The Editorial of Goodbye 2022 Round

Nanako & m_99 &
pornjoke & SteamTurbine & triple_a & Nezzar

由于 spoiler 标签仅在 Codeforces 上被支持

中文题解的 Hint 将不是折叠式的

敬请谅解

Problem A. Koxia and Whiteboards

Hint

- **Hint 1**

  最终答案将是 $a_1, \ldots, a_n, b_1, \ldots, b_m$ 中的 n 个值之和。

- **Hint 2**

  做完所有操作后，b_m 总是在最终的 n 个数当中。

- **Hint 3**

  考虑 $n = 2$ 且 $m = 2$ 时的情形。根据 Hint 2，b_2 总是在最终的 n 个数当中，b_1 呢?

Solution

这个问题可以自然地用贪心算法解决——对于 $i = 1, 2, \ldots, m$，我们使用 b_i 替换当前 $a_1, a_2, \ldots, a_n$ 中最小的元素即可。每组测试数据所需的时间复杂度是 $O(nm)$。

另一种做法是，我们先将 b_m 加到最终的和中，而对于其它 $(n + m - 1)$ 个数，我们自由选择其中的 $(n - 1)$ 个并最大化它们的和。因此，sort 或者类似的做法可以在 $O((n + m)\log(n + m))$ 的时间复杂度内解决该问题，这也是出题人预想中的正解。

Code (m_99)

```
#include <stdio.h>
#include <bits/stdc++.h>
using namespace std;
#define rep(i,n) for (int i = 0; i < (n); ++i)
#define Inf32 1000000001
#define Inf64 4000000000000000001

int main(){

    int _t;
    cin>>_t;

    rep(_,_t){
        int n,m;
        cin>>n>>m;
        vector<long long> a(n+m);
        rep(i,n+m)scanf("%lld",&a[i]);

        sort(a.begin(),a.end()-1);
        reverse(a.begin(),a.end());

        long long ans = 0;
        rep(i,n)ans += a[i];
```

```
        cout<<ans<<endl;
    }


    return 0;
}
```

Problem B. Koxia and Permutation

Hint

- **Hint 1**

  $k = 1$ 时，所有排列的代价都是 $2n$。

- **Hint 2**

  $k \geq 2$ 时， 最小代价总是 $n + 1$。

Solution

$k = 1$ 时，每个排列都会有相同的代价。

$k \geq 2$ 时，最小代价将至少是 $n + 1$。这是因为至少会有一个区间包含 n，并对 $\max$ 部分贡献 n 代价，而此时 $\min$ 的部分将至少贡献 1 代价。

最小代价确实总是可以达到这个下界，我们可以构造形如 $[n, 1, n - 1, 2, n - 2, 3, n - 3, 4, \ldots]$ 的排列使得代价为 $n + 1$，无论 k 的取值如何。

每组测试数据所需的时间复杂度是 $O(n)$。经过小心的实现，其他构造方式也可能得出符合条件的答案。

Code (Nanako)

```cpp
#include <iostream>
#define MULTI int _T; cin >> _T; while(_T--)
using namespace std;
typedef long long ll;

int n, k;

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

    MULTI {
        cin >> n >> k;
        int l = 1, r = n, _ = 1;
        while (l <= r) cout << ((_ ^= 1) ? l++ : r--) << ' ';
        cout << endl;
    }
}
```

Problem C. Koxia and Number Theory

Hint

- **Hint 1**

  如果 a_i 不是两两不同的，那么这是 NO 的一种 trivial 情形。

- **Hint 2**

  如果 a_i 是两两不同的， 你能构造一个 $n = 4$ 的样例使得答案是 NO 吗?

- **Hint 3**

  也许你在思考奇偶性之类的性质？尝试一般化你的想法。

- **Hint 3.5**

  考虑每一个质数。考虑中国剩余定理。

- **Hint 4**

  我们应该检查多少质数呢？考虑鸽巢原理。

Solution

首先，a_i 应当两两不同。原因在于，$a_i + x \geq 2$ 且 $\gcd(t, t) = t$，所以这是 NO 的一种 trivial 情形。

对于给定的 x，让我们记 $b_i := a_i + x$。条件「$\gcd(b_i, b_j) = 1$ 对于 $1 \leq i < j \leq n$ 恒成立」等价于「每个质数 p 只能整除至多一个 b_i」。那么，对于一个质数 p，我们能否判断，p 是否总是整除至少两个 b_i（无论 x 的取值）？

一个很小但很有启发性的样例是 $a = [5, 6, 7, 8]$。这个样例的答案是 NO，因为：

- $x \equiv 0 \pmod 2$ 时，有 $\gcd(6 + x, 8 + x) \neq 1$；
- $x \equiv 1 \pmod 2$ 时，有 $\gcd(5 + x, 7 + x) \neq 1$。

也就是说，我们在模 2 意义下考虑数组 $[5, 6, 7, 8]$，则得到 $[1, 0, 1, 0]$。其中 0 和 1 均出现两次，所以 x 的任何取值都导致 b 中有两个元素被 2 整除。

这个想法也可以推广到更大的质数。对于一个质数 p，令 cnt_j 是 j 在 $[a_1 \bmod p, a_2 \bmod p, \ldots, a_n \bmod p]$ 中的出现次数。如果 $\min(cnt_0, cnt_1, \ldots, cnt_{p-1}) \geq 2$，则我们立刻输出 NO。

虽然 $[1, 10^{18}]$ 内有非常多的质数，但我们只需要检查小于等于 $\lfloor \frac{n}{2} \rfloor$ 的质数。这是因为根据鸽巢原理，$\min(cnt_0, cnt_1, \ldots, cnt_{p-1}) \geq 2$ 对于更大的质数是不可能被满足的。由于小于等于 $\lfloor \frac{n}{2} \rfloor$ 的质数至多有 $O(\frac{n}{\log n})$ 个，我们在 $O(\frac{n^2}{\log n})$ 的时间复杂度内解决了本问题。

而 $\min(cnt) \geq 2$ 是本质的的原因是，对于一个质数 p，如果 $a_u \equiv a_v \pmod p$，则我们有必要令 $(x + a_u) \not\equiv 0 \pmod p$，否则 $\gcd(x + a_u, x + a_v)$ 将会是 p 的倍数。所以实际上，$cnt_i \geq 2$ 意味着 $x \not\equiv (p - i) \pmod p$。如果 $\min(cnt) < 2$ 对于所有质数 p 都成立，则我们可以列出一个同余方程组，并使用中国剩余定理求解合适的 x；如果至少存在一

个质数使得 $\min(cnt) \geq 2$，则 x 的任何取值都将导致 p 出现至少两次。

Code (Nanako)

```cpp
#include <iostream>
#include <algorithm>
#define MULTI int _T; cin >> _T; while(_T--)
using namespace std;
typedef long long ll;

const int N = 105;
const int INF = 0x3f3f3f3f;
template <typename T> bool chkmin (T &x, T y) {return y < x ? x = y, 1 :
0;}
template <typename T> bool chkmax (T &x, T y) {return y > x ? x = y, 1 :
0;}

int n;
ll a[N];

int cnt[N];

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

    MULTI {
        cin >> n;
        for (int i = 1;i <= n;++i) {
            cin >> a[i];
        }

        int isDistinct = 1;
        sort(a + 1, a + n + 1);
        for (int i = 1;i <= n - 1;++i) {
            if (a[i] == a[i + 1]) isDistinct = 0;
        }
        if (isDistinct == 0) {
            cout << "NO" << endl;
            continue;
        }

        int CRT_able = 1;
        for (int mod = 2;mod <= n / 2;++mod) {
            fill(cnt, cnt + mod, 0);
            for (int i = 1;i <= n;++i) {
                cnt[a[i] % mod]++;
            }
            if (*min_element(cnt, cnt + mod) >= 2) CRT_able = 0;
```

```
            }
            cout << (CRT_able ? "YES" : "NO") << endl;
        }
    }
```

Problem D. Koxia and Game

Hint

- **Hint 1**

  如果 a, b, c 均已确定，如何判定获胜的一方是谁?

- **Hint 2**

  如果只有 a, b 已确定，尝试设计一个算法判定是否存在 c 使得 Koxia 获胜。

- **Hint 2.5**

  如果你无法解决 Hint 2 中的问题，尝试思考它如何与图论相关。

- **Hint 3**

  尝试讨论图中每个连通分量的结构以计算 c 的数量。

Solution

首先，让我们考虑一个恰当的数组 c 如何使得 Koxia 获胜。

引理 **1** 在每一轮游戏中，Koxia 应当移除 S 中恰当的元素使得 S 中剩余的两个元素是相同的，也即 Mahiru 的选择实际上并不能影响结果。

- 在第 n 轮中，如果 Koxia 留下两个不同的元素，则 Mahiru 总是可以阻止 p 成为排列。
- 这意味着只有当 d_n 对于 Mahiru 只有一种选择时，Koxia 才可能获胜。
- d_n 确定后，$(d_1, d_2, \ldots, d_{n-1})$ 将会是某 $(n-1)$ 个数的排列。对于 d_{n-1} 进行相似的讨论并以此类推，我们得出结论——只有在每一轮游戏中 d_i 对于 Mahiru 都只有一种选择时，Koxia 才可能获胜。

引理 **2** 令 p 是一个长度为 n 的数组，其中 p_i 是 a_i 和 b_i 中的一者。Koxia 获胜当且仅当 p 可能是一个排列。

- 根据引理 1，当 p 可能是一个排列时，我们令 $c_i = p_i$，则 Koxia 可以迫使 Mahiru 在每一轮中都选择 $d_i = p_i$，所以 Koxia 获胜。
- 如果 p 不可能是一个排列，我们可以使用类似于引理 1 的归约证明不存在任何数组 c 使得 Koxia 获胜。

首先，我们需要一个算法判定 p 是否可能是一个排列。

将输入数据 (a_i, b_i) 视为边，则我们将这一问题转化为大小为 n 的图上的一个图论问题。p 可能是一个排列，当且仅当存在一种方式给每条边指定方向使得每个结点都被恰好一条边所指向。不难看出，这等价于，对每个连通分量而言，边的数量等于顶点的数量。这可以通过并查集或者图的遍历在 $O(n\alpha(n))$ 或 $O(n)$ 的时间复杂度内判定。

为了解决计数的问题，让我们考虑每一个连通分量的结构。每个满足 $|V| = |E|$ 的连通分量都可以被视为 一棵树加上一条附加边，而这条边可以分为两类。

- 附加边和其他树边一起形成环。在环上，我们有两种边选点的方案（顺时针和逆时针），而在此之后其他边选点的方案将是固定的（边指向远离环的方向）。

- 附加边形成自环。此时 c_i 的取值并不影响图的结构，所以任何 $[1,n]$ 内的取值都是合法的，而其他边选点的方案将是固定的。

因此，在答案非零时，本问题的最终答案是 $2^{cnt_{\text{cycle components}}} \cdot n^{cnt_{\text{self-loop components}}}$，其中 $cnt_{\text{cycle components}}$ 代表成非自环的连通分量数量，$cnt_{\text{self-loop components}}$ 代表成自环的连通分量数量。时间复杂度是 $O(n\alpha(n))$ 或 $O(n)$。

Code (Nanako, DSU)

```cpp
#include <iostream>
#include <numeric>
#define MULTI int _T; cin >> _T; while(_T--)
using namespace std;
typedef long long ll;

const int N = 1e5 + 5;
const int mod = 998244353;

int n;
int a[N], b[N];

int fa[N], cnt_v[N], cnt_e[N], selfloop[N];
int vis[N];
void init () {
    iota(fa + 1, fa + n + 1, 1);
    fill(cnt_v + 1, cnt_v + n + 1, 1);
    fill(cnt_e + 1, cnt_e + n + 1, 0);
    fill(selfloop + 1, selfloop + n + 1, 0);
    fill(vis + 1, vis + n + 1, 0);
}
int getfa (int x) {
    return fa[x] == x ? x : fa[x] = getfa(fa[x]);
}
void merge (int u, int v) {
    u = getfa(u);
    v = getfa(v);
    cnt_v[u] += cnt_v[v];
    cnt_e[u] += cnt_e[v];
    selfloop[u] |= selfloop[v];
    fa[v] = u;
}

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

    MULTI {
        cin >> n;
        for (int i = 1;i <= n;++i) {
```

```
            cin >> a[i];
        }
        for (int i = 1;i <= n;++i) {
            cin >> b[i];
        }

        init();
        for (int i = 1;i <= n;++i) {
            if (getfa(a[i]) != getfa(b[i])) merge(a[i], b[i]);
            cnt_e[getfa(a[i])]++;
            if (a[i] == b[i]) selfloop[getfa(a[i])] = 1;
        }

        ll ans = 1;
        for (int i = 1;i <= n;++i) if (vis[getfa(i)] == 0) {
            if (cnt_v[getfa(i)] != cnt_e[getfa(i)]) ans = 0;
            else ans = ans * (selfloop[getfa(i)] ? n : 2) % mod;
            vis[getfa(i)] = 1;
        }
        cout << ans << endl;
    }
}
```

Code (zengminghao, DFS)

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
const int P = 998244353;

int n, a[N], b[N];
vector <int> G[N];
bool vis[N];

int vertex, edge, self_loop;
void dfs(int x) {
    if (vis[x]) return ;
    vis[x] = true;
    vertex++;
    for (auto y : G[x]) {
        edge++;
        dfs(y);
        if (y == x) {
            self_loop++;
        }
    }
}

void solve() {
```

```
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    for (int i = 1; i <= n; i++) scanf("%d", &b[i]);

    for (int i = 1; i <= n; i++) G[i].clear();

    for (int i = 1; i <= n; i++) {
        G[a[i]].push_back(b[i]);
        G[b[i]].push_back(a[i]);
    }

    int ans = 1;

    for (int i = 1; i <= n; i++) vis[i] = false;
    for (int i = 1; i <= n; i++) {
        if (vis[i]) continue ;
        vertex = 0;
        edge = 0;
        self_loop = 0;
        dfs(i);
        if (edge != 2 * vertex) {
            ans = 0;
        } else if (self_loop) {
            ans = 1ll * ans * n % P;
        } else {
            ans = ans * 2 % P;
        }
    }

    printf("%d\n", ans);
}

int main() {
    int t;
    scanf("%d", &t);
    while (t--) {
        solve();
    }
    return 0;
}
```

Problem E. Koxia and Tree

Hint

- **Hint 1**

  尝试解决一个经典问题——求树上 k 个指定结点的两两距离之和。

- **Hint 2**

  添加移动操作，但边的方向固定。求树上 k 个指定结点的两两距离之和。

- **Hint 2.5**

  如果你无法解决 Hint 2 中的问题，尝试思考为什么出题人令每条边仅被经过一次。

- **Hint 3**

  当边的方向随机时，维护 p_i（结点 i 存在蝴蝶的概率）如何帮助你计算答案?

Solution

第一眼看来，我们容易联想到一个经典问题——求树上 k 个指定结点的两两距离之和。对于任意一条边，如果其两侧分别有 x 个指定结点和 $n-x$ 个指定结点，则最终将有 $x(n-x)$ 对结点对经过这条边。不失一般性，我们以结点 1 为根，并定义 siz_i 为子树 i 中指定结点的数量。通过对于每条边 (fa, son) 计算 $siz_{son}(n-siz_{son})$ 并求和，我们得到了这一问题的答案，而其在除以 $\binom{k}{2}$ 后也将等于两个结点（从 k 个结点中均匀随机选择）距离的期望值。

接下来让我们考虑 Hint 2——考虑移动操作，但边的方向固定。让我们定义 siz_i^0 和 siz_i 为子树 i 中蝴蝶的数量，但分别表示初始值和实时值。一个非常重要的观察是，虽然蝴蝶在移动，但因为每条边最多被蝴蝶经过一次，所以我们总是可以说 $|siz_{son} - siz_{son}^0| \le 1$。如果我们正确维护了蝴蝶的位置，这一性质允许我们在常数时间复杂度内讨论 siz_{son} 的实际取值并加算得出答案。

当我们进一步引入随机方向时，如果我们定义 p_i 为「结点 i 存在蝴蝶的概率」，则「从结点 u 移动到结点 v」将等价于令 $p_u = p_v = \frac{p_u+p_v}{2}$，这允许我们轻松地维护 p 的实时值。类似地，通过讨论 siz_{son} 的取值（但结合每种情况的概率计算，而非执行具体的移动操作），我们得到最终答案。总时间复杂度是 $O(n)$。

Code (Nanako)

```cpp
#include <iostream>
#include <vector>
using namespace std;
typedef long long ll;

const int N = 3e5 + 5;
const int mod = 998244353;
const int inv2 = 499122177;

ll qpow (ll n, ll m) {
```

```
    ll ret = 1;
    while (m) {
        if (m & 1) ret = ret * n % mod;
        n = n * n % mod;
        m >>= 1;
    }
    return ret;
}
ll getinv (ll a) {
    return qpow(a, mod - 2);
}

int n, k;
int a[N];
int u[N], v[N];

vector <int> e[N];
int fa[N];
ll p[N], sum[N];
void dfs (int u, int f) {
    sum[u] = p[u];
    for (int v : e[u]) if (v != f) {
        dfs(v, u);
        fa[v] = u;
        sum[u] += sum[v];
    }
}

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

    cin >> n >> k;
    for (int i = 1;i <= k;++i) {
        cin >> a[i];
        p[a[i]] = 1;
    }
    for (int i = 1;i <= n - 1;++i) {
        cin >> u[i] >> v[i];
        e[u[i]].push_back(v[i]);
        e[v[i]].push_back(u[i]);
    }
    dfs(1, -1);

    ll ans = 0;
    for (int i = 1;i <= n - 1;++i) {
        if (fa[u[i]] == v[i]) swap(u[i], v[i]);
        ll puv = p[u[i]] * (1 - p[v[i]] + mod) % mod;
        ll pvu = p[v[i]] * (1 - p[u[i]] + mod) % mod;
```

```
        ll delta = 0;
        delta -= puv * sum[v[i]] % mod * (k - sum[v[i]]) % mod;
        delta -= pvu * sum[v[i]] % mod * (k - sum[v[i]]) % mod;
        delta += puv * (sum[v[i]] + 1) % mod * (k - sum[v[i]] - 1) %
mod;
        delta += pvu * (sum[v[i]] - 1) % mod * (k - sum[v[i]] + 1) %
mod;
        ans = (ans + sum[v[i]] * (k - sum[v[i]]) + delta * inv2) % mod;
        ans = (ans % mod + mod) % mod;
        p[u[i]] = p[v[i]] = 1ll * (p[u[i]] + p[v[i]]) * inv2 % mod;
    }
    cout << ans * getinv(1ll * k * (k - 1) / 2 % mod) % mod << endl;
}
```

Problem F. Koxia and Sequence

Hint

- **Hint 1**

  根据对称性，对于任意非负整数 t，$a_1 = t$ 时的良好序列数量、$a_2 = t$ 时的良好序列数量、……都是相同的。

- **Hint 2**

  尝试独立考虑每一位对答案的贡献。

- **Hint 3**

  由于按位异或的性质，我们可以在模 2 意义下计算答案。

- **Hint 4**

  计算按位或恰等于 y 的方案数很困难，但计算按位或为 y 的一个子集的方案数则相对简单。因此，考虑容斥。

- **Hint 5**

  考虑 Lucas 定理 / Kummer 定理。

- **Hint 6**

  「共有 $a + b$ 个球。考虑所有满足 $c + d = n$ 的非负整数对 (c, d)，计算从前 a 个球中选择 c 个并从后 b 个球中选择 d 个的方案数之和。」

  由于这等价于从 $a + b$ 个球中任意选择 n 个，这一问题的答案是 $\binom{a+b}{n}$。这一恒等式即 Vandermonde 恒等式。

Solution

令 $f(i, t)$ 表示 $a_i = t$ 时的良好序列数量，我们有 $f(1, t) = f(2, t) = \ldots = f(n, t)$。因此，如果 n 是偶数，则答案为 0。否则，答案是满足「$a_1 = t$ 时的良好序列数量是奇数」时 t 的异或和。对于每一位独立考虑，则我们将问题转化为「对于每个 i，计算 a_1 的第 i 位模 2 为 1 时的良好序列数量」。

令 $g(y')$ 表示 y 是 y' 的子集（即 $y \mid y' = y'$）时的答案，我们可以用归纳法证明原问题的答案是那些满足 y' 是 y 的子集的 $g(y')$ 的异或和。因此，新的目标是，对于每个 i，计算满足「y' 是 y 的子集且 $g(y')$ 的第 i 位模 2 为 1」的 y' 的数量。

根据 Lucas 定理的推论或 $p = 2$ 时的 Kummer 定理，我们知道 $\binom{a}{b} \bmod 2 = 1$ 等价于「b 是 a 的子集」。长度为 n 且和为 x 且或为 y 的子集的良好序列数量在模 2 意义下等于 $\sum_{t_1+\ldots+t_n=x} \prod \binom{y}{t_i}$。如果存在 t_i 不是 y 的子集，则模 2 意义下 $\binom{y}{t_i} = 0$，乘积也为 0。考虑 Vandermonde 恒等式，该值又应等于 $\binom{ny}{x}$。类似于此前，我们将问题转化为——对于每个 i，计算满足「y' 是 y 的子集且 y' 的第 i 位模 2 为 1 且 $x - 2^i$ 是 $ny' - 2^i$ 的子集（$\binom{ny'-2^i}{x-2^i} \bmod 2 = 1$）」的 y' 的数量。

综上，分别枚举 i 和 y'，得到时间复杂度为 $O(y \log y)$。

Code (errorgorn)

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

#define int long long
#define ll long long
#define ii pair<ll,ll>
#define iii pair<ii,ll>
#define fi first
#define se second
#define endl '\n'
#define debug(x) cout << #x << ": " << x << endl

#define pub push_back
#define pob pop_back
#define puf push_front
#define pof pop_front
#define lb lower_bound
#define ub upper_bound

#define rep(x,start,end) for(int x=(start)-((start)>(end));x!=(end)-((start)>(end));((start)<(end)?x++:x--))
#define all(x) (x).begin(),(x).end()
#define sz(x) (int)(x).size()

mt19937 rng(chrono::system_clock::now().time_since_epoch().count());

int n,a,b;

bool isSub(int i,int j){
    if (i<0 || j<0) return false;
    return (j&i)==i;
}

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin.exceptions(ios::badbit | ios::failbit);

    cin>>n>>a>>b;

    int ans=0;
    for (int sub=b;sub;sub=(sub-1)&b) rep(bit,0,20) if (sub&(1<<bit)){
        if (isSub(a-(1<<bit),n*sub-(1<<bit))){
            ans^=(1<<bit);
        }
    }
```

```
    cout<<ans*(n%2)<<endl;
}
```

G Koxia and Bracket

Hint

- **Hint 1**

  被删除的括号序列具有怎样的特殊性质?
- **Hint 2**

  尝试使用 $O(n^2)$ DP 解决这个问题。
- **Hint 3**

  如果没有正则的要求，能不能用一次操作快速处理多个括号?
- **Hint 4**

  你能不能将上一个想法和分治结合起来?

Solution

$O(n^2)$ Solution

让我们考虑被删除的括号子序列有什么性质。

首先，它一定是一个形如))((((的括号序列。证明很简单：如果有一个被删除的) 在 (的右侧，那么我们可以保留这一对 () 而不破坏剩余序列的正则性。

这个性质意味着我们可以将原序列分为两段，并在左边的一段中仅删除)，在右边的一段中仅删除 (。现在让我们尝试找到两部分的分界点：考虑一个基于括号序列的前缀和，其中的每个 (被替换为 1，而每个) 被替换为 -1。

我们将一个位置定义为 Special 的，当且仅当这个位置对应的数小于先前出现的最小值。不难发现，每当出现一个 Special 位置，我们都必须在这个位置之前额外删除一个) 使得括号序列重新满足条件。

从上述思路考虑，我们可以发现只有最远的 Special 位置之前的) 可能被删除，因此我们可以将这个位置作为分界点。

我们现在分别解决两侧的问题。值得指出的是它们本质相同，因为我们可以将仅删除 (的问题转化为仅删除) 的问题。例如，如果从 (()((())()) 中仅删除 (，则相当于从 (()())))() 中仅删除)。

对于仅删除) 的问题，使得序列正则的充分条件是，操作完成后，前缀和中的每个数必须大于 0。

同样从上述思路考虑，让我们设计状态 $dp_{i,j}$，代表在考虑到第 i 个) 时，我们在满足前缀和限制之余，额外删除的) 的数量为 j（$j \geq 0$） 的方案数。

$$dp_{i,j} = \begin{cases} dp_{i-1,j} + dp_{i-1,j-1}, & \text{if i is not Special;} \\ dp_{i-1,j} + dp_{i-1,j+1}, & \text{if i is Special.} \end{cases}$$

将删除）的部分与删除（的部分得到的 $dp_{end,0}$ 相乘即为答案。时间复杂度 $O(n^2)$，经过编译器优化的方案可以在大约 9 秒内完成运算，但这不足以通过本题。

$O(n\sqrt{n\log n})$ Solution

让我们考虑不存在 Special 位置的转移可以通过什么性质进一步优化。对于状态 $dp_{i,j}$，在处理 k 个）后，有转移状态如下：

$$dp_{i+k,j} = \sum_{l=0}^{k} \binom{k}{l} \times dp_{i,j-l}$$

我们发现这个转移方程表现为卷积形式。因此我们可以通过 NTT 对这个卷积进行优化，单次操作的时间复杂度为 $O(n\log n)$，而由于 Special 位置的存在，这种做法的最坏全局复杂度为 $O(n^2\log n)$。

考虑如何将这个做法与 $O(n^2)$ 做法结合起来。对于状态 $dp_{i,j}$ 与 k 个），我们考虑其对 $dp_{i,k}$ 的贡献。我们发现如果满足 $j \geq k$，那么状态转移无论如何都不会受 Special 位置的影响。

基于上述思路，我们可以采用基于定期重构的分块方式：设定重构周期 B，在一轮周期内，对于 $j \leq B$ 的部分，我们使用 $O(n^2)$ 的DP做法处理，而对于 $j > B$ 的部分，我们在一轮周期结束后用 NTT 计算出答案。

时间复杂度 $O(\frac{n^2}{B} + B \cdot n\log n)$，通过设置合适的 B，我们可以将时间复杂度优化到 $O(n\sqrt{n\log n})$。虽然时间复杂度仍然很高，但考虑到 $O(n^2)$ 的低常数，合适的实现可以通过本题。

$O(n\log^2 n)$ Solution

考虑将提取 $j \geq k$ 部分进行 NTT 的思路与分治结合起来。假设现在需要处理的区间为 (l, r)，同时传递的 dp 多项式为 s，我们进行如下操作：

- 统计区间 (l, r) 中 Special 位置的个数 num，将多项式 s 中对应状态 $j \geq num$ 的部分提取出来，单独与当前区间进行卷积。
- 将多项式 s 中对应状态 $j < num$ 的部分传入区间 (l, mid) 中进行运算，再将得到的结果传入区间 $(mid + 1, r)$ 中继续运算。
- 将上述两步操作得到的多项式直接相加，返回得到的多项式。

如何计算进行上述操作的时间复杂度？让我们分别分析传入左区间与右区间的操作：

- 传入左区间 (l, mid) 时，进行 NTT 运算的多项式的大小为区间 (l, r) 中包含 Special 位置的个数减去左区间 (l, mid) 包含 Special 位置的个数，即右区间 $(mid + 1, r)$ 中包含 Special 位置的个数，这个数不会超过右区间 $(mid + 1, r)$ 的长度。
- 传入右区间 $(mid + 1, r)$ 时，多项式的大小不会超过左区间 (l, mid) 的长度。
- 同时，与 s 相乘的组合数多项式的长度为区间长度 +1.

综上，在区间 (l,r) 中进行 NTT 运算的两个多项式的大小不会超过区间长度 + 1。因此这个做法的时间复杂度为分治 NTT 的时间复杂度，即 $O(n\log^2 n)$。

Code (errorgorn)

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/rope>
using namespace std;
using namespace __gnu_pbds;
using namespace __gnu_cxx;

#define int long long
#define ll long long
#define ii pair<ll,ll>
#define iii pair<ii,ll>
#define fi first
#define se second
#define endl '\n'
#define debug(x) cout << #x << ": " << x << endl

#define pub push_back
#define pob pop_back
#define puf push_front
#define pof pop_front
#define lb lower_bound
#define ub upper_bound

#define rep(x,start,end) for(auto x=(start)-((start)>(end));x!=(end)-
((start)>(end));((start)<(end)?x++:x--))
#define all(x) (x).begin(),(x).end()
#define sz(x) (int)(x).size()

#define indexed_set
tree<ll,null_type,less<ll>,rb_tree_tag,tree_order_statistics_node_update
>
//change less to less_equal for non distinct pbds, but erase will bug

mt19937 rng(chrono::system_clock::now().time_since_epoch().count());

const int MOD=998244353;

ll qexp(ll b,ll p,int m){
    ll res=1;
    while (p){
        if (p&1) res=(res*b)%m;
        b=(b*b)%m;
        p>>=1;
```

```
    }
    return res;
}

ll inv(ll i){
    return qexp(i,MOD-2,MOD);
}

ll fix(ll i){
    i%=MOD;
    if (i<0) i+=MOD;
    return i;
}

ll fac[1000005];
ll ifac[1000005];

ll nCk(int i,int j){
    if (i<j) return 0;
    return fac[i]*ifac[j]%MOD*ifac[i-j]%MOD;
}

//https://github.com/kth-competitive-
programming/kactl/blob/main/content/numerical/NumberTheoreticTransform.h
const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<int> vi;
typedef vector<ll> vl;
void ntt(vl &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, qexp(root, mod >> s, mod)};
        rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
    }
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
            ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
            a[i + j + k] = ai - z + (z > ai ? mod : 0);
            ai += (ai + z >= mod ? z - mod : z);
        }
}
vl conv(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
```

```
    int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s), n = 1 << B;
    int inv = qexp(n, mod - 2, mod);
    vl L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    rep(i,0,n) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
    ntt(out);
    return {out.begin(), out.begin() + s};
}

vector<int> v;

vector<int> solve(int l,int r,vector<int> poly){
    if (poly.empty()) return poly;

    if (l==r){
        poly=conv(poly,{1,1});
        poly.erase(poly.begin(),poly.begin()+v[l]);
        return poly;
    }

    int m=l+r>>1;
    int num=0;
    rep(x,l,r+1) num+=v[x];
    num=min(num,sz(poly));

    vector<int> small(poly.begin(),poly.begin()+num);
    poly.erase(poly.begin(),poly.begin()+num);

    vector<int> mul;
    rep(x,0,r-l+2) mul.pub(nCk(r-l+1,x));
    poly=conv(poly,mul);

    small=solve(m+1,r,solve(l,m,small));
    poly.resize(max(sz(poly),sz(small)));
    rep(x,0,sz(small)) poly[x]=(poly[x]+small[x])%MOD;

    return poly;
}

int solve(string s){
    if (s=="") return 1;
    v.clear();

    int mn=0,curr=0;
    for (auto it:s){
        if (it=='(') curr++;
        else{
            curr--;
```

```
            if (curr<mn){
                mn=curr;
                v.pub(1);
            }
            else{
                v.pub(0);
            }
        }
    }

    return solve(0,sz(v)-1,{1})[0];
}

int n;
string s;
int pref[500005];

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin.exceptions(ios::badbit | ios::failbit);

    fac[0]=1;
    rep(x,1,1000005) fac[x]=fac[x-1]*x%MOD;
    ifac[1000004]=inv(fac[1000004]);
    rep(x,1000005,1) ifac[x-1]=ifac[x]*x%MOD;

    cin>>s;
    n=sz(s);
    pref[0]=0;
    rep(x,0,n) pref[x+1]=pref[x]+(s[x]=='('?1:-1);

    int pos=min_element(pref,pref+n+1)-pref;
    string a=s.substr(0,pos),b=s.substr(pos,n-pos);
    reverse(all(b)); for (auto &it:b) it^=1;
    cout<<solve(a)*solve(b)%MOD<<endl;
}
```

Problem H. Koxia, Mahiru and Winter Festival

Hint

- **Hint 1**

  在怎样的情形下，最小阻塞度为 1？

- **Hint 2**

  假定你有一个黑盒可以给出网格大小为 $n-2$ 时的方案，请尝试给出网格大小为 n 时的方案。

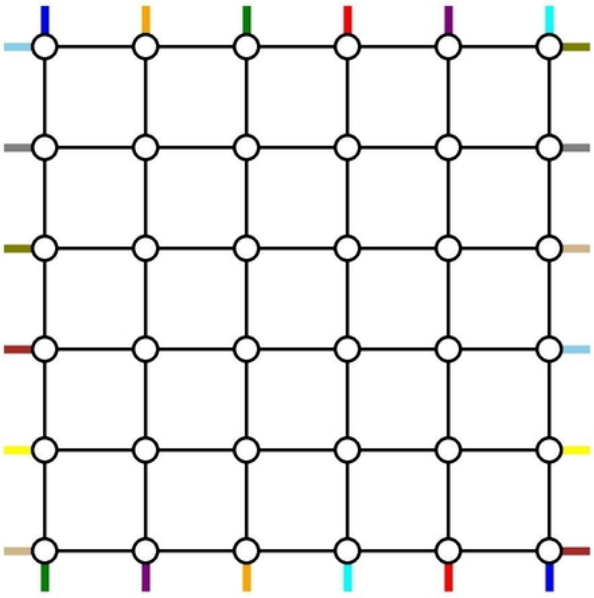
Preface

这是 *congestion minimization* 的特殊情况。这个问题的一般化版本是 NP-Hard 的，但是在本问题的特殊结构上可以解决。

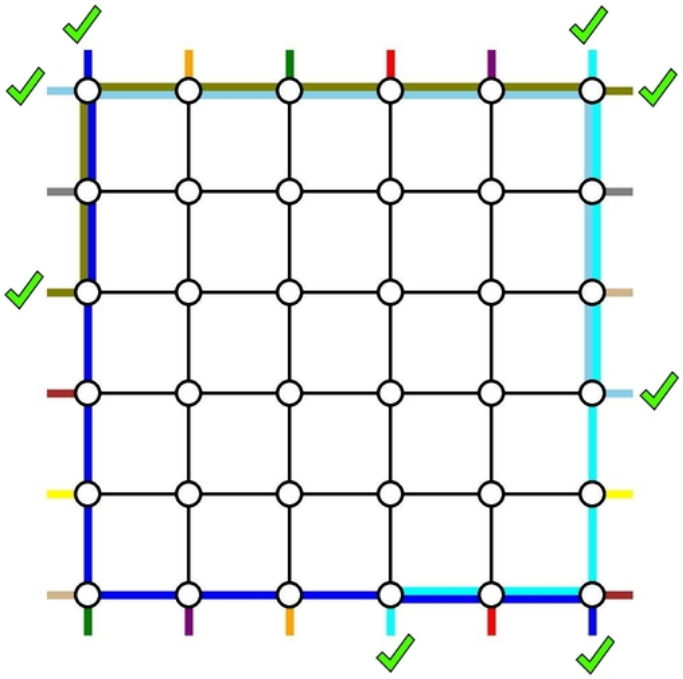
唯一使得最大阻塞度为 1 的情形是 $p=q=[1,2,\ldots,n]$。这可以使用鸽巢原理证明——如果存在 $p_i \neq i$ 或 $q_i \neq i$，则所有 $2n$ 条路线的最小长度之和将超过边的总数，即总存在边被经过多于一次。

我们现在的目标是尝试构造路线方案使得最大阻塞度为 2。我们将证明这对于任意输入数据都是可能的。让我们先展示一些图片作为草案，以表达我们的大致想法，并在稍后完善其细节。

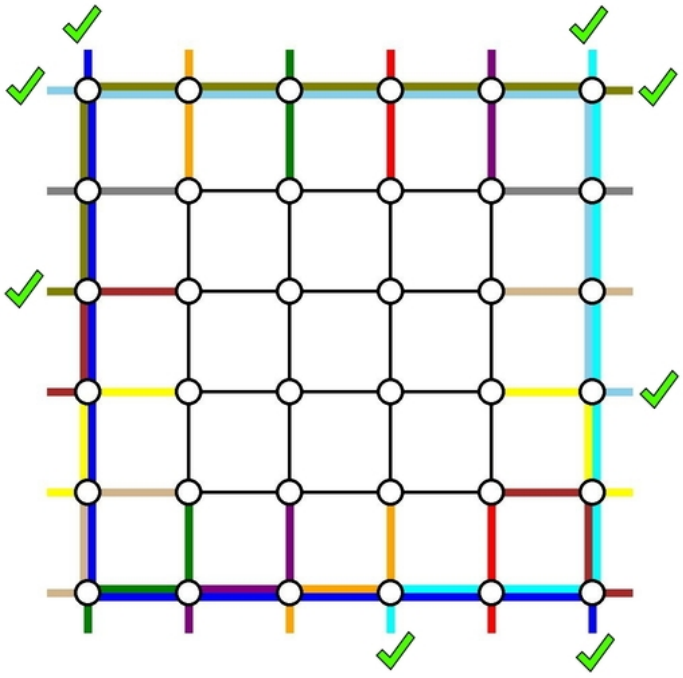
Solution (sketch)



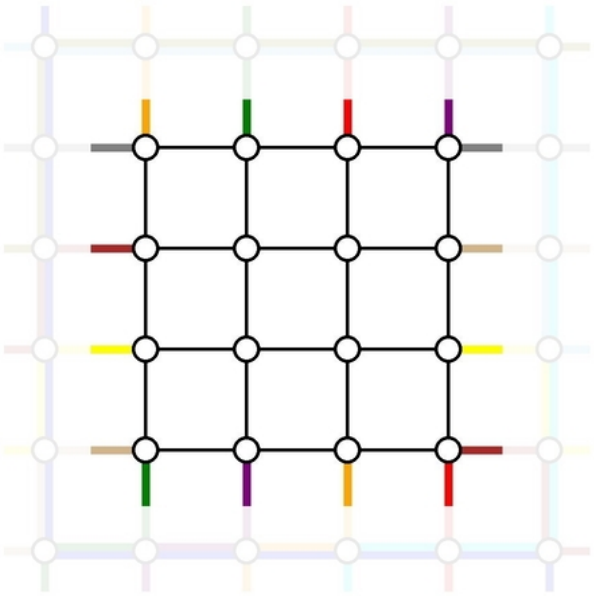
Given a problem instance of size 6



First we route 4 paths using only the outer edges



For the remaining starting/ending points, connect it to the inner grid one step closer



We have reduced the problem to an instance of size 4

Solution (details)

这一做法基于归纳法。基本情形 $n = 0$ 和 $n = 1$ 是 Trivial 的。我们假设我们解决了所有网格大小为 $k - 2$ 时的情形，现在我们将其视为一个黑盒，用于解决网格大小为 k 时的情形。

对于网格大小为 k 时的情形，首先，我们仅用最外侧的边连接以下 4 条路线：

- 使用左侧和下侧的边，连接从 $(1, 1)$ 出发的、从上至下的路线；
- 使用右侧和下侧的边，连接从 $(1, k)$ 出发的、从上至下的路线；
- 使用上侧和右侧的边，连接从 $(1, 1)$ 出发的、从左至右的路线；
- 使用左侧和上侧的边，连接到达 $(1, k)$ 的、从左至右的路线。如果该路线连接的点对和上一条路线相同，我们使用左侧、上侧和右侧的边连接任意从左至右的路线。

到目前为止，还有 $k - 2$ 条从上至下的路线和 $k - 2$ 条从左至右的路线需要连接。我们只需将它们的起点和终点向中心靠近一格，保持它们的相对顺序。这样一来，我们就把原问题归约为一个大小为 $k - 2$ 的问题，而我们已经知道其如何解决。

Code (SteamTurbine)

```
#include <bits/stdc++.h>
#define FOR(i,s,e) for (int i=(s); i<(e); i++)
#define FOE(i,s,e) for (int i=(s); i<=(e); i++)
#define FOD(i,s,e) for (int i=(s)-1; i>=(e); i--)
#define PB push_back
using namespace std;

struct Paths{
    /* store paths in order */
```

```
    vector<vector<pair<int, int>>> NS, EW;

    Paths(){
        NS.clear();
        EW.clear();
    }
};

Paths solve(vector<int> p, vector<int> q){
    int n = p.size();
    Paths Ret;
    Ret.NS.resize(n);
    Ret.EW.resize(n);

    // Base case
    if (n == 0) return Ret;
    if (n == 1){
        Ret.NS[0].PB({1, 1});
        Ret.EW[0].PB({1, 1});
        return Ret;
    }

    // Route NS flow originating from (1, 1) and (1, n) using leftmost
and rightmost edges
    FOE(i,1,n){
        Ret.NS[0].PB({i, 1});
        Ret.NS[n-1].PB({i, n});
    }
    // Routing to final destination using bottom edges
    FOE(i,2,p[0]) Ret.NS[0].PB({n, i});
    FOD(i,n,p[n-1]) Ret.NS[n-1].PB({n, i});

    // Create p'[] for n-2 instance
    vector<int> p_new(0);
    FOE(i,1,n-2) p_new.PB(p[i] - (p[i]>p[0]) - (p[i]>p[n-1]));

    // Route EW flow originating from (1, 1) using topmost and rightmost
edges
    FOE(i,1,n) Ret.EW[0].PB({1, i});
    FOE(i,2,q[0]) Ret.EW[0].PB({i, n});

    // Route EW flow originating in (m, 1) with q[m] as small as
possible
    int m = 1;
    // special handle so congestion is 1 if possible
    if (p[0] == 1 && p[n-1] == n && q[0] == 1 && q[n-1] == n){
        m = n - 1;
        FOE(i,1,n) Ret.EW[n-1].PB({n, i});
    }
```

```
    else{
        FOR(i,1,n) if (q[i] < q[m]) m = i;
        // Route(m+1, 1) --> (1, 1) --> (1, n) --> (q[m], n)

        FOD(i,m+2,2) Ret.EW[m].PB({i, 1});
        FOR(i,1,n) Ret.EW[m].PB({1, i});
        FOE(i,1,q[m]) Ret.EW[m].PB({i, n});
    }

    // Create q'[] for n-2 instance
    vector<int> q_new(0);
    FOR(i,1,n) if (i != m) q_new.PB(q[i] - (q[i]>q[0]) - (q[i]>q[m]));

    if (n > 1){
        Paths S = solve(p_new, q_new);
        int t;

        // connect NS paths
        FOR(i,1,n-1){
            Ret.NS[i].PB({1, i+1});
            for (auto [x, y]: S.NS[i-1]){
                Ret.NS[i].PB({x+1, y+1});
                t = y + 1;
            }
            Ret.NS[i].PB({n, t});
            if (p[i] != t) Ret.NS[i].PB({n, p[i]});
        }

        // connect EW paths
        int l = 0;
        FOR(i,1,n) if (i != m){
            Ret.EW[i].PB({i+1, 1});
            if (i > m) Ret.EW[i].PB({i, 1});

            for (auto [x, y]: S.EW[l]){
                Ret.EW[i].PB({x+1, y+1});
                t = x + 1;
            }

            Ret.EW[i].PB({t, n});
            if (q[i] != t) Ret.EW[i].PB({q[i], n});
            ++l;
        }
    }

    return Ret;
}

int main(){
```

```
    int n;
    vector<int> p, q;

    scanf("%d", &n);
    p.resize(n), q.resize(n);
    FOR(i,0,n) scanf("%d", &p[i]);
    FOR(i,0,n) scanf("%d", &q[i]);

    Paths Solution = solve(p, q);

    for (auto path: Solution.NS){
        printf("%d", path.size());
        for (auto [x, y]: path) printf(" %d %d", x, y);
        puts("");
    }

    for (auto path: Solution.EW){
        printf("%d", path.size());
        for (auto [x, y]: path) printf(" %d %d", x, y);
        puts("");
    }
    return 0;
}
```