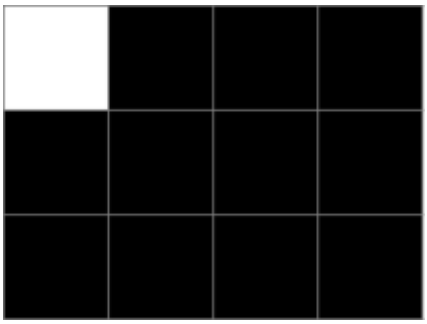
USTC 2020 秋季学期集训第一周题解

A（CF1333A）.



```cpp
#include <bits/stdc++.h>
using namespace std;

void solve() {
    int n, m; cin >> n >> m;
    string black_row(m, 'B');
    vector<string> result(n, black_row);
    result[0][0] = 'W';
    for (int i = 0; i < n; ++i) {
        cout << result[i] << '\n';
    }
}

int main() {
    int t; cin >> t;
    while(t--) solve();
}
```

图片和代码来源：https://codeforces.com/blog/entry/75802

B（CF1333C）.首先考虑求出a_i的前缀和$b_i = \sum_{j=1}^{i} a_j$，那么一个区间 合$(l,r]$法当且仅当不存在使$l \le j < i \le r$得$b_i = b_j$。考虑对于一个b_i，求出上一个与相b_i同的位置lst_i，即$lst_i = \max\{j \mid j < i, b_j = b_i\}$不存在则$lst_i = -1$。那么区间$(l,r]$合法的条件变为对于所有$0 \le i \le n$，满足$l > lst_i$或$r < i$。那么我们只要从左往右枚举$r$,那么$l$需要大于$\max\{lst_j \mid j \le r\}$，直接在枚举的同时维护$lst$数组的前缀max即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int maxn=2e5+10;
ll a[maxn];
int lst[maxn],n;
map<ll,int>p;

int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        scanf("%I64d",&a[i]);
        a[i]+=a[i-1];
    }
    ll ans=0;
    for(int i=0;i<=n;i++){
        if(!p.count(a[i]))lst[i]=-1;
        else lst[i]=p[a[i]];
        p[a[i]]=i;
    }
    int mx=-1;
    for(int i=1;i<=n;i++){
        mx=max(mx,lst[i]);
        ans+=i-mx-1;
    }
    cout<<ans<<endl;
    return 0;
}
```

C（CF1333D）.首先只要每次把所有能翻的就翻就能使得答案最优,记录这个最少翻转次数为m_i。再考虑如何使得答案最差，只要每次能翻只翻转一个,我们记录这个最大翻转次数为m_x。对于无解的情况即：$k < mi$或者$k > mx$。如果$mi < k$，那么就选择最差的方式去反转直到$mi = k$,对于剩下的选择用最优秀的方法翻转即可,这样就构造了一个合法的答案了。

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, k;

vector<int> find_steps(const vector<int>& a) {
    vector<int> steps;
    for (int i = 0; i < n - 1; ++i) {
        if (a[i] == 1 && a[i + 1] == 0) steps.push_back(i);
    }
    return steps;
}

int main() {
    cin >> n >> k;
    string s; cin >> s;
    vector<int> a(n);
    for (int i = 0; i < n; ++i) a[i] = (s[i] == 'L') ? 0 : 1;
    int maxi = 0, mini = 0;
    int cnt = 0;
    int last = -1;
    for (int i = n - 1; i >= 0; --i) {
        if (a[i] == 0) {
            ++cnt;
        } else {
            if (cnt == 0) continue;
            maxi += cnt;
            mini = max(cnt, last + 1);
            last = mini;
        }
    }
    if (k < mini || k > maxi) {
        cout << -1;
        return 0;
    }
    bool is_min = false;
    vector<int> have_step;
    for (int i = 0; i < k; ++i) {
        if (!is_min) {
            auto steps = find_steps(a);
            cout << min(int(steps.size()), maxi - k + i + 1) << ' ';
            int cur = 0;
            while (k - i - 1 < maxi && cur < steps.size()) {
                cout << steps[cur] + 1 << ' ';
                a[steps[cur]] = 0;
                a[steps[cur] + 1] = 1;
                ++cur;
                --maxi;
            }
            if (maxi == k - i - 1) {
                is_min = true;
```

```
                have_step = find_steps(a);
            }
        } else {
            int v = have_step.back();
            have_step.pop_back();
            cout << "1 " << v + 1;
            a[v] = 0;
            a[v + 1] = 1;
            if (v > 0 && a[v - 1] == 1) {
                have_step.push_back(v - 1);
            }
            if (v + 2 < n && a[v + 2] == 0) {
                have_step.push_back(v + 1);
            }
        }
        cout << '\n';
    }
}
```

代码来源：https://codeforces.com/blog/entry/75802

D（CF1295E）.

考虑最后的状态一定满足左部都不大于一个整数k,右部都大于k.

枚举初始的划分点i(将原序列分成$[1,i],[i+1,n]$两部分，$1 \leq i < n$,因为初始时两边均非空).不妨设$f(i,k)$表示以i为初始划分点，最终左部都不大于k,右部都大于k的花费.

考虑j对$f(i,j)$的贡献:

- 若$j \leq i$:若$p_j \leq k$,即j本来在左部，最终也在左部，不需移动。若$p_j > k$,即j本来在左部，最终应在右部，需要花费a_i的代价移动。
- 若$j > i$:若$p_j \leq k$,即j本来在右部，最终在左部，需要花费a_i的代价移动。若$p_j > k$,即j本来在右部，最终也在右部，不需移动。

暴力枚举i,k,j,时间复杂度$\Theta(n^3)$
考虑从划分点$i-1$到i的变化:

- 对于$k \geq p_i$,不需要再花费a_i的代价移动。
- 对于$k < p_i$,需要花费a_i的代价移动。

因此只枚举i,k即可。时间复杂度$\Theta(n^2)$

考虑上面变化的第一条，即对$k \in [p_i,n], f(i,k) = f(i-1,k) - a_i$,对$k \in [0,p_i-1], f(i,k) = f(i-1,k) + a_i$

那这就是一个区间修改，查询全局min的线段树。时间复杂度$\Theta(nlogn)$

```cpp
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;
typedef long long LL;

const int mxN=200344;

int n, p[mxN], id[mxN], a[mxN];

struct N
{
    LL sum, mx;
} t[mxN*4];

void modi(int u, int l, int r, int i, int val)
{
    if(l == r)
    {
        t[u].sum = val;
        t[u].mx = max(0, val);
        return;
    }
    int mid = (l+r)/2;
    if(i<=mid)
        modi(u*2, l, mid, i, val);
    else
        modi(u*2+1, mid+1, r, i, val);
    t[u].sum = t[u*2].sum+t[u*2+1].sum;
    t[u].mx = max(t[u*2].mx, t[u*2].sum+t[u*2+1].mx);
```

```
}

int main()
{
    scanf("%d", &n);
    for(int i=1; i<=n; ++i)
        scanf("%d", p+i), id[p[i]] = i;
    for(int i=1; i<=n; ++i)
        scanf("%d", a+i);
    for(int i=1; i<=n; ++i)
        modi(1, 1, n, p[i], -a[i]);
    LL s=0, ans=1ll<<60;
    for(int i=1; i<n; ++i)
    {
        s += a[i];
        modi(1, 1, n, p[i], a[i]);
        ans = min(ans, s-t[1].mx);
    }
    printf("%lld\n", ans);
    return 0;
}
```

题解来源：https://www.luogu.com.cn/problem/solution/CF1333D

代码部分由曾相如提供

E（CF1326E).

首先从样例也能看出来：**答案是单调不递增的**，这个很重要。也很好证：若当前答案被炸掉后，它就永远成为不了答案，它要么被新放的炸弹炸，要么被曾经炸它的炸弹炸，不可能起死回生。

那么答案什么时候会减。我们考虑当前答案x，如果我们希望x留着，就得保证**存在一个**i，使得i后面“$\geq x$的数”比i后面的“炸弹数”要多。这样说明i后面的炸弹炸不完“$\geq x$的数”，x当然就可以留下了。（这里要注意，如果你想如果后面的炸弹先把x炸了留下更大的数怎么办？这个就不需要考虑了，因为答案是单调的，既然你现在正在考虑答案是否为x，那么说明比x大的那些数，之前肯定是被炸过的，翻不了身。）

显然如果不存在这么一个i满足条件，说明**从任何位置开始**，x都避免不了被炸的悲剧，那么答案就要减了，这里得循环判断，因为答案可能减少不止一次。

所以我们要求是否**所有的位置**：“$\geq x$的数”小于等于“炸弹数”，若是，答案x减1。用线段树每一个位置表示当前位置后面“$\geq x$的数”减“炸弹数”，维护全局最大值，若最大值小于等于0，满足条件，答案减。每改变一个x，x位置前面值加一，每安放一个炸弹，炸弹位置前面值减一。

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define INF 1e9
const int maxn=1.2e6+10;
int ma[maxn],a[maxn],id[maxn],b[maxn],addv[maxn];
int n;

inline int lson(int o){return o<<1;}
inline int rson(int o){return o<<1|1;}

inline void pushup(int o){
    ma[o]=max(ma[lson(o)],ma[rson(o)]);
}

inline void pushdown(int o){
    if(addv[o]!=0){
        addv[lson(o)]+=addv[o];
        addv[rson(o)]+=addv[o];
        ma[lson(o)]+=addv[o];
        ma[rson(o)]+=addv[o];
    }
    addv[o]=0;
}

inline void change(int o,int l,int r,int ql,int qr,int v){
    if(ql<=l&&r<=qr){
        addv[o]+=v;
        ma[o]+=v;
        return;
    }
    pushdown(o);
    int mid=(l+r)>>1;
    if(ql<=mid)change(lson(o),l,mid,ql,qr,v);
    if(mid<qr)change(rson(o),mid+1,r,ql,qr,v);
    pushup(o);
}
```

```
38
39 int main(){
40     scanf("%d",&n);
41     for(int i=1;i<=n;i++)scanf("%d",&a[i]),id[a[i]]=i;
42     for(int i=1;i<=n;i++)scanf("%d",&b[i]);
43     change(1,1,n,1,id[n],1);
44     printf("%d ",n);
45     int ans=n;
46     for(int i=1;i<n;i++){
47         change(1,1,n,1,b[i],-1);
48         while(ma[1]<=0){
49             ans--;
50             change(1,1,n,1,id[ans],1);
51         }
52         printf("%d ",ans);
53     }
54     return 0;
55 }
```

题解来源：https://www.luogu.com.cn/problem/solution/CF1326E

F(CF1276D).

首先，不同的序列数量等同于**消除标记的方案数**。

考虑树形 dp。我们称一个点被一条边**覆盖**，表示这个点在考虑到这条边时选择消除了这个点上的标记。

设 $f_{x,0/1/2/3}$ 分别表示：点 x 是被自己的父亲边之前的边覆盖的、点 x 是被自己的父亲边覆盖的、点 x 是被自己的父亲边之后的边覆盖的、点 x 没有被覆盖。

那么对于点 $y \in \text{son}_x$，考虑 $f_{x,0/2}$：

- 点 y 不能被覆盖，即 $f_{y,2/3}$。
- 定义点 $z(\in \text{son}_x) < y$ 是指边 (x,z) 在边 (x,y) 之前，则点 z 一定要被覆盖，即 $f_{z,0/1}$。
- 定义点 $z(\in \text{son}_x) > y$ 是指边 (x,z) 在边 (x,y) 之后，则点 z 可以被覆盖也可以不被覆盖，但不能被边 (x,z) 覆盖，即 $f_{z,0/2/3}$。

因此有转移：

$$f_{x,0/2} = \sum_{y \in \text{son}_x} \left(f_{y,2/3} * \prod_{z<y} f_{z,0/1} * \prod_{z>y} f_{z,0/2/3} \right)$$

考虑 $f_{x,1}$，有转移：

$$f_{x,1} = \prod_{y<fa_x} f_{y,0/1} * \prod_{y>fa_x} f_{y,0/2/3}$$

考虑 $f_{x,3}$，有转移：

$$f_{x,3} = \prod_{y} f_{y,0/1}$$

时间复杂度 $\mathcal{O}(n)$。

```cpp
#include <bits/stdc++.h>
#define ui unsigned int
#define ll long long
#define ul unsigned ll
#define ld long double
#define pi pair< int, int >
#define fi first
#define se second
#define mp make_pair
#define ls (p << 1)
#define rs (ls | 1)
#define md ((t[p].l + t[p].r) >> 1)
#define pq priority_queue
#define pb push_back
#define vi vector< int >
#define si set< int >::iterator
#define fl(x) freopen(x".in", "r", stdin), freopen(x".out", "w", stdout);
using namespace std;

namespace io {
    const int SI = 1 << 21 | 1;
    char IB[SI], *IS, *IT, OB[SI], *OS = OB, *OT = OS + SI - 1, c,
ch[100];
    int f, t;
    #define gc() (IS == IT ? (IT = (IS = IB) + fread(IB, 1, SI, stdin), IS
== IT ? EOF : *IS++) : *IS++)
```

```
    inline void flush() {
        fwrite(OB, 1, OS - OB, stdout), OS = OB;
    }
    inline void pc(char x) {
        *OS++ = x;
        if (OS == OT) flush();
    }
    template<class I>
    inline void rd(I &x) {
        for (f = 1, c = gc(); c < '0' || c > '9'; c = gc()) if (c == '-') f = -1;
        for (x = 0; c >= '0' && c <= '9'; x = (x << 3) + (x << 1) + (c & 15), c = gc());
        x *= f;
    }
    inline void rds(char *s, int &x) {
        for (c = gc(); c < 33 || c > 126; c = gc());
        for (x = 0; c >= 33 && c <= 126; s[++x] = c, c = gc());
        s[x+1] = '\0';
    }
    template<class I>
    inline void print(I x, char k = '\n') {
        if (!x) pc('0');
        if (x < 0) pc('-'), x = -x;
        while (x) ch[++t] = x % 10 + '0', x /= 10;
        while (t) pc(ch[t--]);
        pc(k);
    }
    inline void prints(string s) {
        int x = s.length();
        while (t < x) pc(s[t++]);
        pc('\n'), t = 0;
    }
    struct Flush {
        ~Flush() {
            flush();
        }
    } flusher;
}
using io::rd;
using io::rds;
using io::print;
using io::prints;

const int P = 998244353;

struct modint {
    int x;
    inline modint(int x = 0) : x(x) {}
    inline modint &operator = (int o) { return x = o, *this; }
    inline modint &operator += (modint o) { return x = x + o.x >= P ? x + o.x - P : x + o.x, *this; }
    inline modint &operator -= (modint o) { return x = x - o.x < 0 ? x - o.x + P : x - o.x, *this; }
    inline modint &operator *= (modint o) { return x = 1ll * x * o.x % P, *this; }
    inline modint &operator ^= (ll b) {
        modint a = *this, c;
```

```
        if (!~b) b = P - 2;
        c.x = 1 % P;
        while (b) {
            if (b & 1) c *= a;
            a *= a, b >>= 1;
        }
        return x = c.x, *this;
    }
    inline modint &operator /= (modint o) { return *this *= o ^= -1; }
    inline modint &operator += (int o) { return x = x + o >= P ? x + o - P : x + o, *this; }
    inline modint &operator -= (int o) { return x = x - o < 0 ? x - o + P : x - o, *this; }
    inline modint &operator *= (int o) { return x = 1ll * x * o % P, *this; }
    inline modint &operator /= (int o) { modint y = modint(o); return *this *= y ^= -1; }
    template<class I>
    inline friend modint operator + (modint a, I b) { return a += b; }
    template<class I>
    inline friend modint operator - (modint a, I b) { return a -= b; }
    template<class I>
    inline friend modint operator * (modint a, I b) { return a *= b; }
    inline friend modint operator ^ (modint a, ll b) { return a ^= b; }
    template<class I>
    inline friend modint operator / (modint a, I b) { return a /= b; }
    inline friend bool operator == (modint a, int b) { return a.x == b; }
    inline friend bool operator != (modint a, int b) { return a.x != b; }
    inline bool operator ! () { return !x; }
    inline modint operator - () { return x ? P - x : 0; }
};
inline void rd(modint &x) { rd(x.x); }
inline void print(modint x, char k = '\n') { print(x.x, k); }

const int N = 2e5 + 7;
int n, m, k;
vi e[N];
modint f[N][4], a[N], b[N], c[N];

void dfs(int x, int fa) {
    for (int y : e[x]) if (y != fa) dfs(y, x);
    a[m=k=0] = 1;
    for (int y : e[x])
        if (y == fa) k = m;
        else  ++m, a[m] = a[m-1] * (f[y][0] + f[y][1]), b[m] = f[y][2] + f[y][3], c[m] = f[y][0] + b[m];
    c[m+1] = 1;
    for (int i = m; i; i--) c[i] *= c[i+1];
    for (int i = 1; i <= m; i++) f[x][(i>k)<<1] += a[i-1] * b[i] * c[i+1];
    f[x][1] = a[k] * c[k+1], f[x][3] = a[m];
}

int main() {
    rd(n);
    for (int i = 1, x, y; i < n; i++) rd(x), rd(y), e[x].pb(y), e[y].pb(x);
    dfs(1, 0), print(f[1][0] + f[1][2] + f[1][3]);
    return 0;
```

```
130 }
```

题解和代码来源：https://www.luogu.com.cn/problem/solution/CF1276D

edit by jyz