



中國計算機學會
CHINA COMPUTER FEDERATION



NOI2024冬令营讲义

(内部资料)

主办单位：中国计算机学会

承办单位：重庆市育才中学校



中國計算機學會
CHINA COMPUTER FEDERATION



NOI2024冬令营讲义

(内部资料)

主办单位：中国计算机学会

承办单位：重庆市育才中学校

NOI2024 冬令营讲义

目 录

第一课堂	(1-160)
信息学竞赛中的持久化数据结构与技巧	李欣隆 (1)
OI 在 TCS 中的应用	许庭强 (46)
题目选讲	戚朗瑞 (78)
量子计算和量子通讯介绍	袁 骁 (87)
组合计数中的递推问题	李白天 (113)
杂题选讲	刘一平 (147)
随机化和近似算法选讲	程思元 (151)
计算机教育漫谈 —— 算法与教育	刘润达 (160)
第二课堂	(198-372)
树的基础算法	史钊镭 (198)
DFS 序的经典应用	祝敏磊 (259)
最短路算法及其应用	肖 然 (282)
树上问题选讲	谢秋锋 (319)
矩阵在信息学中的典型应用	胡伟栋 (355)
信息学竞赛中的字符串问题选讲	诸一行 (372)
营员交流	(402-581)
浅谈树拓扑序计数相关问题的一些方法	陈昕阳 (402)
浅谈一种互质数对与最大公约数的维护算法	杜冠成 (414)
浅谈一些信息合并的处理方式	冯政玮 (425)
线性拟阵奇偶	郭羽冲 (439)
浅谈静态数据结构的合并与分裂	黄洛天 (453)
浅谈 -analogue	李铭乐洋 (464)
范围修改查询问题相关算法及其应用	沈吉溆 (473)
拉格朗日乘数及对偶的应用	施开成 (487)
不相交格路与反射容斥	吴 畅 (510)
浅谈复杂度及其在解决问题方面的应用	杨敏行 (523)
用生成函数刻画随机过程停时	叶 开 (539)
《世界沉睡童话》命题报告	叶李蹊 (548)
关于积性函数求和的一些探讨	周康阳 (565)
浅谈一类树上范围相关问题	朱屹帆 (581)



信息学竞赛中的持久化数据结构与技巧

清华大学 李欣隆

持久化的概念

有时希望对数据结构修改后，修改前的版本仍然存在。

对数据结构的操作一般分为几类：

- 查询：输入一个或多个数据结构；
- 修改：输入一个数据结构，以及修改方式，输出修改后的数据结构；
- 合并：输入多个数据结构，输出一个数据结构；

部分持久化

数据结构的不同版本构成一个序列，其中每一个元素都可以被查询，但只有最后一个元素可以被修改。

每次修改导致序列末尾增加一个新元素。

这个序列称为版本序列。

完全持久化

数据结构的不同版本构成一棵有根树，每个结点是由其父亲修改得到的。

每次修改导致有根树新增一个叶子。

这个树结构称为版本树。

可合并的持久化

- 数据结构的不同版本构成一个有向无环图，每个结点由其前驱合并或修改得到。对于一次合并操作，几个数据结构版本被合并为一个新的数据结构版本，新建的结点的前驱是被合并的结点。
- 对一次修改操作，新建的结点的前驱是被修改的一个结点。
- 这个图称为版本图。

离线处理持久化问题

- 部分持久化

离线后按时间升序扫版本序列，相应进行修改查询。

- 完全持久化

离线后对版本树进行 DFS，相应进行修改、撤销修改、查询；修改一般不能带均摊。

持久化常见方法

- 路径复制
- 肥结点
- 结点分裂

路径复制

- 要求数据结构按有根树存储
- 每个结点有指向每个孩子的指针，没有其它指针
- 修改或查询中，访问一个非根的结点必须先访问它的父亲
- 修改时，如果需要修改某个结点，则复制一份再修改
- 修改时，如果一个结点的孩子被修改，则这个结点也发生修改，需要将指向孩子的指针改为指向修改后的孩子的指针
- 如果需要新建结点，则直接新建
- 如果需要删除结点，则只能删除不被任何指针指向的结点

路径复制

- 不改变原数据结构的单次修改/查询的时间复杂度
- 空间复杂度和时间复杂度相同
- 对数据结构的结构有限制
- 支持部分持久化、完全持久化、可合并的持久化
- 对于带均摊时间复杂度的数据结构，因为同个版本可以被多次修改，可能被卡掉最坏情况

路径复制的应用

- 栈 (push/pop/top)
- 线段树 (单点/区间 修改/查询)
- 平衡树 (单点/区间 修改/查询，分裂/合并)
- 可并堆

肥结点 部分持久化

- 对原数据结构的内存中的每个位置的值，维护其经历的每次修改的时刻和修改后的值
- 相当于部分持久化数组
- 记对值的修改次数为 m_1 ，对值的查询次数为 m_2
- 使用带末端插入的数组，查询时二分查找，时间复杂度 $O((m_1+m_2)\log(m_1))$ ，空间 $O(m_1)$
- 若使用 vEB 树，则时间复杂度可以降低到 $O((m_1+m)\log\log(m_1))$

$O(\log n)-O(1)$ 的动态标号法

- 问题描述：维护双向链表，支持在给定结点前后插入，查询两个结点的先后关系
- 使用重量平衡树（单点插入时只改变 $O(\log n)$ 个结点与根的相对位置）维护结点的顺序
- 在平衡树上每个结点维护一个区间，根对应 $[0, 2^w-1)$
- $[l, r)$ 的左右孩子分别对应 $[l, (l+r)/2)$, $[(l+r)/2, r)$
- 如果一个结点与根的相对位置发生改变，则重新计算结点对应的区间
- 每个结点对应的标号是 $(l+r)/2$
- 插入结点的时间复杂度是 $O(\log n)$
- 比较结点的先后关系只需 $O(1)$ 比较标号

肥结点 完全持久化

- 考虑版本树的括号序列，将其视为一个等效的修改序列
- 每次对值的修改相当于在序列中插入两次修改操作
- 使用 $O(\log n)-O(1)$ 的动态标号法维护等效的修改序列中的时刻
- 使用平衡树维护每个值的修改序列的插入和查询
- 时间复杂度 $O((m_1+m_2)\log(m_1))$ ，空间 $O(m_1)$
- 改进？

$O(1)-O(1)$ 的小规模动态标号法

- 问题描述：维护双向链表，支持在给定结点前后插入，查询两个结点的先后关系；结点不超过 w 个
- 初始化时，给链表每个结点分配等差数列编号，且链表首尾的差充分大
- 插入结点时，标号为前驱和后继的标号的平均数
- 插入结点的时间复杂度是 $O(1)$
- 比较结点的先后关系只需 $O(1)$ 比较标号
- 可以嵌套使用，处理结点不超过 $w^{O(1)}$ 个的情况，每层结点超过 w 个就分裂后重新初始化，并在上层插入一个结点

肥结点 完全持久化

- 对修改序列选出一些代表元
- 用 $O(\log n)-O(1)$ 的动态标号法维护代表元的标号
- 维护性质：
 - 修改序列中，相邻 $\log^2(m_1)$ 个位置有至少一个代表元
 - 每个值对应的修改序列中，相邻 $\log^2(m_1)$ 个位置有至少一个代表元

肥结点 完全持久化

- 每次插入修改操作，均摊增加 $1/\log^2(m_1)$ 个代表元，改变 $1/\log(m_1)$ 个代表元的标号
- 修改序列中，两个代表元间的 $O(\log^2(m_1))$ 个修改操作，用 $O(1)-O(1)$ 的小规模动态标号法（嵌套两层）维护动态标号
- 每个值对应的修改序列中，用vEB树维护代表元，在标号变化时删除并重新插入；用平衡树维护两个代表元间的 $O(\log^2(m_1))$ 个修改操作，在非代表元变成代表元时分裂；支持 $O(\log \log(m_1))$ 查询后继
- 时间复杂度 $O((m_1+m_2)\log \log(m_1))$

结点分裂

- 可以将路径复制和肥结点结合，得到结点分裂方法
- 要求数据结构按有向图存储
- 如果每个结点的入度和出度分别有常数上界，则可以使用结点分裂高效实现完全持久化

结点分裂

- 每个结点维护原数据结构上结点的常数个版本
- 完全持久化中，单个结点的修改对应于在相应结点上插入两个版本（如果修改了指针，指针两侧的结点都要插入版本）
- 一个结点 x 上版本过多时，分裂为两个结点 x, x' ，此时需要相应修改与这个结点相邻的结点（给相邻结点插入一个修改操作，把指向 x 的指针改为指向 x' ）
- 结点上的版本个数足够多时，分裂导致的递归分裂均摊是少的，单次修改是 $O(1)$ 的
- 另外用 $O(\log n) - O(1)$ 的动态标号法套小规模动态标号法维护每个版本的标号，得到 $O(1) - O(1)$ 的动态标号

结点分裂 应用

- 双端队列
- 维护了父指针的平衡树
- 并查集
- 自底向上的动态树的持久化

持久化数据结构

- 1.持久化数组
- 2.持久化线段树
- 3.持久化平衡树
- 4.持久化可并堆
- 5.持久化动态树

洛谷P3919 可持久化数组

- 维护一个数组
- 现在从1时刻到m时刻，每个时刻进行一次操作：
 1. 在某个历史版本上修改某一个位置上的值
 2. 访问某个历史版本上的某一位置的值强制在线

Solution1

- 可以使用路径复制的方法，通过持久化线段树维护这个问题。
- 时间复杂度 $O(\log n)$

Solution2

- 持久化数组和肥结点问题是等价的
- 使用 vEB 树维护的完全持久化肥结点即可
- 时间复杂度 $O(\log \log n)$

持久化线段树

- 分为两种常见的应用：
 1. 持久化线段树维护区间信息
 2. 持久化值域线段树维护值域信息

维护区间信息的持久化线段树

- 有一个常见误区是持久化线段树不能标记下放，这个说法是错的
- 根据路径复制的原则，持久化线段树标记下放时需要新建两个孩子，不然会影响其他版本的信息正确性
- 这样的实现单次操作时空复杂度都是 $O(\log n)$ 的，常数较大
- 使用标记永久化的方法可以减少空间常数，但是标记永久化的功能经常比标记下放弱

维护值域信息的持久化值域线段树

- 被称为所谓的“主席树”
- 利用值域线段树结构相同的特性，可以在值域上多棵树一起二分来降低时间复杂度

洛谷P3834 【模板】可持久化线段树 2

- 给定一个长度 n 的序列，有 m 次询问，每次询问给定 k ，查询区间的第 k 小值。

Solution

持久化平衡树

- 分为两种常见的应用：
 1. 强制在线，支持访问历史版本的平衡树问题
 2. 持久化平衡树的可合并持久化特性

访问历史版本的持久化平衡树

- 从第一种问题的意义上来看，几乎所有平衡树都可以一定程度上持久化：
- treap：复杂度期望 $O(\log n)$ 的完全持久化
- splay：不能直接持久化，可以针对性设计低于暴力的时间复杂度的持久化
- 替罪羊树：复杂度均摊 $O(\log n)$ 的部分持久化
- WBT (BB(α)T)，WBLT，AVL，RBT：复杂度严格 $O(\log n)$ 的完全持久化

访问历史版本的持久化平衡树

- 有一个常见误区是带旋转的平衡树不能持久化，这个说法是错的
- 实际上带父亲指针的平衡树都可以通过结点分裂的方法持久化，而且一般来说带旋转的平衡树也是自顶向下访问的，不需要维护父亲指针，可以直接用路径复制持久化
- 平衡树的分裂合并与旋转是完全等价的操作，二者没有高低之分

洛谷P3835 【模板】可持久化平衡树

- 您需要写一种数据结构（可参考题目标题），来维护一个可重整数集合，其中需要提供以下操作（对于各个以往的历史版本）：
 - 1、插入 x
 - 2、删除 x （若有多个相同的数，应只删除一个，如果没有请忽略该操作）
 - 3、查询 x 的排名（排名定义为比当前数小的数的个数 $+1$ ）
 - 4、查询排名为 x 的数
 - 5、求 x 的前驱（前驱定义为小于 x ，且最大的数）
 - 6、求 x 的后继（后继定义为大于 x ，且最小的数）
- 和原本平衡树不同的一点是，每一次的任何操作都是基于某一个历史版本，同时生成一个新的版本。（操作3, 4, 5, 6即保持原版本无变化）

Solution

洛谷P5055 【模板】可持久化文艺平衡树

- 您需要写一种数据结构，来维护一个序列，其中需要提供以下操作（对于各个以往的历史版本）：
 1. 在第 p 个数后插入数 x 。
 2. 删除第 p 个数。
 3. 翻转区间 $[l, r]$ ，例如原序列是 $\{5, 4, 3, 2, 1\}$ ，翻转区间 $[2, 4]$ 后，结果是 $\{5, 2, 3, 4, 1\}$ 。
 4. 查询区间 $[l, r]$ 中所有数的和。
- 和原本平衡树不同的一点是，每一次的任何操作都是基于某一个历史版本，同时生成一个新的版本（操作4即保持原版本无变化），新版本即编号为此次操作的序号。
- 本题强制在线。

Solution

访问历史版本的持久化平衡树

- 这种持久化数据结构一般用处很小，因为绝大多数序列问题都可以用线段树维护，而值域上的问题又可以用值域线段树维护。
- 持久化平衡树维护值域一般不比持久化值域线段树维护值域有优势。

可合并持久化的平衡树

- 支持合并两个历史版本的持久化平衡树。
- 由于平衡树的合并需要保证值域不相交，否则时间复杂度没有保证，所以一般这种问题出现在序列维护中。

区间复制的持久化平衡树

- 持久化平衡树可以通过分裂合并的方式截取一个区间，将被截取的区间视为一个新的版本的序列
- 通过路径复制实现的持久化平衡树可以直接合并两个历史版本的持久化平衡树
- 实际上持久化线段树也可以做到同样的功能，但是线段树无法高效截取区间

可合并持久化的特性

- 因为可以合并历史版本，所以经过常数操作之后，当前版本中的元素个数可以翻倍。
- 如果一个数据结构的时间复杂度是 $O(\log(\text{元素个数}))$ 的，则经过 $\Theta(n)$ 次操作后，当前版本中的元素个数可能是 $\Theta(2^{(n/c)})$ ，这个时候 $O(\log(\text{元素个数})) = O(n)$ 。
- 这样的特性导致可合并持久化数据结构功能受限，一般需要限制每个版本中的元素个数。

可合并持久化的平衡树

- 从第二种问题的意义上来看，一部分平衡树可以支持可合并持久化：
- treap：复杂度未知的可合并持久化
- splay：不能
- 替罪羊树：不能
- WBT ($BB(\alpha)T$) , WBLT , AVL , RBT：复杂度严格 $O(\log n)$ 的可合并持久化

Treap 实现区间复制

- Treap 被普遍认为是唯一的持久化平衡树，其实是持久化平衡树中性质最差的一个。
- 如果 treap 采用常规的方法实现可合并持久化，则在构造数据下时间复杂度 $\Theta(n)$ 。
- 直接初始序列长度为 1，通过 $\Theta(\log n)$ 次操作自我复制合并，可以让 treap 中所有元素的随机权值相同，treap 由于随机堆权值的性质被破坏，所以时间复杂度没有保证。

Treap 实现区间复制

- Treap 的分裂方式是当 treap 确定形态后唯一的，所以只能修改合并的方式。
- 合并时不再维护堆权值，若合并两个节点 $T1$ 和 $T2$ ，则以 $\text{size}(T1)/(\text{size}(T1)+\text{size}(T2))$ 的概率认为 $T1$ 的随机权值比 $T2$ 优先级大。
- 因为区间复制会同时复制树的结构，所以在区间复制的情况下时间复杂度未知。

其他可合并持久化的平衡树

- WBT ($BB(\alpha)T$), WBLT, AVL, RBT 等这些平衡树可以保证深度 $O(\log n)$ 且每次操作时间复杂度严格 $O(\log n)$, 所以可以简单实现可合并持久化的需求。

洛谷P8263 [Ynoi Easy Round 2020] TEST_8

给出一个长度为 n 的 01 串 S (一个由 0 和 1 组成的序列, 下标为 1 到 n 的整数)。

支持以下几种操作:

操作1: 给出 l, r, k , 将 01 串下标为 l 到 r 的一段重复 k 次并放回原位;

操作2: 给出 l, r, k , 将 01 串下标为 l 到 r 的一段带翻转地重复 k 次 (具体地说, 第 i ($1 \leq i \leq k$) 次重复时, 若 $i-1$ 的二进制表示中有奇数个 1, 则这次重复时要左右反转, 否则不变), 最后放回原位;

操作3: 给出 l, r , 将 01 串下标为 l 到 r 的一段删除;

操作4: 给出 k , 求 01 串中从左到右第 k 个 1 的位置, 若 k 超过 01 串中 1 的个数, 则输出 -1 。

对于 100% 的数据, $1 \leq n, m \leq 10^5$, 01 串的长度始终不超过 10^8 ;

Solution

- 区间复制 k 次应该如何解决?
- 直接复制 k 次则时间复杂度和 k 相关。

Solution

- 可以考虑类似快速幂的倍增合并。
- 假设要将区间复制 k 次, 则初始设平衡树 $A(1)$ 表示一个要被复制的区间, 我们要得到 $A(k)$ 。
- 可合并持久化的平衡树可以实现 $A(2i)=A(i)+A(i)$ 。
- 则从 $A(1)$ 开始, 求出 $A(2), A(4), A(8) \dots$, 然后对 k 做二进制拆位即可通过 $O(\log n)$ 次合并得到 $A(k)$ 。

Solution

- $A(2i)=A(i)+A(i)$ 的复杂度是什么？
- 在 treap 上，这个操作的时间复杂度为 $O(\log n)$ 。
- 在 WBLT，AVL 上，这个操作的时间复杂度为 $O(1)$ 。
- WBLT 可以直接 $O(\log \alpha)$ 时间复杂度合并两个大小比例为 α 的平衡树。
- AVL 可以 $O(h)$ 时间复杂度合并两个深度差不超过 h 的平衡树。
- 于是这个做法中，treap 的时间复杂度为 $O(\log^2(n))$ ，其他持久化平衡树为 $O(\log(n))$ 。

洛谷P8524 [Ynoi2078] 《A theory of consciousness from...》阅读报告（更新中...）

给定一个长度 n 的序列 a_1, \dots, a_n ，需要进行 m 次操作，操作共有三种类型：

操作 1：给定 l, r, x ，先新建一个数组 b 满足 $b_i = a_i$ ，之后将 a_x, \dots, a_{x+r-l} 同时修改为 b_l, \dots, b_r ；

操作 2：给定 l, r ，将 a_l, \dots, a_r 同时修改为除以 2 下取整后的值；

操作 3：给定 l, r ，求 a_l, \dots, a_r 的和。

对于 100% 的数据，满足 $1 \leq n \leq 3 \cdot 10^6$ ， $1 \leq m \leq 3 \cdot 10^6$ ，

序列的初值满足 $1 \leq a_i \leq 10^9$ 。

对每个操作满足 $1 \leq l \leq r \leq n$ 。

对每个操作 1 满足 $1 \leq x \leq x+r-l+1 \leq n$ 。

以上所有数值为整数。

Solution

- 这个问题如果不支持可合并持久化，则可以使用均摊时间复杂度的做法解决。
- 区间复制操作破坏了均摊时间复杂度的保证。
- 考虑到区间除操作进行超过 $O(\log(a_i))$ 次后所有数都会变成 0，于是我们可以同时维护区间进行 $0, 1, 2, \dots, \log(a_i)$ 次除操作的和，这样去均摊。
- 总时间复杂度为 $O((n+m\log n)\log(a_i))$

可合并持久化的空间优化

- 如果要同时维护多个历史版本，则可合并持久化难以实现空间优化，参照持久化线段树维护 n 个历史版本时需要 $\Theta(n\log n)$ 的空间。
- 但是在区间复制的大部分情况下，可合并的持久化只涉及到一个版本，只需要维护当前的版本，不需要维护所有历史版本。
- 可以利用这一点进行空间优化。

智能指针

- 对每个节点开一个额外的域，表示有多少个指针指向其，如果一个点没有被指针指向，就将其回收，在没有环的图上这个方法是可以保证空间复杂度的，而可持久化的过程是一个 DAG 结构，所以可以保证回收掉没有意义的节点。
- 这个方法带来的额外常数很小。
- STL 中有这个数据结构，但是使用 STL 会导致这里访存代价多一倍，所以比较慢，如果可以内嵌在数据结构中则不会多访存代价。

定期重构

- 对区间复制类问题，可以每 $\Theta(n/\log n)$ 次重构一次整个序列，并且回收内存。
- 这样空间复杂度是： $\Theta(n/\log n) * O(\log n) = O(n)$ 的。
- 总时间复杂度是 $m / \Theta(n/\log n) * n = \Theta(m \log n)$ 的。
- 于是时间复杂度 $O(m \log n)$ ，空间复杂度 $O(n)$ 。

底层分块

- 当数据结构一个结点维护 $C = \omega(1)$ 的信息时，可以进行底层分块，即当结点大小比维护的信息大小还小时，不递归建立子树。
- 这样数据结构一个结点可能大小是 C 的。
- 也可以理解为先对序列按 C 大小分块，然后对这个块建立数据结构。
- 这样原本数据结构结点数是 $O(n)$ 的，空间复杂度是 $O(nC)$ ，可以降低到结点数 $O(n/C)$ ，空间复杂度 $O(n)$ 。

可合并持久化的空间优化

- 注意到以上优化对序列长度超线性时均无用。
- 因为持久化平衡树可以通过合并快速增加序列长度，若序列长度和 n 不同阶，则上述优化均无用。

HDU 6087 Rikka with Sequence

题意：维护一个序列 a ，要求支持：

操作1：区间求和

操作2： $\text{for}(\text{int } i = l; i \leq r; ++i) a[i] = a[i - k]$

操作3：区间 $a[i] = b[i]$ ，其中 $b[i]$ 等于一开始输入的 $a[i]$

Solution

- 区间复制 k 次直接用上述方法实现即可。
- 回到初始状态则只需要同时维护两个历史版本，当前的版本和初始的版本。
- 于是可以用节省空间的方法做到线性空间复杂度。

CF702F T-Shirts (加强版)

有 n 种T恤，每种有价格 c_i 和品质 q_i 。

有 m 个人要买 T 恤，第 i 个人有 v_i 元，每人每次都会买一件能买得起的 q_i 最大的T恤。一个人只能买一种T恤一件，所有人之间都是独立的。

问最后每个人买了多少件 T 恤？如果有多个 q_i 最大的T恤，会从价格低的开始买。

reformed by Rainy_chen。

- 强制在线，人做为询问在线给出，每次查询需要在线输出答案。

Solution

- 对所有衣服按照 $q[i]$ 从大到小排序；
- 对每个人维护 $a[i]$ 表示其剩余的钱， $b[i]$ 表示其买的衣服数量；
- 对所有人按照 $a[i]$ 从小到大排序，初始 $b[i]=0$ ；
- 之后从 1 到 n 枚举每件衣服 j ；
- 衣服 j 对人的影响就是把全局 $a[i] \geq c[j]$ 的人都进行 $a[i] -= c[j]$ 的操作，并且进行 $b[i]++$ 的操作。

Solution

- 用一棵持久化平衡树倒着维护所有初始资金最后买了多少件衣服。
- 初始情况是映射 $b[i]=0$
- 每次在前面增加衣服的时候相当于 $[0, C)$ 的区间完全不变， $[C, \max v)$ 的区间由 $[0, \max v - C)$ 的区间平移过来，并且区间的 b 加上 1。
- 这个直接用持久化平衡树即可实现，预处理结束后对每个人的查询相当于单点查询。

Solution

- 其实本质上是在优化一个如下形式的 DP :
- 如果 $j < c[i]$, 则 $f[i][j] = f[i+1][j]$
- 否则 $f[i][j] = f[i-c[i]][j] + 1$
- 然后 $f[n+1][j] = 0$
- 总时间复杂度 $O((n+m)\log n)$

Hdu5118 GRE Words Once More (改)

- 给定一个 DAG , 每条边有边权 , 每次查询从一个点出发到任意点 , 经过的路径中所有边权按顺序连接构成的字符串 , 字典序第 k 小的路径 , 是在哪个点结束的。
- $n, m \leq 1e5, k \leq 1e18$ 。
- 每个点的每种边权的出边只有一条。
- 强制在线。

Solution

- 注意到每个节点开始 , 假设其有两条出边 , 边权为 1,2
- 则任何先经过 1 边的路径都比先经过 2 边的路径的字典序小
- 每个节点可以用一个持久化平衡树维护答案。
- 每次在 DAG 上按拓扑顺序合并一个点到的所有边的平衡树 , 每个平衡树是按原顺序合并过来的。
- 这里合并的时候保留序列前 $\max k$ 个位置即可 , 不然由于可合并持久化会导致元素个数指数级上涨 , 时间复杂度会出现问题。

Solution

- 考虑先写出一个暴力 DP， $f[i][j]$ 表示从 i 结点开始走字典序第 j 小的路径会到哪个点。
- 这个 DP 转移需要按照拓扑顺序转移。
- 本质上这道题的做法就是持久化平衡树优化这个 DP 的过程。

树套持久化平衡树

- 可合并历史版本的持久化平衡树功能非常强大，经常作为树套树的内层树使用。
- 这里的树套树的定义比 Range Tree 的定义更广泛。
- 有三种已知的应用：
 1. 将持久化平衡树作为标记下传
 2. 将持久化平衡树作为信息合并
 3. 外层树作为分治，内层用持久化平衡树优化 DP

将持久化平衡树作为标记下传

- 正常情况下线段树的标记大小都是常数，如果尝试线段树每个结点维护一个标记构成的序列，则标记下放到叶子结点时，每个结点都有 n 个标记，时间复杂度退化至 $O(n^2)$ 。
- 如果不暴力维护标记，而是使用一个数据结构维护标记，同样会遇到上述问题。
- 但是持久化平衡树可以利用可合并持久化的特性规避这个问题。

将持久化平衡树做为标记下传

- 考虑线段树，每个结点维护一个持久化平衡树，表示这个结点上所有标记构成的序列（有序）。
- 每次标记下放时需要将儿子的标记和自己的标记合并，并清空自己的标记。
- 可以将儿子的持久化平衡树和自己的持久化平衡树合并成一棵新的平衡树，然后将自己的结点信息和标记合并。
- 这里需要将自己的一个持久化平衡树与两个儿子合并，所以暴力实现会复制 n 个元素，但是持久化平衡树可以以 $O(\log n)$ 的代价实现这个操作。

BZOJ3946 无聊的游戏（改）

- 维护一个长为 n 的序列，每个位置是个长度 ≤ 10 的字符串，有 m 次操作：
- 1. 给区间字符串前面加上一个输入的字符串 s 。
- 2. 求区间 LCP 长度。
- $n \leq 1e6, m \leq 3e4, \sum |S| \leq 1e7$
- 强制在线

Solution1

- 第一维的区间操作线段树分治掉，套一个持久化平衡树维护标记，一个持久化平衡树维护信息（即外层树当前这个区间的 LCP）。
- 标记下放的时候，标记的持久化平衡树直接区间复制合并到儿子的标记的持久化平衡树上，同时把标记的持久化平衡树区间复制合并到节点的答案持久化平衡树上。
- 区间 LCP 直接在 $O(\log n)$ 棵持久化平衡树上二分。
- 总时间复杂度 $O(n + m \log^3 n + |S|)$

Solution2

- 考虑利用差分的思想，维护相邻两个位置的 LCP 长度。
- $f[i]$ 表示 i 和 $i+1$ 位置的字符串的 LCP 长度。
- 询问区间 $[l, r]$ 的 LCP 长度，可以表示为 $f[l] \sim f[r-1]$ 的区间 min。
- 区间 `push_front` 的时候只影响边界位置的 f 。
- 还是沿用线段树套可持久化平衡树的方法，每次只需要在两个位置求两棵可持久化平衡树维护的字符串的 LCP，二分哈希即可。
- 总时间复杂度 $O(n+m\log^2 n+|S|)$

将持久化平衡树做为信息合并

- 考虑线段树，每个结点维护一个持久化平衡树，表示这个结点对应的区间信息。
- 这种信息比较特殊，不像普通的线段树套平衡树，平衡树维护的是区间值域信息，导致因为值域相交，所以无法合并两棵平衡树。
- 这种信息是以序列区间为序的信息，但是信息比较复杂，无法通过维护亚线性于区间长度的信息来实现区间信息合并。

将持久化平衡树做为信息合并

- 如果线段树上合并两个区间的信息的复杂度是 $O(\text{区间长度})$ ，则复杂度退化至暴力，但是持久化平衡树可以利用可合并持久化的特性规避这个问题。
- 每次线段树需要合并信息时，将左右儿子的持久化平衡树合并时进行某种简单的修改，之后合并成一个新版本，做为父亲的信息。

单侧递归模型

- 类似“楼房重建”问题，合并两个区间信息时，递归进其中一侧，一直递归到叶子结束。
- 线段树每次操作合并 $O(\log n)$ 个区间，每次合并需要 $O(\log n)$ 的代价，单次操作时间复杂度 $O(\log^2 n)$ 。

区间不匹配括号信息

- 维护一个括号序列，有一些修改操作，询问区间括号匹配后的信息。
- 这个是单侧递归的一种通用模型。
- 已知做法有：
 1. 线段树套持久化平衡树
 2. 单侧递归
 3. 类 top tree 的结构

CF1340F Nastya and CBS

你需要动态维护一个多种括号组成的括号序列。需要支持两种操作：

- 1 i t : 修改 i 位置的括号为 t (绝对值表示种类，正数表示是左括号，负数表示是右括号)。
- 2 l r : 查询 $[l, r]$ 区间是否是一个合法的括号序列。

序列长度为 n ，不同的括号种类数为 k ，操作次数为 q 。

一个多种括号组成的括号序列 S 是合法的当且仅当：

1. S 为空。
2. S 可以表示为 $A + B$ 的形式，其中 A, B 都是合法的。
3. S 可以表示为 $(^ + A +)^*$ 的形式，其中 A 是合法的并且 $(^$ 与 $)^*$ 是一对相同种类的左右括号。

例如 $0(00)$ 和 $((((()))$ 是合法的，而 $)))$ 和 $((((($ 不是。

Solution1

- 考虑线段树维护序列，对一个节点维护其经过括号匹配后的结果，如果一个节点出现了两个不匹配的左右括号挨在一起的情况（如“(j)”），则这个节点不可能被成功匹配，可以直接标记匹配失败
- 所以一个节点如果可能和别的节点合并后成功匹配，其不匹配括号一定是一些左括号然后一些右括号
- 为了方便判括号是否可以匹配，使用字符串哈希的方法，考虑我们对每个节点维护了其不匹配括号中左括号与右括号分别的哈希值，如何合并节点信息

Solution1

- 这里用小括号，只表示括号方向
- 如))(((与))(((合并得到))(((，可以发现合并时，一定会抵消掉左儿子的所有右括号，或者右儿子的所有左括号，否则匹配失败
- 如上述例子合并时，我们需要知道右儿子这x个左括号的哈希值，这个可以通过在右儿子的子树中递归实现：
- 若右儿子的左儿子的左括号大于等于x个，我们只需要递归右儿子的左儿子即可
- 否则右儿子的左儿子的左括号一定全部包含在这x个括号中，因为维护了所有不匹配左括号的哈希值，所以只需要递归右儿子的右儿子

Solution1

- 这样合并时构成了一个单侧递归结构，合并两个节点的时间复杂度为 $O(\log n)$ ，总时间复杂度为 $O(\log^2 n)$ 。
- 建树的时间复杂度为 $T(n)=2T(n/2)+O(\log n)$ ，解得 $T(n)=O(n)$ 。
- 总时间复杂度 $O(n+m\log^2 n)$ 。

Solution2

- 考虑用线段树套持久化平衡树维护。
- 线段树维护序列，持久化平衡树维护区间合并后剩下的括号。
- 每次合并两个括号序列时，需要处理中间新产生的匹配，这里需要做一个类似单侧递归的操作，合并时间复杂度 $O(\log n)$ 。
- 总时间复杂度 $O(n+m\log^2 n)$ 。

Luogu6781 [Ynoi2008] rupq

给定一个长为 n 的括号序列，每个位置有一个括号 a_i ，其中 $a_i = 1$ 代表是左括号 '(', $a_i = 0$ 代表是右括号 ')', a_i 位置的括号有一个权值 b_i 。

对任意括号序列，通过不断删除形如 '()' 的子段直到无法操作，最终可以得到一个唯一的序列，这个序列称为不匹配括号，如 $a_1 = '(', a_2 = ')', a_3 = '(', a_4 = ')', a_5 = '(',$ 这个括号序列的不匹配括号序列为 '()((('，其不匹配括号对应位置为 1, 2, 5，其不匹配括号对应位置的 b 为 b_1, b_2, b_5 。

有 m 次操作，操作有以下三类：

1 x y : 进行单点修改，即 $a_x := 1 - a_x; b_x := y$ 。

2 l r : 求 $a[l..r]$ 中不匹配括号对应位置的 b 从左到右的 NAND (32 位的按位与非，可以见 https://en.wikipedia.org/wiki/NAND_logic)，max (最大值)，将二者 xor 后输出。如果不匹配括号为空序列，则 NAND 的答案为 $2^{32} - 1$ ，max 的答案为 0。

NAND 即以 $c_0 = 2^{32} - 1$ 为初值，然后依次 $c_i = \text{NAND}(c_{i-1}, b_i)$ ，对于一个 n 个值的序列 b ，最后答案为 c_n 。

3 l r : 将 $a[l..r]$ 和 $a[r+1..n]$ 交换位置， b 也做同样的操作。

对于 100% 的数据，

$$0 \leq a[i] \leq 1,$$

$$0 \leq b[i] \leq 10^9,$$

$$1 \leq n \leq 2 \times 10^6, 1 \leq m \leq 2 \times 10^5.$$

Solution

- 使用平衡树维护序列，这样可以支持分裂合并
- 每个区间的括号合并完成后应该形如 "))))((((("，考虑怎么合并两个儿子的信息，不失一般性，设左儿子的 "(" 数量多于右儿子的 ")" 数量，其它的情况类似
- 那么匹配完成后分为三部分，左儿子的 "))))"，左儿子的 "(" 和右儿子的 ")" 匹配之后左儿子剩下的 "((((("，右儿子的 "))))"。假如我们分别维护了 "))))" 和 "(((((" 对应的信息，那么我们只需要考虑中间的部分如何维护

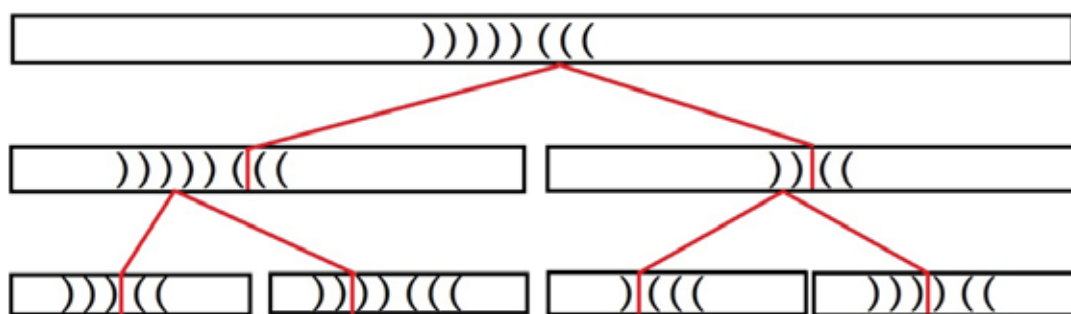
Solution

- 考虑中间的部分相当于在计算左儿子的前若干个 "(" 的信息，于是实现函数 $\text{solve}(u, K)$ 表示计算节点 u 的前 K 个 "(" 的信息。
- 考虑 u 的两个儿子的情况，如果右儿子的 ")" 不少于左儿子的 "("，那么意味着左儿子的 "(" 不会造成贡献，递归到 $\text{solve}(\text{rson}[u], K)$ 。
- 否则，设左儿子的 "(" 个数比右儿子的 ")" 个数多 t ，那么如果 $K \leq t$ ，那么全部由左儿子贡献，递归到 $\text{solve}(\text{lson}[u], K)$ ；否则，贡献由中间剩余的 "(" 和右儿子的 $\text{solve}(\text{rson}[u], K-t)$ 合并得到，由于中间剩余的 "(" 我们已经维护出来了，所以只需要递归右边。
- 于是每次只需要向一边递归，复杂度 $O(\log n)$ 。

Solution

- 这里 max 和 NAND 都可以写做半群信息的形式，所以直接 $O(1)$ 合并都行。
- 总时间复杂度 $O(n+m\log^2 n)$

线段树上维护线性个word的信息的特殊的
二分的支持



- 这道题里我们就是维护了如 $(()))((()))(((), (()))((()))$ 这两个节点中间那一段的答案，避免了两侧递归，动态凸包比这个结构复杂，但也是在这样的合并结构上二分。

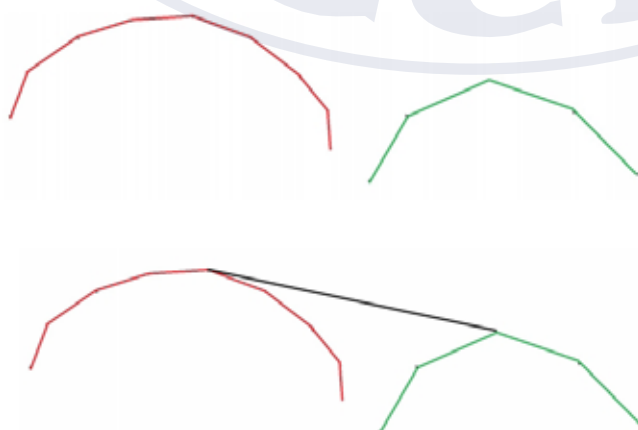
完全动态凸包

- 支持插入一个点，删除一个点，维护凸包上点数，并且能支持一般的可二分操作。
- 强制在线。

Solution1

- 因为外层带插入，所以外层是平衡树。
- 我们使用平衡树套持久化平衡树。
- 第一维用平衡树按 x 把点排序，然后平衡树的一个区间对应 x 的一个区间，平衡树的两个儿子对应两个不相交的 x 的区间，持久化平衡树维护的是区间凸包。
- 合并的时候一定是左儿子凸包的一个后缀和右儿子凸包的一个后缀拼起来。
- 需要 $O(\log n)$ 合并两个 x 区间不相交的凸包。

$O(\log n)$ 合并 x 轴不相交凸包



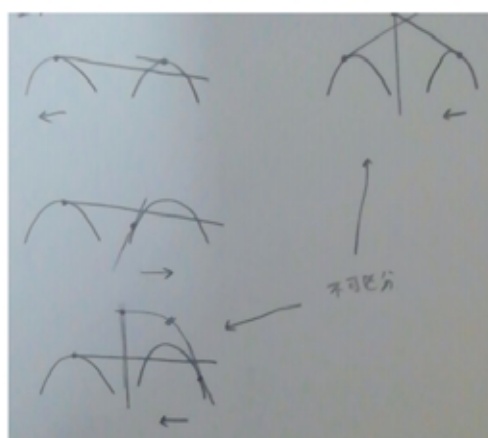
Solution1

- 如果左边切到了右边
- 第一种情况是右边的点在左边切线上方
- 这种情况把右边的点向左二分
- 第二种情况是右边的点在左边切线下方
- 这种情况把右边的点向右二分
- 剩下两种情况不可区分



Solution1

- 我们找一个 $x=k$ 的线，这个 $x=k$ 不在两端的 x 区间内，看两个切线和 $x=k$ 交点
- 交点 y 坐标高的一边往另一边二分
- 右边切到了左边类似讨论



Solution1

- 这里外层树要支持插入元素，按惯例使用一棵重量平衡树维护。
- 当不平衡的时候外层树旋转，直接使用两个儿子的持久化平衡树复制一个区间合并即可。
- 这里注意到由于两棵持久化平衡树合并是 $O(\log n)$ 的，所以外层的重量平衡树可以换成任意的平衡树。
- 总时间复杂度 $O(n+m\log^2 n)$ ，空间复杂度 $O(n+m\log n)$

Solution2

- 用单侧递归的结构也可以实现完全动态凸包，优势是空间是线性的，但是目前按照我的理解，单侧递归只能维护凸包点数和支持二分，不能维护凸包上的点的半群信息。

单侧递归方法无用论

- 线段树套持久化平衡树的方法在功能上是严格不弱于单侧递归结构。
- 使用智能指针或者定期重构的方法，可以让线段树套持久化平衡树的方法空间复杂度也为线性并且时间复杂度保持不变。

单侧递归方法无用论

- 建树时空复杂度： $T(n)=2T(n/2)+O(\log n)$ ，解得 $T(n)=O(n)$ 。
- m 次操作的时空复杂度：
- $O(n/\log^2 n)$ 次操作重构一次，单次操作的空间复杂度为 $O(\log^2 n)$ ，总空间复杂度 $O(n)$ ；
- 时间复杂度方面每次重建建树 $O(n)$ ，重构次数 $O(m\log^2 n/n)$ ，重构的时间复杂度 $O(m\log^2 n)$ 。
- 目前看来单侧递归的方法是无用的，从动态凸包的例子也可以看出线段树套持久化平衡树的方法的确更强。

分治套持久化平衡树优化 DP

- 常用于区间函数复合类问题，初始给定一个函数序列，第 i 个位置是一个函数 $f(i, x)$ ，多次询问时给定区间 l, r 以及初始参数 a ，求 a 依次经过 l 到 r 中每个函数，经过第 i 个函数时， $a := f(i, a)$ ，求最终的 a 。

分治套持久化平衡树优化 DP

- 可以对外层建立一棵线段树，线段树的结点如果对应序列区间 $[l, r]$ ，则维护一个映射 $g(a)$ 表示初始参数 a ， a 依次经过 l 到 r 中每个函数，经过第 i 个函数时， $a := f(i, a)$ ，最终的 a 。
- 如果函数形式符合条件，则可以使用持久化平衡树优化 DP 来求出区间的 g 映射。

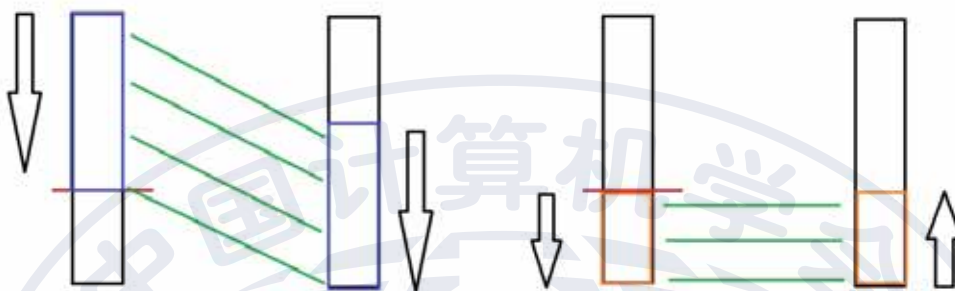
洛谷P8264 [Ynoi Easy Round 2020] TEST_100 (加强版)

- 给定一个序列 a ，每次查询时给出 l, r, v 。
- 令 i 从 l 到 r ，依次做 $v = |v - a[i]|$ 的迭代，求最后的 v 。
- 强制在线。
- $n, m \leq 1e5$, 值域为 $[0, 1e9]$
- 时间限制：6s，空间限制：2G

Solution

这个问题的分段函数段数很大，只需 $O(\log v)$ 个序列上的位置就可以构造出 $\Theta(v)$ 段的分段函数，所以需要使用一种 DAG 结构的数据结构，比如树套树或者带区间复制的可持久化平衡树。

先考虑全局查询怎么做，可以从右往左扫描线，维护一个可持久化平衡树，初始为恒等映射，左边加入一个线性变换 $x = |x - a_i|$ 时，等价于将所有左边 $\geq a_i$ 的元素 x' 会映射为 $x' - a_i$ ，所有 $< a_i$ 的元素 x' 会映射为 $a_i - x'$ 。于是等价于将右边的映射中一个区间复制过来，另一个区间翻转后复制过来，然后合并，这个使用可持久化平衡树可以高效地维护。



Solution

为了处理区间查询，只需将序列进行一次分治即可，最后的数据结构是一个线段树套可持久化平衡树的结构。

std 有进行一些常数优化，如线段树最底层按 $\Theta(\log n \log v)$ 大小分块，线段树最上层去掉 $O(\log \log n)$ 个节点，改为分块，线段树可以直接从右儿子中插入左子树部分的操作而得到父亲结点的可持久化平衡树。

总时间复杂度 $O((n + m) \log n \log v)$ 。

持久化可合并堆

- 维护一个数据结构，支持以下操作：
 1. 插入
 2. 删除最小值
 3. 查询最小值
 4. 合并两个堆
- 目前最优复杂度是除了 2 操作 pop 的时间复杂度为 $O(\log n)$ ，其他操作时间复杂度均为 $O(1)$ 。
- 可以完全持久化，并且一定程度上可以合并历史版本。

持久化可合并堆

Brodal Queue

- 这是什么？
- http://en.wikipedia.org/wiki/Brodal_queue
- 插入、查询最小值、合并、减小某个元素的值都是最坏复杂度 $O(1)$ 的，只有删除操作是 $O(\log n)$ 的。
- 并且它能够可持久化。
- 网上流传的一个关于持久化可并堆的pdf
- 这个说法也是错的，持久化的Brodal Queue无法减小某个元素的值，只能做上述的四个操作，而且Brodal论文中给出的是一个新的数据结构，叫做斜二项堆
- 如果对Brodal queue感兴趣可以看<https://www.luogu.com.cn/problem/P6019>

持久化可合并堆

- 斜二项堆在除了decrease key 以外的操作可以做到最优理论复杂度，并且代码非常简单，效率也不差。
- 一般用持久化的左偏树就够用了，持久化可并堆的应用主要在 k 短路 问题上。

斜二项堆

- 这是一种被设计用于支持可持久化的可并堆，除了不支持减小权值外，其余操作都是最坏情况最优。
- 斜二项堆的结构可如下递归定义：
 1. 一棵 0 级斜二项树包含一个非空的斜二项堆；
 2. 一棵 $n+1$ 级斜二项树是一棵二叉树，根的左右孩子为两棵 n 级斜二项树，根存储着 0 或 1 个非空的斜二项堆，根还同时维护了子树内的最小值；
 3. 一个非空的斜二项堆 T 由一个值 T_0 和一个序列 T_1, T_2, \dots, T_k 组成，其中 T_i 是 r_i 级斜二项树，满足 $r_1 \leq r_2 < r_3 < \dots < r_k$ 。 T_0 是最小值，即 T_i 中的值都不小于 T_0 。

斜二项堆

- 插入
- 新建一个只有一个元素的堆表示待插入的元素，然后和待插入的堆合并，时间复杂度 $O(1)$ 。

斜二项堆

- 合并
- 只需考虑待合并的两个堆 T, T' 非空且 $T_0 \leq T'_0$ 的情况。若 $r_1 = r_2$ ，则将 T', T_1, T_2 合为一棵 $r_1 + 1$ 级的斜二项树取代 T_1, T_2 ；否则将 T' 作为一棵 0 级的斜二项树插入到 T_1 前面。时间 $O(1)$ 。

斜二项堆

- 查询最小值
- T_0 为最小值，时间复杂度 $O(1)$ 。

斜二项堆

- 删除最小值
- 删去最小值 T_0 ，从 T_1, \dots, T_k 中找到新的最小值所在的树 T_i ，找到 T_i 中的最小值 T'_0 。
- 将 T'_0 到根路径上的结点取出，将 T_i 拆分为多棵树。将 T'_0 对应的斜二项堆 T' 取出，将 T'_1, \dots 等树、 T_i 拆分出的树、以及原有的 $T_1, \dots, T_{i-1}, T_{i+1}, \dots, T_k$ 中同级的树合并，重复至没有同级的树，然后 T'_0 成为新的 T_0 ，将之前取出的结点中的斜二项堆插入回去，完成删除。
- 时间复杂度 $O(\log n)$ ，其中 n 为当前堆的结点数。

指针歧义问题

- Decrease Key 的操作是给定一个指针，将这个指针对应的权值减小，从而向堆顶移动。
- 在可合并的持久化数据结构中，一个指针可能有多个父亲结点，修改之后向哪一个父亲结点移动有歧义。
- 如果给出一个时刻，将这个时刻的指针 decrease key，则这里可以归约动态前驱问题，时间复杂度下界 $O(\log \log n)$ 。
- 指针歧义问题在可合并持久化的自底向上平衡树，自底向上 top tree 中也存在。

持久化可并堆的功能过弱

- 部分持久化：等价于持久化变量。
- 完全持久化：不比持久化值域线段树，平衡树有优势。
- 可合并持久化：可以支持值域相交的合并，但如果用于数据结构问题上，由于数据结构本身自带分治结构，经常没有用处。
- 可以 $O(1)$ 合并的数据结构不应该能支持强功能。

BZOJ3489 (加强版)

- 给出一个长度为 n 的序列，给出 m 个询问：在 $[l, r]$ 之间找到一个在这个区间里只出现过一次的数，并且要求找的这个数尽可能大。如果找不到这样的数，则直接输出 0。
- $n \leq 1e5, m \leq 1e6, 256MB$
- 强制在线

Solution

- 还是通过前驱后继来算出每个位置的贡献范围
- 设 i 的前驱为 p_i ，后继为 q_i ，把询问看成点 (L, R) ，有贡献的 i 满足 $L \in (p_i, i]$ 且 $R \in [i, q_i)$ ，询问的就是覆盖这个点的矩形的最大值。

Solution

- 那么可以用持久化树套堆，插入矩形时一维持久化，一维区间插入，用堆维护最大值。
- 注意这里的“持久化堆”只需要查询历史，因此只需要把最大值记下来就好了。
- 因为这里“持久化堆”询问无自由度，所以部分持久化的堆和持久化的变量完全等价。
- 总时间复杂度 $O(n \log^2 n + m \log n)$ ，空间复杂度 $O(n \log n)$ ，可以更优。

Luogu7561 「JOISC 2021 Day2」道路建设

JOI 王国坐落于一个 xy 平面上，王国里有 N 座小城，分别编号为 $1 \dots N$ ，其中第 i 个小城的坐标为 (X_i, Y_i) 。

现在王国正筹划着在小城之间建 K 条路，而连接两个不同的小城 i 和 j 将花费 $|X_i - X_j| + |Y_i - Y_j|$ 元。在这里我们认为「连接小城 i 和 j 」和「连接小城 j 和 i 」本质相同。

和往常一样，你成为了这个项目的主管。为了估算花费情况，你了解连接一些小城所需的花费。在这 $\frac{N(N-1)}{2}$ 条可能的道路中，你了解最便宜的 K 条道路的花费。

你的任务是，给出小城的坐标以及 K 值，编写一个程序计算最便宜的 K 条道路的花费。

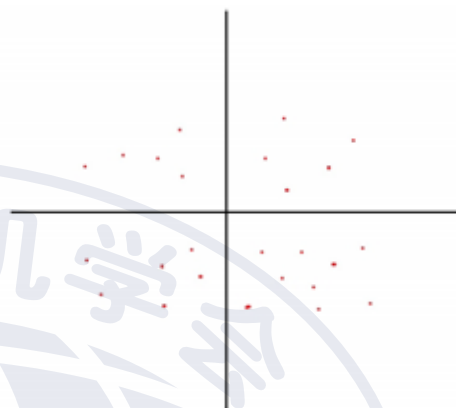
第一行两个整数 N, K 。

接下来 N 行每行两个整数 X_i, Y_i 。

对于 100% 的测试数据，有 $2 \leq N \leq 250000$ ， $1 \leq K \leq \min(250000, \frac{N(N-1)}{2})$ ， $-10^9 \leq X_i, Y_i \leq 10^9$ ， $(X_i, Y_i) \neq (X_j, Y_j) (1 \leq i < j \leq N)$ 。

Solution

- 考虑每次询问拆出四个象限；
- 从左往右维护一棵持久化线段树；
- 这样的扫描线加持久化线段树；
- 可以去掉一维的限制；
- 这样我们可以对单点支持查询了。



Solution

- 每棵持久化线段树维护；
- 结点内到对应区间两端最近点；
- 这样我们已经可以 $O(\log n)$ 对一个点；
- 查与其第 $k=1$ 近的点情况了。



Solution1

- 我们需要对一个点，支持找到距离其最近的点，并且支持删掉这个最近的点。
- 这里我们尝试一种全新的树套树，用外层线段树，内层套四个持久化可并堆，分别维护区间内所有点中 $x+y, -x+y, x-y, -x-y$ 的最小值。
- 由于持久化可并堆合并是 $O(1)$ 的，所以这里时间复杂度不会变高。

Solution1

- 每次需要支持在线段树根结点上找到全局最近的点，并且支持删除最小值，这两个操作我们可以直接在线段树根结点的持久化可并堆上做。
- 持久化可并堆删除最小元素是 $O(\log n)$ 的，所以可以支持 $O(\log n)$ 删除最小元素同时维护出新的最小值。
- 然后我们发现，可以顺便记录下最小值在序列上的位置，然后将序列上对应位置的线段树叶子的堆（只有 1 个元素）清空，然后向上合并到根结点，由于持久化可并堆合并是 $O(1)$ 的，所以这样做居然复杂度还是 $O(\log n)$ 的。

Solution2

- 这里持久化线段树是可以直接标记删除一个点的。
- 这里因为持久化了所以可以直接单点修改然后变成一个新的版本。
- 于是我们对 k 的情况每次找出最近距离的点，然后删掉即可。
- 这样我们可以 $O(k \log n)$ 查与一个点距离前 k 近的点。
- 因为求的是全局的 k 近点对，所以套个堆维护一下当前所有点与最近的点距离，每次找最近的一个二元组。
- 总时间复杂度 $O((n+k) \log n)$ 。

未公开题目

给你一个长为 n 的序列 a 。

定义一个合法二元组 (i, j) 需要满足 i, j 为整数, 且 $i \leq j$ 。如果 $i > j$, 则 (i, j) 不构成合法二元组。

定义一个合法二元组 (i, j) 的分数为 $a_i - a_j$ 。

定义一个合法二元组 (i, j) 在区间 $[l, r]$ 内, 当且仅当 $l \leq i, j \leq r$ 。

有 m 次操作:

`l r x`: 表示将序列中第 l 个位置到第 r 个位置都加上 x 。

`q r k`: 表示询问选出 k 个不同的合法二元组, 每个合法二元组都在区间 $[l, r]$ 内, 这些合法二元组的分数和最高是多少。

即选出一些合法二元组 $(i_1, j_1), (i_2, j_2), \dots, (i_k, j_k)$, 满足这些选出的合法二元组两两不同, 且分数和尽可能大。

两个合法二元组 (i_1, j_1) 与 (i_2, j_2) 不同, 当且仅当 $i_1 \neq i_2$ 或 $j_1 \neq j_2$ 。

换句话说, 两个合法二元组 (i_1, j_1) 与 (i_2, j_2) 相同, 等价于 $i_1 = i_2$ 且 $j_1 = j_2$ 。

对于 100% 的数据, 满足 $1 \leq n, m \leq 10^5$, $1 \leq l \leq r \leq n$, 所有询问的 k 的和不超过 3×10^5 , $-10^6 \leq a_i, x \leq 10^6$ 。

数据保证区间 $[l, r]$ 中一定可以找到 k 个合法二元组。

Solution1

- 还记得序列上查询前 k 大的 $a[i]-a[j]$, 满足 $i < j$ 怎么做吗?

P1631

序列合并

题目+题解

Solution1

- 我们先用一棵线段树支持区间查询。
- 线段树合并两个区间时, 需要考虑跨过中线的贡献, 即左边选一个 $a[i]$, 右边选一个 $a[j]$ 的贡献。
- 这种情况下就是左边有一些数 $A[i]=a[i]$, 右边有一些数 $B[j]=-a[j]$, 左边的数 $A[i]$ 和右边的数 $B[j]$ 两两匹配后权值为 $A[i]+B[j]$, 求前 k 大。
- 和洛谷 P1631 序列合并 一样。

Solution1

- 将左右的数从大到小排序，对左边的每个数 $A[i]$ ，维护目前和右边的哪个最大的数 $B[j]$ 匹配。
- 维护一个全局的堆，存了目前所有最大的 $(A[i]+B[j])$ 。
- 我们还记得持久化可并堆是 $O(1)$ 合并的吗？
- 利用这个性质，我们可以把线段树区间询问的 $O(\log n)$ 个区间的持久化可并堆直接合并起来。
- 每次从全局堆中找到最大的 $(A[i]+B[j])$ ，因为线段树分治时一定满足 $i < j$ ，所以正确性有保证。

Solution1

- 全局堆找出答案后需要删除。
- 则我们递归找到对应的 i 和 j 后，在 i 的堆中删除最大值，接下来 i 需要找到删掉最大值后，与其匹配的新的最大值。
- 直接给 i 一个持久化可并堆，反正是 $O(1)$ 的。
- 一开始持久化可并堆需要包含所有 $B[j]$ ，直接复制一个右区间的堆过来，反正是 $O(1)$ 的。
- 删除元素后找到新的最大值是 $O(\log n)$ 的，但对 k 次操作，只进行 k 次。
- 于是总时间复杂度 $O(n+(m+k)\log n)$

Solution2

- 可以用矩形 $[l1, r1] * [l2, r2]$ 表示从 $[l1, r1]$ 中选一个点 i ，从 $[l2, r2]$ 中选一个点 j ，对应的最大 $a[i]-a[j]$ 。
- 每次取出这个点之后，矩形需要拆成常数个新的小的范围。
- 这里我们发现两种范围方便维护：
 1. $[l, r] * [l, r]$
 2. $[l1, r1] * [l2, r2]$ 满足 $r1 < l2$
- 如果两个区间有重合会不方便，因为需要注意 $i < j$ 的限制。
- 可以证明每次只会拆出 $O(1)$ 个新的小的范围。
- 于是总时间复杂度 $O(n+(m+k)\log n)$ 。

持久化可并堆无用论

- 我们花式利用可以 $O(1)$ 合并的数据结构做了两个题，实际上我们能用数据结构套持久化可并堆解决题目都可以在原本的数据结构上维护。
- 持久化可并堆可以 $O(1)$ 合并，所以实际上我们合并的是一个类似索引的东西，因为 pop 还是 $O(\log n)$ 的，而堆只支持查询堆顶元素，查询的信息和一个变量是一样的。
- 这种问题可以先思考如何使用持久化可并堆解决，然后再思考如何用原本的数据结构维护同样的信息来达到同样的效果。

持久化动态树

- 动态树分为自底向上和自顶向下的结构。
- 自顶向下的结构可以直接使用路径复制的方法持久化。
- 自底向上的结构需要通过结点分裂的方法持久化。

持久化动态树

- 持久化动态树的难点在于去均摊，而不在于持久化本身。
- 已知存在单次操作严格时间复杂度 $O(\log n)$ 的 Link Cut Tree，以及单次操作严格时间复杂度 $O(\log n)$ 的 Worst Case Top Tree，所以直接套用持久化技巧，即可得到持久化的动态树。
- 注意对 Link Cut Tree 直接使用持久化平衡树维护辅助树是错误的，因为还有 access 的段数的均摊。

持久化动态仙人掌

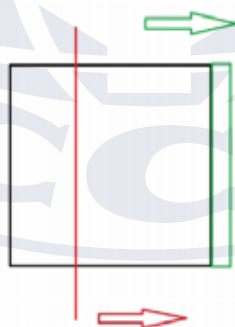
- 目前存在单次操作严格 $O(\log n)$ 的重量平衡的动态仙人掌。
- 所以我们可以得到持久化的重量平衡的动态仙人掌这样的数据结构。

随机 access

- 如果使用持久化平衡树维护辅助树，每次随机 access 一个点，是否正确呢？
- 如果是均匀随机 access 一个点，则可以通过 $O(n^{0.5})$ 次操作构造一条长度 $O(n^{0.5})$ 的路径，有常数概率这段路径没有被 access。
- 这样可以构造数据让随机 access 失效。
- 正确（卡不掉）的随机 access 需要根据树的形态来选择 access 的点。

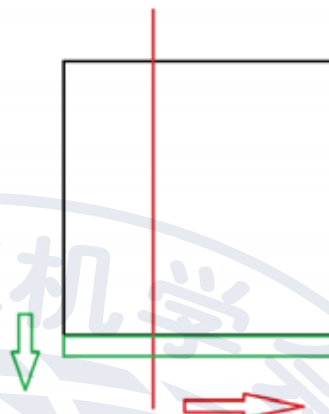
扫描线与持久化

- 一般来说扫描线的问题都可以直接对扫描线的过程部分持久化。
- 实际上 OI 里面的强制在线很多时候都是弱的，比如先给一个序列，然后强制在线进行询问，这种情况下问题只是部分持久化。
- 强一些的问题会让序列向一侧加点，则我们维护一个向着加点一侧扫的扫描线即可。



扫描线与持久化

- 如果向扫描线方向垂直的方向加点会怎么样？



洛谷P8525 [Ynoi2078] 《A Path Towards Autonomous Machine Intelligence》阅读报告

你需要支持 m 次操作，第 i 次操作给出 $L_i, R_i, a_i, b_i, l_i, r_i, X_i, x_i$,

相应的答案定义为：

$$(\text{ans}_{i,0}, \text{ans}_{i,1}) := F_{i,r_i}(F_{i,r_i-1}(\dots F_{i,l_i}(x_i, 0) \dots)),$$

其中

若 $L_j \leq X_i \leq R_j$ 则 $F_{i,j}(x, y) = ((a_j x + b_j) \bmod 2677114440, \max(b_j, y))$,

其余情况 $F_{i,j}(x, y) = (x, y)$ 。

你需要输出 $\text{ans}_{i,0}$ 与 $\text{ans}_{i,1}$ 的异或和。

输入格式

第一行两个整数 n, m 。

接下来 m 行，第 i 行为 $L_i, R_i, a_i, b_i, l_i, r_i, X_i, x_i$ 表示第 i 次操作。

本题强制在线，对 $2 \leq i \leq m$ ，第 i 次操作的输入的每个数需要与第 $i-1$ 次操作的答案异或。

对于 100% 的数据，满足：

$$1 \leq n \leq 10^6$$

$$1 \leq m \leq 3 \times 10^5$$

对于 $1 \leq i \leq m$ ：

$$1 \leq L_i \leq R_i \leq n$$

$$1 \leq a_i \leq 10^6$$

$$1 \leq b_i \leq 10^6$$

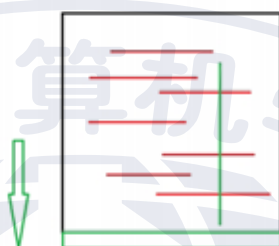
$$1 \leq l_i \leq r_i \leq i$$

$$1 \leq X_i \leq n$$

$$1 \leq x_i \leq 10^6$$

Solution

- 问题可以看做是：
- 进行 m 次操作，进行第 i 次操作时有以下两种可能的操作：
- 1. 向区间 $[l, r]$ 上加入一个新的修改操作（可以看作是线段树的可合并标记），时间为 i 。
- 2. 求 x 位置的信息经过时间区间 $[l, r]$ 的修改操作影响后的值。



Solution1

- 修改操作是区间 `push_back` 标记，可以使用线段树套持久化平衡树的方法。
- 每个结点维护标记的持久化平衡树，查询的时候询问一个点到根路径上的持久化平衡树拼起来后的一个区间信息。
- 总时间复杂度 $O(n + m \log^2 n)$

Solution2

题意：

序列，每个位置是一个序列，初始为空。

支持操作：区间内每个序列末尾添加一个元素。查询单点在一个时间区间中添加的元素的信息合并，信息满足结合律。

要求在线。

离线算法可以对外层序列用扫描线，线段树维护当前扫到的位置，每个时刻的信息，可以支持时间区间查询，时间复杂度是 $O(m \log m)$ 的。

这个算法可以将线段树换成可持久化线段树，从而支持在线处理。

但由于在线的限制，可持久化线段树必须按在线的顺序构建，每一时刻对所有版本新增一个叶子，而不是离线时每扫一步新增一个版本。

查询对应某个版本的线段树区间查询，可以用分散层叠维护可持久化线段树当前已建立的最右链（类似二进制分组的结构）上的版本序号，从而可以在 $O(\log m)$ 时间算出查询的版本在最右链上每个节点对应的具体的版本，于是可以在 $O(\log m)$ 时间进行一次普通的线段树区间查询得到答案。

修改操作对应的代价也在均摊 $O(\log m)$ 。

总时间复杂度 $O(m \log m)$ ，空间复杂度 $O(m \log m)$ 。

扫描线垂直加点

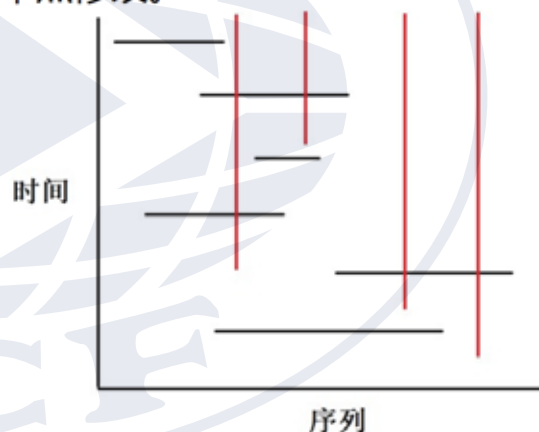
- 很多情况下扫描线需要换维扫，不沿着题目预定的时间轴扫更好做。
- 这种方法一般是扫描线扫序列，数据结构维护时间轴。
- 上面的这个算法是一类换维扫描线问题的在线解。
- 由于扫描线有很多种问题形式，所以还有很多其他类型的换维扫描线问题目前没有在线解。
- 这个思路也很有意思，我们思考的时候先换维扫描线将问题离线解决，然后通过持久化技巧将问题在线化，得到在线算法。

LOJ3489 JOISC2021 饮食区

- 有一个长为 n 的序列，序列每个位置有个队列
- 有 m 个操作
- 每个操作形如 $[l, r]$ 的每个队列中进来了 k 个 $\text{type}=c$ 的人
- 或者 $[l, r]$ 的每个队列中出去了 k 个人（不足 k 个则全部出去）
- 还有查询某个队列中第 k 个人的 type （不足 k 个输出 0）

Solution

- 考虑离线，扫描线扫序列维，数据结构维护时间维。
- 离线后，一次区间修改等价于两次单点修改。
- 如图黑色是修改红色是询问。
- 直接按照时间顺序扫描线。
- 是区间修查单点。
- 扫描线扫序列数据结构维护时间。
- 是单点修查区间。



Solution

- 假设有一次询问是在 i 时刻询问序列 $x[i]$ 位置队列的第 $k[i]$ 个元素。
- 我们假设区间插入 t 个数是单点 $+t$ ，区间弹出 t 个数是单点 $-t$ 。
- 需要维护的是时间维上每次单点修改，当扫描线跑到序列的 $x[i]$ 位置时，维护 i 时刻前面最近的一次队列空的时刻，假设为 $f[i]$ 时刻现在我们开在时间维上的线段树需要维护的是：
- 1. 单点修改
- 2. 查询一个点向左走，第一个位置 ans ，满足 ans 时刻队列空了

Solution

- 假设线段树维护的是序列 $b[i]$ ，每个位置代表其后缀和。
- 每次单点修改即前缀的 $b[i]$ 加上一个数。
- 查询即查询一个点左边后缀和最大的位置。
- 线段树维护区间 $b[i]$ 的 \max ，然后用一种特殊的方法二分。



Solution

- 定义队列空：pop 的时候没东西 pop 了。
- 假设最大后缀和位置为 x 。
- 先证在这里队列空了：
- 因为是后缀最大的和，所以这之前队列一定是空的，如果不是空的，那前面有一个区间 $[i, x]$ 和 > 0 ，我们可以再选前面的一些位置
- 再证这之后队列不空：
- 假设 y 时刻队列空了， $x < y$ ，则 $[x, y]$ 区间的和 < 0 ，这个 $[x, y]$ 区间是 $[x, n]$ 这个后缀的前缀， $[x, n]$ 做为最大后缀和不应该有一个负数前缀。

Solution

- 找到了这个最近一次队列空的 $t[i]$ 时刻时，这个区间中所有 pop 都一定 pop 到了元素。
- 于是我们求区间总 pop 量，然后找到 push 量达到 pop 量的那个位置就是我们的答案了。
- 总时间复杂度 $O((n+m)\log n)$ 。

Solution

- 可以发现这个问题形式符合我们扫描线垂直加点的算法。
- 于是利用这个算法可以将这个题同时复杂度强制在线。

总结

- 今天我讲了一些信息学竞赛中对持久化数据结构与相关技巧，希望可以抛砖引玉，引起同学们对持久化数据结构的兴趣，一起研究出更强的数据结构。

OI 在 TCS 中的应用

清华大学 许庭强

今天讲什么？

- OI: Olympiad in Informatics
- TCS: Theoretical Computer Science
- 叠甲：本人其实也不会什么东西。
- 目的：让大家觉得 TCS 并没有那么困难。

许庭强

OI 在 TCS 中的应用

2024.1

2 / 65

广为人知题

- 相信很多同学都对下面这道题很熟悉了：

问题 (广为人知题^{a)})

^a<https://uoj.ac/problem/697>

给定长度为 n 的字符串 s 。给定 d 个 s 的子串作为模式串。 q 次查询，每次给出 ql_i, qr_i ，求所有模式串在 $s[ql_i, qr_i]$ 中的出现次数之和。

- 本题可以用基本子串结构在 $O(n + d)$ 预处理转化为二维数点问题。
- 复杂度 $O(n + (d + q) \log n)$ 。

许庭强

OI 在 TCS 中的应用

2024.1

3 / 65

广为人知题 2

- 略微修改一下题意：

问题

广为人知题 2 给定长度为 n 的字符串 s 。给定 d 个 s 的子串作为模式串。 q 次查询，每次给出 ql_i, qr_i ，求有多少模式串在 $s[ql_i, qr_i]$ 中出现过。

- 当模式串集合为所有串时，题目变为询问区间本质不同子串个数。
- 当模式串集合为所有平方串时，题目变为「ZJOI2020」字符串。

许庭强

OI 在 TCS 中的应用

2024.1

4 / 65

广为人知题 2

- 类似区间本质不同子串个数，我们给所有串 t 赋一个权值 v_t 。我们直接将所有模式串对应的 v_t 变为 1，其余为 0。
- 查询的答案即为查询串 $s[l, r]$ 所有本质不同子串中 v 的和。
- 考虑 SAM 对应的那棵未压缩过的 trie。
- 记 f_u 为 u 到根对应的所有串的 v 之和。答案即为：

$$\sum_{u \text{ 是叶子}} f_u - \sum_u (u \text{ 儿子个数} - 1) f_u$$

许庭强

OI 在 TCS 中的应用

2024.1

5 / 65

广为人知题 2

- 先考虑第二项怎么求.
- 按照 r 扫描线, 维护 SAM 上连续上一次出现位置相同的节点.
- 每次 access 遇到一个连续段时, 仅有段尾的点可能儿子 +1 造成贡献.
- 这需要这个儿子未出现过且段尾原来有至少一个别的儿子. 一个关于 l 的区间加.
- 查询即为单点查询. 复杂度 $O(n \log^2 n + q \log n)$.

许庭强

OI 在 TCS 中的应用

2024.1

6 / 65

广为人知题 2

- 再考虑第一项. 首先叶子一定是 $s[l, r]$ 的前缀.
- 并且一定存在一个 x 使得 $s[l, t]$ 是叶子当且仅当 $t \geq x$.
- 考虑如何找到该 x .
- 即为找到最小的 x 使得 $s[l, x]$ 的下一次出现位置右端点 $> r$.
- 按照 l 从右往左扫描线, 在反串 SAM 上维护上一次出现位置. 每次对所有重链 lowerbound 即可.

许庭强

OI 在 TCS 中的应用

2024.1

7 / 65

广为人知题 2

- 第一项变为

$$\sum_{i=x}^r f_s[l, i] = \sum_{i=l}^r f_s[l, i] - \sum_{i=l}^{x-1} f_s[l, i]$$

- 两部分分别等价于出现次数之和. 把广为人知题当作黑盒容易在 $O((n+q)\log n)$ 时间内解决.
- 加上第二部分, 总复杂度为 $O(n\log^2 n + d\log n + q\log n)$.

许庭强

OI 在 TCS 中的应用

2024.1

8 / 65

广为人知题 2

- 凑巧的是, 学术界真的有人研究过这个问题! [CKMRRSWZ20] [CKMRRW21]
- 不幸的是, 他们的复杂度不太优秀.

Operation	Our Results		Previous Results	
	Preprocessing	Query time	Preprocessing	Query time
EXISTS	$O(n+d)$	$O(1)$	$O(n+d)$	$O(1)$
REPORT	$O(n+d)$	$O(1+x)^1$	$O(n+d)$	$O(1+x)^1$
COUNT	$O(n+d\sqrt{\log n})$	$O(\frac{\log n}{\log \log n})$	$O(\frac{n \log n}{\log \log n} + d \log^{3/2} n)$	$O(\frac{\log^2 n}{\log \log n})$
REPORTDISTINCT	$O(n \log n + d)$	$O(1+x)^1$	$O(n \log n + d)$	$O(\log n + x)^1$
COUNTDISTINCT	$O(n \log^2 n + d\sqrt{\log n})$	$O(\log n)$	$\tilde{O}(n^2/m + d)$	$\tilde{O}(m)$
			$\tilde{O}(nd/m + d)$	$\tilde{O}(m)$
			$O(n \log^{1-\epsilon} n + d \log^{3/2} n)^2$	$O(\frac{\log^3 n}{\log \log n})^2$

¹ x denotes the length of output.

² 2-approximate

许庭强

OI 在 TCS 中的应用

2024.1

9 / 65

APSP(All Pairs Shortest Path)

- 给定一张图, 求两两之间最短路.
- $O(n^3)/O(mn + n^2 \log n)$
- 无向无权图: $O(n^\omega \log n)$.
- 任意边权: 等价于 $(\min, +)$ 矩阵乘法. $O(n^3/2^{\Omega(\sqrt{\log n})})$.

许庭强

OI 在 TCS 中的应用

2024.1

10 / 65

APSP(All Pairs Shortest Path)

- 我们讲一下无向无权图的 $O(n^\omega \log n)$.

定理

可以在 $O(n^\omega \log n)$ 时间内求出无向无权图的 APSP.

- 对于邻接矩阵 A , 如果 A 除对角线外均为 1, 那么自然两两最短路均为 1.
- 否则计算 $B' = A^2$. 令 B 为矩阵使得 $B_{ij} = [B'_{ij} > 0]$. 递归计算邻接矩阵 B 的 APSP.
- 令 $\delta_A(u, v)$ 表示邻接矩阵 A 下 u, v 间的最短路长度. 有

$$2\delta_B(u, v) - 1 \leq \delta_A(u, v) \leq 2\delta_B(u, v)$$

许庭强

OI 在 TCS 中的应用

2024.1

11 / 65

APSP(All Pairs Shortest Path)

- 需要区分是 $2\delta_B(u, v) - 1$ 还是 $2\delta_B(u, v)$.
- 计算 $\sum_{(u,w) \in E} \delta_B(w, v)$.
- 如果 $\delta_A(u, v) = 2\delta_B(u, v) - 1$, 那么所有 $\delta_B(w, v) \leq \delta_B(u, v)$ 且至少一个 $\delta_B(w, v) < \delta_B(u, v)$.
- 如果 $\delta_A(u, v) = 2\delta_B(u, v)$, 那么所有 $\delta_B(w, v) \geq \delta_B(u, v)$.
- 因此只需对所有 u, v 计算 $\sum_{(u,w) \in E} \delta_B(w, v)$. 而这可以用另外一次矩阵乘法解决.

许庭强

OI 在 TCS 中的应用

2024.1

12 / 65

APSP(All Pairs Shortest Path)

- 看起来没什么改进的空间了.
- 那么, 求近似解呢?
- 考虑无向无权图. 令 $\delta(u, v)$ 为 u, v 之间的最短路.
- 如果 $\delta(u, v) \leq d(u, v) \leq k\delta(u, v)$, 那么称该解为 stretch k 的.
- 如果 $\delta(u, v) \leq d(u, v) \leq \delta(u, v) + k$, 那么称该解为 surplus k 的.

许庭强

OI 在 TCS 中的应用

2024.1

13 / 65

APASP(All Pairs Almost Shortest Path)

- 先看 surplus k .
- $k = 1$: 等价于矩阵乘法.
- 直接构造三层的分层图, 第一部分向第二部分连若干边, 第二部分向第三部分连若干边.
- 查询第一层的一个点和第三层的一个点最短距离是否为 < 4 就能规约到布尔矩阵乘法.

许庭强

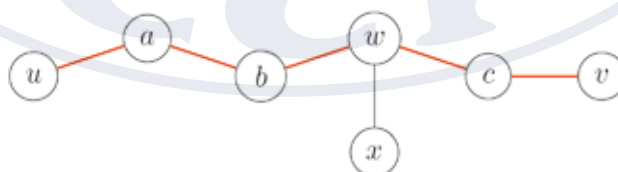
OI 在 TCS 中的应用

2024.1

14 / 65

APASP(All Pairs Almost Shortest Path)

- $k = 2$.



- 有 $\delta(u, v) \leq \delta(u, x) + \delta(x, v) \leq \delta(u, v) + 2$.
- 我们希望利用这种性质求出 surplus 2 近似解.

许庭强

OI 在 TCS 中的应用

2024.1

15 / 65

APASP(All Pairs Almost Shortest Path)

- 设定常数 B . 令 V_1 为所有 $\deg(v) > B$ 的点 v 构成的集合.
- 找到点集 D_1 使得对任意 $v \in V_1$ 存在一个点 $u \in D_1$ 使得 u, v 相邻.
- D_1 的大小不超过 $\tilde{O}(n/B)$. 这是因为如果我们随机一个 $5n/B \log n$ 大小的集合, 对于一个 $v \in V_1$, 不存在 $u \in D_1$ 与 v 相邻的概率不超过 $(1 - B/n)^{5n/B \log n} \leq e^{-B/n \cdot 5n/B \log n} \leq n^{-5}$.
- 根据 Union Bound ($P(A \cup B) \leq P(A) + P(B)$), 至少一个 $v \in V_1$ 使得不存在 $u \in D_1$ 与 v 相邻的概率不超过 n^{-4} .

许庭强

OI 在 TCS 中的应用

2024.1

16 / 65

APASP(All Pairs Almost Shortest Path)

- 从 D_1 内的所有点开始 bfs, 找到其到所有其他点之间的最短路. 该部分复杂度 $O(nm/B)$.
- 令 E_1 为所有满足 $\deg(u) \leq B$ 或 $\deg(v) \leq B$ 的 (u, v) 的集合. 对每个 $v \in V_1$, 找到其对应的一个 $u \in D_1, (u, v) \in E$, 将这些 (u, v) 放入 E_2 .
- 对每个点 $v \in V$, 定义新的带权无向图 $G' = (V, E')$, 其中 $E' = E_1 \cup E_2 \cup (D_1 \times v)$. E_1 与 E_2 的权值仍为 1, $D_1 \times v$ 的权值即为对应点到 v 间的最短路距离.
- 在新图上跑 dijkstra 即为最后与 v 相关的解. 复杂度为 $\tilde{O}(n^2 B)$.
- 取 $B = (m/n)^{1/2}$, 复杂度为 $\tilde{O}(n^{3/2} m^{1/2})$.

许庭强

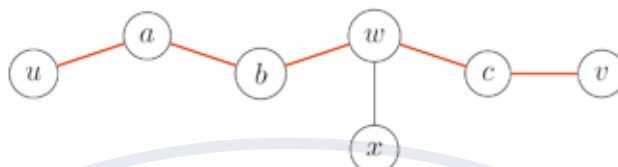
OI 在 TCS 中的应用

2024.1

17 / 65

APASP(All Pairs Almost Shortest Path)

- 证明一下解是 surplus 2 的.



- 如果路径上所有点度数均不超过 B , 那么 E_1 中的边直接给出最短路.
- 如果路径上存在点度数超过 B , 考虑 u 到 v 最短路中最后一次出现的度数超过 B 的点 w . $w \in V_1$, 那么存在点 $x \in D_1$ 与 w 相邻. u, x 间的最短路在 $D_1 \times u$ 中出现, $(x, w) \in E_2$, w 到 v 的剩余部分在 E_1 中出现. 得到的答案不超过 $\delta(u, x) + 1 + \delta(w, v)$.

许庭强

OI 在 TCS 中的应用

2024.1

18 / 65

APASP(All Pairs Almost Shortest Path)

- 刚才的做法在 m 越小时越好. 如果 m 太大了怎么办?
- 选一个点集 D' 覆盖所有度数 $\geq B'$ 的邻域, 同样 bfs 计算这些点到所有其他点的距离. 对所有 (u, v) 计算 $\min_{w \in D'} (\delta(u, w) + \delta(w, v))$. 这样就处理了路径上有度数 $> B'$ 的情况. 复杂度 $O(n^2 |D'|) = O(n^3 / B')$.
- 路径上没有度数 $> B'$ 就只需保留两端度数 $\leq B'$ 的边. 剩余边数 m' 不超过 $O(nB')$.
- 结合一下前面的做法, 令 $B' = n^{2/3}$, 复杂度即为 $\tilde{O}(n^3 / B' + n^2 B'^{1/2}) = \tilde{O}(n^{7/3})$.
- 很古老的结果. [DHZ96]

许庭强

OI 在 TCS 中的应用

2024.1

19 / 65

APASP(All Pairs Almost Shortest Path)

- 最优了吗?
- 直到 2022 年. [DKRWZ22] [Dur23]
- 注意到最后关于 B' 的部分, 有两个瓶颈. 第一个是要对 D' 中的所有点跑 bfs, 复杂度 $O(n^3/B')$; 第二个是对所有 (u, v) 枚举 $w \in D'$ 计算 $\delta(u, w) + \delta(w, v)$, 复杂度同样为 $O(n^3/B')$.

许庭强

OI 在 TCS 中的应用

2024.1

20 / 65

APASP(All Pairs Almost Shortest Path)

- 先考虑第一部分.
- 抛开选支配集的步骤, 我们一个一个点删除. 每次删除度数最大的点. 假设为 x . 那么对于所有 u, v , 如果 u, v 最短路中的一个点和 x 相邻, 用 $\delta(u, x) + \delta(x, v)$ 即可满足条件. 因此我们以 x 为起点跑 bfs 后, 可以把 x 以及 x 的邻居全部删除. 不断做该过程直到删除了 $O(n/B')$ 个点.
- 因为最后图没有删光, 所有点的度数一定不超过 B' , 总边数仍然不超过 $O(nB')$, 可以使用 $O(n^{3/2}m^{1/2})$ 的算法.
- bfs 的复杂度为 $O(m)$, 而每次删除至少删除了 $O(m/n)$ 个点. 因此 bfs 的复杂度总和不超过 $O(n^2)$.

许庭强

OI 在 TCS 中的应用

2024.1

21 / 65

APASP(All Pairs Almost Shortest Path)

- 还差第二部分, 对所有 (u, v) 枚举 $w \in D'$ 计算 $\delta(u, w) + \delta(w, v)$.
- 注意到这是一个 $n \times n/B'$ 大小矩阵与一个 $n/B' \times n$ 大小矩阵的 $(\min, +)$ 矩阵乘法.

定理 ([Dur23])

对于一个 $n \times n^\beta$ 大小的矩阵 A 与另一个 $n \times n^\beta$ 大小的每行内单调递增的矩阵 B , 其中所有权值是不超过 $O(n^\mu)$ 的非负整数, A 与 B 的 $(\min, +)$ 矩阵乘法可以在 $\tilde{O}(n^{\frac{1+\beta+\mu+\omega(\beta)}{2}})$ 时间内计算.

- 随便选一棵生成树找到一个欧拉序, 再给第二个矩阵的第 i 列整体 $+i$ 即可变为每行单调递增.
- 取 $B' = O(n^{0.5185})$, 整体复杂度变为 $\tilde{O}(n^{2.2593})$.

许庭强

OI 在 TCS 中的应用

2024.1

22 / 65

APASP(All Pairs Almost Shortest Path)

- 考虑 k 更大的情况.
- 和 $k = 2$ 的 $O(n^{3/2}m^{1/2})$ 做法类似.
- 设参数 $B_1 > B_2 > \dots > B_t$. 令 V_i 为所有 $\deg(v) > B_i$ 的点 v 构成的集合.
- 同样令 D_i 为 V_i 每个点邻域的 hitting set.
- 从小到大考虑所有 D_i . 每个阶段的目标是找到 D_i 中的点到所有其他点的 surplus $2(i-1)$ 近似.
- 类似的, 在 D_i 到其他点的最短路径中, 按照路径上是否存在一个在 V_{i-1} 的点分类.
- 那么在计算 $u \in D_i$ 到其他点的答案时, 就要保留与度数 $\leq B_{i-1}$ 的点相邻的所有边, 再加上 u 到 D_{i-1} 中所有点形成的带权边跑 dijkstra.

许庭强

OI 在 TCS 中的应用

2024.1

23 / 65

APASP(All Pairs Almost Shortest Path)

- 这样, 如果路径上没有 V_{i-1} 中的点, 由于保留了与 $\deg \leq B_{i-1}$ 的点相邻的所有边, 能够直接找到答案.



- 如果有 V_{i-1} 中的点, 那么由于我们刚才已经求出了 D_{i-1} 中的点到 u 的 surplus $2(i-2)$ 最短路长度, 从 u 出发求出的最短路长度就不超过 $2(i-1)$.
- 对于 V_1 我们需要直接在整张图上跑 bfs. 在最后需要令 $D_{t+1} = V$ 再做一次.

许庭强

OI 在 TCS 中的应用

2024.1

24 / 65

APASP(All Pairs Almost Shortest Path)

- 我们现在来分析一下复杂度.
- $|D_i| = \tilde{O}(n/B_i)$. 保留的边数不超过 $O(nB_{i-1})$. dijkstra 复杂度 $\tilde{O}(n^2B_{i-1}/B_i)$.
- 计算 D_1 需要原图上 bfs, 复杂度 $\tilde{O}(mn/B_1)$.
- 计算 D_{t+1} 需要对所有点求答案, 复杂度 $\tilde{O}(n^2B_t)$.
- 简单平衡一下, 可以得到 $B_i = O((m/n)^{(t+1-i)/(t+1)})$. 总复杂度 $\tilde{O}(n^{2-1/(t+1)}m^{1/(t+1)})$.
- 为 surplus $2t$ 近似.

许庭强

OI 在 TCS 中的应用

2024.1

25 / 65

APASP(All Pairs Almost Shortest Path)

- 现在回到 stretch k .
- 介绍一个 $\delta(u, v) \leq d(u, v) \leq 2\delta(u, v) + 1$ 的做法.
- 还是类似刚才用到的思想, 对一个阈值 B , 找到度数 $> B$ 的所有点邻域的 hitting set D . 同样计算 D 到所有其他点的距离.
- 但此时我们计算 $d(u, v)$ 时, 直接找到距离 u 最近的 D 中的点和距离 v 最近的 D 中的点计算贡献.
- 直观上来看, 如果最短路上有一个度数 $> B$ 的点, 那么其邻域内那个 D 中的点与 u, v 之间的距离一定有一个不超过一个差不多 $\delta(u, v)/2$ 的数字.
- 这样我们算出的 $\delta(u, x) + \delta(x, v)$ 也大致不超过 $2\delta(u, v)$.

许庭强

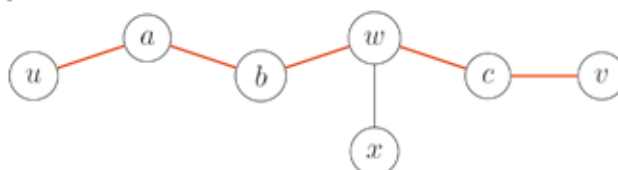
OI 在 TCS 中的应用

2024.1

26 / 65

APASP(All Pairs Almost Shortest Path)

- 仔细分析一下.
- 不妨设 $n = 2^k$. 令 $B_i = 2^{k-i}, 0 \leq i \leq k$.
- 对 $1 \leq i \leq k$, 同样找到 D_i , 并保留 $\deg(u) \leq B_{i-1}$ 周围的边对所有 D_i 内点跑 bfs.
- 对任意 u, v , 考虑其最短路上度数最大或第二大的点. 设其度数在 $(B_t, B_{t+1}]$ 内.



- 那么对 D_t 做 bfs 那张图包含了 u, v 间的最短路. 计算 $\delta(u, x) + \delta(x, v)$ 即可. 其中 x 是 D_t 中距离 u 或 v 最近的点.

许庭强

OI 在 TCS 中的应用

2024.1

27 / 65

APASP(All Pairs Almost Shortest Path)

- 最坏情况是最短路长度为奇数 $2d + 1$, 且中间两个点为度数前两大点. 此时 $\delta(u, x)$ 可能达到 $d + 1$.
- 那么 $\delta(u, x) + \delta(x, v)$ 可能达到

$$(d + 1) + (d + 1) + (2d + 1) = 4d + 3 = 2(2d + 1) + 1$$

- 因此该算法能够达到 $\delta(u, v) \leq d(u, v) \leq 2\delta(u, v) + 1$.

许庭强

OI 在 TCS 中的应用

2024.1

28 / 65

APASP(All Pairs Almost Shortest Path)

- 考虑复杂度. 类似的, 对 D_i bfs 时, 有 $\tilde{O}(n/B_i)$ 个点, 边数为 nB_{i-1} . 复杂度 $O(n^2 B_{i-1}/B_i)$.
- 对每个点 u 计算 D_i 中距离 u 最近的点是简单的. 对每对 u, v 需要枚举所有的 $1 \leq i \leq k$, 复杂度 $\tilde{O}(1)$.
- 由于 $B_{i-1}/B_i = O(1)$, 总复杂度 $\tilde{O}(n^2)$.
- 在指数上达到 optimal.

许庭强

OI 在 TCS 中的应用

2024.1

29 / 65

APASP(All Pairs Almost Shortest Path)

- 这个做法同时还是 $\tilde{O}(n^2)$ 复杂度的 stretch $7/3$.
- 难点在于距离小的点对. 尤其是 $\delta(u, v) = 3$ 的那些.
- 其实很接近 stretch 2 了, 但现在 stretch 2 还只会 $\tilde{O}(n^{2.032})$.
[DFKNWV23]
- 同时和 surplus 1 时的理由相同, $k < 2$ 的 stretch 近似也等价于矩阵乘法.
- 很接近 optimal 了对吧?
- 可以由你们来终结这个问题.

许庭强

OI 在 TCS 中的应用

2024.1

30 / 65

Unbounded SubsetSum

问题 (Unbounded SubsetSum)

给定 w_1, w_2, \dots, w_n . 保证 $1 \leq w_i \leq u$. 对 $c \in [0, t]$ 求出是否存在 $m_1, m_2, \dots, m_n \in \mathbb{N}$ 使得 $\sum_{i=1}^n m_i w_i = c$.

- 很简单, 对吧?
- 多项式求逆可以直接做到 $\tilde{O}(u + t)$.

许庭强

OI 在 TCS 中的应用

2024.1

31 / 65

Related Problems

问题 (Unbounded Knapsack)

给定 $(w_1, v_1), (w_2, v_2), \dots, (w_n, v_n)$. 保证 $1 \leq w_i \leq u$. 对 $c \in [0, t]$ 求出所有 $m_1, m_2, \dots, m_n \in \mathbb{N}$ 使得 $\sum_{i=1}^n m_i w_i = c$ 的方案中, $\sum_{i=1}^n m_i v_i$ 的最大值. 如果不存在这样的方案, 认为最大值为 $-\infty$.

问题 (CoinChange)

给定 w_1, w_2, \dots, w_n . 保证 $1 \leq w_i \leq u$. 对 $c \in [0, t]$ 求出所有 $m_1, m_2, \dots, m_n \in \mathbb{N}$ 使得 $\sum_{i=1}^n m_i w_i = c$ 的方案中, $\sum_{i=1}^n m_i$ 的最小值. 如果不存在这样的方案, 认为最小值为 ∞ .

问题 (Residue Table)

给定 w_1, w_2, \dots, w_n . 保证 $1 \leq w_i \leq u$. 对 $c \in [0, w_1)$ 求出最小的 c' 使得 $c' \equiv c \pmod{w_1}$ 且存在 $m_1, m_2, \dots, m_n \in \mathbb{N}$ 使得 $\sum_{i=1}^n m_i w_i = c'$.

许庭强

OI 在 TCS 中的应用

2024.1

32 / 65

下界

- 我们希望能够在尽量短的时间内解决问题. 复杂度一定会包含 u, t 的多项式项.
- 首先对于第一个问题——背包问题, 其不弱于 $(\min, +)$ 卷积.
- 对于 a_1, a_2, \dots, a_n 和 b_1, b_2, \dots, b_n , 我们可以构造 $2n$ 个物品, $(4n + i, a_i)$ 和 $(2n + i, b_i)$. 这样 $c = 6n + s$ 的答案即为 $(\min, +)$ 卷积在 s 处的答案.
- 那么我们就有了一个简单的下界.
- Unbounded Knapsack: $\tilde{O}(T(u) + t)$.
- CoinChange: $\tilde{O}(u + t)$.
- Residue Table: $\tilde{O}(u + t)$.
- 幸运的是, 这些下界我们都能达到. [DMZ22]

许庭强

OI 在 TCS 中的应用

2024.1

33 / 65

字典序

- 定义一些符号.
- 一个解的形式为 (x_1, x_2, \dots, x_n) , 表示每个物品用了几次.
- 我们称在字典序 σ 下解 $A = (a_1, a_2, \dots, a_n)$ 的字典序小于 $B = (b_1, b_2, \dots, b_n)$ 当且仅当存在 $j \in [1, n]$ 使得 $a_{\sigma_j} > b_{\sigma_j}$ 且对任意 $k < j$ 有 $a_{\sigma_k} = b_{\sigma_k}$.
- 我们后面会用到它.

许庭强

OI 在 TCS 中的应用

2024.1

34 / 65

一个有趣的结论

- 那么我们从一个趣味而神秘的结论开始:

定理

对于 *Unbounded SubsetSum* 问题, 如果我们给定字典序 σ 和一个可能达到的子集和 c , 找到其对应的字典序最小的解 (x_1, x_2, \dots, x_n) . 那么我们有

$$\prod_{i=1}^n (x_i + 1) \leq c + 1$$

许庭强

OI 在 TCS 中的应用

2024.1

35 / 65

一个有趣的结论

- 证明并不困难.
- 考虑所有形如 (y_1, y_2, \dots, y_n) 的解, 其中 $0 \leq y_i \leq x_i$. 一共有 $\prod_{i=1}^n (x_i + 1)$ 组解.
- 如果解的数量大于 $c + 1$, 就一定存在两组解得到的子集和相同. 设为 (y_1, y_2, \dots, y_n) 和 $(y'_1, y'_2, \dots, y'_n)$.
- 考虑 $(x_i - y_i + y'_i)$ 和 $(x_i - y'_i + y_i)$ 两组解. 一定有一个字典序更小. 得证.

许庭强

OI 在 TCS 中的应用

2024.1

36 / 65

一个有趣的结论

- 类似的, 对于背包问题, 也有如下结论:

定理

对于 *Unbounded Knapsack* 问题, 如果我们给定字典序 σ 和一个可能达到的重量和 c , 找到达到最大权值和中字典序最小的解 (x_1, x_2, \dots, x_n) . 那么我们有

$$\prod_{i=1}^n (x_i + 1) \leq c + 1$$

- 需要略微修改一下证明过程, 先看 $(x_i - y_i + y'_i)$ 与 $(x_i - y'_i + y_i)$ 中是否有更大权值. 如果没有, 说明权值均相同, 找到字典序更小的即可.

许庭强

OI 在 TCS 中的应用

2024.1

37 / 65

一个有趣的结论

- 进一步, 对于 $c > u^2$, 我们至少会选一个性价比最高的物品. 找到其中字典序最小的, 不断选择直到低于 u^2 即可.
- 再根据刚才得到的结论, 可以发现, 对于字典序最小的最优解, 选择的物品种类数一定不超过 $2 \log u + 1 = O(\log u)$.
- 十分有用的性质.
- 我们记这样的解为 $\text{sol}(c, \sigma)$.

许庭强

OI 在 TCS 中的应用

2024.1

38 / 65

一些其他重要性质

- 对于 c 和 σ , 对于一个满足 $\text{sol}(c, \sigma)_i > 0$ 的 i , 我们有

$$\text{sol}(c - w_i, \sigma)_k = \begin{cases} \text{sol}(c, \sigma)_k - 1 & k = i \\ \text{sol}(c, \sigma)_k & k \neq i \end{cases}$$

- 首先背包权值和一定相同.
- 其次看字典序. 如果 $c - w_i$ 的最优解比该解字典序小, 那么该最优解加上 i 之后可以获得一个 c 的字典序更小的解. 矛盾.
- 同理, 反过来也成立.

许庭强

OI 在 TCS 中的应用

2024.1

39 / 65

算法

- 现在我们着眼于解决问题.
- 我们将算法分为两个步骤:
 - ① 找到所有 $|\text{sol}(c, \sigma)| \leq B$ 的 $\text{sol}(c, \sigma)$.
 - ② 通过这些 $\text{sol}(c, \sigma)$ 找到所有的 $\text{sol}(c, \sigma)$.
- 这里 $|\text{sol}(c, \sigma)|$ 指用到的物品个数之和.
- 取 $B = 2 \log u + 1$.

许庭强

OI 在 TCS 中的应用

2024.1

40 / 65

第二步

- 我们先考虑第二步怎么做.
- 很简单, 我们直接枚举 i 从 1 到 t , 假设当前整个前缀 $[1, i]$ 的 sol 已经计算完毕, 并尝试从 $\text{sol}(i, \sigma)$ 添加一个物品转移.
- 添加物品的方式即为枚举一个满足 $\text{sol}(i, \sigma)_k > 0$ 的物品 k .
- 由于一共只有最多 $B = 2 \log u + 1$ 种物品, 复杂度仍然正确.
- 现在考虑正确性.
- 对于那些 $|\text{sol}(c, \sigma)| > B$ 的 c , 由于其最多用到了 B 种物品, 一定至少有一种物品用了超过 1 次, 那么它就会被去掉 1 个这个物品的方案转移到, 从而算出正确答案.
- 复杂度 $O(t \log^2 u)$.

许庭强

OI 在 TCS 中的应用

2024.1

41 / 65

第二步

- 对于 CoinChange 和 Residue Table 问题也是类似的.
- CoinChange 只是 Unbounded Knapsack 的特殊情况.
- 对于 Residue Table, 我们可以用类似 dijkstra 的方式找到所有解. 每次选择一个最小的转移即可. 复杂度 $O(u \log^2 u)$.

许庭强

OI 在 TCS 中的应用

2024.1

42 / 65

第一步

- 现在我们回到第一步.
- 由于我们只需考虑总个数 $\leq B$ 的方案, 可以尝试做 B 次卷积.
- 对于 Unbounded Knapsack, 每次卷积都是一次 $(\max, +)$ 卷积, 但在卷积的同时, 我们需要找到对每个得到的贡献, 找到贡献给它的字典序最小的物品.
- 具体来说, 固定 f_i 为重量等于 i 的物品中权值的最大值. 初始令数组 s 在 0 处为 0, 其余为 $-\infty$. 我们需要做 B 次 $s = s \times f$, 此处卷积为 $(\max, +)$ 卷积.
- 每次卷积后, 对每个 s_i , 需要找到一个最小的 j 使得新的 s_i 能够由 f_j 贡献过来.

许庭强

OI 在 TCS 中的应用

2024.1

43 / 65

第一步

- 可以通过设置权值找到该最小 j . 复杂度为

$$O(BT(uB)) = O(T(u) \log^3 u)$$

- 结合第二步的算法, 我们可以在 $O(T(u) \log^3 u + t \log^2 u)$ 时间内解决 Unbounded Knapsack 问题.

许庭强

OI 在 TCS 中的应用

2024.1

44 / 65

第一步

- 还有两个问题.
- 对于 CoinChange, 由于我们的卷积次数自动表示了物品个数, 我们无需再保留权值, 只用考虑是否存在合法解.
- 变成了一个 Boolean Convolution. 可以用 FFT 轻松做到 $O(u \log^3 u)$.
- 但是, 我们应该如何找到字典序最小的转移方式呢?
- 这对于任意情况而言是困难的. 我们只有一个 $\tilde{O}(n^{1.5})$ 的做法.

许庭强

OI 在 TCS 中的应用

2024.1

45 / 65

第一步

- 但是, 我们还有一个武器.
- 我们可以自己选择 σ .
- 实际上, 随机选择 σ 就能做到足够优秀.

许庭强

OI 在 TCS 中的应用

2024.1

46 / 65

第一步

- 在所有排列中随机选择一个作为 σ .
- 形式化的, 有一个随机排列 p . 我们需要做 Boolean Convolution $C = A \times B$, 并对所有 $C_i = 1$ 的 i 找到 j 使得 $A_j = B_{i-j} = 1$ 且 p_j 最小.
- 设卷积长度为 n . 算法分为 $\log n$ 步. 第 i 步时, 我们只保留 A 中 $p_j \leq 2^i$ 的那些位置与 B 做卷积.
- 这样我们能对每个 $C_i = 1$ 找到我们想要的 p_j 在哪个 $(2^k, 2^{k+1}]$ 区间内.

许庭强

OI 在 TCS 中的应用

2024.1

47 / 65

第一步

- 我们实际上有如下结论:

定理

对任意 $C_i = 1$ 的 i , 找到能贡献给 C_i 的位置 A_j 中最小的 p_j 在哪个 $(2^k, 2^{k+1}]$ 区间内. 那么所有 $p_j \in (2^k, 2^{k+1}]$ 的 j 中能贡献给 C_i 的 j 的个数大概率不超过 $O(\log n)$.

- 证明并不困难.
- 我们可以把随机 p 看作这样一个过程.
- 每次随机将位置分成两个部分, 选择一个部分继续递归直到仅剩一个位置.

许庭强

OI 在 TCS 中的应用

2024.1

48 / 65

第一步

- 那么我们会在一个步骤停下来当且仅当所有的贡献都被分到了同一边, 且我们选择了另一边递归下去.
- 如果当前有 r 个贡献位置, 那么这个概率就大约为 2^{-r} .
- 如果 $r = c \log n$, 概率就不会超过 $O(n^{-c})$.
- 再次由 Union Bound, 我们能够使得想要的 $(2^k, 2^{k+1}]$ 中的贡献 with high probability 不超过 $O(\log n)$.

许庭强

OI 在 TCS 中的应用

2024.1

49 / 65

第一步

- 那么现在我们需要找到这 $O(\log n)$ 个贡献并选出 p_j 最小的 j .
- 一个简单的办法是, 我们直接计算所有贡献的个数, 1 次方和, 2 次方和, ..., $O(\log n)$ 次方和. 解方程找到所有贡献的位置.
- 仍然可以用 FFT 来解决.
- 算一下复杂度.
- $O(\log u)$ 次卷积, 卷积长度 $O(u \log u)$, 每次卷积需要分 $O(\log u)$ 段, 每段内部算 k 方和需要 $O(\log u)$ 次 FFT.
- 总复杂度 $O(u \log^5 u)$.
- OI 里不太 practical, 对吧?
- 但这就是 TCS.
- (尝试去点 $\log u$ 说不定真能扔 OI 里呢?)

许庭强

OI 在 TCS 中的应用

2024.1

50 / 65

最终结论

- 对于 Residue Table 也是一样, 我们不关心权值了, 我们只用对前 $O(u \log u)$ 算出大小不超过 B 的字典序最小的合法解, 然后更新出我们想要的初始值即可.
- CoinChange 问题可以在 $O(u \log^5 u + t \log^2 u)$ 复杂度内解决.
- Residue Table 问题可以在 $O(u \log^5 u)$ 复杂度内解决.

许庭强

OI 在 TCS 中的应用

2024.1

51 / 65

趣味小问题

- 讲一些看起来很简单的问题.
- 以一个趣味小问题开始:

问题

给定 n 位三进制数, 我们想要用尽量少的位数来存储, 支持单点查询.

- 明显我们能用 $\lceil n \log_2 3 \rceil$ 位来记录所有信息.
- 但这样我们无法做到快速查询. 怎么办呢?
- 我们在后面再给出这个问题的答案.

许庭强

OI 在 TCS 中的应用

2024.1

52 / 65

一个看起来更简单的问题

- 给出一个看起来更简单的问题:

问题

给定长度为 n 的 01 数组 a_0, a_1, \dots, a_{n-1} . 查询前缀和 $\sum_{i=0}^x a_i$.

- 目标仍然是用尽量少的位数来存储信息.
- 当然, 时间复杂度不能变成 $O(n)$.

许庭强

OI 在 TCS 中的应用

2024.1

53 / 65

一个看起来更简单的问题

- 直接存前缀和: $O(n \log n)$ 位.
- 分成大小为 B 的块, 仅计算分界处的前缀和. 查询时用 popcount 快速计算块内贡献.
- 空间复杂度 $n + O(n \log n / B)$, 时间复杂度 $O(B / \log n)$.
- 相当于 $O(n)$ 空间, $O(1)$ 查询.
- 还是不满意, 我们想要做到 $n + o(n)$.
- 称该类型数据结构为 Succinct Data Structure.
- 具体来说, 对于一个问题, 设 OPT 为其至少要用的位数, 算法用到的位数为 $OPT + R$. Succinct 即指 $R = o(OPT)$.
- 我们称 R 为冗余空间.

许庭强

OI 在 TCS 中的应用

2024.1

54 / 65

Word RAM Model

- 将空间卡到 bit 级别时, 我们就需要重新审视对于复杂度的定义.
- 我们在 Word RAM Model 下定义问题.
- 内存被划分为 w -bit words, 可以对 w -bit 整数 $O(1)$ 做加减乘除和位运算. 与我们平常的理解相同.
- 认为 $w = O(\log n)$.
- 在 Word RAM Model 下, 很多问题能够获得更好的复杂度.

许庭强

OI 在 TCS 中的应用

2024.1

55 / 65

一个看起来更简单的问题

- 回到前缀和, 我们给出一个更为优秀的算法.
- 将原序列分为 $B' = \log^{100} n$ 长度的块. 记录块的前缀和需要的空间仅有 $n / \log^{99} n$.
- 块内与刚才相同, 但此时空间复杂度变为 $n + O(n \log B' / B)$.
- 取 $B = \log n$, 有空间复杂度 $n + O(n \log \log n / \log n)$.

许庭强

OI 在 TCS 中的应用

2024.1

56 / 65

一个看起来更简单的问题

- 还能更好!
- 如果查询时间复杂度为 $O(t)$, 那么冗余空间大小可以达到 $O(\frac{nt^t}{\log^t n})$.
[Pat08]
- 如果查询时间复杂度为 $O(t)$, 那么冗余空间大小可以达到 $O(\frac{n}{\log^{\Omega(t)} n})$. [Yu19]
- 已经达到了问题的下界. [PV10]
- 算法有些复杂.
- 我们通过之前的 3 进制数问题来阐释 $O(\frac{nt^t}{\log^t n})$ 算法中用到的部分思想和科技.

许庭强

OI 在 TCS 中的应用

2024.1

57 / 65

趣味小问题

- 回到这里, 一个简单的想法是分块. 对于每个大小为 B 的块, 我们需要 $\lceil B \log_2 3 \rceil$ 个二进制位, 每次查询需要 $O(B/w)$ 的时间复杂度.
- 但此时每个块都可能多出一位, 导致总体多出 $O(n/B)$ 个二进制位.
- 那么我们此时就有了一个新点子: 我们能不能把多出来的这部分信息递归地继续做下去?

许庭强

OI 在 TCS 中的应用

2024.1

58 / 65

Succincter

- 对于一个集合 \mathcal{X} , 我们希望用 M 个二进制位加上大小为 K 的信息, 共 $K2^M$ 种情况来处理所有集合 \mathcal{X} 中的元素.
- 固定 $r \leq |\mathcal{X}|$. 我们想让 $r \leq K \leq 2r$.
- 那么自然的, 我们选择 M 使得 $2^M r < |\mathcal{X}| \leq 2^{M+1} r$.
- 这样, $K = \lceil \frac{|\mathcal{X}|}{2^M} \rceil$. 就有 $r \leq K \leq 2r$.
- 我们来计算下我们损失的信息量.

$$\begin{aligned} \log_2 \left(\frac{K2^M}{|\mathcal{X}|} \right) &= \log_2 \left(\frac{\lceil \frac{|\mathcal{X}|}{2^M} \rceil 2^M}{|\mathcal{X}|} \right) \\ &= \log_2 \left(1 + \frac{2^M}{|\mathcal{X}|} \right) \leq \frac{2}{r} \end{aligned}$$

许庭强

OI 在 TCS 中的应用

2024.1

59 / 65

Succincter

- 看起来还不错.
- 让我们应用到刚才的 3 进制问题上.
- 待定参数 r . 首先把序列分 w 个一块, 每个块内得到其对应的 K_0 和 M_0 .
- 递归向上, 现在有 n/w 个元素, 每个元素有一个 $[0, K_0 - 1]$ 内的值. 把序列分为 $B = \Theta(w/\log r)$ 一块, 同样得到其对应的 K_1 和 M_1 .
- 此时每个块内的种类数为 $O(K_0^B) \leq (2r)^{\Theta(w/\log r)} \leq 2^{O(w)}$.
- 再继续向上, 仍然是 $B = \Theta(w/\log r)$ 一块, 后面的 K 与 M 将会全部相同.

许庭强

OI 在 TCS 中的应用

2024.1

60 / 65

Succincter

- 重新捋一遍.
- 我们的目标是: 查询一个 3 进制数的第 i 位.
- 经过第一层, 我们的目标变为: 查询第 i 位所在的块压缩出来的所有 bit. 因其长度为 $O(w)$, 我们可以 $O(1)$ 做加减乘除操作解压.
- 那么第 i 位所在的块压缩出来的所有 bit 由两部分组成, K 与 2^M . 其中 2^M 可以直接读取, 而 K 的部分则变为下一层的内容.
- 目标此时变为: 第二层有一个 K 进制数, 查询第 i 位所在的数字.
- 同理, 只需查询第 i 位在第二层所在的块压缩出来的所有 bit. 由于 $K_0^B = 2^{O(w)}$, 仍然可以 $O(1)$ 做加减乘除操作解压.
- 可以继续将 K 的部分归到下一层, 不断递归下去.

许庭强

OI 在 TCS 中的应用

2024.1

61 / 65

Succincter

- 假设我们递归 t 层. 在最后时, 还剩余 $O(\frac{n}{B^t w})$ 个元素.
- 直接对每个元素用 $\lceil \log_2 K \rceil$ 记录下来. 对每一个元素都可能浪费 1 位.
- 现在计算总共浪费的位数.

$$R = O\left(\frac{n/w}{r} + \frac{n/Bw}{r} + \cdots + \frac{n/B^t w}{r} + \frac{n}{B^t w}\right)$$

$$= O\left(\frac{n}{wr} + \frac{n}{B^t w}\right)$$

- 取 $r = B^t$. 那么 $B = \Theta(w/\log r) = \Theta(w/(t \log B)) = \Theta(\frac{w/t}{\log(w/t)})$.
- 总空间复杂度即为 $n + O(\frac{n}{(w/t)^t})$.

许庭强

OI 在 TCS 中的应用

2024.1

62 / 65

Succincter

- 这被称为 Succincter.
- 利用 Succincter, 可以进一步解决很多其他问题, 例如刚才我们讨论过的前缀和等等.
- 一个未解决的问题是, 前缀和查询 (Rank) 还有一个与之相对应的查询: 查询第 k 小的 1 位置 (Select).
- 冗余空间下界 $O(\frac{n}{\log^{\Omega(t)} n})$ 对 Rank 与 Select 均成立. [PV10] 但仅有 Rank 达到了下界, Select 还仅停留在 Succincter 得到的 $O(\frac{nt^t}{\log^t n})$, 未被完全解决.

许庭强

OI 在 TCS 中的应用

2024.1

63 / 65



后记

- 今天想说的大约是：许多学术界前沿的问题似乎离我们并不遥远，有许多问题在 OI 中早已出现，即使是一些 OI 中并未出现的内容，有 OI 背景的学生也能够很快地理解并跟上前辈的脚步。
- 这或许是因为 TCS 的历史过短吧，又或是源于那些毕业进入科研领域的学长前辈们在 OI 发展中发挥的至关重要的作用。
- 但无论如何，希望大家都能够保持自信，无怨无悔地走到路的尽头。

题目选讲

威海实验高级中学 戚朗瑞

CF1753C

给定一个长度为 n 的 01 序列 a 和一种操作，你需要用这种操作将序列从小到大排序。

操作如下：

- 等概率随机选取两个位置 i, j ($i < j$)，若 $a_i > a_j$ ，则交换 a_i, a_j 。

注意：当 $a_i \leq a_j$ 时，交换失败，也算作一次操作。

请你求出操作被执行的期望次数。对 998244353 取模。 $1 \leq n \leq 2 \times 10^5$, $a_i \in \{0, 1\}$ 。

Solution

- 设序列中总共有 k 个 0，并且序列前 k 个位置总共有 x 个 1。
- 那么显然，序列后 $n-k$ 个位置总共有 x 个 0。
- 定义一次有效交换为：交换了一个前 k 个位置中的 1，和一个后 $n-k$ 个位置中的 0。
- 那么我们每次进行完一个有效交换， x 就会减少 1。
- 求出现第一次有效交换的期望次数，显然为 $\frac{n(n-1)/2}{x^2}$ 。
- 那么答案为 $\sum_{i=1}^x \frac{n(n-1)/2}{i^2}$ 。

DMOPC '22 Contest 2 P6 - Yogyakarta Elevators

- 有一个 n 层的楼和 m 个电梯，每个电梯停靠在一个子集楼层。
- 称一个区间 $[l, r]$ 是连通的当且仅当可以通过电梯在不经 $[l, r]$ 外的楼层从任意一个 $[l, r]$ 内的楼层到达另一个。
- 你需要找出最长连通区间。

$$1 \leq N \leq 5 \times 10^4$$

$$1 \leq M \leq 500$$

Solution

- 考虑区间 $[l, r]$ ，从大往小扫描 l 。
- 我们建出一个图，点集为每个电梯。对于每层，对应一个其所在的电梯的子图。（令边权为点的编号）
- 发现一个区间合法当且仅当其生成子图仅有一个连通块。
- 因此，我们扫描 l 的过程中，仅需要保留每个点出现的最小的 $>= l$ 的位置，以及最小生成树上的边。
- 然后再考虑保留哪些点及边，仅有 $O(m)$ 种本质不同的方案。
- 用并查集实现可得到 $O(nm \alpha(n))$ 的复杂度。

CF1753D

有一个 $n \times m$ 的网格图，场地上有许多不可移动的障碍，标记为 #，还有一些可以移动的 1×2 障碍，现在需要在这个网格图中空出一个 1×2 的空位，可以移动的障碍移动如下：

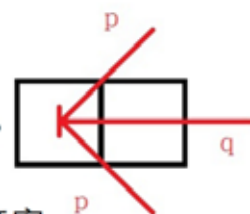
- 固定它的一格并任意方向旋转 90° ，花费 p 单位，前提是旋转后不准与其他障碍重合。
- 将其平移一个单位，花费 q 单位，前提是平移后不准与其他障碍重合。

现在要求出在这个网格图里空出 1×2 的空位，至少需要花费多少单位，如果不可能，输出 -1。

$$1 \leq n, m \leq 300000$$

Solution

- 将模型转换为把空位移动。
- 按照右侧方式连边，表示空格移动花费的代价。
- 显然，如果我们将网格黑白染色，黑白格互相独立。
- 我们初始把所有是空格的位置 $dis=0$ ，跑最短路。
- 然后找到一对相邻的点，最短路加和，取 \min 即为答案。
- 但是否有可能某条边移动是不合法的？（床的另一个端点已经被移动走了）
- 稍微分析可得知这种情况没影响，因为如果另一端已经被移走，那么不需要继续移动，已经有 1×2 的位置了。因此上述算法是正确的。



[BJOI2018]染色

- <https://www.luogu.com.cn/problem/P4429>

Solution

- 考虑什么情况会无解。
- 第一种比较显然，非二分图。找到一个奇环并选取集合 $\{A,B\}, \{B,A\}, \{A,B\}, \dots$ 即可。
- 接下来都是二分图的情况。
- 先不断剔除掉叶子，显然没有影响。
- 然后所有点 $\deg \geq 2$ 。
- 如果图不连通，拆成连通块分别判。

Solution

- 我们发现，对于一个偶环，我们可以利用它固定一个点的颜色。
- 这样，如果有两个偶简单环仅交于 ≤ 2 点或不交，直接无解。
- 仅有一个偶简单环显然有解。
- 在仍未被判解的case中，存在点 $\deg \geq 3$ ，并能从每个 $\deg \geq 3$ 的点找到两个环，交于 ≥ 3 个点，且交的部分为链，链的一端为该点。（先找到一个环，然后从这个点往外扩展，由于上述无解情况，因此最终到达的点肯定满足条件）

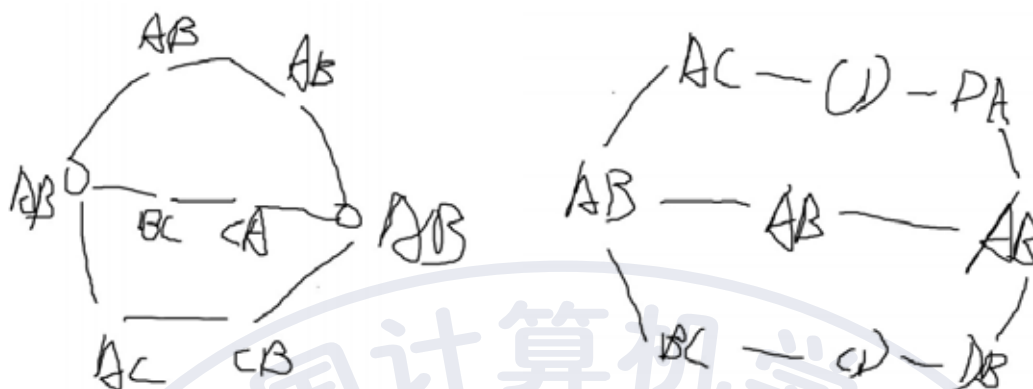
Solution

- 考虑有点 $\deg \geq 4$ 的情况。
- 扩展后该点已经连了三条边，还有一条。
- 扩展最后一条边，发现无论连向哪都无解。
- 因此若有解，所有点 $\deg \leq 3$ 。

Solution

- 继续考虑 $\deg = 3$ 的点。
- 还是使用上述方法扩展，这样可视为三条互不相交的从一个三度点到另一个三度点的路径。
- 也可以从这看出
- 由于是二分图，三条路径长度的奇偶性相同。
- 注意前面分析出的性质，三条路径长度均 ≥ 2 。
- 如果有 ≥ 2 条路径长度 > 2 ，也无解。
- 分奇数偶数讨论：

Solution



Solution

- 同时，如果有 >2 个三度点，也无解。
- （从第三个三度点开始扩展，不管到哪都无解）
- 这样无解的情况都差不多讨论完了，剩下了什么？
 - 所有点 $\deg=2$ ，环，已经说了显然有解。
 - 仅有两个三度点，并且至少两条路径长度为 2。
 - 接下来证明这种情况有解。

Solution

- 考虑先不管两条长度为 2 的路径，剩下一条链。
- 如果链上的全都是 AB，那么两端相同，剩下两个在 2 上的点也没用了。
- 否则，链上至少有三种填法，而两条 2 只能禁掉两种，因此总有解。
- 综上所述：有解当且仅当是二分图，并且每个连通块是一个环或仅有两个三度点且有两条路径长度为 2。

CF1570F

定义一个01串 s 是好的，当且仅当 s 可以通过以下操作变成全是1的串，可以操作无数次

选择 i, j 满足 $i < j, s_i = s_j = 1, 2 \sum_{k=i}^j s_k \geq j - i + 1$ ，然后将 $k \in [i, j]$ 的 s_k 全部改为1

给定 n 和 m ，求有多少个长度为 n 的01串是好的，对 m 取模

$$1 \leq n \leq 5000, 10 \leq m \leq 10^9$$

Solution

- 考虑不断对一个串进行操作，直到操作不了，或者全1。
- 对其不断操作，最终会形成若干个1的连续段，并且两两间隔 $>$ 两边连续段的长度之和。
- 我们考虑用所有串，减掉不合法的串。
- 记 $f[i]$ 表示长度为 i 的合法串个数。
- DP转移的话就用 2^i 减掉不合法的串个数即可，用一些简单的优化即可做到 $O(n^2)$ 。

ARC148D

场上 $2N$ 个整数，Alice, Bob 轮流取数，Alice 先手，如果最终 Alice 取出数的和取模 M 和 Bob 取出数的和相等，那么 Bob 获胜，否则 Alice 获胜。

$$1 \leq N \leq 2 \times 10^5$$

$$2 \leq M \leq 10^9$$

$$0 \leq A_i \leq M - 1$$

Solution

- 由于 $2N$ 为偶数，最后一次操作的肯定是 Bob.
- 考虑最后剩余两个数，如果 $2x$ 和 $2y$ 对 M 取模的结果不相同，那么 Alice 的两种选法中至少有一个可以获胜。
- 因此，如果存在 v 满足共有奇数个数 k 满足 $2k \bmod M = v$ ，则 Alice 必胜。（每次随便取出一个不属于这类的即可，便增加了一个，而 Bob 每次至多减少一个，最后剩的两个数肯定可胜）
- 否则如果 Bob 想获胜，Alice 每取一个 x Bob 就必须跟着取一个满足 $2x$ 和 $2y$ 对 M 取模结果相等的 y ，否则就会变为上面的情况。这种情况下最终两个人取走的差是固定的，只可能是 0 或 $M/2$ （若 M 为偶数），算出即可。

ARC151E

有一个序列 A 。 X, Y 是给定的 A 的两个子串，每次可以在 X 的开头或末尾增添或删除一个数字，且需满足任意时刻 X 非空且为 A 的子串，求把 X 变成 Y 的最少次数。

$$1 \leq N \leq 2 \times 10^5$$

$$1 \leq A_i \leq N$$

$$1 \leq P, Q \leq N$$

Solution

- 如果两个串中有相同字符，则显然最优解为将 X 变为最长公共子串再变到 Y 。（由于两个串都是 A 的子串，最长公共子串更容易求出）。
- 否则肯定先将 X 变为一个单个字符，然后后续操作不断交替串长为 $1, 2, 1, 2, \dots$ （可以发现为了保持 A 的子串更长显然不优），最终变为 Y 所含有一个字符。
- 只需要对相邻字符连边，把 X 中所含有的字符距离设为 0 ，跑最短路即可。

[IOI2023] 封锁时刻

- <https://uoj.ac/problem/824>

Solution

- 显然一个点的值只可能恰好 X 可达或恰好 Y 可达或 0 。
- 找到 $X \rightarrow Y$ 这条链中间的一条边（记其连接 a, b ，其中 a 靠近 X ），使得其分开了距离 X 更近的点 and 距离 Y 更近的点。
- 第一种情况是没有点贡献为 2 ，此时贪心即可。
- 否则 $X \rightarrow a$ 和 $Y \rightarrow b$ 这些点一定至少贡献 1 ，此时可以发现最优解不再会出现一个点选了但前驱没选的情况。
- 接下来可以直接转化为每个点贡献 1 和贡献 2 均有不同代价 p, q ，拆成 $p \leq q - p$ 和 $p > q - p$ 的两类，前者可以直接贪心，后者至多有一个仅贡献 1 。
- 排序后枚举选了几组 2 ，以及是否将一个改为 1 /添加一个 1 即可。

[IOI2023] 机器人比赛

- <https://uoj.ac/problem/829>

Solution

- 这里参考了 [IOI 2023 Robot Contest in \$Z^*\$ = 5 - Codeforces](#) 这篇文章。该做法得到了 $Z^* = 5$ 的结果。
- 为了方便描述，我们把数字 0~5 换成 '0, 1, U (2), D (3), L (4), R (5)' 的表示。
- 在第一阶段，通过 bfs 的方式搜索网格图并找到右下角。每轮只找距离已访问格子相邻的那些格子。
- 初始将左上角（通过 U,L 方向都是边界且标为 0 判断）指向 D（如果是墙就改为 E），并向下走。
- 当我们新到达一个格子（标为 0）时，我们将其指针指向上一轮所在的格子（bfs 树上的父节点）并返回（通过找唯一的一个指向自己的格子判断）。
- 如果该格子已访问，那么将其指向的方向逆时针旋转，直到遇到下列情况：
 - 该格子未被访问，为空。此时指向它并走过去。
 - 该格子已被访问并且是其 bfs 树上的父节点（通过找唯一的一个指向自己的格子判断），此时指向它并离开该表。
- 当走回左上角并指向初始时的方向，一轮就结束了。
- 可以发现，在一轮中，所有非叶节点（新发现的点）和根的指针都旋转了一周并从父节点指回了父节点，可以遍历整棵树并增广一层。
- 特别的，一旦走到了右下角（通过 D,R 方向都是边界判断），立即进入第二阶段。

Solution

- 在第二阶段中，我们需要将树中所有不在路径上的点都标为 0 并将路径上的点标为 1。
- 可以发现进入第二阶段时整张图形成了一个以右下角为根的树。（在进入时将该节点标为一个向右或下的方向防止出现错误）。
- 在这一阶段中，不再会更改箭头的方向，而只会判断每个格子是否在路径上来改为 0 或 1。
- 进行 dfs，每当到达一个节点时：
 - 如果存在某个相邻的节点指向自己，暂时不修改该格子，并走过去。（有多个的话随便选一个都行）
 - 否则进行一次判断：该节点在路径上（将标为 1）当且仅当其为左上角或存在一个相邻的标为 1 的节点。
- 根据该遍历的性质，可以发现所有不在路径上且与（某个在路径上点）相邻的点的标记时间均早于（其相邻的在路径上的点），因此不会受到影响。

量子计算和量子通讯介绍

北京大学前沿计算研究中心 袁晓

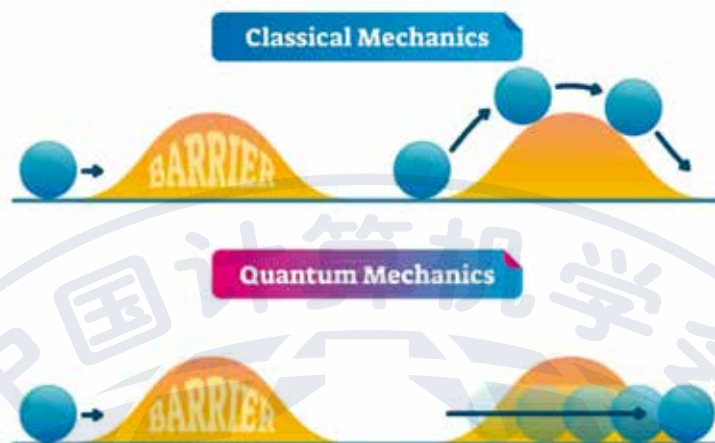
量子信息概念介绍



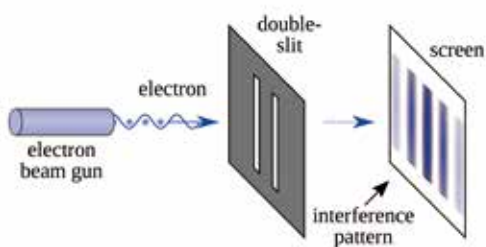


Background: quantum mechanics

Some interesting quantum effects

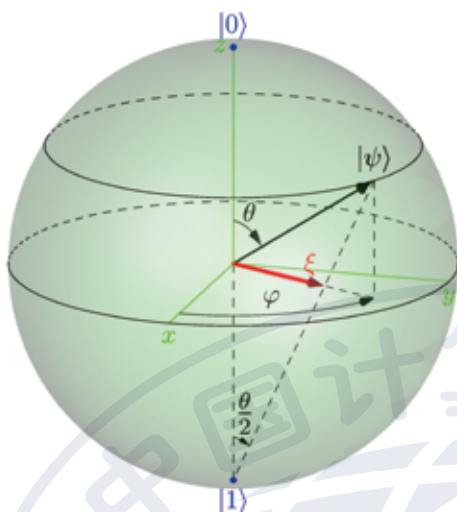


Key concepts: superposition





Key concepts: qubit



<http://stla.github.io/stlapblog/posts/BlochSphere.html>

THE BLOCH SPHERE A stereographic representation of qubits

The simplest quantum state, namely the (pure) qubit, can be written

$$|\psi\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle$$

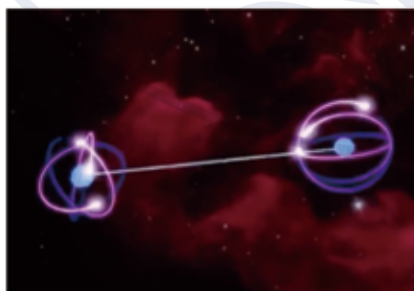
and shown on the Bloch sphere as the vector with spherical polar coordinates θ and φ . Of course, this representation of $|\psi\rangle$ on the sphere is *not* a linear combination of the representations of the basis states $|0\rangle$ and $|1\rangle$ at the poles of the sphere. However this graphical representation is not an artificial one. Indeed, taking the ratio of the two coordinates

$$\xi = \frac{e^{i\varphi} \sin \frac{\theta}{2}}{\cos \frac{\theta}{2}} = \tan \frac{\theta}{2} e^{i\varphi}$$

provides the stereographic projection of $|\psi\rangle$, which is shown in red on the picture. This ratio takes its value in the (xy) -plane plus "a point at infinity", corresponding to the stereographic projection of $|1\rangle$. The other basis state $|0\rangle$ is sent to the origin of the (xy) -plane.



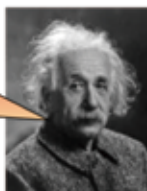
Key concepts: entanglement



$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|0\rangle_A \otimes |0\rangle_B + |1\rangle_A \otimes |1\rangle_B)$$

A	B
0	0
1	1

Spooky
action at a
distance



Measuring on one particle
would instantaneously affect
the state of the other

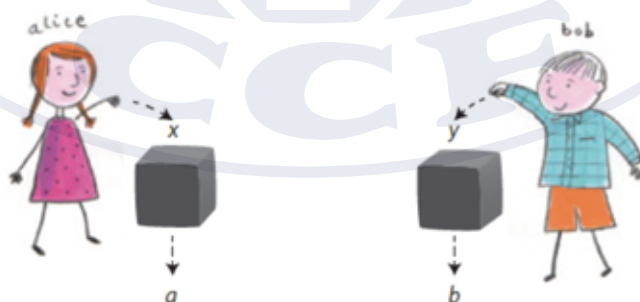
Local hidden variable



Key concepts: non-locality



Key concepts: non-locality



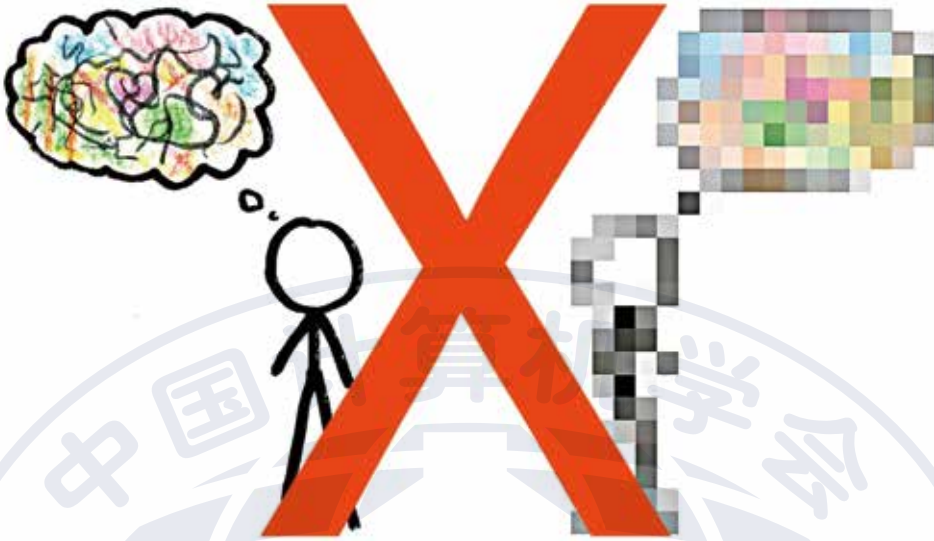
x	y	a	b
0	0	0	0
0	1	1	1
1	0	0	0
1	1	0	1

Classical strategy: always output 00 or 11
Winning probability: 0.75

Quantum strategy: measure an entangled state
Winning probability: 0.85

Key concepts: no-cloning

北京大学 PEKING UNIVERSITY

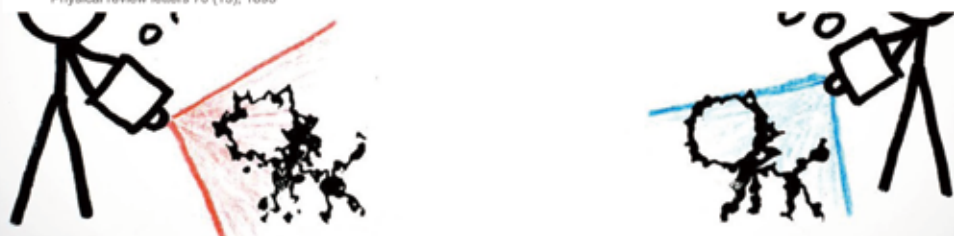


Key concepts: teleportation

北京大学 PEKING UNIVERSITY

Teleportation Explained

[Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels](#) 13118 1993
CH Bennett, G Brassard, C Crépeau, R Jozsa, A Peres, WK Wootters
Physical review letters 70 (13), 1895



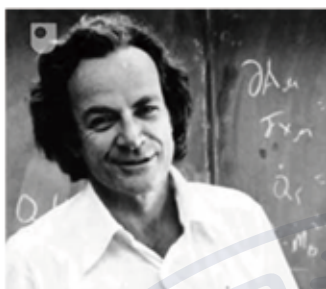


量子计算简介





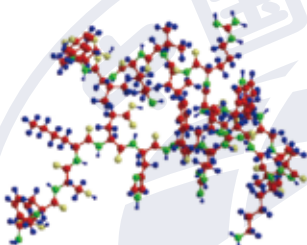
Quantum computer to simulate quantum systems



The dimension of many-body quantum system increases exponentially to the number of particles.

Many-body problem is general hard to classical computers.

Quantum computing can inherently simulate quantum systems and solve the dimension problem.



The first indications of something exciting...



In 1985 David Deutsch found an example of a problem which a quantum machine using qubits could solve faster (in fewer steps) than a conventional computer.

In 1992 David and Richard Jozsa found a stronger example, now called the Deutsch-Jozsa algorithm. It established that quantum machines can (for suitable problems) do profoundly better.



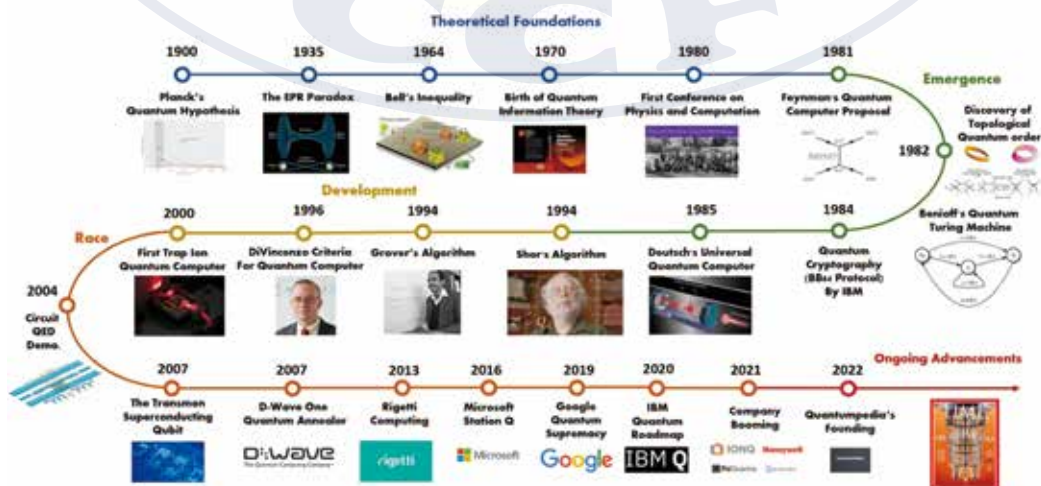


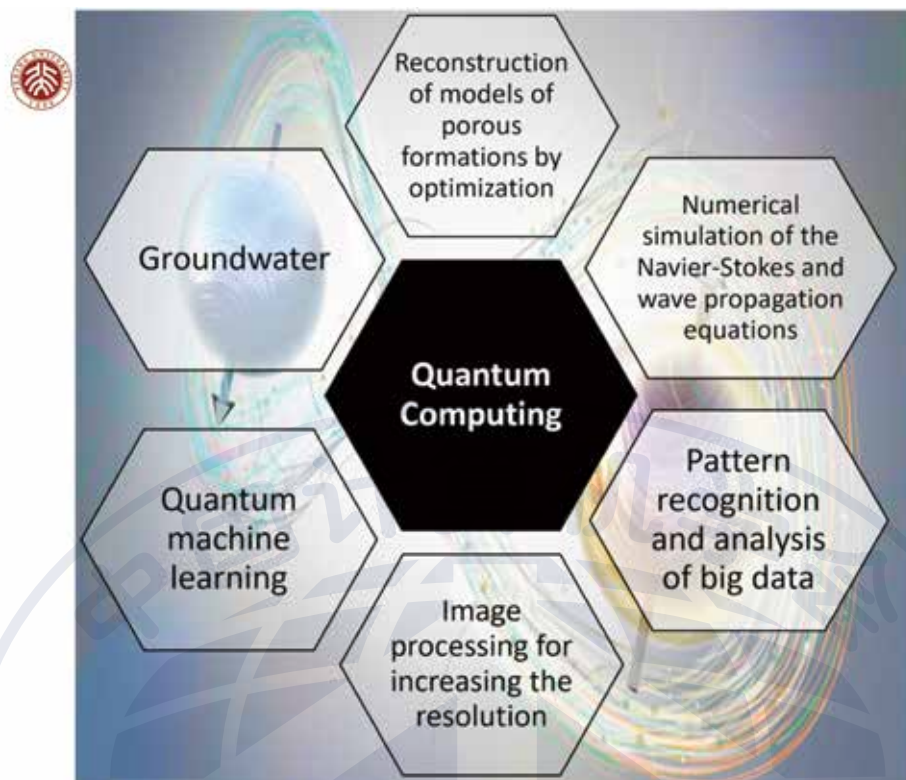
Then came the most famous algorithm



In 1994-96 Peter Shor published his ideas for using a quantum machine to perform fast factoring.

Factoring is *hard* (for a conventional computer). This provides the security for the RSA cryptosystem — the protocol that protects most of the world's information.





Progress: quantum computing

	超导	离子阱	光量子	硅量子点	中性原子
看好技术的潜在用户%	61%	35%	34%	26%	16%
量子比特相干时间	~1 ms	~50+ s	N/A	~1-10 s	~1 s
双量子比特门保真度	~99.6%	~99.9%	~99.9%	~99.99%	99%
门速度	~10-50 ns	~1-50 us	~1 ns	~1-10ns	~100 ns
连接性	附近连接	全连接	全连接	附近连接	附近连接
量子比特	128Q	32Q	18Q	2Q	256Q
优点	1.工程化成熟 2.可扩展性	1.保真度高 2.可连接 3.相干时间长	1.可扩展性 2.现成半导体技术	1.稳定 2.现成半导体技术	1.可扩展性 2.可连接
缺点	1.超低温 2.互连受限	1.门速度慢 2.超高真空 3.单阱水平互连	1.光子损耗 2.难以纠缠	1.超低温 2.工程化初期 3.难以纠缠	1.门保真度和速度 2.超高真空
代表公司	IBM, Google, Q, IONQ, AQT, XANADU, rigetti, D-WAVE, Honeywell, qci, intel, PsiQuantum			Silicon Quantum Computing, intel, PASQAL	ColdQuanta

国家政策和工业界



US

2018年，白宫科学技术政策办公室（OSTP）发布了《量子信息科学国家战略概述》；特朗普总统签署了两党制的《国家量子计划法案》；成立了国家量子协调办公室（NQCO）；

2020年7月22日，白宫科学技术政策办公室和美国国家科学基金会（NSF）周二宣布，将投资7500万美元（约合5.2亿RMB）在全国范围内建立三个量子计算中心。

2020年8月27日，为多学科人工智能和量子计算提供约10亿美元（约70亿RMB）的新资金以建设研究中心；量子信息科学（QIS）：政府6.25亿美元+私营科技公司3亿美元，共计9.25亿美元

这是特朗普政府连续四年，政府预算提议大幅削减联邦研究，包括气候变化等紧迫课题的研究的情况下，只有在人工智能和量子计算领域的投资有所增加。

2020年10月7日，美国白宫量子协调办公室发布了《国家量子信息科学战略投入的量子前沿报告》

美国已将在量子信息科学（QIS）领域保持领导地位作为确保美国长期经济繁荣和国家安全的关键优先事项





US

2021年《无尽前沿法案》在2022至2026财年被授权的拨款共计1124.1亿美元，用于实施本法案及其修订

- (i)人工智能、机器学习和其他软件进步;
- (ii)高性能计算、半导体和先进的计算机硬件;
- (iii)量子计算和信息系统;
- (iv)机器人、自动化和先进制造业;
- (v)预防或减轻自然和人为灾害;
- (vi)先进的通信技术;
- (vii)生物技术、医疗技术、基因组学和合成生物学;
- (viii)网络安全、数据存储和数据管理技术;
- (ix)先进的能源、电池和工业效率;
- (x)与关键技术重点领域有关的先进材料科学、工程和探索。



China

中共中央政治局2020年10月16日下午就量子科技研究和应用前景举行第二十四次集体学习

习近平在主持学习时发表了讲话。他指出，近年来，量子科技发展突飞猛进，成为新一轮科技革命和产业变革的前沿领域。加快发展量子科技，对促进高质量发展、保障国家安全具有非常重要的作用。

5月28日，中国科学院第二十次院士大会、中国工程院第十五次院士大会和中国科学技术协会第十次全国代表大会在北京人民大会堂隆重召开。中共中央总书记、国家主席、中央军委主席习近平出席大会并发表重要讲话。

要在事关发展全局和国家安全的基础核心领域，瞄准人工智能、量子信息、集成电路、先进制造、生命健康、脑科学、生物育种、空天科技、深地深海等前沿领域，前瞻部署一批战略性、储备性技术研发项目，瞄准未来科技和产业制高点。

北京大学
PEKING UNIVERSITY

专栏2 科技前沿领域攻关

01 新一代人工智能	前沿基础理论突破，专用芯片研发，深度学习框架等开源算法平台构建，学习推理与决策，图像图形，语音视频，自然语言识别处理等领域创新。
02 量子信息	城域、城际、自由空间量子通信技术研发，通用量子计算原型机和实用化量子模拟机研制，量子精密测量技术突破。
03 集成电路	集成电路设计工具、重点装备和高纯靶材等关键材料研发，集成电路先进工艺和绝缘栅双极型晶体管（IGBT）、微机电系统（MEMS）等特色工艺突破，先进存储技术升级，碳化硅、氮化镓等宽禁带半导体发展。
04 脑科学与类脑研究	脑认知原理解析，脑介观神经联接图谱绘制，脑重大疾病机理与干预研究，儿童青少年脑发育，类脑计算与脑机融合技术研发。
05 基因与生物技术	基因组学研究应用，遗传细胞和遗传育种，合成生物，生物药等技术创新，创新疫苗、体外诊断，抗体药物等研发，农作物、畜禽水产，农业微生物等重大新品种创制，生物安全关键技术研究。
06 临床医学与健康	癌症和心脑血管、呼吸、代谢性疾病等发病机制基础研究，主动健康干预技术研发，再生医学，微生物组，新型治疗等前沿技术研发，重大传染病、重大慢性非传染性疾病防治关键技术研究。
07 深空深海深地和极地探测	宇宙起源与演化，透视地球等基础科学研究，火星环绕、小行星巡视等星际探测，新一代重型运载火箭和重复使用航天运输系统，地球深部探测装备，深海运维保障和装备试验船、极地立体观测监测平台和重型破冰船等研制，探月工程四期、蛟龙探海二期、雪龙探极二期建设。

China

北京大学
PEKING UNIVERSITY

Industry progress

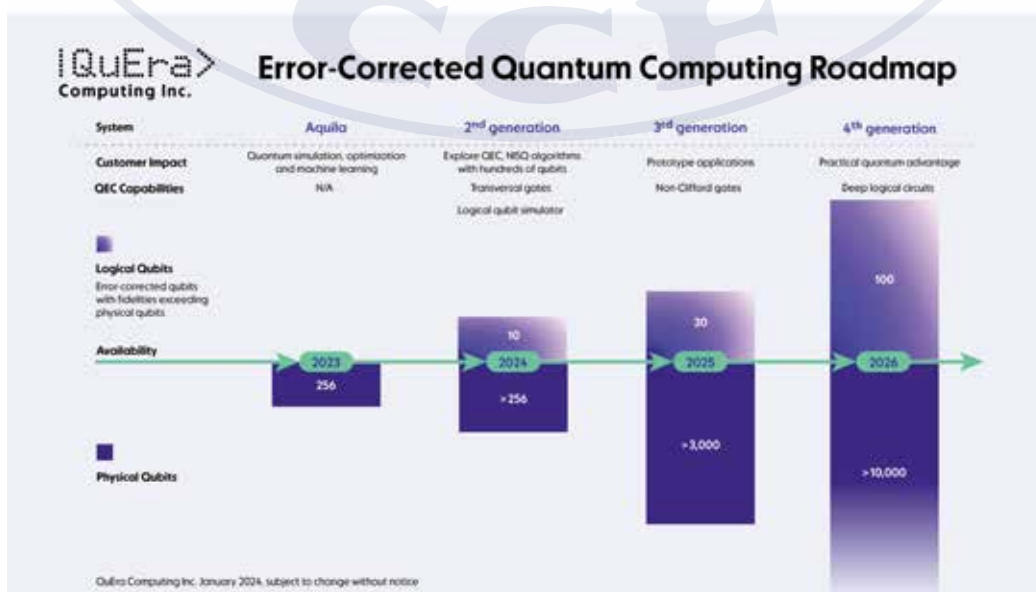




Industry progress



Industry progress





Recap: complex numbers and complex vectors

In general, we require complex numbers to describe quantum phenomena.

Any $z \in \mathbb{C}$ is of the form $z = a + ib$ for some $a, b \in \mathbb{R}$ and $i = \sqrt{-1}$.

\mathbb{C}^n is the vector space of n -tuples of complex numbers $\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$.

With addition:

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} + \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} z_1 + w_1 \\ z_2 + w_2 \\ \vdots \\ z_n + w_n \end{bmatrix}$$

and scalar multiplication: W

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} W z_1 \\ W z_2 \\ \vdots \\ W z_n \end{bmatrix}$$

Matrices

A matrix is an array of (in general) complex numbers:

$$A = \begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & & a_{nm} \end{bmatrix}$$

With addition:

$$\begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & & a_{nm} \end{bmatrix} + \begin{bmatrix} b_{11} & \dots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{n1} & & b_{nm} \end{bmatrix} = \begin{bmatrix} a_{11} + b_{11} & \dots & a_{1m} + b_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} + b_{n1} & & a_{nm} + b_{nm} \end{bmatrix}$$

and scalar multiplication:

$$B \begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & & a_{nm} \end{bmatrix} = \begin{bmatrix} Ba_{11} & \dots & Ba_{1m} \\ \vdots & \ddots & \vdots \\ Ba_{n1} & & Ba_{nm} \end{bmatrix}$$

5 / 20

Matrix multiplication

If A is a $n \times m$ matrix and B is a $m \times l$ matrix then $C = A \times B$ is the $n \times l$ matrix with entries given by

$$C_{ik} = \sum_{j=1}^m A_{ij} B_{jk}$$

for all $i = 1, \dots, n$ and $k = 1, \dots, l$.

Matrix multiplication is

- associative: $(A \times B) \times C = A \times (B \times C) = ABC$
- distributive: $A(B + C) = AB + AC$; $(A + B)C = AB + BC$
- not commutative: $AB \neq BA$

6 / 20

Tensor multiplication

As well as scalar multiplication and matrix multiplication, to describe quantum computation we must consider a third form of multiplication on matrices, tensor multiplication. Let A and B be matrices of any dimension:

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1m}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \dots & a_{nm}B \end{bmatrix}$$

where \otimes denotes the tensor product. For example:

$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \otimes \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 4 & 6 \end{bmatrix}$$

In general if A is $n \times m$ and B is $n' \times m'$ then $A \otimes B$ is $nn' \times mm'$.

7 / 20

Combining matrix and tensor multiplication

As a (column) vector is just a $n \times 1$ matrix, we can equally well apply tensor products to vectors. This reveals an important property of tensor products when combined with matrix products. Let A and B be $n \times m$ and $n' \times m'$ matrices respectively, and \mathbf{x} and \mathbf{y} be m and m' dimension column vectors respectively:

$$(A \otimes B)(\mathbf{x} \otimes \mathbf{y}) = (A\mathbf{x}) \otimes (B\mathbf{y})$$

The second exercise sheet asks you to prove this for the case of 2×2 matrices.

8 / 20

“Bras” and “Kets”

A “Ket” is a column vector:

$$|\psi\rangle = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

Each “Ket” has a corresponding “Bra”, which is its conjugate transpose, the row vector:

$$\langle\psi| = [a_1^* \ a_2^* \ \dots \ a_n^*]$$

We continue to denote matrix operations with a capital letter, i.e., the matrix A operating on the state $|u\rangle$ would be written $A|u\rangle$.

When tensor multiplying vectors expressed as kets, the following are all equivalent: $|\psi\rangle \otimes |\phi\rangle$, $|\psi\rangle |\phi\rangle$, $|\psi\phi\rangle$. Note also that tensor multiplication is associative, so $(|\psi\rangle \otimes |\phi\rangle) \otimes |\omega\rangle = |\psi\rangle \otimes (|\phi\rangle \otimes |\omega\rangle) = |\psi\phi\omega\rangle$.

11 / 20

The qubit: an information theoretic way to represent superposition

A classical bit is an intuitive concept, it is either equal to:

$$0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle \text{ or } 1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

Even if we are uncertain about whether a classical bit, B , is in state 0 or 1, we can characterise it by a probability mass function, or a *mixture*

$$p(B = 0) = p_0 \ ; \ p(B = 1) = p_1$$

where $p_0 + p_1 = 1$. A qubit, $|\psi\rangle$, is quite different, it can be in a *superposition* of the 0 and 1 states:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

where α and β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$.

10 / 17

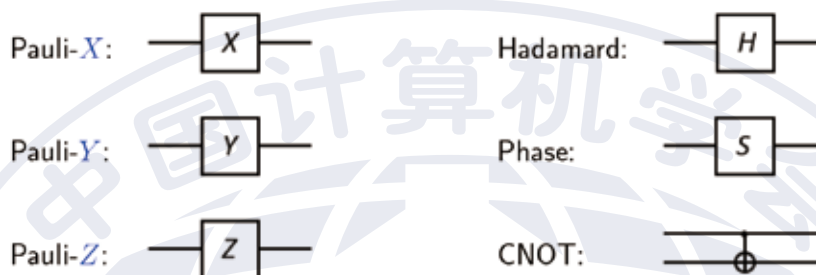
Quantum circuits: from matrices to gates

In the tensor network, we have that:

- Wires are qubits (possibly entangled).
- Gates are unitary matrices.

We have already met the Pauli and Hadamard single-qubit unitary matrices as well as the CNOT two-qubit unitary, and the *phase gate*

$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$ is also a useful primitive.



5 / 22

The Pauli matrices

The Pauli matrices X , Y and Z are important one-qubit unitary matrices.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Which has the following effect on the computational basis states:
 $X|0\rangle = |1\rangle$; $X|1\rangle = |0\rangle$.

$$Y = i \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Which has the following effect on the computational basis states:
 $Y|0\rangle = i|1\rangle$; $Y|1\rangle = -i|0\rangle$.

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Which has the following effect on the computational basis states:
 $Z|0\rangle = |0\rangle$; $Z|1\rangle = -|1\rangle$.

9 / 20

The Hadamard matrix

Another important one-qubit unitary is the Hadamard matrix:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Which has the following effect on the computational basis states:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \equiv |+\rangle$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \equiv |-\rangle$$

i.e., it puts the computational basis states in superposition. H is self-inverse, therefore:

$$H|+\rangle = |0\rangle \quad H|-\rangle = |1\rangle$$

i.e., it interferes the superposition to recover the original computational basis states.

11 / 20

Entangled states

As we shall see, quantum computing draws its advantage from the fact that not all quantum states are separable. Consider the *two qubit unitary*

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

applied to the state $|+\rangle \otimes |0\rangle$:

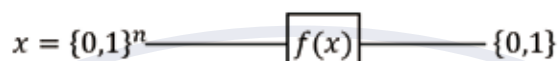
$$\begin{aligned} \text{CNOT}(|+\rangle \otimes |0\rangle) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ 1/\sqrt{2} \\ 0 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ 0 \\ 1/\sqrt{2} \end{bmatrix} \\ &= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \end{aligned}$$

an *entangled state* which cannot be separated as tensor product.

19 / 20

Computing mathematical functions on quantum computers

In order to design quantum algorithms we need to know how to execute functions on a quantum computer. In particular, we are interested in functions that take a (binary) number, and output a truth value (i.e., $\{0,1\}$). That is, functions of the form $f : \{0,1\}^n \rightarrow \{0,1\}$.


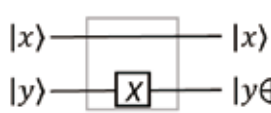
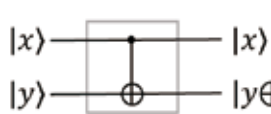
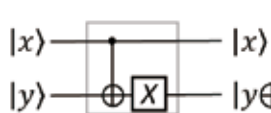


However, we know that a quantum circuit is composed of quantum gates, which are unitary operations, and apart from for trivial functions, this is not a unitary.

3 / 18

Constant and balanced function on a single bit

If the input, x , is a single bit (as is the output), then we have four possible functions:

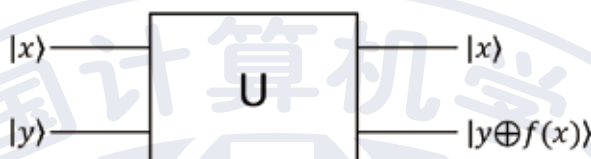
Function	x	$f(x)$	Type	Unitary
$f(x) = 0$	0 1	0 0	Constant	
$f(x) = 1$	0 1	1 1	Constant	
$f(x) = x$	0 1	0 1	Balanced	
$f(x) = x \oplus 1$	0 1	1 0	Balanced	

6 / 18

Deutsch's algorithm set-up

We want to find out whether a particular function, with one input bit and one output bit is constant or balanced. Classically, we need to evaluate the function *twice* (i.e., for input = 0 and input = 1), but remarkably, we only need to evaluate the function *once* quantumly, by using Deutsch's algorithm.

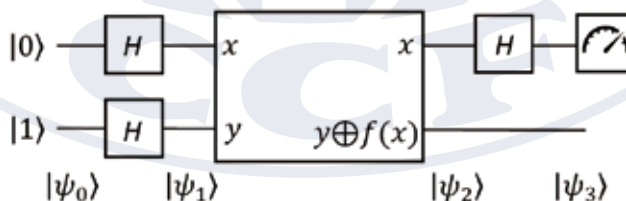
We have a two qubit unitary, which is one of the four on the previous slide (we don't know which):



Which we are going to incorporate into a quantum circuit.

7 / 18

Deutsch's algorithm (1)



Initially we prepare the state:

$$|\psi_0\rangle = |01\rangle$$

Which the initial Hadamard gates put in the superposition state:

$$\begin{aligned} |\psi_1\rangle &= \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \\ &= \frac{1}{2} (|00\rangle - |01\rangle + |10\rangle - |11\rangle) \end{aligned}$$

8 / 18

Deutsch's algorithm (2)

$$|\psi_1\rangle = \frac{1}{2} (|00\rangle - |01\rangle + |10\rangle - |11\rangle)$$

Next the unitary is implemented, which sets the second qubit to $y \oplus f(x)$, so we have four options for $|\psi_2\rangle$:

$$\begin{array}{ll} f(x) = 0 & |\psi_2\rangle = \frac{1}{2} (|00\rangle - |01\rangle + |10\rangle - |11\rangle) \\ f(x) = 1 & |\psi_2\rangle = \frac{1}{2} (|01\rangle - |00\rangle + |11\rangle - |10\rangle) \\ f(x) = x & |\psi_2\rangle = \frac{1}{2} (|00\rangle - |01\rangle + |11\rangle - |10\rangle) \\ f(x) = x \oplus 1 & |\psi_2\rangle = \frac{1}{2} (|01\rangle - |00\rangle + |10\rangle - |11\rangle) \end{array}$$

which factorises as:

$$|\psi_2\rangle = \begin{cases} \pm \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) & \text{if } f(0) = f(1) \\ \pm \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) & \text{if } f(0) \neq f(1) \end{cases}$$

That is the two balanced cases differ only by an unobservable global phase (and likewise for the two constant cases).

9 / 18

Deutsch's algorithm (3)

$$|\psi_2\rangle = \begin{cases} \pm \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) & \text{if } f(0) = f(1) \\ \pm \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) & \text{if } f(0) \neq f(1) \end{cases}$$

The next step is to use the Hadamard gate to interfere the superposition on the first qubit, which yields:

$$|\psi_3\rangle = \begin{cases} \pm |0\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) & \text{if } f(0) = f(1) \\ \pm |1\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) & \text{if } f(0) \neq f(1) \end{cases}$$

The final step is to measure the first qubit, and we can see that the outcome will always be 0 if the function is constant, and 1 if balanced.

We can see that superposition and interference, in some sense, play complementary roles: we prepare a state in superposition, perform some operations, and then use interference to discern some global property of the state.

10 / 18

Quantum computing jargon

Query complexity:

- In Deutsch's algorithm we are not using a quantum computer to evaluate a "classically difficult" function *per se*, but rather using quantum phenomena to reduce the number of queries we need to make to an unknown function, to ascertain some information thereabout.

Oracles and black boxes:

- In Deutsch's algorithm, and other query complexity algorithms, we query U , which is known as a "black box", or often in quantum computing an "oracle". The oracle in Deutsch's algorithm is sufficiently simple that we can explicitly express each possible option, but frequently in quantum computing problems are framed in terms of oracles, even when this is not the case.

11 / 18

Deutsch-Jozsa algorithm



Academia Europaea

Richard Jozsa

Together with Richard Jozsa, who is now a Professor here in DAMTP, David Deutsch generalised the algorithm to apply to constant / balanced functions of any input size.

12 / 18

Deutsch's problem

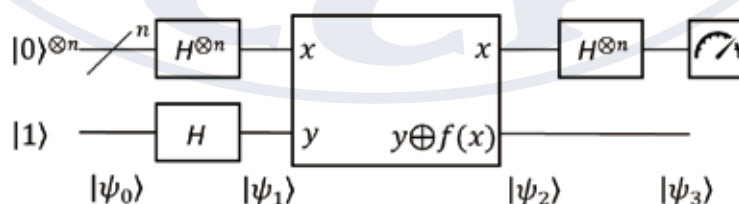
The Deutsch-Jozsa algorithm is usually motivated in terms of Deutsch's problem:

- We have a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$.
- We are **promised this function is either constant** (same output for each x) **or balanced** ($f(x)$ is equal to each of 0 and 1 for exactly half of the possible values of x).
- Classically, we can see that we may need to query the function $\frac{2^n}{2} + 1$ times to be sure whether the function is constant or balanced.
- But quantumly, if the function is encoded as a quantum oracle, then the Deutsch-Jozsa algorithm allows us to determine whether the function is constant or balanced with only a single oracle call.

13 / 18

Deutsch-Jozsa algorithm (1)

The circuit of the Deutsch-Jozsa algorithm closely resembles that of Deutsch's algorithm:



The initial state of which can be expressed:

$$|\psi_0\rangle = |0\rangle^{\otimes n} |1\rangle$$

which is then put into superposition, which can conveniently be expressed:

$$|\psi_1\rangle = \sum_{x \in \{0,1\}^n} \frac{1}{\sqrt{2^{n+1}}} |x\rangle (|0\rangle - |1\rangle)$$

14 / 18

Deutsch-Jozsa algorithm (2)

$$|\psi_1\rangle = \sum_{x \in \{0,1\}^n} \frac{1}{\sqrt{2^{n+1}}} |x\rangle (|0\rangle - |1\rangle)$$

The unitary transforms $|\psi_1\rangle$ to:

$$|\psi_2\rangle = \sum_{x \in \{0,1\}^n} \frac{1}{\sqrt{2^{n+1}}} (-1)^{f(x)} |x\rangle (|0\rangle - |1\rangle)$$

We now address the interference $H^{\otimes n}$ on the first n wires, for which we use the expression:

$$H^{\otimes n} |x\rangle = \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} (-1)^{x \cdot z} |z\rangle$$

which allows us to express:

$$|\psi_3\rangle = \sum_{x \in \{0,1\}^n} \sum_{z \in \{0,1\}^n} \frac{1}{2^n} (-1)^{x \cdot z + f(x)} |z\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

15 / 18

Deutsch-Jozsa algorithm (3)

We can now determine whether the function is constant or balanced by measuring the first n qubits of the final state (i.e., we neglect the final qubit which is in the $|-\rangle$ state):

$$|\psi_3\rangle = \left(\sum_{z \in \{0,1\}^n} \sum_{x \in \{0,1\}^n} \frac{1}{2^n \sqrt{2}} (-1)^{x \cdot z + f(x)} |z\rangle \right) \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

Specifically, we consider the probability of measuring zero on every qubit, which corresponds to the term in the superposition where $|z\rangle$ is $|0\rangle^{\otimes n}$:

- In the case where the function is constant, then the co-efficient of $|0\rangle^{\otimes n}$, $\sum_x (-1)^{f(x)} / 2^n$ is equal to ± 1 ... as this has amplitude 1, then we measure $|0\rangle^{\otimes n}$ with probability one.
- In the case where the function is balanced then $\sum_x (-1)^{f(x)} / 2^n = 0$, and so we will never measure $|0\rangle^{\otimes n}$.

So it follows that measuring the first n qubits allows us to determine with certainty whether the function is constant (measure all zeros) or balanced (measure at least one 1).

16 / 18

Potential for exponential speed-up using a quantum computer



Imagine now that the oracle is held by a person, “Bob”, who is spatially separated from the person, “Alice”, who is trying to determine whether the function is constant or balanced.

- To resolve an instance of Deutsch's problem, classically Alice transmits $\frac{2^n}{2} + 1$ messages, each of size n bits, and each of which Bob replies to with a one bit message.
- Whereas quantumly the Deutsch-Jozsa algorithm requires only the transmission of a single $n + 1$ qubit message by Alice, to which Bob replies with a n qubit message.

So there is an exponential reduction in the amount of information transfer required to solve an instance of Deutsch's problem.

17 / 18

Summary

- We can encode any mathematical function as a unitary matrix.
- Deutsch's algorithm was the first algorithm that demonstrated a quantum advantage: specifically a reduction in query complexity compared to the classical case.
- The Deutsch-Jozsa algorithm generalises Deutsch's algorithm, and reveals the possibility of exponential speed-ups using quantum computers.

18 / 18

组合计数中的递推问题

清华大学交叉信息研究院 李白天

今天讲什么？

- 组合计数明明处处都有递推, 所以几乎什么都可以讲.
- 基于生成函数的处理手法, 部分经典算法, 以及它们一些 OI 之外的故事.
- 不追求困难性, 所以只会有比较简单但典型的例子.
- 所以, 以下内容全都不会讲:
 - UOJ593 新年的军队
 - UOJ633 你将如闪电般归来
 - Codeforces1687F Koishi's Unconscious Permutation
 - SDOI2022 多边形
 - CTS2023 另一个欧拉数问题
- 当然, 欢迎大家补题!

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

2 / 69

1 组合类与生成函数

- 组合构造的字典
- 连通图计数
- n 王问题

2 有关递推式的算法

- 半在线卷积的更快算法 — 超越 “CDQ 分治”
- 线性递推的 Bostan-Mori 算法
- 多项式 Euclid 算法
- Hermite-Padé 逼近

3 整式递推的理论

- 为什么要研究整式递推
- 线性空间的表述方式
- 代数幂级数
- 多元微分有限
- 整式递推在 OI 中的未来

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

3 / 69

从组合类到生成函数

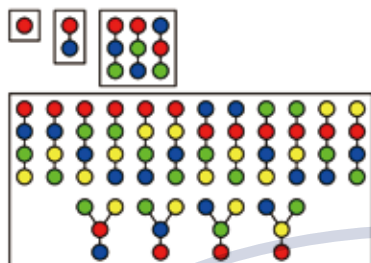


图: n 个顶点的 Cayley 树

Kilom691, CC BY-SA 3.0, via Wikimedia Commons

数列

$$\{a_n\}_{n \geq 0} = \{0, 1, 1, 3, 16, \dots\} \quad (1)$$

Generating Function
生成函数

$$A(x) = \sum_{n=1}^{\infty} n^{n-2} \cdot x^n \quad (2)$$

• 组合类:

$$\mathcal{A} = \{\alpha_1, \alpha_2, \dots, \alpha_k, \dots\} \quad (3)$$

• 生成函数:

$$A(x) = \sum_{\alpha \in \mathcal{A}} x^{|\alpha|}. \quad (4)$$

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

4 / 69

基本运算

	\mathcal{A} $= \{\alpha_1, \alpha_2, \dots\}$	$A(x)$ $= \sum_{\alpha \in \mathcal{A}} x^{ \alpha }$	a_n $= \#\{\alpha : \alpha = n\}$
无交并	$\mathcal{C} = \mathcal{A} \sqcup \mathcal{B}$	$C = A + B$	$c_n = a_n + b_n$
积	$\mathcal{C} = \mathcal{A} \times \mathcal{B}$	$C = A \cdot B$	$c_n = \sum_{k=0}^n a_k b_{n-k}$
序列	$\mathcal{B} = \text{Seq } \mathcal{A}$ $= \epsilon \sqcup \mathcal{A} \times \mathcal{B}$	$B = 1 + A + A^2 + \dots$ $= 1 + A \cdot B$ $= \frac{1}{1-A}$	$b_n = \sum_{k=1}^n a_k b_{n-k}$
多重集	$\mathcal{B} = \text{MSet } \mathcal{A}$ $= \prod_{\alpha \in \mathcal{A}} (\text{Seq } \alpha)$	$= \prod_{n=1}^{\infty} (1 - x^n)^{-a_n}$?
幂集	$\mathcal{B} = \text{Set } \mathcal{A}$ $= \prod_{\alpha \in \mathcal{A}} (\epsilon \sqcup \alpha)$	$= \prod_{n=1}^{\infty} (1 + x^n)^{a_n}$?

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

5 / 69

同一个世界, 不同的梦想

- 如果 $\mathcal{A} = \text{MSet } \mathcal{B}$, 那么

$$A(x) = \exp \left(\sum_{k=1}^{\infty} \frac{B(x^k)}{k} \right), \quad (5)$$

$$B(x) = \sum_{k=1}^{\infty} \frac{\mu(k)}{k} \log A(x^k). \quad (6)$$

- 考虑同一个生成函数

$$A(x) = \frac{1}{1-qx} = 1 + qx + q^2x^2 + \dots, \quad (7)$$

- 对于组合类 \mathcal{A} 的两种解释: 字符集为 $|\Sigma| = q$ 的字符串 Σ^* , 或有限域上的多项式 $\mathbb{F}_q[T]$.
- 组合类 \mathcal{B} 的解释: 字符串的 Lyndon 分解, 或者 $\mathbb{F}_q[T]$ 分解成不可约因子之乘积.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

6 / 69

同一个世界, 不同的梦想

- 组合类 \mathcal{B} 的解释:

$$\Sigma^* = \text{MSet}[\mathcal{L}_{\text{Lyndon}}] \quad (8)$$

$$\mathbb{F}_q[T] = \text{MSet}[\mathcal{I}_{\text{irreducible}}]. \quad (9)$$

- 得到同样的生成函数和数列:

$$B(x) = \sum_{k=1}^{\infty} \frac{\mu(k)}{k} \log \frac{1}{1-qx^k} \quad (10)$$

$$b_n = \frac{1}{n} \sum_{k|n} \mu(k) q^{n/k}. \quad (11)$$

- Lyndon 串的组合意义还算好理解, 但后者的组合意义恐怕需要一点更多的知识.
- Lyndon 串和不可约多项式之间的双射也是不太显然的, 这个双射也不是很典则, 一般来说要选取域扩张 $\mathbb{F}_{q^n}/\mathbb{F}_q$ 的一个正规基.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

7 / 69

指数生成函数的基本运算

如果 α, β 有 $\binom{|\alpha|+|\beta|}{|\alpha|}$ 种组合方式, 那么就要考虑

$$\frac{x^n}{n!} \cdot \frac{x^m}{m!} = \binom{n+m}{n} \frac{x^{n+m}}{(n+m)!}. \quad (12)$$

	\mathcal{A}	$A(x)$	a_n
	$= \{\alpha_1, \alpha_2, \dots\}$	$= \sum_{\alpha \in \mathcal{A}} x^{ \alpha }$	$= \#\{\alpha : \alpha = n\}$
无交并	$\mathcal{C} = \mathcal{A} \sqcup \mathcal{B}$	$C = A + B$	$c_n = a_n + b_n$
积	$\mathcal{C} = \mathcal{A} \times \mathcal{B}$	$C = A \cdot B$	$c_n = \sum_{k=0}^n \binom{n}{k} a_k b_{n-k}$
k 元集	$\mathcal{B} = \text{MSet}_k \mathcal{A}$	$B = A^k / k!$?
多重集	$\mathcal{B} = \text{MSet} \mathcal{A}$	$B = \exp A$?
	$= \bigsqcup_{k=0}^{\infty} \text{MSet}_k \mathcal{A}$		

微分算子

- 定义

$$\partial \left(\sum_{n=0}^{\infty} a_n x^n \right) = \sum_{n=0}^{\infty} a_n \cdot n x^{n-1}, \quad (13)$$

- 对于普通生成函数, 有

$$\partial \cdot x^n = n x^{n-1}. \quad (14)$$

- 对于指数型生成函数, 有关递推式的算法

$$\partial \cdot \frac{x^n}{n!} = \frac{x^{n-1}}{(n-1)!}. \quad (15)$$

- 从组合意义的角度, 它们相当于对于组合类进行了什么变换?

和微分有关的运算律

直接从组合意义的角度, 解释如下运算律:

- 可加性:

$$\partial(A + B) = \partial A + \partial B. \quad (16)$$

- Leibniz 律:

$$\partial(A \cdot B) = (\partial A) \cdot B + A \cdot (\partial B). \quad (17)$$

- 复合:

$$\partial(A \circ B) = ((\partial A) \circ B) \cdot (\partial B). \quad (18)$$

多重集构造的递推式

- 微分方程 \iff 组合解释.

$$\mathcal{B} = \text{MSet } \mathcal{A} \quad (19)$$

$$B = \exp A \quad (20)$$

$$B' = B \cdot A' \quad (21)$$

$$b_n = \sum_{k=1}^n \binom{n-1}{k-1} a_k b_{n-k}. \quad (22)$$

例子 — 连通图

设 \mathcal{G} 为无向图构成的组合类, \mathcal{C} 为无向连通图构成的组合类, 那么

$$\mathcal{G} = \text{MSet } \mathcal{C}. \quad (23)$$

尝试解释以下两种不同的递推式, 分别从 **组合意义** 和 **代数推导**:

$$C_n = 2^{n(n-1)/2} - \sum_k \binom{n-1}{k-1} C_k 2^{(n-k)(n-k-1)/2}, \quad (24)$$

$$C_n = \sum_k \binom{n-2}{k-1} C_k C_{n-k} \cdot (2^k - 1). \quad (25)$$

例子 — n 王问题

- 有多少 n 阶排列 σ 使得相邻两项的差的绝对值不是 1?
- 查表发现这被收录于 **整数序列百科** 的第 A002464 项.
设这个数列叫做 A_n , 有如下递推式:

$$A_n = (n+1)A_{n-1} - (n-2)A_{n-2} - (n-5)A_{n-3} + (n-3)A_{n-4}. \quad (26)$$

如何证明？

组合证明？有的，但是...

一道组合题的线性时间做法 - 递推树上递推果，递推树下你和我

狗雷布是真的伊
真的狗雷布真的伊

93 人赞同了该文章

前言：

这篇文章写的是BZOJ上的4321号题目《昨天可爱的小灰机 @FFjet 给我的》，然后本菜做了一天，写了一下午，改了半个晚上，才出来这篇文章。

题目里给的 n 是有范围的， $1 \leq n \leq 1000$ ，也就是说，随缘无望，题目自要求基本上 $O(n^3)$ 到顶了。下面解法的线性的复杂度已经超越我所想了。现在网上大部分做法是 $O(n^2)$ 的动态规划。（事实上我算是最后不化简也是 $O(n^2)$ 的，最后化简成 $O(n)$ 的算法）

好吧这个题目真的难，如果改成证明题的话，放到CMO里面也可以当3A6了qwq

现在，启动发动机，开始起飞——

<https://zhuanlan.zhihu.com/p/56537011>

长达几页纸的双射... 有没有更简单的方法？

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

14 / 69

生成函数与递推式

对于相邻关系容斥，可以写出生成函数

$$\sum_n A_n x^n = \left(\sum_{n=0}^{\infty} n! T^n \right) \circ (x - 2x^2 + 2x^3 - 2x^4 + 2x^5 + \dots) \quad (27)$$

$$= \left(\sum_{n=0}^{\infty} n! T^n \right) \circ \left(x \frac{1-x}{1+x} \right). \quad (28)$$

记 $S(T) = \sum_{n=0}^{\infty} n! T^n$ ，将递推式 $S_n = nS_{n-1} + [n=0]$ 转化为生成函数的微分方程

$$S(T) = 1 + (S(T) \cdot T)' \cdot T \quad (29)$$

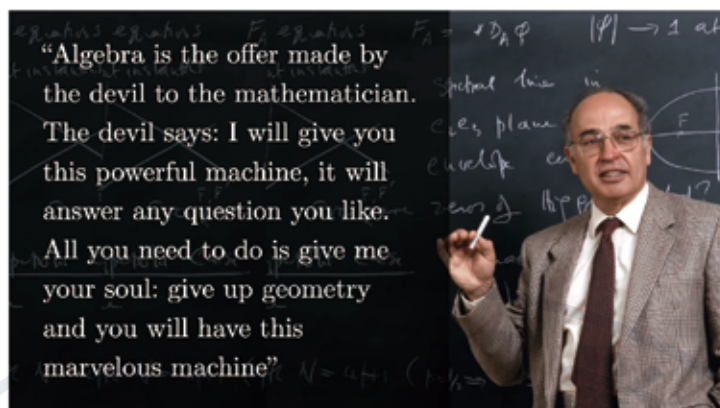
$$S = 1 + TS + T^2 S'. \quad (30)$$

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

15 / 69



— Michael Atiyah

生成函数是魔鬼和我们的交易。魔鬼说：我给你这个强大的机器，它能回答任何你想问的问题。但是，你必须付出代价，你必须给我你的灵魂：放弃组合意义，然后你就能得到这台威力无穷的机器！

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

16 / 69

课间休息, 思考题

记 \mathcal{A} 为 2-正则图构成的组合类, 证明其指数型生成函数是

$$A(x) = \exp\left(-\frac{x}{2} - \frac{x^2}{4}\right) \frac{1}{\sqrt{1-x}}. \quad (31)$$

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

17 / 69

生成函数 / 多项式 — 算法

以下内容不会讲, 但是只假设它存在, 对后续内容的理解也基本没有影响.

- 快速 Fourier 变换: 高效计算 $A(x)B(x) \bmod x^n$.
- Newton 迭代法: 高效计算 $A(x)^{-1}$, $\log A(x)$, $\exp A(x)$ 等基本初等函数.

时间一般认为是 $\mathcal{O}(n \log n)$ 的, 但仔细思考计算模型会发现并不显然, 这里就记作 $M(n)$.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

18 / 69

小故事: 整数乘法的长征

- Колмогоров 的猜测: $M(N) = \Omega(N^2)$

分治乘法!

- $M(N) = \mathcal{O}(N^{\log 3 / \log 2})$: [Karatsuba 1962]
- $M(N) = N 2^{\mathcal{O}(\sqrt{\log N})}$: [Toom 1963], [Schönhage 1966], [Knuth 1969]

快速 Fourier 变换 [Gauß 1876], [Cooley-Tukey 1965], 但单位根怎么存?

- $M(N) = \mathcal{O}(N \log N \log \log N \log \log \log N \dots)$: [Pollard 1971]
- $M(N) = \mathcal{O}(N \log N \log \log N)$: [Schönhage-Strassen 1971] **GMP**
- $M(N) = N \log N 2^{\mathcal{O}(\log^* N)}$: [Fürer 2007], Harvey, van der Hoeven, Lecerf...
- $M(N) = \mathcal{O}(N \log N)$: [Harvey-van der Hoeven 2019]

模型是多带 Turing 机, 可以大概理解成要衡量一个一个 bit 的操作.

Network Coding Conjecture

- 如果 网络编码猜想 成立, 那么这是不可改进的.
[Afshani-Freksen-Kamma-Larsen 2019]

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

19 / 69

半在线卷积

- 一般的卷积式

$$c_n = \sum_k a_k b_{n-k} \quad (32)$$

也可以看做是一个递推式, 如果我们只有知道了 c_n 才知道

a_{n+1}, b_{n+1} . 这是 Relaxed Convolution 在线卷积问题.

- 如果序列 b 是一开始就完全知道的, 这是 Semi Relaxed Convolution 半在线卷积问题.
- 通过外层分治可以发现, 在线卷积并不比半在线卷积要难.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

20 / 69

半在线卷积的算法 [van der Hoeven 2002, 2007]

- 直接的半在线卷积算法: 每次分治成两半, 递归求解, 时间复杂度 $\mathcal{O}(N \log^2 N)$.
- 每次分成 $B = \mathcal{O}(\log N)$ 块: 时间复杂度 $\mathcal{O}\left(\frac{N \log^2 N}{\log \log N}\right)$.
- 形如 $T(N) = 3\sqrt{N}T(\sqrt{N}) + \mathcal{O}(N \log N)$ 的递归式:

$$\mathcal{O}\left(N(\log N)^{\log 3 / \log 2}\right) \quad (33)$$

- 形如 $T(N) = 2\ell N^{1-1/\ell}T(N^{1/\ell}) + \mathcal{O}(\ell N \log N)$ 的递归式:

$$\mathcal{O}\left(N \log N \exp\left(2\sqrt{\log 2 \log \log N}\right)\right) \quad (34)$$

- 没有平衡?

$$R(n) = \mathcal{O}\left(N \log N \exp\left(\sqrt{2 \log 2 \log \log N}\right) \sqrt{\log \log N}\right) \quad (35)$$

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

21 / 69

线性递推的简洁算法 [Bostan-Mori 2021]

- ① 将生成函数写作 $P(x)/Q(x)$ 的形式
- ② 不妨分子分母同乘 $Q(-x)$, 得到 $P(x)Q(-x)/Q(x)Q(-x)$, 分母有什么特点?
- ③ 求第 K 项的时间复杂度: $\mathcal{O}(M(N)\log K)$. 只需要实现多项式乘法.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

22 / 69

多项式 Euclid

- 给定多项式 $A(T), B(T)$, 求 $X(T), Y(T)$ 使得

$$AX + BY = \gcd(A, B). \quad (36)$$

- 说真的, 我们除了辗转相除法以外没有别的什么思路.

$$A_0, B_0 \quad (37)$$

$$A_1 = B_0, B_1 = A_0 \bmod B_0 \quad (38)$$

$$A_2 = B_1, B_2 = A_1 \bmod B_1 \quad (39)$$

$$\dots \quad (40)$$

$$A_\ell = B_{\ell-1}, B_\ell = A_{\ell-1} \bmod B_{\ell-1} \quad (41)$$

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

23 / 69

不妨设 $\deg A > \deg B$.

$$A_0, B_0 \quad (42)$$

$$A_1 = B_0, B_1 = A_0 - B_0 \cdot Q_1 \quad (43)$$

$$A_2 = B_1, B_2 = A_1 - B_1 \cdot Q_2 \quad (44)$$

$$\dots \quad (45)$$

$$A_\ell = B_{\ell-1}, B_\ell = A_{\ell-1} - B_{\ell-1} \cdot Q_\ell, \quad (46)$$

- 如果有一个函数 $\text{HalfGCD}_N(A, B)$ 将两个次数 $< N$ 的多项式求出 Q_1, \dots, Q_ℓ 使得 $\deg B_\ell < N/2$, 但 $\deg A_\ell \geq N/2$...
- 那么调用 $\log N$ 次就可以得到完整的 Euclid 过程商的序列, 而且复杂度可以被主定理控制.

折半 Euclid 算法 [Knuth 1970] [Schönhage 1971] [Moenck 1973]

不妨设 N 是 2 的幂, $\deg A > \deg B$.

- 函数 $\text{HalfGCD}_N(A, B)$ 的目的: 将两个次数 $< N$ 的多项式求出 Q_1, \dots, Q_ℓ 使得 $\deg B_\ell < N/2$, 但 $\deg A_\ell \geq N/2$.
- 前期的计算不会影响太低位: $\deg(Q_1 \cdots Q_\ell) = \deg A_0 - \deg A_\ell < N/2$.
- 如果对于 $A = T^L A' + O(T^{L-1})$, $B = T^L B' + O(T^{L-1})$ 做 $\text{HalfGCD}_{N/2}(A', B')$...
- 由于我们可以写成矩阵,

$$A_i = B_{i-1}, B_i = A_{i-1} - B_{i-1} \cdot Q_i \iff \begin{pmatrix} A_i \\ B_i \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -Q_i \end{pmatrix} \begin{pmatrix} A_{i-1} \\ B_{i-1} \end{pmatrix}. \quad (47)$$

- 有

$$\begin{pmatrix} 0 & 1 \\ 1 & -Q_\ell \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 \\ 1 & -Q_1 \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix} = T^L \begin{pmatrix} A' \\ B' \end{pmatrix} + O(T^{L-1}) \cdot O(T^{N/4-1}). \quad (48)$$

折半 Euclid 算法 [Knuth 1970] [Schönhage 1971] [Moenck 1973]

- 如果对于 $A = T^L A' + O(T^{L-1})$, $B = T^L B' + O(T^{L-1})$ 做 $\text{HalfGCD}_{N/2}(A', B')$, 有

$$\begin{pmatrix} 0 & 1 \\ 1 & -Q_\ell \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 \\ 1 & -Q_1 \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix} = T^L \begin{pmatrix} A' \\ B' \end{pmatrix} + O(T^{L-1}) \cdot O(T^{N/4-1}). \quad (49)$$

- 第一次取 $L = N/2$, 将 (A, B) 约化到 $\deg A \geq 3/4 N > \deg B$, 然后做一次取模.
- 第二次取 $L = N/4$, 将 (A, B) 约化到 $\deg A \geq 1/2 N > \deg B$.
- 这总共调用了两次分治, 有

$$T(N) = 2T(N/2) + \mathcal{O}(M(N)), \quad (50)$$

解得 $T(N) = \mathcal{O}(M(N) \log N)$.

连分式展开

我们求出的序列的一种直观解释:

$$\frac{A}{B} = Q_1 + \frac{A \bmod B}{B} \quad (51)$$

$$= Q_1 + \frac{1}{B/(A \bmod B)} \quad (52)$$

$$= Q_1 + \frac{1}{Q_2 + \frac{1}{Q_3 + \ddots}} \quad (53)$$

$$:= [Q_1; Q_2, \dots, Q_\ell]. \quad (54)$$

线性递推式重建

- 已知一个 $\leq N$ 阶线性递推式的前 $2N$ 项 a_0, \dots, a_{2N-1} , 求递推式.
- 我们知道, 这是要找 P, Q 满足 $A \equiv P/Q \pmod{x^{2N}}$, 且 $\deg P < N, \deg Q \leq N$.

定义 (Padé 逼近)

给定 $A(x)$, 求 $\deg P \leq N_1, \deg Q \leq N_2$ 满足

$$P - AQ \equiv 0 \pmod{x^{N_1+N_2+1}}. \quad (55)$$

定义 (有理函数重建)

给定 $A(x)$ 和模 $M(x)$, 求 $\deg P \leq N_1, \deg Q \leq N_2$ 满足

$$P - AQ \equiv 0 \pmod{M}, \quad (56)$$

其中 $\deg M = N_1 + N_2 + 1$.

连分式展开 \Rightarrow 有理函数重建

定义 (有理函数重建)

给定 $A(x)$ 和模 $M(x)$, 求 $\deg P \leq N_1, \deg Q \leq N_2$ 满足

$$P - AQ \equiv 0 \pmod{M}, \quad (57)$$

$$P = AQ + BM, \quad (58)$$

其中 $\deg M = N_1 + N_2 + 1$.

$$A_0 = M \quad (59)$$

$$A_1 \quad \overbrace{A_1}^{\deg < \deg A_0} = A = O(x^{\deg A_0 - \deg A_1}) \cdot (M, A) \quad (60)$$

$$A_2 \quad \overbrace{A_2}^{\deg < \deg A_1} = A_0 - A_1 \cdot Q_1 = O(x^{\deg A_0 - \deg A_1}) \cdot (M, A) \quad (61)$$

$$\dots \quad (62)$$

间奏：纠错码的快速算法

纠错码是通信的基础, 考虑固定的一个有限的字符集 Σ , 一个纠错码可以看做一个函数 $C: \Sigma^n \rightarrow \Sigma^m$ ($m \geq n$).

所以为了保证纠错码的可靠性, 我们关心“距离”:

$$d = \min_{\substack{x, y \in \Sigma^n \\ x \neq y}} \delta(C(x), C(y)), \quad (63)$$

其中 δ 是 Σ^m 上的 Hamming 距离. 易见, 如果传输中错误的字符数量 $< d/2$, 那么我们可以完美地恢复原来的信息.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

30 / 69

基于多项式求值的编码 [Reed-Solomon 1960]

令字符集为有限域 \mathbb{F} 满足 $|\mathbb{F}| \geq m$, 令 $\alpha_1, \dots, \alpha_m$ 是 \mathbb{F} 的 m 个不同的元素, 考虑如下的映射:

$$C(a_0, \dots, a_{n-1}) \mapsto \left(\sum_{j=0}^{n-1} a_j \alpha_i^j \right)_{i=1}^m, \quad (64)$$

也即将信息 a_0, \dots, a_{n-1} 看做一个 $n-1$ 次多项式 $f(x) = \sum_{j=0}^{n-1} a_j x^j$, 在 m 个给定点处的取值.

注意到 n 个点值就足够确定一个 $n-1$ 次多项式, 所以 **Reed-Solomon** 码的距离满足 $d \geq m - n + 1$.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

31 / 69

Reed-Solomon 编码的快速纠错 [Berlekamp-Welch 1986]

给一个 $n-1$ 次多项式 $f(x)$, 如果 $f(\alpha_1), \dots, f(\alpha_m)$ 中有 $\leq (m-n)/2$ 个错误, 一定可以唯一地还原出 f 的系数, 但是如何快速计算?

不妨设给定的 y_1, \dots, y_m 确定出来的多项式是 $g(x)$, 那么 $g(x) - f(x)$ 只在 β_1, \dots, β_k 上非零, 所以

$$(g(x) - f(x))(x - \beta_1) \cdots (x - \beta_k) \equiv 0 \pmod{(x - \alpha_1) \cdots (x - \alpha_m)}, \quad (65)$$

注意写作 $r(x) = (x - \beta_1) \cdots (x - \beta_k)$, 那么

$$\deg r + \deg(f \cdot r) = k + (n-1+k) \leq m-1, \quad (66)$$

可以考虑直接对 $g(x)$ 进行有理函数重建.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

32 / 69

更加一般的问题

定义 (Hermite-Padé 逼近)

给定多项式 A_1, \dots, A_m 和度数限制 d_1, \dots, d_m , 求 P_1, \dots, P_m 使得

- $\deg P_i < d_i$,
- $P_1 A_1 + \cdots + P_m A_m \equiv 0 \pmod{x^{d_1 + \cdots + d_m - 1}}.$

Min25 BM

- 找寻整式递推式 可以直接对 $(A, A', \dots, A^{(m-1)})$ 调用 Hermite-Padé 逼近的算法.
- 找寻最小多项式 可以直接对 $(1, A, \dots, A^{m-1})$ 调用 Hermite-Padé 逼近的算法.
- 记 $\sigma = d_1 + \cdots + d_m - 1$, Hermite-Padé 逼近的时间复杂度可以在 $\tilde{O}(m^{\omega-1}\sigma)$ 的时间内完成, 其中 ω 是矩阵乘法的指数¹. [Labahn-Zhou 2012]

¹现在 $\omega \leq 2.371552$. [Williams-Xu-Xu-Zhou 2023]

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

33 / 69

解 Toeplitz 方程

定义 (Toeplitz 矩阵)

形如 $(a_{i-j})_{i,j}$ 的矩阵, 其中 a 是下标从 $-(N-1)$ 到 $N-1$ 的数列.

- Toeplitz 方程可以写作

$$a(T) \cdot x(T) = b(T) + \Omega(T^{N-2}) + O(T^{2N-1}), \quad (67)$$

- 转化成

$$a(T) \cdot \underbrace{x(T)}_{\deg < N} - b(T) \cdot \underbrace{1}_{\deg < 1} - 1 \cdot \underbrace{r(T)}_{\deg < N-1} \equiv 0 \pmod{T^{2N-1}}. \quad (68)$$

- 刚好 $N+1+(N-1)-1=2N-1$, 符合 Hermite-Padé 逼近的形式.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

34 / 69

解的多项式基

- 给定 $A \in \mathbb{F}[x]^m$, 定义 $V_s = \{P \in \mathbb{F}[x]^m : P \cdot A \equiv 0 \pmod{x^s}\}$.
- 记 $\deg(P) = \max_{1 \leq i \leq m} \{\deg(P_i)\}$, $\text{type}(P)$ 为取到最大值的最大的 i .
- V_s 的 **极小基**: 对每个 i , 选取 Q_i 是 $\text{type}(P)=i$ 中次数最小的一个.

性质:

- 确实是基: Q_1, \dots, Q_m 可以组合出 V_s 中的元素.
- 最优性: V_{md-1} 的一组极小基中, 次数最小的 Q_i 是 $\deg < d$ 的 Hermite-Padé 逼近的形式的一组解.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

35 / 69

极小基的维护

- 按照 s 从小到大的顺序逐渐维护 V_s 的一组极小基。
- V_0 是平凡情况, 有 $Q_{ij} = [i = j]$.
- 从 V_s 推到 V_{s+1} 的极小基 $\{\tilde{Q}_i\}$:
 - 如果 $x^{s+1} \mid Q_i \cdot A$, 可以直接保留, $\tilde{Q}_i = Q_i$.
 - 否则存在 x^s 次项, 设 ℓ 是这种情况的 i 中按照 (\deg, type) 的字典序比较下最小的那个.
 - 对于这样的 $i \neq \ell$, 通过 $\tilde{Q}_i = Q_i - \lambda \cdot Q_\ell$ 消去 x^s 次项 (为什么 $\text{type}(\tilde{Q}_i) = i$?)
 - 对于 ℓ , 必须有 $\tilde{Q}_\ell = x \cdot Q_\ell$. (总得牺牲一个)
- 正确性: 考虑消元.
- 时间复杂度: $\mathcal{O}(m^3 d^2) = \mathcal{O}(m \sigma^2)$. [Derksen 1994]
- 改进成一般情况: 把 \deg 的定义改为 $\deg(P) = \max_i \{\deg(P_i) - d_i\}$.
- 改进复杂度: 将 HalfGCD 的思想应用到上述过程!

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

36 / 69

大炮现状

定义 (Hermite 标准型)

任何一个多项式矩阵 $A \in \mathbb{F}[x]^{m \times m}$, 存在 $H = AU$ ($\det(U) \in \mathbb{F}^\times$) 使得

- H 是下三角矩阵.
- $\deg(H_{ij}) < \deg(H_{ii})$.
- 在 $\tilde{\mathcal{O}}(m^\omega \deg(A))$ 时间内计算 Hermite 标准型和 $\det(A)$. ($\deg(A)$ 是“平均次数”!) [Labahn-Neiger-Zhou 2012]
- Popov 标准型, s -约化型, ...
- Hermite-Padé 逼近, 但是从 $\text{mod } x^\sigma$ 改成 $\text{mod } M(x)$.
- 多项式矩阵的 “gcd”.
- ...

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

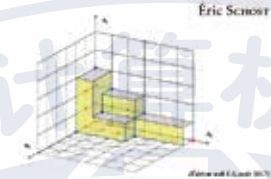
37 / 69

延伸阅读



Algorithmes Efficaces en Calcul Formel

Alain BOSTAN
Frédéric CHYZAK
Marc GUSTI
Romain LEBRATON
Grégoire LECHE
Bruno SALVY
Éric SCHOST



- von zur Gathen & Gerhard: 经典之作.
- AECF: 成书时间较新, 有网络版, 但是是法语.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

38 / 69

定义

定义 (整式递推)

P-Recursive

对于一个数列 $\{a_n\}_{n \geq 0}$, 我们称其为 **整式递推** 的, 当且仅当存在多项式 $p_0(x), p_1(x), \dots, p_m(x)$ 使得, 对于 $n \geq m$, 有

$$p_0(n)a_n + p_1(n)a_{n-1} + \dots + p_m(n)a_{n-m} = 0. \quad (69)$$

定义 (微分有限)

D-Finite

对于一个生成函数 $A(x)$, 我们称其为 **微分有限** 的, 当且仅当存在多项式 $f_0(x), \dots, f_m(x)$, 有

$$f_0(x)A(x) + f_1(x)A'(x) + \dots + f_m(x)A^{(m)}(x) = 0. \quad (70)$$

定理: $\{a_n\}$ 整式递推 $\iff A(x)$ 微分有限.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

39 / 69

为什么要研究整式递推？

- 大量的组合计数问题的数列最终都被发现是整式递推的。
- 以 k -正则图计数为例。
- 之前的思考题里提到了, $k=2$ 的时候生成函数形如

$$A(x) = \exp\left(-\frac{x}{2} - \frac{x^2}{4}\right) \frac{1}{\sqrt{1-x}}, \quad (71)$$

- 微分有限的生成函数关于各种运算 具有良好的封闭性, 我们之后会看到有办法直接证明上述序列是整式递推的。
- 随后, $k=3$ 和 $k=4$ 的情况也被证明是整式递推的, 这一方法并非出于对于正则图组合结构的归纳. [Goulden-Jackson 1986]
- 之后, 任意 k 的情况也被证明. [Gessel 1990] 他们的证明方法源于发展了 无穷元对称微分有限生成函数 的理论。

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

40 / 69

作为一个关心计算的人, 我为什么要研究整式递推？

- ∂ 连同四则运算, 很大程度上勾勒了我们有哪些工具计算一个序列。
- 生成函数的层级:

有理 \subset 代数 \subset 微分有限 \subset ^{D-Algebraic} 微分代数 $\subset \mathbb{F}[x]$

- 如果是有理函数, 我们可以 $O(\log N)$ 就计算出第 N 项, 相当快速。
- 如果微分有限, 我们可以 $O(M(\sqrt{N}))$ 就计算出第 N 项 [Chudnovsky-Chudnovsky 1988] [Bostan-Gaudry-Schost 2007], 或者 $O(N)$ 计算出前 N 项。
- 如果微分代数, 我们至少可以用半在线卷积的方法 $O(R(N))$ 计算出前 N 项。

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

41 / 69

题外话：疑难数列之整数拆分

- 整数拆分的生成函数

$$F(x) = \prod_{n=1}^{\infty} \frac{1}{1-x^n}, \quad (72)$$

看起来应该是非常困难的序列, 但它居然是 微分代数的!

$$4F^3F'' + 5xF^3F''' + x^2F^3F^{(4)} - 16F^2F'^2 - 15xF^2F'F'' + 20x^2F^2F'F''' - 39x^2F^2F''^2 + 10xFF'^3 + 12x^2FF'^2F'' + 6x^2F'^4 = 0. \quad (73)$$

Modular Form

- 深层原因或许要涉及 模形式, 这是现代数学的重要分支之一, 远远超出了本次讲故事的狩猎范围.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

42 / 69

题外话：疑难数列之禁子结构排列

- 给定一个有限个排列构成的集合 \mathcal{P} , 记 $A_n(\mathcal{P})$ 是所有没有出现过 \mathcal{P} 中结构的 n 阶排列的数量.
- 例如 $A_n(\{(123)\})$ 就是 Catalan 数 C_n .
- 但到了形态有 4 阶排列, 就已经出现了没法解决的问题了, 例如 $A_n(\{(1324)\})$.
 - ▶ 人们现在 不知道 如何在多项式时间内计算 $A_n(\{(1324)\})$. 尽管人们现在已经确认计算 $A_n(\mathcal{P})$ 很可能是困难的. (如果存在多项式时间算法计算 $A_n(\mathcal{P}) \bmod 2$, 那么 $\text{EXP} = \oplus \text{EXP}$. [Garra-brant-Pak 2015])
 - ▶ 人们现在 不知道 如何证明 $A_n(\{(1324)\})$ 并非整式递推. 尽管已经知道存在 $\mathcal{P} \subset \mathfrak{S}_{80}$ 使得 $A_n(\mathcal{P})$ 不是整式递推. [Garra-brant-Pak 2015]
 - ▶ 人们现在甚至不能证明 $A_n(\{(1324)\})$ 的渐进行为, 只能根据 $n \leq 50$ 的数值结果做出猜测: [Conway-Guttmann-Zinn-Justin 2017]

$$A_n(\{(1324)\}) \sim C \cdot \lambda^n \mu^{\sqrt{n}} n^\alpha. \quad (74)$$

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

43 / 69

作为一个关心封闭形式的人, 我为什么要研究整式递推?

- 首先, 大部分数列如果有比较 正常 的封闭形式, 这个封闭形式也很多时候是整式递推的.
- 如何证明 $A = B$? 其中 A, B 都是组合求和式.
 - 在计算机的辅助下, 设计对应的消元算法, 分别得到 A, B 的一个整式递推式.
 - 找到它们公共满足的整式递推式.
 - 暴力验证前面充分多项, 说明两个数列相等.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

44 / 69

Gessel 游走

- 定义 $q(i, j; n)$ 是满足如下要求的 n 步游走, 从 $(0, 0)$ 到达 (i, j) 的方案数:
 - 途中位置只能在 $\mathbb{Z}_{\geq 0}^2$ 上.
 - 每一步只能是 $\{\leftarrow, \rightarrow, \nearrow, \searrow\}$.
- Gessel 猜想 $q(0, 0; 2n) = 16^n \frac{(5/6)_n (1/2)_n}{(2)_n (5/3)_n}$, 在计算机的辅助下被证明. [Kauers-Koutschan-Zeilberger 2008]
- 过了很久才得到一个完全由人类完成的证明, 并且过程并不初等, 用到了椭圆函数. [Bostan-Kurkova-Raschel 2017]

Proof of the algebraicity of the trivariate GF. We start by proving the algebraicity of $Q(0, y)$ as a function of y, z . We consider the representation of $r_y(\omega)$ given in Theorem 3 and apply eight times the addition theorem (P4) for ζ -functions, namely (for suitable values of $k \in \mathbb{Z}$ that can be deduced from (21))

$$\zeta_{1,3}(\omega - k\omega_2/8) = \zeta_{1,3}(\omega) - \zeta_{1,3}(k\omega_2/8) + \frac{1}{2} \frac{\wp'_{1,3}(\omega) + \wp'_{1,3}(k\omega_2/8)}{\wp_{1,3}(\omega) - \wp_{1,3}(k\omega_2/8)}.$$

We then make the weighted sum of the eight identities above (corresponding to the good values of k in (21)); this way, we obtain

$$r_y(\omega) = U_1(\omega) + U_2 + U_3(\omega).$$

图: 证明一瞥

- 大结局: 在计算机的辅助下被证明 整个 3 维序列的生成函数

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

45 / 69

线性代数 101

- 域 \mathbb{F} 上的一个线性空间 V 是一个集合, 配备加法和对 \mathbb{F} 的数乘运算, 满足线性性 ($\alpha, \beta \in \mathbb{F}, u, v \in V$):
 - $\alpha \cdot (u + v) = \alpha \cdot u + \alpha \cdot v,$
 - $(\alpha + \beta) \cdot u = \alpha \cdot u + \beta \cdot v,$
 - $\alpha \cdot (\beta \cdot v) = (\alpha\beta) \cdot v.$
- 我们会用到的线性空间: 多项式 $\mathbb{F}[x]$, 形式幂级数 $\mathbb{F}[[x]]$, 形式 Laurent 级数 $\mathbb{F}((x))$, 数列 $\mathbb{F}^{\mathbb{Z}_{\geq 0}}, \dots$
- 线性变换 $T: V \rightarrow W$ 是一个函数, 保持线性性:
 - $T(u + \alpha v) = Tu + \alpha Tv.$
- 我们关心的运算 ∂ 在 $\mathbb{F}[x]$, $\mathbb{F}[[x]]$ 和 $\mathbb{F}((x))$ 都是 \mathbb{F} -线性的.
- 每个线性空间都有个维数, 是它的基的大小, 如果基的大小是有限的, 称是有限维线性空间 ($\dim V = n < \infty$).
- 比如 $\mathbb{F}[x]$ 不是有限维的, \mathbb{F}^n 是 n 维的, 次数 $\leq n$ 的多项式是 $n+1$ 维的.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

46 / 69

微分有限的线性空间表述

- 有理分式 $\mathbb{F}(x)$ 由于可以做除法, 所以它是个域.
- $\mathbb{F}((x))$ 不仅是 \mathbb{F} -线性空间, 还具有 $\mathbb{F}(x)$ -线性结构!
 - 商 $\mathbb{F}[x]^{-1}\mathbb{F}[x] \hookrightarrow \mathbb{F}((x))$ 容易识别, 这一嵌入进一步是同构, 但在多元情况并不典范.

定义 (微分有限)

一个生成函数 $F(x)$ 是微分有限的当且仅当

$$\mathcal{D}F := \text{span}_{\mathbb{F}(x)} \{ \partial^k F : k \in \mathbb{Z}_{\geq 0} \} \quad (75)$$

是有限维 $\mathbb{F}(x)$ -线性空间.

- 这和我们之前的定义等价, 但更容易推广到多元情况.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

47 / 69

微分有限的基本性质

定理

若 $F(x), G(x)$ 微分有限, 则 $F+G$ 微分有限.

证明.

令 $\{\alpha_1, \dots, \alpha_n\}$ 为 $\mathcal{D}F$ 的基, $\{\beta_1, \dots, \beta_m\}$ 为 $\mathcal{D}G$ 的基, 那么

$$\partial^k(F+G) = \partial^k F + \partial^k G = \sum a_i \alpha_i + \sum b_i \beta_i. \quad (76)$$

故 $\dim_{\mathbb{F}(x)} \mathcal{D}(F+G) \leq n+m < \infty$. □

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

48 / 69

微分有限的基本性质

定理

若 $F(x), G(x)$ 微分有限, 则 FG 微分有限.

证明.

令 $\{\alpha_1, \dots, \alpha_n\}$ 为 $\mathcal{D}F$ 的基, $\{\beta_1, \dots, \beta_m\}$ 为 $\mathcal{D}G$ 的基, 那么 $\partial^k(FG)$ 被 $\partial^i F \partial^j G$ 线性表出, 因此被 $\{\alpha_i \beta_j\}$ 线性表出.

故 $\dim_{\mathbb{F}(x)} \mathcal{D}(FG) \leq nm < \infty$. □

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

49 / 69

微分有限的基本性质

定理

若 $F(x), G(x)$ 微分有限, 则 $F+G$ 微分有限.

定理

若 $F(x), G(x)$ 微分有限, 则 FG 微分有限.

- 如何在计算上得到之前定义的对应的方程?
- F/G 并不一定是微分有限的:

$$\frac{x}{\exp x - 1}. \quad (77)$$

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

50 / 69

代数幂级数

定义 (代数幂级数)

$F(x) \in \mathbb{F}[[x]]$ 是代数的当且仅当有多项式方程 $P \in \mathbb{F}[X, T]$ 满足

$$P(x, F) = 0. \quad (78)$$

代数幂级数关于除法是封闭的!

定理

代数幂级数构成域, 也即 $F+G, FG, F/G$ 都是代数幂级数.

加法和乘法略去, 仅勾勒除法的封闭性.

证明.

如果 $P(x, F) = 0$, 写作 $P(X, T) = \prod_{\alpha} (T - \alpha(x))$, 那么

$$\prod_{\alpha} (T - 1/\alpha) = 0 \implies \prod_{\alpha} (\alpha T - 1) = 0 \implies T^{\deg P} P(x, 1/T) = 0. \quad (79)$$

□

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

51 / 69

代数幂级数的微分有限性

定理

如果 $u(x)$ 是代数的, 那么它是微分有限的.

证明 (勾勒).

设 u 满足一个次数为 n 的代数方程 $P(x, u) = 0$, 求导可以得到 $\mathcal{D}u \in \text{span}_{\mathbb{F}(x)}\{u^k : 0 \leq k < n\}$. □

后者其实就是代数扩张 $\mathbb{K} := \mathbb{F}(x)(u)$.

定理

如果 $F(x)$ 是微分有限的, $u(x)$ 是代数的, 那么 $F(u(x))$ 是微分有限的.

证明 (勾勒).

只需证明 $\mathcal{D}^k(F \circ u)$ 张成的是有限维 \mathbb{K} -空间. □

大炮开火!

- 2-正则图构成的组合类, 其指数型生成函数是

$$A(x) = \exp\left(-\frac{x}{2} - \frac{x^2}{4}\right) \frac{1}{\sqrt{1-x}}. \quad (80)$$

- $\exp\left(-\frac{x}{2} - \frac{x^2}{4}\right)$ 是微分有限的, 因为
 - $\exp x$ 是微分有限的.
 - $-\frac{x}{2} - \frac{x^2}{4}$ 是有理的, 所以是代数的.
- $\frac{1}{\sqrt{1-x}}$ 是微分有限的, 因为它是代数的.
- 因此 $A(x)$ 是微分有限的.

更多代数幂级数

二元分式

$$\frac{1}{1-x-y} = \sum_{n,m \geq 0} \binom{n+m}{n} x^n y^m \quad (81)$$

的对角线是

$$\sum_{n=0}^{\infty} \binom{2n}{n} x^n = \frac{1}{\sqrt{1-4x}}, \quad (82)$$

根据后见之明, 可以用广义二项式, 但如何直接得到?

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

54 / 69

有理分式的对角线

定理

对于有理分式 $Q(x, y)$, 其对角线 $\sum_{n=0}^{\infty} ([x^n y^n] Q) T^n$ 是代数的.

证明 (勾勒).

欲提取 $Q(S, T/S)$ 的 S^0 次项, 分式分解展开成

$$Q(S, T/S) = \sum_{\alpha_i, \beta_i, n_i} \frac{\alpha_i}{(S - \beta_i)^{n_i}}, \quad (83)$$

提取其常数项, 一些代数函数的和, 还是代数的. □

为什么有的向 $+\infty$ 方向展开, 有的向 $-\infty$ 方向展开?

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

55 / 69

代数方程

- 在 $\mathbb{Z}_{\geq 0}$ 上行走, 有 a, b, c, d 种方法走 $+1, +2, -1, -2$ 的位移, 有多少种走法走 n 步从原点回到原点?
- 设 $F_{0,0}, F_{1,0}, F_{0,1}, F_{1,1}$, 分别表示最初和最终欠几步.

$$F_{0,0} = \frac{1}{1 - x^2(acF_{0,0} + adF_{1,0} + bcF_{0,1} + cdF_{1,1})}, \quad (84)$$

$$F_{1,0} = x(cF_{0,0} + dF_{0,1}) \cdot F_{0,0}, \quad (85)$$

$$F_{0,1} = x(aF_{0,0} + bF_{1,0}) \cdot F_{0,0}, \quad (86)$$

$$F_{1,1} = F_{0,0} + x(cF_{0,0} + dF_{0,1}) \cdot F_{0,0} \cdot x(aF_{0,0} + bF_{1,0}). \quad (87)$$

- 数值实验表明 $F_{0,0}$ 满足一个次数 ≤ 6 的代数方程.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

56 / 69

代数方程

- $F_{0,0}$ 满足一个次数 ≤ 6 的代数方程.
- 我们有四个未知量, 列了四个方程.
- 记 $\mathbb{K} = \mathbb{F}(x, a, b, c, d)$.

Bézout 定理

“一般”的 n 个 n 元多项式方程组, 次数分别为 d_1, \dots, d_n , 在 \mathbb{K} 上有 $d_1 \cdots d_n$ 个解.

- 所以容易用 Hermite–Padé 逼近猜出最小多项式.
- 一个构成证明的计算过程可以考虑使用 Gröbner 基, 给 $\mathbb{K}[F_{0,0}, F_{1,0}, F_{0,1}, F_{1,1}]$ 设置主元的“顺序” $F_{0,0} < F_{1,0} < F_{0,1} < F_{1,1}$, 最后会消出一个只含 $F_{0,0}$ 的多项式, 这就给出了 $F_{0,0}$ 满足的一个多项式方程.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

57 / 69

多元整式递推: 如何定义?

- Zeilberger 最初的尝试: 一个 $f: \mathbb{Z}_{\geq 0}^n \rightarrow \mathbb{F}$ 的 n 元数列, 对于每个 $1 \leq i \leq n$, 存在一组多项式 $P_j^{[i]} (1 \leq j \leq r_i)$, 使得满足递推式

$$\sum_{j=0}^{r_i} P_j^{[i]}(\mathbf{m}) f(m_1, \dots, m_i - j, \dots, m_n) = 0. \quad (88)$$

- 这个定义有严重的问题! Stanley 给出了一个例子: 对于 $f(n, m)(n^2 - m) = 0$ 这个方程, 有一组解

$$\sum_{n=0}^{\infty} x^n y^{n^2}, \quad (89)$$

但是这个函数的性质相当复杂, 是我们想要排除的. [Gessel 90]

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

58 / 69

多元微分有限

定义 (多元微分有限)

一个 n 元生成函数 $F(x_1, \dots, x_n) \in \mathbb{F}[[x_1, \dots, x_n]]$ 是 **微分有限** 的当且仅当

$$\mathcal{D}F := \text{span}_{\mathbb{F}(x_1, \dots, x_n)} \left\{ \partial_{x_1}^{k_1} \cdots \partial_{x_n}^{k_n} F : \mathbf{k} \in \mathbb{Z}_{\geq 0}^n \right\} \quad (90)$$

是有限维 $\mathbb{F}(x_1, \dots, x_n)$ -线性空间.

- 等等, 如何赋予 $\mathbb{F}[[x_1, \dots, x_n]]$ 以 $\mathbb{F}(x_1, \dots, x_n)$ -线性结构? (考虑如何展开 $1/(x-y)$)
- 当然嵌入到 $\mathbb{F}\langle x_1 \rangle \cdots \langle x_n \rangle$ 是一种办法, 不过这钦定了一个顺序, 并不典范.
- 一个重要的 等价定义: 放宽成对于每个 $1 \leq i \leq n$,

$$\dim \text{span}_{\mathbb{F}(x_1, \dots, x_n)} \left\{ \partial_{x_i}^k F : k \in \mathbb{Z}_{\geq 0} \right\} < \infty \quad (91)$$

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

59 / 69

多元微分有限的基本性质

定理

如果 $F, G \in \mathbb{F}[x_1, \dots, x_n]$ 是微分有限的, $u_1, \dots, u_n \in \mathbb{F}[t_1, \dots, t_m]$ 是代数的, 那么以下生成函数微分有限:

- $F + G$.
- $F \cdot G$.
- 良定义的 $F(u_1, \dots, u_n)$.
- 对角线 [Lipshitz 1988]

$$\sum_{i_1, \dots, i_{n-1} \in \mathbb{Z}_{\geq 0}} f_{i_1, \dots, i_{n-1}, i_{n-1}} x_1^{i_1} \dots x_{n-1}^{i_{n-1}} \quad (92)$$

终结比赛的对角线 [Lipshitz 1988]

- 几乎所有正常的和式都是微分有限的!

$$c_{i,j} = \sum_k a_{i,k} b_{k,j}. \quad (93)$$

- 首先 $A(X, Y)B(Y', Z)$, 然后缩并 Y 和 Y' , 然后带入 $Y = 1$.

证明勾勒: 对角线的微分有限性 [Lipshitz 1988]

为了使得呈现更加清晰, 我们只证明二元情况 $(F(x, y))$, 多元情况可以照猫画虎.

- ① 换元为 $G = s^{-1}F(x/s, s)$, 这是关于 x, s “微分有限” 的, 不是形式幂级数, 但是仍然满足前述的 $\dim \mathcal{D}G < \infty$.
- ② 证明存在非零解满足

$$\sum_{k, \ell} p_{k, \ell}(x) \partial_x^k \partial_s^\ell G = 0. \quad (94)$$

- ③ 设 o 是 ∂_s^ℓ 的系数不为零的最小的 ℓ , 那么上式的 s^{-o-1} 次项系数给出等式

$$\sum_k p_{k, o}(x) \partial_x^k ([s^{-1}]G) = 0. \quad (95)$$

- ④ 说明 $[s^{-1}]G$ 也即 F 的对角线微分有限! □

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

62 / 69

关键引理 [Lipshitz 1988]

欲证 $\sum_{k, \ell} p_{k, \ell}(x) \partial_x^k \partial_s^\ell G = 0$.

- 根据微分有限性条件, 得到多项式方程 ($\deg L = \ell$)

$$L(x, s) \partial_x^d G = O(s^d, \partial_x^{d-1})G, \quad (96)$$

$$L(x, s) \partial_s^d G = O(s^d, \partial_s^{d-1})G. \quad (97)$$

- 考虑一个大 N , 以及所有 $\alpha + \beta \leq N$, 考虑 $L^N \partial_x^\alpha \partial_s^\beta G$, 通过上述方程不断约化为

$$L^N \partial_x^\alpha \partial_s^\beta G = O(s^{(d+\ell)N}, \partial_x^{d-1}, \partial_s^{d-1})G. \quad (98)$$

- 全体 α, β 一共有 $\Omega(N^2)$ 种选择, 但右侧的 $s^i \partial_x^j \partial_s^k$ 只有 $\mathcal{O}(N)$ 种情况, 所以当 N 充分大, 一定可以将左侧 $\mathbb{F}(x)$ -线性组合得到右侧为 0.

- 计算这种解的任务一般被称作计算 ^{syzygy} 合冲.

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

63 / 69

来不及讲的话题

- 小模数 p ?
 - 固定模数 p , 代数幂级数的单项求值都有“数位 DP”算法.
 - p -自动机和代数幂级数的等价性.
 - 整式递推除以 0 ?
- Weyl 代数 $\mathbb{F}[x, \partial]$ 和 Ore 代数 $\mathbb{F}[x_1, \dots, x_n, \partial_1, \dots, \partial_n]$?
 - 快速计算乘法 (矩阵乘法)?
 - 不交换的代数结构, 但是可以定义一个方向的 Euclid 算法和 gcd.
- q -整式递推?
 - 咬文嚼字: [二项式 / 整式递推 / 超几何级数 / ...] 的 q -类比, 或者 q -[二项式 / 整式递推 / 超几何级数 / ...], 而不是单独说“ q -类比”?

“还有许多问题我愿意告诉你们, 但是你们现在尚不能接受.”

— A. Кострикин, 代数学引论

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

64 / 69

延伸阅读



李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

65 / 69

后日谈：整式递推在 OI 中的未来

- “很多序列都是整式递推的”，这是一个对于我们理解问题的正面消息，同时也是对出题人的新考验。
- 出题人：我动用了许多智慧，最后得到了这个问题答案的递推式！
- 选手：跑几项暴力，Min25 BM 直接秒了，真简单！

- 出题人：

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

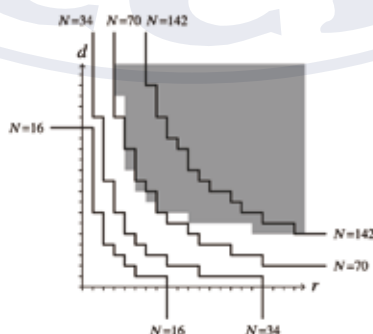
66 / 69

没有绝对的“最小递推式”，只有 Pareto 最优！

Example 1.36 The formal power series

$$\frac{1+x^5}{\sqrt{x+1}} + \frac{2x+3}{\sqrt{1-x}} + (3x^4 - 4x^3 + 8) \exp\left(\frac{x}{1-x}\right) = 12 + 11x + \frac{29}{2}x^2 + \dots$$

satisfies a differential equation of order r with polynomial coefficients of degree d for every point (r, d) in the gray region in the figure below.



图：递推式的长度-多项式次数的权衡 [Kauers 2023]

李白天 (IIIS, THU)

组合计数中的递推问题

2024 年 2 月 2 日

67 / 69

没有一劳永逸的方法

- 微分有限这一定义本身并不能完整捕捉 ^{effective} 有效 的递推式。
- 回归一个古老的启蒙问题: 给定多项式 $f(x)$, 求出 $f(x)^n$ 的各项系数.
 - 记 $g(x) = f(x)^n$, 那么可以利用 $g'f = nf'g$ 来递推.
 - 如果 $f(x)$ 不是低次多项式, 而是 稀疏多项式, 方法仍然奏效, 但难以用 Min25 BM 解决.
 - 稀疏整式递推, 需要理解操作原理才能得到 有效 递推式的问题.
- 隐式整式递推: 答案序列 $f(n)$ 本身并非整式递推, 但是计算某一项的时候具有整式递推的内核.
 - 截取-Taylor-截取: $\mathcal{O}(n)$ 计算 Bernoulli 数 $B_n = [x^n/n!]_{\exp x - 1}$.
 - 思考题: $\mathcal{O}(n)$ 计算 n 个顶点的, 不存在 2 度点的图的数量.
- 相信大家的智慧!



杂题选讲

清华大学 刘一平

杂题选讲

Longest Trip

来源：IOI2023 Day1 T2

交互题。有一张 n ($1 \leq n \leq 256$) 个点的未知的无向图，保证任意三个不同点构成的子图中至少有 D ($1 \leq D \leq 3$) 条边。你每次可以给出两个不交点集 A, B ，查询图中有无 A 到 B 的边。请通过至多 400 次查询，求出图中的任意一条最长简单路径（不经过重复点的路径）。

1s, 2048M

杂题选讲

Soccer Stadium

来源：IOI2023 Day1 T3

给定一个 $n \times n$ ($n \leq 2 \times 10^3$) 的棋盘，每个格子是空的或障碍，你要选择一些空格子形成一个球场，问最多选择多少格子。

一个球场满足：任意两个球场内的格子可以通过至多两步移动到达。一步移动为：水平或竖直移动若干个格子，中途不能改变方向，只能经过球场内的格子。

3.5s, 2048M



杂题选讲

Yet Another Partiton Problem

来源：CF1175G

给定长度为 n ($n \leq 2 \times 10^4$) 的非负整数序列，要划分为恰好 k ($k \leq 100$) 段，每段的代价是它的长度乘最大值，问最小代价和。

5s, 512M

杂题选讲

RPSRobots

来源：TopCoder 14379

有 n ($n \leq 10^5$) 个剪刀石头布机器人，每个机器人的策略 s_i 是一个长度为 k ($k \leq 18$) 的由剪刀石头布组成的序列。

两个机器人 A, B 决斗时，他们会首先将各自的序列循环移位，然后按照各自的串比 k 轮，第 i 轮出自己串中第 i 个手势。称这对机器人是“稳定的”当且仅当无论它们如何移位，最后 A 赢、B 赢、平局的次数都不变。

求有多少个非空集合，满足集合中任意一对不同机器人都是稳定的，答案模 $10^9 + 7$ 。

杂题选讲

完美的集合

来源: loj2462

给定一棵 n ($n \leq 60$) 个点的树, 边有长度 ($\leq 10^4$), 点有重量 $1 \leq w_i \leq 10^9$ 和价值 $0 \leq v_i \leq 10^9$,

定义一个连通块是完美的, 当且仅当其中点的重量之和 $\leq M$ ($M \leq 10^4$) 且在此前提下价值和最大。

求选择 K ($K \leq 10^9$) 个完美连通块的方案数, 模 5^{23} , 使得存在一个节点 x 满足:

- x 被所有连通块包含。
- 所有连通块中的点 y 都满足 $\text{dist}(x, y) \cdot v_y \leq \text{Max}$ 。 ($\text{Max} \leq 10^{18}$)

2s, 512M

杂题选讲

完美的队列

来源: loj2461

有 n ($n \leq 10^5$) 个队列, 队列 i 有容量限制 a_i , 初始所有队列为空。

有 m ($m \leq 10^5$) 次操作, 每次操作先给某区间里的队列 push 一个数, 再对超过容量限制的队列 pop, 然后查询所有队列中有多少种不同的数。注意每次操作后都要输出答案。

5s, 256M

杂题选讲

通道

来源：loj2339

给定三棵 n ($n \leq 10^5$) 个点的树，边带非负整数权值。设 $d_i(u, v)$ 表示第 i 棵树中 u 到 v 的最短距离。求

$$\max_{1 \leq u, v \leq n} (d_1(u, v) + d_2(u, v) + d_3(u, v))$$

4s, 1024M

杂题选讲

Mr. Kitayuta's Gift

来源：CF506E

给定由小写字母构成的字符串 s ($1 \leq |s| \leq 200$)，要执行 n ($1 \leq n \leq 10^9$) 次操作，每次往其中任意位置插入一个小写字母。问最后能得到多少种不同的回文串，答案模 10007。

6s, 768M。

◀ ▶ 🔍 ↺ ↻ ⌂

随机化和近似算法选讲

南京外国语学校 程思元

[IOI2023] overtaking

- 有一条长度为 L 公里的公路， $N + 1$ 辆巴士在这条公路上行驶，巴士从 0 到 N 依次编号。巴士 i ($0 \leq i < N$) 计划在活动的第 $T[i]$ 秒从机场出发，行驶一公里用时 $W[i]$ 秒。巴士 N 是备用巴士，行驶一公里用时 X 秒。
- 公路上一共有 M 个调度站，巴士只能允许在调度站超车。起点和终点一定时调度站。
- 每辆巴士都以指定的最快速度行驶，除非它遇到前面有比它慢的巴士且无法超车。
- Q 次询问若备用巴士在时刻 Y 从起点出发，到底终点的时间。
- $1 \leq L \leq 10^9, 1 \leq N \leq 1000, 2 \leq M \leq 1000, 1 \leq Q \leq 10^6, 0 \leq T[i], Y \leq 10^{18}, 1 \leq W[i], X \leq 10^9$ 。

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

2 / 19

[IOI2023] beechtree

- 有一棵以 0 为根的 N 个点的有根树，每条边都有某种颜色。
- 对于某个 r 以及子树 $T(r)$ 中结点的某个置换 $v_0, v_1, \dots, v_{|T(r)|-1}$ 。令 $f(i)$ 为颜色 $C[v_i]$ 在长为 $i - 1$ 的颜色序列 $C[v_1], C[v_2], \dots, C[v_{i-1}]$ 中的出现次数。
- 置换 $v_0, v_1, \dots, v_{|T(r)|-1}$ 被称为是一个绝妙置换，当且仅当 $v_0 = r$ 且对于所有满足 $1 \leq i < |T(r)|$ 的 i ，结点 v_i 的父结点是 $v_{f(i)}$ 。
- 子树 $T(r)$ 是一棵绝妙子树，当且仅当 $T(r)$ 中结点存在某个绝妙置换。求每棵子树是否是绝妙的。
- $3 \leq n \leq 2 \times 10^5$ 。

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

3 / 19

「THUSCH 2017」巧克力

- 有一个 $n \times m$ 的矩阵，每个位置有一个 -1 或正整数的颜色和一个权值。
- 你需要选择一个连通块，使得连通块里不存在颜色是 -1 的格子，且连通块包含至少 k 种颜色。
- 最小化选出的连通块的大小，同时在此基础上最小化连通块权值的中位数。
- $1 \leq n \times m \leq 233, 1 \leq k \leq 5$ 。

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

4 / 19

「2020-2021 集训队作业」Storm

- 给出一个带点权 v_x 和边权 w_e 的无向图 $G(V, E)$ ($|V| = n, |E| = m$)，你需要选出一个大小不超过 K 的边集 $S \subseteq E$ (可以为空)，使得被边集中的边覆盖的点权值和减去边集的边权和尽量大。
- 形式化的说，你需要求出下面式子的值：

$$\max_{S \subseteq E, |S| \leq K} \sum_{x \in I(S)} v_x - \sum_{e \in S} w_e$$

其中 $I(S)$ 为 S 中至少有一条边相连的点集。

- $\sum 2^K (n + m) \leq 10^6$ 。

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

5 / 19

一道模拟赛题

- 有一个 $n \times n$ 的矩阵，第 i 行第 j 列的元素为 $a_{i,j}$ 。
- 给出两个长度为 n 的数组 k, w ，令 f_i 为第 i 行所有 $\binom{n+1}{2}$ 个区间中第 k_i ($1 \leq k_i \leq \binom{n+1}{2}$) 大的 mex 值，求 $\max_{i=1}^n f_i + w_i$ 。
- $1 \leq n \leq 10^4$ 。

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

6 / 19

[WC/CTS2023] 比赛

- 有 n 名学生，标号从 1 到 n ，他们一共加入了 m 个社团。任意两个社团至多只包含一名公共成员。
- 现在学校要组织一场比赛，想让这 n 名学生围成一个圈。为了防止作弊，校长希望圈上任意连续三个人不来自同一个社团。
- 构造一组圆排列学生的方案，或指出这样的方案并不存在。
- $\sum n \leq 2000$ 。

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

7 / 19

[CTSC2009] N^2 数码游戏

- 有一个 $n \times n$ 的网格，其中填的数是 $0 \sim n^2 - 1$ 的排列。你每次可以把 0 和它上、下、左、右中的一个数字交换。
- 给定初始状态，求尽可能小的操作次数使得 0 在右下角且其它位置第 i 行第 j 列上的数是 $(i-1) \times n + j$ 。
- 提交答案题，有 $n = 3, 4, 5, 6$ 的数据，其中 $n = 4, 5$ 各有一个数据满足初始状态恰好是最终状态的翻转。

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

8 / 19

低密度背包

- 0/1 背包问题：有 n 个正整数，选出一个子集使得它们的和等于 m ，或判断无解。
- 定义 0/1 背包问题的密度为 $\frac{n}{\log m}$ 。

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

9 / 19

低密度背包

- 0/1 背包问题：有 n 个正整数，选出一个子集使得它们的和等于 m ，或判断无解。
- 定义 0/1 背包问题的密度为 $\frac{n}{\log m}$ 。
- 求解密度为 n^2 的数据随机的 0/1 背包。

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

10 / 19

SQRT Problem¹

- 给出三个整数 n, a, b ，保证 n 是奇数且 $\gcd(n, a) = 1$ 。
- 求出一个 $x \in [1, n)$ 使得 $x^2 \equiv a \pmod n$ 且 $\left\lfloor \sqrt[3]{x^2} \right\rfloor = b$ 。
- $3 \leq n < 10^{100}, 1 \leq a, b < n$ 。

¹The 2023 ICPC Asia Hefei Regional Contest, A

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

11 / 19

[ZJOI2020] 密码

- 给定一个大质数 p 和一个随机范围值 err 。有一个未知的 $[0, p)$ 之间均匀随机生成的整数 x 。
- 给出 x 组信息 (a_i, c_i) ，每组信息满足 $a_i x + b_i \equiv c_i \pmod{p}$ ，其中 a_i 在 $[0, p)$ 之间随机生成， b_i 在 $[-err/2, err/2]$ 之间随机生成。
- 求出 x 的值。
- $10^{15} \leq p \leq 10^{18}, err \geq \frac{p}{100}, m \geq 50$ 。

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

12 / 19

卡字符串哈希

- 字符串哈希：哈希函数 $hash(\cdot) : \text{string} \rightarrow \mathbb{Z}$ 。
- 在 OI 中，一般要求能够快速求一个子串的哈希值，快速求两个串拼接的哈希值等操作。

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

13 / 19

卡字符串哈希

- 2-独立性：对于两个足够长的随机字符串 s, t ,
 $\Pr_{s,t}[\text{hash}(s) = \text{hash}(t)] = \frac{1}{p^2}$ 。
- 弱抗碰撞性：对于一个字符串 s , 不存在高效的算法找到一个字符串 t 使得 $s \neq t$ 但 $\text{hash}(s) = \text{hash}(t)$ 。
- 强抗碰撞性：不存在高效的算法找到两个字符串 s, t 使得 $s \neq t$ 但 $\text{hash}(s) = \text{hash}(t)$ 。
- 在OI中，我们只要求这个 hash function 是 2-独立的，但这个哈希函数不一定是抗碰撞的。

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

14 / 19

卡字符串哈希

- 一种 OI 中常见的字符串哈希方法：

$$\text{hash}(s) = \sum_{i=0}^{n-1} \text{ord}(s_i) e^{n-i-1} \bmod p$$

其中 $\text{ord}(s_i)$ 指 s_i 的 ASCII 码。

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

15 / 19

卡字符串哈希

- 一种 OI 中常见的字符串哈希方法：

$$\text{hash}(s) = \sum_{i=0}^{n-1} \text{ord}(s_i) e^{n-i-1} \bmod p$$

- 其中 $\text{ord}(s_i)$ 指 s_i 的 ASCII 码。
- OI 中常取多个不同的 e, p 分别计算哈希值，只有每个哈希值都相同才算相同。下面把选取 k 对 (e, p) 的哈希函数称作 k -哈希。
- 这个 hash 函数是 2-独立的，但它抗碰撞吗？

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

16 / 19

卡字符串哈希

- 生日攻击：随机 M 个字符串，计算它们的哈希值，找其中是否有哈希值相同的。取 $M = \Theta(\sqrt{p})$ 时有 $\Theta(1)$ 概率成功。
- 对于 1-哈希，直接使用生日攻击可以快速找到一对碰撞的 s, t 。
- 对于 2-哈希，生日攻击运行一段时间能找到碰撞。

程思元 (南京外国语学校)

随机化和近似算法选讲

2024 年 1 月 14 日

17 / 19



卡字符串哈希

- 对于 1-哈希，直接使用生日攻击可以快速找到一对碰撞的 s, t 。
- 对于 2-哈希，生日攻击运行一段时间能找到碰撞。
- 3-哈希？
- 5-哈希？
- 10-哈希？

计算机教育漫谈 —— 算法与教育

清华大学 刘润达

多重主题

以大学计算机专业的教学为讨论背景，探讨下列问题：

1. 有什么算法已经成为/可能成为教学的技术工具？
2. 优秀的教学技术工具 / 产品，除了算法，还需要什么？
3. 解决教学中的问题，除了算法工具，还需要什么？

讨论范围

- 数字化教学环境的“赛博空气”：重要，但本次课的体量无法覆盖。
- 网络、操作系统等基础设施；视频会议、办公软件等工具。
- 其中包含了很多算法和计算机技术，也有很多有趣的故事。
- 我们着重讨论：专门用于教育教学的算法和技术，以及它们如何与教育互动。
- 除了算法本身，还尝试讨论技术与人的关系。
- 不会面面俱到，讨论若干典型案例。
- 一个案例相对比较成熟（代码查重）
- 另外两个案例还在演进当中（学习数据分析、编程教学与大语言模型）

代码查重

code plagiarism detection

代码查重难在哪里？

- 计算机程序的“等价变换”难以穷尽。
 - C++: if - switch, for/while, 函数式编程lambda, 模板/继承.....
- 难以抑制学生对往年作业代码的传播
- 学生只要愿意，也可以设法获得查重工具。
- 足够短的代码片段，可能因为巧合而完全一致
- 不得已走到“查重”这一步，可能已经意味着教学的失败.....

问题背景

- 大学计算机课程普遍布置编程作业，作为重要的学习环节。
- 《数据结构》课程的编程作业，可能是类似信息竞赛的算法题
- 《计算机网络原理》课程的编程作业，可能是编写“迷你路由器”
- 作业布置下去之后，学生自己在OJ上提交测试。
- 这个过程中，可能查阅资料、和同学交流讨论。
- 问题：如何判断提交上来的作业是独立完成，而非复制粘贴？
- 问题：学生根据可能的“查重”方法，采取何种方式进行对抗？

极简版本：识别完全复制粘贴的情况

- 采用简单的字符串哈希算法，可识别原样复制粘贴的情况。
- 对抗查重的方法：增加任意修改，多打几个空格，修改几个字符的大小写。
- 查重方法升级：过滤空白符、无视大小写。
- 对抗查重的方法升级：修改变量名、函数名，调换函数的顺序。
- 查重方法升级：不能直接做字符串匹配，必须理解程序的含义

极简案例（实际查重的片段要长的多）

```
#include <stdio.h>
int main(){
    int a, b;
    scanf("%d %d", &a,
    &b);
    printf("%d", a+b);
    return 0;
}
```

(原版代码)

```
#include <stdio.h>
int main(){
    int a, b;
    scanf("%d %d", &a,
    &b);
    printf("%d", a+b);
    return 0;
}
```

(原样复制粘贴)

极简案例（实际查重的片段要长的多）

```
#include <stdio.h>
int main(){
    int num1, num2;
    scanf("%d %d", &num1,
&num2);
    printf("%d", num1+num2);
    return 0;
}
(更改变量名)
```

```
#include <stdio.h>
int main(){
    int    A,B;
    scanf("%d %d", &A, &B);
    printf("%d", A+B);
    return 0;
}
(修改空白符、修改大小写)
```

简化后的“编译原理”

- 编译器如何“理解”程序？
- 首先，将程序拆分为一串“单词”（token）。（词法分析）
- 一个变量名、一个括号、一个常数，都是一个“单词”。
- 然后，根据语法，将一串单词转换为“语法树”。
- （编译的后续步骤：将语法树转换为正确、高效的汇编语言机器码）
- 修改变量名、增加空白符，尽管从字符串匹配的角度上变动很大，
- 但是：不影响词法分析的结果，不影响语法树的形态

右侧为对应的词法分析结果

<pre>int main(){ int a, b; scanf("%d %d", &a, &b); printf("%d", a+b); return 0; }</pre> <p>(原版代码)</p>	<p>类型int 函数名main 左括号 右括号 左大括号 类型int 变量名a 逗号 变量名b 分号 函数名scanf 左括号 字符串"%d%d" 逗号 取地址符 变量名a 逗号 变量名b 右括号 分号 函数名printf 左括号 字符串"%d" 逗号 变量名a 加号 变量名b 右括号 分号 返回关键字 整数常量0 分号 右大括号</p>
---	---

右侧为对应的语法树，缩进表示父子关系

<pre>int main(){ int a, b; scanf("%d %d", &a, &b); printf("%d", a+b); return 0; }</pre> <p>(原版代码)</p>	<p>树根 函数定义 main() --- 变量声明 int a, b --- 函数调用 scanf --- 参数 "%d%d" --- 参数 &a --- 参数 &b --- 函数调用 printf --- 参数 "%d" --- 参数 加法表达式 a + b --- 加数 a --- 加数 b</p>
---	---

如何实现？

- 编译器的词法分析、语法分析均有成熟的算法实现
- (乃至封装后的工具：yacc, flex)
- 只需基于满足一定条件的语法，构建出自动机.....
- 自学参考：《编译原理：原理、技术与工具》（龙书？）
- （未来程序改？）
- 今天的相关研究领域，更关注的一些事情：
- 如何结合目标机器架构，生成正确、高效的可执行代码(汇编机器码)？
- 自动证明程序的正确性（形式化验证）

Winnowing (MOSS)

- Winnowing算法 (Stanford MOSS) 是使用最广泛的代码查重算法/工具。
- <https://theory.stanford.edu/%7Eaiken/moss/>
- MOSS帮助了大量的程序设计教师进行查重，
- 从而，促使很多学生独立完成作业、获得了更好的学习效果。
- 提升 MOSS工具的维护者 Alex Aiken教授 个人的声誉，
- 提升了Stanford计算机系的声誉。
- 尽管并不收费/盈利，但MOSS无疑是一个成功的产品

MOSS 作为产品为什么成功？

- 也许有一天，你也会设计自己的产品，希望它有MOSS一样的影响力.....
- 恰当的定位, 向用户明确这个工具的局限性(最终需要人判断)
 - Measure Of Software Similarity, not "Plagiarism Detection"
- 持续的维护、迭代。
 - 保持服务的稳定性，根据用户需求增加特性。
- 找到用户的需求/痛点：
 - 公益查重服务，程序设计的课程教师均可免费申请账号
 - 提供易用的程序上传脚本
 - 根据教师的需求，支持各种语言
- 重视安全隐私(十几年来没有发生过数据泄露事件)
- 先发优势（MOSS的诞生和流行足够早，用户缺乏切换到其他工具的动机）
- 当然，也离不开其本身的查重算法足够优秀.....

Winnowing算法(2003年论文发表版本)

- Winnowing算法的基础：
 - k-gram
 - 字符串hash
- 假设：
 - 两份文档，其中一份文档的一些片段和另一份文档的一些片段相同。
 - 大小写、空白符等无意义的特征应当被过滤掉（对于程序：词法分析）
 - 只需要找出“足够长”的相同片段。
 - 较短的相同片段可能只代表固定用法
 - 将文档片段的顺序重排，不应当影响判断的结果（if-else分支颠倒）

k-gram

- 后缀树/后缀数组求最长公共子串？太慢了.....
- 假设我们认为长度为k的相同片段，就构成值得注意的雷同.....
- 将做完前置过滤处理的两份文档，看作长为n和m的字符串
- 分别找出其中所有长度为k的子串，看看是否有相同的子串。
- abcdefghijklmn的5-gram，abcde, bcdef, cdefg, defgh.....
- 完整比对需要比较 $O(nm)$ 对 k-gram。
- 字符串hash能够加速吗？

字符串hash？

hash $H(c_1 \dots c_k)$ of $c_1 \dots c_k$ is this number:

$$c_1 * b^{k-1} + c_2 * b^{k-2} * \dots + c_{k-1} * b + c_k$$

To compute the hash of the k -gram $c_2 \dots c_{k+1}$, we need only subtract out the high-order digit, multiply by b , and add in the new low order digit. Thus we have the identity:

$$H(c_2 \dots c_{k+1}) = (H(c_1 \dots c_k) - c_1 * b^{k-1}) * b + c_{k+1}$$

- $O(n)$ 的时间找到所有k-gram的hash值，排序后线性扫描找相同值
- 如果要进行全部k-gram的比较，还是太慢了。
- 一次提交q份作业，则需要比对 $C(q, 2)$ 对代码（有时还加上往年代码）

选取部分hash值做指纹 (fingerprint)

- 一种加速的想法是，不要选取全部 n 个hash值
- 按照某种规则选择其中的一部分。
- 只比较这一部分hash值，也能保证大概率发现雷同的文档/代码。
- 如果仅仅每隔 L 个位置选择一个hash值？
- 简单颠倒一下顺序，增加一些无意义字符，就能通过这样的查重。

winnowing之前的k-gram指纹选取思路：

- 选取所有模 P 为0的hash值。 P 是一个不大的数。(Manber)
- 这样的选取方式和“位置”无关。
- 但比较看运气。发生雷同的片段可能不包含hash值模 P 为0的k-gram。
- 选取所有k-gram hash值中, 最小的 M 个 (Heintze)
- 缺点：如果两份文档长度相差悬殊....
- (比如，在提交代码的时候，塞进去一堆其实没有被用到的函数，最小的 M 个hash值大概率会出现在这些其实没用到的函数里.....)

Winnowing算法

- 按照一种方式选取k-gram指纹，
- 确保：
 1. 当一对文档包含长度至少为T的雷同片段时，一定能够识别出来
 2. 不识别长度短于K的雷同片段（“噪音”）
 - 比如，查重工具给教师汇报了大量的“int main()”重复片段
- 性质2 容易保证，仅考虑通过长度为K的子串形成的hash值 构建指纹。
- 实践中，选取合适的K、T取值非常重要。在误报和漏报之间取得平衡。

如何识别长度至少为T的雷同片段？

- 一个长度为T的雷同片段，当我们选取长度为K的K-gram时，
- 会产生 $(T-K+1)$ 个连续的hash值。
- 在这 $(T-K+1)$ 个hash值当中，至少有一个需要被选中在指纹当中。
- 基于这样的观察，Winnowing算法相当简单：
 - 对于整个文档的所有hash值中，每个长度为 $(T-K+1)$ 的窗口，选择这 $(T-K+1)$ 个hash值中，最小的hash值加入指纹。
 - 大家熟知的滑动窗口/单调队列算法。
- 实践中，这个算法很高效！

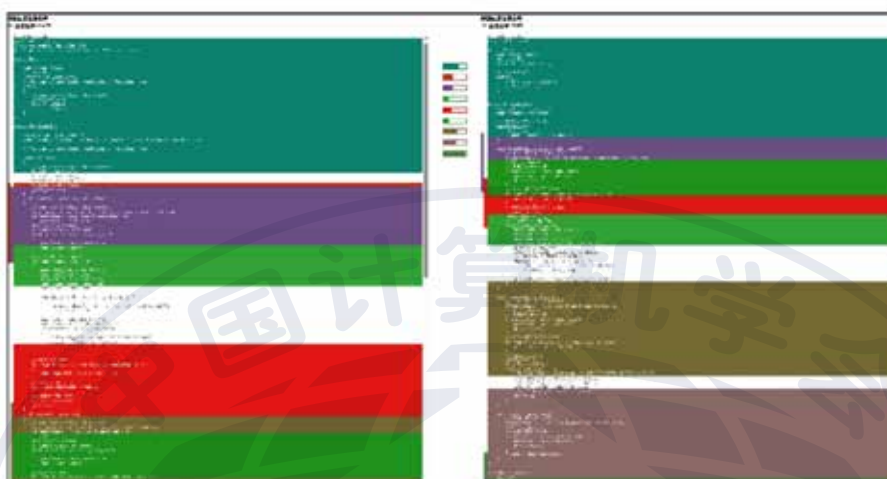
平均情况下，需要选取多少个hash值？

- 最坏情况下，hash值单调递增，几乎所有hash值都需要被选中
- 但实际上几乎不会产生。
- 假设k-gram的hash值大小关系是随机的 窗口长度为 $W = L - K + 1$
- 忽略hash值相同的情况，
- 记随机变量 $X[i]$ 表示， $[i, i+W-1]$ 和 $[i+1, i+W]$ 两个窗口的最小值是否相同
- $\text{sum}(X[i])$ 就是最终指纹选取的数量
- 如果 $[i, i+W]$ 这个区间的最小值落在 $[i+1, i+W-1]$: $X[i] = 0$
- 否则： $X[i] = 1$
- $E[X[i]] = 2/(W+1)$

将winnowing算法集成到MOSS系统中

- 将具体语言的处理 和 指纹选取的模块分离
 - 这样，可以更方便地增加不同语言/文档类型的支持.....
- 排除特定的内容：
 - 固定的实验框架，固定的版权声明....
 - 由教师提供这部分内容，对这部分内容提取指纹，忽略这些匹配
- 可视化地呈现查重的结果
 - 最终仍需要教师判定代码是否雷同
 - 将相邻的雷同片段合并为更长的片段进行展示

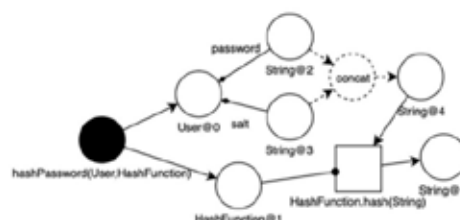
易于阅读的结果（供教师人工阅读）



更多查重算法: 关注数据的流动

- A Novel Graph-Based Program Representation for Java Code Plagiarism Detection
- 利用一个解释器，模拟程序运行时，数据在变量之间的流动，获得一张图
- 问题转化为“最大相同子图”：本身是一个 NP-hard问题

```
public static String hashPassword (
    User user, HashFunction hasher
) {
    String password = user.getPassword();
    String salt = user.getSalt();
    String input = password + salt;
    String hash = hasher.hash(input);
    return hash;
}
```



更多查重算法: 关注数据的流动

- 利用这张图的特殊性，首先找到两个程序的“数据图”之间对应的若干节点
- 例如，特定的函数调用、特定的运算符等，可以选为初始的对应节点
- 给定一对程序的数据图，从初始节点出发，不断迭代加入满足条件的节点：
 - 在两张图里，具有相同的语法属性
 - 在两张图里，都和之前加入的节点有相同类型的边连接
- 最后根据被加入的节点数目评估相似度

更多查重算法: 在语法树上匹配

- Syntax-based Improvements to Plagiarism Detectors and their Evaluations
- 尝试将语法分析的结果加入到查重中，通过语法分析获得语法树
- 对语法树进行后序遍历。在这个过程中，将函数按长度排序。
- 将若干等价类识别为相同的节点，例如if-else的分支和switch的分支。
- 之后再运行字符串的查重算法

```

public class A {
    public void foo() {
        for (int i = 1; i < 10; i++) {
            System.out.println(i);
        }
    }

    public int bar() {
        int sum = 0;
        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                sum += i * j;
            }
        }
        return sum;
    }
}

public class B {
    public int func1(int dummy1, int dummy2) {
        (int extraTest1 = 5; int extraTest2 = 42;
        int x = 0; int i = 1;
        while ((i < 1) {
            int j = 1; while ((j < 2) {
                x += (i - 1) * (2 - j);
                j = j * 7;
            }
            ++i;
        }
        return x;
    }

    public void func2() {
        (int z = 1; while ((10 > 20) {
            System.out.println(x); x++;
        })
    }
}

public class C {
    public void hello() {
        System.out.println("hello, world!");
    }

    public int sum2(int x) {
        int a = 0;
        for (i) {
            if (x == 2) break;
            a += 4;
            x++;
        }
        return x;
    }
}

```

Figure 2: Three contrived example programs: A and B for a plagiarized pair, while C has no plagiarized information taken from A or B.



Figure 3: A bird's eye view of the parse tree structure for Programs A, B, and C.

更多查重算法：跨语言查重

- Plagiarism Detection across Programming Languages
- 如果编程作业允许使用不同编程语言...
- 一个学生用C编程，另一个学生翻译成了java....
- 解决方案：在“中间代码”的层次上查重。
- 编译器的处理流程：词法分析（一串token单词），语法分析（语法树），中间代码生成（和语言/最终机器架构无关的中间码），目标代码生成
- 在中间代码中，过滤掉不重要的关键词（变量名、常数），仅保留和程序结构相关的那些中间代码。

更多查重算法：利用键盘数据

- Plagiarism Deterrence in CS1 Through Keystroke Data
- 算法竞赛常常不考虑查重问题，因为学生完成程序的过程是受控的。
- 如果要求提交完成编程作业过程的录屏，将是对抄袭行为的震慑。
- 录屏太过繁琐/文件太大：重要的只是学生的键盘输入行为。
- 设计一个IDE插件，学生编程的时候启动插件，自动记录键盘行为，最后提交键盘行为记录的文件。
- 教师可以根据记录的文件，回放学生编程的过程。
- 可能比较适合用其他方式识别出雷同案例后的确认。
- （这种技术还可以创作更有交互性的编程文档，vscode扩展storyteller）

更多查重算法：利用时间

- Cheating Detection in Online Assessments via Timeline Analysis
- 对于代码查重，时间线可以作为辅助的证据。对在线小测的效果更好。

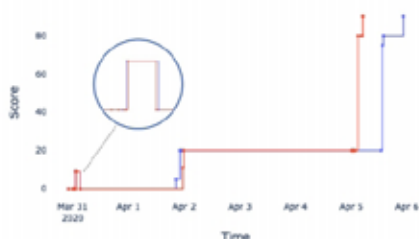


Figure 6: Project Timeline Pattern Full Collaboration (insert – zoomed) Students' work shows high time synchrony



Figure 7: Inconclusive project timeline: although students confessed to cheating, the timeline comparison was not helpful as supporting evidence

促进学术诚信，不只依靠技术：

- Q: 有了“精确 高效的代码查重算法”就能杜绝抄袭现象吗？
- A: 不能。也没有100%精确的代码查重算法。
- 理由 1. 抑制抄袭，除了“代码查重算法”，还需要一套完整的处理制度。
- 理由 2. 错误的行为也有具体的诱因，可以尝试发现和消除深层的诱因。

学术诚信制度

- 今天的学术出版，依赖同行评议制度，基础是互相信任。
- 学术诚信(academic integrity)有很多定义，不妨定义为“不滥用信任”
- 学术不端代价高昂（心肌干细胞：金钱、时间、病人的希望）
- 以足够独立的程度完成课程作业，是学术诚信的第一步
- 未来的雇主、研究生招生学校，信任学生的成绩单反映了学生的能力。
- 维护学术诚信，狭义的作用是确保成绩单的可信度。
- 广义的作用是确保学生的道德品质足以担当重任。

学术诚信制度: Honor Code

- 明确、严格的制度和处理机制，学校/院系层面有统一的处理机构。
 - 如果每个教师独立处理学术不端，容易造成标准的混乱和不统一。
 - 一些学校统一采取的方式：一经发现，挂科起步
- 在学校/院系/课程的层面，提出学生需要遵守的明确要求。
 - 例如：
 - 明确标识出自己曾经获得过的帮助（尽管并不构成抄袭）
 - 不阅读其他人的代码，也不将作业代码提供给其他人/传播
 - 明确什么样的交流形式是可以接受、不构成抄袭的

学生为什么抄袭？

- 尝试发现和消除深层的诱因.....
 - 对分数的追求和对失败的恐惧；
 - 缺乏足够的学术诚信教育，有不被发现的侥幸心理；
 - 学业压力过大，难以独立完成作业/缺乏合适的求助途径；
 - 周围同学的影响；
 - 参考优秀的代码，本身是很好的学习方式（worked example）
- 你认为呢？

促进学术诚信的非技术手段

- Impact of Several Low-Effort Cheating-Reduction Methods in a CS1 Class
- 在课上拿出时间讨论学术诚信，要求学生完成一个学术诚信自测
- 期中再次强调学术诚信，强调被发现学术不端的后果
- 在课上给学生展示代码查重的工具和流程(减少学生侥幸心理)
- 允许学生(在ddl之后、公布查重结果之前)主动撤回提交
- 强化合理求助途径(习题课、答疑等)，避免学生做不出作业而选择抄袭。
- 提供不同难度的作业选项，完成简单难度的作业获得大部分分数。
- 每个学期微调作业的细节，使得往年代码不能直接复用

休息片刻, 接下来还有两个话题...

- 其实都可以从代码查重引申出来!
- 学习行为的数据分析:
 - 收集提交时间数据、收集键盘输入数据可以进行查重
 - 收集其他类型的数据，能否对学生的行为进行更细致的刻画....
 - 从而帮助改进学生的学习？
- 编程教学与大语言模型:
 - 能“自动生成代码”的对话助手，似乎对查重提出了挑战.....

学习行为的数据分析

learning analytics

书接上回

- 根据OJ平台时间数据来进行查重，是“学习行为的数据分析”的一种案例。
- 相关领域的研究者提出了很多近义词，
- 计算教育学，教育数据科学，教育大数据，学习行为分析.....
- 尽管这些名词的提出者有不同的立场和出发点.....
- 但最终的信念都是“万物皆数”。

“万物皆数!” “真的吗?”

- 数据科学的信条：
 - 当我们拥有足够多的高质量数据.....
 - 并用足够好的方式建模分析.....
 - 我们就能更好地理解世界、改变世界。
- 将信条应用于教育领域：
 - 在教学场景中收集足够多的高质量数据，建模分析.....
 - 我们就能更好地理解学生的学习行为、让学生学得更好。
 - 甚至颠覆教育的模式，提供千人千面的个性化教育.....
- 激进的支持者还相信，一套方案会适合于各种教学场景。
- 目前，这些豪言壮语还没能兑现。
- 高质量数据的来源、建模分析的方式..... 诸多前提，如何满足？

数据的来源

- 学习行为分析的兴起，和MOOC的火热有关联（产生了大量的数据！）
- MOOC平台的运营者开始利用用户数据进行分析：
 - 什么样的学生能坚持自学完MOOC、哪个地方学生容易看不下去
- 但在传统的课堂中，并非天然就有这么多数据。
- 加入数字化的学习平台：OJ，网课平台，git仓库，IDE插件.....
- 在传统学习场景中增加数据收集设备：录像、录音、甚至神经生理数据.....
- 收集数据也需要注意数据安全和隐私问题

Q: 数据仅仅包括“数”吗？

- 传统的方式：问卷调查、教师和学生聊天、观察课上的学生反应....
- 广义来说，和学生对话交流，也是一种数据收集过程！
- 传统意义上的教育实证研究：
 - 定量研究，质性研究/定性研究。
 - 对应：量化数据（数字），质性数据（文本、视频、图像）
 - 有人认为基于大量的数据开展教育研究，是一种全新的研究范式。
- 计算机视觉、自然语言处理等领域，早已经将文本、图像作为数据的一种。

案例：用数据对学生分类，进行分层教学

- Using Learning Analytics to Investigate Patterns of Performance and Engagement in Large Classes
- 大学低年级的编程基础课程，学生背景多样，基础多样。
- 将学生分层，给予适合他们的帮助（合适的建议和材料），就能学得更好。
- 可行思路：在期中时，根据前半学期的表现，后半学期采取针对措施。
- 直接根据期中测验的分数来分层：并不完美。

分层教学的更多数据来源

- 在慕课热潮之下，这门课程进行了“翻转课堂”的尝试。
- 学生课前根据材料进行自学。课上进行小测和小组讨论。
- 好处是，获得了很多学生学习的过程数据：
 - 选择题小测分数、编程作业分数、各类课件和参考答案的浏览次数.....
- 对“翻转课堂”做法的一种理解：
 - 用一些新花样，让学生和教师在这门课上投入更多时间精力。
 - 如果有条件，师生乐意在这门课投入更多的精力，是有效果的。
 - 当学生和教师的时间已经高度紧张时，翻转课堂往往难以成功。

三类数据

- 结果性评价 summative assessment
 - 期中笔试 + 多次限时的编程作业
- 形成性评价 formative assessment
 - 课前小测、课上小测
- 行为数据 behaviorial data
 - 课件浏览次数、可选题目的浏览次数、参考答案浏览次数
- 对每个数据维度都进行归一化（归一化之后的数据均值为0，标准差为1）
 - 将原始数据减去均值，除以标准差
- 随后，对归一化之后得到的向量，执行K-means聚类算法

聚类问题

- 聚类（clustering）：一种无监督的机器学习算法
- “聚类”这个翻译很可能是来自“物以类聚，人以群分”
- 说白了就是在不知道有具体哪些类别的情况下做分类。
- 比如：
 - 小明的手机里有1000张照片，想把照片整理成一些有主题的相册。
 - 天文观测，星星 -> 星系
 - 社区发现：在社交软件中，找到哪些人具有相同的兴趣
 - 把用户划分成若干个小群体
 - 随后以这些隐藏的“社区”为依据，给他们推送可能感兴趣的内容

K - means 聚类算法

- 在N维空间中，有若干个点，每个点拥有N个属性（坐标）。
- 两个点A, B之间的距离：
 - for $i = 1 \dots n$,
 - $sum += (a[i] - b[i]) * (a[i] - b[i])$
- return \sqrt{sum}
- 现在希望按照某种规则，将这些点归为K类，要求同一类的点彼此相近，不同类的点差别较大。

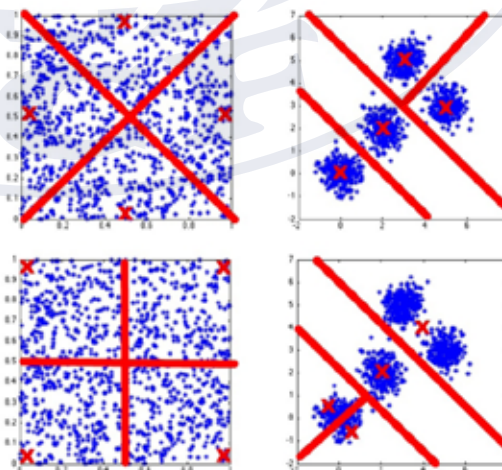
一维聚类

- 给定n个数，要求你将这些数分为K类，同一类的数彼此接近....
- 结果有点平凡，一定是数轴上的K个区间。
- 我们用某个区间的“中心点”代表这个区间....
- 给定K个中心点，每个数和距离最近的中心点分到一起。
- Q: 如何选定这K个中心点？排序？随机？
- 中心点的选取会影响最终的结果....

二维聚类

- 平面上有很多点，对它们进行聚类，要求结果为K类。
- 0. 选中K个初始的“中心点”
- 1. 对每个待聚类的点，找到距离它最近的中心点，归为一组。
- 2. 对每一组，计算所有点的平均位置
- 3. 获得K个新的“中心点”，回到第1步重复进行，直到分类结果不变。
- 可以证明，必然收敛到一个局部最优解（但未必是全局最优解）
 - “中心点与分配给它的所有点的距离平方和”是单调递减的。

初始点很重要



K-means 难以处理的情况？

- 尝试现场画一些出来.....

对更多类似算法感兴趣？

- 更多聚类算法：层次聚类
- 更多的“经典”机器学习算法：决策树、支持向量机、K近邻、贝叶斯学习...
- (“经典”：不同于今天火热的、人工神经网络主导的人工智能模型)
- 寻找标题为“机器学习”/machine learning 的经典教材和网课

分析结果和干预措施

Id	Features	Conjecture	Appraisals
C ₁	H, H, M	Strongly engaged and achieving students infers strong interest in the course content with strong technical-, soft-, and academic skills.	Recognize their accomplishments. Provide additional, optional research or peer-mentoring opportunities to sustain engagement.
C ₂	H, M, L	Moderate levels of engagement and relatively high achievements infers possibility of previous experience in coding, allowing them to perform well without high engagement. Significant drop in performance on S_1 compared to S_2 potentially because of lack of previous knowledge of content covered later in the term.	Recognize their accomplishments with an alert on drop in performance. Provide additional, advanced challenges early in the semester to improve engagement.
C ₃	M, H, H	Strongly engaged and moderate achievements infers lack of soft-, academic skills, and/or issues with the technical content. Significant improvement on S_3 compared to S_2 shows students' desire to do well.	Recognize their hard work and efforts to ("catch-up"). Provide discussions/training on how they can study more effectively. Recommend peer-mentors to them.
C ₄	L, M, M	Moderate levels of engagement and low achievements infers lack of soft-skills and issues in the technical contents as they are not benefiting from the in-class, on-line material. Possibly also a lack of interest in deeply learning the topic.	Recognize their effort to engage in-class and on-line, noting their low performance. Provide discussion/training on soft- skills; additional fundamental content and challenges to work on technical content.
C ₅	VL, VL, L	Disengaged and low achievements infers lack of soft-, academic skills, non-academic issues, or problems with the technical content. Students may be at risk.	Reach out to early-alert or student consultation services or invite students to meet the course staff in a 1 on 1 session.

Table 2: Feature abstraction, conjectures, and appraisals for each of the clusters in the entire class population.

从这个案例中我们能学到什么？

- 好的“学习行为分析”，是一门优秀课程的结果(锦上添花)
 - 很多平庸的课程本身，没有提供丰富的编程作业和各类学习材料。
- 教师决心投入精力给不同类型的学生设计对应的干预措施。
 - 因此才有给学生按照行为数据分类的必要。
 - 需求 -> 解决方案，而非“拿着锤子看什么都像钉子”
- 如果教师自己没有用心准备课程材料，也没有采取干预措施的动机...
 - 此时，多么精妙的数据收集和分析，也不会让教学质量有所改善。

了解学情、改革教学：数据不是一切

- 尽管我们看到很多“学习行为分析”的成功案例
- 但“publication bias”（出版偏差）也客观存在
- 在教育中用好数据，需要：好的数据 + 好的算法 + 优秀的教育者。
- No silver bullet.
- 并非一切都能量化
- 一个结论能用量化的形式得出来 \neq 这个结论是对的
- “统计数字会撒谎”

了解学情、改革教学：数据不是一切

- 《数据结构》课程的助教经历：
 - 询问学生为什么期中退课：编程作业的调试困难
 - 尝试提供更多的线下答疑，帮助调不出程序的同学
 - 尝试给学生提供调试方法和工具的相关文档
 - 尝试给学生的实验报告提供更详细的反馈
 - 尝试问卷调查“最喜欢的/最讨厌的题目”
 - 优化编程作业
 - 尝试分析“何时动手完成作业和成绩的关系”
 - 将作业切成更多份

了解学情、改革教学：数据不是一切

- 很多教学改革并不需要把数据摆在面前才知道该怎么做。
- 如果平时和学生多交流，自然能了解学习的痛点。
- 难的是，如何激励教师投入对应的时间、精力？
 - 面对一整个教师群体/助教群体，不能只靠兴趣/热爱。
- 或者，给教师提供相应的外部支持，帮助教师了解学生。

书接上回

- “学习数据分析”的很多算法，都属于“判别式人工智能”
 - 从数据中挖掘特征、分类……“化繁为简”
- 与之相比，近年风头正盛的AI绘画、AI对话助手、AI编程，
- 属于“生成式人工智能”，
- 基于大量数据训练模型，生成丰富的结果，“化繁为繁”
- 此类“生成式人工智能”，对计算机教育（更具体地，编程教学）有什么影响？
- 休息一下，稍后继续

编程教学与大语言模型

LLM and programming instruction

大语言模型：未来往何处去？

- 这一波生成式人工智能/大语言模型的热潮，未来往何处去，还不清晰。
- 有人觉得，通向AGI（通用人工智能）的阶梯已经出现在了眼前。
- 有人认为“就这”？无非是一张“模糊的JPEG图像”，和智能无关。
- 谁说的对，咱也不知道。
- 由于涉及的预备知识较多，我们不讨论更加细节/前沿的大语言模型技术，
- 主要还是通过编程教学的案例，尝试粗浅地探讨：
 - 究竟什么是现在的人工智能还不能替代的？

搜索练习：ChatGPT用到哪些计算机技术？

- 一些网页并非人类创作.....
 - 而是AI生成的产物.....
- 生成似是而非的内容，前所未有的容易。
- 网络空间/搜索引擎的信噪比，似乎在恶化。
- 下面是一些例子

哪里不太对？

ChatGPT 背后包含了哪些技术？

发布于 2023-05-01 16:28:03 · 452 · 0

△ 编辑

ChatGPT 是由OpenAI开发的一款基于GPT-3 (Generative Pre-trained Transformer 3) 的人工智能语言模型。这个模型是使用多种编程语言和技术组合编写的。首先，ChatGPT 使用了 Python 作为主要的编程语言。Python 是一种流行的高级编程语言，特别适合于数据科学、机器学习和自然语言处理等领域。OpenAI 的研究团队使用 Python 来编写 ChatGPT 的核心算法和模型架构。Python 在数据科学和机器学习方面的优势主要体现在其丰富的库和工具集上。这些工具可以加快开发速度，提高代码质量和效率。

其次，ChatGPT 使用了 TensorFlow 作为深度学习框架。TensorFlow 是由 Google 公司开发的一款流行的深度学习框架。它可以帮助研究人员快速开发和训练深度神经网络。OpenAI 使用 TensorFlow 来搭建 ChatGPT 的神经网络模型，并使用 TensorFlow 的分布式训练功能来加速模型的训练过程。TensorFlow 的优势在于它的灵活性和可扩展性，可以适应不同规模和复杂度的模型。

除此之外，ChatGPT 还使用了其他一些编程语言和技术，包括：

- CUDA：用于在 NVIDIA GPU 上进行并行计算，以加速深度学习模型的训练和推理。
- C++：用于优化模型的计算性能和内存管理，特别是在模型部署和推理时。
- Rust：用于编写高性能的系统级代码，以提高模型的稳定性和安全性。
- JavaScript：用于开发 ChatGPT 的 Web API，使其可以通过网络接口与其他应用程序集成。

哪里不太对？

国内版 国际版

Q does chatgpt use tensorflow?



ALL IMAGES VIDEOS TOOLS

About 1,410,000 results

Yes, ChatGPT is using **TensorFlow and PyTorch**^{1 2}. These frameworks provide essential tools and libraries for building and training deep learning models¹. PyTorch and TensorFlow are the most commonly used tools for creating language models². ChatGPT stops generating the output when the transformer layers output a special token called a stop token².

Learn more: [1. msn.com](#) [2. dataquest.io](#)

从用户角度，大语言模型能做什么？

- 下面是一个“幻觉”的例子，回答的语法通顺，但它是错的。



尝试从教学角度描述ChatGPT的能力：

- 基于大量的语料库训练，具有大量的参数。
- 对几乎所有问题，都可以给出语法上通顺、语义上连贯的回答。
- 能生成一些较短的代码片段，解决简单的编程问题。
- 由于随机的排列组合，有时你会感觉它的回答具有新意。
- 对于大量领域的基本问题，由于语料库中已有的知识，它的回答常常是对的。
- 此时，它表现得像一个更加智能、更懂人性的搜索引擎。
- 但对于新的问题，**不能保证回答的正确性。**“幻觉”问题。

尽管有局限，但还是带来了更多的可能.....

- 现在的大语言模型还不是我们期望的AGI，能力相当有限
- 但已经足以在编程初学阶段，完成很多任务.....
 - 生成一份助教并不会仔细阅读的实验报告
 - 生成大一程序设计课程的作业代码，或简单算法题的代码。
 - 帮助初学者调试简单的程序
 - 帮助初学者理解IDE的报错信息
- 也足以带来很多风险....
 - 初学者更容易被大语言模型给出的似是而非的信息误导
 - 初学者依赖大语言模型完成简单的作业，影响其形成自身的编程能力
 - 独立完成简单任务，才有能力独立完成模型不能做的复杂任务？
 - 使用人工智能生成的代码带来的查重问题.....



如何出ChatGPT不会做的作业题？

- Sam Lau and Philip Guo. 2023. From "Ban It Till We Understand It" to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot. In Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1 (ICER '23),
- 沙行勉, 王寒, 徐珑珊, 宋玉红, 诸葛晴凤. ChatGPT对计算机基础教育的挑战分析与应对策略[J]. 计算机教育, 2023, (11): 51-54.

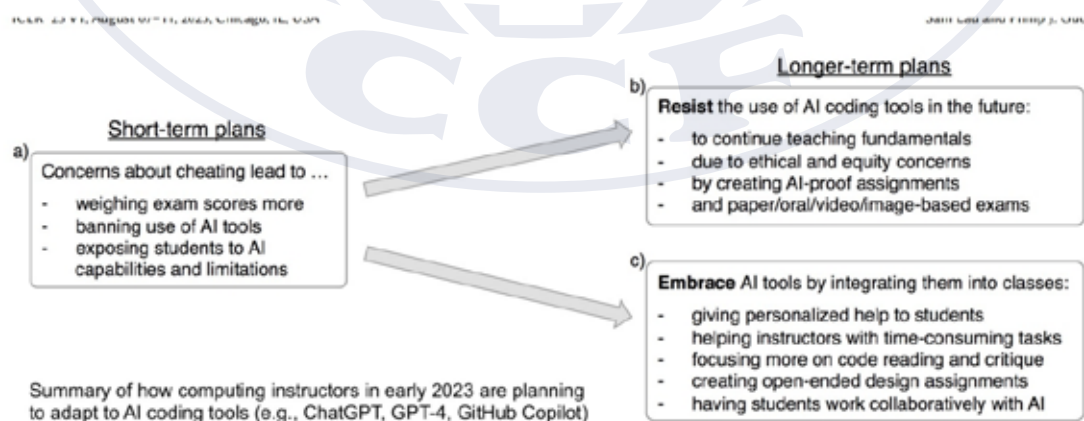


Figure 1: Summary of our study findings. We interviewed 20 introductory programming instructors and present both a) their short-term plans, and their longer-term plans to either b) resist or c) embrace the use of AI coding tools in their classes.



图2 面向Chat GPT的计算机基础教学中的应对策略设计

大语言模型能代替助教答疑吗？

- 在大学课程学习遇到困难时，通常鼓励大家向助教寻求帮助。
- 助教答疑制度的完善程度、助教答疑的质量，是课程质量的重要部分。
- 那么，助教也是和学生对话，LLM也和学生对话，能代替助教做答疑吗？
- 从Stack Overflow说起.....
 - 最著名的计算机领域在线问答网站
- 很多人都知道Stack Overflow上能找到很多高质量的回答
- 但并不是每个人都知道Stack Overflow的回答质量是如何维系的。
- 除了大众用户点赞/点踩，算法分析用户和答案的特征做推荐.....
 - 大批的社区志愿者，无私地承担了审核答案质量的工作。

Stack Overflow背后的英雄: 社区志愿者

- Stack Overflow这样的社区依靠人们互助的善意运作
- 大家最容易看到的是编写回答的答主
- 不容易看到的：
 - 修订者 (curator)：进行公共编辑，帮助改进其他人答案的质量
 - 审查者 (moderator)：处理被举报 (flagged posts) 低质量的答案
- 2023年，在不引人注意的角落，社区志愿者们进行了罢工.....
- 从2023年6月5日持续到2023年8月7日
- <https://meta.stackexchange.com/questions/389811/moderation-strike-stack-overflow-inc-cannot-consistently-ignore-mistreat-an>

背景

- 在ChatGPT火爆初期（2022年底），stack overflow社区审查者的共识：
 - 在全站禁止使用AI生成内容
 - 任何普通用户都可以将自己感到怀疑的内容进行标记 (flag)
 - 社区审查者将会仔细阅读 flagged post, 如果很有AI的味道，就会删掉
 - 这个做法在2023年上半年运行良好
- 但在2023年5月，Stack Exchange公司尝试发布政策，阻止志愿者这样做
 - 这个政策没有和社区志愿者充分沟通就发布了
 - 明面上的理由：GPT Detector之类的自动化程序不准确
 - 但实际上，社区志愿者从未依赖类似的自动化程序来判定
 - Stack Exchange公司的角色：商业化运营，平台服务器维护等。

AI生成内容有什么问题？

- Stack Overflow审查者 Machavity：AI聊天机器人是只鹦鹉。
- 不理解自己所说的话，不能自己检验正确性。
- 那些复制AI聊天机器人回答到Stack Overflow的人，也是鹦鹉。
- 虚假的令人信服语气、虚假的参考引用。
- 严重危害StackOverflow的愿景：集合高质量的问题和答案。
- 诚信问题：提交AI生成内容作为回答，而不注明这是AI生成，是一种抄袭。
- （如果AI生成内容是令人满意的回答，问问题的人为啥自己不问AI呢？）
- Stack Exchange公司尝试推行的新规，
- 严重限制了社区审查者对抗这些问题的能力。

社区审查者的个人案例

- Stevoisiak (2023.5)：过去六个月，处理超过10000条疑似AI生成内容
- 平均每天花费了40分钟处理相关问题
- 从未使用自动化检测工具
- 但能够以非常高的准确率识别出AI生成的答案（即使做过润色）
- 检测AI生成内容的困难，在于低成本程序自动检测。
- 领域专家具备检测的能力，但大多数领域支付不了这个成本。
- 比如，用AI写课程论文，在认真批阅的前提下，很容易被发现。

大语言模型不改变什么？

- “你为什么不Google/百度”？可能变成“为什么不问ChatGPT/文心一言”
- 但不会改变：曾经不懂用搜索引擎的初学者，现在也不会用大语言模型。
- 大一程序设计课程的教师，依然需要对初学者进行引导
 - 引导其积极、合理使用当前可用的信息检索工具。
 - 看到学生问的问题背后，他们真正遇到的麻烦。
 - 在不知道答案的时候，告诉学生自己不知道，搞清楚之后再回答。
- 不会改变：初学者从前会遇到搜不到的问题
- 现在也会遇到大语言模型回答不了、甚至答错的问题。
- 教师/助教依然需要积极和学生交流，提供兜底答疑渠道。

大语言模型不改变什么？

- 如果助教不用心批改实验报告，那么不管学生自己写实验报告还是用工具生成实验报告，学生从中的收获都有限。
- 即使在大语言模型诞生之前，世界上也已经有足够多的似是而非的信息，可能误导学生的学习。大语言模型只不过让这个问题更明显了。
- 在课程里用好任何一项新的教育技术，哪怕是二十年前的幻灯片，都需要教师结合课程特点、学生特点，投入时间精力，做针对的设计。

教育技术，天使还是魔鬼？

- 乐观主义者：XXX 将会颠覆教育。XXX可以是：
 - 印刷术（中世纪），电影(1923年，爱迪生)，计算机（90年代末的很多人）
 - 网课/慕课(2011，Sal Khan, 可汗学院)
 - 元宇宙（2021年的很多人），人工智能（2023年的很多人）
- 悲观主义者：
 - 柏拉图：书写给人带来惰性，外界的符号抑制人自身的智慧.....
 - Oversold and underused: Computers in classrooms Larry Cuban
 - 制造白痴, 电子产品如何威胁下一代 米歇尔·德米尔热
 - Failure to disrupt, Technology Alone Can't Transform Education Justin Reich

结语：

教育中的技术与算法，
偶尔颠覆，常常新奇，总在演进

树的基础算法

南京外国语学校 史钊镭

 中國計算機學會
CHINA COMPUTER FEDERATION 

01

树的定义、表示

树的定义

【定义】连通且无环的无向图。

【等价条件】

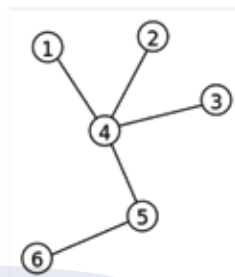
1. 连通，且含有 n 个点、 $n-1$ 条边
2. 任意两点间恰有一条路径
3. 无向无环的连通图





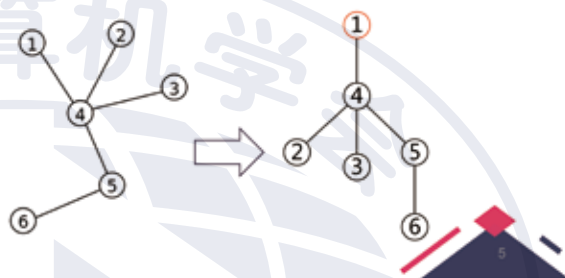
无根树

- 满足上述定义、没有其他限制
- 每个节点的地位是相同的
- 当做普通的无向图来处理



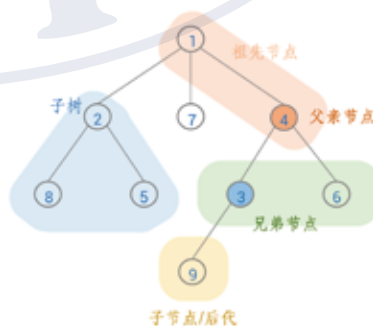
有根树

- 在定义的基础上、指定一个节点为“根”
- 相比无根树，有根树更多地利用树的性质，组织结构更加清晰
- 树上的问题一般先转化成有根树再解决



树结构的特点

- 树是具有递归结构的。
- 一个节点即是一棵树，若根节点为 a 节点，其任意一个孩子节点 b ，与所有不用通过 a 节点即可和 b 相连的节点也构成一棵树，称为以 b 为根的 a 的一棵子树。 b 的任意一棵子树也是 a 的一棵子树。
- 如果一棵树是有根的，那么某个节点 a 的深度 $d(a) = d(\text{father}(a)) + 1$ ，其中 $\text{father}(a)$ 为 a 的父亲节点，另 $d(\text{root})=1$ 。



有根树的存储

[方法一]

除了根没有父亲，所有节点都有唯一的父亲。
记录每个节点的父节点即可。

缺点：不能从根开始遍历整棵树

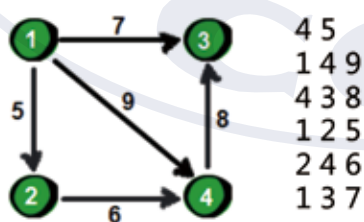
应用：并查集

[方法二]

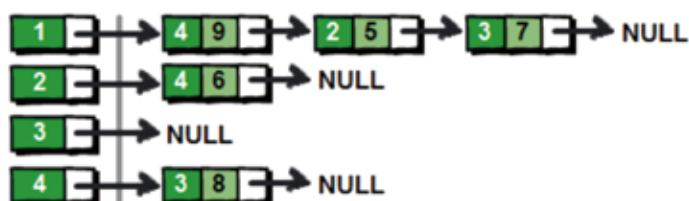
用邻接表存储(同普通的无向图)。

应用：(很多OI题中)只知道树边的两个端点、不知道父子关系的情况，任选一个点为根。

树的表示和存储—邻接表



	U	V	W
1	1	4	9
2	4	3	8
3	1	2	5
4	2	4	6
5	1	3	7



树的表示和存储—邻接表

```
1 vector<int>g[100010];
2 int main(){
3     scanf("%d",&n);
4     for(int i=1;i<n;i++){
5         int x,y;
6         scanf("%d%d",&x,&y);
7         g[x].push_back(y);
8         g[y].push_back(x);
9     }
10
11     for(int i=0;i<n;i++) //枚举点
12         for(int j=0;j<g[i].size();j++)
13             { //枚举顶点i的链表
14                 ....
15             }
16 }
17
18 ..
```

```
1 struct Edge{
2     int v,len,nxt;
3 } edge[maxn<<1]; //记录边
4 int head[maxn],k=0; //每个顶点都有一条链表
5 void addedge(int u,int v,int l){
6     edge[++k].v=v; //右端点
7     edge[k].len=l; //边长
8     edge[k].nxt=head[u]; //下一条边
9     head[u]=k; //头插到 u 节点的链表
10 }
11 int main(){
12     scanf("%d",&n);
13     for(int i=1;i<n;i++){
14         scanf("%d %d %d",&u,&v,&len);
15         addedge(u,v,len);
16         addedge(v,u,len);
17     }
18     for(int u=1;u<n;u++) //枚举顶点
19         for(int i=head[u]; i;i=t[i].nxt){
20             //遍历顶点u的边链表
21             int v=t[i].x;
22             ...
23         }
24 }
```



02

Part Two

树的遍历

深度优先(DFS)

最常用的遍历方式，遍历一个结点后
接下来遍历这个结点的第一个没有被
遍历过的子节点，如果没有这样的点
就退出，时间复杂度 $O(n)$ 。
(先序、中序、后序)

广度优先(BFS)

树上的广度优先遍历实质上是对树上的
所有结点按照到根结点的距离进行分层
进入队列依次处理，时间复杂度 $O(n)$ 。

```
void dfs(u){
```

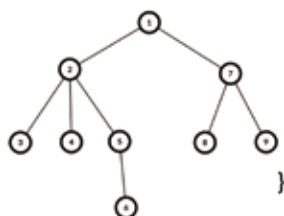
①
对于 u 的每个孩子 v {

②
dfs(v);

③
}
//过去 现在 将来



9
1 2
2 3
2 4
2 5
5 6
1 7
7 8
7 9



```
#include <bits/stdc++.h>
using namespace std;
int const MaxN = 1005;
vector <int> g[MaxN];
int n;
```

```
int main(){
    cin >> n;
    for (int i=1;i<n;i++){
        int x,y;
        cin >> x >> y;
        g[x].push_back(y);
    }
    dfs(1);
    bfs(1);
    return 0;
}
```

```
void dfs(int u){
    cout << u << " ";
    for (int i=0;i<g[u].size();i++)
        dfs(g[u][i]);
}
```

```
void bfs(int root){
    queue <int> q;
    q.push(root);
    while (!q.empty()) {
        int u = q.front();
        cout << u << " ";
        q.pop();
        for (int i=0;i<g[u].size();i++)
            q.push(g[u][i]);
    }
}
```



[例01] 联合权值(NOIP2014)

无向连通图 G 有 n 个点， $n-1$ 条边。点从1到 n 依次编号，编号为 i 的点的权值为 W_i ，每条边的长度均为1。图上两点 (u, v) 的距离定义为 u 点到 v 点的最短距离。对于图 G 上的点对 (u, v) ，若它们的距离为2，则它们之间会产生 $W_u \times W_v$ 的联合权值。

请问图 G 上所有可产生联合权值的有序点对中，联合权值最大的是多少？所有联合权值之和是多少？

[输入格式]

第一行包含1个整数 n 。

接下来 $n-1$ 行，每行包含2个用空格隔开的正整数 u, v ，表示编号为 u 和编号为 v 的点之间有边相连。

最后1行，包含 n 个正整数，每两个正整数之间用一个空格隔开，其中第 i 个整数表示图 G 上编号为 i 的点的权值为 W_i 。

[输出格式]

输出共1行，包含2个整数，之间用一个空格隔开，依次为图 G 上联合权值的最大值和所有联合权值之和。

由于所有联合权值之和可能很大，输出它时要对10007取余。

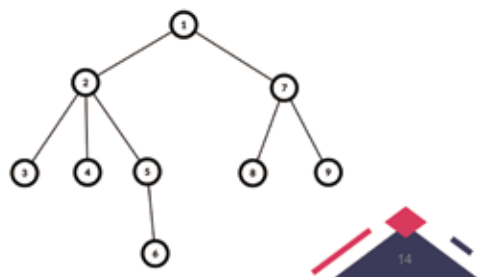
对于100%的数据， $1 < n \leq 200,000$ ， $0 < w_i \leq 10,000$ 。



距离为2的有序点对有(1,3)、(2,4)、(3,1)、(3,5)、(4,2)、(5,3)。

其联合权值分别为2、15、2、20、15、20。其中最大的是20，总和为74。

输入样例	输出样例
5 1 2 2 3 3 4 4 5 1 5 2 3 10	20 74



思路

容易发现，这个图其实是一棵树，树上距离为2的节点只有两种情况：

1. 两个节点为“祖父-孙子”
2. 两个节点互为兄弟
3. 联合的两个节点距离为2，可以枚举中间点。
4. 假设每个中间点周围有两个点，权值分别为a、b，则联合权值为 $2ab$ ，我们可以用 $(a+b)^2 - (a^2 + b^2)$ 来替换。
5. 若有三个点，权值分别为a、b、c，则联合权值为 $2ab + 2bc + 2ac = (a+b+c)^2 - (a^2 + b^2 + c^2)$ 。
6. 依此类推，以某个节点为中转点的联合权值之和等于权值之和的平方减去权值的平方和。
7. 为了找到最大的联合权值，贪心一下，只需找到周围最大的两个权值最大值和次大值，相乘即可。

注意：题目对权值和%10007，但最大联合权值不%10007



[例02] 树上平均距离[hdu2376]

给你一棵 n 个顶点的带权树，计算任意两点的平均距离。

[输入格式]

第一行一个整数 n。

接下来n行，每行三个数字，表示 a 起点、b 终点和 c 边权。

[输出格式]

输出顶点的平均距离，保留小数点后三位。

[数据范围]

$2 \leq n \leq 20000$ ， $0 \leq a < n$ ， $0 \leq b < n$ ， $1 \leq c \leq 1000$



```

graph LR
    1((1)) ---|6| 0((0))
    2((2)) ---|3| 0
    0 ---|7| 3((3))
    3 ---|2| 4((4))
    4 ---|1| f((f))

```





树形动规

树型动态规划

树型动态规划就是在“树形”数据结构上进行动态规划，一般的动态规划都是线性的或者是建立在图上的，线性的动态规划有二种方向既向前和向后，相应的线性的动态规划有二种方法（顺推、逆推），而树型动态规划是建立在树上的，所以也相应的有二个方向：

1. 叶→根：即根的子节点传递有用的信息给根，之后由根得出最优解的过程。
2. 根→叶：通常我们可以先任选某顶点dfs遍历一遍树，再用二次扫描换根的方式将信息传递到所有节点，过程中利用每条边来父子交换转移信息。



特点

- 1) 一般按照后序遍历的顺序，即处理完儿子再处理当前节点，这符合树的子结构的性质。
- 2) 实现方式：树形DP是通过记忆化搜索实现的，因此采用的是递归方式。
- 3) 时间复杂度：树形DP复杂度基本上是 $O(n)$ ；若有附加维 m ，则是 $O(nm)$ 。



动态规划算法必须满足的条件

- 1. 阶段性。**整个问题的求解可以划分为若干个阶段的一系列决策过程（与树的层次性相对应）
- 2. 最优子结构性质。**一个最优化策略具有这样的性质，不论过去状态和决策如何，对前面的决策所形成的状态而言，余下的诸决策必须构成最优策略。简而言之，一个最优化策略的子策略总是最优的（与树的递归结构相对应）。
- 3. 无后效性。**将各阶段按照一定的次序排列好之后，对于某个给定的阶段状态，它以前各阶段的状态无法直接影响它未来的决策，换句话说，每个状态都是过去历史的一个完整总结（与dfs的时间戳特点相对应）。



经典问题：树的独立集、支配集

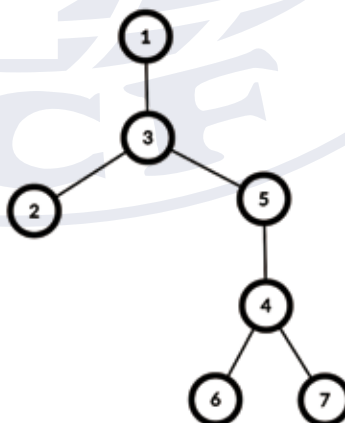
- 对于一棵有 N 个结点的无根树，选出尽量多的结点，使得任何两个结点均不相邻（称为最大独立集）。
- 对于最大独立集，每个点也只有两种状态，即属于点 i 属于独立集或者不属于独立集两种情况，通常我们可以设状态：
 - 1): $dp[i][0]$ 表示点 i 属于独立集的情况下，最大独立集中点的个数。
 - 2): $dp[i][1]$ 表示点 i 不属于独立集的情况下，最大独立集中点的个数。



[例03] 没有上司的舞会

- Ural 州立大学的校长正在筹备学校的 80 周年纪念聚会。由于学校的职员有不同的职务级别，可以构成一棵以校长为根的人事关系树。每个资源都有一个唯一的整数编号，从 1 到 N 编号，且对应一个参加聚会所获得的欢乐度。
- 为使每个职员都感到快乐，校长设法使每个职员和其直接上司不会同时参加聚会。
- 你的任务是设计一份参加聚会者的名单，使总欢乐度最高。

输入样例	输出样例
7 1 1 1 1 1 1 1 1 3 2 3 6 4 7 4 4 5 3 5 0 0	5
输入格式	输出格式
第一行是一个整数 N； 接下来 N 行对应 N 个职员的欢乐度，第 i 行 的一个整数为第 i 个职员的欢 乐度 p _i ； 接着是学校的人事关系树，每 一行格式为 L K，表示第 K 个职员是第 L 个职员的直接 上司，输入以 0 0 结束。	输出参加聚会者获得的 最大欢乐度。 1 ≤ N ≤ 6000, -128 ≤ p _i ≤ 127



- 每个节点给定一个值，子节点和父亲节点不能同时选取，求能够选取的最大价值。
- $f[u][0]$ 表示不选取 u 节点后 u 子树能够得到的最大价值；
- $f[u][1]$ 表示选取了 u 节点后 u 子树能够得到的最大价值。
- 在回溯时可以通过子节点向上更新信息。
- 设 i 号节点的价值为 $a[i]$ ， (u, v) 为一条有向边。

$$f[u][0] = \sum \max\{f[v][0], f[v][1]\}$$

$$f[u][1] = a[u] + \sum f[v][0]$$



[例04]电话网络[USACO2008JAN]

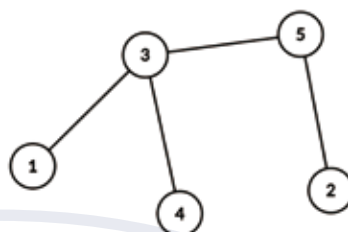
FJ 必须在奶牛们居住的 N ($1 \leq N \leq 10,000$) 块草地中选一些建上无线电通讯塔，来保证任意两块草地间都存在手机信号。

草地中只有 $N-1$ 对是相邻的，不过对任意两块草地，都可以找到一个以 A 开头以 B 结尾的草地序列，并且序列中相邻的编号所代表的草地相邻。无线电通讯塔只能建在草地上，一座塔的服务范围为它所在的那块草地，以及与该块草地相邻的所有草地。

请你帮 FJ 计算一下，为了建立能覆盖到所有草地的通信系统，他最少要建多少座无线电通讯塔。



输入样例	输出样例
5 1 3 5 2 4 3 3 5	2
输入格式	输出格式
第一行是一个整数 N ； 每行为 2 个用空格隔开的整数 A, B ，为两块相邻草地的编号。	输出 1 个整数，即 FJ 最少建立无线电通讯塔的数目。



FJ 可以选择在草地 2 和草地 3，或是草地 3 和草地 5 上建通讯塔。



[例05]士兵的放置[BZOJ2314]

八中有 N 个房间和 $N-1$ 双向通道，任意两个房间均可到达。

现在出了一件极 BT 的事，就是八中开始闹鬼了。

老大决定加强安保，现在如果在某个房间中放一个士兵，则这个房间以及所有与这个房间相连的房间都会被控制。

现在，老大想知道至少要多少士兵可以控制所有房间。以及有多少种不同的方案数。

[输入格式]

第一行一个数字 N ，代表有 N 个房间，房间编号从 1 开始到 N ($N \leq 500000$)。下面将有 $N-1$ 行，每行两个数，代表这两个房间相连。

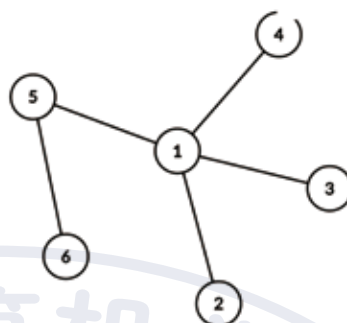
[输出格式]

第一行输出至少有多少个士兵才可以控制所有房间。

第二行输出有多少种方案数，方案数会比较大，输出除以 1032992941 的余数。



输入样例	输出样例
6	2
1 2	2
1 3	
1 5	
1 4	
5 6	



第一种方案是将士兵放在1号房间及6号房间。

第二种方案是将士兵放在1号房间及5号房间。

经典问题二：树上背包

树形背包解决的问题是给你几个物品，物品有依赖关系，a依赖b，b依赖c，选a就必须选b，选b就必须选c，一个物品只能依赖一个物品，但一个物品可以被多个物品依赖，其中依赖关系可以构建一棵树，同时要求你选择 n 物品可以使得价值最大。

[例06] 选课

大学里实行学分制，每门课程都有一定的学分，其中有些课程可以直接选修，有些课程需要一定的基础知识，必须在选了其他的一些课程的基础上才能选修。一门课的直接先修课最多只有一门。两门课也可能存在相同的先修课。每个学生必须在入学时选定自己要学的课程。每个学生可选课程的总数是给定的。现在请你找出一种选课方案，使得你能得到学分最多，并且必须满足先修课优先的原则。

课号	先修课号	学分
1	无	1
2	1	1
3	2	3
4	无	3
5	2	4

课号 1 是课号 2 的先修课，即如果要先修课号 2，则课号 1 必定已被选过。同样，如果要选修课号 3，那么课号 1 和课号 2 都一定被选修过。



这道题有多个课程，我们需要在树根设定一个虚拟课程，而且所有科目的先修课程，使得一共有 $n+1$ 个节点。使得 n 个点构成的森林结构能够形成一棵有根树，有利于树形DP。

设 $f[u][j]$ 为以第 u 个节点为根节点课程（即 u 是所有 u 的子节点的先修课程），在这棵子树总共修完 j 个课程所能获得的最多的学分，显然有 $f[u][0]=0$ 。

u 节点同时为多个节点的父亲， v 是 u 的儿子之一，如果给 v 所在的子树分配 i 门课， u 和剩余儿子剩下了 $j-i$ 个课程。

状态转移：

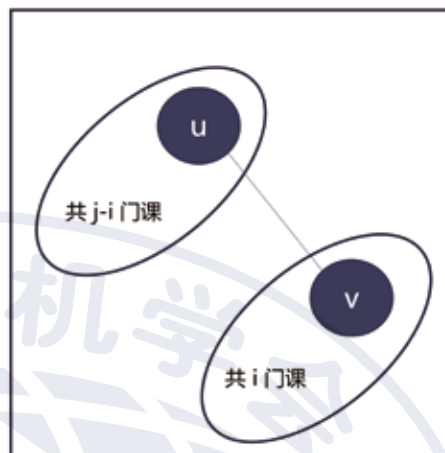
$$f[u][j] = \max\{f[u][j], f[u][j-i] + f[v][i]\}, v \in \text{son}(u), 0 \leq i \leq j-1$$

目标状态：

$f[0][m+1]$ ，0 节点是虚拟节点，需要额外分配1门课。



```
1 #include<bits/stdc++.h>
2 using namespace std;
3 const int MAXN=305;
4 int w[MAXN],dp[MAXN][MAXN],m,n;
5 vector<int> edge[MAXN];
6 inline void dfs(int x) {
7     for(int i=0;i<edge[x].size();++i) {
8         int s=edge[x][i];
9         dfs(s);
10        for(int j=m;j>0;--j) {
11            for(int k=1;k<=j;++k) {
12                dp[x][j]=max(dp[x][j],dp[x][j-k]+dp[s][k]);
13            }
14        }
15    }
16    return ;
17 }
18 int main() {
19     scanf("%d%d",&n,&m);
20     ++m;
21     for(int i=1;i<=n;++i) {
22         int input;
23         scanf("%d",&input,&dp[i][1]);
24         edge[i].push_back(input);
25     }
26     dfs(0);
27     printf("%d\n",dp[0][m]);
28     return 0;
29 }
```



[例07]要糖

万圣节那天，小 Q 想去邻居要糖果，玩下“Trick or Treat”。小 Q 家和邻居家够成一棵树，即邻居间可以相互到达，并且只有一种方案。

小 Q 的妈妈将在 m 分钟后回来，如果发现小 Q 不在家会很生气。小 Q 希望获得最多的糖果，并且在 m 分钟内回到家中。现在告诉你每条路要花的时间，并假设小 Q 到了邻居家后就能拿到糖果，不需要花时间，当然每个邻居只会给一次糖果，以后经过的时候，就不再给了。

请你帮她算算，她最多能拿到多少糖果。

[输入格式]

输入数据一行，一个正数 n 表示小 Q 所在的地方有多少户人家。

接下一行 n 个数，表示每户人家给她的糖果数 val_i 。

接下来 $n-1$ ，每行三个数 a,b,c ，表示通过连接编号为 a,b 的人家的路要花费 c 分钟的时间， $1 \leq a,b \leq n$ 。

最后一行，两个整数 k,m ， k 表示小 Q 家的编号（小 Q 也能从自己家获得糖果）， m 表示妈妈 m 分钟后回家。

[输出格式]

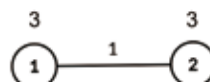
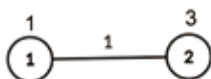
输出一个整数表示小 Q 能获得最多糖果数量。

[数据范围]

$1 \leq n \leq 100, 1 \leq m \leq 200, 0 \leq val_i \leq 1000, 1 \leq c \leq 10$;



输入样例	输出样例
2 1 3 2 1 1 2 1	3 小Q只能获得自己家的糖果，不能在m分钟去其他地方再赶回来。
2 3 3 1 2 1 2 2	6 小Q能获得自己家的糖果和编号为1邻居家的糖果，并恰好在m分钟赶回家中。



小Q只能获得自己家的糖果，不能在m分钟去其他地方再赶回来。

小Q能获得自己家的糖果和编号为1邻居家的糖果，并恰好在m分钟赶回家中。



经典问题三：树的直径

定义：树上最长的路径。

给定一棵树，树中每条边都有一个权值，树中两点之间的距离定义为连接两点的路径边权之和。树中最远的两个节点之间的距离被称为树的直径，连接这两点的路径被称为树的最长链。树的直径可能不唯一。

时间复杂的 $O(n)$ 做法：

方法1--两遍dfs或者bfs:

方法2--树形dp



方法1：两遍dfs或者bfs

第一次任意选一个点进行dfs(bfs)找到离它最远的点，此点就是最长路的一个端点，再以此点dfs(bfs)，找到离它最远的点，此点就是最长路的另一个端点，这两点之间的最短路径就是**树的直径**。

定理1：在一棵树中，以任意结点出发所能到达的最远结点，一定是该树直径的端点之一。

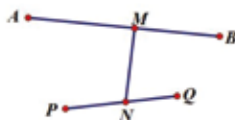
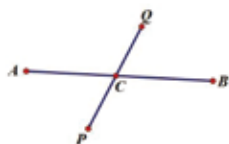
证明：先从任意一点P出发，找离它最远的点Q，再从点Q出发，找离它最远的点W，W到Q的距离就是直径。

①若P已经在直径上，根据树的直径的定义可知Q也在直径上且为直径的一个端点；

②若P不在直径上，我们用反证法，假设此时WQ不是直径，AB是直径且更长。

若AB与PQ有交点C，由于P到Q最远，那么 $PC+CQ > PC+CA$ ，所以 $CQ > CA$ ，易得 $CQ+CB > CA+CB$ ，即 $CQ+CB > AB$ ，与AB是直径矛盾，不成立，如左图（其中AB，PQ不一定是直线，画成直线是为了方便）。

若AB与PQ没有交点，M为AB上任意一点，N为PQ上任意一点。首先还是 $NP+NQ > NQ+MN+MB$ ，同时减掉NQ，得 $NP > MN+MB$ ，易知 $NP+MN > MB$ ，所以 $NP+MN+MA > MB+MA$ ，即 $NP+MN+MA > AB$ ，与AB是直径矛盾，所以这种情况也不成立，如右图。

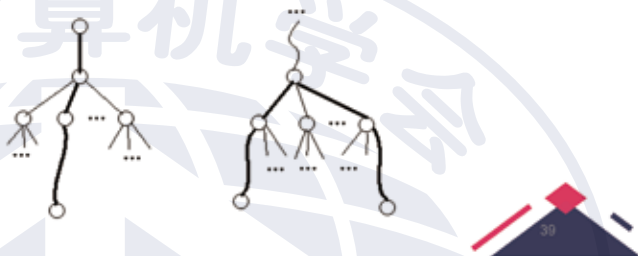


方法2:树形dp

- 考虑一棵树的直径与其根的关系，它要么经过根，要么完全在根的一个子树里。
- 如果直径经过根，考虑根将直径分成的两个部分，它们是子树中从根向下的最长的两条路径，对于直径不经过根的情况递归求解即可。
- 设 $dp[i]$ 为从节点 i 向下的最长路径长度（最长链），dfs过程中统计即可，时间复杂的 $O(n)$ 。

优点：代码量少实现方便。

不足：不容易记录路径。



```

1 void dfs(int u,int fa){//树的直径模板
2     for(int i=head[u];i;i=e[i].nxt){
3         int v = e[i].to, w = e[i].w;
4         if(v==fa) continue;
5         dfs(v,u);
6         //v子树的最长 + w + u子树过去最长链
7         ans = max(ans,dp[v]+dp[u]+w);
8         dp[u] = max(dp[u],dp[v]+w);
9     }
10 }
```



[例08] 马拉松

在听说了传染病在美国的快速传播后，FJ希望他的奶牛们能得到更多的锻炼，因此他决定创建一个马拉松比赛来让他的奶牛们锻炼身体。

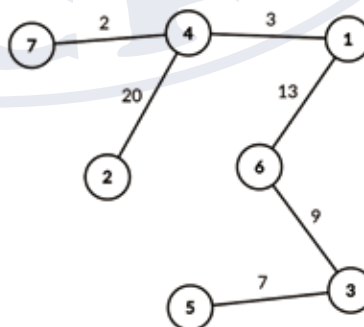
FJ的地图中有很多的农场，农场之间有一些路连接它们。这次的马拉松行程将会包含一对农场以及这两个农场之间的所有道路。由于FJ希望他的奶牛们能得到尽可能多的锻炼，所以他希望找到两个在地图上相隔最远的农场作为马拉松行程中的那一对农场。现在他希望你能帮助他找到这个最远的距离的大小。

（这里要注意的是，FJ的地图是很特殊的，在这个图上，任意两点间有且只有一条通路可以互相到达）

[数据范围]

N为农场数， $1 \leq N \leq 100000$

输入样例	输出样例
7 6 1 6 13 E 6 3 9 E 3 5 7 S 4 1 3 N 2 4 20 W 4 7 2 S	52



[例09] 买房子

有天小 C 突发奇想，自己是不是也该考虑一下买房子的问题了。小 C 所在的城市被划分成 n 个区域，这 n 个区域是连通的，并且从任意一个区域到达另外区域的方案数只有一种。现在这 n 个区域都有房卖，小 C 想，如果他要选择买房区域的话，他所在的区域到其他的区域的距离的最远距离是多少。

现在告诉你 n 个区域的连接情况，请你帮他算算，输出对于每个区域离它最远的距离是多少？

[输入格式]

输入第一行，一个整数 n ;

接下来 $n-1$ 行，每行三个整数 a, b, c ，表示连接区域 a 和 b 的路长为 c ，其中 $0 \leq a, b < n, 0 < c \leq 10000$ 。

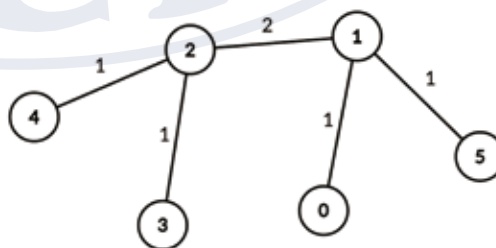
[输出格式]

输出对于每个区域离它最远的距离是多少？

数据范围

N 为农场数， $1 < N \leq 20000$

输入样例	输出样例
6	4
0 1 1	3
1 5 1	3
1 2 2	4
2 3 1	4
2 4 1	4



[例10] 赛道修建[NOIP2018D1T3]

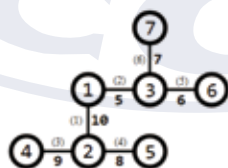
C城将要举办一系列的赛车比赛。在比赛前，需要在城内修建 m 条赛道。

C城一共有 n 个路口，这些路口编号为 $1, 2, \dots, n$ ，有 $n-1$ 条适合于修建赛道的双向通行的道路，每条道路连接着两个路口。其中，第 i 条道路连接的两个路口编号为 a_i 和 b_i ，该道路的长度为 l_i 。借助这 $n-1$ 条道路，从任何一个路口出发都能到达其他所有的路口。

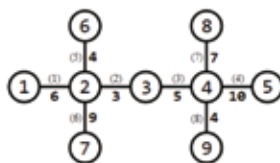
一条赛道是一组互不相同的道路 e_1, e_2, \dots, e_k ，满足可以从某个路口出发，依次经过道路 e_1, e_2, \dots, e_k （每条道路经过一次，不允许调头）到达另一个路口。一条赛道的长度等于经过的各道路的长度之和。为保证安全，要求每条道路至多被一条赛道经过。

任务：设计一种赛道修建的方案，使得修建的 m 条赛道中长度最小的赛道长度最大（即 m 条赛道中最短赛道的长度尽可能大）

输入样例	输出样例
7 1 1 2 10 1 3 5 2 4 9 2 5 8 3 6 6 3 7 7	31
9 3 1 2 6 2 3 3 3 4 5 4 5 10 6 2 4 7 2 9 8 4 7 9 4 4	15



需要修建 1 条赛道。可以修建经过第 3,1,2,6 条道路的赛道(从路口 4 到路口 7)，则该赛道的长度为 $9 + 10 + 5 + 7 = 31$ ，为所有方案中的最大值。



需要修建 3 条赛道。可以修建如下 3 条赛道：

1. 经过第 1,6 条道路的赛道(从路口 1 到路口 7)，长度为 $6 + 9 = 15$;
2. 经过第 5,2,3,8 条道路的赛道(从路口 6 到路口 9)，长度为 $4 + 3 + 5 + 4 = 16$;
3. 经过第 7,4 条道路的赛道(从路口 8 到路口 5)，长度为 $7 + 10 = 17$ 。长度最小的赛道长度为 15，为所有方案中的最大值。

题意：给出有 n 个节点的一棵树的信息，从中选出 m 条链，满足链之间互不相交，并最大化最短链长。

20pt : $m=1$ ，求树的直径

20pt: $b_i=a_i+1$ ，一条链

把所有边权记录下来，这种情况等价于将序列分割成 m 段，使 m 段区间和的最小值最大。二分 m 段区间和的最小值，然后 $O(n)$ 贪心扫一遍，时间复杂度 $O(n \log n)$ 。

20pt: $a_i=1$ 菊花，要么一条，要么2条，贪心配对。

100pt：二分+树形DP+贪心

【数据规模与约定】

所有测试数据的范围和特点如下表所示

测试点编号	n	m	$a_i = 1$	$b_i = a_i + 1$	分支不超过 3
1	≤ 5	$= 1$	否	否	是
2	≤ 10	$\leq n - 1$	是	是	否
3	≤ 15		否	否	是
4	$\leq 1,000$		是	否	否
5	$\leq 30,000$	$= 1$	否	否	否
6		否			
7					
8	$\leq 50,000$	$\leq n - 1$	是		
9	$\leq 1,000$		否	是	是
10	$\leq 30,000$				
11	$\leq 50,000$				
12	≤ 50				
13	≤ 200		否	否	否
14					
15					
16					
17	$\leq 1,000$				
18	$\leq 30,000$				
19	$\leq 50,000$				
20					

其中，“分支不超过3”的含义为：每个路口至多有3条道路与其相连。

对于所有的数据， $2 \leq n \leq 50,000$ ， $1 \leq m \leq n-1$ ， $1 \leq a_i \leq n$ ， $1 \leq b_i \leq 10,000$ 。

算法一： $m=1$ ，求树的直径，20分

算法二：二分+树形DP+贪心

江苏省得分情况：

100分42人

20分以上306人

0分252人

[100pt思路]

题目要求使 m 条赛道中最短赛道的长度尽可能大，不难想到二分最短赛道的长度 len ，并判定是否能修建出 m 条不互相覆盖的赛道长度达到 len ，转换为判定问题。

然后我们可以使用树形dp，自底向上考虑每棵子树对答案的贡献：

1. 以 u 为根的子树可以建出几条达标的赛道；
2. 另外子树上有可利用未匹配链也可以向上构成赛道。

对于 $f[v] + w(u, v) \geq len$ ($v \in son(u)$)的情况，可以直接将其作为一条赛道。而剩余的链，贪心一下，还可以将它们两两拼接成赛道，最后剩余废料赛道中最长的一条还可以跟 u 的父亲去拼接，在剩余的无法匹配的链中取最长值作为 $f[u]$ 。

定义 $f[u]$ 为自节点 u 向下延伸的未匹配赛道的最长链长度，即 u 子树中无法构成满足长度的链中最长的一条。

贪心操作可以用multiset实现，因为有的链可能长度一样。



经典问题四：树的重心

定义1：找到一个点，其所有的子树中最大的子树节点数最少，那么这个点就是这棵树的重心。

定义2：以这个点为根，那么所有的子树（不算整个树自身）的大小都不超过整个树大小的一半。

性质1：树中所有点到某个点的距离和中，到重心的距离和是最小的；如果有两个重心，那么他们的距离和一样。

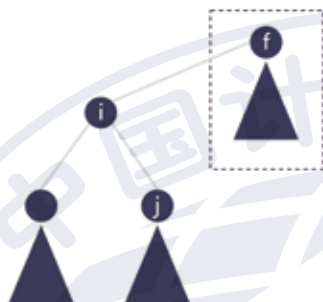
性质2：把两棵树通过一条边相连得到一棵新的树，那么新的树的重心在连接原来两棵树的重心的路径上。

性质3：把一棵树添加或删除一个叶子，那么它的重心最多只移动一条边的距离。



找重心的算法

任选一个结点作为根结点，把无根树变成有根树，然后设 $d(i)$ 表示以 i 为根的子树的结点的个数： $d(i) = \sum d(j) + 1, j \in s(i)$ ， $s(i)$ 为 i 结点的所有儿子结点的编号的集合。算法：只需要DFS一次，在无根树有根数的同时计算即可，连记忆化都不需要——因为本来就没有重复计算，时间复杂度 $O(n)$ 。



删除结点 i 后，最大的连通块有多少个呢？结点 i 的子树中最大有 $\max\{d(j)\}$ 个结点， i 的“上方子树”中有 $n-d(i)$ 个结点

$$d_i = \sum_{j \in s(i)} d(j) + 1,$$



[例11] 黑手党

去年的芝加哥充满了黑帮争斗和奇怪的谋杀。警察局长真的厌倦了所有这些罪行，决定逮捕黑手党领袖。

不幸的是，芝加哥黑手党相当复杂的结构。没有人知道黑手党的信息。警方已经追踪他们一段时间的活动，并且知道他们中的一些人互相通信。根据收集到的资料，警察局长表明黑手党的层次结构可以表示为一棵树。黑手党首脑是树的根，每个节点表示一个人，一个节点的孩子即为这个节点表示的人的直接下属。

更不幸的是，虽然警方知道了匪徒的通讯，他们不知道谁是黑手党首脑。因此他们只有通信关系的无向树。

基于这样的思想，警察局长猜测可能表示黑手党首脑的节点必须满足：在删除它后，包含最多节点的剩余连通块的节点数最小。帮助警察找到所有可能成为黑手党首脑的节点。



[输入格式]

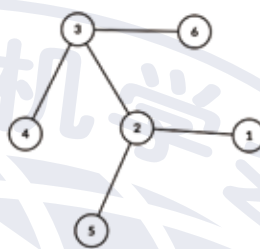
第一行包含一个整数 $n(2 \leq n \leq 50000)$ ，表示有 n 个人，编号为 $1 \sim n$ 。

接下来 $n-1$ 行，每行两个数 x, y ，表示编号为 x 的人和编号为 y 的人之间有通信。

[输出格式]

输出所有可能成为黑手党首脑的人的编号，按升序排列，两个编号之间用空格隔开。

输入样例	输出样例
6	2 3
1 2	
2 3	
2 5	
3 4	
3 6	



[问题分析]

其实本题就是给定一棵树，将树上的一个节点删除后会得到多个连通块，求删去哪些点后，所得到的连通块的节点的最大值最小。这个被删除的点通常把它称为树的重心。

暴力的方法：枚举删去节点 u ，对剩余节点，进行一遍BFS或DFS，找到每个联通块，计算其节点数大小，最大值记为 $f[u]$ ，用 $f[u]$ 去更新答案，求出最小值 ans 。将 $f[u] == ans$ 按照 u 升序输出即可，时间复杂度 $O(n^2)$ 。

DFS的时候记录以每个点 u 为根的子树中， u 删除后的最大连通块的节点数 $f[u]$ ，则有：

$f[u] = \max\{size[v]\}$ ，其中 v 为 u 的孩子， $size[v]$ 为以 v 为根的子树的节点数。

那么，我们同样能在 $O(n)$ 的复杂度内得到根的答案 $f[u]$ ，同时如果将 u 删去，那么 u 的父亲所在的联通块的节点数就是 $n - size[u]$ 。

所以，我们进行一遍DFS就能求出所有点的答案时间复杂度 $O(n)$ 。



二次扫描换根

问题：如果我们知道当前节点 x 为根的结果，能否快速求出以它某儿子 y 为根的结果？

当 y 从根的儿子变成根，它和它的子树都向上提了一层，深度-1，而 x 往下了一层，它除了 y 以外的其他子树也一起往下了一层，深度加1。所以从以 x 为根变为以 y 为根，深度之和减少了 y 的子树的大小，增加了其他节点的个数。



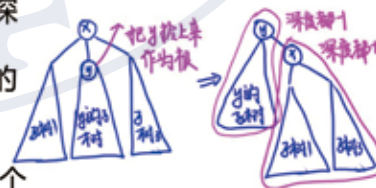
二次扫描换根

(医院设置为例) 我们用 $f[i]$ 表示以 i 为根的时候每个点深度的和，我们可以用一遍 dfs 把 $f[1]$ 求出来 (以 1 为根)，每个点记录一个 $dep[i]$ ，显然 $dep[y] = dep[x] + 1$ (y 是 x 的儿子)，dfs 结束之后求和就好。

根据之前的分析在这次 dfs 的时候我们还需要同时维护一个数组 $cnt[i]$ 表示 i 的子树节点数量 (包括 i)， $cnt[x] = \sum cnt[y]$ ，(其中 y 是 x 的儿子)。

二次扫描：再来一次dfs，遍历到 x 点的时候，去计算 x 的每个儿子 y 的 $f[y]$ 值只需：

$$f[y] = f[x] - cnt[y] + (n - cnt[y])$$



二次扫描换根: 即先算出固定某一点为根的答案后，再考虑把根与它的某个儿子交换会发生什么样的变化，如果这个变化是比较好算的，那么我们就可考虑每个点 v 为根的答案都根据以它父亲 u 为根的结果去推，树边的数量是 $n-1$ ，时间复杂的为 $O(n)$ ，避免重复计算。

[例12] 医院设置

设有一棵二叉树（如图，其中圈中的数字表示结点中居民的人口，圈边上数字表示结点编号。现在要求在某个结点上建立一个医院，使所有居民所走的路程之和为最小，同时约定，相邻结点之间的距离为 1。

就本图而言：

若医院建在 1 处，则距离和
 $= 4 + 12 + 2 \times 20 + 2 \times 40 = 136$ ；

若医院建在 3 处，则距离和
 $= 4 \times 2 + 13 + 20 + 40 = 81$...



输入样例	输出样例
5	81
13 2 3	
4 0 0	
12 4 5	
20 0 0	
40 0 0	

题意：求带权树的重心。

定义几个数组：

- $f[u]$ 表示以 u 为根的总距离和；
- $size[u]$ 表示以 u 为根的子树的大小，显然， $ans = \min(f[i], 1 \leq i \leq n)$

首先我们以 1 号节点为根，求出以该点为医院，所有人到医院的总距离和。

换根：当根从 u 变为 v 的时候， v 的子树的所有节点原本的距离要到 u ，现在只要到 v 了， v 子树每个结点的距离都减少 1，那么总距离就减少 $size[v]$ ，同时，以 v 为根的子树以外的所有节点，原本只要到 u 就行了，现在要到 v ，每个节点的路程都增加了 1，总路程就增加了 $size[1] - size[v]$ ，其中 $size[1]$ 就是我们预处理出来的整棵树的大小，减去 $size[v]$ 就是除去 v 为根的子树 以外的结点数。

对于每个 u 能达到的点 v ，有： $f[v] = f[u] + size[1] - size[v] - size[v]$ ，红色就是 v 为根子树以外节点到数量，对所有节点取 $f[]$ 最小值，时间复杂度 $O(n)$ 。



[例13] 树的双中心 [Shoi2005]

给定一棵树 $T = (V, E)$ ，其中 V 为节点集合， E 为边集合。对于 V 中的每个节点 v ，有一个权值函数 $W(v)$ ，该函数的值均为正整数。记 $d(u, v)$ 为节点 u 和 v 之间的距离，表示它们之间唯一的一条路径的边数。若 u 和 v 为同一个节点，则 $d(u, v) = 0$ 。你的任务是找出不同的节点 x 和 y ，使得以下表达式 $S(x, y)$ 的值最小：

$$S(x, y) = \sum_{v \in V} (W(v) \times \min\{d(v, x), d(v, y)\})$$

$1 < N \leq 50000$ ，树的深度 ≤ 300



输入样例	输出样例
5 1 2 1 3 3 4 3 5 5 7 6 5 4	14 选中2、3节点作为 双中心



- ① $5 * \min(1, 1)$
- ② $7 * \min(0, 2)$
- ③ $6 * \min(0, 2)$
- ④ $5 * \min(1, 3)$
- ⑤ $4 * \min(1, 3)$
- 总共14



暴力：分析样例，设选的两个点为 x, y ，则一定可以将树分成两个集合 A, B ，使得 A 集合所有点都去 x ， B 集合所有点都去 y ，而这两个集合的分界点就是树上的一条边。于是考虑枚举断哪条边，然后对两边分别跑一遍带权树的重心，统计答案加起来取最小值就行了，但是这样做复杂度 $O(n^2)$ 。

优化：以1号节点为根建树， $f[u]$ 表示所有点到 u 的总距离。（人数乘以距离）转移方程是： $f[v] = f[u] + \text{size}[1] - \text{size}[v] - \text{size}[v]$ ，其中 $\text{size}[1]$ 是全树节点总数。

根据上式可以发现，只有当 $\text{size}[v] * 2 > \text{size}[1]$ 时 v 比 u 更优，回忆一下重心定义，找重心的时候一定是往权和大的子树找的，预处理出每个节点的子树大小**最大的儿子**和**次大的儿子**，因为最大子树可能被割断了。每次断边时自下而上修改其所有祖先的 size 大小，这时最大儿子可能变小，进而被次大儿子替代，直接判断一下，然后走此时的大儿子就行。

题目给出树高只有300，所以我们可以从树根开始，不断向靠近重心的方向移动，方法类似“医院设置”，不能移动时就找到了重心，复杂度是 $O(n * h)$ ， h 为树高。



[例14] 树的重心[NOIP2019D2T3]

1. 一个大小为 n 的树由 n 个结点与 $n - 1$ 条无向边构成，且满足任意两个结点间有且仅有一条简单路径。在树中删去一个结点及与它关联的边，树将分裂为若干个子树；而在树中删去一条边（保留关联结点，下同），树将分裂为恰好两个子树。

2. 对于一个大小为 n 的树与任意一个树中结点 c ，称 c 是该树的重心当且仅当在树中删去 c 及与它关联的边后，分裂出的所有子树的大小均不超过 $\text{floor}(L/2)$ 。对于包含至少一个结点的树，它的重心只可能有 1 或 2 个。

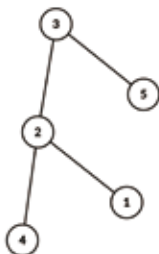
问题：给出了一个大小为 n 的树 S ，树中结点从 $1 \sim n$ 编号，求出 S 单独删去每条边后，分裂出的两个子树的重心编号和之和。

$$\sum_{(u,v) \in E} \left(\sum_{\substack{1 \leq x \leq n \\ \text{且 } x \text{ 号点是 } S'_u \text{ 的重心}}} x + \sum_{\substack{1 \leq y \leq n \\ \text{且 } y \text{ 号点是 } S'_v \text{ 的重心}}} y \right)$$

E 表示树 S 的边集， (u,v) 表示一条连接 u 号点和 v 号点的边。 S'_u 与 S'_v ，分别表示树 S 删去边 (u,v) 后， u 号点与 v 号点所在的被分裂出的子树。

整数 n 表示树 S 的大小，接下来 $n-1$ 行，每行两个以空格分隔的整数 u_i, v_i ，表示树中的一条边 (u_i, v_i) 。

输入样例	输出样例
5 1 2 2 3 2 4 3 5	32
7 1 2 1 3 1 4 3 5 3 6 6 7	56



删去边 (1,2), 1 号点所在子树重心编号为 {1}, 2 号点所在子树重心编号为 {2,3}。
删去边 (2,3), 2 号点所在子树重心编号为 {2}, 3 号点所在子树重心编号为 {3,5}。
删去边 (2,4), 2 号点所在子树重心编号为 {2,3}, 4 号点所在子树重心编号为 {4}。
删去边 (3,5), 3 号点所在子树重心编号为 {2}, 5 号点所在子树重心编号为 {5}。
因此答案为 $(1 + 2 + 3) + (2 + 3 + 5) + (2 + 3 + 4) + (2 + 5) = 32$ 。



删去边 (1,2), 1 号点所在子树重心编号为 {3}, 2 号点所在子树重心编号为 {2}。
删去边 (1,4), 1 号点所在子树重心编号为 {3}, 4 号点所在子树重心编号为 {4}。
删去边 (1,3), 3 号点所在子树重心编号为 {3, 6}, 1 号点所在子树重心编号为 {1}。
删去边 (3,5), 3 号点所在子树重心编号为 {1,3}, 5 号点所在子树重心编号为 {5}。
删去边 (3,6), 3 号点所在子树重心编号为 {1}, 6 号点所在子树重心编号为 {6,7}。
删去边 (6,7), 6 号点所在子树重心编号为 {1,3}, 7 号点所在子树重心编号为 {7}。
因此答案为 $(3 + 2) + (3 + 4) + (3 + 6 + 1) + (1 + 3 + 5) + (1 + 6 + 7) + (1 + 3 + 7) = 56$ 。

测试点编号	$n =$	特殊性质
1 ~ 2	7	无
3 ~ 5	199	无
6 ~ 8	1999	无
9 ~ 11	49991	A
12 ~ 15	262143	B
16	99995	无
17 ~ 18	199995	无
19 ~ 20	299995	无

A : 树的形态是一条链。即 $\forall 1 \leq i < n$, 存在一条边 (p_i, p_{i+1}) 。

B : 树的形态是一个完美二叉树。即 $\forall 1 \leq i \leq (n-1)/2$, 存在两条边 (p_i, p_{2i}) 与 (p_i, p_{2i+1}) 。

保证给出的图是一个树。

题意：给定一棵树，求出删除每一条边后分裂出的两个子树的重心编号和。

40pt(1~8):

最显然的暴力，枚举删除的边($n-1$ 条)，暴力求两个子树的重心，这样是 $O(n^2)$ 的。

15pt(9~11A):

对于链的数据，先把链转成序列，然后在序列上枚举断哪条边，两边的重心就是最中间1~2个点，分奇偶讨论。

20pt(12~15B)：对于二叉树的数据

- 1.删除叶子节点连边，根会成为重心之一。
- 2.删除某个除根节点外的节点连边时，这个点会成为重心之一。
- 3.删除根的左子树某条边时，根的右儿子会成为重心之一，对称同理。

江苏省当年得分：60以上57人，55分62人，40分100人，10~40分70人，10分以下500人。

```

29 int cheA(){
30     memset(in,0,sizeof(head));
31     for(int i=1;i<n;i++)in[e[i].u]++,in[e[i].v]++;
32     for(int i=1;i<n;i++)if(in[i]>2)return 0; //度>2 不是链
33     for(int i=1;i<n;i++)if(in[i]==1)return i; //两个端点
34 }
35 void dfs_link(int x,int fa){
36     p[++tot]=x;
37     for(int i=head[x];i!=-1;i=edge[i].next)
38         if(edge[i].to!=fa)
39             dfs_link(edge[i].to,x);
40 }
41
42 read(T);
43 while(T--){
44     read(n),memset(head,-1,sizeof(head)),cnt=ans=tot=0;
45     for(int i=1,x,y;i<n;i++){
46         read(x),read(y);
47         ae(x,y),ae(y,x);
48         e[i].u=x,e[i].v=y;
49     }
50     int tmp=cheA();//判断是否是链 0不是链, tmp是端点之一
51     if(tmp){//是条链
52         dfs_link(tmp,1);
53         for(int i=1;i<n;i++){
54             if(i&1)ans+=p[(i+1)>>1]; //奇数算一点
55             else ans+=p[i>>1]+p[(i>>1)+1]; //偶数算两点
56             if((n-i)&1)ans+=p[n-((n-i)>>1)];
57             else ans+=p[n-((n-i)>>1)]+p[n-((n-i)>>1)+1];
58         }
59         printf("%lld\n",ans);
60         continue;
61     }

```

15pt：A 一条链

思路：链除了两头入度为1，中间节点的入度为2，可以用 $O(n)$ 遍历确定是否为链，然后枚举断开位置，把链分成两部分，左右的重心由节点奇偶数决定，节点编号也可以用 $O(n)$ 遍历预先存储好。

时间复杂度 $O(n)$

空间复杂度 $O(n)$

代码实现难度：低



```

20 void dfs(int x,int fa){
21     sz[x]=1,SZ++;
22     for(int i=head[x];i!=-1;i=edge[i].next){
23         if(edge[i].to==fa)continue;
24         dfs(edge[i].to,x);
25         sz[x]+=sz[edge[i].to]; // x 的大小
26         f[x]=max(f[x],sz[edge[i].to]); //最大子树大小
27     }
28 }
62 if(n<=2000){
63     for(int i=1;i<=n;i++){//枚举边,分两侧
64         SZ=0;//统计当前子树大小
65         dfs(e[i].u,e[i].v);
66         for(int j=1;j<=n;j++){//找重心
67             if(!sz[j])continue;
68             f[j]=max(f[j],SZ-sz[j]); //
69             if(f[j]<=SZ/2)ans+=j; //当前子树重心 性质
70             f[j]=0,sz[j]=0;
71         }
72         SZ=0,dfs(e[i].v,e[i].u);
73         for(int j=1;j<=n;j++){
74             if(!sz[j])continue;
75             f[j]=max(f[j],SZ-sz[j]);
76             if(f[j]<=SZ/2)ans+=j; //当前子树重心
77             f[j]=0,sz[j]=0;
78         }
79     }
80     printf("%lld\n",ans);
81     continue;
82 }
83 }

```

40pt : $n \leq 2000$

思路：

枚举断开边 $\langle u,v \rangle$ ，树分为 u 和 v 为根的两棵树，分别用树形 dp 找重心，注意重心可能有2个。

时间复杂度 $O(n^2)$

空间复杂度 $O(n)$



```

85 void ecs(){
86     long long rt;
87     for(long long i=1;i<=n;i++){
88         if(A[i].size()==2){
89             rt=i;
90             break;
91         }
92     }
93     long long t1=A[rt][0],t2=A[rt][1];
94     long long ci=(n+1)/2;
95     long long ans=0;
96     for(long long i=1;i<=n;i++){
97         if(i==rt||i==t1||i==t2){//根和根的左右儿子
98             ans+=ci*i;
99         }else{
100             ans+=i;
101         }
102     }
103     writeln(ans);
104 }

```

20pt : $n=262143$ 即 $(2^{18}-1)$ 完全二叉树

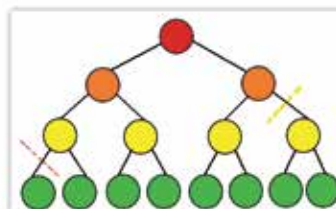
思路：

画图分析，试着分别割去图中带红黄切点的两种边，可以归纳出结论：

删除叶子节点连边，根会成为重心之一。

删除某个除根节点外的节点连边时，这个点会成为重心之一。

删除根的左子树某条边时，根的右儿子会成为重心之一，对称同理。



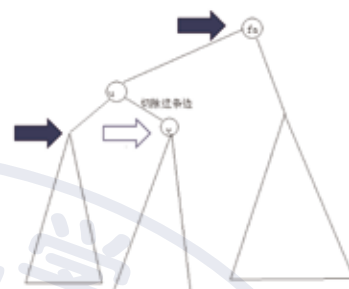
[100pts思路]

有一个性质：重心一定在这个树的重链上

对于以 v 为根的子树的这部分，根据结论，我们沿着以 v 为起点的重链往下跳就可以了。

对于剩余的部分，我们考虑换根，重心应该在重链或者次重链上，或者 u 的父亲节点，将剩余的部分提成以 u 为根的树，修改重儿子和重链即可。

可以用倍增来优化寻找重心的这一部分。时间复杂度 $O(n \log n)$ 。



Part Four

最近公共祖先 (Lowest Common Ancestor)

预备知识--倍增

倍增，每次将范围扩大或减少一倍而达到加速的效果。

举个例子，你想要跳到15米远的地方，你怎么过去？

预设一个k使 $2^k > 15$ 值，这里我们假设 $k=5$ ，

$2^5=32 > 15$ k--; $k=4$; 跳过界了，不跳

$2^4=16 > 15$ k--; $k=3$; 跳过界了，不跳

$2^3=8 \leq 15$ $n=15-8=7$; k--; $k=2$; 没跳到，可以跳

$2^2=4 \leq 7$ $n=7-4=3$; k--; $k=1$; 没跳到，可以跳

$2^1=2 \leq 3$ $n=3-2=1$; k--; $k=0$; 没跳到，可以跳

$2^0=1 \leq 1$ $n=1-1=0$; 跳到了，停

这样我们只跳了4次，与朴素的15次相比，优化了11次！而且随着数据越大优势约明显，这就是 $O(n)$ 和 $O(\log 2n)$ 的区别了，每一个数都可以由二进制来表示，所以每一个距离我们都可以把他拆分成几个 2^k 相加的形式，这个思想在树状数组中也会用到。



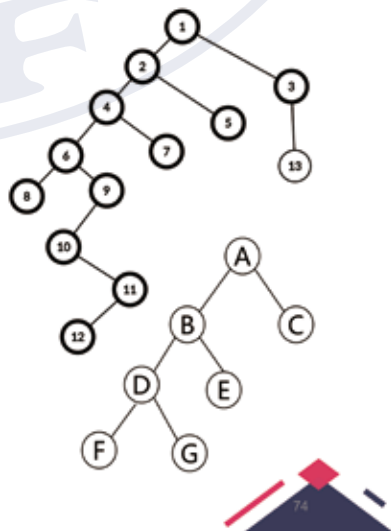
最近公共祖先(LCA)

LCA (Least Common Ancestors)：对于有根树T的两个结点u、v，最近公共祖先 $LCA(u,v)$ 表示一个结点x，满足x是u、v的祖先且x的深度尽可能大(深)。

举例：右图中 $LCA(E, F) = B$ ，同时是 E, F 的祖先结点的点有 {A, B} 其中B 点的深度是最大的。

应用：已知树上最近公共祖先可以快速求出两个点在树上的距离。相当于在树上找两点的最短路，因为最近公共祖先肯定处于两点的最短路径上。

$dis[A] + dis[B] - 2 * dis[LCA(A,B)]$



0.暴力枚举：

首先将两个点中深度较深的那个点向上跳使得两个点的深度相同，然后两个点同时向上跳直到两个点跳到同一个点，这个点就是原来两点的LCA。

1.倍增算法：

属于暴力枚举法的优化，我们没有必要每一次向上跳一步，通过预处理 $fa[i][j]$ 表示结点 i 向上跳 2^j 步到达的结点，就可以将每次跳跃的距离拆分为若干个 2 的幂，这样子求一次LCA的复杂度就优化到了 $O(\log n)$

2.欧拉序ST表+RMQ:

对有根树 T 进行深度优先遍历，每次到达一个节点就把编号记录下来，得到一个长度为 $2N-1$ 的序列，成为树 T 的欧拉序列 F ，同时得到节点深度序列 B ，将 LCA 问题转化为RMQ问题： $LCA(T, u, v) = RMQ(B, first(u), first(v))$ 。

预处理复杂度 $O(n+n\log n)$ ，每次询问的复杂度为 $O(1)$ ，在线算法。



3.Tarjan算法：

将询问保存到被询问的点上，进行DFS，某个子树内部的询问都处理完毕后，这个子树内部的点与外部的点求LCA，答案一定在这个子树之外，这时把子树所形成的集合与当前结点的集合合并，将当前结点设为集合的祖先。

当前结点的所有子树搜索时把当前结点设为检查过的，处理有关当前结点的LCA询问，如果有一个从当前结点到已被检查过的结点 v 的询问，则由于最近公共祖先的包含 v 的子树已经搜索过了，那么这个最近公共祖先一定是 v 所在集合的祖先。

复杂度 $O(n+q)$ ，属于离线算法。

4.树链剖分算法：

利用树链剖分求解，复杂度预处理 $O(n)$ ，单次查询 $O(\log n)$ ，属于在线算法。树链剖分求解LCA中的行为与倍增法类似。



暴力求LCA

- 给定一棵有根树，回答若干询问，每次询问两个点 x, y 的最近公共祖先(LCA)。
- 不难想到的暴力做法：将 x, y 中较深的向根移到同一深度；然后一起一步一步向根移动，直到两者重合。当树的深度比较大，查询数量也比较大，就比较慢了。

```
int LCA (int x, int y) {
    if (depth[x] < depth[y]) swap(x, y);
    while(depth[x] != depth[y]) x = fa[x]; //到达同层
    while(x != y) x = fa[x], y = fa[y]; //一起往上跳
    return x; //LCA
}
```

- 优化：用倍增加速这个过程。



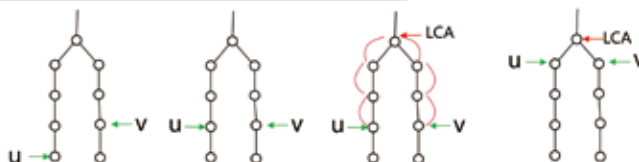
预处理倍增表

令 $f[x][i]$ ($i=0 \dots \log n$) 为从 x 向根移动 2^i 步到达的位置 (如 $f[x][0]$ 就是 x 的父亲)。

```
void dfs(int u, int father) { // 对应深搜预处理 f 数组
    dep[u] = dep[father] + 1; // 深度 + 1
    for (int i = 1; (1 < i) <= dep[u]; i++) // 倍增祖先表
        f[u][i] = f[f[u][i-1]][i-1];
    // 因 u 移动  $2^i$  次就相当于从 u 移动  $2^{i-1}$  次后再移动  $2^{i-1}$  次
    for (int i = head[u]; i != 0; i = edge[i].next) {
        int v = edge[i].to;
        if (v == father) continue; // 往下走
        f[v][0] = u; // u 是 v 的父亲
        dfs(v, u);
    }
}
```



倍增求LCA



求x,y两点的LCA：

首先将x,y提到同一深度：不妨设x当前比y深，从大到小枚举步长 2^i ，若y不比 $f[x][i]$

深则更新x。这里 $i=0 \dots \log n$ ，复杂度 $O(\log n)$

若此时 $x=y$ 则返回结果(y)

从大到小枚举步长 2^i ，若 $f[x][i] \neq f[y][i]$ 则更新x,y。这里 $i=0 \dots \log n$ ，因此复杂度 $O(\log n)$

最后答案即为当前x/y的父节点，即 $f[x][0]$



code 模板

```
const int logN = 18;
int LCA (int x, int y) {
    if (depth[x] < depth[y]) swap(x, y); //让x更深点
    for (int i = LogN; i >= 0; --i) //将x跳到和y同一深度
        if (depth[f[x][i]] >= depth[y])
            x = f[x][i];
    if (x == y) return x;
    for (int i = LogN; i >= 0; --i) //一起向上跳
        if (f[x][i] != f[y][i])
            x = f[x][i], y = f[y][i];
    return f[x][0]; //两个点均在其LCA的下方，再往上跳一次
}

int c = depth[a] - depth[b];
for (int i = 0; i <= 14; i++) {
    if (c & (1 << i)) {
        a = up[a][i];
    }
} //跳同层 位运算优化
```

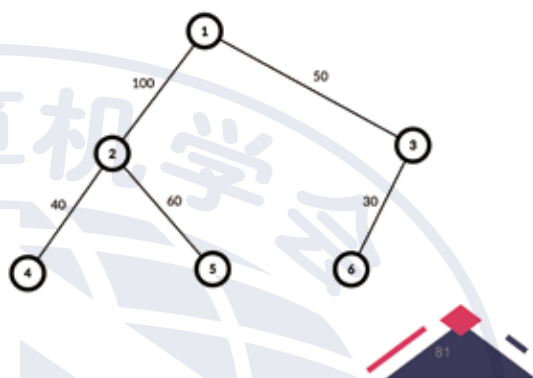


[例15] 点的距离

给定一棵 n 个点的树，带边权， q 个询问，每次询问点 x 到点 y 两点之间的距离。

$1 \leq q, n \leq 10^4$

输入样例	输出样例
6 2	180
1 2 100	240
1 3 50	
2 4 40	
2 5 60	
3 6 30	
2 6	
5 6	

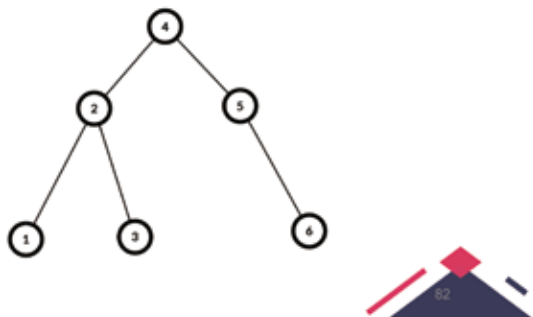


[例16] 聚会

Y 岛上有 N 个城市，有 $N-1$ 条城市间的道路连接着它们。每一条道路都连接某两个城市。幸运的是，小可可通过这些道路可以走遍 Y 岛的所有城市。神奇的是，乘车经过每条道路所需要的费用都是一样的。小可可，小卡卡和小 Y 经常想聚会，每次聚会，他们都会选择一个城市，使得三个人到达这个城市的总费用最小。

$1 \leq q, n \leq 5 \times 10^5$

输入样例	输出样例
6 4	5 2
1 2	2 5
2 3	4 1
2 4	6 0
4 5	
5 6	
4 5 6	
6 3 1	
2 4 4	
6 6 6	



结论1：集合点一定在某两个点的lca上；

结论2：3个点两两算出lca，至少有2个lca相同；

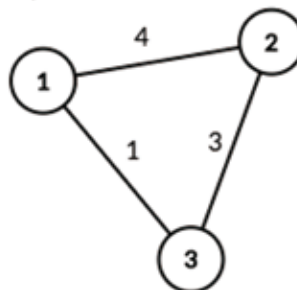
结论3：不同的那个lca（或3个都相同的lca）就是集合点，3个点到这个点的总距离最小。



[例17] 货车运输[noip2013]

- A 国有 n 座城市，编号从 1 到 n ，城市之间有 m 条双向道路。每一条道路对车辆都有重量限制，简称限重。现在有 q 辆货车在运输货物，司机们想知道每辆车在不超过车辆限重的情况下，最多能运多重的货物。
- $0 < n < 10,000$, $0 < m < 50,000$, $0 < q < 30,000$

输入样例	输出样例
4 3	3
1 2 4	-1 //不能到达
2 3 3	3
3 1 1	
3	
1 3	
1 4	
1 3	



解题思路

题意：求图中任意两点间所有路径中最小边权的最大值。

1.要使得货车装最重的货物，那么就必须货车走的路载重要尽可能大——水桶原理，也就是说，载重较小的路是不会被走过的。可以构建原图的最大生成树，这样就可以在满足各城市之间连通情况不变的前提下，使得城市之间路载重最大。求货车能装最大货物重量的问题，就转化成了求树上两点路径上权值最小的边的问题。

2.查询需要优化， u 到 v 的最小边权就是： $\min(u$ 到 lca 的最小边权， v 到 lca 的最小边权)。

u, v 到 lca 的最小边权可以倍增处理：

$\minw[i][j]$ 表示节点到其 2^j 级祖先之间的最小边权

$\minw[i][j] = \min(\minw[i][j-1], \minw[g[i][j-1]]); // g = fa[i][j-1]$

3.最后就是计算答案的部分：

- 对于不连通的两个点很好判断，用并查集判一下，如果不连通直接输出-1
- 如果两点连通：倍增过程中每次跳到其 2^j 级节点时，进行操作： $ans = \min(ans, \minw[u][j])$



[例18] 保卫王国(NOIP2018D2T3)

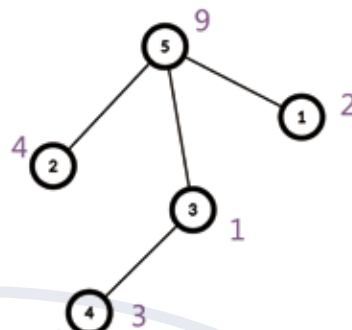
Z 国有 n 座城市， $n-1$ 条双向道路，每条双向道路连接两座城市，且任意两座城市 都能通过若干条道路相互到达。Z 国的国防部长小 Z 要在城市中驻扎军队。驻扎军队需要满足如下几个条件：

- 一座城市可以驻扎一支军队，也可以不驻扎军队。
- 由道路直接连接的两座城市中至少要有一座城市驻扎军队。
- 在城市里驻扎军队会产生花费，在编号为 i 的城市中驻扎军队的花费是 p_i 。

小 Z 很快就规划出了一种驻扎军队的方案，使总花费最小。但是国王又给小 Z 提出了 m 个要求，每个要求规定了其中两座城市是否驻扎军队。小 Z 需要针对每个要求逐一 给出回答。具体而言，如果国王提出的第 j 个要求能够满足上述驻扎条件（不需要考虑 第 j 个要求之外的其它要求），则需要给出在此要求前提下驻扎军队的最小开销。如果 国王提出的第 j 个要求无法满足，则需要输出 $-1 (1 \leq j \leq m)$ 。现在请你来帮助小 Z。



输入样例	输出样例
5 3 C3 2 4 1 3 9 1 5 5 2 5 3 3 4 1 0 3 0 2 1 3 1 1 0 5 0	12 7 -1
0不许驻扎 1必须驻扎 n,m ≤ 100000	对于第一个要求，在 4 号和 5 号城市驻扎军队时开销最小。 对于第二个要求，在 1 号、2 号、3 号城市驻扎军队时开销最小。 第三个要求是无法满足的，因为在 1 号、5 号城市都不驻扎军队就意味着由道路直接连接的两座城市中都没有驻扎军队。 A: 链 B: 深度 < 100 1: 约束 2 第一个点固定 2: 约束 2 两个点相邻



测试点编号	type	n =	m =
1~2	A3	10	10
3~4	C3		
5~6	A3	100	100
7	C3		
8~9	A3	2000	2000
10~11	C3		
12~13	A1	100000	100000
14~16	A2		
17	A3		
18~19	B1		
20~21	C1		
22	C2		
23~25	C3		

题意：给一棵树染色，每个节点染色需要一定的花费，要求相邻两个节点至少有一个被染色，给出一些限制条件，求满足每个限制条件的最小花费为多少。

44pt思路：最大独立集问题，强制选、不选的限制很容易满足，只需要先把询问中节点的反向状态权值 $+\infty$ ，堵塞相反状态，输出答案时分情况讨论，询问结束撤销修改。

$n, m \leq 2000$ ， $O(m \cdot n)$ 做法，对于 m 个约束条件，进行 $O(n)$ 的树上dp。

考虑先写出普通的 DP 方程，用 $f(x, 0/1)$ 表示 x 不选/选的话，有：

$$f(x, 0) = \sum_{v \in \text{son}(x)} f(v, 1)$$

$$f(x, 1) = \sum_{v \in \text{son}(x)} \min\{f(v, 0), f(v, 1)\} + a_x$$



思路

对于每一次询问，除了被询问的两个点之间的链上，各节点的答案会发生改变，其余节点的答案是不变的，没有修改点权，可以倍增。

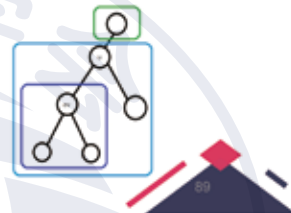
令 T 为整棵树，设 $f[i][0/1]$ 表示（以 i 为根的子树），其中 i 选/不选的最小代价，
设 $g[i][0/1]$ 为（ T - 以 i 为根的子树），其中 i 选/不选的最小代价。
这两个数组可以两遍dfs树形dp求出。

然后令 anc 表示 i 的 2^j 祖先， $dp[i][j][0/1][0/1]$ 表示（ anc 的子树 - i 的子树），其中 i 的状态为 $0/1$ ， anc 的状态为 $0/1$ 的最小代价，这个数组可以用 i 的 $2^{(j-1)}$ 祖先的状态直接转移。然后有了dp数组我们就可以处理询问了。

如果 a 是 b 的祖先，那么可以直接倍增上去（还是像dp一样合并倍增数组，枚举中间点的状态即可），然后不要忘了加上 $g[a][x]$ 。否则我们需要先把 a 和 b 都倍增到 lca 的儿子处，然后枚举 lca 和两个儿子的状态，具体可以见代码。

复杂度 $O((n+q)\log n)$ 。

当年江苏没有满分，52分以上33人，44分以上141人，0分515人。






Part Five

树上差分



05

一维差分

- 一维差分，对于区间 $[l, r]$ 来说，在 l 处加上 x ，在 $r+1$ 处减去 x ，再求前缀和即可。

原数列	9	4	7	5	9
前缀和	9	13	20	25	34
差分数组	9	-5	3	-2	4
前缀和的差分数组	9	4	7	5	9
差分数组的前缀和	9	4	7	5	9

- 原数列的前缀和的差分数组还是原数列，原数列的差分数组的前缀和也是原数列，这就是差分被称为前缀和的逆运算。



树上差分

树上差分算法，是一个适用于树上区间操作的算法，它是差分数组，前缀和求解的树上拓展，利用差分的性质，对路径上的重要节点进行修改（而不是暴力逐个全改），作为其差分数组的值，最后在求值时，利用 *dfs* 遍历求出差分数组的前缀和，就可以达到降低复杂度的目的，另外通常需要 LCA 的配合。

对点和边的树上差分原理相同，实现略有不同，主要是处理两种操作：

- 找每条边被所有路径覆盖的次数
- 找每个点被所有路径覆盖的次数



点差分

设将两点 u, v 之间路径上的所有点权增加 x ， $lca = LCA(u, v)$ ， lca 的父亲节点为 p ，则操作如下：

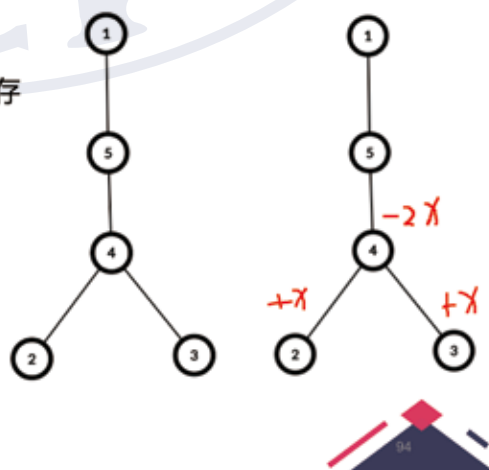
$diff[u] += x, diff[v] += x;$
 $diff[lca] -= x, diff[p] -= x;$



边差分

设将两点 u, v 之间路径上的所有边权增加 x ， $lca = LCA(u, v)$ ，以每条边两端深度较大的节点存储该边的差分数组，则操作如下：

$diff[u] += x, diff[v] += x;$
 $diff[lca] -= 2 * x;$

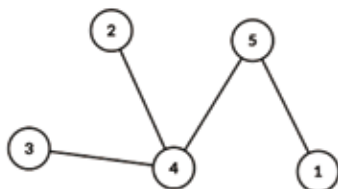



```
void dfs(int x){//子树求和
    //sum[x]=...
    for(int i=head[x];i!=-1;i=edge[i].nt)
    {
        int v=edge[i].to;
        if(v==fa[x])continue;
        dfs(v);
        sum[x] += sum[v]; //累加子树信息
    }
}
```

[例19] 最大流量

FJ给他的牛棚的 N 个隔间之间安装了 $N-1$ 根管道，隔间编号从 1 到 N 。所有隔间都被管道连通了。

FJ 有 K 条运输牛奶的路线，第 i 条路线从隔间 S_i 运输到隔间 T_i 。一条运输路线会给它的两个端点处的隔间以及中间途径的所有隔间带来一个单位的运输压力，你需要计算压力最大的隔间的压力是多少。

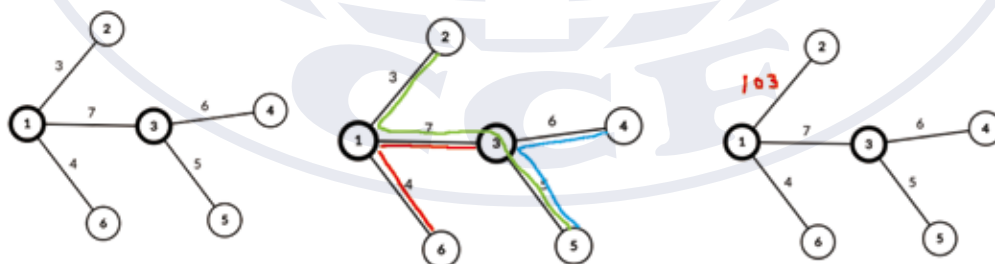


输入样例	输出样例
5 10	9
3 4	
1 5	
4 2	
5 4	
5 4	
5 4	
3 5	
4 3	
4 3	
1 3	
3 5	
5 4	
1 5	
3 4	

[例20]运输计划transport[NOIP2015D2T3]

- 1 L 国有 n 个星球，还有 $n - 1$ 条双向航道，每条航道建立在两个星球之间，这 $n - 1$ 条航道连通了 L 国的所有星球。
- 2 小 P 掌管一家物流公司，该公司有很多个运输计划，每个运输计划形如：有一艘物流飞船需要从 u_i 号星球沿最快的宇航路径飞行到 v_i 号星球去。显然，飞船驶过一条航道是需要时间的，对于航道 j ，任意飞船驶过它所花费的时间为 t_j ，并且任意两艘飞船之间不会产生任何干扰。
- 3 L 国国王同意小 P 的物流公司参与 L 国的航道建设，即允许小 P 把某一条航道改造成虫洞，飞船驶过虫洞不消耗时间。
- 4 小 P 的物流公司就预接了 m 个运输计划。在虫洞建设完成后，这 m 个运输计划会同时开始，所有飞船一起出发。
- 5 如果小 P 可以自由选择将哪一条航道改造成虫洞，试求出小 P 的物流公司完成阶段性工作所需要的最短时间是多少？

题意：给定一棵有 n 个节点的树（带有边权），再给定 m 条树中的路径。允许将一条树边的权改为 0，并使得这些路径的最长距离最小。输出该修改方案中这些路径中最长的一条。（ $n, m \leq 300000$ ）



输入样例	输出样例
6 3	11
1 2 3	
1 6 4	
3 1 7	
4 3 6	
3 5 5	
3 6	
2 5	
4 5	

将第1条航道改造成虫洞：则三个计划耗时分别为：11、12、11，故需要花费的时间为12。
 将第2条航道改造成虫洞：则三个计划耗时分别为：7、15、11，故需要花费的时间为15。
 将第3条航道改造成虫洞：则三个计划耗时分别为：4、8、11，故需要花费的时间为11。
 将第4条航道改造成虫洞：则三个计划耗时分别为：11、15、5，故需要花费的时间为15。
 将第5条航道改造成虫洞：则三个计划耗时分别为：11、10、6，故需要花费的时间为11。
 故将第3条或第5条航道改造成虫洞均可使得完成阶段性工作的耗时最短，需要花费的时间为11。

思路

容易想到，可以从路径中枚举删边，但不一定是所有路径会经过被删边(难点)。

要求最大值最小，首先想到的是二分答案，即二分最终这些路径的长度中最长的一条，设为 k ，然后再判断是否可以通过删除一条边，让所有路径长度 $\leq k$ 。那么该如何判断可行性呢。

1. 我们首先要预处理出来每一条路径的长度。

2. 二分答案的时候检查，删除掉共同覆盖的最长边后(贪心)，能否所有路径都 $\leq \text{mid}$ ，先找出路径的长度大于 mid (二分的答案)的边，也就是超标的边，小于 mid 显然不需要管，然后用边差分标记路径所经过边的覆盖次数，然后找所有超标路径都经过并且边中最长的一条删除(贪心)。

3. 然后扫描所有的路径，枚举删除，如果删去某条边能让所有超 mid 的路径都变得小于等于 mid 了，那么这个答案就是合法的，然后通过二分去找临界值。



优化、避免被卡常

最后一个问题，数据比较大，一定要注意常数优化(下面是建议)：

- 1、将用 `vector` 存的邻接表改成链式前向星，
- 2、读写优化
- 3、可以使用 Tarjan 算法来求 LCA， $O(n)$ 。
- 4、二分的时候可将初始范围尽量缩小。

使用的是倍增算法求 LCA 时间复杂度 $O(n \log n)$ ，空间复杂度 $O(n \log n)$ 。



[例21]天天爱跑步[NOIP2016D1T2]

这个游戏的地图可以看作一棵包含 n 个节点和 $n-1$ 条边的树，任意两个节点存在一条路径互相可达。树上节点的编号是 $1 \sim n$ 之间的连续正整数。

现在有 m 个玩家，第 i 个玩家的起点为 S_i ，终点为 T_i 。每天打卡任务开始时，所有玩家在第 0 秒同时从自己的起点出发，以每秒跑一条边的速度，不间断地沿着最短路径向着自己的终点跑去，跑到终点后该玩家就算完成了打卡任务（因为地图是一棵树，所以每个人的路径是唯一的）。

每个节点上都有一个观察员。在节点 j 的观察员会选择在第 w_j 秒观察玩家，一个玩家能被这个观察员观察到当且仅当该玩家在第 w_j 秒也正好到达了节点 j 。

注意：一个玩家到达自己的终点后，该玩家就会结束游戏，他不能等待一段时间后再被观察员观察到。即对于把节点 j 作为终点的玩家：若他在第 w_j 秒前到达终点，则在节点 j 的观察员不能观察到该玩家；若他正好在第 w_j 秒到达终点，则在节点 j 的观察员可以观察到这个玩家。

请计算每个观察员会观察到多少人？



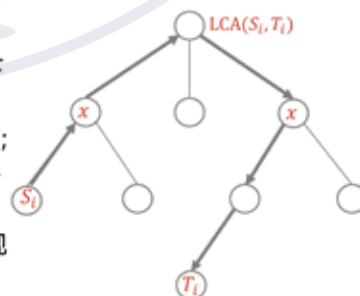
思路

每个玩家跑步的路线可以拆成“上升”和“下降”两段，即 $[S_i, LCA(S_i, T_i)]$ 和 $(LCA(S_i, T_i), T_i]$ 。

位于节点 x 的观察员能观察到第 i 个玩家，当且仅当满足以下两个条件之一：

- 点 x 处于 $S_i \sim LCA(S_i, T_i)$ 上，并且满足 $d[S_i] - d[x] = w[x]$ ；
- 点 x 处于 $LCA(S_i, T_i) \sim T_i$ 上（不含 $LCA(S_i, T_i)$ 这个端点），并且满足 $d[S_i] + d[x] - 2 \times d[LCA(S_i, T_i)] = w[x]$ 。

这两个条件代表玩家从 S_i 跑到 x 所用的时间，等于观察员出现的时间 $w[x]$ 。



思路

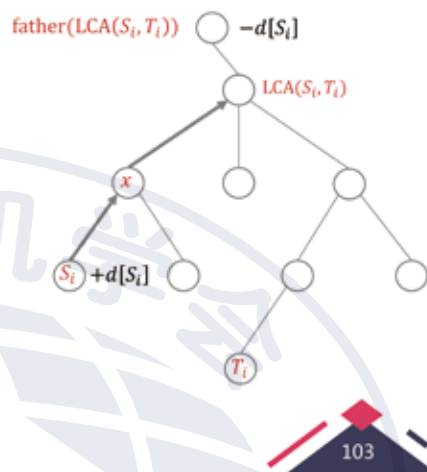
以上升路径为例, 等式 $d[S_i] - d[x] = W[x]$ 移项为

$$d[S_i] = W[x] + d[x]$$

转化为以下模型：

- 有 m 个玩家，其中第 i 个玩家给 $S_i \sim LCA(S_i, T_i)$ 上的每个节点增加一个类型为 $d[S_i]$ 的物品。最终求每个点 x 处类型为 $W[x] + d[x]$ 的物品有多少个。

通过树上差分转化为：起点 S_i 物品增加 $d[S_i]$ 和终点 $father(LCA(S_i, T_i))$ 物品 $d[S_i]$ 消失。



思路

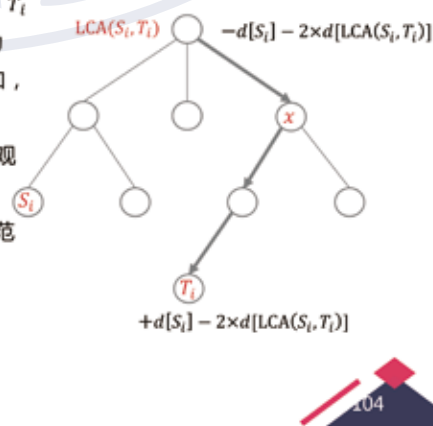
“下降”路径：

- $d[S_i] - 2 \times d[LCA(S_i, T_i)] = W[x] - d[x]$

通过树上差分转化为：物品 $d[S_i] - 2 \times d[LCA(S_i, T_i)]$ 在点 T_i 产生，在点 $LCA(S_i, T_i)$ 处消失，最后求每个点 x 处类型为 $W[x] - d[x]$ 的物品数量。与“上升”路径得到的结果相加，就是点 x 的观察员能观察到的玩家总数。

在树上每个点处使用树上前缀和即可算出子树树根上能观察到的玩家总数。

注意：此时物品类型可能是负数，需要对计数数组的下标范围进行平移或离散化。



思路

如果是计算区间最值，线段树是较为适合的数据结构，而本题仅需计算求和。可以在树上每个节点处建立 4 个 *vector*，分别记录上升、下降路径的增加和消失物品编号。

通过统计“子树递归和回溯之间的差值”来实现计算，算法如下：

扫描 m 个玩家，在每个玩家活动路径的 4 个端点的对应 *vector* 上记录“产生”和“消失”的物品编号；

建立全局数组 c ，对每种类型的物品进行计数，初值全为 0；

对整棵树进行 DFS：

① 递归进入每个点 x 时，用局部变量 val_1 记录 $c[W[x] + d[x]]$ ， val_2 记录 $c[W[x] - d[x]]$ ；

② 递归遍历 x 的所有子树，在子树中更新全局数组 c 。

③ 扫描点 x 的 4 个 *vector*，在 c 中执行修改（按照物品类型进行增减）；

④ 从 x 回溯之前，用新的 $c[W[x] + d[x]]$ 减去 val_1 ，加上 $c[W[x] - d[x]]$ 减去 val_2 的值，就是“子树和”，即点 x 处类型为 $W[x] + d[x]$ 和 $W[x] - d[x]$ 的物品数量。

“遍历以 x 为根的子树过程中累加的值”等于“遍历完成时的值”与“遍历开始时的值”之差，也是树上差分的一种形式。

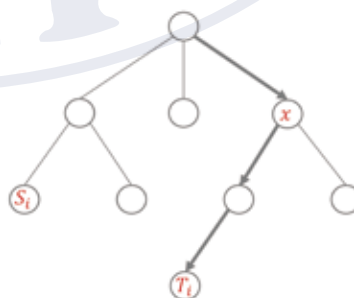


思路

x 点	vector	物品1编号	物品2编号	物品3编号
	上升增加			
	上升减少			
	下降增加			
	下降减少			

开始从 x 点递归	
物品1数量	
物品2数量	
.....	
物品101数量	
.....	
物品 n 数量	

从 x 点回溯前	
物品1数量	
物品2数量	
.....	
物品101数量	
.....	
物品 n 数量	





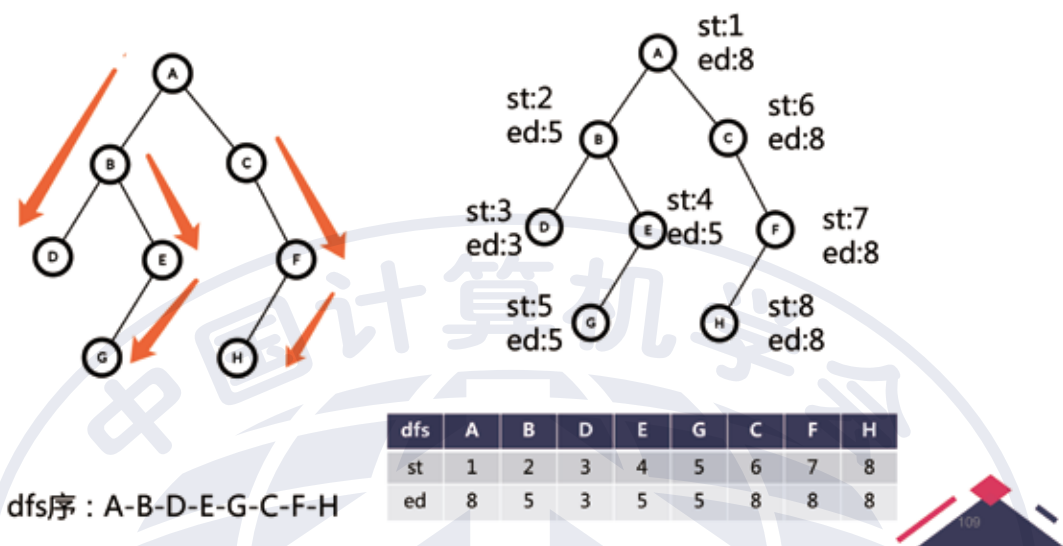
DFS序的定义

dfs序就是按照dfs的顺序对一棵树上的结点进行编号，把树形结构映射成线性结构，然后通过树状数组、线段树等数据结构维护子树区间信息。

dfs序优点在于每一棵树/子树的编号是连续的，这样可以方便的进行任意子树的修改。

dfs序主要是用于处理对于整棵子树的修改，比如子树每个节点权值加减 v ，或查询树上某个点的值。





[例22] Apple Tree

有一棵苹果树，上面一开始长满了苹果，需要维护两种操作

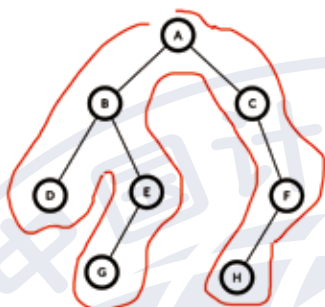
1 : C x , 表明编号为 x 的结点上的苹果存在状态发生了改变，如果原来是存在的则摘下，如果原来不存在则长出新的苹果。

2 : Q x , 统计并输出以 x 为根的子树中有的苹果的个数。



欧拉序的定义

DFS时，第一次到达该结点记录一次，随后每访问完该结点的一棵子树就再记录一次，一共 $2n-1$ （每有一条边就有两个结点要访问，外加根节点被多访问一次）个编号。



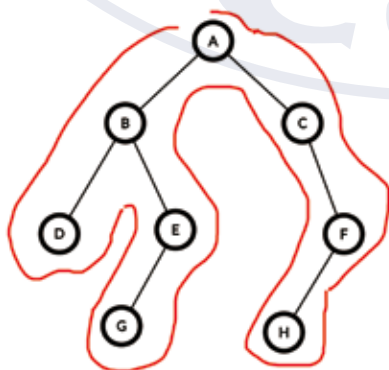
欧拉序1为A-B-D-B-E-G-E-B-A-C-F-H-F-C-A

```
void dfs(int u, int fa)
{
    a[++len] = u;
    int sz = g[u].size();
    for (int i = 0; i < sz; i++)
    {
        if (g[u][i] != fa)
        {
            dfs(g[u][i], u);
            a[++len] = u;
        }
    }
}
```



DFS时，每个节点入栈与出栈时分别记录一次，共 $2n$ 个编号。

欧拉序2为A-B-D-D-E-G-G-E-B-C-F-H-H-F-C-A

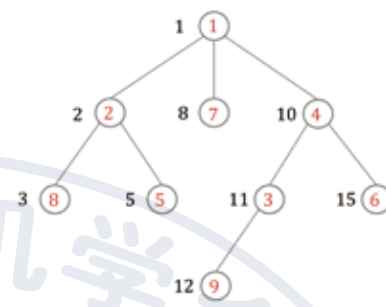


```
void dfs(int x, int fa) {
    dfn[cnt++] = x;
    for (int i = 0; i < ve[x].size(); i++) {
        int to = ve[x][i];
        if (to == fa) continue;
        dfs(to, x);
    }
    dfn[cnt++] = x;
}
```



欧拉序+RMQ求 LCA

- 对一棵树进行 DFS，无论是第一次访问还是回溯，每次到达一个结点时都将编号记录下来，可以得到一个长度为 $2n - 1$ 的序列，这个序列被称作这棵树的欧拉序。
- 把结点 u 在欧拉序中第一次出现的位置编号记为 $pos(u)$ （也称作节点 u 的欧拉序），把欧拉序本身记作 $E[1..2n - 1]$ 。
- 从 u 走到 v 的过程中一定会经过 $LCA(u, v)$ ，但不会经过 $LCA(u, v)$ 的祖先。
- 因此，从 u 走到 v 的过程中经过的欧拉序中深度最小的结点就是 $LCA(u, v)$ 。
- $pos(LCA(u, v)) = m \text{ in } \{pos(k) \mid k \in E[pos(u)..pos(v)]\}$
- 用 DFS 计算欧拉序列的时间复杂度是 $O(n)$ ，且欧拉序的长度也是 $O(n)$ ，所以 LCA 问题可以在 $O(n)$ 的时间内转化成等规模的 RMQ 问题。



请写出欧拉序



根结点总在中间，而根结点是该段序列深度最小的点。

两个点的LCA，就是在该序列上两个点第一次访问出现的区间内，深度最小的那个点。

$$LCA(9, 6) = 4$$

dfn	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
id	1	2	8	2	5	2	1	7	1	4	3	9	3	4	6	4	1
dep	1	2	3	2	3	2	1	2	1	2	3	4	3	2	3	2	1

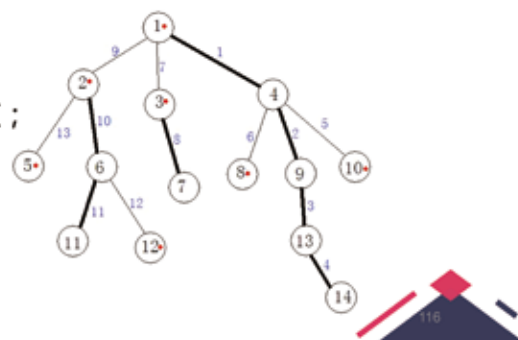


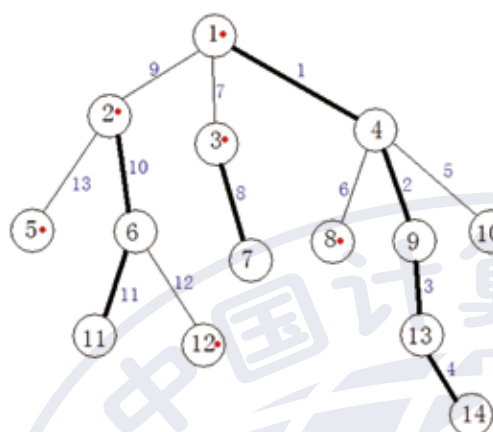


树链剖分（轻重树链剖分）：
把一棵树按重儿子优先方式剖分为若干条链，再按重链优先产生 dfs 序，
然后利用数据结构(树状数组、线段树、Splay等)去维护每一条链，复杂度为 $O(\log n)$ ，思想还是化繁为简。

概念：

- 重结点：子树结点数最多的结点；
- 轻结点：父亲结点中除重结点以外的结点；
- 重边：父亲结点和重结点连成的边；
- 轻边：父亲结点和轻结点连成的边；
- 重链：由多条重边连接而成的路径；
- 轻链：由多条轻边连接而成的路径；





如图所示，较粗的为重边，较细的为轻边。

节点编号旁边有个红色点的表明该节点是其所在链的顶端节点。

边旁的蓝色数字表示该边在线段树中的位置。

图中1-4-9-13-14为一条重链。



树链剖分的第一步当然是对树进行轻重边的划分。

定义 $size[x]$ 为以 x 为根的子树节点个数，令 v 为 u 的儿子中 $size$ 值最大的节点，那么 (u,v) 就是重边，其余边为轻边。

当然，关于这个它有两个重要的性质：

(1) 轻边 (u,v) 中， $size[v] \leq size[u]/2$

(2) 从根到某一点的路径上，不超过 $\log n$ 条轻边和不超过 $\log n$ 条重边。

剖分过程分为两次 **dfs**，或者 **bfs** 也可以。

第一次 **dfs** 就是找重边，也就是记录下所有的重边，记录每个节点的链头。

第二次 **dfs** 就是连接重边形成重链，具体过程就是：以根节点为起点，沿着重边向下拓展，拉成重链，不在当前重链上的节点，都以该节点为起点向下重新拉一条重链。

剖分完毕后，每条重链相当于一段区间，然后用数据结构去维护，把所有重链首尾相接，放到数据结构上，然后维护整体。



第一遍dfs：

dep: 维护每个点的深度，这个在进行最后操作的时候要用；

son: 重儿子序号；

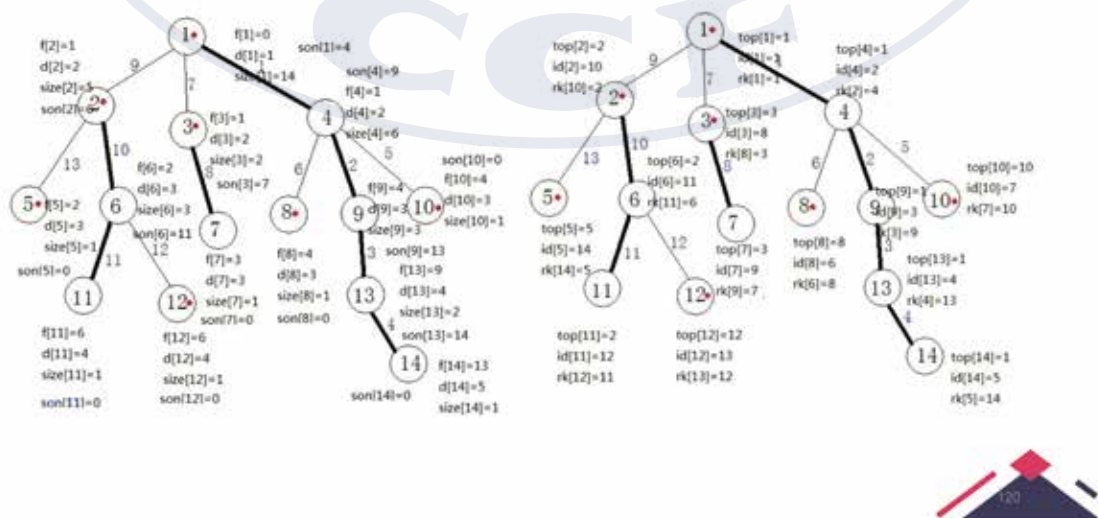
fath: 当前节点的父亲节点编号；

siz: 用来存以当前节点为根的子树的大小，最后利用这个来找重儿子；

第二遍dfs：

id: 每个节点剖分后的新编号，重儿子优先的dfs序；

top: 用来记录每个节点所在树链的顶端节点是谁。



[例23]树上操作[HAOI2015]

有一棵点数为 N 的树，以点 1 为根，且树点有边权。然后有 M 个操作，分为三种：

操作 1：把某个节点 x 的点权增加 a 。

操作 2：把某个节点 x 为根的子树中所有点的点权都增加。

操作 3：询问某个节点 x 到根的路径中所有点的点权和。

任务：对于每个询问操作，输出该询问的答案。

对于 100% 的数据， $N, M \leq 100000$ ，且所有输入数据的绝对值都不会超过 10^6 。

- 轻重链的树链剖分，将树上关于链的问题转化成线性区间问题进行求解，化繁为简。
- 区间问题，我们可以采用一些数据结构求解，比如线段树、树状数组等。
- 一般来讲，树链剖分可以解决以下问题：
 - 修改两点路径上各点的值
 - 查询两点路径上各点的值
 - 修改某点子树上各点的值
 - 查询某点子树上各点的值
 - 求解LCA问题





Return 0;

参 考 资 料

- 《树的算法及其应用》 金靖
- 《树论》 符水波
- 网上资料 不——列举



DFS 序的经典应用

成都外国语学校 祝敏磊

DFS 序

对于一棵树，我们可以通过深度优先搜索遍历它。我们记录到达每一个点的时间，由时间戳构成的序列就是 dfs 序，每个时间戳代表一个节点，序列上的先后关系就是我们在树上遍历到达它的时间的先后顺序。

$t \leftarrow 0$

procedure DFS(x)

$t \leftarrow t + 1$

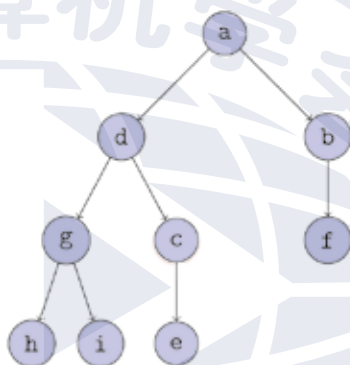
$seq[t] \leftarrow x$

$dfn[x] \leftarrow t$

for all $y \in x$'s sons **do**

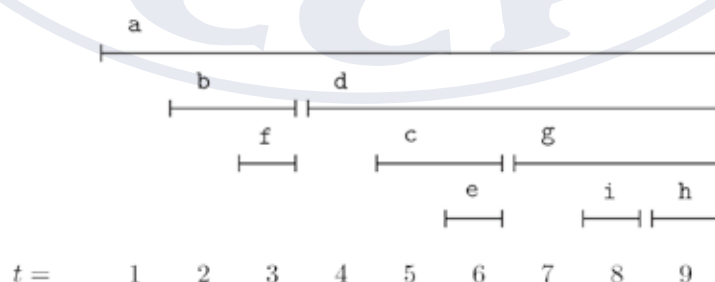
 DFS(y)

$end[x] \leftarrow t$



一种可能的 dfs 序列：

x	a	b	f	d	c	e	g	i	h
dfn	1	2	3	4	5	6	7	8	9
end	9	3	3	9	6	6	9	8	9



我们对一棵大小为 n 的树的根节点执行过程 DFS，得到一个长度为 n 的访问序列 seq ，以及对每个节点的（进入）时间戳 dfn 、结束时间戳 end 。虽然访问某个节点所有儿子的顺序可能不固定，但对于节点 x ，它子树中的所有节点一定就是 $seq[dfn[x], \dots, end[x]]$ 。也就是说，子树和区间形成了一一对应。这样，我们对子树的修改和查询，变成了序列上的区间问题。

例题

例题 1 - LibreOJ 144. DFS 序 1

n 个节点的树, m 次操作, 单点加, 子树求和。

dfs 序转化为序列上单点加, 区间求和。

```
1  #include <bits/stdc++.h>
2  using ll = long long;
3
4  constexpr int N = 1e6 + 10;
5  int n, m, root, a[N], dfn[N], t, end[N];
6  std::vector<int> e[N];
7
8  struct BIT {
9      void init(int n);
10     void add(int i, ll a);
11     ll ask(int i) const;
12 } bit;
13
14 void dfs(int x, int fr) {
15     dfn[x] = ++t;
16     for (const int y : e[x])
17         if (y != fr)
18             dfs(y, x);
19     end[x] = t;
20 }
21
22 void input();
23 void add(int x, int w) {
24     bit.add(dfn[x], w);
25 }
26 ll query(int x) {
27     return bit.ask(end[x]) - bit.ask(dfn[x] -
28     1);
29 }
30 int main() {
31     input();
32     dfs(root, 0);
33     bit.init(n);
34     for (int x = 1; x <= n; ++x)
35         bit.add(dfn[x], a[x]);
36
37     for (int i = 0, op, x, w; i < m; ++i) {
38         scanf("%d%d", &op, &x);
39         if (op == 1) {
40             scanf("%d", &w);
41             add(x, w);
42         } else {
43             printf("%lld\n", query(x));
44         }
45     }
46
47     return 0;
48 }
```

例题 2

n 个节点的树, m 次操作, 子树加, 单点求值。

dfs 序转化为序列上区间加, 单点求和。

例题 3 - LibreOJ 145. DFS 序 2

n 个节点的树， m 次操作，子树加，子树求和。

dfs 序转化为序列上区间加，区间求和。

```
1  #include <bits/stdc++.h>
2  using ll = long long;
3
4  constexpr int N = 1e6 + 10;
5  int n, m, root, a[N], c[N], dfn[N], t,
    ↪ end[N], seq[N];
6  std::vector<int> e[N];
7
8  struct segment_tree {
9      void build(int a[], int n);
10     void add(int l, int r, ll delta);
11     ll ask(int l, int r);
12 };
13 segment_tree segtree;
14
15 void dfs(int x, int fr) {
16     dfn[x] = ++t;
17     seq[t] = x;
18     for (const int y : e[x])
19         if (y != fr)
20             dfs(y, x);
21     end[x] = t;
22 }
23
24 void input();
25 void add(int x, int w) {
26     segtree.add(dfn[x], end[x], w);
27 }
28 ll query(int x) {
29     return segtree.ask(dfn[x], end[x]);
30 }
31
32 int main() {
33     input();
34     dfs(root, 0);
35     for (int i = 1; i <= n; ++i)
36         c[i] = a[seq[i]];
37     segtree.build(c, n);
38
39     for (int i = 0, op, x, w; i < m; ++i) {
40         scanf("%d%d", &op, &x);
41         if (op == 1) {
42             scanf("%d", &w);
43             add(x, w);
44         } else {
45             printf("%lld\n", query(x));
46         }
47     }
48
49     return 0;
50 }
```

例题 4

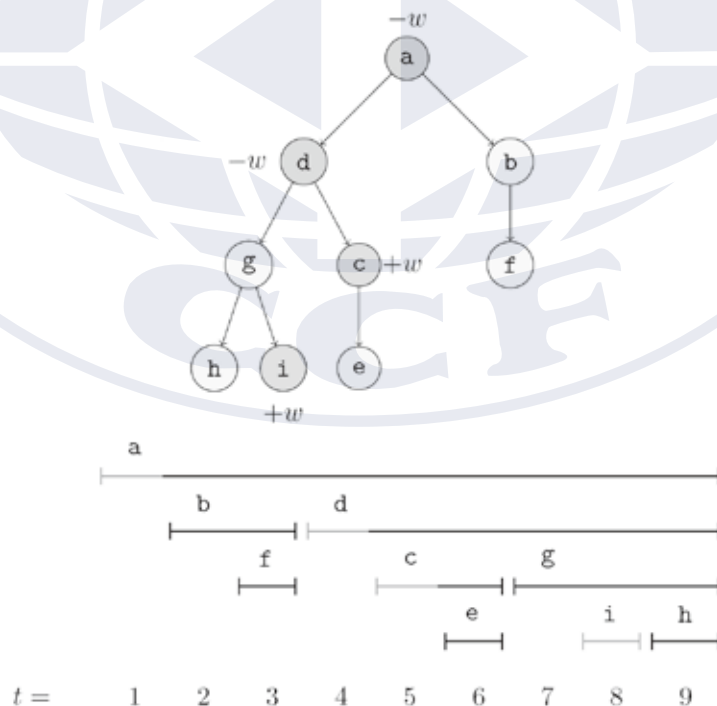
n 个节点的树， m 次操作，路径加，单点求值。

这是树上差分的经典应用。设这条路径两端点为 x, y ，设其 lca 为 z 。我们发现，对于 z 子树中（不含 z ）的任意节点 u ，只要 x 在 u 的子树中，则这条路径必然经过 x ， u 的权值加上了 w ；同理，只要 u 包含了 y ，权值也加上 w ；而 z 同时包含了 x, y ，权值也加上 $1 \times w$ ； z 的所有祖先同时包含了 x, y ，但权值不受到影响。

因此，我们可以将修改转化为：若 x 在 u 的子树中，则 u 的权值加上 w ；若 y 在 u 的子树中，则 u 的权值加上 w 。特别地，若 $u = z$ ，权值只加上 $1 \times w$ ；若 u 是 z 的祖先，则权值加上 0。

我们记录 $d[v]$ 表示 v 的所有祖先（包含 v ）的权值应当加上的权值。节点 u 在某时刻的实际权值是 $\sum_{v \text{ in subtree of } u} d[v]$ 。那么，对于路径 (x, y) (lca 为 z) 的修改，我们就将其转化为路径上两个端点 $d[x] \leftarrow d[x] + w, d[y] \leftarrow d[y] + w$ ； z 统计到子树中加了 $2 \times w$ ，而只能计算一次，因此令 $d[z] \leftarrow d[z] - w$ ；同理，再由 z 的所有祖先不受影响，得到 $d[\text{fa}[z]] \leftarrow d[\text{fa}[z]] - w$ 。

查询转化为区间求和。



对 (c, i) 加上权值 w 的例子。

例题 5

n 个节点的树， m 次操作，单点加，路径求和。

同上文类似。若在点 u 加上 w ，则从 u 的子树出发的路径 (x, y) 都会加上 w ，除非其 lca z 在 u 的子树内部（不含 u ）。

因此，设 $g[x]$ 表示 x 的所有祖先（含 x ）的权值之和。自然有 (x, y) 的权值和为 $g[x] + g[y] - g[z] - g[\text{fa}[z]]$ ；对 u 的权加上 w 时，修改所有 u 子树内节点 v ，令 $g[v] \leftarrow g[v] + w$ （区间加）。

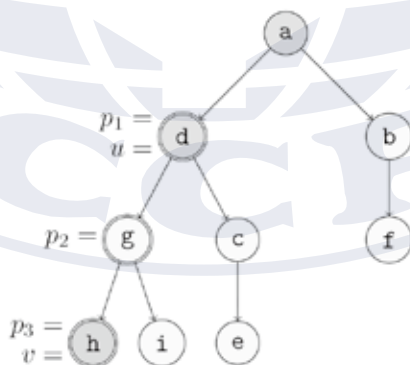
例题 6 - LibreOJ 146. DFS 序 3

n 个节点的树， m 次操作，路径加，子树求和。

修改和例题 4 相同，按照同样的方法转化就可以。只不过查询从对一个点求和变成了对该点子树内所有点求和，也就是从求 $\sum_{v \text{ in subtree of } u} d[v]$ 变成了求 $\sum_{p \text{ in subtree of } u} \sum_{v \text{ in subtree of } p} d[v]$ 。那从 v 到 u 的祖先—后代路径上，一共 $\text{dep}[u] - \text{dep}[v] + 1$ 个可能的 p 都加上了一次 $d[v]$ 。因此考虑 $d[v]$ 的贡献：

$$d[v] \times (\text{dep}[v] - \text{dep}[u] + 1) = d[v] \times (\text{dep}[v] + 1) - d[v] \times \text{dep}[u]$$

拆开的两个式子都有常数系数——对于同一个 v ， $\text{dep}[v]$ 始终不变。因此分别维护 $d[v]$ ， $d[v] \times \text{dep}[v]$ 的区间和即可。转化为序列上单点加，区间求和。



当 $u = d$ 时，考虑哪些不同的 p 会使用 $d[v = h]$ 。


```
1  #include <bits/stdc++.h>
2  using ll = long long;
3
4  constexpr int N = 1e6 + 10;
5  int n, m, root, a[N], fa[N], dep[N], dfn[N],
    ↳ t, end[N];
6  std::vector<int> e[N];
7
8  struct BIT {
9      void init(int n);
10     void add(int i, ll a);
11     ll ask(int i) const;
12     ll ask(int l, int r) const { return ask(r)
    ↳ - ask(l - 1); }
13 } bit, weighted_bit;
14
15 void dfs(int x, int fr) {
16     dfn[x] = ++t;
17     fa[x] = fr;
18     dep[x] = dep[fr] + 1;
19     for (const int y : e[x])
20         if (y != fr)
21             dfs(y, x);
22     end[x] = t;
23 }
24
25 void lca_init();
26 int lca(int x, int y);
27 void input();
28
29 void increase(int x, int b) {
30     bit.add(dfn[x], b);
31     weighted_bit.add(dfn[x], b * (dep[x] +
    ↳ 1));
32 }
33
34 void add(int x, int y, int w) {
35     int z = lca(x, y);
36     increase(x, w);
37     increase(y, w);
38     increase(z, -w);
39
40     if (z != root)
41         increase(fa[z], -w);
42 }
43
44 ll query_subtree(int x) {
45     ll res = 0;
46     res += weighted_bit.ask(dfn[x], end[x]);
47     res -= dep[x] * (bit.ask(dfn[x], end[x]));
48     return res;
49 }
50
51 ll query_single(int x) {
52     return bit.ask(dfn[x], end[x]);
53 }
54
55 int main() {
56     input();
57     dfs(root, 0);
58     lca_init();
59     bit.init(n);
60     weighted_bit.init(n);
61     for (int x = 1; x <= n; ++x) {
62         increase(x, a[x]);
63         if (x != root)
64             increase(fa[x], -a[x]);
65     }
66
67     for (int i = 0, op, x, y, w; i < m; ++i) {
68         scanf("%d%d", &op, &x);
69         if (op == 1) {
70             scanf("%d%d", &y, &w);
71             add(x, y, w);
72         } else if (op == 2) {
73             printf("%lld\n", query_single(x));
74         } else {
75             printf("%lld\n", query_subtree(x));
76         }
77     }
78     return 0;
79 }
```

例题 7 - LibreOJ 147. DFS 序 4

n 个节点的树， m 次操作，子树加，路径求和。

和例题 6 相似。查询和例题 5 一样转化；修改考虑以 u 为根的子树加 w 对子树内每个点 v 到根的链的贡献：

$$w \times (\text{dep}[v] - \text{dep}[u] + 1) = w \times (\text{dep}[v] + 1) - w \times \text{dep}[u]$$

对于单次查询， $\text{dep}[u] \times w$ 是常数； $\text{dep}[v]$ 对于单个 v 也固定。故容易转化为序列上区间加、单点求和。

```
1  #include <bits/stdc++.h>
2  using ll = long long;
3
4  namespace fast_IO {
5  int read();
6  void putchar(char c);
7  void write(ll x);
8  } // namespace fast_IO
9
10 constexpr int N = 1e6 + 10;
11 int n, m, root, a[N], fa[N], dep[N], dfn[N],
    ↪ t, end[N];
12 int head[N], ecnt;
13 struct edge {
14     int v, next;
15 } e[N << 1];
16 void add_edge(int x, int y) {
17     e[++ecnt] = {y, head[x]};
18     head[x] = ecnt;
19 }
20
21 struct BIT {
22     void init(int n);
23     void add(int i, ll delta);
24     void add(int l, int r, ll delta) {
25         add(l, delta);
26         add(r + 1, -delta);
27     }
28     ll ask(int i) const;
29 } bit, weighted_bit, single_bit;
30
31 void dfs(int x, int fr) {
32     dfn[x] = ++t;
33     fa[x] = fr;
34     dep[x] = dep[fr] + 1;
35     for (int i = head[x]; i; i = e[i].next)
36         if (e[i].v != fr)
37             dfs(e[i].v, x);
38     end[x] = t;
39 }
40
41 void input_with_fast_IO();
42 void init_lca();
43 int lca(int x, int y);
44
45 void add_single(int x, int w) {
46     single_bit.add(dfn[x], end[x], w);
47 }
48 void add_subtree(int x, int w) {
49     bit.add(dfn[x], end[x], w);
50     weighted_bit.add(dfn[x], end[x], w *
    ↪ (ll)dep[x]);
51 }
52 ll ask(int x) {
53     ll res = 0;
54     res += single_bit.ask(dfn[x]);
55     res += bit.ask(dfn[x]) * (dep[x] + 1ll);
56     res -= weighted_bit.ask(dfn[x]);
57     return res;
58 }
59 ll query(int x, int y) {
60     int z = lca(x, y);
61     ll res = ask(x) + ask(y) - ask(z);
62     if (z != root)
63         res -= ask(fa[z]);
64     return res;
65 }
66
67 int main() {
68     input_with_fast_IO();
69     dfs(root, 0);
70     init_lca();
```

```
71 bit.init(n);
72 single_bit.init(n);
73 weighted_bit.init(n);
74 for (int x = 1; x <= n; ++x)
75     add_single(x, a[x]);
76
77 for (int i = 0, op, x, y, w; i < m; ++i) {
78     op = fast_IO::read();
79     x = fast_IO::read();
80     if (op == 1) {
81         w = fast_IO::read();
82         add_single(x, w);
83     } else if (op == 2) {
84         w = fast_IO::read();
85         add_subtree(x, w);
86     } else {
87         y = fast_IO::read();
88         fast_IO::write(query(x, y));
89         fast_IO::putchar('\n');
90     }
91 }
92
93 return 0;
94 }
```

思考

下面这三个问题仅靠 dfs 序可做吗？

- 单点加，子树加，路径加，单点求值，子树求和
- 单点加，子树加，单点求值，子树求和，路径求和
- 单点加，子树加，路径加，单点求值，子树求和，路径求和

前两个问题实际上是上面某些例题的结合体，显然能做。dfs 序的强大之处在于将每棵子树对应一个区间，所以单点加，单点求值，子树加，子树求和都能做。

但是路径并不能在 dfs 序上表示成一个区间，所以只有路径加或者只有路径求和的情况可以通过差分等方法转化成一个区间。

但是如果同时出现路径加和路径求和，仅靠 dfs 序确实无能为力。所以第三个问题可以考虑重链剖分求解。

变体

求取 LCA

$t \leftarrow 0$

procedure DFS_LCA(x)

$t \leftarrow t + 1$

$seq[t] \leftarrow x$

$dfn[x] \leftarrow t$

for all $y \in x$'s sons do

DFS_LCA(y)

$t \leftarrow t + 1$

$seq[t] \leftarrow x$



一种可能的变体序列: a, b, f, b, a, d, c, d, e, d, g, d, a

x	a	b	f	d	c	e	g
$dfn[x]$	1	2	3	6	7	9	11

我们不仅在进入节点 x 时向序列里添加元素, 还在从它的某个儿子返回时也添加 x 。设 $dfn[x]$ 是 x 第一次加入序列时 t 的大小。

这样得到的长为 $2n - 1$ 的序列满足一个性质: 若两个不同节点 x, y 满足 $dfn[x] < dfn[y]$, 则序列中 $seq[dfn[x], \dots, dfn[y]]$ 深度最小的节点 z 就是 x, y 的最近公共祖先。

考虑 x 和 y 在 z 的两棵不同子树中, 从 z 先遍历的子树返回时序列中加入了 z , 因此序列的这一部分必有至少一个 z ; 而在此过程中没有返回到 z 的父亲, 所以 z 就是这其中深度最小的节点; z 是 x 的情况类似。

因此, 我们在序列上同时记录节点深度, 就把 lca 转化为序列上的区间最小值问题。

欧拉序结合树上莫队等

参见例题 P4074 [WC2013] 糖果公园。

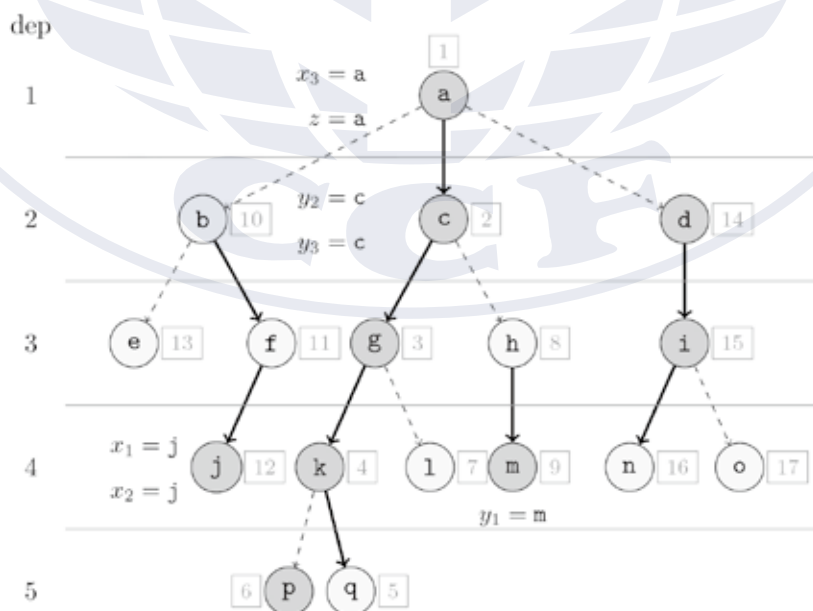
重链剖分

在 dfs 序上, 对于 $i = 1, 2, \dots, n-1$, 如果 $x = \text{seq}[i]$ 和 $y = \text{seq}[i+1]$ 是父亲—儿子关系 (y 是 x 第一个访问的儿子), 则我们可将 (x, y) 并到一条链上。这样, 这棵树就被剖分成若干条链。观察发现, 对于同一棵树, dfs 序和剖分方式是一一对应的。

我们称 y 是 x 的重儿子, 也即在 dfs 的过程中第一个被访问的儿子, 记作 $\text{hson}[x] = y$; 其他的儿子, 称为轻儿子。像 (x, y) 这样由 x 和 x 的重儿子 (x 是叶子节点除外) 构成的边, 称为重边; 其他的边 (由 x 和所有轻儿子构成的边), 称为轻边。我们将由重边构成的链称为重链。某个点 x 一定属于一条唯一的重链: 根据它和父亲之间的关系, 要不然与父亲同属一条重链, 要不然是一条重链的链顶。也就是说, 链顶或者是根节点, 或者是某个节点的一个轻儿子。将 x 所属重链的链顶记作 $\text{top}[x]$ 。非叶子节点一定有至少一个儿子, 也就一定有重儿子, 因此重链的终点一定是某个叶子。

树上差分可以通过转换差分和前缀和的方式解决上文的问题。对于更加一般的、在序列上有解决方案的数据结构问题, 如果将它扩展到树上两点间的路径 (如路径加、路径求和), 我们可以根据某种剖分方式, 将路径剖分成若干部分, 将其合并后解决。

具体地, 设路径 (x, y) 的 lca 为 z 。记 $x_1 = x, x_i = \text{fa}[\text{top}[x_{i-1}]]$ 。则我们将从 x 到 z 的路径分成 $(x_1, \text{top}[x_1]), (x_2, \text{top}[x_2]), \dots$ 直到 x_{k_1} 和 z 同属一条重链, 最后一段就是 (x_{k_1}, z) 。两段路径之间 $(\text{top}[x_i], x_{i+1})$, 必然是一条轻边。对于 z 到 y 的部分 (不包含 z) 同理: 剖成 $(z, y_{k_2}), \dots, (\text{top}[y_1], y_1 = y)$ 。那么, 只要我们维护的信息具有结合律, 就可以用 $k = k_1 + k_2$ 次查询/修改在树上维护信息。



一棵树的重链剖分。

路径 (p, i) 被划分成 3 个部分。

求 m, j 的 lca 需要比较和跳跃 2 次。

前文提到, dfs 序具有不唯一性。而对于一种任意的 dfs 序, 这样的剖分不具有特殊性质。

一条路径可能被剖分成任意数量的段， k 在最坏情况下是 $O(n)$ 的。有没有办法使得对于任意一条路径， k 都在“可接受的范围”内？

答案是肯定的。我们记 $\text{siz}[x]$ 表示 x 的子树大小。我们令 $\text{hson}[x]$ 是 x 的所有儿子 y_i 中 $\text{siz}[y_i]$ 最大的一个（若有并列，则任取一个）。在 dfs 求得 dfs 序——也就是剖分方案时，我们规定必须先走已经求好的 $\text{hson}[x]$ ，然后再遍历其他轻儿子。

不难发现，当 $\text{siz}[\text{hson}[x]] \geq \text{siz}[x]/2$ 时，对于任意一个轻儿子 y ，都有 $\text{siz}[y] \leq \text{siz}[x] - \text{siz}[\text{hson}[x]] \leq \text{siz}[x]/2$ ；当 $\text{siz}[\text{hson}[x]] < \text{siz}[x]/2$ 时，也有 $\text{siz}[y] \leq \text{siz}[\text{hson}[x]] < \text{siz}[x]/2$ 。因此，只要我们在上述剖分方式中增多一段路径（从 $\text{top}[x_i]$ 跳到 x_{i+1} ），必然有 $\text{top}[x_i] \leq \text{siz}[x_{i+1}]/2$ ，子树大小至少翻倍；而整棵树的大小为 n ，因此至多在轻边上切换 $\log_2 n$ 次。这样， k 最坏也是 $O(\log n)$ 的。它的最坏情况在一棵完全二叉树的两个叶子节点作为路径端点时出现： $k = k_1 + k_2 \leq \log_2 n \times 2$ 。

procedure DFS(x)

```

siz[x] ← 1
dep[x] ← dep[fa[x]] + 1
hson[x] ← null
for all  $y \in x$ 's sons do
    DFS( $y$ )
siz[x] ← siz[y] + siz[x]
if  $\text{hson}[x] = \text{null} \vee \text{siz}[y] > \text{siz}[\text{hson}[x]]$  then
    hson[x] ←  $y$ 

```

$t \leftarrow 0$

procedure DECOMPOSE(x, u)

```

 $t \leftarrow t + 1$ 
dfn[x] ←  $t$ 
seq[t] ←  $x$ 
top[x] ←  $u$ 
if  $\text{hson}[x] \neq \text{null}$  then
    DECOMPOSE(hson[x],  $u$ )
for all  $y \in x$ 's sons do
    if  $y \neq \text{hson}[x]$  then
        DECOMPOSE( $y, y$ )
end[x] ←  $t$ 

```

例题

求取 LCA

已知两节点 x, y ，求其最近公共祖先 z 。

假设 x, y 此时不在同一条链上。则若 x 所在链的链顶 $\text{top}[x]$ ，其深度大于等于 $\text{top}[y]$ 的，我们就可以断言“ z 不在 x 所在链上”。否则， y 现在所在链向上跳若干次，必然可以到达一条链，这条链以 z 的某个轻儿子作为链顶。但其深度大于 z ，与假设相悖。因此我们令 $x \leftarrow \text{fa}[\text{top}[x]]$ ；反之亦然。如此往复，最后得到 x, y 在同一条链上，取深度较小者就是 z 。

复杂度 $O(\log n)$ 。

procedure LCA(x, y)

```

while  $\text{top}[x] \neq \text{top}[y]$  do
    if  $\text{dep}[\text{top}[x]] \geq \text{dep}[\text{top}[y]]$  then
         $x \leftarrow \text{fa}[\text{top}[x]]$ 
    else
         $y \leftarrow \text{fa}[\text{top}[y]]$ 
if  $\text{dep}[x] < \text{dep}[y]$  then
    return  $x$ 
else
    return  $y$ 

```

路径/子树修改，路径/子树查询 - P3384 【模板】重链剖分/树链剖分

按照上文介绍的方式，剖分后在 $O(\log n)$ 个段上区间加、区间求和即可。

```
1  #include <bits/stdc++.h>
2  using ll = long long;
3
4  constexpr int N = 1e5 + 10;
5  int n, m, root, mod, a[N], b[N];
6  int siz[N], fa[N], dep[N], hson[N];
7  int t, dfn[N], end[N], seq[N], top[N];
8  std::vector<int> e[N];
9
10 struct segment_tree {
11     void build(int a[], int n);
12     void add(int l, int r, int delta);
13     int query(int l, int r);
14 } segtree;
15 void input();
16
17 void dfs(int x, int fr) {
18     fa[x] = fr;
19     siz[x] = 1;
20     dep[x] = dep[fr] + 1;
21     hson[x] = 0;
22     for (const int y : e[x]) {
23         if (y == fr)
24             continue;
25         dfs(y, x);
26         siz[x] += siz[y];
27         if (siz[hson[x]] < siz[y])
28             hson[x] = y;
29     }
30 }
31
32 void decompose(int x, int u) {
33     dfn[x] = ++t;
34     seq[t] = x;
35     top[x] = u;
36     if (hson[x] != 0)
37         decompose(hson[x], u);
38     for (const int y : e[x])
39         if (y != fa[x] && y != hson[x])
40             decompose(y, y);
41     end[x] = t;
42 }
43
44 void add_path(int x, int y, int w) {
45     while (top[x] != top[y]) {
46         if (dep[top[x]] < dep[top[y]])
47             std::swap(x, y);
48         segtree.add(dfn[top[x]], dfn[x], w);
49         x = fa[top[x]];
50     }
51     if (dep[x] > dep[y])
52         std::swap(x, y);
53     segtree.add(dfn[x], dfn[y], w);
54 }
55
56 void add_subtree(int x, int w) {
57     segtree.add(dfn[x], end[x], w);
58 }
59
60 int query_path(int x, int y) {
61     int sum = 0;
62     while (top[x] != top[y]) {
63         if (dep[top[x]] < dep[top[y]])
64             std::swap(x, y);
65         (sum += segtree.query(dfn[top[x]],
66                               dfn[x])) %= mod;
67         x = fa[top[x]];
68     }
69     if (dep[x] > dep[y])
70         std::swap(x, y);
71     (sum += segtree.query(dfn[x], dfn[y])) %=
72         mod;
73     return sum;
74 }
75
76 int query_subtree(int x) {
77     return segtree.query(dfn[x], end[x]);
78 }
79
80 int main() {
81     input();
82     dfs(root, 0);
83     decompose(root, root);
84     for (int i = 1; i <= n; ++i)
85         b[i] = a[seq[i]];
86     segtree.build(b, n);
87
88     for (int i = 0, op, x, y, w; i < m; ++i) {
89         scanf("%d%d", &op, &x);
90         if (op == 1) {
91             scanf("%d%d", &y, &w);
92             add_path(x, y, w);
93         } else if (op == 2) {
94             add_subtree(x, w);
95         }
96     }
97 }
```



```
87     scanf("%d", &y);
88     printf("%d\n", query_path(x, y));
89 } else if (op == 3) {
90     scanf("%d", &w);
91     add_subtree(x, w);
92 } else {
93     printf("%d\n", query_subtree(x));
94 }
95 }
96
97 return 0;
98 }
```



应用 - DFS 序

CF383C Propagating tree

题目大意

给定一棵以 1 为根的 n 个点的有根树，点带权 a_i ，你需要完成这两种操作 m 次：

- 1 u 将 u 权值增加 v ， u 的儿子权值增加 $-v$ ， u 的儿子的儿子权值增加 $-(-v) = v \cdots$ 依次类推。形式化的，设子树中的点 y 到 x 的距离为 d ， $a_y \rightarrow a_y + v \times (-1)^d$ 。
- 2 u 查询点 u 的权值。

$n, m \leq 2 \times 10^5$ 。

解法

显然，每次修改操作对奇数与偶数层内部的改变是相同的，我们可以将他们分开维护。具体方法很简单，求 dfs 序时，将奇数和偶数层的点的编号分别记录在两个序列 A, B ，并分别记录每个点的子树在 A, B 中的区间。

还有一种做法，考虑对于奇数层结点维护其增加值，偶数层结点维护其减少值。若修改的点 u 的深度为奇数，则奇数层的增加值与偶数层的减少值都是 v ，可以直接在原 dfs 序上做区间加单点查，在查询时根据层数答案判断该加上还是减去查询结果。两种做法时间复杂度都是 $O(n \log n)$ 。

CF877E Danil and a Part-time Job

题目大意

给定一棵 n 个点的有根树，根为 1，点有点权 $a_i \in \{0, 1\}$ ，进行下面 m 次操作。

- get 查询点 x 的子树中有多少点权为 1。
- pow 将点 x 的子树中所有点的点权取反，即 $a_i \rightarrow a_i \text{ xor } 1$ 。

$n, m \leq 2 \times 10^5$ 。

解法

记点 x 的 dfs 序为 dfn_x ，子树大小为 siz_x 。

观察题目可以发现，本题只涉及子树操作、查询，可以利用 dfs 序将子树问题转化为序列问题，即操作、查询子树 x 可以转化成操作、查询 dfs 序上的区间 $[dfn_x, dfn_x + siz_x - 1]$ 。考虑序列问题如何解决。维护取反操作可以使用带懒标记的线段树，在线段树每个结点 p 上记录区间长度 len_p ，区间和 sum_p ，标记 $tag_p \in \{0, 1\}$ ， tag_p 为 1 表示该区间需要进行一次反转。每次 *pushdown* 下传标记时，只需将其子节点 q 的 $sum_q \rightarrow len_q - sum_q$ ， $tag_q \rightarrow tag_q \text{ xor } 1$ 即可。

时间复杂度 $O(n \log n)$

P2982 [USACO10FEB] Slowing down G

题目大意

有一棵 n 个点的树，编号为 i 的点有一只编号为 i 的奶牛。现在奶牛将按照由 1 到 n 的顺序进行以下行动：对于编号为 i 的奶牛，它将从点 i 走到点 P_i ，并离开牧场。求出每只奶牛行动中会经过几只其它奶牛。 $n \leq 10^5$ 。

解法

题目简化为：每次求出 $i \rightarrow P_i$ 上的奶牛个数，并删除 i 上的奶牛。也就是要支持单点删路径查。显然可以用树剖在 $O(n \log^2 n)$ 时间复杂度内解决，考虑更优秀的做法。

设 $l = lca(i, p_i)$ ，每一条路径可以拆解成 $i \rightarrow l, l \rightarrow p_i$ 两条祖先后代链。定义 c_x 为 x 到根的奶牛个数，则单点加操作会对 x 子树中的所有 y 的 c_y 产生影响，使 c_y 减 1，用 dfs 序转化为序列问题即可。这是树上差分思想的又一次经典应用。

P3178 [HAOI2015] 树上操作

题目大意

有一棵点数为 N 的树，以点 1 为根，且树有点权。然后有 M 个操作，分为三种：

- 操作 1：把某个节点 x 的点权增加 a 。
- 操作 2：把某个节点 x 为根的子树中所有点的点权都增加 a 。
- 操作 3：询问某个节点 x 到根的路径中所有点的点权和。

$n, m \leq 10^5$ 。

解法

题目显然可以用树链剖分在 $O(n \log^2 n)$ 时间复杂度内解决，考虑有没有更优秀的做法。

定义 c_x 为 x 到根的路径中所有点的点权和。考虑将操作 1, 2 对答案的贡献分开计算最后加起来。

对于操作 1，一次操作会对 x 子树中所有点 y 的 c_y 产生影响，即 $c_y \rightarrow c_y + a$ ，可以用 dfs 序转化为区间加单点查。

对于操作 2，一次操作对 x 子树中所有点 y 的 c_y 产生的影响不一样，取决于 y 到 x 的点数，也就是与 y 的深度 dep_y 相关，即 $c_y \rightarrow c_y + (dep_y - dep_x + 1) \times a$ 。

分离系数，增加值就是 $a \times (dep_y + 1) - dep_x \times a$ ，同样用 dfs 序转化为分别维护两个支持区间加单点查的数据结构，其中第一个数据结构的值在查询时整体乘上 $dep_y + 1$ 即可。

时间复杂度 $O(n \log n)$ 。

CF1132G Greedy Subsequences

题目大意

定义一个序列的最长贪心严格上升子序列为：若选出的子序列为 a ，对于其中相邻两项 i, j ，不存在 $i < k < j$ ，满足在原序列 b 中，有 $b_i < b_k$ 。换句话说就是选择一个元素后必须选择它之后第一个大于它的元素。

给定一个长度为 n 的序列，同时给定一个常数 k ，求该序列的所有长度为 k 的子区间的最长贪心严格上升子序列的长度。

$$1 \leq k \leq n \leq 10^6, 1 \leq a_i \leq n。$$

解法

乍一看，这个东西似乎与树没有什么关系。但考虑到不论怎么选 l, r 点 x 的后继都是确定的，也就是它后面第一个比它大的数的位置。所以我们定义 p_i 表示 i 后面第一个比 a_i 大的数的位置。若不存在则 $p_i = n + 1$ 。

我们把 i 向 p_i 连一条边。显然这会形成一棵树。考虑使用扫描线的思想，把 r 依次向右挪动。我们定义 $f(i)$ 表示在限制内，以 i 为开头的最长贪心严格上升子序列的长度。 l 挪动到 $l + 1$ 相当于 $f(l)$ 不能做出任何贡献，故直接将 $f(l)$ 设为负无穷即可。

将 r 挪到 $r + 1$ ，相当于 r 子树中所有点的 $f(x)$ 都将加一。由于子树的 dfs 序是一个区间，故直接使用你喜欢的数据结构维护区间加，单点改，区间最大值即可。

时间复杂度 $O(n \log n)$ 。

CF1467E Distinctive Roots in a Tree

题目大意

给定一个大小为 n 的树 ($1 \leq n \leq 2 \times 10^5$)，每个点有一个权值 a_i ($1 \leq a_i \leq 10^9$)，求有多少个点 u 满足从 u 出发的任意一条路径上的点的权值不重复。

解法

第一种做法：考虑点 u 能计入答案得充要条件：以 u 为根时，不存在一对颜色相同的祖先后代关系。

所以很自然能想到换根。首先先求出初始的祖先后代关系的个数。我们考虑换一次根会怎么样。发现贡献的变化值只有两种：

- 在未换根时，新根的子树与老根形成的点对。
- 在已经把根换到新根时，老根的子树与新根形成的点对。

由于子树的 dfs 序是连续的，故问题转化为查询区间 x 的出现次数。离散化之后主席树一下就好了。时间复杂度 $O(n \log n)$ 。当然你也可以选择不写主席树，开个桶，在桶里面二分子树 dfs 序的左端点和右端点。

第二种做法：考虑每一对同色点将会使多少点 u 不能计入答案。

我们分为两种情况考虑。第一种情况，若除去 u 的子树之外，还存在点权 a_u ，则说明 u 的子树中所有点都不能作为答案，标记一下就好了。

第二种情况，若 u 的子树中存在 2 个及以上的 a_u 则说明 u 及 u 子树外的所有点都不能作为答案，标记一下就好了。由于我们的标记都是与子树相关的，又因为子树的 dfs 序是连续的一段区间，故直接差分标记即可。时间复杂度 $O(n \log n)$ ，瓶颈在于离散化。

CF1528C Trees of Tranquillity

题目大意

给你两棵 n 个节点的树，根节点都为 1，要你找一个最大的点集 S ，使得对于任意 $u, v \in S$ ，都满足 u, v 在第一棵树上一个是另一个的祖先并且在第二棵树上互相都没有祖先关系，输出 S 的大小即可。

解法

首先观察一下两个限制，将两个限制翻译一下。第一个限制就是你选的点必须在一个链上。第二个限制乍一看不是很好维护，但如果我们用 dfs 序来刻画，就会发现，两个点子树的 dfs 序是连续的一段，且要么包含，要么不交。故两个点没有祖先后代关系，就是说他们子树的 dfs 序形成的连续段不交。

先不考虑第一个限制，那么第二个限制是个经典的贪心：每次加入一个连续段，若没有相交，就直接加入。若已经加入了这个连续段的子集，那么我们一定不会加入这个连续段。如果加入了这个连续段的超集，我们直接将超集的那一段给删除，加入这一段。证明不难。

那么考虑上第一个限制，我们在第一棵树上 dfs，我们现在需要支持加入一个连续段，删除一个连续段，查询最多能选多少个连续段使得它们不交。直接用 set 维护上述贪心即可。时间复杂度 $O(n \log n)$ 。

P3459 MEG-Megalopolis

题目大意

给你一棵初始时边权全为 1 的树。你需要支持以下两个操作：

- 将某条边的边权设为 0。
- 查询 1 到某点的路径上的边权和。

解法

显然，有一个 $O(n \log^2 n)$ 的经典做法：树链剖分。这里不多赘述。

我们定义 $f[u]$ 表示 u 到 1 的路径上的边权和。初始时，若 $\text{dep}[1] = 0$ 则 $f[u] = \text{dep}[u]$ 。考虑把 u 到 v 这条边的权值改为 0。由于 u, v 顺序不重要，我们钦定 u 是深度大的点。不难发现，一次修改只会对 u 子树内的点 x 的 $f[x]$ 产生影响。由于一棵子树的 dfs 序是连续的，故直接用你喜欢的数据结构，维护区间加，单点查即可。时间复杂度为 $O(n \log n)$ 。

P3833 魔法树

题目大意

给你一棵有根树，你需要支持以下两种操作：

- 给 u 到 v 的路径上的点的点权都加上 x 。
- 查询 u 子树中点的权值之和。

解法

显然，有一个 $O(n \log^2 n)$ 的经典做法：树链剖分。这里不多赘述。

首先 u 到 v 的路径可以拆为 1 到 u ，1 到 v ，1 到 u, v 的最近公共祖先这三条链。我们定义 $t[x]$ 表示 1 到 x 的链上的点权都加上 $t[x]$ 的权值。那么操作一就转化为修改 $t[u], t[v], t[lca(u, v)]$ 的值。

考虑怎么查询答案。对于 u 子树中的一点 v ， $t[v]$ 能做出的贡献就是 v 到 u 的路径上的点数，即为 $\text{dep}[v] - \text{dep}[u] + 1$ 。所以查询的答案就是 $\sum (\text{dep}[v] - \text{dep}[u] + 1) \times t[v]$ 。其中 v 在 u 的子树内。

拆开式子得到两个部分 $\sum \text{dep}[v] \times t[v]$ ，以及 $(1 - \text{dep}[u]) \times \sum t[v]$ 。由于在一个子树内的点的 dfs 序是连续的，所以只需要使用两个你喜欢的数据结构分别维护 $\text{dep}[v] \times t[v]$ 的区间和以及 $t[v]$ 的区间和即可。

时间复杂度为 $O(n \log n)$ 。

P4175 [CTSC2008] 网络管理

题目大意

给出一棵 n 个点带边权的树， q 次操作，每次操作可能是：

- 修改一条边的边权。
- 查询树上从 u 到 v 的路径上第 k 大的权值是多少。

$n, q \leq 8 \times 10^4$ 。

解法

对于树是一条链的情况，其实就是带修改操作的区间第 k 大问题，可以使用整体二分算法，用树状数组实现单点加和区间查询，在 $O(q \log^2 n)$ 的时间内解决问题。

对于本题中的树上问题，需要单点修改、路径查询，可以使用树链剖分算法，但复杂度为 $O(q \log^3 n)$ 。

注意到修改是单点权值加，查询时是路径权值和，信息具有可减性，所以使用树上差分算法，可以转化为子树加、单点查询，接着使用 dfs 序，将子树转化为序列问题中的区间，即区间加、单点查询，可以用树状数组做到 $O(q \log^2 n)$ 的复杂度。

P3242 [HNOI2015] 接水果

题目大意

给出一棵 n 个点的树和树上的 p 条路径，每条路径从 a_i 到 b_i ，权值是 c_i 。 q 次询问，每次给出一条路径 u_i 到 v_i ，查询在 p 条路径中是 (u, v) 的子路径的所有路径的权值第 k 小的权值。 $1 \leq n, p, q \leq 4 \times 10^4$ 。

解法

考虑怎么判断一条路径是否被另一条路径包含。先求出这棵树的 dfs 序，设 st_u 是节点 u 的 dfs 序， ed_u 是 u 子树中最大的 dfs 序。这样做的好处是可以把一条路径 (x, y) 对应到平面直角坐标系中 (st_x, st_y) 这个点，而子树关系在 dfs 序上是一个区间信息。

考虑包含路径 (x, y) 的路径 (u, v) dfs 序的特点，不妨设 $st_x < st_y$ ， $st_u < st_v$ ，分类讨论：

- x 是 y 的祖先。那么要求 v 在 y 的子树内， u 不在 x 的子树中 y 所在的那一棵内，可以用 dfs 序描述为平面上的两个矩形。
- x 不是 y 的祖先。那么就是要求 u 在 x 的子树内， v 在 y 的子树内，对应的是平面上的一个矩形范围。

这样我们就把路径的包含关系转化为了点是否在矩形内部的关系，仍然使用整体二分算法，需要矩形加单点查询，可以使用差分 + 扫描线解决问题，时间复杂度 $O(q \log^2 n)$ 。

P3320 [SDOI2015] 寻宝游戏

题目大意

一棵 n 个节点的树，边有边权。每个点可能是关键点，每次操作会改变一个点是否是关键点。求所有关键点形成的极小连通块的边权和的两倍。

解法

求出每个点的 dfs 序，对于包含一个点集的极小连通块，考虑将点集中的点按照 dfs 序排序后得到一个序列，将序列首尾相连得到一个环，那么这个连通块的边权和就是环上相邻两个点的距离求和。

证明可以考虑 dfs 这个连通块的过程，以第一次进入关键点的时间为分界点，整个 dfs 路径被分为点集大小 + 1 段，并且每段都是简单路径（如果不是简单路径，那么这段路径上重复部分的点可以从连通块中删去而不影响合法性）。最初的一段和最末的一段长度和又恰好等于 dfs 序最小的关键点和 dfs 序最大的关键点之间的距离。

对于修改操作，考虑使用 `std::set` 维护排好序后的关键点序列，修改时查询前驱和后继更新答案，时间复杂度 $O(q \log n)$ 。

P2414 [NOI2011] 阿狸的打字机

题目大意

一个初始为空的字符串，接下来有 n 次操作，每次操作可能是：

- 在字符串末尾添加一个字符
- 在字符串末尾删除一个字符
- 将当前的字符串作为一个新版本。

q 次询问，第 i 次询问查询第 x 个版本的字符串在第 y 个版本里出现了多少次。 $n, q \leq 10^5$ 。

解法

题目相当于是在 Trie 上进行 dfs 操作，所以先建出 Trie；由于要查询出现次数，所以对 Trie 建立 AC 自动机。那么相当于查询 Trie 上 y 串所在的路径上有多少个点是 fail 树上 x 串的末尾节点的后代。

将询问离线下来，考虑在 Trie 上 dfs，进入一个点时加入这个点的贡献，退出一个点时撤销这个贡献，那么这样当 dfs 到某个点 u 的时候，保存的就是根到 u 路径上的信息。那么就相当于要支持 fail 树上的单点加和子树求和，使用 dfs 序转化为序列问题上单点加区间求和。用树状数组进行维护，时间复杂度 $O(n \log n)$ 。

P4074 [WC2013] 糖果公园

题目大意

给定一棵树，点有颜色，每次询问一条路径 (x, y) ，求路径上的

$$\sum_c val_c \sum_{i=1}^{cnt_c} w_i$$

其中 val_c 表示颜色 c 的价值， cnt_c 表示颜色 c 的出现次数， w_i 表示颜色出现 i 次后的价值。有单点修改颜色的操作。询问和修改的总次数以及树的大小 $\leq 10^5$ 。

解法

我们将树变成序列：这是 dfs 序列的一个变体，我们在进入和退出节点 x 时都将 x 加入序列中。这样，序列长度是 $2n$ 。仍然记录 $dfn[x]$ 是 x 第一次被加入序列的时间。

这样有什么好处呢？如果有两节点 x, y 满足 $dfn[x] < dfn[y]$ ，那么在 $seq[dfn[x], \dots, dfn[y]]$ 中，除 x, y 的最近公共祖先 z 以外，在 x, y 路径上的点都恰好出现 1 次（从 x 到 z 是“退出”时加入的点；从 z 到 y 是“进入”时加入的点）不在路径上的点，出现了 0 或 2 次（没遍历到，或者“进入”又“退出”）。

如果我们维护好了一条已知路径的答案（包括各个点、各个颜色的出现次数），往这条路径中加入/删除一个点，新的答案是容易计算的——变化量只和该点颜色的新的出现次数有关。那



么，我们令“出现了偶数次的点”贡献清零，直接在 seq 上做带修改莫队就可以求解本题。（可能需要补上 z 的贡献）

时间复杂度 $O(n^{5/3})$ 。



应用 - 树链剖分

P1505 [国家集训队] 旅游

题目大意

给定一棵 n 个节点的树，边带权，编号 $0 \sim n-1$ ，需要支持五种操作：

- 将输入的第 i 条边权值改为 w ；
- 将 u, v 节点之间的边权都变为相反数；
- 询问 u, v 节点之间边权和；
- 询问 u, v 节点之间边权最大值；
- 询问 u, v 节点之间边权最小值。

保证任意时刻所有边的权值都在 $[-1000, 1000]$ 内， $1 \leq n, m \leq 2 \times 10^5$ 。

解法

需要维护树上路径的信息，可以使用树链剖分把树上问题转化为序列上的问题。通过树链剖分，我们把一条路径划分为了 $O(\log n)$ 个区间，就可以使用线段树维护区间和，最大值，最小值等信息，这道题还有一个区间取反的操作，需要打标记。下放标记时交换最大值和最小值并取反，对区间和取相反数即可。复杂度 $O(n \log^2 n)$ 。

P3038 [USACO11DEC] Grass Planting G

题目大意

给出一棵有 n 个节点的树，有 m 个如下所示的操作：

- 将两个节点之间的路径上的边的权值均加一。
- 查询两个节点之间的那一条边的权值，保证两个节点直接相连。

初始边权均为 0。 $2 \leq n \leq 10^5, 1 \leq m \leq 10^5$ 。

解法

方法一：这道题可以同上题一样用树链剖分 + 线段树维护，查询的时候记录每个点与父亲的连边的边的编号就可以在线段树上直接查询，复杂度 $O(n \log^2 n)$ 。

方法二：还有一种更加优秀的做法，也就是树上差分，对于每个点维护它的值减去他所有儿子的值，修改就只需要将 u, v 的值加 1，将 LCA 的值减去 2 即可。每个点的答案是子树的和，通过树链剖分将一个点的子树划分成了一个区间，那么就只需要维护单点改，区间和。用线段树或树状数组可以轻易做到 $O(n \log n)$ 。

CF1254D Tree Queries

题目大意

给定一棵 N 个节点的树，有 Q 次操作：

- 1 $v\ d$ 给定一个点 v 和一个权值 d ，等概率地选择一个点 r ，对每一个点 u ，若 v 在 u 到 r 的路径上，则 u 的权值加上 d （权值一开始为 0）。
- 2 v 查询 v 的权值期望，对 998244353 取模。

$1 \leq N, Q \leq 150000$ 。

解法

考虑分析一次修改的本质是什么：

- 对于一次修改 u, d ，我们删去 u ，会出现若干联通块。
- 对于一个大小为 s 的联通块，每个点加上 $\frac{n-s}{n}$ 。

不妨让每个点加的值为 $n-s$ ，最后乘上 n 的逆元即可。

对于一个点，它的父亲所在的连通块是很好处理的。全局加上，再给 u 的子树减去就可以了。考虑儿子怎么处理，如果暴力遍历每一个儿子，那么复杂度将会变成 $O(n^2)$ ，考虑树链剖分的性质，一个点最多向上跳 $O(\log n)$ 条轻边，这是可以直接暴力处理的，我们只需要维护重儿子的贡献就行了，所以只需要修改点 u 的重儿子的子树的值，再给 u 打上标记，当 u 的轻儿子中的点查询时，正好会跳到这个点，直接加上贡献就可以了。复杂度 $O(n \log n)$ 。

CF1260F Colored Tree

题目大意

给定一棵树，每个节点有一个颜色 h ， h_i 为 $[L_i, R_i]$ 内的一个整数。

现在，对于所有 $\prod_i (R_i - L_i + 1)$ 种不同的染色方案，求出下列式子之和：

$$\sum_{h_i=h_j, 1 \leq i < j \leq n} dis(i, j)$$

$n \leq 10^5, 1 \leq L_i \leq R_i \leq 10^5$ ，答案对 $10^9 + 7$ 取模。

解法

考虑枚举每一个颜色，点 i 会在 L_i 时刻加入， $R_i + 1$ 时刻删除，颜色 h 的贡献为当前的所有点两两之间的距离，也就是 $\sum_u \sum_v dep_u + dep_v - dep_{lca(u,v)}$ 。 $dep_u + dep_v$ 的贡献是容易的，考虑如何求 $dep_{lca(u,v)}$ 。这要求出 LCA，并不太好做，考虑 dep_u 的另一种定义是 u 到根路径上所经过的点数。所以就可以把 $dep_{lca(u,v)}$ 写成有多少个点，满足 u, v 均在它的子树中。假设每个点子树中有 siz_u 个点，那么贡献就是 $\sum \frac{siz_u \times (siz_u - 1)}{2}$ 。我们考虑维护每个点的 siz ，加入一个点 u 就相当于把它到根节点的路径上所有的点的 siz 加一，同样可以用树剖 + 线段树维护。注意线段树不仅要维护区间和，还要维护区间平方和，复杂度 $O(n \log^2 n)$ 。

最短路算法及其应用

北京大学附属中学 肖然

前置知识

- 基础图论
 - 图的存储方法
 - 图的dfs/bfs
 - DAG拓扑排序
- 动态规划
- 基础贪心思想
- 时间复杂度分析
- 数据结构
 - 队列queue
 - 优先队列priority_queue

北京大学附属中学 肖然 2024-1-13

前置知识

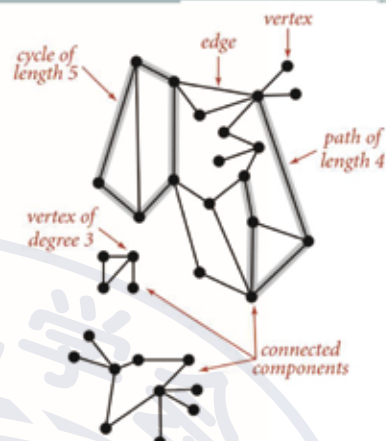
- 基础图论
 - 图的存储方法
 - 图的dfs/bfs
 - DAG拓扑排序
- 动态规划
- 基础贪心思想
- 时间复杂度分析
- 数据结构
 - 队列queue
 - 优先队列priority_queue

北京大学附属中学 肖然 2024-1-13

5

图相关概念

- 图：由两个集合 (V, E) 所组成，记作 $G(V, E)$
 - V 是图中顶点 (Vertex) 的非空集合
 - E 是图中边 (Edge) 的集合
 - 边 $\langle u, v, w \rangle$ 代表 u 指向 v ，长度为 w 的一条边；无向边视为正反2条有向边
 - 若 u, v 之间有边，称 u, v 邻接
 - 简单图：没有重边、自环



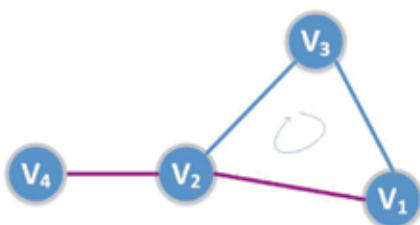
重边/自环



北京大学附属中学 肖然 2024-1-13

6

- 路径 (path)：图中一个点序列称为路径，序列相邻点之间都有边。
 - 简单路径：路径上点和边都不重复出现
 - 环 (cycle)：路径起点和终点相同
 - 路径长度：路径上所有边权总和



$\langle V_4, V_2, V_1 \rangle$ 是一条路径
 路径 $\langle V_2, V_3, V_1 \rangle$ 是一个环

北京大学附属中学 肖然 2024-1-13

7

最短路问题

(Single-Source Shortest Paths, SSSP)

- 给定带权图 $G(V, E)$ 和起点 s ，求从 s 到图中其余各点的最短路（长度/方案）
 - 注意图中可以有负权边，但不能有负环
- $dis[i] = s \rightarrow i$ 的最短路长度
- 输入： V, E
- 输出： $dis[1 \sim n]$



An SPT with 250 vertices

北京大学附属中学 肖然 2024-1-13

8

图的输入方式（略）

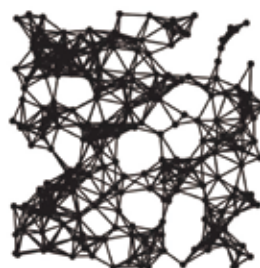
- 图一般以边的列表形式输入（给出点数 N ，边数 M ；然后给出每条边的起点、终点、权值）

V → 13
13 ← E
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3



V → 250
1273 ← E
244 246
239 240
238 245
235 238
233 240
232 248
231 248
229 249
228 241
226 231
...

(1263 additional lines)



肖 2024-1-13

9

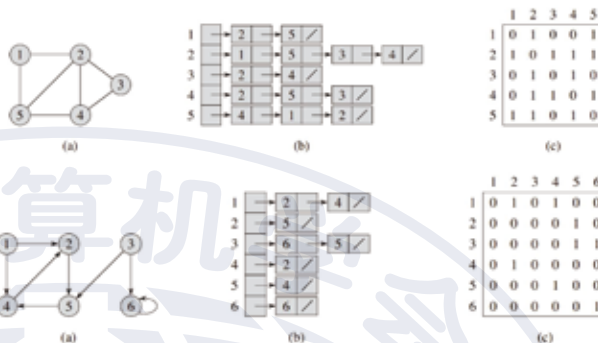
图的存储方法1 - 邻接矩阵 (略)

- 图可以用一个 $N \times N$ 的二维数组 (矩阵) 表示, 称为邻接矩阵

- 无向图的邻接矩阵是对称矩阵

$$G[i][j] = G[j][i]$$

```
int G[MAXN][MAXN];
void add_edge(int u, int v, int w){
    G[u][v] = w;
    //G[v][u] = w; //如果是双向边
}
```



北京大学附属中学 肖然 2024-1-13

10

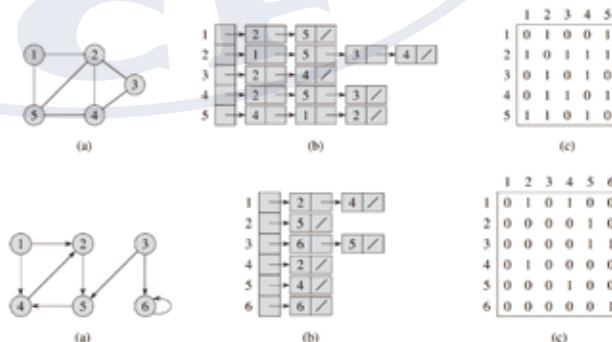
图的存储方法2 - 邻接表 (略)

- 开vector的数组adj表示邻接表, adj[u]包含u的所有出边,

- edge由2个属性组成

- 终点 v 和边权 w

```
struct edge{
    int v,w;
};
vector<edge> adj[MAXN];
void add_edge(int u, int v, int w){
    adj[u].push_back(edge(v,w));
    //adj[v].push_back(edge(u,w)); //如果是双向边
}
```



北京大学附属中学 肖然 2024-1-13

符号

- 图 $G(V, E)$
 - V : 点集或点数(n)
 - E : 边集或边数(m)
- 边 $\langle u, v, w \rangle$: u 指向 v , 长度为 w 的一条边; 无向边视为正反2条有向边
- 邻接: u, v 邻接当且仅当 u, v 之间有边
- 单源最短路
 - s : 最短路问题中的起点
 - $dis[i] = s \rightarrow i$ 的最短路径长度; $d[i] =$ 算法过程中, $s \rightarrow i$ 当前最短路径长度
 - $cnt[u] = s \rightarrow u$ 最短路径上的边数
- 所有点对最短路
 - $d_{ij}^{(k)} = i \rightarrow j$ 的最短路长度、且路径的中间结点只包含前 k 个结点 ($1 \sim k$)
 - $d[u][v] = u \rightarrow v$ 的最短路长度

北京大学附属中学 肖然 2024-1-13

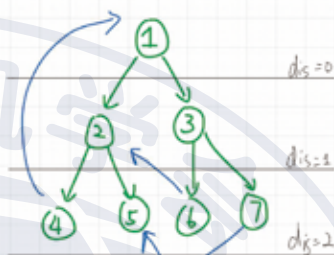
无权图的最短路

- ✓ BFS
- ✓ 0/1 BFS

13

广度优先搜索 (BFS)

- 按层次遍历过程中，若 v_1 在 v_2 之前访问，则 v_1 的邻接点也将在 v_2 的邻接点之前访问，这里蕴含了一个排队关系。
- BFS需要避免重复访问一个结点：可以用 vis/d 判重
- BFS算法流程：
 - 起点 $dis[s] = 0$
 - s 入队
 - while(!q.empty()): // 如果队列不为空
 - 从队首pop出一个顶点做为 u ;
 - 访问与 u 邻接的所有（还没有访问过的）顶点 v
 - $dis[v] = dis[u] + 1$
 - 将 v 入队
- 复杂度 $O(V+E)$ ，注意队列中的点始终关于 dis 升序



BFS遍历序列如下（不唯一）：
1→2→3→4→5→6→7

北京大学附属中学 肖然 2024-1-13

14

0/1 BFS

- 若边带权，但边权只有0/1两种取值，BFS时可以使用两个队列、或者双端队列
- 经过一条边 (u, v, w) 访问点 v 时：
 - 若 $w=0$ ， v 从队头入队， $q.push_front(v)$
 - 若 $w=1$ ， v 从队尾入队， $q.push_back(v)$
- 正确性：保持了队列中 dis 的有序性
- 复杂度 $O(V+E)$

北京大学附属中学 肖然 2024-1-13

15

问题：1/2 BFS?

- 正确性：保持了队列中 dis 的有序性
- 1/2 BFS?
- a/b BFS? $a, b > 0$

北京大学附属中学 肖然 2024-1-13

16

问题：[NOIP2017 普及组] 棋盘

- $m \times m$ 棋盘，格子有 0/1/2 三种，每步可以 4 联通走。
- 任意时刻所处格子颜色不能为 0，每走一步：
 - 如果 2 个格子颜色相同，代价为 0
 - 如果 2 个格子颜色不同，代价为 1
 - 可以暂时改变某个格子颜色，代价为 2，不能连续使用
- 求从 (1,1) 走到 (m,m) 最小总代价
- 拆点最短路：一个点拆成若干个，代表这个点的不同状态（对未来的不同影响）
 - $d[x][y][1/2]$ = 在 (x,y) 格子，当前颜色为 1/2
 - 下一步有 3 种走法：同色（代价为 0），异色（代价为 1），改变为同色（代价为 2）
- 边权有 0/1/2 三种，使用 3 个队列，每次比较三个队头选小的出队，复杂度线性。

北京大学附属中学 肖然 2024-1-13

17

问题：[USACO06JAN] The Grove S

- 给一个网格图，X构成一个四联通连通块
- 给定起点*，每步可以走到相邻八联通点
- 求起点出发的最小环路使得其围住X构成的连通块

```

...+.
..+X+.
.+XXX+.
↑↑↑↑+XXX+
..+X..+
...+++*
    
```

北京大学附属中学 肖然 2024-1-13

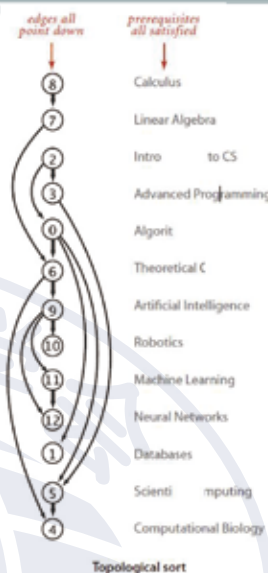
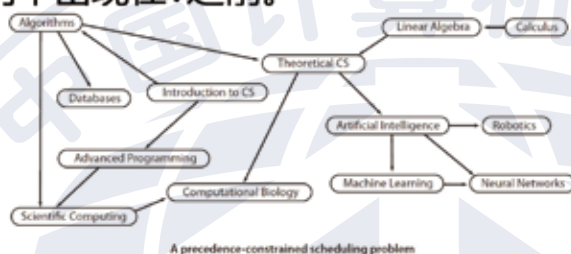
有向无环图的最短路

- ✓ 拓扑排序

19

有向无环图 (DAG)

- Directed Acyclic Graph(DAG)
 - 有向边/不存在环
- DAG可以进行拓扑排序：将图中所有顶点排成一个拓扑序，对于图中任意边 $\langle u, v \rangle$ ， u 在序列中出现在 v 之前。



20

DAG上的最短路问题

- 使用动态规划的思想，求 $dis[v]$ 就是对其所有入边 $\langle u, v, w \rangle$ 进行转移：
 - $dis[v] = \min(dis[u] + w)$
 - 转移一条边 $\langle u, v, w \rangle$ 时需要保证已经算出 $dis[u]$ 的值
- 由于是DAG，DP转移不存在环，可以记忆化搜索/拓扑排序实现
 - 记忆化搜索：需要建反图
 - 拓扑排序：按拓扑序遍历，访问 u 时转移 u 的所有出边

21

拓扑排序

- 拓扑排序求DAG上最短路：
 - 所有入度=0的点入队
 - while(!q.empty()):
 - 从队首pop出一个顶点做为 u
 - 访问与 u 的所有出边 $\langle u, v, w \rangle$
 - $dis[v] = \min(dis[v], dis[u] + w)$
 - 将 v 的入度减 1
 - 若 v 的入度减至 0, 将 v 入队
- 复杂度 $O(V+E)$
- 可以做最长路、路径计数等

```
queue<int> q;
for(int i=1;i<=n;i++){
    if(ind[i] == 0) q.push(i);
}

while(q.size()){
    int u = q.front(); q.pop();
    for(edge e:adj[u]){
        v = e.v; w = e.w;
        dis[v] = min(dis[v], dis[u] + w);
        --ind[v];
        if(ind[v] == 0) q.push(v);
    }
}
```

北京大学附属中学 肖然 2024-1-13

22

问题：DAG最短路

- 在一般图（非DAG）上运行拓扑排序/记忆化搜索求单源最短路，会发生什么？
 - 拓扑排序：环上的点访问不到
 - 记忆化搜索：dfs递归死循环
- 给DAG图 $G(V, E)$ 和起点/终点 s, t , q 次询问：删除某个点 x 之后 $dis[t]$ 的值
 - 求出拓扑序；求出正反最短路 d/dt
 - 边 $\langle u, v, w \rangle$ 对拓扑序上一个区间的点有贡献 $val = d[u] + w + dt[v]$
 - 区间和 val 取min/单点求值

北京大学附属中学 肖然 2024-1-13

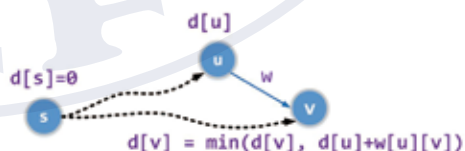
一般图最短路

- ✓ 基本操作：松弛
- ✓ Bellman-Ford
- ✓ SPFA

24

基本操作：松弛 (relaxation)

- 对于有向边 $\langle u, v, w \rangle$ 执行：
 - $d[v] = \min(d[v], d[u] + w)$
- 称为边 $\langle u, v, w \rangle$ 上的【松弛操作】

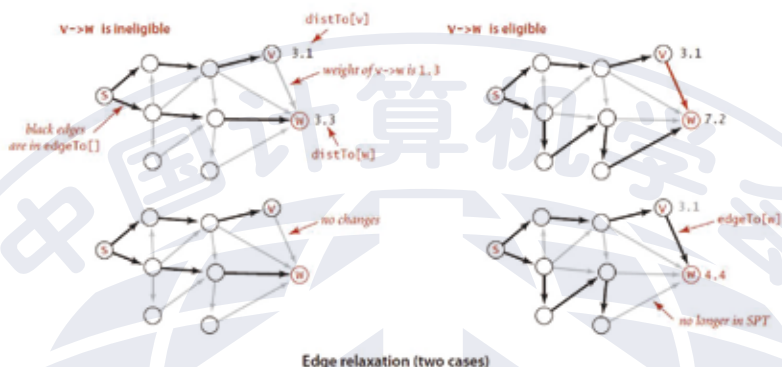


- 松弛一条边 $\langle u, v, w \rangle$ ，即检查是否可以通过中间点u，对目前找到的v的最短路 $d[v]$ 进行改进。若可以改进，更新 $d[v]$ 值。
- 注意松弛只会减小d值
- 松弛是改变最短路径d值的唯一方式

25

基本操作：松弛 (relaxation)

- 最短路算法过程就是通过不断松弛使 $d[i]$ 从初始 inf 收敛至 $\text{dis}[i]$ 的过程



肖然 2024-1-13

26

松弛的性质

- 三角不等式：
 - 对任意边 $\langle u, v, w \rangle$, 有 $\text{dis}[v] \leq \text{dis}[u] + w$
- 收敛性质：
 - 如果图 G 中边 $\langle u, v \rangle$ 在 $s \rightarrow v$ 最短路径上, 考察路径 $s \rightarrow u \rightarrow v$, 若算法过程中在松弛边 $\langle u, v \rangle$ 之前的任何时刻有 $d[u] = \text{dis}[u]$, 则在松弛过后有 $d[v] = \text{dis}[v]$.
- 路径松弛性质：
 - 如果 $p = \langle s, v_1, \dots, v_k \rangle$ 是从 $s \rightarrow v_k$ 的最短路径, 而且 p 中的边按照 $\langle s, v_1 \rangle, \langle v_1, v_2 \rangle, \dots, \langle v_{k-1}, v_k \rangle$ 的顺序进行松弛, 那么最后 $d[v_k] = \text{dis}[v_k]$. 这个性质并不受其他松弛操作的影响, 即使他们与 p 中边上的松弛操作混合在一起.

北京大学附属中学 肖然 2024-1-13

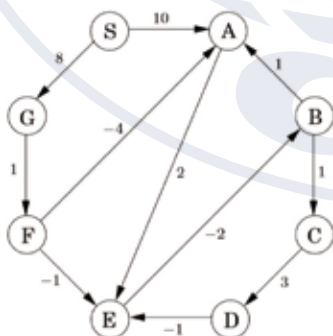
27

Bellman-Ford算法

- Bellman-Ford算法：解决含负权边的带权有向图的单源最短路径问题，并且图G保证无负环。
 - 最短路最多只经过(起点不算) $V-1$ 个结点，根据路径松弛性质，可以通过 $V-1$ 轮松弛操作得到。
- Bellman-Ford算法流程：
 - $d[s] = 0$ ，其他 $d[i] = \text{inf}$
 - 重复 $V-1$ 轮：
 - 松弛图中所有边 $\langle u, v, w \rangle$: $d[v] = \min(d[v], d[u] + w)$
- 复杂度 $O(VE)$

北京大学附属中学 肖然 2024-1-13

28



	Iteration							
Node	0	1	2	3	4	5	6	7
S	0	0	0	0	0	0	0	0
A	∞	10	10	5	5	5	5	5
B	∞	∞	∞	10	6	5	5	5
C	∞	∞	∞	∞	11	7	6	6
D	∞	∞	∞	∞	∞	14	10	9
E	∞	∞	12	8	7	7	7	7
F	∞	∞	9	9	9	9	9	9
G	∞	8	8	8	8	8	8	8

- Bellman-Ford会执行很多无用的松弛操作
- 很多点的dis很早就确定了，之后一直不变
- 带负权图只能使用Bellman-Ford算法



北京大学附属中学 肖然 2024-1-13

第k轮松弛的结果

- 第 k 轮松弛之后，数组 $d[i]$ 的意义是什么？
 - 此时 $d[i]$ 为起点出发最多经过 k 条边到达点 i 的最短路径长度
- 算法的最终目的是在 $V-1$ 轮松弛之后，计算出起点出发最多经过 $V-1$ 条边到达结点 i 的最短路径长度。
- 一些优化：
 - 若 $d[i]$ 上一轮没变，则本轮不用松弛 i 的出边
 - 如果在某轮松弛之中所有结点的 $d[i]$ 都没有改变，那么算法可以提前结束

北京大学附属中学 肖然 2024-1-13

Bellman-Ford判负环

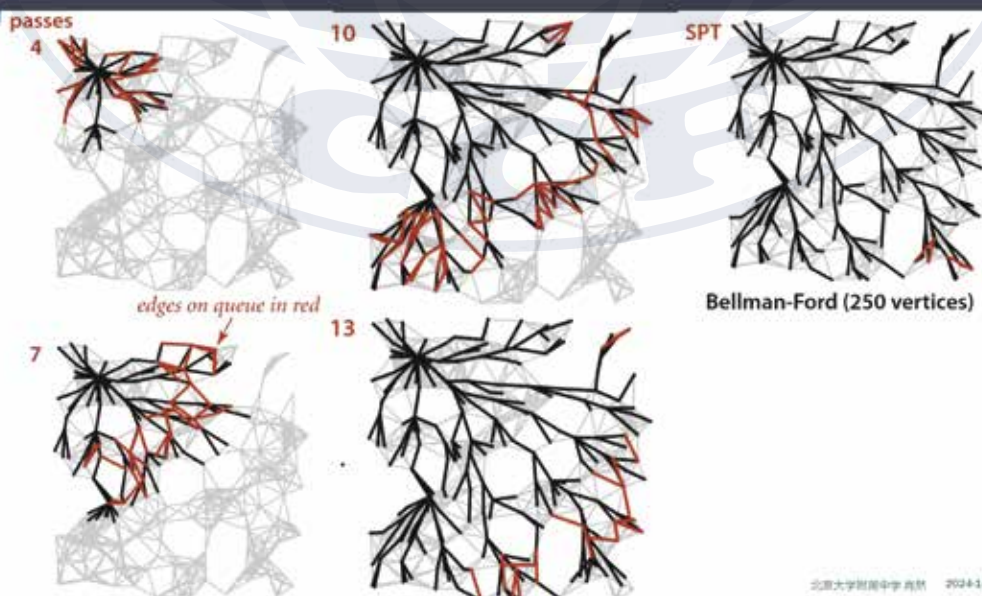
- 在进行完 $V-1$ 轮松弛之后，再加一轮松弛，如果此时有的边仍旧可以被松弛，意味着 G 包含 s 可达的【负环】，最短路不存在。
- 证明：
 - 如果成立，则说明找到了一条经过了 n 条边的从 u 到 v 的路径，且其比任何少于 n 条边的从 u 到 v 的路径都短。
 - 一共 n 个顶点，路径却经过了 n 条边，则必有一个中间顶点 k 经过了至少两次。则 k 是一个环的起点和终点。走这个环比不走这个环路径更短，只能说明这个回路是负环。

北京大学附属中学 肖然 2024-1-13

SPFA算法

- SPFA(Shortest Path Faster Algorithm), 是Bellman-Ford基础上的队列优化, 减少了冗余的松弛操作, 常数较小。就是将BFS进行到底, 能更新d就更新d然后进队, 直到所有结点都无法被更新 (队列为空)
- SPFA算法流程:
 - 起点 $d[s] = 0$, 其他 $d[i] = \text{inf}$
 - s入队
 - while(!q.empty()):
 - 从队首pop出一个顶点做为u;
 - 访问与u邻接的所有 (还没有访问过的) 顶点v
 - 松弛 $\langle u, v, w \rangle$, 若松弛成功将 v 入队
- 能够证明, SPFA算法会执行Bellman-Ford算法中的所有有效松弛操作

北京大学附属中学 肖然 2024-1-13



北京大学附属中学 肖然 2024-1-13

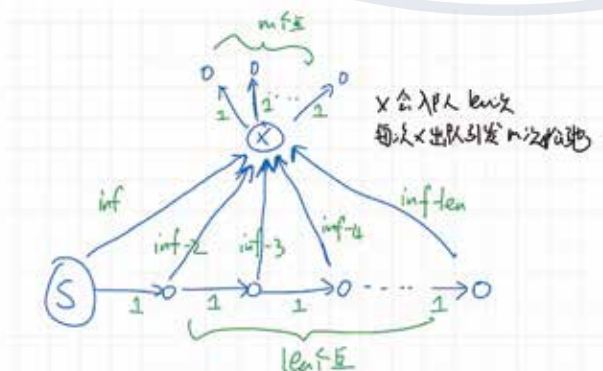
SPFA算法

- 复杂度理论上界 $O(VE)$ ，随机图表现非常好，但是会被卡(?)
- 仍然需要学：判负环，费用流
- 各种优化都可以被定向hack
- inque优化并没有想象中的强
 - 不加inque优化的spfa和bellman-ford基本等价
 - inque优化甚至可能会增加松弛次数，随机图上一般能减少10%左右；inque优化起作用主要是因为stl_queue比较慢，开O2之后几乎没作用
- 1个点被松弛的次数上界仍然是 $V-1$

北京大学附属中学 肖然 2024-1-13

问题：卡SPFA

- 链套菊花
 - 造一条边数len很多的最短路，路径上所有点 $\rightarrow x$ 连边
 - x 会进队len次， x 再连出去1个菊花



北京大学附属中学 肖然 2024-1-13

SPFA判负环

- 设 $\text{cnt}[u] = s \rightarrow u$ 最短路径上的边数
 - $\langle u, v, w \rangle$ 松弛成功时: $\text{cnt}[v] = \text{cnt}[u] + 1$
- SPFA过程中, 若发现【 $\text{cnt}[u] \geq n$ 】, 则有负环
 - 由于最短路边数 $\leq n-1$, 正确性显然
 - 注意只能找到 s 能到达的负环
- 另外还有一种方法: 加Inqueue优化的前提下, 记录每个点进队次数、若 $\geq n$ 则有负环
 - 正确性不显然 (每次进队最短路径边数递增?)
 - 常数较大

北京大学附属中学 肖然 2024-1-13

无负权图最短路

✓ Dijkstra算法

37

Dijkstra(1930-2002)

- 荷兰计算机科学家，早年钻研物理及数学，而后转为计算学。获1972图灵奖。
- 1968年Dijkstra给ACM通讯写了一篇短文，这就是具有历史意义的、著名的《Go To Letter》
 - “Go To语句太容易把程序弄乱，应从一切高级语言中去掉；只用三种基本控制结构就可以写各种程序，而这样的程序可以由上而下阅读而不会返回”
 - 这封信引起了激烈的讨论。人们逐渐认识到：不是一个简单地去掉Go To的问题，而是促进一种新的程序设计观念、方法和风格，以期显著提高软件生产率和降低软件维护代价

北京大学附属中学 肖然 2024-1-13

38

Dijkstra算法流程

1. 设置两个点集的 U 和 $V - U$
 - U 中存放已经确定最短路径 ($d[i]=dis[i]$) 的点
 - $V - U$ 存放还未确定最短路径 ($d[i]>dis[i]$) 的点
2. 初始状态时，集合 U 中只包含起点 s
3. 重复直到集合 U 中包含全部点
 - 3.1 从集合 $V - U$ 中选取当前 d 值最小的点 u 加入到 U
 - 3.2 松弛 u 的所有出边

北京大学附属中学 肖然 2024-1-13

39

Dijkstra算法正确性

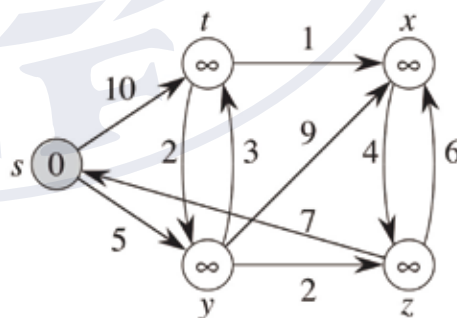
- 能够证明：当点 u 被加入 U 集合时，其最短路径长度确定，即 $\text{dis}[u] = d[u]$
- 可以使用归纳反证说明
 - 注意此时 u 是 $V-U$ 集合中 d 值最小的点，且 U 集合中点的出边都已松弛，根据松弛的收敛性质：若最优路径仍未被找到，路径在 u 之前必然经过至少1个 $V-U$ 中的点 x ，这与 $d[u] \leq d[x]$ 矛盾



40

Dijkstra例子

- 输入一个无负权图
- 初始化：
 - 起点 $d[s] = 0$ ，其他 $d[] = \infty$
- $U = \{s\}$

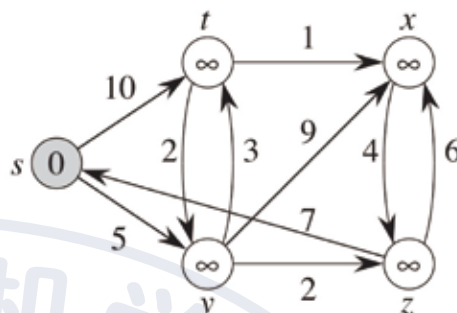


s	t	x	y	z
0	∞	∞	∞	∞

41

Dijkstra例子

- 选出d值最小的点 s 加入 U 集合
- $U = \{s\}$



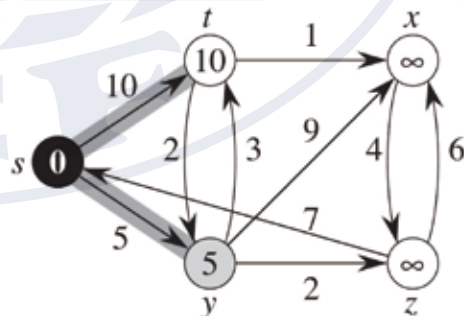
s	t	x	y	z
0	∞	∞	∞	∞

北京大学附属中学 肖然 2024-1-13

42

Dijkstra例子

- 松弛 s 的所有出边
- $U = \{s\}$



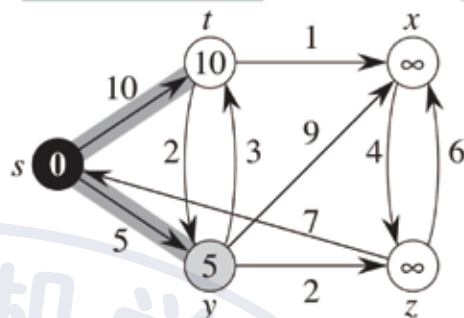
s	t	x	y	z
0	10	∞	5	∞

北京大学附属中学 肖然 2024-1-13

43

Dijkstra例子

- 选出d值最小的点 y 加入 U 集合
- $U = \{s, y\}$



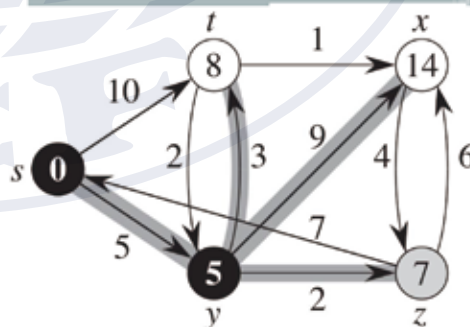
s	t	x	y	z
0	10	∞	5	∞

北京大学附属中学 肖然 2024-1-13

44

Dijkstra例子

- 松弛 y 的所有出边
- $U = \{s, y\}$



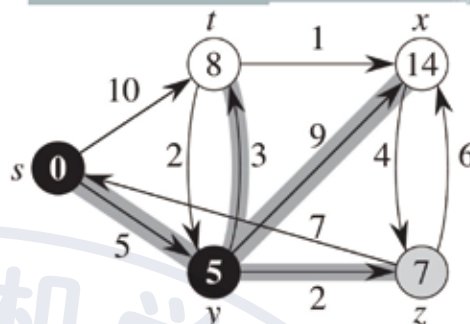
s	t	x	y	z
0	8	14	5	7

北京大学附属中学 肖然 2024-1-13

45

Dijkstra例子

- 选出d值最小的 z 加入 U 集合
- $U = \{s, y, z\}$



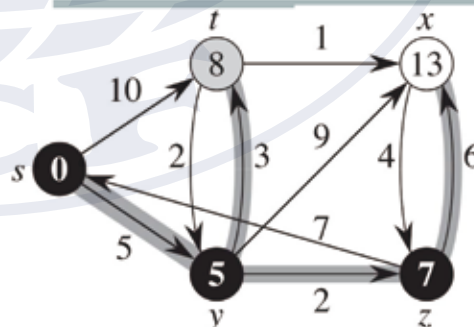
s	t	x	y	z
0	8	14	5	7

北京大学附属中学 肖然 2024-1-13

46

Dijkstra例子

- 松弛 z 的所有出边
- $U = \{s, y, z\}$



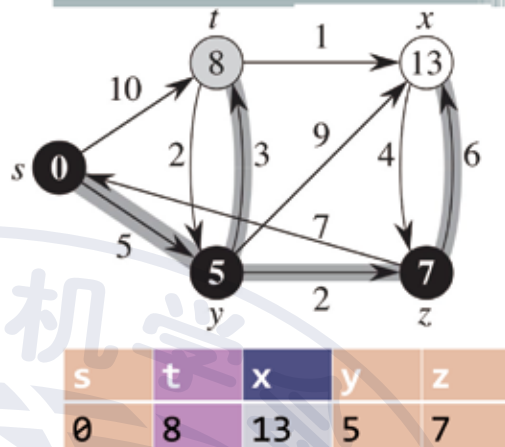
s	t	x	y	z
0	8	13	5	7

北京大学附属中学 肖然 2024-1-13

47

Dijkstra例子

- 选出d值最小的 t 加入U集合
- $U = \{s, y, z, t\}$

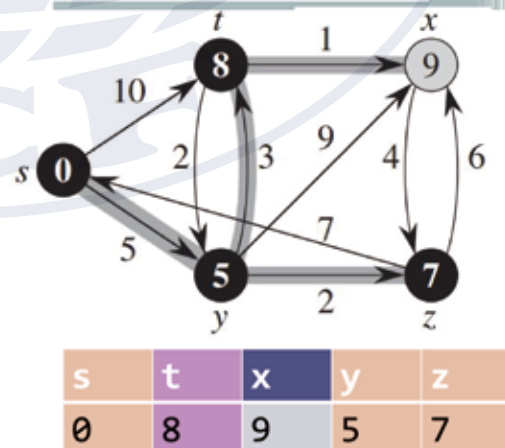


北京大学附属中学 肖然 2024-1-13

48

Dijkstra例子

- 松弛 t 的所有出边
- $U = \{s, y, z, t\}$

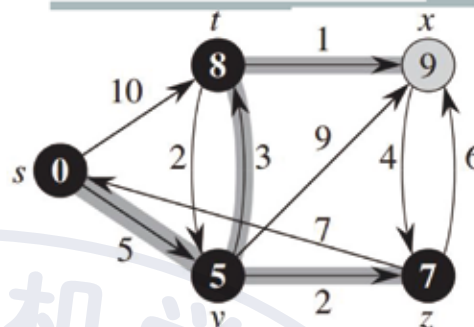


北京大学附属中学 肖然 2024-1-13

49

Dijkstra例子

- 选出d值最小的 x 加入 U 集合
- $U = \{s, y, z, t, x\}$



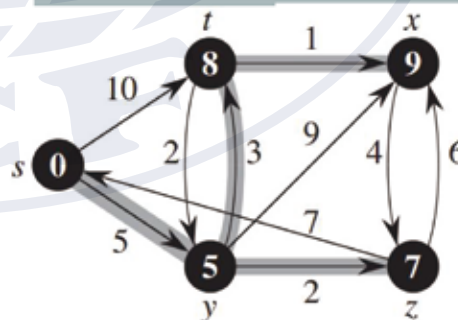
s	t	x	y	z
0	8	9	5	7

北京大学附属中学 肖然 2024-1-13

50

Dijkstra例子

- 最终所有点的d = dis, $U = V$
- $U = \{s, y, z, t, x\}$



s	t	x	y	z
0	8	9	5	7

北京大学附属中学 肖然 2024-1-13

51

Dijkstra复杂度

- 基于优先队列的实现
 - 每个点可能出队多次，只在首次出队时松弛其出边
 - 每条边被松弛恰好1次
 - 点v最多入队次数=入度，优先队列的size=E
 - 优先队列的push()/pop()操作是 $\log(\text{size})$ 的
- 时间复杂度 $O(E \log E)$
- 空间 $O(V+E)$

```
struct node{
    int i,d;
    bool operator < (const node& _) const{
        return d > _.d;
    }
};

struct edge{
    int v,w;
};

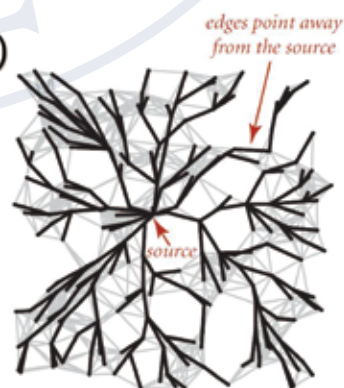
vector<edge> adj[MAXN];
int d[MAXN], vis[MAXN];

void dijkstra(){
    for(int i=1;i<=n;i++) d[i] = 2e9;
    priority_queue<node> q;
    d[s] = 0;
    q.push(node(s,0));
    while(!q.empty()){
        int u = q.top().id; q.pop();
        if(vis[u]) continue; // 有可能多次出队
        vis[u] = 1;
        for(edge e:adj[u]){
            int v = e.v, w = e.w;
            if(d[u] + w < d[v]){
                d[v] = d[u] + w;
                q.push(node(v,d[v]));
            }
        }
    }
}
```

52

最短路树

- 从结点s出发的单源最短路构成一棵树 (DAG)
- DAG上的边 $\langle u,v,w \rangle$ 满足
 - $d[u] + w = d[v]$
- 最短路计数 = DAG上路径计数
- 把图上问题转化为DAG上问题



An SPT with 250 vertices

北京大学附属中学 肖然 2024-1-13

问题：Dijkstra相关

- 如何输出方案
- 多次出队不判vis
 - 正确性？是正确的
 - 复杂度？ $O(nm\log(nm))$
- 若有负边权、没有负环
 - 判vis会发生什么？会WA
 - 不判vis会发生什么？会卡成指数

```

struct node{
    int i,d;
    bool operator < (const node& _) const{
        return d > _.d;
    }
};

struct edge{
    int v,w;
};

vector<edge> adj[MAXN];
int d[MAXN], vis[MAXN];

void dijkstra(){
    for(int i=1;i<=n;i++) d[i] = 2e9;
    priority_queue<node> q;
    d[s] = 0;
    q.push(node(s,0));
    while(!q.empty()){
        int u = q.top().i; q.pop();
        if(vis[u]) continue; // 判vis
        vis[u] = 1;
        for(edge e:adj[u]){
            int v = e.v, w = e.w;
            if(d[u] + w < d[v]){
                d[v] = d[u] + w;
                q.push(node(v,d[v]));
            }
        }
    }
}
    
```

问题：2021CSP-S 第一轮

- 给定整数 n ，使用数量最少的4进行加法、减法、整除运算得到 n 。
- 运算过程中只能出现不超过 $M=10000$ 的数。
- 设 $d[i]$ = 得到数字 i 最少需要的4数量
- 维护 d 值确定的数字集合 U
- 每次选出 $V-U$ 中 d 值最小的 u 放入 U 中， u 和其他数 x 进行运算更新运算结果的 d 值：
 - $d[u+x] = \min(d[u+x], d[u] + d[x])$
 - $d[\text{abs}(u-x)] = \min(d[\text{abs}(u-x)], d[u] + d[x])$
 - $d[u/x] = \min(d[u/x], d[u] + d[x])$
 - $d[x/u] = \min(d[x/u], d[u] + d[x])$

多源最短路

✓ Floyd算法

56

Floyd算法

- 给一个图G，可以有负权（无负环），求任意点对之间最短路径长度 $d[u][v]$
- 实际上是一个DP

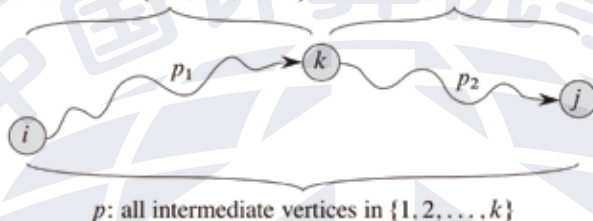
```
void floyd(){
    //初始d[i][i]=0, d[i][j]=边权, 其他d[i][j] = inf
    for(int k=1;k<=n;k++){
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n;j++){
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
            }
        }
    }
}
```

北京大学附属中学 肖然 2024-1-13

57

- 定义 $d_{ij}^{(k)}$ 为: $i \rightarrow j$ 的最短路且路径的中间结点只包含前 k 个结点 ($1 \sim k$)
- 考察 $d_{ij}^{(k)}$ 与 $d_{ij}^{(k-1)}$ 之间的转移。对于 $d_{ij}^{(k)}$ 对应的最短路 p :
 - 若结点 k 不是 p 的中间结点: $d_{ij}^{(k)} = d_{ij}^{(k-1)}$
 - 若结点 k 是 p 的中间结点 (不会多次经过), 那么可将路径 p 分解为 $p_1: i \rightarrow k$, $p_2: k \rightarrow j$ 。 p_1, p_2 的中间结点都不包含结点 k : $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

all intermediate vertices in $\{1, 2, \dots, k-1\}$ all intermediate vertices in $\{1, 2, \dots, k-1\}$



北京大学附属中学 肖然 2024-1-13

58

- 原问题的解可以表示为矩阵 $D^{(n)} = \text{dis}(i, j) = d_{ij}^{(n)}$
- 初始条件:
 - $D^{(0)} = G$
 - $\langle i, j \rangle$ 有边/无边: $d_{ij}^{(0)} = w/\text{inf}$
 - 对角线: $d_{ii}^{(0)} = 0$
- 动态规划的递推过程: $D^{(0)} \Rightarrow D^{(1)} \Rightarrow \dots \Rightarrow D^{(n)}$
- $D^{(k-1)} \Rightarrow D^{(k)}$ 的状态转移方程:

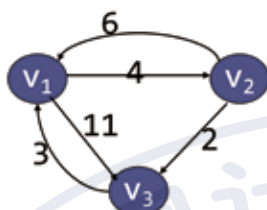
$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

北京大学附属中学 肖然 2024-1-13

59

Floyd例子

$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$



$D^0 = G$

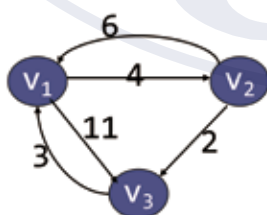
	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

北京大学附属中学 肖然 2024-1-13

60

Floyd例子

$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$



$D^0 =$

	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

$\Rightarrow D^1 =$

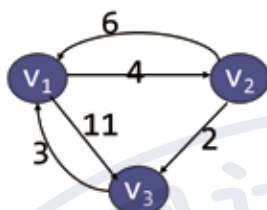
	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

北京大学附属中学 肖然 2024-1-13

61

Floyd例子

$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$



$D^1 =$

	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

$\Rightarrow D^2 =$

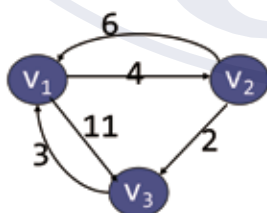
	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

北京大学附属中学 肖然 2024-1-13

62

Floyd例子

$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$



$D^2 =$

	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

$\Rightarrow D^3 =$

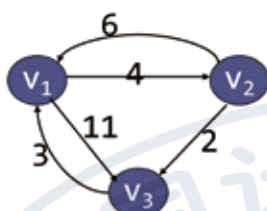
	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

北京大学附属中学 肖然 2024-1-13

63

Floyd例子

$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
实现时上标这一维可以省略（滚动数组）
时间复杂度 $O(n^3)$ ，空间 $O(n^2)$



$D^0 = G =$

	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

$D^1 =$

	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

$D^2 =$

	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

$D^3 =$

	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

北京大学附属中学 肖然 2024-1-13

64

问题：Floyd算法1

- Floyd算法是否能处理负边权？能否判负环？
- k的循环从 $n \rightarrow 1$ 是否可以？是任意排列是否可以？

```
void floyd(){
    //初始d[i][i]=0, d[i][j]=边权, 其他d[i][j] = inf
    for(int k=1;k<=n;k++){
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n;j++){
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
            }
        }
    }
}
```

北京大学附属中学 肖然 2024-1-13

问题：Floyd算法2

- 输出方案？最短路计数？
- 给一个边权都为正的图 $G(V,E)$ ，删除最多的边使得 d 数组不变；如果不限边权为正，有什么问题？

同余最短路

67

同余最短路

- 给定 n 个整数，求：
 - n 个整数能拼凑出多少的其他整数（可以重复取）
 - n 个整数不能拼凑出的最小（最大）的整数
 - 至少要拼几次才能拼出 $\%p = k$ 的数
- 利用同余来构造一些状态：按照 $\%p$ 将状态分类（状态压缩），优化复杂度
- 利用同余构造的状态看作单源最短路中的点，转移形式：
 - $f(u) + w \rightarrow f(v), v = (u + w) \% p$
 - 类似单源最短路中 $f(u) + w \rightarrow f(v)$

北京大学附属中学 肖然 2024-1-13

68

问题：一个例子

- 给定3个正整数 x, y, z ，问在 $[1, h]$ 范围内有多少个数能表示成 x, y, z 线性组合
 - 线性组合： $ax+by+cz=k$ ， a, b, c 为任意整数
 - $1 \leq x, y, z \leq 100000$ ， $h \leq 2^{63}-1$
- 按 $\%x$ 余数将 $[1, h]$ 中的数分类
- 对于 $\%x=i$ 的数，设 $d[i] =$ 能被 y, z 表示出的最小数
 - 即对于 $\%x=i$ 的数，合法的 k 是 $\{d[i], d[i]+x, d[i]+2x, \dots\} \cap [1, h]$
- 初始 $d[0]=0$ ，转移：
 - $d[i]+y \rightarrow d[(i+y)\%x]$ ， $d[i]+z \rightarrow d[(i+z)\%x]$
- 跑最短路求出 $d[0 \sim x-1]$ ，每类分别统计答案

北京大学附属中学 肖然 2024-1-13

问题：[NOIP2017 提高组] 小凯的疑惑

- 给正整数 x, y ，求不能被 $ax+by, a \geq 0, b \geq 0$ 表出的最大整数
- 保证答案存在
- $x, y \leq 10^9$
- 保证答案存在： x, y 互质
- 不妨设 $x < y$ ；按 $\%x$ 的余数分类，转移图是一个环
- 不能表示的最大数 = 最大的 d 值 $-a$
- 环上最后一个点 d 值 = $(a-1)b$

北京大学附属中学 肖然 2024-1-13

总结

- ✓ 各算法比较
- ✓ CSP真题
- ✓ 练习题单
- ✓ 致谢

71

算法	算法思想	适用范围	时间复杂度	空间复杂度	备注
BFS	搜索	无权图 01边权图	$O(V+E)$	$O(V+E)$	a/b边权图
拓扑排序	DP/记忆化搜索	DAG, 可以负权	$O(V+E)$	$O(V+E)$	
Bellman-Ford	迭代/DP	一般图	$O(VE)$	$O(V+E)$	判负环
SPFA	迭代/DP	一般图	$O(VE)$	$O(V+E)$	随机图表现很好 判负环 费用流
Dijkstra	贪心	无负权图	$O(E \log E)$	$O(V+E)$	
Floyd	DP	一般图	$O(V^3)$	$O(V^2)$	常数很小 求最小环 求传递闭包

72

问题：[CSP-J2019] 加工零件

- 给一个 n 点 m 条边无向图，每条边长度为 1，起点 $s=1$
- q 次询问 (a, l) ：判定 $1 \rightarrow a$ 是否存在长度 l 路径
- $n, m, q \leq 10^5$, $l \leq 10^9$
- 无向边（大小=2环）：如果 $1 \rightarrow a$ 存在长度 l 路径，则必然存在长度 $l+2, l+4, \dots$ 路径
- 路径按长度奇偶性分类（无向边是长度=2的环）
 - $d[u][0] = 1 \rightarrow u$ 偶数最短路径
 - $d[u][1] = 1 \rightarrow u$ 奇数最短路径
- 无权图，BFS求最短路
- 回答询问：判定 $d[a][l \% 2] \leq l$ 是否成立
- 复杂度 $O(n+m+q)$

北京大学附属中学 肖然 2024-1-13

问题：[CSP-J2023] 旅游巴士

- 给定 n 点 m 条边有向图，边权=1，边有出现时间 w ，起点1处有长度= k 自环
- 求一条 $1 \rightarrow n$ 长度为 k 整数倍的最短路径
- 路径长度按 k 分类
 - $d[u][j] = 1 \rightarrow u$ 长度模 k 余 j 的最短路径
- 一条边 $\langle u, v, w \rangle$ 的松弛：
 - 若 $d[u][j] \leq w$: $d[u][j]+1 \rightarrow d[v][j+1]$
 - 若 $d[u][j] > w$: $d[u][j]+xk+1 \rightarrow d[v][j+1]$, $x = (w-d[u][j])/k$ 上取整
- Dijkstra复杂度 $O(E \log E)$, $E = mk$

北京大学附属中学 肖然 2024-1-13

练习题单

无权图的最短路: BFS

- ✓ P2864 [USACO06JAN] The Grove S
- ✓ P3958 [NOIP2017 提高组] 奶酪
- ✓ P3956 [NOIP2017 普及组] 棋盘

有向无环图的最短路: topo-Sort

- ✓ P1807 最长路

一般图最短路: Bellman-Ford(SPFA)

- ✓ P3385 【模板】负环

无负权图最短路: Dijkstra

- ✓ P4779 【模板】单源最短路径 (标准版)
- ✓ P1629 邮递员送信
- ✓ P1144 最短路计数

卡SPFA

- ✓ P3640 [APIO2013] 出题人
- ✓ P4768 [NOI2018] 归程

多源最短路: Floyd

- ✓ B3647 【模板】Floyd
- ✓ P1119 灾后重建
- ✓ P2047 [NOI2007] 社交网络

同余最短路

- ✓ P3403 跳楼机
- ✓ P5663 [CSP-J2019] 加工零件
- ✓ P9751 [CSP-J2023] 旅游巴士

拓展

- ✓ P3953 [NOIP2017 提高组] 逛公园
- ✓ P5100 [JOI 2017 Final] 足球
- ✓ P5468 [NOI2019] 回家路线
- ✓ P6190 [NOI Online #1 入门组] 魔法

北京大学附属中学 肖然 2024-1-13

参考资料与致谢

- ✓OI-WIKI: oi-wiki.org/
- ✓Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-Wesley, 4th edition, 2011
- ✓Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009

感谢CCF的邀请

xiaoran@pkuschool.edu.cn

北京大学附属中学 肖然 2024-1-13

树上问题选讲

长沙市长郡中学 谢秋锋

01

树的定义与相关概念

树的定义与相关概念

- 如果一个无向简单图 G 满足以下相互等价的条件之一，那么 G 是一棵树。
- 1. G 是没有回路的连通图。
- 2. G 没有回路，但在 G 内添加任意一条边就会形成一个回路。
- 3. G 是连通的，且去掉任意一条边后都不再连通。
- 4. G 内的任意两个顶点能被唯一简单路径所连通。
- 5. 如果 G 有有限个顶点（设为 n 个顶点），那么 G 是一棵树还等价于 G 是连通的且具有 $n-1$ 条边的图。

树的定义与相关概念

- **根**：一棵树上指定的特殊节点。
- **有根树和无根树**：有根的树叫作有根树，否则叫作无根树。
- **深度**：从根节点到任意节点的路径长度称为该节点在树中的深度。
- **高度**：节点在树中的高度是该节点向下到某个叶子节点最长简单路径中边的条数。
- 树的高度是根节点的高度，也等于树中节点的最大深度。

树的定义与相关概念

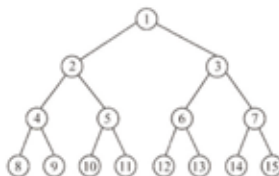
- **叶子节点**：没有子节点的节点。
- **分支节点**：有子节点的节点，也称为内部节点。
- **父节点和子节点**：某个内部节点与自己的子节点形成父节点与子节点关系。
- **兄弟节点**：同一个父节点的子节点之间互称兄弟节点。
- **祖先节点**：从该节点到根节点的简单路径上经过的节点称为该节点的祖先节点。
- **子孙节点**：以该节点为根的子树下的所有节点称为该节点的子孙节点。

树的定义与相关概念

- **无序树**：一棵树中任意节点的子节点之间没有顺序关系。
- **有序树**：一棵树中任意节点的子节点之间有顺序关系。
- **森林**：若干棵互不相交的树的集合就是森林，只有一棵树也可以称为森林。

树的定义与相关概念

- **二叉树**：每个节点最多只有两个子树的树。
- **完全二叉树**：一棵二叉树只有最下面两层节点的度数小于 2 且最下面一层的节点都处于该层左边的连续位置上，这样的二叉树称为完全二叉树。
- **满二叉树**：一棵完全二叉树，除了最下面一层没有子节点外，其余每一层上的所有节点有且都有两个子节点，这样的完全二叉树就是满二叉树。





02 树的重心与直径

树的重心

- 在树中选择某个节点并删除，这棵树将分为若干棵子树。
- 此时可以统计每棵子树的节点数并记录最大值。
- 取遍树上所有节点，使此最大值取到最小的节点称为树的重心。

树的重心

```
• void dfs(int x) {  
•     vis[x]=true, siz[x]=1; // 是否被访问/子树大小  
•     int maxx=0;  
•     for(int i:e[x]) {  
•         if(vis[i]) continue; dfs(i);  
•         siz[x]+=siz[i]; maxx=max(maxx,siz[i]);  
•     }  
•     maxx=max(maxx,n-siz[x]);  
•     if(maxx<=n/2) pos=x; // 更新重心的位置  
• }
```

树的重心

- 树的重心具有以下性质：
 1. 以树的重心为根时，所有子树的大小都不超过整棵树的一半。
 2. 树中所有点到某个点的距离和中，到重心的距离和是最小的，如果有两个重心，那么到他们的距离和一样。
 3. 把两棵树通过一条边相连得到一棵新的树，那么新的树的重心在连接原来两棵树的重心的路径上。
 4. 在一棵树上添加或删除一个叶子，那么它的重心最多只移动一条边的距离。

树的重心

- CF685B
- 给定一棵有根树，求每一棵子树的重心是哪一个节点。

树的重心

- CF685B
- 给定一棵有根树，求每一棵子树的重心是哪一个节点。
- 考虑当前在求以 x 为根的子树的重心。
- 若 y 是 x 的一个子节点，且以 y 为根的子树大小超过了以 x 为根的子树大小的一半。
- 将以 y 为根的子树的重心表示为 p ，以 x 为根的子树的重心一定在 p 到 x 的路径上。
- 若不存在满足条件的节点 y ，则以 x 为根的子树的重心就是 x 。

树的重心

```
• void dfs(int x) {  
•     siz[x]=1, ans[x]=x;  
•     for(int to:e[x]) { dfs(to); siz[x]+=siz[to]; maxx[x]=max(maxx[x],maxx[to]); }  
•     for(int to:e[x]) {  
•         if(siz[to]<=siz[x]/2)continue; int p=ans[to];  
•         while(p^x) {  
•             if(max(maxx[p],siz[x]-siz[p])>siz[x]/2) p=fa[p];  
•             else { ans[x]=p; return; }  
•         }  
•     }  
• }
```

树的直径

- 图中任意两点之间的最短简单路径的最大值，称为图的直径。
- 树作为一种特殊的图，其直径可以通过两次 BFS 和树型 DP 两种方法求出。

树的直径

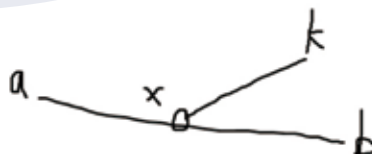
- BFS 做法（能记录路径，不能处理负边权）
- 任选一点 x 作为起点，搜出树上距离起点最远的点 a 。
- 以 a 为起点搜出树上距离 a 点最远的点 b 。
- 则 $a-b$ 之间的简单路径为当前这棵树的直径。

树的直径

- 证明：从 x 出发搜到的最远点 k 一定是直径的端点。

- 情况一： x 在直径上

- 假设 k 不是 a 或 b ，则 $xk > xb$ 。
- 于是 $ab = ax + xb < ax + xk = ak$ 。
- 这与 ab 是直径的已知条件矛盾，所以 k 必然是 a 或 b 。



树的直径

• 证明：从 x 出发找到的最远点 k 一定是直径的端点。

• 情况二： x 不在直径上， xk 与 ab 相交于 t

• 由于 k 是 x 找到的最远点，有 $tk > tb$ 。

• 于是 $ab = at + tb < at + tk = ak$ 。

• 这与 ab 是直径的已知条件矛盾。



树的直径

• 证明：从 x 出发找到的最远点 k 一定是直径的端点。

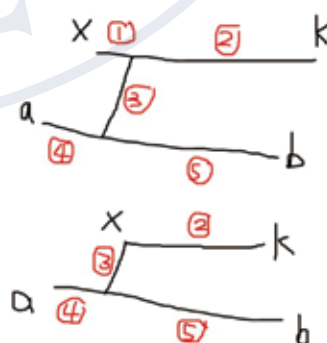
• 情况三： x 不在直径上， xk 与 ab 不相交

• 由于是一棵树，一定可以找到一条路径从 x 走到 ab 上。

• 此时 $② > ③ + ⑤$ ，即 $② > ⑤$ 。

• 那么 $④ + ③ + ② > ④ + ② > ④ + ⑤ = ab$ ，即存在一条比直径更长的路径。

• 与已知条件矛盾。



树的直径

- 树型 DP 做法（能处理负边权，不能记录路径）
- 定义 $d[x]$ ：从 x 出发走向以 x 为根的子树能够到达的最远点的距离。
- 定义 $f[x]$ ：经过点 x 的最长链的长度。
- 在处理的 x 的某棵以 y 为根的子树时，先用 $d[y]+w(x \rightarrow y)+d[x]$ 更新 $f[x]$ 的值，再用 $d[y]+w(x \rightarrow y)$ 更新 $d[x]$ 的值。

树的直径

```
• void dp(int x,int fa) {  
•     for(auto i:e[x]) {  
•         int y=i.first, w=i.second; // y 表示目标点 w 表示边权  
•         if(y==fa) continue;  
•         dp(y,x);  
•         f[x]=max(f[x],d[x]+d[y]+w); // 更新经过 x 的最长链  
•         d[x]=max(d[x],d[y]+w); // 更新 x 出发能走的最远距离  
•     }  
• }
```


树的直径

- APIO2010 巡逻
- 有一棵 n 个节点的树，树上的边权均为 1。
- 要从点 1 出发经过每条边至少一次，最后回到点 1。
- 新加 k 条边，新加的边必须走。
- 求最小的移动距离。

• $n \leq 10^5, 1 \leq k \leq 2$

树的直径

- 新加上一条连接两个不同点的边，移动距离一定会减少。
- 若加上 1 条连接 a 和 b 的边，则总距离减少 $\text{dist}(a,b)-1$ 。
- 当 k 为 1 时， a 和 b 分别为树的直径的两个端点时是最优的。

树的直径

- 当 k 为 2 时，需要加入第 2 条边，构成第 2 个环。
- 若新的环和之前的环没有重复部分，则两个环上除新边外都只走 1 次。
- 如果有重复部分，则重复的边要走 2 次。

树的直径

- 假设第 2 条边连接的是 c 和 d 。
- 如果路径 $c-d$ 和路径 $a-b$ 有重合的部分，则重合的部分在考虑加第 1 条边时，经过的次数已经从 2 变成了 1，重合后又要变回 2。
- 相当于重合部分的边在第 2 次计算 $\text{dist}(c,d)$ 的贡献为 -1。
- 所以，将原树直径上的边的边权修改为 -1 后，可以得到一棵新树。
- 第 2 条边应当连接新树的直径的两个端点。
- 综上，第一次找直径时要记录路径，用搜索。第二次找直径时存在负边权，用树型 DP。



点分治

点分治

- 树的点分治是分治思想在树形结构上的典型应用，常用于解决树上与路径相关的问题。
- 树的点分治的基本思想为：
 - 找到树的一个重心，则问题的解可以分为路径经过重心的解和路径不经过重心的解两部分。
 - 1. 路径经过重心的解：设重心为树根，用数据结构记录重心到每个节点的路径信息，从中取出根的不同子树中的路径信息组合起来即为路径经过重心的解。
 - 2. 路径不经过重心的解：将重心和其关联的边删除，对分割出来的每棵树递归处理即可求出路径不经过重心的解。

点分治

- IOI2011 Race
- 给一棵树，每条边有权。求一条简单路径，权值和等于 k ，且边的数量最小。
- $1 \leq n \leq 2e5$

点分治

- 用桶 $have[i][0/1]$ 记录边权和为 i 的路径是否存在/最少边数。
- 当遍历到某个子节点时：
 1. 先枚举当前子树的所有路径，若枚举到的路径的边权之和为 dis ，边数为 dep ，就在桶中寻找边权之和为 $k-dis$ 的路径是否存在，若存在就用其更新答案。
 2. 再枚举一遍当前子树的所有路径，将枚举到的路径的信息更新到桶中去。
- 所有子节点遍历完毕后，就求完了路径经过当前重心的解。
- 此时需要将桶中信息清空，并将当前点从树中删去。
- 然后，对于每一棵子树，找出重心并递归求解，这部分就是路径不经过当前重心的解。

```
void cal(int now){
    vis[now]=true,have[0][0]=1;
    vector<int>v;
    for(auto i:e[now]){
        int to=i.first,w=i.second;
        if(vis[to])continue;
        dis[to]=w;
        lena=0;
        dfsdis(to,0,1); // 遍历当前子树的所有路径
        for(int j=1;j<=len; ++j){ // 之前的子树中存在一条路径和当前枚举的路径凑出答案
            if(!have[k-q[j].dis][0])continue;
            ans=min(ans,have[k-q[j].dis][1]+q[j].dep);
        }
        for(int j=1;j<=len; ++j){ // 更新值
            if(!have[q[j].dis][0]){
                have[q[j].dis][0]=1;
                have[q[j].dis][1]=q[j].dep;
            }
            else have[q[j].dis][1]=min(have[q[j].dis][1],q[j].dep);
            v.push_back(q[j].dis);
        }
    }
    for(int i:v)have[i][0]=0; // 清空树
    for(auto i:e[now]){
        int to=i.first,w=i.second;
        if(vis[to])continue;
        res+=weightasiz[to]; // 分别表示：找重心过程中最大子树的最小值，原树的子树大小
        dfsrt(to,0);
        cal(rt);
    }
}
```

点分治

- COCI2018 Transport
- 一个国家有 n 个城市，每个城市有一个加油站，燃料储量为 a_i 。
- 有 $n-1$ 条路径，这些城市连接成一个树形结构，每条路径有边权。
- 货车能从 u 走到 v ，货车的燃料量必须不小于 u 和 v 之间的距离。
- 货车抵达一个城市，就可以补充不超过加油站燃料储量的燃料。假设货车的油箱无限大。
- 求有序数对 (u,v) 的个数，满足一辆邮箱初始燃料量为 0 的货车可以从 u 到 v 。
- 货车只能走简单路径。
- $1 \leq n \leq 1e5$

点分治

- COCI2018 Transport

- 一个国家有 n 个城市，每个城市有一个加油站，燃料储量为 a_i 。
- 有 $n-1$ 条路径，这些城市连接成一个树形结构，每条路径有边权。
- 货车能从 u 走到 v ，货车的燃料量必须不小于 u 和 v 之间的距离。
- 货车抵达一个城市，就可以补充不超过加油站燃料储量的燃料。假设货车的油箱无限大。
- 求有序数对 (u, v) 的个数，满足一辆邮箱初始燃料量为 0 的货车可以从 u 到 v 。
- 货车只能走简单路径。
- $1 \leq n \leq 1e5$

点分治

- 设 rt 为当前的分治重心。分别统计 $i \rightarrow rt$ 的路径和 $rt \rightarrow i$ 的路径。
- 对于 $i \rightarrow rt$ 的路径：
 - 设 j 是路径上的点，则满足 $w_i - w_j > d_i - d_j$ 即 $w_i - d_i > w_j - d_j$ 。
 - 所以遍历时记录从根出发的 $w - d$ 的最大值， $w_i - d_i$ 不小于最大值时，记录下当前路径。
- 对于 $rt \rightarrow i$ 的路径：
 - 设 x 是出发时的燃油量， j 是路径上的点，则满足 $x + w_{f[j]} > d_j$ 即 $x + (w_{f[j]} - d_j) > 0$ 。
 - 因为 x 未知，所以所有的路径都可能存在，都先记录下来。

点分治

- $i \rightarrow rt$ 记录下来的值为 $w_j - d_j$ ，存到 $s1$ 数组中。
- $rt \rightarrow i$ 记录下来的值为 $\min\{w_{rt[j]} - d_j\}$ ，存到 $s2$ 数组中。
- 将 $s1$ 和 $s2$ 数组分别排序后，可用双指针求出所以可行的方案。
- 可行的方案满足 $s1[p1] + s2[p2] - a[rt] \geq 0$ 。
- 由于只能走简单路径，所以还需要将同一子树内的匹配方案减去。

01

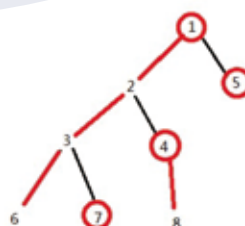
树链剖分和树上启发式合并

树链剖分

- 树链剖分简称为树剖或链剖。
- 是将一棵树分割成若干条链，以便于维护树上路径信息的一种方法。
- 树链剖分有多种形式，如重链剖分，长链剖分和实链剖分。
- 大多数情况下，树链剖分都指重链剖分。

重链剖分

- 重子节点：子节点中子树最大的一个子节点。
- 轻子节点：子节点中除了重子节点以外的节点。
- 重边：节点及其重子节点的连边。
- 轻边：节点及其轻子节点的连边。
- 重链：由多条重边连接而成的路径。
- 链顶：一条重链中深度最小的节点。



重链剖分

- 具体流程：
 - 1. 从根节点开始 DFS，得到树的基本信息：节点深度、子树大小、父节点、重儿子等。
 - 2. 从根节点开始 DFS，搜索时优先搜索重儿子，得到 DFS 序，并记录链顶和每个点在 DFS 序中的位置等。
- 对于 n 个点构成的树，任意点到根的路径上的轻边数目和重链数目均不超过 $\log n$ 条。

重链剖分

```
• void dfs1(int now,int f) {  
•     dep[now]=dep[f]+1, siz[now]=1, fa[now]=f; // 深度、子树大小、父节点  
•     for(int to:e[now]) {  
•         if(to==f) continue;  
•         dfs1(to,now);  
•         siz[now]+=siz[to];  
•         if(siz[to]>siz[son[now]]) son[now]=to;  
•     }  
• }
```

重链剖分

```
• void dfs2(int now,int t) {  
•     dfn[now]=++cnt, top[now]=t; // 树上的点在 dfs 序上的位置、链顶  
•     id[cnt]=now; // dfs 序上的点在树上的位置  
•     if(!son[now]) return;  
•     dfs2(son[now],t); // 处理重儿子  
•     for(int to:e[now]) {  
•         if(to==f||to==son[now]) continue;  
•         dfs2(to,to); // 处理轻儿子  
•     }
```

长链剖分

- 长链剖分和重链剖分的区别在于取子树深度最大的子节点作为重儿子。
- 在长链剖分下：
 - 对于 n 个点构成的树，任意点到根的路径上的轻边数目不超过 \sqrt{n} 条。

长链剖分

- 洛谷 P5903 树上 K 级祖先
- 一棵 n 个点的树， q 次询问，求节点 x 的 k 级祖先。
- $n \leq 5e5, q \leq 5e6$

长链剖分

- 性质：节点 x 的 k 级祖先 y 所在链的长度一定大于 k 。
- 证明：
 1. 当 x 和 y 在同一条重链上时， $\text{dis}(x, y) = k + 1 > k$ 。
 2. 当 x 和 y 不在同一条重链上时， y 所在重链的长度一定大于 $\text{dis}(\text{top}[y], x)$ ，显然也大于 k 。

长链剖分

- 进行长链剖分，对于每条重链的链顶 x ，记对应的重链长度为 $\text{len}[x]$ 。
- 记录链顶往上跳 $1 \sim \text{len}[x]$ 步到达的点，及向下跳 $1 \sim \text{len}[x]$ 步到达的点。
- 需要记录的信息是 $O(n)$ 的。

长链剖分

- 求 x 的 k 级祖先时，设 $2^i \leq k < 2^{i+1}$ ：
- 1. 让 x 跳到 2^i 级祖先，问题转化成求 $x = \text{bz}[x][i]$ 的 $k - 2^i$ 级祖先
- 2. 由于 $k - 2^i < 2^{i+1} - 2^i = 2^i < \text{len}[\text{top}[x]]$ ，所以需要的信息一定已经被预处理过了
- 令 $k = k - (1 < i) - (\text{dep}[x] - \text{dep}[\text{top}[x]])$ 。
- 1. 当 $k \geq 0$ 时，最终答案为 $\text{top}[x]$ 的 k 级祖先
- 2. 当 $k < 0$ 时，最终答案为 $\text{top}[x]$ 的 k 级重子节点

长链剖分

```
• void dfs(int now,int t) {  
•     top[now]=t;  
•     if(now==t) {  
•         for(int i=1,j=now;i<=len[now];++i)u[now].push_back(j),j=fa[j];  
•         for(int i=1,j=now;i<=len[now];++i)v[now].push_back(j),j=son[j];  
•     }  
•     if(son[now]) dfs(son[now],t);  
•     for(int to:e[now]) if(to!=son[now]) dfs(to,to);  
• }
```

长链剖分

```
• int ask(int x,int k) {  
•     if(!k) return x;  
•     int tmp=log2(k);  
•     x=bz[x][tmp], k-=1<tmp;  
•     k-=dep[x]-dep[top[x];  
•     x=top[x];  
•     return k>=0?u[x][k]:v[x][-k];  
• }
```

树上启发式合并

- 树上启发式合并 (DSU on tree) 可用于统计静态子树信息。
- 算法流程为：
 1. 预处理树的基本信息 (如重儿子等)。
 2. 统计以 x 为根的子树的答案时，先递归求解 x 的所有轻儿子所在子树的答案，求解完毕后将产生的贡献删去。随后递归求解 x 的重儿子所在子树的答案，求解完毕后将产生的贡献不删，并在此基础上加入 x 所有轻儿子所在子树的贡献和节点 x 的贡献，从而得到以 x 为根的子树的答案。

树上启发式合并

- 整个过程用 DFS 实现，每个点都会被遍历至少一次。
- 轻儿子所在子树还会额外遍历一遍，这部分被遍历的次数等价于它属于多少个轻子树。
- 即点到根的路径上的轻边的数目。
- 假设树有 n 个节点。
- 那么点到根路径上轻边的数目不会超过 $\log n$ 条。
- 对于所有点来说，被遍历的总次数约为 $n \log n$ 次。

树上启发式合并

- CF600E
- 有一棵 n 个节点以 1 号节点为根的有根树。
- 每个节点都有一个颜色，第 i 个节点的颜色表示为 c_i 。
- 若一种颜色在以 x 为根的子树中出现次数最多，称其在以 x 为根的子树中占主导地位。
- 同一子树中可能有多种颜色占主导地位。
- 对于每个节点，求以 i 为根的子树中，占主导地位的颜色的编号和。
- $c_i \leq n \leq 1e5$

树上启发式合并

- 显然，对于每个节点，都遍历一遍以该节点为根的子树，就可以求出答案。

```
void cal(int x, int f) {  
    for (int to : e[x]) if (to != f) cal(to, x);  
    // 遍历以 x 为根的子树并用 cnt[i] 统计颜色 i 出现的次数  
    // 清空 cnt 数组  
}
```


树上启发式合并

- void cal(int x,int f) {
- for(int to:e[x]) if(to!=f) cal(to,x);
- // 遍历以 x 为根的子树并用 cnt[i] 统计颜色 i 出现的次数
- // 清空 cnt 数组
- }
- 假设计算 cal(x,f) 时最后遍历到的 x 的子节点是 y。
- 则计算 cal(y,x) 时得到的 cnt 数组在回溯后可继续用于计算 cal(x,f)，无需清空。
- 显然当 y 为 x 的重儿子时，就能最大化可重复利用的信息的规模。

```
void cal(int now,int f)
{
    for(int to:e[now]) // 统计轻儿子的答案
    {
        if(to==f||to==son[now])continue;
        cal(to,now);
        // 回溯轻儿子状态
        for(int i=dfn[to],lim=dfn[to]+siz[to]-1;i<=lim;++i)--cnt[a[id[i]]];
        p=sum=0; // p:表示颜色的最大出现次数 sum:表示子树的答案
    }
    if(son[now])cal(son[now],now); // 统计重儿子的答案
    ++cnt[a[now]];
    if(cnt[a[now]]>p)p=cnt[a[now]],sum=a[now];
    else if(cnt[a[now]]==p)sum+=a[now];
    for(int to:e[now])
    { // 把轻儿子的贡献加回来
        if(to==f||to==son[now])continue;
        for(int i=dfn[to],lim=dfn[to]+siz[to]-1;i<=lim;++i)
        {
            ++cnt[a[id[i]]];
            if(cnt[a[id[i]]]>p)p=cnt[a[id[i]]],sum=a[id[i]];
            else if(cnt[a[id[i]]]==p)sum+=a[id[i]];
        }
    }
    ans[now]=sum;
}
```

树上启发式合并

- CF208E
- 有一个家庭关系森林，描述了 n 人的家庭关系。
- 如果存在一个人 z ，是两个人 a 和 b 共同的 p 级祖先，称 a 和 b 为 p 级表亲。
- m 次询问，每次询问给出一对整数 v 和 p ，求编号为 v 的人有多少个 p 级表亲。
- $n, m \leq 1e5$

树上启发式合并

- 将 m 次询问离线下来。
- 对于一次询问 (v, p) ，可以利用倍增等方式求出 v 的 p 级祖先 y 。
- 问题就转化成了求 y 的 p 级孩子有多少个。
- 在遍历子树的过程中，可以用 $cnt[i]$ 记录深度为 i 的节点的数目。
- 那么一次询问的解就是遍历完以 y 为根的子树后， $cnt[dep[y]+p]-1$ 的值。
- 用树上启发式合并进行这个遍历的过程即可。



02 动态DP

动态DP

- 动态 DP 可用来解决树上的带有修改操作的 DP 问题。
- 具体方式是把状态转移写成矩阵形式，再用数据结构进行维护。

动态DP

- 洛谷 P4719 动态 DP
- 给定一棵 n 个点的树，点带点权。
- 有 m 次操作，每次操作给定 x, y ，表示修改点 x 的权值为 y 。
- 需要在每次操作之后求出这棵树的最大权独立集的权值大小。
- $1 \leq n, m \leq 1e5$

动态DP

- 通过树链剖分预处理出每个节点的重子节点。
- 定义 $f[i][0/1]$ 表示节点 i 不选/选时对应子树的最大权独立集。
- 定义 $g[i][0]$ 表示节点 i 的所有轻儿子可选可不选的最大权独立集。
- 定义 $g[i][1]$ 表示节点 i 选上且它的所有轻儿子都不选的最大权独立集。
- 设 j 为 i 的重子节点，可得：
- $f[i][0] = g[i][0] + \max(f[j][0], f[j][1])$
- $f[i][1] = g[i][1] + f[j][0]$

动态DP

- $f[i][0] = g[i][0] + \max(f[j][0], f[j][1])$
- $f[i][1] = g[i][1] + f[j][0]$

- 定义 $(+, \max)$ 形式的广义矩阵乘法，可以构造出矩阵形式的转移：

$$\begin{bmatrix} g[i][0] & g[i][0] \\ g[i][1] & -\infty \end{bmatrix} * \begin{bmatrix} f[j][0] \\ f[j][1] \end{bmatrix} = \begin{bmatrix} f[i][0] \\ f[i][1] \end{bmatrix}$$

动态DP

- 定义 $(+, \max)$ 形式的广义矩阵乘法，可以构造出矩阵形式的转移：

$$\begin{bmatrix} g[i][0] & g[i][0] \\ g[i][1] & -\infty \end{bmatrix} * \begin{bmatrix} f[j][0] \\ f[j][1] \end{bmatrix} = \begin{bmatrix} f[i][0] \\ f[i][1] \end{bmatrix}$$

- 当节点 x 的值改变时，只有从 x 到根路径上的状态信息会随之改变。
- 当向上的转移发生在一条重边上时，转移系数只和轻儿子节点的状态有关。
- 因此，当在一条重链上进行转移时，只要乘上 dfs 序上一段连续的值即可。
- 可以使用线段树进行维护。

动态DP

- 当向上的转移发生在一条重边上时，转移系数只和轻儿子节点的状态有关。
- 因此，可以预处理出 dfs 序上每个点的转移矩阵，并用线段树维护区间乘积。
- 在一条重链上进行转移时，只要乘上一段区间乘积就可以转移到链顶。
- 当向上的转移发生在一条轻边上时，通过转移方程直接修改即可。
- 特别的，重链的链底节点存储的矩阵信息就是当前节点对应子树的答案。
- 所以，总答案就是根节点所在重链的链底节点乘上该重链对应区间乘积后的结果。

```
void dfs1(int now,int f)
{
    fa[now]=f,siz[now]=1,dp[now][1]=w[now];
    for(int to:e[now])
    {
        if(to==f)continue;
        dfs1(to,now);
        dp[now][1]+=dp[to][0];
        dp[now][0]+=max(dp[to][0],dp[to][1]);
        siz[now]+=siz[to];
        if(siz[to]>siz[son[now]])son[now]=to;
    }
}

void dfs2(int now,int f)
{
    dfn[now]++;cnt,id[cnt]=now;
    top[now]=f,tail[f]=cnt;
    v[now][1][0]=w[now];
    if(!son[now])return;
    dfs2(son[now],f);
    for(int to:e[now])
    {
        if(to==fa[now]||to==son[now])continue;
        dfs2(to,to);
        v[now][0][0]+=max(dp[to][0],dp[to][1]);
        v[now][1][0]+=dp[to][0];
    }
    v[now][0][1]=v[now][0][0];
}
```



```
int x=read<int>(),y=read<int>();
v[x][1][0]+=y-w[x]; // 修改 x 的值为 y
w[x]=y;
while(x)
{
    node xx=ask(1,1,n,dfn[top[x]],tail[top[x]]); // 修改前链顶的值
    update(1,1,n,dfn[x],x); // 在线段树上修改
    node yy=ask(1,1,n,dfn[top[x]],tail[top[x]]); // 修改后链顶的值
    x=fa[top[x]]; // 跳到链顶的父节点
    v[x][0][0]+=max(yy[0][0],yy[1][0])-max(xx[0][0],xx[1][0]);
    v[x][0][1]=v[x].a[0][0];
    v[x][1][0]+=yy[0][0]-xx[0][0]; // 根据修改前后的差值,更新节点信息
}
node ans=ask(1,1,n,1,tail[1]);
printf("%d\n",max(ans[0][0],ans[1][0]));
```

动态DP

- NOIP2018 提高组 保卫王国
- 给定一棵 n 个点的树, 点带点权。
- 有 m 次询问, 每次询问指定两个点选/不选。
- 对于每次询问, 回答这棵树的最小权覆盖集的权值大小。
- $1 \leq n, m \leq 1e5$

动态DP

- 若钦定的两个点都不选，且它们相邻，无解。
- 对于钦定的点，若选，则权值 $+\text{inf}$ ，若不选，则权值 $-\text{inf}$ 。
- 最后处理答案时需要把 inf 产生的影响变回去。
- 方法和上一题类似。

动态DP

- 定义 $f[i][0/1]$ 表示节点 i 不选/选时对应子树的最小权覆盖集。
- 定义 $g[i][0]$ 表示节点 i 的所有轻儿子可选可不选的最小权覆盖集。
- 定义 $g[i][1]$ 表示节点 i 选上且它的所有轻儿子都选的最小权覆盖集。
- $f[i][0] = g[i][1] + f[j][1]$
- $f[i][1] = g[i][0] + \min(f[j][0], f[j][1])$
- 定义 $(+, \min)$ 形式的广义矩阵乘法，可以构造出矩阵形式的转移：

$$\begin{bmatrix} \infty & g[i][1] \\ g[i][0] & g[i][0] \end{bmatrix} * \begin{bmatrix} f[j][0] \\ f[j][1] \end{bmatrix} = \begin{bmatrix} f[i][0] \\ f[i][1] \end{bmatrix}$$

动态DP

- CSP-S2022 数据传输
- 给定一棵 n 个点的树，点带点权。
- 有 q 次询问，每次询问给定两个点 x 和 y 。
- 对于每次询问，寻找一条从 x 出发到 y 结束的路径，每次可以移动不超过 k 条边。
- 求路径上经过点的点权之和的最小值。
- $1 \leq n, q \leq 2e5, 1 \leq k \leq 3$

动态DP

- 把 x 到 y 的简单路径拿出来，设为 t_1, t_2, \dots, t_l 。
- 定义路径上第 i 个点的点权为 v_i 。
- 可以证明，当 $k \leq 2$ 时，最优方案一定只经过路径上的点。
- 当 $k=1$ 时： $f[i] = f[i-1] + v[i]$
- 当 $k=2$ 时： $f[i] = \min(f[i-1], f[i-2]) + v[i]$

动态DP

- 当 $k=3$ 时，有可能经过路径之外的点了。
- 设 $\text{minx}[i]$ 表示到第 i 个点距离为 1 的最小点权。
- 设 $f[i][0/1/2]$ 表示到路径上第 i 个点距离为 0/1/2 条边的最小值。
- $f[i][0] = \min(f[i-1][0], f[i-1][1], f[i-1][2]) + v[i]$
- $f[i][1] = \min(\min(f[i-1][0], f[i-1][1]) + \text{minx}[i], f[i][0] + \text{minx}[i], f[i-1][0])$
- $f[i][2] = f[i-1][1]$

动态DP

- 定义 $(+, \min)$ 形式的广义矩阵乘法，可以构造出矩阵形式的转移：

$$\begin{bmatrix} v[i] & v[i] & v[i] \\ 0 & \text{minx}[i] & \text{minx}[i] + v[i] \\ \infty & 0 & \infty \end{bmatrix} * \begin{bmatrix} f[i-1][0] \\ f[i-1][1] \\ f[i-1][2] \end{bmatrix} = \begin{bmatrix} f[i][0] \\ f[i][1] \\ f[i][2] \end{bmatrix}$$

- 因为不涉及修改，所以可以通过树上倍增等方式记录转移矩阵的乘积。
- 分别求出 $x \sim \text{lca}$ 和 $\text{lca} \sim y$ 两部分的答案再合并即可。

```
void dfs(int now,int f)
{
    fa[now][0]=f,dep[now]=dep[f]+1;
    bz[now][0][0][0]=bz[now][0][0][1]=bz[now][0][0][2]=v[f];
    bz[now][0][1][1]=minx[f],bz[now][0][1][0]=0,bz[now][0][1][2]=minx[f]+v[f];
    bz[now][0][2][0]=bz[now][0][2][2]=inf,bz[now][0][2][1]=0;
    for(int i=1;dep[now]-(1<<i)>0;++i)
    {
        fa[now][i]=fa[fa[now][i-1]][i-1];
        bz[now][i]=bz[fa[now][i-1]][i-1]*bz[now][i-1];
    }
    for(int to:e[now])
    {
        if(to==f)continue;
        dfs(to,now);
    }
}
```

```
int aread<int>(),bread<int>();
if(dep[a]<dep[b])swap(a,b);
int lca=lCA(a,b);
if(lca==b)
{
    node ans;
    ans[0][0]=v[a],ans[1][0]=ans[2][0]=inf;
    for(int i=17;-1;--i)if(dep[fa[a][i]]>dep[b])ans=bz[a][i]*ans,a=fa[a][i];
    printf("%lld\n",ans[0][0]);
}
else
{
    node ansa,ansb;
    ansa[0][0]=v[a],ansa[1][0]=ansa[2][0]=inf;
    ansb[0][0]=v[b],ansb[1][0]=ansb[2][0]=inf;
    for(int i=17;-1;--i)
    {
        if(dep[fa[a][i]]>dep[lca])ansa=bz[a][i]*ansa,a=fa[a][i];
        if(dep[fa[b][i]]>dep[lca])ansb=bz[b][i]*ansb,b=fa[b][i];
    }
    long long Ans=ansa[0][0]+ansb[0][0]-v[lca];
    for(int i=0;i<3;++i)
    for(int j=0;j<3;++j)
    {
        if(!i&&!j)continue;
        if(i+j>3)continue;
        Ans=min(Ans,ansa[i][0]+ansb[j][0]);
    }
    printf("%lld\n",Ans);
}
```

矩阵在信息学中的典型应用

北京师范大学附属实验中学 胡伟栋

矩阵的定义

- 将 $n \times m$ 个数排列成 n 行 m 列的形式称为一个矩阵。

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, C = \begin{bmatrix} 9 & 9 & 8 \\ 2 & 4 & 4 \\ 3 & 5 & 3 \\ 9 & 9 & 8 \end{bmatrix}$$

- 矩阵一般用大写字母 A, B, C 表示，矩阵中的一个元素用小写字母 a_{ij}, b_{ij}, c_{ij} 表示。
 - C 是一个 4 行 3 列的矩阵， $c_{11} = 9, c_{32} = 5$
- 行数与列数相等矩阵称为方阵



北京师范大学附属实验中学

矩阵的典型运算

- 矩阵乘法
 - $C = AB$
 - $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j} + \dots$
 - 当 A 的列数等于 B 的行数时才能相乘
 - $p \times q$ 的矩阵乘以 $q \times r$ 的矩阵结果为 $p \times r$ 的矩阵



北京师范大学附属实验中学

矩阵的基本意义

- 矩阵用于简化表示多个数量之间的关联关系，典型的包括

- 方程组

- $\begin{bmatrix} 2 & 3 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 20 \\ 3 \end{bmatrix}$

- 代表的方程：

- $\begin{cases} 2x_1 + 3x_2 = 20 \\ x_1 - 2x_2 = 3 \end{cases}$

- 递推关系



北京师范大学附属实验中学
BEIJING NORMAL UNIVERSITY EXPERIMENTAL MIDDLE SCHOOL

矩阵的基本意义

- 矩阵用于简化表示多个数量之间的关联关系，典型的包括

- 方程组

- 递推关系

- $\begin{bmatrix} a_{i1} \\ a_{i2} \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & -5 \end{bmatrix} \begin{bmatrix} a_{i-1,1} \\ a_{i-1,2} \end{bmatrix}$

- 代表的递推关系：

- $a_{i1} = 2a_{i-1,1} + 3a_{i-1,2}$

- $a_{i2} = 4a_{i-1,1} - 5a_{i-1,2}$



北京师范大学附属实验中学
BEIJING NORMAL UNIVERSITY EXPERIMENTAL MIDDLE SCHOOL

理解矩阵乘法

$$\bullet \begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & -5 \end{bmatrix} \begin{bmatrix} r \\ s \end{bmatrix}$$

$$\bullet \begin{bmatrix} r \\ s \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\bullet \begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & -5 \end{bmatrix} \begin{bmatrix} r \\ s \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & -5 \end{bmatrix} \left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \right)$$

$$\bullet \begin{bmatrix} p \\ q \end{bmatrix} = \left(\begin{bmatrix} 2 & 3 \\ 4 & -5 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$



北京师范大学附属实验中学
THE EXPERIMENTAL SCHOOL AFFILIATE TO BEIJING NORMAL UNIVERSITY

矩阵乘法

• 重要性质：结合律

• 特别注意：不一定满足交换律



北京师范大学附属实验中学
THE EXPERIMENTAL SCHOOL AFFILIATE TO BEIJING NORMAL UNIVERSITY

单位矩阵

- 单位矩阵

$$\bullet I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



北京师范大学附属实验中学
BEIJING NORMAL UNIVERSITY EXPERIMENTAL MIDDLE SCHOOL

矩阵的幂

- 矩阵幂的定义

- 对于一个方阵 A ，记 $A^n = A \cdot A \cdot A \cdots A$ 表示 n 个 A 连乘的结果，称为 A 的 n 次幂。
- $A^n = A^{n-1} \cdot A$
- 特别地： $A^0 = I$



北京师范大学附属实验中学
BEIJING NORMAL UNIVERSITY EXPERIMENTAL MIDDLE SCHOOL



矩阵快速幂

- 矩阵的幂和整数的幂类似，可以使用分治思想快速实现

中国计算机学会



北京师范大学附属实验中学
BEIJING NORMAL UNIVERSITY AFFILIATE EXPERIMENTAL MIDDLE SCHOOL

78

例：斐波那契数列

- 对于斐波那契数列：
 - $F_1 = F_2 = 1$
 - $F_i = F_{i-1} + F_{i-2}$
- 求 $F_n \bmod M$



北京师范大学附属实验中学
BEIJING NORMAL UNIVERSITY AFFILIATE EXPERIMENTAL MIDDLE SCHOOL

11

解析

- 对于递推关系

- $F_i = F_{i-1} + F_{i-2}$

- 构造递推矩阵

- $\begin{bmatrix} F_i \\ F_{i-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{i-1} \\ F_{i-2} \end{bmatrix}$

- 令 $A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, 则

- $\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = A \begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix} = A^2 \begin{bmatrix} F_{n-2} \\ F_{n-3} \end{bmatrix} = A^3 \begin{bmatrix} F_{n-3} \\ F_{n-4} \end{bmatrix} = \dots = A^{n-2} \begin{bmatrix} F_2 \\ F_1 \end{bmatrix}$

- 用快速幂求 A^{n-2} 即可



北京师范大学附属实验中学
THE EXPERIMENTAL SCHOOL AFFILIATED TO BEIJING NORMAL UNIVERSITY

例：复杂一点的递推

- 对于递推关系：

- $F_1 = F_2 = F_3 = 1$

- $F_i = 5F_{i-1} - 3F_{i-3}$

- 求 $F_n \bmod M$



北京师范大学附属实验中学
THE EXPERIMENTAL SCHOOL AFFILIATED TO BEIJING NORMAL UNIVERSITY

70

解析

- 对于递推关系

$$F_i = 5F_{i-1} - 3F_{i-3}$$

- 构造递推矩阵

$$\begin{bmatrix} F_i \\ F_{i-1} \\ F_{i-2} \end{bmatrix} = \begin{bmatrix} 5 & 0 & -3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} F_{i-1} \\ F_{i-2} \\ F_{i-3} \end{bmatrix}$$

$$\text{令 } A = \begin{bmatrix} 5 & 0 & -3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \text{ 则}$$

$$\begin{bmatrix} F_n \\ F_{n-1} \\ F_{n-2} \end{bmatrix} = A \begin{bmatrix} F_{n-1} \\ F_{n-2} \\ F_{n-3} \end{bmatrix} = A^2 \begin{bmatrix} F_{n-2} \\ F_{n-3} \\ F_{n-4} \end{bmatrix} = \dots = A^{n-2} \begin{bmatrix} F_2 \\ F_1 \\ F_0 \end{bmatrix}$$

- 用快速幂求 A^{n-2} 即可



北京师范大学附属实验中学
BEIJING NORMAL UNIVERSITY EXPERIMENTAL MIDDLE SCHOOL

例：带常数的递推

- 对于递推关系：

$$F_1 = F_2 = F_3 = 1$$

$$F_i = 5F_{i-1} - 3F_{i-3} + 7$$

- 求 $F_n \bmod M$



北京师范大学附属实验中学
BEIJING NORMAL UNIVERSITY EXPERIMENTAL MIDDLE SCHOOL



25

解析

- 对于递推关系

$$F_i = 3F_{i-1} - 5F_{i-2} + 7$$

- 构造递推矩阵

$$\begin{bmatrix} F_i \\ F_{i-1} \\ F_{i-2} \\ 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 & -5 & 7 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} F_{i-1} \\ F_{i-2} \\ F_{i-3} \\ 1 \end{bmatrix}$$

$$\text{令 } A = \begin{bmatrix} 3 & 0 & -5 & 7 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{ 则}$$

$$\begin{bmatrix} F_n \\ F_{n-1} \\ F_{n-2} \\ 1 \end{bmatrix} = A \begin{bmatrix} F_{n-1} \\ F_{n-2} \\ F_{n-3} \\ 1 \end{bmatrix} = A^2 \begin{bmatrix} F_{n-2} \\ F_{n-3} \\ F_{n-4} \\ 1 \end{bmatrix} = \dots = A^{n-2} \begin{bmatrix} F_2 \\ F_1 \\ F_0 \\ 1 \end{bmatrix}$$

- 用快速幂求 A^{n-2} 即可



北京师范大学附属实验中学
THE EXPERIMENTAL MIDDLE SCHOOL AFFILIATE TO BEIJING NORMAL UNIVERSITY



例：多个数列递推

- 对于递推关系：

$$a_1 = b_1 = a_2 = b_2 = 1$$

$$a_i = 2a_{i-1} + 3b_{i-1}$$

$$b_i = a_{i-1} - 5b_{i-2}$$

- 求 $a_n \bmod M$



北京师范大学附属实验中学
THE EXPERIMENTAL MIDDLE SCHOOL AFFILIATE TO BEIJING NORMAL UNIVERSITY



解析

- 对于递推关系

- $$x_i = 2x_{i-1} + 3x_{i-2}$$

$$y_i = x_{i-1} - 5x_{i-2}$$

- 构造递推矩阵

- $$\begin{bmatrix} x_i \\ x_{i-1} \\ y_i \\ y_{i-1} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 0 & 0 \\ 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ x_{i-2} \\ y_{i-1} \\ y_{i-2} \end{bmatrix}$$

- $$\text{令 } A = \begin{bmatrix} 2 & 3 & 0 & 0 \\ 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

- $$\begin{bmatrix} x_i \\ x_{i-1} \\ y_i \\ y_{i-1} \end{bmatrix} = A \begin{bmatrix} x_{i-1} \\ x_{i-2} \\ y_{i-1} \\ y_{i-2} \end{bmatrix} \Rightarrow \begin{bmatrix} x_i \\ x_{i-1} \\ y_i \\ y_{i-1} \end{bmatrix} = A^{i-1} \begin{bmatrix} x_1 \\ x_0 \\ y_1 \\ y_0 \end{bmatrix}$$

- 用快速幂求 A^{i-1} 即可



北京师范大学附属实验中学

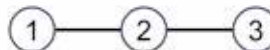
例：图上递推计数[TJOI2017] 可乐

- 问题简述

- 给一个包含 n 个点（不超过30）， m 条边（不超过100）的无向图
- 有一个可乐机器人在图上移动，第0秒在1号点，每秒可能存在三种行为
 - 停在当前点
 - 移动到图上的相邻点
 - 自爆，结束
- 问经过 t 秒（不超过 10^6 ），机器人的行动路径有多少种可能

- 例

- 下图两秒内有8种可能



北京师范大学附属实验中学

解析

- 递推
 - 状态表示：用 S_{ij} 表示时刻 i 到达结点 j 的方案数，用 B_{ij} 表示时刻 i 到在结点 j 爆炸的方案数，用 Z_i 表示时刻 i 爆炸的方案数
 - $S_{ij} = S_{i-1,j} + \sum_{(k,j) \in E} S_{i-1,k}$
 - $B_{ij} = S_{i-1,j}$
 - $Z_i = \sum B_{ij} = \sum S_{i-1,j}$
- 直接递推即可的满分
- 转换成矩阵可以解决！很大的情形



北京师范大学附属实验中学
BEIJING NORMAL UNIVERSITY EXPERIMENTAL MIDDLE SCHOOL

矩阵用于序列的动态维护



北京师范大学附属实验中学
BEIJING NORMAL UNIVERSITY EXPERIMENTAL MIDDLE SCHOOL

21

例：整数变换

- 对于两个整数 A, B ，利用一些指令对整数进行变换
 - 指令U
 - 将 A 变为两数之和， B 不变
 - 指令D
 - 将 B 变为两数之和， A 不变
- 给定一个确定的变换序列 S
- 对于 Q 个询问
 - 问对于某一对给定的 (A, B) ，经过 S 中从 L 到第 R 个指令的变换后，变成了什么数



北京师范大学附属实验中学
THE EXPERIMENTAL MIDDLE SCHOOL OF BEIJING NORMAL UNIVERSITY

解析

- 构造变换矩阵，用 A_i, B_i 表示 A, B 在时刻 i 的值
 - 指令U
 - 将 A 变为两数之和， B 不变
 - $$\begin{bmatrix} A_i \\ B_i \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} A_{i-1} \\ B_{i-1} \end{bmatrix}$$
 - 指令D
 - 将 B 变为两数之和， A 不变
 - $$\begin{bmatrix} A_i \\ B_i \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} A_{i-1} \\ B_{i-1} \end{bmatrix}$$



北京师范大学附属实验中学
THE EXPERIMENTAL MIDDLE SCHOOL OF BEIJING NORMAL UNIVERSITY

解析

- 指令序列和矩阵序列的关系：
 - 例UDDD
 - $$\begin{bmatrix} A_i \\ B_i \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} A_{i-4} \\ B_{i-4} \end{bmatrix}$$
- 算法：使用线段树维护矩阵序列
 - (现场演示)



北京师范大学附属实验中学

BEIJING NORMAL UNIVERSITY AFFILIATED EXPERIMENTAL SCHOOL

例：线段树练习

- 输入两个长为n的数列a[i], b[i]。维护m个操作，操作分3类：
 - (1) 对于其中连续一段执行：a[i]赋值为a[i]+b[i]
 - (2) 对于其中连续一段执行：b[i]赋值为a[i]*b[i]
 - (3) 对于其中连续一段求a[i]b[i]的和。mod 10301输出



北京师范大学附属实验中学

BEIJING NORMAL UNIVERSITY AFFILIATED EXPERIMENTAL SCHOOL

解析

- (现场推导)

中国计算机学会



北京师范大学附属实验中学
BEIJING NORMAL UNIVERSITY EXPERIMENTAL MIDDLE SCHOOL

36

矩阵用于方程组

- 方程组的矩阵表示

- 方程组：
$$\begin{cases} 2x_2 - x_3 = 7 \\ 2x_1 + 2x_2 - 4x_3 = 2 \\ 2x_1 + x_2 + x_3 = 12 \end{cases}$$

- 矩阵表示：
$$\begin{bmatrix} 0 & 2 & -1 \\ 2 & 2 & -4 \\ 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 2 \\ 12 \end{bmatrix}$$

- 增广矩阵表示：
$$\left[\begin{array}{ccc|c} 0 & 2 & -1 & 7 \\ 2 & 2 & -4 & 2 \\ 2 & 1 & 1 & 12 \end{array} \right]$$



北京师范大学附属实验中学
BEIJING NORMAL UNIVERSITY EXPERIMENTAL MIDDLE SCHOOL



高斯消元法

• 方程组：
$$\begin{cases} 2x_2 - x_3 = 7 \\ 2x_1 + 2x_2 - 4x_3 = 2 \\ 2x_1 + x_2 + x_3 = 12 \end{cases}$$
，增广矩阵表示：
$$\left[\begin{array}{ccc|c} 0 & 2 & -1 & 7 \\ 2 & 2 & -4 & 2 \\ 2 & 1 & 1 & 12 \end{array} \right]$$

• 基本操作

- 1. 交换两个方程的顺序，不改变方程组
- 2. 将某个方程左右两边乘以一个非零常数，不改变方程组
- 3. 将某个方程左右两边乘以一个常数，加到另一个方程两边，不改变方程组



北京师范大学附属实验中学
THE EXPERIMENTAL MIDDLE SCHOOL AFFILIATE TO BEIJING NORMAL UNIVERSITY

高斯消元法：算法

• 方程组：
$$\begin{cases} 2x_2 - x_3 = 7 \\ 2x_1 + 2x_2 - 4x_3 = 2 \\ 2x_1 + x_2 + x_3 = 12 \end{cases}$$
，增广矩阵表示：
$$\left[\begin{array}{ccc|c} 0 & 2 & -1 & 7 \\ 2 & 2 & -4 & 2 \\ 2 & 1 & 1 & 12 \end{array} \right]$$

• (现场演示)



北京师范大学附属实验中学
THE EXPERIMENTAL MIDDLE SCHOOL AFFILIATE TO BEIJING NORMAL UNIVERSITY

29

例：[JSOI2008] 球形空间产生器

- 有一个球形空间产生器能够在 n 维空间中产生一个坚硬的球体。
- 现在，你被困在了这个 n 维球体中，你只知道球面上 $n+1$ 个点的坐标，你需要以最快的速度确定这个 n 维球体的球心坐标，以便于摧毁这个球形空间产生器。
- 输入格式
 - 第一行是一个整数 n ($1 \leq n \leq 10$)，接下来的 $n+1$ 行，每行有 n 个实数，表示球面上一点的 n 维坐标，每一个实数精确到小数点后 6 位，且其绝对值都不超过 20000。



北京师范大学附属实验中学

THE EXPERIMENTAL MIDDLE SCHOOL AFFILIATED TO BEIJING NORMAL UNIVERSITY

解析

- (现场推导)



北京师范大学附属实验中学

THE EXPERIMENTAL MIDDLE SCHOOL AFFILIATED TO BEIJING NORMAL UNIVERSITY

31



小结

- 矩阵的定义
- 矩阵的基本运算
- 矩阵在递推中的典型应用
- 矩阵用于解方程（高斯消元法）



大纲中关于矩阵的内容

- 提高级
 - 【5】向量与矩阵的概念
 - 【6】向量的运算
 - 【6】矩阵的初等变换
 - 【6】矩阵的运算：加法、减法、乘法与转置
 - 【6】特殊矩阵的概念：单位阵、三角阵、对称阵和稀疏矩阵
 - 【7】高斯消元法





大纲中关于矩阵的内容

- NOI级

- 【9】逆矩阵
- 【9】行列式
- 【9】向量空间与线性相关



信息学竞赛中的字符串问题选讲

浙江省余姚中学 诸一行

DFA(确定有限状态自动机)

- 什么是DFA？
- 一个 DFA(Deterministic Finite Automaton) 可以用来判断输入的字符串是否符合该DFA 设定的条件。一个 DFA 由以下几部分构成：
 - 1、状态集 Q (必须是有限的集合)：如果把一个 DFA 看成一张有向图，那么 DFA 中的状态就相当于图上的顶点。
 - 2、字符集 Σ ：该自动机只能输入这些字符。
 - 3、可接受状态集 F ： $F \subseteq Q$ ，是一组特殊的状态。
 - 4、起始状态 q_0 ： $q_0 \in Q$ ，是一个特殊的状态。
 - 5、转移函数 $\delta: Q \times \Sigma \rightarrow Q$ ，是一个接受两个参数返回一个值的函数，其中第一个参数和返回值都是一个状态，第二个参数是字符集中的一个字符。如果把一个 DFA 看成一张有向图，那么 DFA 中的转移函数就相当于顶点间的边，而每条边上都有一个字符。

DFA(确定有限状态自动机)

- DFA 的作用就是识别字符串，那么一个 DFA 如何处理输入的串呢？
- 一开始，状态位于起始状态 q_0 ，接下来，将待判断的串的每个字符依次输入。
- 每输入一个字符 c ，就将当前状态从 q 移动至 $\delta(q, c)$ 。
- 若将给定串的所有字符输入完毕后，最终所在的状态状态 $q_{\text{end}} \in F$ ，则输入的串是可接受串，否则就不是。

构建简单的DFA——

例1、判断一个二进制数字字符串是不是偶数

- 要求构建一个DFA，识别并判断一个二进制整数字符串，该字符串所对应的二进制数是否为偶数。
- 分析：
- 判断一个二进制数是否为偶数，只要判断其最后一位二进制数是否为0。
- 构造一个DFA，最最重要的就是构造状态。对于已经输入的串 T ，状态 q 用来描述 $\{x | T + x \text{ 是可接受串}\}$ ，即在 T 后面跟些什么东西能让串合法。

构建简单的DFA——

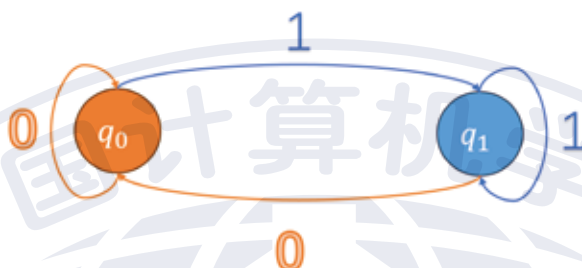
例1、判断一个二进制数字字符串是不是偶数

- 要求构建一个DFA，识别并判断一个二进制整数字符串，该字符串所对应的二进制数是否为偶数。
- 构建DFA：
- 1、状态集 Q （必须是有限的集合）： $\{q_0, q_1\}$ ，表示末位是0，还是1。
- 2、字符集 Σ ： $\{0, 1\}$ ，字符串仅有字符‘0’，‘1’组成。
- 3、可接受状态集 F ： $\{q_0\}$ ，末位是0。
- 4、起始状态 q_0 ： $\{q_0\}$ 。
- 5、转移函数 $\delta : Q \times \Sigma \rightarrow Q$ ，每个状态都有两个字符集的转移

构建简单的DFA——

例1、判断一个二进制数字字符串是不是偶数

- 要求构建一个DFA，识别并判断一个二进制整数字符串，该字符串所对应的二进制数是否为偶数。
- 构建DFA：



- 从起始结点开始，从高到低接受这个数的二进制序列，然后看最终停在哪里。如果最终停在橙色结点，则是偶数，否则不是。

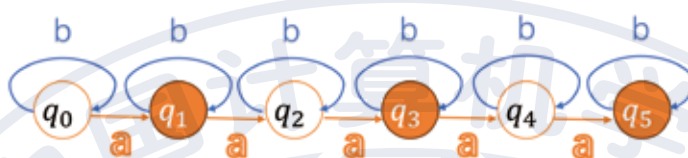
构建简单的DFA——

例2、指定个数的ab串

- 构造一个 DFA，使得它接受且仅接受只含字符 ab，且字符 a 的数量恰好为 1, 3 或 5 的字符串。
- 分析：
- 构造一个 DFA，最最重要的就是构造状态。对于已经输入的串 T ，状态 q 用来描述 $\{x | T + x \text{ 是可接受串}\}$ ，即在 T 后面跟些什么东西能让串合法。此题的关键是正式字符 a 的数量。
- 设状态 $q =$ 已经出现过的字符 a 的数量 $(0 \leq q \leq 5)$
- 在 T 后面跟上一个只含字符 ab
- 若输入字符 a，则走向 $q + 1$ ，若输入字符 b，则留在原地 q ，若输入 ab 以外的字符，则丢弃，另外，若 $q > 5$ ，也丢弃。

构建简单的DFA—— 例2、指定个数的ab串

- 构造一个 DFA，使得它接受且仅接受只含字符 ab，且字符 a 的数量恰好为1,3 或 5 的字符串。

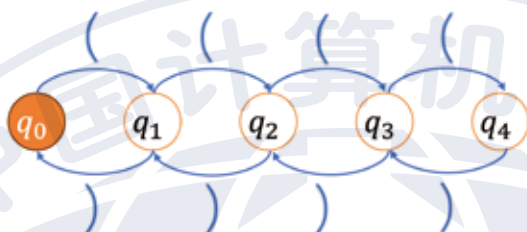


构建简单的DFA—— 例3、合法的括号序列

- 构造一个 DFA，使得它接受且仅接受符合以下条件的字符串：
 - 1) 是个合法的括号序列；
 - 2) 括号嵌套的最深层数不超过 4
- 分析：
 - 一个合法的括号序列可以用栈来维护
 - 括号嵌套深度不超过 4，可以用栈深度控制，也就是DFA中的状态集数量。
 - 检验一个序列是否合法，也就是可接受状态。
 - 接收到左括号还是右括号，就可以用转移函数在状态间移动。

构建简单的DFA—— 例3、合法的括号序列

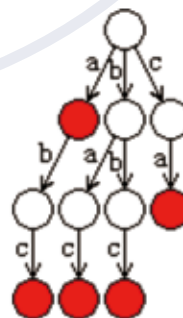
- 构造一个 DFA，使得它接受且仅接受符合以下条件的字符串：
- 1) 是个合法的括号序列；
- 2) 括号嵌套的最深层数不超过 4



- 思考：能否构造出一个 DFA，使得它接受且仅接受合法的括号序列呢？

Trie自动机

- Trie树就是一个像字典一样的树。
- 例如：要存储五个模式串：a,abc,bac,bbc,ca。
- 右图中，红色表示单词终止的位置。



- 相信大家都接触过Trie树,
- 现在我们就用自动机的形式来描述Trie树。

Trie自动机

- Trie自动机：接受且仅接受指定的字符串集合中的元素。
- 1、状态集 Q ：建树过程中产生的结点总数，不会超过 $\sum |T| + 1$
- 2、字符集 Σ ：字符串中出现的可能字符集合。
- 3、可接受状态集 F ：每个字符串的结束状态结点。
- 4、起始状态 q_0 ：root状态结点。
- 5、转移函数 δ ：Trie 上的边，我们用 $\delta(u, c)$ 表示结点 u 的 c 字符指向的下一个结点，或着说是结点 u 代表的字符串后面添加一个字符 c 形成的字符串的结点。（ c 的取值范围和字符集大小有关，不一定是 $0 \sim 25$ 。）

Trie树的实现

```
1 struct trie {
2     int nxt[100000][26], cnt;
3     bool exist[100000]; // 该结点结尾的字符串是否存在
4
5     void insert(char *s, int l) { // 插入字符串
6         int p = 0;
7         for (int i = 0; i < l; i++) {
8             int c = s[i] - 'a';
9             if (!nxt[p][c]) nxt[p][c] = ++cnt; // 如果没有，就添加结点
10            p = nxt[p][c];
11        }
12        exist[p] = 1;
13    }
14
15    bool find(char *s, int l) { // 查找字符串
16        int p = 0;
17        for (int i = 0; i < l; i++) {
18            int c = s[i] - 'a';
19            if (!nxt[p][c]) return 0;
20            p = nxt[p][c];
21        }
22        return exist[p];
23    }
24 }
```

Trie自动机的应用

例1：Phone List (POJ3630, Nordic 2007)

- 题目大意：
- 给你 n 个电话号码，判断存不存在某个电话号码是另外一个电话号码的前缀。
- 分析：
- 根据电话号码，构建一个Trie树，看是否所有可接受状态都是叶子结点。如果存在非叶子结点，那么就存在一个串是另一个串的前缀。
- 前缀问题一般都用Trie树解决。

Trie自动机的应用

例2：The xor-longest Path (POJ 3764)

- 题目大意：
- 给你一棵带边权的树，求 (u, v) 使得 u 到 v 的路径上的边权异或和最大，输出这个最大值。点数不超过 10^5 ，边权在 $[0, 2^{31})$ 内。
- 分析：
- 利用树上LCA，随便指定一个根 $root$ ，用 $dis(u, v)$ 表示 u 和 v 之间的路径的边权异或和，那么 $dis(u, v) = dis(root, u) \oplus dis(root, v)$ ，因为LCA以上的部分异或两次抵消了。
- 问题就转化成所有 $dis(root, u)$ 中寻找两个异或值最大的。(The XOR Largest Pair)

Trie自动机的应用

例2：The xor-longest Path (POJ 3764)

- 问题转化：
- 在给定的 N 个整数 A_1, A_2, \dots, A_N 中选出两个进行异或运算，得到的结果最大是多少？
- 将每个数按照31位二进制数理解，要让某个 A_i 与另一个 A_j 的异或值最大，就要高位尽可能不同。
- 将所有 A_i 按二进制数从高位开始依次插入一棵Trie树，对每个 A_i 在Trie树中直接找与其每一位01尽可能相反的数。
- 时间复杂度： $O(N \cdot 31)$

Trie自动机的应用

例3：【IOI2008】Type Printer

- 题目大意：
- 一个打印机，支持向其加减最后一个字母和打印的操作。给出 N 个单词，求打印这些单词的最小操作数。注意，打印结束后允许打印机内有字母，且打印次序任意。 $(1 \leq N \leq 25000)$ ，单词均小写，最大长度20
- 分析：
- 由于打印操作是无法省去的，操作数最小就要在加减字母考虑。显然若两单词有**相同前缀**，则可以通过共用前缀的方式解决。这样可减少一次加减共两个操作。同时，由于最后一个单词打印后可不删除，因此最后一个单词要尽量长。

Trie自动机的应用

例3：【IOI2008】Type Printer

- 分析：
- 根据相同前缀这一条件，我们联想到Trie树，故使用Trie维护单词。
- 可以证明，本题答案就是Trie上边的个数 $\times 2$ （加，减各一次）+ 单词数（用于打印）- 最长单词长度（不用删除）。
- 打印方案：采用将最后打印的单词的节点最后遍历，其他的先遍历，在遍历到单词的结束标记时打印的方式。

KMP自动机

什么是KMP自动机？

- 用线性时间在字符串 T 中查找模式串 S 。
- Knuth–Morris–Pratt 算法：该算法由 Knuth、Pratt 和 Morris 在 1977 年共同发布。相信大家已经学过了。。。
- 如果用自动机的思想去理解 KMP，KMP 就相当于一个 DFA。
- 它接受一个字符串 T 当且仅当给定的模式串 S 是 T 的后缀。
- 由于 T 是一个字符一个字符输入的，KMP 算法就可以判断 S 是否是 T 的每个前缀的后缀，即找出 S 在 T 中每一次出现的位置。比如说， $S = ABC, T = ABCABCD$ ，KMP 算法可以找出 S 在 T 中的出现位置是 $1 \dots 3$ 与 $4 \dots 6$ 。

KMP自动机

KMP算法的自动机构造

- 显然，我们需要对模式串 S 构建自动机，想必大家也都已经知道KMP算法，自然也知道最终DFA的状态数。那我们还是根据现在的需求，重新探索一番。
- 首先明确，我们要构建的DFA功能：
- 设目前已经输入的串是 T ，那么若存在一个 x 使得 $T[|T| - x + 1 \dots |T|] = S[1 \dots x]$ ，那么在 T 后接着输入 $S[x + 1 \dots |S|]$ ，即可到达一个可接受串。
- 对于一个字符串 T ，设集合 $right(T) = \{x | T[|T| - x + 1 \dots |T|] = S[1 \dots x]\}$ 。
- 例如，若 $S = abaabaabaab$, $T = cabaab$ ，那么 $right(T) = \{0, 2, 5\}$ ，这说明， $T + \text{任意串} + S$, $T + S[3 \dots 11]$, $T + S[6 \dots 11]$ 都是可接受串。
- 对于一个已经输入的串 T ，需要关心的有且仅有 $right(T)$ ，那么，我们设 $q = right(T)$ 。

KMP自动机

KMP算法的自动机构造

- 现在考虑怎么转移。
- $\delta(q, c) = \{0\} \cup \{x + 1 | x \in q \text{ 且 } x < |S| \text{ 且 } S[x + 1] = c\}$ 。
- 这个式子来直观理解一下。
- $x + 1 \in \delta(q, c)$ ，当且仅当原来的 T 的末 x 个字符与 $S[1 \dots x]$ 相同，且 $c = S[x + 1]$ 。
- 当然，空串永远等于空串，0 永远属于 q 。另外，初始状态 $q_0 = \{0\}$ 。

KMP自动机

KMP算法的自动机构造

- 这样子的状态数是多少呢？
- 首先，可以直观地看到， q 是一个 $\{0, 1, \dots, |S|\}$ 的子集。
- 那么， $|Q|$ 难道是 $O(2^{|S|})$ 级别的吗？
- 显然，没有这么夸张。我们注意到，若 $x \in q$ ，那么 $0 \leq i < x$ 的所有整数是否属于 q 都是确定的（只取决于 S ），那么，只要知道了 q 中的最大元素，整个 q 都是确定的。
- 现在，对于 q ，我们只需关心其最大元素，而 q 中的元素一定在 0 到 $|S|$ 之间，那么，状态数就成功降到了 $|S| + 1$ 。
- 接下来，设 $q' = \max\{x | x \in q\}$ 。

KMP自动机

KMP算法的自动机构造

- $q = \{x | S[1 \dots x] = S[q' - x + 1 \dots q']\}$ 。
- 设 $fail[q'] = \max\{x | x \in q \text{ 且 } x < q'\}$ ，那么， $q = \{q', fail[q'], fail[fail[q']], \dots, 0\}$ 。
- 转移已经十分明显！
- $\delta(q', c) = \max(0, \max\{x + 1 | x \in q \text{ 且 } x < |S| \text{ 且 } S[x + 1] = c\})$
- $$= \begin{cases} q' + 1 & (q' < |S| \text{ 且 } S[q' + 1] = c) \\ 0 & (q' = 0 \text{ 且 } S[1] \neq c) \\ \delta(fail[q'], c) & \text{其他情况} \end{cases}$$

KMP自动机

KMP算法的自动机构造

- $fail[i] = \begin{cases} 0 & (i = 1) \\ \delta(fail[i-1], S[i]) & (i > 1) \end{cases}$
- δ 和 $fail$ 都是可以在 $O(|S|)$ 的时间内递推求出，这样，一个 $|S| + 1$ 个状态的 KMP 算法的 DFA 就构造完了。

KMP自动机

KMP算法的自动机构造

```

1 //s: 模式串
2 //t: 文本串
3 //n: 文本串长度
4 //m: 模式串长度
5 //fail: 存放部分失效匹配的数组
6 //cnt: 记录模式串出现的次数
7 void computePartialMatchTable(){
8     fail[1]=0;
9     for(int q = 2, k = 0; q <= m; q++){
10         while(k && s[k+1] != s[q]) k = fail[k];
11         if(s[k+1]==s[q]) k++;
12         fail[q] = k;
13     }
14 }

15 void kmp(){
16     cnt = 0;
17     computePartialMatchTable();
18     for(int i = 1, q = 0; i <= n; i++){
19         while(q && s[q+1]!=t[i]) q = fail[q];
20         if(s[q+1]==t[i]) q++;
21         if(q==m){
22             cnt++;
23             q=fail[q];
24         }
25     }
26 }
    
```

KMP自动机的应用

例1：Radio Transmission (BOI2009)

- 题目大意：
- 给你一个字符串 s_1 ，它是由某个字符串 s_2 不断自我连接形成的（保证至少重复 2 次），求 s_2 最小长度
- 分析
- 先进行kmp，答案为 $n - fail[n]$ 。
- 设答案为 ans ，由于 $s_1[fail[n] + 1 \dots n]$ 一定是一个 s_2 ，故 $ans \leq n - fail[n]$ 。
- 又如果 $ans < n - fail[n]$ ，那么根据 $fail[n]$ 定义，我们取 $fail[n] = n - ans$ 会更优，矛盾。
- 故 $ans = n - fail[n]$ 。

KMP自动机的应用

例2：Censoring S (USACO15FEB)

- 题目大意：
- 给定一个字符串 S 和一个模式串 T ，我们需要在 S 中删去 T ，删完之后 S 会将剩余部分按照原顺序合并，有可能会产生新的匹配串，现在问删完后 S 会变什么样？ $|S| \leq 10^6$
- 分析
- 构造KMP，容易发现，在匹配过程中，匹配到的位置与数组位置对应，我们只需要知道匹配到第 i 位所对应的数组位置就可以知道之前匹配到的结果。
- 我们只需要用栈维护匹配的数组位置，删除时就弹出栈顶元素。
- 时间复杂度 $O(n)$ 。

KMP自动机的应用

例3：SZA-Template (POI2005)

- 题目大意：
- 给你一个长度为 n 的字符串 S ，求印章长度的最小值，这个印章可以重复印刷同一个位置，但是每次必须恰好把所有字母都印到纸上。 $n \leq 5 \times 10^5$ 。
- 分析
- 令 $f[i]$ 表示 $S[1 \dots i]$ 的最小印章长度，进行kmp
- 证明： $f[i] = i$ 或 $f[fail[i]]$
- 显然 $f[i] \leq i$
- 如果 $f[i]$ 不能覆盖 $S[1 \dots fail[i]]$ ，那么显然它也无法覆盖 $S[1..i]$ ，故 $f[i] \geq f[fail[i]]$
- 考虑什么时候取等，显然，对于 $S[i - fail[i] + 1..n]$ ，我们可以用 $f[fail[i]]$ 覆盖，剩下 $S[1..i - fail[i]]$ 部分，我们也希望可以覆盖，而且可以重叠，可以通过判断是否存在 $f[j] = f[fail[i]]$ ，其中 $j \geq i - fail[i]$

KMP自动机的应用

例3：SZA-Template (POI2005)

- 题目大意：
- 给你一个长度为 n 的字符串 S ，求印章长度的最小值，这个印章可以重复印刷同一个位置，但是每次必须恰好把所有字母都印到纸上。 $n \leq 5 \times 10^5$ 。
- 分析
- 若存在，则 $f[i] = f[fail[i]]$ ；反之，假设 $f[i] < i$
- 根据 $fail$ 数组的定义，又 $S[1..fail[i]]$ 能被 $f[fail[i]]$ 表示，故 $f[i] \leq f[fail[i]]$
- 故此时 $f[i] = f[fail[i]]$ ，矛盾，假设不成立
- 综上，此时 $f[i] = i$

AC自动机

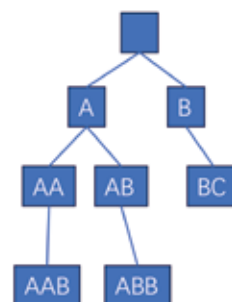
什么是AC自动机？

- KMP 算法，它给定了一个模式串 S ，接受一个字符串 T 当且仅当 S 是 T 的后缀。
- AC 自动机给定一个字符串集合 $S = \{S_1, S_2, \dots, S_N\}$ ，接受一个字符串 T 当且仅当 $\exists S_i \in S, S_i$ 是 T 的后缀。
- KMP 是 AC 自动机在 $|S| = 1$ 时的简化版本。
- AC 自动机相当于在 Trie 树构建 KMP 自动机

AC自动机

AC自动机的构造

- 如果有 (a, b) 满足 $T[|T| - b + 1 \dots |T|] = S_a[1 \dots b]$ ，那么 T 之后再添上 $S_a[b + 1 \dots |S_a|]$ 即可使得 S_a 是 T 的后缀，到达一个可接受状态。举个例子，若 $S = \{AAB, ABB, BC\}$ 。
- 设 $H = \{S_a[1 \dots b] \mid 1 \leq a \leq K, 0 \leq b \leq |S_a|\}$ ，代表所有可能成为某个模式串的前缀的字符串的集合，
- 本例子中 $H = \{\emptyset, A, AA, AAB, AB, ABB, B, BC\}$ 。
- 设 $q = \text{right}(T) = \{s \mid s \text{ 是 } T \text{ 的后缀且 } s \in H\}$ ，这是我们需要关心的。假设当前 $T = CAAB$ ，那么 $q = \{\emptyset, B, AB, AAB\}$ 。如果在 T 的末尾添上一个 B ，变成 $CAABB$ ，那么此时转移 $\delta(q, B) = \{\emptyset, B, ABB\}$ 。



AC自动机

AC自动机的构造

- 接下来我们要优化AC自动机的状态数。
- 类似于KMP，我们设 q' 代表 q 中的最长的字符串。
- 若 q' 确定，整个 q 都确定。
- $q = \{x | x \text{ 是 } q' \text{ 的后缀且 } x \in H\}$ 。
- 类似于KMP，我们定义一个 $fail[q']$ ，代表 q 中最长的字符串是 q' 时， q 中第二长的字符串。
- 那么有 $q = \{q', fail[q'], fail[fail[q']], \dots, \emptyset\}$ 。

AC自动机

AC自动机的构造

- 设 $fa[i]$ 代表字符串 i 右边一个字符删去后得到的字符串，如 $fa[ABCD] = ABC$ 。类似于KMP，我们只需用 q' 来代表整个状态，有了转移：
- $$\delta(q', c) = \begin{cases} q' + c & (q' + c \in H) \\ \emptyset & (q' = \emptyset \text{ 且 } c \notin H) \\ \delta(fail[q'], c) & \text{其他情况} \end{cases}$$
- $$fail[q'] = \begin{cases} \emptyset & (|q'| = 1) \\ \delta(fail[fa[q']], q'[|q'|]) & (|q'| > 1) \end{cases}$$
- 另外， $q' \in F$ 当且仅当 $q' \in S$ 或 $fail[q'] \in F$ 。正如KMP的 δ 与 $fail$ 可以递推得到，AC自动机的 δ 与 $fail$ 也可以在 H 形成的 trie 树上 BFS 得到，时间复杂度 $O(|H| |\Sigma|)$ ，其中 $|H| \leq 1 + \sum_{i=1}^K |S_i|$

AC自动机

AC自动机的构造

```
1 void build() {  
2     for(int i = 0; i < 26; i++)  
3         if (tr[0][i]) q.push(tr[0][i]);  
4     while (q.size()) {  
5         int u = q.front();  
6         q.pop();  
7         for (int i = 0; i < 26; i++) {  
8             if (tr[u][i])  
9                 fail[tr[u][i]] = tr[fail[u]][i], q.push(tr[u][i]);  
10            else  
11                tr[u][i] = tr[fail[u]][i];  
12        }  
13    }  
14 }
```

AC自动机应用

例1：Keywords Search (HDU 2222)

- 题目大意
- 给定一个字符串 T 与 K 个字符串 S_1, S_2, \dots, S_k ，问 S_1, S_2, \dots, S_k 分别在 T 中以 T 的子串出现了多少次。 $(1 \leq |T| \leq 10^6, \sum_{i=1}^K |S_i| \leq 10^5)$ 。
- 建出 $\{S_1, S_2, \dots, S_k\}$ 的AC自动机，然后将 T 逐个字符输入。 $fail$ 形成了一个树形结构， q 中含有元素 s 当且仅当 q' 位于元素 s 在 $fail$ 树上的子树内。 T 输入的过程中每个节点开个计数器，然后为 $fail$ 树上的每个子树求和。
- 时间复杂度 $O(|T| + \sum_{i=1}^K |S_i| |\Sigma|)$ 。

AC自动机应用

例2：病毒（POI2000）

- 题目大意
- 已知某些特定的01串是病毒的特征代码。如果一个01串不含有任何病毒特征代码，则称它为一段安全代码。给定病毒特征库，判断是否存在无限长的安全代码。
【输入】 第一行为一个整数 n ，表示病毒特征代码的条数。下面 n 行，每行一段病毒特征代码。所有代码长度之和不超过30000。
【输出】 若存在无限长的安全代码，输出一行“TAK”，否则输出一行“NIE”。

AC自动机应用

例2：病毒（POI2000）

- 分析
- “无限长”的安全代码是什么意思呢？
- 就是说从根结点出发，在安全图中可以走无限步。
- “无限步”又是什么意思呢？
- 就是说安全图中有环。因此我们建立一个AC自动机并对其安全图进行拓扑排序，若成功，则安全图无环，输出“NIE”，否则输出“TAK”。

AC自动机应用

例3：Censoring G (USACO15 FEB)

- 题目大意
- 给定一个字符串 S 和若干模式串 T_i ，我们需要在 S 中删去 T_i ，删完之后 S 会将剩余部分按照原顺序合并，有可能会产生新的匹配串，现在问删完后 S 会变什么样。 $|S| \leq 10^5, \sum |T_i| \leq 10^5$ 。
- 分析
- 构造AC自动机，容易发现，在匹配过程中，匹配到的位置与结点对应，我们只需要知道匹配到第 i 位所对应的结点就可以知道之前匹配到的结果
- 我们只需要用栈维护匹配的结点，删除时就弹出栈顶元素。
- 时间复杂度 $O(n)$ 。

AC自动机应用

例4：阿狸的打字机 (NOI 2011)

- 题目大意
- 给定一颗trie树，多组询问，每次询问两个结点 x, y ，求 x 代表的字符串在 y 中出现的次数
- 分析
- 构造AC自动机，在fail指针树上， x 的子树中的点代表的字符串都是 x 的后缀。
- 我们可以通过判断有多少后缀为 x 的不同字符串为 y 的前缀来求出次数。
- 故而我们只需要求出 x 在fail指针树的子树中有多少点是 y 的祖先。
- 问题就变成了链上修改、子树查询——每次我们先将 y 到根上 $+1$ ，再对 x 子树求和，最后将 y 到根上 -1 。树剖可以在 $O(\text{nl}g^2n)$ 实现
- 该题还可以离线解决，我们按照询问离散到 y 上，接着按照trie树的构建顺序维护上述的01序列，接着对询问继续求和，每个位置只会从0变成1，从1变成0一次，故总修改次数为 $O(n)$ ，我们用树状数组维护，总时间复杂度为 $O(\text{nl}gn)$ 。

SAM(后缀自动机)

什么是SAM?

- SAM(Suffix Automaton) 是一个 DFA, 给定一个字符串 S , 该 DFA 接受且仅接受 S 的后缀。
- 转移图是有向无环图, 且每个转移接受且只接受一个字符
- 在符合以上性质的基础上, 节点数最少。
- 如果没有节点数最少的限制?

SAM(后缀自动机)

SAM的构造

- 不管对于什么自动机, 它的状态都是用来体现出, 对于已输入的串 T , T 的后面接着输入哪些串后是可接受串, 输入哪些串后不是可接受串。
- 若 x 满足 $T = S[x - |T| + 1 \dots x]$, 那么 $T + S[x + 1 \dots |S|]$ 就是一个可接受串, 否则就不是。
- 类似 KMP, 我们需要关心的有且仅有集合 $right(T) = \{x | T = S[x - |T| + 1 \dots x]\}$ 。
- 类似 KMP, 看上去, 不同的 $right(T)$ 有 $O(2^{|S|})$ 种。
- 思考一下, 注意到, 对于任何可能出现的两个 $right(T_1)$ 和 $right(T_2)$, 它们要么无交, 要么一个包含另一个。
- 既然所有可能的 $right$ 集合有如上的性质, 那么所有可能的 $right$ 集合就可以形成一个根节点是 $\{0, 1, 2, \dots, |S|\}$ 的树形结构, 所有可能的 $right$ 集合数量 (即 $|F|$) 最多是 $2^{|S|} + 1$ 。

SAM(后缀自动机)

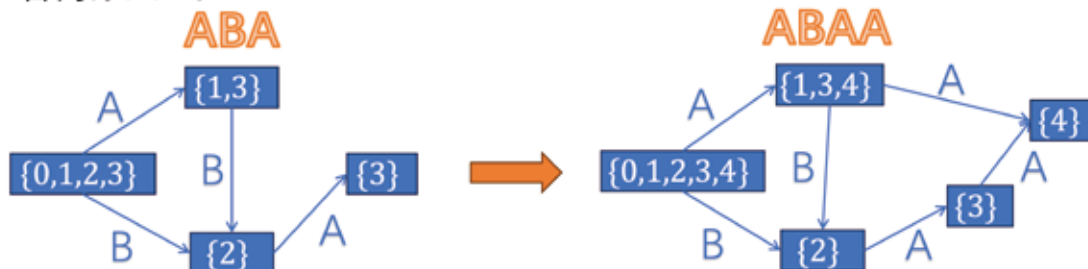
SAM的构造

- 所有 *right* 集合形成了树形结构，然而我们手头只有 S 这个串，并不知道这个树形结构长啥样，更不知道所有 *right* 集合分别是什么。
- 直接构造 SAM 很困难，不过，有一点还是可以知道的。方便起见令 $q = \text{right}(T)$ ，则有
- $\delta(q, c) = \{x + 1 | x \in q \text{ 且 } x < |S| \text{ 且 } S[x + 1] = c\}$
- 另外， $q_0 = \{0, 1, 2, \dots, |S|\}$ ， $q \in F$ 当且仅当 $|S| \in q$ 。

SAM(后缀自动机)

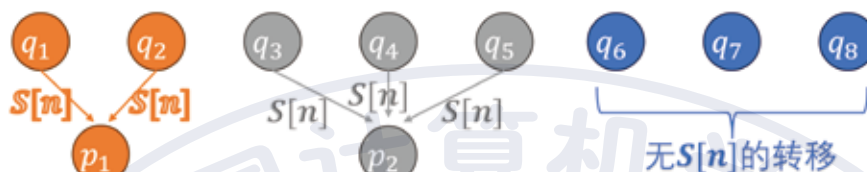
SAM的构造

- 如果我们已经得到了 $S[1 \dots n - 1]$ 对应的 SAM，如何通过一些微小的改变，得到 $S[1 \dots n]$ 对应的 SAM 呢？
- 在原先的 S 末尾加入一个字符 $S[n]$ 后，SAM 发生的变化有哪些？对于原先 *right* 集合中含有 $n - 1$ 的状态 q ，在 $\delta(q, S[n])$ 的 *right* 集合内添入 n 。



SAM(后缀自动机) SAM的构造

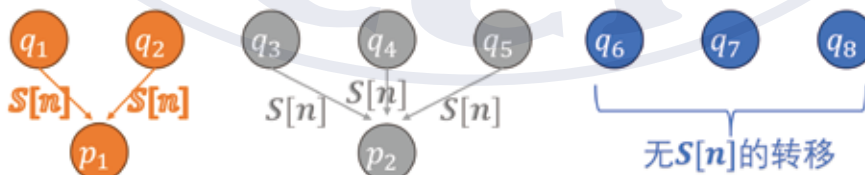
- 更一般的情况。



- 如上图, $q_{1...8}$ 是所有包含 $n - 1$ 的状态。
- $\{n - 1\} = q_8 \subset q_7 \subset \dots \subset q_1 = \{0, 1, \dots, n - 1\}$ 。
- 由 $\forall q_a \subset q_b, c \in \Sigma, \delta(q_a, c) \subseteq \delta(q_b, c)$ 可知 $p_2 \subset p_1$ 。

SAM(后缀自动机) SAM的构造

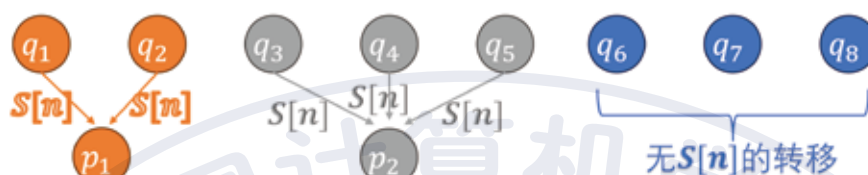
- 更一般的情况。



- 显然, 应当新建一个状态 $q_{new} = \{n\}$, 并让 $\delta(q_{6,7,8}, S[n]) = q_{new}$ 。
- 另外, p_1, p_2 的 *right* 集合内要加入 n 。不过有一个特殊情况: 如果存在一个状态 q_s 使得 $\delta(q_s, S[n]) = p_2$ 且 $n - 1 \notin q_s$, p_2 应当被分裂成两个点, 一个是原来不含 n 的 p_2 , 一个是加入了 n 之后的 p_2 , 记作 p_{split} 。
- 令 $\delta(q_{3,4,5}, S[n]) = p_{split}, \delta(q_s, S[n]) = p_2$ 。

SAM(后缀自动机) SAM的构造

- 更一般的情况。



- 这个例子中，虽然 p_1 的 $right$ 集合内添了 n ，但不存在不含 $n - 1$ 的状态 q_s 满足 $\delta(q_s, S[n]) = p_1$ ，所以 p_1 不用分裂，和 p_1 有关的图的结构没有发生变化。
- 这样， S 的长度每增加 1，我们都可以正确地处理出 DFA 的形态变化。那从一个空的字符串开始不断增加，即可构建出任意一个字符串的后缀自动机。

SAM(后缀自动机) SAM的构造

- 刚才，我们已经将 SAM 的增量法构造的基本流程讲完了，现在是实现的问题。
- 刚才那个比较抽象的例子里， $q_8 = \{n - 1\}$ ，当当前串长为 n 时，我们将状态 $\{n\}$ 的点号记下，并且对所有状态编号 i 记下 $fail[i]$ ，代表集合包含状态 i 的最小的集合的点号。
- 那样， $q_7 = fail[q_8]$ ， $q_6 = fail[q_7]$ ，...，就可以把 $q_{1...8}$ 全都列出来。
- 接着，按照 $\delta(q, S[n])$ 把 $q_{1...8}$ 分成若干段，相同的为一段。
- 一开始一定有一段满足 $\delta(q, S[n]) = \emptyset$ ，因为 $\delta(\{n - 1\}, S[n]) = \emptyset$ ，这些在 S 末尾添加一个字符后都会指向 $\{n\}$ 。

SAM(后缀自动机) SAM的构造

- 接下来，找出 $q_1 \dots q_8$ 中最小的字符 $S[n]$ 的转移不是空集的状态（例子中为 q_5 ）。
- 怎样判断是否存在 q_s 使得 $q_s \subset q_5$ 且 $\delta(q_s, S[n]) = p_2$ 呢？
- 在每个状态 i 处记下 $\text{maxlen}[i]$ ，表示能够走到状态 i 的最长的字符串 T 的长度。能够走到状态 i 的所有字符串 T ，右端点为 i 的 right 集合中的任意一个元素，长度位于区间 $(\text{maxlen}[\text{fail}[i]], \text{maxlen}[i])$ 。
- 若 $\text{maxlen}[q_5] + 1 < \text{maxlen}[p_2]$ ，则这样的 q_s 存在， p_2 需要分裂。
- fail 与 maxlen 都可以方便维护。

SAM(后缀自动机) SAM的构造

```

1 // 每加入一个字符ch调用extend(ch)
2 void extend(int ch)
3 {
4     int cur=siz++;
5     st[cur].maxlen=st[lst].maxlen+1;
6     int p=lst;
7     while(p!=-1&&!st[p].delta[ch]){
8         st[p].delta[ch]=cur;
9         p=st[p].fail;
10    }
11    if(p==-1){
12        st[cur].fail=0;
13    }
14    else{
15        int q=st[p].delta[ch];
16        if(st[p].maxlen+1==st[q].maxlen){
17            st[cur].fail=q;
18        }
19        else{
20            int pp=siz++;
21            st[pp].maxlen=st[p].maxlen+1;
22            memcpy(st[pp].delta, st[q].delta, sizeof(st[pp].delta));
23            st[pp].fail=st[q].fail;
24            while(p!=-1&&st[p].delta[ch]==q){
25                st[p].delta[ch]=pp;
26                p=st[p].fail;
27            }
28            st[q].fail=st[cur].fail=pp;
29        }
30    }
31    lst=cur;
32 }

```

SAM的应用

例1：不同子串个数（luogu P2408）

- 给定一个字符串 S ，问它有多少个本质不同的非空子串。
- 分析
- 首先，造出 S 的 SAM。将一个字符串 T 输入该 SAM 后，状态没有被丢弃当且仅当 T 是 S 的子串（否则 $right$ 集合就空了）。
- 那么，为每个 SAM 上的状态分别统计能够走到该状态的本质不同的字符串的数量。
- 更简洁的方法就是，答案 = $\sum_{i \in Q \text{ 且 } i \neq q_0} \maxlen[i] - maxlen[fail[i]]$ 。

SAM的应用

例2：回文串（APIO2014）

- 题目大意
- 给定长为 n 的串 S ，求 S 的所有回文子串，长度乘出现次数的最大值。
- $1 \leq n \leq 3 \times 10^5$
- 分析
- 考虑回文串——倒着和正着都一样，设 S' 为字符串倒过来形成的字符串。
- 则如果 $S[l \dots r]$ 是回文串，则有 $S[l \dots r] = S[l \dots r]' = S'[l' \dots r']$ 其中 l', r' 表示 l, r 倒过来后应该在的位置。
- 这是一个将 S' 与 S 进行子串匹配的过程。
- 对 S 从前往后构造后缀自动机。

SAM的应用

例2：回文串（APIO2014）

- 在后缀自动机上，我们只需要可以找到所有本质不同的回文串就可以计算出最后的答案。
- 本质不同的回文串数量为 $O(n)$ 。
- 法一、manacher找回文串，后在后缀自动机上用倍增快速锁定回文串所在的结点，时间复杂度为 $O(n \lg n)$

SAM的应用

例2：回文串（APIO2014）

- 法二、我们将 S 从后往前依次在后缀自动机中遍历，假设当前已经遍历了 $S[i+1..n]'$ ，下一个是 $S[i]$ ，
- 我们可以在后缀自动机中找到与 S 匹配的最大子串所在的结点，假设极大长度为 l ，即满足 $S[i..i+l-1]'$ 在 S 中出现，而 $S[i..i+l]'$ 并未出现
- 由回文串的性质不难发现，如果 $S[x..y]' = S[p..q]$ ，其中 $p \leq x \leq q$ ，则 $S[x..q]$ 为回文串
- 故我们只需要判断 $right$ 集合中是否有介于 i 和 $i+l'-1$ (l' 为结点字符串长度) 之间的元素 y ，就可以快速找到回文串 $S[i..y]$ ，可以通过找到所在结点后跳 $fail$ 树进行遍历。
- 又要求本质不同，我们每个结点只需要计算一次，所以我们只需要对 $right$ 集合中的最大值进行判断，并且在遍历的时候，每个点只需要遍历一遍，遇到遍历过的可以直接结束，时间复杂度为均摊 $O(n)$

SAM的应用

例3：你的名字（NOI2018）

- 题目大意
- 给定长为 n 的串 S ， q 次询问，每次问一个子串 $S[l, r]$ 以及另一个串 T ，求 T 有几个本质不同的子串，未在 $S[l, r]$ 中出现过。
- $1 \leq l \leq r \leq n \leq 5 \times 10^5, 1 \leq q \leq 10^5, 1 \leq |T| \leq 5 \times 10^5, \sum |T| \leq 10^6$ 。
- 分析
- 考虑对于 T 的每个前缀，求出在 $S[l, r]$ 中出现过的最长后缀。这样只需对 T 的反串求 SA 或后缀树，即可求出“部分不计入的本质不同子串数”。
- 如果 $l = 1, r = n$ 则可以建出 S 的 SAM，然后逐个扫描 T 的字符，维护当前最长后缀（记录所在节点 u 与目前长度 now ）。对于目前字符 T_i ，如果 u 有 T_i 的出边则走，并使 l 加一；否则 u 跳至其后缀连接，赋 now 为新的 u 对应的最长串长，反复进行直到找到出边或 $now = 0$ 。

SAM的应用

例3：你的名字（NOI2018）

- 如果 $l = 1$ ，可以离线后建 SAM 边回答询问，但 l 也会变时就不行了。
- 当 l, r 不固定时，如果用同样方法，这时得到的最长后缀可能在 $S[l, r]$ 中不存在。设点 u 对应的结束位置集合为 R_u ，最短串长度为 t 。若 $\exists i \in R_u$ ，满足 $l - t + 1 \leq i \leq r$ ，则当前节点中存在匹配上的串，但 now 要与 $\max i - l + 1$ 取 \min 。否则 u 需要跳后缀连接。
- 为找到 $\max_{i \in R_u, l-t+1 \leq i \leq r} i$ ，可以使用线段树合并求出每个点的 R 集合，也可以将问题抽象为“所有前缀 $S[1, i]$ 满足 $i \leq r$ ，且对应点在 parent 树上的 dfs 序在一段区间内，这样的 i 的最大值”，用可持久化线段树或离线扫描 r 维护线段树即可求解。
- 暴力跳后缀连接的时间复杂度可以均摊证明。
- 总复杂度 $O((n + \sum |T|) \log n)$

ST (后缀树)

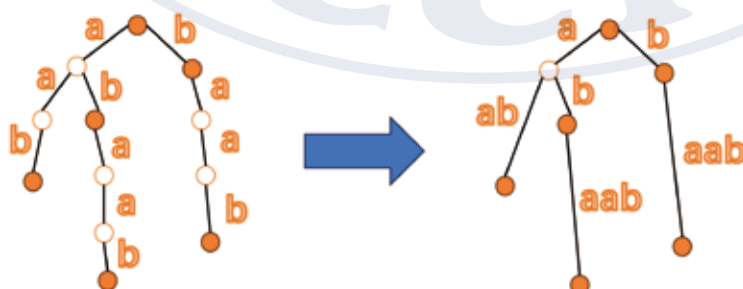
什么是后缀树

- 接下来，我们要讲后缀树 (Suffix Tree, ST) 了。
- 后缀树是什么？
- 就是给定一个长度为 N 的字符串 S ，将 S 的每一个后缀列出来，建成一个 trie 树。显然，后缀的长度和是 $O(N^2)$ 的，那么后缀树的节点数也是 $O(N^2)$ 的。
- 幸好有一个性质。将只有一个儿子，且本身不是某个后缀的结束位置的节点和其儿子合并后，剩下的树的节点个数是 $O(N)$ 的。

ST (后缀树)

什么是后缀树

- 我们来举一个后缀树的例子。以下是串 abaab 的后缀树：



- 其中橙色点为某个后缀的结束位置的节点。
- 为什么缩完以后节点个数是 $O(N)$ 呢？接下来会证明。

ST (后缀树)

后缀树的构造

- 如果 T 是 S 的子串, T 一定出现在 S 的后缀树的某个节点上。
- 如果 T 的反串在 S 的反串的 SAM 中的 $right$ 集合为 $\{a, b, c, d, e\}$, 那么, T 在 S 的后缀树上的节点的子树中, 存在着 $S[N - a + 1 \dots N], S[N - b + 1 \dots N], S[N - c + 1 \dots N], S[N - d + 1 \dots N], S[N - e + 1 \dots N]$ 这 5 个后缀的终止节点。
- T 在 S 的后缀树中的节点的子树中的终止节点集合与 T 的反串在 S 的反串的 SAM 中的 $right$ 集合一一对应。
- S 的后缀树中, 子树中的终止节点集合相同的节点, 都被缩起来了!
- 所以, 后缀树的节点树与其反串的后缀自动机的状态数相同。
- 既然反串的不同 $right$ 集合数量 $O(N)$, 那么后缀树的节点数量也是 $O(N)$ 。

ST (后缀树)

后缀树的构造

- 后缀树的构造方法, 在刚刚节点数量 $O(N)$ 的证明中已经透露出来了。
- 结合以下事实, 请同学们自觉思考后缀树的构造方法:
- 子树中终止节点集合相同的节点, 都被缩起来啦!
- 一个节点的子树中终止节点集合与反串的 SAM 中的 $right$ 集合一一对应。
- SAM 中, 一个节点的 $fail$ 节点为最小的包含它的 $right$ 集合。
- 缩完的后缀树中, 一个节点的父亲是……
- 建出反串的后缀自动机, 反串的后缀自动机上的 $fail$ 形成的树, 和原串的后缀树的形态一模一样!

后缀树的应用

例1：品酒大会（NOI2015）

- 题目大意
- 给定长为 n 的串 S ，序列 a_1, \dots, a_n 。对于 $r = 0, \dots, n-1$ ，求 $i \neq j, |LCP(S[i, n], S[j, n])| \geq r$ 的有序对 (i, j) 数量，以及这些 (i, j) 对应的 $a_i \cdot a_j$ 最大值。
- $1 \leq n \leq 3 \times 10^5, |a_i| \leq 10^9$ 。

后缀树的应用

例1：品酒大会（NOI2015）

- 建立后缀树用于刻画各后缀间 LCP 的结构：两个后缀的 LCP 即为它们在后缀树上对应点的 LCA。因此只需对于每个点，统计所有以它为 LCA 的后缀数以及对应的 a 乘积 \max 即可。求得信息会贡献给答案的一段前缀。
- 为求出这些信息，对于每个点，统计其子树内有几个后缀，以及这些后缀对应的 a 的最小、最大值即可。
- 根据建后缀树方法不同，时间复杂度为 $O(n \log n)$ 、 $O(n|\Sigma|)$ 或 $O(n)$ 。



浅谈树拓扑序计数相关问题的一些方法

杭州第二中学 陈昕阳

前言	带多元钦定的叶向树拓扑序计数	逆向树形背包合并	更精巧的容斥原理	致谢
●	○○○○○	○○○○○○○	○○○○○○○○○○○	○

前言

叶向树拓扑序计数问题是算法竞赛中的经典问题，在此之上也已有许多拓展问题与方法，我在我的集训队论文中对此进行了一些总结和研究。

接下来将介绍三种拓展问题以及解决的方法，由于时间限制，默认大家都对叶向树拓扑序计数问题有基本了解。

杭州第二中学 陈昕阳
浅谈树拓扑序计数相关问题的一些方法

前言	带多元钦定的叶向树拓扑序计数	逆向树形背包合并	更精巧的容斥原理	致谢
○	●○○○○	○○○○○○○	○○○○○○○○○○○	○

问题

给定一棵 n 个点以 1 为根的叶向树和一个常数 k 。你需要对所有长度为 k 的序列对 a, b 满足 a 递增, b 互不相同且 a, b 值域均为 $[1, n]$, 算出有多少长度为 n 排列 p 满足

$\forall 1 \leq i \leq k, p_{a_i} = b_i$, 且对叶向树的每条边 $u \rightarrow v$, $p_u < p_v$ 。答案模 998244353。

$$n^{2k} \leq 3 \times 10^7, k \leq 4。$$

杭州第二中学 陈昕阳
浅谈树拓扑序计数相关问题的一些方法



考虑以下针对给定叶向树的过程：

- ① 每个点有黑或白的颜色，初始所有点都是白色，另有一个长度为 n 的辅助序列 a 。
- ② 进行 n 轮操作，每轮选一个白色点（之前选过的点也可以再选），找到其最浅的白色点祖先 p ，将其涂黑。假设这是第 i 轮，记录 $a_i = p$ 。

可以发现，无论怎么选每一轮的白色点，最终得到的序列 a 都是原叶向树的一个拓扑序。但一个拓扑序可能对应多种 n 轮每轮选择一个白色点的方案。



看上去算重了，但其实可以观察到，当 a 序列是一个拓扑序时，会生成其的选择方案永远是 $\prod_{i=1}^n siz_i$ 种。因为拓扑序第 i 位是 u ，代表过程中第 i 轮被涂黑的点是 u ，在 u 父亲已经被涂黑的前提下，这充要于选择的白色点 v 在 u 子树内，这有 siz_u 种方案，而不同轮独立可以直接相乘。

所以钦点 k 个点在拓扑序上的位置可以转化为在染色问题中钦点它们分别应当在哪一轮被涂黑。

这样转化的好处在于如果只关心 k 个点在哪一轮被涂黑，只需要知道这 k 个点中每个点，最浅的白色祖先是哪个即可。



前言 ○ 带多元钦定的叶向树拓补序计数 ○○○●○ 逆向树形背包合并 ○○○○○○ 更精巧的容斥原理 ○○○○○○○○○ 致谢 ○

DP 实现

设 $dp_{i,S,T}$ 表示已经完成了前 i 轮涂色，在这 i 轮中作出限制 $T = \{(a_j, b_j)\}$ ，点 a_j 必须在第 b_j 轮被涂黑，而 S 集合是所有尚未被涂黑的带有钦定的点祖先中首个白点构成的集合，这样的涂色方案数。

转移考虑第 i 轮涂黑了哪个点，可以发现被涂黑的点要么是 S 集合中某个点 p 的父亲，那么前驱状态就是向 S 集合加入 fa_p 并删去 fa_p 所有孩子；要么被涂黑的点在 T 中被记录；要么第 i 轮操作前 S 集合和第 i 轮操作后一模一样。

只保留 $|T| + |S| \leq k$ 的 DP 状态，并且对于 $|T| = k$ 的状态直接结算答案不再继续转移，即可做到 $O(n^{2k} poly(k))$ 的复杂度。

杭州第二中学 陈昕阳
浅谈树拓补序计数相关问题的一些方法

前言 ○ 带多元钦定的叶向树拓补序计数 ○○○●○ 逆向树形背包合并 ○○○○○○ 更精巧的容斥原理 ○○○○○○○○○ 致谢 ○

总结

事实上在 k 远小于 n 时这个做法差不多是达到了复杂度下界的，因为要计算 $\binom{n}{k} n^k$ 个信息，而消耗的复杂度则是

$$O(n^{2k} poly(k))$$

不过这个做法在 $k > 1$ 时实现起来可能比较痛苦。

前言	带多元钦定的叶向树拓扑序计数	逆向树形背包合并	更精巧的容斥原理	致谢
○	○○○○○	●○○○○○	○○○○○○○○○	○

方法的引入

树上问题，自下至上 DP，进行背包合并，这是非常常见的做法。往往能说明暴力背包合并的复杂度整体就是 $O(n^2)$ 的，与只合并一次背包的复杂度相同因此不会成为瓶颈。

但有时不得不自上至下 DP，然后需要做逆向树形背包合并（背包分裂？），此时复杂度就可能出现一些问题。

杭州第二中学 陈昕阳				
浅谈树拓扑序计数相关问题的一些方法				

前言	带多元钦定的叶向树拓扑序计数	逆向树形背包合并	更精巧的容斥原理	致谢
○	○○○○○	○●○○○○○	○○○○○○○○○	○

例题

给定一棵 n 个点，以 1 为根的叶向树。对所有 $1 \leq u < v \leq n$ 求出 $f(u, v)$ 表示所有合法拓扑序 a 的 $|a_u - a_v|$ 之和。合法拓扑序的定义是所有是长度为 n 的排列满足 $\forall 2 \leq u \leq n, a_u > a_{fa_u}$ 。答案模 998244353。
 $n \leq 500$ 。

杭州第二中学 陈昕阳				
浅谈树拓扑序计数相关问题的一些方法				

前言

带多元钦定的叶向树拓扑序计数

逆向树形背包合并

更精巧的容斥原理

致谢

初步考虑解法

首先 $|x - y|$ 类贡献考虑枚举值域线拆成 $(\sum_V [x \leq V][V < y]) + (\sum_V [y \leq V][V < x])$ 。把所有 $a_u \leq V$ 的点涂成白色，剩下的涂成黑色。考虑枚举一种黑白染色方案，那么不能有一条边 $u \rightarrow v$ 满足 u 黑色而 v 白色。只要满足这些边的限制，设一共涂了 k 个黑色点，该染色方案对应合法排列 a 的数量就是 $\frac{k!(n-k)!}{\prod_{i=1}^n \text{siz}'_i}$ ，其中 siz'_u 是 u 子树内与 u 同色点的数量。设这个对应排列数为该染色方案权值。

那么相当于需要对所有 $p \neq q$ 算出 p 为白色 q 为黑色的所有染色方案的权值和。

杭州第二中学 陈昕阳

浅谈树拓扑序计数相关问题的一些方法

前言

带多元钦定的叶向树拓扑序计数

逆向树形背包合并

更精巧的容斥原理

致谢

初步考虑解法

直接枚举 q ， $O(n^2)$ DP 算出所有 $g_{u,i}$ 表示对 u 子树内所有点染色，一共有 i 个白色点，并且 q 点如果在子树内则必须为黑色的所有染色方案权值和。

接着可以考虑自上至下地 DP，设 $dp_{u,i}$ 表示对 u 子树外所有点染色，如果 q 在子树外则必须为黑色，已知 u 子树内有 i 个白色点的所有染色方案权值和。

这样如果能 $O(n^2)$ 求出 dp ，钦定 p 白色， q 黑色的所有方案权值和就是 $\sum_{i=1}^{\text{siz}_p} dp_{p,i} g_{p,i}$ 。

杭州第二中学 陈昕阳

浅谈树拓扑序计数相关问题的一些方法

前言

带多元钦定的叶向树拓扑序计数

逆向树形背包合并

更精巧的容斥原理

致谢

初步考虑解法

考虑如何求出 dp ，首先 $\forall 1 \leq i \leq n, dp_{1,i} = i!(n-i)!$ 。对于 $u > 1$ ，首先要去掉 fa_u 的影响，会导致 i 的项以 $\frac{1}{i}$ 的系数移到 $i-1$ ，这样得到 $dp'_{fa_u,*}$ 。接着应该是将 $dp'_{fa_u,*}$ 在第二维上与 fa_u 除 u 以外的其他孩子 $v_1 \sim v_k$ 的 $g_{v_i,*}$ 逐个进行减法卷积，因为从 fa_u 换到 u ， u 的其他孩子也由子树内变为子树外。

此时就需要进行上文提到的逆向树形背包合并，怎么做复杂度才正确呢？

杭州第二中学 陈昕阳

浅谈树拓扑序计数相关问题的一些方法

前言

带多元钦定的叶向树拓扑序计数

逆向树形背包合并

更精巧的容斥原理

致谢

逆向树形背包合并的正确做法

这相当于给出 k 个多项式 $f_1 \sim f_k$ 以及大多项式 F ， $\deg(F) = \sum_{i=1}^k \deg(f_i)$ ，要对所有 $1 \leq i \leq k$ 算出 $subconv(F, \prod_{j \neq i} f_j)$ 的 $0 \sim \deg(f_i)$ 次项系数， $subconv(P, Q)$ 表示对 P 和 Q 执行减法卷积。

考虑对 $1 \leq i \leq k$ 算出 $prd_i = \prod_{j=i}^k f_j$ ，那么对于 $i=1$ 结果就是 $subconv(F, prd_2)$ 。接下来要计算的所有信息都需要除去 f_1 ，所以可以直接令 $F \leftarrow subconv(F, f_1)$ 然后不管 f_1 这一项，这时只需保留 F 的 $0 \sim \sum_{j=2}^k \deg(f_j)$ 项系数即可。接下来 $i=2$ 的结果就是 $subconv(F, prd_3)$ ，以此类推。

杭州第二中学 陈昕阳

浅谈树拓扑序计数相关问题的一些方法



这样做的时间复杂度为 $O(\deg^2(F) - \sum_{i=1}^k \deg^2(f_i))$ ，根据树形背包的时间复杂度分析整体是 $O(n^2)$ 的。

所以例题可以 $O(n^3)$ 解决。

事实上从转置原理的角度来看，逆向树形背包合并这样的解决方法是非常自然的，因为从上至下 DP 可以看作是对经典的从下至上 DP 进行了转置，这里我们描述的过程就是转置了从下至上 DP 时每次考虑合并一个孩子信息的过程。



有向树拓扑序计数问题的做法是 well-known 的——通过容斥原理转化到一些叶向森林拓扑序计数的子问题，然后带上容斥系数把答案加起来。

事实上只要更精巧地使用容斥原理，可以把有向边仙人掌拓扑序计数问题、甚至有向广义串并联图拓扑序计数问题也转化到一些叶向森林拓扑序计数的子问题。



前言
○

带多元权定的叶向树拓扑序计数
○○○○○

逆向树形背包合并
○○○○○○○

更精巧的容斥原理
○○●○○○○○○○

致谢
○

有向边仙人掌拓扑序计数

给定一张 n 个点 m 条边的有向图，保证将所有有向边视为无向边后，图是一个边仙人掌。求原图有多少拓扑序，答案模 998244353。
 $n \leq 500, m \leq 2(n-1)$ 。

杭州第二中学 陈昕阳
浅谈树拓扑序计数相关问题的一些方法

前言
○

带多元权定的叶向树拓扑序计数
○○○○○

逆向树形背包合并
○○○○○○○

更精巧的容斥原理
○○●○○○○○○○

致谢
○

容斥分析

任意定根 r ，我们可以把仙人掌上的边分为两类：在环上的边，称为环边；不在任意环上的边，称为树边。

通过对一条边的方向设定一个预期，可以保证在子问题中要么这条边不存在，那么该边的方向与预期一致。

求出 r 到每个点 u 的最短路 dis_u ，我们这样设定每条边的预期：

- 对于树边，预期方向是两个点中 dis 较小的指向较大的。
- 对于环边，依次考虑每个简单环 C ，确定这个环上所有边在子问题中的方向。记该环上 dis 最小的点是 u ，从 u 开始顺时针考虑环上的每条边，一开始设定预期方向是顺时针方向。每考虑一条边，一定会拆出一条顺时针方向的边，可能还有一条无限制边。如果在新子问题中边是顺时针方向，预期方向不变；否则，从环上的下一条边开始，预期方向变为逆时针方向。

杭州第二中学 陈昕阳
浅谈树拓扑序计数相关问题的一些方法



这里边的预期方向事实上与前面分解出的多个子问题中选择了哪个有关。可以看作是一开始只有一个子问题，所有边均未被拆开。每次新考虑一条边，如果是顺时针方向那该子问题不分裂；如果是逆时针方向，分裂到一个顺时针方向的子问题以及一个无限制的子问题，在无限制的子问题中预期方向被反转了。

这样容斥的好处在于最终每个子问题每个点入度都 ≤ 1 ，这个通过对点和边进行一些分类即可证明。那么排除掉有环的情况，剩下的子问题就都是叶向森林了。



子问题中所有边当然都在原图中出现过。原图中所有树边在子问题中要么不存在要么是由距 r 较近的点指向较远的；所有环边在子问题中，与 dis 最小点 u 弱连通的部分也一定是由距 u 较近的点指向较远的（这里的距离是在子问题中的）。

所以转移顺序也是比较明确的，对仙人掌建圆方树后自下至上 DP 即可，DP 的状态设计与叶向树问题是类似的。

更多 DP 细节在此略去，最终可以得到一个 $O(n^3)$ 的做法。



前言
带多元权定的叶向树拓扑序计数
逆向树形背包合并
更精巧的容斥原理
致谢

有向广义串并联图计数

给定一张 n 个点 m 条边的有向图，保证将所有有向边视为无向边后，图是广义串并联图。求原图有多少拓扑序，答案模 998244353。
 $n \leq 100, m \leq 2(n-1)$ 。

杭州第二中学 陈昕阳
浅谈树拓扑序计数相关问题的一些方法

前言
带多元权定的叶向树拓扑序计数
逆向树形背包合并
更精巧的容斥原理
致谢

容斥分析

广义串并联图肯定要考虑其对应的三种操作了，不过事实上如果一开始图是简单图，叠合重边操作总是发生于一次缩二度点操作之后，可以一起考虑。

这样一来可以看作是要不断进行删 0, 1, 2 度点的操作。

还是希望通过容斥分解到很多每个点入度 ≤ 1 的子问题，一个简单的想法是当要删去点 u 时，确定其所有邻边的预期方向为指向 u 。这样每个点只有在被删去时才有可能增加入度，所以入度总是 ≤ 2 ，但我们希望的是入度 ≤ 1 。

前言 ○ 带多元钦定的叶向树拓扑序计数 ○○○○○ 逆向树形背包合并 ○○○○○○ 更精巧的容斥原理 ○○○○○○○●○○ 致谢 ○

容斥分析

显然只有缩二度点时才会出现入度 $= 2$ 的情况，此时是容斥后两个邻居 v, w 均存在 $v \rightarrow u, w \rightarrow u$ 的边。但我们注意到如果此时 v, w 之间存在边，比方说存在 $v \rightarrow w$ 的边，那么就可以删去 $v \rightarrow u$ 的边，因为只要有 $v \rightarrow w, w \rightarrow u$ 的边， $v \rightarrow u$ 的限制是自然成立的，这样一来 u 的入度就变为 1 了。

但本来 v, w 之间可能不存在边，为了解决这一情况，我们可以创造两个子问题，这两个子问题中分别有 $v \rightarrow w$ 的边和 $w \rightarrow v$ 的边，通过把两个子问题答案加和得到原问题答案。

注意：该做法中容斥会创造子问题，拆 v, w 间的边也会创造子问题。

杭州第二中学 陈昕阳
浅谈树拓扑序计数相关问题的一些方法

前言 ○ 带多元钦定的叶向树拓扑序计数 ○○○○○ 逆向树形背包合并 ○○○○○○ 更精巧的容斥原理 ○○○○○○○●○○ 致谢 ○

DP 实现

删 0, 1, 2 度点会导致一些点获得子树大小增量，但增量对应的点已经不在当前的图上，需要记录这个增量信息。

任何时刻有 m 条边的图事实上都存在 2^m 个子问题，即 m 条边每条的定向选择哪一种导出的 2^m 种情况。

记录 DP 信息 $dp_{id,0/1,i,j}$ 表示若将第 id 条边 $= (u, v)$ 定向为正/反向，在容斥拆边过程中，使得最终叶向森林 u 子树大小恰增加 i ， v 子树大小恰增加 j 的所有方案的权值和。注意这里的 i, j 都是这条边带来的影响，对应的点事实上是在先前的三种操作过程中被收缩到这条边上的点。

还需记录 $val_{u,i}$ 表示不考虑所有边带来的子树大小增量，只考虑删一度点操作对 u 子树大小影响的话，容斥拆边过程中 u 子树大小为 i 的所有方案的权值和。

杭州第二中学 陈昕阳
浅谈树拓扑序计数相关问题的一些方法



前言
带多元钦定的叶向树拓扑序计数
逆向树形背包合并
更精巧的容斥原理
致谢

DP 实现

希望删除一个度数 ≤ 2 的点 u 时，要对涉及边选取的方向进行讨论，考虑涉及的点的 val 信息以及涉及边的 dp 信息，并结算这个点的 $\frac{1}{siz_u}$ 贡献。 u 的子树大小增量也可能需要传递到某个与它有边的点上，这个增量可能记在边上，即更新 dp （当增量与某条边方向有关时），也可能记在点上，即更新 val （该增量与任何边方向都无关）。

在转移过程中需要进行二维背包合并，对两维均暴力卷积最终做法是 $O(n^4)$ 的，对一维暴力卷积一维 FFT 可以做到 $O(n^3 \log n)$ 。



浅谈一种互质数对与最大公约数的维护算法

南京外国语学校 杜冠成

摘要

定义与约定

互质数对的维护

最大公约数的维护

算法应用

总结

致谢

中国计算机学会

南京外国语学校 杜冠成
浅谈一种互质数对与最大公约数的维护算法

摘要

CCF

- 本文首创了一种互质数对与最大公约数的维护算法，并将后者命名为 Dynamic-gcd algorithm，同时结合例题对这两个算法的应用加以阐述，以体现其广泛的应用前景。



目录

定义与约定

互质数对的维护

最大公约数的维护

算法应用

总结

总览

致谢

1 摘要

2 定义与约定

2.1 互质数对的维护

2.2 最大公约数的维护

2.3 算法应用

2.4 总结

南京外国语学校 杜冠成

浅谈一种互质数对与最大公约数的维护算法

目录

定义与约定

互质数对的维护

最大公约数的维护

算法应用

总览

致谢

定义与约定

- $Prime$ 为所有质数构成的集合。
- $P(x)$ 表示 x 的最大质因子。

南京外国语学校 杜冠成

浅谈一种互质数对与最大公约数的维护算法

目录

定义与约定

互质数对的维护

最大公约数的维护

算法应用

总结

致谢

1 摘要

2 定义与约定

3 互质数对的维护

4 最大公约数的维护

5 算法应用

6 总结

南京外国语学校 杜冠成
浅谈一种互质数对与最大公约数的维护算法

目录

定义与约定

互质数对的维护

最大公约数的维护

算法应用

总结

致谢

引入

■ 给定 n ，你需要对初始所有元素均为 1 的序列 a_1, a_2, \dots, a_n 进行若干次修改，使得 $\forall i \in [1, n]$ ，存在一个时刻满足 $\forall a_j = [\gcd(i, j) = 1]$ 。

南京外国语学校 杜冠成
浅谈一种互质数对与最大公约数的维护算法

摘要	定义与约定	互质数对的维护	最大公约数的维护	算法应用	总结	致谢
○○	○○	○○○○●○○○○○○○○	○○○○○○○○○○○○○○	○○○○○○○	○○	○

维护方式

- 注意到, i 与 j 不互质, 当且仅当 $\exists p \in \text{Prime}, p|i, p|j$.
- 考虑列出 n 以内的所有质数, 并采用暴力搜索的方式, 对于每个质数 p , 考虑 p 是否能整除 i , 并将其作为两个分支继续搜索下去。
- 具体来说, 我们在搜索的过程中, 维护 x_0, i_0 及序列 a , 表示目前考虑到了第 x_0 小的质数, 且钦定需要整除 i 的所有质数乘积为 i_0 。初始 $x_0 = i_0 = 1$ 且 a 中所有元素均为 1。令 p_0 为第 x_0 小的质数, 在 $i_0 p_0 \leq n$ 的前提下, 若钦定 p_0 整除 i , 则将 a_{p_0}, a_{2p_0}, \dots 赋值为 0, 并向分支 $(x'_0, i'_0) = (x_0 + 1, i_0 p_0)$ 继续搜索; 然后, 从 (x'_0, i'_0) 回溯后, 撤销对 a_{p_0}, a_{2p_0}, \dots 的修改, 并向分支 $(x_0 + 1, i_0)$ 搜索。

摘要	定义与约定	互质数对的维护	最大公约数的维护	算法应用	总结	致谢
○○	○○	○○○○○○●○○○○○○○○	○○○○○○○○○○○○○○	○○○○○○○	○○	○

正确性证明

- 可以发现, 若在搜索过程中, 所有钦定整除 i 的质数构成的集合恰好与某个 i 的质因子集合相同, 则该时刻满足条件。换言之, 若对于每个 i , 求出 i 的所有质因子的乘积 t_i , 则当 $i_0 = t_i$ 时, 该时刻恰好满足关于 i 的要求。
- 注意到这个时刻总存在, 因此前述算法的正确性得到了保障。

摘要	定义与问题	互质数对的维护	最大公约数的维护	算法应用	总结	致谢
○○	○○	○○○○○○○○○○●○	○○○○○○○○○○○○	○○○○○○○	○○	○

减枝及时间复杂度证明

- 注意到，当 $i_0 p_0 > n$ 时，此后总有 $i_0 p_0 > n$ ，那么不会再额外钦定任意质数整除 i 。此时直接返回即可。
- 考虑分析算法的时间复杂度。首先，dfs 本身的时间复杂度是 $\Theta(n)$ 的，复杂度的瓶颈在于枚举 p_0 倍数的部分。
- 注意到 $i_0 p_0$ 的最大质因子为 p_0 ，而 dfs 过程中所有 $i_0 p_0$ 两两不同，因此时间复杂度为 $O\left(\sum_{i=1}^n \frac{n}{P(i)}\right)$ 。
- Erdos, Ivic, Pomerance 的论文中给出¹

$$\sum_{i=1}^n \frac{1}{P(i)} = n\delta(n) \left(1 + O\left(\frac{\log \log n}{\log n}\right)^{\frac{1}{2}}\right)$$
- 这个算法的实际运行效率相当优秀，在 $n \leq 10^6$ 时，其时间复杂度可以看做 $\Theta(n\sqrt{n})$ ，常数较小。

¹[1] P. Erdos, A. Ivic and C. Pomerance, On sums involving reciprocals of the largest prime factor of an integer. Glasnik Matematicki. 21(41) 1986, 283-300.

摘要	定义与问题	互质数对的维护	最大公约数的维护	算法应用	总结	致谢
○○	○○	○○○○○○○○○○●○	○○○○○○○○○○○○	○○○○○○○	○○	○

算法伪代码

Algorithm 1 互质数对的维护

```

1:  $p_0 :=$  the  $x_0$ -th smallest prime
2: if  $i_0 p_0 > n$  then
3:   for each  $l(i) == i_0$  do
4:     claim requirements of  $i$  have been satisfied
5:   end for
6: else
7:   dfs( $x_0 + 1, i_0$ )
8:   for each  $j$  which is a multiple of  $p_0$  do
9:      $a_j := 1$ 
10:  end for
11:  dfs( $x_0 + 1, i_0 p_0$ )
12:  for each  $j$  which is a multiple of  $p_0$  do
13:    revoke operations on  $a_j$  to recover
14:  end for
15: end if

```



The image shows two slides from a presentation. The top slide is a table of contents with sections: 1. 摘要 (Summary), 2. 定义与约定 (Definitions and Conventions), 3. 互质数对的维护 (Maintenance of coprime pairs), 4. 最大公约数的维护 (Maintenance of GCD), 5. 算法应用 (Algorithm Application), and 6. 总结 (Conclusion). The bottom slide is titled '引入' (Introduction) and contains a problem statement: '给定 n ，你需要对初始所有元素均为 1 的序列 a_1, a_2, \dots, a_n 进行若干次修改，使得 $\forall i \in [1, n]$ ，存在一个时刻满足 $\forall a_j = \gcd(i, j)$.'

摘要	定义与约定	互质数对的维护	最大公约数的维护	算法应用	总结	致谢
○○	○○	○○○○○○○○○○○○○○○○	○○○○●○○○○○○	○○○○○○	○○	○

维护方式

- 注意到, $\gcd(i, j)$ 为所有满足 $p^\alpha | i, p^\alpha | j (p \in \text{Prime}, \alpha \in \mathbb{N}^+)$ 的 p 的乘积。
- 类似互质数对的维护, 我们列出 n 以内的所有质数, 并暴力搜索。对于每个质数 p , 枚举 i 质因数分解后 p 的最高幂次 β , 将其作为多个分支继续搜索下去。在此过程中, 维护 x_0, i_0 及序列 a_i , 表示考虑到了第 x_0 小的质数, 而 i_0 表示所有 p^β 的乘积。
- 在枚举 β 的过程中, 每当 β 增大 1 后, 若 $i_0 p^\beta \leq n$, 则将所有满足 $p^\beta | j$ 的 a_j 乘 p , 向分支 $(x_0 + 1, i_0 p^\beta)$ 继续搜索。在 (x_0, i_0) 的所有分支处理完毕后, 对于所有满足 $p | j$ 的 a_j , 撤销对 a_j 的所有乘 p 操作。

南京外国语学校 杜冠成
浅谈一种互质数对与最大公约数的维护算法

摘要	定义与约定	互质数对的维护	最大公约数的维护	算法应用	总结	致谢
○○	○○	○○○○○○○○○○○○○○○○	○○○○○○○○●○○○	○○○○○○	○○	○

算法证明

- 根据算术基本定理, $\forall i \in [1, n]$, 都存在某个时刻满足 $i_0 = i$, 算法的正确性得到了保障。
- 2.2 节提出的减枝与时间复杂度证明同样适用于此处。
- 在实际运行效率方面, 当 $n \leq 10^6$ 时, 其时间复杂度可看做 $\Theta(n\sqrt{n})$, 但常数较大。

南京外国语学校 杜冠成
浅谈一种互质数对与最大公约数的维护算法

摘要	定义与约定	互质数对的维护	最大公约数的维护	算法应用	总结	致谢
○○	○○	○○○○○○○○○○○○○○	○○○○○○○○●○○	○○○○○○	○○	○

算法伪代码

Algorithm 2 最大公约数的维护

```

1:  $p_0 :=$  the  $x_0$ -th smallest prime
2: if  $i_0 p_0 > n$  then
3:   claim requirements of  $i = i_0$  have been satisfied
4: else
5:   dfs( $x_0 + 1, i_0$ )
6:   for each  $\beta \in \mathbb{N}^+$  satisfying  $i_0 p_0^\beta \leq n$  do
7:     for each  $j$  which is a multiple of  $p_0^\beta$  do
8:        $a_j := pa_j$ 
9:     end for
10:    dfs( $x_0 + 1, i_0 p_0^\beta$ )
11:  end for
12:  for each  $j$  which is a multiple of  $p_0$  do
13:    revoke operations on  $a_j$  to recover
14:  end for
15: end if

```

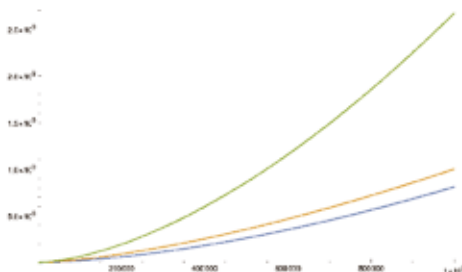
南京外国语学校 杜冠成

浅谈一种互质数对与最大公约数的维护算法

摘要	定义与约定	互质数对的维护	最大公约数的维护	算法应用	总结	致谢
○○	○○	○○○○○○○○○○○○○○	○○○○○○○○○○●	○○○○○○	○○	○

算法总结

- 可以看出，此算法能够通过一定次数的修改，动态维护每一对 n 以内正整数的最大公约数，因此我将其命名为 Dynamic-gcd algorithm。
- 下图蓝线、绿线分别为互质数对维护算法、Dynamic-gcd algorithm 中 $a_j \leftarrow 1$ 和 $a_j \leftarrow pa_j$ 的执行次数关于 n 的函数图像，黄线表示 $f(x) = x\sqrt{x}$ 的函数图像。



南京外国语学校 杜冠成

浅谈一种互质数对与最大公约数的维护算法



目录

正文与行号

互质数对的维护

最大公约数的维护

算法应用

总结

致谢

1 摘要

2 定义与约定

3 互质数对的维护

4 最大公约数的维护

5 算法应用

6 总结

南京外国语学校 杜冠成
浅谈一种互质数对与最大公约数的维护算法

目录

正文与行号

互质数对的维护

最大公约数的维护

算法应用

总结

致谢

例题

例

■ 给定长度为 n 的序列 A , T 次询问, 每次给定正整数 x, l, r , 求 $A_{\gcd(x, l)}, A_{\gcd(x, l+1)}, \dots, A_{\gcd(x, r)}$ 的最大子段和。

■ 保证 $1 \leq T \leq 10^6, 1 \leq n \leq 5 \times 10^4, 1 \leq x, l, r \leq n, l \leq r$ 。

■ 本题为笔者的原创题。

南京外国语学校 杜冠成
浅谈一种互质数对与最大公约数的维护算法

摘要	定义与约定	互质数对的维护	最大公约数的维护	算法应用	总结	致谢
○○	○○	○○○○○○○○○○○○○○	○○○○○○○○○○○○	○○○○●	○○	○

解法

解

- 考虑 Dynamic-gcd algorithm，在维护 a 的同时，维护序列 b ，满足 $b_i = A_{a_i}$ 。
- 每次对 a 的单点修改后，对 b 也进行单点修改。
- 将所有询问统一离线处理后，问题转化为： $\Theta(n\sqrt{n})$ 次单点修改， T 次查询区间最大子段和。
- 使用线段树维护即可，时间复杂度 $\Theta((n\sqrt{n} + T) \log n)$ 。

南京外国语学校 杜冠成
浅谈一种互质数对与最大公约数的维护算法

摘要	定义与约定	互质数对的维护	最大公约数的维护	算法应用	总结	致谢
○○	○○	○○○○○○○○○○○○	○○○○○○○○○○○○	○○○○○●	○○	○

- 1 摘要
- 2 定义与约定
- 3 互质数对的维护
- 4 最大公约数的维护
- 5 算法应用
- 6 总结

南京外国语学校 杜冠成
浅谈一种互质数对与最大公约数的维护算法



摘要	定义与约定	互质数对的维护	最大公约数的维护	算法应用	总结	致谢
00	00	0000000000000000	00000000000000	0000000	0●	0
总结						

- 本文提出了互质数对的维护算法，与最大公约数的维护算法 Dynamic-gcd algorithm，通过例题可以看出，它们可以通过较少次数的修改，动态维护每一对 n 以内正整数间互质性、最大公约数的信息，在与互质、gcd 有关的数论、数据结构等题型中，均有广阔的应用前景。
- 笔者希望本文能起到抛砖引玉的作用，吸引读者对数论信息的动态维护进行深入研究。



浅谈一些信息合并的处理方式

重庆市巴蜀中学 冯政玮

声明与主要内容

声明与主要内容

信息合并与信息学竞赛中是非常重要的部分，今天我们将从朴素信息合并，结构信息合并共两个角度简要探讨一下在算法竞赛中常见的一些信息合并在具体问题中的处理方式与技巧。
今天的内容比较通俗易懂，希望同学们能认真听，也希望同学们能理解交流分享内容仅为论文一部分更多例题和内容详见我的论文
作者才疏学浅，若所述内容有误请指出。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

2 / 29

前置知识

前置知识

信息一般指在具体问题中需要维护的东西，它可以是一个权值，一个节点，一条边，一个式子等。
而通常信息合并的过程指将信息类比于集合间的元素，将若干信息按照某些方式并在一起，维护一个信息结构，使得可以在结构中可以快速查找想要的信息，不同的信息对应着不同的处理方式。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

3 / 29

朴素信息合并 启发式合并

启发式合并

启发式合并是一种信息合并方式，通常形如有若干个信息集合总大小为 n ，操作形如合并两个存在的集合。

具体地，对于两个大小分别是 $A, B (A < B)$ 的信息集合，我们将 A 集合的元素取出，插入到 B 集合中。

复杂度分析：对于每个信息元素，当它被执行插入操作时，其所处的集合大小一定翻倍，所以对于每个信息元素只会被执行至多插入 $\log n$ 次。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

4 / 29

朴素信息合并 启发式合并

启发式合并与整体二分结合

例题：

给定一张 n 个点 m 条边的带权无向图， q 次询问，每次给定 x, y 询问 x 到 y 的路径的最小权值，一条路径的权值定义为路径上边权中的次大值，如果仅经过一条边，次大值定义为 0。可视为 n, m, q 同阶。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

5 / 29

朴素信息合并 启发式合并

启发式合并与整体二分结合

特殊讨论最大值的一条边，最小化次大值，整体二分处理。

问题转化为如何快速验证通过所有边权在 $1 \sim mid$ 和一条额外边使得 x, y 相互可达。

维护连通块之间边的映射关系，启发式合并递归再撤回。

注意到启发式合并的复杂度为均摊 $O(n \log n)$ 次插入操作，单次合并的时间复杂度并没有确切保证

注意到对于 i 点而言，每次加入时总是在加入完 $1 \sim i-1$ 后再加入 i 节点信息，可以视为顺序的执行了 $\log n$ 次从 $1 \sim n$ 的启发式合并，所以复杂度正确。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

6 / 29

朴素信息合并 启发式合并

朴素信息合并

更多朴素信息合并的不同角度与例题详见我的论文，限于篇幅不在今天介绍。

接下来我们从平衡树与线段树的角度简单介绍以下结构信息合并。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

7 / 29

结构信息合并 平衡树合并

平衡树合并

平衡树一般维护有序信息集合，在集合上可以支持若干操作如区间求和，区间权值更新等，平衡树的合并可以视为两集合求并集后得到的一颗新的平衡树。

一些平衡树合并可以做到较优的复杂度，但是由于其考察较少，平衡树自身常数较大，所以在OI中较优复杂度的平衡树合并少有人提及与实现，通常对于平衡树的合并都是以启发式合并的方式进行处理。

我们将详细介绍复杂度较优的平衡树合并如何实现及具体细节并简要说明其复杂度。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

8 / 29

结构信息合并 平衡树合并

finger search

finger search 指搜索的一种扩展，即维护上一次搜索得到的元素称为finger，再基于finger元素做搜索操作。

具体地对于有序序列，定义 $d(x, y)$ 表示元素 x, y 集合中的排名之差，若上一次查找到的元素为 x ，下一次查找元素为 y ，其查找复杂度可以描述为一个关于 $d(x, y)$ 的函数如 $O(f(d(x, y)))$ ，我们称这个过程为finger search。

finger search tree 定义为一种二叉搜索树，保留指向上一次操作的节点的指针，保留的节点称为finger，同样的，我们定义 $d(x, y)$ 为中序遍历排名之差。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

9 / 29

结构信息合并 平衡树合并

Splay

Splay 是一类二叉搜索树，在执行操作过程中使用旋转平衡其树高，保证其均摊时间复杂度，其结构比较经典，这里不再介绍其操作及复杂度证明。

由于其每次操作后会将操作节点旋转到根，可以视根为 finger，于是每次操作可视为在上次的基础上做新的 finger search 操作。

在 [10, 11] 中有证明，其单次操作复杂度可以描述为 $O(\log d)$ 。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

10 / 29

结构信息合并 平衡树合并

Splay

考虑 Splay 合并，我们将小的一颗平衡树按照中序遍历取出（即取出元素有序）顺序将其直接按照 Splay 的 insert 插入至第二棵平衡树中。

不妨设小树的大小为 m 大树的大小为 n ，设 m 个节点在 n 中的 rk 分别为 D_1, D_2, \dots, D_m ，对于第 $i (i > 1)$ 次插入的复杂度为

$O(\log d_i), d_i = D_i - D_{i-1}$ 。

讨论第一次，剩下插入复杂度为 $\sum_{i=2}^m \log d_i$ 。

有 $\log A + \log B \leq 2 \log \frac{A+B}{2}$ ，根据类似均值的推导，我们有

$\sum_{i=2}^m \log d_i \leq m \log \frac{n}{m}$ 。

即合并时总插入复杂度不超过 $O(\log n + m \log \frac{n}{m})$ 。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

11 / 29

结构信息合并 平衡树合并

Splay

对于一个元素，考虑其若干次合并的贡献，设其所属集合 siz 的变化量为一个序列 $p_1 = 1 < p_2 < \dots < p_k = n$ ，故当前节点消耗总复杂度为

$$O\left(\sum_{i=1}^{k-1} \log \frac{p_{i+1}}{p_i}\right) = O(\log n)$$

总合并复杂度为 $O(n \log n)$ 。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

12 / 29

结构信息合并 平衡树合并

无旋 Treap

Treap 是一种通过随机权值优化的二叉搜索树，每个节点有一个键值和关联的随机优先级，每个节点的键值满足二叉搜索树的性质，每个节点的随机优先级满足堆的性质，顾名思义 $\text{Tree} + \text{Heap} = \text{Treap}$ 。

其保证复杂度的直接一种理解方式为视为当前是一个序列，然后每次选择随机优先级最高的作为根，然后划分为左右子树，注意到找随机优先级最高形如随机划分，则期望树高为 $O(\log n)$ 。

无旋 Treap 由于其易于实现，且能支持信息可持久化，在 OI 中比较常用。

这里简单介绍无旋 Treap 的基本操作与启发式合并的方式，注意当前一颗平衡树维护的内部键值形如一个集合（互不相同）。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

13 / 29

无旋 Treap

- $\text{Split}(A, \text{val}) = \{C, D, x\}$ 将 A 按照 val 分裂成 $<\text{val}$ 与 $>\text{val}$ 的两个部分， $=\text{val}$ 权值的点 x （不存在则为空节点）。
- $\text{Merge}(A, B) = C$ 对于 A, B 集合满足 $\max A < \min B$ （这里比较 A, B 的键值）新集合为 A, B 集合的并。
- $\text{Union}(A, B) = C := A \cup B$ 求 A, B 集合的并集合。

对于前两个函数是为人熟知的无旋 Treap 的普通合并方式，这里不再详细介绍，其期望复杂度分别为 $O(\log |A|)$, $O(\log |A| + \log |B|)$ 。

无旋 Treap

对于启发式合并的过程我们可以视为 Treap 的 Union，这里详细介绍其处理方式：

对于合并 A, B Treap，不妨 A, B 根中 A 根随机权值优先级更高，取 A 根作为合并后的根，令 A 根的键值 k ，将 B 按照键值 k 分为 C, D, x 分别对应 $<k, >k, =k$ 的信息，舍弃 x 节点，将 C 递归与 A 左子树合并得到当前左子树， D 递归与 B 右子树合并得到当前右子树。

对于 Union 合并两颗大小分别为 $n, m (m \leq n)$ 的平衡树其复杂度均为 $O(m \log \frac{n}{m})$ 。

上述复杂度分析在原论文 [14] 有详细介绍，较为复杂不在此处说明。

根据类似 Splay 合并的上述分析有，总合并复杂度为 $O(n \log n)$ 。

结构信息合并 平衡树合并

无旋 Treap

这里请大家提醒注意，当存在键值重复元素，一般的写法是重定义 Split 函数为分裂成 $\leq val$ 和 $> val$ 的两个部分。

注意，此时使用 Union 函数进行合并时可能导致树高退化，以下是一个极端情况的例子：

A,B 两树的所有节点的键值相同，考虑 $\text{Union}(A,B)=D$ 的过程：

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

16 / 29

结构信息合并 平衡树合并

无旋 Treap



图: 相同权值 Union 的例子

不妨设 A 树随机优先级高于 B 树，此时 Split 的过程就是将 B 树和 A 的右子树 C 进行合并，通过归纳，可以发现合并出来的 D 树其右链长度为 A,B 右链长度之和，此时树高并不满足高度期望为 $O(\log n)$ 。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

17 / 29

结构信息合并 线段树合并

线段树合并

线段树同样维护有序信息集合，相对平衡树其优势在于结构简洁明确，实现简单，可以支持大部分平衡树可处理的操作。

线段树的结构是为人熟知的，此处不再赘述。

线段树合并用于维护线段树间信息集合合并，由于相比平衡树其常数较小，时间复杂度易于分析，有较优拓展性，所以我们常用线段树合并。

复杂度分析：对于两颗合并的线段树，结构类似，两颗线段树合并的时候当一颗线段树节点为空时可以直接继承另一颗线段树的信息，每次合并后两个节点会变成一个节点，以节点个数作为势能，注意到设插入的操作次数为 $O(n)$ 次，总插入节点个数为 $O(n \log n)$ ，总线段树合并的过程复杂度也为 $O(n \log n)$ 。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

18 / 29

结构信息合并 线段树合并

带新建节点标记的线段树合并

考虑以下问题：给定一棵树，对于树上节点有权值 A ，需要维护支持区间 $[1, A], [A + 1, n], [1, n]$ 区间加，信息合并为下标对位乘法，对于所有点需要求得整个子树的信息。

对位乘积可以描述为线段树合并的过程，若一边为一个节点，则可以对另一侧打区间乘法标记，区间加法仅建立遍历到的节点，下放加法标记时需要新建线段树节点。

考虑其合并复杂度：对于任意线段树节点，要么同时有左右儿子，要么同时没有左右儿子，在所有标记下放的时候均不会造成节点的新建，故总复杂度仍 = 节点个数，即 $O(n \log n)$ 。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

19 / 29

结构信息合并 线段树合并

复杂度分析

这个例子很好的告诉我们当线段树标记下放需要新建节点时复杂度仍尝试分析节点个数变化。

考虑对于不满足此初始条件时，此类线段树合并时间复杂度，相似的，我们假定在合并时，存在一边只有单个节点（没有左右儿子）时可以快速将影响应用到另一棵树上。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

20 / 29

结构信息合并 线段树合并

复杂度分析

若直接考虑合并时分析其节点的新建情况，有例子形如，合并两条一致的黑色部分链，需要做标记下放而新建虚线节点，此时我们递归深度为 $\log n$ 但节点个数仅减少 1，似乎此时导致总复杂度为 $O(n \log^2 n)$ 。

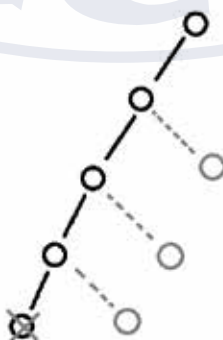


图: 合并的例子

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

21 / 29

结构信息合并 线段树合并

复杂度分析

其实不然，注意到当前建立的叶子总是不存在左右儿子的，即会新建节点的线段树节点总是在原结构上只有一个儿子的节点，我们将其补齐，相似的，初始节点为 $O(n \log n)$ 个且满足例题性质。
有在此类线段树合并新建节点过程中我们仅会补齐上述节点，故总复杂度仍为 $O(n \log n)$ 。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

22 / 29

结构信息合并 线段树合并

线段树合并空间优化

考虑线段树合并的空间消耗，通常情况下我们空间复杂度的上界为 $O(n \log n)$ 。

我们尝试改变线段树合并的顺序，并在删除节点时回收利用空间，得到紧的空间上界。

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

23 / 29

线段树合并空间优化

具体地，考虑当前问题形如一棵树，每个节点有若干个单点修改需要对于每个子树都求出子树内所有操作在线段树合并后的形式。
若当前形如区间修改 $[l, r]$ ，我们考察其新建节点的情况，其新建的节点不会超过单点加入 l 并补全路径上右儿子与单点加入 r 并补全路径上左儿子，于是区间修改可以大致视为两次单点修改。
我们尝试按照每个点的操作次数做重链剖分，当前处理方式为先遍历重儿子，遍历完后递归轻儿子子树，与重儿子子树合并。
注意在空间回收意义下即消耗空间复杂度为任意时刻最多存在的节点数量。

线段树合并空间优化

可以发现任意时刻形如至多维护 $O(\log n)$ 棵线段树，且第 i 棵大小不会超过 $\frac{n}{2^i}$ ，我们尝试精细分析其占用的空间。
考虑线段树一共有 $\log n$ 层，对于加入 x 个节点的线段树，其第 i 层新建的节点数量为 $\min(x, 2^i)$ ，此时视为上界维护了 $\log n$ 棵线段树，第 $i \in [1, \log n]$ 棵有 2^i 个节点。
当前形如总节点数的上界为

$$\begin{aligned} \sum_{i=0}^{\log n} 2^{i+1} + (\log n - i)2^i &\leq O(n) + \sum_{i=0}^{\log n} \sum_{j=i+1}^{\log n} 2^i = \\ O(n) + \sum_{j=0}^{\log n} \sum_{i \leq j} 2^i &\leq O(n) + \sum_{j=0}^{\log n} 2^{j+1} = O(n) \end{aligned}$$

即总消耗空间复杂度为 $O(n)$ 。

参考文献与资源

参考文献 I

- 2015 年集训队论文, 任之洲, 《浅谈启发式思想在信息学竞赛中的应用》
- 2021 年集训队论文, 周镇东, 《浅谈一类树上路径相关问题》
- OI Wiki, dsu on tree, <https://oi-wiki.org/graph/dsu-on-tree/>
- xiaoziyao, 浅谈一类信息的暴力重构手法
<https://www.cnblogs.com/xiaoziyao/p/17413029.html>
- 2013 年集训队论文, 许昊然 《浅谈数据结构题的几个非经典解法》
- Wikipedia, leftist tree, <https://oi-wiki.org/ds/leftist-tree/>
- Wikipedia, Finger search, https://en.wikipedia.org/wiki/Finger_search

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

27 / 29

参考文献与资源

参考文献 II

- Wikipedia, Finger search tree, https://en.wikipedia.org/wiki/Finger_search_tree
- D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. Journal of the ACM
- R. Cole. On the dynamic finger conjecture for splay trees. part II: The proof. SIAM Journal of Computing, 30(1):44-85, 2000.
- R. Cole, B. Mishra, J. Schmidt, and A. Siegel. On the dynamic finger conjecture for splay trees. part I: Splay sorting log n-block sequences. SIAM Journal of Computing, 30(1):1-43, 2000.
- R. Seidel and C. R. Aragon. Randomized search trees. Algorithmica, 16(4/5):464-497, 1996.

冯政玮 (重庆市巴蜀中学)

浅谈一些信息合并的处理方式

2024.1.14

28 / 29

参考文献与资源

参考文献 III

-  Gerth Stølting Brodal. Finger Search Trees. University of Aarhus, 2005
-  Guy E. Blelloch and Margaret Reid-Miller. Fast set operations using treaps. In 10th Annual ACM Symposium on Parallel Algorithms and Architectures, Puerto Vallarta, Mexico, June -July 1998. ACM.
-  2018 年集训队论文, 董炜旻, 《浅谈 Splay 与 Treap 的性质及其应用》
-  约瑟夫用脑玩, finger-search+ 启发式合并, <https://www.luogu.com.cn/blog/ICANTAKIOI/finger-search-qi-fa-shi-ge-bing>
-  dpair, 记录一个线段树合并的 trick, <https://dpair.gitee.io/articles/sgtmrgt/>

线性拟阵奇偶

华东师范大学第二附属中学 郭羽冲

线性拟阵奇偶
●○○○○○○○○○

带权线性拟阵奇偶
○○○○○○○○○○○○

例题
○○○○○

线性拟阵奇偶

定义

给定 m 个二元组 (a_i, b_i) ，其中 a_i, b_i 均为 n 维向量。选出最多的二元组，使得所有被选出的二元组中包含的向量线性无关。

华东师范大学第二附属中学 郭羽冲

线性拟阵奇偶

线性拟阵奇偶
○●○○○○○○○○○

带权线性拟阵奇偶
○○○○○○○○○○○○

例题
○○○○○

线性拟阵奇偶

稀疏型构造

令 v 为原问题的答案。

令 $x'_1 \dots x'_m$ 为未定元。构造矩阵 A, X ：

$$A = (a_1, b_1, a_2, b_2, \dots, a_m, b_m)$$

$$X = \text{diag} \left(\begin{pmatrix} 0 & x'_1 \\ -x'_1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & x'_2 \\ -x'_2 & 0 \end{pmatrix}, \dots, \begin{pmatrix} 0 & x'_m \\ -x'_m & 0 \end{pmatrix} \right)$$

将矩阵放在 $x'_1 \dots x'_m$ 的有理分式域下考虑，则有

$$2v + 2m = \text{rank} \begin{pmatrix} 0 & A \\ -A^T & X \end{pmatrix}.$$

华东师范大学第二附属中学 郭羽冲

线性拟阵奇偶

线性拟阵奇偶
○○●○○○○○○○

带权线性拟阵奇偶
○○○○○○○○○○○○○

例题
○○○○○○○

线性拟阵奇偶

紧凑型构造

令 $x_1 \dots x_m$ 为未定元。构造矩阵 M ：

$$M = \sum_{i=1}^m x_i (a_i b_i^T - b_i a_i^T)$$

将矩阵放在 $x_1 \dots x_m$ 的有理分式域下考虑则有 $2v = \text{rank}(M)$ 。
相比稀疏型构造，紧凑型构造得到的矩阵规模较小，在计算的时间复杂度上有优势。

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○●○○○○○○○

带权线性拟阵奇偶
○○○○○○○○○○○○○

例题
○○○○○○○

线性拟阵奇偶

算法

为了得到较好的时间复杂度，我们需要对未定元 $x_1 \dots x_m$ 进行一些特殊处理。
任取一个大质数 p ， $x_1 \dots x_m$ 均从 $[0, p) \cap \mathbb{Z}$ 中随机选取，在 M 中将 $x_1 \dots x_m$ 的值代入后得到 M' 。 M' 为 \mathbb{F}_p 上的矩阵，可以用高斯消元 $O(n^3)$ 地计算 $\text{rank}(M')$ 。我们认为 $\text{rank}(M) = \text{rank}(M')$ ，即可进一步得到答案。

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○○○●○○○○

带权线性拟阵奇偶
○○○○○○○○○○○○○○

例题
○○○○○○

线性拟阵奇偶

正确率分析

上述算法是有一定错误概率的，考虑对其进行分析。
 有 $\text{rank}(M') \leq \text{rank}(M)$ ，下证明两者有较大概率取等。
 令 $r = \text{rank}(M)$ ，则 M 一定有一个 r 阶可逆子矩阵 $M_{S,T}$ 。
 $\det(M_{S,T})$ 为关于 x_1, \dots, x_m 的非零多元多项式，而 $\det(M'_{S,T})$ 即为将 x_1, \dots, x_m 的值代入多项式得到的结果。因此考虑如下引理：
 引理 (Schwartz-Zippel) 若 $f(x_1, \dots, x_m)$ 为不超过 d 次的非零多元多项式，且 x_1, \dots, x_m 均从集合 S 中随机选取，则 $f(x_1, \dots, x_m) = 0$ 的概率不超过 $\frac{d}{|S|}$ 。
 于是 $\det(M'_{S,T}) = 0$ 的概率不超过 $\frac{r}{p}$ 。因此我们有至少 $1 - \frac{r}{p}$ 的概率能够得到正确的结果。

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○○○●○○○○

带权线性拟阵奇偶
○○○○○○○○○○○○○○

例题
○○○○○○

线性拟阵奇偶

构造方案——算法

依次考虑每个二元组，若将当前二元组删除之后答案不会变小，则将其删除，否则保留。最终一定会剩下恰好 v 个二元组，它们即构成了一组答案。
 若使用暴力算法，则我们需要在删除每个二元组之后在 $O(n^3)$ 的时间复杂度内求 $\text{rank}(M)$ 。则总时间复杂度为 $O(mn^3)$ 。

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○○○○○●○○○

带权线性拟阵奇偶
○○○○○○○○○○○○○○

例题
○○○○○○

线性拟阵奇偶

构造方案——优化

考虑到 $\text{rank}(a_i b_i^T - b_i a_i^T) \leq 2$ ，我们可以使用经典的矩阵低秩扰动方法加速计算。

具体地，我们有如下引理：

引理（Sherman-Morrison-Woodbury, SMW）对于可逆 n 阶矩阵 A, B ， $n \times m$ 矩阵 U ， $m \times n$ 矩阵 V ，有：

$$(A - UB^{-1}V)^{-1} = A^{-1} + A^{-1}U(B - VA^{-1}U)VA^{-1}$$

且 $(A - UB^{-1}V)$ 可逆等价于 $(B - VA^{-1}U)$ 可逆。

华东师范大学第二附属中学 郭羽冲

线性拟阵奇偶

线性拟阵奇偶
○○○○○○●○○○

带权线性拟阵奇偶
○○○○○○○○○○○○○○

例题
○○○○○○

线性拟阵奇偶

构造方案——优化

M 未必可逆，无法直接利用上述引理。

由 M 的反对称性， M 一定有一个 r 阶可逆主子矩阵 $M_{S,S}$ 。

将所有向量在 S 上截取，即只保留 S 所给出的 r 个分量。

任取 r 个线性无关的行作为 S ，容易证明 $M_{S,S}$ 即为满足条件的主子矩阵。问题归结为 M 可逆的情况。

华东师范大学第二附属中学 郭羽冲

线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○●○
带权线性拟阵奇偶
○○○○○○○○○○○○
例题
○○○○○○

线性拟阵奇偶

构造方案——优化

下假设 M 可逆。实时维护 $N = M^{-1}$ 。
 令 $U = (a_i, -b_i), V = (b_i, a_i)^T$, 则 $a_i b_i^T - b_i a_i^T = UV$ 。
 需要判断 $M - UV$ 是否可逆。由 SMW 引理可转化为判断 $I - VNU$ 是否可逆，可以在 $O(r^2)$ 的时间内计算出 $I - VNU$ 。而这是一个 2×2 的矩阵，可以 $O(1)$ 判断其可逆性。
 若可逆，则将 N 更新为 $N' = (M - UV)^{-1}$ 。由 SMW 引理得：

$$N' = (M - UV)^{-1} = N + NU(I - VNU)^{-1}VN$$

可以在 $O(r^2)$ 的时间复杂度内计算出 N' 。总时间复杂度降为 $O(mr^2)$ 。

线性拟阵奇偶
○○○○○○○○●○
带权线性拟阵奇偶
○○○○○○○○○○○○
例题
○○○○○○

线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○●○
带权线性拟阵奇偶
○○○○○○○○○○○○
例题
○○○○○○

线性拟阵奇偶

时间复杂度

上述算法时间复杂度组成如下：

- 求 $M = \sum_{i=1}^m x_i(a_i b_i^T - b_i a_i^T)$, $O(mn^2)$ 。
- 求 $\text{rank}(M)$, $O(n^3)$ 。

若构造方案，则需要增加：

- 求 M^{-1} , $O(n^3)$ 。
- 每次低秩扰动后判断是否可逆，更新 N , $O(mr^2)$ 。

实际情况中根据 a_i, b_i 的特殊性质可能可以进一步优化时间复杂度。

线性拟阵奇偶
○○○○○○○○●○
带权线性拟阵奇偶
○○○○○○○○○○○○
例题
○○○○○○

线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○○○

带权线性拟阵奇偶
●○○○○○○○○○○

例题
○○○○○○

带权线性拟阵奇偶

定义

顾名思义，是线性拟阵奇偶问题的带权版本。

学术界已经提出了本问题的 $O(\text{poly}(n, m))$ 算法。但此算法非常复杂，超出了笔者目前的能力范围。因此本节中我们将介绍一个相对简单但时间复杂度较高（非多项式）的算法。

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○○○

带权线性拟阵奇偶
●○○○○○○○○○○

例题
○○○○○○

带权线性拟阵奇偶

稀疏型构造

令 v 为原问题的答案。

令 w_i 的值域为 $[1, V] \cap \mathbb{Z}$ 。新引入一个未定元 z ，利用它的次数来刻画权值和。类似于不带权情况下的稀疏型构造，令：

$$A = (a_1 z^{w_1}, b_1, a_2 z^{w_2}, b_2, \dots, a_m z^{w_m}, b_m)$$

$$X = \text{diag} \left(\begin{pmatrix} 0 & x'_1 \\ -x'_1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & x'_2 \\ -x'_2 & 0 \end{pmatrix}, \dots, \begin{pmatrix} 0 & x'_m \\ -x'_m & 0 \end{pmatrix} \right)$$

$$Y = \text{diag} (y_1, y_2, \dots, y_n)$$

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○○○
带权线性拟阵奇偶
○○●○○○○○○○○
例题
○○○○○○

带权线性拟阵奇偶

稀疏型构造

则我们有如下定理：
定理

$$2v = \deg_z \left(\det \begin{pmatrix} Y & A \\ -A^T & X \end{pmatrix} \right)$$

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○○○
带权线性拟阵奇偶
○○●○○○○○○○○
例题
○○○○○○

带权线性拟阵奇偶

稀疏型构造——正确性证明

Y 为对角矩阵，因此将行列式展开得：

$$\det \begin{pmatrix} Y & A \\ -A^T & X \end{pmatrix} = \sum_S \det \begin{pmatrix} 0 & A \\ -A^T & X \end{pmatrix}_{S,S} \det(Y_{\bar{S},\bar{S}})$$

其中 $\bar{S} = ([1, n + 2m] \cap \mathbb{Z}) \setminus S$ 。

而反对称矩阵中有 $\det(A) = \text{pf}(A)^2$ ，因此：

$$\deg_z \left(\det \begin{pmatrix} Y & A \\ -A^T & X \end{pmatrix} \right) = 2 \deg_z \left(\sum_S \text{pf} \begin{pmatrix} 0 & A \\ -A^T & X \end{pmatrix}_{S,S} \det(Y_{\bar{S},\bar{S}}) \right)$$

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○○○

带权线性拟阵奇偶
○○○○●○○○○○○

例题
○○○○○○

带权线性拟阵奇偶

稀疏型构造——正确性证明

由 Pfaffian 的定义展开可得：

$$\text{pf} \begin{pmatrix} 0 & A \\ -A^T & X \end{pmatrix}_{S,S} = \sum_{T \subseteq S} \text{pf} \begin{pmatrix} 0 & A \\ -A^T & 0 \end{pmatrix}_{T,T} \text{pf} \begin{pmatrix} 0 & 0 \\ 0 & X \end{pmatrix}_{S \setminus T, S \setminus T}$$

因为 X, Y 的非零主子式均互不相同，所以左式中 z 的最高次数一定是通过选择一组合适的 S, T 使得 $\det(Y_{S,S}) \neq 0$, $\text{pf} \begin{pmatrix} 0 & 0 \\ 0 & X \end{pmatrix}_{S \setminus T, S \setminus T} \neq 0$ 且

$\deg_z \left(\text{pf} \begin{pmatrix} 0 & A \\ -A^T & 0 \end{pmatrix}_{T,T} \right)$ 最大所贡献而来。

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○○○

带权线性拟阵奇偶
○○○○●○○○○○○

例题
○○○○○○

带权线性拟阵奇偶

稀疏型构造——正确性证明

由 Y 右下角均为 0 可得 $n+1 \dots n+2m$ 均在 S 中。由 S 左上角均为 0 可得 $1 \dots n$ 均不在 T 中。

因此 $\text{pf} \begin{pmatrix} 0 & 0 \\ 0 & X \end{pmatrix}_{S \setminus T, S \setminus T}$ 非零当且仅当 $\forall i \in [1, m] \cap \mathbb{Z}$, 有 $n+2i, n+2i-1$ 同时在或同时不在 T 中。

而这与原问题的形式是高度吻合的。 $T \cap [n+1, n+2m]$ 刻画了每个二元组是否被选择（在 T 中的元素对应的二元组均被选择，否则不选择）。

固定 $T \cap [n+1, n+2m]$ 。为了使得 $\text{pf} \begin{pmatrix} 0 & A \\ -A^T & 0 \end{pmatrix}_{T,T}$ 确实能够贡献出它所对应的 \deg_z ，我们还需要将 $[1, n] \cap \mathbb{Z}$ 的一个子集加入 T 中以确保其 Pfaffian 非零。

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○○○

带权线性拟阵奇偶
○○○○○○●○○○○○

例题
○○○○○○

带权线性拟阵奇偶

稀疏型构造——正确性证明

若 $\text{pf} \begin{pmatrix} 0 & A \\ -A^T & 0 \end{pmatrix}_{T,T} \neq 0$, 则有:

$$\deg_z \left(\text{pf} \begin{pmatrix} 0 & A \\ -A^T & 0 \end{pmatrix}_{T,T} \right) = \sum_{n+2i \in T} w_i$$

能够找到这样的子集当且仅当所有被选择的向量线性无关, 而这恰好是原题的要求。因此 $v = \max \left(\deg_z \left(\text{pf} \begin{pmatrix} 0 & A \\ -A^T & 0 \end{pmatrix}_{T,T} \right) \right)$ (要求 S, T 满足之前列出的条件)。定理得证。

线性拟阵奇偶
○○○○○○○○○○

带权线性拟阵奇偶
○○○○○○●○○○○○

例题
○○○○○○

带权线性拟阵奇偶

紧凑型构造

考虑将稀疏型构造化为紧凑型构造。我们有如下引理:

引理 (Schur 补) 若 D 可逆, 则如下等式成立:

$$\det \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \det(A - BD^{-1}C) \det(D)$$

由上述引理可得:

$$2v = \deg_z \left(\det \begin{pmatrix} Y & A \\ -A^T & X \end{pmatrix} \right) = \deg_z (\det(Y + AX^{-1}A^T))$$

线性拟阵奇偶
○○○○○○○○○○

带权线性拟阵奇偶
○○○○○○○○●○○○

例题
○○○○○○

带权线性拟阵奇偶

紧凑型构造

令 $x_i = -\frac{1}{x'_i}$ ，展开可得：

$$Y + AX^{-1}A^T = Y + \sum_{i=1}^m -\frac{1}{x'_i} (a_i b_i^T - b_i a_i^T) z^{w_i} = Y + \sum_{i=1}^m x_i (a_i b_i^T - b_i a_i^T) z^{w_i}$$

令此矩阵为 M ，则得到了带权情况下的紧凑型构造，有 $2v = \deg_z(\det(M))$ 。

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○○○

带权线性拟阵奇偶
○○○○○○○○●○○○

例题
○○○○○○

带权线性拟阵奇偶

算法

求 $\det(M)$ 。

未定元 x_i, y_i 均可使用随机取值的方式处理。

M 中每个元素均为关于 z 的不超过 V 次多项式，因此 $\det(M)$ 为关于 z 的不超过 Vn 次多项式。

任选 $Vn+1$ 个不同的 z 的取值 $z_1 \dots z_{Vn+1}$ ，并求出代入每个 z_i 后 $\det(M)$ 的值，再使用拉格朗日插值法还原出 $\det(M)$ 对应的多项式。

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○○○

带权线性拟阵奇偶
○○○○○○○○○○●○○

例题
○○○○○○

带权线性拟阵奇偶

构造方案

维护代入每个 z_i 时的矩阵 $M_1 \dots M_{Vn+1}$ 以及它们的逆 $N_1 \dots N_{Vn+1}$ 。

需要支持对矩阵低秩扰动，并维护其行列式。我们有如下引理：

引理（Matrix Determinant Lemma）若 A, B 可逆，则如下等式成立：

$$\det(A - UB^{-1}V) = \frac{\det(A)}{\det(B)} \det(B - VA^{-1}U)$$

可能在某个时刻出现 M_i 不可逆的情况，即 $\det(M_i) = 0$ 。

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○○○

带权线性拟阵奇偶
○○○○○○○○○○●○○

例题
○○○○○○

带权线性拟阵奇偶

构造方案

由于 $\det(M)$ 为关于 z 的不超过 Vn 次多项式，若我们从 $[0, p) \cap \mathbb{Z}$ 中随机选取 $z_1 \dots z_{Vn+1}$ ，则由 Schwartz-Zippel 引理， $\det(M_i) = 0$ 的情况期望会发生 $O\left(\frac{V^2 mn^2}{p}\right)$ 次。

在发生 $\det(M_i) = 0$ 时更换 z_i 的取值，不断地从 $[0, p)$ 中随机选取新的 z_i ，直到 z_i 与其它 z_j 均不同且 $\det(M_i) \neq 0$ 即可。

若取足够大的 p ，则重构的时间复杂度可以忽略不计。

此时已实时维护出了 $\det(M_1) \dots \det(M_{Vn+1})$ ，每次使用拉格朗日插值法还原出 $\det(M)$ 并进行判断即可。

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○○○

带权线性拟阵奇偶
○○○○○○○○○○●

例题
○○○○○○

带权线性拟阵奇偶

时间复杂度

上述算法时间复杂度组成如下：

- 求矩阵中每个元素 $z_1 \dots z_{V_{n+1}}$ 处的值（多点求值）， $O(Vn^3 \log^2(Vn))$ 。
- 求每个 z_i 处的行列式， $O(Vn^4)$ 。
- 插值还原 $\det(M)$ ， $O(Vn \log^2(Vn))$ 。

若构造方案，则需要增加：

- 求 $N_1 \dots N_{V_{n+1}}$ ， $O(Vn^4)$ 。
- 每次低秩扰动后插值还原 $\det(M)$ ， $O(Vmn \log^2(Vn))$ 。
- 更新 $\det(M_1) \dots \det(M_{V_{n+1}})$ 以及 $N_1 \dots N_{V_{n+1}}$ ， $O(Vmn^3)$ 。

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○○○

带权线性拟阵奇偶
○○○○○○○○○○●

例题
●○○○○○

例题

一般图最大匹配

给定一个无向图，要求选出最多的边满足任意两条选出的边均无公共点。

华东师范大学第二附属中学 郭羽冲
线性拟阵奇偶

线性矩阵奇偶 带权线性矩阵奇偶 例题
○○○○○○○○○○ ○○○○○○○○○○○○ ○●○○○○

一般图最大匹配——算法

解决此问题的经典方法为带花树。

若第 i 条边为 (u_i, v_i) , 则令 $a_i = e_{u_i}, b_i = e_{v_i}$.

若选出的边有公共点 u , 则 e_u 这个向量一定出现了至少两次, 一定线性相关。否则每个 e_u 至多出现一次, 一定线性无关。

有趣的是，观察我们得到的矩阵 M ，与一般图最大匹配的另一经典解法：Tutte 矩阵 完全一致。

令边数为 n ，点数为 m 。则直接使用通用算法构造方案的时间复杂度为 $O(mn^2)$ 。

华东师范大学第二附属中学 郭羽冲
线性回归方程

线性矩阵奇偶 带权线性矩阵奇偶 例题

一般图最大匹配——优化

考虑利用 a_i, b_i 的性质进一步优化, 我们有如下引理:

引理 (Harvey) 若 A 可逆, 且 A 与 B 只有集合 S 对应的主子矩阵不同。令 $D = B - A$, 则有:

$$(B^{-1})_{T,T} = (A^{-1})_{T,T} - (A^{-1})_{T,S}(I + D_{S,S}(A^{-1})_{S,S})^{-1}D_{S,S}(A^{-1})_{S,T}$$

且 B 可逆当且仅当 $I + D_{S,S}(A^{-1})_{S,S}$ 可逆。

Harvey 引理启示我们，只需要知道局部的信息即可处理局部的修改。

而在一般图最大匹配问题中, (a_i, b_i) 只会影响 $M_{u_i, v_i}, M_{v_i, u_i}$ 这两个位置。于是考虑分治地进行计算。

华东师范大学第二附属中学 郭羽冲
线性回归方程

线性拟阵奇偶
○○○○○○○○○○

带权线性拟阵奇偶
○○○○○○○○○○○○

例题
○○○●○○

例题

一般图最大匹配——优化

定义过程 $\text{solve}(S, T)$ 表示处理所有满足 $u_i \in S, v_i \in T$ 的二元组，判断它们是否在答案中。

令 $X = S \cup T$ ，则只需要关注 $M_{X,X}$ 以及 $(M^{-1})_{X,X}$ 。

若 $|S| = |T| = 1$ ，则直接暴力，时间复杂度为 $O(1)$ 。

否则将 S 均分为 S_1, S_2 ，将 T 均分为 T_1, T_2 。依次枚举每一对 $(i, j) \in \{(1, 1), (1, 2), (2, 1), (2, 2)\}$ ，并进行如下过程：

- 递归 $\text{solve}(S_i, T_j)$ 。
- 在上一步中我们求出了 S_i, T_j 对应的子问题中所有不在答案中的二元组。删除它们后会将 M 变为 $M + D$ ，显然 D 只可能在 X 对应的主子矩阵内有值。因此利用 Harvey 引理在 $O(|X|^3)$ 的时间复杂度内计算出 D 对 $(M^{-1})_{X,X}$ 的影响即可。

华东师范大学第二附属中学 郭羽冲

线性拟阵奇偶

线性拟阵奇偶
○○○○○○○○○○

带权线性拟阵奇偶
○○○○○○○○○○○○

例题
○○○●○○

例题

一般图最大匹配——时间复杂度

为了方便分析时间复杂度，不妨令 $S = T$ 或 $S \cap T = \emptyset$ ，令两种情况的时间复杂度分别为 $T_1(|S|), T_2(|S| + |T|)$ ，则有：

$$T_2(n) = O(n^3) + 4T_2\left(\frac{n}{2}\right) = O(n^3)$$

$$T_1(n) = 2T_1\left(\frac{n}{2}\right) + 2T_2(n) + O(n^3) = O(n^3)$$

因此 $\text{solve}(\{1, 2, \dots, n\}, \{1, 2, \dots, n\})$ 的时间复杂度为 $T_1(n) = O(n^3)$ 。

华东师范大学第二附属中学 郭羽冲

线性拟阵奇偶

浅谈静态数据结构的合并与分裂

中国人民大学附属中学 黄洛天

线段树合并

线段树合并大家都了解，那么什么样的信息是线段树合并可以维护的？如何抽象出信息的性质？

为了维护信息，对于暴力的合并的节点我们有两种选择：

- ① 在合并完成子树后，从合并的后的两个儿子的信息合并上来。
- ② 直接合并对应节点的信息。

Examples

对于 01 序列，维护区间中 $[010, 01, 10, 0, 1]$ 子序列的出现次数。

Examples

线段树套平衡树，平衡树支持对 key 的平移和区间查询。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

线段树合并

对于带懒标记的线段树合并，由于动态开点线段树在下放标记时会建立新的节点从而影响时间复杂度，因此我们需要用一些方法限制住新建点的数量。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

线段树合并

对于我们正在维护的动态开点线段树，时刻保持每个节点要么没有儿子，要么有两个儿子。这样对于非叶子节点下放标记不会新建节点。于是问题变成如何处理叶子节点与线段树子树的合并。一个想法是由于叶子节点代表的区间往往有统一的性质，因此考虑提炼出这个统一的性质并作为标记打到另一个子树上。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

点分树、边分树合并

类似于动态开点线段树，我们可以定义动态开点的点分树、边分树。为了方便叙述，我们只讨论点分树，边分树也是类似的。由于点分树每个节点有很多儿子，我们难以支持自上而下的检索，一些信息也难以维护。因此考虑将点分树的儿子建哈夫曼树，不难证明树高是 $O(\log n)$ 的。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

点分树、边分树合并

对于一类点分树问题，经常需要维护重心到当前连通块其他点的信息。那么对于每个节点 u ，他会影响的节点就是点分树上的父亲。由此可见如果我们做单点修改操作只有 $O(\log n)$ 个节点在点分树上维护的信息会变化。

不同于线段树，点分树不支持从儿子上维护的信息合并到当前节点，因此点分树只能采用前文的第二种合并。合并方法同线段树，自顶向下递归即可。

黄洛天

中国人民大学附属中学

浅谈静态数据结构的合并与分裂

静态 Top Tree 合并

Definition

对于无向树 T ，定义 T 上一个簇为一个三元组 $C = (V, E, B)$ ，其中 (V, E) 连通且是 T 的子图，集合 $B \subseteq V, |B| \leq 2$ ，且对于每个 $x \in V$ ，若 $\exists y \notin V, (x, y) \in E(T)$ ，则 $x \in B$ 。 B 中元素称为 C 中的界点。我们用 $V(C), E(C)$ 分别表示 C 的点集和边集，用 $B(C)$ 表示 C 中的界点集合。

黄洛天

中国人民大学附属中学

浅谈静态数据结构的合并与分裂

静态 Top Tree 合并

Definition

compress 操作是指将两个簇合并为一个簇的操作。其中，将两个界点集合交为 1 的簇合并成一个簇，新的界点集合为两个界点集合的对称差。

Definition

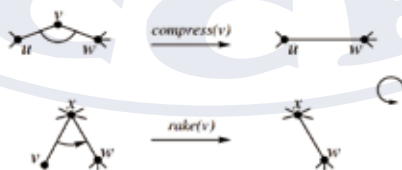
rake 操作是指将两个簇合并为一个簇的操作。其中，将两个界点集合交为 1 的簇合并成一个簇，新的界点集合为其中一个簇的界点集合。

黄洛天

中国人民大学附属中学

浅谈静态数据结构的合并与分裂

静态 Top Tree 合并



对于一棵树来说，我们可以用上面两种操作，将整棵树进行收缩。具体来说，可以将一个簇看成是缩成一条界点 (u, v) 之间的边，初始每一条 (u, v) 的边都看成一个簇，通过用上面两种操作不断合并簇，最后可以把整棵树合并成一个簇。

收缩过程中，簇的合并结构形成了一个树，我们把这个树称为 Top Tree。

黄洛天

中国人民大学附属中学

浅谈静态数据结构的合并与分裂

静态 Top Tree 合并

如何构建静态 Top Tree ?

对原树轻重链剖分，dfs 地自底向上处理每一个重链，并认为每一个轻子树都已经合并成一个簇。对于重链上的每一个节点，将这个节点的父亲的轻子树的簇分治地 rake，得到的总簇再用一次 rake 操作合并到这个节点的父边。需要保证深度不会太大，考虑每次分成两半，rake 完两半之后再 rake 成一个簇。我们称这样操作形成的树为 rake tree。然后，我们得到了排成一条链的簇。接着我们就可以将这些簇 compress 为一个，同样需要分治操作来保证深度，得到的树称为 compress tree。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

静态 Top Tree 合并

Lemma

如果我们按簇的大小选择带权中点分治，那么这样建立出来的 Top Tree 深度为 $O(\log n)$ 。

无论是 rake tree 还是 compress tree，每向上跳常数层，簇的大小至少会翻倍，因此深度为 $O(\log n)$ 。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

静态 Top Tree 合并

做链定位时需要注意以下 case。

先定位到 rake tree 或 compress tree 上，设要定位到 u 到 v 之间的区间。

- ① u 或 v 是另一方的祖先。
- ② u 或 v 均不是另一方的祖先。
- ③ u, v 分别在重儿子左右的两个 rake tree 上。
- ④ u, v 其中一个是重儿子，即不在 rake tree 上。
- ⑤ u, v 在同一个 rake tree 上。

这五类描述了静态 Top Tree 定位过程中遇到的各种情况，对于每类如何计算贡献并不难，在本文不详细展开。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

静态 Top Tree 合并

现在有了动态开点的静态 Top Tree，于是考虑怎么合并两个动态开点的静态 Top Tree。我们仍然自上而下的去合并 T_1, T_2 。

对于非叶子节点之间的合并，沿用之前线段树合并的思想，暴力递归下去。

我们要面对叶子节点和一个子树的合并。此时叶子节点上面带有两种标记——簇界点之间路径的标记和整个簇除了前者的标记。较为常见的情况是可以直接将两种标记打给另一个静态 Top Tree 的对应节点上。但如果不行需要针对簇界点之间路径的标记和整个簇除了前者的两套信息设计一个势能函数。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

线段树分裂

线段树分裂支持把线段树分裂成两个值域不交的部分。注意线段树分裂与线段树合并并不是逆操作的关系。

具体而言如果需要把序列 a_i 划分为

$$b_i = \begin{cases} a_i & i \leq p \\ 0 & i > p \end{cases}, c_i = \begin{cases} 0 & i \leq p \\ a_i & i > p \end{cases}, \text{ 同时需要维护 } b_i, c_i \text{ 的区间}$$

信息，考虑通过对维护 a 序列的线段树分裂实现。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

线段树分裂

注意到如果线段树的某个节点代表的区间 $[l, r]$ 满足 $r \leq p$ ，则这个区间的信息可以完全一样的复制到 b 序列对应的线段树节点上；同理若 $l > p$ ，则区间信息可以复制到 c 序列对应的线段树节点上。两者都不满足的节点显然只有 $O(\log n)$ 个，把他们复制成两份放在对应位置。

此做法只会增加 $O(\log n)$ 个节点，因此原来的势能分析仍然成立。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

点分树、边分数分裂

点分树、边分树的局限性

数据结构分裂之后信息从儿子合并而来是很简单的。但如果想直接用原来的信息分裂，则必须维护的信息支持分裂。因此点分树边分树分裂对信息的要求高并且应用范围窄，不作为本文讨论的重点。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

静态 Top Tree 分裂

考虑静态 Top Tree 的分裂出子树和子树补。

假设我们要分裂出 u 子树，可以在定位出 u 子树在静态 Top Tree 上对应的子树。即 u 节点所在 compress tree 中深度在 u 后面的节点，及其的 rake tree 子树，这一部分可以完整的分裂出来。对于 u 在静态 Top Tree 上的祖先节点由于子树内一部分在原树上的 u 子树，另一部分不在，因此需要复制到两棵树上。

每次只会新建 $O(\log n)$ 个节点。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

静态 Top Tree 分裂

考虑静态 Top Tree 的分裂出子树和子树补。

假设我们要分裂出 u 子树，可以在定位出 u 子树在静态 Top Tree 上对应的子树。即 u 节点所在 compress tree 中深度在 u 后面的节点，及其的 rake tree 子树，这一部分可以完整的分裂出来。对于 u 在静态 Top Tree 上的祖先节点由于子树内一部分在原树上的 u 子树，另一部分不在，因此需要复制到两棵树上。每次只会新建 $O(\log n)$ 个节点。

黄洛天

中国人民大学附属中学

浅谈静态数据结构的合并与分裂

静态 Top Tree 分裂

处理完了子树的分裂，另一个想法是处理链的分裂。

面临的问题

也就是说要将静态 Top Tree 中的一部分 compress tree 复制到 T' 的对应位置，但是这些 compress tree 连接的 rake tree 并不应该复制到 T' 的对应位置，而是应该保留在原来的位置。因此 T' 需要复制的节点并不是若干子树的形式，暴力剥离掉 compress tree 连接的 rake tree 的时间代价是不可接受的。

黄洛天

中国人民大学附属中学

浅谈静态数据结构的合并与分裂

静态 Top Tree 分裂

由于静态 Top Tree 本身的结构导致了此问题，因此需要扩展出一个新的结构。由于本题不需要维护子树信息，因此对于 compress node 的 rake tree 子树内的信息并不关键。也就是说不一定要让 rake tree 作为当前 compress node 的儿子。

考虑维护双层静态 Top Tree，将原本的每个点划分为 u 和 u' ，分别表示第一层和第二层。第一层内部的结构同静态 Top Tree 一样。第二层内部只建立 compress tree。对于每个 compress tree 的根节点 rt 有额外的一条边指向 rt' 。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

静态 Top Tree 分裂

只有第二层维护链信息，第一层仅作为定位和重量平衡的用途。

不难发现上述结构，可以很好的维护链修改查询，并且具有每个节点儿子个数 $O(1)$ 且树高为 $O(\log n)$ 的性质。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

静态 Top Tree 分裂

有了动态开点的双层静态 Top Tree，就可以合并和支持分裂出一个链了。前者是类似于普通静态 Top Tree 的。后者考虑我们要分裂的链在第二层的若干 compress tree 的区间上，而第二层的 compress tree 并不包含多余的 rake tree，因此这些子树可以被直接复制为 T' 的对应节点。这些 compress tree 上共用的节点，包括第一层静态 Top Tree 上公用的节点总数为 $O(\log n)$ ，可以新建出来。对于其余节点则保留了除了链以外的信息。

势能函数设计为 $\Phi = \sum |T_i|$ ，即静态 Top Tree 的节点个数，可以分析得到时间复杂度为 $O(n \log n)$ 。

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

参考文献

- 2023 年集训队论文, 程思元, 《浅谈静态 Top Tree 在树和广义串并联图上的应用》
- 赵雨扬, 《Top tree 相关东西的理论、用法和实现》, <https://uoj.ac/blog/4912>

黄洛天

浅谈静态数据结构的合并与分裂

中国人民大学附属中学

浅谈 -analogue

湖南省长沙市第一中学 李铭乐洋

基本 q -analogue

在 q -analogue 中，我们有时将 q 视作满足 $0 < q < 1$ 的实参数。

对于实数 x ，我们定义其 q -analogue 为 $[x] = \frac{1-q^x}{1-q}$ 。此处，根据洛必达法则，当 $q \rightarrow 1$ 时 $[x] \rightarrow x$ 。

类似地定义 q -factorial：对于正整数 n ，我们定义 $n!$ 的 q -analogue 为

$$[n]! = \prod_{i=1}^n [i] = \prod_{i=1}^n \frac{1-q^i}{1-q}.$$

partition 与 Ferrers graph

我们记 partition 为序列 λ 满足 $\lambda_1 \geq \lambda_2 \geq \dots$ ，接着记 $\ell(\lambda)$ 为最大的 x 使得 $\lambda_x > 0$ ，然后记 $|\lambda|$ 为 $\sum \lambda_i$ 。

对于一个 partition 定义其 Ferrers graph 为：共 $\ell(\lambda)$ 行，从下往上第 i 行长度为 λ_i ，且所有行均左对齐的图形。

Gaussian polynomial

我们对于 $n, k \in \mathbb{N}, 0 \leq k \leq n$ 定义 Gaussian polynomial 为：

$$\begin{bmatrix} n \\ k \end{bmatrix} = \begin{bmatrix} n \\ k \end{bmatrix}_q = \frac{[n]!}{[n-k]![k]!} = \frac{(1-q^n)(1-q^{n-1}) \cdots (1-q^{n-k+1})}{(1-q^k)(1-q^{k-1}) \cdots (1-q)}.$$

这是 q -binomial coefficients 的特殊情况。后者是对于 $x \in \mathbb{R}, k \in \mathbb{N}$ 定义的，有：

$$\begin{bmatrix} x \\ k \end{bmatrix} = \frac{(q^{x-k+1}; q)_k}{(q; q)_k}$$

其中 $(a; q)_k = (a)_k = (1-a)(1-qa) \cdots (1-q^{k-1}a)$ ，被称做 q -rising factorial。

Gaussian polynomial

而事实上 Gaussian polynomial 是 q 中的多项式，因为我们对于 $n, k \in \mathbb{N}$ 有：

$$\begin{bmatrix} n+k \\ k \end{bmatrix} = \sum_{\lambda \subseteq n^k} q^{|\lambda|}$$

其中 $\lambda \subseteq n^k$ 即 $\lambda_1 \leq n, \ell(\lambda) \leq k$ 。

Gaussian polynomial

考虑归纳。我们令 $P(n, k) = \sum_{\lambda \subseteq n^k} q^{|\lambda|}$ ，即有：

$$P(n, k) = \sum_{\lambda \subseteq n^k, \lambda_1 = n} q^{|\lambda|} + \sum_{\lambda \subseteq n^k, \lambda_1 < n} q^{|\lambda|} = q^n P(n, k-1) + P(n-1, k)$$

边界条件 $P(n, 0) = P(0, k) = 1$ 是显而易见的。

Gaussian polynomial

另一方面：

$$\begin{aligned} q^n \begin{bmatrix} n+k-1 \\ k-1 \end{bmatrix} + \begin{bmatrix} n-1+k \\ k \end{bmatrix} &= q^n \frac{[n+k-1]!}{[k-1]![n]!} + \frac{[n+k-1]!}{[k]![n-1]!} \\ &= \frac{[n+k-1]!}{[k]![n]!} \left(q^n \frac{1-q^k}{1-q} + \frac{1-q^n}{1-q} \right) \\ &= \frac{[n+k]!}{[k]![n]!} \end{aligned}$$

The Catalan numbers and Dyck paths

我们记所有 $(0, 0)$ 到 (n, m) 的格路集合为 $L_{n,m}$ ，以及 $L_{n,m}^+ \subseteq L_{n,m}$ 为那些永远不低于到 $y = \frac{n}{m}x$ 的格路。对于任意的 $\pi \in L_{n,m}^+$ 我们称 π 是一个 Dyck path，又称作 Catalan path。

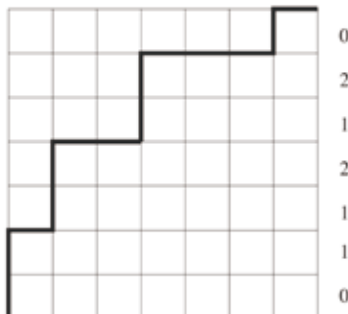
我们记 $C_n = \frac{1}{n+1} \binom{2n}{n}$ 为 Catalan number。Catalan 数的一种组合解释是 $L_{n,n}^+$ 中的元素个数即 $|L_{n,n}^+| = C_n$ 。

The area statistic

对于 $\pi \in L_{n,n}^+$ ，我们给出如下定义：

- 记 $\sigma(\pi)$ 为：初始为空，从 $(0, 0)$ 开始，每走一步 North (即 $(1, 0)$) 就往后加入字符 0，每走一步 East (即 $(0, 1)$) 就往后加入字符 1。最终走到 (n, n) 结束时得到的序列。
- 记 $a_i(\pi)$ 为：第 i 行中，在 π 右侧、在 $y = x$ 左侧的完整格子个数。这里我们还将 $a_i(\pi)$ 称做 π 的第 i 行的长度。同时， π 的 area statistic 被定义为 $\sum a_i(\pi)$ (记做 $\text{area}(\pi)$)。

The area statistic



上图每行右侧的值即为对应的 $a_i(\pi)$ 。

C_n 的 q -analogue

我们有：

$$\sum_{\pi \in L_{n,n}^+} q^{\text{maj}(\sigma(\pi))} = \frac{1}{[n+1]} \begin{bmatrix} 2n \\ n \end{bmatrix}$$

C_n 的 q -analogue

考虑所有非法路径 $L_{n,n}^- = L_{n,n} \setminus L_{n,n}^+$ 。对于任一 $\pi \in L_{n,n}^-$ ，我们找到其经过的格点中， $j-i$ 最小的（如有多个，选择最早的一个），记为 P 。

那么到达 P 的一步一定是 East。将这一步 East 变成 North，后面的部分平移，得到了一个 $\pi' \in L_{n-1,n+1}$ 。可以看到此时 $\text{maj}(\sigma(\pi'))$ 相比 $\text{maj}(\sigma(\pi))$ 少了 1。

事实上，对于任一 $\pi' \in L_{n-1,n+1}$ ，只要找到 $j-i$ 最小的点（如有多个，选择最晚的一个）。下一步必然是 North，将其改为 East，后面的部分平移即可。

C_n 的 q -analogue

那这样 $L_{n-1,n+1}$ 和 $L_{n,n}^-$ 中的路径就是一一对应的了，即有：

$$\sum_{\pi \in L_{n,n}^-} q^{\text{maj}(\sigma(\pi))} = \sum_{\pi \in L_{n-1,n+1}} q^{\text{maj}(\sigma(\pi))+1} = q \begin{bmatrix} 2n \\ n-1 \end{bmatrix}$$

于是就有：

$$\begin{aligned} \sum_{\pi \in L_{n,n}^+} q^{\text{maj}(\sigma(\pi))} &= \sum_{\pi \in L_{n,n}} q^{\text{maj}(\sigma(\pi))} - \sum_{\pi \in L_{n,n}^-} q^{\text{maj}(\sigma(\pi))} \\ &= \begin{bmatrix} 2n \\ n \end{bmatrix} - q \begin{bmatrix} 2n \\ n-1 \end{bmatrix} \\ &= \frac{1}{[n+1]} \begin{bmatrix} 2n \\ n \end{bmatrix} \end{aligned}$$

C_n 的 q -analogue

然后来看另一个 C_n 的 q -analogue:

$$C_n(q) = \sum_{\pi \in L_{n,n}^+} q^{\text{area}(\pi)}$$

其满足:

$$C_n(q) = \sum_{k=1}^n q^{k-1} C_{k-1}(q) C_{n-k}(q), n \geq 1$$

C_n 的 q -analogue

考虑 $(0,0)$ 出发后, 第一次碰到 $y=x$ 的位置为 (k,k) 。那么 $(0,1)$ 到 $(k-1,k)$ 的路径可以被 $q^{k-1}C_{k-1}(q)$ 描述, (k,k) 到 (n,n) 的路径可以被 $C_{n-k}(q)$ 描述。

The bounce statistic

对于路径 $\pi \in L_{n,n}^+$ 定义其 bounce path 为如下算法生成的路径:

从 $(0,0)$ 开始, 跟着 π 走直到遇见一个 E, 转向并走到 $y=x$ 。此时再次转向, 跟着 π 走直到再次遇见某个 E, 重复上述过程。

可以将 bounce path 想象成从 $(0,0)$ 出发、初始向 N 走的、碰到 π 或 $y=x$ 会反弹的小球走出的路径。

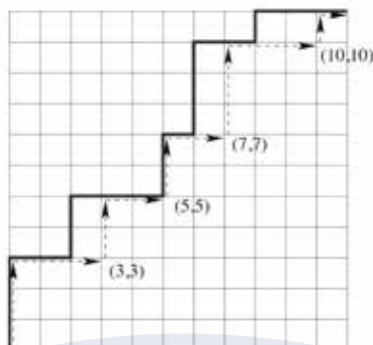
The bounce statistic

我们记下所有碰到 $y=x$ 的位置:

$$(0,0), (j_1, j_1), \dots, (j_b, j_b) = (n, n)$$

我们称 b 为反弹次数, 称 j_1 为第一次反弹的长度, 称 $j_2 - j_1$ 为第二次反弹的长度 .. 剩下的以此类推。

The bounce statistic



如图所示。

The bounce statistic

对此，我们定义 bounce statistic $\text{bounce}(\pi)$ 为：

$$\text{bounce}(\pi) = \sum_{i=1}^{b-1} n - j_i$$

q, t -Catalan numbers

我们定义：

$$F_n(q, t) = \sum_{\pi \in L_{n,n}^+} q^{\text{area}(\pi)} t^{\text{bounce}(\pi)}$$

q, t -Catalan numbers

记 $L_{n,n}^+(k)$ 为满足 $\pi \in L_{n,n}^+$ 且恰好以 k 个 N 开头，然后紧接着下一步是 E 的那些 π 的集合。

特别地，对于 $L_{0,0}^+(k)$ 。当 $k = 0$ 时其仅包含空路径，否则为空。

我们定义：

$$F_{n,k}(q, t) = \sum_{\pi \in L_{n,n}^+(k)} q^{\text{area}(\pi)} t^{\text{bounce}(\pi)}$$

特别地， $F_{n,0}$ 仅在 $n = 0$ 时取 1，否则取 0。

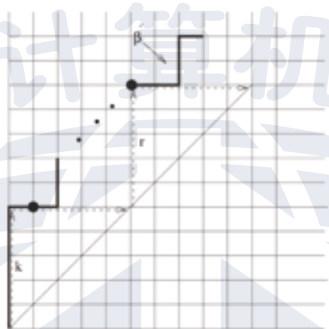
q, t -Catalan numbers

那么我们有如下递归关系：

$$F_{n,k}(q, t) = \sum_{r=0}^{n-k} \begin{bmatrix} r+k-1 \\ r \end{bmatrix}_q t^{n-k} q^{\binom{k}{2}} F_{n-k,r}(q, t)$$

q, t -Catalan numbers

考虑反弹至少两次的 $\beta \in L_{n,n}^+(k)$ ，其满足第一次反弹长度为 k ，第二次反弹长度为 r ——这意味着其经过点 $(1, k), (k, k+r)$ ，即下图中的两个大点：



q, t -Catalan numbers

将 β 分成两个部分，第一个部分是 $(0, 0)$ 到 $(k, k+r)$ ，第二个部分是 $(k, k+r)$ 到 (n, n) 。

对于第二个部分，我们在其前面加入 r 个 N 并记其为 β' 。此时有 $\beta \in L_{n-k, n-k}^+(r)$ ，同时也有 $\text{bounce}(\beta') = \text{bounce}(\beta) + n - k$ 。

q, t -Catalan numbers

而面积可以拆成三部分： $x = k$ 右侧的面积， $y = k$ 下侧的面积以及剩余面积。

1. 第一部分只和 β' 有关系。
2. 第二部分一定是 $\binom{k}{2}$ 。
3. 第三部分，只要考虑 $(1, k)$ 走到 $(k, k+r)$ 的方案。根据定理 3.2，所有方案的面积和为 $\begin{bmatrix} r+k-1 \\ r \end{bmatrix}_q$ 。

因此有：

$$q^{\text{area}(\beta)} = q^{\text{area}(\beta')} q^{\binom{k}{2}} \begin{bmatrix} r+k-1 \\ r \end{bmatrix}_q$$

q, t -Catalan numbers

于是就有：

$$\begin{aligned} F_{n,k}(q, t) &= \sum_{r=0}^{n-k} \sum_{\beta' \in L_{n-k, n-k}^+(r)} q^{\text{area}(\beta')} q^{\binom{k}{2}} \begin{bmatrix} r+k-1 \\ r \end{bmatrix}_q t^{\text{bounce}(\beta')} t^{n-k} \\ &= \sum_{r=0}^{n-k} \begin{bmatrix} r+k-1 \\ r \end{bmatrix}_q t^{n-k} q^{\binom{k}{2}} F_{n-k,r}(q, t) \end{aligned}$$

而对于 $n = k$ 的情况（即唯一需要考虑反弹恰好一次的情况）上述结果是显然的。

q, t -Catalan numbers

另外，我们有：

$$F_n(q, t) = \sum_{b=1}^n \sum_{\alpha_1 + \dots + \alpha_b = n, \alpha_i > 0} t^{\sum_{j=2}^b (j-1)\alpha_j} q^{\sum_{j=1}^b \binom{\alpha_j}{2}} \prod_{i=1}^{b-1} \begin{bmatrix} \alpha_i + \alpha_{i+1} - 1 \\ \alpha_{i+1} \end{bmatrix}_q$$

对称性问题

我们有：

$$\sum_{\pi \in L_{n,n}^+} q^{\text{area}(\pi)} = \sum_{\pi \in L_{n,n}^+} q^{\text{bounce}(\pi)}$$

对称性问题

对于 $\pi \in L_{n,n}^+$ ，其 $a_i(\pi)$ 需要满足的必要条件是 $a_i(\pi) \leq a_{i-1}(\pi) + 1$ 。于是，我们转而枚举 $a_i(\pi) = j$ 的次数，分别记为 b_0, b_1, \dots, b_k 。接着考虑它们组合的方案数计算如下：

先将 b_0, b_1 组合，方案数为 $\binom{b_0+b_1-1}{b_1}$ 。然后将 b_2 加入，不能出现 0 后面紧邻一个 2 的情况。方案数为 $\binom{b_1+b_2-1}{b_2}$.. 以此类推。

因此，我们可以得到：

$$\sum_{\pi \in L_{n,n}^+} q^{\text{area}(\pi)} = \sum_{k=1}^n \sum_{b_1 + \dots + b_k = n, b_i > 0} q^{\sum_{i=2}^k b_i(i-1)} \prod_{i=1}^{k-1} \binom{b_i + b_{i+1} - 1}{b_{i+1}}$$

结果与 $F_n(1, q)$ 一致。

The dinv statistic

我们接下来介绍另外一个关于 q, t -Catalan 的 Dyck path 的 statistic。

对于 $\pi \in L_{n,n}^+$ 有：

$$\text{dinv}(\pi) = |\{(i, j) : 1 \leq i < j \leq n, a_i(\pi) = a_j(\pi)\}| + |\{(i, j) : 1 \leq i < j \leq n, a_i(\pi) = a_j(\pi) + 1\}|$$

对此我们有：

$$\sum_{\pi \in L_{n,n}^+} q^{\text{dinv}(\pi)} t^{\text{area}(\pi)} = \sum_{\pi \in L_{n,n}^+} q^{\text{bounce}(\pi)} t^{\text{area}(\pi)}$$

The dinv statistic

我们构造 Dyck path 的双射 ζ 使其满足：

$$\begin{aligned} \text{dinv}(\pi) &= \text{area}(\zeta(\pi)) \\ \text{area}(\pi) &= \text{bounce}(\zeta(\pi)) \end{aligned}$$

对于 $\pi \in L_{n,n}^+$ ，令 $b-1$ 为其最长行的长度（即 $\max a_i(\pi)$ ），记 c_0, \dots, c_{b-1} 为每个长度对应的出现次数。

我们构造对应路径 π' 使其有恰好 b 次反弹，且反弹长度依次为 c_0, \dots, c_{b-1} 。对于每个 $0 \leq i < b-1$ 的 $(c_i, c_i + c_{i+1})$ 处，我们构造其与 $(c_{i-1}, c_{i-1} + c_i)$ （当 $i=0$ 即 $(0,0)$ ）之间的路径：从上到下考虑所有 $a_i(\pi)$ ，若是 $i+1$ 则加入一个 S（South step），若是 i 则加入一个 W（West step），其他情况不做考虑。

The dinv statistic



如图所示，左侧的 π 对应了右侧的 π' 。

The dinv statistic

ζ 很显然是一个双射：对于 π' ，我们通过反弹次数得到了 $b-1$ ，并且得到了每个长度的出现次数。此时再看具体路径，对于 $0 \leq i < b-2$ 可以得到仅有 $i, i+1$ 的子序列。而组合原路径的方案是唯一的，这和定理 4.3 的证明是对应的。

我们将 $\text{area}(\pi')$ 按照 bounce path 分成两部分：下侧是 $\binom{c_0}{2} + \dots + \binom{c_{b-1}}{2}$ ，对应了 $\text{dinv}(\pi)$ 中的 $|\{(i, j) : 1 \leq i < j \leq n, a_i(\pi) = a_j(\pi)\}|$ 。上侧，分析每一段，不难发现也恰好对应了 $|\{(i, j) : 1 \leq i < j \leq n, a_i(\pi) = a_j(\pi) + 1\}|$ 。

因此 $\text{area}(\pi') = \text{dinv}(\pi)$ 。

The dinv statistic

另一方面，我们有：

$$\text{bounce}(\pi') = \sum_{i=1}^{b-1} ic_i = \text{area}(\pi)$$

The dinv statistic

一个推论是：

$$F_{n,k}(q, t) = \sum_{\pi \in L_{n,n}^+, \pi \text{ has exactly } k \text{ rows of length } 0} q^{\text{dinv}(\pi)} t^{\text{area}(\pi)}$$

这是因为 ζ 对应的限制即为有恰好 k 行长度为 0（即第一次反弹长度为 k ）。

范围修改查询问题相关算法及其应用

杭州学军中学教育集团文渊中学 沈吉源

引入
●○○

抽象信息
○○

离线范围修改查询
○○○○○○○○○○○○○○

例题
○○○

半平面范围查询问题
○○○○○○

一个经典问题

从一个经典问题引入：

问题

维护一个序列，序列上的每个元素为一个 2×2 矩阵，要求做以下操作：

修改操作：给定 l, r, v ，将区间 $[l, r]$ 中的矩阵乘上矩阵 v 。

查询操作：给定 l, r ，查询区间 $[l, r]$ 的矩阵之和。

沈吉源
范围修改查询问题相关算法及其应用

杭州学军中学教育集团文渊中学

引入
●●○

抽象信息
○○

离线范围修改查询
○○○○○○○○○○○○○○

例题
○○○

半平面范围查询问题
○○○○○○

上述问题是线段树的经典问题，可以使用带懒惰标记的线段树解决。对于线段树上的每个节点，需要维护区间矩阵和，以及对区间中的矩阵乘的一个懒标记。

现在我们考虑抽象数据结构问题：不考虑具体维护什么信息，考虑在维护的信息满足某些特定条件下的通用问题形式。

沈吉源
范围修改查询问题相关算法及其应用

杭州学军中学教育集团文渊中学

引入
●●○

抽象信息
○○

离线范围修改查询
○○○○○○○○○○○○○○

例题
○○○

半平面范围查询问题
○○○○○○

引入 ○○●	抽象信息 ○○	离线范围修改查询 ○○○○○○○○○○○○○○○○	例题 ○○○	半平面范围查询问题 ○○○○○○
-----------	------------	------------------------------	-----------	---------------------

对于线段树能维护问题，抽象后的问题是：

维护一个序列，可以用某种数据表示序列上一个区间的信息，可以用某种数据表示序列区间的一次修改，需要满足的性质：

1. 序列上两个区间 $[l, mid]$, $[mid + 1, r]$ 的信息可以 $O(1)$ 合并出 $[l, r]$ 的信息。
2. 对一个区间进行的相邻两次修改操作可以 $O(1)$ 合并为一次修改操作。
3. 如果知道了一个区间的信息，可以 $O(1)$ 计算出这个区间经过一次修改后的信息。

矩阵的乘法满足结合律 $(A \times B) \times C = A \times (B \times C)$ 与分配律 $A \times (B + C) = A \times B + A \times C$ ，但不满足交换律。

在上面的问题中，信息的合并是矩阵加法；而结合律、分配律分别使这个问题能满足 2 和 3 的性质。

沈吉源	杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用	

引入 ○○○	抽象信息 ●○○	离线范围修改查询 ○○○○○○○○○○○○○○○○	例题 ○○○	半平面范围查询问题 ○○○○○○
抽象信息的定义				

定义半群为：

给定一个集合 D ，以及 D 上的一个二元运算： $D * D \rightarrow D$ 。

运算满足结合律： $\forall a, b, c \in D, (a * b) * c = a * (b * c)$ 。

定义幺半群为：

给定一个集合 D ，以及 D 上的一个二元运算： $D * D \rightarrow D$ 。

运算满足结合律： $\forall a, b, c \in D, (a * b) * c = a * (b * c)$ 。

有幺元： $\forall a \in D, a * \epsilon = \epsilon * a = a$ 。

定义交换半群为：

给定一个集合 D ，以及 D 上的一个二元运算： $D * D \rightarrow D$ 。

运算满足结合律： $\forall a, b, c \in D, (a * b) * c = a * (b * c)$ 。

运算满足交换律： $\forall a, b \in D, a * b = b * a$ 。

沈吉源	杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用	

引入 ○○○	抽象信息 ○●	离线范围修改查询 ○○○○○○○○○○○○○○○○	例题 ○○○	半平面范围查询问题 ○○○○○○
-----------	------------	------------------------------	-----------	---------------------

前面的例子维护的信息可以看作半群信息。
可以发现维护的是双半群信息，一个表示区间的信息，一个表示修改的懒标记的信息。
不难发现，只要区间的信息与修改的信息均为半群信息，就可以用线段树维护。

沈吉源 杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用

引入 ○○○	抽象信息 ○○	离线范围修改查询 ●○○○○○○○○○○○○○○	例题 ○○○	半平面范围查询问题 ○○○○○○
-----------	------------	-----------------------------	-----------	---------------------

更广的范围

在问题中，修改和询问的范围不一定是序列的区间，而可能为树简单路径，二维平面矩形，高维正交范围，半平面范围，曲线范围等。
初始有 n 个点，一个范围代表了初始点集的一个子集。
可以用一个信息表示范围内点的信息和，用一个信息表示对范围的一个修改。这两个信息满足的条件是：

1. 两个范围的信息 S, T 可以 $O(1)$ 合并成一个信息 $S + T$ 。
2. 对一个范围的相邻的两次修改可以 $O(1)$ 合并成一次修改。
3. 如果知道了一个范围的信息，可以 $O(1)$ 计算出这个范围经过一次修改后的信息。

沈吉源 杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用

问题的形式

初始给出 n 个点的一个点集，每个点有一个初始权值。
每次修改给一个范围以及一个修改的值，表示把范围内的点的权值与修改权值做一个二元运算。
每次查询给一个范围，求范围内的点的权值求和后得到的值。其中点的权值和修改的权值满足双半群的性质。

沈吉源 杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用

问题的形式

形式化的问题如下：
给定交换半群 $(\mathcal{D}, +)$ ，半群 $(\mathcal{M}, *)$ ，二元运算符 $*$ ：
 $\mathcal{D} * \mathcal{M} \rightarrow \mathcal{D}$ 。
其中二元运算符满足：
结合律： $\forall a \in \mathcal{D}, b, c \in \mathcal{M}, (a * b) * c = a * (b * c)$ 。
分配律： $\forall a, b \in \mathcal{D}, c \in \mathcal{M}, (a + b) * c = a * c + b * c$ 。
给定一个初始集合 I_0 ，一个有限集合 $I \subseteq I_0$ ，询问集合 Q 。
 I 中每个点有初始权值 $d_0(x) \in \mathcal{D}$ 。
第 t 个操作定义为：
1. 范围修改：给出范围 $q_t \in Q$ 和 $f_t \in \mathcal{M}$ ，如果 $x \in q_t$ 则
 $d_t(x) = d_{t-1}(x) * f_t$ ，否则 $d_t(x) = d_{t-1}(x)$ 。
2. 范围查询：给出范围 $q_t \in Q$ ，设 $d_t(x) = d_{t-1}(x)$ ，答案为
 $\sum_{x \in q_t} d_t(x)$ 。
用 $n = |I|$ 表示初始权重的个数， m 表示修改查询的操作数。

沈吉源 杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用

引入 抽象信息 离线范围修改查询 例题 半平面范围查询问题

离线范围修改查询算法

下面将介绍一种离线范围修改查询算法，由清华大学李欣隆与蔡承泽提出¹。这个算法可以被简单实现，并且常数较小。可以证明这个算法的操作次数复杂度是最优的。

¹Xinlong Li and Chengze Cai. Offline optimal range query and update algorithm. 2021.

沈吉源 杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用

引入 抽象信息 离线范围修改查询 例题 半平面范围查询问题

等价关系

定义等价关系：

假设有一个操作序列 q_1, q_2, \dots, q_n ，定义点 x_1 与点 x_2 的等价关系：若 $R(x_1, x_2) = \forall_{i \in [1, n], x_1 \in q_i \leftrightarrow x_2 \in q_i}$ ，其中 q_i 为第 i 次操作的范围，则称 x_1 与 x_2 等价。

沈吉源 杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用

引入
○○○

抽象信息
○○

离线范围修改查询
○○○○○●○○○○○○○○

例题
○○○

半平面范围查询问题
○○○○○○

等价类



我们将等价的点归于一个等价类中，可以得到若干个等价类。
假设进行了 q_l, q_{l+1}, \dots, q_r 这些操作，设划分为的等价类集合为 $R_{l,r}$ 。
如果 $[l', r'] \in [l, r]$ ，则 $R_{l,r}$ 比 $R_{l',r'}$ 将点集划为更细的部分，称这个关系为 $R_{l,r} \subseteq R_{l',r'}$ 。

沈吉源

杭州学军中学教育集团文渊中学

范围修改查询问题相关算法及其应用

引入
○○○

抽象信息
○○

离线范围修改查询
○○○○○○●○○○○○○○○

例题
○○○

半平面范围查询问题
○○○○○○

算法介绍

由于修改操作是给定的，我们考虑设计一个在修改操作序列上分治的算法。

在该问题中，考虑维护如下的一个动态有根森林：

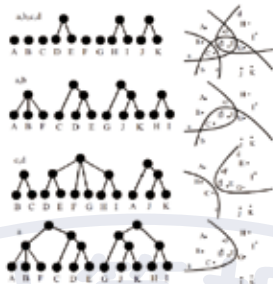
- 每个节点都对应一个操作序列的某个等价类；
- 每个叶子节点存储的信息为大小为 1 的等价类的信息，即 n 个初始给定点各自的信息。

沈吉源

杭州学军中学教育集团文渊中学

范围修改查询问题相关算法及其应用

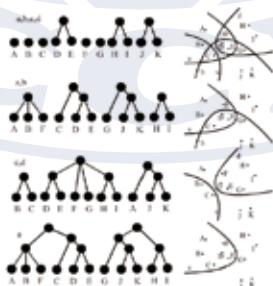
引入 ○○○	抽象信息 ○○	离线范围修改查询 ○○○○○○○○●○○○○○○	例题 ○○○	半平面范围查询问题 ○○○○○○○
-----------	------------	-----------------------------	-----------	----------------------



假设我们想将当前等价类集合从 R_1 变为 R_2 ，并满足 $R_1 \subseteq R_2$ (R_2 的等价类个数更少)。
若我们对当前有根森林做若干次合并操作，每次合并若干有根树，就可以将状态 R_1 移动到目标状态 R_2 。合并若干棵以 x_1, x_2, \dots, x_k 为根的有根树的方法是新建一个点 y 作为 x_1, x_2, \dots, x_k 的父亲节点。

沈吉源	杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用	

引入 ○○○	抽象信息 ○○	离线范围修改查询 ○○○○○○○○●○○○○○○	例题 ○○○	半平面范围查询问题 ○○○○○○○
-----------	------------	-----------------------------	-----------	----------------------



同样，我们可以通过撤销状态 R_2 的若干次合并操作（即删除根节点），将其改变为 R_1 的状态。称这两种操作为有根森林之间的移动。

沈吉源	杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用	

引入 ○○○	抽象信息 ○○	离线范围修改查询 ○○○○○○○○○○●○○○	例题 ○○○	半平面范围查询问题 ○○○○○○○
-----------	------------	----------------------------	-----------	----------------------

为了支持修改查询操作，考虑在有根森林的非叶子节点上维护额外的域。需要记录两个额外的值，一个存储修改的懒惰标记 $u(x)$ ，一个存储子树的和 $d(x)$ （即为等价类中元素的和）。在有根森林加点时，我们会新建一个点 y 作为 x_1, x_2, \dots, x_k 的父亲，此时设置 $u(y) = \epsilon_M, d(y) = \sum_{i=1}^k d(x_i)$ ，即可以上传信息。在有根森林删点时，需要删除点 y 并分裂出 x_1, x_2, \dots, x_k 这 k 棵有根树，此时需要下放点 y 的懒惰标记并删除节点 y 。如果有一棵以 x 为根的树，其包含的叶子节点恰好是一次修改或查询中包含的点，则 $d(x)$ 即为本次询问的答案，修改则可以对 x 打上标记 $(d(x), u(x)) \rightarrow (d(x) * U, u(x) * U)$ 。

沈吉源	杭州学军中学教育集团文渊中学
-----	----------------

范围修改查询问题相关算法及其应用

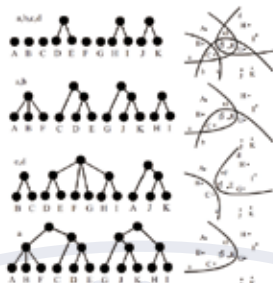
引入 ○○○	抽象信息 ○○	离线范围修改查询 ○○○○○○○○○○●○○○	例题 ○○○	半平面范围查询问题 ○○○○○○○
-----------	------------	----------------------------	-----------	----------------------

对于原问题，考虑分治。将当前操作序列分成左右两部分。对于两部分操作，由于操作数量更少，拥有的等价类个数也更少，这意味着分治后的两个区间都只需要处理更少的信息。当我们需要进行 q_l, \dots, q_r 的分治时，设 $mid = \lfloor \frac{l+r}{2} \rfloor$ ，先将等价类集合从 $R_{l,r}$ 移动到 q_l, \dots, q_{mid} 的状态 $R_{l,mid}$ ，这会在有根森林上加若干个点。处理完 q_l, \dots, q_{mid} 后，将状态回退到 $R_{l,r}$ ，然后移动到 $R_{mid+1,r}$ ，对 q_{mid+1}, \dots, q_r 执行分治算法，再将状态回退到 $R_{l,r}$ 。当递归到 $l = r$ 时只有一个操作，也就只有 2 个等价类。修改就在需要操作的等价类对应的有根树的根上打上标记，查询则提取该根节点 d 的信息即可。

沈吉源	杭州学军中学教育集团文渊中学
-----	----------------

范围修改查询问题相关算法及其应用

引入 ○○○	抽象信息 ○○	离线范围修改查询 ○○○○○○○○○○●○○	例题 ○○○	半平面范围查询问题 ○○○○○○○
-----------	------------	---------------------------	-----------	----------------------



图中演示了等价关系为 $R_{1,4}, R_{1,2}, R_{3,4}, R_{1,1}$ 时有根森林的结构。

我们从 $R_{1,4}$ 移动到 $R_{1,2}$ 时，将等价类 $\{A\}, \{B\}, \{F\}$ 合并为 $\{A, B, F\}$ ； $\{C\}, \{D, E\}$ 合并为 $\{C, D, E\}$ ； $\{G\}, \{J, K\}$ 合并为 $\{G, J, K\}$ 。从 $R_{1,2}$ 移动回 $R_{1,4}$ 时，撤销上述合并。递归到 $R_{1,1}$ 时，需要修改的树即为 a 树，代表了等价类，设 x 为 $\{A, B, C, D, E, F\}$ 即 a 树的根，我们打上修改标记并查询 x 的值。

沈吉源	杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用	

引入 ○○○	抽象信息 ○○	离线范围修改查询 ○○○○○○○○○○●○○	例题 ○○○	半平面范围查询问题 ○○○○○○○
算法的复杂度				

定义离散函数 F ， $F(x)$ 表示 x 个操作最多将点集分成多少个等价类。

不难发现 $F(x)$ 是单调递增的，且 x 个操作最多将集合 I 分成 $\min(F(x), |I|)$ 个等价类。

定义离散函数 G ， $G(n)$ 满足 $F(G(n)) \leq n, F(G(n) + 1) > n$ 。
 $G(n)$ 表示在给定范围下，至少需要多少个操作能把 n 个点分成最细的等价关系，即 n 个等价类。

沈吉源	杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用	

引入 ○○○	抽象信息 ○○	离线范围修改查询 ○○○○○○○○○○○○○○●	例题 ○○○	半平面范围查询问题 ○○○○○○
-----------	------------	-----------------------------	-----------	---------------------

假设有 n 个点与 m 次操作，我们将 m 个操作分成每 $G(n)$ 个操作一组，在每组中使用上述分治算法，在代数结构上的运算次数为

$$T(n, m) = O(n) + \frac{m}{G(n)} T_1(G(n)), T_1(n) = 2T_1(n/2) + O(F(n)).$$

在序列上的问题中，有 $F(n) = O(n)$ ，此时

$$T(n, m) = O(n + m \log \min(n, m)).$$

对于修改查询范围为半平面，多边形，圆，椭圆，双曲线等情况，有 $F(n) = O(n^2)$ ，此时 $T(n, m) = O(n + m\sqrt{n})$ 。

对于修改查询范围为 d 维的空间范围时，其中 $d > 1$ ，有 $F(n) = O(n^d)$ ，此时 $T(n, m) = O(n + mn^{1-1/d})$ 。

特别的，若修改查询范围为点集的任意子集，有 $F(n) = O(2^n)$ ，此时 $G(n) = O(\log n)$ ，可得 $T(n, m) = O(n + m \frac{n}{\log n})$ 。这意味着修改查询为任意子集可以做到比暴力更优的复杂度。

沈吉源	杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用	

引入 ○○○	抽象信息 ○○	离线范围修改查询 ○○○○○○○○○○○○○○●	例题 ●○○	半平面范围查询问题 ○○○○○○
-----------	------------	-----------------------------	-----------	---------------------

[CTS2021] 测测你的半平面修改查询²

给定交换半群 $(\mathcal{D}, +)$ ，半群 $(\mathcal{M}, *)$ ，二元运算符 $*$ ：

$\mathcal{D} * \mathcal{M} \rightarrow \mathcal{D}$ ，其中 $*$ 满足结合律与分配律。

给定平面上的 n 个点 (x_i, y_i) ，每个点有初始权值 $d_i \in \mathcal{D}$ 。

需要实现 m 个操作：

第 j 次操作给出 a_j, b_j, c_j 与半群信息 $o_j \in \mathcal{M}$ 。

对于所有满足 $a_j x_i + b_j y_i < c_j$ 的 i ，需要先回答 $\sum_i d_i$ ，后将所有 d_i 修改为 $o_j * d_i$ 。

操作离线。

²<https://qoj.ac/problem/9020>, <https://qoj.ac/problem/4817>

沈吉源	杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用	

引入 ○○○	抽象信息 ○○	离线范围修改查询 ○○○○○○○○○○○○○○○○	例题 ●●●	半平面范围查询问题 ○○○○○○
-----------	------------	------------------------------	-----------	---------------------

对操作序列每 B 个分一组进行分治，并应用上面的算法，设 $B = \sqrt{n}$ ，可以得到操作次数复杂度为 $O(n + m\sqrt{n})$ 。

接下来我们要处理的问题就是计算几何问题，可以使用平面图点定位的算法。

具体的，对于 \sqrt{m} 个询问的直线，处理出每两条直线的交点，并将交点按照 x 坐标排序后扫描线。

扫描线过程中，实时维护出所有直线在当前 x 坐标下，按照 y 坐标排序后的顺序，在遇到一个交点后就交换两条直线的顺序。同时对所有点按 x 坐标排序，在扫描到该点时二分定位出该点属于哪个区域。这样就得到了 $O(m)$ 个区域的初始信息。

沈吉源	杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用	

引入 ○○○	抽象信息 ○○	离线范围修改查询 ○○○○○○○○○○○○○○○○	例题 ●●●	半平面范围查询问题 ○○○○○○
-----------	------------	------------------------------	-----------	---------------------

对于每条直线，将直线上的交点按 x 坐标增大排序，维护一个链表，每个交点属于两条直线的链表。

当删除该直线时，遍历链表上的每个交点，在该交点所属的另一个链表中删除该点，并合并两边的区域。

这样就完成了删直线并合并区域的过程。撤回删掉的直线时，也同样遍历该直线的链表，依次撤回操作即可。

这样做的时间复杂度为 $O(n + m\sqrt{n} \log n)$ ，瓶颈在对交点排序与扫描线二分。不过该题是交互题，只需要保证操作次数正确即可。

沈吉源	杭州学军中学教育集团文渊中学
范围修改查询问题相关算法及其应用	

引入

抽象信息

离线范围修改查询

例题

半平面范围查询问题

问题的形式

在该问题中，没有修改操作，只有查询操作，查询为半平面的一侧的交换半群信息。

若可以离线，则使用上述的修改查询算法，就可以达到运算次数的下界 $O(n + m\sqrt{n})$ 。

但在仅有查询，没有修改的情况下，有另一些复杂度可能更低、或可以强制在线的算法。下面将介绍一些针对该问题的算法。

沈吉源

杭州学军中学教育集团文渊中学

范围修改查询问题相关算法及其应用

引入

抽象信息

离线范围修改查询

例题

半平面范围查询问题

旋转扫描线

将点集随意划分为大小为 B 的块。

定义半平面的标准型为将询问的半平面先一直向下平移，直到碰到第一个点，之后将询问的半平面一直顺时针方向旋转，直到碰到第一个点。可以证明这样的变换是不会改变这个半平面一侧的点集的。

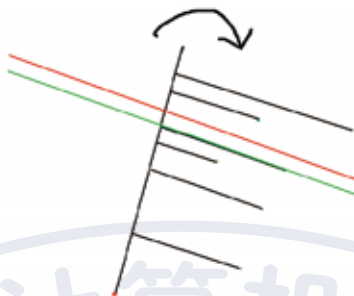
在这样的变换下，在 B 个点的情况下，只存在 $O(B^2)$ 个不同的半平面的标准型。如果两个半平面变换到了相同的标准型，则这两个平面对应同一个点集，询问的答案相同，即只有 $O(B^2)$ 个需要预处理的答案。

沈吉源

杭州学军中学教育集团文渊中学

范围修改查询问题相关算法及其应用

引入 ○○○	抽象信息 ○○	离线范围修改查询 ○○○○○○○○○○○○○○○○	例题 ○○○	半平面范围查询问题 ○○●○○○
-----------	------------	------------------------------	-----------	---------------------



旋转扫描线的过程为：假设当前有一条直线，斜率从 $-\infty$ 逐渐移向 $+\infty$ ，在过程中维护当前所有点的顺序。
对于一次半平面范围询问，可以在扫描线扫到对应斜率时，在点的序列上二分求出前驱后继。然后相当于要在序列上支持前缀，后缀查询。动态维护当前旋转扫描线的序列，每次交换的元素都是相邻的，可以 $O(1)$ 进行修改。

沈吉源	杭州学军中学教育集团文渊中学
-----	----------------

引入 ○○○	抽象信息 ○○	离线范围修改查询 ○○○○○○○○○○○○○○○○	例题 ○○○	半平面范围查询问题 ○○●○○○
-----------	------------	------------------------------	-----------	---------------------

取 $B = \sqrt{m}$ ，代数结构上的运算次数为 $O(n\sqrt{m})$ ，总时间复杂度 $O(n\sqrt{m} \log n)$ 。时间复杂度的瓶颈在于对斜率的排序以及点定位的二分。
注意这里复杂度与修改查询问题的区别：范围修改查询问题的复杂度下界为 $O(n + m\sqrt{n})$ ，不会因为操作多而降低复杂度；而仅支持查询的复杂度可以做到 $O(n\sqrt{m})$ ，会因为操作多而降低复杂度。

沈吉源	杭州学军中学教育集团文渊中学
-----	----------------



引入 ○○○	抽象信息 ○○	离线范围修改查询 ○○○○○○○○○○○○○○○○	例题 ○○○	半平面范围查询问题 ○○○○●○
-----------	------------	------------------------------	-----------	---------------------

此外，还有一些其他的处理范围修改查询问题的方法，如最优点集划分树、平面分块等。由于时间所限，在这里不进行展开，可以阅读我的集训队论文。



拉格朗日乘数及对偶的应用

宁波市镇海中学 施开成

拉格朗日乘数 ●○○○ ○○○○○○○	对偶 ○○○ ○○○○○○○ ○○○○○○○○○	纳什均衡 ○○○○○○○ ○○○○○○○	Thanks ○
松弛变量			

- 1 拉格朗日乘数
 - 松弛变量
 - 拉格朗日乘数定理
 - 拉格朗日乘数法

2 对偶

3 纳什均衡

4 Thanks

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	2 / 45

拉格朗日乘数 ●●○○ ○○○○○ ○○○○○○○	对偶 ○○○ ○○○○○○○ ○○○○○○○○○	纳什均衡 ○○○○○○○ ○○○○○○○	Thanks ○
松弛变量			

考虑一类具有等式约束的最优化问题：

$$\begin{aligned} & \max f(\mathbf{x}) \\ & s.t. \begin{cases} g_1(\mathbf{x}) = 0 \\ g_2(\mathbf{x}) = 0 \\ \dots \\ g_m(\mathbf{x}) = 0 \end{cases} \end{aligned}$$

其中 $\mathbf{x} = (x_1, x_2, \dots, x_n)$ 为 n 元实向量, f, g_1, \dots, g_m 均为 $\mathbb{R}^n \rightarrow \mathbb{R}$ 的连续可微函数。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	3 / 45

拉格朗日乘数 ○○● ○○○○○○○	对偶 ○○○ ○○○○○○○ ○○○○○○○○○	纳什均衡 ○○○○○○○ ○○○○○○○	Thanks ○
松弛变量			

对于这类最优化问题，我们可以通过在等式约束前加入松弛变量，将原问题变为如下形式：

$$\max_{\mathbf{x}} \min_{\lambda_i} f(\mathbf{x}) + \lambda_1 g_1(\mathbf{x}) + \cdots + \lambda_m g_m(\mathbf{x})$$

其中系数 $\lambda_1, \dots, \lambda_m \in \mathbb{R}$ 被称作拉格朗日乘数，函数 $F(\mathbf{x}, \lambda_1, \dots, \lambda_m) = f(\mathbf{x}) + \lambda_1 g_1(\mathbf{x}) + \cdots + \lambda_m g_m(\mathbf{x})$ 被称为拉格朗日函数。

此方法可以将等式约束并入要最大化的式子里。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	4 / 45

拉格朗日乘数 ○○○ ●○○○ ○○○○○○○	对偶 ○○○ ○○○○○○○ ○○○○○○○○○	纳什均衡 ○○○○○○○ ○○○○○○○	Thanks ○
拉格朗日乘数定理			

- 1 拉格朗日乘数
 - 松弛变量
 - 拉格朗日乘数定理
 - 拉格朗日乘数法

2 对偶

3 纳什均衡

4 Thanks

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	5 / 45

拉格朗日乘数 ○○○ ●○○○ ○○○○○○○	对偶 ○○○ ○○○○○○○ ○○○○○○○○○	纳什均衡 ○○○○○○○ ○○○○○○○	Thanks ○
拉格朗日乘数定理			

Theorem 2.1 (拉格朗日乘数定理)

假设 $f, g_1, \dots, g_m: \mathbb{R}^n \rightarrow \mathbb{R}$ 均为连续可微函数。则对于 f 在 $\forall i \in [1, m] \cap \mathbb{N}, g_i(\mathbf{x}_\star) = 0$ 时的任意极值点 \mathbf{x}_\star ，都有 $\nabla f(\mathbf{x}_\star), \nabla g_1(\mathbf{x}_\star), \dots, \nabla g_m(\mathbf{x}_\star)$ 线性相关。

由于时间有限，且该定理的证明需要用到隐函数定理等微积分内容，故此处不作展示。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	6 / 45

拉格朗日乘数 ○○○ ○○●○○ ○○○○○○○	对偶 ○○○ ○○○○○○○ ○○○○○○○○○	纳什均衡 ○○○○○○○ ○○○○○○○	Thanks ○
拉格朗日乘数定理			

Corollary 2.1

对于 f 在 $g_i(x_1, \dots, x_n) = 0$ ($1 \leq i \leq m$) 时的任意一组极值点 \mathbf{x}_\star ，当 $\nabla g_1(\mathbf{x}_\star), \dots, \nabla g_m(\mathbf{x}_\star)$ 线性无关时，存在唯一一组系数 $\lambda_1, \dots, \lambda_m$ 使得 $\nabla f(\mathbf{x}_\star) = \lambda_1 \nabla g_1(\mathbf{x}_\star) + \dots + \lambda_m \nabla g_m(\mathbf{x}_\star)$ 。

这一推论是拉格朗日乘数定理的等价形式，其结论常常被写为拉格朗日函数 $F(x_1, \dots, x_n, \lambda_1, \dots, \lambda_m) = f(x_1, \dots, x_n) + \lambda_1 g_1(x_1, \dots, x_n) + \dots + \lambda_m g_m(x_1, \dots, x_n)$ 关于所有 x_i 和 λ_i 的偏导数均为 0。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	7 / 45



拉格朗日乘数 ○○○ ○○○● ○○○○○○○	对偶 ○○○ ○○○○○ ○○○○○○○	纳什均衡 ○○○○○ ○○○○○ ○○○○○	Thanks ○
拉格朗日乘数定理			

注意条件中的 $\nabla g_1(x_*), \dots, \nabla g_m(x_*)$ 线性无关不可忽略。

如果忽略条件，考虑函数 $f(x, y) = x + y, g(x, y) = x^2 + y^2$ ，求 $f(x, y)$ 在 $g(x, y) = 0$ 时的最大值。

此时拉格朗日函数 $F(x, y, \lambda) = x + y + \lambda(x^2 + y^2)$ 不存在任何临界点，然而 $g(x, y) = 0$ 时 $f(x, y)$ 的最大值应为 $f(0, 0) = 0$ 。这是因为在 $x = 0, y = 0$ 时有 $\nabla g(x, y) = 0$ ，不满足 ∇g 线性无关的条件。

此条件在相关题目中一般都是满足的，放在此处仅为了保证定理的严谨性。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	8 / 45

拉格朗日乘数 ○○○ ○○○ ●○○○○○	对偶 ○○○ ○○○○○ ○○○○○○○	纳什均衡 ○○○○○ ○○○○○ ○○○○○	Thanks ○
拉格朗日乘数法			

- 1 拉格朗日乘数
 - 松弛变量
 - 拉格朗日乘数定理
 - 拉格朗日乘数法

2 对偶

3 纳什均衡

4 Thanks

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	9 / 45

拉格朗日乘数 ○○○ ○○○○ ●○○○○○	对偶 ○○○ ○○○○○ ○○○○○○○	纳什均衡 ○○○○○ ○○○○○ ○○○○○	Thanks ○
拉格朗日乘数法			

对于问题

$$\begin{aligned} \max f(x) \\ \text{s.t. } g_1(x) = 0, g_2(x) = 0, \dots, g_m(x) = 0 \quad (m < n) \end{aligned}$$

在答案存在的情况下，我们可以使用拉格朗日乘数法求解这类问题，具体过程见下一页。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	10 / 45

拉格朗日乘数 ○○○ ○○○○ ○○●○○○○	对偶 ○○○ ○○○○○ ○○○○○○○	纳什均衡 ○○○○○ ○○○○○ ○○○○○	Thanks ○
拉格朗日乘数法			

定义拉格朗日函数

$$F(x, \lambda_1, \dots, \lambda_m) = f(x) + \lambda_1 g_1(x) + \dots + \lambda_m g_m(x).$$

列出 $n + m$ 个方程 $\frac{\partial F}{\partial x_i}(x) = 0$ 和 $\frac{\partial F}{\partial \lambda_i}(x) = 0$ ，其中 $\frac{\partial F}{\partial \lambda_i}(x) = 0$ 等价于 $g_i(x) = 0$ 。取方程的所有解的 x 和使得 $\nabla g_1(x), \dots, \nabla g_m(x)$ 线性相关且满足 $g_i(x) = 0$ 的所有 x ，分别代入 $f(x)$ 计算并取最大值，该值即为 $f(x)$ 在 $g_i(x) = 0$ 约束下的最大值。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	11 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○●○○○	对偶 ○○○ ○○○○○○ ○○○○○○○○	纳什均衡 ○○○○○○ ○○○○○○	Thanks ○
拉格朗日乘数法			
[NOI2012] 骑行川藏			

给定实数 s_i, k_i, v'_i, E , 最小化 $f(v_1, \dots, v_n) = \sum_{i=1}^n \frac{s_i}{v_i}$ 。其中

v_1, \dots, v_n 均为正实数而且需要满足 $\sum_{i=1}^n k_i(v_i - v'_i)^2 s_i = E$ 。

$1 \leq n \leq 10000, 0 \leq E \leq 10^8, s_i \in [0, 10^5], k_i \in (0, 15], v'_i \in (-100, 100)$ 。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	12 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○●○○○	对偶 ○○○ ○○○○○○ ○○○○○○○○	纳什均衡 ○○○○○○ ○○○○○○	Thanks ○
拉格朗日乘数法			

稍作分析可以发现, 当 $f(v_1, \dots, v_n)$ 在题目的条件下取到最小值时, 必然有 $v_i \geq v'_i$ 。

因此我们定义 $g(v_1, \dots, v_n) = \sum_{i=1}^n k_i(v_i - v'_i)^2 s_i - E$, ∇g 线性相关

当且仅当所有 $v_i = v'_i$ 成立, 此时特判处理即可。

则可以使用拉格朗日乘数法求解

$F(v_1, \dots, v_n, \lambda) = f(v_1, \dots, v_n) - \lambda g(v_1, \dots, v_n)$ 的临界点。解偏导方程会求出全部极值点, 但由于题目要求 v_i 为正实数, 所以我们只需要考虑满足 $0 < v_i$ 的极值点。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	13 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○●○	对偶 ○○○ ○○○○○○ ○○○○○○○○	纳什均衡 ○○○○○○ ○○○○○○	Thanks ○
拉格朗日乘数法			

$$\text{由 } \frac{\partial F}{\partial v_i} = -\frac{s_i}{v_i^2} + 2\lambda s_i k_i(v_i - v'_i), \frac{\partial F}{\partial \lambda} = \sum_{i=1}^n k_i(v_i - v'_i)^2 s_i - E \text{ 可得}$$

$$2\lambda s_i k_i(v_i - v'_i) = \frac{s_i}{v_i^2}$$

$$\sum_{i=1}^n k_i(v_i - v'_i)^2 s_i = E$$

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	14 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○●○	对偶 ○○○ ○○○○○○ ○○○○○○○○	纳什均衡 ○○○○○○ ○○○○○○	Thanks ○
拉格朗日乘数法			

$$2\lambda s_i k_i(v_i - v'_i) = \frac{s_i}{v_i^2} \quad \sum_{i=1}^n k_i(v_i - v'_i)^2 s_i = E$$

当 λ 固定时，考虑左式等式两端的函数图像。当 $\lambda > 0$ 时，等式两端的函数在 $v_i > 0$ 时有且仅有一个交点，且必然有 $v_i \geq v'_i$ 成立；当 $\lambda \leq 0$ 时，等式两端的函数在 $v_i \geq v'_i$ 时不存在交点。

故必然有 $\lambda > 0$, $v_i \geq v'_i$ ，并且由函数图像可知 λ 递增时 v_i 会递减，此时右式的等式左侧也会递减。

因此可以通过二分 λ 的方式求解以上方程组。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	15 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ●○○ ○○○○○○○ ○○○○○○○○○	纳什均衡 ○○○○○○○ ○○○○○○○	Thanks ○
拉格朗日对偶			

1 拉格朗日乘数

2 对偶

■ 拉格朗日对偶

■ 性质

■ 线性规划对偶

3 纳什均衡

■ Thanks

施开成

宁波市镇海中学

拉格朗日乘数及对偶的应用

16 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ●○○ ○○○○○○○ ○○○○○○○○○	纳什均衡 ○○○○○○○ ○○○○○○○	Thanks ○
拉格朗日对偶			

对于 $f: \mathbb{R}^n \rightarrow \mathbb{R}, g: \mathbb{R}^n \rightarrow \mathbb{R}^m, h: \mathbb{R}^n \rightarrow \mathbb{R}^p$ ，考虑最优化问题

$$\begin{aligned} \max f(\mathbf{x}) \\ s.t. \quad g(\mathbf{x}) \geq \mathbf{0} \\ h(\mathbf{x}) = \mathbf{0} \end{aligned}$$

对于不等式约束，也可以加入松弛变量，将问题变为

$$\max_{\mathbf{x}} \min_{\lambda \geq \mathbf{0}, \nu} f(\mathbf{x}) + \lambda^T g(\mathbf{x}) + \nu^T h(\mathbf{x})$$

记拉格朗日函数 $F(\mathbf{x}, \lambda, \nu) = f(\mathbf{x}) + \lambda^T g(\mathbf{x}) + \nu^T h(\mathbf{x})$ 。

施开成

宁波市镇海中学

拉格朗日乘数及对偶的应用

17 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○● ○○○○○○○ ○○○○○○○○○	纳什均衡 ○○○○○○○ ○○○○○○○	Thanks ○
拉格朗日对偶			

$$\max_x \min_{\lambda \geq 0, \nu} F(x, \lambda, \nu)$$

对于上式，其拉格朗日对偶问题即为交换 max 和 min 的顺序后的问题，即

$$\min_{\lambda \geq 0, \nu} \max_x F(x, \lambda, \nu)$$

原函数 $f(x)$ 的拉格朗日对偶函数即为内层的式子

$$L(\lambda, \nu) = \max_x F(x, \lambda, \nu)$$

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	18 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ●○○○○○ ○○○○○○○	纳什均衡 ○○○○○○○ ○○○○○○○	Thanks ○
性质			

1 拉格朗日乘数

2 对偶

- 拉格朗日对偶
- 性质
- 线性规划对偶

3 纳什均衡

4 Thanks

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	19 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○●○○○ ○○○○○○○	纳什均衡 ○○○○○ ○○○○○ ○○○○○	Thanks ○
性质			
弱对偶性			

Theorem 2.1 (min-max 不等式)

对于 $F: D_x \times D_y \rightarrow \mathbb{R}$, 有 $\max_{y \in D_y} \min_{x \in D_x} F(x, y) \leq \min_{x \in D_x} \max_{y \in D_y} F(x, y)$ 。

Proof

$$\begin{aligned} \forall i, j: F(i, j) &\leq F(i, j) \\ \forall i, j: \min_x F(x, j) &\leq F(i, j) \\ \forall i: \max_y \min_x F(x, y) &\leq \max_y F(i, y) \\ \max_y \min_x F(x, y) &\leq \min_x \max_y F(x, y) \end{aligned}$$

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	20 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○●○○○ ○○○○○○○	纳什均衡 ○○○○○ ○○○○○ ○○○○○	Thanks ○
性质			
强对偶性			

在一些特殊情况下，交换 min 和 max 的顺序不会改变式子的值，这一性质被称为强对偶性。

Definition 1

\mathbb{R}^n 的子集是紧致的当且仅当它是闭集合且有界。
 $S \subseteq \mathbb{R}^n$ 是凸集当且仅当 $\forall p, q \in S, \forall \lambda \in [0, 1], \lambda p + (1 - \lambda)q \in S$ 。
 对于凸集 C , $f: C \rightarrow \mathbb{R}$ 是凸函数当且仅当 $\forall p, q \in C, \forall \lambda \in [0, 1], f(\lambda p + (1 - \lambda)q) \leq \lambda f(p) + (1 - \lambda)f(q)$, f 是凹函数当且仅当 $\forall p, q \in C, \forall \lambda \in [0, 1], f(\lambda p + (1 - \lambda)q) \geq \lambda f(p) + (1 - \lambda)f(q)$ 。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	21 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○●○○ ○○○○○○○	纳什均衡 ○○○○○ ○○○○○ ○○○○○	Thanks ○
性质			

Lemma 2.1

若 $D_x \subseteq \mathbb{R}^n, D_y \subseteq \mathbb{R}^m$ 为紧致凸集，连续函数 $f: D_x \times D_y \rightarrow \mathbb{R}$ 关于 x 是凸函数，关于 y 是凹函数，则存在 $x_0 \in D_x, y_0 \in D_y$ ，满足：

$$\min_{x \in D_x} F(x, y_0) = F(x_0, y_0) = \max_{y \in D_y} F(x_0, y)$$

由于该引理的证明过于困难，故此处不作展示。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	22 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○●○○ ○○○○○○○	纳什均衡 ○○○○○ ○○○○○ ○○○○○	Thanks ○
性质			

Theorem 2.2 (minimax 定理)

若 $D_x \subseteq \mathbb{R}^n, D_y \subseteq \mathbb{R}^m$ 为紧致凸集，连续函数 $f: D_x \times D_y \rightarrow \mathbb{R}$ 关于 x 是凸函数，关于 y 是凹函数，则有：

$$\max_{y \in D_y} \min_{x \in D_x} F(x, y) = \min_{x \in D_x} \max_{y \in D_y} F(x, y)$$

在实际应用中 F 常常是个双线性函数。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	23 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○● ○○○○○○○○○	纳什均衡 ○○○○○○ ○○○○○○	Thanks ○
性质			

Proof

由引理 2.1 可知, 存在 $x_0 \in D_x, y_0 \in D_y$ 满足

$$\max_{y \in D_y} F(x_0, y) = F(x_0, y_0) = \min_{x \in D_x} F(x, y_0)$$

$$\begin{aligned} & \text{则有 } \min_{x \in D_x} \max_{y \in D_y} F(x, y) \leq \max_{y \in D_y} F(x_0, y) = F(x_0, y_0) = \min_{x \in D_x} F(x, y_0) \\ & \leq \max_{y \in D_y} \min_{x \in D_x} F(x, y). \end{aligned}$$

由弱对偶定理可知 $\max_{y \in D_y} \min_{x \in D_x} F(x, y) \leq \min_{x \in D_x} \max_{y \in D_y} F(x, y)$ 。

因此有 $\max_{y \in D_y} \min_{x \in D_x} F(x, y) = \min_{x \in D_x} \max_{y \in D_y} F(x, y)$ 。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	24 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○● ●○○○○○○○	纳什均衡 ○○○○○○ ○○○○○○	Thanks ○
线性规划对偶			

1 拉格朗日乘数

2 对偶

- 拉格朗日对偶
- 性质
- 线性规划对偶

3 纳什均衡

4 Thanks

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	25 / 45

拉格朗日乘数 ○○○ ○○○○○	对偶 ○○○ ○○○○○ ●○○○○○	纳什均衡 ○○○○○ ○○○○○	Thanks ○
线性规划对偶			

对于线性规划标准型

$$\begin{aligned} \max_x \quad & c^T x \\ \text{s.t.} \quad & x \geq 0 \\ & Ax \leq b \end{aligned}$$

我们可以通过添加拉格朗日乘数的方式来进行改写：

$$\max_{x \geq 0} \min_{y \geq 0} c^T x + y^T (b - Ax)$$

并得到其对偶形式

$$\min_{y \geq 0} \max_{x \geq 0} c^T x + y^T (b - Ax)$$

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	26 / 45

拉格朗日乘数 ○○○ ○○○○○	对偶 ○○○ ○○○○○ ○○●○○○○○	纳什均衡 ○○○○○ ○○○○○	Thanks ○
线性规划对偶			

$$\min_{y \geq 0} \max_{x \geq 0} c^T x + y^T (b - Ax)$$

将式子关于 x 进行整理得：

$$\min_{y \geq 0} \max_{x \geq 0} (c^T - y^T A)x + y^T b$$

将 x 视为拉格朗日乘数并从式子中取出，即得到线性规划问题的对偶形式：

$$\begin{aligned} \min_{y \geq 0} \quad & y^T b \\ \text{s.t.} \quad & y^T A \geq c^T \end{aligned}$$

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	27 / 45



拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○○ ○○○●○○○	纳什均衡 ○○○○○○ ○○○○○○ ○○○○○○	Thanks ○
线性规划对偶			

对于一个非标准型线性规划问题，如果我们将先将其转为标准型再进行常规的线性规划对偶，不仅步骤繁杂，而且可能会在矩阵中增加大量冗余元素，阻碍对问题形式的观察和判断。

而这种利用拉格朗日乘数进行对偶的方法，既便于记忆和使用，又能在很大程度上保留原问题的简洁形式，使得对偶问题的组合意义更加清晰。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	28 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○○ ○○○●○○○	纳什均衡 ○○○○○○ ○○○○○○ ○○○○○○	Thanks ○
线性规划对偶			
[AtCoder World Tour Finals 2022 Day1] Welcome to Tokyo!			

给定 m 对 l_i, r_i ，你需要对于每个 $k = 1, 2, \dots, n$ ，求出：如果在 $1, 2, \dots, n$ 中标记 k 个数，最多能有多少个 i 使得 $[l_i, r_i]$ 中存在被标记的数。

$$1 \leq n, m \leq 10^6, 1 \leq l_i, r_i \leq n。$$

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	29 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○○○ ○○○○○○●○○	纳什均衡 ○○○○○○○ ○○○○○○○	Thanks ○
线性规划对偶			

对于一个固定的 k 考虑问题。令 a_i 表示是否标记 i , b_i 表示 $[l_i, r_i]$ 中是否存在被标记的数, 则可以转化为求以下线性规划问题的整数解:

$$\max \sum_{i=1}^m b_i \quad s.t. \quad a_i \leq 1, \sum_{i=1}^n a_i = k, b_i \leq 1, b_i \leq \sum_{j=l_i}^{r_i} a_j, 0 \leq a_i, b_i$$

稍作改写得到

$$\max_{0 \leq a_i, b_i} \sum_{i=1}^m b_i \quad s.t. \quad a_i - 1 \leq 0, \sum_{i=1}^n a_i - k = 0, b_i - 1 \leq 0, b_i - \sum_{j=l_i}^{r_i} a_j \leq 0$$

由于该线性规划问题的矩阵是全幺模矩阵, 因此我们忽略整数性的限制。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	30 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○○○ ○○○○○○●○○	纳什均衡 ○○○○○○○ ○○○○○○○	Thanks ○
线性规划对偶			

对限制加入拉格朗日乘数可得 $\max_{0 \leq a_i, b_i} \min_{p_i, q_i, s_i \leq 0, \lambda}$

$$\sum_{i=1}^m \left(b_i + q_i(b_i - 1) + s_i \left(b_i - \sum_{j=l_i}^{r_i} a_j \right) \right) + \sum_{i=1}^n p_i(a_i - 1) + \lambda \left(\sum_{i=1}^n a_i - k \right).$$

使用 minimax 定理交换 \max 和 \min 并重新整理得 $\min_{p_i, q_i, s_i \leq 0, \lambda} \max_{0 \leq a_i, b_i}$

$$\sum_{i=1}^n \left(p_i + \lambda - \sum_{j \in [l_i, r_i]} s_j \right) a_i + \sum_{i=1}^m (1 + q_i + s_i) b_i - \sum_{i=1}^n p_i - \sum_{i=1}^m q_i - \lambda k.$$

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	31 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○ ○○○○○○○●	纳什均衡 ○○○○○ ○○○○○ ○○○○○	Thanks ○
线性规划对偶			

将 a_i, b_i 视为拉格朗日乘数后得到

$$\begin{aligned} \min_{p_i, q_i, s_i \leq 0, \lambda} & - \sum_{i=1}^n p_i - \sum_{i=1}^m q_i - \lambda k \\ \text{s.t. } & p_i + \lambda - \sum_{j \mid i \in [l_j, r_j]} s_j \leq 0 \\ & 1 + q_i + s_i \leq 0 \end{aligned}$$

此时对偶已经完成，后续可以通过人类智慧去除 p_i, q_i 并思考该问题的组合意义。由于这部分与主题无关，故不作叙述。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	32 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○ ○○○○○○○	纳什均衡 ●○○○○ ○○○○○	Thanks ○
定义			

1 拉格朗日乘数

2 对偶

3 纳什均衡

■ 定义

■ 零和博弈

4 Thanks

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	33 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○○ ○○○○○○○○	纳什均衡 ○○●○○○ ○○○○○○	Thanks ○
定义			

为简化讨论，我们将一个策略空间有限的完全信息双人静态非合作博弈视为以下模型：

两人各自独立地进行一次行动，第一个人有 n 种可能的行动，第二个人有 m 种可能的行动。如果第一个人选了第 i 种行动，第二个人选了第 j 种行动，则第一个人获得 A_{ij} 的收益，第二个人获得 B_{ij} 的收益。（ A, B 是事先给出的 $n \times m$ 的收益矩阵）。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	34 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○○ ○○○○○○○○	纳什均衡 ○○●○○○ ○○○○○○	Thanks ○
定义			

参加博弈的人可以制定混合策略，即在操作前给每一种行动分配一个概率。如果第一个人给第 i 种行动分配了 a_i 的概率，则所有的 a_i 可以看作一个 n 维列向量 a ，其每个元素都介于 $[0, 1]$ 之间且所有元素总和为 1。类似的，第二个人的策略也可以用一个 m 维列向量 b 表示，其每个元素都介于 $[0, 1]$ 之间且所有元素总和为 1。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	35 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○○ ○○○○○○○○	纳什均衡 ○○●○○○ ○○○○○○	Thanks ○
定义			

参加博弈的人可以制定混合策略，即在操作前给每一种行动分配一个概率。如果第一个人给第 i 种行动分配了 a_i 的概率，则所有的 a_i 可以看作一个 n 维列向量 a ，其每个元素都介于 $[0, 1]$ 之间且所有元素总和为 1。类似的，第二个人的策略也可以用一个 m 维列向量 b 表示，其每个元素都介于 $[0, 1]$ 之间且所有元素总和为 1。

假如双方的最优策略分别为 a 和 b ，则第一个人的期望收益为 $a^T A b$ ，第二个人的期望收益为 $a^T B b$ 。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	35 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○○ ○○○○○○○○	纳什均衡 ○○●○○○ ○○○○○○	Thanks ○
定义			

Definition 2 (纳什均衡)

如果博弈的每个参与者都选择了自己的混合策略，且没有玩家能在只调整自己混合策略的情况下增大自己的期望收益，则称当前的混合策略集合与相应的博弈结果构成了纳什均衡（或纳什均衡点）。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	36 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○○ ○○○○○○○○	纳什均衡 ○○○○●○ ○○○○○○	Thanks ○
定义			

Example 1 (囚徒困境)

一个监狱中有两名嫌疑犯，他们每人有两个选择：认罪并检举对方和保持沉默。如果两人都检举对方，则各判刑 8 年；如果一人检举对方，另一人保持沉默，则检举对方者立刻出狱，保持沉默者判刑 10 年；如果两人都保持沉默，则各判刑 2 年。两名囚犯无法交流，并且都希望最小化自身刑期。

如果我们用刑期的相反数表示收益，则根据问题描述，有

$$n = m = 2, A = \begin{bmatrix} -8 & 0 \\ -10 & -2 \end{bmatrix}, B = \begin{bmatrix} -8 & -10 \\ 0 & -2 \end{bmatrix}.$$

此时存在唯一的纳什均衡点，即双方都选择检举对方，混合策略为

$$a = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	37 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○○ ○○○○○○○○	纳什均衡 ○○○○●○ ○○○○○○	Thanks ○
定义			

Example 2 (性别博弈)

有一对夫妻，丈夫希望看球赛，妻子希望逛街，但他们都想待在一起而不是分开各做各的。具体而言，两个人会分别选择看球赛或者逛街，如果双方选择不一致则均获得 0 的满意度，否则与自己希望相符的一人获得 3 的满意度，另一人获得 2 的满意度。

根据问题描述，有 $n = m = 2, A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}, B = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}.$

该博弈有三个纳什均衡点，分别为 $a = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ ，此时两人收

益分别为 3, 2; $a = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}, b = \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix}$ ，此时两人期望收益分别为 1.2, 1.2;

$a = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ ，此时两人收益分别为 2, 3。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	38 / 45



拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○ ○○○○○○○	纳什均衡 ○○○○○ ●○○○○○	Thanks ○
零和博弈			

1 拉格朗日乘数

2 对偶

3 纳什均衡

■ 定义

■ 零和博弈

4 Thanks

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	39 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○ ○○○○○○○	纳什均衡 ○○○○○ ●○○○○○	Thanks ○
零和博弈			

Definition 3 (零和博弈)

如果对于任意博弈结果都有所有人的收益之和为 0，则称这一博弈是零和博弈。

对于之前描述的模型，如果其为零和博弈，则有 $A = -B$ 。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	40 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○ ○○○○○○○	纳什均衡 ○○○○○ ○●○○○	Thanks ○
零和博弈			

Definition 3 (零和博弈)

如果对于任意博弈结果都有所有人的收益之和为 0，则称这一博弈是零和博弈。

对于之前描述的模型，如果其为零和博弈，则有 $A = -B$ 。

因此可以只用一个矩阵 A 来描述收益函数，对于双方的混合策略 a, b ，有第一个人的期望收益等于 $a^T A b$ ，第二个人的期望收益等于 $-a^T A b$ 。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	40 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○ ○○○○○○○	纳什均衡 ○○○○○ ○●○○○	Thanks ○
零和博弈			

Theorem 2.1

对于任意一个双人零和博弈，不可能存在两个纳什均衡点，使得参与者在两个纳什均衡点处的期望收益不同。

Proof

反证法。假设双方在第一个纳什均衡点的混合策略分别为 a_1, b_1 ，在第二个纳什均衡点的混合策略分别为 a_2, b_2 ，且假设第一位玩家在第一个纳什均衡点的期望收益更高，即 $a_1^T A b_1 > a_2^T A b_2$ 。

由于纳什均衡的定义是没有玩家可以通过改变自身策略来增加期望收益，因此有

$$-a_1^T A b_1 \geq -a_1^T A b_2, \quad a_2^T A b_2 \geq a_1^T A b_2$$

即 $a_2^T A b_2 \geq a_1^T A b_2 \geq a_1^T A b_1$ ，与 $a_1^T A b_1 > a_2^T A b_2$ 矛盾。故双人零和博弈不可能存在两个期望收益不同的纳什均衡点。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	41 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○○ ○○○○○○○○	纳什均衡 ○○○○○○ ○○○○●○	Thanks ○
零和博弈			

Theorem 2.2

双人零和博弈必然存在纳什均衡点。

Proof

设该博弈中两人的策略数量分别为 n, m ，第一个人的收益矩阵为 A 。令 D_a 表示第一个人的所有混合策略构成的集合， D_b 表示第二个人的所有混合策略构成的集合，则有 D_a, D_b 均为凸集。定义函数 $F: D_a \times D_b \rightarrow \mathbb{R}$ 满足 $F(a, b) = a^T A b$ 。

由之前的引理可知，必然存在 $a_0 \in D_a, b_0 \in D_b$ ，使得

$$\max_{a \in D_a} F(a, b_0) = F(a_0, b_0) = \min_{b \in D_b} F(a_0, b)$$

则双方的混合策略 a_0, b_0 构成了该博弈的一个纳什均衡。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	42 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○○ ○○○○○○○○	纳什均衡 ○○○○○○ ○○○○●○	Thanks ○
零和博弈			

由上述讨论可知，任意双人零和博弈均存在且仅存在一个期望收益不同的纳什均衡点。因此实际应用中常常把纳什均衡定义为双人零和博弈的解。对于 OI 中的双人零和静态博弈问题，题目中的“双方均采用最优策略”即指代达到纳什均衡的混合策略。

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	43 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○ ○○○○○○○	纳什均衡 ○○○○○ ○○○○○ ○○○○●	Thanks ○
零和博弈			

由上述讨论可知，任意双人零和博弈均存在且仅存在一个期望收益不同的纳什均衡点。因此实际应用中常常把纳什均衡定义为双人零和博弈的解。对于 OI 中的双人零和静态博弈问题，题目中的“双方均采用最优策略”即指代达到纳什均衡的混合策略。

Corollary 2.1

对于一个双人零和博弈，令 D_a 表示第一个人的所有混合策略构成的集合， D_b 表示第二个人的所有混合策略构成的集合， A 表示第一个人的收益矩阵， S 表示纳什均衡情况下第一个人的期望收益。则有

$$S = \min_{b \in D_b} \max_{a \in D_a} a^T A b = \max_{a \in D_a} \min_{b \in D_b} a^T A b.$$

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	43 / 45

拉格朗日乘数 ○○○ ○○○○ ○○○○○○○	对偶 ○○○ ○○○○○ ○○○○○○○	纳什均衡 ○○○○○ ○○○○○ ○○○○●	Thanks ○
零和博弈			

该推论还提供了一种把双人零和博弈的纳什均衡问题转化为线性规划问题的方法。第一个人在纳什均衡点的期望收益可以用以下线性规划问题表示：

$$\begin{aligned} & \max t \\ & s.t. \quad x_i \geq 0, \quad \sum_{i=1}^n x_i = 1, \quad \forall i \in [1, n] \cap \mathbb{Z}, \quad \sum_{j=1}^n x_j A_{ji} \geq t \end{aligned}$$

施开成	宁波市镇海中学
拉格朗日乘数及对偶的应用	44 / 45



不相交格路与反射容斥

北京市十一学校 吴畅

Overview

- ① 定义与约定
- ② 不相交格路
 - 起点和终点固定
 - Pfaffian
 - 终点不固定
 - 起点和终点均不固定
- ③ 反射容斥
 - 两条斜率为 1 的直线边界
 - 更一般的描述
 - 两个例子

吴畅 (北京市十一学校)

不相交格路与反射容斥

January 12, 2024

2 / 26

定义与约定

定义

Definition (步长, 格路)

称 \mathbb{Z}^d 上的一个向量为一种**步长**, 若干步长构成的集合为一个**步长集合**. 考虑 \mathbb{Z}^d 上的序列 $P = (P_0, P_1, \dots, P_l)$, 以及步长集合 \mathbb{S} , 若满足 $\forall 0 \leq i < l, \overrightarrow{P_i P_{i+1}} \in \mathbb{S}$, 则称 P 是步长集合 \mathbb{S} 下的一条长度为 l 的**格路**.

更常见的是下面的概念:

Definition (简单步, 简单格路)

若 \mathbb{Z}^d 上的步长 $s = (s_1, s_2, \dots, s_d)$ 满足 $\forall 1 \leq i \leq d, s_i \in \{0, 1\}$ 且 $\sum_{i=1}^d s_i = 1$, 则称 s 是**简单步**. 步长集合由 \mathbb{Z}^d 上所有简单步构成的格路, 称为**简单格路**.

约定

从 A 到 E , 长度为 m 、步长集合为 \mathbb{S} 、约束集合为 R 的所有格路 (组) 构成的集合:

$$L_m(A \rightarrow E; \mathbb{S} \mid R)$$

例如 $L((0, 0) \rightarrow (1, 1) \mid x \geq y) = \{((0, 0), (1, 0), (1, 1))\}$.

带权计数, w 是定义在 \mathcal{M} 上的函数:

$$GF(\mathcal{M}; w) := \sum_{x \in \mathcal{M}} w(x)$$

通常情况下 \mathcal{M} 是路径或路径组的集合, 而 w 同时定义在边、路径和路径组上: 路径的权值为所有边权的乘积, 路径组的权值为所有路径权值的乘积.

不相交格路

起点和终点固定

当起点集合和终点集合固定时，有广为人知的 LGV 引理：

Lemma (Lindström–Gessel–Viennot lemma)

令 G 是有向无环图， $\mathbf{A} = (A_1, A_2, \dots, A_n)$, $\mathbf{E} = (E_1, E_2, \dots, E_n)$ 均是节点组成的序列， w 是边权函数。 \mathfrak{S}_n 表示长度为 n 的排列构成的集合。

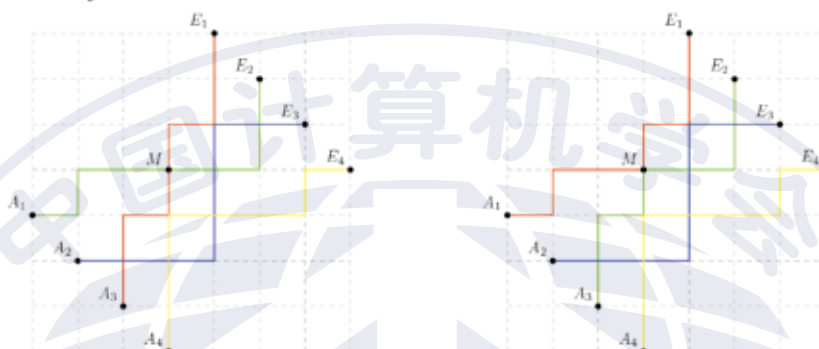
$$\begin{aligned} \sum_{\sigma \in \mathfrak{S}_n} (\text{sgn } \sigma) GF(L_G(\mathbf{A}_\sigma \rightarrow \mathbf{E} \mid \text{non-intersecting}); w) \\ = \det_{1 \leq i, j \leq n} (GF(L_G(A_i \rightarrow E_j); w)) \end{aligned}$$

其中 $L_G(\dots)$ 含义与 $L(\dots)$ 基本相同，唯一的区别是语境不再是格路而是有向无环图 G 上的路径。

LGV 引理的简要证明

构造一个对合 ϕ , 定义在二元组 (σ, \mathbf{P}) 上, 其中 $\mathbf{P} \in L_G(\mathbf{A}_\sigma \rightarrow \mathbf{E})$ 是相交的路径组, 满足符号相反、权值不变。

ϕ 的构造: 取任一 G 的拓扑序 \mathcal{A} 。找到最小的交点 (M, i, j) , 表示 P_i 和 P_j 在 M 处有一交点, 其中第一维的比较按照 \mathcal{A} 上的排名进行。交换 P_i, P_j 在 M 前的子路径。



吴畅 (北京市十一学校)

不相交格路与反射容斥

January 12, 2024

8 / 26

G-协调

LGV 帮我们抵消了相交格路的贡献, 但是仍只能求出所有不相交格路带符号的求和。

Definition

若有向无环图 G 以及其上的顶点集合 \mathbf{A}, \mathbf{E} 满足, $\forall i < j, k < l$, 任意 $P \in L_G(A_i \rightarrow E_l)$ 和 $Q \in L_G(A_j \rightarrow E_k)$ 相交, 则称 \mathbf{A} 和 \mathbf{E} 是 G -协调的。

Theorem

令 G 是一张有向无环图, $\mathbf{A} = (A_1, A_2, \dots, A_n), \mathbf{E} = (E_1, E_2, \dots, E_n)$ 是 G -协调的节点序列, w 是边权函数 (同样定义在路径和路径组上)。

$$GF(L_G(\mathbf{A} \rightarrow \mathbf{E} \mid \text{non-intersecting}); w) = \det_{1 \leq i, j \leq n} (GF(L_G(A_j \rightarrow E_i); w))$$

在简单格路的环境下, 这个条件更容易被满足, 例如 $A_i = (1, a_i), E_i = (n, e_i)$ 。

吴畅 (北京市十一学校)

不相交格路与反射容斥

January 12, 2024

9 / 26

完美匹配

Definition (完美匹配)

称一个长度为 $2n$ 的排列 $\pi \in \mathfrak{S}_{2n}$ 是完美匹配，当且仅当其满足

- $\forall 1 \leq i \leq n, \pi_{2i-1} < \pi_{2i}$
- $\forall 1 \leq i < n, \pi_{2i-1} < \pi_{2i+1}$

对所有 $1 \leq i \leq n$ ，称 π_{2i-1} 和 π_{2i} 是一对匹配的元素。

记 \mathfrak{M}_{2n} 表示全体长度为 $2n$ 的完美匹配构成的集合。

类似排列，对匹配 π 定义 $\text{inv } \pi$ 和 $\text{sgn } \pi$ 。此外我们还有一种递归的方法计算 $\text{inv } \pi$ 的奇偶性：

取 π 中任意一对匹配的元素 (i, j) ，将它们删去，其余元素在下标和值域上填补空缺。可以得到一个 $\pi' \in \mathfrak{M}_{n-2}$ ，那么
 $\text{inv } \pi \equiv \text{inv } \pi' + j - i - 1 \pmod{2}$ 。

吴畅 (北京市十一学校)

不相交格路与反射容斥

January 12, 2024

10 / 26

Pfaffian

Definition (反对称矩阵)

令 $A = (a_{i,j})_{n \times n}$ 是一个方阵，若 $A^T = -A$ ，则称 A 是一个反对称矩阵。

可以发现，反对称矩阵满足主对角线上为 0，关于主对角线对称的元素互为相反数。因此，我们可以只用主对角线上方的元素 $(a_{i,j})_{1 \leq i < j \leq n}$ 来描述一个 $n \times n$ 的反对称矩阵 A 。

Definition (Pfaffian)

令 n 是偶数， $A = (a_{i,j})_{1 \leq i < j \leq n}$ 是一个反对称矩阵，定义 A 的 Pfaffian 为

$$\text{Pf}(A) = \sum_{\pi \in \mathfrak{M}_n} (\text{sgn } \pi) \prod_{i=1}^{n/2} a_{\pi(2i-1), \pi(2i)}$$

吴畅 (北京市十一学校)

不相交格路与反射容斥

January 12, 2024

11 / 26

Pfaffian 的性质

Proposition

假设 n 是偶数, A 是大小为 n 的反对称矩阵, X 是大小为 n 的方阵

- ① $\text{Pf}(1)_{1 \leq i < j \leq n} = 1$, 等价地, $\sum_{\pi \in \mathfrak{M}_n} \text{sgn } \pi = 1$ 。
- ② $\text{Pf}(X^T A X) = \det(X) \text{Pf}(A)$ 。
- ③ $\text{Pf}(A)^2 = \det(A)$ 。

为了求 Pfaffian, 可以考虑令 (b) 中的 X 是初等行变换矩阵, 得到与 \det 类似的消元过程, 复杂度同高消。

终点不固定的不相交路径

Theorem

令 G 是一张有向无环图, $\mathbf{A} = (A_1, A_2, \dots, A_{2n})$, $\mathbf{E} = (E_1, E_2, \dots, E_m)$ 是 G -协调的节点序列, w 是边权函数。

$$\sum_{\mathbf{E}' \subseteq \mathbf{E}, |\mathbf{E}'| = 2n} GF(L_G(\mathbf{A} \rightarrow \mathbf{E}' \mid \text{non-intersecting}); w) \\ = \text{Pf}_{1 \leq i < j \leq 2n} (Q_G(i, j; w))$$

其中 $Q_G(i, j; w)$ 表示对所有不交路径对 (P', P'') , 满足起点分别为 A_i 和 A_j , 终点均为 \mathbf{E} 中的点, $\sum_{(P', P'')} w(P')w(P'')$ 。

我们希望仿照 LGV 引理的证明, 构造对合 ϕ 将二元组 (π, \mathbf{P}) 中 \mathbf{P} 相交的那些消掉。

一个引理

Lemma

令 n, i, j 是三个正整数, 满足 $1 \leq i < j \leq n$. 对完美匹配 $\pi \in \mathfrak{M}_n$, 满足在 π 中 i 和 j 不匹配, 令 π' 表示交换 π 中 i 和 j 的位置, 然后在保持匹配关系不变的情况下, 重排形成的新完美匹配。假如对所有 $i < k < j$, 在 π 中 k 和 i, j 均不匹配, 那么 $\text{sgn } \pi = -\text{sgn } \pi'$ 。

Proof.

根据引理中所给的条件, i, j 及其匹配元素的变换情况, 只可能为下图两种中的一种。用之前提到的方法比较 $\text{inv } \pi$ 和 $\text{inv } \pi'$ 的奇偶性, 也就是分别删去 i, j 及其匹配元素并检查贡献的不同。不难验证, 在两种情况下, 该引理均是正确的。



吴畅 (北京市十一学校)

不相交格路与反射容斥

January 12, 2024

14 / 26

定理的证明

Proof.

ϕ 的构造: 取任一 G 的拓扑序 \mathcal{A} . 对于一个相交路径组 P , 我们将每个交点描述为 (M, i, j) 的形式, 表示 P_i 和 P_j 在 M 处有一交点。找到所有这样的三元组中字典序最小的一个, 其中第一维的比较按照 \mathcal{A} 上的排名进行。交换 P_i, P_j 在 M 前的子路径; 同时交换 i, j 在 π 中的位置, 保持元素的匹配关系不变的情况下重排。

不难发现 $w(P)$ 不变, 接下来只需要说明符号取反。首先 $Q_G(i, j; w)$ 的定义保证了 i 和 j 在 π 中不匹配。

假设存在 $i < k < j$ 和 i, j 其一匹配, 不妨设 k 和 i 匹配。记 P_i, P_k 对应的终点编号分别为 id_i, id_k , P_i, P_j 分别被 M 分割成 AB 和 CD 。根据假设 P_k 和 A, B 均不交, 而由于 M 是拓扑序最小的交点, P_k 和 C 也不相交。由于 A 和 E 是 G -协调的, 必然有 $id_k > id_i$ 。现在考虑路径 P_k 以及 CB , 由于 CB 的终点 id_i 在 id_k 前面, P_k 和 CB 必然相交, 而前面推出了 P_k 和 C, B 均不相交, 矛盾。 \square

吴畅 (北京市十一学校)

不相交格路与反射容斥

January 12, 2024

15 / 26

续

Proof.

这样一来， ϕ 对 π 的影响，也就是交换 i 和 j 的位置，满足引理中的条件，因此符号必然取反。由于 A, C 上不存在其它的交点，也容易说明再进行一次变换会回到初始状态，进而 ϕ 是对合。
最后，待验证的是，每组合法的不交路径恰好贡献 1。由之前的命题这也是显然的。

□

假如路径数量（同起点集合）是奇数，我们只需向 G 加一个没有任何邻边的虚点 v ，同时将 v 添加到 A 和 E 的末尾。这样一来，起点集合的 v 只能和终点集合的自己以唯一的方式匹配，答案不变但转化为了偶数的情况。

吴畅（北京市十一学校）

不相交格路与反射容斥

January 12, 2024

16 / 26

起点也不固定

事实上，起点也可以不固定。更多信息可以参考本人的集训队论文以及互测题《网格图最大流计数》。

Theorem

令 G 是一张有向无环图， $A = (A_1, A_2, \dots, A_n)$, $E = (E_1, E_2, \dots, E_m)$ 是节点序列，满足 A 和 E 是 G -协调的， w 是边权函数。

$$\sum_{s \geq 0} x^s \sum_{A' \subseteq A, E' \subseteq E, |A'| = |E'| = s} GF(L_G(A' \rightarrow E' \mid \text{non-intersecting}); w)$$

当 n 是奇数时 $= Pf_{1 \leq i < j \leq n+1}((-1)^{i+j-1} + x^2 Q_G(i, j; w))$

其中 $Q_G(i, j; w)$ 在 $j \leq n$ 时与前述定义一致，而

$$Q_G(i, n+1; w) = x^{-1} \sum_{E \in E} GF(L(A_i \rightarrow E); w);$$

当 n 是偶数时 $= Pf_{1 \leq i < j \leq n+2}((-1)^{i+j-1} + x^2 Q_G(i, j; w))$

其中 $Q_G(i, j; w)$ 在 $j \leq n+1$ 时定义同上，而 $Q_G(i, n+2; w) = 0$ 。

吴畅（北京市十一学校）

不相交格路与反射容斥

January 12, 2024

17 / 26

反射容斥

两条斜率为 1 的直线边界

比较常见的反射容斥：

Theorem

令 n, m 是非负整数, l, r 是整数, 满足 $l < 0 < r, n + l < m < n + r$, 从 $(0, 0)$ 到 (n, m) , 始终不与 $y = x + l$ 和 $y = x + r$ 相交的格路数量为

$$\begin{aligned} & |L((0, 0) \rightarrow (n, m) \mid x + l < y < x + r)| \\ &= \sum_{k \in \mathbb{Z}} \left(\binom{n+m}{n-k(r-l)} - \binom{n+m}{n-k(r-l)+r} \right) \end{aligned}$$

简单来说, 将一条路径与两直线相交的情况写成字符串 s :

$$\begin{aligned} [s = \emptyset] &= [\emptyset \subseteq s] - [L \subseteq s] - [R \subseteq s] + [LR \subseteq s] + [RL \subseteq s] \\ &\quad - [LRL \subseteq s] - [RLR \subseteq s] + \dots \end{aligned}$$

一些约定

对反射容斥比较理想的描述，需要借助 root system 和 Weyl group 的概念进行刻画。这里只给出一些简化后的、最表层的定义和结论。

在 \mathbb{R}^d 上，令 \mathcal{H} 是一个超平面构成的有限集合， W 是将 \mathcal{H} 中所有元素看成边界，生成的反射群。注意到 W 中的一些反射变换并不沿着 \mathcal{H} 中的超平面进行，那么记 \mathcal{R} 表示 W 中的所有元素对应的超平面的集合。

\mathcal{R} 中的超平面将 \mathbb{R}^d 分割为许多区域， $\mathbb{R}^d - \bigcup_{H \in \mathcal{R}} H$ 中的连通块被称为腔。不难发现，对于任意腔 C ，取 $w \in W$ ，那么 $w(C)$ 两两不同且覆盖了全部的腔。

经过如上的若干定义，我们已经具备了描述更一般的反射容斥的工具。然而，为了使反射容斥适用，步长集合也需要加以约束，使得路径无法在不接触边界的情况下越过边界。为此，我们通过定义 k_H 和 r_H 来实现这一点，其中 $H \in \mathcal{H}$ 。

吴畅 (北京市十一学校)

不相交格路与反射容斥

January 12, 2024

20 / 26

反射容斥

Theorem

令 C 是反射群 W 对应的一个腔，由超平面集合 \mathcal{H} 生成。取步长集合 \mathbb{S} ，使得对任意 $w \in W$ ， $w(\mathbb{S}) = \mathbb{S}$ ，且对任意 $H \in \mathcal{H}$ ， $s \in \mathbb{S}$ ，内积 $\langle s, r_H \rangle$ 的值为 0 或 $\pm k_H$ 。其中 r_H 是一和 H 垂直的非零向量， k_H 是固定常数，它们都只和 H 有关。

对于 C 中任意两个整点 A, E ，满足对任意 $H \in \mathcal{H}$ ， $\langle A, r_H \rangle$ 和 $\langle E, r_H \rangle$ 都是 k_H 的整数倍。从 A 到 E ，恰好 m 步，且始终在 C 内部的格路数量为

$$|L_m(A \rightarrow E; \mathbb{S} \mid \text{inside } C)| = \sum_{w \in W} (\det w) |L_m(w(A) \rightarrow E; \mathbb{S})|$$

下面给出两个具体例子来理解这个定理。

吴畅 (北京市十一学校)

不相交格路与反射容斥

January 12, 2024

21 / 26

Example1

考虑在 \mathbb{Z}^2 中，步长集合为 $\{(1, -1), (1, 1), (F, 0)\}$ ，且始终不低于 x 轴的一类格路。当 $F = 1$ 时，对应的格路称为 **Motzkin 路**。当 $F = 2$ 时，对应的格路称为 **Schröder 路**。

Corollary

令 a, c 是整数， b, d 是非负整数，从 (a, b) 到 (c, d) 的 Schröder 路的数量为

$$|L((a, b) \rightarrow (c, d); S = \{(1, -1), (1, 1), (2, 0)\} \mid y \geq 0)| = \sum_{k=0}^{(c-a)/2} \binom{c-a-k}{k} \left(\binom{c-a-2k}{(c+d-2k-a-b)/2} - \binom{c-a-2k}{(c+d-2k-a+b+2)/2} \right)$$

吴畅 (北京市十一学校)

不相交格路与反射容斥

January 12, 2024

22 / 26

Example1

Proof.

如果没有 $y \geq 0$ 的限制，可以直接枚举 $(2, 0)$ 走了多少步，而剩下的步长和简单步本质相同，只需将坐标系旋转 45 度。那么我们有

$$|L((a, b) \rightarrow (c, d); S)| = \sum_{k=0}^{(c-a)/2} \binom{c-a-k}{k} \binom{c-a-2k}{(c+d-2k-a-b)/2}$$

应用反射容斥，将问题转化为从 (a, b) 到 (c, d) 的自由 Schröder 路数量，减去从 (a, b) 到 $(c, -d-2)$ 的自由 Schröder 路数量，代入上式即可完成证明。 \square

这类格路均适用反射容斥解决，其本质在于较为特殊的 $(F, 0)$ 步长与边界 $y = -1$ 平行，对应了 $\langle s, r_H \rangle = 0$ 的情况。

另外，此时可以取 $r_H = (0, 1), k_H = 1$ ，那么所有满足 $\langle s, r_H \rangle = k_H$ 的步长都可以成对加入进步长集合，即 $(x, 1)$ 和 $(x, -1)$ 。

吴畅 (北京市十一学校)

不相交格路与反射容斥

January 12, 2024

23 / 26

Example2

Corollary

令 d 是正整数, $\mathbf{a} = (a_1, a_2, \dots, a_d)$ 和 $\mathbf{e} = (e_1, e_2, \dots, e_d)$ 是 \mathbb{Z}^d 中的两格点, 且满足 $\forall 1 \leq i < d, a_i < a_{i+1}, e_i < e_{i+1}$. 从 \mathbf{a} 走到 \mathbf{e} , 且始终保持 $x_1 < x_2 < \dots < x_d$ 的路径数量为

$$|L(\mathbf{a} \rightarrow \mathbf{e} \mid x_1 < x_2 < \dots < x_d)| = \left(\sum_{i=1}^d (e_i - a_i) \right)! \det_{1 \leq i, j \leq d} \left(\frac{1}{(e_i - a_j)!} \right)$$

Example2

Proof.

在该情境中, 上述定理中的概念

$\mathcal{H} = \{x_i = x_{i+1} \mid 1 \leq i < d\}, \mathcal{R} = \{x_i = x_j \mid 1 \leq i < j \leq d\}$, 而 W 对应了全体 $\{1, 2, \dots, d\}$ 的置换, 因此我们有

$$\begin{aligned} & |L(\mathbf{a} \rightarrow \mathbf{e} \mid x_1 < x_2 < \dots < x_d)| \\ &= \sum_{\sigma \in \mathfrak{S}_d} (\text{sgn } \sigma) |L(\mathbf{a}_\sigma \rightarrow \mathbf{e})| \\ &= \sum_{\sigma \in \mathfrak{S}_d} (\text{sgn } \sigma) \frac{\left(\sum_{i=1}^d (e_i - a_i) \right)!}{\prod_{i=1}^d (e_i - a_{\sigma(i)})!} \\ &= \left(\sum_{i=1}^d (e_i - a_i) \right)! \det_{1 \leq i, j \leq d} \left(\frac{1}{(e_i - a_j)!} \right) \end{aligned}$$

□



后话

最后的例子是一个非常经典的问题，但在许多地方都是用 LGV 引理来解释的（事实上其形式确实非常像 LGV）。

这里给出了用反射容斥的证明，也在一定程度上，揭示了不相交格路与反射容斥的关系。

本课件仅是抛砖引玉的简单介绍了格路计数中两个比较常见的概念，更多的内容可以参考本人的集训队论文。也期待格路计数在 OI 中的更广泛的应用。

谢谢大家！



浅谈复杂度及其在解决问题方面的应用

南京外国语学校 杨敏行

前言

复杂度的概念

由数据范围推测复杂度

从特殊条件反推复杂度

精细复杂度问题初探

前言

- 因为笔者能力有限，所以本文的难度和创新程度可能有所不足，具有较高水平的读者可能不会有很多收获，还请多多海涵。
- 本文旨在介绍利用复杂度来解决问题的方法，提供一种可行的寻找正解的思路。其将包含从数据范围分析可能的复杂度、从几个特殊的条件推测复杂度、以及精细复杂度理论等数个方面的内容。

南京外国语学校 杨敏行

浅谈复杂度及其在解决问题方面的应用

前言

复杂度的概念

由数据范围推测复杂度

从特殊条件反推复杂度

精细复杂度问题初探

引入复杂度

一元复杂度

多元复杂度

简化复杂度

为什么要引入复杂度？

- 算法设计希望可以更快速地得到正确的答案。因此，我们需要寻找效率更优异的算法。
- 在现代计算机领域，通常认为四则运算、取模、左右移、与或非、异或、输入输出、赋值等操作，如果所操作的变量不过大，不超过计算机的字长（常见为 2^{32} , 2^{64} , 2^{128} 等），则可以认为只需一次操作即可完成。本文中理想地认为所有上述运算的效率相等，所以一份算法的效率就可以被看作是其执行的操作数目。
- 计算理论方面往往认为，输入规模更大时的算法效率是更重要的，因此，设计一个通过算法在输入规模 n 充分大时的表现来衡量其效率高低的方式是有必要的。
- 因此，我们引入复杂度的概念。

南京外国语学校 杨敏行

浅谈复杂度及其在解决问题方面的应用

前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精確复杂度问题初探
引入复杂度	一元复杂度	一元复杂度	多元复杂度	简化复杂度

一元复杂度的定义

- 对于两个函数 $f(n), g(n)$ ，如果存在实数 $M > 0$ 以及一个 $N > 0$ ，使得 $\forall n \geq N$ ，都有 $|f(n)| \leq M|g(n)|$ ，此时称 g 是 f 的渐进上界，使用 big-O-notation 记号可记作 $f = O(g)$ 。此时，认为函数 f 有着 g 或者 $O(g)$ 的复杂度。
- 注意到 $O(g)$ 其实指代了所有满足上述条件的 f 构成的集合，因此正确的写法其实应该是 $f \in O(g)$ ，但因为历史习惯原因，通常采取 $f = O(g)$ 的写法，但是这里的 $=$ 符号仍然表示 \in ，因此没有反向的 $O(g) = f$ 成立。

前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精確复杂度问题初探
引入复杂度	一元复杂度	一元复杂度	多元复杂度	简化复杂度

一元复杂度相关定理

- 易验证一元复杂度有着如下定理：
 - 传递性：若 $f = O(g), g = O(h)$ ，则 $f = O(h)$ 。
 - 线性性：若 $f = O(h), g = O(h)$ ，那么 $\lambda f + \mu g = O(h)$ ，其中 $\lambda, \mu \in \mathbb{R}$ 。
 - 乘法结合性：若 $f = O(F), g = O(G)$ ，那么 $f \times g = O(F \times G)$ 。
 - 常底数对数的互换：由换底公式，底数为常数的 $O(\log_a n)$ 可以互换底数，也即 $\log_b n = \frac{\log_a n}{\log_a b}$ ，而 $\log_a b$ 是在渐进复杂度中无影响的常数。因此，在研究底数为常数的对数时，我们常常忽略底数，统一记作 $O(\log n)$ 。

前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精確复杂度问题初探
引入复杂度	一元复杂度	一元复杂度	多元复杂度	简化复杂度

前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精确复杂度问题初探
引入复杂度	一元复杂度	多元复杂度	简化复杂度	

一元复杂度扩展定义

- 粗略定义一些常见的复杂度级别：
 - 对数多项式复杂度 ($O(\text{polylog}(f))$): 复杂度是关于 $\log f$ 的多项式。
 - 多项式复杂度 ($O(\text{poly}(f))$): 复杂度是关于 f 的多项式。
 - 指数复杂度: 复杂度是 a^f , 其中 a 是一个大于 1 的实数。
- 同时, 定义软复杂度概念: 如果 $f = O(g(n)\text{polylog}(g(n)))$, 则可记 $f = \tilde{O}(g(n))$ 。软复杂度的另一种定义是 $O(g(n)\text{polylog}(n))$, 但是为了有 $O(n^\alpha \beta^n)$ 可以被写作 $\tilde{O}(\beta^n)$, 所以采取第一种写法。

前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精确复杂度问题初探
引入复杂度	一元复杂度	多元复杂度	简化复杂度	

一元复杂度的问题

- 但是, 一元复杂度也有缺憾。例如, n 个数相加, 一共做了 $O(n)$ 次加法, 但是还需考虑加法自身的复杂度: 其与值域和字长有关, 因此要想精确刻画执行的操作数, 仅使用一个变量可能是不太合适的。
- 另一种常见思路是, 认为 n 是输入规模: 输入一个大小为 A 的数需要 $\log_2 A$ 个 bit; 这样描述虽然消减了变量数目, 但是当问题的输入更加繁杂时, 这样分析出的结果会比较混乱, 因此我们还是有必要引入多元复杂度的概念。

前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精确复杂度问题初探
引入复杂度	一元复杂度	多元复杂度	简化复杂度	

前言	复杂度的概念	由数据范围测复杂度	从特殊条件反推复杂度	精确复杂度问题初探
引入复杂度	一元复杂度	多元复杂度	简化复杂度	
邻域				

- 在定义多元复杂度的概念前，首先需要完善邻域的概念。
- 对于 $a \in \mathbb{R} \cup \{\pm\infty\}$ ， a 的一个邻域是指：
 - $(a - \delta, a + \delta)$ ，如果 $a \in \mathbb{R}$ ；其中 δ 是任何正实数。
 - $(N, +\infty)$ ，如果 $a = +\infty$ ；其中 N 是任何实数。
 - $(-\infty, N)$ ，其中 $a = -\infty$ ；其中 N 是任何实数。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用				
前言	复杂度的概念	由数据范围测复杂度	从特殊条件反推复杂度	精确复杂度问题初探
引入复杂度	一元复杂度	多元复杂度	简化复杂度	
多元复杂度				

- 对于两个多元函数 $f(x_1, x_2, \dots, x_n), g(x_1, x_2, \dots, x_n)$ ，若对于确定的一组 $y_1, y_2, \dots, y_n \in \mathbb{R} \cup \{\pm\infty\}$ ，有：
 - 存在一个实数 $M > 0$ ，使得存在 y_1, \dots, y_n 的邻域，对于该邻域中的任何 z_1, \dots, z_n ，都有 $|f(z_1, \dots, z_n)| \leq M|g(z_1, \dots, z_n)|$ 。
 此时，称 g 是 f 在 y_1, y_2, \dots, y_n 处的渐进上界，记作 $f = O(g)$ 或者 $f \in O(g)$ 。
- 同时，一元复杂度的对数多项式复杂度、多项式复杂度和软复杂度的概念也可以简单推广到多元复杂度。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用				
-------------------------------	--	--	--	--

前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精确复杂度问题初探
引入复杂度	一元复杂度	多元复杂度	简化复杂度	

多元复杂度相关定理

- 首先，易知传递性、线性性、乘法结合性、常底数对数的互换等性质可以推广至多元复杂度的场合。
- 其次，如果变量的值域是 V ，计算机的字长是 $w = 32/64/128 \dots$ ，那么加减、与或非、异或、输入输出、赋值等操作的复杂度按位处理是 $O(\log_2 V)$ ，而乘除、取模等操作，使用 FFT 算法可以在 $O(\log_2 V \log_2 \log_2 V)$ ；整理可得加减类运算的复杂度 $O(\frac{\log V}{w})$ ，乘除类运算的复杂度 $O(\frac{\log V}{w} \log \frac{\log V}{w})$ 。

南京外国语学校 杨敏行	浅谈复杂度及其在解决问题方面的应用
-------------	-------------------

前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精确复杂度问题初探
引入复杂度	一元复杂度	多元复杂度	简化复杂度	

为什么要简化复杂度？

- 复杂度可以直接代入变量的范围，得到一个对操作数目的估计。这是复杂度最常见的应用：实际应用时各个变量往往是有范围限制的，并不会真的趋向无穷。
- 比如说，在一个问题中，如果有 $n \leq 10^5$ ，那么基本可以立刻放弃所有复杂度为 $O(n^2)$ 的做法，并且认为所有复杂度为 $O(n \log n)$ 的算法基本上都是可以通过的。复杂度就像一个标签，它帮助我们快速过滤掉一批算法，并判定另一些算法有成为正解的可能。
- 标签之间也有好坏之分：例如 $O(n)$ 就明显比 $O(\frac{n \log A}{w})$ 要更加简洁。而假如在给定的数据范围下， $\frac{\log A}{w}$ 一项的值不算很大，那么把它看成一个常数也未尝不可。比如说，在前述的 n 个数求和问题中，一个常见的数据范围是 $n \leq 10^5, A \leq 10^9, w = 32$ ，那么就有 $O(\frac{n \log A}{w}) \in O(\frac{n \log 10^9}{32}) = O(n)$ 。

南京外国语学校 杨敏行	浅谈复杂度及其在解决问题方面的应用
-------------	-------------------

前言	复杂度的概念	由数据范围推复杂度	从特殊条件反推复杂度	精确定义问题初探
引入复杂度	一元复杂度	多元复杂度	简化复杂度	

如何简化复杂度

- 从另一个角度说，也有 $O(\frac{n \log A}{w}) \in O(\frac{10^5 \log 10^9}{32}) = O(1)$ 。但是，此时这个 $O(1)$ 的标签就不是一个合适的标签，因为向其中代入得到的结果并非对实际操作数的合格估计。
- 这说明，将复杂度式子中一些量看成常数有助于简化问题，但是如果不分青红皂白就将一切变量都看作常数，则并非一个好的近似。
- 同样的简化复杂度形式的操作还有，将一些变量用其它变量表示（比如说，简单图的边数 m 数目不超过 n^2 ，于是涉及到 m 的复杂度 $f(m)$ 可以被写作 $f(n^2)$ ）；将数据范围相同的变量合并（比如说一道题元素个数 n 和询问次数 q 的范围都是 10^5 ，那么它们就可以统一以 n 表示）。这些操作以减少复杂度的泛用性和严谨性为代价，大大降低了复杂度描述的繁琐度。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用

前言	复杂度的概念	由数据范围推复杂度	从特殊条件反推复杂度	精确定义问题初探
----	--------	-----------	------------	----------

前提

- 信息学竞赛遇到的问题与实际生活中遇到的问题相比，区别之一是竞赛的问题总是有解的。这意味着存在一个算法可以在题目的数据范围下通过问题，于是数据范围也可以成为辅助我们寻找答案的工具之一。
- 从数据范围到算法，我们需要复杂度作为桥梁。所以我们才需要适当地对复杂度加以简化，否则枚举各种复杂得毫无意义的复杂度是舍本逐末的。
- 出于同样的道理，我们仅考虑最基础的场合，即认为输入仅有一维 n 是重要的。至于更多变量的场合，如果不存在乘积量，可以尝试对于每个变量分开分析，因为复杂度具有线性性。否则，可以尝试把乘积整体当作变量分析。如果上述分析无法成立，则这种过于复杂的场合或许并不是根据数据范围反推复杂度的合适情景。另外，下文仅提供思路的参考，具体问题还是应该具体分析。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用

前言

复杂度的概念

由数据范围推测复杂度

从特殊条件反推复杂度

精读复杂性问题的初探

数据范围对应的复杂度

- $n \leq 1 \sim 12$: 可能对应着一些比较高的复杂度, 例如 $4^n, 5^n$ 或者 $Bell(n), n!$ 等。当然, 这些复杂度都可以是软复杂度。
- $n \leq 15 \sim 17$: 可能对应着各种大到小 $\text{poly}(n)$ 系数的 $\tilde{O}(3^n)$, 或者具有较大 $\text{poly}(n)$ 系数的 $O(2^n)$ 。
- $n \leq 20$: 2^n , 带有一般来说不超过二次的 $\text{poly}(n)$ 系数。需要注意的是, 也有一种偶见的 $\binom{n}{n/2}$ 的复杂度, 例题如 UNR#6 D2T2 神隐、九省联考 2018 一双木棋, 可自行阅读论文查看相关解法。
- $n \leq 26 \sim 28$: 这是 $O(2^n)$ 复杂度的上限。需要注意的是, 有时复杂度可以是 $\sum_{i=0}^n 2^i = O(2^{n+1})$ 。

南京外国语学校 杨敏行

浅谈复杂度及其在解决问题方面的应用

前言

复杂度的概念

由数据范围推测复杂度

从特殊条件反推复杂度

精读复杂性问题的初探

数据范围对应的复杂度

- $n \leq 30 \sim 60$: 这些数据范围往往不会有多项式复杂度。例如 DP on DP、划分数 DP、剪枝后的搜索、meet in middle 等算法是此区间内较主流的算法。
同时, 对于所有的非多项式复杂度算法, 永远都存在一种搜索所有本质不同的状态并通过压缩、分类等操作将状态数削减至可以承受的范围的方法。分析此种方法的复杂度往往是毫无意义且与实际大相径庭的, 此时最佳策略莫过于亲自写一份暴力得到状态数: 可能会发现真实的状态数其实出乎意料得小。大部分 DP on DP 题都能在这种策略下找到思路。
- $n \leq 50 \sim 80$: 可能存在一些具有高次多项式复杂度 (例如 $\tilde{O}(n^6)$ 或者 $\tilde{O}(n^7)$) 的 DP 在此范围内。需要注意的是, 高次多项式复杂度往往可能拥有一个较小的常数 (因为循环嵌套时, 内层循环的起讫值依赖于外层循环, 进而会产生一个小于 1 的常数), 所以此时在理论的大 O 复杂度内代入 n 得到的结果和实际运行的效率可能会有较大的分野, 因此有时不能因为理论复杂度大就贸然放弃一个算法的可能性。

南京外国语学校 杨敏行

浅谈复杂度及其在解决问题方面的应用

前言

复杂度的概念

由数据范围推导复杂度

从特殊条件反推复杂度

精确复杂度问题初探

数据范围对应的复杂度

- $n \leq 100$: 随着计算机性能的提高, 近年来 $n \leq 100$ 的数据范围往往已经对应着 $O(n^4)$ 的复杂度了。
- $n \leq 200 \sim 500$: 可能对应着带有不同级别 $\text{polylog}(n)$ 的 $\tilde{O}(n^3)$ 、 $O(n^{3+\varepsilon})$ (其中 $\varepsilon \in (0, 1)$)、直到严格的 $O(n^3)$ 。需要注意的是, $n \leq 150 \sim 200$ 可能有一种特殊的技巧, 将在下文提到。
- $n \leq 1000 \sim 10000$: 带有不同级别的 $\text{polylog}(n)$ 的 $\tilde{O}(n^2)$ 、 $O(n^{2+\varepsilon})$ 、直到严格 $O(n^2)$ 。除此之外, 在 $n \leq 1000$ 左右, 有一些复杂度为 $\frac{n^3}{w}$ 的使用 bitset 的做法, 或者复杂度为 $\frac{n^3}{\log n}$ 的做法。

南京外国语学校 杨敏行

浅谈复杂度及其在解决问题方面的应用

前言

复杂度的概念

由数据范围推导复杂度

从特殊条件反推复杂度

精确复杂度问题初探

数据范围对应的复杂度

- $n \leq 5 \times 10^4$: 这是一个几乎所有 $\tilde{O}(n^{1+\varepsilon})$ 都可以通过的复杂度级别; 虽然有一些例如 $O(n\sqrt{n}\log^2 n)$ 的复杂度或许不太可行。在 $5 \times 10^4 \sim 5 \times 10^5$ 的范围内, $O(\log n)$ 和 $O(n^{0.25})$ 的表现通常是相近的, 虽然前者具有较优的理论复杂度, 但是后者 (通常在 $O(n\log^2 n)$ 和 $O(n\sqrt{n})$ 作比较时) 往往有着较小的常数, 综合而言, 二者效果一般大致相同。同时, 像 $O(n\sqrt{n}\log^2 n)$ 或 $O(n\log^4 n)$ 之类的复杂度, 其对应的算法即使效率或许可以承受, 代码的复杂程度也是不可接受的。
另外, 存在一些 $O(\frac{n^{t=1/2/3}}{\log n})$ 的算法, 比如说 The Method of Four Russians。 $t = 2$ 时的该技巧有可能在这种数据范围下应用。
- $n \leq 10^5$: 例如 $n\sqrt{n}\log n, n\log^3 n$ 这样的复杂度有可能通过这一级别, 前提常数必须较小。另外, 此数据范围下一个不可忽略的算法是 $\frac{n^2}{w}$ 的 bitset 算法。同时, 这往往是 KDT 能处理的数据范围的极限: KDT 具有过大的常数。

南京外国语学校 杨敏行

浅谈复杂度及其在解决问题方面的应用

前言

复杂度的概念

由数据范围推测复杂度

从特殊条件反推复杂度

精确复杂度问题初探

数据范围对应的复杂度

- $n \leq 2 \times 10^5$: $n\sqrt{n\log n}$ 有可能通过本数据范围。常数中等的 $n\sqrt{n}$ 亦可。某些常数很大的 $n\log^2 n$ 可能只能通过这种范围。
- $n \leq 5 \times 10^5$: 常数很小的 $n\sqrt{n}$ 有可能可以通过。常数适中的 $n\log^2 n$, 以及 LCT 这种常数较大的 $n\log n$ 也会有这种数据范围。
- $n \leq 10^6$: 常数较小的 $n\log^2 n$ (例如二分、树状数组、排序、分治的 $\log n$), 常数不过大的 $n\log n$ 。
- $n \leq 2 \times 10^6 \sim 5 \times 10^6$: 常数中等至很小的 $n\log n$ 。也可能是某些亚 $n\log n$ 做法, 例如 $n\sqrt{\log n}$, 或者 $O(n\alpha(n))$ 的并查集。
- $n \leq 10^7 \sim 5 \times 10^7$: 往往意味着 $O(n)$ 的复杂度, 其中有可能同时包含着对空间的限制。

南京外国语学校 杨敏行

浅谈复杂度及其在解决问题方面的应用

前言

复杂度的概念

由数据范围推测复杂度

从特殊条件反推复杂度

精确复杂度问题初探

数据范围对应的复杂度

- $n \leq 10^8 \sim 10^9$: 往往任何低于 $O(n)$ 的复杂度都可以通过, 但一般 $\text{polylog}(n)$ 不会只出这么小的复杂度。某些分块打表的做法也可能通过这种数据范围。筛法是这种数据范围的常客: 其中 $O(n^{2/3})$ 的杜教筛往往在这种范围下出现。
- $n \leq 10^{10} \sim 10^{11}$: 一般是 min25 筛的范围, 其它算法出这个级别的比较少见。
- $n \leq 10^{12} \sim 10^{14}$: 常见筛法中能通过这种范围的或许只有 $O(\sqrt{n})$ 的 powerful number 筛。其它 $\tilde{O}(\sqrt{n})$ 的算法也可能出这个数据范围。
- 更大的 n : 此时往往不会有 n^α 的复杂度 (但或许可能会出现 $n^{1/3}$ 之类), 更常见的还是 $\text{polylog}(n)$, 例如数位 DP。这种数据范围下, 一种可行的分析方式是取 $n' = \log n$ 然后对 n' 展开如上文的分析。

南京外国语学校 杨敏行

浅谈复杂度及其在解决问题方面的应用



前言 复杂度的概念 由数据范围推测复杂度 从特殊条件反推复杂度 精研复杂性问题的初探

总结

- 上述数据范围对应的复杂度仅供参考。同时，经常也能见到一些题目，如果复杂度并非题目考察的关键， $n \log n$ 的复杂度只出 $n \leq 10^5$ 的数据范围也是有可能的：也即，复杂度可以向下兼容数据范围，但不能向上。同时，一些很复杂的复杂度可能是光靠猜想不可能得到的，不能想当然地认为复杂度只有上述列的几种。笔者希望达到的效果是，看到 10^6 不去考虑根号算法，看到 10^5 记得试一试 bitset 优化，看到 40 想想 meet in middle，而不是遇到问题就来查表。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用

前言 复杂度的概念 由数据范围推测复杂度 从特殊条件反推复杂度 精研复杂性问题的初探

前言

- 本节将介绍一些很有特征的条件，其常常对应着一些比较优秀的复杂度。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用

前言 复杂度的概念 由数据范围推测复杂度 从特殊条件反推复杂度 精确复杂度问题初探

很小的 K

- 一些问题要求统计 n 个元素中，所有大小不超过 K 且额外满足某些特殊性质的元素集合，对应的某种函数的最值。如果这个 K 很小，一种常见的方法是对元素随机染色，计算不同颜色间的上述集合的最值。答案集合期望在不过多次随机内被正确染色，进而被求出。
- 例：恰经过 K 个不重复点的最优路径。其余例题可见论文。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用

前言 复杂度的概念 由数据范围推测复杂度 从特殊条件反推复杂度 精确复杂度问题初探

$n \times m \leq \text{定值}$

- 这是一类常见限制，令 $n \times m \leq C$ ，则 $\min(n, m) \leq \sqrt{C}$ 。于是，算法可能包含一些与 \sqrt{C} 有关的复杂度，例如 $\tilde{O}(2^{\sqrt{C}})$ 或者 $O(C\sqrt{C})$ 之类。
- 此时，解法往往与 n, m 二维构成的网格图有关：设计算法分别从 n, m 两维进行处理。有时 n, m 两维是对称的因此只需设计一个算法，但更多的时候是不对称的。
- 例：网格图上状压，状压 n, m 较小一侧的结果。其余例题可见论文。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用

前言 复杂度的概念 由数据范围推测复杂度 从特殊条件反推复杂度 精细复杂度问题初探

$\sum_{i=1}^n a_i = A$ 的启示

- $\sum_{i=1}^n a_i = A$ 的条件能带来很多有趣的结论：
 - 不同的 a_i 数目是 $O(\sqrt{A})$ 的。
 - 上一条可以扩展为，所有的 a_i 可以分为两类： $\leq \sqrt{A}$ 的，和 $O(\sqrt{A})$ 个大于 \sqrt{A} 的。
 - 如果额外有 $a_i \leq n$ (例： a_i 是简单图的边数)，那么 $\min a_i$ 是 $O(\sqrt{A})$ 的。
- 这几条限制都有着很多用处。第一条可以用于构建压缩 trie (Radix Tree, Radix 树)：用长度集合为 a_i 的串建立的压缩 Trie 的深度是 \sqrt{A} 级别的，进而一些暴力算法可能有较优的复杂度；第二条是很经典的根号分治的思想，在包括众数问题、图论等情形都经常被应用；第三条体现在降低一些搜索问题的复杂度，其应用可见论文。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用

前言 复杂度的概念 由数据范围推测复杂度 从特殊条件反推复杂度 精细复杂度问题初探
前言 暴力算法及其相关问题 其它经典问题的复杂度下界

为什么要研究精细复杂度问题

- 除了从数据范围来推理复杂度，还可以从待解决的的问题来推理复杂度。在思考问题的时候，一个常见的思路是将原问题转化成其它的问题；而转化至的问题有可能是一些经典的、有着复杂度下界的问题。因此，牢记这些经典问题的下界，也可以助力迅速判定原问题转化的思路是否正确。
- 这方面的理论，被称作 Fine Grained Complexity 问题，即精细复杂度问题。本节将介绍数个精细复杂度理论中较为常用的结论。
- 本章专注于实用解的概念，基本不关注在信息学竞赛方面几乎没有实用性的做法。因此，一些问题给出的复杂度可能与现在提出的最优复杂度不同。相反，此处认为 bitset 优化是很有价值的，并会特别指出某些算法存在 bitset 优化的解法。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用

前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精细复杂度问题初探
前言		矩阵乘法及其相关问题		其它经典问题的复杂度下界

矩阵乘法的复杂度

- 两个 $n \times n$ 矩阵的乘法是精细复杂度问题中极其基础的一个环节。截至完稿日，现行复杂度最优的矩阵乘法已可以在 $O(n^{2.37188})$ 的复杂度内解决，将来也必然存在更优秀的算法。但是，在信息学竞赛方面，基本上任何 $\tilde{O}(n^{3-\epsilon})$ 的算法都实现过于复杂或常数过大，基本不实用。
- 因此竞赛方面认为两个 $n \times n$ 的矩阵乘法的实用复杂度是 $O(n^3)$ 。特别地，0/1 矩阵的乘法（其中 0/1 间的乘法看作与，加法看作或）可以用 bitset 优化至 $O(\frac{n^3}{w})$ ，其中 w 是计算机字长。

		南京外国语学校 杨敏行	浅谈复杂度及其在解决问题方面的应用	
--	--	-------------	-------------------	--

前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精细复杂度问题初探
前言		矩阵乘法及其相关问题		其它经典问题的复杂度下界

可规约至矩阵乘法的简单问题

- 有若干问题都是可以归约至矩阵乘法的，例如行列式、求逆、有向图上传递闭包、高斯消元。因此，这些问题同样只有 $O(n^3)$ 的实用解，其中传递闭包更常见的是 bitset 优化解法。
- 除此之外，还存在一种方法，其将问题弱化或转化，使得转化后的问题一旦被解决则矩阵乘法被解决。于是原问题不弱于矩阵乘法问题，进而其复杂度也就拥有了下界。这种手法被称作规约矩乘。

		南京外国语学校 杨敏行	浅谈复杂度及其在解决问题方面的应用	
--	--	-------------	-------------------	--

前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精確复杂度问题初探
前言		矩阵乘法及其相关问题		其它经典问题的复杂度下界
规约矩乘的例子				

- 例：给定一棵 n 个点、点上带颜色树，多次询问链上不同颜色数目。
- 令 $m = \sqrt{n/2}$ 。考虑自根挂下 $2m$ 条长度为 m 的链，令 $A_{i,j}$ 表示第 i 条链上是否有第 j 种颜色， $B_{j,i}$ 表示第 $i+m$ 条链上是否有第 j 种颜色（其中 $1 \leq i, j \leq m$ ），则一切 A, B 矩阵都可以被构造。而对第 i 条链底点和第 j 条链底点的询问，就等价于询问 A, B 两矩阵乘积中下标为 i, j 处的值，询问 m^2 次即可问出整个矩阵。因此，如果原问题中点数和询问数同阶，则原问题的最劣情况不弱于计算 $m \times m$ 矩阵乘法，那么其不存在低于 $O(m^3) = O(n\sqrt{n})$ 的实用解。
- 可以发现，规约矩乘是一个很强大的工具，运用得当的话可以排除相当一部分错误的做法：假如用规约矩乘证明了复杂度不低于 $n\sqrt{n}$ ，显然再去考虑 polylog 解法就是很荒谬的。
- 其余可以用规约矩乘证明的例子有区间逆序对计数、区间相等对计数等。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用				
前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精確复杂度问题初探
前言		矩阵乘法及其相关问题		其它经典问题的复杂度下界
LCS 问题				

- 给定两个长度为 n 的串，寻找其最长公共子序列。
- 结论：不存在低于 $O(n^2)$ 的解。特别地，存在 $O(\frac{n^2}{w})$ 的 bitset 做法，读者可自行查找 LOJ#6564 的题解。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用				
-------------------------------	--	--	--	--

前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精确定义性问题初探
前言		矩阵乘法及其相关问题		其它经典问题的复杂度下界
(min, +) 卷积				

- 给定长度为 n 的序列 a, b , 计算 $c_k = \min_{i+j=k} \{a_i + b_j\}$ 。
- 结论：不存在低于 $O(n^2)$ 的解。特别地，如果 a, b 满足凸性，可以用类 Minkowski 和做法做到 $O(n)$ 。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用				
前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精确定义性问题初探
前言		矩阵乘法及其相关问题		其它经典问题的复杂度下界
KSUM 问题				

- 在长为 n 的数列中，寻找大小恰为 K 的零和子集。
- 3SUM、4SUM 都只有 $O(n^2)$ 做法。还有一些问题与之等价，例如长为三的等差数列计数、三点共线计数、三线共点计数、 $a_i + a_j = a_u + a_v$ 计数等。
- 需要注意的是，存在与 $A = \max a$ 有关的高速做法，例如朴素的 FFT 卷积做法，当 $K = O(1)$ 时，结合容斥可以做到 $O(A \log A)$ 。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用				
-------------------------------	--	--	--	--



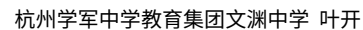
前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精细复杂度问题初探
前言		矩阵乘法及其相关问题		其它经典问题的复杂度下界
APSP 问题				

- 全源最短路问题，其复杂度下界是 $O(n^3)$ 的 Floyd 算法。特别地，使用 bfs，可以达到 $O(n^2 w)$ 的时间复杂度，其中 w 是最大边权。
- 一个与其类似的问题是矩阵的 $(\min, +)$ 乘法，复杂度下界也是 $O(n^3)$ 。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用				
前言	复杂度的概念	由数据范围推测复杂度	从特殊条件反推复杂度	精细复杂度问题初探
前言		矩阵乘法及其相关问题		其它经典问题的复杂度下界
进一步学习精细复杂度理论				

- 读者可自行在网上搜索相关内容，或者参阅 MIT 的线上课程资料 <https://people.csail.mit.edu/virgi/6.s078/>。

南京外国语学校 杨敏行 浅谈复杂度及其在解决问题方面的应用				
-------------------------------	--	--	--	--



杭州学军中学教育集团文渊中学 叶开

信息学竞赛中，我们总是会见到许多关于随机过程停时的问题。我们知道，生成函数是一个很有力的工具，我们能否使用生成函数来刻画随机过程的停时呢？

现有一随机数生成器，每次生成 $1 \leq j \leq n$ 的概率为 $A_j / \sum_{i=1}^n A_i$ ，使用其不断独立随机生成。
当对任意 $1 \leq j \leq n$ 都有 j 被生成了至少 B_j 次时过程停时。
求期望停时。
 $A_i, B_i \in \mathbb{N}^+$, $1 \leq n$, $\sum A_i, \sum B_i \leq 400$ 。

引入

这类问题（而不单是这道题目）的常见思路？

- ▶ min-max 容斥。
- ▶ 生成函数。
- ▶ 势函数与鞅的停时定理。

接下来着重探讨第二种做法，并给出一类较为通用的方法。

由于时间限制，概率论的基本概念不在此赘述，默认大家都已掌握。

用 PGF 刻画停时

设对于一个总会发生停时的过程， $F(z) = \sum_j P(j \text{ 时刻停时})z^j$ ，

那么总有：

结论 1: $F(1) = 1$ 。

结论 2: 期望停时为 $F'(1)$ 。

理由显然。

因此问题往往就被转化为刻画出 $F(z)$ 的信息。

用 EGF 刻画时间轴

由于 EGF 乘法能够分配标号，使用 EGF 刻画时间轴往往会比较容易。
但是问题来了：使用 PGF 刻画停时信息时往往需要 OGF 形式，怎么办呢？

Laplace 变换

我们规定 $\hat{F}(z) = \sum_j A_j z^j / j!$ 的 Laplace 变换结果为

$$F(z) = \mathcal{L}\hat{F}(z) = \sum_j A_j z^j$$

其中 \mathcal{L} 称为 Laplace 变换算子。
容易发现这实现了从 EGF 到 OGF 的变换。

Laplace 变换

结论 1: Laplace 变换算子是线性算子。

结论 2: $\mathcal{L}z^a e^{bz} = a! z^a / (1 - bz)^{a+1}$ 。

因此对于形如 $z^a e^{bz}$ 的线性组合的 EGF，我们总可以将其改写成 OGF 形式。

例题

以上内容已经可以解决一大批题目了。
我们先看一下引言中的题目。

Gachapon

现有一随机数生成器，每次生成 $1 \leq j \leq n$ 的概率为 $A_j / \sum_{i=1}^n A_i$ ，使用其不断独立随机生成。
当对任意 $1 \leq j \leq n$ 都有 j 被生成了至少 B_j 次时过程停时。
求期望停时。
 $A_i, B_i \in \mathbb{N}^+$, $1 \leq n$, $\sum A_i, \sum B_i \leq 400$ 。

Gachapon

直接做不好做，不妨枚举最后一步生成的数，最后把 1 的贡献加回去即可。

设 $S = \sum_i A_i$ ，那么立刻就有答案为

$$1 + \left(\mathcal{L} \left(\sum_{i=1}^n \frac{(A_i z / S)^{B_i-1}}{(B_i-1)!} \prod_{j \neq i} \left(\exp(A_j z / S) - \sum_{k < B_j} \frac{(A_j z / S)^k}{k!} \right) \right) \right)' \circ 1$$

设 $u = \exp(z/S)$ ，容易把内部的式子经过暴力乘法表示成关于 u, z 的多项式，复杂度不会超过 $O(v^3)$ （其中 $v = \max\{S, \sum B\}$ ）。最后直接把每个 $z^a u^b$ 对答案的贡献单独计算一遍即可。

例题

事实上可以发现，这样不仅能求出停时，还能求出以其中某些事件发生停时的概率。
求出对应的 PGF 后直接代入 1 即为答案。

猎人杀

现在有 n 个人，每人有一个权值 w_i 。
每次会在剩下的人集合中，以 w_i 比例的概率独立随机选择第 i 个人让其离开，重复直到所有人都离开。
求最后离开的人是第一个人的概率，对 998244353 取模。
 $w_i \in \mathbb{N}^+$, $1 \leq \sum_{i=1}^n w_i \leq 10^5$ 。

猎人杀

由于概率会变化，看上去这并不好直接做。
但是注意到，如果允许选择一个已经离开的人再次离开，那么这就变成了和上一题类似的问题，只是 $B_i = 1$ 。
这样，令 $S = \sum w_i$ ，就有答案为

$$\frac{w_1}{S} (\mathcal{L}(\prod_{j=2}^n (\exp(w_j z/S) - 1))) \circ 1$$

设 $u = \exp(z/S)$ ，使用分治卷积算法在 $O(S \log^2 S)$ 时间内计算出关于 u 的多项式后即可对每个 u^a 算出对答案的贡献，进而求出答案。

使用 $f = \exp \ln f$ （其中 $f = \prod_{j \neq 1} (1 - u^{w_j})$ ）的技巧可以进一步做到 $O(S \log S)$ ，此处略去。

通用测评号

现在有 n 个盒子，每个盒子最多放 a 个球，初始均为空。
不断独立均匀随机选取一个不满的盒子放入一个球，直到所有盒子内的球数均 $\geq b$ 发生停时，问最后期望有多少盒子被填满，对 998244353 取模。

$1 \leq n \leq 250, 1 \leq b < a \leq 250$ 。

通用测评号

先特判掉 $n = 1$ 的情况：此时答案显然为 0，无须考虑。
注意到答案即为每个盒子被填满的概率和，也即 n 乘上单个盒子被填满的概率，因此只需求出单个盒子被填满的概率。
容易发现假如允许继续往已经被填满的盒子里填，答案是不变的。枚举最后一步选择的是哪个盒子，就得到概率为

$$\frac{n-1}{n} (\mathcal{L}_z((\exp t - \sum_{j=0}^{a-1} \frac{t^j}{j!})(\exp t - \sum_{j=0}^{b-1} \frac{t^j}{j!})^{n-2} \frac{t^{b-1}}{(b-1)!}))_z(1)$$

其中 $t = z/n$ 。

通用测评号

设 $u = \exp t$ ，那么发现核心就是要快速求出

$$(u - \sum_{j=0}^{a-1} \frac{t^j}{j!})(u - \sum_{j=0}^{b-1} \frac{t^j}{j!})^{n-2} \frac{t^{b-1}}{(b-1)!}$$

容易使用 ODE 在 $O(n^2 a)$ 的复杂度下求出，设为 $\sum_{i,j} F_{i,j} t^i u^j$ ，那么容易得到概率即为

$$(n-1) \sum_{i,j} \frac{i! F_{i,j}}{(n-j)^{i+1}}$$

更多例题

在解决问题时，有时还要结合一类特殊的符号化方法。这样还可以解决更多题目，譬如下面这题。

Slime and Biscuits

n 个人，第 i 个人有 a_i 个球。

设 $S = \sum a_i$ ，每次会在 S 个球中独立均匀随机选择一个并传给任意一个不是当前人的人，直到所有球都归一人所有后过程停时。

求期望停时对 998244353 取模后的值。

$a_i \in \mathbb{N}$, $1 \leq n \leq 10^5$, $1 \leq S \leq 3 \times 10^5$ 。

本题使用这类方法，可以通过一些繁复的技巧做到 $O(n + S)$ 。但由于时间所限，在这里不进行展开了，感兴趣的同学可以阅读我的集训队论文。

《世界沉睡童话》命题报告

中国人民大学附属中学 叶李蹊

题目大意

给定一棵大小为 n 的有标号有根树 T ，其中结点 Rt 是 T 的根。
 设最初结点 u 的点权是 a_u ，保证 $a_1 \sim a_n$ 构成一个 $1 \sim n$ 的排列。
 注意一个结点的儿子之间是有顺序的。设结点 u 有 k_u 个儿子，它们从左到右依次称为 $ch(u, 1), ch(u, 2), \dots, ch(u, k_u)$ 。
 由于结点的儿子之间有顺序，以 Rt 为根 DFS 这棵树将得到唯一一个 DFS 序列 s ，其中结点 s_i 第 i 个被访问到。
 接下来进行 $n-1$ 次删边操作，第 i 次删边操作删去 $(s_{i+1}, Fa(s_{i+1}))$ 。
 在此之前，你需要选择是否交换 $a_{s_{i+1}}$ 与 $a_{Fa(s_{i+1})}$ 。
 当所有边尽数被删去以后，设最终结点 u 的点权是 a'_u 。定义 S 表示所有可能的排列 $a'_1 \sim a'_n$ 构成的集合。
 最后给定一个 $1 \sim n$ 的排列 p ，你需要在下列三个小问中选择一个回答：

- ① 判定 p 是否 $\in S$ ；
- ② 求出 S 中字典序大于 p 的最小排列 q ；
- ③ 求出 S 中字典序小于 p 的排列个数，对 998244353 取模。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

2 / 36

数据范围

以下是各个子任务的具体情况以及特殊性质：

子任务编号	分值	$n =$	树的形态	特殊性质	子任务依赖
1	10	20			
2	10	8×10^4	完全二叉树	DFS 序 $1 \sim n$	
3	10	8×10^4	完全二叉树		2
4	10	8×10^4	二叉树	DFS 序 $1 \sim n$	2
5	10	8×10^4	二叉树		2 ~ 4
6	10	8×10^4		DFS 序 $1 \sim n$	2, 4
7	10	8×10^4			1 ~ 6
8	10	1.2×10^5			1 ~ 7
9	10	1.6×10^5			1 ~ 8
10	10	2×10^5			1 ~ 9

- 二叉树：保证 T 中每个结点的儿子个数都 ≤ 2 。
- 完全二叉树：保证 T 与一棵大小同样为 n 的完全二叉树同构，并且 Rt 对应完全二叉树的根。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

3 / 36

评分标准

在一个测试点中：

- 如果你正确回答了第 1 小问，将获得该测试点满分 10% 的分数；
- 如果你正确回答了第 2 小问，无论第 1 小问是否回答、回答正确与否，都将获得该测试点满分 70% 的分数；
- 如果你正确回答了第 3 小问，无论第 1, 2 小问是否回答、回答正确与否，都将获得该测试点满分 100% 的分数。

定义与记号

- 记 b_x 表示初始时点权为 x 的结点的标号，则 b 与 a 互为逆排列。
- 再记 t_i 表示标号为 i 的结点在 DFS 的过程中第几个被访问到，则 t 与 s 也互为逆排列。

Definition

染色方案 假如一条边在被删去之前交换了两端点的点权，我们把这条边染黑；否则把这条边染白。我们记 c_u 表示 $(u, Fa(u))$ 这条边的颜色； $c_u = 1$ 表示黑色， $c_u = 0$ 表示白色。

- 做出 $n - 1$ 次选择一共有 2^{n-1} 种方案，一一对应 2^{n-1} 种不同的染色方案。我们用染色方案代指选择方案。
- 根据一种染色方案交换点权，称最终点权构成的排列 p 为该种染色方案导出的结果。

不同染色方案导出的结果一定不同

Theorem

$|S| = 2^{n-1}$ 。即任意两种不同的染色方案导出的结果也一定不同。

Proof.

两种染色方案是不同的，说明至少存在一条边 $(u, Fa(u))$ ，在第一种方案中 $c_u = 0$ ，在第二种方案中 $c_u = 1$ 。
连接 u 子树和外部的途径只有 $(u, Fa(u))$ 一条。我们将 u 子树内结点的初始点权标为 0， u 子树外的标为 1。
如果交换 $(u, Fa(u))$ ，最终 u 子树内就会恰有一个 1；否则就会全部是 0。因此交换 $(u, Fa(u))$ 与不交换 $(u, Fa(u))$ ，最终点权肯定不会完全相同。 \square

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

6 / 36

第一小问的做法

我们可以由此得出第 1 小问的做法：给定一个排列 p ，我们先 DFS 一遍，对每个结点 u 求出 u 子树内所有 $t_{b_{p_i}}$ 的最大值、最小值。
若最小值 $= t_u$ 且最大值 $= t_u + \text{siz}(u) - 1$ ，则说明 $(u, Fa(u))$ 必须染白；否则说明 $(u, Fa(u))$ 必须染黑。
得到每条边的颜色之后，我们再检查一遍交换过后的结果是否确实 $= p$ 即可。时间复杂度 $O(n)$ ，可以获得 10 分。

Definition (部分染色方案)

在一个部分染色方案当中，一条边可以被确定染白，也可以被确定染黑，或者尚未确定被染成什么颜色。对于第三种情况，我们记 $c_u = ?$ 。

Definition (自由度)

定义一个部分染色方案的自由度等于未染色的边的总个数 cur ，那么将该部分染色方案完善成完整的染色方案的方式共有 2^{cur} 种，且这 2^{cur} 种完整的染色方案导出的结果一定两两不同。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

7 / 36

点权的运输

固定一个染色方案，考虑交换点权的过程。

Definition (始发地和目的地)

对于同一个权值 x ，如果最初结点 u 拥有点权 x ，最终结点 v 拥有点权 x ，我们称权值 x 从 u 运输到了 v ，称 v 是 u 的目的地， u 是 v 的始发地，记 $\text{destination}(u) = v$ ， $\text{origin}(v) = u$ 。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

8 / 36

点权的运输

Definition (运输路线)

在交换点权的过程中， x 一定依次经过从 u 到 v 的树链上所有的结点各一次，我们称树链 $u \rightsquigarrow v$ 是权值 x 的运输路线。

设树链 $u \rightsquigarrow v$ 上的结点依次为 $h_1 \sim h_m$ 。拿出其中一个 h_i 来看：

- h_i 的 $k_{h_i} + 1$ 条邻边被删去的顺序为：
 $(h_i, \text{Fa}(h_i)), (h_i, \text{ch}(h_i, 1)), (h_i, \text{ch}(h_i, 2)), \dots, (h_i, \text{ch}(h_i, k_{h_i}))$ 。
- h_i 作为一个枢纽结点，起到中转权值的作用：在权值 x 首次到达结点 h_i 之后，找到 h_i 的邻边中第一条尚未被删去的黑边，那么 x 将被运往这条边的另一个端点。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

9 / 36

运输路线的正序描述

钦定 $\text{destination}(u) = v$ ，以下是 $\text{destination}(u) = v$ 对染色方案的限制：

- 所有 (h_i, h_{i+1}) 必须染黑；
- 对于 h_1 ， (h_1, h_2) 是 h_1 的邻边中第一条黑边；
- 对于 h_m ， (h_{m-1}, h_m) 是 h_m 的邻边中最后一条黑边；
- 对于中间的 h_i ， (h_{i-1}, h_i) 和 (h_i, h_{i+1}) 是 h_i 的邻边中相邻两条黑边，其中 (h_{i-1}, h_i) 在前， (h_i, h_{i+1}) 在后。

例外：当 $u = v$ 时要求 u 的所有邻边染白。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

10 / 36

运输路线的倒序描述

钦定 $\text{origin}(u) = v$ ，以下是 $\text{origin}(u) = v$ 对染色方案的限制：

- 所有 (h_i, h_{i+1}) 必须染黑；
- 对于 h_1 ， (h_1, h_2) 是 h_1 的邻边中最后一条黑边；
- 对于 h_m ， (h_{m-1}, h_m) 是 h_m 的邻边中第一条黑边；
- 对于中间的 h_i ， (h_{i-1}, h_i) 和 (h_i, h_{i+1}) 是 h_i 的邻边中相邻两条黑边，其中 (h_i, h_{i+1}) 在前， (h_{i-1}, h_i) 在后。

例外：当 $u = v$ 时要求 u 的所有邻边染白。

Theorem

钦定 $\text{origin}(u) = v$ 对染色方案的限制是两个集合 E_0, E_1 ，要求 E_0 中的边全部染白， E_1 中的边全部染黑。 E_0, E_1 的限制既是充分条件，也是必要条件。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

11 / 36

追溯过程的描述

固定一个染色方案，我们追溯出 $\text{origin}(u)$ 的值。
我们知道 (h_i, h_{i+1}) 是 h_i 的邻边中位于 (h_{i-1}, h_i) 之前的第一条黑边。
利用这一点，我们可以从 $h_1 = u$ 开始，顺藤摸瓜追溯出 $h_1 \sim h_m$ 。
注意到如果 h_{i+1} 是 h_i 的儿子，那么追溯到 h_{i+1} 就结束了！因此在 h_m 之前， h_{i+1} 一直是 h_i 的父亲。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

12 / 36

追溯过程的描述

经过整理，我们得到以下描述：

- 倒序访问 u 到其所有儿子的连边，然后访问 $(u, \text{Fa}(u))$ ；
- 倒序访问 $\text{Fa}(u)$ 到其所有位于 u 左侧儿子的连边，然后访问 $(\text{Fa}(u), \text{anc}(u, 2))$ ；
- 倒序访问 $\text{anc}(u, 2)$ 到其所有位于 $\text{Fa}(u)$ 左侧儿子的连边，然后访问 $(\text{anc}(u, 2), \text{anc}(u, 3))$ ；

以此类推。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

13 / 36

追溯过程的描述

追溯过程结束的条件有两个：

- ① 访问一个结点到其儿子的连边 (u, v) 时，如果 $c_v = 1$ ，则追溯过程在 v 结束；
- ② 访问一个结点到其父亲的连边 $(u, Fa(u))$ 时，如果 $c_u = 0$ ，则追溯过程在 u 结束。

称第一种边的预期颜色是白色，第二种边的预期颜色是黑色。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

14 / 36

追溯过程的描述

设将要访问的边依次为 $e_1 \sim e_m$ ，它们的预期颜色分别为 $c_{e_1}^* \sim c_{e_m}^*$ 。访问到第一条 $c_{e_i} \neq c_{e_i}^*$ 的边时，追溯过程结束， $\text{origin}(u)$ 是这条边靠下的端点。

由于 $(Rt, Fa(Rt))$ 一定是白边，不符合预期颜色黑色，故而追溯过程一定会结束。

Definition (追溯过程)

总结上述的讨论，我们定义针对 $\text{origin}(u)$ 的追溯过程是一个二元组序列 $(e_1, c_{e_1}^*) \sim (e_m, c_{e_m}^*)$ ，按照访问的顺序依次表示每条边靠下的端点以及每条边的预期颜色。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

15 / 36

逐步确定染色方案

固定 $p_u = x$ 意味着钦定 $\text{origin}(u) = b_x$ 。

我们维护一个部分染色方案，初始时所有 $c_u = ?$ 。随着对排列 p 的扫描，逐步确定染色方案。

我们需要提供以下接口：

- 操作 $\text{fix}(u, c)$ 表示将 $(u, \text{Fa}(u))$ 染成颜色 c ；
- 操作 $\text{assign}(u, x)$ 表示固定 $p_u = x$ ，即钦定 $\text{origin}(u) = b_x$ 。
 $\text{assign}(u, x)$ 可能与现有的部分染色方案矛盾。假如 $\text{assign}(u, x)$ 希望将确定的白边染黑，或者将确定的黑边染白，那么 $\text{assign}(u, x)$ 就是矛盾的。
对于没有矛盾的 $\text{assign}(u, x)$ ，记 $\text{reduce}(u, x)$ 表示 $\text{assign}(u, x)$ 将会确定的无色边个数，即自由度的减小量。
- 记 $\text{valid}(u)$ 表示当前执行 $\text{assign}(u, x)$ 不会导致矛盾的 x 构成的集合。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

16 / 36

第二小问的做法

首先求出 p 和 q 不相等的最高位是哪一位，我们希望这个最高位越靠后越好。

从 1 到 n 依次考虑每个结点。考虑到 u 的时候：

- ① 如果 $\max \text{valid}(u) > p_u$ ，说明 p_u 可以 $\neq q_u$ ；
- ② 执行 $\text{assign}(u, p_u)$ 。假如 $\text{assign}(u, p_u)$ 矛盾，立刻跳出循环。

记 λ 表示最大的 u 使得 p_u 可以 $\neq q_u$ （假如 λ 不存在则输出 -1 ）。重置所有 $c_u = ?$ ，再次从 1 到 n 依次考虑每个结点。考虑到 u 的时候：

- ① 对于 $u < \lambda$ ，直接执行 $\text{assign}(u, p_u)$ ；
- ② 对于 $u = \lambda$ ，求出 $\text{valid}(u)$ 中 $> p_u$ 的最小元素 x ，然后执行 $\text{assign}(u, x)$ ；
- ③ 对于 $u > \lambda$ ，记 $x = \min \text{valid}(u)$ ，执行 $\text{assign}(u, x)$ 。

唯一 $u = \lambda$ 的那一步可以直接暴力。因此后续数据结构优化需要解决的问题有三个：

- ① 执行 $\text{assign}(u, x)$ ；
- ② 求出 $x = \min \text{valid}(u)$ ；
- ③ 求出 $x = \max \text{valid}(u)$ 。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

17 / 36

第三小问的做法

维护当前自由度 cur 和累计答案 ans ，初始时 $cur = n - 1$ ， $ans = 0$ 。
从 1 到 n 依次考虑每个结点。考虑到 u 的时候：

- ① 对于 $valid(u)$ 中每个 $< p_u$ 的 x ，将 $2^{cur - reduce(u, x)}$ 计入答案；
- ② 执行 $assign(u, p_u)$ 。假如 $assign(u, p_u)$ 矛盾，立刻跳出循环。

对于第三小问，后续数据结构优化需要解决的问题有两个：

- ① 执行 $assign(u, x)$ ；
- ② 求出 $contribution(u) = \sum_{x \in valid(u)} [x < p_u] 2^{cur - reduce(u, x)}$ 。

上述做法的暴力实现

针对 $origin(u)$ 进行追溯。设访问到的边靠下的端点依次为 $e_1 \sim e_m$ ，它们的预期颜色分别为 $c_{e_1}^* \sim c_{e_m}^*$ 。

- 如果 $c_{e_i} \neq ?$ 并且 $c_{e_i} = c_{e_i}^*$ ，那么称 e_i 是绿色的；
- 如果 $c_{e_i} \neq ?$ 并且 $c_{e_i} \neq c_{e_i}^*$ ，那么称 e_i 是红色的；
- 如果 $c_{e_i} = ?$ ，那么称 e_i 是未染色的；

暴力找出 $e_1 \sim e_m$ 。如果追溯过程在 e_i 结束，说明 $origin(u) = e_i$ ，即 $a'_u = a_{e_i}$ 。

- $assign(u, x)$ 设 $a_{e_i} = x$ 。如果这样的 i 不存在则 $assign(u, x)$ 是矛盾的。否则将 e_i 染成红色，并将 $e_1 \sim e_{i-1}$ 染成绿色。
- $reduce(u, x)$ 等于 $e_1 \sim e_i$ 这 i 条边中未染色的个数。
- $valid(u)$ 找到第一条红色边 e_i 。那么 $valid(u)$ 等于 $e_1 \sim e_i$ 中所有未染色边构成的集合，再并上 e_i 。

这里染红一条边指的是通过 fix 操作将其染成不等于预期颜色的那种黑白颜色。染绿指的是染成预期颜色。

对于完全二叉树的特殊情形，时间复杂度 $O(n \log n)$ ，可以获得 30 分。

DFS 序 $1 \sim n$ 的情况

对于 DFS 序 $1 \sim n$ 的特殊情况，我们可以随着 $1 \sim n$ 的扫描，实时维护针对 $\text{origin}(u)$ 进行追溯的过程。

Definition (追溯过程的差异度)

我们认为针对 $\text{origin}(u)$ 的追溯过程是一个二元组序列 $(e_1, c_{e_1}^*) \sim (e_m, c_{e_m}^*)$ 。定义针对 $\text{origin}(u)$ 的追溯过程 A 和针对 $\text{origin}(u+1)$ 的追溯过程 B 的差异度 $\text{diff}(u, u+1)$ 等于这两个序列不完全一致的前缀长度之和，即 $|A| + |B| - 2|\text{LCS}(A, B)|$ 。给定两个栈 A, B ，通过弹出栈顶和压入栈中两种操作从 A 得到 B 所需要的最小操作次数等于 $\text{diff}(A, B)$ 。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

20 / 36

DFS 序 $1 \sim n$ 的情况

Theorem

$$\sum_{u=1}^{n-1} \text{diff}(u, u+1) = O(n).$$

Proof.

我们可以用下列三种操作按 DFS 序遍历一棵树：

- 下降操作：从 u 走到 $\text{ch}(u, 1)$ 。变化量是 $k_u + (k_{\text{ch}(u, 1)} + 1)$ 。
- 转换操作：从 $\text{ch}(u, i)$ 走到 $\text{ch}(u, i+1)$ 。变化量是 $(k_{\text{ch}(u, i)} + 1) + (k_{\text{ch}(u, i+1)} + 1)$ 。
- 上升操作：从 $\text{ch}(u, k_u)$ 走到 u 。变化量是 $(k_{\text{ch}(u, k_u)} + 1) + 1$ 。

由于每个结点的度数对总差异度之和的贡献次数 ≤ 3 ，因此总差异度之和是 $O(n)$ 的。 \square

直接使用线段树维护对 $\text{origin}(u)$ 进行追溯的过程，时间复杂度 $O(n \log n)$ ，可以获得 30 分，结合完全二叉树 50 分。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

21 / 36

定义与记号

记 $\text{top}(u)$ 表示 u 所在的重链顶端， $\text{bottom}(u)$ 表示 u 所在的重链底端。

Definition (二元组序列 $\text{circle}(u)$)

我们定义 $\text{circle}(u)$ 是一个长度为 $k_u + 1$ 的二元组序列。按照如下的过程：

- 倒序访问 u 到其所有儿子的连边，然后访问 $(u, \text{Fa}(u))$ ；

其中 u 到其所有儿子的连边的预期颜色都是白色，而 $(u, \text{Fa}(u))$ 的预期颜色是黑色。

定义与记号

Definition (二元组序列 $\text{trace}(u, v)$)

对于树上的一条从下到上的直链 $u \rightsquigarrow v$ ，我们定义 $\text{trace}(u, v)$ 是一个二元组序列。按照如下的过程：

- 倒序访问 $\text{Fa}(u)$ 到其所有位于 u 左侧儿子的连边，然后访问 $(\text{Fa}(u), \text{anc}(u, 2))$ ；
- 倒序访问 $\text{anc}(u, 2)$ 到其所有位于 $\text{Fa}(u)$ 左侧儿子的连边，然后访问 $(\text{anc}(u, 2), \text{anc}(u, 3))$ ；
- 以此类推，直到访问完 $(v, \text{Fa}(v))$ 为止。

在这个过程中，所有结点到其儿子的连边预期颜色都是白色，所有结点到其父亲的连边预期颜色都是黑色。

Definition (二元组序列 $\text{chain}(u)$)

最后我们对于每个重链顶端结点 u ，定义 $\text{chain}(u) = \text{trace}(\text{bottom}(u), u)$ 。

用重链剖分拆分追溯过程

针对 $\text{origin}(u)$ 的追溯过程可以拆成两个部分： $\text{circle}(u)$ 和 $\text{trace}(u, \text{Rt})$ 。我们知道 $u \rightsquigarrow \text{Rt}$ 的直链可以划分成 $\leq \log n$ 条轻边和重链前缀。按照重链剖分，我们把 $\text{trace}(u, \text{Rt})$ 拆分成 $\leq \log n$ 段 $\text{trace}(u, p)$ 和 $\text{trace}(p, q)$ 的并，其中 $p = \text{Fa}(u)$ ， $q = \text{top}(u)$ 。我们有：

- $\text{trace}(u, p)$ 一定是 $\text{circle}(p)$ 的一个后缀，并且所有边的预期颜色也和 $\text{circle}(p)$ 相一致；
- $\text{trace}(p, q)$ 一定是 $\text{chain}(q)$ 的一个后缀，并且所有边的预期颜色也和 $\text{chain}(q)$ 相一致；

我们称这样的后缀叫做重合子段，于是我们把针对 $\text{origin}(u)$ 的追溯过程拆成了 $O(\log n)$ 个重合子段。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

24 / 36

用数据结构维护重合子段

对于每个 $\text{circle}(u)$ 和每个重链顶端的 $\text{chain}(u)$ ，我们各开一个数据结构来维护。

由于每条边只会有效染色一次，我们可以暴力定位那些需要被染色的边进行染色。所有 n 个过程总共只会染色 $n - 1$ 条边。

以 $\text{trace}(p, q)$ 为例，假设 $\text{trace}(p, q)$ 的第一条边在 $\text{chain}(q)$ 中位于位置 l ，那么这个重合子段就是后缀 $[l, m]$ 。既然预期颜色一致，我们可以直接称呼边的红色和绿色，用一条边的红色和绿色代指这条边的黑色和白色。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

25 / 36

用数据结构维护重合子段

我们用数据结构维护重合子段：

- 我们开一个 `set` 维护所有无色边的位置，用于 `assign(u, x)` 定位所有无色边进行染色；
- 我们开一个 `set` 维护所有红色边的位置，用于寻找 l 之后第一条红色边的位置；
- 再开一棵线段树维护满足 $c_{e_i} = 1$ 或 $c_{e_i} = ?$ 之一的 a_{e_i} 的区间 `max`、区间 `min`，用于求出 `maxvalid(u)` 和 `minvalid(u)`。

处理一个重合子段的时间复杂度是 $O(\log n)$ 。由于重合子段的个数也是 $O(\log n)$ 的，单次操作的时间复杂度是 $O(\log^2 n)$ 。我们可以正确回答第二小问，时间复杂度 $O(n \log^2 n)$ ，可以获得 70 分。
使用全局平衡二叉树可以优化到 $O(n \log n)$ 。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

26 / 36

单个数据结构的任务

我们将单个数据结构的任务做一个总结：

- 维护两个序列 $a_{e_1} \sim a_{e_m}$ 、 $c_{e_1} \sim c_{e_m}$ ，其中 $a_{e_1} \sim a_{e_m}$ 的值不会变化，所有 c_{e_i} 初值为 $?$ 。
- 每次修改操作会将某个 c_{e_i} 变为 0/1。
- 每次查询操作一个后缀 $[l, m]$ 和一个权值 x ，查询
$$\sum_{i=l}^m [a_{e_i} < x] w'_i \prod_{j=l}^{i-1} w_j$$
。其中：
 - 对于 $c_{e_i} = ?$ ： $w_i = w'_i = \frac{1}{2}$ ；
 - 对于 $c_{e_i} = 0$ ： $w_i = 1$ ， $w'_i = 0$ ；
 - 对于 $c_{e_i} = 1$ ： $w_i = 0$ ， $w'_i = 1$ 。
- 另外还需要返回 $\prod_{j=l}^m w_j$ ，以供后续查询使用。

注意到单个数据结构的任务可以归约 $x < A$ 乘 $y < B$ 和问题，因此追求一个 $O(n\sqrt{n})$ 算法是明智的。

叶李蹊 (中国人民大学附属中学)

《世界沉睡童话》命题报告

January 14, 2024

27 / 36

静态情况

数点问题

静态情况指的是 c_{e_i} 具有 0/1/? 初值，但是不会变化的情况。
开一棵线段树维护序列 $a_{e_1} \sim a_{e_m}$ 。在每个线段树结点 u 上记一个 vector M_u 表示 $a_{e_{l_u}} \sim a_{e_{r_u}}$ 排序后的结果，那么 M_u 可以由 $M_{lch(u)}$ 和 $M_{rch(u)}$ 直接归并得到。
查询时在 $[l, m]$ 分解为的 $O(\log n)$ 个线段树结点上各自二分即可。
假如采用主席树的办法，这种办法没有希望扩展到动态情况。

带权问题

定义一个 i 在 $[l_u, r_u]$ 中的贡献是 $w_i' \prod_{j=l_u}^{i-1} w_j$ 。

那么再开一个与 M_u 等长的 vector F_u 表示 $[l_u, r_u]$ 中所有 $a_{e_i} < x$ 的 i 的贡献之和，再记录 $\prod_{i=l_u}^{r_u} w_i$ 的值，即可支持 pushup。

可以看出线段树的结构处理带权问题游刃有余。

动态情况

Lemma (重链剖分的性质)

对于任意一个结点 $u \rightsquigarrow Rt$ 的直链，我们可以把 $u \rightsquigarrow Rt$ 分解为 $k \leq \log n$ 段重链前缀。假设这 k 段重链的顶端按照从上到下的顺序依次为 z_1, z_2, \dots, z_k ，那么一定有 $\text{siz}(z_i) \leq \frac{n}{2^{i-1}}$ 。

根据这个性质我们可以得到，跳到从上往下数第 i 条重链的时候，所访问的两个数据结构所维护的序列长度必然分别 $\leq \frac{n}{2^{i-1}}$ 。因此为了得到一个 $O(n\sqrt{n})$ 做法，只需要能在 $O(\sqrt{n})$ 的时间复杂度内处理单个数据结构的修改查询即可。

AKEE 线段树

以下线段树结构的名字叫 AKEE 线段树¹：

- 记一个线段树结点的长度 $\text{len}(u) = r_u - l_u + 1$ ；
- 设置一个阈值 B ，只对 $\text{len}(u) \leq B$ 的线段树结点维护出 M_u 。
- 修改的时间消耗来源于不断向上 `pushup`。由于单次修改重构的区间长度和 $= B + \frac{B}{2} + \frac{B}{4} + \dots = 2B$ ，总复杂度 $O(B)$ 。
- 查询只在当前节点完全包含于查询区间且 $\text{len}(u) \leq B$ 时，在该点上相应查询。访问的结点分为两种：
 - 父亲已经完全包含于查询区间，但是父亲长度 $> B$ 暴力递归下来的结点。至多 $O(\frac{n}{B})$ 个。
 - 位于查询区间的两侧，父亲并不属于查询区间的结点。至多 $O(\log n)$ 个。

¹<https://www.cnblogs.com/dmoransky/p/14957063.html>

AKEE 线段树

我们把访问到的结点个数 $O(\frac{n}{B} + \log n)$ 乘上暴力二分的 $O(\log n)$ 得到在单个数据结构上查询的时间复杂度 $O(\frac{n \log n}{B} + \log^2 n)$ 。

在 $O(\log n)$ 个数据结构上查询的总时间复杂度 $O(\frac{n \log n}{B} + \log^3 n)$ 。取

$B = O(\sqrt{n \log n})$ 平衡得时间复杂度 $O(n \sqrt{n \log n})$ ，并不令人满意。

AKEE 线段树是很好实现的，只需要在 `pushup` 和 `query` 的时候加一个 `if` 条件就行。相比于直接分块，按照线段树结点维护的区间 `len` 根号分治并不会带来任何复杂度优化，这样的线段树结构唯一的作用在于简化代码。

常数正常的话可以获得 70 ~ 90 分。

分散层叠技术

最后我们用分散层叠技术优化 AKEE 线段树。

我们希望利用 x 在父亲结点 u 上二分的结果 $O(1)$ 确定出在两个子结点 $lch(u)$ 和 $rch(u)$ 上二分的结果。

- 对于长度 $\leq B$ 的结点，我们只需要保留归并的过程即可做到。具体来说我们在归并 M_u 的时候顺便记录两个 vector L_u, R_u ，表示归并到这个位置的时候左右子结点两个指针分别指到哪了，就可以 $O(1)$ 确定。
- 对于长度 $> B$ 的结点，我们扩展定义 M_u 表示 $M_{lch(u)}$ 和 $M_{rch(u)}$ 各取 $1/3$ 的元素归并起来的结果。我们将 $M_{lch(u)}$ 和 $M_{rch(u)}$ 分别每隔 3 个元素取一个元素，然后归并起来。这样只需要暴力往后找 ≤ 2 个元素就可以定位到，仍然是 $O(1)$ 的。

于是只需要在线段树的根处做一次二分即可。

分散层叠技术

- 对于 $\text{len} > B$ 的部分，单次修改重构的区间长度和 $= B + \frac{2B}{3} + \frac{4B}{9} + \dots = 3B$ ，时间复杂度 $O(B)$ 。
- 查询操作递归到长度 $\leq B$ 的结点时应该已经知道了二分位置，单个复杂度 $O(\frac{n}{B} + \log n)$ ，一次询问总计 $O(\frac{n}{B} + \log^2 n)$ 。

取 $B = O(\sqrt{n})$ 平衡得时间复杂度 $O(n\sqrt{n})$ ，可以获得满分 100 分。

随着对排列 p 的扫描，上述做法在线统计出了每个 i 的贡献（一个 i 的贡献指的是满足 $q_i < p_i$ 且 $\forall j < i$ 都有 $q_j = p_j$ 的合法排列 q 的个数）。

同时也有一种离线的办法可以避免分散层叠。既然已经知道了完整的排列 p ，我们可以把整个统计过程中所有的染色操作和查询操作全部离线下来。这样一来，每个数据结构只需要用相对简单的序列分块实现即可。

关于积性函数求和的一些探讨

学军中学 周康阳

前置知识	狄利克雷卷积	积性函数求和
●○○○○○○○○○○○○○○○○	○○○○○○○○	○○○○○○○○○○○○
Contents		

1 前置知识

2 狄利克雷卷积

3 积性函数求和

周康阳

关于积性函数求和的一些探讨

学军中学

前置知识	狄利克雷卷积	积性函数求和
○●○○○○○○○○○○○○○○	○○○○○○○○	○○○○○○○○○○○○
定义与约定		

Definition

数论函数是定义域为正整数集，陪域为复数域的函数。

对于一个数论函数 f ，我们约定 $S_f(n) = \sum_{i=1}^n f(i)$ 。

Definition

积性函数是指一个定义域为正整数 n 的数论函数 $f(n)$ ，且满足如下性质： $f(1) = 1$ ，且当 a 和 b 互质时， $f(ab) = f(a)f(b)$ 。

对于一个积性函数 f ，只要知道了对于所有 $p \in \mathbb{P}, k \in \mathbb{Z}^+$ 的 $f(p^k)$ ，我们就可以知道整个 f 函数。

周康阳

关于积性函数求和的一些探讨

学军中学

前置知识
○○●○○○○○○○○○○○○○○
块级卷积
○○○○○○○○
积性函数求和
○○○○○○○○○○○○○○○○

定义与约定

一些常见的积性函数在 $p \in \mathbb{P}, k \in \mathbb{Z}^+$ 的取值如下：

- ϵ 函数（单位函数），满足 $\epsilon(p^k) = 0$ 。
- μ 函数，满足 $\mu(p^k) = -[k = 1]$ 。
- φ 函数，满足 $\varphi(p^k) = (p-1)p^{k-1}$ 。
- I 函数，满足 $I(p^k) = 1$ 。
- id_t 函数，满足 $id_t(p^k) = p^{kt}$ 。

周康阳

关于积性函数求和的一些探讨

学军中学

前置知识
○○●○○○○○○○○○○○○○○
块级卷积
○○○○○○○○
积性函数求和
○○○○○○○○○○○○○○○○

Dirichlet 卷积

Definition

对于数论函数 f 和 g ，定义其 *Dirichlet* 卷积

$$(f * g)(x) = \sum_{d|x} f(d)g\left(\frac{x}{d}\right)$$

如果将 *Dirichlet* 卷积视为数论函数的乘法，函数的直接加和为数论函数的加法。那么数论函数的加法和乘法是满足交换律、结合律和分配律的。

周康阳

关于积性函数求和的一些探讨

学军中学

前置知识
○○○○●○○○○○○○○

块级卷积
○○○○○○○

积性函数求和
○○○○○○○○○○

Powerful Number

Definition

Powerful Number 是指一个正整数 m ，满足对于任意 m 的质因数 p ，均有 $p^2 \mid m$ 。

接下来，我们用 PN 简称 Powerful Number。在公式中，用 PN 代表所有 Powerful Number 构成的集合。

Theorem

对于所有 PN ，我们都可以将其表示成 a^2b^3 的形式（其中 a 和 b 都是正整数）。

对于一个 PN 数 $m = \prod_{i=1}^k p_i^{\alpha_i}$ ，对每个 $p_i^{\alpha_i}$ 都构造出 a^2b^3 的即可。这个只要把 α_i 拆成 $2x+3y$ 的形式就行了。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识
○○○○●○○○○○○○○

块级卷积
○○○○○○○

积性函数求和
○○○○○○○○○○

Powerful Number

Theorem

n 以内的 PN 个数为 $O(\sqrt{n})$ 。

Proof.

把所有 n 以内的 PN 表示成 a^2b^3 的形式。枚举 a ，我们就可以得到 PN 数不会超过 $\sum_{a=1}^{\sqrt{n}} (\frac{n}{a^2})^{1/3}$ 。这个可以估计为

$$\begin{aligned} \int_1^{\sqrt{n}} (\frac{n}{x^2})^{1/3} dx &= n^{1/3} \int_1^{\sqrt{n}} x^{-2/3} dx \\ &= n^{1/3} (3n^{1/6} - 3) \end{aligned}$$

即 $O(\sqrt{n})$ 级别。 □

可以通过深搜在 $O(\sqrt{n})$ 内找出所有 n 以内的 PN 。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识
○○○○○○●○○○○○○

快速卷积
○○○○○○○

积性函数求和
○○○○○○○○○○

Powerful Number

下面看一道例题。

Problem

(UOJ Long Round 1 校验码的 $n=1$ 部分分) 给定 c 。积性函数函数 $q(x)$ 满足 $q(p^k) = p^{c\lfloor \frac{k}{2} \rfloor}$ 。求 $\sum_{i=1}^m q(i)$ ，答案对 2^{32} 取模。
 $n \leq 1.2 \times 10^{11}$

周康阳

关于积性函数求和的一些探讨

学军中学

前置知识
○○○○○○○●○○○○○○

快速卷积
○○○○○○○

积性函数求和
○○○○○○○○○○

Powerful Number

设计积性函数 $f(x)$ 满足 $f(p) = q(p)$ ，并构造积性函数 g 满足 $g * f = q$ 。

对于素数 p ，有 $g(p)f(1) + g(1)f(p) = q(p)$ ，解得 $g(p) = 0$ 。因此 g 函数只在 PN 处有值！

$$\begin{aligned} \sum_{i=1}^m q(i) &= \sum_{i=1}^m \sum_{j|i} g(j)f(k) \\ &= \sum_{j \in \text{PN}, j \leq m} g(j) \sum_{k=1}^{\lfloor \frac{m}{j} \rfloor} f(k) \\ &= \sum_{j \in \text{PN}, j \leq m} g(j) \lfloor \frac{m}{j} \rfloor \end{aligned}$$

周康阳

关于积性函数求和的一些探讨

学军中学

前置知识
○○○○○○○○●○○○○○
块筛卷积
○○○○○○○
积性函数求和
○○○○○○○○○○○

Powerful Number

对于一个质数 p ，我们有 $h(p^k) = \sum_{i=0}^k f(p^i)g(p^{k-i})$ 。
 因此 $g(p^k) = h(p^k) - \sum_{i=1}^k f(p^i)g(p^{k-i})$ ，我们可以按 k 从小到大的顺序解出 $g(p^k)$ 。
 在 DFS 找 PN 的同时维护 g 函数值。这样，这个问题就可以 $O(\sqrt{m})$ 解决了。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识
○○○○○○○○○○●○○○
块筛卷积
○○○○○○○
积性函数求和
○○○○○○○○○○○

杜教筛

对于一些特殊的积性函数（如 μ 和 ϕ ），我们可以通过构造 Dirichlet 卷积在 $O(n^{\frac{2}{3}})$ 的时间复杂度计算其前缀和。这种算法在国内信息学竞赛界被称为“杜教筛”。

对于 μ 函数，我们有 $\mu * I = \epsilon$ 。

不妨考虑更一般的问题。如果有数论函数 A, B, C 满足 $A(1) = B(1) = C(1) = 1$ 且 $A * B = C$ ，并且我们能快求出 S_B 和 S_C 的点值，我们该如何求出 $S_A(n)$ 呢？

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识
○○○○○○○○○○●○○○

块筛卷积
○○○○○○○

积性函数求和
○○○○○○○○○○○

杜教筛

注意到

$$\begin{aligned}\sum_{i=1}^n C(i) &= \sum_{i=1}^n \sum_{j|k=i} A(j) B(k) \\ &= \sum_{k=1}^n B(k) \sum_{j|k} A(j) \\ &= S_A(n) + \sum_{k=2}^n B(k) S_A\left(\left\lfloor \frac{n}{k} \right\rfloor\right)\end{aligned}$$

因此 $S_A(n) = S_C(n) - \sum_{k=2}^n B(k) S_A\left(\left\lfloor \frac{n}{k} \right\rfloor\right)$, 可以递归求解 $S_A(n)$ 。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识
○○○○○○○○○○●○○○

块筛卷积
○○○○○○○

积性函数求和
○○○○○○○○○○○

杜教筛

注意到 $\left\lfloor \frac{\frac{n}{a}}{b} \right\rfloor = \left\lfloor \frac{n}{ab} \right\rfloor$, 在递归过程中, 需要被计算的实际上只有 $S_A\left(\left\lfloor \frac{n}{k} \right\rfloor\right)$ (k 是正整数)。

Theorem
对于正整数 n , 集合 $\{\left\lfloor \frac{n}{i} \right\rfloor | i \in \mathbb{Z}^+\}$ 的大小是 $\mathcal{O}(\sqrt{n})$ 的。

Proof.
对于 $k \leq \sqrt{n}$, 只有 \sqrt{n} 个 $\left\lfloor \frac{n}{k} \right\rfloor$, 显然只有 $\mathcal{O}(\sqrt{n})$ 种;
对于 $k > \sqrt{n}$, $\left\lfloor \frac{n}{k} \right\rfloor \leq \frac{n}{\sqrt{n}} = \sqrt{n}$, 因此也只有 $\mathcal{O}(\sqrt{n})$ 种。 \square

因此, 我们只需要计算 S_A 的 $\mathcal{O}(\sqrt{n})$ 个点值。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识
○○○○○○○○○○●○○

块筛卷积
○○○○○○○

积性函数求和
○○○○○○○○○○○

杜教筛

在计算单个 $S_A(m)$ 时，将连续的相同的 $\lfloor \frac{m}{k} \rfloor$ 一起计算（即整除分块），就能在 $\mathcal{O}(\sqrt{m})$ 的复杂度内递归到子问题了。

对于 $m \leq n^{\frac{2}{3}}$ 的 $S_A(m)$ ，我们通过提前预处理出 A 的前 $n^{\frac{2}{3}}$ 个点值来解决。对于 $m > n^{\frac{2}{3}}$ ，使用上面的递归式计算，这部分的时间复杂度 $\mathcal{O}(\sum_{i=1}^{\frac{1}{n^{\frac{2}{3}}}} \sqrt{\frac{n}{i}})$ 。这个可以估算为 $\mathcal{O}(\int_1^{\frac{1}{n^{\frac{2}{3}}}} \sqrt{\frac{n}{x}} dx) = \mathcal{O}(n^{\frac{2}{3}})$ 。总复杂度即为 $\mathcal{O}(n^{\frac{2}{3}})$ 加上预处理出 $\mathcal{O}(n^{\frac{2}{3}})$ 预处理出 A 的前 $n^{\frac{2}{3}}$ 个点值的复杂度（ μ 和 φ 都是可以 $\mathcal{O}(n^{\frac{2}{3}})$ 预处理的）。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识
○○○○○○○○○○●○○

块筛卷积
○○○○○○○

积性函数求和
○○○○○○○○○○○

杜教筛

Definition

定义数论函数 f 关于 n 的块筛为 $S_f(n)$ 在 $\{\lfloor \frac{n}{i} \rfloor | i \in \mathbb{Z}^+\}$ 的点值。

在上述过程当中，我们只用了 B 和 C 关于 n 的块筛，就求出了 A 关于 n 的块筛。

周康阳

学军中学

关于积性函数求和的一些探讨



前置知识

2 块筛卷积

3 积性函数求和



对于数论函数 A, B, C 满足 $A(1) = B(1) = C(1) = 1$ 且 $A * B = C$ ，上文的杜教筛根据 B 和 C 关于 n 的块筛解出了 A 关于 n 的块筛。

在这里，我们先考虑一个更加简单的问题：已知数论函数 A 和 B 关于 n 的块筛，要求求出满足 $C = A * B$ 关于 n 的块筛。我们将 C 称为 A 和 B 的块筛卷积。块筛上的运算仍然具有交换律，结合律和分配律。

这个问题实际上是可以 $O(n^{\frac{1}{2}} \text{poly} \log n)$ 解决的。



前置知识
○○○○○○○○○○○○○○○○

块筛卷积
○○●○○○

积性函数求和
○○○○○○○○○○

块筛卷积

下文中，我们默认块筛是关于 n 的块筛。
我们要解决的是 $C(z) = \sum_{xy=z} A(x)B(y)$ 在所有 $\lfloor \frac{n}{t} \rfloor$ 上的前缀和。
首先考虑 $x > \sqrt{n}$ 的情况 ($y > \sqrt{n}$ 是对称的)。如果 $xy \leq \lfloor \frac{n}{t} \rfloor$ ，那么 $x \leq \frac{n}{ty}$ 。所以枚举 t 和 y ，有贡献的 x 是一段前缀。这部分的复杂度是 $\mathcal{O}(\sqrt{n} \log n)$ 。
这样我们就只需要解决 $x \leq \sqrt{n}$ 且 $y \leq \sqrt{n}$ 的情况了。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识
○○○○○○○○○○○○○○○○

块筛卷积
○○●○○○

积性函数求和
○○○○○○○○○○

块筛卷积

$xy \leq \frac{n}{t}$ 等价于 $\ln(x) + \ln(y) \leq \ln(\frac{n}{t})$ 。
我们做一个估计：选取一个块长 $S(3 \leq S \leq n)$ ，我们认为 $\ln(x) + \ln(y) \leq \ln(\frac{n}{t})$ 当且仅当 $\lceil S \ln(x) \rceil + \lceil S \ln(y) \rceil \leq S \ln(\frac{n}{t})$ 。
这个估计可以用多项式乘法完成。设计多项式 $F(z), G(z)$ ，其中 $[z^k]F(z)$ 为满足 $\lceil S \ln(x) \rceil = k$ 的 $A(x)$ 的和， $[z^k]G(z)$ 为满足 $\lceil S \ln(y) \rceil = k$ 的 $B(y)$ 的和。计算多项式 $H(z) = F(z)G(z)$ ，则 $[z^k]H(z)$ 就是 $\lceil S \ln(x) \rceil + \lceil S \ln(y) \rceil = k$ 的 $A(x)B(y)$ 和。使用一次长度为 $S \ln(n)$ 的卷积即可。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识
○○○○○○○○○○○○○○○○

块筛卷积
○○○○●○○

积性函数求和
○○○○○○○○○○○○

块筛卷积

但是这样做显然是不对的。观察我们在什么情况下会估计错：只有 $\ln(x) + \ln(y) \leq \ln(\frac{n}{t})$ 且 $\lceil S\ln(x) \rceil + \lceil S\ln(y) \rceil > S\ln(\frac{n}{t})$ 时才会估错。此时有 $S\ln(x) + S\ln(y) \in (S\ln(\frac{n}{t}) - 2, S\ln(\frac{n}{t}))$ 。注意到这个时候 x, y, t 必须满足 $\ln(x) + \ln(y) + \ln(t) \in (\ln(n) - \frac{2}{S}, \ln(n)]$ ，即 $xyt \in (ne^{-\frac{2}{S}}, n]$ 。其中 $e^{-\frac{2}{S}}$ 是 $1 - \mathcal{O}(\frac{1}{S})$ 级别的。因此， xyt 是在一个长度为 $\mathcal{O}(\frac{n}{S})$ 的区间内的！

周康阳

关于积性函数求和的一些探讨

学军中学

前置知识
○○○○○○○○○○○○○○○○

块筛卷积
○○○○●○○

积性函数求和
○○○○○○○○○○○○

块筛卷积

不妨假设这个区间是 $[n-L, n]$ 。我们使用区间筛得到这个区间内所有数的质因数分解。更具体地说，先筛出所有不超过 \sqrt{n} 的小质数，对于每个质数，我们都可以求得区间 $[n-L, n]$ 中被它整除的数。这样我们就筛出了 $[n-L, n]$ 的所有数的 $\leq \sqrt{n}$ 的质因子。而 $> \sqrt{n}$ 的质因子最多只有一个，所以只要看这个数除掉所有小质数后的值就行了。得到质因数分解后，我们就可以通过在质因子上 DFS 快速找出所有可能的 (x, y, t) 了，对于一个 (x, y, t) 可以 $\mathcal{O}(1)$ 检查它是否被正确地估计了。对于一个 $v \in [n-L, n]$ ，它贡献时间复杂度是 $d_3(v)$ （其中 $d_3(v) = \sum_{xyz=v} 1$ ）。

周康阳

关于积性函数求和的一些探讨

学军中学

前置知识
○○○○○○○○○○○○○○○○

块筛卷积
○○○○○○●

积性函数求和
○○○○○○○○○○○○

块筛卷积

而根据解析数论中的结论¹，
 $\sum_{i \leq n} d_3(i) = nP_3(\ln n) + O(n^{43/96+\epsilon})$ 。所以
 $\sum_{i=n-L}^n d_3(i) = LP_3(\ln n) + O(n^{43/96+\epsilon})$ ，其中 P_3 是二次多项式。算法的输出已经有 $O(\sqrt{n})$ ，比 $O(n^{43/96+\epsilon})$ 大，所以可以忽略这部分的复杂度。剩下部分的复杂度就是 $O(L \log^2 n)$ ，即 $O(\frac{n}{S} \log^2 n)$ 。
 这么做的总复杂度是 $O(S \log^2 n + \frac{n}{S} \log^2 n)$ ，取 $S = \sqrt{n}$ 即可得到 $O(\sqrt{n} \log^2 n)$ 的时间复杂度。

¹https://en.wikipedia.org/wiki/Divisor_summatory_function

周康阳

关于积性函数求和的一些探讨

学军中学

前置知识
○○○○○○○○○○○○○○○○

块筛卷积
○○○○○○○

积性函数求和
●○○○○○○○○○○

Contents

1 前置知识

2 块筛卷积

3 积性函数求和

周康阳

关于积性函数求和的一些探讨

学军中学

前置知识
○○○○○○○○○○○○○○○○

块筛卷积
○○○○○○○○

积性函数求和
○○●○○○○○○○○

积性函数求和

对于一个满足一些特殊性质（比如满足 $f(p^k) = p^k(p^k - 1)$ 的积性函数）的积性函数 f ，该如何求 f 的块筛呢？

首先， $f(p^k) (k \geq 2)$ 的值并不重要。如果另一个积性函数 g 满足 $f(p) = g(p)$ ，且 g 的前缀和容易计算，那么我们只需要设计满足 $h * g = f$ 的积性函数 h ，根据前文的结论， h 只在 PN 处有值且这些位置上的值可以 $O(\sqrt{n})$ 算出。因此容易 $O(\sqrt{n})$ 算出 h 的块筛，然后通过 h 和 g 的块筛卷积算出 f 的块筛。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识
○○○○○○○○○○○○○○○○

块筛卷积
○○○○○○○○

积性函数求和
○○●○○○○○○○○

积性函数求和

我们先设计数论函数 q 满足 q 只在质数处的值非零，且 $q(p) = f(p)$ 。

定义一个数论函数 q ($q(1) = 0$) 的 $\exp(q)$ 为 $\sum_{i=0}^{\infty} \frac{q^i}{i!}$ ，其中卷积是块筛卷积。观察 $d = \exp(q)$ 。对于一个数 $x = \prod_{i=1}^k p_i^{\alpha_i}$ ，

$d(x)$ 只有在 $q^{\sum_{i=1}^k \alpha_i}$ 项上有值，且值为

$$\frac{\prod_{i=1}^k \frac{q(p_i)^{\alpha_i}}{(\sum_{i=1}^k \alpha_i)!}}{\prod_{i=1}^k \frac{q(p_i)^{\alpha_i}}{\alpha_i!}} = \frac{\prod_{i=1}^k q(p_i)^{\alpha_i}}{\prod_{i=1}^k \alpha_i!}。$$

因此， d 函数是一个积性函数，且 $d(p^k) = \frac{f(p)^k}{k!}$ ，满足 $d(p) = f(p)$ 。这样，如果我们能求出 q 的块筛，我们就可以通过 $\log n$ 次（在 $i > \log n$ 时 q 函数的前 n 项没有值）块筛卷积得到 d 函数的块筛，从而得到 f 的块筛。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识
○○○○○○○○○○○○○○○○

块筛卷积
○○○○○○○

积性函数求和
○○○●○○○○○

积性函数求和

怎么得到 q 的块筛呢？考虑一个类似 Min_25 筛的想法。将 q 函数拆成若干个 q_i 的带权和，并通过这些 q_i 的块筛来算出 q 的块筛。比如 $q(p) = p(p-1)$ ，我们就可以拆成 $q_1(p) = p^2$ ， $q_2(p) = p$ ，然后就有 $q = q_1 - q_2$ 。
当 $q(p)$ 是关于 p 的常数次多项式时，我们都可以按照这种方法拆函数。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识
○○○○○○○○○○○○○○○○

块筛卷积
○○○○○○○

积性函数求和
○○○●○○○○○

积性函数求和

如何求出 q_i 的块筛呢？这里我们反过来。
以求 $q_1(p) = p^2$ 为例。对于一个满足 $d(p^k) = q_1(p)^k$ 的积性函数 d ，它满足 $d(x) = x^2$ ，块筛很好算；而对于满足 $d(p^k) = \frac{q_1(p)^k}{k!}$ 的积性函数 d ，由于它和上个函数在 p 处的点值相同，所以也可以用 PN 在一次块筛卷积的复杂度内相互转化，所以它的块筛也是可以算出的。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识
○○○○○○○○○○○○○○○○

块状卷积
○○○○○○○○

积性函数求和
○○○○○○●○○○○

积性函数求和

而满足 $d(p^k) = \frac{q_1(p)^k}{k!}$ 的函数 d ，正是 $\exp(q_1)$ 。这启发我们用一
个类似 \ln 的思路来从 d 反推出 q_1 。

定义数论函数 d ($d(1) = 1$) 的 $\ln(d)$ 为 $\sum_{i=1} \frac{(-1)^{i-1} (d - \epsilon)^i}{i}$ 。
我们有 $q_1 = \ln(d)$ ，证明如下。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识
○○○○○○○○○○○○○○○○

块状卷积
○○○○○○○○

积性函数求和
○○○○○○●○○○○

积性函数求和

我们只要分别说明 $\exp(\ln(d)) = d$ 和满足 $\exp(q) = d$ 的 q 是唯
一的即可。

如果满足 $\exp(q) = d$ ，那么 $\epsilon + q + \sum_{i \geq 2} \frac{q^i}{i!} = d$ 。我们可以从小
到大依次确定 $q(i)$ 的值：设函数 $p = \sum_{i \geq 2} \frac{q^i}{i!}$ ，则 $p(i)$ 只和
 $q(j) (j < i)$ 有关，所以可以用 $q(i) = d(i) - [i = 1] - p(i)$ 解出
 $q(i)$ 。因此 q 函数是可以被唯一确定的。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识

块筛卷积

积性函数求和

积性函数求和

然后说明 $\exp(\ln(d)) = d$ 。

$$\exp(\ln(d)) = \sum_{j=0}^{\infty} \frac{(\sum_{i=1}^{\infty} \frac{(-1)^{i-1}(d-e)^i}{i})^j}{j!}$$

使用块筛的分配律和结合律将其展开，我们将得到一个 $\sum_i a_i d^i$ 形式的式子。不妨用多项式 $A(z)$ 来刻画它，其中 $[z^k]A(z) = a_k$ 。而 \exp 和 \ln 在块筛卷积中的定义和多项式相同（即 $\exp(z) = \sum_{i=0}^{\infty} \frac{z^i}{i!}$, $\ln(1+z) = \sum_{i=1}^{\infty} \frac{(-1)^{i-1} z^i}{i}$ ）， $A(z) = \exp(\ln(z)) = z$ ，上面的 $a_i = [i=1]$ ，因此 $\exp(\ln(d)) = d$ 。

周康阳

学军中学

关于积性函数求和的一些探讨

前置知识

块筛卷积

积性函数求和

积性函数求和

上述的 \ln 中也只需要枚举 $i \leq \log n$ ，所以也只需要 $\log n$ 次块筛卷积。

因此，该算法的时间复杂度是 $\mathcal{O}(\sqrt{n} \log^3 n)$ 。

周康阳

学军中学

关于积性函数求和的一些探讨



直接实现这个算法实际上是完全跑不动的...
在我的集训队论文中，还提到了对这个算法的一些优化。
目前的时间复杂度是 $O(\frac{\sqrt{n \log^2 n}}{\sqrt{\log \log n}})$ ，而且很可能可以被进一步优化。欢迎大家来和我探讨这个问题。





浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

前言

树上问题作为一类经典的问题，在信息学竞赛中经常出现，而树上范围相关问题是其中比较重要且具有广泛运用的一类问题。笔者对树链剖分，点分树，基础 Top Tree 进行研究，详细分析其与维护树上范围相关的一些性质，并结合例题进行总结。本文中在通常情况下定义树的大小为 $O(n)$ 级别。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

树链剖分

重链剖分是树链剖分的一种类型，在处理树上范围相关问题中，重链剖分有着重要的作用。我们默认读者已经掌握重链剖分相关的若干定义和性质。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

重链剖分的一种理解：树上链分治

我们尝试从类似点分治的角度来理解重链剖分——树上链分治，点分治可以将树上的一条路径唯一对应到点分治中经过一个分治重心的两条路径，我们希望用类似的方式将树上的一条路径唯一对应到重链剖分上的一条重链区间上，将一条路径的唯一对应到的重链区间设置为路径中深度最小的节点的所在重链与路径的交即可。

点分树可以处理一些涉及单点范围信息的维护，因此我们不妨尝试使用树链剖分来处理一些涉及路径范围信息的维护。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

例题

给定一个 n 个节点的树，树边权均为 1，每个节点有点权，初始点权均为 0，有 m 次操作，需要支持两种操作：

1. 给定路径端点 u, v ，范围 d 和权值 w ，将所有满足与路径 (u, v) 上的任意节点的最短距离 $\leq d$ 的节点点权增加 w 。
2. 给定单点 u ，查询点 u 的权值。

数据范围： $1 \leq n, m \leq 10^5$ 。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

重链剖分的一种理解：树上链分治

在点分树中，对于当前分治层上分治重心的标记，是作用在以分治重心为根的子树上的；因此考虑在树链剖分结构上进行类似的标记，即节点 x 的标记作用在节点 x 的所有轻儿子的子树中，修改时分析路径在重链剖分上的标记位置及其影响，查询时对单点本身的标记和单点的祖先上所有轻边对应的父亲节点的标记进行查询。

先考虑直接维护贡献，对原树进行重链剖分，对于一次修改操作，将路径 (u, v) 经过的所有重链区间取出，并对每一个重链区间进行标记，形如对于一个在重链区间内的点 x ，其所有轻儿子的子树内与点 x 距离在范围 d 内的所有节点点权增加 w ，注意受点 x 的标记影响的节点为点 x 及其轻儿子子树内的所有节点。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

重链剖分的一种理解：树上链分治

但注意到这样会有三个部分的贡献需要处理：

第一个部分是重链区间之后的部分的贡献，形如与重链区间右端点距离在范围 d 内的右端点重子树内的所有节点点权增加 w ；

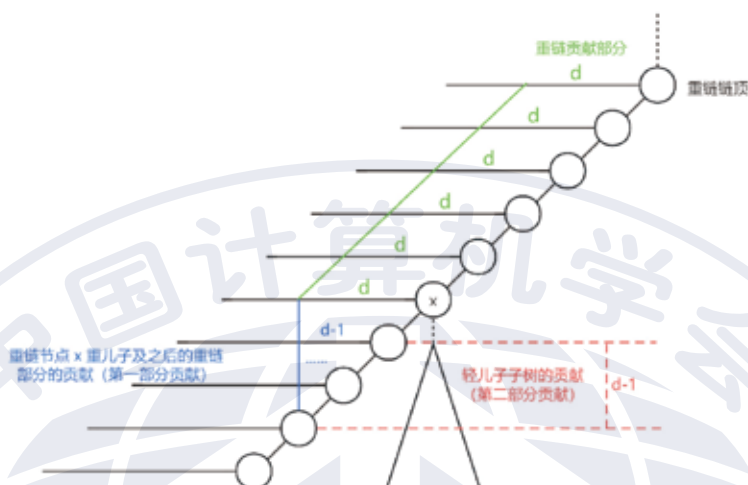
第二个部分是通过轻边跳到一条新的重链时，对重链区间上轻儿子子树的操作会影响到跳上来时的轻儿子子树，原来的轻儿子的贡献需要去掉重复计算部分；

第三个部分是跳到路径深度最小的节点之后其上部分需要添加贡献。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

重链剖分的一种理解：树上链分治



浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

重链剖分的一种理解：树上链分治

考虑一条重链，对于一个节点的标记为 d ，可以发现：直接维护贡献部分的标记形如重链区间上一段标记为 d ；第一个部分设重链区间右端点为 r ，形如重链以 r 为端点的一段后缀上的点 x 的标记为 $d - \text{dist}(r, x)$ ；第三个部分设路径深度最小的节点为 lca ，形如重链以 lca 到该重链的轻边父节点 y 为端点的一段前缀上的点 x 的标记为 $d - \text{dist}(lca, y) - \text{dist}(x, y)$ 。对每条重链维护区间所有相同标记信息，区间左端点向右标记递减标记信息，区间右端点向左标记递减标记信息。

第二个部分注意到重复计算的部分恰好为一个重链的顶部节点的范围为 $d - 1$ 内的节点，在每条重链顶部维护标记。

查询单点时，对单点本身标记和单点祖先上所有轻边对应父亲节点的标记进行查询，对每条重链顶部的标记进行查询即可。

使用线段树维护每条重链的标记，时间复杂度 $O(n \log^3 n)$ ，使用全局平衡二叉树维护每条重链的标记，时间复杂度 $O(n \log^2 n)$ 。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

基于调整搜索顺序的重标号树链剖分

树链剖分的性质非常丰富且非常广泛，在一些题目中我们需要维护一些复杂的信息，树链剖分的其中一些性质非常适合维护这类信息，但受到其结构的影响会导致一些信息不太方便很好的合并，此时可以考虑通过改变其结构来保留其中一些有用的性质并更好的维护信息。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

例题：数据结构

小 Z 有一棵 n 个节点构成的树，这棵树以点 1 为根，边权均为 1，每个节点 x 有点权 a_x ，初始所有节点的权值为 0。小 Z 想对这棵树进行总共 m 次操作，并且小 Z 希望能够支持以下四种类型的操作：

1. 给定 x, y, k, v ，对满足节点 i 到树上唯一的最短路径 (x, y) 的最短距离 $\leq k$ 的所有节点 i 的权值增加 v ，即 $a_i \leftarrow a_i + v$ 。
2. 给定 x, v ，将点 x 的子树内的所有节点 i 的权值增加 v ，即 $a_i \leftarrow a_i + v$ 。
3. 给定 x, y, k ，查询所有满足节点 i 到树上唯一的最短路径 (x, y) 的最短距离 $\leq k$ 的所有节点 i 的权值的和。
4. 给定 x ，查询点 x 的子树内的所有节点 i 的权值和。

由于答案会比较大，输出对 2^{64} 取模即可。数据范围：
 $1 \leq n, m \leq 2 \times 10^5$ ， $1 \leq x, y \leq n$ ， $1 \leq v \leq 10^9$ ， $0 \leq k \leq 3$ 。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

基于调整搜索顺序的重标号树链剖分

考虑到当 $k = 0$ 时，我们可以依据重链剖分的性质，由于路径至多跳 $\log n$ 条轻边，因此在 $\log n$ 条重链上进行处理，而重链的标号具有连续性，因此可以支持快速对点进行修改和查询，这启发我们尝试在保持重链剖分考虑 $\log n$ 条重链进行处理的性质的基础上，通过稍微调整重链的标号，使得重链上与重链上节点距离较小的节点标号大致连续，方便快速修改。

笔者在下文中将这种在保持树链剖分部分性质的基础上，通过调整标号顺序便于维护信息的方式称为“重标号树链剖分”，这里介绍本题的一种标号方式：

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

基于调整搜索顺序的重标号树链剖分

按照重链中深度最小的点的 dfs 顺序来处理每一条重链，考虑对于一条重链，先将这条重链上的所有点按照深度从小到大排序进行标号，然后依次枚举距离 d 为 1 和 2 和 3，每次将这条重链上的所有点按照深度从小到大处理每个点，处理到点 x 时将点 x 的所有轻儿子子树中的与点 x 距离 $= d$ 的所有点进行标号。注意到有可能处理一条重链时与重链顶部节点的距离 ≤ 3 的节点已经被标号，此时已经被标号的点不会再进行标号。

可以发现这样的结构保留了树链剖分的许多性质，比如保证了至多在 $\log n$ 条重链上进行处理，并且这样的标号方式使得树上路径的小范围信息具有了一些连续性。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

基于调整搜索顺序的重标号树链剖分



图：进行上述标号方式后，该树的一种合法的标号

基于调整搜索顺序的重标号树链剖分

对于每一个节点 x ，考虑维护线段集合 $s_{0,x,d}$ 表示与点 x 距离 $= d$ 的在点 x 的子树内的所有节点编号区间（当 $d = 4$ 时描述的是距离 > 4 的点），线段集合 $s_{1,x,d}$ 表示与点 x 距离 $= d$ 的在点 x 的轻儿子的子树内的所有节点编号区间（当 $d = 4$ 时描述的是距离 > 4 的点），可以发现转移是简单的。

根据标号的连续性，可以发现 $s_{1,x,d} (0 \leq d \leq 4)$ 均满足线段数量是 $O(1)$ 的，具体来说均满足线段数量 ≤ 2 ，因为一个节点的与其距离 ≤ 3 的距离相同的轻儿子一定会在同一时刻被连续标号，但由于可能并不是处理当前节点所在重链时对其的标号（即与其距离 < 3 的祖先方向的节点与其并不在一条重链上，并且会将这一层的节点进行标号），因此这个节点的重儿子的编号可能会将连续编号划分成两段，因此线段数量 ≤ 2 ；而当距离 ≥ 4 时显然编号是连续的，因此线段数量 ≤ 1 。

基于调整搜索顺序的重标号树链剖分

同时也可以发现 $s_{0,x,d}(0 \leq d \leq 4)$ 均满足线段数量是 $O(k)$ 的，具体来说满足线段数量 ≤ 4 ，因为一个节点的与其距离 ≤ 3 的距离相同的所有节点中，对于相同的重儿子来说，每一层一定都是连续的，因此即使每一层之间都是独立的，线段数量仍然满足 ≤ 4 。

因此如果我们对于单点修改其 $\leq d$ 的轻儿子子树的范围内的所有点，则线段总数为 $O(k)$ ，如果我们对于单点修改其 $\leq d$ 的子树范围内的所有点，则线段总数为 $O(k^2)$ 。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

基于调整搜索顺序的重标号树链剖分

考虑一次链邻域修改操作：考虑每次对同一条重链上的点 x 和点 y 之间的信息进行处理，首先对重链上的顶部的至多三个节点进行特殊处理，需要对其轻儿子子树信息进行处理；重链的尾部节点需要对其子树信息进行处理；中间的部分可以通过编号的连续性进行处理；重链贡献重复部分对其子树信息进行处理；总共需要处理 $O(\log n)$ 条重链，因此该部分时间复杂度为 $O(k^2 \log^2 n)$ ；最后需要对最近公共祖先之上的部分进行处理；因此单次修改总时间复杂度为 $O(k^2 \log^2 n)$ 。

考虑一次链邻域查询操作：可以发现和修改操作是类似的，同样对重链顶部上至多三个节点进行特殊处理；重链尾部节点进行子树处理；中间部分通过编号连续性来处理；重链贡献重复部分进行子树处理；最后对最近公共祖先之上的部分进行处理；因此单次查询总时间复杂度为 $O(k^2 \log^2 n)$ 。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

基于调整搜索顺序的重标号树链剖分

子树修改和子树查询是简单的，由于我们按照 dfs 顺序给每条重链标号，因此我们可以扫描节点 x 的子树信息进行处理，单次修改和查询的总时间复杂度为 $O(k^2 \log n)$ 。该做法可以支持查询链上范围 $\leq k (0 \leq k \leq 3)$ 的点权最值和子树的点权最值，做法和查询点权和基本相同。时间复杂度 $O(mk^2 \log^2 n)$ ，常数较大，可以通过本题。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

总结

通过树链剖分的树上链分治的角度，我们可以得到许多关于树上范围相关问题的思考和启发。

树链剖分具有非常丰富和广泛的性质，包含重链剖分、长链剖分等多种类型，还可以深入发掘其中更具有拓展性的思想。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆



点分树

点分树在处理树上范围相关问题中具有广泛应用。我们默认读者已经掌握点分树的相关内容和基本应用。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

点分树与树上范围传递性相关问题

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

例题：地雷

给定一棵 n 个节点的树，边 i 连接 u_i 和 v_i ，边权为 w_i 。点 i 有半径 r_i 和点权 a_i ，定义 $\text{dist}(i, j)$ 表示点 i 和点 j 在树上的最短路径长度。当点 i 被激活时，所有满足 $\text{dist}(i, j) \leq r_i$ 的尚未被激活的点 j 将会被激活，反复进行该过程，直到没有节点被激活为止。

对于每一个节点 i ，计算如果初始时激活了点 i ，最终所有被激活的点的点权之和。

数据范围： $1 \leq n \leq 10^5, 1 \leq a_i \leq 10^9, 0 \leq r_i \leq 10^{18}, 1 \leq u_i, v_i \leq n, 1 \leq w_i \leq 10^{12}$ 。

点分树与树上范围传递性相关问题

定义点 i 直接可达点 j 表示 $\text{dist}(i, j) \leq r_i$ ，定义点 i 可达点 j 表示点 i 直接可达点 j 或存在一个点 k 满足点 i 可达点 k 且点 k 可达点 j 。

注意到点 i 直接激活的点在树上点 i 的半径为 r_i 的范围，建立点分树。定义 rt 为当前点分树中的一个子树的根。

设 $f_{rt,x}$ 表示初始激活点 x 后，最终能给根 rt 提供的激活的范围余量，即设点 y 最终被激活，则 $f_{rt,x} = \max_y \{r_y - \text{dist}(y, rt)\}$ ；

设 $g_{rt,x}$ 表示至少需要初始时 rt 提供的激活范围 $g_{rt,x}$ 能够使得最终点 x 被激活，即存在点 y 满足 $\text{dist}(rt, y) \leq g_{rt,x}$ 且点 y 可达点 x 。

点分树与树上范围传递性相关问题

首先考虑给 f 和 g 添加点分树子树的限制，即在上述定义的基础上限制访问节点的范围为 rt 的子树，按深度由深到浅依次处理出 f 和 g 。

对于一个根 rt ，考虑逐渐增大其初始提供的激活范围 r' ，由于其子树内的 f 和 g 更新完毕，因此当点 x 被激活时，设 rt' 为点分树上 rt 子树内且在点 x 祖先中的节点，则对于点 y 满足 $f_{rt',x} \geq g_{rt',y}$ 时，点 y 将被激活，按照上述过程，可以处理出当前点分树子树的 g 的信息。

对于一个点 x ，我们可以处理出其在 rt 对应儿子节点子树的范围内能够提供给 rt 的最大激活范围，即设 rt' 为点分树上 rt 子树内且在点 x 祖先中的节点，则对于点 y 满足 $f_{rt',x} \geq g_{rt',y}$ 时，点 y 可以给 $f_{rt,x}$ 提供 $r_y - \text{dist}(y, rt)$ 的贡献。

对 rt 的 f 和 g 排序后维护前缀信息即可转移。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

点分树与树上范围传递性相关问题

接下来考虑去掉给 f 和 g 添加的点分树子树范围限制，此时注意到点分树上的根（全局重心）的信息是正确的，因此在点分树上换根处理，按深度由浅到深依次处理 f 和 g 。

对于一个根 rt ，其点分树上的祖先的 f 和 g 是正确信息，考虑如何更新其 f 和 g 的信息。

对于一个点 x ，考虑其激活后首次激活到子树外的点，激活到一个子树外的点 y ，且 x 和 y 路径上对应点分树祖先为 rt' ，枚举 rt' ，则当 $f_{rt',x} \geq g_{rt',y}$ 时，点 y 可以给 $f_{rt,x}$ 提供 $r_y - \text{dist}(y, rt)$ 的贡献；同样的，对于一个点 x ，考虑其最终被激活前最后一次由子树外的点激活到子树内的点，由一个子树外的点 y 激活，且 x 和 y 路径上对应点分树祖先为 rt' ，枚举 rt' ，则当

$f_{rt',y} \geq g_{rt',x}$ 时，点 y 可以给 $g_{rt,x}$ 提供 $\text{dist}(rt, y)$ 的贡献。

按照上述方式进行换根处理，维护出所有 rt 的 f 和 g 的信息，查询时从点 x 向点分树上跳父亲节点，去掉点 x 所在儿子节点子树后查询经过该重心的点权和，时间复杂度 $O(n \log^2 n)$ 。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆



总结

点分树可以处理许多较为基础的树上范围相关问题。从上述例题中还可以发现，点分树在处理一类树上范围传递性相关问题中具有较强的性质。

点分树由于其优秀的树形结构，在处理一类树上范围相关问题时，能够维护许多范围相关的信息，因此具有广泛应用。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

Top Tree

Top Tree 功能非常强大，本文仅介绍较为基础的 Top Tree 与树上范围相关问题的一些应用。我们默认读者已经掌握 Top Tree 的相关定义，下文中簇的端点表示簇的界点。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

静态 Top Tree 的一种构建（全局平衡二叉树）

感兴趣的读者可以阅读我的集训队论文。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

例题：树上邻域数点

给定一棵 n 个节点的树，你可以使用一些 compress 和 rake 操作进行簇运算，可以进行不超过 M 次簇运算的预处理，有 m 次询问，每次询问给出节点 u 和范围 d ，要求使用至多 1 次簇运算，得到一个点集没有限制，边集恰好为节点 u 的范围为 d 的所有边构成的集合的合法的簇。

数据范围： $1 \leq n \leq 2 \times 10^5$ ， $1 \leq m \leq 10^6$ ，预处理的簇运算次数限制 $M = 3 \times 10^7$ 。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

Top Tree 与树上范围相关的应用

首先构建原树的 Top Tree，考虑当节点 u 为原树的根时的做法，我们在每个簇上对该簇的两个端点，都分别维护出簇的边集中与端点距离为 d 的合法簇信息，由于我们只维护簇内的信息，因此需要维护的 d 的范围上界为该簇的端点对应的最长路径长度，方便起见我们使用每个簇的大小作为 d 的上界。

设簇类型 $f_{x,0/1,d}$ 表示簇内距离簇 x 的左端点或右端点距离 $\leq d$ 的簇的信息，要求信息的一端为对应左右端点，且当 $d \geq$ 簇的左右端点距离时信息两端点分别为该簇的左右端点，发现 compress 和 rake 类型节点可以进行来自左右儿子的信息合并，因此可以不使用簇运算回答每次关于根的询问。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

Top Tree 与树上范围相关的应用

当询问点为原树任意点 x 时，我们选择该点在原树上的任意一条与之相连的边，找到一个满足条件的簇使得该簇是该边在 Top Tree 对应节点的祖先且点 x 的范围为 d 可以完全包含该簇信息，要求该簇在 Top Tree 上的深度最小，此时答案为点 x 到该簇的两个端点再向该范围外进行拓展的簇的信息合并。

设簇类型 $g_{x,0/1,d}$ 表示簇外距离簇 x 的左端点或右端点距离 $\leq d$ 的簇的信息，其中 d 的上界为簇 x 在 Top Tree 上父节点的大小，当簇 x 不为 Top Tree 的根时，父亲节点大小 $\geq d$ ，因此可以使用类似换根 dp 的方式通过簇自身的 f 和父亲节点对应簇的 g 维护出簇自身的 g ，然后通过一个簇的 f 和 g 的信息即可在至多 1 次簇运算内快速回答询问。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

Top Tree 与树上范围相关的应用

按照上述方式预处理，由于该 Top Tree 深度为 $O(\log n)$ ，即节点子树大小的和的级别为 $O(n \log n)$ ，因此预处理簇运算次数为 $O(n \log n)$ ，可以通过本题。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

总结

Top Tree 功能强大，从上述例题中可以发现 Top Tree 在处理相关问题时的特殊的结构和特殊的信息合并方式，进一步提升了对树上范围相关问题的处理能力。

浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆



总结

本文介绍了树链剖分和基础 Top Tree 在树上范围相关问题中的运用，事实上很多树上问题经典算法在树上范围相关问题中具有广泛运用的潜力，笔者希望本文能够激发读者思考，启发读者对树上范围相关问题进行更加深入的研究。



全国青少年信息学奥林匹克竞赛 (CCF NOI)

E-mail: noi@ccf.org.cn

网址: <http://www.noi.cn>

