《运筹帷幄》解题报告

江苏省常州高级中学 赵海鲲

October 25, 2024

Contents

1 题目

1.1 题目描述

给定一棵 n 个点的树，第 i 个节点有 b_i 个棋子，且最多能放 a_i 个棋子。现在有一个点 k 是根。每次操作你可以选择一个点，将它的一个棋子，移到它的父亲上，需要满足它父亲的棋子数没有超过限制，然后需要最小化每个棋子到 k 的距离和。

对 $k = 1, 2, \cdots, n$ 都求出答案。

1.2 数据范围

$0 \le b_i \le a_i$，$1 \le a_i \le 10^7$，为了避免答案爆 long long，将 a_i 的范围开小了一点。

$1 \le n \le 5 \times 10^5$。

subtask 1（1 分）：$b_i = 0$；

subtask 2（5 分）：$n \le 100$；

subtask 3（11 分）：$n \le 5000$；

subtask 4（3 分）：链，保证 $\forall i \in [1, n-1] \cap \mathbb{Z}$，满足 i 和 $i+1$ 有边；

subtask 5（3 分）：菊花，保证 $\forall i \in [2, n] \cap \mathbb{Z}$，满足 1 和 i 有边；

subtask 6（6 分）：保证树随机；

subtask 7（16 分）：$a_i \le 5$；

subtask 8（22 分）：$n \le 5 \times 10^4$；

subtask 9（16 分）：$n \le 10^5$；

subtask 10（11 分）：$n \le 2 \times 10^5$；

subtask 11（5 分）：$n \le 3 \times 10^5$；

subtask 12（1 分）：无。

这里说明随机树的生成方式：对于结点 $i \in [2, n]$，在 $[1, i-1]$ 内等概率随机一个点 p，将 i, j 连一条边。

2 解题过程

2.1 算法 1

对于 $b_i = 0$，没有棋子，所以答案一定为 0。

于是，只需要输出 n 个 0 即可。

可以通过 subtask 1，期望得分 1 分。

2.2 算法 2

我们考虑对每个根 k 分别处理。

首先，由于 $a_i \geq 1$，所以我们可以把题目转化成每次选择一个棋子，移到其祖先。

考虑贪心。

2.2.1 流程

我们 DFS 整棵树，假设当前在结点 x，先对于 x 的子树进行贪心的过程，然后再将子树内最深的棋子移到当前位置。

2.2.2 证明

首先，由于操作都是将深度大的棋子，移到深度小的结点上去，所以我们可以将操作按照移到的结点的深度从大到小重排。

于是，我们可以把过程看成 DFS 整棵树，假设当前在结点 x，先对于 x 的子树进行操作，然后进行将子树内的棋子移到当前位置的操作。

假设当前子树内有 k 棵棋子，到 x 的距离按照**从小到大排序**为 $\{a_1, a_2, \cdots, a_k\}$。

假设当前可以选择移动一枚棋子到 x，我们可以说明：

1. 选择棋子移动到 x 是不劣的，因为假设选择了第 p 个棋子移动到 x，那么距离序列就变成了 $a' = \{0, a_1, a_2, \cdots, a_{p-1}, a_{p+1}, a_{p+2}, \cdots, a_k\}$。

   注意到，a' 序列能全维偏序 a 序列（这里全维偏序是指所有对应位置的数都有 $a'_i \leq a_i$，下同）。

   所以，我们可以说明选择棋子移动到 x 是不劣。

2. 选择第 k 个棋子移动到 x 是不劣的，因为假设选择了第 $p \neq k$ 个棋子移动到 x，那么距离序列就变成了 $b = \{0, a_1, a_2, \cdots, a_{p-1}, a_{p+1}, a_{p+2}, \cdots, a_k\}$。

   假设选择了 a_k 移动到 x，那么距离序列就变成了 $b' = \{0, a_1, a_2, \cdots, a_{k-1}\}$。

   注意到，b' 序列能全维偏序 b 序列。

   所以，我们可以说明选择距离最远的棋子移动到 x 是不劣。

整个过程可以使用可并堆或者线段树合并来维护棋子，做到 $O(n \log n)$ 的时间复杂度。

由于需要对 n 个根分别做，所以总时间复杂度 $O(n^2 \log n)$。

可以通过 subtask 2，期望得分 5 分。

2.3 算法 3

考虑优化算法 2 的过程，我们考虑使用长剖来维护，我们维护 $f_{i,j}$ 表示当前 i 子树内，距离 i 为 j 的棋子数。

如果使用 `std::vector <int>` 来维护 f_i，那么对于移动棋子的操作，我们可以直接在 `vector` 上做 `pop_back` 的操作。

这样时间复杂度可以做到均摊 $O(n)$。

由于需要对 n 个根分别做，所以总时间复杂度 $O(n^2)$。

可以通过 subtask 2、3，期望得分 16 分。

2.4 算法 4

对于树是一条链，假设当前 k 是根。

那么，除了 k 结点，其它结点的分配都是相当固定的，于是我们可以先忽略 k 结点的操作，找到当前左侧删到 l，右侧删到 r。

我们分类讨论，对于 k 结点，由于肯定是删远的，所以如果全在大的一边删完，距离还比另一边远，那么就可以直接计算；否则，存在分界点 p，满足距离 $> p$ 的全被删完了，$= p$ 的删了一部分，可以二分找到分界点 p。

时间复杂度 $O(n \log n)$。

可以通过 subtask 4，期望得分 3 分。

2.5 算法 5

对于树是一条菊花，，假设当前 k 是根。

首先对于 $k = 1$，只需要计算距离为 1 的棋子数即可。

对于 $k \neq 1$，我们也可以简单计算出最终距离为 1 的棋子数，和最终距离为 2 的棋子数。

时间复杂度 $O(n)$。

可以通过 subtask 5，期望得分 3 分。

2.6 算法 6

对于题目描述的随机树生成方式，我们可以证明整棵树的最大深度期望为 $O(\log n)$。

所以，对于处理 $k = 1$ 的状态，我们可以直接记 $f_{x,i}$ 表示到结点 x 距离为 i 的点数。

考虑如何换根，我们还是按照 DFS 的顺序去计算答案，假设当前在结点 x，我们维护 $g_{x,i}$，表示子树外到 x 距离为 i 的点数。

时间复杂度 $O(n \log n)$。

可以通过 subtask 5、6，期望得分 9 分。

2.7 算法 7

对于 $a_i \leq 5$，我们考虑对全局维护一棵线段树，按照 DFS 序维护第 i 个点上的棋子数量。

假设当前 k 是根，我们先维护出 $k = 1$ 的状态，那么只需要 DFS 整棵树，假设当前在结点 x，先对于 x 的子树进行操作，然后再将 x 子树内对应的 DFS 区间上的前 $a_x - b_x$ 个棋子，全都移到 x 上，这个可以直接在线段树上维护区间前 5，也可以查询 5 次区间最大值。

考虑如何换根，我们还是按照 DFS 的顺序去计算答案，假设当前在结点 x，我们先撤销 x 对于其子树内的棋子移动操作，再进行相应的操作。

时间复杂度 $O(na \log n)$。

可以通过 subtask 7，期望得分 16 分。

2.8 算法 8

上述算法除了最基础的贪心过程，其余部分跟正解毫无关系，所以设置的分值较少。

从算法 8 开始，我们引入一个重要的想法，我们在预处理的过程中持久化线段树合并；在换根的过程中，DSU on tree，并记录相应的线段树编号。

我们先处理子树 x 被操作后的结果，使用持久化线段树合并来记录子树 rt_x 表示 x 子树操作完后的结果。

然后，我们考虑换根，假设当前在结点 x。

我们考虑在过程中维护：

- 线段树根结点编号的集合 S_x；
- 用**数组**来维护 f_i 表示到 x 距离为 i 的点的数量。

我们考虑 DSU on tree。

我们考虑转移到儿子 y，这里有两种处理方法：

- 对于重儿子：我们考虑将其它轻儿子子树在 f 上加入影响；
- 对于轻儿子：我们将 x 的线段树根结点编号加入 S 集合内，再将这棵子树在 f 中减去影响，因为这棵子树被重复算了，离开这个子树，再将这棵子树在 f 中加上影响。

类似地，我们也可以先将所有轻儿子子树在 f 中加上影响。接着，

- 对于重儿子：我们不进行任何操作；
- 对于轻儿子：我们将重儿子的线段树根结点编号加入 S 内，再将这个棵子树在 f 中减去影响，离开这个子树，再将这棵子树在 f 中加上影响。

总之，我们可以用一个集合 S 和一个数组 f 表示信息，我们需要对 f 维护当前删了多少个，并对 f 维护后缀和。

由于 DSU on tree 的性质，我们加入轻子树的时间复杂度是 $O(n\log n)$，并且 $|S|$ 是 $O(\log n)$ 的。

我们考虑如何处理删除最大的 k 个，由于有 $|S|$ 个线段树和一个数组 f，我们需要先在外层二分，然后由于 S 内的线段树存在一个下标偏移，我们只能分别在每个线段树上二分。

时间复杂度 $O(n\log^3 n)$。

可以通过 subtask 2、3、4、5、8，取决于实现优劣决定是否能通过 subtask 6、9，期望得分 $44\sim66$ 分。

2.9 算法 9

我们定义 size_x 表示 x 子树内的结点数量。

从算法 9 开始，我们再引入一个重要的想法，注意到在换根的过程中，我们一定是优先删除到 x 距离 $>\text{size}_x$ 的点，这样可以保证我们的删除是有顺序的。

我们还是要先处理子树 x 操作完后的结果，这里不再赘述。

然后，我们考虑换根，假设当前在结点 x。

我们考虑在过程中维护：

- 子树外的线段树根结点编号集合 S_x；
- 用**线段树**来维护 f_i（$i\le\text{size}_x$）表示距离 x 为 i 的棋子数量。

注意到在线段树上一段长度为 len 的区间上修改和查询，时间复杂度都是 $O(\log n+\text{len})$ 的。

我们考虑转移到儿子 y：

- 对于重儿子
  - 对于距离 $>\text{size}_y$：对于 f 超过的部分，我们可以直接加入 S_x 内的最后一棵线段树上（如果没有就新开一棵加入 S），这部分 f 超过部分的大小其实就是轻子树大小。
  - 对于距离 $\le\text{size}_y$：我们可以将轻儿子在 f 上修改。
- 对于轻儿子
  - 对于距离 $>\text{size}_y$：
    - 对于来自 x 子树内的：我们由于 y 子树内到 y 的距离不可能 $>\text{size}_y$，所以我们直接对 x 线段树分裂即可。
    - 对于来自 x 子树外的：我们相当于是截取 f 的一段区间，只需要对 f 做线段树分裂即可。

- 对于距离 $\leq \text{size}_y$：我们可以遍历 x 线段树到 x 距离不超过 size_y 的部分，然后再减去来自 y 子树的贡献。

此外，对于删除距离前 k 大的，注意到我们肯定是依次删 S_x 最早加入的线段树。

不过很遗憾，问题在于我们每次加入了两棵线段树，并且单独求 x 的答案需要涉及到 x 子树内的线段树和 f 这两棵线段树，都结构不同，只能做到时间复杂度 $O(n\log^2 n)$。

可以通过 subtask 2、3、4、5、8、9，取决于实现优劣决定是否能通过 subtask 1、6、9、10、11，期望得分 $55\sim 99$ 分。

2.10 算法 10

其实算法 9 已经相当接近了！

我们再做一些改进，结合一下算法 8、9 的优势。

我们还是要先处理子树 x 操作完后的结果，这里不再赘述。

然后，我们考虑换根，假设当前在结点 x。

我们考虑在过程中维护：

- 子树外的线段树根结点编号集合 S_x；
- 用**数组**来维护 f_i 表示距离 x 为 i 的棋子数量。

我们考虑转移到儿子 y：

我们先将所有轻儿子都加入 f 里面。

- 对于重儿子：不需要做额外处理；
- 对于轻儿子：我们先将 y 在 f 内的贡献删除，然后再截取重儿子距离 $\leq \text{size}_y$ 的部分，对于重儿子 $> \text{size}_y$ 的部分，我们将重儿子的线段树分裂，然后将对应的编号加入 S_y 内。

此外，对于删除距离前 k 大的，注意到我们肯定是依次删 S_x 最早加入的线段树，我先判断这棵子树能否删空，如果能删空，我们就直接在 S 中删除，否则再进行线段树二分。这样，我们就只需要进行至多一次线段树二分。

对于 f 我们维护**后缀和**，以及当前删了多少个，离开子树的时候对 f 进行撤销。

在 f 做删除的时候，我们可以二分处理，若 S_x 集合有线段树，也可以配合线段树二分操作。

如果加入轻儿子时，在已经被删除的地方增加了棋子，则我们可以直接重构 f。

时间复杂度 $O(n\log n)$，不过很遗憾，空间复杂度同样也是 $O(n\log n)$。

可以通过所有子任务，期望得分 100 分。

2.11 算法 11

从算法 11 开始，我们再引入一个重要的想法，我们抛弃算法 8 ～ 算法 10 的基础想法。

具体地，我们不再使用线段树合并，取而代之地，我们维护数组 $f_{x,i}$ 表示 x 子树内操作完后距离 x 为 i 的棋子数量。

然后我们考虑重链剖分。对于 f_x，我们先继承重儿子 son_x 的 f_{son_x} 的信息，再暴力加入轻儿子。我们考虑这样做的问题在于会破坏之前的 f_{son_x} 的信息。于是，我们考虑在换根到结点 x 的时候，对 f_x 进行撤销，这样就可以得到正确的 f_{son_x} 了。

于是，我们再结合算法 8，就可以把在 $O(\log n)$ 棵线段树上二分，改成在 $O(\log n)$ 个数组上二分。

时间复杂度 $O(n \log^2 n)$。

可以通过 subtask 2、3、4、5、8、9，取决于实现优劣决定是否能通过 subtask 1、6、9、10、11，期望得分 55 ～ 99 分。

2.12 算法 12

我们再将算法 11，结合下算法 10，就可以保证删除的顺序。

时间复杂度 $O(n \log n)$，空间复杂度 $O(n)$。

可以通过所有子任务，期望得分 100 分。

2.13 算法 13

其实我们也可以使用长链剖分来处理，不过现在问题在于 $|S_x|$ 可能是 $O(\sqrt{n})$ 级别。

不过，我们可以把依次判断 S_x 内的第一个数组是否能删空的过程，换成二分处理，这样可以保证这部分的复杂度正确。

时间复杂度 $O(n \log n)$，空间复杂度 $O(n)$。

可以通过所有子任务，期望得分 100 分，不过相比于算法 12 并没有任何优势。

3 命题思路

本题的原型来自于Roars III，不过已经是天差地别，原题做法只能获得 subtask 7 的 16 分。

笔者在命制此题时，从算法 8 开始，逐步优化到算法 9，最后得到了算法 10。

算法 11 是徐恺同学的赛时想法，算法 12 是我在算法 11 基础上加以改进。

算法 13 是刘海峰同学赛时想法，并加以改进。

本题难度较大，代码量较大，涉及知识面广，对选手水平有较为综合的考察。

最终的算法 12 只用到了数组，没有用到任何高级数据结构，形式简洁优美。

4 参考资料

- CCF NOI 科学委员会，《全国青少年信息学奥林匹克系列竞赛大纲（2023 年修订版）》：
  https://www.noi.cn/upload/resources/file/2023/03/15/1fa58eac9c412e01ce3c89c761058a43.pdf；
- The 3rd Universal Cup. Stage 2: Zielona Góra B. Roars III：
  https://qoj.ac/contest/1699/problem/8518。

特别感谢 **章弥炫**、**朱鹏睿**、**徐恺**、**刘海峰**、**黄建恒**、**许淇文**同学与我讨论做法。

感谢 **叶李蹊**、**葛致远**、**翟鹏昊**、**陈诺**、**邱梓轩**同学参与本题的验题工作。