2019 Multi-University Training Contest 6 Editorial

By Claris

2019 年 8 月 7 日

1 Salty Fish

Shortest judge solution: 1233 Bytes.

假设要偷走所有的苹果，那么现在需要放弃一些苹果，并且黑掉一些监控相机以保证能偷走没有放弃的苹果，我们需要最小化放弃的苹果的收益之和加上黑掉相机支付的代价的总和。

考虑最小割建图：

- 源点 S 向每个监控相机连边，割掉这条边的代价为黑掉它的代价，割掉这条边表示黑掉这个监控相机。
- 每个节点向汇点 T 连边，割掉这条边的代价为这个点的苹果数，割掉这条边表示放弃这个节点的苹果。
- 每个监控相机向其监控范围内的所有节点连边，割掉这条边的代价为 $+\infty$，表示不能破坏监控关系。

那么每条 S 到 T 的路径都表示某个节点既没有并放弃，又被一些相机监控着，这是不合法的。我们需要割掉代价之和最少的边，使得 S 和 T 不连通，因此最终的答案就是所有节点的苹果数量之和减去这个图的最小割。

因为最小割 = 最大流，从叶子向根节点依次考虑每棵子树，计算最大流。设 $v[i][j]$ 表示 i 的子树中离 i 距离为 j 的那些节点还能提供多少流量，那么考虑监控范围最高点在 i 的每个监控相机，显然应该优先接收距离较大的那些节点的流量。

用 `std::map` 存储每个值非零的 $v[i]$，可以在总计 $O(m \log n)$ 的时间内求出最大流。至于 $v[i]$ 的计算，可以由 i 的儿子的 v 启发式合并而来。

注意到这是关于深度的一个启发式合并，如果将这棵树长链剖分，计算出每个点 x 子树内离 x 距离最远的点到 x 的距离 $d[x]$，那么可以选择将 d 最大的儿子的 v 继承给 x，然后将其它儿子的 v 暴力插入到 $v[x]$ 中。因为每个点仅属于一条长链，且一条长链只会在链顶位置作为短儿子被暴力合并一次，所以合并的时间复杂度为 $O(n)$。

总时间复杂度为 $O((n+m)\log n)$。

2 Nonsense Time

Shortest judge solution: 980 Bytes.

考虑时间倒流，看作一个完整的排列按照一定顺序依次删除每个数，然后每次需要计算 LIS 的长度。

首先在 $O(n \log n)$ 的时间内求出 LIS，并找到一个 LIS。当删除 x 时，如果 x 不在之前找到的那个 LIS 中，那么显然 LIS 的长度是不会变化的，否则暴力重新计算出新的 LIS 即可。

因为数据随机，因此 LIS 的期望长度是 $O(\sqrt{n})$，删除的 x 位于 LIS 中的概率是 $\frac{1}{\sqrt{n}}$，也就是说期望删除 $O(\sqrt{n})$ 个数才会修改 LIS，那么 LIS 变化的次数不会很多。

期望时间复杂度为 $O(n\sqrt{n} \log n)$。

3 Milk Candy

Shortest judge solution: 2345 Bytes.

建立一张 $n+1$ 个点的图，点的编号为 0 到 n，点 i 表示 $s_i = x_1 + x_2 + \cdots + x_i$。如果我们知道了 $x_l + x_{l+1} + \cdots + x_r$，那么我们就知道了 $s_r - s_{l-1}$ 的值，在 $l-1$ 和 r 之间连一条边。如果这个图是连通的，那么我们就能根据 $s_0 = 0$ 推出所有 s，从而推出所有 x。问题转化为从每个 NPC 手中恰好购买 k_i 条边，使得这个图连通，且代价之和最小。

从另外一个角度考虑这个问题：先购买所有边，然后从每个 NPC 手中删除不超过 $c_i - k_i$ 条边，总计删除恰好 $\sum(c_i - k_i)$ 条边，使得剩下的图仍然连通，且删去的边代价之和最大。由于删去边后图连通等价于剩下的边存在生成树，生成树是图拟阵的基，所以这是图拟阵的对偶拟阵 M_1；而从每个边集中选择不超过若干条边的条件，则是划分拟阵 M_2。

所以我们的目标就是找到这两个拟阵的交的大小为 $\sum(c_i - k_i)$ 的权值和最大的独立集，可以用拟阵交算法解决：

- 令初始解 I 为空集，即没有边被删除。
- 每条边作为有向图中的一个点，并新建源点 S 和汇点 T。
- 对于 $x \notin I$ 的某条边 x，将 x 的点权设置为 w_x，表示额外删掉这条边的代价。若 $I \cup x$ 满足 M_1，则连边 $S \to x$；若 $I \cup x$ 满足 M_2，则连边 $x \to T$。
- 对于 $x \in I$ 的某条边 x，将 x 的点权设置为 $-w_x$，表示取消删除这条边的代价。
- 对于 $x \in I$ 的某条边 x 以及 $y \notin I$ 的某条边 y，若 $I \setminus x \cup y$ 满足 M_1，则连边 $x \to y$；若 $I \setminus x \cup y$ 满足 M_2，则连边 $y \to x$。
- 在构造出来的图中 SPFA 找到 S 到 T 的最长路作为增广路，将上面每条边的选择情况取反，得到新的解 I'，此时 I' 的大小比 I 刚好大 1。不断重复构图找增广路直至 I 的大小为 $\sum(c_i - k_i)$。

不妨认为 $n, m, \sum c$ 同阶，则一共 $O(n)$ 次增广，每次增广建图需要 $O(n^3)$ 的时间，寻找增广路需要 $O(n^3)$ 的 SPFA，总时间复杂度为 $O(n^4)$。

4 Speed Dog

Shortest judge solution: 2134 Bytes.

问题等价于找到一堆 $x_i(0 \le x_i \le 1)$，使得下面式子的值最小：

$$\max\left(\sum_{i=1}^{n} a_i x_i, \sum_{i=1}^{n} b_i(1-x_i)\right)$$

因为

$$\max(A,B) = \max_{0\le k\le 1}(kA + (1-k)B)$$

所以

$$\begin{aligned} & \max\left(\sum_{i=1}^{n} a_i x_i, \sum_{i=1}^{n} b_i(1-x_i)\right) \\ = & \max_{0\le k\le 1}\left(\sum_{i=1}^{n} k a_i x_i + (1-k) b_i(1-x_i)\right) \end{aligned}$$

根据 Minimax Theorem，有

$$\begin{aligned} & \min_{0\le x_1,x_2,\ldots,x_n\le 1}\left(\max_{0\le k\le 1}\left(\sum_{i=1}^{n} k a_i x_i + (1-k) b_i(1-x_i)\right)\right) \\ = & \max_{0\le k\le 1}\left(\min_{0\le x_1,x_2,\ldots,x_n\le 1}\left(\sum_{i=1}^{n} k a_i x_i + (1-k) b_i(1-x_i)\right)\right) \\ = & \max_{0\le k\le 1}\left(\sum_{i=1}^{n}\min_{0\le x_i\le 1}(k a_i x_i + (1-k) b_i(1-x_i))\right) \\ = & \max_{0\le k\le 1}\left(\sum_{i=1}^{n}\min(k a_i, (1-k) b_i)\right) \end{aligned}$$

注意到对于固定的 i 来说，$\min(ka_i, (1-k)b_i)$ 关于 k 的函数是一个凸函数，而凸函数的和 $f(k)$ 也为凸函数，因此可以通过三分这个 k 得到答案。

对于固定的 i 来说，当 $ka_i = (1-k)b_i$，也就是 $k = \frac{b_i}{a_i+b_i}$ 时取得极值，所以只有 $O(n)$ 个这样的 k 是有用的，只需要在它们之间三分，也因此避免了浮点数运算。注意这里需要对这些 k 进行去重，否则三分时可能会出现函数平台导致三分失败。

现在剩下的问题就是如何快速得到答案。对于这些 k 建立一棵权值线段树，将每个二元组 (a_i, b_i) 放在分界线 $\frac{b_i}{a_i+b_i}$ 的位置上。线段树每个节点维护对应区间内 a 的和、b 的和以及区间左端点和右端点对应的 a 和 b 的和，插入一个新的二元组的时间复杂度为 $O(\log n)$。

查询最优解时，只需要从线段树根节点开始，假设左子树表示 $[l, mid]$，右子树表示 $[mid+1, r]$，那么通过比较 $f(mid)$ 和 $f(mid+1)$ 的大小即可知道极值位于左子树还是右子树，通过线段树维护的信息可以 $O(1)$ 算出 $f(mid)$ 和 $f(mid+1)$ 的值。

时间复杂度 $O(n \log n)$。

5 Snowy Smile

Shortest judge solution: 1383 Bytes.

首先将纵坐标离散化到 $O(n)$ 的范围内，方便后续的处理。

将所有点按照横坐标排序，枚举矩形的上边界，然后往后依次加入每个点，这样就确定了矩形的上下边界。设 $v[y]$ 表示矩形内部纵坐标为 y 的点的权值和，则答案为 v 的最大子段和，用线段树维护带修改的最大子段和即可。

时间复杂度 $O(n^2 \log n)$。

6 Faraway

Shortest judge solution: 938 Bytes.

将 $|x_i - x_e| + |y_i - y_e|$ 的绝对值拆掉，则每个点 (x_i, y_i) 会将平面分割成 4 个部分，每个部分里距离的表达式没有绝对值符号，一共 $O(n^2)$ 个这样的区域。

枚举每个区域，计算该区域中可能的终点数量。注意到 $lcm(2,3,4,5) = 60$，所以只需要枚举 x_e 和 y_e 模 60 的余数，$O(n)$ 判断是否可行，然后 $O(1)$ 计算该区域中有多少这样的点即可。

时间复杂度为 $O(60^2 n^3)$。

7 Support or Not

Shortest judge solution: 3013 Bytes.

首先考虑找到第 k 小的球对距离，二分答案 mid，统计有多少对球的距离不超过 mid。我们需要找到最小的 mid，使得有至少 k 对球的距离不超过 mid。

将每个球的半径都加上 $\frac{mid}{2}$，那么我们的目标是统计有多少对球存在公共点。

假设最大的球半径为 R，以 $2R$ 为棱长将三维空间划分为一个个立方体格子，那么每个球只需要检查球心在附近 27 个格子内部的所有球。考虑所有球的半径相等的情况，那么每个格子内部一旦有超过 $O(\sqrt{k})$ 个球时，我们必然已经找到了 k 对相交的球。因此在找到 k 对相交的球时及时结束二分答案的检查过程即可。

但是当球的半径不尽相同时，上述分析不成立。那么在当前球的半径不足 $\frac{R}{2}$ 时重构网格，则最多会重构 $O(\log r)$ 次，且每个球依然只会检查均摊 $O(\sqrt{k})$ 个球与它是否相交。

找到第 k 小解 ans 后，我们只需要取 $mid = k - 1$，继续运行检查算法，将找到的这些相交球对之间的距离作为最终的答案的即可，如果不足 k 个，那么剩下的答案肯定都是 ans。

利用 Hash 表定位格子，则总时间复杂度为 $O(n \log^2 r + n\sqrt{k} \log r)$，常数较小。

8 TDL

Shortest judge solution: 472 Bytes.

考虑枚举 $f(n,m) - n$ 的值 t，则 $n = t \oplus k$，$O(t \log n)$ 检查这个 n 是否满足条件即可。

注意到 t 显然不会超过第 m 个与 n 互质的质数，而 n 最多只有 $O(\log \log n) < m = 100$ 个质数，根据质数密度可以得到 t 的一个比较松的上界 $O(m \log m)$。

时间复杂度 $O(m^2 \log^2 m \log n)$。

9 Three Investigators

Shortest judge solution: 713 Bytes.

考虑将数字 $a[i]$ 拆成 $a[i]$ 个 $a[i]$，比如 "`4,1,2`"→"`4,4,4,4,1,2,2`"，则问题转化为：找到最多 5 个不共享元素的不下降子序列，使得这些子序列包含的元素总量最多。可以证明，这等于杨氏图表前 5 层的长度之和。

考虑杨氏图表求解答案的过程：

- 从 1 到 n 依次考虑序列中的每个数，将其插入杨氏图表的第一层中。
- 插入 x 时，如果 x 不小于这一层的最大的数，则将 x 放在这一层的末尾；否则找到大于 x 的最小的数 y，将 y 替换为 x，并将 y 插入下一层。

因为每一层的元素都有序，所以可以用数组维护，寻找 y 的过程可以用二分查找加速。

但是对于本题来说，我们不能暴力地插入 $a[i]$ 个 $a[i]$。考虑将杨表每一层中相同的元素合并，用 `std::map` 记录每个元素的个数，那么当我们一次性插入 x 个 x 时，只需要将其插入 `std::map` 中，然后不断消费后继，将后继的元素个数减少即可，在减少的时候要将其作为 "p 个 q" 插入下一层中。

每一类数字被消费完毕后需要及时从 `std::map` 中删除，而每次插入会导致最多一种其它数字被拆分，所以每层的插入次数至多为上一层的两倍。

假设要求不超过 k 个子序列的答案，本题中 $k = 5$，则时间复杂度为 $O(2^k n \log n)$。

10 Ridiculous Netizens

Shortest judge solution: 1610 Bytes.

取一个根，将这棵树转化为有根树，考虑连通块包含根节点的情况，那么对于一个点来说，如果它选了，它的父亲就必须选。

求出 DFS 序括号序列，设 $f[i][\lfloor \frac{m}{j} \rfloor]$ 表示考虑了 DFS 序的前 i 项，目前连通块点权乘积为 j 的方案数。因为当 $j \geq \sqrt{m}$ 时 $\lfloor \frac{m}{j} \rfloor$ 只有 $O(\sqrt{m})$ 种取值，所以状态数为 $O(n\sqrt{m})$。注意到 $\lfloor \frac{m}{jk} \rfloor = \lfloor \frac{\lfloor \frac{m}{j} \rfloor}{k} \rfloor$，所以可以转移。

如果 i 是一个左括号，那么把 f 传给儿子，并强制选择儿子；如果 i 是个右括号，那么这个子树既可以选又可以不选，将对应状态的方案数累加即可，转移 $O(1)$。

接下来考虑连通块不包含根节点的情况，那么可以去掉这个根，变成若干棵树的子问题。取重心作为根进行点分治，则考虑的总点数为 $O(n \log n)$。

时间复杂度 $O(n\sqrt{m} \log n)$。

11 11 Dimensions

Shortest judge solution: 2102 Bytes.

设 $f[i][j]$ 表示 $[1, i-1]$ 这些位的数字已经确定，且 $[1, i-1]$ 的数字模 $m = j$ 时，有多少种在 $[i, n]$ 这些位填数字的方法使得最终的数字模 $m = 0$。

初始值：$f[n+1][0] = 1$。

转移：$f[i][j] = \sum f[i+1][(10j+k) \bmod m]$，其中 $0 \le k \le 9$，且第 $i+1$ 位可以填 k。

最终满足条件的总方案数即为 $f[1][0]$，这样就可以判断每个询问是否有解。

考虑在有解的情况下如何找到第 k 小的方案。我们称如果状态 i 由状态 j 等累加得到，则 j 是 i 的一个后继状态。从初始状态 $(1,0)$ 开始，按照下一位填的数字从小到大枚举当前状态 S 的每个后继状态 T，如果 T 的 DP 值 $\ge k$，则说明我们要找的方案在 T 中，且这个方案下这一位已经确定，然后走到 T 状态即可；否则 T 的 DP 值 $< k$，那么将 k 减去 T 的 DP 值，然后继续考虑其它更大的后继即可。这样单次询问是 $O(n)$ 的，不能接受。

类似树的轻重链剖分，对于每个状态，取其后继状态中 DP 值最大的状态作为重后继，则每个状态最多只有一个重后继，我们可以倍增求出每个状态往后走 2^k 次重后继后会到达哪个状态，以及那个状态相对当前来说是第几小的方案。对于每个询问，我们先在倍增数组中沿着重后继不断往前走直到必须要走轻后继为止，然后走一次轻后继，再接着沿着倍增数组走重后继。

因为重后继是 DP 值最大的后继，这意味着每个轻后继的 DP 值不超过总方案数的一半，所以每走一次轻后继，k 至少会除以二，最多 $O(\log k)$ 次。

时间复杂度 $O(nm \log n + q \log n \log k)$。

12 Stay Real

Shortest judge solution: 376 Bytes.

小根堆中，每个点的权值总是不小于父亲节点的权值。所以无论怎么取，先拿走的数一定不小于后面拿走的数。

此时双方的最优策略就是：贪心选择能取的数字之中最大的数。

时间复杂度 $O(n \log n)$。