寻找第 k 优解的几种方法

绍兴市第一中学 俞鼎力

摘 要

求第 k 优解是信息学竞赛中十分常见的题目类型。本文对这类题目的解法进行了分类归纳，使得此类问题可以一般化地转化成较普通、较简单的问题。此外，本文给出了 k 短路的 $O(n\log n+m+k\log k)$ 算法和 k 小生成树的 $O(m\log n+k^2)$ 算法以及 k 小简单路径的 $O(kn(m+n\log n))$ 算法。对于文中的每一项内容（除去算法介绍的章节），都给出了例题，便于读者理解和归纳。

1 引言

好奇心是学者的第一美德。——居里夫人

在信息学竞赛中，解决完某个问题之后，我们往往会考虑原问题的一些变种，比如改变题目中某个条件，改变题目中的目标函数，甚至寻求该问题的第 k 优解。

信息学竞赛中，求解第 k 大值或第 k 小值的题型已经非常普遍。对于这类题型，我们称之为求第 k 优解。求第 k 优解可能需要用到的算法很多：比如数据结构、动态规划、图论……

我们有一些已知的方法来寻找第 k 优解或者将问题转化得较为简单。比如二分、权值线段树、字母树上统计等等。也有一些大家并不是非常熟悉的算法，比方说 k 短路算法、k 小生成树算法、k 小简单路径算法。本文对这些内容进行了全面的描述，对大家熟悉的部分进行了总结归纳，对大家并不熟悉的部分进行了详细的讲述。希望能给所有参加信息学竞赛的同学带来或多或少的帮助。

在本文第二节，作者对第 k 优解进行了数学化的定义，为之后的论述打下基础。紧接着，作者证明了使用二分法求解第 k 优解的正确性，并给出了两道有趣的例题供读者深入研究。

在本文第三节，作者首先引用了 2013 年许昊然对整体二分的论述[3]，着重讲了权值线段树与整体二分的联系，以及它在求第 k 优解时的两种使用方法，对这两种方法分别给出了一道例题。

在本文第四节，作者归纳了在字母树上统计来求第 k 优解的方法。

在本文第五节，作者详细地讲述了优先队列在求第 k 优解时的应用，并对辅助用的图 P 下了定义并进行了扩展。之后的所有内容都利用了优先队列，并且都会使用到辅助用的图 P.

在本文第六节，作者对 k 短路算法进行了详细的讲解，并提及了堆的可持久化。k 短路算法的应用在本文第七节中提到，即一类动态规划问题第 k 优解的通用做法。

在本文第八节，作者耗费大量的篇幅介绍了 k 小生成树算法。刘汝佳在《算法艺术与信息学竞赛》一书中提及过 k 小生成树，但是只给出了一个不加证明的定理，且没有给出具体做法。作者查阅了诸多国外的文献资料，得出了本文中 k 小生成树的 $O(m\log n + k^2)$ 算法。

在本文第九节，作者对 k 小生成树的构图 P 的方法进行总结，并在此基础上提出了 k 短简单路径的 $O(kn(m + n\log n))$ 算法。

2 二分法求第 k 优解

2.1 第 k 优解的定义

首先，我们需要给第 k 优解一个数学化的定义。

先给出最优化问题的数学描述：

定义2.1. *最优化问题即在一给定集合上求某函数的极值(极大化或极小化)问题. 对于极小化情形，即为求*

$$\min_{u\in U} J(u),$$

其中 u 表示问题的一个解，$u \in U$ 表示解 u 需要满足给定的约束(组成集合 U 的条件称为约束条件)，函数 J 表示对解的优劣判断的指标(称为问题的目标函数).

类似的我们可以给出第 k 优解的数学描述：

定义2.2. *第 k 优解问题即在一给定集合上求某函数的第 k 极值(极大化或极小化)问题. 对于极小化情形，即为求*

$$\max_{u\in K} J(u),$$

其中解集 K 满足

$$|K| = k \quad 且 \quad \max_{u\in K} J(u) \le \min_{v\in U-K} J(v).$$

2.2 一个转化

定理2.1. *第 k 优解问题在只需求 $\max_{u\in K} J(u)$ 而不需要求解 u 的具体方案时，可以花费 $O(\log(\sup J(U) - \inf J(U)))$[1] 的代价转化为一个计数问题。其中 $J(U)$ 表示函数 J 在定义域 U 下的值域，且 $J(U) \subseteq \mathbb{Z}$.*

Proof. 这里只证明第 k 优解的极小化情形。设集合

$$S(x) = \{u \in U : J(u) \le x\},$$

则

$$\max_{u\in K} J(u) = \min_{|S(x)|\ge k} x.$$

由于 $|S(x)|$ 关于 x 是一个单调函数，所以我们可以二分 x 求出 $\min_{|S(x)|\ge k} x$.

由于 $J(U) \subseteq \mathbb{Z}$，根据整数的二分复杂度可以得到，问题能通过花费 $O(\log(\sup J(U) - \inf J(U)))$ 的代价转化为：给定 x, U, J，求

$$\sum_{u\in U, J(u)\le x} 1.$$

第 k 优解的极大化情形同理。 □

定理 2.1 中要求 $J(U) \subseteq \mathbb{Z}$，如果 $J(U) \subseteq \mathbb{R}$ 且题目要求四舍五入到 d 位小数，那么我们就可以令 $J'(u) = \text{round}(J(u) \cdot 10^d)$[2] 将其转成满足条件的函数。

注意到定理 2.1 中所求的计数问题为

$$\sum_{u\in U, J(u)\le x} 1,$$

[1] $\sup S$ 和 $\inf S$ 分别表示集合 S 的上界和下界

[2] $\text{round}(x)$ 表示 x 四舍五入之后的值

它是在 $u \in U$ 的基础上添加了一个条件 $J(u) \leq x$，因此往往在 $J(u)$ 比较简单的情况下，可以使用定理 2.1 来简化问题；而对于 $J(u)$ 计算非常复杂的情况，我们需要简化 $J(u) \leq x$ 这个条件(如例题 2)，如果无法简化，运用定理 2.1 可能会使得问题变得更加复杂。

2.3 例题一

例1 (COCI 2011/2012 4^{th} round BROJ). 给定一个质数 p，求最小质因子等于 p 的第 k 小正整数。若答案大于 10^9 则输出 0.

此时 $J(u) = u$，$U = \{u \in \mathbb{Z}^+ : u \leq 10^9 \text{ 且 } u \text{ 的最小质因数为 } p\}$. 可以发现 $U = \{vp : v \leq \frac{10^9}{p} \text{ 且 } v \text{ 的最小质因数不小于 } p\} \subseteq \{vp : v \leq \frac{10^9}{p}\}$.

当 $p \geq 67$ 时，$|\{vp : v \leq \frac{10^9}{p}\}|$ 不大，可以直接枚举其中的所有元素，然后判断其是否在 U 中，找到满足条件的第 k 小即可。

当 $p \leq 61$ 时，根据定理 2.1，可以在 $[0, 10^9]$ 中二分 x，并统计

$$\sum_{v=1}^{\lfloor \frac{x}{p} \rfloor} (1 - [v\text{的最小质因数小于}p])^{3}.$$

然后根据容斥原理可以得到

$$\sum_{v=1}^{\lfloor \frac{x}{p} \rfloor} \left(1 - [v\text{的最小质因数小于}p]\right) = \sum_{d=1}^{\lfloor \frac{x}{p} \rfloor} \left([d\text{的质因子均小于}p] \cdot \mu(d) \left\lfloor \frac{x}{pd} \right\rfloor \right).$$

而质因子均小于 p，且 $\mu(d) \neq 0$ 的 d 的个数不超过 $2^{\pi(p)}$ 个[4]，可以顺利通过此题。

复杂度 $O\left(\min\left\{\frac{L}{p}, 2^{\pi(p)} \log L\right\}\right)$，此题中 $L = 10^9$.

2.4 例题二

例2 (TopCoder SRM 607 Div. 1 - Level 3). 桌面上有两个钉子和 n 个滑轮。两个钉子的坐标分别为 $(0, 0)$ 和 $((n+1)d, 0)$. 每个滑轮的半径均为 r，滑轮的圆心坐标为 $(d, 0), (2d, 0), \ldots, (n \cdot d, 0)$. 保证滑轮不会重叠。

[3] $[S]$ 当命题 S 为真时等于1，否则等于0

[4] $\pi(n)$ 表示不超过 n 的素数个数

要将两个钉子用一个**紧绷**的绳子连起来，绳子可以在滑轮上绕任意圈。现给定 n, d, r, k，问在所有方案中，第 k 短的绳子长度。要求与答案误差不超过 10^{-9}.

数据范围：$1 \le r \le 499\,999\,999, 2r+1 \le d \le 1\,000\,000\,000, 1 \le n \le 50, 1 \le k \le 10^{18}$.

同样套用定理 2.1，首先二分绳子的长度 *bound*，统计绳子长度不超过 *bound* 的方案数。

将绳子的每一段分成 4 类：e, A, B, R，如图 1。四者的长度均能通过简单几何得出。

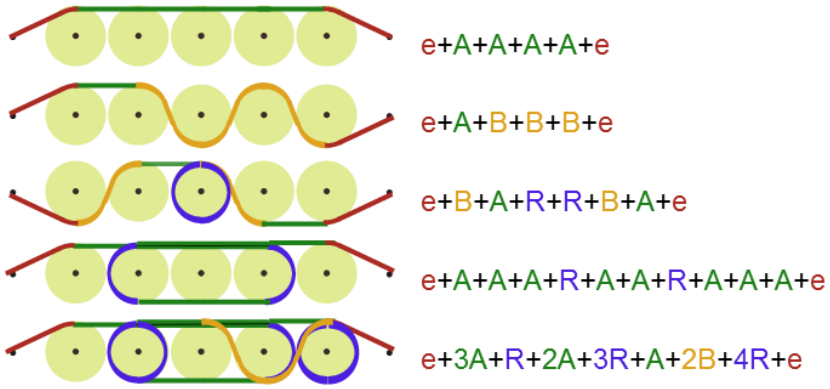


Figure 1

所以，我们需要统计的就是所有满足 $2e + xA + yB + zR \le bound$ 的 (x, y, z) 三元组的方案数。

这样，我们可以设计一个比较暴力的动态规划：用六元组 (x, y, z, p, d, b) 来表示当前用了 x 个 A、y 个 B、z 个 R，在第 p 个滑轮上，方向是 d(0 表示向右，1 表示向左)，在轮子上的位置为 b(0 表示在上方，1 表示在下方)时的方案数。转移显然。

第一点，可以发现方向 d 完全是由 R 的个数 z 决定的，即 $d = z \mod 2$.

其次有：b 的取值不会影响转移，并且两种情况始终是对称的，如图 2。

在注意到 b 的取值无所谓后，可以发现 A 和 B 的转移完全相同，所以在动态规划时，我们可以只记录 $w = x + y$，然后在计算 (x, y, z) 三元组的方案数时，我们需要在动态规划的基础上乘上 $\binom{x+y}{y}$.

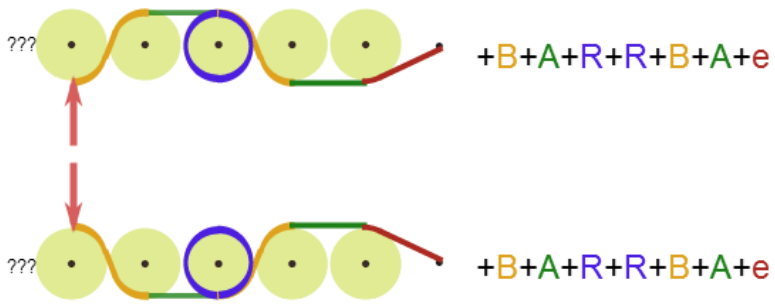


Figure 2

因此，在统计时，可以先枚举 w, z，再枚举 x，如果 (x, y, z) 满足 $2e + xA + yB + zR \leq bound$，就将 (w, z) 的方案数乘上 $\binom{x+y}{y}$ 加入答案。所以 w 枚举的范围不大，枚举到 $O(\log k)$ 约 70 已经足够。

但是由于 z 仍然可能很大，无法记录，考虑进一步优化：在 A 或 B 之后出现多个 R 是没有意义的，因为转两次相当于没转。而去掉的多余的 R，可以在最后乘上一个组合数来分配。即，枚举 w, z, x 之后$(z \leq w)$，计算最多能再加入的 R 的个数，即

$$rest = \frac{bound - (2e + xA + yB + zR)}{2R},$$

将 (w, z) 的方案数乘上 $\binom{x+y}{y}$ 再乘上 $\binom{w+1+rest}{w+1}$[5] 加入答案。

最后一个小细节，在计算组合数 $\binom{n}{m}$ 时$(m \leq 70)$：对于 $m < 4$ 的情况，我们可以 $O(m)$ 暴力计算；而对于 $m \geq 4$ 的情况，我们可以预处理 $n \leq 70000$ 的所有组合数，因为如果 $n > 70000$，则 $\binom{n}{m}$ 一定大于 10^{18}.

至此，问题得到很好的解决。对于二分的上界，可以取 $(n-1)A + 2kR$，时间复杂度为 $O(\log^3 k \log(dn + rk) + k^{\frac{1}{4}} \log k)$.

3 整体二分和权值线段树

3.1 整体二分

在数据结构题中，往往题目要求回答多个询问，比如询问 $[l_i, r_i]$ 区间中第

[5]这个组合数可以使用隔板法得到，也可以直接借助生成函数。

k_i 极值。我们仍然能够使用二分来解决，但是单纯的二分是不够的，这时我们需要使用**整体二分**。整体二分的具体做法请参见许昊然的2013集训队论文[3]。

这里引用它需要满足的性质：

1. 询问的答案具有可二分性；
2. 修改对判定答案的贡献互相独立，修改之间互不影响效果；
3. 修改如果对判定答案有贡献，则贡献为一确定的与判定标准无关的值；
4. 贡献满足交换律、结合律，具有可加性；
5. 题目允许离线算法。

其中在求第 k 优解时，第一条性质已经被定理 2.1 证明。

3.2 使用权值线段树求第 k 优解

整体二分需要满足的性质中，第二条和第五条显得非常苛刻。因为很多问题都需要在线进行，而且修改某个点的权值。对于这种情况，我们可以运用**权值线段树**将问题转化为计数问题，同样花费 $O(\log W)$ 的代价($W = \sup J(U) - \inf J(U)$)。

建以权值为关键字的线段树，在线段树节点上维护 $J(u)$ 在 $[l, r]$ 之间的 u 的数据结构 $T_{l,r}$.

询问时，在 $T_{l,r}$ 中询问 $u \in U_i$ 的解 u 的个数 s (U_i 表示第 i 个询问解的约束条件，例如解在某个区间内)，若 $s \leq k$，则在权值线段树中往右走，否则往左走，并将 k 减去 s.

插入一个新的解 v，就在 $\log W$ 个满足 $l \leq J(v) \leq r$ 的 $T_{l,r}$ 中插入 v. 删除同理。修改一个点的权值就是先删除再加入。

但是此方法不能修改 $u \in U_j$ 的所有 $J(u)$ (U_j 表示第 j 个修改操作中需要修改的解的约束条件)。

可以发现整体二分时，我们也利用了这个模型，但是整体二分是离线地在权值线段树上走，不维护这些数据结构，所以其无法支持可能会影响的修改；但是好处是它可以继续在线段树的节点上套用分治等离线做法，并且节省内存。

3.3 例题三

例3 (BZOJ 3065 带插入区间 k 小值). 开始 n 个数：$a_1, \ldots, a_n$. 执行 q 个操作，操作有三种：询问从左至右第 x 个数到从左至右第 y 个数中，第 k 小的数；将从左至右第 x 个数改为 val；在从左至右第 x 个数之前插入一个值为 val 的数。

要求强制在线。

数据范围：$n \leq 35000$，插入个数 ≤ 35000，修改个数 ≤ 70000，询问个数 ≤ 70000，$0 \leq$ 每时每刻权值 ≤ 70000.

维护权值线段树，每个节点是一棵平衡树 $T_{l,r}$，按照原来顺序维护权值在区间 $[l, r]$ 内的数，如果一个数权值 $\leq m = \left\lfloor \frac{l+r}{2} \right\rfloor$，那么它在平衡树中的特征值为 0，否则特征值为 1.

询问时，首先找到 x 和 y 在 $T_{0,W}$ 中的位置，这样可以得到 x 和 y 之间特征值为 0 的数的个数 s，如果 $s \leq k$，则走到 $T_{l,m}$ 中，否则走到 $T_{m+1,r}$ 中，并将 k 减去 s. 而 x, y 在下一层平衡树中的位置可以通过做类似这样的询问得到：在 x (包含 x)之前(之后)的第一个特征值为 c 的数在下一层平衡树的位置。因此需要维护每个数在权值线段树的每一层的平衡树中的位置。递归直到 $l = r$ 即可。

插入和修改的方法与询问类似。

总复杂度为 $O((n+q) \log W \log n)$. 但与其他复杂度相同的算法相比代码复杂度较低。

3.4 权值线段树的另一种用法

如果换一个思路，将权值线段树作为某一数据结构的点，来维护某一区间的解的权值分布，将会产生一个问题：很多数据结构都需要合并点权，而权值线段树无法高效地合并。我们需要一系列手段来解决这个问题。

对于答案的合并，可以这样做：由于只需要询问第 k 极值，所以我们可以假设已经合并好了，并得到了答案所代表的权值线段树。在权值线段树上走的时候，再去计算当前需要统计的节点的值。

对于标记也是权值线段树的情况：由于不能下传标记了，所以应该将标记永久化，详见例 4。

对于数据结构中需要用子节点更新父节点的情况，如平衡树左旋或右旋之后需要更新时，可以选择重量平衡的数据结构，例如 Treap，替罪羊树等等，暴力重构整棵子树做到期望或均摊较优的复杂度。

3.5 例题四

例4 (ZJOI2013 K 大数查询). 有 n 个位置，m 个操作。操作有两种：在第 a 个位置到第 b 个位置，每个位置加入一个数 c；或者询问从第 a 个位置到第 b 个位置，第 k 大的数是多少。

数据规模：$n, m \leq 50000, |c| \leq n$.

此题做法很多，作者在考场上写了线段树套权值线段树的做法，即对位置维护线段树，线段树上每一个节点维护两棵权值线段树，一棵设为 $C_{l,r}$ 表示当前节点的懒标记。它的意义在于：当前节点对应的区间上每个位置都加了这些权值的数。另一棵设为 $T_{l,r}$ 维护了当前节点对应的区间上所有数的权值。

对于询问操作，如果 $a = l$ 且 $b = r$，那么就直接返回 $T_{l,r}$，否则返回 $[a, b]$ 递归到左儿子和右儿子的结果加上 $(b - a + 1) \cdot C_{l,r}$.

对于修改操作，在 $T_{l,r}$ 中加入 $(b - a + 1)$ 个 c，如果 $a = l$ 且 $b = r$，那么在 $C_{l,r}$ 中加入 c，否则递归到左儿子和右儿子。

时间复杂度 $O(m \log^2 n)$.

值得一提的是，这个做法可以再花费 $O(\log n)$ 的代价来支持区间加一个值，同样也是利用懒标记。

在线段树上多维护一个标记 D，表示当前节点对应的区间全部加了 D. 修改时，一旦进入一个节点，c 就要减去当前节点的 D. 询问时，由于答案是 $\sum s_i(T_i + d_i)$ 的形式(其中 T_i 表示一棵权值线段树)，所以在最后的权值线段树上走的时候需要额外付出 $O(\log n)$ 的代价。复杂度为 $O(m \log^3 n)$.

4 字母树上的统计

信息学竞赛中还有一类题目，它往往要求满足某一条件的字典序第 k 小字符串。

当然对这类问题，我们可以套用二分，将问题转化为满足某一条件且字典序小于某个串的字符串个数。但是这样反而使问题变得复杂，不如平时我们在说的**逐位确定**来得方便、高效。

而逐位确定的实质就是在字母树上走，时刻保持当前子树中所有满足条件的方案个数不小于 k.

每次可以尝试着按字典序从小到大，向当前节点的儿子走，设其儿子对应的子树方案数为 s，如果 $s \geq k$，那么就确定走到该儿子，否则就将 k 减去 s.

而某一节点的子树内满足条件的字符串个数，相当于求前缀确定的满足条件的字符串个数。

不光是字符串，比如求第 k 小的排列，也是利用这种方法来做的。

4.1 例题五

例5. 给一个字符串 str，求其第 k 小子串。两个子串的起始位置不同或结束位置不同均视为不同的子串。

数据范围：$|str| \leq 100\,000$.

建字符串 str 的后缀树，由于后缀树也是字母树，所以可以在后缀树上走得到答案。而后缀树上某一子树内子串个数可以用树形动态规划求得，这里不再赘述。

也可以尝试使用其他做法，但是其本质不变，都是在后缀树上走。如果选手只会后缀数组，可以用后缀数组加单调栈建后缀仙人掌来做，由于后缀仙人掌即为后缀树的“左儿子右兄弟”形式，在本题的实现上非常方便。

4.2 例题六

例6. 对于一个包含 W 个单词的语句，相邻两个单词之间用一个空格隔开，单词只包含小写字母，其信息量为所有字符的信息量之和。空格信息量为 1，‘a’信息量为 2,……，‘z’信息量为 27. 问信息量为 V 且字典序第 k 小的语句。

数据规模：$1 \leq V \leq 1000, 1 \leq W \leq 300, 1 \leq k \leq 10^{18}$.

可以直接套用之前所述，在字母树上走，并询问前缀确定的满足条件的字符串个数。

但是如果每次暴力做动态规划无法通过，因此，我们需要预处理来优化。总复杂度为 $O(VW)$.

5 优先队列在求第 k 优解中的应用

5.1 一个使用堆的例子

首先，引用算法导论中的一道习题：

例7 (Introduction to Algorithms 6.5-8). 请给出一个时间为 $O(n \log m)$、用来将 m 个已排序链表合并为一个排序链表的算法。此处 n 为所有输入链表中元素总数。

做法非常显然。首先，我们将 m 个链表头放入一个小根堆中。进行 n 次操作，第 i 次操作，将堆中最小节点的从堆中取出，作为答案链表的第 i 个，然后在堆中加入该节点在原链表中的下一个。

例8. 给出 n 个数 $a_1, a_2, \ldots, a_n$ 和 m 个数 $b_1, b_2, \ldots, b_m$，问对于所有 $1 \le i \le n, 1 \le j \le m$，$a_i + b_j$ 的第 k 小值。

数据范围：$n, m, k \le 500000$.

将 b 排序，并设 $c_{ij} = a_i + b_j$，那么 $c_1, c_2, \ldots, c_n$ 均为长度为 m 的有序表。运用例 7 的做法，我们执行 k 次操作即可得到前 k 小值。时间复杂度 $O(m \log m + k \log n)$.

当然也可以二分答案来得到，每次枚举 i，维护可行的最大的 j，就可以计算满足 $a_i + b_j \le x$ 的 (i, j) 对数。复杂度为 $O(n \log W)$. 但是二分只能得到第 k 小值，而无法得到前 k 小值。

5.2 使用优先队列求第 k 优解的一般方法

构造解的一张图 P，满足所有解均是图中的一个或多个点，图中每个点都是一个合法的解，且这张图满足堆性质：即如果 u 连向 v，那么 $J(u) \le J(v)$.

新建一个节点 *source*，设其权值为 $-\infty$，从 *source* 向所有没有入度的点连边。那么从 *source* 出发能到达所有解。如图 3，即为例 8 对应的图 P 的其中一种。

用一个优先队列维护已扩展的点，初始只有一个点 *source*，进行 $k+1$ 次操作，第 i 次弹出当前优先队列中的最优解 u，作为第 $i-1$ 优解，将所有 u 到 v 有边，且之前还没被遍历到的 v 放入优先队列中。注意到这张图 P 并不需要

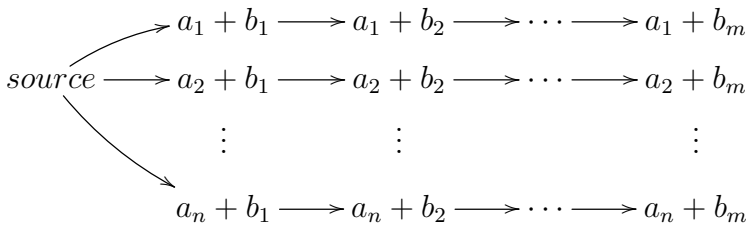


Figure 3

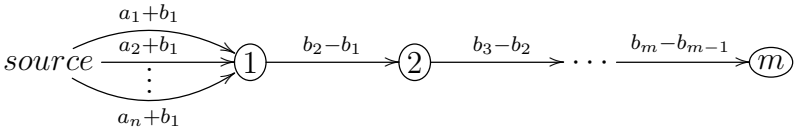


Figure 4

完全构出，可以在需要的时候再建边。要判断是否多次被遍历，可以使用散列表。设最终遍历到的解个数为 $State$，那么整个算法复杂度为 $O(State \log k)$.

注意到图的堆性质，我们可以设 $c(u,v) = J(v) - J(u)$，即 (u,v) 拥有边权 $J(v) - J(u)$. 在边有权值的意义下，对图 P 的定义进行扩展：从起点 $source$ 开始的路径与问题的解**一一对应**，解的值为路径上所有边权和。此时图 P 不一定是拓扑图。图 4 即为例 8 对应的图 P 的另一种。

例 8 还有一个 $O(k \log k + n + m)$ 的做法(如图 5)：首先对 b 建小根堆 h，建堆复杂度为 $O(m)$，详细请参见算法导论[5]；堆中父节点 u 向儿子 v 连边权为 $h_v - h_u$ 的边；$source$ 向堆的根连 n 条边，第 i 条边的边权为 $a_i + \min\{b\}$. 这样建得 P，由于除 $source$ 外所有点的出度不超过 2，对于 $source$，可以只加入权值前 k 小的边，这样复杂度即为 $O(k \log k + n + m)$.

在之后的章节中我们将看到优先队列在求第 k 优解时的应用，以及更多构建 P 的巧妙方法。

6 k 短路算法

定理6.1. *在一张有向带权图 G 中，从起点 s 到终点 t 的可重复经过同一点的不严格递增的第 k 短路的长度，可以在 $O(n \log n + m + k \log k)$ 的复杂度内得到。*

Proof. 对于一条边 e，定义它的边权(长度)为 $\ell(e)$，定义它的起始点为 $head(e)$、终止点为 $tail(e)$，用 $d(u,v)$ 表示 u 到 v 的最短距离。

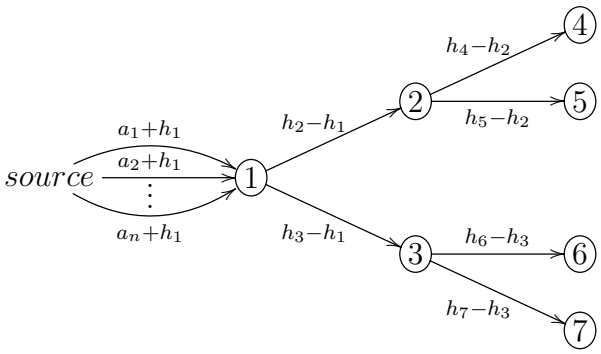


Figure 5

6.1 问题转化

首先定义 T 为 G 中以 t 为终点的最短路径构成的最短路径树。对于一条边$e \in G$，定义：

$$\delta(e) = \ell(e) + d(tail(e), t) - d(head(e), t),$$

那么有：

$$\forall e \in G, \delta(e) \geq 0; \qquad \forall e \in T, \delta(e) = 0.$$

例如图 6，A 为起点，L 为终点；右图中，所有实箭头构成最短路径树 $\delta = 0$，而虚箭头上所标即为该边的 δ 值。

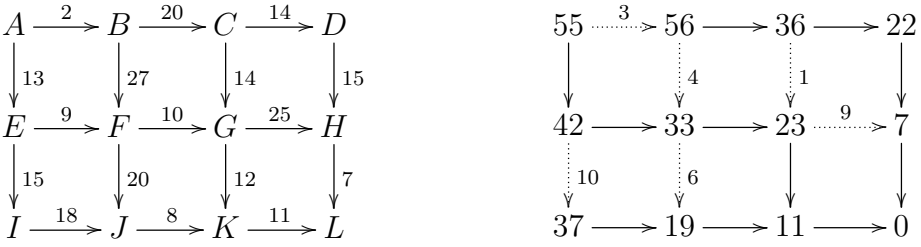


Figure 6

但是最短路径树可能并不一定是一棵树，在此我们要将 T 严格定义为一棵树；对于一个节点 $v \neq t$，定义 $next_T(v)$ 为 $v \to t$ 的最短路径（不唯一则任选一条）上 v 的后继节点。那么 G 中所有的点就以 $next_T$ 形成了一个严格的树结构 T 。

对于一条从 s 到 t 的路径 p，去掉 p 中所有在 T 中的边，将其定义为 $sidetracks(p)$.

对于 $sidetracks$ 我们有以下三条性质：

性质6.1. 对于 $sidetracks(p)$ 中任意一条边 e，都有 $e \in G-T$；对于 $sidetracks(p)$ 中任意相邻的两条边 e, f，都满足 $head(f)$ 是 $tail(e)$ 在 T 上的祖先或 $head(f) = tail(e)$.

性质6.2. p 的路径长度 $l(p) = d(s,t) + \sum_{e \in sidetracks(p)} \delta(e) = d(s,t) + \sum_{e \in p} \delta(e)$.

性质6.3. 对于一个满足性质 6.1的边的序列 q，有且仅有一条 s 到 t 的路径 p 满足 $sidetracks(p) = q$.

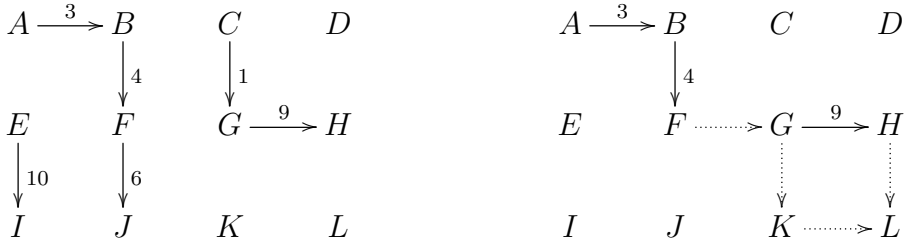


Figure 7

以上三条性质比较显然。例如图 7 中 $A \to B \to F \to G \to H \to L$ 这条路径，它的 $sidetracks$ 为 δ 值为 $3, 4, 9$ 的三条边。它的长度为 $dist(s,t)+3+4+9 = 71$.

至此，我们将问题转化为：求第 k 小的满足性质 6.1 的边的序列。

即，序列中任意一条边 e 都有 $e \in G - T$；任意相邻的两条边 e, f，都满足 $head(f)$ 是 $tail(e)$ 在 T 上的祖先或 $head(f) = tail(e)$.

序列 q 的权值定义为

$$w(q) = \sum_{e \in q} \delta(e).$$

6.2 构建 P

算法一 初始解为空序列。对于一个序列 q，令 q 最后一条边的 $tail$ 为 v (若是空序列则 $v = s$)。在 q 之后加入一条边 e 得到新序列 $q'(head(e)$ 在 T 中 v 到 t 的路径上且 $e \in G - T$)。q 向 q' 连边权为 $\delta(e)$ 的边。如图 8。

这样构成的图 P，每个点的出度最多为 m，所以复杂度为 $O(n\log n + km(\log k + \log m))$[6].

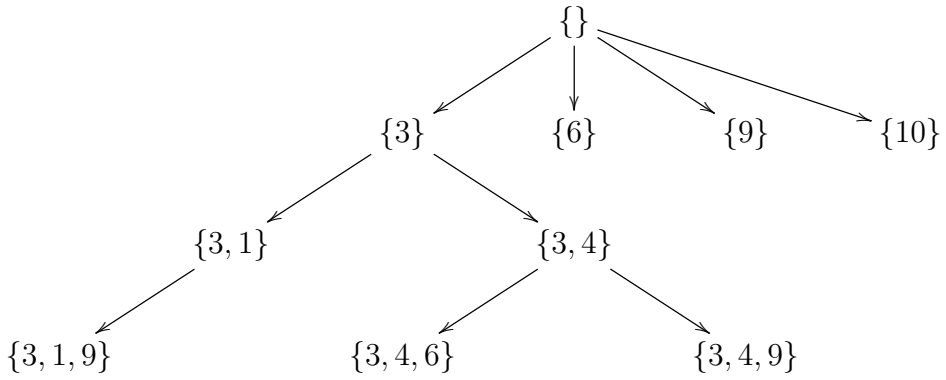


Figure 8

算法二 上图可以对每个点 v 的所有边排序，然后用左儿子有兄弟的方法将每个点的出度减小为 2，兄弟之间的边的边权为两者之差。如图 9。

它的意义在于建立一张有序表 $g(v)$，按权值从小到大记录所有在 $G-T$ 中且满足 $head(e)$ 在 T 中 v 到 t 的路径上的边 e.

对于序列 q，令其最后一条边为 e、$v = tail(e)$、倒数第二条边的 $tail$ 为 u. 将 e 替换为 $g(u)$ 中 e 的后一条边或者在 q 中新加入 $g(v)$ 中的第一条边得到新序列 q'.

复杂度为 $O(n\log n + nm\log m + k\log k)$.

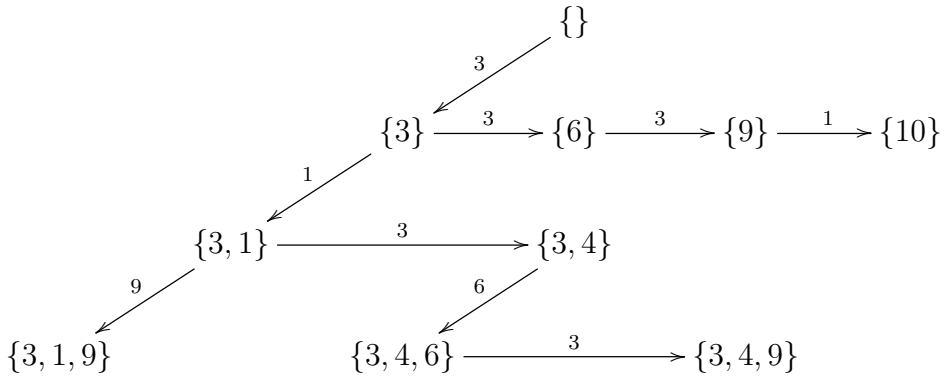


Figure 9

[6]这里默认第一步的最短路算法复杂度为 $O(n\log n + m)$.

算法三 上一个算法的瓶颈在于对每个点建立有序表。注意到 $g(v)$ 和 $g(next_T(v))$ 是有大部分相同的。事实上我们是在 $g(next_T(v))$ 中添上所有在 $G-T$ 的由 v 连出去的边来求得 $g(v)$ 的。但是有碍于有序表的添加复杂度，我们不能有效得来求得 g.

如果我们对于点 v 不建立 $g(v)$ 转而建立一个堆 $H(v)$ (类似例 8 的 $O(k\log k+n+m)$ 做法)，并通过在 $H(next_T(v))$ 中**可持久地**加入所有在 $G-T$ 的由 v 连出去的边来得到 $H(v)$，那么我们就有能解决上面的问题。

这里可持久化的原因是我们时刻需要用到插入之前的堆。

而对于上一算法中“将 e 替换为 $g(u)$ 中 e 的后一条边”这一步，我们直接将 e 在 $H(u)$ 中的两个儿子替换 e 即可。

复杂度为 $O(n\log n+m\log m+k\log k)$.

算法四 例 8 的 $O(k\log k+n+m)$ 做法中，我们用到了建堆的优越性，这里我们也可以利用。

对于点 v 用堆 $h(v)$ 维护所有在 $G-T$ 的由 v 连出去的边。并将 $h(v)$ 中最小元素的权值作为 $h(v)$ 的权值。

把 $h(v)$ 当作一个节点，可持久地插入 $H(next_T(v))$ 得到堆的堆 $H(v)$.

这样建 $h(v)$ 的时间为 $O(m)$，建 $H(v)$ 减少为$O(n\log n)$. 所以总复杂度为 $O(n\log n+m+k\log k)$.

□

关于可持久化堆的实现，作者在冬令营营员讨论[7]上讲解了二叉堆和二项堆的可持久方法，但是作者当时误以为左偏树和斜堆都是均摊复杂度的，所以没有介绍左偏树的可持久化，其实左偏树的复杂度与斜堆不同，是单次操作严格 $O(\log n)$. 所以左偏树是可以可持久化的。

左偏树的可持久化并不困难，设 $\textsc{Merge}(a,b)$ 返回将左偏树 a 和 b 合并得到的左偏树。如果 a 的权值大于 b，那么 $\textsc{Merge}(a,b)$ 返回 $\textsc{Merge}(b,a)$，否则新建一个节点，将其左儿子设为 $a.left$，右儿子设为 $\textsc{Merge}(a.right,b)$，并调整维护左右儿子及其键值，最后返回该节点即可。

7 一类动态规划问题

众所周知，拓扑图的最短路可以通过动态规划来实现。反过来说，如果某一动态规划能转成拓扑图，且它构出的图的边 (a,b) 的意义为 $b\le a+\ell(a,b)(a,b$

均为动态规划时的变量(状态)，对应于动态规划转移方程即为 $b = \min\{b, a + \ell(a, b)\}$)，那么此动态规划的最优解就可以用最短路来求解。

定理7.1. *如果某一动态规划能转成拓扑图，且它构出的图的边 (a, b) 的意义为 $b \leq a + \ell(a, b)$(对应于动态规划转移方程即为 $b = \min\{b, a + \ell(a, b)\}$)，且此动态规划的所有决策方案与该拓扑图中所有起始点到终止点的路径一一对应，那么此动态规划的第 k 优解就可以用 k 短路来求解。*

设该动态规划的状态数为 n，转移数为 m，则其第 k 优解可以在 $O(n \log n + m + k \log k)$ 的复杂度内求出。

Proof. 由于动态规划的初始状态和终止状态可能不止一个，所以其构成的拓扑图中初始点和终止点也可能不止一个，需要新建节点 s, t，从 s 向所有初始点连长度为 0 的边，所有终止点向 t 连长度为 0 的边，此时此动态规划的第 k 优解即为拓扑图上 s 到 t 的 k 短路。

设该动态规划的状态数为 n，转移数为 m，则此时拓扑图上点数为 $O(n)$，边数为 $O(n + m)$，根据定理 6.1 可以得出在此拓扑图上求解 k 短路的复杂度 $O(n \log n + m + k \log k)$. □

如果动态规划转移方程为 $b = \max\{b, a + \ell(a, b)\}$，那么也是可以用 k 短路解的，只需将所有边权取原来的相反数，最后答案也取相反数。这也是求解最长路的方法。

7.1 例题九

例9. *给出 n 个一维上的点 $x_i (1 \leq i \leq n)$，要将这些点划分成两个集合 S_A, S_B. 对于一个集合 $S = \{p_1, p_2, \ldots, p_{|S|}\}$，其中 $p_1 < p_2 < \cdots < p_{|S|}$，设这个集合的代价*

$$\ell(S) = \sum_{k=0}^{|S|-1} \left| x_{p_k} - x_{p_{k+1}} \right|.$$

其中默认 $x_{p_0} = 0$.

现在要求所有方案中第 k 小的

$$\ell(S_A) + \ell(S_B).$$

数据范围：$n \leq 10000, k \leq 500000$.

解法一

如果 k 等于 1，那么我们可以设计动态规划来解决这个问题：

用 $f(i,j)$ 来表示两个集合的末尾分别为 i 和 j 时的最小代价。这里，可以默认 $i \geq j$，因为 $f(i,j) = f(j,i)$.

转移时，枚举 $i+1$ 会放到哪个集合中。即

$$f(i,j) + |x_{i+1} - x_i| \to f(i+1,j) \quad [7]$$
$$f(i,j) + |x_{i+1} - x_j| \to f(i+1,i).$$

初始化 $f(0,0) = 0$，其他均为 ∞. 答案等于 $\min_{j<n} f(n,j)$.

这样，我们就可以通过定理 7.1 来建拓扑图，然后用 k 短路来得到第 k 优解，由于图的点数和边数均为 $O(n^2)$，复杂度为 $O(n^2 \log n + k \log k)$.

容易看出 $f(i+1,j) - f(i,j)$ 只有当 $j = i$ 时不确定，对于其他的 $j \neq i$ 都有

$$f(i+1,j) = f(i,j) + |x_{i+1} - x_i|.$$

而对于 $f(i+1,i)$ 我们可以写成

$$f(i+1,i) = \min\{f(i,j) + |x_{i+1} - x_j| - |x_{i+1} - x_i|\} + |x_{i+1} - x_i|.$$

如果我们求出 $M_{i+1} = \min\{f(i,j) + |x_{i+1} - x_j|\}$，就可以这样转移：$M_{i+1} - |x_{i+1} - x_i| \to f(i,i)$，然后将整个 $f(i)$ 加上 $|x_{i+1} - x_i|$ 得到 $f(i+1)$.

这样我们优化点数至 $O(n)$，并且仍然满足定理 7.1 的所有条件，所以复杂度为 $O(n \log n + n^2 + k \log k)$.

解法二

可以发现，求 M_i，可以用线段树来做：设 $g(i,x_j) = f(i,j) - x_j, h(i,x_j) = f(i,j) + x_j$，那么有

$$M_i = \min\left\{x_i + \min_{j<x_i} g(i,j), \min_{j>x_i} h(i,j) - x_i\right\}$$

这样一来，每次只会在线段树中新建 $\log n$ 个节点(需要离散)，边数也优化至 $O(n \log n)$，复杂度为 $O(n \log^2 n + k \log k)$.

[7]这里 $a + x \to b$ 表示用 $a + x$ 去更新 b 这个变量的值，等同于 $b = \min\{a + x, b\}$.

但是这并非能做到的最好复杂度，可以这样做：在线段树上，每个节点的儿子个数增加为 d，也就是说，我们不再平分一段区间，而是将它 d 等分，这样修改时，修改的点数会减少至 $O(\log_d n)$，而询问时的区间个数会增加至 $O(d\log_d n)$. 这样，我们的点数为 $O(n\log_d n)$，而边数为 $O(nd\log_d n)$，总复杂度就是

$$O\left(\frac{n\log^2 n}{\log d}+\frac{nd\log n}{\log d}+k\log k\right).$$

如果取 $d=\log n$，那么复杂度为

$$O\left(\frac{n\log^2 n}{\log\log n}+k\log k\right).$$

其实 d 取 $O(\log n)$ 并非最优，可以证明再将 d 除一个朗伯 W 函数[8]，即 $d=O\left(\frac{\log n}{W(O(\log n))}\right)$ 时取到最优，详细请见[8]。

8 k 小生成树

8.1 最小生成树

Prim 和 Kruskal 算法是信息学竞赛中喜闻乐见的最小生成树做法，其复杂度分别为 $O(n\log n+m)$ 和 $O(m\log n+m\alpha(n))$.

但是也有一些比两者更优的算法，可以做到期望线性，或者最坏 $O(m\alpha(m,n))$.

8.2 次小生成树

首先引用唐文斌在冬令营 2012 上的讲课：

- 先求出最小生成树 MST；
- 然后枚举不在 MST 上的边 (u,v)，若将 (u,v) 替换掉 MST 上节点u 与节点 v 之间权值最大的边，那么得到的生成树的权值为 $w(MST)+w(u,v)-maxw(u,v)$;
- 取最小值即得到次小生成树。

[8]朗伯 W 函数满足：$z=W(z)e^{W(z)}$.

意思就是说，我们使用一条非树边去替换最小生成树上的树边，并使其仍是生成树。次小生成树一定是这些生成树之一。具体的证明请见下文。

而 $maxw(u,v)$ 可以使用倍增在 $O(m\log n)$ 的复杂度内求得。所以求次小生成树的复杂度为 $O(m\log n)$.

8.3 k 小生成树

这里先给出一个 $O(m\log n+k^2)$ 的做法。

8.3.1 收缩必要边

对于一张无向图 G 和 G 上的一条边 $e=(x,y)$，定义收缩图 $G\cdot e$ 为点集为 $V(G)-y$、边集为 $c_e(E(G))$ 的图。其中函数 c_e 表示去掉所有 (x,y) 边，并将所有 (y,z) 边改为 (x,z).

引理8.1. *T 为 G 的一棵生成树，e 是 T 上的任意一条边，$c_e(T)$ 是 $G\cdot e$ 的最小生成树当且仅当 T 是 G 的最小的包含 e 的生成树。*

Proof. 可以直接根据定义得出。 □

引理8.2. *v 是 G 中任意一个点，e 是所有与 v 相连的边中最小的一条。令 T 为 $G\cdot e$ 的最小生成树，那么 $T+e$ 即为 G 的最小生成树。*

Proof. 令 $e=(u,v)$，若 e 不在 G 的最小生成树中，那么 v 到 u 必有路径，设其与 v 相连的边为 f，那么 $\ell(f)\geq\ell(e)$，由于将 f 换成 e 之后仍为生成树，所以 e 必在 G 的其中一棵最小生成树中。

根据引理 8.1，$T+e$ 为 G 的最小的包含 e 的生成树，所以 $T+e$ 不大于 G 的最小生成树，所以 $T+e$ 是 G 的最小生成树。 □

对于一条生成树上的边 e，它将生成树分成 T_1,T_2，定义 $r_G(e)$ 为除 e 以外最小的跨越 T_1 和 T_2 的边。

引理8.3. *对于任意一条最小生成树 T 上的边 e，如果 $G-e$ 连通，那么 $T-e+r_G(e)$ 是 $G-e$ 的最小生成树。*

Proof. 可以通过对点数数学归纳得出。每次收缩一个叶节点。 □

引理8.4. 给出 G 的最小生成树 T，所有 T 上的边 e 的 $r_G(e)$ 能在 $O(m\log n)$ 的时间复杂度内计算得到。

Proof. 枚举所有非树边 $e=(u,v)$，在 T 上 u 到 v 的路径覆盖 e，最后求每条树边上被覆盖的最小非树边即可。

可以暴力使用动态树来做，也可以将所有非树边按边长从小到大排序，暴力覆盖路径，并将已覆盖的路径使用并查集合并，每次跳到已覆盖的最高点。 □

引理8.5. 给出 G 的最小生成树和所有树边的 r_G，我们可以在线性时间内得出 $n-k$ 条一定在 k 小生成树中的边。

Proof. 对于一条树边 e，令 $\ell'(e)=\ell(r_G(e))-\ell(e)$. 换句话说，$\ell'$ 即为将 e 删掉对图的最小生成树的额外代价。然后，我们可以在 $O(n)$ 时间内得到前 $k-1$ 小的 ℓ'，设其余的树边的集合为 S.

对于 S 中任意一条边 e，至少存在 $k-1$ 条边 e' 满足 $\ell(T-e'+r_G(e'))\le\ell(T-e+r_G(e))$，所以至少有 k 棵生成树大小不劣于 $\ell(T-e+r_G(e))$. 根据引理 8.3 得到，至少有 k 棵生成树大小不劣于任意一棵不包含边 e 的生成树。所以 e 必在 k 小生成树中。 □

引理8.6. 给出图 G 的最小生成树 T，在 $O(m\log n)$ 的时间内，我们可以找到一个边集 S 和一张 k 个点的图 G'，使得 G 的 k 小生成树恰好为 G' 的 k 小生成树加上边集 S.

Proof. 令 S 为引理 8.5 中所建的边集，令 $G'=G\cdot S$，即 G 收缩 S 中所有的边。 □

8.3.2 去掉无用边

与 $r_G(e)$ 类似的，对非树边 $e=(u,v)$，定义 $R_G(e)$ 为树上 u 到 v 路径上的最大边。

引理8.7. 如果 T 是图 G 的最小生成树，那么 $c_e(T-R_G(e))$ 为 $G\cdot e$ 的最小生成树。

Proof. 对点数进行数学归纳。每次收缩一条非 $R_G(e)$ 的树边，根据引理 8.1 可得。 □

同样，R_G 也可以被快速算出。

引理8.8. *给出 G 的最小生成树 T，所有非树边 e 的替换边 $R_G(e)$，能在 $O(m\log n)$ 的复杂度内求出。*

Proof. 即求树上 u 到 v 路径最大值，使用倍增在 $O(m\log n)$ 的复杂度内得到。 □

引理8.9. *给出 G 的最小生成树和所有非树边的 R_G，我们可以在线性时间内得出 $m-n-k$ 条一定不在在 k 小生成树中的非树边边。*

Proof. 定义 $L(e)$ 为 $\ell(e)-\ell(R_G(e))$. 与之前类似，令 f 为所有非树边中 L 第 $k-1$ 小的边。对于任意一条边 e，若 $L(e)>L(f)$，那么至少有 k 棵生成树的大小不超过 $T-R_G(e)+e$，因此也不超过任意包含边 e 的生成树。所以 e 一定不在 G 的 k 小生成树中。 □

这样，我们就将 n 减少至 k，m 减少至 $2k-2$，所以之后，我们默认 $n\le k, m\le k-1$.

对于 $k=2$ 的情况，n,m 均可减少至 2，所以之前次小生成树的做法正确性可以直接推得。

8.3.3 构建 P

定义 P 中一个节点为 R，它由三部分组成：(生成树 T，强制在生成树中的边集 I，强制不在生成树中的边集 X)。

首先给出第一种建 P 的方法(如果对 P 的定义有些遗忘或模糊，请回顾 5.2 节)：

起始点 *source* 为 $R_1=(T_1,\varnothing,\varnothing)$，$T_1$ 为 G 的最小生成树。

对于一个 $R_i=(T_i,I_i,X_i)$，枚举下一次进行的替换，$T_i-e+r_{G-X_i}(e)$，其中 $e\in T_i-I_i$. 对这些替换按长度进行排序，设排序后被替换边分别为 $e_1,e_2,\cdots\cdots$ 那么 R_i 扩展出节点

$$(T_i-e_j+r_{G-X_i}(e_j), I_i+e_1+e_2+\cdots+e_{j-1}, X_i+e_j).$$

根据 P 的定义，需要证明所有 R_i 与所有可行的生成树一一对应，且图满足堆性质。

首先证明所有 R 与所有可行的生成树一一对应：

Proof. 令 R 对应其定义中的生成树 T. 现只需证明一棵生成树有且仅有一个 R 与之对应。

先证明一棵生成树 T 至少存在一个 R 与之对应：设 R_i 为当前的生成树，满足 $I_i \subseteq T$ 且 $X_i \cap T = \varnothing$. 一开始 $i = 1$，即 $R_i = R_1 = (T_1, \varnothing, \varnothing)$，满足条件。如果 $T = T_i$，那么我们就找到了 R_i 与 T 对应，否则由于 $T \neq T_i$，所以 R_i 中至少有一个儿子 R_j 满足 $I_j \subseteq T$ 且 $X_j \cap T = \varnothing$，将 R_i 变为 R_j 重复之前即可。由于 $|I_i| + |X_i|$ 每次至少增加 1，所以这个过程不会无限制地进行下去，所以一定能找到一个 R 与 T 对应。

再证明一棵生成树 T 不会与多个 R 相对应，即每个 R 对应的 T 互不相同：对于一个节点 R，设其儿子按顺序分别为 $R_1, R_2, \cdots\cdots$ 那么由于 $X_i = X + e_i$ 且 $e_i \in T$，所以 T 与 R_i 及其子树内所有点对应的生成树均不相同。对于节点 R_i, R_j，假设 $i < j$，则 $e_i \in X_i$，$e_i \in I_j$，所以 R_i 及其子树内所有点的生成树与 R_j 及其子树内所有点的生成树不相同。对 $|I_i| + |X_i|$ 进行数学归纳，通过 R_i 为根的子树内所有节点的生成树互不相同，加上之前的即可得出 R 为根的子树内所有节点的生成树互不相同。 □

再证明图满足堆性质：

Proof. 对于 I_i 中的所有边，我们视其边权为 $-\infty$；同理对于 X_i 中的所有边，视其边权为 $+\infty$. 那么对于 P 中节点 R，满足 T 是改边权之后图的最小生成树。

数学归纳证明，首先 R_1 满足。考虑 P 中节点 R_i，若 R_i 满足，那么对于 $I_i + e_1 + e_2 + \cdots + e_{j-1}$，$T_i - e_j + r_G(e_j)$ 为次小生成树，所以对于 $I_i + e_1 + e_2 + \cdots + e_{j-1}, X_i + e_j$，$T_i - e_j + r_G(e_j)$ 为最小生成树，所以 R_j 满足。

由于随着边集 I 增大，最小生成树不会变优，而 X 增大会使最小生成树变劣，因此图 P 满足堆性质。 □

第二种方法是对第一种方法的优化，也就是使用左儿子右兄弟的形式来减少单个点的出度，详细见下文。

8.3.4 算法流程

首先，求出最小生成树，设其为 T_1，令 $R_1 = (T_1, \varnothing, \varnothing)$. 如图 10 中，实线部分为最小生成树，虚线部分为非树边。

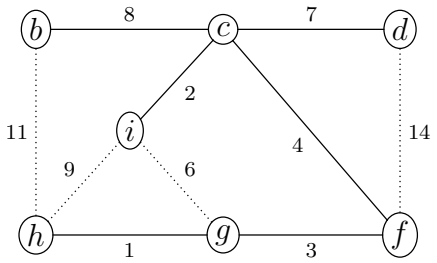


Figure 10

假设现在已经求出前 k 优解：$R_1, R_2, \ldots, R_k$. 找一个最小的替换 (i, e, f)，即 $T_i - e + f$ 仍然是一棵树且 $\ell(T_i - e + f)$ 在所有替换中是最小的，其中 e, f 需要满足 $e \notin I_i$ 且 $f \notin X_i$.

那么 $R_{k+1} = (T_i - e + f, I_i, X_i + e)$，并将 R_i 改为 $(T_i, I_i + e, X_i)$. 如图 11，将 (c, f) 替换成 (i, g)，那么 T_2 中 (c, f) 就被列入 X 集合无法再出现在树中，T_1 中 (c, f) 将强制在树中，所以我们用两条线来表示。

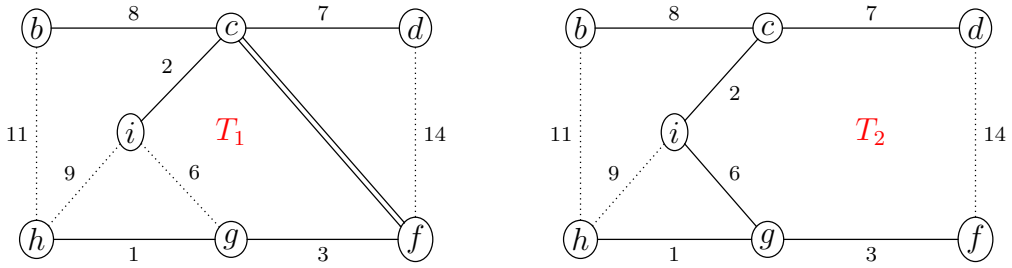


Figure 11

下一步，如图 12，我们将把 T_1 中的 (b, c) 替换成 (b, h) 得到 T_3.

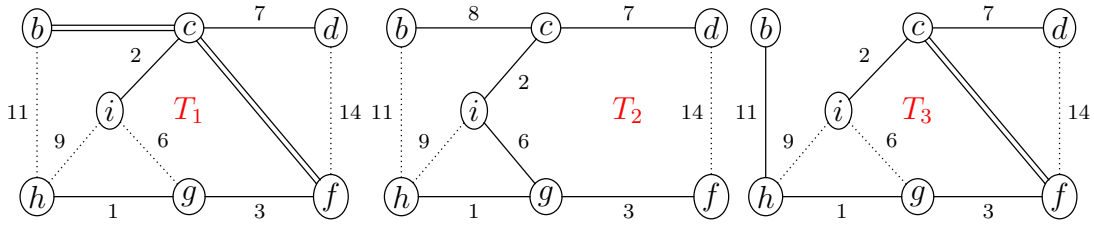


Figure 12

再下一步，如图 13，我们将把 T_1 中的 (g, f) 替换成 (i, g) 得到 T_4.

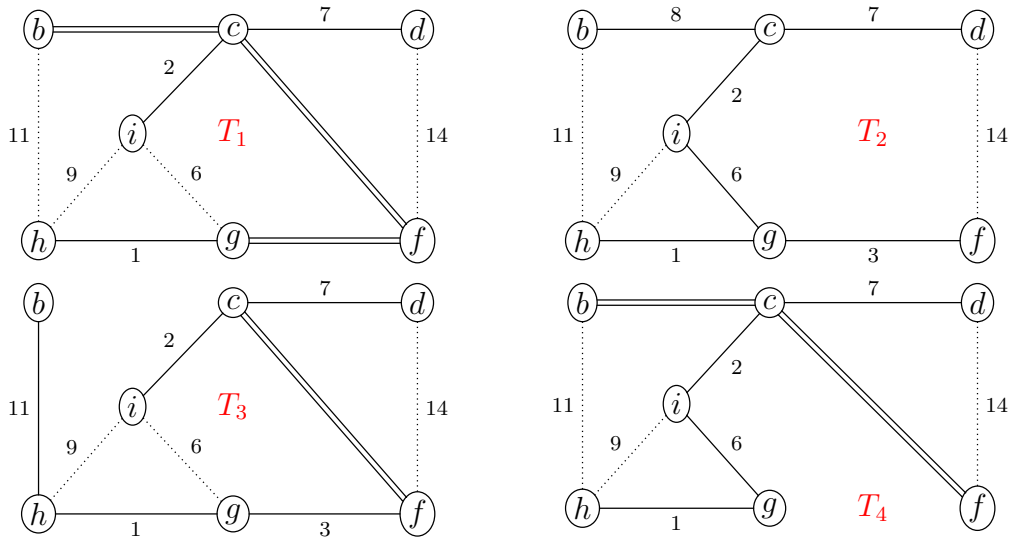


Figure 13

现在问题是：如何快速的得到最小的替换。有两种方法：

第一种：维护所有树边的 r_G 值。如果暴力维护，根据引理 8.4 可知复杂度为 $O(km\log n)$. 可以预先将所有边排序，将其优化至 $O(km\alpha(n))$. 但是无法做到 $O(km)$.

第二种：维护所有非树边的 R_G 值。如果暴力维护，根据引理 8.8 可知复杂度为 $O(km\log n)$，现将其优化至 $O(km)$.

I 集合增加一条边 e 时，只有 u 到 v 树上路径经过 e 的非树边 (u,v) 的 R_G 值才会改变。因此假设以 e 的某个端点为根，那么只有最近公共祖先为根的路径的 R_G 值才会改变。要判断最近公共祖先是否为根，只需把根删掉，判断是否联通即可；如果已知最近公共祖先为根，那么 R_G 值的计算就非常简单了。

将 e 换成 f，并在 X 集合中加入 e 的做法如下：

设 $e=(u_0,v_0)$，去掉 e，树被划分成 T_1,T_2，设 u_0 在 T_1 中，v_0 在 T_2 中。设在 T_1 中 u_0 到 f 的路径为 $u_0,u_1,\ldots,u_p$，在 T_2 中 v_0 到 f 的路径为 $v_0,v_1,\ldots,v_q$，其中 $f=(u_p,v_q)$.

如果将 e 换成 f，那么只有横跨 T_1,T_2 的非树边在树上的路径才会改变。更具体地，横跨 T_1,T_2 的非树边边在树上的路径只改变在环 $u_0\to u_1\to\cdots\to u_p\to v_q\to\cdots\to v_1\to v_0\to u_0$ 上的部分(实际上是取反)。所以，只需更改所有横跨 T_1,T_2 的非树边的 R_G 值。

对树进行一次遍历，求出每个点距离它最近的环上的点。对于一条非树边 $e=(u,v)$，设距离 u,v 最近的环上的点分别为 u',v'，那么 $R_G(e)$ 将由五部分构成：$u\sim u'$, $u'\sim u_p$, f, $v_q\sim v'$, $v'\sim v$. 这些都可以通过维护前缀信息在 $O(n+m)$ 时间内得到。所以复杂度为 $O(km)$.

所以我们有：

定理8.1. *k 小生成树能在时间复杂度 $O(m\log n+k^2)$ 复杂度内求出。*

Proof. 最小生成树能在该复杂度内完成。使用引理 8.6 在该复杂度内将点数和边数都减少为 $O(k)$，再根据本节中的方法，在 $O(k^2)$ 内得出 k 小生成树。 □

最后偷偷说一句：如果存在一种数据结构能支持快速并可持久地更换树边、更改边权，并维护树上最小的替换，那么就可以在该数据结构 $O(k)$ 次操作的复杂度内得出 k 小生成树。在 [15] 中就有这个做法，它使用的数据结构是可持久化 topology tree.

9 k 小简单路径

k 小简单路径，是求在一张有向带权图 G 中，从起点 s 到终点 t 的**不**可重复经过同一点的不严格递增的第 k 短路。下面我们将默认图 G 不包含负环。

虽然它与 k 短路的定义只相差了一个“不”字，但是求解 k 小简单路径与求解 k 短路的方法相差很大，反而与 k 小生成树的解法有异曲同工之妙，因此作者选择在 k 小生成树之后提出 k 小简单路径的做法。

9.1 构建 P

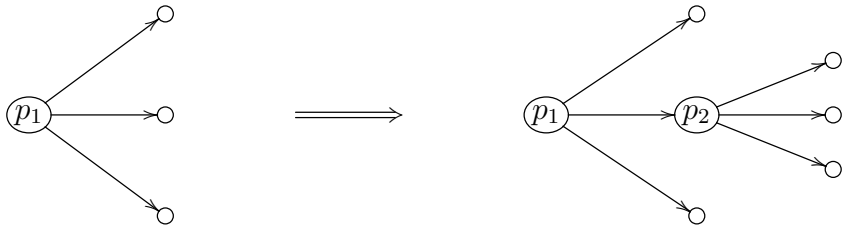
分析之前 k 小生成树的构造，P 是一棵树，P 中节点 v 始终优于所有在其子树内的点。

换一个角度，就可以这样理解：

- 初始有一个集合 $\mathcal{T}$ 表示所有生成树的全集；
- 开始找到 $\mathcal{T}$ 中的最优解，即最小生成树 T_1；
- 将 $\mathcal{T}-\{T_1\}$ 分成 m 个集合 $\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_m$，使得这 m 个集合中每个集合的最优解容易求出；

- 这样得到次小生成树 T_2，假设 $T_2 \in \mathcal{T}_j$，那么将 $\mathcal{T}_j - \{T_2\}$ 分成若干集合与 $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{j-1}, \mathcal{T}_{j+1}, \ldots, \mathcal{T}_m$ 一起继续之前操作得到第三小生成树 T_3……

所以类似的，可以这样构建 k 小简单路径的图 P：



- 初始有一个集合 $\mathcal{P}$ 表示所有 s 到 t 的简单路径的全集；
- 开始找到 $\mathcal{P}$ 中的最优解，即 s 到 t 的最短路 p_1；
- 将 $\mathcal{P} - \{p_1\}$ 分成 m 个集合 $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m$，使得这 m 个集合中每个集合的最优解容易求出；
- 这样得到次小简单路径 p_2，假设 $p_2 \in \mathcal{P}_j$，那么将 $\mathcal{P}_j - \{p_2\}$ 分成若干集合与 $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{j-1}, \mathcal{P}_{j+1}, \ldots, \mathcal{P}_m$ 一起继续之前操作得到第三小生成树 p_3 ……

现在的问题就是如何将某个集合进行划分，使得划分之后所得的集合的最优解容易求出。在 k 小生成树中，我们使用了强制包含和强制排除的方法，在求 k 小简单路径时也是相同的：

令图 P 中一个节点 R 由三部分组成：(当前简单路径 p，强制包含的边集 I，强制不包含的边集 X)。其中令 $p = (v_1 = s, v_2, \ldots, v_{|p|-1}, v_{|p|} = t)$，保证 $I = \{(s, v_2), (v_2, v_3), \ldots, (v_{q-1}, v_q)\}$ 且 $q \leq |p| - 1$，X 中全是从 v_q 出发的边。

R 共扩展出 $|p| - q$ 个点，设其为 $R_i = (p_i, I_i, X_i)$，那么有

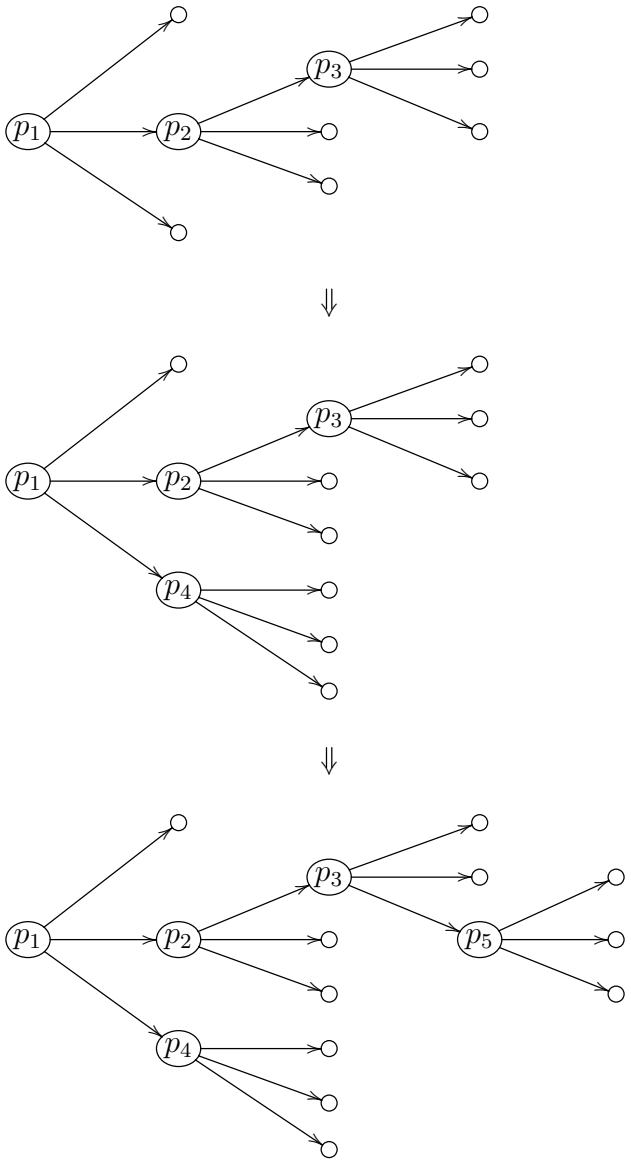
$$I_1 = I, X_1 = X + \{(v_q, v_{q+1})\},$$

对于 $i > 1$ 有

$$I_i = I_{i-1} + \{(v_{q+i-2}, v_{q+i-1})\}, X_i = \{(v_{q+i-1}, v_{q+i})\},$$

p_i 是条件 I_i, X_i 下的最短简单路径。

很可惜，我们无法像 k 小生成树那样使用左儿子右兄弟的形式来减少单个节点的出度，因为我们无法做到快速地判断兄弟之间的大小关系。



9.2 算法流程

首先，如果所有边权非负，就使用 Dijkstra 算法求出 s 到 t 的最短路；否则使用 Bellman-Ford 或 SPFA 算法求出 s 到所有点的最短路，将 (u,v) 的边权 $\ell(u,v)$ 替换为 $\ell(u,v)+dist(s,u)-dist(s,v)$，类似于之前 k 短路算法中的 δ. 此时 s 到 t 的任意一条路径长度需要加上原先 s 到 t 的最短路，与性质 6.2 类似。将 (s 到 t 的最短路,$\varnothing,\varnothing$) 加入优先队列中。

每次从优先队列中弹出最小的点 $R = (p, I, X)$，并将它能扩展出的点 $R_i = (p_i, I_i, X_i)$ 全部加入优先队列中。

条件 I_i, X_i 下的最短简单路径 p_i 可以通过 Dijkstra 算法在 $O(n \log n + m)$ 复杂度内得到。由于 k 个点最多扩展出 kn 个点，所以需要计算 kn 次最短路，所以复杂度为 $kn(n \log n + m)$.

对于优先队列的实现，可以直接使用堆，时间复杂度为 $O(kn \log kn)$，空间复杂度可以优化到 $O(kn)$.

因此我们有

定理9.1. *在一张有向带权图 G 中，从起点 s 到终点 t 的不可重复经过同一点的不严格递增的第 k 短路的长度，可以在 $O(kn(m + n \log n))$ 的复杂度内得到。*

10 感谢

感谢中国计算机学会提供学习和交流的平台。

感谢绍兴一中的陈合力老师、邵红祥老师、游光辉老师、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胡伟栋和余林韵的指导。

感谢清华大学的徐捷、鲁逸沁、裘捷中、梁佳文和上海交通大学的蒋舜宁学长对我的帮助。

感谢绍兴一中的何奇正、郑钟屹、徐正杰同学对我的帮助和启发。

感谢绍兴一中的董宏华、张恒捷、王鉴浩、郭雨等同学对我的帮助和启发。

感谢镇海中学的杜瑜皓同学为本文审稿。

感谢其他对我有过帮助和启发的老师和同学。

感谢我的父母二十年如一日无微不至的关心和照顾。

参考文献

[1] 维基百科，最优化问题。

[2] Wheeler，PulleyTautLine，TopCoder Algorithm Problem Set Analysis.

[3] 许昊然，浅谈数据结构题的几个非经典解法，2013 集训队论文。

[4] 吕凯风，带插入区间 K 小值系列题解。

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

[6] D. Eppstein. Finding the k shortest paths. *SIAM J. Computing*, 28(2):652–673, 1998.

[7] 俞鼎力，堆的可持久化和 k 短路，2014 冬令营营员讨论。

[8] 俞鼎力，无聊的邮递员解题报告，2014 集训队第二次作业。

[9] WIKIPEDIA，Prim's algorithm.

[10] WIKIPEDIA，Kruskal's algorithm.

[11] WIKIPEDIA，Minimum spanning tree.

[12] 唐文斌，图论专题之生成树，2012 冬令营讲课。

[13] 刘汝佳，黄亮，算法艺术与信息学竞赛。

[14] D. Eppstein. Finding the k smallest spanning trees. *BIT* 32 (2): 237 - 248.

[15] Frederickson, Greg N. Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees, *SIAM J. Computing*, 26(2):484 - 538, 1997.

[16] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.*

[17] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17:712-716, 1971.