目录

ICPC2026_Wuhan 1
D.Travel Plan 1
G.Shuffle Playlist 5
J.Number Game 8
K.Salary Adjustment 10

ICPC2026_Wuhan

D.Travel Plan

题目描述

小 R 是一个喜欢旅行的女孩子，她来到了一座新的城市。

这座城市有 n 个景点，编号为 $1 \sim n$，景点之间由 m 条双向道路连接。保证每条道路连接两个不同的景点，任意两个景点之间至多只有一条道路连接，且任意两个景点之间都可以通过若干条道路相互到达。也就是说，这座城市的所有景点和它们之间的道路构成一张 n 点 m 边的简单无向连通图。

小 R 希望从某一个景点出发，按照深度优先搜索（DFS）的顺序前往所有景点。具体地，对于每个景点 u，她会把从 u 出发的每一条道路通向的节点按照某个顺序排列为有序列表 $\mathrm{adj}(u)$。接着，她会从选定的起点 s 出发，按照以下伪代码进行旅行：

```
Algorithm 1 Depth-First Search
1: function DFS(u, d)
2:     visited[u] ← true
3:     depth[u] ← d
4:     for each v ∈ adj(u) do
5:         if ¬visited[v] then
6:             dfs-tree ← dfs-tree ∪ {(u, v)}
7:             DFS(v, d + 1)
8:         end if
9:     end for
10: end function
11: dfs-tree ← ∅
12: visited[1..n] ← false
13: depth[1..n] ← 0
14: DFS(s, 0)
```

她希望旅行完成后，每个景点 u 在 DFS 树上的深度（即伪代码中的 $\mathrm{depth}[u]$）都恰好等于 a_u。

请你判断是否存在一种符合要求的方案。如果存在，请构造出一种符合要求的 s 和 $\mathrm{adj}(u)$。

输入格式

第一行包含两个整数 n, m $(2 \le n \le 2 \times 10^5,\ 1 \le m \le 2 \times 10^5)$，表示景点数量和道路数量。

第二行包含 n 个整数 $a_1, a_2, ..., a_n$ $(0 \le a_i < n)$，表示每个景点在 DFS 树上的深度。

接下来 n 行，其中第 i 行首先包含一个整数 d_i $(1 \le d_i < n)$，表示从节点 i 出发的道路数量，随后包含 d_i 个整数 $\mathrm{adj}'(i,1), \mathrm{adj}'(i,2), ..., \mathrm{adj}'(i,d_i)$ $(1 \le \mathrm{adj}'(i,j) \le n)$，表示从节点 i 出发的每一条道路通向的节点。

保证 $\sum_{i=1}^{n} d_i = 2m$，且所有景点和道路构成一张简单无向连通图。

输出格式

若无解，输出一行，包含一个整数 -1。

若有解，第一行输出一个整数 s，表示选择的起点；接下来 n 行，其中第 i 行输出 d_i 个整数 $\mathrm{adj}(i,1),\mathrm{adj}(i,2),...,\mathrm{adj}(i,d_i)$，相邻两个整数之间用一个空格分隔，表示节点 i 对应的有序列表。$\mathrm{adj}(i)$ 必须是 $\mathrm{adj}'(i)$ 的一个重排列。

样例

1.in / 1.out

```
5 7
2 0 1 3 1
3 3 4 2
4 1 4 5 3
3 4 1 2
3 1 3 2
1 2
```

```
2
4 3 2
5 3 1 4
1 4 2
1 3 2
2
```

2.in / 2.out

```
5 7
1 0 2 2 1
3 5 2 4
3 4 1 5
1 5
3 2 5 1
4 1 2 3 4
```

```
-1
```

题解

这么一道签到题写了 $+4$ 出来，我有点服气我自己了。

这个题起初我是打算用拓扑排序把顺序约束出来的，后来发现环状的样例会卡我的拓扑排序所以放弃。

思考一个这样的问题：`dfs` 我们在刚学 OI 的时候就学过它的性质，是“不撞南墙不回头”。 这意味着如果你走到一个点，你要返回的话，你必须**已经走过它的所有边**或者**已经到达过所有指向的点**。

在你走到这个点之前，你不可能会走过这条边，所以我们考虑其他点有没有被访问过。

具体是什么样的点呢？

很显然，我在 `dfs` 中只走 `depth[x] -> depth[x] + 1` 的边（我们将这些边记作“一类边”，其余称为“二类边”）就可以满足题目的约束，但如何确定走点的顺序，如何判断无解呢？

你思考：如果你的“二类边”都指向你 `dfs` 树中的儿子，是不是可以说明这些边一定不会被走过，可以放在你当前 dfs 顺序的最后？

只要有一条“二类边”没有指向儿子，那不管从哪个节点递归下来都必定会走这条边，破坏题目约束。

所以，我们得到了本题的解法，同时也明白了：本题的有解性与 `dfs` 顺序无关。

可以欣赏我的丑陋代码了……

```cpp
#include <stdio.h>
#include <vector>
#define maxn 200007

using namespace std;

vector<int> edge[maxn], ans[maxn];
int req[maxn], mk[maxn], vis[maxn];
int n, m, u, v, tp;

int sol = 1;
void dfs(int index, int fa) {
    if (not sol or vis[index])
        return;
    mk[index] = 1;
    vis[index] = 1;

    for (auto v : edge[index]) {
        if (v != fa) {
            if (req[v] == req[index] + 1) {
                ans[index].push_back(v);
                dfs(v, index);
            }
        }
    }

    for (auto v : edge[index]) {
        if (v != fa and req[v] != req[index] + 1) {
            ans[index].push_back(v);
            if (req[v] <= req[index]) {
                if (not mk[v]) {
                    sol = 0;
                    return;
                }
            }
        }
    }

    if (fa)
        ans[index].push_back(fa);

    mk[index] = 0;
}

```

```
int st;
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%d", req + i);
        if (req[i] == 0) {
            if (st) {
                printf("-1");
                return 0;
            } else {
                st = i;
            }
        }
    }

    if (not st) {
        printf("-1");
        return 0;
    }

    for (int u = 1; u <= n; u++) {
        scanf("%d", &tp);
        for (int i = 1; i <= tp; i++)
            scanf("%d", &v),
                //  printf("Edge: %d <-> %d\n", u, v),
                edge[u].push_back(v);
    }

    dfs(st, 0);

    if (not sol) {
        printf("-1");
        return 0;
    }

    for (int i = 1; i <= n; i++)
        if (not vis[i]) {
            printf("-1");
            return 0;
        }

    printf("%d\n", st);

    for (int u = 1; u <= n; u++) {
        for (auto v : ans[u])
            printf("%d ", v);
        puts("");
    }
```

```
93
94      return 0;
95  }
```

G.Shuffle Playlist

题目描述

在卡牌异变中，埴安神袿姬收集了 n 种能力卡牌，编号分别为 $1, 2, ..., n$，其中编号为 i 的卡牌有 a_i 张。此外，天弓千亦还赠与了她 k 张空白卡牌。

现在袿姬想在灵长园里播放她精心挑选的歌单，歌单包含 n 首歌曲，歌曲编号分别为 $1, 2, ..., n$。播放编号为 i 的歌曲时，如果还有编号为 i 的卡牌，则消耗一张；否则，如果还有空白卡牌，则消耗一张空白卡牌；如果两者都没有，则停止播放。

为避免同一首歌连续播放两次，从该歌单生成播放列表时采用如下随机播放规则：

- 一开始，歌单中的 n 首歌曲会以任意顺序组成一个播放列表，并按列表顺序播放；
- 当当前列表的最后一首歌曲即将开始播放时，若其可播放，则播放列表立即更新为一个新的列表，其中这首歌曲作为第一首，其余 $n-1$ 首歌曲任意排列在后面。

袿姬可以让所有随机排列的结果都取最有利的情况，请你帮她求出她最多可以播放多少首歌曲。

输入格式

本题包含多组测试数据。

第一行包含一个整数 T $(1 \le T \le 10^5)$，表示测试数据组数。

对于每组测试数据：

- 第一行包含两个整数 n, k $(2 \le n \le 10^5, 0 \le k \le 10^{12})$，分别表示卡牌种数与空白卡牌的数量。
- 第二行包含 n 个正整数 $a_1, a_2, ..., a_n$ $(1 \le a_i \le 10^{12})$，表示每种编号的卡牌的数量。

保证所有测试数据中 n 的总和不超过 2×10^5。

输出格式

对于每组测试数据，输出一行，包含一个整数，表示最优情况下最多能播放的歌曲数。

样例

1.in / 1.out

```
8
3 0
1 1 1
4 0
4 4 4 1
4 1
4 4 4 1
2 3
147154220721 1
4 9
1 6 11 16
3 1
1 5 4
3 1000000000000
1 5 4
```

```
16 6 2
17 2 1 1 1 1 1
```

```
1 3
2 9
3 14
4 9
5 41
6 10
7 1000000000010
8 9
```

在样例的第一组数据中，没有空白卡牌。不妨设初始播放列表为 [1，2，3]。先播放歌曲 1 和 2。当歌曲 3 即将开始播放时，播放列表更新为 [3，1，2]（或 [3，2，1]）。接着播放歌曲 3。播放完歌曲 3 后，下一首是歌曲 1（或歌曲 2），但对应的卡牌已经在之前被消耗，因此播放停止。共播放了 3 首歌。

在样例的第二组数据中，一种可能的最优情况为：播放列表的变化为 [1，2，3，4] → [4，1，2，3] → [3，1，2，4]，其中最后一个播放列表中的歌曲 4 无法播放，共播放了 9 首歌。注意，播放至列表的最后一首歌曲时如果无法播放，那么播放会立即停止，播放列表不会更新。

在样例的第三组数据中，最优情况下第二次播放歌曲 4 会消耗一张空白卡牌。

题解

本题的考点是边界。

很显然这不是一个随机问题，所以我们要采取最优的方案。

直接出一个方案可能有点困难，但检验答案应该相对简单一点。

所以，我们将问题转化为二分答案。二分答案的正确性是显然的，因为如果能对某个播放顺序成立，当曲目更少的时候也显然成立。

二分什么？我原本打算二分轮数，但我发现末尾的数字实在太难讨论。

然后我去压力大肥鱼，它告诉我建议我直接二分歌曲数，这样就可以直接把零和整都得到，然后一并处理。

这道题最困难的部分我认为应该是对于播放列表的解释。

实际上每一轮播放都剔除了一个数，但是这个“剔除”的事情可就多了。

很显然，相邻两次剔除不能剔除同一个数，这意味着如果一共有 m 轮，一个数最多被剔除 $\lfloor \frac{m}{2} \rfloor$ 次，反过来说，一个数至少出现 $m - \lfloor \frac{m}{2} \rfloor = \lceil \frac{m}{2} \rceil$ 次。

这是一个很重要的限制，所以我们在一开始假定每一张卡牌都出现 $\lceil \frac{m}{2} \rceil$ 次。

AI（具体地，Deepseek V4.1 Pro）给了我一个相当不自然的想法：考虑所有卡牌中，没有被用上，也就是被“浪费”的卡牌数量.

然后使用所有卡牌数量减去浪费，再加上空白卡，就是我们实际能使用的卡牌数量。如果它不少于歌曲数量，我们就可以认为检验成立.

我的疑问是，如果某些卡牌少了，没有达到 $\lceil \frac{m}{2} \rceil$ 次，这个约束怎么成立呢？

后来发现，这个卡牌少了，其他卡牌不会因为这一点就变多，因为它的上限被轮数限制了。所以你就没有足够的卡牌来满足上面的约束.

所以，我们做出这个假设，在此情况下，被浪费的卡牌数量为 $d_i = \max(a_i - \lceil \frac{m}{2} \rceil, 0)$.

不过实际上我们会用更多的卡牌，具体地，我们实际上可以再用 $\min(d_i, m - \lceil \frac{m}{2} \rceil)$ 张卡牌（因为我们使用的卡牌数量不能多于轮数）.

这里还有一个“不过”，我们使用更多卡牌也是有限制的，具体地，最多不能超过 $t - n \cdot \lceil \frac{m}{2} \rceil$ 首歌（这个限制是显然的，我们最多只能用 t 张卡牌）。所以我们减去浪费值的时候也需要减去 $\min(t - n \cdot \lceil \frac{m}{2} \rceil, \min(d_i, m - \lceil \frac{m}{2} \rceil))$.

此时就涉及到末尾的讨论，如果 $a_i \geq b + 1$，意味着我们的末尾部分可以有这首歌，我们进行一个计数。

由于我们的末尾只有 $\lfloor \frac{t-1}{n-1} \rfloor$ 首歌，即使我们有更多的选项，我们也不得不选择至多这么多首歌。

这个题目的讨论相当毒瘤且精巧，认真学习会有很大进步。

cpp

```cpp
#include <stdio.h>
#define maxn 200007

typedef long long i64;
typedef __int128 i128;

int T;
int n;
i64 k, li[maxn], s;

inline i64 max(i64 a, i64 b) { return a > b ? a : b; }
inline i64 min(i64 a, i64 b) { return a < b ? a : b; }

bool check(i64 p) {
    // m = 0, 公式不适用
    if (p < n) {
        i64 cnt_pos = 0;
        for (int i = 1; i <= n; i++)
            if (li[i] >= 1)
                cnt_pos++;
        return p <= cnt_pos + k;
    }

    i64
        // 完整的块数
        blocks = (p - 1) / (n - 1),
        // 此外剩余的数量
        remain = (p - 1) % (n - 1),
        // 每首曲目（实际）最少的出现次数
        min_played = (blocks + 1) / 2,
        // 单曲实际抵扣的最大数量
        max_discount = blocks / 2;

    i64 waste_cards = 0, capped = 0, cnt_big = 0;
    for (int i = 1; i <= n; i++) {
        // 这里若 d>=0，实际含义是我的播放次数多于最少的次数，
        // 这些卡片也就被“浪费”了
```

```
        i64 d = max(li[i] - min_played, 0);
        waste_cards += d;

        // ceil(m/2) + floor(m/2) = m，所以说多出来的部分就是浪费的，但是 capped
        // 记录的部分是“实际没有浪费的卡片”
        capped += min(d, max_discount);

        // blocks+1 意味着可以到最后剩余的部分
        if (li[i] >= blocks + 1)
            cnt_big++;
    }

    // 在剩余的部分内，如果有能播放的曲目，这些卡片也不是“浪费”的
    capped += min(remain, cnt_big);

    i64 ini_waste = p - n * min_played;
    i64 waste = waste_cards - min(ini_waste, capped);

    return p <= k + s - waste;
}

int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%lld", &n, &k);

        s = 0;
        for (int i = 1; i <= n; i++)
            scanf("%lld", li + i), s += li[i];

        i64 l = 0, r = 1e18;
        while (r - l) {
            i64 mid = (l + r + 1) >> 1;
            if (check(mid))
                l = mid;
            else
                r = mid - 1;
        }

        printf("%lld\n", l);
    }
    return 0;
}
```

J.Number Game

题目描述

小 R 正在玩一个数字游戏。

她在黑板上写了 n 个非负整数 $a_1, a_2, ..., a_n$，它们构成可重集 A。

每次操作中，她可以选择黑板上的若干个数，把它们同时修改为它们构成的可重集的 mex。对于由非负整数构成的可重集 S，定义 $\mathrm{mex}(S)$ 为最小的不属于 S 的非负整数。例如，$\mathrm{mex}\{1,2,3\} = 0$，$\mathrm{mex}\{0,1,1,4,5\} = 2$。

她另外有一个可重集 $B = \{b_1, b_2, ..., b_n\}$ 作为这个数字游戏的目标。请你判断她是否能通过有限次操作，使黑板上的数从 A 变为 B。

输入格式

本题包含多组测试数据。

第一行包含一个整数 T $(1 \le T \le 2 \times 10^5)$，表示测试数据组数。

对于每组测试数据：

- 第一行包含一个整数 n $(1 \le n \le 2 \times 10^5)$，表示可重集 A, B 的大小。
- 第二行包含 n 个整数 $a_1, a_2, ..., a_n$ $(0 \le a_i \le 10^9)$，表示可重集 A。
- 第三行包含 n 个整数 $b_1, b_2, ..., b_n$ $(0 \le b_i \le 10^9)$，表示可重集 B。

保证所有测试数据中 n 的总和不超过 2×10^5。

输出格式

对于每组测试数据，输出一行，包含一个字符串 YES 或 NO。如果该组测试数据可以通过有限次操作将 A 变为 B，输出 YES，否则输出 NO。大小写不敏感。

样例

1.in / 1.out

```
4
3
1 2 3
2 2 3
3
0 0 0
2 3 3
3
3 3 2
2 3 3
4
10000 10000 10000 10000
2 3 3 3
```

```
YES
NO
YES
YES
```

在样例的第一组数据中，可以通过以下操作将 $\{1,2,3\}$ 变为 $\{2,2,3\}$：

- 选择元素 2，其 $\mathrm{mex}\{2\} = 0$，得到 $\{1,0,3\}$；
- 再选择元素 1 和 0，其 $\mathrm{mex}\{1,0\} = 2$，得到 $\{2,2,3\}$。

题解

TODO

K.Salary Adjustment

题目描述

你是聪明猪猪公司（Incredibly Clever Pig Corporation，ICPC）的员工。

公司有 n 名员工，编号依次为 $1, 2, ..., n$。员工的上下级关系构成一棵以员工 1 为根的有根树。对于 $i = 2, 3, ..., n$，员工 i 的直接上级是员工 f_i。保证 $1 \le f_i < i$。

公司即将调整全体员工的薪资。员工 i 的当前薪资为 a_i，目标薪资为 b_i。整个调整过程持续恰好 n 天：每天恰好选择一名此前尚未被调整过的员工，并在当天把该员工的薪资由当前值直接改为目标值。因此，调整过程对应一个 1 到 n 的排列 $p = (p_1, p_2, ..., p_n)$，其中 p_k 表示第 k 天被调整的员工。

但是公司对于薪资有着严格的规定。具体地，如果员工 u 是员工 v 的上级，则在任意时刻，员工 u 的薪资不得低于员工 v 的薪资。保证调整前的薪资 a_i 和调整后的薪资 b_i 均满足该项规定。

调整过程对应的排列 p 是合法的当且仅当在每一天的调整完成后，该项规定都能成立。

请你求出合法的排列 p 的数量，答案对 998244353 取模。

输入格式

本题包含多组测试数据。

第一行包含一个整数 T $(1 \le T \le 10^4)$，表示测试数据组数。

对于每组测试数据：

- 第一行包含一个整数 n $(2 \le n \le 2 \times 10^5)$，表示员工数量。
- 第二行包含 $n - 1$ 个整数 $f_2, f_3, ..., f_n$ $(1 \le f_i < i)$，其中 f_i 表示员工 i 的直接上级。
- 第三行包含 n 个整数 $a_1, a_2, ..., a_n$ $(1 \le a_i \le 10^9)$，表示每名员工的当前薪资。
- 第四行包含 n 个整数 $b_1, b_2, ..., b_n$ $(1 \le b_i \le 10^9)$，表示每名员工的目标薪资。

保证所有测试数据中 n 的总和不超过 2×10^5。

输出格式

对于每组测试数据，输出一行，包含一个整数，表示合法的排列 p 的数量对 998244353 取模的结果。

样例

1.in / 1.out

```
2
4
1 1 2
5 3 4 2
7 6 5 4
8
1 1 1 2 5 5 3
6 4 3 5 4 3 4 2
9 5 7 7 2 1 1 5
```

```
4
1680
```

在样例的第一组数据中，上下级关系为：员工 1 是员工 2、3 的直接上级，员工 2 是员工 4 的直接上级。当前薪资为 $(5,3,4,2)$，目标薪资为 $(7,6,5,4)$。

四个合法的排列 p 分别为 $(1,2,3,4)$、$(1,2,4,3)$、$(1,3,2,4)$、$(3,1,2,4)$。

例如 $p=(1,2,3,4)$ 时，薪资变化如下：

$(5,3,4,2) \to (7,3,4,2) \to (7,6,4,2) \to (7,6,5,2) \to (7,6,5,4)$

上述每一天的调整完成后，上级薪资都不低于下级薪资。

题解

这个题感觉就是卡那一步的性质，想通了就做出来。

你思考，建完一棵树，每条边“是否被约束”是可以根据 a_i 和 b_i 直接推算出来的，然后就可以不用这两个值了。

之后呢，我们相当于在构建一个排列，要求有连约束边的点必须排在它的儿子前面。

思考这样一个我们高中学过的性质：

考虑一个排列 $P=(a_1,a_2,...,a_n)$，如果我们要求 $a_1,a_2,...,a_k$ 在 a_n 之前，我们的方案数为 $\frac{n!}{k+1}$。

怎么证明？思考排列组合中“除法消序”的含义。

当这 $k+1$ 个值任意排列的时候，共有 $(k+1)!$ 种排列方式，如果我们要求 a_n 在排列的最后，我们就只剩下 $k!$ 种排列方式。

当然对于不是全排列的情况，这个性质也成立。

所以说，我们只需要统计每个节点有多少全部由“约束边”连接的儿子，记作 c_i，最终的答案为：

$$\frac{n!}{\prod_{i=1}^{n} c_i}$$

其实我们可以将约束边考虑为实边，在这个森林上计算子树大小，答案是等价的，这样想可能直观一点。

```cpp
#include <stdio.h>
#include <vector>
#define maxn 200007

const int mod = 998244353;

using namespace std;

int T, n;

int tsize[maxn], fas[maxn], prv[maxn], nxt[maxn], invs[maxn], facs[maxn],
    facsinv[maxn];
std::vector<int> edge[maxn];

inline int nointersect(int al, int ar, int bl, int br) {
    return max(al, ar) <= min(bl, br);
}

void dfs(int index) {
```

```
    tsize[index] = 1;
    for (auto v : edge[index])
        if (v != fas[index]) {
            dfs(v);
            if (not nointersect(prv[v], nxt[v], prv[index], nxt[index]))
                tsize[index] += tsize[v];
        }
}

int main() {
    invs[1] = 1;
    for (int i = 2; i < maxn; i++)
        invs[i] = (mod - (mod / i)) * 1ll * invs[mod % i] % mod;

    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);

        for (int i = 1; i <= n; i++)
            edge[i].clear(), tsize[i] = 0;

        for (int i = 2; i <= n; i++)
            scanf("%d", fas + i);
        for (int i = 1; i <= n; i++)
            scanf("%d", prv + i);
        for (int i = 1; i <= n; i++)
            scanf("%d", nxt + i);

        for (int i = 2; i <= n; i++)
            edge[fas[i]].push_back(i);

        dfs(1);

        int ans = 1;
        for (int i = 1; i <= n; i++)
            ans = 1ll * ans * i % mod * invs[tsize[i]] % mod;
        printf("%d\n", ans);
    }
    return 0;
}
```