2021“MINIEYE 杯”中国大学生算法设计超级联赛（3）题解

By Claris

2021 年 7 月 27 日

1 Bookshop

Shortest judge solution: 3945 Bytes.

对树进行轻重链剖分并求出 DFS 序，在 DFS 的过程中先 DFS 一个点的重儿子，再 DFS 它的轻儿子们，那么每条重链在 DFS 序上是连续的。对于每个询问 (x, y, z)，令 t 为 x 和 y 的 LCA，那么从 x 出发走到 t 的过程在 DFS 序里对应从右往左的 $O(\log n)$ 个区间，从 t 出发走到 y 的过程也对应 DFS 序里从左往右的 $O(\log n)$ 个区间。考虑离线询问，把所有询问分成两个步骤来统一处理：

- 对于每个询问，处理从 x 往上走到 LCA 的过程，即从右往左拆成 $O(\log n)$ 个区间。
- 对于每个询问，处理从 LCA 往下走到 y 的过程，即从左往右拆成 $O(\log n)$ 个区间。

上述两个步骤是类似的，以第一步为例，可以维护一个数据结构 T，从右往左扫描整棵树的 DFS 序，假设扫到了 DFS 序的第 i 个位置，那么需要依次处理以下几类事件：

- 对于每个询问拆出来的右端点为 i 的区间，将对应询问插入数据结构 T 中。
- 令第 i 个位置代表的点为 x，则需要将数据结构 T 中所有 z 值至少为 p_x 的询问对应的 z 值都减去 p_x。
- 对于每个询问拆出来的左端点为 i 的区间，将对应询问从数据结构 T 中删除。

因此我们需要维护一个数据结构 T，支持插入数字、删除数字、将所有至少为 k 的数字都减去 k。考虑用平衡树 T 按 z 从小到大维护所有询问，那么插入删除是基本功能，将所有至少为 k 的 z 值都减去 k 时：

- z 值在 $[0, k)$ 的数不需要改动。
- z 值在 $[k, 2k)$ 的数需要减去 k，其值减少至少一半，可以暴力修改，每个询问总计会被修改 $O(\log z)$ 次。
- z 值在 $[2k, +\infty)$ 的数需要减去 k，其相对排名不会发生改变，可以打标记实现。

因此最多会有 $O(n + q \log n + q \log z)$ 次平衡树操作，每次操作时间复杂度为 $O(\log q)$，总时间复杂度为 $O((n + q \log n + q \log z) \log q)$。

2 Destinations

Shortest judge solution: 1910 Bytes.

每个人有三条可行的路线，第 j 条起点为 s_i，终点为 $e_{i,j}$，代价为 $c_{i,j}$，容易发现这三条路线存在公共点（起点），因此与下述问题等价：给定 $3m$ 条链，选择 m 条没有公共点的链，使得代价之和最小。注意到最多只能选择 m 条公共点的链，因此这又等价于选择最多条没有公共点的链，在此基础之上需要最小化代价之和。对于一条链，将其收益设置为 $10^6(m+1)-c_{i,j}$，则该收益值在 10^6m 进制下为 $(1, 10^6-c_{i,j})$，总收益不会发生进位。我们的目标是最大化总收益，如果总收益在 10^6m 进制下的高位等于 m 则有解，对应的代价和为 10^6m 减去总收益 10^6m 进制下的低位。

现在问题转化为：给定若干条树链，选择总收益最大的一些链使得两两没有公共点。这是经典问题，一个简单的解法是考虑其对偶问题：在树上选择尽量少的点，每个点可以重复选多次，满足每条树链上选择的点数至少为其收益值。那么对偶问题可以贪心解决：从深到浅 DFS 这棵树，在考虑 x 点时，处理所有 LCA 为 x 的链 (u, v, w)，统计 u 到 v 路径上选择的点数 cnt，若不足 w，则在 LCA（即 x 点）处补充 $w-cnt$ 个点。因此需要一个数据结构支持单点修改、查询链和，令 f_x 表示 x 到根路径上的点权和，那么单点修改对 f 的影响是子树加，可以通过树状数组维护差分实现。

时间复杂度 $O((n+m)\log n)$。

3 Forgiving Matching

Shortest judge solution: 2114 Bytes.

对于 S 的每个长度为 m 的子串，统计其与 T 匹配的位置数 f_i，即可得到该子串被认为匹配的最小的 k 值。两个字符匹配当且仅当它们字符相等，或者至少有一个是通配符。假设没有通配符的存在，枚举 0 到 9 每个字符 c，那么如果 $S_i = T_j = c$，则 f_{i-j} 应该加上 1，可以翻转 T 串后通过 FFT 求出 f。

现在考虑通配符的影响，S 一个子串与 T 通过通配符匹配的位置数 $= S$ 对应子串中通配符的数量 $+T$ 中通配符的数量 $-$ 对应位置都是通配符的位置数量。S 对应子串中通配符的数量可以使用前缀和求得，对应位置都是通配符的位置数量同样可以通过 FFT 求得。

时间复杂度 $O(11n\log n)$。

4 Game on Plane

Shortest judge solution: 872 Bytes.

两条直线存在公共点当且仅当它们重合或者它们斜率不同，因此 Bob 的最优策略一定是避开斜率出现次数最多的那些直线。Alice 为了让 Bob 与尽量多的直线相交，最优策略就是最小化斜率出现次数的最大值，所以不断从每种斜率的直线中各选一种即可。

时间复杂度 $O(n\log n)$。

5 Kart Race

Shortest judge solution: 2612 Bytes.

从 1 号点开始 DFS 整个图，并把出栈序列记下来，那么若 x 能到达 y，显然 x 晚于 y 出栈。因为图是平面图，考虑最有代表性的两种遍历方式：顺时针遍历和逆时针遍历，那么可以得到两个出栈序列。设 a_i 表示顺时针遍历图时 i 点的出栈序，b_i 表示逆时针遍历图时 i 点的出栈序，那么 x 能到达 y 当且仅当 $a_x > a_y$ 且 $b_x > b_y$。

按照题意，选出来的点应当满足两两不可达，因此把 a_i 看作横坐标，b_i 看作纵坐标，那么选了 (a_i, b_i) 就不能选它左下角以及右上角的所有点，因此选出来的点一定满足从左往右 a 递增且 b 递减，问题转化为求价值和最大的下降子序列，可以用树状数组优化朴素 DP 在 $O(n \log n)$ 的时间内得到最优解。

对于字典序最小最优解的求解，有一个方法是在 DP 值里直接用长度为 n 的 01 串来记录当前方案里选了哪些点，转移的时候需要支持字典序大小的比较、拷贝以及把单点从 0 修改成 1。因此考虑使用可持久线段树来记录方案，那么单点修改的时间复杂度为 $O(\log n)$，比较两个方案的字典序时根据左子树是否相同来决定往左还是往右递归比较，时间复杂度也为 $O(\log n)$。在这里，注意到每个点只会加入一次，因此如果两棵线段树的根节点的指针不同，那么表示的 01 串一定不同，不需要额外维护 Hash 值。

时间复杂度 $O(n \log^2 n)$。

6 New Equipments II

Shortest judge solution: 1711 Bytes.

网络流建图：

- 左边 n 个点表示 n 个工人，右边 n 个点表示 n 台设备，引入源汇 S 和 T。
- 由 S 向第 i 个工人连边，容量 1，费用 a_i。
- 由第 i 个设备向 T 连边，容量 1，费用 b_i。
- 如果工人 i 可以匹配设备 j，那么由工人 i 向设备 j 连边，容量 1，费用 0。

我们的目标就是求出流量 $= 1, 2, 3, \ldots, n$ 时的最大费用流，那么依次增广 n 次，每次需要高效地找到增广路然后进行增广，由于增广路长度为 $O(n)$，因此算法的瓶颈位于寻找增广路之上。

在每一次增广中，我们需要找到一条 S 到 T 的费用最大的路径，注意到非 0 费用只会出现在每个点连到源汇的边上，因此等价于寻找一个工人 x 和一个设备 y，满足它们在之前的增广中没有被使用过，x 通过一开始的边以及增广后添加的 $O(n)$ 条用于反悔的匹配边可以到达 y，且 $a_x + b_y$ 最大。枚举右侧每个点 y，只要找到左侧能到达它的点权最大的点，那么就可以找到最优的点对以及对应的增广路。按照 a 从大到小依次从左侧每个点开始 BFS，记录每个点第一次被左侧哪个点遍历到，那么它就表示左侧能到达它的点权最大的点。在这里由于图是用补图的方式给出的，需要使用补图 BFS 的方式，即维护目前还未遍历过的点集 R，遍历到点 x

时，将 R 修改为 x 在补图中的出边集，由于补图只有 m 条边，因此按照这种方式 BFS 的时间复杂度为 $O(n+m)$。

一共要增广 n 轮，每轮寻找增广路的时间复杂度为 $O(n+m)$，故总时间复杂度为 $O(n^2+nm)$。

7 Photoshop Layers

Shortest judge solution: 674 Bytes.

预处理出 f_i 表示图层 i 左侧第一个合成方式为“普通” 的图层。对于每个询问，求出 r 左侧第一个合成方式为 “普通” 的图层 f_r，则中间的部分都是 “线性减淡”，可以用前缀和求出结果，最后与 255 取最小值。

时间复杂度 $O(n+q)$。

8 Restore Atlantis II

Shortest judge solution: 4143 Bytes.

考虑一维的情况，随机数据下 n 条线段的线段并可以用特别少的极长线段来表示，在二维情况下也是成立的。一个简单的实现方式是在 x 方向将所有关键点提取出来，相邻两个关键点之间用 `std::vector` 记录 y 方向的线段并由哪些极长线段构成，接着依次考虑相邻两段 x 方向的关键点中间的部分，如果它们记录的 `std::vector` 相同，那么就把这两段合并成同一段，起到压缩的效果。如此一来，随机数据下，任何一个区间的矩形并都可以用大约几十的信息来记录。

剩下的问题就是查询区间矩形并，如果使用线段树维护区间矩形并，那么预处理需要 $O(n)$ 次信息合并，每次查询需要 $O(\log n)$ 次信息合并，效率较低。考虑将 n 个矩形分成 $O(\frac{n}{\log n})$ 块，每块 $O(\log n)$ 个矩形，预处理出每一块块内的前后缀矩形并，同时用 ST 表预处理出任意两块之间的矩形并，那么预处理需要 $O(n)$ 次信息合并。对于每个查询 $[l,r]$：

- 如果 l 和 r 位于不同的块里，那么最终需要的信息等于 l 所在块内的后缀信息并 $+r$ 所在块内的前缀信息并 $+$ 中间跨越的整块的信息并，需要 $O(1)$ 次信息合并。
- 如果 l 和 r 位于同一块里，由于数据随机，这样的询问数量期望为 $O(\log n)$，直接暴力求出信息并即可。

最终一共需要 $O(n+q)$ 次信息合并，程序运行效率受信息合并实现方式的影响较大。

9 Rise in Price

Shortest judge solution: 1326 Bytes.

设 $f_{i,j,k}$ 表示从 $(1,1)$ 走到 (i,j)，一路上收集了 k 个钻石时，钻石的单价最高能涨到多少，则 $ans=\max(k\times f_{n,n,k})$。

对于固定的 (i,j) 来说，考虑两个状态 $f_{i,j,x}$ 和 $f_{i,j,y}$，其中 $x<y$，如果 $f_{i,j,x}\le f_{i,j,y}$，则状态 $f_{i,j,x}$ 一定不可能发展为最优解，可以剔除。对于每个 (i,j)，用列表按照 k 升序保存所有状态，并剔除不可能成为最优解的状态即可。

随机数据下当 $n=100$ 时，单个 (i,j) 的有效状态的峰值 k 大约为几千。时间复杂度 $O(n^2k)$。

10 Road Discount

Shortest judge solution: 1425 Bytes.

将原始边作为白边，折扣边作为黑边，由于同一条边不可能选择两次，那么问题等价于求包含恰好 k 条黑边的最小生成树。这是一个经典问题，令 $f(k)$ 表示包含恰好 k 条黑边的最小生成树的边权和，则 $f(k)$ 是一个凸函数，求出 $f(k)$ 的方法为：

- 选择参数 c，将每条黑边的边权都加上 c。
- 求出修改边权后的图的最小生成树，令 $sum(c)$ 为对应的边权和，$l(c)$ 为最小生成树中使用黑边数量的最小值，$r(c)$ 为最小生成树中使用黑边数量的最大值。
- 二分找到合适的参数 c，满足 $l(c)\le k\le r(c)$，则 $f(k)=sum(c)-k\times c$。

由于边权在 $[1,1000]$ 之间，因此可以预处理出 $c=-1000\ldots1000$ 的所有信息，一共需要求 $O(c)$ 次最小生成树。注意到如果对黑边或者白边单独求最小生成树，则非树边不可能用到，因此可以将边数缩减至 $O(n)$。

总时间复杂度 $O(m\log n+nc\log n)$。

11 Segment Tree with Pruning

Shortest judge solution: 371 Bytes.

线段树上代表区间长度相同的节点的子树点数相同，且最多只有 $O(\log n)$ 种本质不同的区间长度，对区间长度记忆化搜索即可。

时间复杂度 $O(\log n\log\log n)$。

12 Tree Planting

Shortest judge solution: 2850 Bytes.

算法一：设 $f_{i,S}$ 表示考虑了前 i 个位置，i 往前 k 个位置的种树情况为 S 的总贡献，时间复杂度 $O(n2^k)$。

算法二：将位置 i 放在二维矩阵的 $(i \bmod k,\lfloor\frac{i}{k}\rfloor)$ 处，那么这个矩阵有 k 行 $\lceil\frac{n}{k}\rceil$ 列。如果 (i,j) 种了树，那么 $(i-1,j)$、$(i+1,j)$、$(i,j-1)$、$(i,j+1)$ 都不能种树，除此之外，第一行和最后一行也存在冲突。从上往下，从左往右轮廓线 DP，设 $f_{i,j,S,T}$ 表示考虑到了 (i,j) 这个格子，第一行种树情况为 S，轮廓线上种树情况为 T 的总贡献，时间复杂度 $O(n2^{\frac{2n}{k}})$。

注意到两种算法里 S 和 T 必定是独立集，而 k 个点的链的独立集数量为斐波那契数的第 k 项，大约为 $O(1.618^k)$，因此算法一可以降至 $O(n1.618^k)$，算法二降至 $O(n1.618^{\frac{2n}{k}})$。

当 $k \le \frac{2n}{k}$，即 $k \le \sqrt{2n}$ 时使用算法一，当 $k > \sqrt{2n}$ 时使用算法二，可以得到最优复杂度 $O(n1.618^{\sqrt{2n}})$。