2023 年牛客多校第三场题解

出题人：北京航空航天大学

1 A World Fragments I

题意：给定两个二进制数 x, y，每次可以选择 x 二进制表达中的其中一位 b，然后执行 $x \leftarrow x - b$ 或 $x \leftarrow x + b$。问 x 最少经过多少次操作变成 y。$1 \le x, y \le 10^9$。

解法：只要数字不为 0，就一直存在 $b = 1$，因而首先特判 $x = y$ 的情况，再判断 x 是否为 0，若不为 0 则输出 $|x - y|$。

2 B Auspiciousness

题意：给定 n 表示有一个由 $\{1, 2, 3, \cdots, 2n\}$ 构成的 $2n$ 张牌的初始牌堆，同时自己这里有一个空的牌堆。执行以下的操作：

1. 翻开牌堆顶的一张牌，并放在自己牌堆的堆顶。
2. 记自己牌堆堆顶的一张牌大小为 x。如果初始牌堆为空，结束。否则执行 3 操作。
3. 若 $x \le n$，则猜测初始牌堆翻出的下一张牌 y 比 x 大，否则猜测翻出来的下一张牌比 x 小。这时翻开初始牌堆的堆顶，并将这一张牌放在自己的牌堆的堆顶。如果猜测正确，则跳转到 2 操作，否则结束游戏。

问对于 $(2n)!$ 种牌的排列，总的抽取牌数有多少，对特定数字 m 取模。多测，$\sum n \le 300$，$1 \le m \le 10^9$。

解法：首先不考虑空间复杂度，定义 $f_{i,x,y,k}$ 表示当前在抽取第 $2n - x - y + 1$ 张牌时，小于等于 n 的牌张数还剩 x 张，大于 n 的牌数还剩 y 张，在抽第 $2n - x - y$ 张（上一轮）牌时（1）如果是小于等于 n 的牌，则选了这小于等于 n 的剩下来的 x 张牌里面第 k 小的牌；（2）如果是大于 n 的牌，则选了这大于 n 的剩下来的 y 张牌里面第 k 小的牌；（该状态用 $i \in \{0, 1\}$ 区分）作为的现有方案总数（**不考察后续初始牌堆的牌顺序**）。

这样构造方案的思路：首先比较容易想到的是，维护大于 n 和小于等于 n 的个数，然后维护一个状态表示上一次是抽到大于的还是小于等于的。但是这样在枚举当前轮的时候，就无法避免重复选择的问题。因而修正到**去掉已经选过的**之后排多少，这样操作就不会导致重复问题，并且也方便刻画当前的大小。

既然不考虑重复问题，那么可以写出下面的转移方程：

$$
\begin{cases}
f_{0,x,y,k} \leftarrow \sum_{i=1}^{k} f_{0,x+1,y,i} + \sum_{i=0}^{y+1} f_{1,x,y+1,i} \\
f_{1,x,y,k} \leftarrow \sum_{i=k+1}^{x+1} f_{1,x+1,y,i} + \sum_{i=0}^{y+1} f_{0,x,y+1,i}
\end{cases}
$$

对于答案统计，可以考虑对每一步成功抽牌进行分步统计——即不再统计有多少种抽牌方式能抽到 i 张牌，而是在能抽到第 i 张牌的状态中，每次叠加它的次数（贡献）。

不难发现状态数有 $O(n^3)$ 个，对于单个状态的计算如果按照上式直接计算，会达到 $O(n^4)$ 的复杂度。不难注意到第四个维度可以前缀和，因而可以使用前缀和优化掉第四维，得到下面一个未经过空间优化的代码（会 MLE）。

注意下面的代码中，为了统一 dp 方程形式，在 $[1, n]$ 部分是维护的第 k 大，在 $[n+1, 2n]$ 部分维护的是第 k 小。

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

```cpp
#define int long long
int dp[2][310][310][310];
int solve()
{
    int n, mod;
    cin >> n >> mod;
    // A: 阶乘 C: 组合数
    vector<int> fac(n * 2 + 3);
    vector<vector<int>> C(n * 2 + 3, vector<int>(n * 2 + 3));
    C[0][0] = fac[0] = 1;
    for (int i = 1; i <= n * 2; i++)
    {
        fac[i] = fac[i - 1] * i % mod;
        C[i][0] = 1;
        for (int j = 1; j <= i; j++)
            C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;
    }
    // 清空
    for (int i = 0; i <= n + 3; i++)
        for (int j = 0; j <= n + 3; j++)
            for (int k = 0; k <= n + 3; k++)
                dp[0][i][j][k] = dp[1][i][j][k] = 0;
    // 快速取模
    function<void(int &, int)> add = [&](int &x, int y)
    {
        if ((x += y) >= mod)
            x -= mod;
    };
    function<void(int &, int)> del = [&](int &x, int y)
    {
        if ((x -= y) < 0)
            x += mod;
    };
    // 任何情况，都能拿走一张牌
    int ans = fac[2 * n];
    // 初始条件：在还没开始抽牌之前，大（[n+1,2n]）还剩n张，小（[1,n]）也剩n张。
    for (int i = 1; i <= n; i++)
    {
        dp[0][n][n][i] = i % mod; // 一开始是小，当前选择的牌是[1,i]范围，有i种
        dp[1][n][n][i] = i % mod; // 一开始是大，同理
    }
    // 枚举大小两类牌在本轮抽取完成后，各剩多少（x,y），倒序枚举
    for (int x = n; x >= 0; x--)
    {
        for (int y = n; y >= 0; y--)
        {
            if (x + y >= 2 * n) // 初值设定过
```

```cpp
                continue;
            // 本次抽的是[1,n]中的牌
            for (int k = 1; k <= x; k++)
            {
                if (x != n) // 上一轮抽到一张范围在[1,n]的牌，如果要继续游戏，如果当前抽到k，则上一轮抽出来的牌必须在[1,k]的范围（比当前小）
                    add(dp[0][x][y][k], (dp[0][x + 1][y][x + 1] - dp[0][x + 1][y][k] + mod) % mod);
                if (y != n) // 再枚举当前抽到的是一张大的，这时一定可以接着抽
                    add(dp[0][x][y][k], dp[1][x][y + 1][y + 1]);
                // 能走到当前这一步的状态，都统计一下这一次抽牌对答案的贡献
                add(ans, dp[0][x][y][k] * fac[x + y - 1] % mod);
            }
            // 前缀和一下
            for (int k = 1; k <= x; k++)
                add(dp[0][x][y][k], dp[0][x][y][k - 1]);
            // 本次抽到的是[n+1,2n]中的牌
            for (int k = 1; k <= y; k++)
            {
                if (y != n) // 上一轮抽到一张范围在[n+1,2n]的牌，如果要继续游戏，如果当前抽到k，则上一轮抽出来的牌必须在[k+1,y+1]的范围（比当前大）
                    add(dp[1][x][y][k], (dp[1][x][y + 1][y + 1] - dp[1][x][y + 1][k] + mod) % mod);
                if (x != n) // 上一轮抽到一张[1,n]的牌，都可以继续抽牌
                    add(dp[1][x][y][k], dp[0][x + 1][y][x + 1]);
                // 能走到当前这一步的状态，都统计一下这一次抽牌对答案的贡献
                add(ans, dp[1][x][y][k] * fac[x + y - 1] % mod);
            }
            // 前缀和一下
            for (int k = 1; k <= y; k++)
                add(dp[1][x][y][k], dp[1][x][y][k - 1]);
        }
    }
    // 全部能抽完的部分多算了一步
    add(ans, (fac[2 * n] - dp[0][1][0][1] - dp[1][0][1][1] + mod) % mod);
    return ans;
}
signed main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int T;
    cin >> T;
    while (T--)
        cout << solve() << "\n";
    return 0;
}
```

由于每次 x 只会从 $x+1$ 转移，因而可以考虑再滚动掉最外层的一维数组（即 x），即使用 $0,1$ 两个状态来表示当前的 x 和 $x+1$。

```cpp
#include <bits/stdc++.h>
using namespace std;
long long dp[2][2][310][310];
void Solve()
{
    int n, mod;
    scanf("%d%d", &n, &mod);
    vector<long long> fac(n * 2 + 3);
    vector<vector<long long>> C(n * 2 + 3, vector<long long>(n * 2 + 3));
    C[0][0] = fac[0] = 1;
    for (int i = 1; i <= n * 2; i++)
    {
        fac[i] = fac[i - 1] * i % mod;
        C[i][0] = 1;
        for (int j = 1; j <= i; j++)
            C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;
    }
    for (int i = 0; i <= 1; i++)
        for (int j = 0; j <= n + 3; j++)
            for (int k = 0; k <= n + 3; k++)
                dp[0][i][j][k] = dp[1][i][j][k] = 0;
    // 快速取模
    function<void(long long &, long long)> add = [&](long long &x, long long y)
    {
        x += y;
        if (x >= mod)
            x -= mod;
    };
    function<void(long long &, long long)> del = [&](long long &x, long long y)
    {
        x -= y;
        if (x < 0)
            x += mod;
    };

    long long ans = fac[2 * n];
    // 在下面的代码中，为方便编写，因而考虑用(n-x)的奇偶性来作为滚动数组的标识
    // 初始条件
    for (int i = 1; i <= n; i++)
    {
        dp[0][0][n][i] = i % mod;
        dp[1][0][n][i] = i % mod;
    }
    int op = 0; // 滚动后x的代表
```

```cpp
    // 枚举两个各剩多少
    for (int x = n; x >= 0; x--)
    {
        for (int y = n; y >= 0; y--)
        {
            if (x + y == 2 * n || x + y == 0)
                continue;
            // 第一类转移
            for (int k = 1; k <= x; k++)
            {
                if (x != n)
                    add(dp[0][op][y][k], (dp[0][op ^ 1][y][x + 1] - dp[0][op ^ 1][y][k] + mod) % mod);
                if (y != n)
                    add(dp[0][op][y][k], dp[1][op][y + 1][y + 1]);
                add(ans, dp[0][op][y][k] * fac[x + y - 1] % mod);
            }
            for (int k = 1; k <= x; k++)
                add(dp[0][op][y][k], dp[0][op][y][k - 1]);
            // 第二类转移
            for (int k = 1; k <= y; k++)
            {
                if (y != n)
                    add(dp[1][op][y][k], (dp[1][op][y + 1][y + 1] - dp[1][op][y + 1][k] + mod) % mod);
                if (x != n)
                    add(dp[1][op][y][k], dp[0][op ^ 1][y][x + 1]);
                add(ans, dp[1][op][y][k] * fac[x + y - 1] % mod);
            }
            for (long long k = 1; k <= y; k++)
                add(dp[1][op][y][k], dp[1][op][y][k - 1]);
        }
        op ^= 1;
        for (long long i = 0; i <= n + 3; i++)
            for (long long j = 0; j <= n + 3; j++)
                dp[0][op][i][j] = dp[1][op][i][j] = 0;
    }
    // 去掉抽走2n张牌的重复贡献
    add(ans, (fac[2 * n] - dp[1][op ^ 1][1][1] * 2 + mod * 2) % mod);
    printf("%lld\n", ans);
}
int main()
{
    int T;
    scanf("%d", &T);
    while (T--)
        Solve();
```

```
    return 0;
}
```

3 D Ama no Jaku

题意：给定一 $n \times n$ 的 01 矩阵，每次可以翻转一行或一列，执行若干次操作。若将操作完成后的矩阵的每一行从做到右视为一个二进制数 $\{r\}_{i=1}^{n}$，每一列从上到下视为 $\{c\}_{i=1}^{n}$，要求 $\min(r_i) \geq \max(c_i)$，问是否可以实现，可以实现求最小操作次数。$1 \leq n \leq 300$。

解法：假设第一行有一个 1，则 $\max(c_i) \geq 2^{n-1}$，此时 $\min(r_i) \geq 2^{n-1}$，即第一列每个数字都是 1，则此时 $\max(c_i) = 2^n - 1$，则要求整个矩阵都是 1。因而全 1 矩阵是合理的。

如果第一行全 0，则 $\min(r_i) = 0$，则 $\max(c_i) = 0$，整个矩阵全 0。

因而整个矩阵必须所有数字都相同。

考虑维护方程 $\{c_i\}$ 和 $\{r_j\}$。如果最后是全 0，如果当前 $a_{i,j} = 1$，则 $c_i \neq r_j$，反之相等。因而维护一个 01 种类并查集即可。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 2000;
char a[N + 5][N + 5];
// [1, n] row0; [n+1,2n] row1
// [2*n+1,3*n] col0
int father[4 * N + 5], n;
int getfather(int x)
{
    return father[x] == x ? x : father[x] = getfather(father[x]);
}
int getid(int id, int flag, int col)
{
    int ans = id;
    if (flag)
        ans += 2 * n;
    if (col)
        ans += n;
    return ans;
}
void merge(int x, int y)
{
    x = getfather(x);
    y = getfather(y);
    if (x != y)
        father[x] = y;
}
bool check(int x, int y)
```

```
{
    return getfather(x) == getfather(y);
}
int solve(int op)
{
    int m = n * 2;
    for (int i = 1; i <= 2 * m; i++)
        father[i] = i;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            if (a[i][j] == ('1' ^ op))
            {
                merge(getid(i, 0, 0), getid(j, 1, 1));
                merge(getid(i, 0, 1), getid(j, 1, 0));
            }
            else
            {
                merge(getid(i, 0, 0), getid(j, 1, 0));
                merge(getid(i, 0, 1), getid(j, 1, 1));
            }
    for (int i = 1; i <= n; i++)
        if (check(i, i + n))
            return -1;
    for (int i = 1; i <= n; i++)
        if (check(i + 2 * n, i + 3 * n))
            return -1;
    map<int, int> siz;
    for (int i = 1; i <= n; i++)
        siz[getfather(i)]++;
    for (int i = 2 * n + 1; i <= 3 * n; i++)
        siz[getfather(i)]++;
    int ans = 2 * n;
    for (auto [x, y] : siz)
        ans = min(ans, y);
    return min(ans, 2 * n - ans);
}
int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%s", a[i] + 1);
    if (solve(0) == -1)
    {
        printf("-1");
        return 0;
    }
    printf("%d", min(solve(0), solve(1)));
```

```
    return 0;
}
```

4 E Koraidon, Miraidon and DFS Shortest Path

题意：给定一张 $G(n,m)$ 的有向图，使用 dfs 算法求解从 1 开始的单源最短路，问给定的图能否在任何边遍历顺序下都正确输出。$1 \le n, m \le 10^5$。

解法：为什么我们要找支配树？可以考虑以下三个例子：

基本错误型：一个点（2）有多种到达方式。

一个环：多次遍历到的点不一定是有重复的。

破环：多种遍历方式会导致破环方式不同（如 $6 \to 4$，$4 \to 2$，$5 \to 3$），从而影响答案。

首先我们依然可以建立一个 bfs 树，找到从 1 出发到每个点的最短距离。然后考虑原图中的每一条边：

1. 如果当前一条边 (u,v) 连接的两点是同层的——必然出现了图 1 基本型，因而输出 `No`。
2. 如果当前一条边 (u,v) 连接的两点是 u 浅 v 深的——该边会在 bfs 树中用到，且一定满足 $\text{dep}_u + 1 = \text{dep}_v$。不关心它对答案的影响。
3. 如果当前一条边 (u,v) 连接了不同层的两点（u 深 v 浅）——可能出现图 2（正确）或者图 3（错误）的情况。这时要分析从 1 出发是不是必须经过 v 到 u。如果必须经过，则是图 2 情况。否则，一定可以经过一条不经过 v 的道路到达 u（支配关系定义），这时更新 v 的答案会出错（$\text{dep}_u + 1 > \text{dep}_u > \text{dep}_v$）。因而这个在支配树上维护支配关系即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 5e5 + 10;
namespace dtree
{
    const int MAXN = 500020;
    vector<int> E[MAXN], RE[MAXN], rdom[MAXN];

    int S[MAXN], RS[MAXN], cs;
    int par[MAXN], val[MAXN], sdom[MAXN], rp[MAXN], dom[MAXN];

    void clear(int n)
    {
        cs = 0;
        for (int i = 0; i <= n; i++)
        {
            par[i] = val[i] = sdom[i] = rp[i] = dom[i] = S[i] = RS[i] = 0;
            E[i].clear();
            RE[i].clear();
            rdom[i].clear();
        }
```

```
    }
    void add_edge(int x, int y) { E[x].push_back(y); }
    void Union(int x, int y) { par[x] = y; }
    int Find(int x, int c = 0)
    {
        if (par[x] == x)
            return c ? -1 : x;
        int p = Find(par[x], 1);
        if (p == -1)
            return c ? par[x] : val[x];
        if (sdom[val[x]] > sdom[val[par[x]]])
            val[x] = val[par[x]];
        par[x] = p;
        return c ? p : val[x];
    }
    void dfs(int x)
    {
        RS[S[x] = ++cs] = x;
        par[cs] = sdom[cs] = val[cs] = cs;
        for (int e : E[x])
        {
            if (S[e] == 0)
                dfs(e), rp[S[e]] = S[x];
            RE[S[e]].push_back(S[x]);
        }
    }
    int solve(int s, int *up)
    {
        dfs(s);
        for (int i = cs; i; i--)
        {
            for (int e : RE[i])
                sdom[i] = min(sdom[i], sdom[Find(e)]);
            if (i > 1)
                rdom[sdom[i]].push_back(i);
            for (int e : rdom[i])
            {
                int p = Find(e);
                if (sdom[p] == i)
                    dom[e] = i;
                else
                    dom[e] = p;
            }
            if (i > 1)
                Union(i, rp[i]);
        }
        for (int i = 2; i <= cs; i++)
```

```
            if (sdom[i] != dom[i])
                dom[i] = dom[dom[i]];
        for (int i = 2; i <= cs; i++)
            up[RS[i]] = RS[dom[i]];
        return cs;
    }
}
int up[MAXN];
vector<int> G[MAXN];
int dep[MAXN], f[21][MAXN];
void dfs(int x, int fa)
{
    dep[x] = dep[fa] + 1;
    for (int i = 0; i <= 19; i++)
        f[i + 1][x] = f[i][f[i][x]];
    for (auto it : G[x])
    {
        if (it == fa)
            continue;
        f[0][it] = x;
        dfs(it, x);
    }
}
int lca(int x, int y)
{
    if (dep[x] < dep[y])
        swap(x, y);
    for (int i = 20; i >= 0; i--)
    {
        if (dep[f[i][x]] >= dep[y])
            x = f[i][x];
        if (x == y)
            return x;
    }
    for (int i = 20; i >= 0; i--)
        if (f[i][x] != f[i][y])
            x = f[i][x], y = f[i][y];
    return f[0][x];
}
string solve()
{
    int n, m;
    cin >> n >> m;
    dtree::clear(n);
    for (int i = 1; i <= n; i++)
        G[i].clear();
    for (int i = 1; i <= m; i++)
```

```cpp
    {
        int x, y;
        cin >> x >> y;
        dtree::E[x].push_back(y);
    }
    dtree::solve(1, up);
    for (int i = 2; i <= n; i++)
        G[up[i]].push_back(i);
    dfs(1, 0);
    const int inf = 1e9;
    vector<int> dis(n + 1, inf);
    dis[1] = 0;
    queue<int> q;
    q.push(1);
    bool flag = true;
    while (!q.empty())
    {
        int x = q.front();
        q.pop();
        for (auto it : dtree::E[x])
            if (dis[it] == inf)
            {
                q.push(it);
                dis[it] = dis[x] + 1;
                continue;
            }
            else if (dis[it] != dis[x] + 1)
            {
                if (lca(it, x) != it)
                    flag = false;
            }
    }
    if (flag)
        return "Yes";
    else
        return "No";
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int T;
    cin >> T;
    while (T--)
        cout << solve() << "\n";
    return 0;
}
```

5 F World Fragments II

题意：给定两个十进制数 x, y，每次可以选择 x 十进制表达中的其中一位 b，然后执行 $x \leftarrow x - b$ 或 $x \leftarrow x + b$。问 x 最少经过多少次操作变成 y。多次询问，$1 \le q \le 3 \times 10^5$，$1 \le x, y \le 3 \times 10^5$，强制在线。

解法：显然，每个点可以向外连出若干条边模拟一次操作。如果数字范围足够小那么是一个简单的全源最短路问题，但是本题数据范围较大，但是我们仍然需要这一建图的思想。

考虑固定枚举出中间某个特定的变化步骤，即取出一段长度为 9 的一段——因为一步操作至多使得 x 变化 9。然后采用分治的思想：

定义起始数字 x 和结束数字 y 以及中间变化过程都在区间 $[l, r]$，考虑中间的 9 个数字 $[m, m+8]$。这时可以考虑以 $[m, m+8]$ 依次每个点跑单源最短路（bfs），求得每个点到 $[m, m+8]$ 中每个点到其他点的正向（即 $x \to [m, m+8]$）和反向（即 $[m, m+8] \to x$）的距离。这时这个图的大小和源都相对较少，可以承受。

如果起始数字和结束数字分列在 $[l, m)$ 和 $(m+8, r]$ 区间，则必然需要在变化过程中经过 $[m, m+8]$。那么答案可以枚举 $x \to [m, m+8]$ 中某个点的正向距离，加上 $y \to [m, m+8]$ 的反向距离之和。

如果都分在同一侧，则递归在两个子区间进行分治处理，预处理记录每个点在 $O(\log n)$ 层中到枢纽点 $[m, m+8]$ 的两个距离。

考虑一次查询操作，则需要对每一个包含 $[l, r]$ 的区间都做一次答案更新操作：$x \to [m, m+8]$ 的正向距离，加上 $y \to [m, m+8]$ 的反向距离之和。这样单次查询操作仅需要查询 $O(\log n)$ 个区间，复杂度就是 $O(k \log n)$。整体复杂度 $O(k(n+q)\log n)$。

6 G Beautiful Matrix

题意：给定一个 $n \times m$ 的字符矩阵 $\{S\}_{(i,j)=(1,1)}^{(n,m)}$，定义一个 $n \times n$ 的子矩阵是优美的当且仅当 $\forall i, j \in [1, n]$，$S_{i,j} = S_{n-i+1,n-j+1}$。问 $\{S\}$ 中有多少个优美的子矩阵。$1 \le n, m \le 2 \times 10^3$。

解法：本题卡常，请使用 $\mathcal{O}(n^2)$ 的做法通过。

首先枚举哪一行是中央对称行。当固定中央行之后，考虑在这一行做 Manacher。传统的 Manacher 是仅单字符匹配成立即可扩展回文半径，在本题这种矩阵对称中，可以考虑对一个矩阵区域（正向绿色等于反向紫色）的匹配作为扩展条件：

因而使用二维哈希判断子矩阵对称相等，结合 Manacher 算法即可通过。复杂度同每行 Manacher 算法，即 $O(nm)$。

本题严重卡常。

7 H Until the Blue Moon Rises

题意：给定一个长度为 n 的序列 $\{a\}_{i=1}^{n}$，一次操作可以选择两个不同的数字 a_i 和 a_j 执行 $a_i \leftarrow a_i + 1, a_j \leftarrow a_j - 1$。问能不能有限次操作内让序列中每个数字都是质数。$1 \le n \le 10^3$，$1 \le a_i \le 10^9$。

解法：当 $n = 1$ 时，显然和 s 为质数才行。

当 $n \ge 2$ 时，$s \ge 2n$（每个数字都是最小的质数 2）。

当 $n = 2$ 时，如果 s 为偶数，则由哥德巴赫猜想一定成立。如果 s 为奇数，则必然是 $2 + (n-2)$。检查 $n-2$ 是否为质数即可。

当 $n \ge 3$ 时，可以首先先安排 $n-2$ 个 2，问题退化到 $n = 2$ 的情形。当 $s - 2n$ 为偶数时，哥德巴赫猜想使得一定有解。如果 $s - 2n$ 是奇数，则可以让前面 $n-2$ 个 2 中其中一个 2 变成 3，则此时 $s - 2n - 1$ 为偶数，如果 $s - 2n - 1 \ge 4$，则同样利用哥德巴赫猜想可以证明成立。

8 I To the Colors of the Dreams of Electric Sheep

题意：有一个 n 个点的树，第 i 个点拥有的颜色种类由二进制数 c_i 定义。q 次询问，每次从 u 出发到 v，初始自选颜色，一秒的时间里可以移动到相邻同色节点（即该节点有自身的一种颜色），或者在一个点变换自身颜色（必须是这个点已经有的颜色）。问花费的最少时间。$1 \le n, q \le 5 \times 10^5$，$0 \le c_i < 2^{60}$。

解法：一个贪心的想法是，能尽可能维持当前颜色不变就尽可能不变。因而首先处理出从当前节点 u 向上最多能不变色走到哪里，然后再倍增处理出变 2^k 次颜色会跳到哪里（因为可能不变色一次只能走一条边）。

考虑 $u \to v$ 可以拆分成 $u \to \mathrm{lca}(u,v) \to v$，因而拆分成祖孙链上的问题。对于单程 $u \to \mathrm{lca}(u,v)$，首先倍增跳到最靠近 $\mathrm{lca}(u,v)$ 的变色点 w，然后再维护不变色段 $w \to \mathrm{lca}(u,v)$ 的可行颜色，对 v 做同样的考虑。两侧取交集非空则不需要在转折点变色。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 5e5 + 10;
vector<int> ve[maxn];
int dep[maxn], f[21][maxn];
int jmp[21][maxn];
int col[60][maxn]; // 对于每个颜色能跳的最高的节点
long long a[maxn];
int Fa[maxn];
void dfs(int x, int fa)
{
    dep[x] = dep[fa] + 1;
    Fa[x] = fa;
    int up = -1;
    for (int i = 0; i < 60; i++)
    {
        col[i][x] = col[i][fa];
        if ((a[x] >> i & 1) && col[i][x] == -1)
            col[i][x] = x;
        if (~a[x] >> i & 1)
            col[i][x] = -1;
        if (up == -1 || (col[i][x] != -1 && dep[up] > dep[col[i][x]]))
            up = col[i][x];
    }
    jmp[0][x] = up;
    for (int i = 0; i <= 19; i++)
```

```
    {
        f[i + 1][x] = f[i][f[i][x]];
        if (jmp[i][x] != -1)
            jmp[i + 1][x] = jmp[i][jmp[i][x]];
    }
    for (auto it : ve[x])
    {
        if (it == fa)
            continue;
        f[0][it] = x;
        dfs(it, x);
    }
}
int lca(int x, int y)
{
    if (dep[x] < dep[y])
        swap(x, y);
    for (int i = 20; i >= 0; i--)
    {
        if (dep[f[i][x]] >= dep[y])
            x = f[i][x];
        if (x == y)
            return x;
    }
    for (int i = 20; i >= 0; i--)
        if (f[i][x] != f[i][y])
            x = f[i][x], y = f[i][y];
    return f[0][x];
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n, q;
    cin >> n >> q;
    memset(col, -1, sizeof(col));
    memset(jmp, -1, sizeof(jmp));
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 1; i < n; i++)
    {
        int x, y;
        cin >> x >> y;
        ve[x].push_back(y);
        ve[y].push_back(x);
    }
    dfs(1, 0);
```

```
    function<int(int, int)> solve = [&](int x, int y)
    {
        int L = lca(x, y);
        int ans = dep[x] + dep[y] - dep[L] * 2;
        if (x == L)
            swap(x, y);
        if (x == L)
            return 0;
        if (y == L)
        {
            for (int i = 20; i >= 0; i--)
                if (jmp[i][x] != -1 && dep[jmp[i][x]] > dep[L])
                {
                    ans += 1 << i;
                    x = jmp[i][x];
                }
            if (a[x] & a[Fa[x]])
                return ans;
            return -1;
        }
        for (int i = 20; i >= 0; i--)
            if (jmp[i][x] != -1 && dep[jmp[i][x]] > dep[L])
            {
                ans += 1 << i;
                x = jmp[i][x];
            }
        for (int i = 20; i >= 0; i--)
            if (jmp[i][y] != -1 && dep[jmp[i][y]] > dep[L])
            {
                ans += 1 << i;
                y = jmp[i][y];
            }
        if (jmp[0][x] == -1 || jmp[0][y] == -1)
            return -1;
        if (dep[jmp[0][x]] > dep[L] || dep[jmp[0][y]] > dep[L])
            return -1;
        bool flag = false;
        for (int i = 0; i < 60; i++)
            if (col[i][x] != -1 && col[i][y] != -1)
            {
                if (dep[col[i][x]] <= dep[L] && dep[col[i][y]] <= dep[L])
                    flag = true;
            }
        if (flag)
            return ans;
        else
            return ans + 1;
```

```
    };
    while (q--)
    {
        int x, y;
        cin >> x >> y;
        cout << solve(x, y) << "\n";
    }
    return 0;
}
```

9 J Fine Logic

题意：给定 n 个点和 m 对偏序关系 $\langle u,v\rangle$，构造最少的排列数目 k，使得在这 k 个排列中至少有一个排列满足 $u<v$ 。$1\le n,m\le 10^6$。

解法：数据范围过于庞大意味着 k 必然不大。其实很容易发现一种任意情况下都成立的方案：$k=2$，一个排列是 $\{1,2,3,\cdots,n\}$，另一个排列是 $\{n,n-1,n-2,\cdots,1\}$。因为 $u<v$ 的情况在第一个排列中，$u>v$ 在第二个排列中。

考虑什么时候满足 $k=1$。如果 $k=1$，则必然存在一个拓扑序列。考虑依照 $u\to v$ 建图，即必须访问完 u 才能访问 v。如果该 DAG 有拓扑序，那么这个排列就是合法的。否则则按 $k=2$ 的方案构造。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1000000;
int in[N + 5];
vector<int> G[N + 5];
int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 1, u, v; i <= m; i++)
    {
        scanf("%d%d", &u, &v);
        G[u].push_back(v);
        in[v]++;
    }
    queue<int> q;
    for (int i = 1; i <= n; i++)
        if (!in[i])
            q.push(i);
    vector<int> ans;
    while (!q.empty())
    {
        int tp = q.front();
        q.pop();
        ans.push_back(tp);
```

```
        for (auto x : G[tp])
        {
            in[x]--;
            if (!in[x])
                q.push(x);
        }
    }
    if (ans.size() == n)
    {
        printf("1\n");
        for (auto x : ans)
            printf("%d ", x);
    }
    else
    {
        printf("2\n");
        for (int i = 1; i <= n; i++)
            printf("%d ", i);
        printf("\n");
        for (int i = n; i >= 1; i--)
            printf("%d ", i);
        printf("\n");
    }
    return 0;
}
```