Whalica Cup (Round 2) 题解

说在前面

这里是 Whalica，很抱歉在不懈努力下 Whalica Cup (Round 2) 还是出锅了。

说说最严重的 B 题，作为 B 题的出题人，由于验题主要是在 Codeforces 验的，而 Polygon 导出的 package 的测试数据和 checker 并不在一个文件夹，而录题的肌肉记忆加上我当天缺乏睡眠脑子不太清醒，导致 B 题的 checker 就这样自然地被遗忘了，而再在牛客上检验数据时确实还是不够仔细，对此非常非常抱歉！好在这个问题算是在比较前面发现的，所以还是有一定的挽救余地。最后，对被这个问题影响的所有选手道歉，非常对不起！

另外对提出这个问题的小伙伴致以真心的感谢！

第二题真的有spj吗 问题链接

这场的题的难度个人感觉还是非常变态的，首先题面全员恶人，大部分题目都是大段的文字和保证题目容易被理解的定义，规范性的语言也使得选手在读题上就有很大障碍。其次，从 D 题开始，后面就是板子板题大杂烩，没学习过的很大可能会被卡住而完全停滞，而学过且熟练的几乎就是秒的感觉，所以有出题人认为这场其实比较 edu，而且这也导致了这场的榜比较混乱。这些难点还有其他的一些影响因素（包括出锅）导致这场其实真的没那么简单。

也再次感谢所有来参加 Whalica Cup (Round 2) 的小伙伴，这一场相比 Round 1 我真的投入了太多的精力，感谢你们能来参加这个比赛！下面是所有题的详细题解和参考代码。

A 世界，是充满色彩的

出题人：Whalica (@Whalica)

预估难度：Codeforces 800

解题思路

简单模拟题，考察了十六进制相关知识。

由于色号的 `R`，`G`，`B` 值的取值范围均为 $[0, 255]$，所以所有色号字符串均为 `#RRGGBB` 的形式。只需要把对应位置的数提取出来，假设华黎卡给出的颜色的 `R`，`G`，`B` 值分别为 R, G, B，那么该颜色的互补色的 `R`，`G`，`B` 值就分别为 $255 - R, 255 - G, 255 - B$，再按十六进制输出即可。

时间复杂度：$O(t)$。

参考代码（C++）

```cpp
//Code from Whalica
#include <bits/stdc++.h>

using i64 = long long;
using u64 = unsigned long long;

std::map<char, int> f; // 从字符到整数的映射
std::map<int, char> g; // 从整数到字符的映射

void solve() {
    std::string s;
    std::cin >> s;

    int R = f[s[1]] * 16 + f[s[2]];
```

```
    int G = f[s[3]] * 16 + f[s[4]];
    int B = f[s[5]] * 16 + f[s[6]];
    R = 255 - R;
    G = 255 - G;
    B = 255 - B;

    std::string ans = "#";
    ans.push_back(g[R / 16]);
    ans.push_back(g[R % 16]);
    ans.push_back(g[G / 16]);
    ans.push_back(g[G % 16]);
    ans.push_back(g[B / 16]);
    ans.push_back(g[B % 16]);

    std::cout << ans << "\n";
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    for (int i = 0; i <= 9; i++) {
        f[i + '0'] = i;
        g[i] = i + '0';
    }
    for (int i = 10; i <= 15; i++) {
        f[i - 10 + 'a'] = i;
        g[i] = i - 10 + 'a';
    }

    int t = 1;
    std::cin >> t;

    while (t--) {
        solve();
    }

    return 0;
}
```

B 华黎卡的排列构造

出题人：Whalica (@Whalica)

预估难度：Codeforces 1000 ~ 1400

解题思路

构造题，由于构造题经常出现大个人差的情况，本人特地将预估难度区间拉大了。

我们不妨按照题意打表，打出 $1 \le n \le 10$ 的所有情况，所得结果如下：

n	b
1	$\{1\}$
2	$\{2,1\}$
3	$\{2,1,3\}$
4	$\{3,1,4,2\}$
5	$\{3,1,4,2,5\}$

n	p
6	$\{4,1,5,2,6,3\}$
7	$\{4,1,5,2,6,3,7\}$
8	$\{5,1,6,2,7,3,8,4\}$
9	$\{5,1,6,2,7,3,8,4,9\}$
10	$\{6,1,7,2,8,3,9,4,10,5\}$

不难看出，当 n 为奇数时，可以在 $n-1$ 的排列基础上在最后添上一个 n 构造，而此时我们不妨假设 $1 \le i \le n-1$ 的 i 均满足 $p_i | S_i$，因此我们只需证明 $i=n$ 时也满足此条性质，则说明该构造法合法：

当 $i=n$ 时，$p_n=n, S_n=\dfrac{n(n+1)}{2}$，而 $\dfrac{S_n}{p_n}=\dfrac{n+1}{2}$，此时 n 为奇数，所以 $\dfrac{n+1}{2}$ 是整数，故 $p_i|S_i$ 对于 $i=n$ 也成立，所以该构造法合法。

否则 n 为偶数。通过打表我们可以总结出构造法：

$$p_i=\begin{cases}\dfrac{n}{2}+\dfrac{i+1}{2}, & i \text{ 是奇数}\\ \dfrac{i}{2}, & i \text{ 是偶数}\end{cases}$$

下面我们证明该构造法的正确性：

首先求得数组 S：

$$S_i=\begin{cases}\dfrac{i+1}{2}\cdot(\dfrac{n}{2}+\dfrac{i+1}{2}), & i \text{ 是奇数}\\ \dfrac{i}{2}\cdot(\dfrac{n}{2}+\dfrac{i}{2}+1), & i \text{ 是偶数}\end{cases}$$

观察排列 p 和数组 S 的表达式，我们不难发现 $p_i|S_i$ 对于所有的 $1 \le i \le n$ 是恒成立的，所以该构造法合法。

所以最终排列 p 的构造可以参考上文的表达式，且综上所述，没有无法构造的情况。可以证明，这也是字典序最小的排列。

单组数据的时间复杂度：$O(n)$。

参考代码（C++）

```cpp
//Code from Whalica
#include <bits/stdc++.h>

using i64 = long long;
using u64 = unsigned long long;

void solve() {
    int n;
    std::cin >> n;

    std::vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) {
        if (i % 2 == 1) {
            a[i] = n / 2 + (i + 1) / 2;
        } else {
            a[i] = i / 2;
        }
    }

    if (n % 2 == 1) {
        a.push_back(n);
    }

    std::cout << "YES\n";
    for (int i = 1; i <= n; i++) {
```

```
        std::cout << a[i] << " \n"[i == n];
    }
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int t = 1;
    std::cin >> t;

    while (t--) {
        solve();
    }

    return 0;
}
```

C 七影蝶的指引

出题人：雨沐 (@绫雨沐)

预估难度：Codeforces 1500

解题思路

1. 预处理蝴蝶

首先计算每个节点 u 包含的 “有效蝴蝶” 数量 $W[u]$。只有满足 $H - dep[u] \leq t_u$ 的蝴蝶才有可能被指引。我们将每个节点上满足条件的蝴蝶数量求和，记为 $W[u]$。

2. 定义状态

我们需要通过特殊能力来最大化路径上的 $W[u]$ 之和。

- $uval[u]$：从根节点 1 到节点 u 路径上所有 W 的总和。
  $uval[u] = W[u] + uval[parent(u)]$
- $dval[u]$：从节点 u 开始，向其子树方向延伸到某个叶子节点路径上的 W 最大总和。
  $dval[u] = W[u] + \max_{v \in children(u)}\{dval[v]\}$
  （若 u 是叶子，则 $dval[u] = W[u]$）

3. 计算答案

如果不使用特殊能力，或者特殊能力在同一条垂直路径上使用，最大数量就是所有叶子节点中 $uval[L]$ 的最大值。

如果使用特殊能力进行 “跨枝条” 跳跃：

假设我们在深度 d 处从节点 u 跳到深度 $d - 1$ 的节点 v。

- 收集到的蝴蝶总数为：**（从叶子到 u 的最大蝴蝶数）+（从 v 到根节点的蝴蝶数）**。
- 即：$dval[u] + uval[v]$。

为了最大化结果，对于每一个可能的跳跃深度 d（$1 \leq d \leq H$），我们取：

$\text{Max\_at\_depth\_d} = \left(\max_{dep[u]=d} dval[u]\right) + \left(\max_{dep[v]=d-1} uval[v]\right)$

最终答案即为所有层级跳跃方案与不跳跃方案中的最大值。

时间复杂度：$O(n + m)$。

参考代码（C++）

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

using i64 = long long;

const int N = 200005;
vector<int> adj[N];
vector<int> F[N];
int dep[N], fa[N];
i64 W[N], dval[N], uval[N];
int H;

void Dfs(int u, int p, int d)
{
    dep[u] = d;
    fa[u] = p;
    H = max(H, d);
    F[d].push_back(u);
    for (int v : adj[u])
    {
        if (v != p)
            Dfs(v, u, d + 1);
    }
}

void solve()
{
    int n, m;
    cin >> n >> m;

    H = 0;
    for (int i = 0; i <= n; ++i)
    {
        adj[i].clear();
        F[i].clear();
        W[i] = 0;
        dval[i] = 0;
        uval[i] = 0;
    }

    for (int i = 0; i < n - 1; ++i)
    {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    Dfs(1, 0, 0);

    for (int i = 0; i < m; ++i)
    {
        int a, t;
        cin >> a >> t;
        if (H - dep[a] <= t)
        {
            W[a] += 1;
        }
    }
```

```
    for (int d = H; d >= 0; --d)
    {
        for (int u : F[d])
        {
            i64 mx = 0;
            for (int v : adj[u])
            {
                if (v != fa[u])
                    mx = max(mx, dval[v]);
            }
            dval[u] = W[u] + mx;
        }
    }

    for (int d = 0; d <= H; ++d)
    {
        for (int u : F[d])
        {
            uval[u] = W[u] + uval[fa[u]];
        }
    }

    i64 ans = uval[F[H][0]];

    vector<i64> mxup(H + 1, 0);
    for (int d = 0; d <= H; ++d)
    {
        for (int u : F[d])
            mxup[d] = max(mxup[d], uval[u]);
    }

    for (int d = 1; d <= H; ++d)
    {
        i64 mxd = 0;
        for (int u : F[d])
            mxd = max(mxd, dval[u]);

        ans = max(ans, mxd + mxup[d - 1]);
    }

    cout << ans << endl;
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    int _T;
    cin >> _T;
    while (_T--)
    {
        solve();
    }
    return 0;
}
```

D 华黎卡的开彩游戏

出题人：Inkstrings (@MaxBlazeResFire)

预估难度：Codeforces 1600

解题思路

处理单个字符串与多个字符串的 LCP 之和的问题，最直接的做法就是使用 `trie` 树。

将所有字符串插入 `trie`，插入的过程中每经过一个节点/新建一个节点，就在给这个节点的权值加 1，也即把贡献拆到每个点上。这样当询问串在 `trie` 树上走的时候，就只需要把沿途的权值加起来就可以。注意当询问串走空的时候及时跳出，防止回到开头造成错误统计。

字符的替换可以看作字符集上按替换顺序的循环移位，而字符集上的循环移位不改变字符的相对关系，且是个可逆的操作。不难发现只需要对每个询问统计到当前询问之前共进行了多少次全局循环移位操作，将其对 6 也即字符集大小取余，查询前将查询串 **反向** 循环移位回去再进行查询是完全等价的。

时间复杂度：$O(\sum |s_i| + |\Sigma| \sum |T_i|)$，其中 $|\Sigma|$ 表示字符集大小。

参考代码（C++）

```cpp
#include<bits/stdc++.h>
using namespace std;
#define MAXN 100005
#define int long long
int F[200],nxt[MAXN][6],edp[MAXN];
int n,q,nodecnt,A[MAXN];
char s[MAXN],T[MAXN];

inline void insert(char *s){
    int len=strlen(s+1);
    int now=0;
    for(int i=1;i<=len;i++){
        int ch=F[(int)s[i]];
        if(!nxt[now][ch]) nxt[now][ch]=++nodecnt;
        //每走一步记录一次产生lcp贡献
        now=nxt[now][ch],edp[now]++;
    }
}

inline void solve(){
    scanf("%lld%lld",&n,&q);
    //建trie
    for(int i=1;i<=n;i++) scanf("%s",s+1),insert(s);
    int cyc=0;
    for(int i=1;i<=q;i++){
        int op; scanf("%lld",&op);
        if(op==1){
            scanf("%s",T+1);
            int len=strlen(T+1),now=0,ans=0;
            //全局循环移位看作询问串反向循环移位
            for(int j=1;j<=len;j++) A[j]=(F[(int)T[j]]-cyc+6)%6;
            //走一轮就是答案
            for(int j=1;j<=len;j++){
                now=nxt[now][A[j]];
                ans+=edp[now];
                if(!now) break;
            }
            printf("%lld\n",ans);
        }
        else cyc=(cyc+1)%6;
    }
}
```

```
signed main(){
    //映射
    F['w']=0,F['h']=1,F['a']=2,F['l']=3,F['i']=4,F['c']=5;
    solve();
    return 0;
}
//Assigned by Inkstrings
```

E 八千年的等待

出题人：mahiro_zcy (@mahiro_zcy)

预估难度：Codeforces 1600

解题思路

设 $f(i,v,j)$ 为 $[a_1,a_2,\cdots,a_i]$ 的，最后一项的值等于 v，且闪光次数为 j 的子序列的个数。

注意，由于空子序列没有 "最后一项"，从而 f 的任何状态都是不含空子序列的。

$g(i,j)$ 为 $[a_1,a_2,\cdots,a_i]$ 的，闪光次数为 j 的子序列的个数。

初值

$$\begin{aligned}\forall 1\le v\le n,0\le j\le k,f(0,v,j)&=0\\ g(0,0)&=1\\ \forall 1\le j\le k,g(0,j)&=0\end{aligned}$$

转移时，只需要考虑新增的子序列，即以 a_i 结尾的子序列即可。

$$\begin{aligned}\forall 1\le i\le n,1\le v\le n,v\ne a_i,0\le j\le k,f(i,v,j)&=f(i-1,v,j)\\ \forall 1\le i\le n,f(i,a_i,0)&=f(i-1,a_i,0)+g(i-1,0)-f(i-1,a_i,0)\\ &=g(i-1,0)\\ \forall 1\le i\le n,1\le j\le k,f(i,a_i,j)&=f(i-1,a_i,j)+g(i-1,j)-f(i-1,a_i,j)+f(i-1,a_i,j-1)\\ &=g(i-1,j)+f(i-1,a_i,j-1)\\ \forall 1\le i\le n,0\le j\le k,g(i,j)&=g(i-1,j)+f(i,a_i,j)-f(i-1,a_i,j)\end{aligned}$$

答案为 $g(n,k)$。

由于转移只涉及 $f(i,a_i,*)$ 和 $g(i,*)$ 的计算，从 $i-1$ 转移到 i 的时间复杂度是 $O(k)$ 。

从而总的时间复杂度是 $O(nk)$。

实现时，不需要开 i 这一维，采用滚动更新即可，这样实现的空间复杂度也是 $O(nk)$，就可以通过了。

参考代码（C++）

```
#include <bits/stdc++.h>

using i64 = long long;
using u64 = unsigned long long;
using u32 = unsigned int;

template<class T>
constexpr T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}

template<int P>
struct MInt {
```

```
    int x;
    constexpr MInt() : x{} {}
    constexpr MInt(i64 x) : x{norm(x % getMod())} {}

    static int Mod;
    constexpr static int getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(int Mod_) {
        Mod = Mod_;
    }
    constexpr int norm(int x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
        return x;
    }
    constexpr int val() const {
        return x;
    }
    explicit constexpr operator int() const {
        return x;
    }
    constexpr MInt operator-() const {
        MInt res;
        res.x = norm(getMod() - x);
        return res;
    }
    constexpr MInt inv() const {
        assert(x != 0);
        return power(*this, getMod() - 2);
    }
    constexpr MInt &operator*=(MInt rhs) & {
        x = 1LL * x * rhs.x % getMod();
        return *this;
    }
    constexpr MInt &operator+=(MInt rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    constexpr MInt &operator-=(MInt rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr MInt &operator/=(MInt rhs) & {
        return *this *= rhs.inv();
    }
    friend constexpr MInt operator*(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr MInt operator+(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr MInt operator-(MInt lhs, MInt rhs) {
```

```
        MInt res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr MInt operator/(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr std::istream &operator>>(std::istream &is, MInt &a) {
        i64 v;
        is >> v;
        a = MInt(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MInt lhs, MInt rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MInt lhs, MInt rhs) {
        return lhs.val() != rhs.val();
    }
};

template<>
int MInt<0>::Mod = 998244353;

template<int V, int P>
constexpr MInt<P> CInv = MInt<P>(V).inv();

constexpr int P = 998244353;
using Z = MInt<P>;

void solve() {
    int n, k;
    std::cin >> n >> k;

    std::vector<int> a(n, 0);
    for (int i = 0; i < n; ++i) {
        std::cin >> a[i];
        --a[i];
    }

    std::vector<std::vector<Z>> f(n, std::vector<Z>(k + 1, 0));
    std::vector<Z> g(k + 1, 0);
    g[0] = 1;
    for (int i = 0; i < n; ++i) {
        std::vector<Z> nf(k + 1, 0), ng(k + 1, 0);
        for (int j = 0; j <= k; ++j) {
            if (j == 0) {
                nf[j] = g[j];
            } else {
                nf[j] = g[j] + f[a[i]][j - 1];
            }
            ng[j] = g[j] + nf[j] - f[a[i]][j];
        }
        f[a[i]] = nf;
        g = ng;
    }

    std::cout << g[k] << "\n";
}
```

```cpp
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int t = 1;
    std::cin >> t;

    while (t--) {
        solve();
    }

    return 0;
}
```

F 魔学

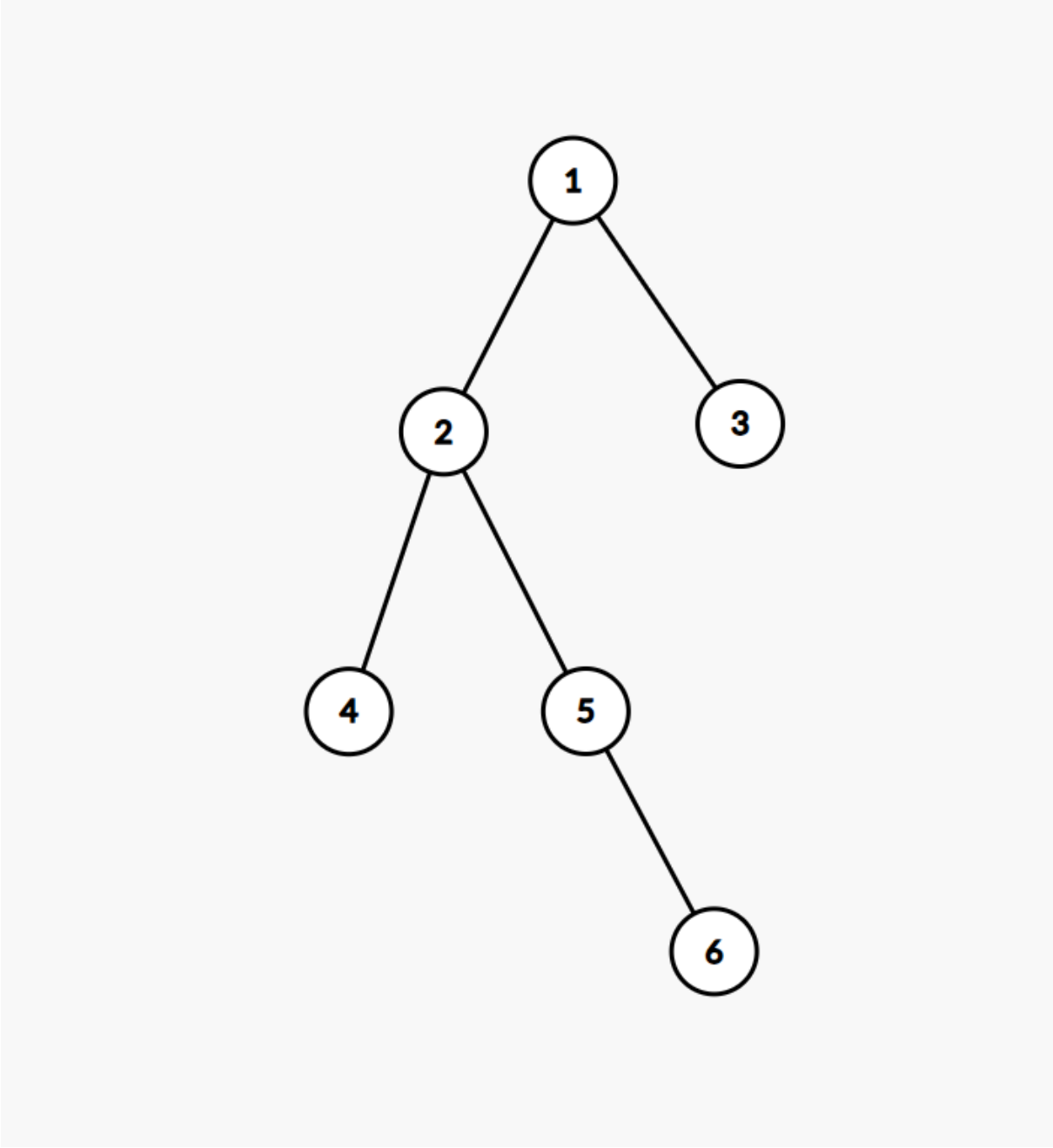
出题人：mahiro_zcy (@mahiro_zcy)

预估难度：Codeforces 1700 ~ 1900

解题思路

考虑 $k \geq 1$ 时，对于每个 $v \in \text{Subtree}(u)$，a_v 在 $x_u^{(k)}$ 中的系数是多少.

例如，对于下面的树，考察 a_6 在 $x_1^{(3)}$ 中的系数是多少.



可以采用回溯的方式来计算：

$x_1^{(3)}$ 中的 a_6 可以来自 $x_1^{(2)}$，$x_2^{(2)}$，$x_5^{(2)}$，$x_6^{(2)}$

$x_1^{(2)} \to x_1^{(3)}$ 中的 a_6 可以来自 $x_1^{(1)}$，$x_2^{(1)}$，$x_5^{(1)}$，$x_6^{(1)}$

$x_2^{(2)} \to x_1^{(3)}$ 中的 a_6 可以来自 $x_2^{(1)}$，$x_5^{(1)}$，$x_6^{(1)}$

$x_5^{(2)} \to x_1^{(3)}$ 中的 a_6 可以来自 $x_5^{(1)}$，$x_6^{(1)}$

$x_6^{(2)} \to x_1^{(3)}$ 中的 a_6 可以来自 $x_6^{(1)}$

$x_1^{(1)} \to x_1^{(2)} \to x_1^{(3)}$ 中的 a_6 可以来自 $x_6^{(0)}$，带来 1 个 a_6

$x_2^{(1)} \to x_1^{(2)} \to x_1^{(3)}$ 中的 a_6 可以来自 $x_6^{(0)}$，带来 1 个 a_6

$x_5^{(1)} \to x_1^{(2)} \to x_1^{(3)}$ 中的 a_6 可以来自 $x_6^{(0)}$，带来 1 个 a_6

$x_6^{(1)} \to x_1^{(2)} \to x_1^{(3)}$ 中的 a_6 可以来自 $x_6^{(0)}$，带来 1 个 a_6

$x_2^{(1)} \to x_2^{(2)} \to x_1^{(3)}$ 中的 a_6 可以来自 $x_6^{(0)}$，带来 1 个 a_6

$x_5^{(1)} \to x_2^{(2)} \to x_1^{(3)}$ 中的 a_6 可以来自 $x_6^{(0)}$，带来 1 个 a_6

$x_6^{(1)} \to x_2^{(2)} \to x_1^{(3)}$ 中的 a_6 可以来自 $x_6^{(0)}$，带来 1 个 a_6

$x_5^{(1)} \to x_5^{(2)} \to x_1^{(3)}$ 中的 a_6 可以来自 $x_6^{(0)}$，带来 1 个 a_6

$x_6^{(1)} \to x_5^{(2)} \to x_1^{(3)}$ 中的 a_6 可以来自 $x_6^{(0)}$，带来 1 个 a_6

$x_6^{(1)} \to x_6^{(2)} \to x_1^{(3)}$ 中的 a_6 可以来自 $x_6^{(0)}$，带来 1 个 a_6

因此，$x_1^{(3)}$ 中 a_6 的系数是 10，分别来自下面的过程：

$$
\begin{aligned}
&x_6^{(0)} \to x_1^{(1)} \to x_1^{(2)} \to x_1^{(3)}\\
&x_6^{(0)} \to x_2^{(1)} \to x_1^{(2)} \to x_1^{(3)}\\
&x_6^{(0)} \to x_5^{(1)} \to x_1^{(2)} \to x_1^{(3)}\\
&x_6^{(0)} \to x_6^{(1)} \to x_1^{(2)} \to x_1^{(3)}\\
&x_6^{(0)} \to x_2^{(1)} \to x_2^{(2)} \to x_1^{(3)}\\
&x_6^{(0)} \to x_5^{(1)} \to x_2^{(2)} \to x_1^{(3)}\\
&x_6^{(0)} \to x_6^{(1)} \to x_2^{(2)} \to x_1^{(3)}\\
&x_6^{(0)} \to x_5^{(1)} \to x_5^{(2)} \to x_1^{(3)}\\
&x_6^{(0)} \to x_6^{(1)} \to x_5^{(2)} \to x_1^{(3)}\\
&x_6^{(0)} \to x_6^{(1)} \to x_6^{(2)} \to x_1^{(3)}
\end{aligned}
$$

因此，要计算 a_v 在 $x_u^{(k)}$ 中的系数是多少，只需要计算有多少个这样的过程.

我们记从 v 到 u 的简单路径上有 d 条边，即形成 $p_1, p_2, p_3, \cdots, p_d, p_{d+1}$ 的路径，其中 $p_1 = v, p_{d+1} = u$，那么，我们的过程就可以写成：

$$
x_v^{(0)} \to x_{p_{l_1}}^{(1)} \to x_{p_{l_2}}^{(2)} \to \cdots \to x_{p_{l_{k-1}}}^{(k-1)} \to x_u^{(k)}
$$

约束条件是

$$
1 \le l_1 \le l_2 \le \cdots \le l_{k-1} \le d+1
$$

我们只需要统计有多少个这样的 $(l_1, l_2, \cdots, l_{k-1})$ 即可。设 $f_i = l_i + i$，则上式变为

$$
2 \le f_1 < f_2 < \cdots < f_{k-1} \le d+k
$$

也就是从 $\{2, 3, \cdots, d+k\}$ 中选取 $k-1$ 个互不相等的整数，因此选法种数是

$$
\binom{d+k-1}{k-1}
$$

所以，a_v 在 $x_u^{(k)}$ 中的系数是

$$
c_d = \binom{d+k-1}{k-1}
$$

记 $D(u)$ 为结点 u 的深度，即 u 到根结点的最短路径的边数。则上式中 $d = D(v) - D(u)$.

接下来考虑实现.

可以先预处理

$$
\begin{aligned}
c_0 &= 1\\
\forall d \ge 1, c_d &= \frac{d+k-1}{d} c_{d-1}
\end{aligned}
$$

该预处理对 $k = 0$ 也是成立的.

最多只需要处理到 $d = \max\limits_{1 \le i \le n} D(i)$，即整个树的最大深度差 h.

然后可以直接暴力计算每个点

$$
x_u^{(k)} = \sum_{v \in \text{Subtree}(u)} c_{D(v)-D(u)} a_v
$$

这里，枚举子树中的点，可以通过 DFS 序等方法实现.

对于每个 v，满足 $v \in \text{Subtree}(u)$ 的 u 的个数是 $D(v)+1 \leq h+1$。

因此，这个做法的时间复杂度是 $O(nh)$，最坏情况下为 $O(n^2)$，可以通过.

参考代码（C++）

```cpp
#include <bits/stdc++.h>

using i64 = long long;
using u64 = unsigned long long;
using u32 = unsigned int;

using i128 = __int128;
using u128 = unsigned __int128;

template<class T>
constexpr T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}

template<int P>
struct MInt {
    int x;
    constexpr MInt() : x{} {}
    constexpr MInt(i64 x) : x{norm(x % getMod())} {}

    static int Mod;
    constexpr static int getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(int Mod_) {
        Mod = Mod_;
    }
    constexpr int norm(int x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
        return x;
    }
    constexpr int val() const {
        return x;
    }
    explicit constexpr operator int() const {
        return x;
    }
    constexpr MInt operator-() const {
        MInt res;
        res.x = norm(getMod() - x);
        return res;
    }
    constexpr MInt inv() const {
        assert(x != 0);
```

```
        return power(*this, getMod() - 2);
    }
    constexpr MInt &operator*=(MInt rhs) & {
        x = 1LL * x * rhs.x % getMod();
        return *this;
    }
    constexpr MInt &operator+=(MInt rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    constexpr MInt &operator-=(MInt rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr MInt &operator/=(MInt rhs) & {
        return *this *= rhs.inv();
    }
    friend constexpr MInt operator*(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr MInt operator+(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr MInt operator-(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr MInt operator/(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr std::istream &operator>>(std::istream &is, MInt &a) {
        i64 v;
        is >> v;
        a = MInt(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MInt lhs, MInt rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MInt lhs, MInt rhs) {
        return lhs.val() != rhs.val();
    }
};

template<>
int MInt<0>::Mod = 998244353;

template<int V, int P>
constexpr MInt<P> CInv = MInt<P>(V).inv();

constexpr int P = 998244353;
using Z = MInt<P>;

struct Comb {
    int n;
```

```
    std::vector<Z> _fac;
    std::vector<Z> _invfac;
    std::vector<Z> _inv;

    Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
    Comb(int n) : Comb() {
        init(n);
    }

    void init(int m) {
        if (m <= n) return;
        _fac.resize(m + 1);
        _invfac.resize(m + 1);
        _inv.resize(m + 1);

        for (int i = n + 1; i <= m; i++) {
            _fac[i] = _fac[i - 1] * i;
        }
        _invfac[m] = _fac[m].inv();
        for (int i = m; i > n; i--) {
            _invfac[i - 1] = _invfac[i] * i;
            _inv[i] = _invfac[i] * _fac[i - 1];
        }
        n = m;
    }

    Z fac(int m) {
        if (m > n) init(2 * m);
        return _fac[m];
    }
    Z invfac(int m) {
        if (m > n) init(2 * m);
        return _invfac[m];
    }
    Z inv(int m) {
        if (m > n) init(2 * m);
        return _inv[m];
    }
    Z binom(int n, int m) {
        if (n < m || m < 0) return 0;
        return fac(n) * invfac(m) * invfac(n - m);
    }
} comb;

void solve() {
    i64 n, k;
    std::cin >> n >> k;

    std::vector<Z> a(n, 0);
    for (int i = 0; i < n; ++i) {
        std::cin >> a[i];
    }

    std::vector<std::vector<int>> g(n, std::vector<int>());
    for (int i = 0; i < n - 1; ++i) {
        int u, v;
        std::cin >> u >> v;
        --u, --v;
        g[u].push_back(v);
        g[v].push_back(u);
    }

    std::vector<int> d(n, 0);
    std::vector<int> node(n, 0);
    std::vector<int> l(n, 0);
```

```
    std::vector<int> r(n, 0);
    int now = -1;
    auto dfs = [&](auto &&self, int u, int p) -> void {
        ++now;
        node[now] = u;
        l[u] = now;
        for (int v : g[u]) {
            if (v != p) {
                d[v] = d[u] + 1;
                self(self, v, u);
            }
        }
        r[u] = now;
    };
    dfs(dfs, 0, -1);

    int h = *std::max_element(d.begin(), d.end());
    std::vector<Z> c(h + 1, 0);
    c[0] = 1;
    for (int i = 1; i <= h; ++i) {
        c[i] = c[i - 1] * (i + k - 1) * comb.inv(i);
    }

    std::vector<Z> ans(n, 0);
    for (int u = 0; u < n; ++u) {
        for (int i = l[u]; i <= r[u]; ++i) {
            int v = node[i];
            ans[u] += c[d[v] - d[u]] * a[v];
        }
    }

    for (int i = 0; i < n; ++i) {
        std::cout << ans[i] << " \n"[i == n - 1];
    }
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int t = 1;
    std::cin >> t;

    while (t--) {
        solve();
    }

    return 0;
}
```

G 真蓝

出题人：小七爱英语 (@小七爱英语)

预估难度：Codeforces 1700 ~ 1800

解题思路

维护区间 $\max$ 线段树即可。

设询问的区间为 $[l, r]$，询问的值为 x，则先递归查询出 $l \le L$ 且 $R \le r$ 的区间，如果该区间 $\max \ge x$，则暴力递归。

因为对于任何一个将被删除的节点，递归将其设置为 0 并更新的时间复杂度是 $O(\log n)$，暴力递归到叶子节点的次数不超过 n 次，所以时间复杂度为 $O((n+q)\log n)$。

参考代码（C++）

```cpp
#include <bits/stdc++.h>

using namespace std;

using i64 = int64_t;

constexpr int MAXN = 2e5, N = MAXN + 4;

int b[N];
bitset<N> del$;

constexpr int lb(int x) { return x & -x; }
constexpr int rb(int x) noexcept { return x - 1 & 1 - x; } // lb(x-1)

basic_string<int> tr[N], tl[N];

template <typename T>
i64 op(T &v, int x) {
    i64 res = 0;
    while (!v.empty() and b[v.back()] >= x) {
        if (!del$[v.back()]) {
            del$[v.back()] = true;
            res += b[v.back()];
        }
        v.pop_back();
    }
    return res;
}

i64 ask(int l, int r, int x) { // 双·树状数组，缺点是不能访问 1 号点
    i64 res = 0;
    while (l - 1 <= r - lb(r)) res += op(tl[r], x), r -= lb(r);
    while (l + rb(l) <= r + 1) res += op(tr[l], x), l += rb(l);
    return res;
}

void solve() {
    int n, q;
    cin >> n >> q, ++ n;
    for (int i = 2; i <= n; ++ i) {
        cin >> b[i];
    }
    for (int i = 2; i <= n; ++ i) {
        tl[i].reserve(lb(i));
        tr[i].reserve(rb(i));
    }
    {
        vector<int> t(n - 1);
        iota(t.begin(), t.end(), 2);
        std::sort(t.begin(), t.end(), [](int lhs, int rhs) { return b[lhs] < b[rhs]; });
        for (int i : t) {
            for (int x = i; x <= n; x += lb(x))
                tl[x] += i;
        }
        for (int i : t) {
            for (int x = i; x >= 2; x -= rb(x))
                tr[x] += i;
        }
    }
    while (q --) {
        int l, r, x;
        cin >> l >> r >> x, ++ l, ++ r;
```

```
        cout << ask(l, r, x) << '\n';
    }
}

signed main() {
    int T = 1;
    cin.tie(0)->sync_with_stdio(false);
    while (T --) {
        solve();
    }
    return 0;
}
```

H 华黎卡的神秘集合

出题人：Whalica (@Whalica)

预估难度：Codeforces 1900 ~ 2000

解题思路

一道稍微绕了一点的数据结构题。

分析操作 $1, 2$ 可知我们需要维护一个 **出现** 的数的集合 S，而分析操作 $3, 4$ 可知，我们需要维护一个 **未出现** 的数的集合 T。但是在要求上，操作 $3, 4$ 的要求明显比操作 $1, 2$ 的要求更复杂，所以我们不妨只维护 **未出现** 的数的集合 T，初始集合 T 有从 1 到 n 的所有元素。

我们不妨维护一个长度为 n 的数组 cnt，$cnt_i = 0$ 表示数字 i 出现，$cnt_i = 1$ 表示数字 i 未出现。这样，操作 $1, 2$ 就变成了数组 cnt 上的单点修改，操作 3 就可以变为查找 cnt 数组从左到右第一个大于 0 的位置。这个步骤可以使用线段树上二分实现，也就是用线段树维护 cnt 数组。可以证明，在这种写法下，操作 $1, 2, 3$ 的单次复杂度均为 $O(\log n)$。比赛中观测到，操作 3 的维护也可以使用 `std::set` 实现。

接下来考虑操作 4，每次遍历 cnt 数组求和因为最坏复杂度太高肯定不太现实，我们不妨考虑操作 $1, 2$ 对原集合权值的影响，由于操作无效的情况可以通过检查 cnt_x 的值简单实现，我们下面仅讨论操作有效的情况。方便起见，我们定义元素 i 的权值为 $i \cdot 2^j$，其中 j 表示元素 i 是集合 T 中从小到大第 j 个元素。

- 对于每次操作 1，元素 x 被插入原集合 S，相当于被从集合 T 中删除。元素 x 的删除将导致在集合 T 中所有比 x 大的元素的权值除以 2，而元素 x 的权值从集合 S 的权值直接删去。
- 对于每次操作 2，元素 x 被从原集合 S 中删除，相当于被插入集合 T。元素 x 的加入将导致在集合 T 中所有比 x 大的元素的权值乘以 2，而元素 x 的权值直接加入集合 S 的权值。

以上所述，相当于我们在实现一个区间加、乘的操作，而操作 4 又相当于区间求和的操作，从而我们很容易联想到线段树之类的数据结构。

具体地，我们用线段树维护一个长度为 n 的 seg 数组，代表所有元素自己的权值，操作 4 就等效为对区间 $[1, n]$ 内所有元素的权值求和，可以证明操作 $1, 2$ 在维护 seg 数组时的时间复杂度为 $O(\log n)$，操作 4 本身的时间复杂度也为 $O(\log n)$。

综上所述，总时间复杂度为 $O(q \log n)$，可以通过本题。另外，比赛时观测到还存在类似根号复杂度的解法。

参考代码（C++）

```
//Code from Whalica
#include <bits/stdc++.h>

using i64 = long long;
using u64 = unsigned long long;

constexpr i64 P = 998244353;

i64 power(i64 a, i64 b) {
    a %= P;
    i64 res = 1;
    for (; b; a = a * a % P, b >>= 1) {
        if (b & 1) {
```

```
            res = res * a % P;
        }
    }
    return res;
}

i64 inv(i64 a) {
    assert(a != 0);
    return power(a, P - 2);
}

template <class Info, class Tag>
class LazySegmentTree { // 1 - index
public:
    template <class Container>
    LazySegmentTree(const Container& c) : n(c.size() - 1), info((n + 1) << 2, Info()), tag((n + 1)
<< 2, Tag()) {
        auto build = [&](auto&& build, int l, int r, int x) -> void {
            if (l == r) {
                info[x] = Info(l, c[l]);
                return;
            }
            int mid = (l + r) >> 1;
            build(build, l, mid, x << 1);
            build(build, mid + 1, r, x << 1 | 1);
            pushup(x);
        };

        if (n >= 1) {
            build(build, 1, n, 1);
        }
    }

    void rangeApply(int l, int r, const Tag& t) {
        RangeApply(l, r, t, 1, n, 1);
    }

    Info rangeQuery(int l, int r) {
        return RangeQuery(l, r, 1, n, 1);
    }

    void Modify(int p, const Tag& t) {
        RangeApply(p, p, t, 1, n, 1);
    }

    template<class F>
    int findFirst(int l, int r, F&& f) {
        return findFirst(l, r, 1, n, 1, f);
    }

    template<class F>
    int findLast(int l, int r, F&& f) {
        return findLast(l, r, 1, n, 1, f);
    }

private:
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;

    void pushup(int x) {
        info[x] = info[x << 1] + info[x << 1 | 1];
    }

    void pushdown(int l, int r, int x) {
```

```
        info[x << 1].Apply(tag[x]);
        info[x << 1 | 1].Apply(tag[x]);
        tag[x << 1].Apply(tag[x]);
        tag[x << 1 | 1].Apply(tag[x]);
        tag[x].status = false;
    }

    void RangeApply(int l, int r, const Tag& t, int s, int e, int x) {
        if (s >= l && e <= r) {
            info[x].Apply(t);
            tag[x].Apply(t);
            return;
        }
        pushdown(s, e, x);
        int mid = (s + e) >> 1;
        if (mid >= l) {
            RangeApply(l, r, t, s, mid, x << 1);
        }
        if (mid + 1 <= r) {
            RangeApply(l, r, t, mid + 1, e, x << 1 | 1);
        }
        pushup(x);
    }

    Info RangeQuery(int l, int r, int s, int e, int x) {
        if (s >= l && e <= r) {
            return info[x];
        }
        pushdown(s, e, x);
        Info ans = Info();
        int mid = (s + e) >> 1;
        if (mid >= l) {
            ans = (ans + RangeQuery(l, r, s, mid, x << 1));
        }
        if (mid + 1 <= r) {
            ans = (ans + RangeQuery(l, r, mid + 1, e, x << 1 | 1));
        }
        return ans;
    }

    template<class F>
    int findFirst(int l, int r, int s, int e, int x, F&& f) {
        if (e < l || s > r) {
            return -1;
        }
        if (s >= l && e <= r && f(info[x]) == false) {
            return -1;
        }
        if (s == e) {
            return s;
        }
        pushdown(s, e, x);
        int mid = (s + e) >> 1;
        int res = findFirst(l, r, s, mid, x << 1, f);
        if (res == -1) {
            res = findFirst(l, r, mid + 1, e, x << 1 | 1, f);
        }

        return res;
    }

    template<class F>
    int findLast(int l, int r, int s, int e, int x, F&& f) {
        if (e < l || s > r) {
            return -1;
```

```
        }
        if (s >= l && e <= r && f(info[x]) == false) {
            return -1;
        }
        if (s == e) {
            return s;
        }
        pushdown(s, e, x);
        int mid = (s + e) >> 1;
        int res = findLast(l, r, mid + 1, e, x << 1 | 1, f);
        if (res == -1) {
            res = findLast(l, r, s, mid, x << 1, f);
        }

        return res;
    }
};

struct Tag {
    i64 add;
    i64 mul;
    bool status = false;

    Tag() : add(0), mul(1) {}
    Tag(i64 add_, i64 mul_) : add(add_), mul(mul_), status(true) {}

    void Apply(const Tag& t) {
        if (!t.status) {
            return;
        }
        if (!status) {
            (*this) = t;
            return;
        }
        mul = (mul * t.mul) % P;
        add = add * t.mul % P;
        add = (add + t.add + P) % P;
    }
};

struct Info {
    int l, r;
    i64 val;

    Info() : l(0), r(0), val(0) {}
    Info(int x, i64 v) : l(x), r(x), val(v) {}

    void Apply(const Tag& t) {
        if (!t.status) {
            return;
        }
        val = val * t.mul % P;
        val = (val + t.add * (r - l + 1) % P + P) % P;
    }

    friend Info operator+(const Info& info1, const Info& info2) {
        if (info1.l == 0) {
            return info2;
        }
        Info res;
        res.l = info1.l;
        res.r = info2.r;
        res.val = (info1.val + info2.val + P) % P;

        return res;
```

```
    }
};

constexpr int N = 1E5;

std::vector<i64> Pow(N + 1);

void solve() {
    int n, q;
    std::cin >> n >> q;

    LazySegmentTree<Info, Tag> cnt(std::vector<int>(n + 1, 1));
    LazySegmentTree<Info, Tag> seg(std::vector<i64>(n + 1));
    for (int i = 1; i <= n; i++) {
        seg.Modify(i, Tag(1LL * i * Pow[i] % P, 1));
    }
    for (int i = 0; i < q; i++) {
        int op;
        std::cin >> op;

        if (op == 1) {
            int x;
            std::cin >> x;

            if (cnt.rangeQuery(x, x).val == 0) {
                continue;
            }
            seg.rangeApply(x, n, Tag(0, inv(2)));
            seg.Modify(x, Tag((P - seg.rangeQuery(x, x).val) % P, 1));
            cnt.Modify(x, Tag((P - 1) % P, 1));
        }
        if (op == 2) {
            int x;
            std::cin >> x;

            if (cnt.rangeQuery(x, x).val == 1) {
                continue;
            }
            int precnt = cnt.rangeQuery(1, x).val + 1;
            seg.rangeApply(x, n, Tag(0, 2));
            seg.Modify(x, Tag(x * Pow[precnt] % P, 1));
            cnt.Modify(x, Tag(1, 1));
        }
        if (op == 3) {
            std::cout << cnt.findFirst(1, n, [&](const Info& info) -> bool {
                return info.val > 0;
            }) << "\n";
        }
        if (op == 4) {
            std::cout << seg.rangeQuery(1, n).val << "\n";
        }
    }
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    Pow[0] = 1;
    for (int i = 1; i <= N; i++) {
        Pow[i] = Pow[i - 1] * 2 % P;
    }

    int t = 1;
    // std::cin >> t;
```

```
    while (t--) {
        solve();
    }

    return 0;
}
```

I 萤火引路

出题人：mahiro_zcy (@mahiro_zcy)

预估难度：Codeforces 2000 ~ 2200

解题思路

考虑动态规划。

设 $f(s,i)$ 是通过**不超过** s 次 "萤火引路" 到达第 i 盏灯所需的最少体力。

初值：

1. 第 1 盏灯不需要任何操作就能到达

$$\forall 0 \le s \le k, f(s,1) = 0$$

2. 如果不使用 "萤火引路"，则只能通过 "沿岸漫步" 一步步到达 i

$$\forall 2 \le i \le n, f(0,i) = \sum_{j=1}^{i-1} a_j$$

接下去考虑转移。

对于 $1 \le s \le k$，$2 \le i \le n$，考虑如何通过不超过 s 次 "萤火引路" 到达第 i 盏灯。枚举最后一步操作：

1. 通过不超过 s 次 "萤火引路" 到达 $i-1$，再 "沿岸漫步" 到达 i。
2. 通过不超过 $s-1$ 次 "萤火引路" 到达 j，这里 $1 \le j \le i-1$，再 "萤火引路" 到达 i。

因此

$$\forall 1 \le s \le k, 2 \le i \le n, f(s,i) = \min\left\{f(s,i-1) + a_{i-1}, \min_{1 \le j \le i-1}\left\{f(s-1,j) + b_j + (x_i - x_j)^2\right\}\right\}$$

考虑如何计算

$$\min_{1 \le j \le i-1}\left\{f(s-1,j) + b_j + (x_i - x_j)^2\right\}$$

先分离与 j 无关的项，得到

$$x_i^2 + \min_{1 \le j \le i-1}\left\{-2x_j x_i + f(s-1,j) + b_j + x_j^2\right\}$$

设

$$h(t) = \min_{1 \le j \le i-1}\left\{-2x_j t + f(s-1,j) + b_j + x_j^2\right\}$$

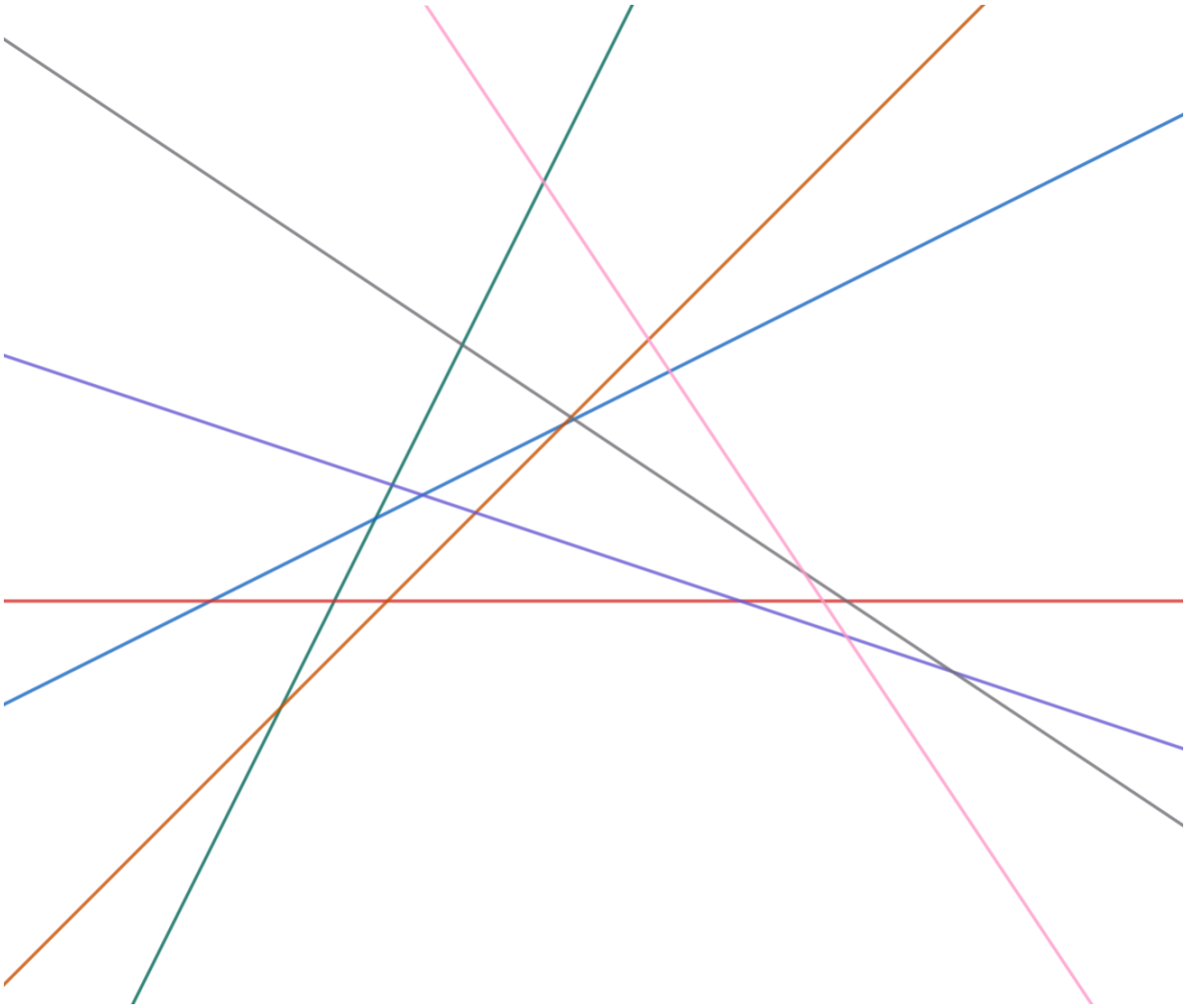
我们要求 $h(x_i)$。可以发现这其实就是 $i-1$ 个一次函数的 $\min$。

考虑如何维护这些一次函数。

这里发生的事情其实就是，到达 i 时，会添加一个 $j = i-1$ 对应的一次函数，然后查询 x_i 处的值。

我们不妨从几何角度来思考如何处理添加直线和查询。

为方便理解，我们举一个例子，画图如下：



由于第 j 条直线的斜率是 $-2x_j$，而 x_j 随 j 严格单调增加，所以 j 对应的直线的斜率随 j 严格单调减少。

因此，可以确定这些直线按 j 从小到大的顺序是：绿色直线，橙色直线，蓝色直线，红色直线，紫色直线，灰色直线，粉色直线。

约定一些用词：

直线 l 的有效区域 满足该条件的 t 的集合：直线 l 的取值在 t 处小于或等于其余所有的直线在 t 处的取值。

有效直线 有效区域不为空集的直线。

在上图中，绿色直线，橙色直线，红色直线，紫色直线，粉色直线 都是有效直线，而蓝色直线，灰色直线 不是有效直线。

由图可知，一个有效直线的有效区域，必然是一个区间。

并且由于越大的 j 对应的直线的斜率越小，所以，对于 2 个有效直线，j 更大的那条的有效区间在 j 更小的那条的有效区间的右边。

考虑把有效直线存成一个序列 $F_1, F_2, \cdots, F_m$，按 j 从小到大排列，即 F_1 是有效直线中 j 最小的那个。

可以发现：

1. t 在 F_1 的有效区域，当且仅当 $m = 1$ 或 $F_1(t) \le F_2(t)$。
2. 对于 $2 \le p \le m-1$，t 在 F_p 的有效区域，当且仅当 $F_p(t) \le F_{p-1}(t)$ 且 $F_p(t) \le F_{p+1}(t)$。
3. t 在 F_m 的有效区域，当且仅当 $m = 1$ 或 $F_m(t) \le F_{m-1}(t)$。

接下去考虑如何进行具体操作。

1. 查询

   朴素的想法是直接对 $p = 1, 2, \cdots, m$ 依次判定 x_i 是否在 F_p 的有效区域，直到判定到某个在的为止。

   但这样的时间复杂度过高，需要优化。

   由于查询点 x_i 随着 i 单调增加，如果此时 x_i 在某个直线的有效区域右侧，之后的查询点也不会处于这条直线的有效区域，可以从 F 中丢弃这条直线，即之后可以认为不存在这条直线。

这带来了下面的算法：

如果 x_i 处于 F 的第一项的有效区域，得到答案，结束。否则，从 F 中丢弃这一项，重复本步骤。

2. 添加直线

只需要考虑添加这条直线后哪些直线是有效直线。

记要加入的直线是 L。

由于 j 越大斜率越小，L 是斜率最小的，t 足够大就会进入他的有效区域，所以 L 一定是有效直线。

接下来考虑哪些之前的有效直线，在加入这条直线后还是有效直线。

考虑判断之前的直线 F_p 在加入 L 后是否仍然是有效直线。

首先 F_1 仍然是有效直线，因为 F_1 是斜率最大的，只要 t 足够小就会进入他的有效区域。接下来考虑对于 $p \geq 2$ 的判定。

记 $X(l_1, l_2)$ 为 l_1 和 l_2 的交点的横坐标，即满足 $l_1(t) = l_2(t)$ 的 t。

作图可以得到，如果 $X(F_{p-1}, F_p) \leq X(F_p, L)$，则 F_p 仍然是有效直线，否则不是。

同时还会发现：

1. 对于任意 $2 \leq p \leq m$，如果 F_p 仍然是有效直线，则 F_{p-1} 也仍然是有效直线。
2. 对于任意 $1 \leq p \leq m-1$，如果 F_p 不再是有效直线，则 F_{p+1} 也不再是有效直线。

因此，被去掉的其实是 F 的一个后缀，效率最高的方法就是从后面往前面删。

这带来了下面的算法：

如果 F 的最后一项，在加入 L 后仍然是有效直线，将 L 加入 F 的末端，结束。否则，从 F 中丢弃这一项，重复本步骤。

可以使用双端队列实现这个 F。注意上面比较交点横坐标的过程，应该写出式子后移项，变成纯整数之间的比较，避免浮点误差。

这样我们就能实现之前的转移了。

在从 $s-1$ 到 s 的转移的整个过程中，因为每个直线都被加入 1 次和至多被丢弃 1 次，维护这些直线的总时间复杂度是 $O(n)$。

整题的总时间复杂度是 $O(nk)$。

除了上面介绍的方法，我们也可以使用李超树来直接求若干个直线的 $\min$ 在某个点的值。李超树对于这个问题具有更强的通用性。这里不进行详细介绍，如果有兴趣可以自行了解。

参考代码 （C++）

```cpp
#include <bits/stdc++.h>

using i64 = long long;
using u64 = unsigned long long;
using u32 = unsigned int;

using i128 = __int128;
using u128 = unsigned __int128;

void solve() {
    int n, k;
    std::cin >> n >> k;

    std::vector<i64> x(n, 0);
    for (int i = 0; i < n; ++i) {
        std::cin >> x[i];
    }

    std::vector<i64> a(n - 1, 0);
    for (int i = 0; i < n - 1; ++i) {
        std::cin >> a[i];
    }

    std::vector<i64> b(n - 1, 0);
```

```
    for (int i = 0; i < n - 1; ++i) {
        std::cin >> b[i];
    }

    std::vector<i64> f(n, 0);
    for (int i = 1; i < n; ++i) {
        f[i] = f[i - 1] + a[i - 1];
    }
    for (int s = 1; s <= k; ++s) {
        std::deque<std::array<i64, 2>> lines;
        auto check = [&](const std::array<i64, 2> &l1, const std::array<i64, 2> &l2, const
std::array<i64, 2> &l3) -> bool {
            return (l2[1] - l1[1]) * (l2[0] - l3[0]) <= (l3[1] - l2[1]) * (l1[0] - l2[0]);
        };
        auto add = [&](const std::array<i64, 2> &l) -> void {
            while (lines.size() >= 2 && !check(lines[lines.size() - 2], lines[lines.size() - 1],
l)) {
                lines.pop_back();
            }
            lines.push_back(l);
        };
        auto query = [&](i64 t) -> i64 {
            while (lines.size() >= 2 && lines[0][0] * t + lines[0][1] > lines[1][0] * t + lines[1]
[1]) {
                lines.pop_front();
            }
            return lines[0][0] * t + lines[0][1];
        };
        std::vector<i64> nf(n, 0);
        for (int i = 1; i < n; ++i) {
            add({-2 * x[i - 1], f[i - 1] + b[i - 1] + x[i - 1] * x[i - 1]});
            nf[i] = std::min(nf[i - 1] + a[i - 1], x[i] * x[i] + query(x[i]));
        }
        f = nf;
    }

    std::cout << f[n - 1] << "\n";
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int t = 1;
    std::cin >> t;

    while (t--) {
        solve();
    }

    return 0;
}
```

最后，感谢所有出题人的努力！