2016 年信息学奥林匹克

中国国家队候选队员论文集

教练：余林韵、陈许旻

中国计算机学会

目录

1 积性函数求和的几种方法
任之洲　绍兴市第一中学 **1**

2 网络流的一些建模方法
姜志豪　东营市胜利第一中学 **17**

3 浅谈线性规划与对偶问题
董克凡　福建省福州第一中学 **33**

4 浅谈无向图最小割问题的一些算法及应用
王文涛　绍兴市第一中学 **63**

5 浅谈线性规划在信息学竞赛中的应用
邹逍遥　宁波市镇海中学 **81**

6 区间最值操作与历史最值问题
吉如一　杭州学军中学 **99**

7 再探快速傅里叶变换
毛啸　雅礼中学 **123**

8 从 **Unknown** 谈一类支持末尾插入删除的区间信息维护方法
罗哲正　安徽师范大学附属中学 **141**

9 小 **C** 的后缀数组命题报告
洪华敦　绍兴市第一中学 **163**

10 消消看　命题报告
张浩威　浙江省余姚中学 **171**

11 《**strakf**》命题报告
李子豪　佛山石中 **179**

12 《过去的集合》命题报告
汪文潇　厦门双十中学 **189**

13 火车司机出秦川　命题报告
吴作凡　安徽师范大学附属中学 **195**

14 基础排序算法练习题
金策　浙江省杭州学军中学 **209**

15 move 命题报告
袁宇韬　长沙市雅礼中学 **221**

积性函数求和的几种方法

绍兴市第一中学 任之洲

摘 要

积性函数是一类特殊的数论函数，本文归纳整理了一些常用的积性函数相关问题的计算技巧，并在这些算法的启发下得到了一种适用范围较广的积性函数求和方法。

1 引言

在信息学竞赛中，有时会遇到一些定义在正整数域下的函数，即数论函数。积性函数作为一类特殊的数论函数，经常成为我们讨论的对象。在这一类问题中，选手们往往需要先进行一些推导，并选择合适的方法计算整理后的式子，这时就需要一些高效的算法来帮助我们完成最后的计算。

在本文中，作者首先对信息学竞赛中常用的一些计算方法进行了整理。

第2节中，介绍了积性函数的定义，并列举了一些常见的例子。

第3节中，介绍了利用线性筛对积性函数前n项进行快速求值的方法。

第4节中，介绍了*Dirichlet*卷积的定义以及*Möbius*反演，为后一节做铺垫。

第5节中，结合了前两节的内容，给出了一种高效的数论函数求和方法，这个算法在国内信息学竞赛中一般被称为杜教筛，是一种利用*Dirichlet*卷积进行构造的算法，虽然和积性没有很大联系，且需要被计算函数本身拥有特殊的性质，使用条件略为苛刻，但可以给我们许多启发。

在本文的第6节中，作者在前几节算法的启发下得到了一种新颖的积性函数求和方法，这种方法的效率虽然略差于第5节中的算法，但更加广泛适用。在这一节中，作者将从得到该算法的设计初衷谈起，逐步构筑出整个流程，并介绍这个算法在积性函数求和以外的应用。

2 定义

定义2.1. 定义域为正整数域，陪域为复数域的函数被称为**数论函数**。

定义2.2. 设$f(x)$为一个数论函数，若对于每一对互质的正整数a, b都满足$f(ab) = f(a)f(b)$，则$f(x)$是**积性函数**。

定义2.3. 设$f(x)$为一个数论函数，若对于每一对正整数a, b都满足$f(ab) = f(a)f(b)$，则$f(x)$是**完全积性函数**。

本文涉及的函数若无特殊说明，均为数论函数。

2.1 常见的积性函数

以下为一些常见的积性函数：

- 除数函数 $\sigma_k(n)$，表示n的所有约数的k次幂之和。
- $Euler$函数 $\varphi(n)$，表示不超过n且与n互质的正整数个数。
- $Möbius$函数 $\mu(n) = \begin{cases} 1 & n = 1 \\ (-1)^k & n\text{为}k\text{个不同质数之积} \\ 0 & \text{其它情况} \end{cases}$

2.2 常见的完全积性函数

以下为一些常见的完全积性函数：

- 幂函数 $Id_k(n) = n^k$
- 单位函数 $\epsilon(n) = \begin{cases} 1 & n = 1 \\ 0 & n > 1 \end{cases}$

3 线性筛

线性筛是计算积性函数时最常用的算法，可以$O(n)$计算出$2 \sim n$的最小质因子，利用这些信息可以对前n项函数值完成递推。

3.1 计算最小质因子

考虑将一个正整数n质因子分解，并将质因子降序排列。

即设$n = \prod_{i=1}^{k} p_i$，其中p_i为质数，且对于$1 \le i < k$满足$p_i \ge p_{i+1}$。

这样的分解方式是唯一的，可以利用唯一性得到以下算法：

- 设pr_i为i的最小质因子，i从2开始枚举计算。
- 假如当前的pr_i还没有被计算出来，那么i为质数，$pr_i = i$。
- 枚举每一个不超过pr_i的质数p，$pr_{ip} = p$。

每个数只会被最小质因子筛去，所以这个算法的时间复杂度为$O(n)$。

3.2 计算积性函数前n项

设$f(x)$为一个积性函数，现在需要计算$f(x)$前n项的值。

假如可以将x分解为一对互质的a, b且$a, b > 1$，那么$f(x)$的值可以根据定义2.2 递推得到，剩下只需要计算x只有一种质因子的情况。

可以利用线性筛$O(n)$计算出每个数的最小质因子，再递推得到每个数最小质因子的幂次。

设x的最小质因子为p，幂次为c，且$p^c \ne x$，那么$f(x) = f(p^c) f(\frac{x}{p^c})$。

3.3 例题一

例1. *计算Euler函数$\varphi(n)$的前n项。*

对于$n > 1$，设$n = \prod_{i=1}^{k} p_i^{c_i}$，其中$p_i$均为质数且对于$i \ne j$满足$p_i \ne p_j$。

*Euler*函数有经典的计算式

$$\varphi(n) = n \prod_{i=1}^{k} \frac{p_i - 1}{p_i} = \prod_{i=1}^{k} (p_i - 1) p_i^{c_i - 1}$$

当n只有一种质因子的情况可以方便地计算，剩下的部分可以参考 3.2 节中的方法利用线性筛递推，时间复杂度$O(n)$。

4 *Möbius*反演

4.1 *Dirichlet*卷积

定义4.1. 定义两个数论函数$f(x), g(x)$的*Dirichlet*卷积（记作$(f * g)(x)$）

$$(f * g)(n) = \sum_{d|n} f(d)g(\frac{n}{d})$$

*Dirichlet*卷积具有如下运算性质：

- 交换律：$f * g = g * f$
- 结合律：$(f * g) * h = f * (g * h)$
- 分配律：$f * (g + h) = f * g + f * h$
- 单位元[1]：$f * \epsilon = f$
- 若f, g均为积性函数，则$f * g$也为积性函数。

以上性质的证明比较简单，留给读者自己思考。

4.2 *Möbius*反演

引理4.1. $\mu * 1 = \epsilon$，即

$$\sum_{d|n} \mu(d) = \epsilon(n)$$

其中函数1是返回值恒为1的常数函数。

Proof. 设n有k种不同质因子，那么

$$\begin{aligned} \sum_{d|n} \mu(d) &= \sum_{i=0}^{k} (-1)^i \binom{k}{i} \\ &= (1-1)^k \\ &= \epsilon(k) = \epsilon(n) \end{aligned}$$

□

[1] ϵ为2.2节中提到的单位函数。

定理4.1 (*Möbius*反演)**.**

如果有两个数论函数f, g满足

$$f(n) = \sum_{d|n} g(d)$$

则它们也满足

$$g(n) = \sum_{d|n} \mu(d) f\left(\frac{n}{d}\right)$$

反之亦然，即

$$f = g * 1 \Leftrightarrow g = \mu * f$$

Proof. 设

$$f = g * 1$$

两侧都卷上函数μ，得

$$\mu * f = \mu * g * 1$$

根据引理 4.1 整理得

$$\mu * f = \epsilon * g = g$$

两侧都卷上函数1，得

$$g * 1 = \mu * f * 1 = \epsilon * f = f$$

□

4.3 *Möbius*反演的推广

*Möbius*反演还有许多扩展形式，接下来介绍一种在数论函数方面的推广。

引理4.2. 设正整数x, y, a, b，其中$a, b \le x$，$y = \left\lfloor \frac{x}{a} \right\rfloor$，那么有$\left\lfloor \frac{y}{b} \right\rfloor = \left\lfloor \frac{x}{ab} \right\rfloor$。

Proof.

设$x = kab + c$，其中k, c均为非负整数且$c < ab$，即$k = \left\lfloor \frac{x}{ab} \right\rfloor$。

$y = \left\lfloor \frac{x}{a} \right\rfloor = kb + \left\lfloor \frac{c}{a} \right\rfloor$，其中$\left\lfloor \frac{c}{a} \right\rfloor < b$，所以$\left\lfloor \frac{y}{b} \right\rfloor = k$。 □

定理4.2. 设f,g为两个数论函数，t为一个完全积性函数，且$t(1)=1$，有

$$\begin{aligned} f(n) &= \sum_{k=1}^{n} t(k) g\left(\left\lfloor \frac{n}{k} \right\rfloor\right) \\ \Leftrightarrow \quad g(n) &= \sum_{k=1}^{n} \mu(k) t(k) f\left(\left\lfloor \frac{n}{k} \right\rfloor\right) \end{aligned}$$

Proof. 考虑将原式代入，根据引理 4.2 得

$$\begin{aligned} &\sum_{k=1}^{n} \mu(k) t(k) f\left(\left\lfloor \frac{n}{k} \right\rfloor\right) \\ = &\sum_{k=1}^{n} \mu(k) t(k) \sum_{i=1}^{\lfloor \frac{n}{k} \rfloor} t(i) g\left(\left\lfloor \frac{n}{ki} \right\rfloor\right) \\ = &\sum_{j=1}^{n} g\left(\left\lfloor \frac{n}{j} \right\rfloor\right) \sum_{i|j} \mu(i) t(i) t\left(\frac{j}{i}\right) \\ = &\sum_{j=1}^{n} g\left(\left\lfloor \frac{n}{j} \right\rfloor\right) t(j) \sum_{i|j} \mu(i) \end{aligned}$$

由引理 4.1 得

$$\begin{aligned} \sum_{k=1}^{n} \mu(k) t(k) f\left(\left\lfloor \frac{n}{k} \right\rfloor\right) &= \sum_{j=1}^{n} g\left(\left\lfloor \frac{n}{j} \right\rfloor\right) t(j) \epsilon(j) \\ &= g(n) t(1) = g(n) \end{aligned}$$

反之亦然。 □

5 利用*Dirichlet*卷积构造

这个算法在国内信息学竞赛中一般被称为杜教筛，可以高效地完成一些常见数论函数的计算。

5.1 主要形式

设$f(n)$为一个数论函数，需要计算

$$S(n) = \sum_{i=1}^{n} f(i)$$

根据函数$f(n)$的性质，构造一个$S(n)$关于$S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$的递推式，如下例。

找到一个合适的数论函数$g(n)$

$$\sum_{i=1}^{n}\sum_{d|i}f(d)g\left(\frac{i}{d}\right)=\sum_{i=1}^{n}g(i)S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$$

可以得到递推式

$$g(1)S(n)=\sum_{i=1}^{n}(f*g)(i)-\sum_{i=2}^{n}g(i)S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$$

引理5.1. 对于一个正整数n，考虑所有正整数$i\in[1,n]$，$\left\lfloor\frac{n}{i}\right\rfloor$的取值只有$O(\sqrt{n})$种。

Proof.

对于所有$i\le\sqrt{n}$，$\left\lfloor\frac{n}{i}\right\rfloor\ge\sqrt{n}$，这一部分取值为$O(\sqrt{n})$种。

对于所有$i>\sqrt{n}$，$\left\lfloor\frac{n}{i}\right\rfloor<\sqrt{n}$，这一部分取值也为$O(\sqrt{n})$种。 □

根据引理 4.2 和引理 5.1，在递推计算$S(n)$的整个过程中，需要被计算的$S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$只有$O(\sqrt{n})$种。

假如可以快速对$(f*g)(i)$以及$g(i)$完成求和，可以根据$\left\lfloor\frac{n}{i}\right\rfloor$的取值进行分段，计算一个$S(n)$的复杂度即为$O(\sqrt{n})$，整个过程的复杂度如下

$$\sum_{i=1}^{\lfloor\sqrt{n}\rfloor}O\left(\sqrt{i}\right)+\sum_{i=1}^{\lfloor\sqrt{n}\rfloor}O\left(\sqrt{\frac{n}{i}}\right)$$

后一部分的复杂度一定大于前一部分，所以只需考虑

$$\sum_{i=1}^{\lfloor\sqrt{n}\rfloor}O\left(\sqrt{\frac{n}{i}}\right)\approx O\left(\int_{0}^{\sqrt{n}}\sqrt{\frac{n}{x}}\mathrm{d}x\right)=O(n^{\frac{3}{4}})$$

构造递推式的方法需要根据函数具体分析，可以参考下面几个例子。

5.2 例题二

例2. 计算

$$S(n)=\sum_{i=1}^{n}\mu(i)$$

根据引理 4.1 ，用 5.1 节中的方法可以得到递推式

$$S(n)=\sum_{i=1}^{n}\epsilon(i)-\sum_{i=2}^{n}S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)=1-\sum_{i=2}^{n}S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$$

直接计算的复杂度为$O(n^{\frac{3}{4}})$，考虑可以线性筛预处理前$n^{\frac{2}{3}}$项，那么递推部分的复杂度为

$$O\left(\int_0^{n^{\frac{1}{3}}}\sqrt{\frac{n}{x}}\mathrm{d}x\right)=O(n^{\frac{2}{3}})$$

结合线性筛部分，总复杂度为$O(n^{\frac{2}{3}})$。

5.3 例题三

例3 (SPOJ DIVCNT2). 计算

$$S(n)=\sum_{i=1}^{n}\sigma_0(i^2)$$

同样考虑构造*Dirichlet*卷积关系式，设$f(n)=\sigma_0(n^2)$，$g(n)=2^{\omega(n)}$，其中$\omega(n)$表示n的不同质因子个数，有$f=g*1$，即

$$\sigma_0(n^2)=\sum_{d|n}2^{\omega(d)}$$

Proof. 考虑n的每个约数d，可以在d^2中去掉d的一个质因子集合，也就是$2^{\omega(d)}$种去除方法，这样可以枚举出n^2的所有约数。 □

容易得到$g=\mu^2*1$，那么可以得到如下递推关系[2]

$$\begin{aligned}\sum_{i=1}^{n}\sigma_0(i^2)&=\sum_{i=1}^{n}2^{\omega(i)}\left\lfloor\frac{n}{i}\right\rfloor\\ \sum_{i=1}^{n}2^{\omega(i)}&=\sum_{i=1}^{n}\mu^2(i)\left\lfloor\frac{n}{i}\right\rfloor\\ \sum_{i=1}^{n}\mu^2(i)&=\sum_{i=1}^{\lfloor\sqrt{n}\rfloor}\mu(i)\left\lfloor\frac{n}{i^2}\right\rfloor\end{aligned}$$

与例 2 类似，可以利用线性筛$O(n^{\frac{2}{3}})$完成计算。

[2]其中μ^2求和的计算式可以根据容斥得到。

5.4 小结

根据引理 4.2 和引理 5.1，这一类递推计算第n项时只会涉及到$O(\sqrt{n})$个状态，通过积分可以计算出递推转移的复杂度为$O(n^{\frac{3}{4}})$，利用线性筛预处理后可以进一步降低复杂度。

以个思想为基础可以构筑起下一节中的算法，关于这一节的内容，更详细的可见参考文献 2，本文不做更多展开。

6 一种基于质因子分解的求和方法

在很多时候，我们并不能轻易地找到被求和函数的性质，这时就希望能有一种适用范围更广的高效求和方法，接下来本文将会对形式较为一般的积性函数给出一种求和算法。

6.1 回顾

设$F(n)$为一个积性函数，n的质因子分解为

$$n = \prod_{i=1}^{k} {p_i}^{c_i}$$

考虑这样来描述一个积性函数：

- 当n为质数时，即$n = p$，$F(p) = G(p)$。
- 当n为质数的幂时，即$n = p^c$且$c > 1$，$F(p^c) = T(p^c)$。
- 剩下的情况根据积性，$F(n) = \prod_{i=1}^{k} F({p_i}^{c_i})$。

例如，对于以下两个常见的积性函数：

- *Euler*函数 $\varphi(n)$ 中，$G(p) = p - 1$，$T(p^c) = (p - 1)p^{c-1}$。
- *Möbius*函数 $\mu(n)$ 中，$G(p) = -1$，$T(p^c) = 0$。

更一般地，$G(p)$可以是一个与p相关的多项式，$T(p^c)$可以是一个与p和c相关的多项式，对于这些情况或是更复杂的表达式，前一节中介绍的算法将可能不再适用。考虑不再依赖函数自身的性质，而是直接根据积性来完成计算。

6.2 引入问题

以这一类问题的一个简单形式为例。

例4. 定义一个欧拉函数的变种$\phi(n,d)$，n的质因子分解为$\prod_{i=1}^{k} {p_i}^{c_i}$。

其中p_k为互不相同的质因子（$c_k > 0$，即把n质因子分解），那么

$$\phi(n,d) = \prod_{i=1}^{k}({p_i}^{c_i} + d)$$

特别地，定义$\phi(1,d) = 1$。

对于给定的n,d，求

$$\left(\sum_{i=1}^{n}\phi(i,d)\right) mod\ 10^9 + 7$$

设$F(n) = \phi(n,d)$，沿用 6.1 节中的表示方法，$G(p) = p + d$，$T(p^c) = p^c + d$。

6.3 初步转化

引理6.1. 对于一个正整数$x \le n$，x最多拥有一个大于$\sqrt{n}$的质因子。

Proof. 假设x拥有两个大于$\sqrt{n}$的质因子p_1, p_2，那么$p_1p_2 > n$，与$x \le n$不符。 □

根据引理 6.1 ，可以将$F(x)$分为两类考虑：

- x没有大于$\sqrt{n}$的质因子。
- x有一个大于$\sqrt{n}$的质因子。

根据函数的积性，可以得到

$$\sum_{i=1}^{n} F(i) = \sum_{\substack{x \le n \\ x\text{没有大于}\sqrt{n}\text{质因子}}} F(x)\left(1 + \sum_{\substack{\sqrt{n} < p \le \left\lfloor \frac{n}{x} \right\rfloor \\ p\text{为质数}}} G(p)\right)$$

$F(x)$后乘的系数只与$\left\lfloor \frac{n}{x} \right\rfloor$的值有关，即不同的系数只有$O(\sqrt{n})$种，可以依据$\left\lfloor \frac{n}{x} \right\rfloor$的取值将$F(x)$分为$O(\sqrt{n})$段，每一段对应一种系数。

设y为一组固定的$\left\lfloor \frac{n}{x} \right\rfloor$的值，也就是需要计算这两部分

$$\sum_{\substack{\sqrt{n} < p \le y \\ p\text{为质数}}} G(p) \qquad \sum_{\substack{\left\lfloor \frac{n}{x} \right\rfloor = y \\ x\text{没有大于}\sqrt{n}\text{质因子}}} F(x)$$

6.4 $G(p)$的计算

6.4.1 初步分析

在例 4 的式子中，$G(p) = p + d$，不妨令$G(p)$是一个关于p的低阶多项式，考虑对多项式的每一项分别计算。

设不超过$\sqrt{n}$的质数共m个，升序排列为$p_1 \sim p_m$。

设$g_k[i][j]$为$[1, j]$范围内与前i个质数互质的所有数的k次幂之和，其中$i = 0$时的部分可以用插值$O(k)$计算。

对于$i \geq 1$的情况，考虑$[1, j]$范围内有多少含有p_i质因子的数与前$i - 1$个数互质，设$l = \left\lfloor \frac{j}{p_i} \right\rfloor$，有如下递推式

$$g_k[i][j] = g_k[i-1][j] - {p_i}^k g_k[i-1][l]$$

递推最终得到的$g_k[m][j] - 1$即为$[1, j]$范围内大于$\sqrt{n}$的质数的k次幂之和。

根据引理 4.2 和引理 5.1，计算$g_k[m][n]$过程中需要的第二维状态共$O(\sqrt{n})$种，并且这些状态恰为每一种需要被计算的$G(p)$之和。

6.4.2 朴素实现

朴素实现这个递推式需要依次枚举质数p_i，对每一个状态进行计算。

考虑每个$\left\lfloor \frac{n}{x} \right\rfloor$有多少质数需要转移，其中$\left\lfloor \frac{n}{x} \right\rfloor > \sqrt{n}$的部分需要转移$\sqrt{n}$范围内的所有质数[3]

$$\sum_{i=1}^{\lfloor \sqrt{n} \rfloor} O\left(\frac{i}{\log i}\right) + \sum_{i=1}^{\lfloor \sqrt{n} \rfloor} O\left(\frac{\sqrt{n}}{\log \sqrt{n}}\right) \approx O\left(\frac{n}{\log n}\right)$$

6.4.3 优化

当$p_{i+1} > j$时，一定满足$g_k[i][j] = 1$。

所以，当${p_i}^2 > j \geq p_i$，即$l = \left\lfloor \frac{j}{p_i} \right\rfloor < p_i$时

$$g_k[i-1][l] = 1$$

$$g_k[i][j] = g_k[i-1][j] - {p_i}^k$$

[3]本文中，质数密度按$O\left(\frac{1}{\log n}\right)$估计，$O\left(\frac{1}{\log n}\right)$与$O\left(\frac{1}{\log \sqrt{n}}\right)$同阶。

因此$p_i^2 > j$时的计算实际上是可以被省去的，对于每个j记录上一次转移时的i的值，在调用$g_k[i][j]$这个位置时将没有被计算的这一段质数的贡献计入。

对于每一组$\left\lfloor \frac{n}{x} \right\rfloor$只需要转移不超过$\sqrt{\left\lfloor \frac{n}{x} \right\rfloor}$的质数，复杂度可估计为

$$\sum_{i=1}^{\lfloor\sqrt{n}\rfloor} O\left(\frac{\sqrt{i}}{\log i}\right) + \sum_{i=1}^{\lfloor\sqrt{n}\rfloor} O\left(\frac{\sqrt{\frac{n}{i}}}{\log\sqrt{\frac{n}{i}}}\right)$$

后半部分的贡献一定大于前半部分，且$\sqrt{\frac{n}{i}} \ge n^{\frac{1}{4}}$，故复杂度可以估计为

$$\sum_{i=1}^{\lfloor\sqrt{n}\rfloor} O\left(\frac{\sqrt{\frac{n}{i}}}{\log\sqrt{\frac{n}{i}}}\right) \approx O\left(\frac{\int_0^{\sqrt{n}} \sqrt{\frac{n}{x}}\mathrm{d}x}{\log n}\right) = O\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$$

6.5 $F(x)$的计算

6.5.1 初步分析

由于$\left\lfloor \frac{n}{x} \right\rfloor$值相同的$F(x)$需要乘的系数是相同的，可以直接用$\left\lfloor \frac{n}{x} \right\rfloor$来表示状态。

同样设不超过$\sqrt{n}$的质数共m个，升序排列为$p_1 \sim p_m$。

设$f[i][j]$为只包含前i种质因子，且$\left\lfloor \frac{n}{x} \right\rfloor = j$的$F(x)$之和，其中$f[0][n] = 1$。

根据引理 4.2 和引理 5.1 ，第二维状态数仍然可以缩减到$O(\sqrt{n})$种，每一种对应着一组系数。

6.5.2 朴素实现

考虑由$f[i-1][j]$用质数p_i进行转移，枚举转移幂次c，设$l = \left\lfloor \frac{j}{p_i^c} \right\rfloor$，对$f[i][l]$的贡献为$T(p_i^c)f[i-1][j]$或$G(p_i)f[i-1][j]$（$c = 1$的情况）。

同样计算每个$\left\lfloor \frac{n}{x} \right\rfloor$有多少质数需要转移，考虑枚举每一种$c$的计算量，设

$$h(n) = \sum_{k=2}^{\lfloor\log_2 n\rfloor} O(n^{\frac{1}{k}}) \approx O(\sqrt{n})$$

复杂度估计和 6.4 节中类似

$$\sum_{i=1}^{\lfloor\sqrt{n}\rfloor} O\left(\frac{i+h(i)}{\log i}\right) + \sum_{i=1}^{\lfloor\sqrt{n}\rfloor} O\left(\frac{\sqrt{n}+h(\frac{n}{i})}{\log n}\right) \approx O\left(\frac{n}{\log n}\right)$$

6.5.3 优化

在前一小节中，通过省去了$p_i^2 > j$时的计算而将复杂度降低至$O\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$，考虑采取类似的优化措施。

注意到，当$\left\lfloor \frac{n}{x} \right\rfloor$不超过$\sqrt{n}$时

$$1 + \sum_{\substack{\sqrt{n} < p \le \left\lfloor \frac{n}{x} \right\rfloor \\ p\text{为质数}}} G(p) = 1$$

而最后计算答案的式子为

$$\sum_{i=1}^{n} F(i) = \sum_{\substack{x \le n \\ x\text{没有大于}\sqrt{n}\text{质因子}}} F(x) \left(1 + \sum_{\substack{\sqrt{n} < p \le \left\lfloor \frac{n}{x} \right\rfloor \\ p\text{为质数}}} G(p) \right)$$

这一部分$F(x)$对答案的贡献系数是相同的，可以以此为突破口进行优化。

当$p_i^2 > y = \left\lfloor \frac{n}{x} \right\rfloor$时，$\left\lfloor \frac{y}{p_i} \right\rfloor < p_i \le \sqrt{n}$，所以状态$y$最多只能用一个超过$p_i$的质数转移，并且转移后的值对答案的贡献系数一定为1。

优化后的转移策略如下：

- 对于一个质数p_i，只转移$y \ge p_i^2$的状态。
- 设$l = \left\lfloor \frac{y}{p_i^c} \right\rfloor$，假如$l < p_i^2$，那么这个状态在之后的转移中会被忽略，需要现在计算它对答案的贡献，也就是需要统计$[p_{i+1}, l]$范围内的$G(p)$之和。
- 维护每个状态最后一次转移时的p_i，统计被忽略的那一段$G(p)$之和。

复杂度估计和 6.4 节中类似

$$\sum_{i=1}^{\lfloor \sqrt{n} \rfloor} O\left(\frac{h(i)}{\log i}\right) + \sum_{i=1}^{\lfloor \sqrt{n} \rfloor} O\left(\frac{h(\frac{n}{i})}{\log n}\right) \approx O\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$$

6.5.4 另一种实现[4]

注意到只有当$x < \sqrt{n}$时，才能够再乘入一个大于$\sqrt{n}$的质数，并且这一部分的x也不可能含有任意大于$\sqrt{n}$的质因子。

[4]感谢毛啸同学提供这种实现方法。

可以先用线性筛处理出所有$x < \sqrt{n}$的$F(x)$值，乘上相应的系数计入答案，剩下的问题是计算

$$\sum_{\substack{\sqrt{n} \le x \le n \\ x\text{没有大于}\sqrt{n}\text{质因子}}} F(x)$$

与前一节不同，将不超过$\sqrt{n}$的质数降序排列为$p_1 \sim p_m$。

设$f'[i][j]$为只包含前i种质因子，且$x \le j$的$F(x)$之和，容易得到$f'[0][j] = 1$，且$f'[m][n]$减去$x < \sqrt{n}$的$F(x)$之和就是需要求的结果。

容易得到一个递推转移，枚举质因子p_i的幂次c，设$l_c = \left\lfloor \frac{j}{p_i{}^c} \right\rfloor$，有

$$f'[i][j] = f'[i-1][j] + G(p_i)f'[i-1][l_1] + \sum_{c \ge 2} T(p_i{}^c)f'[i-1][l_c]$$

参考前一节，这样递推的第二维状态同样只有$O(\sqrt{n})$种，朴素实现的复杂度为$O\left(\frac{n}{\log n}\right)$，考虑采取类似的优化措施。

当$p_i{}^2 > j$时，由于p_i是降序排列的，只需要统计$[p_i, j]$范围内的质数的函数值之和，所以仍然可以省掉这一部分的计算，将复杂度降到$O\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$。

对比一下这两种实现方法，主要的区别在于枚举质因子的顺序，可以根据具体情况选择。

6.6 扩展

回顾一下这个算法中的几步主要转化：

- 根据引理 6.1 ，利用积性将每个数的质因子分为两部分来考虑。
- 根据引理 4.2 和引理 5.1 ，将递推的第二维状态缩小到$O(\sqrt{n})$种。
- 分析递推的性质，通过省去$p_i{}^2 > j$时的计算将复杂度降到$O\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$。

上述的几步实际上对所有数完成了质因子拆分，并且在递推的时候质因子是有序加入的，所以这个算法可以完成一些筛法的功能。

6.6.1 例题五

例5 (UR13C Sanrd)**.**

求n以内所有合数的次大质因子之和，在这里次大质因子的定义如下：

设$n = \prod_{i=1}^{m} p_i$，其中p_i均为质数且$p_i \le p_{i+1}$。
次大质因子为p_{m-1}，若$m < 2$则不计算。

同样把质因子以$\sqrt{n}$为阈值分成两部分来考虑，并且根据引理 6.1，$x \le n$的次大质因子一定不超过$\sqrt{n}$。

回顾 6.4 节和 6.5 节中的第一种算法，构成每个数x的流程为：先升序加入x不超过$\sqrt{n}$的质因子，最后再计算x大于$\sqrt{n}$的质因子，即在整个过程中，x的质因子是被升序枚举的。

在计算不超过$\sqrt{n}$的质因子时，可以认为每一个状态中维护的是一个数集的信息，一次转移是将这个数集中所有数都乘上当前枚举的质因子，得到一个新集合，并入目标状态。由于质因子是有序枚举的，故每个状态中都不会有数被重复计算，并且还保证状态之间的数集不相交。

容易发现，转移时新加入的质因子一定是新生成的数的最大质因子，被转移状态的最大质因子就是次大质因子，所以只要记录每个状态表示的数集的最大质因子之和，就可以得到次大质因子。

6.7 总结

这个算法是本文主要想要展示的内容，解决了形式较为一般的积性函数求和问题，对于原函数的主要约束如下：

- 需要被求和的函数$F(n)$为积性函数。
- $G(p)$可以分解为若干完全积性函数之和，由于递推数组的初始化需要，这些完全积性函数的求和要能快速完成，一个简单的例子为：$G(p)$是关于p的低阶多项式。
- 由于第 6.5 节中的转移需要，转移系数$G(p)$和$T(p^c)$的计算要能快速完成。

6.4 和 6.5 节中的递推转移都可以使用滚动数组，质数也只需要预处理$\sqrt{n}$范围以内的，所以空间复杂度为$O(\sqrt{n})$。

这个算法可以帮助我们$O\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$对大多数积性函数完成求和，在那些不具备特殊性质或推导难度较大的积性函数求和问题中有很大的优势。

感谢

感谢计算机协会提供学习和交流的平台。
感谢绍兴一中的陈合力老师、董烨华老师多年来给予的关心和指导。
感谢清华大学的俞鼎力、董宏华、张恒捷、王鉴浩学长对我的帮助。
感谢毛啸同学与我讨论这个算法，并提供了一种简洁的实现方法。
感谢其他对我有过帮助和启发的老师和同学。

参考文献

[1] 贾志鹏, 线性筛法与积性函数, WC2012营员交流

[2] 吉如一, http://jiruyi910387714.i11r.com/posts/195270.html, jiry_2's Blog。

[3] Project Euler forum, Problem 521

网络流的一些建模方法

东营市胜利第一中学 姜志豪

摘 要

网络流在信息学竞赛中有着广泛的应用，很多网络流模型的建立方法十分巧妙。本文对一些比较常用的网络流建模方法进行了总结。

1 引言

网络流(network-flows)是一种类比水流的解决问题方法，在信息学竞赛中应用广泛。常见的网络流问题有最大流、最小费用最大流、有上下界的网络流等。网络流问题的巧妙之处往往不在于算法实现过程，而在于网络流的建模方法。本文对一些比较常用的网络流建模方法进行了总结，分为最大流建模、最小割建模、费用流建模和流量平衡思想四部分。希望能够给参加信息学竞赛的同学带来或多或少的帮助。

2 从最大流角度建模

一般来说，从最大流角度进行建模最直观。往往是用一条源点 $S \to$ 汇点 T 的流来表示一种方案。例如用最大流求二分图的最大匹配时，一条 $S \to T$ 的流就表示一个匹配。

2.1 建模举例

例 1. *(士兵占领)*

有一个 $n \times m$ 的棋盘，有的格子是障碍。现在你要选择一些格子来放置一些士兵，一个格子里最多可以放置一个士兵，障碍格里不能放置士兵。我们称这

些士兵占领了整个棋盘，当满足第i行至少放置了 r_i 个士兵, 第j列至少放置了 c_j 个士兵。现在你的任务是使用最少个数的士兵来占领整个棋盘。

$1 \le n, m \le 100$

最坏情况是先在每一行安排上指定数量的士兵，再在每一列安排上指定数量的士兵。这个方案中，每个士兵的贡献是1。

有些士兵可以既对行有贡献，又对列有贡献，贡献是2。这类士兵越多，需要的士兵总数就越少。考虑使这一类士兵尽可能多。第 i 行这类士兵的数量不能超过 r_i ，否则就有士兵对这一行没有贡献。同样，第 i 列这类士兵的数量不能超过 c_i 。考虑建图：

每一行建立一个点 A_i ，与源点 S 相连，容量是 r_i 。每一列建立一个点 B_i ，与汇点 T 相连，容量是 c_i 。

若第 i 行第 j 列可以放置士兵，就从 A_i 向 B_j 连一条容量为1的边。

这样建图后，一个贡献是2的士兵对应着一条 $S \to T$ 的流。边的容量限制了一个格子最多放置一个士兵，并且每行每列贡献是2的士兵的数量不超出要求。

求最大流，也就是贡献是2的士兵的最大数量，从而求出最小总士兵数。

例 2. *(Dining)*

有 f 种食物和 d 种饮料，每种食物或饮料只能供一头牛享用，且每头牛只享用一种食物和一种饮料。现在有 n 头牛，每头牛都有自己喜欢的食物种类列表和饮料种类列表，问最多能使多少头牛同时享用到自己喜欢的食物和饮料。

$1 \le n, f, d \le 100$

考虑用一条 $S \to T$ 的流来表示满足一头牛的要求，可以得到建图方式：

每种食物建立一个点 A_i ，与 S 相连，容量是1。每种饮料建立一个点 B_i ，与 T 相连，容量是1。

每头牛建立两个点 C_i 、 D_i ， C_i 与 D_i 之间连一条容量为1的边。

若第 i 头牛喜欢第 j 种食物，就将 C_i 与 A_j 相连，容量是1。

若第 i 头牛喜欢第 j 种饮料，就将 D_i 与 B_j 相连，容量是1。

用两个点来表示一头牛，中间连容量为1的边，是为了限制一头牛只被满足一次。

有一些题目的建模需要对问题进行分析，将问题简化或转化成可以用网络流解决的问题。我们来看下面这个问题。

例 3. *(Collector's Problem)*

*Bob*和他的朋友从糖果包装里收集贴纸。*Bob*和他的朋友总共 n 人。共有 m 种不同的贴纸。

每人手里都有一些（可能有重复的）贴纸，并且只跟别人交换他所没有的贴纸。贴纸总是一对一交换。

*Bob*比这些朋友更聪明，因为他意识到只跟别人交换自己没有的贴纸并不总是最优的。在某些情况下，换来一张重复贴纸更划算。

假设*Bob*的朋友只和*Bob*交换（他们之间不交换），并且这些朋友只会出让手里的重复贴纸来交换他们没有的不同贴纸。你的任务是帮助*Bob*算出他最终可以得到的不同贴纸的最大数量。

$2 \le n \le 10, 5 \le m \le 25$

Bob的朋友只会出让手里的重复贴纸来交换他们没有的不同贴纸。所以，对于Bob的某个朋友Friend，Bob只能把一种Friend没有的贴纸给他，并且一种最多给一次。Friend只会把他手里重复的贴纸给Bob，如果Friend有 i $(i \ge 2)$ 张某种贴纸，他至多给Bob $(i-1)$ 张。

那么，Bob的朋友的作用是，将Bob手中的贴纸 X 变成另一种贴纸 Y。

可以进行如下建图：

对每种贴纸 i 建立点 A_i。源点 S 向 A_i 连边，容量为Bob拥有的贴纸 i 的数量。A_i 向汇点 T 连边，容量为1。

对Bob的每个朋友 j 建立点 B_j。若朋友 j 没有贴纸 i，就从 A_i 向 B_j 连边，容量为1。若朋友 j 有 k $(k \ge 2)$ 张贴纸 i，就从 B_j 向 A_i 连边，容量为 $k-1$。

我们用 A 表示点 A_i 的集合，用 B 表示点 B_i 的集合。

一条 $S \rightarrow T$ 的流，是先从 S 到 A，表示Bob最初拥有的某种贴纸。然后经过若干次（可能是0次）到 B 再到 A 的过程，表示的是和朋友进行了交换。最后从 A 到 T，表示交换结束后Bob手中的贴纸的种类。

2.2 小结

最大流构图的特点是直观容易理解。$S \rightarrow T$ 的流，有着实际的意义，表示方案或操作方式。

不过，最大流问题的变化也非常多。有些时候，需要认真分析问题，发现问题的实质，将问题简化或转化，才能够得出网络流模型。

3 从最小割角度建模

3.1 用容量为正无穷的边表示冲突

例 4. *(NOI2006, 最大获利)*

新的技术正冲击着手机通讯市场，对于各大运营商来说，这既是机遇，更是挑战。*THU*集团旗下的*CS&T*通讯公司在新一代通讯技术血战的前夜，需要做太多的准备工作，仅就站址选择一项，就需要完成前期市场研究、站址勘测、最优化等项目。在前期市场调查和站址勘测之后，公司得到了一共 n 个可以作为通讯信号中转站的地址，而由于这些地址的地理位置差异，在不同的地方建造通讯中转站需要投入的成本也是不一样的，所幸在前期调查之后这些都是已知数据：建立第 i 个通讯中转站需要的成本为 p_i。另外公司调查得出了所有期望中的用户群，一共 m 个。关于第 i 个用户群的信息概括为 a_i, b_i 和 c_i：这些用户会使用中转站 a_i 和中转站 b_i 进行通讯，公司可以获益 c_i。*THU*集团的*CS&T*公司可以有选择的建立一些中转站（投入成本），为一些用户提供服务并获得收益（获益之和）。那么如何选择最终建立的中转站才能让公司的净获利最大呢？（净获利 = 获益之和 – 投入成本之和）

$1 \le n \le 5000, 1 \le m \le 50000$

将中转站、用户群看成点。建立点 A_i 表示中转站 i，从源点 S 向 A_i 连接容量为 p_i 的边，割这条边表示建立中转站 i，需要 p_i 的费用。建立点 B_i，表示第 i 个用户群，从 B_i 向汇点 T 连接容量为 c_i 的边，割这条边表示不满足这个用户群的要求，损失了 c_i 的收益。

若第 i 个用户群会使用中转站 j，那么 S 向 A_j 连接的边不能与 B_i 向 T 连接的边同时保留，所以可以从 A_j 向 B_i 连接一条容量为正无穷的边，这样就限制了那两条边不会同时保留。

所有用户群的获益之和减去最小割就是最大净获利。

在这道题目中，借助容量为正无穷的边，使冲突的收益无法同时保留。容量为正无穷的边还可以限制其他的冲突，我们看一下下面这道题目。

例 5. *(Codechef Dec14, Course Selection)*

铃现在正在大学学习。

课业计划共包含 n 项课程，每项课程都需要在 m 个学期里的某一个完成。

一些课程有前置课程，a 是 b 的前置课程表示课程 a 的完成的学期要在课程 b 完成的学期的前面。共有 k 组课程顺序要求。

对于课程 i，在不同学期完成的得分不同。令 $x_{i,j}$ 表示课程 i 在学期 j 完成的得分。若 $x_{i,j}$ 为 -1，表示学期 j 中没有开设课程 i。

计算铃各个课程分数的平均值的最大值。

$1 \le n, m \le 100, 0 \le k \le 100, -1 \le x_{i,j} \le 100$

各课程分数的平均值，就是各课程分数之和除以课程数。所以就是要最大化课程分数之和。

得分最大就是扣分最小，可以假设满分是100分，最小化扣分之和。用 $y_{i,j}$ 表示课程 i 在学期 j 完成的扣分。$x_{i,j} = -1$ 时，$y_{i,j}$ 的值是正无穷，否则 $y_{i,j} = 100 - x_{i,j}$。

可以进行如下建图：

对于课程 i、学期 j，建立点 (i, j)。

从源点 S 连向 $(i, 1)$ 一条边，容量为 $y_{i,1}$。

对于 $2 \le j \le m$，从 $(i, j-1)$ 连向 (i, j) 一条边，容量为 $y_{i,j}$。

从 (i, m) 连向汇点 T 一条边，容量为正无穷。

割去连入 (i, j) 的边，表示课程 i 在学期 j 学习。

这样求出的最小割，就是不考虑前置课程要求的情况下的最小扣分。

若 a 是 b 的前置课程，那么 a 的割边位置要在 b 的割边位置的前面。

从 S 向 $(b, 1)$ 连一条容量为正无穷的边。

对于 $2 \le j \le m$，从 $(a, j-1)$ 向 (b, j) 连一条容量为正无穷的边。

这样建图，就能使方案满足所有前置课程的要求。

容量为正无穷的边不会出现在最小割中。所以可以借助容量为正无穷的边，限制“S 与边的起点连通”、“边的终点与 T 连通”这两个条件不会同时满足。从而使问题中的一些冲突情况在网络流图中表现出来。

3.2 从两点关系的角度进行最小割建模

例 6. *(happiness)*

高一一班的座位表是个 $n \times m$ 的矩阵，经过一个学期的相处，每个同学和前后左右相邻的同学互相成为了好朋友。

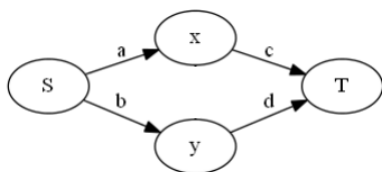
这学期要分文理科了，每个同学选择理科的喜悦值是 p_i，选择文科的喜悦值是 q_i。

一对好朋友如果同时选理科，他们又将共同获得喜悦值 v_{i1}。一对好朋友如果同时选文科，他们又将共同获得喜悦值 v_{i2}。

如何分配可以使得全班的喜悦值总和最大。

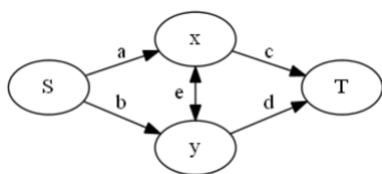
$1 \le n, m \le 100$

对每个同学建立一个点 x，点 x 向源点 S 连一条边，向汇点 T 连一条边，分别表示选择文科或理科。如下图所示。



每个同学选择文理科的收益可以最后再加进来，所以暂且不考虑。

若两个同学 x、y 相邻，并且他们没有选择相同的科类，就会少获得收益，或者认为是有损失。我们可以在 x、y 之间连边，来表示这种损失。



如上图，我们假设保留与 S 相连的边表示理科，保留与 T 相连的边表示文科。用 v_1 表示 x、y 同时选择理科的收益，用 v_2 表示 x、y 同时选择文科的收

益。那么，x、y的选择共有四种情况，情况及损失如下。

$$a + b = v_1 \quad (x、y \text{都选择文科}) \tag{1}$$

$$c + d = v_2 \quad (x、y \text{都选择理科}) \tag{2}$$

$$a + d + e = v_1 + v_2 \quad (x \text{选择文科，} y \text{选择理科}) \tag{3}$$

$$b + c + e = v_1 + v_2 \quad (x \text{选择理科，} y \text{选择文科}) \tag{4}$$

$$(3) + (4) - (1) - (2) = 2 \cdot e = v_1 + v_2$$
$$\therefore e = \frac{v_1 + v_2}{2}$$
$$a = b = \frac{v_1}{2}$$
$$c = d = \frac{v_2}{2}$$

解出边的容量后，再考虑上每个同学选择文理科的收益，可以这样来建图：

对于每个同学，建立一个点。从 S 向这个点连一条边，容量是这个同学选理科的收益，再加上他与所有相邻同学都选理科的共同收益和的一半。从这个点向 T 连一条边，容量是这个同学选文科的收益，再加上他与所有相邻同学都选文科的共同收益和的一半。

对于相邻两个同学 x、y，在 x、y之间连一条双向边，容量是他们都选文科的共同收益与他们都选理科的共同收益的平均数。

所有收益的和减去最小割就是最大喜悦值之和。

这个问题中，收益至多涉及两个人。可以先考虑两个人之间的关系，然后把所有关系综合起来，进行建图。

例 7. *(Google Code Jam 2008 Final E, The Year of Code Jam)*

*Sphinny*正在看新一年的日程表。*Sphinny*生活的世界与我们的世界有所不同，那个世界里一年有 n 个月，每个月恰有 m 天。

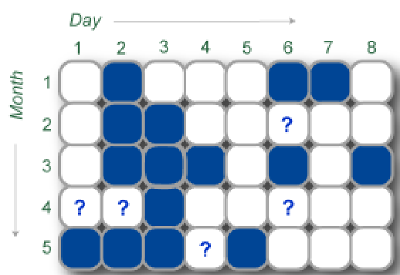
她将这一年的每一天都用以下三种方式之一在日程表上打标记：

*1.*白色：这一天她将不参加竞赛。

*2.*蓝色：这一天她将参加一场竞赛。

3.问号：这一天有预定好的竞赛，但她还没有决定好是否参加。

下面的图片是一张5个月，每个月有8天的日程表。



Sphinny想最大化所有竞赛的喜悦值之和。一场竞赛的喜悦值的计算方式是：

1.初始为4。

2.每有相邻（有公共边）的一天也要参赛，喜悦值减1。

Sphinny决定把每一个问号标记都改为白色标记或蓝色标记，并最大化总喜悦值。帮助Sphinny求出总喜悦值最大是多少。

$1 \le n, m \le 50$

这个题目和上一个题目类似，我们从两点之间的关系入手，建立最小割模型。

初始化 ans 为所有问号标记都为白色标记时的喜悦值，再将 ans 加上问号标记个数的4倍。那么：

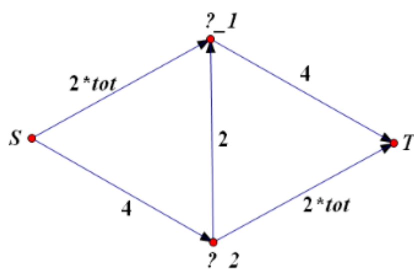
1.问号标记变为白色标记， ans 要减4。

2.问号标记变为蓝色标记，若它周围有 tot 个一开始就是蓝色的标记， ans 要减去 $2 \cdot tot$ 。

3.两个相邻的问号标记 ?_1 和 ?_2 都变为蓝色标记， ans 要减去2。

我们尝试像上一个题目一样求边的容量，但难以求出。

可以使用下图这样的建图方式。



其中，?_1 保留与 S 的连边，表示变成白色标记；保留与 T 的连边，表示变成蓝色标记。?_2 保留与 S 的连边，表示变成蓝色标记；保留与 T 的连边，表示变成白色标记。只有相邻两个问号标记都变为蓝色标记时，连接这两个问号标记的边才会被割掉，使 ans 减小2。

将日程表黑白染色，相邻的格子颜色不同，就会形成二分图。一种点用 ?_1 的连接方式，另一种点用 ?_2 的连接方式。

ans 减去最小割即为最大总喜悦值。

3.3 小结

最小割可以解决一些存在收益冲突的收益最大化问题。在网络流图中，任何一条 $S \rightarrow T$ 的路径上，都需要割掉至少一条边。利用这个性质，可以使收益的冲突在网络流模型中得到体现。

4 从费用流角度建模

给网络流增加一个因素：费用。这就变成了费用流问题。

费用流相比于最大流，能够解决更多的问题。

4.1 建模举例

例 8. *(序列)*

给出一个长度为 n 的正整数序列 A_i。选出一个子序列，使得原序列的任意一个长度为 m 的连续子序列中，被选出的元素数量不超过 k 个。

最大化选出的子序列中元素的和。

$1 \leq n \leq 1000, 1 \leq m, k \leq 100$

一个长度为 m 的连续子序列中至多选择 k 个元素。可以将问题转化一下，不是选择1次，而是选择k次，但在同一次选择中，任意一个长度为 m 的连续子序列中至多选择一个元素。

不难证明，原问题的一种合法方案，在转化后的问题中有等价的合法方案。转化后的问题中的一种合法方案，在原问题中也合法。

转化后的问题就比较好解决了，可以这样来建图：

建立源点 S 、汇点 T 。对序列中的第 i 个元素建立一个点 P_i 。

S 向 P_1 连一条容量为 k ，费用为0的边。

$P_i\ (1 \le i < n)$ 向 P_{i+1} 连一条容量为 k ，费用为0的边。

P_n 向 T 连一条容量为 k ，费用为0的边。

$P_i\ (1 \le i \le n - m)$ 向 P_{i+m} 连一条容量为1，费用为 A_i 的边。

$P_i\ (n - m + 1 \le i \le n)$ 向 T 连一条容量为1，费用为 A_i 的边。

求最大费用最大流，最大费用就是最大元素和。

这道题目需要将原问题转化成更容易进行建模的问题，然后用网络流解决。

例 9. *(WC 2007, 剪刀石头布)*

在一些一对一游戏的比赛（如下棋、乒乓球和羽毛球的单打）中，我们经常会遇到A胜过B，B胜过C而C又胜过A的有趣情况，不妨形象的称之为剪刀石头布情况。有的时候，无聊的人们会津津乐道于统计有多少这样的剪刀石头布情况发生，即有多少对无序三元组(A, B, C)，满足其中的一个人在比赛中赢了另一个人，另一个人赢了第三个人而第三个人又胜过了第一个人。注意这里无序的意思是说三元组中元素的顺序并不重要，将(A, B, C)、(A, C, B)、(B, A, C)、(B, C, A)、(C, A, B)和(C, B, A)视为相同的情况。

有 n 个人参加一场这样的游戏的比赛，赛程规定任意两个人之间都要进行一场比赛：这样总共有 $\frac{n(n-1)}{2}$ 场比赛。比赛已经进行了一部分，我们想知道在极端情况下，比赛结束后最多会发生多少剪刀石头布情况。即给出已经发生的比赛结果，而你可以任意安排剩下的比赛的结果，以得到尽量多的剪刀石头布情况。

$1 \le n \le 100$

直接求(A, B, C)构成剪刀石头布情况的数量不好求，可以考虑(A, B, C)不构成剪刀石头布的情况。

如果(A, B, C)不构成剪刀石头布的情况，那么A、B、C中，一个人赢了2场，一个人赢了1场，一个人赢了0场。

若第 i 个人赢了 w_i 场，那么 $\sum_i \frac{w_i(w_i-1)}{2}$ 就是不构成剪刀石头布情况的(A, B, C)的数量。

最大化剪刀石头布情况数量，就是最小化不满足的数量，就是最小化 $\sum_i \frac{w_i(w_i-1)}{2}$ 。

可以建出网络流模型：

建立源点 S 和汇点 T 。

对第 i 个人建立点 P_i ， P_i 向 T 连一条容量为 n 的边。

对未进行的 i 、 j 之间的比赛建立点 $C_{i,j}$ ， S 向 $C_{i,j}$ 连一条容量为1的边， $C_{i,j}$ 向 P_i 、 P_j 各连一条容量为1的边。

这样求出的最大流就是一种合法方案。 P_i 与 T 相连的边的流量，就是第 i 个人在这些未进行的比赛中赢的次数。

为了最小化 $\sum_i \frac{w_i(w_i-1)}{2}$ ，我们让这个值以费用的形式在网络流图中体现出来。

费用要加在 P_i 与 T 相连的边上。但每当这条边的流量加1时，费用不是不变的，而是一次比一次大。

可以把这条边拆成很多条，每条边的容量是1，费用是流量加1时的费用。

例如，若流量是 x 时，这条边的总费用是 $\frac{x(x-1)}{2}$ ，就把这条边拆成许多条容量为1的边，费用分别是 $0, 1, 2, ...$ 。因为求的是最小费用最大流，所以会先增广费用小的边。这样，流量是 x 时，费用就是 $\frac{x(x-1)}{2}$ 了。

这道题目，用到了补集转化的思想。同时，如果边的费用不固定，可以通过拆边来解决。

4.2 小结

费用流比最大流能够解决更多类型的问题，也更加灵活多变。在很多时候，需要先将原问题进行转化，再通过费用流解决问题。

5 流量平衡思想

有一些问题难以通过直观的方法直接进行网络流建模，但我们可以得到问题中变量之间的一些关系，一般可以写成若干个等式。

在网络流图中，除源点、汇点外，其他顶点都满足流量平衡。流量平衡也可以写成等式的形式。

有些问题中的等式，可以构造出网络流图，将等式以网络流图中流量平衡的形式表示出来。

下面的两个例子，就用到了这种思想。

5.1 建模举例

例 10. *(NOI2008, 志愿者招募)*

中奥成功后，布布经过不懈努力，终于成为奥组委下属公司人力资源部门的主管。布布刚上任就遇到了一个难题：为即将启动的奥运新项目招募一批短期志愿者。经过估算，这个项目需要 n 天才能完成，其中第 i 天至少需要 a_i 个人。布布通过了解得知，一共有 m 类志愿者可以招募。其中第 i 类可以从第 s_i 天工作到第 t_i 天，招募费用是每人 c_i 元。新官上任三把火，为了出色地完成自己的工作，布布希望用尽量少的费用招募足够的志愿者，但这并不是他的特长！于是布布找到了你，希望你帮他设计一种最优的招募方案。

$1 \le n \le 1000, 1 \le m \le 10000$

假设共3天，第 i 天招募 p_i 人。

共有三类志愿者，分别是：

从第1天到第3天，费用为 c_1，招募了 b_1 人。

从第2天到第3天，费用为 c_2，招募了 b_2 人。

从第1天到第2天，费用为 c_3，招募了 b_3 人。

可以列出不等式组：

$$p_1 = b_1 + b_3 \ge a_1$$
$$p_2 = b_1 + b_2 + b_3 \ge a_2$$
$$p_3 = b_1 + b_2 \ge a_3$$

设第 i 天招募的志愿者人数超出最少人数 d_i 人，其中 $d_i \ge 0$。可以得到等式：

$$p_1 = b_1 + b_3 = a_1 + d_1$$
$$p_2 = b_1 + b_2 + b_3 = a_2 + d_2$$
$$p_3 = b_1 + b_2 = a_3 + d_3$$

相邻两个等式相减，得到新的等式组：

$$
\begin{aligned}
p_1 &= b_1 + b_3 = a_1 + d_1 \\
p_2 - p_1 &= b_2 = a_2 - a_1 + d_2 - d_1 \\
p_3 - p_2 &= -b_3 = a_3 - a_2 + d_3 - d_2 \\
-p_3 &= -b_1 - b_2 = -a_3 - d_3
\end{aligned}
$$

整理一下，得到：

$$
\begin{aligned}
b_1 + b_3 - a_1 - d_1 &= 0 \\
b_2 - a_2 + a_1 - d_2 + d_1 &= 0 \\
-b_3 - a_3 + a_2 - d_3 + d_2 &= 0 \\
-b_1 - b_2 + a_3 + d_3 &= 0
\end{aligned}
$$

网络流图中，除了源点、汇点，其他顶点都满足流量平衡。若流出的流量记为正，流入的流量记为负，那么流入流出流量的代数和为0。可以对上面的每一个等式各建立一个点，等式表示的就是这个点的流量平衡。

对于每一个变量 b_i、 d_i，都恰好在两个等式中出现了，而且在一个等式中为正，一个等式中为负。网络流图中，一条连接 x、 y 的边，在 x、 y 点的流量平衡等式中各出现一次，一次为正，一次为负。所以，每一个变量 b_i、 d_i，都可以作为网络流图中的一条边。

对于常量 a_i，也在两个等式中出现，一次为正，一次为负。为正时，表示流出，可以从这个点向汇点连边。为负时，表示流入，可以从源点向这个点连边。

需要最小化 $\sum b_i \cdot c_i$，可以将这个值以费用的形式表示出来。

可以这样来建图：

建立源点 S 和汇点 T。

建立 $n+1$ 个点，点编号为 $1 \sim n+1$，表示 $n+1$ 个等式。

第 $i\,(2 \le i \le n+1)$ 个点向第 $i-1$ 个点连一条容量为正无穷，费用为0的边。

第 i 类志愿者可以从第 s_i 天工作到第 t_i 天，费用是每人 c_i 元。那么就从第 s_i 个点向第 t_i+1 个点连一条容量为正无穷，费用为 c_i 的边。

对于第 i 个点，若 $a_{i-1} - a_i$ 为正数，就从这个点向 T 连一条容量为 $a_{i-1} - a_i$，费用为0的边。否则就从 S 向这个点连一条容量为 $a_i - a_{i-1}$，费用为0的边。在这里，我们认为 a_0 和 a_{n+1} 的值为0。

由于 a_i 是常量，所以 S 连出的边、连入 T 的边都必须满流。原图的最大流满足这个条件。

为了最小化费用，所以需要求最小费用最大流。

例 11. *(World Finals 2011, Chips Challenge)*

在一个 $n \times n$ 网格里放部件。其中一些格子已经放了部件，一些格子不能放部件，其他格子可以放也可以不放。要求第 x 行的总部件数等于第 x 列的总部件数。为了保证散热，任意行/列的部件不能超过总部件数的 $\frac{A}{B}$。求最多能放多少部件。

$1 \le n \le 40$

用 $a_{i,j}$ 表示第 i 行第 j 列的格子是否放部件，1表示放，0表示不放。用 t_i 表示第 i 行的部件总数，同时也是第 i 列的部件总数。可以列出等式：

$$t_i = a_{i,1} + a_{i,2} + ... + a_{i,n} \quad (1 \le i \le n)$$
$$t_i = a_{1,i} + a_{2,i} + ... + a_{n,i} \quad (1 \le i \le n)$$

整理一下，得到：

$$a_{i,1} + a_{i,2} + ... + a_{i,n} - t_i = 0 \quad (1 \le i \le n)$$
$$t_i - a_{1,i} - a_{2,i} - ... - a_{n,i} = 0 \quad (1 \le i \le n)$$

还有一个限制，是任意行/列的部件不能超过总部件数的 $\frac{A}{B}$。枚举总部件数 tot，就能确定出每行、每列最多放的部件数 $maxt$ 了。求出这种情况下最多放的总部件数 ans。若 $ans \ge tot$，说明方案合法。

通过枚举，确定 $maxt$ 后，可以这样建图：

建立点 A_i 表示第 i 行部件数量的计算式$a_{i,1} + a_{i,2} + ... + a_{i,n} - t_i = 0$。

建立点 B_i 表示第 i 列部件数量的计算式$t_i - a_{1,i} - a_{2,i} - ... - a_{n,i} = 0$。

点 B_i 向点 A_i 连接流量上界为 $maxt$，下界为0，费用为1的边。

若 $a_{i,j}$ 必须为1，就从点 A_i 向点 B_j 连接流量上下界均为1，费用为0的边。

若 $a_{i,j}$ 可以为1，也可以为0，就从点 A_i 向点 B_j 连接流量上界为1，下界为0，费用为0的边。

这个网络流图的一个可行流就是一种合法方案。最大费用可行流就是最优方案。

对所有的 $maxt$ 都求出最优方案，取所有合法方案中的最优解作为最终方案。

5.2 小结

将问题中变量之间的关系以等式的形式表示出来，然后与网络流图中描述流量平衡的等式进行联系。可以将原问题对应到网络流图上。

多数网络流模型都可以用流量平衡思想来理解。所以可以用流量平衡的思想解决大部分网络流问题。但流量平衡思想的缺点是不够直观，有时会把问题变复杂。不过在一些直观上难以理解或难以想到的问题上，流量平衡的思想能够发挥很大的作用。

6 总结

网络流的建模角度非常多，如最大流角度、最小割角度、费用流角度、平衡流角度等。

网络流建模时会用到的技巧也有很多，如拆点、拆边、建分层图等。

网络流建模时，有时需要分析问题，将原问题转化为比较容易进行建模的问题。

有时，网络流建模需要和其他算法结合才能解决问题，如二分、补集转化等。

总之，网络流建模的内容十分丰富。

希望有同学能够从本文中获得灵感，发现新的、更加巧妙的网络流建模方法。

7 参考文献

[1] 刘汝佳、陈锋，《算法竞赛入门经典——训练指南》，清华大学出版社

[2] 胡伯涛，《最小割模型在信息学竞赛中的应用》

浅谈线性规划与对偶问题

福建省福州第一中学 董克凡

摘 要

线性规划是一种重要的数学模型，有着广泛的应用。信息学竞赛中，有许多问题均能直观地用线性规划表示。本文从线性规划的角度来探讨这一类问题的性质，通过线性规划的对偶原理实现问题转化，从而使问题得到更有效的解决。

1 引言

在信息学竞赛中，以网络流为代表的一系列线性规划问题的解法往往灵活多变，本文中通过将这些问题表示为线性规划的方式来更深层地理解这类问题。希望能给所有参加信息学竞赛的同学或多或少的启发。

本文第一部分着重介绍线性规划的定义，求解一般线性规划的单纯形方法，以及如何将问题表示为线性规划模型。第二部分着重介绍对偶性，通过实例，叙述从线性规划模型为切入点，利用线性规划的对偶性，实现问题的模型转换，从而高效解决问题的一种新思路。

2 线性规划

2.1 定义

在最优化问题中，有一类问题满足它的限制条件以及目标函数都是自变量的线性函数，我们称这一类问题为线性规划问题。为了叙述方便，在这里首先给出一些定义。

定义2.1. 已知一组实数$a_1, a_2, \ldots, a_n$，以及一组变量$x_1, x_2, \ldots, x_n$，在这些变量上的一个线性函数定义为

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} a_i x_i$$

等式$f(x_1, x_2, \ldots, x_n) = b$，不等式$f(x_1, x_2, \ldots, x_n) \leq b, f(x_1, x_2, \ldots, x_n) \geq b$统称为线性约束。

线性规划问题要求最大化或最小化一个受限于一组有限的线性约束的线性函数。

称满足所有限制条件的解$x_1, x_2, \ldots, x_n$为可行解，使目标函数达到最优的可行解为最优解，所有可行解构成的区域为解空间。

2.2 线性规划的性质

为了描述线性规划的性质与算法，这里需要使用一种更规范化的形式来描述线性规划问题。

定义2.2. 标准型

标准型线性规划要求满足如下形式

$$\begin{aligned}
\text{最大化} \quad & \sum_{j=1}^{n} c_j x_j & \\
\text{满足约束} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, & i = 1, 2, \ldots, m \\
& x_j \geq 0, & j = 1, 2, \ldots, n
\end{aligned}$$

容易发现，经过一些简单的变换，所有线性规划问题都可以用标准型描述。

若目标函数要求取最小值，那么可以对其取相反数变成取最大值。对于限制条件$f(x_1, x_2, \ldots, x_n) = b$，可以用两个不等式$f(x_1, x_2, \ldots, x_n) \leq b, -f(x_1, x_2, \ldots, x_n) \leq -b$描述，对于限制条件$f(x_1, x_2, \ldots, x_n) \geq b$，可以用不等式$-f(x_1, x_2, \ldots, x_n) \leq -b$描述。对于无限制的变量$x$，可以将其拆为两个非负变量$x_0, x_1$，使得$x = x_0 - x_1$。

标准型也可以用矩阵表示：

$$
\begin{array}{lll}
\text{最大化} & \mathbf{c}^T\mathbf{x} & \\
\text{满足约束} & A\mathbf{x} \le \mathbf{b} & \quad (2.2.1) \\
 & \mathbf{x} \ge 0 &
\end{array}
$$

其中向量$\mathbf{x} \le \mathbf{y}$，当且仅当对于这两个向量的每一维$x_i, y_i$，都有$x_i \le y_i$

标准型的优点是简洁明了，而为了求解线性规划，需要所有的约束条件都是等式，这样可以避免在处理负系数时产生的不等号方向改变的问题，所以下面介绍线性规划的另一种表示方法——松弛型。

定义2.3. 松弛型

松弛型线性规划要求满足如下形式

$$
\begin{array}{lll}
\text{最大化} & \sum_{j=1}^{n} c_j x_j & \\
\text{满足约束} & x_{i+n} = b_i - \sum_{j=1}^{n} a_{ij} x_j, & i = 1, 2, \ldots, m \\
 & x_j \ge 0, & j = 1, 2, \ldots, n+m
\end{array}
$$

将标准型转化为松弛型是容易的。对于在标准型中的一个限制$f_i(x_1, x_2, \ldots, x_n) \le b_i$，新增加一个变量$x_{i+n} = b_i - f_i(x_1, x_2, \ldots, x_n)$，那么原来不等式的限制条件就转化为如下的等式限制：$x_{i+n} = b_i - f_i(x_1, x_2, \ldots, x_n) \ge 0$，转化之后仍然满足所有变量的取值仍为非负实数。

考察一个线性规划的解空间，这个解空间是多个线性不等式解空间的交。由于每一个线性不等式的解空间都是一个凸形区域[1]，故可以由归纳证明得，线性规划的解空间是一个凸形区域。这里证明从略。

由于解空间是一个凸形区域，也就是说局部最优解的值有且只有一个[2]，并且等于全局最优解的值。那么在这个区域内进行贪心，使用一个类似于爬山的算法来求解最优值的时候，就不需要担心算法在一个局部最优解中止，由这个性质，便引出了求解线性规划的一个一般性方法——单纯形法。

[1]凸形区域的一个直观定义是，区域内的任意两点连线上的点都属于这个区域

[2]这里暂不考虑线性规划无界（最优解可以取到无穷）的情况；局部最优解可能不止一个，有可能是某个与目标函数平行的边界，但是它们的值一定相同

3 单纯形

3.1 算法描述

单纯形法的基本思想是，通过变量的代换，实现在解空间内沿着边界朝着目标函数增大的方向移动。其核心操作是转轴（pivot）操作，即变量的代换。

首先做一些定义：

基变量：在松弛型等式左侧的所有变量。

非基变量：在松弛型等式右侧的所有变量。

一个线性规划的一组基变量和非基变量就隐含着一个基本解。即所有基变量的值为右侧的常数项[3]，所有非基变量的值为0，在单纯形算法中，我们只考虑基本解。

单纯形算法有两个主要的操作：转轴操作以及simplex操作。转轴操作的作用是，选择一个基变量x_B 以及一个非基变量x_N，将其互换（称这个非基变量为换入变量，基变量为换出变量），即用x_B以及其他非基变量代换x_N。具体来说，一开始有

$$x_B = b_i - \sum_{j=1}^{n} a_{ij} x_j$$

那么

$$x_N = (b_i - \sum_{j \neq N} a_{ij} x_j - x_B) / a_{iN}$$

将目标函数以及其余所有限制中的x_N 用这个等式替换，那么等式右侧就不会出现x_N这个变量，这就做到了互换基变量和非基变量。当然在选择x_N时要保证$a_{iN} \neq 0$

simplex操作即单纯形算法的主过程，从一个基本解出发，经过一系列的转轴操作，达到最优解。通过选择特定的换入变量以及换出变量，可以使得每一次转轴操作都能使目标函数增大，直到达到全局最优解。

[3]常数项为负数的情况将在下一节中考虑

下面用一个例子来说明单纯形的具体流程，考虑这一个线性规划：

$$\begin{aligned}
\text{最大化} \quad & 3x_1 + x_2 + 2x_3 \\
\text{满足约束} \quad & x_4 = 30 - x_1 - x_2 - 3x_3 \\
& x_5 = 24 - 2x_1 - 2x_2 - 5x_3 \\
& x_6 = 36 - 4x_1 - x_2 - 2x_3 \\
& x_j \geq 0, \qquad j = 1, 2, \ldots, 6
\end{aligned} \tag{3.1}$$

这个线性规划隐含的初始基本解为$(x_1, x_2, \ldots, x_6) = (0, 0, 0, 30, 24, 36)$，记目标函数为$z$，那么这时$z = 0$。

第一步选择x_1作为换入变量，因为在目标函数中x_1系数为正，增大x_1必然会增大z。由于所有变量非负的限制，x_1不可能无限制地增大，分别考虑三个限制条件，当x_1增大到30 以上，而x_2, x_3不变时，x_4就会变为负数，所以$x_1 \leq 30$。同样的，由后两个限制可以得到$x_1 \leq 12, x_1 \leq 9$。从中选择一个对x_1限制最紧的约束，即第三个约束，把第三个约束中的基本变量x_6作为换出变量。将限制三改写为$x_1 = 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4}$，代入目标函数以及其余约束，可以得到如下新形式的线性规划：

$$\begin{aligned}
\text{最大化} \quad & 27 + \frac{x_2}{4} + \frac{x_3}{2} - \frac{3x_6}{4} \\
\text{满足约束} \quad & x_4 = 21 - \frac{3x_2}{4} - \frac{5x_3}{2} + \frac{x_6}{4} \\
& x_5 = 6 - \frac{3x_2}{2} - 4x_3 + \frac{x_6}{2} \\
& x_1 = 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4} \\
& x_j \geq 0, \qquad j = 1, 2, \ldots, 6
\end{aligned}$$

这个新的线性规划仍然满足松弛型的形式，并且在这次迭代之后，目标函数增大至$z = 27$。

考虑转轴操作之后的基本解$(x_1, x_2, \ldots, x_6) = (9, 0, 0, 21, 6, 0)$与操作之前的基本解$(x_1, x_2, \ldots, x_6) = (0, 0, 0, 30, 24, 36)$，$x_4, x_5, x_6$在转轴操作前是基变量，它们的值的改变对目标函数的值没有影响，x_1在操作之后由非基变量变成了基变量，它的值由0增大为9，这就使得目标函数的值增大至27。虽然目标函数的值增大了，但是操作前后的两个线性规划是等价的。这时新的基本解为$(9, 0, 0, 21, 6, 0)$，如果将这个基本解代入(3.1)中，同样可以得到$z = 27$。

接下来，因为目标函数中仍然有变量的系数为正，那么也就是说目标函数可以继续被增大。这一次不妨选择x_3作为换入变量，那么换出变量就是x_5，因为x_5限制了$x_3 \leq \frac{3}{2}$，那么将第二个限制改写为$x_3 = \frac{3}{2} - \frac{3x_2}{8} - \frac{x_5}{4} + \frac{x_6}{8}$，代入目标函数以及其余的约束，得：

$$\begin{array}{ll} \text{最大化} & \frac{111}{4} + \frac{x_2}{16} - \frac{x_5}{8} - \frac{11x_6}{16} \\ \text{满足约束} & x_4 = \frac{69}{4} + \frac{3x_2}{16} + \frac{5x_5}{8} - \frac{x_6}{16} \\ & x_3 = \frac{3}{2} - \frac{3x_2}{8} - \frac{x_5}{4} + \frac{x_6}{8} \\ & x_1 = \frac{33}{4} - \frac{x_2}{16} + \frac{x_5}{8} - \frac{5x_6}{16} \\ & x_j \geq 0, \qquad j = 1, 2, \ldots, 6 \end{array}$$

此时这个松弛型对应的的基本解为$(\frac{33}{4}, 0, \frac{3}{2}, \frac{69}{4}, 0, 0)$。接下来，只能选择$x_2$作为换入变量，这是需要注意，由于在第一个限制条件中，x_2 的系数是正的，所以这一个限制条件并不限制x_2 增加到多少，故三个约束给出的上界分别为$\infty, 4, 132$。那么选择x_3作为换入变量，得：

$$\begin{array}{ll} \text{最大化} & 28 - \frac{x_3}{6} - \frac{x_5}{6} - \frac{2x_6}{3} \\ \text{满足约束} & x_4 = 18 - \frac{x_3}{2} + \frac{x_5}{2} \\ & x_2 = 4 - \frac{8x_3}{3} - \frac{2x_5}{3} + \frac{x_6}{3} \\ & x_1 = 8 + \frac{x_3}{6} + \frac{x_5}{6} - \frac{x_6}{3} \\ & x_j \geq 0, \qquad j = 1, 2, \ldots, 6 \end{array}$$

此时对应的基本解为$(8, 4, 0, 18, 0, 0)$。这时，目标函数中所有变量系数均为负，所以我们找到了一个局部最优点，由线性规划的性质，这就是全局最优解。

下面正式地给出单纯形算法的伪代码：其中$(A, \mathbf{b}, \mathbf{c})$意义与(2.2.1)同。

Algorithm 1 Simplex(A, b, c)

```
1: initialization(A, b, c)
2: while ∃e that c_e > 0 do
3:    find the index l that A_le > 0 and minimizes b_l/A_le
4:    if ∀l, A_le ≤ 0 then
5:       return Unbounded
6:    else
7:       pivot(A, b, c, l, e)
8:    end if
9: end while
```

其中，第4,5两行的作用是，若某一步选择的换入变量没有约束条件，那么这个线性规划就是无界的，否则，找到最紧的约束，执行转轴操作。initialization操作是初始化,即找到一组基本解，一般来说，若$b_i \geq 0, i = 1, \ldots, m$，那么可以略去这一操作，直接从$(0, 0, \ldots, 0, b_1, b_2, \ldots, b_m)$这一基本解开始。由于每一步转轴操作我们选择的是一个限制最紧的约束，所以在转轴操作之后，仍能保证$b_i \geq 0, i = 1, \ldots, m$。（证明从略）

初始化操作的基本思想是引入一个辅助线性规划:

$$
\begin{aligned}
\text{最大化} \quad & -x_0 \\
\text{满足约束} \quad & x_{i+n} = b_i - \sum_{j=1}^{n} a_{ij} x_j + x_0, \qquad i = 1, 2, \ldots, m \\
& x_j \geq 0, \qquad j = 0, 1, \ldots, n + m
\end{aligned}
$$

如果原线性规划存在一个可行解$(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_{n+m})$，那么$(0, \overline{x}_1, \overline{x}_2, \ldots, \overline{x}_{n+m})$就是辅助线性规划的一个可行解。又因为$x_0 \geq 0$的限制，这个可行解就是辅助线性规划的最优解。

另一方面，若求得辅助线性规划最优解$x_0 = 0$，那么直接将x_0从这个线性规划中删去不会对约束产生影响，然后再将目标函数替换为原目标函数，就得到了一个满足$b_i \geq 0$的与原线性规划等价的线性规划，这样就完成了初始化操作。

而辅助线性规划的初始解是容易构造的。由于x_0在每一个约束中的系数都为+1，那么我们把x_0作为换入变量，找到b_i的最小值b_l，把x_{l+n}作为换出变量执

行一次转轴操作。操作后，第l个约束变为

$$x_0 = -b_l + \sum_{j=1}^{n} a_{lj}x_j + x_{l+n}$$

满足$-b_l > 0$。对于其余约束变为

$$x_{i+n} = -b_l + b_i + \sum_{j=1}^{n} (a_{lj} - a_{ij})x_j + x_{l+n}, \quad i \neq l$$

由于$b_l \leq b_i$，所以$-b_l + b_i \geq 0$，故操作之后就满足辅助线性规划有一个可行的初始基本解，然后对其执行simplex即可。

举一个例子，比如下面的线性规划：

$$\begin{aligned} \text{最大化} \quad & 2x_1 - x_2 \\ \text{满足约束} \quad & x_3 = 2 - 2x_1 + x_2 \\ & x_4 = -4 - x_1 + 5x_2 \\ & x_j \geq 0, \qquad j = 1, 2, \ldots, 4 \end{aligned} \tag{3.2}$$

这个线性规划不存在一个可行的初始基本解，故对其进行初始化操作。引入辅助线性规划：

$$\begin{aligned} \text{最大化} \quad & -x_0 \\ \text{满足约束} \quad & x_3 = 2 - 2x_1 + x_2 + x_0 \\ & x_4 = -4 - x_1 + 5x_2 + x_0 \\ & x_j \geq 0, \qquad j = 0, 1, \ldots, 4 \end{aligned}$$

找到b_i的最小值是-4，那么以x_0为换入变量，x_4为换出变量执行转轴操作：

$$\begin{aligned} \text{最大化} \quad & -4 - x_1 + 5x_2 - x_4 \\ \text{满足约束} \quad & x_3 = 6 - x_1 - 4x_2 + x_4 \\ & x_0 = 4 + x_1 - 5x_2 + x_4 \\ & x_j \geq 0, \qquad j = 0, 1, \ldots, 4 \end{aligned}$$

这时就找到了一个可行的基本解(4, 0, 0, 6, 0)，接下来对其进行simplex操作，将x_2换入，x_0换出：

$$\begin{aligned}
\text{最大化} \quad & -x_0 \\
\text{满足约束} \quad & x_3 = \frac{14}{5} + \frac{4x_0}{5} - \frac{9x_1}{5} + \frac{x_4}{5} \\
& x_2 = \frac{4}{5} - \frac{x_0}{5} + \frac{x_1}{5} + \frac{x_4}{5} \\
& x_j \geq 0, \qquad j = 0, 1, \ldots, 4
\end{aligned}$$

这个松弛型是辅助线性规划的最优解，$x_0 = 0$，故可以将x_0从辅助线性规划中移除，考虑原目标函数$2x_1 - x_2$，由于这时x_2是基变量，故需要把x_2用非基变量替换，故目标函数需要设定为

$$2x_1 - \left(\frac{4}{5} - \frac{x_0}{5} + \frac{x_1}{5} + \frac{x_4}{5}\right)$$

所以得到的线性规划为：

$$\begin{aligned}
\text{最大化} \quad & \frac{4}{5} + \frac{9x_1}{5} - \frac{x_4}{5} \\
\text{满足约束} \quad & x_3 = \frac{14}{5} - \frac{9x_1}{5} + \frac{x_4}{5} \\
& x_2 = \frac{4}{5} + \frac{x_1}{5} + \frac{x_4}{5} \\
& x_j \geq 0, \qquad j = 1, 2, 3, 4
\end{aligned}$$

这个线性规划就存在可行的初始基本解，可以将其返回给simplex过程。

进行初始化操作同样不会改变原线性规划的本质，这时的基本解$(x_1, x_2, x_3, x_4) = (0, \frac{4}{5}, \frac{14}{5}, 0)$就满足(3.2)的所有约束。

例中恰好满足在最后得到的结果里x_0为非基变量，所以可以直接删去。若最终x_0为基变量，那么就需要再执行一次转轴操作，将x_0换出。由于$x_0 = 0$，所以任意选择一个在x_0等式右侧的非基变量将其换出即可。

3.2 单纯形法的时间复杂度

初始化操作与simplex操作的时间复杂度是同阶的，故只需考虑simplex的时间复杂度。

pivot操作即变量的代入及替换，单次操作的复杂度为$O(NM)$。但是pivot操作的执行次数并不是多项式的，不过在实际应用中，单纯形算法往往能十分快速地给出最优解。

需要注意的一点是，大部分的OI试题得出的限制条件都有一定的性质，而执行初始化操作会破坏这些性质，造成运行时间的无谓增加，且初始化操作较为繁琐，所以作者不建议大家在竞赛中使用，在本文的例题中我会介绍一些避免初始化操作的方法。

但这并不意味着线性规划问题不能在多项式时间内解决。椭球算法和内点算法均为解决线性规划的多项式时间算法。这里不加证明地给出这个定理：

定理3.1. 一般线性规划问题可以在多项式时间内解决。

但单纯形算法实现简单，并且实际运行时间并不明显比上述算法劣，不失为一种好的求解线性规划的方法。

4 将问题表示为线性规划

4.1 一道例题

例1. *precious stones*[4]

A、B两个人在洞穴中发现了n个石头，第i个石头对A来说价值为A_i，对B来说价值为B_i，石头是可以切割的，A、B认为石头的价值是均匀的（即石头的价值与其体积成正比）。现在A、B要分配这些石头，他们要求每个人得到的价值（一个人得到的价值是得到的石头在自己看来的价值）相同。求两人最大能得到的价值。

$n \le 50, 0 \le A_i, B_i \le 100$

设第i块石头有p_i分给了A，$1 - p_i$分给了B，那么两人得到的价值分别记为

$$V_a = \sum_{i=1}^{n} p_i A_i, V_b = \sum_{i=1}^{n} (1 - p_i) B_i$$

限制条件是$V_a = V_b$，要求最大化V_a。简单观察后，$V_a = V_b$这个限制可以放宽为$V_a \le V_b$，这是因为由于$A_i, B_i \ge 0$，所以当$V_a < V_b$时，可以将某部分B得到的

[4]试题来源：TCO08 round2 1000

石头分给A，A得到的价值不会减少，故增加（至少不减）目标函数的同时不违反限制条件，故得到下述线性规划：

$$\begin{aligned} \text{最大化} \quad & \sum_{i=1}^{n} p_i A_i \\ \text{满足约束} \quad & \sum_{i=1}^{n} p_i (A_i + B_i) \le \sum_{i=1}^{n} B_i \\ & p_i \le 1, \quad i = 1, 2, \ldots, n \\ & p_i \ge 0, \quad i = 1, 2, \ldots, n \end{aligned}$$

这样放宽之后，得到的线性规划便不需要执行初始化操作，接下来只需要直接运行单纯形算法即可。

4.2 最大流问题

在最大流问题中，除源汇外的每一个节点都需要满足“流量守恒”，即流入流量等于流出流量。每一条边需要满足流量不超过容量。用$c(u, v)$表示(u, v)这条边的容量，$f(u, v)$表示流量。为了使问题更加简便，我们可以人为增加一条从汇到源的容量为∞的边，这样使得流量守恒对于每一个节点均成立，而且$f(t, s)$就是从源到汇的流量大小，用V表示点集，E表示边集，得到如下线性规划：

$$\begin{aligned} \text{最大化} \quad & f(t, s) \\ \text{满足约束} \quad & f(u, v) \le c(u, v), \quad (u, v) \in E \\ & \sum_{v} f(u, v) = \sum_{v} f(v, u), \quad u \in V \\ & f(u, v) \ge 0, \quad (u, v) \in E \cup \{(t, s)\} \end{aligned}$$

4.3 最小费用流问题

在这个问题中，我们要求的只是最小费用，并不需要满足流量最大的限制，

这时每条边有费用$w(u,v)$。故有如下线性规划：

$$
\begin{array}{lll}
\text{最小化} & \sum_{(u,v)\in E} f(u,v)w(u,v) & \\
\text{满足约束} & f(u,v) \le c(u,v), & (u,v)\in E \\
& \sum_{v} f(u,v) - \sum_{v} f(v,u) = 0, & u \in V - \{s,t\} \\
& f(u,v) \ge 0, & (u,v) \in E \cup (t,s)
\end{array}
$$

可以发现，对于每一个变量$f(u,v)$，除了对变量本身的非负限制、容量限制之外，这个变量只出现在两个限制中，及u,v两个节点的流量守恒方程，且在这两个限制中$f(u,v)$的系数分别为$+1,-1$。

4.4 多物网络流问题

考虑这样一个问题：在最大流问题中，有n对源汇$(s_i,t_i), i=1,2,\dots,n$，每一对源汇对应的物品是不同的，但是它们共享同一个运输网络。对于每一对源汇(s_i,t_i)，要求他们之间的流量不小于d_i，判断是否有可行解。

对于这个问题，要求所有节点对于每一种物品分别满足流量守恒，对于每一条边，所有物品的流量和不超过容量。由于只需要判断是否存在可行解，故目标函数为“空”：

$$
\begin{array}{lll}
\text{最大化} & 0 & \\
\text{满足约束} & \sum_{i} f_i(u,v) \le c(u,v), & (u,v)\in E, i=1,2\dots,n \\
& \sum_{v} f_i(u,v) = \sum_{v} f_i(v,u), & u\in V, i=1,2,\dots,n \\
& f_i(t_i,s_i) \ge d_i, & i=1,2,\dots,n \\
& f_i(u,v) \ge 0, & (u,v)\in E\cup\{(t_i,s_i)|1\le i\le n\}
\end{array}
$$

这个问题目前惟一已知的多项式解法是将其表示为线性规划，然后用多项式时间解决这个线性规划。

5 整数线性规划

在线性规划问题中，有一类特殊的问题，它们的自变量取值范围是整数而

非实数，这样的线性规划称为整数线性规划。

整数线性规划是一个NP-hard问题，所以在信息学竞赛中要求解一般整数线性规划是不实际的。但是有一些特殊的线性规划问题保证了当自变量取值范围为整数和实数得到的答案是一样的，例如容量限制均为整数的最大流问题以及最小费用流问题，虽然这些问题在问题描述中并没有要求流量必须是整数，但是由于问题的特殊性，它一定存在一个最优解满足所有边的流量是整数。这一点可以由针对网络流问题的算法的步骤得出。

但是有些问题并不能满足这一点，若此时直接使用单纯形法求解一个一般的线性规划，得到的结果往往是原问题的限制所不能达到的。下面给出一道作者认为应该存疑的例题：

例2. 志愿者招募加强版[5]

有一个比赛持续N天，共有M种志愿者，每种志愿者的工作时间是连续的几段，第i种志愿者可以工作k_i段时间，每段时间从第s_{ij}天开始到第t_{ij}天结束，招募一个这种志愿者的花费为c_i，比赛要求第i天工作的志愿者数量不小于a_i，求最少花费。

$N \leq 1000, M \leq 10000$

显然题目要求招募的志愿者数量必须是整数，所以这是一个整数线性规划问题。这个问题的原问题是志愿者招募[6]这一题，原题每个志愿者的工作时间是一段连续的区间，经过转化后可以转为网络流建模，故使用单纯性算法解决该问题时可以保证求出的定义在实数上的解等于原问题的解。但是本题并没有这个保证，下面给出一个例子：

比赛共有三天，有三种志愿者：第一种志愿者工作时间为$[1,1],[2,2]$两段时间，招募代价为1。第二种志愿者工作时间为$[1,1],[3,3]$两段时间，招募代价为1。第三种志愿者工作时间为$[2,2],[3,3]$两段时间，招募代价为1。要求每天工作的志愿者数量不少于1。

显然这个问题的最优解为招募任意两种志愿者各一个，代价为2，但是当我们把变量取值范围放宽到实数时，最优解为三种志愿者各招募0.5个，代价为1.5，这确实满足了每天工作的志愿者数目为1，但并不符合原问题的描述。

[5]试题来源：http://www.lydsy.com/JudgeOnline/problem.php?id=3265
[6]试题来源：NOI2008

我并没有好的方法解决这个问题，但是从通过情况上看，绝大部分人都是使用单纯形算法通过了此题，对此作者表示不解。

6 对偶问题

6.1 引言

考虑下面的线性规划：

$$\begin{aligned} \text{最小化} \quad & 7x_1 + x_2 + 5x_3 \\ \text{满足约束} \quad & x_1 - x_2 + 3x_3 \geq 10 \\ & 5x_1 + 2x_2 - x_3 \geq 6 \\ & x_i \geq 0, \qquad i = 1, 2, 3 \end{aligned}$$

由于每一个变量x_i非负，那么容易得到$7x_1 + x_2 + 5x_3 \geq x_1 - x_2 + 3x_3 \geq 10$，这一点可以通过逐项比较变量的系数得到。也就是说，目标函数的值至少是10。

那么，用同样的方法，能否再得到更“紧”的目标函数的下界呢？同时考虑两个限制条件，可以发现，$7x_1+x_2+5x_3 \geq (x_1-x_2+3x_3)+(5x_1+2x_2-x_3) \geq 10+6$。能否做到更好呢？我们考虑如何通过这个方法最大化目标函数的下界。通过这个方法找到的下界一定是如下形式：

$$\begin{aligned} f(y_1, y_2) &= y_1(x_1 - x_2 + 3x_3) + y_2(5x_1 + 2x_2 - x_3) \\ &= (y_1 + 5y_2)x_1 + (-y_1 + 2y_2)x_2 + (3y_1 - y_2)x_3, \qquad y_1, y_2 \geq 0 \end{aligned}$$

同时由于$(x_1 - x_2 + 3x_3) \geq 10, (5x_1 + 2x_2 - x_3) \geq 6$，故$f(y_1, y_2) \geq 10y_1 + 6y_2$。如果对于每一个变量$x_i$，满足$x_i$ 在目标函数中的系数要不小于它在$f(y_1, y_2)$中的系数（例如对于x_1，满足$y_1 + 5y_2 \leq 7$），那么由变量的非负性可以保证

$$\begin{aligned} 7x_1 + x_2 + 5x_3 &\geq (y_1 + 5y_2)x_1 + (-y_1 + 2y_2)x_2 + (3y_1 - y_2)x_3 \\ &= f(y_1, y_2) \geq 10y_1 + 6y_2 \end{aligned}$$

故在满足这个要求的同时我们要求最大化$10y_1 + 6y_2$，可以惊喜地发现，这个问

题同样能表述为一个线性规划问题：

$$
\begin{aligned}
\text{最大化} \quad & 10y_1 + 6y_2 \\
\text{满足约束} \quad & y_1 + 5y_2 \le 7 \\
& -y_1 + 2y_2 \le 1 \\
& 3y_1 - y_2 \le 5 \\
& y_i \ge 0, \qquad i = 1, 2
\end{aligned}
$$

我们称初始的线性规划为原问题，新得到的这个线性规划为对偶问题。如果对对偶问题再进行一次对偶操作，就又回到了原问题，也就是说，对偶过程是相互的，一个最大化问题可以对偶为一个最小化问题，一个最小化问题也同样可以对偶为一个最大化问题。下面给出对偶问题的形式化的定义：

定义6.1. 对偶问题

给定一个原始线性规划：

$$
\begin{aligned}
\text{最小化} \quad & \sum_{j=1}^{n} c_j x_j \\
\text{满足约束} \quad & \sum_{j=1}^{n} a_{ij} x_j \ge b_i, \qquad i = 1, 2, \ldots, m \\
& x_j \ge 0, \qquad j = 1, 2, \ldots, n
\end{aligned}
$$

定义它的对偶线性规划为：

$$
\begin{aligned}
\text{最大化} \quad & \sum_{i=1}^{m} b_i y_i \\
\text{满足约束} \quad & \sum_{i=1}^{m} a_{ij} y_i \le c_j, \qquad j = 1, 2, \ldots, n \\
& y_i \ge 0, \qquad i = 1, 2, \ldots, m
\end{aligned}
$$

用矩阵可以更形象地表示为：

$$
\begin{array}{llcll}
\text{最小化} & \mathbf{c}^T\mathbf{x} & & \text{最大化} & \mathbf{b}^T\mathbf{y} \\
\text{满足约束} & A\mathbf{x} \ge \mathbf{b} & \text{互为对偶} & \text{满足约束} & A^T\mathbf{y} \le \mathbf{c} \\
& \mathbf{x} \ge 0 & & & \mathbf{y} \ge 0
\end{array}
$$

上述两个问题互为对偶问题，其中A^T表示矩阵A的转置。

6.2 线性规划对偶性

首先形式化地给出上一节得到的结论：

定理6.1. （线性规划弱对偶性） 若$\mathbf{x} = (x_1, \ldots, x_n)$是原问题的一个可行解，$\mathbf{y} = (y_1, \ldots, y_m)$是对偶问题的可行解，那么

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i$$

证明. 由于$\mathbf{y}$是对偶问题的一个可行解，那么

$$c_j \geq \sum_{i=1}^{m} a_{ij} y_i$$

由于$x_i \geq 0$，所以

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i \right) x_j$$

同理，由于$\mathbf{x}$是原问题的可行解，$y_i \geq 0$，所以

$$\sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) y_i \geq \sum_{i=1}^{m} b_i y_i$$

原式成立是因为如下等式显然成立

$$\sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i \right) x_j = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) y_i$$

□

这只说明了我们通过对偶问题找到了原问题最优解的一个下界，而下面的定理说明了通过对偶问题求出的下界一定可以达到。

定理6.2. （线性规划对偶性） 若$\mathbf{x}^* = (x_1^*, \ldots, x_n^*)$与$\mathbf{y}^* = (y_1^*, \ldots, y_m^*)$分别是原问题及对偶问题的最优解，那么

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$$

这个定理的证明较为繁琐，此处略去，参考文献[1]中有详细证明。

观察上述的两个定理，若$\mathbf{x}^*, \mathbf{y}^*$分别为原问题与对偶问题的最优解，那么在定理6.1中，等号必然成立。这就引出了下面的定理：

定理6.3. （互松弛定理） 若$\mathbf{x}, \mathbf{y}$分别是原问题及对偶问题的可行解，那么$\mathbf{x}, \mathbf{y}$都是最优解**当且仅当**下列两个条件被同时满足：

- 对于所有$1 \le j \le n$：满足$x_j = 0$或$\sum_{i=1}^{m} a_{ij} y_i = c_j$
- 对于所有$1 \le i \le m$：满足$y_i = 0$或$\sum_{j=1}^{n} a_{ij} x_j = b_i$

对偶原理将一个最大化问题与一个与之对应的最小化问题联系起来，在对偶过程中，原问题的每一个限制条件（即原矩阵的每一行）在对偶问题中产生了一个变量（即转置矩阵的每一列），而原问题的每个变量在对偶问题中就变成了一个限制条件。定理6.2告诉我们这两个问题的最优解相等，而定理6.3则提供了一种原问题最优解与对偶问题最优解之间的关系。

现在考虑一些含有不那么“标准”的线性约束的线性规划对偶问题。若原问题中有约束$\sum_{j=1}^{m} a_{rj} x_j = b_r$，那么将这个约束拆为$\sum_{j=1}^{m} a_{rj} x_j \le b_r, -\sum_{j=1}^{m} a_{rj} x_j \le -b_r$，假设这两个约束对偶之后对应的变量为$y'_r, y''_r$，它们应该满足$y'_r, y''_r \ge 0$，在对偶之后的第$j$个约束为

$$\sum_{1 \le i \le n, i \ne r} a_{ij} y_i + a_{rj} y'_r - a_{rj} y''_r = \sum_{1 \le i \le n, i \ne r} a_{ij} y_i + a_{rj}(y'_r - y''_r)$$

如果令$y_r = y'_r - y''_r$，那么$y_r \in \mathbb{R}$，也就是说，一个等式约束在对偶之后变成了一个无限制的变量。

使用相同的方法可以继续考虑限制$\sum_{j=1}^{m} a_{rj} x_j \ge b_r$，它对偶之后得到了一个非正变量。

6.3 一些经典问题的对偶

6.3.1 最大流问题

接触过网络流的选手应该都知道，最大流的对偶问题是最小割问题，接下来我们来证明这一点。

最大流问题的线性规划描述在4.2中已经给出：

$$
\begin{aligned}
\text{最大化}\quad & f_{ts} \\
\text{满足约束}\quad & f_{uv} \le c_{uv}, && (u,v) \in E \\
& \sum_v f_{uv} = \sum_v f_{vu}, && u \in V \\
& f_{uv} \ge 0, && (u,v) \in E \cup \{(t,s)\}
\end{aligned}
$$

最小割问题可以表述为如下的线性规划：

$$
\begin{aligned}
\text{最小化}\quad & \sum_{u,v\in E} c_{uv} d_{uv} \\
\text{满足约束}\quad & d_{uv} - p_u + p_v \ge 0, && (u,v) \in E \\
& p_s - p_t \ge 1 \\
& p_u, d_{uv} \in \{0,1\}
\end{aligned}
$$

这是因为，一方面，一个割将所有点分为了两个不相交的集合S,T，其中$s \in S, t \in T$，这个割有代价$|cut(s,t)| = \sum_{u\in S, v\in T} c_{uv}$。令$p_u = [u \in S]$[7]$, d_{uv} = \max\{p_u - p_v, 0\}$，那么这个割就对应了这个线性规划的一个可行解。另一方面，要最小化$\sum_{u,v\in E} c_{uv} d_{uv}$，就要求当$u \in S, t \in T$ 时，$d_{uv} = 1$，否则$d_{uv} = 0$，那么这个线性规划的最优解就对应了一个割，故最小割问题可以用这个线性规划解决。

接下来考虑最大流的对偶问题。记由(u,v)的容量限制产生的变量为d_{uv}，由点u的流量守恒产生的变量为p_u，那么对偶之后的问题就是：

$$
\begin{aligned}
\text{最小化}\quad & \sum_{(u,v)\in E} c_{uv} d_{uv} \\
\text{满足约束}\quad & d_{uv} - p_u + p_v \ge 0, && (u,v) \in E \\
& p_s - p_t \ge 1 \\
& d_{uv} \ge 0, && (u,v) \in E
\end{aligned}
$$

这个形式已经非常接近最小割的模型了，但是限制条件$p_u \in \{0,1\}$没有被满足。不过可以证明，一定存在一个最优解满足这个限制。这一部分的证明与主题无关，此处略去，在附录中给出。这里直接给出结论：

[7]方括号内的式子是一个布尔表达式，这个式子表示当$u \in S$ 时，$p_u = 1$，否则$p_u = 0$

定理6.4. （最大流最小割定理）给定一个源为s，汇为t的流网络，则s,t之间的最大流等于s,t之间的最小割。

6.3.2 二分图最大权匹配问题

二分图最大权匹配问题可以表示为下述线性规划：

$$
\begin{aligned}
\text{最大化} \quad & \sum_{u,v\in E} c_{uv}d_{uv} & \\
\text{满足约束} \quad & \sum_{v\in Y} d_{uv} \le 1, & u \in X \\
& \sum_{u\in X} d_{uv} \le 1, & v \in Y \\
& d_{uv} \in \{0,1\} &
\end{aligned}
$$

其中用d_{uv}表示点u,v是否匹配，c_{uv}表示匹配代价，X,Y分别为二分图左右部点集。类似最小割问题的证明，在这里去掉取值为整数的限制并不影响答案[8]，考虑这个问题的对偶问题。我们用p_u,p_v来表示上述线性规划中两类限制对偶之后对应的变量。对偶之后可得：

$$
\begin{aligned}
\text{最小化} \quad & \sum_{u\in X} p_u + \sum_{v\in Y} p_v & \\
\text{满足约束} \quad & p_u + p_v \ge c_{uv} & u \in X, v \in Y \\
& p_u, p_v \ge 0 &
\end{aligned}
$$

这个问题有如下组合解释：给出一张带权二分图，要求给每一个节点定一个“顶标”，对于二分图中的每一条边要求满足这条边连接的两个顶点的顶标之和不小于边权，且所有顶标非负，求最小的顶标之和。我们称其为最小顶标和问题，这一张图与原图相同，故得到以下结论：

定理6.5. 在一张带权二分图中，最大权匹配等于最小顶标和。

考虑这个问题的弱化版，若二分图中每条边的边权都为1，那么原问题就是二分图最大匹配问题。如果限制对偶之后的问题自变量取值为整数，那么对偶

[8]这是因为可以找到一个取值为整数的解得到的目标函数值等于最优解

之后的限制就变成了对于有边相连的两个节点，至少有一个节点的顶标为1，也就是说对于一条边连接的两个顶点，至少要选择一个顶点来“覆盖”这条边。那么有以下结论：

定理6.6. 二分图中，最大匹配数等于最小点覆盖数。

二分图的最大权匹配问题可以用KM算法或者直接套用最小费用最大流的算法高效求解。一般来说，我们倾向于将最小顶标和问题转化为最大权匹配问题解决。

例3. *MST*最小生成树[9]

给出一张带权无向图G以及这张图上的一棵生成树T，你可以修改一条边的边权，代价为权值改变量的绝对值。要求经过修改之后T是G的一棵最小生成树，要求最小化代价。

用n, m分别表示图G的点数以及边数，则$n \leq 50, m \leq 800$

由于目标是让T称为一棵最小生成树，那么对于在树上的边，进行的操作一定是减小它的边权，对于非树边就只能增加边权。考虑一条非树边e，记e连接的两个节点为u, v，那么如果T是最小生成树就要满足e的边权不小于树中u, v路径上的任意一条边的边权。也就是说若边t在树中u, v两点间的路径上，那么就要求$V(e) + \Delta e \geq V(t) - \Delta t$，即$\Delta t + \Delta e \geq V(t) - V(e)$。把原图中的每条边当做一个节点，按照是否出现在$T$中分类，那么这就是一个二分图，把$\Delta t$ 看做顶标，则限制$\Delta t + \Delta e \geq V(t) - V(e)$就是要求顶标和不小于$V(t) - V(e)$。那么这就是一个最小顶标和问题，可以转化为二分图最大权匹配问题解决。

6.4 对偶原理在信息学竞赛中的应用

6.4.1 寻找初始基本解

例4. *Flight Distance* [10]

给出一张n个节点的带权无向图，边(u, v)的边权为w_{uv}，可以随意增加或者减少一条边的边权，代价是边权的变化量。要求这张图在进行操作之后满足任

[9]题目来源：Shoi2004 http://www.lydsy.com/JudgeOnline/problem.php?id=1937

[10]试题来源：https://www.codechef.com/FEB12/problems/FLYDIST

意两点间直接相连的边长度不超过两点之间的最短路长度。求最小代价，以分数形式输出。

$n \le 10, w_{uv} \le 20$

记边的集合为E。对于每一条边$(u,v) \in E$建立两个变量t^+_{uv}, t^-_{uv}，表示这条边的变化量，那么在操作之后的边权为$w_{uv} + t^+_{uv} - t^-_{uv}$。另$d_{uv}$为$u,v$两点在操作之后之间的最短路的长度，那么由于题目要求连接两点的边的长度不超过两点之间最短路长度，同时这两点之间的最短路长度也不能小于直接相连的边的边权。所以得到$d_{uv} = w_{uv} + t^+_{uv} - t^-_{uv}$。另外，最短路之间需要满足$d_{uv} \le w_{ui} + t^+_{ui} - t^-_{ui} + d_{iv} (u \ne v, (u,i) \in E)$，否则$d_{uv}$就可以被继续松弛，得到一条更短的路径，这个条件保证了d_{uv}为u,v之间的最短路长度。所以可以得到如下线性规划：

$$
\begin{aligned}
\text{最小化} \quad & \sum_{(u,v)\in E} t^+_{uv} + t^-_{uv} & \\
\text{满足约束} \quad & d_{uv} - t^+_{uv} + t^-_{uv} \ge w_{uv} & (u,v) \in E \\
& -d_{uv} + t^+_{uv} - t^-_{uv} \ge -w_{uv} & (u,v) \in E \\
& t^+_{ui} - t^-_{ui} + d_{iv} - d_{uv} \ge -w_{ui} & u \ne v, (u,i) \in E \\
& d_{uv}, t^+_{uv}, t^-_{uv} \ge 0 &
\end{aligned}
$$

如果直接求解这个线性规划，那么需要执行单纯形算法的初始化操作，这个操作较为繁琐。由于在目标函数中，所有的系数都是非负的，也就是说对偶之后所有的约束条件的常数项非负，故可以不必进行初始化操作，所以可以对偶之后执行单纯形算法。

因为答案要求输出分数形式，所以需要实现一个分数类。

6.4.2 将线性规划模型转化为半平面交

当一个线性规划只有两个变量的时候，每一个限制条件就将解集限制在了二维平面上的一个半平面内，所以线性规划问题的解空间就是所有限制对应的半平面的交。

例5. *Equations*[11]

[11]试题来源：http://poj.org/problem?id=3689

给出三个长度为n的数组a_i, b_i, c_i以及m次询问，每次询问给出两个参数s, t，求一组非负实数x_i，满足

$$\sum_{i=1}^{n} a_i x_i = s, \sum_{i=1}^{n} b_i x_i = t$$

的同时最大化$\sum_{i=1}^{n} c_i x_i$，对于每次询问输出这个最大值，或者判断无解。

$n \le 10^5, m \le 10^4, 1 \le a_i, b_i, c_i, s, t \le 10^4$

对于每一个询问，写出线性规划模型：

$$\begin{aligned} \text{最大化} \quad & \sum_{i=1}^{n} c_i x_i \\ \text{满足约束} \quad & \sum_{i=1}^{n} a_i x_i = s \\ & \sum_{i=1}^{n} b_i x_i = t \\ & x_i \ge 0, \qquad i = 1, 2, \ldots, n \end{aligned}$$

这个模型只有两个限制条件，故对偶之后只有两个变量，不妨记这两个变量为x, y，那么对偶之后得到如下线性规划：

$$\begin{aligned} \text{最小化} \quad & sx + ty \\ \text{满足约束} \quad & a_i x + b_i y \ge c_i, \qquad i = 1, 2, \ldots, n \\ & x, y \in \mathbb{R} \end{aligned}$$

那么，每一个限制$a_i x + b_i y \ge c_i$就对应了一个半平面，满足所有限制的解x, y就在所有半平面的交内，这些半平面与询问无关，可以预处理出来。那么每一个询问就是要求出这个半平面交内使$sx + ty$最小的点，可以二分求解。当且仅当$sx + ty$无界时，原问题无解。所以总的时间复杂度为$O((n + m) \log n)$

6.4.3 将线性规划模型转化为网络流问题

在最小费用流问题中，线性规划矩阵满足每一列有且仅有两个非零的位置，且它们的值分别为±1。这样每一个变量就可以看为一条边的流量，每一行的限

制可以看成一个节点的流量守恒限制。具体的构图方法在下面的例题中详细说明。

一般来说，经过网络流构图之后使用解决网络流问题的算法求解这一类线性规划问题要比直接用单纯形算法求解快，这是因为解决网络流问题的算法都利用了这些线性规划问题的特殊性，而单纯形算法直接忽略了这些特殊性。这也就是我们考虑将线性规划问题转化为网络流问题的原因。

例6. *Chefbook* [12]

给出一张n个点m条边的有向图，每一条边(u,v)有边权L_{uv}，以及限制S_{uv},T_{uv}。每次操作可以选择：*1.*将一个点u的所有出边权值加上P_u，其中P_u是任意正数。*2.*将一个点u的所有入边权值减去Q_u，其中Q_u是任意正数。每种操作在一个点只能执行一次，两种操作可以在一个点都被执行。可以任意执行操作，记最终每条边边权为L^*_{uv}，要求$S_{uv} \le L^*_{uv} \le T_{uv}$ 的同时最大化$\sum_{uv} L^*_{uv}$，**并输出方案。**

$n \le 100, m \le n^2$

由于这个问题中，每一个操作可能会修改多个位置，难以直接求解。

简单观察后发现，这个问题可以被表示为线性规划模型，每一个限制可以表示为$S_{uv} \le L_{uv} + P_u - Q_v \le T_{uv}((u,v) \in E)$，要求最大化$\sum_{uv} L_{uv} + P_u - Q_v((u,v) \in E)$。计算过程中可以忽略目标函数的常数项，只需在最后加上忽略的常数即可。接下来合并同类项，P_u的系数为$\sum_{uv} 1((u,v) \in E)$，这就等于点u 的出度，同理Q_v的系数就等于点v的入度的相反数。记点u 的入度为$deg_{in}[u]$，出度为$deg_{out}[u]$，即可得到如下线性规划：

$$
\begin{array}{lrr}
\text{最大化} & \sum_u (P_u deg_{out}[u] - Q_u deg_{in}[u]) & \\
\text{满足约束} & P_u - Q_v \le T_{uv} - L_{uv}, & (u,v) \in E \\
 & -P_u + Q_v \le L_{uv} - S_{uv}, & (u,v) \in E \\
 & P_u, Q_u \ge 0, & u \in V
\end{array}
$$

这个线性规划的矩阵满足每一行有且仅有两个系数非零，且分别为±1，所以考虑将其对偶，用x_{uv} 表示前一种限制对偶后对应的变量，y_{uv}表示后一种限制对偶

[12]试题来源：https://www.codechef.com/JUNE15/problems/CHEFBOOK

后对应的变量，那么可以得到：

$$
\begin{aligned}
\text{最小化} \quad & \sum_u ((T_{uv} - L_{uv})x_{uv} + (L_{uv} - S_{uv})y_{uv}) \\
\text{满足约束} \quad & \sum_v (x_{uv} - y_{uv}) \geq deg_{out}[u], \quad u \in V \quad (1) \\
& \sum_u (-x_{uv} + y_{uv}) \geq -deg_{in}[v], \quad v \in V \quad (2) \\
& x_{uv}, y_{uv} \geq 0, \quad (u, v) \in E
\end{aligned}
$$

考虑限制

$$\sum_v (x_{uv} - y_{uv}) \geq deg_{out}[u]$$

在此式中增加辅助变量$a_u \geq 0$，即可得到

$$\sum_v (x_{uv} - y_{uv}) - deg_{out}[u] - a_u = 0, u \in V$$

同理，对于约束(2)，增加辅助变量b_v，可得：

$$\sum_u (-x_{uv} + y_{uv}) + deg_{in}[v] - b_v = 0, v \in V$$

满足上述条件同时，最小化

$$\sum_u ((T_{uv} - L_{uv})x_{uv} + (L_{uv} - S_{uv})y_{uv})$$

经过这样的转化之后，这个问题就变成了最小费用流问题，将每一个变量看为一条边，变量的值是边的流量，这条边的费用就是这个变量在目标函数中的系数；每一个限制看为一个点，那么在限制u中若变量x_{uv}的系数为+1，就相当于变量x_{uv}所对应的边流入点u，若系数为−1，就相当于这条边从点u流出，那么每一个限制就相当于一个节点的流量守恒方程，也就是说这一个线性规划与一个费用流模型等价，接下来考虑如何构造出这个费用流图。

将约束(1)对应的点记作u，约束(2)对应的点记作$v + n$，那么上述线性规划的对应建图应该为：对于每一条边$(u, v) \in E$，新建一条$v + n \to u$的边，费用为$T_{uv} - L_{uv}$，容量为∞（对应变量x_{uv}），以及一条$u \to v + n$的边，费用为$L_{uv} - S_{uv}$，容量为∞（对应变量y_{uv}）。由于变量a_u只出现了一次，那么就建立一条$u \to t$的边，容量为∞，费用为0(其中t表示网络流图中的汇，对于节点t没有流量守恒

限制)，b_v类似。对于限制中的常数项$-deg_{out}[u]$，建立一条$u \to t$的边，容量为$deg_{out}[u]$，并且强制这条边满流（也就是说这条边的流量下界是$deg_{out}[u]$）。对于常数项$deg_{in}[v]$，建立一条$s \to v+n$的边，容量为$deg_{in}[v]$，流量下界为$deg_{in}[v]$。

至此，我们已经将这个线性规划转化为了网络流模型，那么接下来就只需使用有下界的最大费用流算法解决。

上述建模是一个比较一般的建模方法，首先将所有限制变为等式限制，然后分别处理变量、辅助变量以及常数项，将模型转化为一个有下界的费用流问题。不过此题中还有更为巧妙的处理方法：

由于这个线性规划的矩阵满足每一列的和为0（包括常数列），那么考虑任意一个可行解，这个解必然同时满足

$$\sum_v (x_{uv} - y_{uv}) \geq deg_{out}[u], \quad \sum_u (-x_{uv} + y_{uv}) \geq -deg_{in}[v]$$

如果把这两类共$2n$个限制相加，则不等式右侧为$\sum_u deg_{out}[u] - \sum_v deg_{in}[v] = 0$，左侧为$\sum_u \sum_v (x_{uv} - y_{uv}) + \sum_v \sum_u (-x_{uv} + y_{uv}) = 0$，所以得到$0 \geq 0$，由于$0 = 0$，那么要使这个等号成立，约束条件中的等号必然全部成立（否则将所有限制相加之后就得到$0 > 0$），也就是说，对于一个可行解，需要满足

$$\sum_v (x_{uv} - y_{uv}) = deg_{out}[u], \quad \sum_u (-x_{uv} + y_{uv}) = -deg_{in}[v]$$

如此便不需要添加辅助变量，更进一步，由于s的所有出边的容量和为$\sum_u deg_{out}[u]$，t的所有入边容量和为$\sum_v deg_{in}[v]$，而$\sum_u deg_{out}[u] = \sum_v deg_{in}[v]$，且除此之外图中不存在其它的可行流量，故只需要执行**最小费用最大流**算法即可保证所有等式成立，不需要对边加入流量下界的限制，如此建出的网络流模型中的边数比前一种方法少，故运行更快。

至此我们已经可以求出最终的答案，但是输出方案的问题仍没有解决。在这个问题中，我们已经知道了对偶问题的一个最优解的方案（每个变量的取值就等于费用流中对应边的流量），那么，由定理6.3(互松弛定理)可以得到：若$x_{uv} > 0$，那么原问题的最优解必然满足$P_u - Q_v = T_{uv} - L_{uv}$，若$y_{uv} > 0$，那么$-P_u + Q_v = L_{uv} - S_{uv}$，除此之外还应该满足原线性规划的限制条件，即$P_u - Q_v \leq T_{uv} - L_{uv}, -P_u + Q_v \leq L_{uv} - S_{uv}$，那么这就是一个差分约束系统，可以用最短路算法解决，对应的构图如下：对于限制$P_u - Q_v \leq T_{uv} - L_{uv}$，建立一条$v + n \to u$的长度为$T_{uv} - L_{uv}$的边，限制$-P_u + Q_v \leq L_{uv} - S_{uv}$，建立一条$u \to v + n$的长度

为$L_{uv} - S_{uv}$的边，对于等式限制可以将其拆为两个不等式限制再连边。然后随意选择一个起点执行最短路算法，最后每个点的距离顶标就对应着一组最优解。

为了保证求出的最优解P, Q非负，可以将所有的P, Q均加上一个大常数，可以发现这样并不影响答案以及线性规划的限制。至此整个问题得以解决，时间复杂度为$O(\text{mincostflow}(2n, 2(n+m)))$

对于这个问题，我测试了单纯形与网络流的运行时间时间差别，一共运行了$n = 50, m = 1000$的测试数据十组，使用费用流算法运行时间约为$60ms$，而使用单纯形算法运行时间约为$1630ms$。足以体现问题的独特性质对算法运行时间的影响。

例7. *Orz the MST*[13]

给出一个带权的连通无向图，图中有n个点m条边，对于其中的每条边i，在原来边权的基础上，其边权每增加1需要付出的代价为a_i，边权每减少1需要付出的代价为b_i，现在指定该图的一棵生成树，求通过修改边权，使得该生成树成为图的一棵最小生成树，需要付出的最少总代价。

$n \le 300, m \le 1000$

类似例3，把每一条边按照是否为树边分类，记树边集合为T，那么非树边集合为$E \setminus T$，设第i条树边减小了权值x_i，第j条非树边增加了权值y_j，若i被j覆盖[14]，那么要求修改后的i的边权不超过j的边权，所以$w_i - x_i \le w_j + y_j \Leftrightarrow x_i + y_j \ge w_i - w_j$。定义函数$cover(i, j)$表示$i$是否被$j$覆盖，那么可以得到如下线性规划：

$$
\begin{aligned}
&\text{最小化} && \sum_{i \in T} b_i x_i + \sum_{j \in E-T} a_j x_j \\
&\text{满足约束} && x_i + y_j \ge w_i - w_j, \qquad i \in T, j \in E \setminus T, cover(i, j) = 1 \\
& && x_i, y_j \ge 0
\end{aligned}
$$

[13]试题来源：http://www.lydsy.com/JudgeOnline/problem.php?id=3118

[14]不妨设j所连接的两个点为u_j, v_j，那么i被j覆盖即i在u_j, v_j两点间树上的路径上

类似最小顶标和问题，考虑将其对偶，用s_{ij}表示对偶后的对应变量，得到：

$$
\begin{aligned}
\text{最大化} \quad & \sum_{\substack{i\in T, j\in E\setminus T \\ cover(i,j)=1}} (w_i - w_j)s_{ij} & \\
\text{满足约束} \quad & \sum_{\substack{j\in E\setminus T \\ cover(i,j)=1}} s_{ij} \le b_i, & i \in T \\
& \sum_{\substack{i\in T \\ cover(i,j)=1}} s_{ij} \le a_i, & j \in E \setminus T \\
& s_{ij} \ge 0 &
\end{aligned}
$$

由于$T \cap (E \setminus T) = \emptyset$，所以可以将第一类限制看做对左部节点的出度限制，第二类限制看做对右部节点的入度限制，对于左部节点i，最多只能流出b_i的流量，所以建立一条$s \to i$，容量为b_i，费用为0的边，对于右部节点j，建立一条$j \to t$，容量为a_i，费用为0的边，若i被j覆盖，那么由左部的i向右部的j连一条费用为$w_i - w_j$，容量为∞的边，然后执行最大费用流算法。

这个对偶后的模型就是二分图最优多重匹配问题。

例8. 战线防守[15]

战线可以看作一个长度为n的序列，现在需要在这个序列上建塔，在i号位置上建一座塔有c_i的花费，且一个位置可以建任意多的塔。有m限制，每个限制形如区间$[l_i, r_i]$中至少有d_i个塔。求满足所有限制的最少花费。

$n \le 1000, m \le 10000$

由于每一个限制是对一段区间中的数量和的限制，所以为了能使用网络流构图，需要使用一点小技巧：记$s_i = \sum_{j\le i} x_j$，即对x作前缀和，那么每个限制就

[15]试题来源：ZJOI2013

变成了两个变量相减的形式，于是可以得到如下线性规划：

$$
\begin{aligned}
\text{最小化} \quad & \sum_{j=1}^{n}(s_j - s_{j-1})c_j \\
\text{满足约束} \quad & s_{r_i} - s_{l_i-1} \ge d_i, && i = 1, 2, \dots, m \\
& s_j - s_{j-1} \ge 0, && j = 1, 2, \dots, n \\
& s_j \ge 0, && j = 0, 1, \dots, n
\end{aligned}
$$

然后用类似处理例6的方法进行费用流建图即可，这里不再赘述。

6.4.4 小结

要高效解决一些线性规划问题最重要的环节是对特定的线性规划问题的特殊性的运用，例如只有两个变量的线性规划问题可以用半平面交算法解决，每个变量只在两个限制中出现，且系数和为0的线性规划问题可以用网络流算法解决等。对偶原理便是一种运用这些性质的手段。

7 总结

从线性规划的角度看待信息学竞赛中的问题，使得这些问题的描述更加清晰，同时这也带来了一种新的解决问题的方法。但是将问题一般化之后变不免损失了问题的特殊性，难以更高效地解决问题。

通过对具有特殊性的问题（例如网络流问题）的线性规划表示的总结，我们能够将一些具有特殊性的线性规划问题转化为其他问题，从而可以使用针对这些问题的算法来高效解决具有某些特殊性的线性规划问题，这就使我们在用一般性的角度看待问题的同时又能很好地利用问题的特殊性，达到高效解题的目标。

感谢

感谢父母的养育之恩。

感谢陈颖老师，余林韵教练，黄豪硕学长，张瑞喆学长的指导，感谢周聿浩等同学对本文提供意见与帮助。

感谢CCF提供学习和交流的机会。

参考文献

[1] Thomas H.Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein. Introduction to Algorithms. Second Edition.

[2] Introduction to Linear Optimization, Dimitris BersimasJohn N. Tsitsiklis.

[3] Luca Trevisan. Stanford University-CS261: Optimization, Handout 15

[4] Vijay V. Vazirani. Approximation Algorithms: Chapter 10, Introduction to LP-Duality, Chapter 13, The primal-dual schema

[5] 胡伯涛，《最小割模型在信息学竞赛中的应用》

[6] 曹钦翔，《线性规划与网络流》

附录

最大流对偶线性规划的最优解与割的对应关系

最大流的对偶线性规划如下：

$$
\begin{aligned}
\text{最小化} \quad & \sum_{uv} c_{uv} d_{uv} \\
\text{满足约束} \quad & d_{uv} - p_u + p_v \geq 0, \quad (u,v) \in E \\
& p_s - p_t \geq 1 \\
& d_{uv} \geq 0
\end{aligned}
$$

记这个线性规划的最优解为(d^*_{uv}, p^*_u)，最优解的值为$z^* = \sum_{uv} c_{uv} d^*_{uv}$。为了证明存在一个割，使得$cut(s,t) = z^*$，首先需要证明该线性规划存在一个解$(d_{uv}, p_u)$使得$\sum_{uv} c_{uv} d_{uv} = \sum_{uv} c_{uv} d^*_{uv}$且满足$0 \leq p_u \leq 1$。

由于在目标函数中没有出现p_u，且在约束条件中p_u总是以差值$p_u - p_v$的形式出现，那么不妨假设$p_t^* = 0$。考虑解$(d_{uv}, p_u) = (d_{uv}^*, \min\{p_u^*, 1\})$，它得到的目标函数的值与最优解得到的值相同，下面证明这个解仍满足所有限制：

对于限制$d_{uv} - p_u + p_v \geq 0$，若$p_u^* > 1, p_v^* > 1$，那么这个限制等价于$d_{uv} \geq 0$，显然成立。

若$p_u^* > 1, p_v^* \leq 1$，由于$p_u^* > p_u$，所以$d_{uv} - p_u + p_v > d_{uv}^* - p_u^* + p_v^* \geq 0$。

若$p_u^* \leq 1, p_v^* > 1$，由于$p_u \leq p_v$，故$p_u - p_v \leq 0$，所以$d_{uv} \geq 0 \geq p_u - p_v$。

对于剩余限制显然成立，故解(d_{uv}, p_u)为一个可行解，更进一步，解(d_{uv}, p_u)得到的目标函数的值与最优解得到的值相同，故这是一个最优解。同理可证存在一个最优解满足$p_u \geq 0$。

下面证明一定存在一个割$cut(s,t)$，使得$cut(s,t) \leq z^*$。

一个割可以用一个集合S表示，要求满足$s \in S, t \notin S$，表示将点集分割为$S, \complement_V S$ 两部分。设x 是一个在$(0,1]$ 中均匀取值的随机变量，对于一个x，我们定义一个与之对应集合$S_x = \{u | p_u \geq x\}$，这个集合对应了一个割$cut_x(s,t)$，接下来我们考虑$|cut_x(s,t)|$的数学期望。

由于期望的线性性质，可以分别考虑每一条边对期望的贡献。对于边(u,v)，当且仅当$u \in S_x, v \notin S_x$时才会计算为割的代价。若$p_u \geq p_v$，那么这条边在割中的期望为$E_{uv} = c_{uv}(p_u - p_v)$，否则为0。由于在最小割的线性规划模型中限制了$d_{uv} \geq \max\{0, p_u - p_v\}$，故$c_{uv}d_{uv} \geq E_{uv}$，所以$E(|cut_x(s,t)|) = \sum_{uv} E_{uv} \leq \sum_{uv} c_{uv}d_{uv} = z^*$，由于期望值是所有方案的平均值，所以一定存在$x \in (0,1]$，使得$|cut_x(s,t)| \leq z^*$。证毕。

浅谈无向图最小割问题的一些算法及应用

绍兴市第一中学 王文涛

摘 要

本文对信息学竞赛中出现的一类定义在带非负边权的无向图上的最小割问题进行了研究，对处理这类问题的几个算法（Gomory-Hu树(最小割树)的构造算法、等价流树的Gusfield构造算法、Stoer-Wagner算法、Karger算法）进行了介绍，并通过几道例题简要阐述了这类问题的解题思路。

1 引言

在信息学竞赛乃至一些实际问题中，我们不仅会遇到要求带非负边权的有向图中某个源点和汇点之间的最大流或最小割的情况，也会遇到要求带非负边权的无向图上任意两点间最大流或最小割的情况。对于有向图的情况，之前的论文中已有很多相关的算法和这一类的题目的研究，而后一种情况暂时还没有较深入的研究。本文就针对这类问题进行展开。

在第二节中，为了便于之后的描述，作者将会定义一些无向图上与割相关的概念。

在第三节中介绍了用以解决任意两点最小割问题的Gomory-Hu树(最小割树)和等价流树以及它们的构造算法。

在第四节中介绍了用以求出全局最小割的确定性算法Stoer-Wagner算法和随机算法Karger算法。

对于这些算法，作者给出了它们的证明和简要实现。

在第五节中，本文通过几个例题观察了这些算法如何应用在信息学竞赛的解题过程中。

2 割的有关定义

给定带权无向图$G = (V, E, c)$，其中函数c是$E \to \mathbb{R}^*$的映射（也就是权值非负）。本文中所有的图都是指这样的带权无向图。

对于一个边集$S \subseteq E$，定义$c(S) = \sum_{e \in S} c(e)$。

对于两个不相交的点集A, B定义$c(A, B)$为所有满足$u \in A,\ v \in B$的边(u, v)的边权之和。

定义图G的一个割为图G的点集V的一个划分$(U, V - U)$。我们称$U, V - U$为这个割的两侧。

定义这个割的边集为所有满足$u \in U,\ v \in V - U$的边(u, v)的集合，记为$\delta(U)$。

定义这个割的权为$c(\delta(U))$，或者记为$c(U, V - U)$。

我们称割$(U, V - U)$为u, v的一个割如果$u \in U,\ v \in V - U$。我们称U为这个割的u侧，$V - U$为这个割的v侧。

定义u, v的最小割为权最小的u, v的割，记这样的割的边集为$\alpha(u, v)$，记这样的割的权为$\lambda(u, v)$。

易知u, v的最小割等价于v, u的最小割。

注意到自环不会影响到割，所以下面所有图中默认都去掉了自环。

3 任意两点最小割

对于图中任意两点求出最小割。

一个最容易想到的方法是，直接枚举图中的两个点跑一次最大流算法。但这样需要$\Theta(|V|^2)$次最大流，效率较低。为此，我们需要挖掘最小割的性质，用更少的最大流次数来解决。

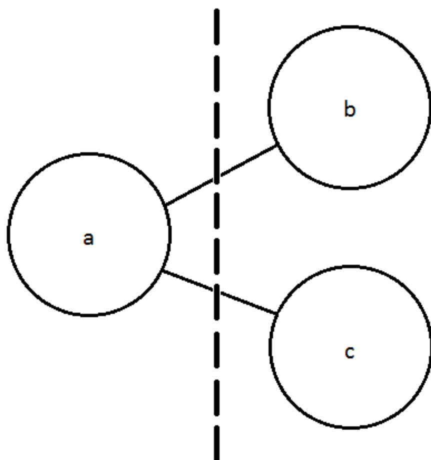
3.1 最小割树

定义一棵树$T = (V, E_T)$为Gomory-Hu树（最小割树）如果对于所有的边$(s, t) \in E_T$，$\delta(W)$是某个$\alpha(s, t)$，其中W为树T删去边(s, t)后产生的两个连通块之一。

当然我们现在还无法确定Gomory-Hu树是否存在。不过后面我们只要证明构造算法的正确性，也就可以证明其存在性了。所以我们不妨暂时先假设存在。

3.1.1 定理1

对于任意不同的三点a, b, c，$\lambda(a, b) \geq \min(\lambda(a, c), \lambda(c, b))$。



证明：对于任意不同的三点a, b, c，考虑$\lambda(a, b), \lambda(b, c), \lambda(c, a)$中最小的一对，不妨假设是$\lambda(a, b)$。讨论$c$在$\alpha(a, b)$的哪一侧，不妨假设是$b$侧，另一侧同理。那么显然有$\lambda(a, c) \leq \lambda(a, b)$。又由于$\lambda(a, b)$是最小的一对，所以$\lambda(a, c) \geq \lambda(a, b)$。得到$\lambda(a, c) = \lambda(a, b)$。由此可见对于任意不同的三点$a, b, c$，$\lambda(a, b), \lambda(b, c), \lambda(c, a)$必然是两个相同的较小值和一个较大值。命题得证。

实际上，我们还可以扩展这个定理，得到：

对于任意不同的两点u, v，有$\lambda(u, v) \geq \min(\lambda(u, w_1), \lambda(w_1, w_2), \ldots, \lambda(w_k, v))$。

3.1.2 定理2

对于所有$u, v \in V$，令(s, t)为u, v在Gomory-Hu树T上唯一的路径上$\lambda(s, t)$最小的边。那么$\lambda(u, v) = \lambda(s, t)$。也就是说，$s, t$的最小割就是Gomory-Hu树上$s$到$t$的路径上权最小的边。

证明：由定理1的扩展可知，$\lambda(u, v) \geq \lambda(s, t)$。又根据定义，$u, v$在$\alpha(s, t)$的两侧，可得$\lambda(u, v) \leq \lambda(s, t)$。因此，$\lambda(u, v) = \lambda(s, t)$。

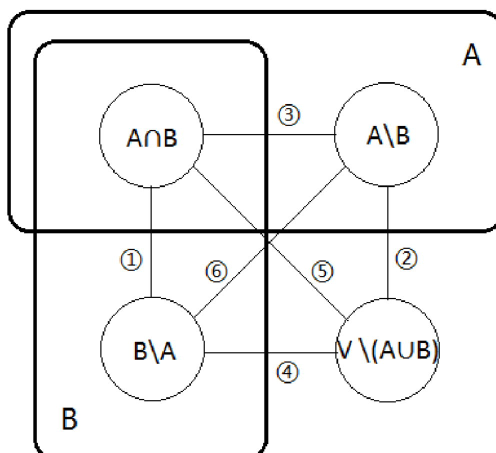
3.1.3 定义1

对于一个集合V，定义一个关于V的子集U的函数$f(U)$，如果f满足对于任

意的$A, B \subseteq V$，$f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$，则称函数$f$为子模[1]函数。

3.1.4 定理3

割的权函数$c(\delta(U))$是子模函数。



证明：对于某两个点集A, B，我们可以把所有点分成4部分：$A \cap B$, $A \setminus B$, $B \setminus A$, $V \setminus (A \cup B)$，如图所示。$f(A) + f(B) =$ ① + ② + ③ + ④ + 2⑤ + 2⑥ $\geq$ ① + ② + ③ + ④ $= f(A \cup B) + f(A \cap B)$。命题得证。

3.1.5 定义2

对于一个集合V上的函数f，如果$f(U) = f(V - U)$，则称f为对称函数。

割的权函数显然是对称函数。

3.1.6 定义3

对于一个集合V上的函数f，如果f满足对于任意的$A, B \subseteq V$，$f(A) + f(B) \geq f(A \setminus B) + f(B \setminus A)$，则称函数$f$为“反模”[2]函数。

[1]英文名：submodular

[2]英文名：posi-modular；作者找不到对应的译名，故用“反模”代替

3.1.7 定理4

如果函数f既是子模函数、又是对称函数，那么f是“反模”函数。

证明：

$$
\begin{aligned}
& f(A) + f(B) \\
= \ & f(V - A) + f(B) \\
\geq \ & f((V - A) \cap B) + f((V - A) \cup B) \\
= \ & f(B \setminus A) + f(V - (A \setminus B)) \\
= \ & f(B \setminus A) + f(A \setminus B)
\end{aligned}
$$

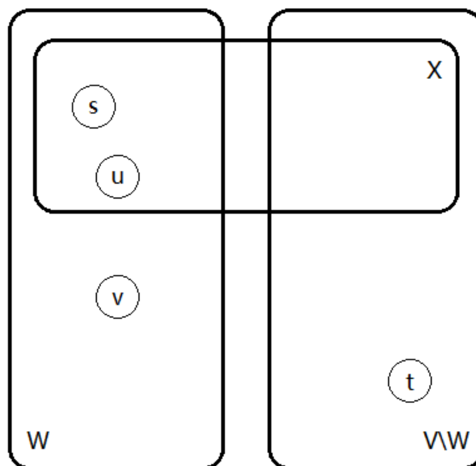
显然，割的权函数也是“反模”函数。

3.1.8 定理5

对于两点s,t的一个最小割$\alpha(s,t)$，设W为$\alpha(s,t)$的一侧，则对于任意的$u,v \in W$, $u \neq v$，存在一个u,v的最小割$\delta(X)$使得$X \subseteq W$。

证明：为了不失一般性，我们令$s \in W$, $u \in X$, $s \in X$。其他情况都可以通过调整转化为这种情况。

接下来，我们分两种情况讨论：



情况1：$t \notin X$。那么，由于割的权函数是子模函数，有：

$$c(\delta(X)) + c(\delta(W)) \geq c(\delta(X \cap W)) + c(\delta(X \cup W))$$

然而注意到$\delta(X \cap W)$也是u, v的一个割，我们有$c(\delta(X \cap W)) \geq c(\delta(X))$；同时$\delta(X \cup W)$也是$s, t$的一个割，我们有$c(\delta(X \cup W)) \geq c(\delta(W))$。因此，上面的所有不等式都应该取等号。这时我们注意到，$\delta(X \cap W)$也是u, v的最小割。

情况2：$t \in X$。那么，由于割的权函数是“反模”函数，有：

$$c(\delta(X)) + c(\delta(W)) \geq c(\delta(W \setminus X)) + c(\delta(X \setminus W))$$

然而注意到$\delta(W \setminus X)$也是u, v的一个割，我们有$c(\delta(W \setminus X)) \geq c(\delta(X))$；同时$\delta(X \setminus W)$也是$s, t$的一个割，我们有$c(\delta(X \setminus W)) \geq c(\delta(W))$。因此，上面的所有不等式都应该取等号。这时我们注意到，$\delta(W \setminus X)$也是u, v的最小割。

综合上述两种情况，原命题成立。

这个定理是构造Gomory-Hu树最重要、最核心的定理。它揭示了最小割可以“不交叉”，某个最小割的两侧中的点的最小割互不相干，从而产生递归的结构。

3.1.9 定义4

令图$G = (V, E)$，点集$R \subseteq V$。定义关于R在G中的Gomory-Hu树为二元组(T, C)，其中$T = (R, E_T)$，$C = (C_r \mid r \in R)$是一个对点集V的划分，其中$r \in R$，使得

- 对于所有的$r \in R$有$r \in C_r$；
- 对于所有的边$(s, t) \in E_T$，$T-(s, t)$（即T删去边s, t后的两部分）表示s, t在G中的最小割。设$T - (s, t)$的其中一部分为X，则s, t在G中的最小割

$$\delta(U) = \bigcup_{r \in X} C_r$$

注意到G的Gomory-Hu树就是$R = V$时的Gomory-Hu树。

3.1.10 算法

伪代码：Gomory-Hu(G,R)

if $|R| = 1$ **then**

$T = (\{r\}, \emptyset)$

$C_r = V$

return (T, C)

end if

任选两点$r_1, r_2 \in R$，$r_1 \neq r_2$，令$\delta(W)$为一个r_1, r_2的最小割，不妨设$r_1 \in W$

令G_1为将G中$V \setminus W$缩成一个点v_1后的图；$R_1 = R \cap W$

令G_2为将G中W缩成一个点v_2后的图；$R_2 = R \setminus W$

令$(T_1, (C_r^1 \mid r \in R_1)) = \text{Gomory-Hu}(G_1, R_1)$

令$(T_2, (C_r^2 \mid r \in R_2)) = \text{Gomory-Hu}(G_2, R_2)$

令r'为满足$v_1 \in C_{r'}^1$的点

令r''为满足$v_2 \in C_{r''}^2$的点

$T = (R_1 \cup R_2, E_{T_1} \cup E_{T_2} \cup \{(r', r'')\})$

for all $r \in R_1$ **do**

$C_r = C_r^1$

end for

for all $r \in R_2$ **do**

$C_r = C_r^2$

end for

$C_{r'} = C_{r'} - \{v_1\}$

$C_{r''} = C_{r''} - \{v_2\}$

return (T, C)

正确性证明：

根据Gomory-Hu树的定义，我们需要证明任意一条边$(s,t) \in E_T$都满足$T - (s,t)$是一个s,t在G中的最小割。当$|R| = 1$时显然成立。对于一般的情况，如果$(s,t) \in T_1$或$(s,t) \in T_2$，由3.1.8可知，T_1或T_2以外的点对其内部的点之间的最小割没有影响，因此T_1定理5和T_2都是合法的Gomory-Hu树。

于是，我们只要证明加上去的那条边(r', r'')合法就可以了。令$\delta(W)$为算法中得到的r_1, r_2的最小割且$r_1 \in W$。我们需要证明的是$\delta(W)$也是r', r''的最小

割。由于$\delta(W)$已是r', r''的一个割，所以只要证$\lambda(r', r'') \geq \lambda(r_1, r_2)$。假定$r_1 \neq r'$且$r_2 \neq r''$（其他情况的证明过程类似）。如果$\lambda(r_1, r') < \lambda(r_1, r_2)$的话，由于在$G_1$中$V \setminus W$被缩成了一个点$v_1$且$v_1 \in C^1_{r'}$，$\lambda(r_1, r')$也就同时是$r_1, r_2$的一个割，这与$\lambda(r_1, r_2)$是$r_1, r_2$的最小割的权矛盾，因此不可能出现这种情况，$\lambda(r_1, r') \geq \lambda(r_1, r_2)$。同理$\lambda(r_2, r'') \geq \lambda(r_1, r_2)$。

于是，根据3.1.1，我们有

$$\lambda(r', r'') \geq \min\left(\lambda(r', r_1), \lambda(r_1, r_2), \lambda(r_2, r'')\right) = \lambda(r_1, r_2)$$

命题得证。

而对于$r' = r_1$或$r'' = r_2$的情况，只是改变了上式min中的一些项，仍然可以得到相同的结论。

根据这个算法，我们立刻有下述引理：

3.1.11 引理1

对于图G中的$R \subseteq V$构造Gomory-Hu树只需计算$|R| - 1$次大小不超过G的图的最小割。

当$R = V$时构出的Gomory-Hu树就是图G的Gomory-Hu树，因此整个算法一共只要$\Theta(|V|)$次最大流。

3.1.12 非递归实现

由于上述递归算法中的连边需要用到划分C，而划分C在具体实现时较难记录，因此我们考虑如何用非递归的方法来实现。

注意到递归的顺序是无关紧要的。我们可以这样看这个算法：每个时刻有很多个点集。每次可以随便选择一个大小大于1的点集拆分成两个点集，不停地拆分直到所有点集的大小都为1 时停止。而对于Gomory-Hu树的连边也可以动态地进行：每个时刻我们让每条边不是连接两个点，而是连接两个点集。把每个点集看成一个大点，那么这些边就构成了这些大点的一棵树，上述算法中缩成的点也就代表着每个点集邻接的子树。而对某个点集拆分时缩成的点的划分也就是这个点集邻接的边的划分。

考虑如何实现这个非递归的算法。对于每个点集，我们选择编号最小的那个点作为这个点集的代表点。对每个点u记录一个它指向的点fa_u。如果点u就

是其所在点集的代表点，u到fa_u的边就代表一条真正的Gomory-Hu树边，连接以u为代表点的点集和以fa_u为代表点的点集，并且已有边权。如果点u不是其所在点集的代表点，那么fa_u表示归属关系，fa_u就是其所在点集的代表点。初始时所有点都在同一个点集里，因此除了1号点外其他点的fa都是1。接下来从2到$|V|$依次枚举点u，如果fa_u表示归属关系，就对u所在点集进行一次拆分。设$v = fa_u$。选择v和u作为拆分时最小割的两个点。算出最小割后，我们要把点集分为两部分。同时我们还要修改所有这个点集邻接的树边。为此，我们把所有被划到u这一侧且$fa_x = v$的点x的fa_x改为u，边权不变。这一步即处理了那些点集内的点，又处理了那些真正的树边。对于树边，我们还要处理fa_v和v这条边。如果fa_v代表的点集被划到了u这一侧，就要把fa_u设为fa_v。最后还要在划出的两个点集之间连上一条权为最小割权的边。如果上一步中fa_v没有被划到u一侧，就把fa_u设成v来代表这条边；否则就把fa_v设成u来代表这条边。这样就完成了一次划分。

注意到每次划分后我们都只把当前点u到fa_u的边从表示归属关系的边转变成了树边，因此枚举过程中一定是之前枚举过的点对应的fa边都是树边而之后没被枚举过的点对应的fa边都是表示归属关系的边。因此每次枚举到一个点时一定会对其进行拆分。

伪代码：Gomory-Hu Incremental Method(G)

$w_1, w_2, \ldots, w_{|V|} = NULL$
$fa_1 = NULL$
$fa_2, fa_3, \ldots, fa_{|V|} = 1$
for $u = 2$ **to** $|V|$ **do**
  $v = fa_u$
  在图G中把其他点集缩点之后的图上计算v, u的最小割$\alpha(v, u)$，得到最小割权为$\lambda(v, u)$
  for $x = 1$ **to** $|V|$ **do**
    if $x \neq u$ **and** $fa_x = v$ **and** x被划到v侧 **then**
      $fa_x = u$
    end if
  end for
  if fa_v被划到u侧 **then**
    $fa_u = fa_v$

$w_u = w_v$
$fa_v = u$
$w_v = \lambda(v, u)$
else
$fa_u = v$
$w_u = \lambda(v, u)$
end if
end for
$E = \emptyset$
for $u = 1$ **to** $|V|$ **do**
if $fa_u \neq NULL$ **then**
$E = E \cup \{(fa_u, u)\},\ c(fa_u, u) = w_u$
end if
end for
return $T = (V, E, c)$

3.2 等价流树的Gusfield构造算法

有时我们不要求Gomory-Hu树反映具体的最小割是什么，只要求出最小割的权即可。也就是说，我们要求一棵树，使得s, t的最小割就是树上路径上权值最小的边，而对于树上的每条边，这条边的边权为两个端点之间最小割的权，但我们不要求删去这条边后的两个连通块就代表这条边的最小割。我们称这种树为等价流树[3]。

等价流树的构建算法Gusfield算法比Gomory-Hu树要简单许多。我们只要对Gomory-Hu 树的构建算法稍加修改：对于递归的过程，我们不按原来的算法选两个点连边，而是直接在源点的s和汇点t之间连相应最小割权值的边。证明也要简单得多：首先递归的两部分构出来的都是合法的等价流树；现在中间连了一条边后，我们只要证明所有从s侧的某个点u到t侧的某个点v的路径都满足等价最小割的性质，即

$$\lambda(u, v) = \min\Big((u, s)\text{路径上最小的边权}, \lambda(s, t), (t, v)\text{路径上最小的边权}\Big)$$

由3.1.1可知，上式左边部分大于等于右边部分，因此只要证上式左边部分小于

[3]英文名：equivalent flow tree

等于右边部分即可。显然，s, t的最小割也是u, v的割，因此$\lambda(u, v) \le \lambda(s, t)$。然后考虑$(u, s)$路径上最小的边所代表的最小割。由3.1.8可知，这个最小割必然不会割开t侧的点集，因此t侧的点集要么属于这个最小割的s侧，要么属于u侧。如果属于s侧，那么这个最小割也是(u, v)的割；如果属于u侧，那么这个最小割也是(s, t)的割，权值不会小于$\lambda(s, t)$。对于(t, v)路径上最小的边所代表的最小割也是如此。综合起来，我们就可以证明刚才的结论了。

注意到递归时求的s, t最小割可以与当前处理的点集之外的最小割相交。因为如果相交必然可以调整得到一个权值相同且不相交的最小割，而这个最小割对当前点集的划分是不变的。因此Gusfield算法不涉及缩点，更容易用代码实现。

事实上，Gusfield算法不仅可以递归实现，也可以非递归实现。

伪代码：Gusfield's Algorithm(G)

$fa_1 = NULL$

$fa_2, fa_3, \ldots, fa_{|V|} = 1$

for $u = 2$ **to** $|V|$ **do**

  $v = fa_u$

  求出$\alpha(v, u), \lambda(v, u)$

  $w_u = \lambda(v, u)$

  for $x = u + 1$ **to** $|V|$ **do**

    if $fa_x = v$ **and** x被划到v侧 **then**

      $fa_x = u$

    end if

  end for

end for

$E = \emptyset$

for $u = 2$ **to** $|V|$ **do**

  $E = E \cup \{(fa_u, u)\},\ c(fa_u, u) = w_u$

end for

return $T = (V, E, c)$

4 全局最小割

求出图中最小的一个割（全局最小割）。

一种方法是，构出Gomory-Hu树，取其中最小的一条边就是全局最小割。

但有复杂度更好的算法。

4.1 Stoer-Wagner算法

对于图中任意两点，它们有可能在全局最小割的同侧，也可能在异侧。如果在同侧，显然我们可以把这两个点缩成一个点，不会对答案造成影响；如果在异侧，这两个点的最小割就是全局最小割。所以，只要我们不停地进行操作：每次任选两个点做一下最小割，更新一下答案，然后把这两个点缩起来，直到只剩一个点为止，就可以求出全局最小割。

如何找到图中任意两个点的最小割呢？下面的算法给出了一个可行的方案。

4.1.1 最大邻接搜索

有一个点集A。初始时A中包含某一个初始的点。每次，选择一个不在A中的点v使得$c(A, \{v\})$最大，将v加入到A中。直到所有点都加入到A中。

4.1.2 定理

设第i个加入A的点为v_i。$(\{v_1, v_2, \ldots, v_{n-1}\}, \{v_n\})$是最大邻接搜索中最后加入的两个点$v_{n-1}, v_n$的最小割。

证明：

如果$n = 2$，显然成立。因此设$n > 2$。考虑最后加入的三个点v_{n-2}, v_{n-1}, v_n。首先我们可以删去边(v_{n-1}, v_n)，因为任何v_{n-1}, v_n的割都要割这条边，删去了对这些割的相对大小没有影响。

现在我们来证明$c(\{v_1, \ldots, v_{n-1}\}, \{v_n\}) \leq \lambda(v_{n-2}, v_{n-1})$。

考虑从图中移除点v_n（及其相连的边）。按原顺序重新进行最大邻接搜索，那么v_{n-2}, v_{n-1}成为最后两点。由数学归纳可知，$(\{v_1, \ldots, v_{n-2}\}, \{v_{n-1}\})$是$v_{n-2}, v_{n-1}$的最小割。

现在把v_n（及其相连的边）加回图中。那么显然v_{n-2}, v_{n-1}的最小割不可能变得更优。而由于v_n与v_{n-1}间没有边，所以我们把v_n加到原最小割的$\{v_1, \ldots, v_{n-2}\}$侧，最小割不变。因此$(\{v_1, \ldots, v_{n-2}, v_n\}, \{v_{n-1}\})$是$v_{n-2}, v_{n-1}$的最小割。

又由最大邻接搜索倒数第二步的选择可知

$$c(\{v_1, \ldots, v_{n-2}\}, \{v_{n-1}\}) \geq c(\{v_1, \ldots, v_{n-2}\}, \{v_n\})$$

由于v_{n-1}与v_n间没有边，

$$c(\{v_1, \ldots, v_{n-2}, v_n\}, \{v_{n-1}\}) \geq c(\{v_1, \ldots, v_{n-2}, v_{n-1}\}, \{v_n\})$$

即

$$\lambda(v_{n-2}, v_{n-1}) \geq c(\{v_1, \ldots, v_{n-1}\}, \{v_n\})$$

接下来我们再用相似的方法来证明$c(\{v_1, \ldots, v_{n-1}\}, \{v_n\}) \leq \lambda(v_{n-2}, v_n)$。

考虑从图中移除点v_{n-1}（及其相连的边）。按原顺序重新进行最大邻接搜索，那么v_{n-2}, v_n成为最后两点。由数学归纳可知，$(\{v_1, \ldots, v_{n-2}\}, \{v_n\})$是$v_{n-2}, v_n$的最小割。

现在把v_{n-1}（及其相连的边）加回图中。那么显然v_{n-2}, v_n的最小割不可能变得更优。而由于v_{n-1}与v_n间没有边，所以我们把v_{n-1}加到原最小割的$\{v_1, \ldots, v_{n-2}\}$侧，最小割不变。因此$(\{v_1, \ldots, v_{n-2}, v_{n-1}\}, \{v_n\})$是$v_{n-2}, v_n$的最小割。即

$$c(\{v_1, \ldots, v_{n-1}\}, \{v_n\}) = \lambda(v_{n-2}, v_n)$$

也就证明了$c(\{v_1, \ldots, v_{n-1}\}, \{v_n\}) \leq \lambda(v_{n-2}, v_n)$。

综合上述两部分，可得

$$\lambda(v_{n-1}, v_n) \leq c(\{v_1, \ldots, v_{n-1}\}, \{v_n\}) \leq \min\left(\lambda(v_{n-1}, v_n), \lambda(v_{n-2}, v_n)\right)$$

又根据3.1.1，

$$\lambda(v_{n-1}, v_n) \geq \min\left(\lambda(v_{n-1}, v_n), \lambda(v_{n-2}, v_n)\right)$$

因此，上述不等式中所有不等号均取等号。也就是说，

$$\lambda(v_{n-1}, v_n) = c(\{v_1, \ldots, v_{n-1}\}, \{v_n\})$$

命题得证。

4.1.3 复杂度

对于最大邻接搜索，我们可以用一个堆来维护所有点v的$c(A,\{v\})$值。如果我们采用二叉堆，复杂度为$O(|E|\log|V|)$，使用斐波那契堆可以做到$O(|E|+|V|\log|V|)$。

4.2 Karger算法

该算法是一个随机算法，不能保证绝对正确，但可以通过多次运行提高正确率。

4.2.1 不带权的图

对于不带权的图，算法如下：每次等概率随机选取图上的一条边，假设这条边不在全局最小割上，因此可以把这条边两端的两点缩成一个大点。缩起来后去掉所有自环。不停地重复缩边的过程直到图上只剩下两个点为止，中间那条边就是我们得到的一个割。

正确率：注意到如果最终得到的是全局最小割，那么中间每步缩的边都不能是在最小割上的边。设全局最小割有k条边。那么显然，每个点的度数都不小于k。（因为如果某个点度数小于k，就形成了更小的的一个割）因此总的边数$|E|$不少于$\frac{k|V|}{2}$。（每条边在两个端点处各算到一次）。每次随机选到的边在全局最小割上的概率

$$\frac{k}{|E|}\le\frac{k}{\frac{k|V|}{2}}=\frac{2}{|V|}$$

于是，每次没有选到全局最小割上的边的概率不小于$1-\frac{2}{|V|}$。由于每次缩边都会使点数减少1，最终得到全局最小割的概率不小于

$$\left(1-\frac{2}{|V|}\right)\left(1-\frac{2}{|V|-1}\right)\cdots\left(1-\frac{2}{3}\right)=\frac{|V|-2}{|V|}\frac{|V|-3}{|V|-1}\cdots\frac{1}{3}=\binom{|V|}{2}^{-1}$$

那么每次做完后没有得到全局最小割的概率为$1-\binom{|V|}{2}^{-1}$。由于$1-x\le e^x$，$1-\binom{|V|}{2}^{-1}\le e^{-\binom{|V|}{2}^{-1}}$。设运行了$m$次该算法取最优答案，则正确率为$1-\left(1-\binom{|V|}{2}^{-1}\right)^m\ge 1-e^{-\binom{|V|}{2}^{-1}m}$。当$m=\binom{|V|}{2}$时就可以做到$1-e^{-1}$的正确率。也就是说只要运行$\Theta(|V|^2)$次就可以得到某个常数的正确率了。

4.2.2 实现

对于随机选边的过程，可以预先随机生成一个所有边的排列，然后按排列中的顺序枚举每条边尝试缩边，直到图中只有两个点为止。如果枚举到的边当前在图中已是自环，就忽略。易知这样的选法和上面的选法是等价的。

主要难点在于缩点。如果用邻接表或邻接矩阵存储边，每次缩点需要$O(|V|)$的时间，总时间复杂度为$O(|V|^2)$。为此我们需要优化。

如果用链表存储边，我们可以像最小生成树的Kruskal算法一样对点维护一个并查集。对于把u, v两点缩成一点的过程，我们可以把v的边表直接接到u的边表后面，并在并查集中把v指向u。每时每刻对于图中的某条边，这条边的两个端点都从并查集中查询一下得到。这样复杂度就是$O(|E|\alpha(|V|))$。

还有一个复杂度更优的方法。在排列上二分什么时候开始图上只有两个点。比如二分的第一步，我们计算依次操作前一半的$|E|/2$条边后是否只剩两个点了。这个可以用DFS来判断。如果是，那么二分左半部分；如果不是，就直接把左半边这些边连接的点在图上缩起来，重新计算右半边所有边在新图中的两个端点，然后二分右半部分。这些操作都可以$O(|E|)$解决。由于每次边数减少一半，所以最终复杂度仍是$O(|E|)$。

4.2.3 带权的图

对于某条带权的边，我们可以把它当成是很多单位边权的重边，这样就可以转化为不带权的图。只是在随机选边的过程中我们要修改概率的分配。一条边被选中的概率应该为该边的边权占所有边边权之和的比例。为此，我们需要维护一个边的有序集合来维护选边操作：每次在剩下的所有边边权之和的范围内随机一个数，找到集合中第一条边使得它以及它之前所有边的边权之和不小于这个数，把这条边作为选择的边进行缩边，然后删去这条边。这样的一个集合可以用树状数组来维护。复杂度为$O(|E|\log|E|)$。

5 应用

5.1 例题1：[ZJOI2011]最小割

给定一张带权无向图。有q次询问，每次给出一个数x，求有多少点对满足

两点之间的最小割容量不超过x。

5.1.1 做法

这是一个关于任意两点间最小割的问题。我们可以构出Gomory-Hu树。然后按树上边的权值从小到大做，每次用并查集合并这条边连接的两个连通块，同时把两个连通块内的点数之积加到答案里。这样就求出了不超过每种可能的权值的点对数了。询问时只要在这个数组上lower_bound一下就好了。

5.2 例题2：Codeforces Round #200 Div.1 E. Pumping Stations

给定一张带权无向图。求一个所有点的排列，使得排列中相邻两个点之间的最大流之和最大。

5.2.1 做法

首先，最大流等于最小割。同样，我们构出Gomory-Hu树。下面我们证明结论：最优排列的答案等于Gomory-Hu树上所有边权之和。一个排列在树上就对应了一条路径。考虑树上边权最小的那条边（如果有多条选任意一条）。那么从这条边一侧的某个点走到另一侧的某个点的权值必然就是这条边的权值。如果一条路径不止一次跨过这条边，那么必然可以通过调整使得答案不变劣：在整条路径上跨过这条边的地方断开，我们会得到若干段不跨过的路径。不妨设跨过了k次，那么就会得到$k+1$段路径。我们可以把同一侧的路径都首尾相接拼起来，这样就在两侧各得到了一条不跨过中间这条边的路径。然后再把两侧的路径首尾相接，这就必然会跨过中间的边。经过调整后，我们就只跨过中间边一次了。而由于中间这条边是最小的，不跨过这条边一定不比跨过劣。因此，这样调整过后必然不会变劣。然后，我们对这条边的左右两边递归进行这个推导。这样就证明了结论，递归的同时也构造出了一组最优解。

5.3 例题3：UVALive 5099 Nubulsa Expo

给定一张无向图，对于某个确定的源，求出与所有可能的汇的最大流的最小值。

5.3.1 做法

首先最大流等于最小割。显然，答案就是全局最小割。因为一个全局最小割分成的两部分中必然有一部分是那个源所在的，只要选择另一部分中的某个点作为汇，就可以把全局最小割作为源汇的最小割了。

5.4 例题3：带非负边权平面图最小环

给定一张平面图，平面图中每条边都有一个非负的权。求权最小的环。

5.4.1 一个做法

构出平面图的对偶图，那么平面图的一个环就对应对偶图的一个割，环的权就是割的权。因此，只要求出对偶图的全局最小割即可。

可能存在复杂度更优的算法。

6 总结

在信息学竞赛中遇到这类问题时，往往首先需要通过题中给出的条件将问题模型转化为无向图最小割的模型，接着，或直接套用合适的算法或结构，或利用无向图最小割的种种性质，再进一步把问题转化为能用其他算法或数据结构解决的问题。

对于这类问题，学界有更多深入的研究，还有其他很多可以高效解决这类问题的算法。本文只展示了冰山一角，有兴趣的读者可以继续深入研究。希望本文能起到抛砖引玉的作用。

7 感谢

感谢中国计算机协会提供学习和交流的平台。

感谢绍兴市第一中学的陈合力老师、董烨华老师、邵红祥老师、游光辉老师多年来给予我的关心和指导。

感谢国家集训队教练余林韵和陈许旻的指导。

感谢清华大学的俞鼎力、董宏华、何其正、张恒捷、王鉴浩、贾越凯等学长给我的帮助。

感谢绍兴市第一中学的任之洲、洪华敦、陶渊政、叶钊珲、李洋等同学对我的帮助和启发。

感谢其他帮助或启发过我的老师和同学。

感谢父母对我的关心和支持。

参考文献

[1] R. E. Gomory, T. C. Hu. Multi-terminal network flows. Journal of the Society for Industrial and Applied Mathematics, vol. 9, 1961.

[2] Dan Gusfield (1990). "Very Simple Methods for All Pairs Network Flow Analysis". SIAM J. Comput. 19 (1): 143-155.

[3] David R. Karger, Global Min-cuts in $\mathcal{RNC}$, and Other Ramifications of a Simple Min-Cut Algorithm, Department of Computer Science, Stanford University.

[4] David Morrison, Chandra Chekuri, "Gomory-Hu Trees", CS 598CSC: Combinatorial Optimization Lecture 6.

[5] Uri Zwick, Lecture notes for "Analysis of Algorithms": Global minimum cuts. School of Computer Science, Tel Aviv University.

[6] Gomory-Hu tree - Wikipedia

[7] Karger's algorithm - Wikipedia

[8] 演算法筆記- Cut，國立台灣師範大學資訊工程學系

浅谈线性规划在信息学竞赛中的应用

宁波市镇海中学 邹逍遥

摘 要

线性规划在实际生活中应用广泛，但是线性规划在目前的OI界普及度不够，所以有关的题目也很少。本文简单介绍了线性规划的实现，并列举了线性规划在不同类型的OI题中的应用。

引言

z 线性规划在早期的算法竞赛中较少被提及，一般是以模板题的形式出现。由于线性规划的应用非常广泛，近几年的OI比赛中已经出现了许多需要使用（或可以使用）线性规划解决（或获取部分分）的题目。

单纯形算法实现起来非常简单，运行效率也不差，在解决一些建模较复杂的网络流问题上并不输给其他算法。同时线性规划的用途比网络流更广，对于某些题目虽然直接使用线性规划解决是错误的，但是可以在出题人没有认真造数据的情况下侥幸AC。

由于网上关于这方面证明的参考资料非常多，所以本文中略去了大部分的证明，介绍算法处只是简单介绍了如何实现。

本文分为五个章节，第一章介绍了线性规划的定义；第二章介绍了常用的解决线性规划的算法——单纯形法；第三至五章分别介绍了线性规划在不同类型的OI题中的应用。

1 什么是线性规划

实际生活中有很多问题都是这样的形式：它们需要最大化或者最小化一个目标；它们通常面临资源，时间等多方面的限制。假如把这些问题的目标简化成一个线性的函数，把限制表示成一些线性的等式或者不等式，那么这些问题就可以被描述成线性规划问题。

1.1 一般线性规划

在一般线性规划问题中，我们希望最优化一个满足一组线性不等式约束的线性函数。

定义**线性函数**为满足$f(x_1, x_2, \cdots, x_n) = \sum_{i=1}^{n} a_i x_i$的函数$f$，其中$a_i$为实数，$x_i$为变量。

定义**线性等式**为$f(x_1, x_2, \cdots, x_n) = b$，其中$b$为实数，$f$为线性函数。

定义**线性不等式**为$f(x_1, x_2, \cdots, x_n) \le b$或$f(x_1, x_2, \cdots, x_n) \ge b$，其中$b$为实数，$f$为线性函数。

后文将使用**线性约束**或简称**约束**来表示线性等式和线性不等式。

正式地说，线性规划是一个需要最小化或最大化一个受限制于有限个数的约束的线性函数的问题。

1.2 标准型与松弛型

了解了线性规划的定义之后，我们需要用更规范的形式来表示一个线性规划。

常用的有两种形式：**标准型**与**松弛型**。简单来说，在标准型中，线性规划被表示成约束均为线性不等式的线性函数最大化；在松弛型中，线性规划被表示成约束均为线性等式的线性函数最大化。

我们一般将标准型表示成这种形式：

最大化$\sum_{i=1}^{n} c_i x_i$，满足约束

$$\sum_{j=1}^{n} a_{i,j} x_j \le b_i \qquad i = 1, 2, \cdots, m$$

$$x_i \ge 0 \qquad i = 1, 2, \cdots, n$$

也可以用一个更紧凑的形式表示：

最大化$\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$，满足约束

$$\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}$$
$$x_i \ge 0$$

其中$\boldsymbol{A} = (a_{i,j})$是一个$m \times n$的矩阵，$\boldsymbol{b} = (b_i)$是一个$m$维向量，$\boldsymbol{c} = (c_j)$和$\boldsymbol{x} = (x_j)$是$n$维向量。

有的时候也会写成$\{max\ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} | \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}, x_i \ge 0\}$

松弛型的一般形式是这样：

最大化$\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$，满足约束

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$
$$x_i \ge 0$$

对于任意一个线性规划问题，都可以将其中的一个等式用两个不等式代替而答案不变。同样，对于任意一个不等式，建立一个辅助变量y_i，变为$\sum_{j=1}^{n} a_{i,j}x_j \pm y_i = b_i$，然后加入$y_i \ge 0$的限制即可转化为一条等式加一条非负性约束的形式。

对于任意一个最小化的线性规划问题，可以加入一个辅助变量$p =$想要最小化的值，然后变为最大化$-p$，就可以实现最小化和最大化问题的互相转变。

那么任何一个线性规划问题都能被写成标准型或者松弛型的形式，也就是说松弛型和标准型只是两种更为规范的表示方法而已。

为了表述更加易懂，后文的线性规划并不会列成非常规范的形式，需要求解时只需要将它转化一下即可。

1.3 如何将一般的问题转化为线性规划

1.3.1 最短路问题

考虑最短路问题的最简单形式：给定一个n个点m条边的带权有向图，第i条边从u_i出发连向v_i长度为l_i，求s到t的最短路。

设n个变量d_i表示s到i的最短路，那么可以列出如下的线性规划：

最大化d_t，满足约束

$$d_s = 0$$
$$d_{v_i} \le d_{u_i} + d_{l_i} \qquad i = 1, 2, \cdots, m$$

1.3.2 最大流

最大流问题：给定一个n个点的有向图，i到j之间的流量限制为$l_{i,j}$，方便起见，设流量满足如果$l_{i,j} > 0$则$l_{j,i} = 0$，求s到t的最大流。

设n^2个变量$f_{i,j}$表示i到j的实际流量，那么可以列出如下的线性规划：

最大化$\sum_u f_{s,u}$，满足约束

$$f_{u,v} \le l_{u,v} \qquad u, v = 1, 2, \cdots, n$$
$$f_{u,v} = -f_{v,u} \qquad u, v = 1, 2, \cdots, n$$
$$\sum_i f_{u,i} = 0 \qquad u \in \{1, 2, \cdots, n\} - \{s, t\}$$

1.3.3 多商品流

给定一个n个点的有向图，i到j之间的流量限制为$l_{i,j}$，方便起见，设流量满足如果$l_{i,j} > 0$则$l_{j,i} = 0$。

有p条商品运输路线，第i条路线要从s_i向t_i运d_i个商品，求是否存在一种可行解。

这个问题用线性规划解决非常方便，但是没有除了线性规划以外的高效解法。

设pn^2个变量$f_{k,i,j}$表示商品k从i到j的实际流量，那么可以列出如下的线性规划：

最大化0（由于只需要求可行解所以不需要目标函数），满足约束

$$f_{k,u,v} \le l_{u,v} \qquad u, v = 1, 2, \cdots, n; k = 1, 2, \cdots, p$$
$$\Sigma_k f_{k,u,v} \le l_{u,v} \qquad u, v = 1, 2, \cdots, n$$
$$f_{k,u,v} = -f_{k,v,u} \qquad u, v = 1, 2, \cdots, n; k = 1, 2, \cdots, p$$
$$\Sigma_i f_{k,u,i} = 0 \qquad u \in \{1, 2, \cdots, n\} - \{s_k, t_k\}; k = 1, 2, \cdots, p$$
$$\Sigma_i f_{k,s_i,v} = d_i \qquad k = 1, 2, \cdots, p$$

检查该线性规划是否有解即可。

2　单纯形算法

本章中使用n代表变量个数，m代表约束个数。

2.1　单纯形算法简介

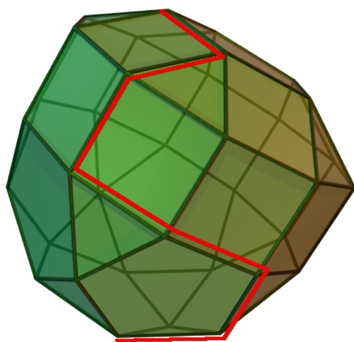
单纯形算法是求解线性规划的古典方法。在最坏情况下，它的时间复杂度为指数级。然而要把单纯形卡到指数级别需要指数级大小的权值，而算法竞赛中涉及的问题一般都有一个具体的模型，并且权值也大多是一定范围内的整数，在满足如此多的条件下就不可能把单纯形卡到指数级了。

由于线性规划的普及度并不高，而卡线性规划比实现线性规划困难得多，不可能每道题的出题人都精通卡单纯形的技巧。而随机情况下单纯形的期望调用pivot过程的次数是O(约束个数)次，也就是说实践中能在1s内跑出范围为几百的数据，足够满足大部分需求了。而在数据满足更多限制的情况下甚至能够只用低于约束个数次的pivot操作就可以获得解。

单纯形算法接受一个标准型线性规划$\{max\ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}|\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}, x_i \ge 0\}$作为输入，返回“无解”（Infeasible），“无界”（Unbounded）或一个取到最优值的方案。

可以将每一种变量的取值对应一个n维空间中的点，那么每一个不等式对应一个n-1维超平面，合法解的集合就对应一个n维的凸多胞体。

单纯形算法首先选取一个初始基本可行解（可以理解为找到一个在凸多胞体上的点），如果找不到则返回无解。接着不停地对这个解进行**转轴**（Pivot）操作（可以理解为走向该点所在的单纯形上的另一个顶点），每次操作保证答案增加，当无论走哪一条边的收益都是负的时候就找到了最优解。



可以证明这个算法一定会停止（凸多胞体上的点是有限的），并且这个算法返回的值一定正确（假如无法继续移动了那么当前点一定是最优解）。

那么也就是说，单纯形算法可以分为以下几个步骤：

- 将输入的线性规划转为松弛型
- 执行初始化过程找到一个初始解，找不到返回无解
- 执行最优化过程找到最优解或者返回无界

接下来将详细介绍每个步骤。

2.2 将输入的线性规划转为松弛型

新建m个变量$x_{n+1}, x_{n+2}, \cdots, x_{n+m}$，并使$x_{n+i} = b_i - \sum_{j=1}^{n} a_{i,j}x_j$，$x_{n+i} \geq 0$。那么这个线性规划就被写成了如下的形式：

最大化$\sum_{i=1}^{n} c_i x_i$，满足约束

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{i,j}x_j \qquad i = n+1, n+2, \cdots, n+m$$

$$x_i \geq 0 \qquad i = 1, 2, \cdots, n+m$$

其中$x_1, x_2, \cdots, x_n$称为**非基变量**，$x_{n+1}, x_{n+2}, \cdots, x_{n+m}$称为**基变量**。

在这种松弛型表示下，将所有的非基变量置为0，所有基变量置为b_i，得到的这组解就是该松弛型代表的解。单纯形算法执行的过程中会不停地选取不同的基变量集合来将线性规划表示为不同形式的松弛型，最终使所有的c_i都变为负数就找到了最优解。

那么可以发现假如一开始所有的b_i都是非负的那么$\{0, 0, \cdots, 0\}$是一组合法解，初始的松弛型表示是合法的，所以可以跳过初始化这一步。否则就需要先执行初始化过程来得到一个基本可行解才能执行最优化过程。

2.3 转轴操作

转轴操作的作用是将一个基变量与一个非基变量进行互换。在初始化和最优化过程中都需要使用它。

方便起见这里假设每次操作时保证$x_1, x_2, \cdots, x_n$为非基变量。

转轴操作接受两个数l,e作为输入，然后交换x_{n+l}和x_e的位置，即x_{n+l}变为非基变量，x_e变为基变量。

一开始有等式$x_{n+l} = b_l - \sum_j a_{i,j}x_j$

移项得$a_{i,e}x_e = b_l - \sum_{j\neq e} a_{i,j}x_j - x_{n+l}$

两边同除$a_{i,e}$得$x_e = \frac{b_l-\sum_{j\neq e} a_{i,j}x_j-x_{n+l}}{a_{i,e}}$

用这条等式代替原等式，然后将这条等式依次代入其他等式和要优化的结果得到只关于$\{x_1, x_2, \cdots, x_n\} - \{x_e\} + \{x_{n+l}\}$的等式。

最后交换x_{n+l}和x_e的下标即可。

注意假如最后需要输出方案那么要记下当前的每个变量分别对应原来的哪个变量。

2.4 最优化过程

假设我们已经获得了这个线性规划的一个基本可行解，那么执行最优化过程即可获得结果：

- 选择一个满足$c_e \geq 0$的非基变量x_e
- 找到那个满足$a_{l,e} > 0$且$b_l/a_{l,e}$最小的基变量x_{n+l}
- 执行$pivot(l, e)$，然后回到第一步

即每次选取一个增大后能使当前状态变优（答案不一定增加）的变量，将它增加到有一个基变量变成0为止，然后互换它们的位置。

这里可以加上一个贪心优化：枚举所有符合条件的(l, e)，选取使答案增大最多的那个进行pivot操作。由于pivot的复杂度为$O(mn)$，所以这样并不会影响总复杂度，但是可以使pivot次数降低许多。

这里假如直接任意选取有可能会出现死循环，可以通过每次选取合法的e中标号最小的，同时选取合法的l中标号最小的就可以避免死循环。

该过程在以下两种情况下会终止：

- 找不到一个合法的e，可以证明此时已经找到了最优解。
- 找不到一个合法的l，此时的最优解是无穷大，返回“无界”。

2.5 初始化过程

为了找到一个初始可行解，需要新建一个辅助线性规划：

最大化$-x_0$，满足约束

$$\sum_{j=1}^{n} a_{i,j}x_j \le b_i + x_0 \qquad i = 1, 2, \cdots, m$$

$$x_i \ge 0 \qquad i = 0, 2, \cdots, n$$

容易看出当且仅当该线性规划最优解为0时原线性规划有解。

将这个辅助线性规划转成松弛型，然后找到一个b_l最小的l，执行$pivot(l, 0)$，就获得了该辅助线性规划的初始解，执行最优化过程就可以解出辅助线性规划的最优解。

执行完初始化过程后直接去掉x_0并接着执行最优化过程即可。

代码实现的时候可以不用真正加入一个新的变量，直接在原线性规划上进行转轴操作即可。

2.6 例题

例题1. *Happiness*[1]

要买一样的m份套餐给m个人吃。套餐由n种食品组成。已知每种食品的单位价格，每个人每得到一单位的某种食品获得的满足度，每个人满足度的上限。求在不超过任何人的满足度上限的条件下，最多能花多少钱。

这题属于线性规划的模板题：将每种食品当成一个变量每个人当成一条不等式即可。

同时$\{0, 0, \cdots, 0\}$是一个满足条件的初始解，所以本题甚至连初始化过程都不需要。

例题2. *FLYDIST*[2]

给定一个n个点m条边的整数边权无向图，你需要使每条边的边权变成它连接的这两个点之间的最短路长度，而且每条边的边权改变之后必须是非负，设边权原来为O_i，改变之后为N_i，代价为$\sum |O_i - N_i|$，求最小化代价。

$n \le 10, O_i \le 20$

[1]来源：Uva 10498

[2]来源：Codechef FEB12

首先将每对点之间的最短路当成变量，设为$d_{i,j}$，并对每条边设两个变量I_k, D_k，表示最终边权为$O_k + I_k - D_k$。那么要最小化的就是$\sum I_k + D_k$。

对于每条边k，设连接的点为x_k, y_k，那么对于每个j都有$d_{x_k,j} \le O_k + I_k - D_k + d_{y_k,j}$，同理反向也有对应的一个不等式。同时为了保证每条边边权非负需要加上$O_k + I_k - D_k \ge 0$，使用单纯形解出这个线性规划即可。

3 线性规划在网络流问题中的应用

3.1 全幺模矩阵

全幺模矩阵（totally unimodular matrix）是指任何一个行数和列数相等的满秩子矩阵行列式的值都是1或-1的矩阵（但它自己的行列数不需要相等）。

还有另外一个判定全幺模矩阵的常用方法（充分条件）：同时满足以下四个条件的矩阵$\boldsymbol{A}$是全幺模矩阵

- $\boldsymbol{A}$仅由-1,0,1构成
- $\boldsymbol{A}$的每列至多包含两个非零数
- $\boldsymbol{A}$的行可以被分为两个集合B,C
- 假如有一列包含了两个同号非零数那么它们所在的行被分在不同的集合，假如有一列包含了两个异号非零数那么它们所在的行被分在同一个集合。

全幺模矩阵有一个非常重要的性质：假如一个线性规划$\{max\ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} | \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}, x_i \ge 0\}$中的$\boldsymbol{A}$为全幺模矩阵，那么可以证明在执行单纯形的过程中涉及到的所有约束中的系数的值都是1,0,-1中的一个。

也就是说，每个变量限制为实数和限制为0或1解出来得到的答案是一样的，所以直接执行线性规划就能得到相同的整数规划问题的正确答案。

同时在这种情况下单纯形的过程中只需要使用乘法，并且可以用链表优化去除那些值为0的项进行常数优化从而大大提升速度。

有很多的问题可以转成全幺模的线性规划问题：比如任何一个最大流问题，任何一个最小费用最大流问题，某些非常特殊的dp问题等等。

有一些网络流题本来就没有比较显然的建图，而是要通过先列出线性规划再经过转对偶等一系列转化最后将线性规划问题转化成网络流解决。但是既然

能够使用网络流解决那么建出的矩阵一定是全幺模的，这时候线性规划就是解决这类问题的一种通用的解法，不需要再对问题模型进行深入的分析。

3.2 线性规划对偶性

线性规划对偶性（linear-programming duality）是指对于任何一个线性规划$\{max\ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}|\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}, x_i \ge 0\}$，它的最优解与$\{min\ \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x}|\boldsymbol{A}^{\mathrm{T}}\boldsymbol{x} \le \boldsymbol{c}, x_i \ge 0\}$的最优解相同。

将最大流问题和最小割问题分别使用线性规划表示出来以后它们就互为对偶问题，所以最大流最小割定理可以使用线性规划对偶性证明。

在大部分的题目中，原问题和对偶问题中通常会有一个能直接找到初始解不需要初始化从而降低码量。

3.3 例题

例题3. 志愿者招募[3]

给出一个长度为$n(n \le 1000)$的序列和$b_1 \sim b_n$，有$m(m \le 10000)$种区间，第i种区间每选取一个可以使l_i到r_i这个区间权值+1并付出a_i的代价。求最后的序列每个位置的权值都不小于b_i的最小代价。

首先列出线性规划式子：

最小化$\sum_{i=1}^{m} a_i x_i$，满足约束

$$\sum_{l_j \le i \le r_j} x_j \ge b_i \qquad i = 1, 2, \cdots, n$$

$$x_i \ge 0 \qquad i = 1, 2, \cdots, m$$

可以发现每个变量在约束中出现的位置是连续的。对约束进行差分之后每个变量仅出现在两个约束中。

对每个约束建一个点，将这条约束视为这个点的流量平衡，那么每个变量就代表一条边，每条约束的常数就代表源和汇的补流。建出图后跑最小费用最大流即可。这是本题的官方费用流解法。

[3]来源：NOI 2008

但是这个建图并没有那么显然，假如你并没有看出应该怎么建图，而是直接使用单纯形解决，那么只需要将这个线性规划转对偶，变成

最大化$\sum_{i=1}^{n} b_i x_i$，满足约束

$$\sum_{l_i \le j \le r_i} x_j \le a_i \qquad i = 1, 2, \cdots, m$$

$$x_i \ge 0 \qquad i = 1, 2, \cdots, n$$

就列出了一个n个变量m条约束的线性规划，而且$\{0, 0, \cdots, 0\}$就是它的一个初始解，不需要写初始化过程。加上乘法优化和链表优化之后运行速度也非常快。

实现效率上是zkw费用流快于单纯形快于暴力最短路费用流，这三种做法都能通过本题的所有数据（最短路费用流写的差可能会TLE）。

例题4. 防守战线[4]

有一个长度为$n(n \le 1000)$的序列，第i个点上权值每加一的代价是a_i。给定$m(m \le 10000)$个区间l_i到r_i和要求值b_i，求最后的序列每个l_i到r_i的区间内的权值和都不小于b_i的最小代价。

最小化$\sum_{i=1}^{n} a_i x_i$，满足约束

$$\sum_{l_i \le j \le r_i} x_j \ge b_i \qquad i = 1, 2, \cdots, m$$

$$x_i \ge 0 \qquad i = 1, 2, \cdots, n$$

由于$\{0, 0, \cdots, 0\}$并不是它的初始解，直接使用线性规划解决效率较低代码复杂度较高。

将这个线性规划转对偶之后变成

最大化$\sum_{i=1}^{m} b_i x_i$，满足约束

$$\sum_{l_j \le i \le r_j} x_j \le a_i \qquad i = 1, 2, \cdots, n$$

$$x_i \ge 0 \qquad i = 1, 2, \cdots, m$$

和上题一样直接使用整数单纯形解决即可。

[4]来源：ZJOI 2013 Day1

3.4 总结

近年来网络流题的各种简单模型已经被大家掌握了，所以出的题目一般都是考建模。而且由于网络流问题的特殊性，没有复杂度比较接近的暴力算法，又因为网络流算法的运行速度本来就比理论下界快很多，题目中的边数和点数都比较大，出题人也不能保证自己的程序一定能在m,n限定的所有图上跑进时限，所以网络流题的时间限制一般比较宽松，数据也一般不会刻意针对某种算法，更不会刻意去卡常数。

对于上面两道题的1000×10000的随机数据，单纯形的期望pivot执行次数不到100次。在效率和代码复杂度上都优于普及最为广泛的最短路网络流算法，美中不足的是空间复杂度为$O(mn)$，假如题目的空间限制比较紧就无法通过了。

不过由于单纯形可以省去思考建图的时间，在考试时间有限的情况下具有非常大的优势。

4 线性规划在特殊整数规划问题中的应用

整数规划是指最后的解中的变量都必须是整数的线性规划。0-1规划是整数规划的特殊形式，即要求每个变量都属于$\{0,1\}$时的最优解。

由于生活中大多数的问题表示成线性规划之后要求的是整数解，所以整数规划的应用更加广泛。

但是整数规划目前还没有通用的多项式解法（比如精确覆盖问题就可以规约到0-1规划），所以OI竞赛中能使用线性规划直接解决的一般都是能转化成全幺模矩阵的问题。不过对于那些不能转化成全幺模矩阵的问题，假如不精心构造数据而是使用随机数据的话，线性规划也能获得很高的正确率。

例题5. *幻想乡Wifi搭建计划*[5]

平面上有一个长无限，宽为R的长方形，内部有$n(n\le 100)$个点。

有$m(m\le 100)$个圆心在长方形外部的圆，半径均为R，第i个圆权值为a_i。

求圆对点的最小权覆盖。

对于40%的数据，所有圆的圆心都在长方形上方。

假如任意确定每个圆覆盖的点的话这就是一个精确覆盖问题。虽然它是NPC的，但是可以利用线性规划解出它的实数解：

[5]来源：ZJOI 2015 Day2

最小化$\sum_{i=1}^{m} a_i x_i$，满足约束

$$\sum_{j能覆盖i} x_j \geq 1 \qquad i = 1, 2, \cdots, n$$

$$x_i \geq 0 \qquad i = 1, 2, \cdots, m$$

如果这个线性规划的矩阵是全幺模的话那么可以证明按实数解出来的最优解和原问题的最优解相同。

对于前40分，可以证明这个做法是正确的，证明方法可以仿照40分的dp做法，即一定存在一种方法将点排序，使得每个圆覆盖的都是一个区间内的点。然后问题就变成了区间覆盖问题，容易看出矩阵是全幺模的。

不过可惜的是本题的矩阵并不是全幺模的，所以会出现错误。不过不经过构造的数据要卡掉单纯形比较困难，比赛中直接实现整数单纯形能获得70分。

例题6. 文学[6]

给定平面上n个点和m个半平面，其中第i个半平面的权值为a_i，求半平面对点的最小权覆盖。

对于40%的数据，$a_i = 1$。

本题可以像上题一样直接转化为精确覆盖模型然后列出线性规划。

不过和上题一样，这个矩阵并不是全幺模的，一个$n = 3, m = 3$的数据就可以轻松hack掉：三个半平面分别覆盖$\{1, 2\}, \{1, 3\}, \{2, 3\}$，权值都是2，解出来的答案就是3，而正解是4。

但当时出题人并没有意识到单纯形这种做法，没有造针对性的数据，所以在最初版本的数据中单纯形通过了所有测试点获得了100分，出题人在比赛中途重新造数据才把单纯形卡到了80分。可见单纯形还是一种非常有效的骗分手段。

同时出题人给了许多$a_i = 1$的部分分，在这种条件下直接输出实数解四舍五入的结果就不能轻易地被随机数据hack掉了，出题人也觉得卡这部分工作量太大就没有卡。

例题7. *Red and Black Tree*[7]

[6]来源：清华集训2014

[7]来源：Codeforces 375E

给定一棵$n(n \le 500)$个点的树，第i个点的权值为$a_i(a_i \in \{0,1\})$，并且恰好有m个点权值为0，剩下的点权值为1。

现在需要选出一个包含m个点的点集，使得对于树上任意一个点都存在一个点集中的点与这个点的距离不超过k，在此基础上要求选出的m个点权值和最小。

直接根据题意列出线性规划：

最小化$\sum_{i=1}^{n} a_i x_i$，满足约束

$$\sum_{i} x_i = m$$
$$\sum_{dis(i,j) \le k} x_j \ge 1 \qquad i = 1, 2, \cdots, n$$
$$x_i \ge 0 \qquad i = 1, 2, \cdots, n$$

首先任意确定一个点作为根，然后任取一个离这个根最远的叶子节点，可以发现能覆盖这个叶子节点的点中覆盖范围是两两包含的。利用这个性质可以很容易地证明这个线性规划得出的解和整数规划的解是相同的。

不过答案相同并不是因为这个线性规划的矩阵是全幺模的，只是在这个题目条件下该线性规划的最优解一定和整数线性规划相同，假如修改一下题目要求输出一种方案就不能使用线性规划解决了。

也就是说这个线性规划的约束所代表的多胞体并不是每个顶点都是整点，而只是至少有一个整点的答案和实数限制下的最优解相同。

这道题还有一种$O(n^3)$的树形dp做法，不过实际效率比单纯形算法慢很多。

5 线性规划在博弈问题中的应用

上面介绍的这些题目都有其他比较优秀的做法，而线性规划的复杂度无法正确估计，空间消耗较大，导致了它在解决这类问题上出场率并不高。不过线性规划是解决接下来要提到的这类问题的一个有力工具。

5.1 纳什均衡

在一个二人或多人参与的博弈游戏中，如果在一个情况下没有人可以通过只改变他们自己的决策而获得更高的收益，那么这个策略组合被称为**纳什均**

衡。

假如在一个策略组合中，每个人的策略都是固定的，那么这个纳什均衡被称为**纯策略纳什均衡**，假如每个人的策略都是由所有策略以一定概率组合起来的策略，那么这个纳什均衡被称为**混合策略纳什均衡**。

可以证明，对于任意一个决策数有限的零和游戏（所有人的收益和为0）都至少存在一个混合策略纳什均衡。

对于一个二人零和游戏，很容易将问题转化为线性规划问题从而利用单纯形法解出纳什均衡。

5.2 例题

例题8. 读心[8]

有$n(n \leq 50)$个选项，A和B要分别从中选择一种。给定所有n^2种情况下A的收益矩阵v（B的收益为$-v_{i,j}$），保证$v_{i,j} \in \{-1, 0, 1\}$，保证$v_{i,j} = -v_{j,i}$。

现在A需要给出一种选取每个策略的概率表，使得不论B选择哪个策略，A的期望收益都不小于0。

由于保证了矩阵对称，所以A和B的纳什均衡是一样的，也就是说一定存在一种满足题意的方案。

将每个决策的概率设成变量，那么容易看出约束就是对方每个决策收益不超过0，以及所有决策的概率之和等于1。

稍微修改一下就可以列出线性规划：最大化$\sum_{i=1}^{n} x_i$，满足约束

$$\sum_{i=1}^{n} x_i \leq 1$$
$$\sum_{j} v_{i,j} x_i \geq 0 \qquad i = 1, 2, \cdots, n$$
$$x_i \geq 0 \qquad i = 1, 2, \cdots, n$$

由于本题矩阵的特殊性，线性规划的运行速度十分快，跑过n为500左右的数据没什么问题。不过当时这道题的官方解法是调整法，所以数据范围非常小。

[8]来源：BJOI 2011 Day2

例题9. *TheKFactor*[9]

有一个长度为$n(n \leq 50)$的数组，你在数组的最左端要向右走。每个格子内有一个整数v_i代表最终停在这个格子上的收益。

系统会在1到n中的某一个格子后面设置一道墙，也就是说存在一个i使得你无法走到i右边的格子，当你试图走到那些格子的时候最终会停留在i号格子上。

设墙最终在i的概率为a_i，a满足对于任意一个j都有$\sum_{i=1}^{j} a_i \leq \frac{j}{n}$，并且有$\sum a_i = 1$。

现在你需要确定一个数组b_i代表你最终选择停在每个格子上的概率（显然必须满足$\sum b_i = 1$），要求最大化（在a任意确定的情况下）你的最小得分。

直接根据每种可能的a数组列不等式显然是不现实的，那么要考虑一下如何转化问题。

这题是比较显然的一个二人零和博弈模型。稍微转化一下然后运用上面的结论就可以证明一定存在一个纳什均衡。也就是说根据每种可能的b数组列出不等式解得的答案和根据a数组解是一样的。

由于b数组是没有前缀和限制的，也就是说只需要保证每个b_i分别等于1的这n种情况的答案满足条件那么任何一个b数组都满足条件。

不等式就很好列了：设ans为你的最大期望得分，那么

最小化ans，满足约束

$$\sum a_i = 1$$

$$\sum_{i=1}^{j} a_i v_i + \sum_{i=j+1}^{n} a_i v_j \leq ans \qquad j = 1, 2, \cdots, n$$

$$\sum_{i=1}^{j} a_i \leq \frac{j}{n} \qquad j = 1, 2, \cdots, n$$

$$a_i \geq 0 \qquad i = 1, 2, \cdots, n$$

解这个线性规划即可。

例题10. 最优决策2[10]

A和B分别获得一个$1 \sim n$且保证不同的随机整数，然后A在$1 \sim m$中选出一个数报出，B在“2”和“A报出的数”这两个选项中选择一个数报出。

[9]来源：Topcoder SRM 682 Div1 Hard

[10]来源：2016集训队互测

假如报出的两个数相同（设都为x），那么随机整数大的一方获得x的收益，另一方获得$-x$的收益。

假如报出的两个数不同（设为x和y，$x < y$），那么选y的一方获得x的收益，另一方获得$-x$的收益。

现在你需要为A确定一个策略表（即拿到某个数字时有多少的概率选某个选项），最大化最坏情况下A的期望收益。

设$n \times m$个变量$x_{i,j}$，表示拿到i时有多少的概率选j，再设m个变量c_i表示B在看到A的策略是i时的总收益。

那么问题是如何计算c_i。注意到B不论看到的是哪一个数字都一定会选取一个k，然后拿到小于k的数字选择放弃，否则选择进攻。那么c_i就是B选择的所有$n + 1$种方案中收益的最小值。

设$s_{i,j,k,p}$为A拿到数字i选择选项j，B拿到数字k并且B选择进攻$(p = 1)$/放弃$(p = 0)$的收益（当$i = k$时$s_{i,j,k,p} = 0$），那么可以列出线性规划：

最大化$\sum_i c_i$，满足约束

$$\sum_i x_{i,j}(\sum_{k=1}^{p-1} s_{i,j,k,0} + \sum_{k=p}^{n} s_{i,j,k,1}) \geq c_j \qquad j = 1, 2, \cdots, m; p = 1, 2, \cdots, n + 1$$

$$\sum_j x_{i,j} = 1 \qquad i = 1, 2, \cdots, n$$

$$x_{i,j} \geq 0 \qquad i = 1, 2, \cdots, n; j = 1, 2, \cdots, m$$

有了这个线性规划之后就能解出n, m为几十的情况，这样就获得了一个很大范围的表，能比较容易地猜出最后的结论从而AC本题。

感谢

感谢CCF提供学习和交流的平台

感谢国家集训队教练陈许旻，余林韵的辛勤付出

感谢父母对我的关心和照顾

感谢学军中学周子鑫同学在论文写作过程中对我的启发

感谢雅礼中学毛啸同学的验稿

感谢所有对我有过帮助和启发的同学

参考文献

[1] 刘汝佳,《算法竞赛入门经典训练指南》, 清华大学出版社。

[2] 李宇骞,《浅谈信息学竞赛中的线性规划——简洁高效的单纯形法实现与应用》, IOI2007国家集训队论文

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest Clifford Stein《Introduction to Algorithms》

[4] 全幺模矩阵的英文维基百科https://en.wikipedia.org/wiki/Unimodular_matrix

[5] 纳什均衡的英文维基百科https://en.wikipedia.org/wiki/Nash_equilibrium

[6] 单纯形算法的英文维基百科https://en.wikipedia.org/wiki/Simplex_algorithm

区间最值操作与历史最值问题

杭州学军中学 吉如一

摘 要

本文主要介绍了有关区间最值操作与历史最值问题在算法竞赛中的一系列转化与处理方法。虽然目前，有关区间最值操作、历史最值询问的问题在算法竞赛中的出现次数还比较少，但是它的出现频率正在逐渐增加，又因为选手对这一类方面涉及较少，所以在竞赛中它也有着较高的难度。本文对目前已经出现过的类似问题进行了归类分析，并介绍了一些将这一类问题转化为一些传统问题的通用方法，例如将区间最值操作转化为区间加减操作，将有关区间历史最值的和的询问转化为普通的区间和询问。希望本文可以将这一类有趣的问题带入选手们的视野中，起到抛砖引玉的作用。

1 前言

数据结构类的问题在算法竞赛竞赛中一直都是一类出现频率极高热点问题。但是因为近几年来，随着选手们对数据结构问题的研究不断深入，留给出题人的命题空间也越来越狭小。因此，在现在国内的竞赛中，大部分数据结构问题除了在不停的包装原题老生常谈外，还有一种趋势是将一些经典的序列问题放到树结构或者仙人掌结构上，强行增加选手的代码量。我认为这种行为是非常无趣的，它破坏了很多数据结构问题简洁优美的性质，让题目变得索然无味。

然而就在最近，区间最值操作（具体指区间取max操作与区间取min操作）与历史最值问题（具体指历史最大值、历史最小值与历史版本和）逐渐进入了大家的视野。实际上早在IOI2014时就出现过了有关区间最值操作的问题[1]，而历史最值问题的雏形更是能追溯到五年之前[2]，然而在那时并没有引起多少选手

[1] http://uoj.ac/problem/25

[2] http://www.tyvj.cn/p/1518

的兴趣也并没有多少人进行了深入的研究。直到去年夏天的多校联合训练，才出现了第一道难度较高的有关区间最值操作的问题[3]，之后这一类问题便渐渐多了起来，在清华集训[4]与Universal Online Judge(UOJ)[5]的比赛中都有出现。

因为这一类问题较为新颖，所以选手在初次遇到这类问题时通常难以解决。因此本文针对这一类问题，介绍了一种对区间最值操作的通用转化方法，并对常见的几类历史最值问题进行了针对性的分析，希望能对大家有所帮助。

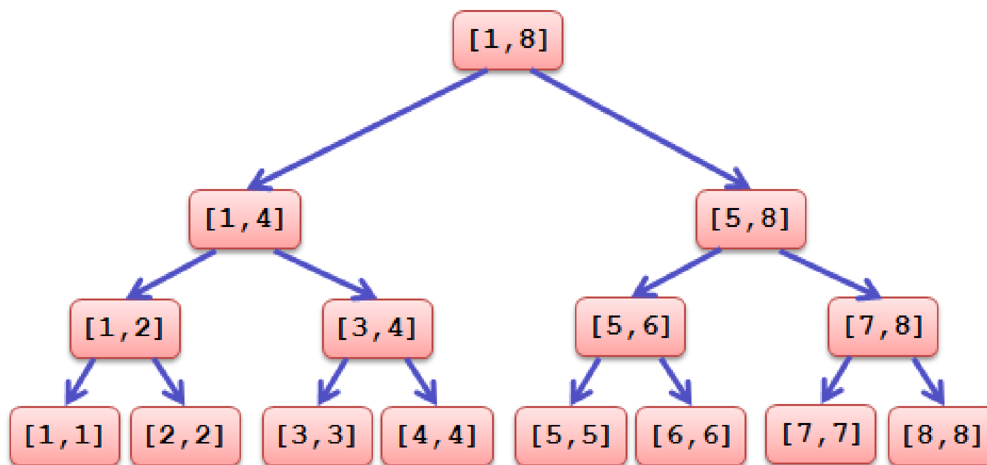
在本文的第二节中，回顾了有关线段树这一工具的定义与性质。而在第三节和第四节中，本文分别针对区间最值操作与历史最值问题进行了讨论。

2 线段树

2.1 简介

线段树是一种在算法竞赛中非常常见的数据结构。线段树的每一个节点都代表了一个区间，特别地，根节点代表总区间，叶子节点代表某个特定位置。

线段树上除了叶子节点以外的每一个节点都有两个儿子：左儿子和右儿子。如果这个节点代表的区间是$[l,r]$，那么它的左儿子代表的区间是$[l,\lfloor\frac{l+r}{2}\rfloor]$，右儿子代表的区间是$[\lfloor\frac{l+r}{2}\rfloor+1,r]$。举例来说，区间$[1,8]$构造的线段树如下图所示：



[3] http://acm.hdu.edu.cn/showproblem.php?pid=5306

[4] http://uoj.ac/problem/164

[5] http://uoj.ac/problem/169

通过利用这个结构的一些性质，我们能在线段树上快速的进行一些操作。

2.2 操作

2.2.1 单点操作

因为线段树的分治结构，孩子节点的区间长度是父节点的一半，所以线段树的树高是$O(\log n)$的。

因此对于单点操作，我们可以直接在线段树上定位这个位置所代表的叶子节点，然后更新它和它的父亲的信息，时间复杂度是单次$O(\log n)$。

2.2.2 区间询问

为了解决区间询问，我们需要在线段树上定位一个区间$[L, R]$。

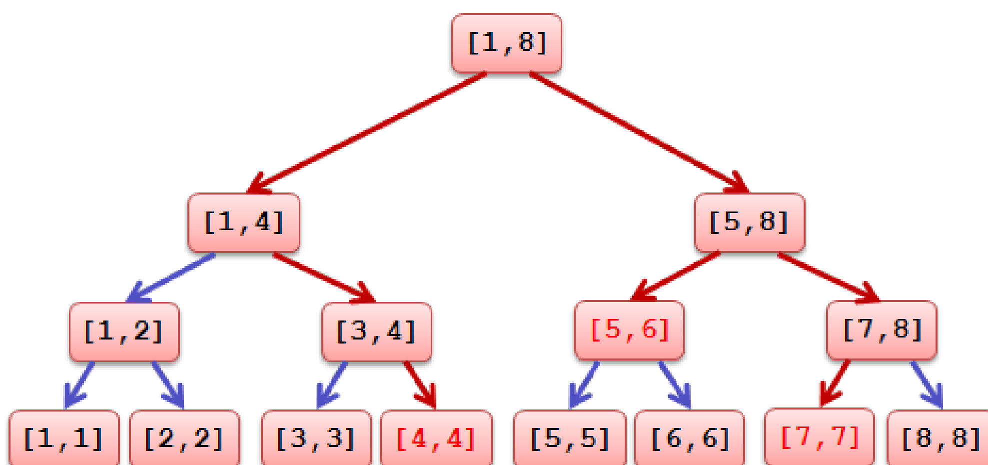
考虑这么一个算法，我们从根节点开始进行搜索，假设现在搜索到的节点的区间是$[l, r]$，那么就有三种情况：

1.如果区间$[L, R]$包含了区间$[l, r]$，即$L \le l \le r \le R$，就把它加入答案。

2.如果区间$[L, R]$与区间$[l, r]$不相交，即$L > r$或者$R < l$，那么就退出。

3.如果不满足上述两种情况，那么就分别搜索这个节点的两个孩子。

举例来说，如果我们在由区间$[1, 8]$建立的线段树中定位区间$[4, 7]$，那么过程如图所示（红色的边表示经过的边，红色字体的区间表示定位得到的区间）：



现在我们来证明这个做法的时间复杂度是$O(\log n)$的：

因为前两种情况是终止条件，只有在第三种情况的时候会继续递归，又因为线段树的每一个节点的儿子个数只有2，所以前两种情况的节点数不会超过第三种情况的两倍，因此我们只需要考虑第三种情况时的复杂度即可。

考虑线段树的每一层（即每一组深度相同的节点），显然这一层的所有节点所代表的区间是互不相交，我们把这些区间按照左端点排序。

考虑询问区间$[L, R]$，显然在排序后，与询问区间相交的所有区间的下标是连续的一段，在这一段中只有最左端的区间与最右端的区间才有可能满足条件三。

因为线段树的深度是$O(\log n)$的，每一层只有$O(1)$个节点满足条件三，所以这个方法的时间复杂度是$O(\log n)$的。

在线段树上定位了区间$[L, R]$后，我们只需要把这些节点的信息合并起来，就得到答案了。

2.2.3 区间修改

对于区间修改，我们自然不可能用单点修改的方法暴力修改每一个区间，这时就要引入懒标记的概念了。

以区间加减举例，我们对线段树中的每一个节点i，都记录一个懒标记A_i，表示给节点i所代表的区间中的所有数都加上了A_i。

现在有一个操作，需要给$[L, R]$加上x。我们先在线段树中定位这个区间，然后给这个区间打上懒标记x，具体来讲就是把A_i加上x并更新这个节点的区间和（加上区间大小乘上x），然后更新它的所有父节点的信息。

不难发现，对于线段树中的任意一个节点，只有当它的所有祖先节点的标记都是0的时候，它所记录的信息才是正确的。因此无论是修改操作还是询问操作，访问到每一个节点的时候，都应该将这个节点的标记下传到它的孩子中。给第i个节点进行标记下传相当于给它的左儿子与右儿子打上懒标记A_i，并把A_i清零。

时间复杂度与区间询问相同，是单次$O(\log n)$的。

懒标记的用处非常大，只要标记可以合并、在打上标记的时候可以快速更新线段树节点的信息，那么就能使用线段树维护，例如区间加减询问区间和、区间覆盖询问区间和、区间覆盖询问区间gcd等等。但是同样它也存在局限性，

例如区间对一个数取max询问区间和，使用基本的懒标记方法就是无法处理的。

3 区间最值操作

对一个序列A，区间最值操作具体指，给出三个数l,r,x，对所有的$i \in [l,r]$，把A_i变成$\max(A_i,x)$或者$\min(A_i,x)$。

接下来我们通过几道例题来对这类问题的处理方法进行讨论。

例题1. *Gorgeous Sequence*[6]

给出一个长度为n的数组A，接下来有m次操作，操作有如下三种：

*1.*给出l,r,x，对所有的$i \in [l,r]$，把A_i变成$\min(A_i,x)$。

*2.*给出l,r，对所有的$i \in [l,r]$，询问A_i的最大值。

*3.*给出l,r，询问$\sum_{i=l}^{r} A_i$。

数据范围：$n,m \le 10^6$

就如第一节中所说明的，因为在给一个节点打上区间取min标记的时候我们无法快速更新区间和，所以这一个问题是无法通过传统的懒标记来解决的。

考虑下面这一种解法：对线段树中的每一个节点除了维护区间和num以外，还要额外维护区间中的最大值ma、严格次大值se以及最大值个数t。

现在假设我们要让区间$[L,R]$对x取min，我们先在线段树中定位这个区间，对定位的每一个节点，我们开始暴力搜索。搜索到每一个节点时候我们分三种情况讨论：

1.当$ma \le x$时，显然这一次修改不会对这个节点产生影响，直接退出。

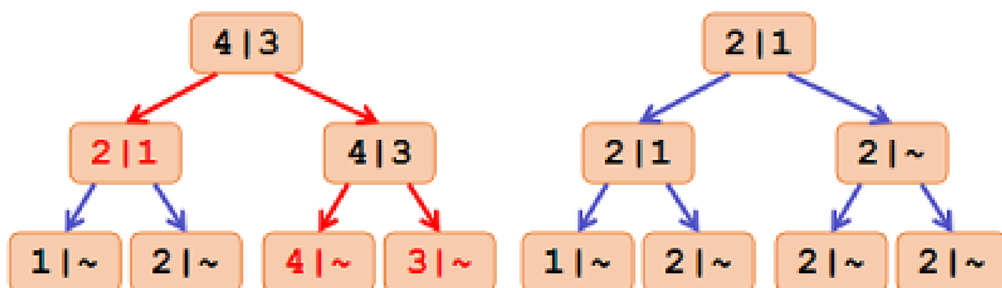
2.当$se < x < ma$时，显然这一次修改只会影响到所有最大值，所以我们把num加上$t \cdot (x - ma)$，把ma更新为x，接着打上标记然后退出。

3.当$se \ge x$时，我们无法直接更新这一个节点的信息，因此在这时，我们对当前节点的左儿子与右儿子进行递归搜索。

如下图所示，左图是一棵建立在区间$[1,4]$上的线段树，每一个节点记录的信息的左侧是区间最大值，右侧是严格次大值。现在我们要让区间$[1,4]$对2取min。

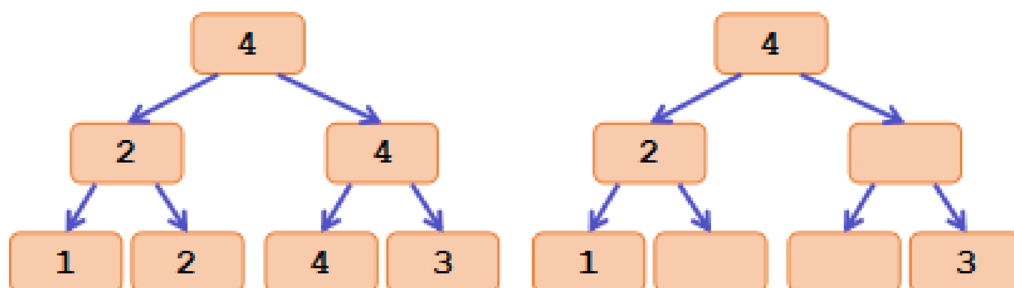
[6]Author：徐寅展，题目可以在这里找到：http://acm.hdu.edu.cn/showproblem.php?pid=5306

那么左图中的红色边表示在搜索时经过的边，红色字体的节点表示搜索终止的节点，右图为更新后的线段树。



如果进行实现的话，我们可以发现这一个算法的运行效率非常高。实际上我们可以证明它的时间复杂度是$O(m \log n)$的。

首先我们把最大值看成标记，对线段树每一个节点，记录一个标记值，它的值等于它所控制的区间中的最大值。接着如果一个点的标记值和它父亲的标记值相同，那么我们就把它这个位置的标记删去，大致转化如下图所示（左图记录的是线段树中的最大值，右图是转化后每一个位置的标记）：



显然在转化之后，线段树中最多存在n个标记。这些标记满足每一个标记的值都大于它子树中的所有标记的值。每一个位置实际的值等于从它对应的线段树的叶子节点出发，向上走遇到的第一个标记的值。

观察上述算法，可以发现我们维护的区间次大值，相当于子树中标记的最大值。而暴力DFS的作用，相当于在打上了新的标记后，为了满足标记值随深

度递减的性质，在子树中回收掉值大于它的标记，即标记回收。现在我们要证明的是标记回收的复杂度。

首先我们定义标记类的概念：

1.同一次区间取min标记产生的标记属于同一类。

2.同一个标记下传产生的新标记属于同一类。

3.不满足前两个条件的任意两个标记属于不同类。

接着定义每一个标记类的权值等于这一类标记加上线段树的根节点在线段树上形成的虚树大小（即所有子树中存在这一类标记的点的个数），定义势函数$\Phi(x)$为线段树中的所有标记类的权值总和。

考虑一次区间取max操作，我们添加了一个新的标记类，它的权值等于我们打标记时经过的点数，这显然单次是$O(\log n)$。

考虑一次标记下传，显然我们只让这一类标记的权值增加了$O(1)$。

考虑DFS的过程，我们一旦开始进行暴力DFS，那么说明这个点的子树中一定存在需要回收的标记，而在回收之后这个点的子树中一定不存在这一类标记，所以我们经过每一个节点，一定至少存在一类标记的权值减少了1，那么必定伴随着势函数减少了1。

因为标记下传只发生于我们在线段树上定位区间的时候，所以标记下传的总次数是$O(m\log n)$的，而打标记时对势函数产生的总贡献也是$O(m\log n)$的，所以势函数总的变化量只有$O(m\log n)$。

到此，我们就证明了这个算法的复杂度是$O(m\log n)$的。

例题2. *Picks loves segment tree*[7]

给出一个长度为n的数列A。接下来进行了m次操作，操作有三种类型：

1. 对于所有的$i\in[l,r]$，将A_i变成$\min(A_i,x)$。

2. 对于所有的$i\in[l,r]$，将A_i加上x（x可能为负数）。

3. 对于所有的$i\in[l,r]$，询问A_i的和。

数据范围$n,m\le 3\times 10^5$。

因为加上了区间加减，区间最大值、严格次大值和最大值个数还是可以维护的，所以我们考虑沿用刚才的做法。

接着我们来证明复杂度。因为有了区间加减这一个操作，例题一的证明就难以沿用了，但是大致的思想可以借鉴。

[7]题目来源：原创

我们可以修改一下标记类的定义：

1.同一次区间取min标记产生的标记属于同一类。

2.同一个标记下传产生的新标记都属于原来的一类。

3.对于一次区间加减操作，对于当前线段树中的任意一类，如果它被修改区间完全包含或者和修改区间完全相离，那么这一类就不变；否则我们就让这一类分裂成两类：一类是被修改的部分，一类是剩下的部分。

重新定义每一类标记的权值：等于这一类标记在线段树上所有节点形成的虚树大小（在这儿不包含根节点）。定义辅助函数$\Phi(x)$等于所有标记类的权值和。

这样一次区间加减操作对辅助函数函数的影响只有单次$O(\log n)$的标记下传（分裂标记的时候势能函数只会减少）；一次标记下传只会让辅助函数增加$O(1)$；一次打标记只会让辅助函数增加$O(\log n)$。

考虑DFS的过程，我们先通过线段树定位了若干个节点，然后从这些节点开始DFS。我们对每一个开始节点考虑每一类会被回收的标记：如果这一类标记只有一部分在这个子树中，那么我们每经过一个节点都会让这一种标记的权值减一；而如果这一类标记全部都在子树中，我们回收的过程相当于先花了若干的代价定位了这一类标记的LCA，然后再在标记的虚树上进行回收。

所以我们可以把复杂度分成两部分，第一部分是减少辅助函数时花费的时间复杂度，这是$O(m\log n)$的；第二部分的复杂度是定位某一类标记的LCA时产生的复杂度，每一类标记都可能对这一部分的复杂度产生$O(\log n)$的贡献，因为标记的总数只有$O(m\log n)$，所以标记的类数也是$O(m\log n)$的，因此这一部分的复杂度的一个上界是$O(m\log^2 n)$的。

到此我们就证明了这一个算法解决这个问题时的复杂度是$O(m\log^2 n)$的（当然，如果使用标记深度和作为势函数可以更方便的得到这个上界，这儿主要是沿用例题一的证明）。

实际上这个上界是比较松的，对目前我构造的数据来说，这个算法的运行效率都更接近与$O(m\log n)$。因此我猜想在这题的证明中标记的总类数是线性的，即这一个算法的时间复杂度是$O(m\log n)$的，然而到目前为止还没有好的方法能够证明，也无法构造出卡到$O(m\log^2 n)$的数据。所以在接下来的讨论中，我们就使用$O(m\log^2 n)$来表示这个算法的复杂度。

3.1 小结

通过上述两道例题，我们得到了一种在区间最值操作中维护区间和的方法（区间取max操作与区间取min操作类似）。

观察例题二的复杂度证明，我们发现在证明的时候并没有使用到区间加减操作的具体性质。因此我们将区间加减操作给替换成其他的修改操作，这一证明还是成立的。

例题3. *AcrossTheSky loves segment tree*[8]

给出一个长度为n的数列A。接下来进行了m次操作，操作有三种类型：

1. 对于所有的$i \in [l,r]$，将A_i变成$\min(A_i,x)$。

2. 对于所有的$i \in [l,r]$，将A_i变成$\max(A_i,x)$。

3. 对于所有的$i \in [l,r]$，询问A_i的和。

数据范围$n,m \leq 3\times 10^5$。

在我们单独解决区间取min操作与区间取max操作的时候，我们需要维护区间最小值、区间严格次小值、区间最小值个数、区间最大值、区间严格次大值以及区间最大值个数，现在这两个操作同时出现，但是这些信息还是可以维护的。

分析复杂度的时候可以把两种标记分开来考虑，显然每种标记对另一种标记的影响只有标记下传（在极端情况的时候还有可能直接回收掉某一棵子树里的部分标记，但是因为这个回收的时间复杂度是$O(1)$的，所以不会产生影响），因此例题二中的证明依然还是成立的，于是这个算法的时间复杂度应当是$O(m\log^2 n)$。

3.2 将区间最值操作转化为区间加减操作

观察我们解决例题二时使用的算法，我们发现在打上区间取min标记时，实际上这一个区间中只有最大值会受到这一个标记的影响，而其他数都是不会变的。同时在最大值被修改之后，原来是区间最大值的所有数依然还是区间最大值，原来不是区间最大值的所有数依然不是区间最大值。

所以实际上，通过上述算法，我们把区间最值操作，转化成了一种只针对区间最大（小）值进行加减修改的操作。

[8]题目来源：原创

因此我们可以把线段树中的每一个节点所维护的区间拆成两类，一类是这个区间中的最大值，一类是这个区间中的其他数。具体操作可以用例题二举例，在例题二中，我们需要处理的操作有：对一个线段树节点内的所有最大值加上或者减去一个数，对一个区间内的所有数加上或者减去一个数。因此我们只需要对线段树中的每一个节点中的最大值与非最大值分别维护标记与信息即可。

在标记下传的时候，需要根据两个儿子中的最大值情况进行讨论，最大值的标记只能下传到最大值较大的那个孩子中（最大值相同的时候同时下传）。

到此，我们就得到了一种在支付一个常数较小的$\log n$的复杂度的情况下，将区间最值操作转化为区间加减操作的方法。接下来我们通过三道不同方面的例题来进一步了解这个方法：

例题4. *Mzl loves segment tree*[9]

给出一个长度为n的数列A，定义一个数组B，最开始数组B的所有数都是0。接下来进行了m次操作，操作有四种类型：

1. 对于所有的$i \in [l, r]$，将A_i变成$\min(A_i, x)$。

2. 对于所有的$i \in [l, r]$，将A_i变成$\max(A_i, x)$。

3. 对于所有的$i \in [l, r]$，将A_i加上x。

4. 对于所有的$i \in [l, r]$，询问B_i的和。

在每次操作之后，对于每一个位置i，如果A_i的值发生了变化，那么就给B_i加上一，否则B_i不变。

数据范围$n, m \le 3 \times 10^5$。

如果只有区间加减操作，那么只要$x \neq 0$，那么每次修改的区间内的所有数都会发生变化，只要给B_i区间加上一就好了。

现在有了区间取min操作和区间取max操作，我们可以把线段树中的每一个节点中的数分成三个部分：最大值、最小值、既不是最大值也不是最小值的数，接着对每一个类数分别维护，这样就能消除区间取min操作与区间取max操作了。

值得注意的是当两个最值操作同时存在的时候，最大值与最小值所控制的数集可能会发生重叠，在这种情况下需要特殊处理避免在答案中重复统计某一个B_i的值。

[9] Author：黄志翱，这儿略有改编。

例题5. *ChiTuShaoNian loves segment tree*[10]

给出两个长度为n的数列A和B。接下来进行了m次操作，操作有五种类型：

1. 对于所有的$i \in [l,r]$，将A_i变成$\min(A_i,x)$。

2. 对于所有的$i \in [l,r]$，将B_i变成$\min(A_i,x)$。

3. 对于所有的$i \in [l,r]$，将A_i加上x。

4. 对于所有的$i \in [l,r]$，将B_i加上x。

5. 对于所有的$i \in [l,r]$，询问A_i+B_i的最大值。

数据范围：$n,m \le 3\times 10^5$

在这个问题中有两个操作的对象：数组A与数组B。在只有单一数组的时候，我们需要在每一个节点中把位置分成两类：区间最大值与非区间最大值。

稍微拓展一下一下，便可以得到这题的做法了。我们在线段树上，把每一个节点中的数给分成四类：同时在A,B中是区间最大值的位置，在A中是区间最大值而在B中不是的位置，在B中是区间最大值而在A中不是的位置，同时不是A,B中的最大值的位置。然后对这四类数分别记录答案与标记即可。

至于复杂度，我们发现两个数组在操作中是完全独立的，两套标记之间并不会产生任何影响，因此时间复杂度还是$O(m\log^2 n)$。

不难发现这个做法拓展到多个数组也是可行的：当我们对K个数组同时操作的时候，我们需要把每一个节点中的数分成2^K类，因此时间复杂度是$O(2^K\cdot m\log^2 n)$，空间复杂度是$O(n2^K)$的。

例题6. *Dzy loves segment tree*[11]

给出一个长度为n的数列A。接下来进行了m次操作，操作有三种类型：

1. 对于所有的$i \in [l,r]$，将A_i变成$\min(A_i,x)$。

2. 对于所有的$i \in [l,r]$，将A_i加上x。

3. 询问$\gcd(A_l,A_{l+1},...,A_{r-1},A_r)$。

数据范围：$n,m \le 10^5$

考虑只有区间加减操作的情况，这是一个经典问题，因为$\gcd(a,b,c)=\gcd(a,b-a,c-b)$，所以我们可以给数组$A$进行差分（即用$A_i-A_{i-1}$来替代$A_i$）。

---

[10]题目来源：原创

[11]题目来源：原创

在差分之后问题就转化成了两部分：第一部分是区间加减询问单点值，第二部分是单点修改询问区间gcd。这两部分都可以用线段树简单的维护。

现在我们来考虑有了区间取min操作后的情况，因为我们对区间最值操作的处理方法是把数拆成两类，但是这样的话数的顺序就会被打乱了，不能直接按照之前的下标顺序进行差分。于是我们考虑用另一种差分方式：把区间最大值单独拿出来维护，将非最大值的所有数进行差分。

具体来说，我们对线段树中的每一个节点额外维护以下的东西：最大值ma、严格次大值se，非最大值序列的开头s、结尾t、差分之后这个序列的最大公约数num。

现在考虑从两个儿子l与r中更新信息，这儿我们只讨论一种情况：$ma_l > ma_r$。在这时显然有$ma = ma_l, se = \max(se_l, ma_r)$。考虑更新差分序列的信息，注意到差分顺序可以是任意的，我们按照左子树序列、右子树序列、右子树最大值的顺序把非最大值序列给拼接起来，即让$s = s_l, t = ma_r, num = \gcd(num_l, num_r, t_l - s_r, t_r - ma_r)$。其他两种情况与这种类似，就不再赘述了。

考虑打上一个区间加减标记，因为我们将非最大值组成的序列差分了，所以num是不会被影响的，只要修改一下其他信息就行了，这是非常简单的。

如果算上gcd的复杂度的话，我们就得到了一个$O(m\log^3 n)$的做法。

3.3 总结

在这一节中，我们得到了一个处理区间最值操作的通用方法：通过维护最大（小）值与严格次大（小）值，在支付一个常数较小的$\log n$的时间复杂度内，将区间最值操作转化为区间加减操作。

但是这样的做法也有一定的局限性。从例题六中可以看出，当涉及到的询问与下标顺序有关的时候，这个做法就难以适用了，因为在这个做法中，我们把一个节点拆开的时候是没有考虑下标的顺序的。例如区间加减，区间取max，询问某一个区间的所有子区间中，权值和的最大值，这一个问题用上述的方法就难以解决。

4 历史最值问题

4.1 简介

在数据结构问题中，我们通常需要对一个给定的数组A进行若干次操作，然后进行一些询问。

目前大多数数据结构题都是对当前版本进行询问，也有一少部分数据结构题需要对历史版本进行询问，这一部分题目中的绝大部分考察的都是可持久化数据结构方面的知识，除此之外，有一类特殊的有关历史版本的问题我们把它称作历史最值问题。

历史最值问题中的询问大致可以分为以下三类：

4.1.1 历史最大值

定义一个辅助数组B，最开始B数组与A数组完全相同。在每一次操作后，对每一个$i \in [1, n]$，我们都进行一次更新，让$B_i = \max(B_i, A_i)$。这时，我们将B_i称作i这个位置的历史最大值。

4.1.2 历史最小值

定义一个辅助数组B，最开始B数组与A数组完全相同。在每一次操作后，对每一个$i \in [1, n]$，我们都进行一次更新，让$B_i = \min(B_i, A_i)$。这时，我们将B_i称作i这个位置的历史最小值。

4.1.3 历史版本和

定义一个辅助数组B，最开始B数组中的所有数都是0。在每一次操作后，对每一个$i \in [1, n]$，我们都进行一次更新，让B_i加上A_i。这时，我们将B_i称作i这个位置的历史版本和。

接下来，我们将历史最值问题按照难度从低到高分成四类进行讨论。

4.2 可以用懒标记处理的问题

例题7. *CPU*监控[12]

给出一个长度为n的数列A，同时定义一个辅助数组B，B开始与A完全相同。接下来进行了m次操作，操作有四种类型：

1. 对于所有的$i \in [l,r]$，将A_i变成x。
2. 对于所有的$i \in [l,r]$，将A_i加上x。
3. 对于所有的$i \in [l,r]$，询问A_i的最大值。
4. 对于所有的$i \in [l,r]$，询问B_i的最大值。

在每一次操作后，我们都进行一次更新，让$B_i = \max(B_i, A_i)$。

数据范围：$n, m \le 10^5$

刚接触这一类问题时，这个例题的难度可能较高，所以我们先忽略第一种操作。

考虑使用传统的懒标记来解决，首先如果只是询问区间最大值，只需要使用区间加减这一个懒标记（用Add表示）就能解决。

现在考虑询问区间历史最大值的最大值。我们定义一种新的懒标记：历史最大的加减标记（用Pre表示）。这个标记的定义是：从上一次把这个节点的标记下传的时刻到当前时刻这一时间段中，这个节点中的Add标记值到达过的最大值。

现在考虑把第i个节点的标记下传到它的儿子l，不难发现标记是可以合并的：$\text{Pre}_l = \max(\text{Pre}_l, \text{Add}_l + \text{Pre}_i), \text{Add}_l = \text{Add}_l + \text{Add}_i$。至于区间历史最大值信息的更新也与标记的合并类似，只需要将当前的区间最大值加上Pre_i然后与原来的历史最大值进行比较即可。

现在回到原题，我们观察在修改操作过程中，被影响到的节点的变化：如果一个节点没有发生标记下传，那么最开始它一直被区间加减操作所影响，这时我们可以用上面描述的Pre标记来记录，直到某一时刻，这个节点被区间覆盖标记影响了，那么这时这个节点中的所有数都变得完全相同，再之后的所有区间加减修改，对这个节点来说，与区间覆盖操作并没有不同。

因此每一个节点受到的标记可以分成两个部分：第一个部分是区间加减，第二个部分是区间覆盖。因此我们可以用(x,y)来表示历史最值标记，它的定义

[12]题目来源：`http://www.tyvj.cn/p/1518`

是当前区间在第一阶段时最大的加减标记是x，在第二个阶段时最大的覆盖标记是y。显然这个标记是可以进行合并与更新的。

到此我们就使用最传统的懒标记方法解决了这个问题，时间复杂度$O(m \log n)$。

例题8. V[13]

给出一个长度为n的数列A，同时定义一个辅助数组B，B开始与A完全相同。接下来进行了m次操作，操作有五种类型：

1. 对于所有的$i \in [l, r]$，将A_i变成$\max(A_i - x, 0)$。
2. 对于所有的$i \in [l, r]$，将A_i加上x。
3. 对于所有的$i \in [l, r]$，将变成x。
4. 给出一个i，询问A_i的值。
5. 给出一个i，询问B_i的值。

在每一次操作后，我们都进行一次更新，让$B_i = \max(B_i, A_i)$。

数据范围：$n, m \le 5 \times 10^5$

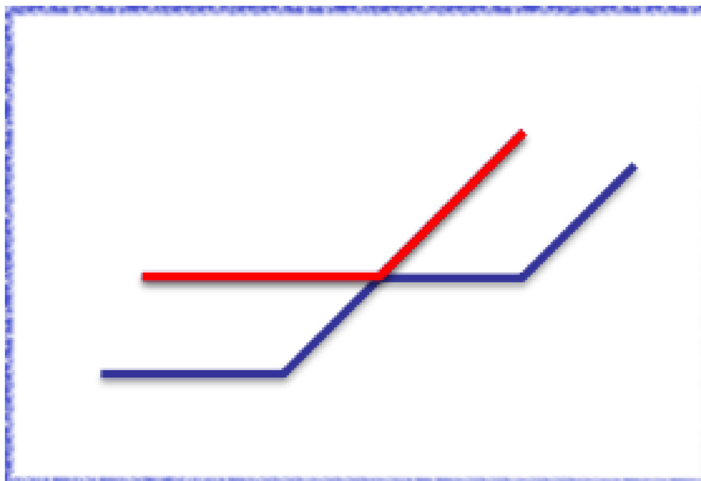
首先我们定义一种标记(a, b)，表示给这个区间中的所有数先加上a，然后再对b取max。不难发现题目中涉及的三种操作都可以用这一个标记来表示，分别为$(-x, 0), (x, -\text{inf}), (-\text{inf}, x)$，其中inf是一个事先设定的足够大数。

考虑合并两个标记(a, b)与(c, d)，那么显然得到的就是$(a + c, \max(b + c, d))$。因此对于询问4，我们只需要把询问的叶子节点到根路径上的所有标记都下传到叶子节点，就能直接求得答案了。

现在考虑历史最大标记。我们可以把标记给看成一个函数，每一个数字对应的函数值表示这个数在作用了这个标记后的答案，在这个问题中，标记(a, b)就相当于是一个分段函数，第一段是一条与x轴平行的直线，第二段是一条斜率为1的直线。

在更新历史最大标记的时候，我们相当于把两个标记的图象同时放到一个坐标系中，然后取出上方的那一段。下图展示的是标记合并时可能出现的一种情况，蓝色的部分是原来的两个标记，红色部分是更新后得到的标记。

[13]Author：彭雨翔，题目可以在这里找到：http://uoj.ac/problem/164



因为在更新标记之后形式和原来相同，所以这个历史最值标记是可以维护的。因此对于询问5，我们也只需要把所有相关的标记给下传下去，就能得到答案了。

时间复杂度$O(m\log n)$。

例题9. *Rikka with Sequences*[14]

给出一个长度为n的数列A，同时定义一个$n\times n$的辅助数组B，最开始$B_{i,j}=\sum_{k=i}^{j}A_k$。接下来进行了$m$次操作，操作有两种类型：

1. 给出两个整数l和x，把A_l的值变成x。

2. 给出两个整数l和r，保证$l<r$，表示询问$B_{l,r}$的值。

在每次操作之后，我们都会进行一次更新：$B_{i,j}=\min(B_{i,j},\sum_{k=i}^{j}A_k)$。

数据范围：$n,m\le 10^5$

因为这题不方便使用在线的方式处理，所以我们可以来考虑离线的算法。

首先我们把所有询问都读进来，并把每一个询问(l,r)都看成二维平面上的一个点。考虑一次对位置i的修改，它会对所有横坐标小于等于i，纵坐标大于等于i的询问产生影响。

所以现在问题就变成了，平面上有一些点，接下来有一些操作，每一次会给一个矩形区域内的所有点加上一个数，或者询问一个点权值的历史最小值。

于是这题相当于是二维情况下的历史最值问题，一维的情况我们可以轻松的用线段树来解决，因此一个非常自然的想法就是使用二维线段树。但是由于

[14]题目来源：原创

历史最小值的标记是时间依赖的，并不能拆成若干个部分，所以二维线段树并不适用。

为了解决这个问题，我们需要一个满足以下条件的数据结构：

1.标记的作用必须是以时间为顺序的。

2.影响到同一个点的标记必须影响在数据结构的同一个点上。

在二维情况下满足以上条件的一个数据结构就是kdtree了。所以我们可以先对原来的所有点建出kdtree，然后打标记的时候和线段树类似：对kdtree 每一个点都维护一个矩形区域，然后如果和标记区域相离就退出，被包含就直接打标记，否则就递归。在询问的时候和线段树一样，把标记下传下去即可。

时间复杂度$O(m\sqrt{n})$

4.3 无法用懒标记处理的单点问题

这儿的单点问题除了询问单点的历史最值外，也包括区间询问历史最小值的最小值与区间询问历史最大值的最大值，因为这两个区间询问的处理方法与单点询问完全相同，所以把它们归入单点问题之中。

例题10. *元旦老人与数列*[15]

给出一个长度为n的数列A，同时定义一个辅助数组B，B开始与A完全相同。接下来进行了m次操作，操作有四种类型：

1. 对于所有的$i \in [l,r]$，将A_i变成$\max(A_i,x)$。

2. 对于所有的$i \in [l,r]$，将A_i加上x。

3. 对于所有的$i \in [l,r]$，询问A_i的最小值。

4. 对于所有的$i \in [l,r]$，询问B_i的最小值。

在每一次操作后，我们都进行一次更新，让$B_i = \min(B_i, A_i)$。

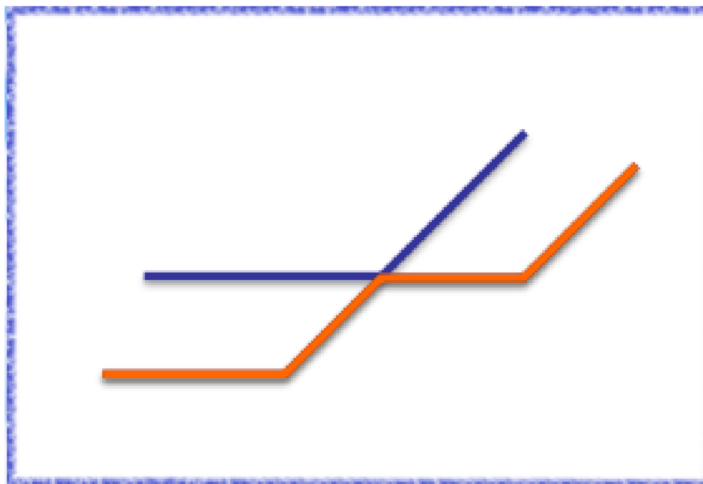
数据范围：$n,m \le 3\times 10^5$。

这儿涉及到了区间最值操作，区间最值操作在第三节中我们已经进行了深入的研究。

因为在这个问题中，区间最值操作的方向与历史最值询问的方向是相反的（一个是取max，一个是历史最小值），所以标记无法合并（如下图所示，两个

[15]题目来源：原创，题目可以在这里找到：`http://uoj.ac/problem/169`

蓝色标记合并出来的部分是下方的橙色曲线，它的段数会随着更新次数线性增长，这是不能接受的），因此我们无法沿用例题8的懒标记做法。



在第三节中我们已经得到了一种把区间最值操作转化为区间加减操作的方法，在这儿我们可以沿用。我们把线段树中每一个节点所控制的位置分成最小值与非最小值两部分，这样要处理的就只有区间加减操作了：这时的历史最值标记是可以轻易维护的。

到此我们就解决了这个问题，时间复杂度$O(m\log^2 n)$。

4.4 无区间最值操作的区间问题

这一标题所指的问题主要是以下三个：区间加减，询问区间历史最小值的和；区间加减，询问区间历史最大值的和；区间加减，询问区间历史版本和的和。

这三个问题都不能使用懒标记来进行维护，而且也不像上一节的区间最值操作一样被我们所熟知。接下来本文将分别对这三个问题介绍转化方法。

4.4.1 历史最小值

设A_i为当前数组，B_i为历史最小值。我们定义一个辅助数组C_i，每一时刻都让$C_i = A_i - B_i$。

那么我们对A_i加上数x时，如果B_i没有发生变化，那么相当于给C_i加上了x，否则C_i变成了0。具体来说，区间加减操作相当于将C_i变成了$\max(C_i + x, 0)$——这是一个我们非常熟悉的操作。

考虑区间和这一询问，只要能够求出A_i和C_i的区间和，直接相减便能得到答案。前者是非常基础的线段树问题，后者我们早已在第三节中研究过了。到此我们就得到了一种$O(m \log^2 n)$的算法。

4.4.2 历史最大值

处理方法与历史最小值类似。设A_i为当前数组，B_i为历史最小值。我们同样定义一个辅助数组C_i，每一时刻都让$C_i = A_i - B_i$。这时区间加减操作相当于将C_i变成$\min(C_i + x, 0)$，直接求和即可。

时间复杂度$O(m \log^2 n)$。

4.4.3 历史版本和

设A_i为当前数组，B_i为历史版本和，同时我们让t为当前结束的操作数（不包括当期操作）。同样，我们考虑定义个辅助数组C_i，每一时刻都让$C_i = B_i - t \cdot A_i$。

不难发现，对于区间加减没有影响到的位置，B_i都不会发生变化。当我们给A_i加上x的时候，我们相当于给C_i减去了$x \cdot t$。于是我们就把历史版本和转化成了简单的区间加减询问区间和的问题，直接懒标记就好了。

时间复杂度$O(m \log n)$。

4.5 有区间最值操作的区间问题

经过上面这些讨论后，这最后一类的算法已经有了雏形：只需要先通过第三节中的方法把区间最值操作转化成区间加减操作，再使用上一小节中的方法来进行求解就能够得到答案了。

因为这一类问题的处理方法比较类似，我们这儿就只选取其中一种作为例题进行分析。

例题11. 赛格蒙特彼茨[16]

[16]题目来源：原创

给出一个长度为n的数列A，同时定义一个辅助数组B，B开始与A完全相同。接下来进行了m次操作，操作有三种类型：

1. 对于所有的$i \in [l, r]$，将A_i变成$\max(A_i, x)$。

2. 对于所有的$i \in [l, r]$，将A_i加上x。

3. 对于所有的$i \in [l, r]$，询问B_i的和。

在每一次操作后，我们都进行一次更新，让$B_i = \min(B_i, A_i)$。

数据范围：$n, m \le 10^5$。

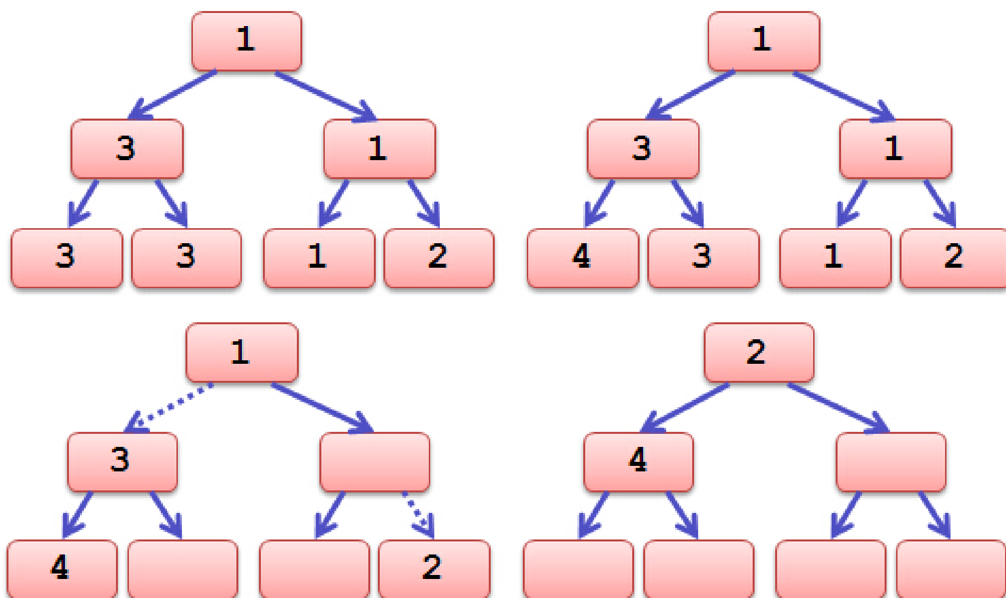
首先，我们使用上一小节中的方法，把询问转化成维护A数组和C数组的区间和。接着我们用第三节中的方法，对数组A维护区间最小值和次小值，从而把区间取max 操作转化成对区间最小值的加减操作，这样就能维护数组A的区间和了。

然后，我们来考虑怎么对C维护区间和，影响到C的操作有：对所有$i \in [l, r]$，把C_i变成$\max(C_i + x, 0)$（接下来称为第一类操作）；对所有$i \in [l, r]$，如果A_i是这个区间中A的最小值，那么把C_i变成$\max(C_i + x, 0)$（接下来称为第二类操作）。

所以还是使用第三节的方法，我们对线段树的每一个节点维护所有A的值是最小值的位置中C_i的最小值、次小值、最小值个数以及所有A的值不是最小值的位置中C_i的最小值、次小值、最小值个数，然后在对C进行修改的时候分别进行暴力DFS就行了。

最后我们来证明这个算法的时间复杂度。我们把A数组的部分称为第一层，C数组的称为第二层。首先第一层的复杂度和第三节的时候一样，我们能够证明出它是$O(m \log^2 n)$的，所以我们接下来只考虑第二部分的复杂度。

仿照第三节中的方法，首先我们先把第二层的最小值给转化成标记。可以对A的值等于最小值的所有位置以及A的值不等于最小值的所有位置分别转化成标记（接下来我们把这两个分别称为第一棵树和第二棵树），转化之后的树如下图所示（上方是原树，两侧分别为A和C的最小值，下方左图是第一棵树，右图是第二棵树）：



可以发现任何时刻，任何一个出现在第二棵树中的标记一定在第一棵树中出现过，所以两棵树上出现过的标记总数的数量级和第一棵树是相同的，所以我们只需要对第一棵树进行分析就行了。

因为第一棵树的定义是依赖数组A的，所以如果在原树中如果某一个节点的最小值和它的父节点不同，那么在第一棵树中这个位置到它父亲的转移是不存在的，我们需要删掉这条边（在图中用虚线表示）。因此在转化后第一棵树被剖成了若干个联通块，每一个联通块都至少存在一个叶子节点且满足标记值随着深度的增加而递增的性质。

因为直接对第一棵树进行标记回收以及区间加减的时候和A数组并没有什么区别，所以我们只考虑A数组中进行暴力DFS的时候产生的影响。在暴力DFS的时候，我们不仅对第一棵树进行了若干次子树加减操作，而且还合并了若干个联通块。不过如果我们在DFS的同时对标记进行下传，那么在合并两个联通块的时候，处在下方的联通块的根节点和上方联通块的根节点之间的路径上应该没有任何标记，所以这时直接把两个联通块合并起来，是不影响标记随着深度递增的限制的。

在第三节中，我们证明了暴力DFS的时候经过的节点的数的一个上界是$O(m\log^2 n)$，因此在第一棵树中标记下传的次数的上界是$O(m\log^2 n)$的，因

此标记总数是$O(m\log^2 n)$的。沿用算法三中的证明，我们可以得到这个算法的一个复杂度上界$O(m\log^3 n)$。

不过不难发现这个上界是非常松的，换句话说常数非常小，实际上在目前构造出来的数据中，暴力DFS的次数都远远没有达到$O(m\log^3 n)$的级别，因此我猜想应该存在更低的上界的证明（实际上如果第三节中我的猜想得到了证明的话，沿用上述证明就可以得到一个$O(m\log^2 n)$的上界），但是因为目前我还没有得到更好的证明方法或者极限数据构造方法，所以目前只能用$O(m\log^3 n)$ 这样一个比较松的上界来衡量这个算法的运行效率。

到此，我们就得到了一个$O(m\log^3 n)$的算法来解决这题。实际上这题就是第三节中的算法的嵌套，不难发现，上述的证明对更多层的嵌套依然有用，即如果我们有K个数组进行嵌套，那么沿用上述算法可以得到一个$O(2^K \cdot m\log^{K+1} n)$复杂度的算法。

4.6 总结

在这一节中，我们针对三种历史最值问题进行了深入的讨论，并介绍了一种通过构造辅助数组把历史最值的区间和这一询问转化为简单的区间和询问的方法。

在目前国内的OI竞赛中，出现过的历史最值相关的问题非常有限，而且大部分都集中在难度较低的前两类中。因此有关这一类问题的挖掘应当还有很大的空间，我所解决的只是其中的一小部分，例如区间加减，询问区间和的历史最小值这一问题，到目前为止我也没有好的解决方案，因此，我希望这篇文章能起到抛砖引玉的作用。同时，如果有人对这一类问题有了新的收获，也欢迎来与我交流。

5 致谢

1. 感谢中国计算机学会提供学习和交流的平台。

2. 感谢学军中学中学的徐先友老师的关心和指导。

3. 感谢徐寅展学长给予的启发与指导。

4. 感谢彭雨翔学长、吕凯风学长、毛啸同学、罗哲正同学给予的帮助。

5. 感谢周子鑫同学、毛潇涵同学的验稿。

6. 感谢其他所有本文所参考过的资料的提供者。

7. 感谢各位百忙之中抽出宝贵的时间阅读。

参考文献

[1] 徐寅展，IOI2014国家集训队论文《线段树在一类分治问题上的应用》

[2] 线段树的英文维基百科

https://en.wikipedia.org/wiki/Segment_tree

[3] 多校联合训练 Gorgeous Sequence 题解

http://blog.sina.com.cn/s/blog_15139f1a10102vnzl.html

再探快速傅里叶变换

雅礼中学 毛啸

摘 要

本文主要介绍了快速傅里叶变换的实现和一些在算法竞赛中有用的技巧和优化。快速傅里叶变换作为一个早已经在算法竞赛中流行起来的算法，对大家来说并不陌生，因此本文题为“再探快速傅里叶变换”。“再探”，顾名思义，本文肯定会介绍许多已经普及了的方法和知识，但仍有相当一部分篇幅将会用在介绍一些在现在算法竞赛中还不是很普及，但是却非常简单而有效的方法。例如，在模10^9+7意义下做两个次数不超过10^5的实多项式的乘法，目前传统的方法是在三个可以用于数论变换的模数下求出结果，最后利用中国剩余定理来求得最终的答案，需要做9次模意义下的离散傅里叶变换，而利用本文3.3节中介绍的方法，只需要做4次复数意义下的离散傅里叶变换即可。

前言

近几年来，各种基于FFT的算法如雨后春笋般流行起来。在2014年的NOI冬令营营员交流上，出现了关于多项式除法的内容[1]，之后在Codeforces上出现了使用多项式除法的题目[2]，之后基于牛顿迭代的各种算法纷纷流行起来，例如在Universal Online Judge(UOJ) 上出现了使用牛顿迭代解微分方程的题目[3]，在2016年NOI冬令营第一课堂的《多项式导论》中，这些算法被再次普及，UOJ上最近还出现了多点求值的题目[4]。基于FFT的各种算法的出现，给解题带来了极大的便利。

[1]余行江、彭雨翔《多项式除法及其应用》

[2]http://codeforces.com/contest/438/problem/E

[3]http://uoj.ac/problem/50

[4]http://uoj.ac/problem/182

然而，所有这些算法都离不开一个东西——FFT，如果没有掌握FFT这个最基本的算法，所有这些基于FFT的算法肯定更加难以掌握。好比盖一栋大楼，如果我们把楼盖得很高，却因为基底不牢固而坍塌，这栋楼再怎么高也没有用。一棵大树，枝叶繁茂，根却因为种种原因坏死，这棵树也只会盖着一头枯枝烂叶。只有夯实了基础，才能走的更远。本人写此文正是基于这样一个思路。希望能与各位读者回到原点，重新研究一下FFT这个最基础而又十分神奇的算法。而什么叫掌握一个算法呢，本人认为，掌握一个算法不仅仅限于写过、甚至背过这个算法，而是既了解这个算法的基本原理，也知道这个算法的各种应用、各种优化和技巧。

1 基本介绍

快速傅里叶变换(Fast Fourier Transform(FFT))是现在比较流行的算法之一，该算法在1965年被普及，但是有一些方法在1805年就有出现[5]。该算法通过计算一个序列的离散傅里叶变换(Discrete Fourier Transform(DFT))来实现将信号从原始域(通常是时间或空间)到到频域的互相转化。该算法在当今算法竞赛中已经十分常见，主要应用于计算两个序列的卷积。

1.1 引入

我们来看一下这样一个问题：

给定两个有限长度的序列a_i，b_i，求序列c使得$c_i = \sum_{k=0}^{i} a_k b_{i-k}$，显然$c$的长度为$a, b$的长度之和减一。本文中的序列下标均从0开始。

1.2 离散傅里叶变换

我们找一个大于c的长度且为2的整数次幂的数n，至于为什么是2的整数次幂后面会有提到。**给定序列s，假设它的长度为$|s|$，那么对于$k \notin [0, |s|)$我们将s_k视为0。**

[5]Heideman, M. T.; Johnson, D. H.; Burrus, C. S. (1984). "Gauss and the history of the fast Fourier transform". IEEE ASSP Magazine 1 (4): 14–21

那么我们有：

$$c_r = \sum_{p,q} [(p+q) \bmod n = r] a_p b_q \tag{1.1}$$

设ω满足$\omega^n = 1$，即ω是n次单位根，**本文中提到的单位根的意思是说，n是最小的使得$\omega^n = 1$ 的数**。易知：

$$\frac{1}{n} \sum_{k=0}^{n-1} \omega^{vk} = [v \bmod n = 0] \tag{1.2}$$

故：

$$\begin{aligned} & [(p+q) \bmod n = r] \\ = \ & [(p+q-r) \bmod n = 0] \\ = \ & \frac{1}{n} \sum_{k=0}^{n-1} \omega^{(p+q-r)k} \\ = \ & \frac{1}{n} \sum_{k=0}^{n-1} \omega^{-rk} \omega^{pk} \omega^{qk} \end{aligned}$$

我们有：

$$\begin{aligned} c_r &= \sum_{p,q} [(p+q) \bmod n = r] a_p b_q \\ &= \sum_{p,q} \frac{1}{n} \sum_{k=0}^{n-1} \omega^{-rk} \omega^{pk} \omega^{qk} a_p b_q \\ &= \frac{1}{n} \sum_{k=0}^{n-1} \omega^{-rk} \sum_{p,q} \omega^{pk} a_p \omega^{qk} b_q \\ &= \frac{1}{n} \sum_{k=0}^{n-1} \omega^{-rk} \sum_{p} \omega^{pk} a_p \sum_{q} \omega^{qk} b_q \end{aligned}$$

我们发现了这样两种关系式：

$$a_m = \sum_{k=0}^{n-1} \omega^{mk} b_k \tag{1.3}$$

$$c_m = \frac{1}{n} \sum_{k=0}^{n-1} \omega^{-mk} d_k \tag{1.4}$$

现在，我们发现我们只要对于每一种关系式，知道右边快速的求得左边。进一步我们发现，我们设多项式$A(x) = \sum_{k=0}^{n-1} b_k x^k$，那么$a_m = A(\omega^m)$，相当于给定一个多项式的系数表示，对于所有$k$求出其在点$\omega^k$的值，即在系数表示和点值表示之间转化，这就是**离散傅里叶变换(DFT)**的过程。

我们后面提到多项式与系数序列的对应关系都是指类似这里$A(x)$与b_k的关系，我们说对$A(x)$进行DFT，实际上指的就是对b_k进行DFT，说对一个东西DFT得到了$A(x)$，指的就是得到了序列b_k。

1.3 快速傅里叶变换

前一节中已经提到了利用DFT的公式，但是按照公式，显然还是太慢了。这时候需要利用单位复根的性质来加速DFT，也就是**快速傅里叶变换(Fast Fourier Transform(FFT))**。

下面仅考虑(1.3)式，我们知道b要求a。

注意到n是2的整数次幂。我们用ω_n^k来表示n次单位复根，那么我们有：

$$\omega_{2n}^{2m} = \omega_n^m \tag{1.5}$$

$$\omega_{2n}^{m} = -\omega_{2n}^{m+n} \tag{1.6}$$

设$A_0(x)$是$A(x)$的偶次项的和，$A_1(x)$是奇次项的和。那么：

$$A(\omega_n^m) = A_0((\omega_n^m)^2) + \omega_n^m A_1((\omega_n^m)^2) \tag{1.7}$$

$$= A_0(\omega_{\frac{n}{2}}^m) + \omega_n^m A_1(\omega_{\frac{n}{2}}^m) \tag{1.8}$$

$$A(\omega_n^{m+\frac{n}{2}}) = A_0((\omega_n^m)^2) + \omega_n^{m+\frac{n}{2}} A_1((\omega_n^m)^2) \tag{1.9}$$

$$= A_0(\omega_{\frac{n}{2}}^m) - \omega_n^m A_1(\omega_{\frac{n}{2}}^m) \tag{1.10}$$

即我们只要对$A_0(x)$和$A_1(x)$求出DFT，就可以在线性时间内求出$A(x)$的DFT，(1.8)(1.10)式常被称为“蝴蝶操作”。

设给长度为n的序列做DFT的复杂度为$T(n)$，则有：

$$T(n) = 2T(\frac{n}{2}) + O(n) \tag{1.11}$$

根据主定理$T(n) = O(n \log n)$。

我们发现，如果要完成上述算法，必须要做到能够把一个多项式的奇数项和偶数项分开，虽然在递归中是可以实现的，但是常数过大，作为一个经常要使用的算法，如果我们的实现方式常数过大是很不好的。

我们观察程序执行的时候每个函数的系数，以长度为8为例：

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$

$(a_0, a_2, a_4, a_6)(a_1, a_3, a_5, a_7)$

$(a_0, a_4)(a_2, a_6)(a_1, a_5)(a_3, a_7)$

$(a_0)(a_4)(a_2)(a_6)(a_1)(a_5)(a_3)(a_7)$

观察$(0, 4, 2, 6, 1, 5, 3, 7)$的二进制表示为$(000, 100, 010, 110, 001, 101, 011, 111)$，我们发现就是把$(0, 1, 2, 3, 4, 5, 6, 7)$的二进制表示$(000, 001, 010, 011, 100, 101, 110, 111)$中的每一位的二进制位翻转，称为“位逆序置换”，那么如果能够较快的求出位逆序置换之后的序列，那么在求解的时候就可以每次处理的都是一段连续的区间，很容易实现非递归的FFT，并且速度比递归版本更快[6]。

此外，为了进一步优化常数，我们应该预处理单位根。

预处理单位根可以利用$\omega^{k+1} = \omega^k \times \omega$递推求，但是这样精度误差稍微有点大，虽然一般情况下可以接受，但是在第3节中介绍的FFT方法中要求精度达到10^{14}，这样预处理单位根会有严重的精度问题。

另外一种方法是设n次单位根为ω_n，那么$\omega_n^k = cos(\frac{2\pi k}{n}) + isin(\frac{2\pi k}{n})$，这里$i = \sqrt{-1}$，这样虽然速度比上述方法慢，但是精度误差更小，可以满足10^{14}的精度要求。

做卷积的过程无非就是将a, b数组均做1次DFT，然后乘起来再做1次逆DFT(Inverse Discrete Fourier Transform(IDFT))，即实现(1.4)式，方法类似不再赘述[7]。

能不能做到更优呢，当然能！通过本文第3节介绍的方法，我们可以在2次的DFT甚至更优的时间内实现实序列[8]的卷积。

1.4 数论变换

现在算法竞赛中常常要求对某一个数取模，如果取模的数是一个质数P，且

[6]这里本人推荐非常好的模板题：`http://uoj.ac/problem/34`，较好的代码可以从提交记录统计中看到。

[7]其实这种做2次变换再乘起来再逆变换的思想在反演中应用广泛，有兴趣的同学可以去看参考文献1。

[8]即每一项都是实数。

这个质数减一是一个比DFT的长度要大的2的整数次幂的倍数，那么我们可以求出$\mathbb{F}_{\mathbb{P}}$下的原根g，并在$\mathbb{F}_{\mathbb{P}}$下用$g^{\frac{P-1}{n}}$来代替单位复根ω，这就是**数论变换(Number Theorem Transform(NTT))**。

如果取模的数不是满足这种条件的质数，方法可以参见3.3节。

对于其他域上的多项式乘法，有$O(n\log n\log\log n)$的做法，限于篇幅不再赘述，有兴趣的同学可以参考2016年的NOI冬令营第一课堂的《多项式导论》的相关内容。

1.5 Bluestein's Algorithm

我们前面提到1.1式都是忽略掉了式子中的mod，如果考虑这个mod的话，c实际上是长度为2^k下的**循环卷积**。有时候题目中要求的操作就是做循环卷积，比如如果我们要做循环卷积意义下的快速幂，注意到如果序列长度是2的整数次幂，那么完全可以直接DFT完之后对每一项快速幂，再IDFT回来，然是如果题目要求做循环卷积且长度n不是2的整数次幂，怎么办呢？

一种方法是直接做一遍普通卷积得到答案c，最后再将c_{k+n}加到c_k上。

注意这种方法仅仅只能做1次循环卷积，如果我们要求做多次循环卷积，就必须一遍一遍的做，如果要求在循环卷积意义下做快速幂还要多一个$O(\log n)$的复杂度，我们当然希望能避免。

如果我们考虑的域较为特殊的存在n次单位复根的域，(如$\mathbb{R}$、$\mathbb{C}$、某些$\mathbb{F}_{\mathbb{P}}$等)，那么我们可以考虑能否直接做长度为n的序列的DFT。

这里我们仅介绍较为好理解的Bluestein's Algorithm，这个算法在目前算法竞赛中还不是很普及。

我们继续考虑(1.3)式：

$$\begin{aligned}a_m &= \sum_{k=0}^{n-1}\omega^{mk}b_k\\ &= \sum_{k=0}^{n-1}\omega^{-\frac{(m-k)^2+m^2+k^2}{2}}b_k\\ &= \omega^{-\frac{m^2}{2}}\sum_{k=0}^{n-1}\omega^{-\frac{(m-k)^2}{2}}\omega^{-\frac{k^2}{2}}b_k\end{aligned}$$

于是我们化成了卷积的形式。换句话说我们平时用DFT实现卷积，这里我们用卷积实现DFT!

具体怎么做呢，注意(1.12)式的上下界并不是卷积的条件，但是我们注意到b_k仅在k在0到$n-1$的时候才有可能非0，这是一种可以很好利用的性质。

令$B_i = \omega^{-\frac{(i-n)^2}{2}}$，我们有：

$$A_{m+n} = \omega^{-\frac{2}{m^2}a_m} \quad (1.12)$$

$$= \sum_{k=0}^{n-1} \omega^{-\frac{(m-k)^2}{2}} \omega^{-\frac{k^2}{2}} b_k \quad (1.13)$$

$$= \sum_{k=0}^{n-1} B_{m+n-k} \omega^{-\frac{k^2}{2}} b_k \quad (1.14)$$

$$= \sum_{k=0}^{m+n} B_{m+n-k} \omega^{-\frac{k^2}{2}} b_k \quad (1.15)$$

(1.14)式到(1.15)式就是利用了b_k仅在k在0到$n-1$的时候才有可能非0的性质。

这样，我们用一遍普通FFT，就可以做任意长度的DFT了，这就是Bluestein's Algorithm的过程。

Bluestein's Algorithm计算的是Chirp Z-Transform(CZT)，CZT是DFT的广义形式。简单来说这个算法不仅可以实现任意长度的DFT，还可以将(1.3)式中的ω换为任意其他的数。

2 基本应用

FFT最大的作用就在于做卷积，卷积不仅仅用于多项式乘法，还可以用于其他各内题目当中，接下来一节中将介绍几种常见的应用。

2.1 最基本的应用

例题1. 力[9]

给出n个数q_i，定义F_j为$F_j = \sum_{i<j} \frac{q_i q_j}{(i-j)^2} - \sum_{i>j} \frac{q_i q_j}{(i-j)^2}$，求所有的$E_i = \frac{F_i}{q_i}$。$n$不超过100000。

题目中要求的式子与普通卷积还有细微区别，但是我们有一个技巧就是设$A_{i+n} = E_i$使之完全符合卷积的形式。

[9]来源：ZJOI2014

根据A序列的定义我们有：

$$
\begin{aligned}
A_{j+n} &= E_j \\
&= \frac{F_j}{q_j} \\
&= \frac{\sum\limits_{i<j}\frac{q_i}{(i-j)^2}-\sum\limits_{i>j}\frac{q_i}{(i-j)^2}}{q_j} \\
&= \sum_{i<j}\frac{q_i}{(i-j)^2}-\sum_{i>j}\frac{q_i}{(i-j)^2} \\
&= \sum_{k=0}^{j+n} q_k P_{j+n-k}
\end{aligned}
$$

其中，P_i的定义如下：

1. 当$i > n$时$P_i = (i-n)^{-2}$。
2. 当$i < n$时$P_i = -(i-n)^{-2}$。
3. 当$i = n$时$P_i = 0$。

直接对q和p做卷积之后得到A，再根据$E_i = A_{i+n}$求得E即可。

例题2. *Fuzzy Search*[10]

给定母串和模式串，字符集大小为4，给定k，模式串在某个位置匹配当且仅当对于任意位置模式串的这个字符所对应的母串的位置的左右k个字符之内有一个与它相等的，求匹配次数。

串长不超过200000。

很容易求出母串的每个位置每种字符是否能匹配。

对于每种字符，我们将母串中能匹配这种字符的和模式串中所有这个位置是这种字符的地方设为1，其他地方设为0，若每个位置母串和模式串如果同时为1则贡献1，否则没有贡献，也就是两个值乘起来，那么匹配当且仅当四种字符的贡献和是串长，考虑怎么计算每种字符的贡献。

接下来将用的是常用的一种技巧，我们将模式串或者母串翻转。我们发现翻转之后，贡献的计算成了卷积的计算，直接用FFT计算卷积即可。

[10]来源：`http://codeforces.com/problemset/problem/528/D`

这题体现到了将匹配问题中的模式串或者母串翻转从而转换为卷积的常用技巧。

2.2 更高层次的应用

接下来将介绍FFT更高层次的应用。

例题3. *Kyoya and Train*[11]

n个点m条边的图，给定起点和终点，如果不能在T时间内到达终点则需要付出给定的代价，走每条边需要付出给定的代价，每条边经过时间的分布列给定，最小化所需要的代价的期望。保证有$n \le 50$，$m \le 100$，$T \le 20000$。

考虑暴力DP怎么做，令$\mathrm{DP}_{x,t}$表示到了第x个点，当前经过的时间是t，到终点需要的最小代价，$P_{e,k}$ 表示这条边经过时间为k的概率。那么转移方程就是枚举每一条边e，假设这条边到的点是y，那么用$\sum_k \mathrm{DP}_{y,t+k} P_{e,k}$ 更新答案即可。

如果$t > T$，那么t的具体值是不会有影响的，所以对于所有$t > T$我们都不妨都用$t = T + 1$表示。

考虑对于一条边$e : x \to y$，我们设$S_{e,t} = \sum_k \mathrm{DP}_{y,t+k} P_{e,k}$，那么对于每个$t$, $\mathrm{DP}_{x,t}$的值就是所有x连出的边的e的$S_{e,t}$的最大值。

现在考虑优化，考虑分治，如果我们要求出所有$l \le t \le r$的$\mathrm{DP}_{x,t}$与$S_{e,t}$，那么我们设$mid = \lceil \frac{l+r}{2} \rceil$，先求出所有$mid \le t \le r$的答案，然后我们注意到$S_{e,t} = \sum_k \mathrm{DP}_{y,t+k} P_{e,k}$很容易变为卷积的形式，我们可以用FFT求出$mid$到$r$这一段的DP值对前面所有$S$的贡献，然后递归求$l \le t < mid$的答案。如果$l = r$，那么直接用$S$更新DP即可。

根据主定理，时间复杂度是$O(mT \log^2 T)$。

这道题巧妙的利用了分治FFT来解决问题，实际上分治FFT是非常常见的解决问题的方法。

分治FFT在多点插值，多点求值等问题上也有应用，但其与本文研究FFT基本应用的出发点相悖，因此具体做法不再赘述。

例题4. *Transforming Sequence*[12]

[11]来源：`http://codeforces.com/problemset/problem/553/E`

[12]来源：`http://codeforces.com/problemset/problem/623/E`

求长度为n的满足前缀按位或单调递增的k位序列。要求每个位置为$[0,2^k-1]$之间的整数。保证$k\leq 30000$，答案要求模10^9+7后输出。

继续从暴力算法出发，考虑用$\mathrm{DP}_{i,j}$表示长度为i，且当前按位或已经有j位的合法的序列个数，那么对$\mathrm{DP}_{i+1,j+l}$有$\mathrm{DP}_{i,j}\times\binom{k-j}{l}$的贡献。

考虑优化，我们如果对所有x求出了$\mathrm{DP}_{i,x}$和$\mathrm{DP}_{j,x}$，考虑对所有x快速求出$\mathrm{DP}_{i+j,x}$。

对于任意l，$\mathrm{DP}_{i,x}$对$\mathrm{DP}_{i+j,x+y}$有$\mathrm{DP}_{i,x}\times 2^{jx}\times\mathrm{DP}_{j,y}\times\binom{k-x}{y}$ 的贡献，2^{jx} 是这样来的，考虑前i位的或值的x位1，后j位这些地方的值已经无所谓了所以可以任意填。这是一个卷积的形式，可以用FFT快速计算。

这样我们就用简单的快速幂在$O(k\log^2 k)$的时间内解决这个问题。

FFT的应用还远远不止这些，限于篇幅本人暂且只介绍这么多。

2.3 一些优化技巧

2.3.1 减少DFT次数

考虑计算$A(x)=B(x)C(x)+D(x)E(x)$。

最简单的方法就是用两个DFT计算出$B(x)C(x)$和$D(x)E(x)$再加起来，但是我们容易发现，在DFT之后给两个序列的每一项进行乘除的组合再IDFT回去，得到的就是这个多项式的对应乘除组合，前提是长度不超过限制，所以我们先把$B(x),C(x),D(x),E(x)$统统DFT得到序列b,c,d,e，再对每一个k令$a_k=b_kc_k+d_ke_k$，最后给a进行IDFT即可得到$A(x)$。

这个方法在复杂的多项式运算中十分有用，缺点是难以与第3节中介绍的合并DFT方法相容。

还有一个简单技巧，如果一个序列多次被用于DFT，那么显然可以预处理好它的DFT，然后每次直接使用预处理的值。

2.3.2 利用循环卷积

考虑对于两个长度为n的序列a,b，计算它们的卷积c的第$\lceil 0.5n\rceil$项到第$\lfloor 1.5n\rfloor$项，传统的做法是全部往后补零扩充为长度为$2n$的序列然后用FFT算卷积，但是因为FFT求的实际上是在1.5节中我们已经提到的循环卷积，我们如果只补零

扩充到长度为$\lceil 1.5n \rceil$以上的序列然后继续调用FFT求卷积，那么第$\lceil 0.5n \rceil$项到第$\lfloor 1.5n \rfloor$ 项的值不受影响。

上述两种优化在基于牛顿迭代的算法中经常出现，能起到比较大的常数优化作用。

2.3.3 分治FFT中的小范围暴力

在分治FFT算法中，由于FFT自带较大的常数，因此在序列长度较小的时候换用暴力能起到比较大的优化效果。其实这种优化不仅用在分治FFT 中，在所有分治算法中都能起到一定的作用。

上述优化在多点插值，多点求值等算法中有很大的作用。

2.3.4 快速幂乘法次数的优化

如果我们要计算一个多项式的n次幂，在条件允许的情况可以通过取多项式ln，再乘上n，再求多项式exp，复杂度减少了一个$O(\log n)$，多项式ln和多项式exp的方法与本文研究FFT基本应用的出发点相悖，因此不再赘述。

假如我们要快速幂的东西无法用上述方法优化的话，有一个现在还不是很普及的技巧：利用addition chain。

我们注意到快速幂最坏需要$2\log_2 n + C$次乘法，但是这并不是下界。我们可以做到更少的乘法次数。我们这里定义addition chain 为一条链，最开始是一个1，后一个数减前一个数的差是这条链上这个数前面的某一个数。我们称这条addition chain能得到这条链上的所有数。

例如$1 \to 2 \to 4 \to 6$，$6-4=2$在6 之前的数中出现过, $4-2=2$在4之前也出现过。那么这条链能得到$\{1,2,4,6\}$，那么根据这条链我们能设计一种方法用乘法从1次幂得到1、2、4、6次幂，比如我们要求6次幂，那么根据这条addition chain ，我们可以从1次幂出发，用1次幂乘1次幂得到2次幂，再乘2次幂得到4次幂，再乘2次幂得到6次幂。

求出能得到一个数的最优的addition chain是NPH的，但是我们有很好的近似算法，例如我们直接BFS，记录每个数对应的addition chain。首先从1出发，1对应的addition chain就是一个数1，每次我们对于当前数x，枚举它对应的addition chain中的数y，如果$x+y$还没有访问过那么将其入队，并且置它对

应的addition chain为x对应的addition chain后面接上$x+y$。这种方法最坏情况下比快速幂优秀得多，但是它有一个缺点是必须线性预处理。

如果我们要对很大的n(例如10^{10000})求出一个东西的n次幂，我们可以通过Method of Four Russians将乘法次数由$2\log_2 n+C$减少至$\log_2 n+O(\frac{\log n}{\log\log n})$。具体如下，设$2^k$ 为最接近$\frac{\log n}{\log\log n}$的二的整数次幂，预处理要求的东西的0到$2^k-1$次方，复杂度显然是$O(\frac{\log n}{\log\log n})$，然后每次倍增，每次倍增完$k$次之后再考虑当前模$2^k$的额外乘一个余数次幂，这样倍增中的乘法次数是$\log_2 n+C$，而额外乘的次数是$O(\frac{\log n}{k})=O(\frac{\log n}{\log\log n})$。

此外，如果询问次数很多的话，可以将k设大。

上述方法虽然只是常数优化，但是实现起来十分简单，效果也非常不错。

3 一个新的技巧

接下来我们将研究一种能显著减少DFT次数从而大幅度优化常数的技巧。

事实上这个技巧在现在算法竞赛中并没有完全普及，但是也有一部分人可能听说过这个方法，这里本人做一些介绍希望能进一步普及这个技巧。

这个技巧是在Codechef MGCH3D一题中使用到的优化方法。事实上Codeforces上俄罗斯的enot1.10(Vladimir Smykalov)在与本人talk中告诉本人了一个看似不同的优化方法。但经过一些推敲即可发现事实上该优化方法与Codechef MGCH3D一题中使用到的优化方法大同小异，故这里只描述Codechef MGCH3D的题解当中的描述，题解原文可见参考文献2。

讨论DFT的次数时我们有时会不区分DFT和IDFT，比如3次IDFT加3次DFT被称为6次DFT。

3.1 基本介绍

我们考虑对长度为n的实多项式$A(x)$, $B(x)$进行DFT，假设n已经调整为2的整数次幂。

我们定义：

$$P(x) = A(x)+iB(x) \tag{3.1}$$

$$Q(x) = A(x)-iB(x) \tag{3.2}$$

这里i指$\sqrt{-1}$，设$F_p[k], F_q[k]$分别表示对P和Q进行DFT 之后得到序列的第k项，即$F_p[k] = P(\omega^k), F_q[k] = Q(\omega^k)$，$\omega$是$n$次单位根。

接下来我们进行一系列推导：

由于排版问题我们用X代替$\frac{2\pi jk}{2L}$这个式子，每个X所对应的j, k的含义在上下文中可以看出。$\mathrm{conj}(x)$表示x的共轭复数。

$$\begin{aligned}
F_p[k] &= A(\omega_{2L}^k) + iB(\omega_{2L}^k) \\
&= \sum_{j=0}^{2L-1} A_j\omega_{2L}^{jk} + iB_j\omega_{2L}^{jk} \\
&= \sum_{j=0}^{2L-1} (A_j + iB_j)(\cos X + i\sin X)
\end{aligned}$$

$$\begin{aligned}
F_q[k] &= A(\omega_{2L}^k) - iB(\omega_{2L}^k) \\
&= \sum_{j=0}^{2L-1} A_j\omega_{2L}^{jk} - iB_j\omega_{2L}^{jk} \\
&= \sum_{j=0}^{2L-1} (A_j - iB_j)(\cos X + i\sin X) \\
&= \sum_{j=0}^{2L-1} (A_j\cos X + B_j\sin X) + i(A_j\sin X - B_j\cos X) \\
&= \mathrm{conj}\left(\sum_{j=0}^{2L-1} (A_j\cos X + B_j\sin X) - i(A_j\sin X - B_j\cos X)\right) \\
&= \mathrm{conj}\left(\sum_{j=0}^{2L-1} (A_j\cos(-X) - B_j\sin(-X)) + i(A_j\sin(-X) + B_j\cos(-X))\right) \\
&= \mathrm{conj}\left(\sum_{j=0}^{2L-1} (A_j + iB_j)(\cos(-X) + i\sin(-X))\right) \\
&= \mathrm{conj}\left(\sum_{j=0}^{2L-1} (A_j + iB_j)\omega_{2L}^{-jk}\right) \\
&= \mathrm{conj}\left(\sum_{j=0}^{2L-1} (A_j + iB_j)\omega_{2L}^{(2L-k)j}\right) \\
&= \mathrm{conj}(F_p[2L-k])
\end{aligned}$$

于是我们仅用1次DFT就可以算出F_p和F_q。

令$\text{DFT}(P[k])$表示对$P(x)$进行DFT之后得到的序列的第k项，我们发现：

$$\text{DFT}(A[k]) = \frac{F_p[k] + F_q[k]}{2} \tag{3.3}$$

$$\text{DFT}(B[k]) = i\frac{F_p[k] - F_q[k]}{2} \tag{3.4}$$

于是我们将2次DFT合并为了1次，可以减少1次DFT。

3.2 更快速的卷积

我们能不能将2次DFT继续往下优化呢？当然能!

前面有提到过我们将多项式$A(x)$分为奇次项和偶次项的技巧，我们继续考虑用这个技巧优化。

同前面的定义，我们设$A_0(x)$是$A(x)$的偶次项的和，$A_1(x)$是奇次项的和，那么$A(x) = A_0(x^2) + xA_1(x^2)$。

假设我们要求$A(x)$与$B(x)$的乘积，且两个多项式的次数均为$n-1$（假设n为2的整数次幂），则有：

$$\begin{aligned} A(x)B(x) &= (A_0(x^2) + xA_0(x^2))(B_0(x^2) + xB_0(x^2)) \\ &= A_0(x^2)B_0(x^2) + xA_0(x^2)B_1(x^2) + xA_1(x^2)B_0(x^2) + x^2A_1(x^2)B_1(x^2) \\ &= A_0(x^2)B_0(x^2) + x(A_0(x^2)B_1(x^2) + A_1(x^2)B_0(x^2)) + x^2A_1(x^2)B_1(x^2) \end{aligned}$$

所以我们要求的式子是：

$$A_0(x^2)B_0(x^2) + x(A_0(x^2)B_1(x^2) + A_1(x^2)B_0(x^2)) + x^2A_1(x^2)B_1(x^2) \tag{3.5}$$

我们需要做4次多项式乘法，但是如果把譬如$A(x^2)$这种式子看作关于x^2的多项式，那么结果多项式的次数均不超过$n-1$。

所以，我们一开始需要做4次长度为n的DFT，两两合并可以优化为2次。

考虑IDFT，看似需要3次IDFT，其实不然，考虑(3.5)式右边的第一项和第三项，它们依然只有偶数项有系数，而第二项只有奇数项有系数。所以我们考虑合并第一项和第三项。

合并第一项和第三项我们显然也可以看成关于x^2的多项式，于是我们要做的就是根据$A(x)$的DFT求$xA(x)$的DFT，容易发现，$xA(x)$的DFT就是将$A(x)$的DFT的第k项乘上ω^k。

于是我们只需要2次长度为n的IDFT，如果我们能两两合并可以优化为1次。

我们考虑所有需要IDFT的序列，它们并不一定所有项都是实数，初看是难以合并的。

但是我们注意到一点，它们IDFT之后的结果是实数！注意到无论序列是否是纯实数，IDFT都是DFT的逆，我们考虑(3.3)(3.4)二式，我们知道右边能求左边，知道左边我们也能恢复右边，我们只需要对所有k恢复出所有的$F_p[k]$(或者$F_q[k]$)，然后复原$P(x)$(或者$Q(x)$)，再根据IDFT的结果是实数，我们可以直接取实部和虚部来得到IDFT的结果。这样，IDFT的次数被减少到了1次，大功告成。

所以我们只需要做3次长度为n即单倍长度的DFT，而前面的方法需要2次两倍长度的DFT，如果我们仍然以两倍长度来衡量的话我们甚至可以用"1.5次DFT"来形容刚才介绍的优化方法。

3.3 任意模数卷积

接下来我们来介绍如何在任意模数下进行卷积，这里假设模数M在10^9级别，要卷积的序列长度在10^5级别。

我们考虑两个长度为10^5级别的序列的卷积，考虑它们在实数域中卷积之后的结果，每个数的大小在10^{23}级别，一般的浮点类型显然会有误差。我们显然可以使用更高精度的浮点数，如python中的decimal等，但是python效率本身较低且目前不能在国内的NOI赛事中使用，而C++等其他很多语言中却没有类似decimal一样的高精度浮点类型，如果手写的话会非常麻烦且效率并不可观。

3.3.1 三模数NTT

考虑找三个大小为10^9且满足NTT性质的模数，分别求出在这三个模数意义下的卷积的结果，因为每个数的大小在10^{23}级别，所以根据中国剩余定理，我们可以唯一确定每个数。中国剩余定理的具体内容这里不再赘述。

具体实现怎么办呢，一种方法是利用128位整型或者高精度，128位整型在很多地方都不能用，而高精度太麻烦也比较慢。

另一种方法是用一种更巧妙的方式来使用中国剩余定理，我们假设模数分别是mod_0, mod_1, mod_2，先合并前两个模数，也就是求出答案在模$mod_0 \times mod_1$意

义下的值，然后用逆元将模$mod_0 \times mod_1 \times mod_2$意义下的数也就是答案表示成$k \times mod_0 \times mod_1 + b$的形式，这个东西我们不必真正求出，我们只需要在模$M$意义下求即可，这样只需要使用64位整型，而64位整型在基本上所有时候都是可以用的，且速度相对前面的方法非常快。

但是由于上述方法需要使用9次模意义下的DFT，效率仅仅只能用一般来形容。

3.3.2 拆系数FFT

我们设$M_0 = \lceil \sqrt{M} \rceil$，根据带余除法我们可以将所有整数$x$ 表示为$x = k[x]M_0 + b[x]$，其中$k[x]$和$b[x]$都是整数。

我们假设多项式$A(x)$的系数序列为a_i，多项式$B(x)$的系数序列为b_i，那么我们把$k[a_i], b[a_i], k[b_i], b[b_i]$形成的四个序列两两做1次卷积，卷积的结果中每个数的大小在10^{14}级别，使用精度较高的浮点数以及注意用2中介绍的精度较高的预处理单位复根方法即可接受，做完卷积之后，对每个卷积的结果乘上相应的系数并贡献到答案中去，如2.3.1节中提到的那样，我们可以预处理这四个序列的DFT，然后对于对应的系数相同的两个卷积我们直接合并，于是我们需要做7次DFT。

我们能不能把做DFT的次数继续减少呢，当然能!

我们考虑前面所述的过程，第一步是对4个数列分别做DFT，我们发现用前面所述的合并DFT技巧DFT是可以两两合并的，这样我们将DFT次数减少到了5次。

IDFT我们需要做3次，两两合并我们可以减少到2次。

于是DFT次数被成功的减少到了4次，大功告成。

本人的C++代码实现可以参见`http://uoj.ac/submission/49836`。

同样的，前面介绍的将普通卷积优化到3次单倍长度DFT的方法，显然也可以应用到任意模数卷积下，即实现所谓的“3.5次DFT”做任意模数卷积，但是式子太复杂且优化效果并不显著，这里不再赘述。

3.4 总结

本节介绍了一种十分有力而暂时并不是很普及的减少DFT次数的方法，最后我们发现**两个DFT只要它们都是实数序列是可以合并的，两个IDFT只要它**

们的结果都是实数序列都是可以合并的。这种优化基本不需要增加任何代码量，效率却有了很可观的提升。

然而这种优化也有一定的局限性：只适用于实数域的FFT，而不适用于$\mathbb{F}_\mathbb{P}$下的NTT。如果有人能想办法让这种做法能适用于NTT，欢迎与本人交流。

致谢

1. 感谢中国计算机学会提供学习和交流的平台。
2. 感谢雅礼中学的汪星明老师多年来给予的关心和指导。
3. 感谢杜瑜皓同学为本人写本文提供的巨大帮助。
4. 感谢金策同学的验稿。
5. 感谢机房里的其他同学为本文检漏挑错。
6. 感谢Vladimir Smykalov、MGCH3D 的出题人等最早提出本文所述的合并DFT 技巧的人。
7. 感谢其他所有本文所参考过的资料的提供者。
8. 感谢其他对我有过帮助和启发的老师和同学。
9. 感谢父母对我的关心和照顾。
10. 感谢各位百忙之中抽出宝贵的时间阅读本文。

参考文献

[1] 《炫酷反演魔术》，吕凯风

`http://vfleaking.blog.uoj.ac/blog/87`

[2] Fast Fourier Transform，彭雨翔

`http://picks.logdown.com/posts/177631-fast-fourier-transform`

[3] Codechef MGCH3D题解

https://discuss.codechef.com/questions/74772/mgch3d-editorial

[4] CZT的英文维基百科

https://en.wikipedia.org/wiki/Chirp_Z-transform

从Unknown谈一类支持末尾插入删除的区间信息维护方法

安徽师范大学附属中学 罗哲正

摘 要

Unknown是我的集训队互测题"我们仍未知道那天所看见的数据结构的名字"。这一类要求支持末尾插入删除以及区间查询的问题，近年来在信息学竞赛中时常出现。本文将通过详细介绍Unknown一题一步步的求解过程，对这一类问题几个不同的分支进行归类分析，总结并提出一些通用算法，并对算法的性能进行分析。本文的核心是总结提出了在不容易快速合并信息的情况下，处理在端点处插入删除并维护区间信息的通用方法。希望对大家有所帮助，并能引发大家对这一类问题的思考。

1 试题大意[1]

有一个元素为向量的序列S，下标从1开始，初始时S为空，现在你需要支持三个操作：

1.在S的末尾添加一个元素(x, y)。

2.删除S的末尾元素。

3.询问下标在$[l, r]$区间内的元素中，$(x, y) \times S_i$的最大值。

其中×表示向量的叉积，$(x_1, y_1) \times (x_2, y_2) = x_1y_2 - x_2y_1$

2 输入格式

第一行一个整数tp表示数据类型，以下多组数据（不超过3组），以$m = 0$结束。

[1]本题改编自SDOI2014向量集 http://www.lydsy.com/JudgeOnline/problem.php?id=3533

对于每组数据，第一行一个整数m表示操作数。

接下来m行，每行有三种形式：

1 x y 在S末尾添加元素(x, y)。

2 删除S末尾元素。

3 l r x y 询问下标在$[l, r]$区间内的元素中，$(x, y) \times S_i$的最大值。

3 输出格式

对于每组数据，你需要把每个询问的结果对$M = 998244353(7 * 17 * 2^{23} + 1$,一个质数)取模之后异或起来输出。

注意数学意义上的取模结果在$[0, M)$区间内，例如$-1 \mod M = M - 1$。

4 数据范围与约定

各测试点数据范围见下表：

测试点编号	m的规模	其他
1	$m \le 1000$	无
2		
3	$m \le 80000$	
4	$m \le 300000$	无2操作，3操作的询问为全部区间
5		所有2操作在1操作的后面，3操作的询问为全部区间
6		所有3操作都在1操作和2操作后面
7	$m \le 500000$	对于所有3操作有$l = 1$，内存限制为128M
8		无
9		内存限制为128M
10		内存限制为64M

对于全部数据，令n为任意时刻序列长度，$n \le 300000; m \le 500000$。

1操作个数不超过300000，且输入的x, y满足$-10^9 \le x \le 10^9; 1 \le y \le 10^9$。

3操作个数不超过300000，且输入的x, y满足$1 \le x \le 10^9; -10^9 \le y \le 10^9$。

3操作中，l, r满足$1 \le l \le r \le n$。

时间限制为5s(Tsinsen)/3s(UOJ)，内存限制为512MB.

5 算法介绍

5.1 算法1：暴力枚举

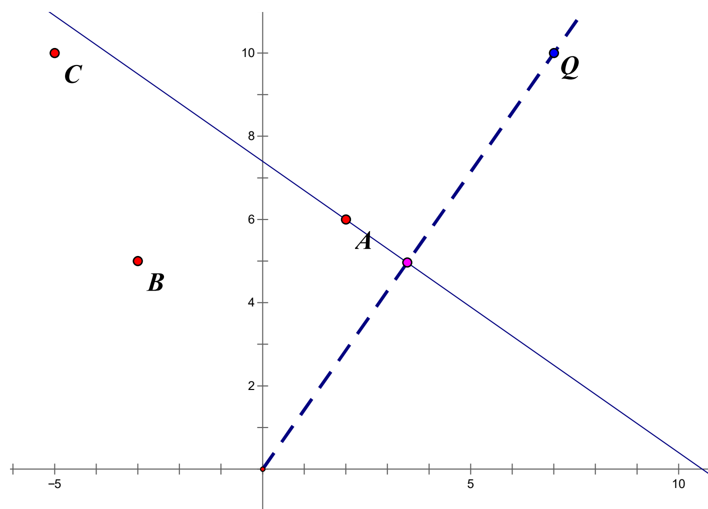
首先我们考虑一下暴力的写法，初看之下这道题是一道数据结构题，数据结构题的暴力一般都比较好写，只要按照题意模拟即可。对于询问区间$[l,r]$，我们可以简单粗暴的枚举$[l,r]$内的所有元素S_i，并统计$(x,y)\times S_i$的最大值即可。

分析这个算法的复杂度，**由于n,m同级，故在复杂度计算时不再区分**，询问次数是$O(n)$的，而区间长度也可以达到$O(n)$级别，所以暴力算法的时间复杂度$O(n^2)$，期望得分20分。

5.2 观察答案的性质

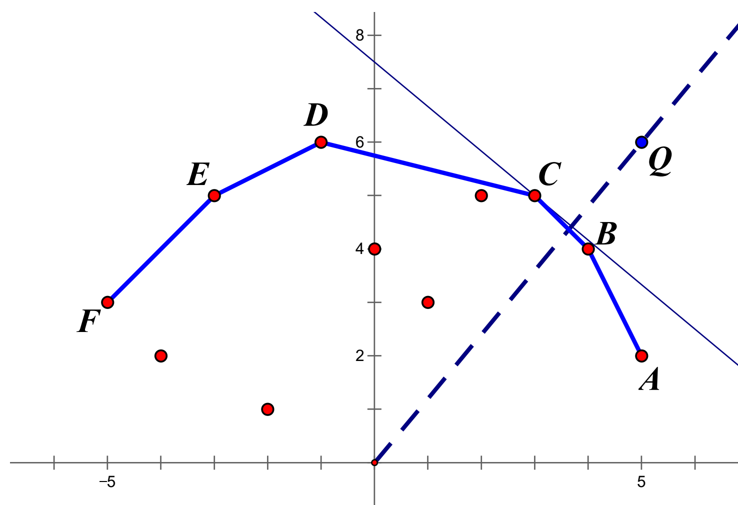
点积较叉积更为简单，我们不妨把叉积写成点积的形式，令$S_i=(x_i,y_i)$，对于询问x,y，有$(x,y)\times(x_i,y_i)=(-y,x)\cdot(x_i,y_i)$。把$(x,y)$变换成$(-y,x)$，则问题变为求出$(x,y)\cdot(x_i,y_i)$的最大值，之后的讨论将不再使用叉积。

由于$\overrightarrow{OA}\cdot\overrightarrow{OB}=\left|\overrightarrow{OA}\right|\cdot\left|\overrightarrow{OB}\right|\cos\angle AOB$，答案与$\overrightarrow{OB}$在$\overrightarrow{OA}$方向上的投影长度成正比，故点积最大值可以考虑这样来求，使用一条与$\overrightarrow{OQ}$垂直的直线，从无穷远处向原点O扫过来，碰到的第一个点即为答案。



例如上图，直线第一个扫到的点为A，则答案就是$\overrightarrow{OQ}\cdot\overrightarrow{OA}$。

继续观察$\overrightarrow{OQ},\overrightarrow{OS_i}$的角度取值，发现它们的角度(arctan)都是在$(0,\pi)$区间内，容易发现答案一定在区间内元素的上凸壳上。



例如上图，询问$\overrightarrow{OQ}$的答案就是$\overrightarrow{OC}$。

如果我们可以快速求得询问区间的上凸壳，那么在凸壳上二分斜率可以做到单次询问$O(\log n)$的复杂度。

于是问题的关键变成了如何维护凸壳。

5.3 以凸壳为代表的一类区间信息的维护

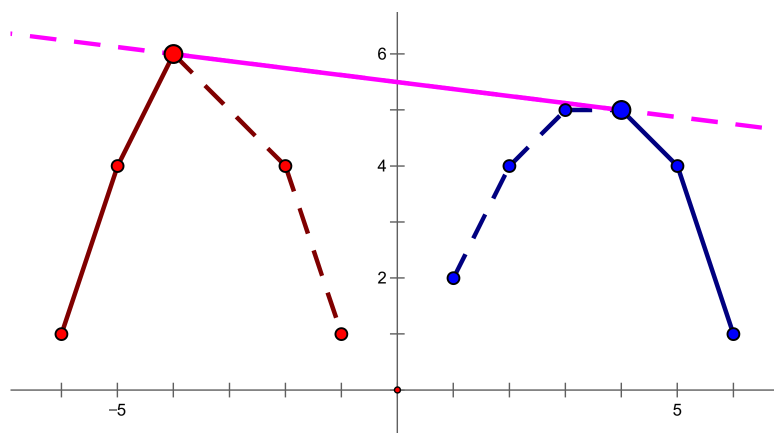
在维护区间信息时，有一些信息本身的特性是需要关注的，并且直接影响了我们维护区间信息的方式与难度。

5.3.1 是否支持快速合并

对于两个区间$[l, mid]$和$[mid+1, r]$，若这两个区间的信息能在$O(1)$或$O\left(poly\left(\log(r-l)\right)\right)$的时间复杂度内合并，我们就称这种信息支持快速合并。

支持快速合并的例子：

- 区间最值（合并复杂度$O(1)$）
- x关于下标单调的凸壳（使用平衡树维护并二分斜率是可以做到在$O\left(\log^2 n\right)$的时间复杂度内合并的，如下图所示）



不支持快速合并的例子：

- 区间点集的凸壳
- 区间数集的mex[2]

支持快速合并的信息都可以使用线段树等分治类区间数据结构进行维护，例如线段树维护区间最大/最小值已经是家喻户晓的技能了。

5.3.2 是否支持快速插入删除

对于一个区间$[l, r]$，若我们能在$O(1)$或$O(poly(\log(r-l)))$的时间内求出$[l-1, r]$和$[l, r+1]$的值，则我们称这种信息支持快速插入。

同理若能在$O(1)$或$O(poly(\log(r-l)))$的时间内求出$[l+1, r]$和$[l, r-1]$的值，称这种信息支持快速删除。

一个显而易见的结论就是：支持快速合并的信息一定支持快速插入（例如在右端插入可以看成合并区间$[l, r], [r+1, r+1]$）。

支持快速插入而不支持快速删除的例子：

- 区间点集的凸壳（使用平衡树等数据结构维护）
- 区间点集的虚树（使用平衡树维护相对dfs序）

支持快速删除而不支持快速插入的例子：

[2]mex:集合中未出现的最小自然数

- mex

同时支持快速插入删除的例子：

- 区间和的常数次多项式函数（如区间和的平方）
- 区间内不同的元素个数

支持区间快速插入删除的信息维护都可以使用莫队算法[3]来解决。由于莫队算法与本文的讨论核心关系不大，这里不展开叙述。

5.4 算法2：分块

由于凸壳本身不支持快速合并和快速删除，而本题是存在删除操作和区间询问的，于是传统的区间分治类数据结构如线段树是不能直接维护的，我们考虑分块。

将序列分为S块，每块大小为$\frac{n}{S}$，对每个块维护块内元素的凸壳。插入和删除的时候重构最后一块的凸壳，于是插入和删除的复杂度都是$O\left(\frac{n}{S}\right)$。

考虑询问区间$[l,r]$，我们需要枚举被区间$[l,r]$完全包含的区间，并在这些区间的凸壳上二分，复杂度是$O(S\log n)$的。

于是总复杂度就是$O\left(\frac{n^2}{S}+nS\log n\right)$，当取$S=\sqrt{\frac{n}{\log n}}$时，时间复杂度最小为$O\left(n\sqrt{n\log n}\right)$。

5.5 算法3：二进制分组

测试点4特殊性质：无2操作，3操作的询问为全部区间。

这个测试点中，我们只要维护全部元素的凸壳，并且只需要支持插入操作即可。

这个问题比较经典，可以使用平衡树维护凸壳。本文在这里介绍一种非传统的方法，采用二进制分组[4]来解决。

我们把序列分为连续的若干组，对每组维护它的凸壳，初始时序列为空。每插入第一个元素时，把这个元素单独分为一组，显然这一组的凸壳就是这个元素本身，如果末尾的两组大小相同，就把末尾的两组合并成一组。

[3]莫队算法：详见2010年集训队作业《小z的袜子》题解

[4]二进制分组：详见许昊然2013年国家集训队论文《浅谈数据结构题的几个非经典解法》

容易证明，这样的分组方式会使得每一组的大小都是2的次幂，并且组的大小严格递减，从而任意时刻组数不会超过$O(\log n)$。

考虑合并两个大小相等的组（不妨设大小为m），由于两个点集并的凸壳一定在两个点集的凸壳的并上，我们可以直接按照x坐标为关键字合并两个组的凸壳，再用Graham算法$O(m)$的计算凸壳即可。

下面分析合并过程的时间复杂度，考虑每个元素对于复杂度的贡献，一个元素被合并一次，所在组的大小一定会翻倍，所以每个元素对复杂度的贡献都不会超过$O(\log n)$，于是复杂度就是$O(n\log n)$。

注意到询问操作需要在每个组内的凸壳上都进行一次二分，所以复杂度是$O\left(n\log^2 n\right)$的，于是这个算法的总复杂度就是$O\left(n\log^2 n\right)$的。

5.6　算法4：时间倒流

测试点5特殊性质：所有2操作在1操作的后面，3操作的询问为全部区间。

二进制分组只涉及到新建与合并，是不能处理删除操作的，观察这个数据类型的性质，所有的删除都在插入后面，这提示我们可以把操作序列分成两半处理。

于是，对于前一半，测试点4的二进制分组做法，对于后一半，我们把时间倒流，这样删除操作就变成了插入操作，依旧套用二进制分组的做法。

时间复杂度为$O\left(n\log^2 n\right)$，把序列分成两部分的想法提示我们往分治的方向思考。

5.7　算法5：线段树维护静态序列

测试点6特殊性质：所有3操作都在1操作和2操作后面。

把所有插入删除操作都处理完之后，询问是在一个静态序列S上的，对于静态序列上的区间询问，我们可以使用经典的分治区间数据结构线段树。

对线段树上每个区间$[a,b]$我们维护该区间内所有元素的凸壳，再建树的时候我们可以直接采用测试点4中提到的线性合并凸壳的做法。这部分的复杂度是$T(n)=2T\left(\frac{n}{2}\right)+O(n)=O(n\log n)$。

接下来考虑查询区间$[l,r]$，一个区间能拆成$O(\log n)$个线段树节点所代表的区间的并，我们在这些节点上存储的凸壳上二分即可，时间复杂度是$O\left(n\log^2 n\right)$。

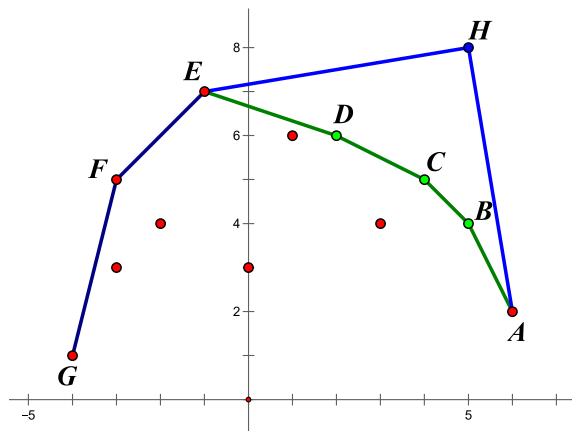
5.8 算法6：在操作树上分治

测试点7特殊性质：对于所有3操作有$l = 1$，内存限制为128M。

5.8.1 数据结构的做法

本题不要求强制在线，那么对于一个询问$[l, r]$，一定存在某个时刻序列长度恰好为r且与询问时序列相同，我们把询问记录在这个时刻上。又因为$l = 1$，所以相当于每次询问全体序列，我们只要维护所有元素的凸壳即可。

考虑使用伸展树(Splay)进行维护，注意到插入一个点时从凸壳中替换下来的点在原凸壳上一定形成一个区间，如果在节点上记录斜率，我们就可以二分出区间的左右端点，然后在Splay上把这一段分割下来并保存，在删除末尾元素的时候，把分割下来的区间再合并回去即可。



例如上图，H点加入之后把B, C, D三个点替换下来并保存。

由于记录了斜率，询问可以直接在Splay上二分，这样所有操作的复杂度都是$O(n \log n)$。

5.8.2 序列上的做法

插入和删除操作考虑起来有一定的困难，我们不妨先考虑序列为静态的情况，每次询问序列为一个前缀$[1, p]$。

我们可以采用分治做法，考虑处理一个区间$[l, r]$，令$mid = \frac{l+r}{2}$，我们把所有$p \geq mid$的询问拿出来，注意这些询问都是覆盖了$[l, mid]$区间的，我们一次性

处理$[l, mid]$区间中的元素对p在$[mid, r]$区间里的询问的贡献。处理完贡献之后，我们按照$p < mid$和$p \leq mid$把询问划分成两部分，分别往左边和右边递归分治计算。当分治到区间内只有一个元素的时候，直接用这个元素更新这个点上的所有询问即可。

下面就是考虑如何快速处理$[l, mid]$内元素对$[mid, r]$区间内询问的贡献：

方法A:我们对$[l, mid]$区间内元素建立出凸壳，然后枚举右边每个询问，在左边已经建立好的凸壳上二分即可，这样一次计算贡献复杂度是$O(n \log n)$的。

方法B:同样先对$[l, mid]$区间的元素建立出凸壳，然后把$[mid, r]$区间里的询问按照极角序排序，从前往后处理询问，凸壳上最优点的移动是单调的。所以我们只要设两个指针i, j分别在询问序列与凸壳上扫描即可，下面给出这部分的伪代码。

Algorithm 1 用两个指针i, j在排好序的询问和凸壳上扫描更新答案

```
for i = 0 to Q.size − 1 do
    while j + 1 < H.size and H_{j+1} · Q_i > H_j · Q_i do
        j = j + 1
    end while
    ans_i = max(ans_i, H_j · Q_i)
end for
```

一次扫描的时间复杂度为$O(n)$，由于要排序，实际复杂度是$O(n \log n)$。注意到排序是可以在分治的同时使用归并排序完成的，所以总复杂度$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n \log n)$。

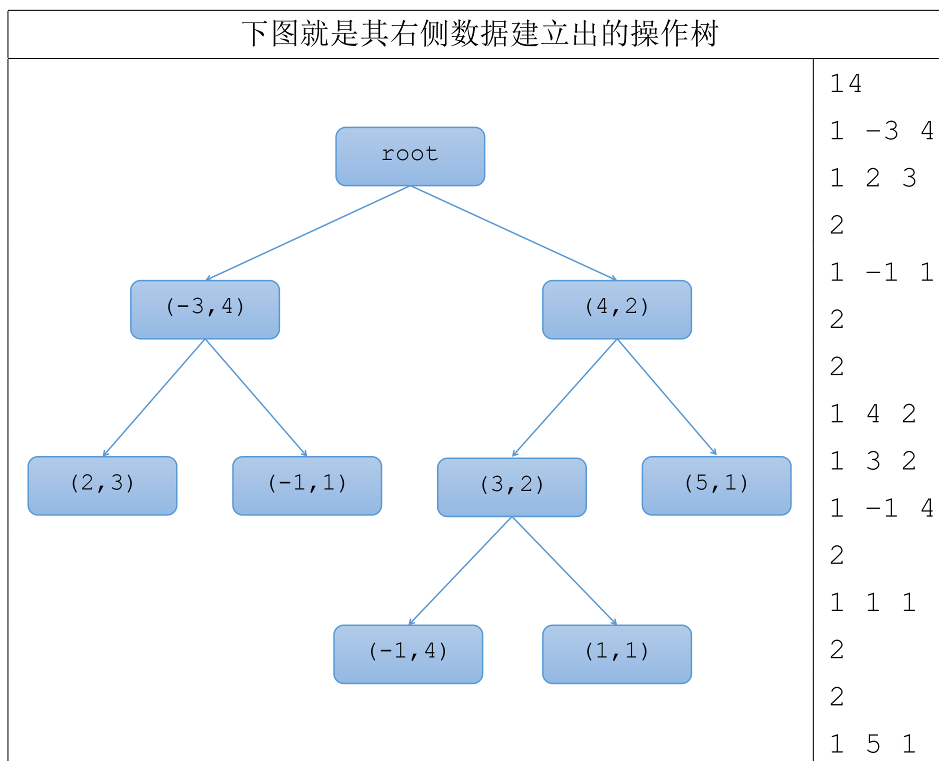
分治算法在静态的序列上复杂度并不比平衡树维护凸壳优秀，但是却有更良好的推广性。

5.8.3 操作树上的推广

观察数据结构做法，发现很像在一棵树上进行dfs的过程。既然问题可以离线，我们干脆把这棵操作树建立出来。操作树建立方法如下：

1. 初始时建立一个根$root$，令指针$p = root$
2. 对于插入操作，新建一个点x，令$father_x = p$，再令$p = x$

3. 对于删除操作，令$p = father_p$



由于只在插入时新建节点，这样建立出的操作树节点数是不超过插入操作的个数的。接着我们可以发现询问区间是树上一条自上而下的链，在本测试点中，链的顶端是$root$，我们把询问链为$root \to p$的询问记录在节点p上。

我们考虑推广分治算法，在操作树上，我们可以直接使用点分治。方便起见我们在点分治中定义一些记号，对于树上一个连通块，定义其中深度最浅的节点作为连通块的根，记为R，重心记为G。

我们在分治结构的每一层一次性处理链$R \to G$上的元素对G子树内的所有询问的贡献，这可以直接套用上文说的方法A或方法B。因为树分治结构不容易使用归并算法优化，所以方法A和方法B一次计算贡献的复杂度都是$O(n \log n)$。

由于重心把树分成了至少两个大小不超过$\frac{n}{2}$的连通块，于是总复杂度$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n) = O\left(n \log^2 n\right)$。

5.9 算法7（满分做法A）

通过思考解决$l = 1$的情况（测试点7），我们拿到了一枚利器——树分治，

接下来我们尝试去解决原问题。

首先套用算法6，建立出操作树，然后在操作树上点分治，唯一不同的是计算链$R \rightarrow G$上的元素对G子树内的所有询问的贡献。因为$l = 1$的条件并不满足，所以G子树内的询问的区间不一定完整包含链$R \rightarrow G$，而是这条链的一个后缀。

我们把这个问题拿出来，发现这个问题与5.8.2中提到的在序列上询问前缀是等价的。于是我们把链$R \rightarrow G$取出并翻转，将询问加入之后采用5.8.2的做法，选用归并优化的做法B，可以做到一次计算$O(n \log n)$的复杂度。

于是这个算法的时间复杂度就是$O\left(n \log^2 n\right)$。

树分治和归并算法的空间复杂度都是$O(n)$，所以空间复杂度是$O(n)$，可以拿到100分。

5.10 算法8（满分做法B）

观察问题的本质，把插入删除操作建立成一棵操作树T之后，询问区间的本质就是树上自上而下的路径。

我们仍旧考虑点分治，一次性处理所有过重心G的所有询问。首先把询问链$u \rightarrow G \rightarrow v$，拆成两条询问链$G \rightarrow u, G \rightarrow v$。观察这个问题，发现如果我们令$G$为根，则与测试点7中的情况完全相同，直接使用5.8.1中的数据结构做法可以得到$O(n \log n)$的复杂度。

于是总复杂度就是$O\left(n \log^2 n\right)$，树分治和Splay的空间复杂度都是$O(n)$，于是空间复杂度也是$O(n)$的，可以拿到100 分。

5.11 算法9（满分做法C）

树分治部分和算法7相同，在计算链$R \rightarrow G$上的元素对G子树内的所有询问的贡献时，使用数据结构维护。从G开始到R向上依次添加节点，使用数据结构维护加入的节点的凸壳。对于每个询问后缀，我们在加入到这个位置的时候在数据结构上二分出答案即可。

由于这里的数据结构只需要支持加点操作，而加点时暴力维护凸壳是能保证复杂度的，所以大多数平衡树或者直接使用set即可完成。

时间复杂度$O\left(n \log^2 n\right)$，可以拿到100分。

5.12 算法10：树链剖分

在5.8.3中我们已经把询问区间转化为树上的链，大多数关于树链的问题，除了点分治之外，还可以采用树链剖分来解决。树链剖分算法本身已经较为普及，这里不再赘述。

对每个询问链，我们使用树链剖分结构将其分成$O(\log n)$个dfs序中的区间。观察剖分出来的这些区间的性质，可以发现除了深度最浅的那个区间之外，所有的区间都是某条重链的前缀。

我们对dfs序按照算法5建立线段树，然后把不是重链前缀的区间在线段树上查询更新。接下来考虑所有重链，对于每条重链，自上而下插入节点，并使用数据结构进行维护。

同样只要支持插入即可，所以大多数平衡树或者set都可以完成。

考虑这个算法的时间复杂度，建立线段树是$O(n\log n)$的，对每个询问不是重链前缀的部分在线段树上查询是$O\left(n\log^2 n\right)$的，扫描每条重链并维护凸壳，使用Splay的话复杂度是$O(n\log n)$的。由于一个询问被拆分成了$O(\log n)$个重链前缀，每个前缀都需要做一次二分，所以复杂度是$O\left(n\log^2 n\right)$的。于是总复杂度就是$O\left(n\log^2 n\right)$的。

注意到这个算法需要把每个询问剖分出的$O(n\log n)$条重链记录下来，于是空间复杂度是$O(n\log n)$的，可以拿到90分。

5.13 算法11：线段树分治

接着考虑算法10，一个询问被拆分成$O(\log n)$个dfs序区间。而序列上的区间修改或询问是可以使用线段树**离线**处理的，这种方法叫线段树分治。我们首先来看一个例子：

5.13.1 例：向量[5]

你要维护一个向量集合，支持以下操作：

1. 插入一个向量(x, y)
2. 删除插入的第i个向量
3. 查询当前集合与(x, y)点积的最大值是多少。如果当前是空集输出0

[5]题目来源：NOI2015模拟题by 杨定澄

允许离线。

100%的数据：$n \le 200000, 1 \le x, y \le 10^6$

题解

对于每个向量，我们都能预处理出它加入集合的时间与从集合中删去的时间，那么每个向量在时间轴上就是一个区间。

考虑对时间轴建立线段树，把每个向量插入线段树可以拆分成线段树上$O(\log n)$个节点。

接下来对每个点建立凸壳，则对于每个询问，自上而下查询每个包含它的区间即可，这样的复杂度是$O\left(n\log^2 n\right)$。

考虑进行一些优化，我们把询问按照极角排序，接着用5.8.2中讲的做法线性求得答案，于是瓶颈在于排序。

事实上，如果我们把所有的向量按照x坐标排序后插入，把所有的询问按照极角排序后插入，那么在线段树的节点上就不用排序了，于是总复杂度就变成了$O(n\log n + Q\log Q + Q\log n)$。

5.13.2　本题中的线段树分治

注意到“向量”与本题有一些区别，“向量”中的询问是单点，而集合中的向量覆盖的是区间，本题中我们询问的是向量序列的区间信息。

我们依然可以沿用线段树分治的做法，首先把询问按照极角序插入线段树，然后在线段树上dfs，对每个线段树节点使用归并求出该节点所代表区间的凸壳，接着仍然使用5.8.2中提到的算法更新答案即可。

这样复杂度是$O(n\log n + Q\log n)$，由于我们使用树链剖分把一个询问拆成了$O(\log n)$个，于是$Q = O(n\log n)$，总复杂度就是$O\left(n\log^2 n\right)$。

由于要在线段树上保存$O\left(n\log^2 n\right)$个询问，故空间复杂度是$O\left(n\log^2 n\right)$的，可以拿到70-90 分。

5.14　算法12：再探二进制分组

算法6-11都要求允许离线，那么有没有一个可以支持在线的做法呢？

在算法3中我们引入了一种在线处理插入的方式——二进制分组，考虑对这个算法进行改进。

我们建立$t+1$层分组结构，t为最小满足$2^t \geq n$的整数（n是当前区间长度），第i层$(i \leq i \leq t)$的组大小为2^i，有$siz_i = \left\lceil \frac{n}{2^i} \right\rceil$组。注意到第$i$层的每一组都在第$i-1$层上被均分成两个等长区间，所以这是一个线段树结构，于是可以按照线段树定义左右孩子和父亲的关系。

如果只有插入还是一样的做法，首先在第0层添加一组，接着从第$i=1$层开始，如果第i层可以增加一组，那么将其左右孩子组合并得到新添加的一组。

删除操作不太好处理，因为删除一个元素可能导致前若干层的最后一个组信息错误，而每次暴力重构的复杂度是$O(n)$，这是不可接受的。我们引入替罪羊树的思想，当一个组信息出现错误的时候，我们不立即重构更新，而是给这个组打上标记，查询到这个组的时候我们递归它的左右孩子，直到碰到没打标记的组才在组内二分计算答案。但是也不能一直不更新，否则查询时递归到底层节点数会达到$O(n)$个，这也是不可接受的。于是考虑这样一种更新机制：

对于每层，我们至多允许最后一个组是有标记的。

这样，当第i层需要增加一组的时候，我们才把原来的最后一组暴力更新一下；删除的时候只要自下而上打标记即可。

接下来计算这个算法的时间复杂度。

5.14.1 重构更新的复杂度

对于第i层，我们统计两种事件：

1. 最后一个组被打标记，复杂度$O(1)$。
2. 重构最后一个组并添加带有标记的新的一组，复杂度$O\left(2^i\right)$。

令len_i表示第i层没有标记的组的总长度，令$E_i = n - len_i$，那么事件1是相当于E_i从0变为2^i，事件2是相当于E_i从$2^i * 2$变为2^i。而一次插入或删除操作对于E_i的改变幅度是± 1。

令势能函数$\delta(i) = \left|E_i - 2^i\right|$，则一次插入或删除操作对$\delta(i)$的贡献至多为1，而一次事件对$\delta(i)$的贡献为$-2^i$。于是两次事件间的操作数至少是$2^i$的，于是均摊复杂度是$O(n)$的。

考虑到有t层，于是重构更新的总复杂度就是$O(tn)$。

5.14.2 查询的复杂度

之前提到这是一个线段树结构，那么首先把询问区间$[l, r]$拆成$O(\log n)$个组。对于每个组，有三种情况，考虑每一种情况的贡献：

1. 没有被打标记。

   对于这种情况，直接查询这个组即可。

2. 有标记而且左右孩子都没有被打标记。

   对于这种情况，我们会在左右孩子中分别查询，仅仅是常数乘以2。

3. 有标记而且右孩子上也有标记。

对于这种情况，我们在左孩子中查询之后递归到右孩子。注意到这里的左孩子一定是某一层的倒数第二个组，这样的组只有$O(\log n)$个。

于是我们查询的区间总数是$O(\log n)$的，于是处理一次询问操作的复杂度就是$O\left(\log^2 n\right)$。那么总复杂度就是$O\left(tn + n\log^2 n\right) = O\left(n\log^2 n\right)$。由于要维护$t$层，所以空间复杂度是$O(tn) = O(n\log n)$的，可以拿到90分。

6 思路小结

本题比较复杂，直接思考起来较为困难，大致分为几个步骤。

首先分析出需要维护的信息，本题中我们需要维护询问集合的凸壳，凸壳是典型的不易快速合并的信息。

接着考虑部分分，并从中获得一些提示。

只有插入且询问全局的部分可以使用二进制分组在线完成。

没有修改的部分可以采用线段树完成。

$l = 1$的部分是比较关键的部分，而且难度较大。我们考虑了序列上的情况，给出了分治的做法。接着引入了操作树的概念，并把分治算法推广到树上即点分治算法。

拿到点分治这一门武器之后，我们尝试解决原问题，观察原问题与$l = 1$的部分不同的地方在于子树内的询问询问不一定是根到重心的链，可能是这条链的一个后缀。继续观察发现这个问题就是$l = 1$的部分在序列上的问题，继续分

治解决。于是我们得到了一个时间复杂度$O\left(n\log^2 n\right)$，空间复杂度$O(n)$的优秀算法。

可见，虽然标题是“我们仍未知道那天所看见的数据结构的名字”，但是我给出的满分算法并没有用到数据结构，而是采用了两次分治简化问题。

其实本质就是把CDQ分治[6]的思想放到树上，序列分治变成点分治，观察到每层分治结构上的贡献统计是序列上的特殊子问题并继续套用分治是求解本题的关键。

以上算法要求离线，考虑强制在线加强版本，空间要求放宽到$O(n\log n)$。

这使得我们回想起只有插入的部分分是支持强制在线的，于是把二进制分组改进成线段树结构，并引入替罪羊树思想。

最后我们设定每一层至多只有最后一个区间被打了标记，并分析出了维护信息的复杂度是$O(n\log n)$的，瓶颈在于二分，于是总复杂度就是$O\left(n\log^2 n\right)$。

应该说这是这一类问题较为通用的在线算法，其空间复杂度是$O(n\log n)$。

本题到此已经完美的解决了，但本题的意义不止于此，在解决本题时用到的思想和方法都有很大的实用与推广价值。

7　简单推广

7.1　双端插入删除

以上的讨论范围都是在末尾插入删除元素的问题。其实我们可以非常容易的推广到两端插入删除的问题上来。

7.1.1　算法7的推广

对于算法7，我们考虑改进操作树的构建方法：

1. 初始时建立一个根$root$，令指针$p = q = root$
2. 对于在开头插入操作，新建一个点x，令$father_x = p$，再令$p = x$
3. 对于在末尾插入操作，新建一个点x，令$father_x = q$，再令$q = x$

[6]CDQ分治：详见陈丹琦2008年国家集训队论文《从Cash谈一类分治算法的应用》

4. 对于在开头删除操作，令$p = father_p$

5. 对于在末尾删除操作，令$q = father_p$

对于每个询问区间$[l, r]$，其在树上对应的链就是链$p \to q$上第l个节点到第r个节点组成的链。但是我们并不能直接使用算法7处理，因为算法7要求询问链必须是树上自上而下的区间，而改进之后的操作树并不能满足这一点。

因此需要改进，只要把一条询问链$u \to v$拆成$w \to u$和$w \to v$两条链即可，其中w是u, v的最近公共祖先(LCA)。

同样算法9也可以推广到改进后的操作树上来，算法8由于是处理过根的询问，所以并不需要把询问拆成两条自上而下的链，直接处理即可。详细的扩展方法与算法7类似，这里不再赘述。

7.1.2 算法12的推广

二进制分组只能支持在一端操作，我们考虑算法12中改进的二进制分组，其实这是一个线段树结构，那我们可以直接使用线段树来代替。

初始时建立长度为$2n$的线段树，并把初始位置放在正中间。我们把更新机制改为每层最多允许两个组有标记（左右各一个），则插入删除时直接沿用算法12的处理方法。所有的复杂度分析在这里依然是成立的（我们只要把左端和右端的δ分开计算即可），只是询问的时候，由于两边两边的倒数第二组可能都被访问，所以常数会乘以2。

7.2 维护其他不能快速合并的区间信息

事实上，在信息不易快速合并时，要求支持末尾插入删除和区间询问的问题大都可以使用本文阐述的算法来完成。我们来看一个例子：

7.2.1 例：吉利树[7]

给一棵n个点的带边权的树，你需要维护一个序列，支持三种操作：

1.在末尾插入一个点x

2.删除末尾的点

[7]题目来源：by吉如一

3.询问区间$[l,r]$中离x最远的点到x的距离。

操作数为m,$n,m \le 100000$。

Subtask1：允许离线

Subtask2：强制在线

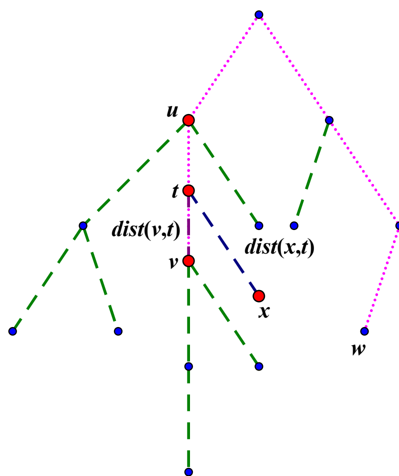
考虑需要维护的信息

若给定一棵树，如何求距x最远的点？我们可以采用树形DP预处理出树上每个点的最远点，这样时间复杂度是$O(n)-O(1)$[8]的。

若给定一棵树和一个树上节点的集合S，求x到S中最远点的距离该如何解决呢？我们依旧采用树形DP预处理，时间复杂度仍然是$O(n)-O(1)$。当$|S|$很小的时候这个算法的复杂度是不够优秀的，考虑使用虚树算法[9]。虚树算法在信息学竞赛中已经较为普及，这里不展开叙述。我们知道建立虚树的时间复杂度是$O(|S|\log n)$的，接着我们在虚树上树形DP预处理出每个虚树上点u到点集中的最远点距离d_u，对于每个询问点x，我们在dfs序上二分求出x到虚树上最近的两个点u,v，显然u,v必为虚树上的一条树边，设u是v的父亲，令$t=LCA(v,x)$。

若x的最远点在s一侧($s=u\ or\ v$)，则答案为$dist(x,t)+d_s-dist(s,t)$，故$ans(x)=\max(dist(x,t)+d_u-dist(u,t),dist(x,t)+d_v-dist(v,t))$。

下面看一个图例理解一下：



如上图所示，$d_v=dist(v,w)$，$ans(x)=dist(x,w)=dist(x,t)+d_v-dist(v,t)$。

[8] $O(n)-O(1)$：即预处理$O(n)$，每次询问$O(1)$

[9] 虚树算法：http://www.lxway.com/40452412.htm

Subtask1题解

首先建立出操作树将询问区间变成树上的一条链。

如果使用算法7点分治套分治，那么需要解决的就是区间信息合并的问题。

虚树的构建是基于dfs序的，那么我们只要维护区间元素按dfs序排序的结果，那么有序表的合并是$O(n)$的，由于建立虚树的复杂度瓶颈是求LCA，而LCA是可以使用欧拉序列+RMQ优化为单次$O(1)$的，于是一次合并的总复杂度就是$O(n\log n)$。

这样内层分治的复杂度是$O(n\log n)$，故外层分治的复杂度为$O\left(n\log^2 n\right)$，可以解决这个子任务。

考虑另一种方法例如算法8，那么要解决插入元素x的问题。

我们采用平衡树按照dfs序维护虚树节点，那么插入元素时在平衡树内二分找到离x最近的边从而找出虚树树边上离x最近的点y，把x和y都插入集合中。那么每次插入元素只要在平衡树中插入至多两个节点，对于虚树边的修改次数也是$O(1)$的，复杂度是$O(\log n)$，同时我们只要记录修改就能方便的支持回退了。

算法9的解法与算法8类似，这里不再叙述。

Subtask2题解

算法12只需要合并区间，而区间信息的合并在7.2.1中已经可以做到$O(n)$的复杂度。于是我们直接把算法12维护的区间信息改为虚树就可以解决本题了，复杂度的分析和Unknown一题完全一样，重构更新均摊$O(n\log n)$，查询$O\left(n\log^2 n\right)$可以解决本题。

8　命题思路与总结

8.1　Unknown命题思路

近年以来出现了很多要求维护区间信息的题目，这类题目大多使用区间数据结构维护。相对不容易维护的信息中以凸壳出现最为频繁，需要通过对时间分治，二进制分组等非传统方法解决的题目也越来越多。

NOI2014第二天第三题“购票”是一道典型的需要维护凸壳并支持区间查询来优化DP的题目，吕凯风学长现场使用树分治算法AC了此题，并在不久之后

把这个思想分享给了我。这个思想对我很有启发，并激发起了我对这一类信息维护方法研究的兴趣。

我研究这一类算法并命制的集训队互测题“我们仍未知道那天所看见的数据结构的名字”是一道有难度的题目，其部分分做法涵盖了多个非传统方法求解传统数据结构题的分支，例如二进制分组，时间倒流，线段树分治等。而得到其满分做法需要选手有敏锐的观察力，严谨的分析能力以及对分治算法的深刻理解和熟练掌握。同时这题对代码能力的要求也比较高，预计选手可能要花上2至4个小时才能拿到本题的全部分数。

8.2 方法总结

本文的最后总结出这一类在端点处插入删除并维护不易快速合并的区间信息的通用思考过程与解决方法。

如果允许离线，建立出操作树并考虑分治，对于能$O(n)$合并区间的可以继续套用分治（算法7）；否则观察信息是否同时支持加入和删除元素，可以的话使用数据结构从重心开始dfs统计（算法8）；如果只能支持快速插入和快速删除中的一个，可以按照某个顺序扫描根到重心的链（算法9）。

以上算法的时间复杂度都是$O\left(n\log^2 n\right)$，空间复杂度都是$O(n)$。

如果强制在线，那么考虑引入替罪羊树思想的二进制分组（算法12）。在复杂度分析中，复杂度瓶颈不在于重构更新，而在于询问。于是我们可以接受$O(n\log n)$的重构复杂度，时间复杂度仍然是$O\left(n\log^2 n\right)$的。对于有的题目，我们可以采取更优的查询方法（例如归并定位），将询问复杂度优化到$O(\log n)$，从而把总复杂度降为$O(n\log n)$

由于要建立$O(\log n)$层分组结构，空间复杂度是$O(n\log n)$ 的。

Unknown只是诸多此类问题的一个代表，存在许多种不同的解法，这些解法各有各的长处并都具有一定的实用性，所以我觉得激发选手思考这方面算法的意义甚至大于题目本身。希望我的这道题起到一个抛砖引玉的作用，能引起大家对于这一类问题的兴趣，并提出一些更有效，更方便的算法。如果在此方面的研究有所收获欢迎大家来找我讨论。

9 感谢

- 感谢叶国平老师在学习生活上的指导和关心。
- 感谢CCF提供交流的平台与机会。
- 感谢吕凯风学长提供命题灵感。
- 感谢吉如一同学，杨家齐同学，吴作凡同学，毛啸同学给予的帮助。
- 感谢吴作凡同学，吉如一同学，倪星宇同学，王蕴韵同学为本文审稿。

参考文献

[1] SDOI2014 向量集 题解 http://www.cnblogs.com/joyouth/p/5350714.html

[2] 许昊然,《浅谈数据结构题的几个非经典解法》,2013年信息奥林匹克中国国家队候选队员论文

[3] 陈丹琦,《从Cash谈一类分治算法的应用》

小C的后缀数组命题报告

绍兴市第一中学 洪华敦

摘 要

2016年集训队互测第三场中，一道传统字符串题的命题思路和题目分析

1 题目大意

小C写了一个后缀数组求两个后缀最长公共前缀(lcp)的程序，他的算法流程是这样子的：

1.读入一个长度为n的字符串$a[1..n]$

2.定义数组$sa[1..n]$，$sa[i]$表示将所有后缀排序后，第i小的后缀是$a[sa[i]..n]$，定义两个字符串的比较方式：逐位比较，如果一个字符串是另一个字符串的前缀，那么前者较小

3.定义数组$rank[1..n]$，令$rank[sa[i]] = i$

4.对于给定的询问l, r，设$x = min(rank[l], rank[r])$，设$y = max(rank[l], rank[r])$

5.对于每个满足$x \le i < y$的i，求出$lcp(a[sa[i]..n], a[sa[i+1]..n])$，最小值就是答案

（注意，第5步中的lcp，使用的是一个暴力的算法，与传统后缀数组的求$heigh$方法无关，你不需要关心他如何实现）

原本这是一个相当完美的算法，但是有个熊孩子在第二步和第三步中间加了一步：将sa数组随机打乱，即sa数组变成了一个随机生成的排列

现在给定字符串和询问，小C想知道被熊孩子打乱后的期望输出是什么

为了避免精度误差，你需要将答案乘上$n!$后对998244353取模后输出

2 读入格式

第一行两个整数n，Q表示串长和询问个数。

第二行一个长度为n的小写字母串。

接下来Q行，每行两个正整数l,r，保证$l \neq r$。

3 输出格式

输出Q行，每行一个整数表示答案。

4 数据范围

对于10%的数据，有$1 \leq n \leq 10$，$1 \leq Q \leq 10$。

对于15%的数据，有$1 \leq n \leq 100$，$1 \leq Q \leq 2$。

对于15%的数据，有$1 \leq n \leq 100$，$1 \leq Q \leq 100$。

对于10%的数据，有$1 \leq n \leq 3 * 10^3$，$1 \leq Q \leq 3 * 10^3$。

对于20%的数据，有$1 \leq n \leq 10^5$，$Q = 1$。

对于30%的数据，有$1 \leq n \leq 10^5$，$1 \leq Q \leq 10^5$。

以上数据组两两互不相交。

5 总体分析

5.1 命题契机

命题是OI竞赛中十分重要的一环，重要比赛的题目质量，直接影响到了这个比赛的公正性，所以探索命题的方法也是一个竞赛选手需要做的事情。

本题探索了一种新的命题思路：对于一个广为人知的传统算法，如果一个部分完全随机化会有什么样的结果。

有一次我做到了一道TopCoder上面的题，那道题是给出了一段归并排序的代码，然后比较函数是随机返回0和1，最后求一个排列出现的概率，那道题我探索了很多方法最后解决掉了，我觉得这是道好题。

于是我仿造着它，将我们竞赛中广为人知的算法随机化，例如后缀树，网络流，线段树，后缀数组等。最后我觉得后缀数组的随机化是最有趣的，也是最有思考的价值的。

在思考这个问题的初期，我试图根据后缀数组的lcp的计算方法去解决它，最后尝试了很多天，只能做到$O(nQ + nlogn)$的复杂度，这也是这道题的最初版本。

有一天，正当我准备写标程时，脑海里突然闪过了一个想法，用后缀数组解决时，是通过维护关于lcp值的笛卡尔树来做的，那如果直接用后缀树呢？

然后我发现，用后缀树的话，就不需要用到之前那个最为核心的结论，因为后缀树中多个串的lcp可以直接用LCA表示，即只需要枚举LCA就行了，且计算答案只需要用一些比较简单的补集转化，方法比之前的算法高明多了。当时我很兴奋，然后就把这道题出出来了，我觉得，一个如此简单大胆的将各种东西随机化的想法，最后需要很多经典算法结合起来解决，是相当优美的。

本题也启发了我们一种做题的思路，当用一种数据结构遭遇了瓶颈时，不妨试试类似相同的数据结构，可能会有优美的性质来解决题目

5.2 得分情况

集训队中大多数选手AC了此题

6 算法介绍

6.1 算法1

题目中提到，顺序数组是随机打乱的，我们可以$O(n!)$枚举顺序数组，然后找到l, r的位置，并两两计算最长公共前缀，时间复杂度$O(Qn!n^2)$。

可以注意到，我们在计算最长公共前缀上花了很多时间，我们可以预处理，对n^2组后缀预处理出最长公共前缀，于是复杂度可以优化到$O(Qn!n)$。

6.2 算法2

对于本题，我们有一个推论：设一个后缀$s[i : n]$的字典序大小排名是$rank[i]$，并且令$sa[rank[i]] = i$。

则设顺序数组中，$[l..r]$内字典序最大的后缀是y，最小的是x，则两两最长公共前缀的最小值是$s[x : n]$和$s[y : n]$的最长公共前缀。

下面我们来证明这个推论：

设$lcp(x, y)$表示$s[x : n]$和$s[y : n]$的最长公共前缀，设$h[i] = lcp(sa[i], sa[i + 1])$。

根据经典的后缀数组算法，我们有：

$$lcp(x, y) = lcp(y, x) \quad (rank[x] > rank[y])$$

$$lcp(x, y) = min(h[rank[x]], h[rank[x] + 1]...h[rank[y] - 1]) \quad (rank[x] < rank[y])$$

设顺序序列中的l到r中间的后缀按顺序分别是$a_1..a_m$。

对于每个满足$rank[x] \leq i < rank[y]$的i，只需要证明存在一个j，使得$h[i]$对$lcp(a_j, a_{j+1})$产生过贡献即可。

我们先找到$a_j = y$，那么设a_{j-1}和a_{j+1}中$rank$较小的一个是p，则$h[rank[p]]..h[rank[y]-1]$都出现过了，于是我们可以去掉y，归纳下去，最后所有满足上面条件的$h[i]$一定都出现过。

于是我们只要枚举最小后缀x，最大后缀y，然后计算随机序列中，l,r中字典序极值是x, y的方案数。

我们这里只讨论l, r, x, y互不相同的情况，有相同的情况类似。

方案数就是

$$\sum_{k=0}^{rank[y]-rank[x]-3} C_{rank[y]-rank[x]-3}^{k} * (k + 2)! * (n - k - 3)! * 2$$

于是我们预处理组合数和阶乘，枚举了x, y后暴力计算即可。

时间复杂度$O(Qn^3 + n^2)$。

6.3 算法3

可以发现，式子$\sum_{k=0}^{rank[y]-rank[x]-3} C_{rank[y]-rank[x]-3}^{k} * (k+2)! * (n-k-3)! * 2$的值只跟$rank[y] - rank[x] - 3$有关。

令

$$f[x] = \sum_{k=0}^{x} C_x^k * (k + 2)! * (n - k - 3)! * 2$$

我们可以用整数域下的快速傅里叶变换在$O(nlogn)$的时间内计算出$f[x]$。

具体步骤如下，令

$$A[x] = \frac{(k+2)!(n-k-3)!}{k!}$$

$$B[x] = \frac{2}{x!}$$

对A, B做卷积可以得到$c[x] = \frac{f[x]}{x!}$。

于是时间复杂度降到了$O(Qn^2 + nlogn)$。

6.4 算法4

之前的算法3中，我们计算答案的方法是累计$lcp(x, y) * f[rank[y] - rank[x] - 3]$，这样复杂度是$O(Qn^2)$的，我们可以发现，根据后缀数组的性质，不同的$lcp(x, y)$最多只有$O(n)$种。

那么现在我们不枚举x, y，改为枚举$lcp(x, y)$，具体就是枚举$lcp(x, y) = h[i]$，我们找出pl, pr，满足$h[i]$是$h[pl..pr]$中的最小值。

这个区间有贡献，当且仅当：

$$rank[x] \in [pl, pr + 1]$$

$$rank[y] \in [pl, pr + 1]$$

$$min(rank[x], rank[y]) \le i < max(rank[x], rank[y])$$

那我们的最小值的取值范围就是$[pl, min(rank[l], rank[r]))$，最大值的取值范围就是$(max(rank[l], rank[r]), pr + 1]$了。

相当于求

$$\sum_{i=1}^{A} \sum_{j=1}^{B} f[i + j + D]$$

其中D是一个常数。

我们设

$$f^2[x] = \sum_{i=1}^{x} f[i]$$

$$f^3[x] = \sum_{i=1}^{x} f^2[i]$$

$$\sum_{i=1}^{A}\sum_{j=1}^{B} f[i+j+D] = \sum_{i=1}^{A} f^2[i+B+D]-f^2[i+D] = f^3[A+B+D]-f^3[B+D]-f^3[A+D]+f^3[D]$$

于是我们可以通过预处理，$O(1)$计算出方案数。

时间复杂度$O(Qn)$。

6.5 算法5

之前我们所有算法都是建立在后缀数组这个算法的基础上的，众所周知，后缀数组是后缀树的简化版，我们可以在后缀树上考虑这个问题。

首先我们可以用经典的后缀自动机算法构造出后缀树。

设顺序序列中$l..r$中的后缀在后缀树上的结点是$a_1, a_2..a_m$，在后缀树上，两个串的最长公共前缀就是对应结点的最近公共祖先的深度。

为了方便表达，下面$LCA(l, r)$表示的是后缀l,r代表的结点的LCA。

而$LCA(a_i, a_{i+1})$中最浅的结点显然就是$LCA(a_1...a_m)$。

那么由于a中必定有后缀l, r对应的结点，我们可以枚举$LCA(l, r)$的祖先，然后计算贡献。

如何计算贡献呢，对于一个祖先x，设他的子树中有$size[x]$个后缀结点，设他的儿子y也是$LCA(l, r)$的祖先。

首先l, r对应的结点是必选的，但是为了使得x成为LCA，我们必须再多选一些点，这里我们可以用容斥思想，在子树x中随便选其他点，对于一个不合法的方案，他选的所有点必然在子树y中，减掉即可。

即方案数就是：

$$2 * \sum_{k=0}^{size[x]-2} C_{size[x]-2}^{k} k!(n-k-1)! - 2 * \sum_{k=0}^{size[y]-2} C_{size[y]-2}^{k} k!(n-k-1)!$$

设$g[x] = 2 * \sum_{k=0}^{x-2} C_{x-2}^{k} k!(n-k-1)!$，$g$我们可以用快速傅里叶变换在$O(nlogn)$内求出。

于是预处理好数组g，然后枚举每个祖先，根据子树里的后缀结点数量，直接计算即可。

时间复杂度$O(Qn + nlogn)$。

6.6 算法6

以上算法的瓶颈是每次要枚举所有祖先，对于每个结点去计算答案，在树退化成链时复杂度会很劣

如果我们可以做到一个结点的贡献只和他本身有关且独立，那我们只要链求和就行了

我们可以发现，一个贡献的组成是两部分，一个是父亲结点，一个是儿子结点。这个贡献可以映射到连接他们的那条边上

于是我么计算答案只需要求出LCA后求LCA到根的所有边的贡献之和即可

我们可以用$O(nlogn)$的时间预处理，之后询问时用倍增的方法求出LCA

于是问题就完美地解决了。

时间复杂度$O(Qlogn + nlogn)$

6.7 算法7

上述算法的瓶颈在于需要用快速傅里叶变换计算系数，众所周知快速傅里叶变换的常数是比较差的

我们把问题一般化，相当于$1..n$的点有三种颜色$0, 1, 2$，其中2类点只有2个，0类点x个，1类点$n - x - 2$个，求满足2类点中间都是0类的排列的方案数

我们可以先将1类点和2类点排列，两个2类点必须相邻，所以捆绑成一个，排列个数是$2 * (n - x - 1)!$种

之后考虑插入0类点，这时无论如何插入，2类点中间一定只有0类点了。

于是先给那$n - x$个点选定位置，方案数是$C(n, n - x)$，然后乘x个点的排列个数$x!$

这个式子毛啸同学给了我一个简单的理解方法：考虑1类点和2类点的子排列，2类点只要相邻即可，也就是说第一个2类点随便放，第二个则是固定的位置，所以是$\frac{2*n!}{n-x+1}$

于是我们可以用$Ukkonen$算法构造后缀树，LCA也可以用$O(1)RMQ$实现，总的复杂度是$O(n+Q)$

7 感谢

感谢计算机协会提供学习和交流的平台。

感谢绍兴一中的陈合力老师，董烨华老师多年来给予的关心和指导。

感谢国家集训队教练余林韵和陈许旻的指导。

感谢清华大学的张恒捷，王鉴浩，贾越凯学长和绍兴一中的任之洲同学对我的帮助。

感谢绍兴一中的孙奕灿同学为本文审稿。

感谢绍兴一中的同学们和杭州外国语学校的陈青云同学为本题验题

感谢雅礼中学的毛啸同学与我交流本题的其他解法

参考文献

[1] 刘汝佳, 黄亮, 《算法艺术与信息学竞赛》, 清华大学出版社。

[2] 刘汝佳, 《算法竞赛入门经典》, 清华大学出版社。

[3] 王鉴浩,《浅谈字符串匹配的几种方法》,2015年信息学奥林匹克中国国家队论文集

[4] 张天扬,《后缀自动机及其应用》,2015年信息学奥林匹克中国国家队论文集

消消看 命题报告

浙江省余姚中学 张浩威

1 提交答案题的特点

近年来，提交答案题已经逐步进入群众们的视野，大到CTSC，小到ZJOI。这一类题目的变化繁多，与其它类型题目最大的差别就是下放了输入数据，或者题目仅仅是要求选手构造一个符合条件的解，例如**UR7水题出题人**。

对于出题人而言，提交答案题比起传统题较为明显的优势是区分度大，出题人可以根据选手输出答案的优劣，设置多档分数。

在做题的过程中，由于独特的评测方式，选手不需要上传程序，通过人类智慧可能会起到意想不到的效果。

消消看是一道数据分析类的提交答案题，考察的不仅是选手的代码能力与算法能力，选手的数据分析能力也显得尤为重要。

本文就**消消看**一题来深入读者对数据分析类提交答案题的理解。

2 试题分析

2.1 题目大意

给定一个n*m的含有不同颜色的方块的图形，每次可以消除一个大小大于1的颜色相同的四联通块，消除完后可能会导致上面的方块掉下来，也有可能使得右边的方块向左移动。每次消除的方块的个数和最后剩下的方块的个数都会获得相应的分数，可以随时结束游戏，目标使得分数尽可能多。

2.2 初步分析

我们注意到如果数据没有任何性质是纯随机的，那么这题状态之多显然是不可能有多项式做法的。因此我们有理由相信，每个测试点都有其特点，根据这些特点我们能得到一个不错的解法。

然而除了第一个点之外，其余点数据量都非常大，无法通过直接观察来得到其特点，因此我们可以考虑抽样调查来得到一些有用的信息。

2.3 数据分析的方法

对于输入数据中有矩形的提交答案题，有以下几种常见的方法。

1、观察每一行。

2、观察每一列。

3、观察子矩形。

就本题而言，还会存在颜色，且每次消除只有可能消除颜色相同的联通块，还存在分数的概念，因此还有以下方法。

4、观察颜色相同的方块所在位置。

5、观察每个颜色相同的联通块的特征。

6、观察消除与剩余方块时得到的分数的大致规律。

通过这些方法，我们可以逐一判断出每个测试点的特点。

2.4 深入分析

首先由于题目并没有给出测试自己输出是否正确的checker，因此我们得先写一个checker来模拟每次消除方块时方块的变动。

2.4.1 checker的写法

对于列与列之间，维护一个双向链表，用来处理某一列方块都被消光的情况。

对于每一列的行与行之间，维护一个双向链表，用来处理方块掉落的情况。

同时要记录每一列的最后一块在第几行。

那么如果要找到一个第x行y列的方块，即相当于是从左下角开始，沿着链走y步，再向上走n-x步。

对于每一个方块的上下方块，能在O(1)时间内找到，对于左右方块，需向下走到底，再向左或右，再向上走若干步到达。

由于每个方块最多会被消一次，因此这个做法的时间复杂度是n^2m的。

当然也可以用平衡树优化，复杂度可以降为$nlog_2nm$，但是在考场上不推荐写。

2.4.2 测试点1,2,3,4,5

这5个测试点有一个共同点就是消除任意多的方块都得不到分数，而剩下的方块个数越少，分数越高。

2.4.3 测试点6,8,9

这3个测试点的共同点是消除方块与得到的分数按平方级别增长，因此将颜色相同的方块一起消掉总是最优的。

2.4.4 测试点1

通过查看数据就能发现这仅仅是一个5*5的矩阵，通过人类智慧手玩就能在几分钟内玩出剩下0个方块的解。

该测试点旨在给选手检验自己checker是否正确，以及送分。

2.4.5 测试点2

通过观察数据的左上角10*10的矩阵，就能发现每个联通块的大小最大为2。只需再写一个代码来验证这个猜想，就能发现除了最底下有一个非常大的底座外，其余都是大小为2的联通块，且每种颜色只有2块。那么我们只需要从上到下，从左到右地去消除方块，使得没有一个方块掉落，没有一个方块向左移动，就能消完所有方块。

该测试点考察的是选手对于数据的初步分析能力，可以看出这个测试点的性质是比较显然的。

2.4.6 测试点3

这个测试点较难观察出规律。实际上看左上角10*10的矩阵就能发现1至7列除了第5列外都是轴对称的。那么可以猜想出，第5列可能仅仅是用来干扰用的。

在进一步的观察中，我们可以得知整个矩阵有20 多列的干扰列。只要我们将这些列消除，整个矩阵就能分为好几块，其中每一块都是轴对称的。

根据轴对称这个性质，只要我们从中间往外面消除，每块都能被消光，只需模拟这个过程就可以了。

该测试点在设计时考虑到消除方块没有分数的点中也应当有考察数据分析能力的点。而该测试点的难点就在于数据分析和结论猜测，选手在知道该测试点性质的前提下还需写一个模拟器来达到输出方案的效果。

2.4.7 测试点4,5

这两个测试点很容易得就能看出仅仅只有3或4种可能的颜色。因此这两个测试点不需要选手高超的数据分析能力，考察的是选手的暴力和乱搞水平。

针对第4个测试点，我们可以每次随机消除一个联通块，只需多随机几次最终就能消除完。

对于第5个测试点，我们发现这样消除完的几率是非常低的。因此可以设置一个参数x，当剩下的方块不超过x个时，暴力枚举所有消除方案，再通过一定的剪枝就能跑出答案。

2.4.8 测试点6

观察除了1以外的每一种颜色，我们能发现每种颜色都是在同一行或者同一列中，其中可能存在几个是消失了的。这很容易就能联想到是被其它颜色覆盖了。

当然我们也可以观察每一行和每一列，能发现存在1种颜色出现多次，存在多种颜色出现1次，这也能得到上述结论。

因此我们只需要枚举若干遍，每次找到能完全覆盖一行或是一列的颜色，消除它即可。

该测试点在设计时主要是让选手适应分数从无到有的变化。

2.4.9 测试点7

这个点是一个一行m列的矩阵。注意到所有颜色几乎都是相邻的，但是中间可能会存在杂碎点，然而这些杂碎点仅仅只有一个是不可能合并的。因此同

一种颜色若不在同一个联通块中是可以默认为两种颜色的。

我们发现最后剩下方块也能得到不错的分数，这样就是一个01背包的经典问题了。

每一种颜色的方块个数可以看作体积，消除后的分数可以看作价值，就可以进行dp了。

该测试点通过消消看的模型考察了经典问题，性质较为显然，旨在考察选手对于经典问题的熟悉程度。

2.4.10 测试点8

这个点的特征较难观察出。容易观察到的是每种颜色都处于同一个联通块中。但是仅仅观察到这个并不能说明什么问题。如果我们把每种颜色所在位置的坐标输出出来，就可以看到每种颜色仅有可能是“L”型或是“一”型或是单点。

如果我们看出了这个性质，这个点就非常方便了。我们无视那些单点，因为最终这些点肯定是消不掉的。然后我们选择一个消除后不会使得联通块分离的方块群，可以证明一定存在这样的方块群。

这样我们就能将所有联通块都删光得到最高的分数。（在这个点中最高分仍是负分）

该测试点比起前面7个测试点难度上升了一个级别，旨在区分高超的提交答案题选手与业余选手。

2.4.11 测试点9

这个点的性质比较容易观察，观察每一列，发现颜色的编号都是连续的，而对于不同的两列，不会出现颜色相同的方块。因此很容易就能发现每一列都互不影响，且最后剩下x个方块就会失去x+1分。

这样就能将一个n*m的方块浓缩成m个1*n方块的子问题，就能够进行dp了。

令$dp_{i,j}$表示消完i到j这些方块后能得到的最大分数，若消不完则为负无穷大。

令f_i表示仅有1到i这些方块时的最大得分，那么一个显然的转移方程是f_i=max$\{f_{i-1}+1, f_{j-1}+dp_{j,i}\}$

那么我们怎么求出dp的值呢。

我们可以将它分为两部分即$dp_{i,j} = max\{dp_{i,k} + dp_{k+1,j}\}$

当然也有可能是方块i与方块j的颜色相同，我们先消完i至j中的一部分，使得剩下的方块颜色均与i和j相同，那么这种颜色就聚集到了一起，可以一起消除，我们令方块i的颜色为c。

这里我们要再设一个g数组，其中$g_{i,j}$表示当前已枚举到第i个方块，最后聚集到一起的有j个颜色c的方块的最大分数，且方块i在j个之中。

当且仅当方块i的颜色与c相同时有转移$g_{i,j} = max\{g_{k,j-1} + dp_{k+1,i-1}\}$

这样我们就能通过g数组来得到dp数组了。

最坏时间复杂度为n^5m，实际上远远不会到这个复杂度，而且提交答案题只要在考试时间内跑出来就可以啦!

该测试点考察的是选手的dp功底与代码能力，性质较为显然，不过这个dp是略高于NOIP难度的。

2.4.12 测试点10

该测试点给了一个既宽又短的矩阵，如果我们将所有颜色所在的位置都打印出来，就能发现每个颜色都构成了一个空心矩阵。

那么实际上这形成了一棵树，对于每个矩形都看作一个点，若矩形i能包含矩形j，则i是j的祖先。构树的过程非常简单不再阐述。

在消除方块的得分中我们能发现除了一个得分是23333333其余都是0。这意味着我们要尽可能地去消这么一个大小的块。

观察这些矩形的特征，对于在树中的任意两个点i,j，若i不是j的祖先，那么消除j这个矩形都不会导致矩形i被破坏。因此我们在消除的过程中，只要从根开始消除，那么消除一个矩形相当于放出若干1号颜色。

这样就转换成了一棵带点权的树，找一个包含根的联通块使得点权和等于给定的值，那么这实际上就是一个树形dp啦。

该测试点考察的是选手的综合水平，其中需要有数据分析能力看出该测试点的性质，根据经典问题的熟悉程度来看出转化后的模型，以及高超的代码能力来输出方案。不得不说在考场中通过该测试点的选手对于提交答案题的理解已经十分深刻了。

2.4.13 乱搞

事实上当选手没有时间能够逐一分析每个测试点性质的时候，考虑到本题具有10个档次的分数，可以写一个通用做法来得到大量分数。

具体地，由于本题所有测试点都没有消除方块减掉分数这种情况，且除了7号与10号点外剩下的方块越少越好，因此我们可以采取能消就消的原则，每次随机消除一个联通块。

只要我们模拟这个过程，就能得到50左右的分数了，这需要选手在平时做题与考试中的经验积累。

3 得分估计

对于不同的点思考时间与码题时间略有不同，得分大致与做题时间成正比。

最高分应当在80至90之间，平均分在60左右。

4 集训队互测得分情况

得分在18至69不等，平均分为45分，略低于预期估分。

5 总结

数据分析类提交答案题经常会出现几种不同的情况。就本题而言，存在“消除方块有分数”与“消除方块没有分数”两种情况，根据不同的情况出题人往往会设置容易点与难点，如果选手能够把握好这一点，而不死磕前面的测试点(例如本题中的测试点3)，就能脱颖而出。

事实上本题的观察数据的方法仍适用于NOI2014消除游戏，根据不同的数据分析类提交答案题，观察数据的方法略有不同，但本质上可以用一句话来概括：**大胆猜测，小心论证**。

6 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢所有辅导过我的老师与帮助过我的同学。

感谢父母对我的关心与照顾。

《strakf》命题报告

佛山石中李子豪

摘 要

本文主要写了两道字符串与数据结构结合的题目，以及KD-tree的一些调参。

1 试题来源

2016国家集训队互测

2 试题大意

给了一个长度为N的仅由小写字母组成的初始字符串S.

之后有M次操作，操作分为三类：

1 A:将原串S的所有等于A的子串权值+1；

2 l r:将原串S的所有等于$S[l,r]$的子串权值+1；

3 a b:询问原串S所有子串$S[l,r]$满足$a \le l \le r \le b$的权值和。

3 数据范围

数据点编号	N, M	操作类型	数据特点
1	5	2,3	无
2	10	2,3	无
3	100	2,3	无
4	$2*10^5$	1,2	无
5	$2*10^4$	1,3	所有的3操作a=1

6	$2*10^4$	1,3	所有的3操作b=N
7-8	$2*10^4$	1,3	所有的1操作的字符串长度为不下降序列
9-10	$6*10^3$	2,3	无
11-12	$2*10^4$	2,3	所有的3操作b=N
13-15	$2*10^4$	2,3	所有的2操作的字符串长度为不下降序列
16-20	$2*10^4$	2,3	无

输入文件大小不超过0.3M。

4 试题考点

字符串、数据结构、分块

5 想法来源

5.1 一道题目

给了一个长度为N的仅有小写字母组成的字符串S.

定义$S[l][r]$为S从l到r的子串。

有q次询问，每次对一个区间$[l_i,r_i]$进行询问，询问最大的x满足$S[l][l+x-1]=S[r-x+1][r]$.

$N,q\le 2*10^5$

5.2 题解

这道题，假设对于一个询问区间$[l,r]$,那么我们实际上就是求一个最小的x满足以l开始的后缀以及与x开始的后缀的最长公共前缀$>r-x$.

因此，我们可以从小到大逐一枚举x,然后离线按顺序处理所有的询问。

我们对询问以左端点进行排序，然后若当前枚举到i,那么则把左端点$<i$的询问插入，然后就删去所有满足答案可以为i的询问。那么，就可以解决了。

因此，我们现在的问题就是要去维护一个可以求出满足答案可以为i的询问。

而根据上面所说，我们有这么一条式子：$i + len - 1 \geq r, len$是最长公共前缀，即$i \geq r - len + 1$.

因此，我们只需要有办法维护$r - len + 1$的值，并从小到大取出即可。

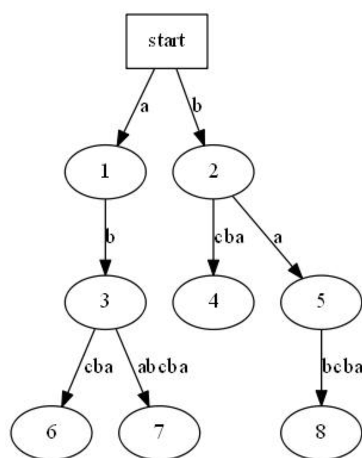
对于这个问题，我们可以先进行后缀数组并求出$height$数组。

然后，根据$height$，我们可以构造出一棵树。大概就是这样：

假设字符串为$ababcba$,那么$height$数组为：

0 a
1 ababcba
2 abcba
0 ba
2 babcba
1 bcba

然后构造的树为：



感受一下大概就懂了。

之后，我们对于一个询问$[l, r]$的插入，则先找出l所对应的节点，然后从这个点不断往根，对于路径上每个节点插入一个pair为$(r - len, num)$,len为这个节点能表示的最长串。

然后，对于处理x为i的询问，则从x对应的节点，不断往根走，对于路径上每个节点询问比i小的所有pair，然后更新对应答案，删除对应询问即可。

然后，这个问题，我们可以用树链剖分和dfs序来解决。

我们大概可以分为三类询问：

1.询问子树的最小值；

2.询问重链上节点的最小值；

3.询问重链上节点的非重儿子的所有儿子的最小值。

然后，对于这三类问题，则可以分别通过整棵树的dfs序、轻重链、轻儿子dfs序来解决。

因此，就可以通过$O((N+M)log_2^2N)$解决了。

5.3 小结

这一题比较好的结合了字符串和数据结构的问题。比较考验选手代码能力。

6 算法介绍

6.1 第1-3个数据点

采取$bruteforce$的方法。

对于每一个修改操作，则把所有符合的子串的权值+1.

对于每一个询问操作，则直接询问范围内的所有子串的权值和。

时间复杂度为$O(n^3)$,预计得分15分。

6.2 第4个数据点

注意到这个数据点并没有操作3，因此直接结束即可。

算上前面的15分，预计得分20分。

6.3 第5-8个数据点

这一类数据点的特点在于只有操作1的修改，而没有操作2的修改。

6.3.1 数据特点

由于输入保证不超过0.3M,可以得到插入总长度不超过$3*10^5$。

因此，我们可以得到一个结论：插入字符串的长度最多只有$O(\sqrt{N})$种。

证明：

假设长度恰好为1到$\sqrt{N}$各一个，那么总长度已达到$O(N)$级别。

证毕。

6.3.2 解决方法

因此，我们对于询问操作，可以对于各个不同长度的询问分开处理。

那么，问题变成要维护各个不同长度的左端点在某一区间范围的方案数。

而对于插入操作，我们可以得出：插入操作所能影响的左端点的后缀排名处于某一个区间范围内。

因此，我们可以对于每一个左端点记录一个二元组(a,b)表示左端点位置在a，对应后缀排名为b。

然后插入则是对于所有$b \in [l, r]$的二元组进行权值+1.询问则是对于所有$a \in [l, r]$的二元组询问权值和。

我们可以采取分块来解决这个问题。

假设我们按照位置序列来分块，块长为L.

那么我们可以对于每一块记录块内权值和以及按照b排序的块内端点排名从而记录对于每一个给定的r得到的最大的k满足$b_k \leq r$.

然后，再对前i块记录权值和sum_i.

对于修改操作，我们直接求出对于每一块的贡献以及对于前i块的贡献值，修改对应的块内权值和以及sum_i.并且对于每一块记录一个修改标记。

对于询问操作，则先利用求好的sum_i求出整块的权值和，然后对于非整块部分，则直接打算标记，暴力询问即可。

分析复杂度，修改操作复杂度为$O(\frac{N}{L})$,询问操作复杂度为$O(L\sqrt{N})$. 当$L = N^{0.25}$时，有最小复杂度为$O(N^{0.75})$.

因此，整个算法复杂度为$O(N^{1.75})$.预计得分20分。

算上上面其余部分的分数，预计得分40分。

6.4 第9-10个数据点

对于这一类数据，同样的，我们可以先构造一个后缀数组。

然后，我们对于每一个位置维护一个结构，维护以该点为左端点的所有修改长度。

对于修改操作，我们可以通过二分求出包含这一子串的后缀排名区间。我们可以对于排名区间里面对应的每个位置，在其对应的结构里面插入当前的修改长度。

然后，对于询问操作，则对操作区间内的所有点分别询问其结构里面长度不超过某一个值的修改个数。

因此，我们需要一个可以维护动态插入以及询问比某个值小的值的个数，可以使用线段树或者平衡树，也可以采用pb_ds库里面的黑科技。

时间复杂度为$O(NMlog_2N)$.预计得分10分。

算上上面其余部分的分数，预计得分50分。

6.5 第11-12个数据点

这一类数据的特点在于右端点固定为N，即不存在右端点越界的情况。因此，我们不需要分开不同长度来处理。

因此，我们可以采取类似第5-8个数据点的解决方法：

对于每一个左端点记录一个二元组(a,b)表示左端点位置在a，对应后缀排名为b。

然后插入则是对于所有$b \in [l,r]$的二元组进行权值+1.询问则是对于所有$a \in [l,r]$的二元组询问权值和。

我们上面原本使用的是分块，但其实这里也是可以采用KD-tree来解决的。使用KD-tree可以减少一定的代码量以及运行速度上也会有一定的提升。

可以证明，对于坐标两两不相同的情况，KD-tree可以保证最坏$O(\sqrt{N})$的复杂度完成二维区间内的问题。

这一点，我们可以分两步来证明：

6.5.1 对于其中一维区间为全集的情况

首先，比较容易可以知道另一维区间为任意区间[L,r]实际上可以等价于[1,r]区间的解决。

这个可以简单证明得到:假设线段树中mid为使得L,r处于两个不同区间的节点，那么下一步问题变成了等价于[L,n]以及[1,r]的问题。

然后，对于[1,r]的情况，如果当前是以全集区间的那维分两边，那么显然两边节点都要遍历，而如果是以[1,r]区间的那维的话，那么必然有一边要么完全不覆盖，要么完全覆盖，都可以一步解决，因此，实际需要往下的只有一个节点。

然后，总层数为log_2N层，然后只有$\frac{log_2N}{2}$层需要遍历两边，因此复杂度为$O(2^{\frac{log_2N}{2}}) = O(\sqrt{N})$.

6.5.2 一般情况

首先，同样容易证明其中一维的任意区间[L,r]可以等价于[1,r]。

然后当当前是以这一维分两边的话，那么必然有一边这一维要么完全不覆盖，要么完全覆盖。最坏情况是完全覆盖那一种，那么这一部分我们已经证明是$O(\sqrt{N})$级别的了。而另一边，则是等价的问题。

因此，我们有$F(N) = F(N/2) + O(\sqrt{N})$,根据等比数列我们可以得到$F(N) = O(\sqrt{N})$.

证毕。

于是，我们可以通过$O(M\sqrt{N})$的复杂度解决。预计得分10分。

算上上面其余部分的分数，预计得分60分。

6.6 第13-15个数据点

这一类数据点，保证了插入长度不下降。这个提示我们可以往长度方向想。

考虑当前询问区间为[L,r]，我们设[1,L-1]为A区间，[L,r]为B区间，[r+1,N]为C区间。然后设(x,y)意义为左端点在x区间，右端点在y区间的权值和。例如(A,B)表示左端点在A区间，右端点在B区间。我们求解的是(B,B)的值。

那么，对于[1,r]区间的询问，我们得出了(A,A)+(A,B)+(B,B)。那么，如果我们有办法求解(A,A)+(A,B)就可以解决了。因为我们已经知道其中一个端点在边界的解法了。

然后我们可以求[1,N]区间的询问，从而得出(A,A)+(A,B)+(A,C)+(B,B)+(B,C)+(C,C).

然后又由(B,B)+(B,C)+(C,C)为[L,N]区间的询问，可以得到。

因此，现在唯一的问题就是(A,C)问题的求解了。但这一类问题似乎没有什么好的解决方法。

这时，我们回过来看长度的提示。然后我们可以得到一个突破点：如果保证长度不超过$r - l + 1$的话，那么不存在(A,C)这个问题。

因此，我们可以根据长度来维护一个可持久化KD-tree，维护只包含长度不超过某个值的答案。

那么，对于询问操作，我们只需要先二分出对应最大长度对应的KD-tree，然后在这棵KD-tree进行询问即可。

时间复杂度为$O(M\sqrt(N))$,预计得分15分。

算上上面其余部分的分数，预计得分75分。

6.7 第16-20个数据点

最后这一类数据点，实际上只需要对第13-15个数据点的解决方法进行少量修改即可。

我们观察到这一题允许离线，并且操作独立，因此，我们只需要在外面套上一个cdq分治，然后就能保证插入长度单调，从而就能直接利用第13-15个数据点的解决方法解决了。

由于多套了个cdq分治，时间复杂度为$O(M\sqrt(N)log_2M)$.

进行少量修改，可以通过全部数据。预计得分100分。

7 对于KD-tree的调参

最后，扯一些别的。

对于前面的第5-8个数据点的分块方法，实际上也可以用KD-tree来解决。

现在问题就是对于某一维询问区间、另一维询问全集的情况，可以根据不同的情况得到不同的复杂度。

其实，这个似乎很好搞，假设我们要求第一维为区间的复杂度为N^a,第二维为区间的复杂度为$N^(1 - a)$的话。

那么，我们只需要选择以第二维为分离标准的层数设为$alog_2N$即可。

但是，要注意安排层的顺序，不然有可能复杂度会多乘了一个$O(log_2N)$。

而一种比较好的安排方法，就是假设$a < 1 - a$,那么先把多出来的$(1 - 2a)log_2N$层先进行分离，之后剩余层数正常的轮流分即可。

用KD-tree来调参的话，相较于分块，应该可以减少一定的空间和常数。

8 总结

总的来说，这道题难度与NOI第二题接近。

主要考察对题目性质的挖掘以及转化为简单问题的解决方法。旨在将字符串类型题目和数据结构类型的题目联系起来。

9 感谢

感谢中国计算机学会提供学习和交流的机会。

感谢佛山石中的江涛老师、梁冠健老师多年来给予的关怀与指导。

感谢国家集训队教练余林韵和陈许旻的指导。

感谢佛山石中的龙耀为、麦景、杨嘉宏等同学对我的帮助和启发。

感谢绍兴一中王鉴浩学长提供的模板。

感谢其他对我有过帮助和启发的老师和同学。

感谢父母对我的关心和照顾。

参考文献

[1] 于纪平：《C++的的pb_ds库在OI中的应用》

[2] 任之洲：《k-d tree在传统OI数据结构题中的应用》

[3] 罗穗骞《后缀数组——处理字符串的有力工具》

[4] 陈立杰《后缀自动机》

[5] 罗剑桥《浅谈分块思想在一类数据处理问题中的应用》

[6] 陈丹琦《从《Cash》谈一类分治算法的应用》

[7] 刘汝佳, 黄亮, 《算法艺术与信息学竞赛》, 清华大学出版社。

[8] 刘汝佳, 《算法竞赛入门经典》, 清华大学出版社。

《过去的集合》命题报告

厦门双十中学 汪文潇

摘 要

不相交集合的合并及查询问题是初学者很早就会遇到的问题之一，这不仅是因为它形式简单，更因为其具有多种不同的解法。其中，用连通性表示是否在同一集合、维护森林的并查集做法和对应的路径压缩、按秩合并的优化更是简洁而极具效率。

这道题是对不相交集合的合并及查询问题的一个简单的扩展，而本文则将脱离试题本身，对该问题的各种解法进行讨论，同时整理出自己命题时的思路和不同解法的一些特点，希望能对其他人有所启发，能对这类问题乃至其他问题有更好的理解。

1 问题大意

有n个元素，最初每个元素属于单独的一个集合，有m次操作，每次操作都是以下两种之一：

1、将两个集合进行合并

2、询问在过去的某个时间点，某两个元素是否在同一个集合

注：这里的问题描述与《过去的集合》这题本身题意略有不同。

2 主要算法介绍

以下将“不相交集合的合并及查询问题”称为原问题。

扩展后的问题与原问题的不同在于其要求部分可持久化，即在原问题基础上要求支持询问历史版本的信息。这里将先考虑原问题的一些解法，再尝试将其应用到要求部分可持久化的情况。对于原问题，一般来说大致有两种类型的方法来实现一个并查集的结构，即标号法和维护森林法。

2.1 标号法

标号法即给每个元素一个标号，用相同的标号表示同一个集合内的元素。

朴素实现的标号法单次询问的复杂度是$O(1)$的，但是合并的总复杂度最坏可以达到$O(n^2)$。

2.1.1 启发式合并

注意到合并两个集合时，我们可以选择将其中一个集合内所有元素的标号都更改为另一个集合的标号，此时时间复杂度与集合大小挂钩。那么一个显然的优化即是每次选择集合大小较小的集合进行修改标号。

此时询问的复杂度仍然是$O(1)$的，而合并的总复杂度则不超过$O(n\log n)$：考虑一个元素的标号每次被修改，其所在集合的大小至少是原来的两倍，而任何一个集合的大小都不会超过元素总数n，因此任意一个元素的标号最多被修改$O(\log n)$次，所以合并的总复杂度不超过$O(n\log n)$。

部分可持久化

注意到，整个做法是很容易使用数组实现的，因此一个方法就是直接在可持久化数组上套用该做法。根据可持久化数组的不同实现，复杂度略有不同。

但是实际上这个问题并不要求完全可持久化，相应的，只需要一个部分可持久化的数组即可。这可以通过对每个位置开一个表记录下每次对其进行的修改来实现。这样实现的部分可持久化数组对某个指定版本进行访问的时间复杂度是$O(\log n)$，对当前版本进行访问时间复杂度是$O(1)$的。

使用了上述结构后，可以做到单次询问复杂度为$O(\log n)$，合并的总复杂度为$O(n\log n)$。

2.2 维护森林法

维护森林法即是通过两个元素之间的连通性来表示它们是否属于一个集合，用森林的结构来存储连通性。要存储一个森林的结构，一个直接的方法就是存储每个节点的父节点。这也是维护森林法的主要形式。

要判断两个元素是否在同一个集合，只需要判断所在树的根是否相同即可。

这样单次询问的复杂度最坏是$O(n)$的，合并的总复杂度最坏可以达到$O(n^2)$。

2.2.1 路径压缩

路径压缩优化即在每次询问某个点所在树的根节点后，暴力忘上走找到根，并将其到根路径上所有点直接变成根的子节点。使用该优化后，每次操作的复杂度是均摊$O(\log n)$的。

由于其均摊性质的复杂度，这个做法很难扩展到部分可持久化的情况。

2.2.2 按秩合并

按秩合并即给每棵树一个秩，只有一个点的树秩为0：当合并两个树时，如果两棵树的秩不同，将秩较小的树的根作为另一棵树的根的子节点，新树的秩设为原来两棵树秩的较大值；如果两棵树的秩相同，任意将一棵的根作为另一棵树的根的子节点，新树的秩设为原来两棵树的秩加一。

不难发现，秩其实等价于一棵树内离根最远的点到根的距离。有归纳法可以得出，一棵秩为i的树，至少要有2^i个节点，因此秩是$O(\log n)$。此时每次操作的复杂度是最坏$O(\log n)$。

部分可持久化

这个做法显然也是可以扩展到部分可持久化情况下的。

一种做法是继续使用可持久化或部分可持久化的数组，但是此时我们有更好的做法。

观察森林的结构,结合修改时的方法，我们可以发现一些性质。

性质1：过去某个时刻的森林的边集一定是当前边集的子集。证明：操作过程中边不会被删除。

性质2：设当前存在的每一条边的权值为其出现的时间点，那么从任意一个点到根的路径上，边权都是单调增的。证明：每次合并两棵树时，加的边一定连接两棵树的根节点。

由此，可以直接记录每条边出现的时间点，询问时在被询问点到根的路径上暴力枚举找到出现时间点早于询问时间点的边中离根最近的那条即可。

此时，合并和询问的复杂度都是最坏$O(\log n)$的。

可以注意到，这个算法在询问集合时是相当暴力的，这也带来了优化的空间。

2.2.3 结合上述两种优化

结合上述两种优化后，复杂度是均摊$O(\alpha(n))$，单次操作最坏复杂度仍然是$O(\log n)$的。

部分可持久化

这个算法在推广到部分可持久化的情况下与只使用按秩合并相比并没有优势，相反路径压缩反而增加了实现难度，并没有太大的意义。

2.2.4 倍增法

考虑在询问时通过倍增的方法解决。

那么现在要解决的就是如何维护倍增所需的信息。

对于每个点x，每个i，记录向根走2^i步所到节点$f_{x,i}$。每次合并操作，实际上是给某个$f_{x,0}$赋值。当某个$f_{x,i}$被确定后，其值便不会再被修改，同时，对于所有满足$f_{y,i} = x$的y，$f_{y,i+1}$也将被确定。对于每对(x, i)，用链表将满足$f_{y,i} = x$的y记录下来，当$f_{x,i}$确定后，就可以根据这个链表更新出所有新确定的$f_{x,i}$。

通过$f_{x,i}$，在询问时可以很容易倍增找到当前的根。

显然对于任意一个x，只有不小于0且不超过$O(\log n)$的i是有意义的，因此有效的位置不超过$O(n \log n)$，又由于每个有效位置最多被赋值一次，合并的总复杂度也不超过$O(n \log n)$。

该算法单次询问的复杂度最坏是$O(\log n)$的，合并的总复杂度为$O(n \log n)$。

部分可持久化

利用点到根的路径上边权单调的性质，可以很容易地将这个算法部分地可持久化。

对于每个点x，每个i，额外记录x与$f_{x,i}$连通的最早时间点$t_{x,i}$。$t_{x,i}$可以在维护$f_{x,i}$时很容易地得到。

询问时利用倍增和点到根的路径上边权的单调性就可以在$O(\log n)$的复杂度内解决。

部分可持久化后，该算法单次询问的复杂度仍然是最坏$O(\log n)$，合并的总复杂度也仍然为$O(n\log n)$。

2.2.5 按秩合并倍增法

注意到按秩合并利用了性质“合并时可以自由选择新树的根”，而倍增法则单纯地使用其他结构来加速了询问过程，这两个算法对朴素算法优化的部分互不相关，显然是可以结合使用的。

同时使用这两个算法，单次询问复杂度为最坏$O(\log\log n)$，合并的总复杂度为$O(n\log\log n)$。

部分可持久化

这两个算法在扩展到部分可持久化情况时，使用的方法也是一致的，都是通过记录、判断森林中边出现的时间，仍然可以结合使用。

此时，我们就得到了一个单次询问复杂度为最坏$O(\log\log n)$，合并的总复杂度为$O(n\log\log n)$的部分可持久化并查集的算法。

3 致谢

感谢我的父母、家人。

感谢曾艺卿老师。

感谢我的同学们。

感谢中国计算机学会。

火车司机出秦川 命题报告

安徽师范大学附属中学 吴作凡

1 试题

1.1 问题描述

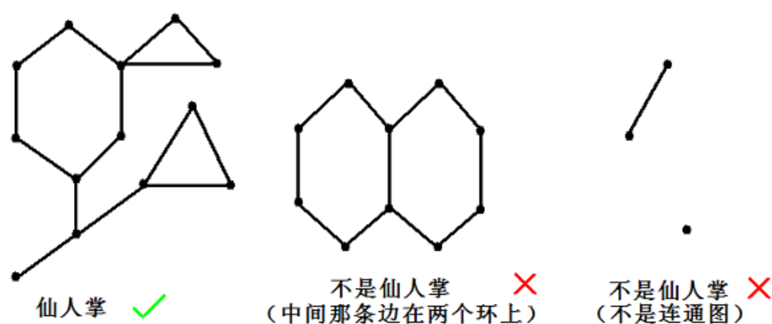
有一个n个点m条边的带边权的仙人掌（点和边的编号都从1开始），有q组操作，每组操作由两个部分构成：

1. 给定k_i条路径，第j条是从s_j到t_j的路径，有一些是经过点最多的简单路径，有一些是经过点最少的简单路径，求出至少被一条路径经过的边的边权和。

2. 在询问结束以后，将第w_i条路径权值修改为x_i。

如果一个无向连通图的任意一条边最多属于一个简单环，我们就称之为仙人掌。所谓简单环即不经过重复的结点的环。

为了让经过点最多或最少的简单路径唯一，本题数据中的仙人掌的每一个环的长度都是奇数。没有自环。



1.2 输入格式

第一行三个整数n,m和q，表示有n个点m条边q组操作。

接下来m行每行三个整数u,v,w，表示一条连接u和v的边权为w的边。

接下来$2q$行，每两行表示一组操作。

每组操作的第一行表示询问，首先是一个整数k_i表示有k_i条路径（k_i可能为0，这时候请忽略这个询问并输出0），接下来会描述每条路径，第j条路径的描述由$s_j,t_j,type_j$三个整数组成，分别表示起点和终点还有路径类型，如果$type_j$是0表示经过点最少的简单路径，为1表示经过点最多的简单路径。路径起点和终点不会相同。

第二行两个整数w_i和x_i，表示第w_i条边的边权会改变为x_i，如果w_i为0则表示没有修改。

1.3 输出格式

输出q行，每行一个整数，表示每次询问的答案。

1.4 数据范围和约定

保证任意时刻边长和都属于int范围内。边长均为正整数。

令$S = \sum_{i=1}^{q} k_i$，数据范围如下表所示：

测试点编号	n	m	q	S	有无修改
1	=2	$n-1 \le m \le 2n-2$	≤ 10	≤ 10	有
2	=100	$n-1 \le m \le 2n-2$	≤ 100	≤ 100	有
3	=100000	$m = n-1$	≤ 100000	≤ 100000	无
4	=100000	$m = n-1$	≤ 100000	≤ 100000	有
5	=300000	$m = n-1$	≤ 300000	≤ 300000	无
6	=300000	$m = n-1$	≤ 300000	≤ 300000	有
7	=100000	$n-1 \le m \le 2n-2$	≤ 100000	≤ 100000	无
8	=100000	$n-1 \le m \le 2n-2$	≤ 100000	≤ 100000	有
9	=300000	$n-1 \le m \le 2n-2$	≤ 300000	≤ 300000	无
10	=300000	$n-1 \le m \le 2n-2$	≤ 300000	≤ 300000	有

2 算法介绍

第1、3、4、5、6测试点的仙人掌是树。在树中两个点间的简单路径是唯一的，这非常方便我们处理，所以先来看一看在我们熟悉的树上应该如何处理这个问题。

2.1 算法一

我们可以先考虑序列的时候该如何处理。

2.1.1 序列算法一

在序列上一条路径就是一个区间，我们将这些区间按照左端点排序，进行一次扫描，不断合并两个相交区间，然后对最后的每个不相交区间求出权值。

所以我们就要支持维护单点修改区间询问，我们可以使用线段树或树状数组等算法，单次时间复杂度均为$O(\log n)$，那么一次操作的总时间复杂度就是$O(k\log n)$。

2.1.2 序列算法二

直接用线段树维护，每个结点打上是否已经被覆盖的标记就好了，比较麻烦的是每次询问后需要将线段树重置，可以记录下每次修改过的结点，在询问结束以后暴力修改回来，当然也可以直接可持久化，时间复杂度也是$O(k\log n)$。

回归到树上，我们同样可以使用轻重链剖分将询问转化为区间求并。

轻重链剖分算法已经相当普及，在信息学竞赛中也屡次出现，为了便于理解下文，这里进行一些简单的介绍[1]。

轻重链剖分的核心思想是将树剖成若干条链，每个点都属于并仅属于一条链，那么一条路径就会由这些链的若干区间构成，我们就将树转化为了序列。

我们需要一种启发式的剖分方式，使得根到每个点的路径经过的链都尽量少，这就有了轻重链剖分——对于一个点u，找到它子树最大的子结点v，

[1]详细可见参考文献[1]

将(u, v)标记为重边，其余边则为轻边，按照重边构成的链剖分，这些链称为重链。

可以发现从根向下走的过程中，如果要通过轻边更换重链，则子树大小至少减小为原来的一半，所以只会经过$O(\log n)$条重链，也就是说任意两点路径被分成的区间数量是$O(\log n)$的。

那么就变成了$O(k \log n)$个区间求并，再利用上面的序列算法，时间复杂度为$O(S \log^2 n)$，期望得分30至50。

2.2 算法二

想要做到更优的时间复杂度，我们可以换一个思路。

如果询问只有一次，对于一条路径$< u, v >$，我们可以在$u, v, Lca(u, v)$（这里的$Lca(u, v)$指的是u和v的最近公共祖先）打上标记，然后遍历整棵树，通过标记判断每条边是否被经过，时间复杂度$O(n)$。

因为关键点的数量是$O(k)$的，很容易想到虚树，这也是经典算法，所以这里也只进行简单介绍[2]。

虚树指的是将原树的一个连通子树，将其未分叉的链缩为一条边形成的树。那么我们就要求出一个包含所有关键点的、点数尽量少的虚树。

一个简单的构造方法是维护一个栈，将关键点按照dfs序排序依次加入虚树，求出当前栈顶和该点的Lca，不断弹栈直到栈顶是Lca的祖先，然后将Lca和该点都入栈并修改结点的父亲。

可以发现这个构造方式构造出的虚树点数是$O(k)$的，时间复杂度是$O(k \log n)$。

那么我们就要对虚树上每条边求出它的边权，可以发现虚树的一条边就是原树上一条链$< u, v >$，而且u是v的祖先。我们可以直接使用Link-Cut Tree来维护这棵树，那么求一条边的时间复杂度是$O(\log n)$。

一个更加简单的方法是我们维护每个点到根的路径长度，修改一条边只会对一棵子树内的点造成影响，一棵子树在dfs序上是连续的，直接用线段树维护区间加法就好了，时间复杂度也是$O(\log n)$。

所有的边权求出以后就可以遍历这个虚树得到答案。

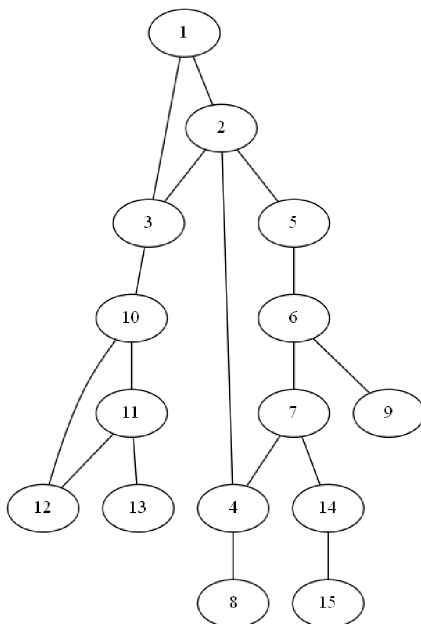
那么我们构建虚树的时间复杂度是$O(k \log n)$，共有$O(k)$条虚树上的边需要

[2]详细可见参考文献[2]

求值，时间复杂度也是$O(k \log n)$，所以单次操作的时间复杂度是$O(k \log n)$，总时间复杂度$O(S \log n)$，期望得分50。

接下来的测试点都是仙人掌，为了便于描述算法我们对仙人掌作出一些定义[3]。

2.3 一些定义



如图是一棵以1号点为根的仙人掌。下文中的例子如果没有特殊说明均指该仙人掌。

2.3.1 最长链

定义两个点u和v的最长链为他们间经过点最多的简单路径，例如10和5的最长链是10-3-1-2-4-7-6-5。

2.3.2 最短链

定义两个点u和v的最短链为他们间经过点最少的简单路径，例如10和5的最

[3]某些定义可能和参考文献[3]不同，请注意区分

短链是10-3-2-5。

2.3.3 环的父亲

定义一个环的父亲是这个环距离根最近的结点，例如环10-11-12的父亲是10。这里的环指的是简单环，下文同。

2.3.4 子仙人掌

定义一个点的子仙人掌是删去它到根的最长链和最短链经过的边以后它所在的连通块，例如点6的子仙人掌就是6和9两个点。

2.4 算法三

对于测试点2，结点数和询问的路径数量都很少，我们可以对于每个最长链或者最短链求出它经过了哪些边，然后统计答案。

首先我们对这棵仙人掌进行一次dfs，形成一个dfs树，那么每个环都由一条dfs树上的链和一条返祖边组成（例如环2-5-6-7-4就是由链2-5-6-7-4和返祖边(4,2)组成）。

对于一条询问的路径，我们先从dfs树上走过去，然后枚举每个环，如果一个环被路径经过那么就会产生两种走法，如果另一种更优就替换。

例如从4到9走最长链，先从树上走过去得到4-7-6-9，环2-5-6-7-4被经过，它存在两种走法，分别是4-7-6和4-2-5-6，显然后者更优，我们将原路径替换得到4-2-5-6-9。

因为任意两个环都不存在相同的边，所以枚举所有环的时间复杂度是$O(n)$（m和n同阶于是不进行区分，下文同），总时间复杂度$O(S n)$，结合树上算法可以得到60分。

2.5 转化

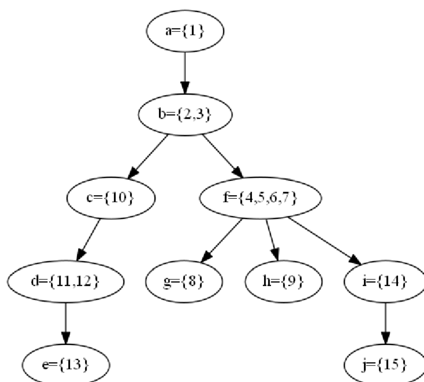
直接处理仙人掌过于困难，我们考虑将仙人掌中的环全部缩掉，使其转化为树，再利用处理树的算法来解决仙人掌的问题。

缩环有一个问题，因为一个点可能出现在多个环中，例如之前的点2。但我们发现环的父亲到根的路径中并不会经过这个环的其它结点，而环中其它结点

到根的最短链和最长链的并则会完全覆盖这个环，那么我们将环的父亲不缩到这个环中。

那么每个点只会被缩到dfs树上它到父亲的边所在的环，而一条边只会在一个环中，这样每个点都会唯一对应一个新点（当然一个新点不只对应原来的一个点），如果这条边不在任何一个环中，这个点就不缩。

我们将这样缩环后形成的树称作缩环树[4]，例子中的仙人掌形成的缩环树如下：



可以看到点b, d, f都是缩环形成的点，都表示一个环。

2.6 例题：静态仙人掌[5]

一个n个点m条边的仙人掌，每个点有一个颜色（黑或白），初始时均为黑色，有q次操作，操作有以下3种：

1. 将给定的点x到根的最短链颜色取反。
2. 将给定的点x到根的最长链颜色取反。
3. 询问给定的点x的子仙人掌里的黑点数目。

$n, q \leq 50000$。

我们将这个问题一般化：给你一个仙人掌，要求支持对给定的一条最长链、最短链或者子仙人掌进行操作和询问（这里以点权为例，边权类似）。

[4]这是作者起的名字

[5]来源：清华集训2015第二天第一题

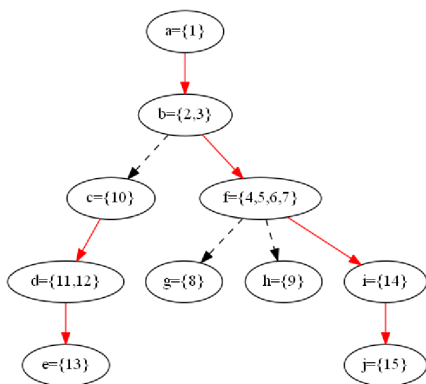
2.6.1 树上的情况

如果仙人掌是树，那么可以使用前文提过的轻重链剖分，按照dfs序将重链排列在一起就可以保证子树也是连续的区间，再用数据结构维护就好了。

链会被划分为$O(\log n)$个区间，子树则是$O(1)$个区间。

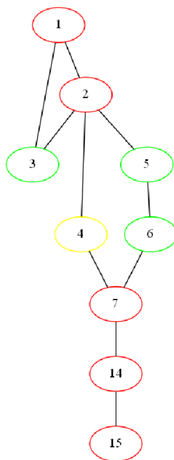
2.6.2 缩环树的轻重链剖分

我们之前将仙人掌转化为了缩环树，那么就可以对这棵缩环树进行轻重链剖分。



这里红边表示重边，虚线表示轻边。

如果这是棵树，我们会将其排列成$\{a, b, f, i, j, h, g, c, d, e\}$的序列，那么对于仙人掌我们要将其排列为什么样的序列呢？



考虑一条重链，比如$a-b-f-i-j$，一个点可能有一种（如果它代表一个点）或两种（如果它代表一个环）走法，我们考虑从15号点走到1号点。

红色的点表示又会被最长链经过也会被最短链经过的点，绿色的点是只会被最长链经过的点，黄色的点是只会被最短链经过的点。

可以发现，如果是缩环树上的一个点，如果它代表一个点，那么一定是红色，如果它代表一个环，这个环和下一个点相交的部分是红色（比如2和7）。

对于一个环，它的两种走法只取决于该环上起点的位置，而且对于同一条重链上的点这个位置是相同的。

那么我们可以对每一条重链分别处理，预处理出每个点的颜色，对于这三种颜色分别建立一个序列，这些序列是将$\{a,b,f,i,j,h,g,c,d,e\}$扩展得到的，将每个点扩展为环上的一段。

对于这个例子，黄色序列就是{4}，红色序列就是{1,2,7,14,15,9,8,10,11,13}，绿色序列就是{3,5,6,12}。

那么对于一条最短链或者最长链，他们在缩环树上的路径会经过$O(\log n)$条重链和$O(\log n)$条轻边，重链可以表示为$O(\log n)$个三种序列上的区间，而每条轻边可以判断应该如何在环上走并且转化为$O(1)$个区间。我们直接用数据结构维护这三个序列就好了。

对于子仙人掌操作，如果这个点在缩环树上不属于环，那么直接在三个序列上找到对应的区间就好了。

如果这个点属于某个环，我们可以将这个环的孩子顺序进行调整，使得每个结点的子仙人掌都是一段区间。

于是我们通过对缩环树进行轻重链剖分，也可以做到和树相同的复杂度，链会被划分为$O(\log n)$个区间，子树则是$O(1)$个区间。

2.7 算法四

我们直接对缩环树的轻重链剖分，利用算法一里序列的维护方式就好了，时间复杂度$O(S\log^2 n)$，期望得分80至100。

2.8 例题：mx的仙人掌[6]

一个n个点m条边的仙人掌，定义两个点的距离为最短路长度，q次询问，每次给定一个点集，询问这个点集中最远点对的距离。

$n, q \leq 300000$，点的个数之和小于等于300000。

2.8.1 单次询问

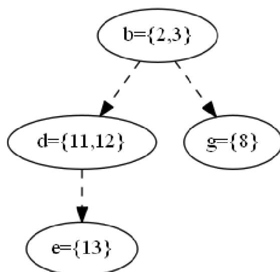
如果只有一次询问，我们该怎么做？

我们可以遍历整个仙人掌，进行一次仙人掌dp[7]，记录下每个点的子仙人掌中距离它最远的关键点，如果遇到了一个环还要再进行一次环dp，时间复杂度$O(n)$。

2.8.2 缩环树的虚树

注意到点集的点数并不多（假设有k个点），仿照树上的情况，我们可以对缩环树也建出虚树，然后对这棵虚树进行dp。

还是以前面的仙人掌和缩环树为例，比如点集为{8,12,13}，它们对应到缩环树上的点是$\{d, e, g\}$，那么虚树如下：



这棵虚树上的点个数是$O(k)$，但是一个点可能代表一个环，对应到仙人掌上点数却是$O(n)$，所以不能直接dp。我们考虑需要dp的点，一定是读入点集中的点，或者是一个环上一个子仙人掌存在关键点的点（比如b里的2，子仙人掌里有8），这样仙人掌上点数也是$O(k)$个，就可以dp了。

[6]来源：UOJ #87. mx的仙人掌

[7]详细可见参考文献[3]

那么我们就要求出虚树上边（也就是仙人掌上一条最短链）的长度，实际上我们特殊考虑最短链的两个端点所在环，剩下部分可以直接通过预处理一个点到根的最短链长度得到。

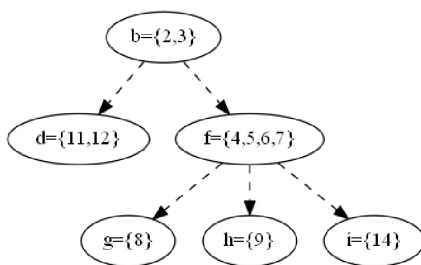
我们发现这样做需要支持询问一个点在一个环的哪个结点的子仙人掌中，这可以通过二分dfs序得到（用类似之前轻重链剖分的方法构建dfs序可以使得子仙人掌是连续一段）。

所以单次询问的时间复杂度是$O(k\log n)$。

2.9 算法五

回到这道题里，我们同样可以对询问的路径端点（假设有k条路径）在缩环树上建出虚树，然后对这个虚树进行遍历，我们举一个例子来考虑哪些部分需要统计并如何维护。

有两条路径，11到14的最长链11-12-10-3-1-2-5-6-7-14和8到9的最短链8-4-7-6-9，对应的缩环树上的点是d, i, h, g，虚树如下：



2.9.1 环内

和树不同，缩环树的环内依然存在边可能对答案作出贡献，在环内产生的贡献有两种方式：

1. 作为一条路径在缩环树上的最近公共祖先产生的贡献：比如8和9所属的点g和h的最近公共祖先f，8是4的子仙人掌中的结点，9是6的子仙人掌中的结点，所以产生了4-7-6的贡献。

2. 一条路径要越过这个环向上走的时候产生的贡献：比如11到14的最长链要越过环f，14是7的子仙人掌中的结点，那么就产生了7到环f的父亲2的最长链7-6-5-2的贡献。

显然第一种贡献只有$O(k)$个，而第二种一个贡献都会对应虚树的一条边，那么也只有$O(k)$个。

这两种贡献都是一些区间，那么可以直接使用算法一中的序列处理方法进行统计，维护修改边权也同样，这部分的时间复杂度是$O(k \log n)$。

2.9.2 虚树的边

和树相同的部分就是虚树的边，一条虚树的边可以看成从一个环的父亲（或一个点）到某个环的结点（或一个点）的最长链、最短链或最长链和最短链的并。比如b和d 的边就可以看成d环的根10到b上结点3的最长链。

我们可以维护每个点到根的最长链、最短链、最长链和最短链的并的长度，边权就是这两个点的差。同树上类似，我们可以按照dfs序进行维护。

如果修改的是某个不属于缩环树上环的点到其父亲的边（例如10到3的边），那么只有这个点的子仙人掌会受到影响，我们修改这个区间的三种权值就好。

如果修改的是某个环上的边，我们可以将这个环分成至多三段，每一段都会被修改最长链或最短链的一种，那么我们将同一个环的dfs序放在一起就可以很好地维护了。

举个例子，如果修改5到6的边，5的子仙人掌的最长链会被修改，6的子仙人掌的最短链会被修改，4和7的子仙人掌的最长链会被修改。

我们可以用线段树来维护，那么需要查询的边数是$O(k)$，每次询问的复杂度是$O(\log n)$，所以总时间复杂度$O(k \log n)$。

那么我们就用$O(S \log n)$的时间复杂度做完了这道题，期望得分100分。

3 命题思路和总结

近年来信息学竞赛出现了不少仙人掌相关的题目，清华集训2015第二天第一题《静态仙人掌》给了我很多启发，虽然我成功想出了比标准算法更优秀的方法（也就是本文的缩环树轻重链剖分方法）却没有成功地实现代码，不过这激起了我研究仙人掌的兴趣。

我通过研究被我称为“缩环树”的结构，发现在这个结构上虚树算法依然成立，时间复杂度也没有变化。本题在树上也算是一个经典问题，所以就将其

扩展并命了这道题。

这道题并不是很容易，选手需要对仙人掌的性质有相当的了解、对经典算法的熟练掌握和较强的代码实现水平才能做出。

我将这道题出到集训队互测中是希望起到一个抛砖引玉的作用，希望大家对于仙人掌，也就是树的一个拓展结构产生一些兴趣。目前仙人掌相关的算法虽然时间复杂度都比较优秀，但是代码都相当难以实现，希望大家能提出一些更为有效、更为方便的算法。

4 感谢

感谢叶国平老师在学习和生活上的帮助和关心。

感谢CCF提供的交流机会。

感谢罗哲正、高闻远和龚子清同学为本文审稿。

参考文献

[1] 杨哲,《SPOJ375 QTREE解法的一些研究》, 2007年国家集训队作业。

[2] 徐寅展,《线段树在一类分治问题上的应用》, 2014年国家集训队论文。

[3] 王逸松,《仙人掌相关算法及其应用》, 2015年国家集训队论文。

基础排序算法练习题 命题报告

杭州学军中学 金策

摘 要

本文作者在此次集训队互测第一轮比赛中命制了一道以数组排序为背景的算法题目。本文将介绍此题的解法，以及命题时的思路，并进行相关的拓展讨论。

1 试题

1.1 题目描述

对于一个长度为n的数组$a[1], a[2], \cdots, a[n]$，定义这样一个算法：它包含m个步骤，其中第i步是将$a[l_i], a[l_i+1], \cdots, a[r_i]$原地升序排序。

共有q个询问，每个询问中给出一个初始数组a，你需要判断上述算法能否将这个数组升序排序（即最终是否满足$a[i] \le a[i+1], i = 1, 2, \cdots, n-1$）。

1.2 输入格式

第一行三个整数n, m, q。

接下来m行按顺序给出了算法中每个步骤的参数l_i, r_i。

接下来q行，每行表示一个询问，包含空格隔开的n个非负整数$a[1], a[2], ..., a[n]$。

1.3 输出格式

对于每个询问输出一行，如果上述算法能将这个数组升序排序，输出`AC`，否则输出`WA`。

1.4 样例输入

```
6 3 2
1 3
3 6
1 3
4 2 2 3 0 7
5 3 8 2 1 9
```

1.5 样例输出

```
AC
WA
```

1.6 样例解释

询问1：$422307 \to [224]307 \to 22[0347] \to [022]347$，正确。

询问2：$538219 \to [358]219 \to 35[1289] \to [135]289$，错误。

1.7 数据规模与约定

编号	$n \le$	$m \le$	$q \le$	备注
1	100	100	100	
2	200	200	200	
3	1400	900000	8	
4	1500	1000000	8	
5	1400	4000	1400	
6	1500	5000	1500	
7	1200	700000	1200	
8	1300	800000	1300	$a[1..n]$包含$\lfloor n/2 \rfloor$个1和$\lceil n/2 \rceil$个0
9	1400	900000	1400	
10	1500	1000000	1500	

对于所有数据，$1 \le n \le 1500, 1 \le m \le 1000000, 1 \le q \le 1500, 1 \le l_i \le r_i \le n, 0 \le a[i] \le 1500$。

1.8 时间限制

每个测试点2.5秒

1.9 空间限制

512MB

2 得分情况

共有12名选手参加了此次互测比赛。

比赛前对本题的分数估计：1 ∼ 3名选手获得80 ∼ 100分的高分，三分之二左右的选手能获得至少40分，所有选手都能获得至少20分。

实际得分情况为：1名选手获得了100分，3名选手获得了50分，7名选手获得了40分，1名同学获得了20分，总体上与预期情况相符。

3 朴素算法

首先考虑最朴素的算法。

对于每次询问，暴力执行算法的m个步骤，每个步骤用$O(n^2)$的冒泡排序法[1]或插入排序法[2]实现。最后检查一遍数组是否有序。时间复杂度是$O(qmn^2)$。

可以考虑对这个算法进行优化，将排序算法改成$O(n \log n)$的快速排序[3]或者`std::sort`[4]。于是时间复杂度变为了$O(qmn \log n)$，可以通过20%的数据。

---

[1] http://en.wikipedia.org/wiki/Bubble%5fsort
[2] http://en.wikipedia.org/wiki/Insertion%5fsort
[3] http://en.wikipedia.org/wiki/Quick%5fsort
[4] http://www.cplusplus.com/reference/algorithm/sort/

4 逆序对

逆序对定义为序列$a[1..n]$中的一对下标(i,j)，满足$1 \le i < j \le n$且$a[i] > a[j]$。它是一个重要的概念，且与排序算法有着紧密的联系。

例题1 (经典问题). 输入一个数组$a[1..n]$，每次允许交换相邻两个数$a[i], a[i+1]$，问最少经过多少次操作可以使数组排好序。$(n \le 10^5)$

注意到，每次操作后，数组的逆序对数量会增加1或者减少1，因此操作次数的下界就是初始数组逆序对数量。显然这个下界是可以达到的，只需每次选取某个$a[i] > a[i+1]$的i，并将这两数交换，直到不存在这样的相邻逆序对为止，此时数组肯定已经被升序排序。

所以只需要计算出初始数组的逆序对数量即可，这可以用归并排序法[5]来解决，每轮归并的同时可以算出跨越数组中点的逆序对数量，剩下的逆序对数可以递归下去计算。

现在回到原题，可以发现在测试点$3,4$中，q的规模较小，而n,m的规模较大，所以需要优化询问一次的复杂度。

我们采用上题中交换相邻逆序对的算法来进行排序，由于a数组的初始逆序对数量是$O(n^2)$级别的，因此在整个算法过程中我们至多进行了$O(n^2)$次交换。

但这样还不够，因为我们每轮排序时要对$a[l_i..r_i]$进行一次扫描，需要$r_i - l_i = O(n)$的时间。如果这段区间长度很长而其中逆序对数较少，那么这趟扫描的时间是浪费的。

因此，为了高效维护，可以把当前所有$a[i] > a[i+1]$的i都插入到一个平衡树（或`std::set`[6]）里。每次将$a[l..r]$进行排序时，找出`set`中不小于l且最小的i，如果$i \ge r$，说明$a[l..r]$已经有序，可以结束操作；否则将$a[i], a[i+1]$交换，然后检查$a[i-1], a[i]$和$a[i+1], a[i+2]$的大小关系，并在`set`中进行更新。

于是总的时间复杂度是$O\left(q(m+n^2)\log n\right)$，可以通过40%的数据。[7]

[5] http://en.wikipedia.org/wiki/Merge%5fsort

[6] http://www.cplusplus.com/reference/set/set/

[7] 将set换成vEB树可以得到更优的理论复杂度。(http://en.wikipedia.org/wiki/Van%5fEmde%5fBoas%5ftree)

5 01序列

01序列指的是元素全部为0或1的序列。在与数列排序有关的问题中，考虑01序列的特殊情况往往会有利于问题解决。

例题2 (BestCoder Round #76 div.1 Problem 1004). 维护一个数组$a[1..n]$，它是一个$1 \sim n$的排列。执行m次操作，操作有两种类型：

*(1)*将$a[l..r]$原地升序排序。

*(2)*将$a[l..r]$原地降序排序。

做完所有m次操作之后，回答一次询问：对于一个给定的k，输出$a[k]$的值。$(n, m \leq 10^5)$

区间操作的常用手段是线段树，但题中的操作不太容易直接在线段树上实现。这道题目的特殊之处在于询问只有$a[k]$一个位置，所以可以考虑二分答案的手段。假设当前二分的值是mid，把大于mid的数字标为1，不大于mid的数字标为0，这样就把n排列转化为了01序列。

在01序列上进行区间排序操作是很容易的：用一个懒标记线段树，支持区间求和、区间赋值。这样要排序一个区间时只要查询一下里面有几个1和几个0，然后把前半段赋值为0，后半段赋值为1即可（降序的话就是反过来）。

对所有操作做完以后，检查一下$a[k]$位置上是0还是1，就能得知实际的$a[k]$是否大于mid。总时间复杂度是$O(n \log n + m \log^2 n)$。

将一个排列转化为01序列的好处是只需要考虑大于mid和不大于mid的元素之间的关系，而在mid同一侧的元素之间的关系不需要考虑，所以会更加方便。

例题3 (经典问题). 给定一个n个元素的比较网络，判断它是否是一个正确的排序网络[8]。

若不是正确的排序网络，统计有多少个n排列不能被正确排序。

如果采用暴力检验的方法，需要对$n!$种不同的n排列分别验证一次；但实际上只需要对2^n种01序列进行检验就可以了。这是排序网络的0-1原理，它的正确性很容易证明：如果有一个n排列不能被正确排序，那么最终结果会存在逆序对$i < j, a[i] > a[j]$。记$x = a[j]$，然后只要把原序列中大于x的标为1，剩下的标为0，将得到的01序列作为输入，那么输出序列中$a[j]$位置上是0，$a[i]$位置上是1，没有被正确排序。

[8]http://en.wikipedia.org/wiki/Sorting%5fnetwork

接下来是个简单的计数问题。显然，一个n排列能被正确排序的充要条件是，对任意的k，把排列中大于k的元素标为1，剩下的标为0，得到的01序列都能被正确排序。所以，预处理出所有合法的01序列，一个合法的n排列就对应一条从$00\cdots0$到$11\cdots1$的路径，中间每步把一个0改成1，且途中经过的都是合法01序列。于是路径条数可以用一个$O(n2^n)$的DP算出。

5.1 逆向递推

现在回到原题。剩下的测试点中q的规模都较大，对每个询问进行模拟看上去比较困难。我们可以换一个思路，转而研究具有什么性质的初始序列是可以被排序的。

有了刚才的经验，我们可以先从01序列的特殊情况入手考虑，解决8号测试点。设01序列中1的数量为k。

考虑能够被正确排序的01序列，它的最终形态是固定的，所以倒过来研究会更加容易。设第m次操作后序列被正确排好序，那么在第m次操作前，序列必然是前面一段0，后面一段1，然后中间$a[l_m], a[l_m+1], \cdots, a[r_m]$这段可以会被任意打乱顺序。同理，如果确定了第$m-1$次操作结束后的结果，那么此时$a[l_{m-1}], \cdots, a[r_{m-1}]$这一段数应该是有序的，我们可以把它们任意打乱顺序，得到第$m-1$次操作前的所有可能结果。如果按这样从后往前递推，一直推到第1次操作之前，理论上就可以得到所有可被排序的初始01序列。

形式化地描述这一过程。用A_k表示所有能被第$k+1, k+2, \cdots, m$次操作正确排序的01序列所组成的集合。集合A_m中只包含一个序列，即$n-k$个0后面跟着k个1。从m到1降序枚举i，对于每一个序列$a \in A_i$，如果$a[l_i..r_i]$这段子区间是有序的，我们就把$a[l_i..r_i]$任意打乱顺序后所有可能的结果添加到集合A_{i-1}中。最终集合A_0就包含了所有可被排序的初始01序列。

5.2 代表元素

但问题在于每次“任意打乱顺序”可以得到的序列有很多种，得到的集合A_i的大小是指数级的，不可能全部显式遍历。因此我们希望找到某些简洁的性质能够描述集合A_i中的所有序列。一种自然的想法是从A_i中寻找一个特定的序列作为“代表元素”，用它来表示出整个集合A_i。也就是说，它是A_i的所有元素中情况最坏的；并且，如果某个序列比代表元素好，那么它也必然属于这个

集合。

现在我们需要一个“好坏”的定义使得这样的代表元素能够存在。显然，1越靠左，0越靠右，序列越接近降序；1越靠右，0越靠左，序列越接近升序。因此容易发现打乱顺序的最坏方式就是将$a[l_i], \cdots, a[r_i]$的1全部移到左边，0放到右边（即按降序排序），其他的方式都可以通过继续将1往左移而达到这一最坏状态。于是可以这样定义：

令全集U是所有包含k个1的n位01序列。对于一个01序列$a \in U$，用$\mathrm{pos}(a, i)$表示a中从左到右第i个1所在的位置。对于两个01序列$a, b \in U$，定义a比b优秀，当且仅当对于$i = 1, 2, \cdots, k$都有$\mathrm{pos}(a, i) \geq \mathrm{pos}(b, i)$。这是一个偏序关系。直观来说，就是可以通过将$b$中的某些1右移而得到$a$。例如，100111就比101101优秀。

为了方便叙述，再给出一些定义：

用$f_{01}(a, l, r)$表示将01序列a的第l项到第r项升序排序后得到的序列，用$f_{10}(a, l, r)$表示将a的第l项到第r项降序排序后得到的序列，$l \leq r$。容易发现这样一些性质：

(1) $f_{01}(a, l, r)$比a优秀，a比$f_{10}(a, l, r)$优秀。

(2) 令$b = f_{01}(a, l, r)$，则$f_{10}(a, l, r) = f_{10}(b, l, r)$；令$c = f_{10}(a, l, r)$，则$f_{01}(a, l, r) = f_{01}(c, l, r)$。

(3) 对于a, b串和l, r，如果a比b优秀，那么$f_{01}(a, l, r)$比$f_{01}(b, l, r)$优秀，$f_{10}(a, l, r)$也比$f_{10}(b, l, r)$优秀。

接下来需要证明代表元素的存在性。把集合A_m中的唯一元素记为a_m。根据之前的最坏递推过程，定义一组01序列$a_{m-1}, a_{m-2}, \cdots, a_0$，其中$a_{i-1} = f_{10}(a_i, l_i, r_i)$。可以用归纳法证明，一个01序列$x \in A_i$当且仅当$x$比$a_i$优秀：

如果序列$x \in A_i$，根据定义有$y = f_{01}(x, l_{i+1}, r_{i+1}) \in A_{i+1}$，即$y$比$a_{i+1}$优秀，再根据性质(3)有$z = f_{10}(y, l_{i+1}, r_{i+1}) = f_{10}(x, l_{i+1}, r_{i+1})$比$f_{10}(a_{i+1}, l_{i+1}, r_{i+1}) = a_i$优秀，又由于$x$比$z$优秀，因此$x$比$a_i$优秀。

另一方面，如果序列u比a_i优秀，根据性质(3)可知它第$i+1$次操作后的结果$v = f_{01}(u, l_{i+1}, r_{i+1})$比$w = f_{01}(a_i, l_{i+1}, r_{i+1})$优秀，又由$a_i = f_{10}(a_{i+1}, l_{i+1}, r_{i+1})$，易知$w$比$a_{i+1}$优秀，于是$v$比$a_{i+1}$优秀，即$v \in A_{i+1}$。因此$u$可被第$i+1, \cdots, m$次操作排好序，即$u \in A_i$。

于是我们不仅证明了代表元素的存在，还将集合A_i和A_{i-1}间的递推转化为了代表元素a_i和a_{i-1}间的递推。

5.3 针对8号点的算法

根据上面发现的结论，我们得到了针对8号测试点的特殊算法：预处理出串a_0，然后每次判断输入的01序列是否比a_0更优秀即可。

其中第二步可以容易地$O(n)$实现；第一步预处理中，我们需要高效实现给一段区间中的01排序，这可以使用消逆序对的算法实现。

时间复杂度是$O\left((m+n^2)\log n+qn\right)$。

5.4 满分算法

不妨假设待排序的数组a是一个n排列。如果不是的话，可以先将它离散化，其中值相同的元素按照下标递增的顺序分配。容易看出这样并不会影响最终的答案。

现在尝试将01序列的算法推广到n排列。对于初始排列$a[1..n]$和给定的k，定义01序列b_k，其中$b_k[i]=1$当且仅当$a[i]\geq k$。之前已经提到过，算法能将a数组正确排序，当且仅当对于所有$k=2,3,\cdots,n$，该算法都能将01序列b_k正确排序。

于是，根据刚才的算法，现在需要对于每个k，预处理出$n-k$个0后跟k个1的串倒着进行m次操作后的结果c_k。直接对每个分别预处理肯定太慢了，但我们可以做一件等价的事情：令初始排列为$1,2,\cdots,n$，按i从m到1的顺序，每次将第l_i项到r_i项降序排序。最后得到的序列中，将小于k的值用0代替，其他值用1代替，得到的01序列就是所求的c_k。这样做的正确性是显然的，而且可以由此得知c_{k-1}是由c_k把某个位置的0改为1而得到的。

现在要对于所有$k=2,3,\cdots,n$检查b_k是否比c_k优秀。这是一个经典问题，可以将b_k中的1取负号，c_k中的1取正号，每次加入一对新的$+1,-1$后，检查数组的前缀和是否非负。这可以用经典的线段树区间加、维护区间最小值来实现。时间复杂度是$O(n\log n)$。

预处理的过程可以用消逆序对的算法实现。因此算法的总时间复杂度为$O\left((n^2+m+qn)\log n\right)$，可以通过100%的数据。

6 其他部分分算法

如果选手发现了上面的结论，但不会使用消逆序对技巧进行预处理，也可

以拿到比较可观的分数。例如在第5、6号测试点中，m的值较小，可以直接每次暴力排序；在8号测试点中，由于待排序的数字都是0和1，所以也可以转化成区间求和与区间赋值操作，用线段树实现。

另外也有一种可能通过5、6号点的做法：考虑用数据结构维护这个序列，将它分成若干段，使得每段都是单调递增的，并把一段中的数字放进一棵平衡树维护。进行区间排序操作时，只要在区间边界处断开，然后将中间的若干段进行归并，归并时使用一些合并平衡树的技巧[9]。但是这个算法的复杂度并不容易分析。

还有一种做法是将永远用不到的操作给去掉，然后暴力模拟每个询问。可以用这样的方法去掉无用操作：令初始数列中$a[i] = n - i + 1$，并按顺序执行排序操作，如果一个操作没有使数列发生改变，则是无用操作；也可以令初始数列为$a[i] = i$，然后倒过来进行降序排序操作，并把没有用到的给去掉。

利用这些部分分算法，结合一定的优化，预计可以得到50~70分，具体得分多少取决于优化的方式以及选手的能力与经验。

7 测试数据构造

本题的数据是有点难造的，下面简单介绍一下每个测试点采用的构造方法。

7.1 1,2号点

这两个点的$n, m, q \le 200$，朴素的模拟就能通过，基本不会有超时或错误做法，所以只需随机构造即可。随机的时候让答案中的`AC`和`WA`的数量尽量平均。

7.2 3,4号点

这两个点是设置给每次用$O\left((n^2 + m)\log n\right)$时间模拟的算法的，$q$比较小；但为了防止选手用爆OJ的方式枚举答案，不能让2^q太小，所以设置为$q \le 8$。

暴力排序的复杂度是$O\left(\sum(r_i - l_i)\log n\right)$，为了把这个做法卡掉，需要让每次排序的区间尽可能长。但是同时需要注意不能让数列太早被排好序，否则选

[9]参考2015年王逸松的互测题解

手可以用适当的`break`来水过。可以在边上留着几个元素不动，等到最后再让它们参与排序。

7.3 询问数组的生成

本题中询问数组是这样生成的：构造完m个排序步骤之后，同标算一样求出对它们倒过来降序排序后的结果a，然后对这个数组进行一些随机调整。如果要生成`AC`的询问，就若干次随机选出一对比较靠近的(i, j)且$i < j, a[i] > a[j]$，然后将$a[i], a[j]$交换；若要生成`WA`的询问，就反过来交换。这样微调可以使`AC`的数据不要太早被排好序，也使`WA`的数据最后排序结果与有序数列较接近，可以防止一些不靠谱的随机做法水过。另外，数据中也掺杂了一部分几乎是降序的询问数组。

7.4 5,6号点

这两个点中m和n数量级差不多，q较大，是让发现了标算结论但不会预处理的选手，以及数据结构能力较强的选手通过的。

这里同样也需要使每次排序区间尽可能长，但要注意防止出现过多无用操作。

可以令$mid = \lceil n/2 \rceil$，然后依次对区间$[1, mid], [2, mid+1], [3, mid+2], \cdots, [n-mid+1, n]$进行排序操作，然后对左右两半分别递归构造。这些操作都是有用操作，操作个数是$\Theta(n)$，平均区间长度是$\Theta(n)$，逆序对的减少量是$\Theta(n^2)$，可以将绝大多数错误做法卡到超时。

7.5 7,8,9,10号点

7,9,10号点是给标算准备的，数据规模有梯度；8号点的构造方法与它们大同小异。这四个点中分布着两种不同类型的数据。

第一种是和3,4号点类似，操作区间都很长，但是无用操作就会变多。

第二种数据的操作区间长度几乎都是2，即每次只排序相邻两个元素，有用操作个数可以做到$\Theta(n^2)$，这个的构造只需要每次交换相邻逆序对即可。这样暴力模拟的复杂度至少是$\Omega(qn^2)$，基本没有可能通过。

由于第一种数据被水过的可能性更大，所以只放了一个9号点，剩下三个点都是第二种数据。

8 命题总结

这道题目题意非常简洁易懂，看上去与以往的许多数据结构题类似，需要维护一个序列并在上面支持区间操作；但是又和某些数据结构裸题不同，这题的瓶颈并不在于高深的数据结构理论或复杂度分析技巧，而是在于研究题中操作的性质，倒过来进行分析，从而发现一些结论。然后再根据这些结论的需要，选用经典的数据结构进行维护。本题标算中用到的数据结构有`set`以及最基本的懒标记线段树，代码难度并不高，大多数NOI水平选手应该都能熟练使用。但本题有一定的思维难度。

感谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练余林韵和陈许旻的帮助。

感谢我的教练徐先友老师多年来的培养。

感谢杜瑜皓、徐寅展、王逸松、毛啸同学在本题命制过程中与我讨论，并提出了宝贵建议。

感谢周子鑫同学验题。

参考文献

[1] Sarbazi-Azad, Hamid (2 October 2000). “Stupid Sort: A new sorting algorithm”. Newsletter (Computing Science Department, Univ. of Glasgow) (599): 4.

[2] BestCoder Round #76 Solution.

[3] 王逸松，Tree and Sets解题报告。

[4] 刘汝佳，黄亮，《算法艺术与信息学竞赛》，清华大学出版社。

move 命题报告

长沙市雅礼中学 袁宇韬

1 题目大意

在一个无限长的01序列中有s个位置为1。所有1的位置为m个区间，给出这些区间。给出k次多项式$f(x)$，定义一个区间的权值为在区间中随机选择一个开始位置，再每次随机选择下一个位置，将这个位置取反，直到区间中所有数相同时，每次移动时经过的距离代入$f(x)$的值的和的期望。有q个询问，每次询问一个长度为n的区间中将所有数任意排列后最多能增加多少权值，并按照能够使权值最大的方案排列，有多种方案时选择1的位置从小到大字典序最小的。

$n \le 10^9$，$k \le 250000$，$m \le 20000$，$q \le 15000$，$s \le 5 \times 10^7$。

2 算法介绍

2.1 基于状态压缩的做法

2.1.1 10分做法

考虑怎样求一个区间的权值。由于一个状态中只要记录当前位置和当前区间，可以得到一个状态数为$O(n2^n)$的做法。用$f_{i,t}$表示当前位置为i，当前区间的二进制表示为t时期望得到的收益。直接解方程可以求出所有状态的权值。这样可以得到10分。

2.1.2 20分做法

由于每一次移动都是随机的，当前一步得到的期望收益只与当前位置i有关。用g_t表示当前状态为t，当前位置任意，不计算当前一步的期望收益。用h_i表示从i位置开始移动一步得到的期望收益，则这个值只与i有关，可以预处理求

出。得到f和g之间的关系：

$$f_{i,t} = g_t + h_i \quad (t \neq 0, t \neq 2^n - 1)$$
$$g_t = \frac{1}{n}\sum_{i=0}^{n-1} f_{i,t \operatorname{xor} 2^i}$$

将所有f用g代入，则状态数减少为2^n。解方程求出所有答案。这样可以得到20分。

2.2 一个多项式复杂度的算法

注意到g_t的方程中有很强的对称性，可以利用这种对称性。

将不同i移动一步得到的收益h_i对g_t的贡献分开计算。令$c_{i,t}$为区间状态为t时从位置i开始移动的期望步数，则有

$$g_t = \sum_{i=0}^{n-1} c_{i,t} h_i$$

由于每一步都是随机移动，$c_{i,t}$与不同位置之间的顺序无关。这样由对称性可以得到除了第i位以外的所有位置中值相同的位置都是等价的。因此$c_{i,t}$ 只与状态t中第i位的值和状态t中1的个数有关。

令$a_{i,x}$为有i位为1的状态t中，一个值为x的位置k的$c_{k,t}$的值，其中x为0或1。

考虑一个前i位为1，后$n-i$位为0的状态t，其中$1 < i < n-1$。将g_t的转移方程用a代入得到：

$$\begin{aligned}
\sum_{k=0}^{i-1} a_{i,1}h_k + \sum_{k=i}^{n-1} a_{i,0}h_k &= \frac{1}{n}\left(\sum_{k=0}^{i-1}\left(h_k + \sum_{p<i,p\neq k} a_{i-1,1}h_p + a_{i-1,0}h_k + \sum_{p=i}^{n-1} a_{i-1,0}h_p\right)\right. \\
&\qquad \left. + \sum_{k=i}^{n-1}\left(h_k + \sum_{p=0}^{i-1} a_{i+1,1}h_p + a_{i+1,1}h_k + \sum_{p\geq i,p\neq k} a_{i+1,0}h_p\right)\right) \\
&= \frac{1}{n}\left(\sum_{k=0}^{i-1}((i-1)a_{i-1,1} + a_{i-1,0} + (n-i)a_{i+1,1} + 1)h_k\right. \\
&\qquad \left. + \sum_{k=i}^{n-1}((n-i-1)a_{i+1,0} + a_{i+1,1} + ia_{i-1,0} + 1)h_k\right)
\end{aligned}$$

由于$a_{i,x}$与h_k无关，则一定有

$$\begin{aligned} a_{i,1} &= \frac{1}{n}((i-1)a_{i-1,1} + a_{i-1,0} + (n-i)a_{i+1,1} + 1) \\ a_{i,0} &= \frac{1}{n}((n-i-1)a_{i+1,0} + a_{i+1,1} + ia_{i-1,0} + 1) \end{aligned} \quad (1 < i < n-1) \tag{1}$$

对于$i = 1$的情况，同样构造t并代入g_t得到：

$$\begin{aligned} a_{1,1}h_0 + \sum_{k=1}^{n-1} a_{1,0}h_k &= \frac{1}{n}\sum_{k=1}^{n-1}\left(h_k + a_{2,1}h_0 + a_{2,1}h_k + \sum_{p\geq i, p\neq k} a_{2,0}h_p\right) \\ &= \frac{1}{n}\left((n-1)a_{2,1}h_0 + \sum_{k=1}^{n-1}((n-2)a_{2,0} + a_{2,1} + 1)h_k\right) \end{aligned}$$

这样可以得到

$$\begin{aligned} a_{1,1} &= \frac{1}{n}(n-1)a_{2,1} \\ a_{1,0} &= \frac{1}{n}((n-2)a_{2,0} + a_{2,1} + 1) \end{aligned} \tag{2}$$

类似对$i = n-1$的情况可以得到

$$\begin{aligned} a_{n-1,1} &= \frac{1}{n}((n-2)a_{n-2,1} + a_{n-2,0} + 1) \\ a_{n-1,0} &= \frac{1}{n}(n-1)a_{n-2,0} \end{aligned} \tag{3}$$

由于$a_{i,x}$只与相邻的项有关，可以在$O(n)$的时间内解出$a_{i,x}$。

容易观察发现当$i < \frac{n}{2}$时$a_{i,1} < a_{i,0}$，当$i = \frac{n}{2}$时$a_{i,1} = a_{i,0}$，当$i > \frac{n}{2}$时$a_{i,1} > a_{i,0}$。这样如果一个询问区间中有i个1，则当$i < \frac{n}{2}$时应该将1 移动到h_k最小的位置，当$i = \frac{n}{2}$时所有方案的答案相同，当$i > \frac{n}{2}$时应该将1移动到h_k最大的位置。

由h_x定义得到

$$\begin{aligned} h_x &= \frac{1}{n}\sum_{i=0}^{n-1} f(|x-i|) \\ &= \frac{1}{n}\left(\sum_{i=0}^{x-1} f(x-i) + \sum_{i=x}^{n-1} f(i-x)\right) \\ &= \frac{1}{n}\left(\sum_{i=1}^{x} f(i) + \sum_{i=0}^{n-x-1} f(i)\right) \end{aligned} \tag{4}$$

由于$f(x)$的每一项系数均为非负整数，且最高次系数非零，得到$f(x)$一定严格递增。由于$h_x - h_{x-1} = \frac{1}{n}(f(x) - f(n-x))$，可以得到$h_x$关于$x = \frac{n}{2}$对称，且在$x < \frac{n}{2}$时递减，在$x > \frac{n}{2}$时递增。容易发现$h_x$最小的一些$x$一定形成一个区间，$h_x$最大的一些$x$一定形成两个区间。

这样每次重新排列的结果中1的位置一定是一个或两个区间。可以用平衡树维护所有区间端点。由于每个区间端点最多被删除一次，总的区间个数为$O(m+q)$，可以得到总共的区间操作次数为$O(m+q)$。每次询问时求出操作前和操作后的答案。预处理h_k的前缀和后可以在$O(1)$得到每个区间对答案的贡献。

这样预处理$a_{i,x}$的时间复杂度为$O(n)$，预处理h_k的时间复杂度为$O(nk)$，处理询问的时间复杂度为$O((m+q)\log(m+q))$。可以得到30–35分。

2.3 优化h_x的求值

2.3.1 使用Bernoulli数

由(4)，要求出h_x，只需要快速求出$\sum_{i=1}^{x} f(i)$。这是关于x的$k+1$次多项式，可以用Bernoulli数求出。如果$f(x) = \sum_{i=0}^{k} u_i x^i$，则可以得到

$$\begin{aligned}
\sum_{i=1}^{x} f(i) &= \sum_{i=0}^{k} \frac{u_i}{i+1} \sum_{j=0}^{i} (-1)^j \binom{i+1}{j} B_j x^{i+1-j} \\
&= \sum_{p=0}^{k+1} x^p \sum_{i=p-1}^{k} \frac{u_i}{i+1} (-1)^{i+1-p} \binom{i+1}{i+1-p} B_{i+1-p} \\
&= \sum_{p=0}^{k+1} \frac{x^p}{p!} \sum_{i=p-1}^{k} \frac{u_i}{i+1} (-1)^{i+1-p} \frac{(i+1)!}{(i+1-p)!} B_{i+1-p}
\end{aligned}$$

这样x^p项的系数可以用$f_i = i!u_i$和$g_{-i} = \frac{(-1)^i B_i}{i!}$两个序列卷积得到。求出Bernoulli数后用一次NTT可以求出。

求Bernoulli数可以用指数生成函数：

$$\begin{aligned} G_e(x) &= \sum_{i=0}^{\infty} \frac{B_i}{i!} x^i \\ &= \frac{x}{e^x - 1} \\ &= \frac{x}{\sum_{i=1}^{\infty} \frac{x^i}{i!}} \\ &= \frac{1}{\sum_{i=0}^{\infty} \frac{x^i}{(i+1)!}} \end{aligned}$$

这样只要求多项式的逆元。多项式求逆的具体做法详见参考文献[1]。

为了求出h_i的多项式，还需要在已知$f(x)$系数时求出$f(n-1-x)$的系数。这同样可以用NTT解决。

预处理h_i的前缀和同样可以用Bernoulli数解决。

这样可以在$O(k \log k)$时间内求出h_i的前缀和的多项式。只要代入需要的值。由于不同的区间端点个数为$O(\min(n, m+q))$的，可以在$O(k \min(n, m+q))$ 的时间内求出所有需要的i对应的值。这样可以得到45–50分。

2.3.2 使用多点求值

对若干不同的x_i求出$f(x_i)$可以使用多点求值算法。下面对多点求值算法进行简单介绍。

由于$f(x_i) = f(x) \mod (x-x_i)$，要求$f(x_1), f(x_2), \ldots, f(x_m)$，可以先求出$g_1(x) = f(x) \mod \prod_{i=1}^{\lfloor \frac{m}{2} \rfloor}(x-x_i)$和$g_2(x) = f(x) \mod \prod_{i=\lfloor \frac{m}{2} \rfloor+1}^{m}(x-x_i)$，再递归求出$g_1(x_1), g_1(x_2), \ldots, g_1(x_{\lfloor \frac{m}{2} \rfloor})$和$g_2$

由于多项式除法可以在$O(n \log n)$时间内求出，时间复杂度为$O(n \log^2 n)$ 。

多项式除法以及多点求值在参考文献[2]中有详细说明。

使用多点求值算法优化，可以得到60–70分。

2.4 进一步的优化

本题中只需要求出一个位置由0变为1或由1变为0对答案的贡献，即$a_{i,1} - a_{i,0}$。

令$d_i = a_{i,1} - a_{i,0}$，则由(1)可以得到：

$$\begin{aligned}
n(a_{i,1} - a_{i,0}) &= ((i-1)a_{i-1,1} + a_{i-1,0} + (n-i)a_{i+1,1} + 1) \\
&\quad - ((n-i-1)a_{i+1,0} + a_{i+1,1} + ia_{i-1,0} + 1) \\
&= (i-1)(a_{i-1,1} - a_{i-1,0}) + (n-i-1)(a_{i+1,1} - a_{i+1,0}) \\
nd_i &= (n-i-1)d_{i+1} + (i-1)d_{i-1} \quad (1 < i < n-1) \qquad (5)
\end{aligned}$$

同样，由(2)和(3)可以分别得到

$$nd_1 = (n-2)d_2 - 1 \qquad (6)$$

$$nd_{n-1} = (n-2)d_{n-2} + 1 \qquad (7)$$

2.4.1 d_i的求值

由对称性可以得到$d_{n-1} = -d_1$。

将(5)对于$i = 2, 3, \ldots, m$相加，并加上(6)，其中$1 \le m < n-1$，得到：

$$\begin{aligned}
\sum_{i=1}^{m} nd_i &= \sum_{i=1}^{m}(n-i-1)d_{i+1} + \sum_{i=2}^{m}(i-1)d_{i-1} - 1 \\
&= \sum_{i=2}^{m+1}(n-i)d_i + \sum_{i=1}^{m-1} id_i - 1 \\
&= \sum_{i=2}^{m-1} nd_i + (n-m)d_m + (n-m-1)d_{m+1} + d_1 - 1
\end{aligned}$$

化简得到：

$$\begin{aligned}
(n-m-1)d_{m+1} - md_m &= (n-1)d_1 + 1 \\
\binom{n-1}{m}((n-m-1)d_{m+1} - md_m) &= \binom{n-1}{m}((n-1)d_1 + 1) \\
\binom{n-2}{m}d_{m+1} - \binom{n-2}{m-1}d_m &= \binom{n-1}{m}(d_1 + \frac{1}{n-1}) \qquad (8)
\end{aligned}$$

将(8)对于$m = 1, 2, \ldots, k-1$求和，得到：

$$\begin{aligned}
\binom{n-2}{k-1}d_k - d_1 &= \sum_{i=1}^{k-1}\binom{n-1}{i}(d_1 + \frac{1}{n-1}) \\
d_k &= \frac{1}{\binom{n-2}{k-1}}\left(\sum_{i=0}^{k-1}\binom{n-1}{i}(d_1 + \frac{1}{n-1}) - \frac{1}{n-1}\right) \qquad (9)
\end{aligned}$$

在(9)中取$k = n - 1$，得到：

$$d_{n-1} = (2^{n-1} - 1)(d_1 + \frac{1}{n-1}) - \frac{1}{n-1}$$

由于$d_{n-1} = -d_1$，可以解出

$$d_1 = \frac{1 - 2^{n-2}}{(n-1)2^{n-2}} \tag{10}$$

将(10)代入(9)，得到：

$$\begin{aligned} d_k &= \frac{1}{\binom{n-2}{k-1}}\left(\sum_{i=0}^{k-1}\binom{n-1}{i}\frac{1}{(n-1)2^{n-2}} - \frac{1}{n-1}\right) \\ &= \frac{\sum_{i=0}^{k-1}\binom{n-1}{i} - 2^{n-2}}{\binom{n-2}{k-1}(n-1)2^{n-2}} \end{aligned} \tag{11}$$

这样可以在$O(k)$的时间内求出所有$1 \le i \le k$的d_i。由于在本题中$k \le s \le 5 \times 10^7$，可以在时间限制内求出所有需要的$d_i$。

2.4.2 优化空间使用

注意到存储所有d_i和中间计算过程中用到的组合数和逆元需要$O(s)$的空间，会超过空间限制，需要减少空间使用。

将(11)变形，得到：

$$\begin{aligned} d_k &= \frac{\sum_{i=0}^{k-1}\frac{(n-1)!}{i!(n-i-1)!} - 2^{n-2}}{\frac{(n-2)!}{(k-1)!(n-k-1)!}(n-1)2^{n-2}} \\ &= \frac{\sum_{i=0}^{k-1}\frac{(n-1)!}{(n-i-1)!}\frac{(k-1)!}{i!} - (k-1)!2^{n-2}}{\frac{(n-2)!}{(n-k-1)!}(n-1)2^{n-2}} \end{aligned}$$

这样需要求出$\sum_{i=0}^{k-1}\frac{(n-1)!}{(n-i-1)!}\frac{(k-1)!}{i!}$和$\frac{(n-2)!}{(n-k-1)!}$的逆元。

令$b_k = \sum_{i=0}^{k-1}\frac{(n-1)!}{(n-i-1)!}\frac{(k-1)!}{i!}$，则有：

$$\begin{aligned} b_0 &= 0 \\ b_k &= (k-1)b_{k-1} + \frac{(n-1)!}{(n-k)!} \quad (k \ge 1) \end{aligned}$$

可以在递推过程中记录$\frac{(n-1)!}{(n-k)!}$，就能在使用$O(1)$额外空间的条件下求出b_k。

令$e_k = \left(\frac{(n-2)!}{(n-k-1)!}\right)^{-1}$，可以得到$e_k = (n-k-2)e_{k+1}$。这样只要对最大的$k$求出$e_k$，再递推求出所有需要的$e_k$。

在计算过程中只需要保留需要求的k的e_k，这样可以将空间使用降低为$O(m+q)$。可以得到90–100分。

2.5 常数优化

由于NTT算法常数较大，需要进行一些常数优化才能通过本题。

2.5.1 常数分析

定义势能函数$\Phi(S)$为状态S中值为1的区间个数的两倍。

所有多点求值的x由每次询问操作前和操作后的答案两部分组成。对于操作前的答案部分，令w为完全包含在询问区间中的1区间个数，z为被部分覆盖的区间个数。显然$z \le 2$。如果询问区间包含于某个1区间，则这次询问的总询问点数为0，势能函数不变。否则这一部分的询问点数为$2w+z$。此时势能函数降低$2w$。

对于操作后的答案部分，考虑在一次询问中状态的改变。如果这次询问中的区间有i个位置为1，可以分为两种情况。

当$i \le \frac{n}{2}$时，产生一个区间。这样需要询问这个区间的两个端点，同时势能函数增加2。

当$i > \frac{n}{2}$时，产生一个前缀区间和一个后缀区间。这样只需要询问前缀区间的右端点和后缀区间的左端点。由于如果一个区间在第一部分中被部分覆盖，则可以和在这一部分中产生的区间合并为一个区间，只会产生最多$2-z$个区间。势能函数增加$4-2z$。

在第一种情况中均摊时间为$4+z$，在第二种情况中均摊时间为$6-z$。可以得到一次询问的均摊时间最多为6。

由于初始势能函数为$2m$，总的询问次数最多为$2m+6q$。在本题的数据范围中最多为130000。

2.5.2 常数优化技巧

在实现NTT时，可以预处理所有需要用到的单位根。这样可以减少乘法运

算的次数。

在实现多点求值时，需要求出一个多项式分别对两个多项式取模的结果。注意到在求出$f(x) \mod g(x)$时需要求出$f(x)$和$g(x)$的NTT结果。对于$f(x)$，可以只进行一次NTT并保存这个结果。对于$g(x)$，由于$g(x)$是由两个多项式相乘得到，在求出$g(x)$时已经得到$g(x)$的NTT结果，可以直接使用。

3 数据生成方式

3.1 数据点1–10

在这10个数据点中，所有数据为完全随机生成。

3.2 数据点11–12

在这两个数据点中，在随机数据的基础上生成了一些长度较大的区间。同时生成了$i = \frac{n}{2}$的询问。

3.3 数据点15–20

在这六个数据点中，初始时只有一个区间。每次会随机选择当前一个区间的后缀询问。询问时一定会将这个后缀向右移动。反复选择后缀操作，当区间长度过短时从初始的左端点重新开始，直到生成了足够的询问。这样可以得到多点求值的点数较多的数据，可以达到大约100000。

3.4 数据点13–14

在这两个数据点中，生成了多点求值的点数较多的数据。

在初始时生成两个区间p和q，长度分别为a和b，$b > \frac{n}{2}$，中间有一定间隔。第一次询问时选择区间q。这样可以使得这一次询问的点数达到6 。同时区间q分为两个区间q_1和q_2，长度均小于$\frac{n}{2}$。第二次询问时选择区间q_1，同时覆盖区间p和q_2的一部分，并满足区间q_1在询问区间的偏左侧。这样这一次询问的点数为6，同时q_1向右移动。可以使得q_1 左端点和q_2右端点的距离不超过n，这样可以将q_1与q_2结合看作q，并继续进行。

但由于这种方法需要满足$b > \frac{n}{2}$，因此生成一些询问后将不能继续进行。可以用一些询问将所有区间合并为类似初始时的形式。

这样生成的数据多点求值的询问点数可以达到大约125000。

4 总结

本题出题思路来源于Topcoder SRM 641 BitToggler。本题将其中的一部分推广到$n = 10^9$，$s = 5 \times 10^7$。得出本题结论有一定的思维难度，同时结合了多项式相关算法，考察了选手的代码能力。

5 感谢

感谢CCF提供学习和交流的平台。

感谢汪星明老师对我的教导。

感谢帮助过我的同学们。

参考文献

[1] 彭雨翔, “Inverse Element of Polynomial”, http://picks.logdown.com/posts/189620-inverse-element-of-polynomial

[2] 彭雨翔, “Polynomial Division”, http://picks.logdown.com/posts/197262-polynomial-division

[3] 刘汝佳,《算法竞赛入门经典》, 清华大学出版社。