2023 年第八场牛客多校题解

出题人：李泽仁，高铭鸿，王冬杨，曾耀辉

1 A Alive Fossils

题意：依次举办 n 场多校，每场多校有一些出题人。问哪些出题人每场都出题了。

解法：用 `set` 维护下一直在出题的人即可。

2 B Bloodline Counter

题意：求 n 个点的竞赛图中最大环大小恰好为 k 的方案数对 998 244 353 取模。$3 \le k \le n \le 5 \times 10^5$。

解法：首先看到这类不同方案求和，且约束条件为恰好的题目，第一反应就是转变成约束条件满足某个至多条件然后相减。因而原问题可以转化为，最大强连通分量至多为 k 个点方案数。

为计算出一个大小为 k 的强连通分量有多少种构成方式，这里直接统计最大环是谁（圆排列数），环内边怎么连（任意连）这种方法是不对的——五角星存在两种哈密顿路，即同一个强连通分量可以有两种这样的构造方式。这种方式仅能求出竞赛图哈密顿路总数，即 $(n-1)!2^{\binom{n}{2}-n}$。

为计算出一个大小为 k 的强连通分量有多少种构成方式，正确做法是考虑容斥。经过缩点后，一个 n 个点的竞赛图一定是由一条链构成。这时仅考察这条链上最末端的一个强连通分量的大小，假设为 i，则有：

$$f_n = \sum_{i=1}^{n} \binom{n}{i} g_i 2^{\binom{n-i}{2}}$$

其中 f_n 表示 n 个点竞赛图的方案数，显然为 $2^{\binom{n}{2}}$，g_i 表示大小为 i 的竞赛图构成一整个强连通分量方案数。该方程含义为，枚举最末端一个强连通分量大小，剩下的任选，就可以构成全部的竞赛图方案。考虑如何快速计算上式：

$$2^{\binom{n}{2}} = \sum_{i=1}^{n} \binom{n}{i} g_i 2^{\binom{n-i}{2}}$$

$$\frac{2^{\binom{n}{2}}}{n!} = \sum_{i=1}^{n} \frac{g_i}{i!} \frac{2^{\binom{n-i}{2}}}{(n-i)!}$$

因而可以看成是 $f(x) = \sum_{i=0}^{+\infty} \frac{2^{\binom{i}{2}}}{i!} x^i$ 等于 $g(x) = \sum_{i=1}^{+\infty} \frac{g_i}{i!} x^i$ 与 $h(x) = \sum_{i=0}^{+\infty} \frac{2^{\binom{i}{2}}}{i!} x^i$ 卷积之后再增补单独的常数项 1——因为 $g(x)$ 没有常数项，而 $f(x)$ 的常数项为 1。不难注意到 $f = h$，因而有 $f(x) = g(x)f(x) + 1$，可得 $g(x) = 1 - \frac{1}{f(x)}$。

由于此处需要每个强连通分量大小不超过 k，因而取 $g(x)$ 的前 k 项出来。由于此处是需要先选点后组成强连通分量，因而考虑指数生成函数 EGF 求取答案：

$$[x^n] \left(\sum_{i=1}^{k} \frac{g_i}{i!} x^i \right)^n$$

此处使用指数生成函数的原因是因为多项式系数下都带有指数。例如，若当前强连通分量大小组成为 $\{a_1, a_2, \cdots, a_k\}$，为选点再排列，则对于该方案，就必须带有该多项式系数：

$$\binom{n}{a_1, a_2, \cdots, a_k} \prod_{i=1}^{k} g_{a_i} = n! \prod_{i=1}^{k} \frac{g_{a_i}}{a_i!}$$

因而使用指数生成函数就可以自动将强连通分量的大小的阶乘引入系数中，而不用单独计算贡献。

因而经过求逆和多项式快速幂可以做到 $\mathcal{O}(n\log n)$ 的复杂度。

```
void Solve() {
    int n, k;
    scanf("%d%d", &n, &k);
    Poly g(k + 1), p2(k + 1);
    g[0] = p2[0] = 1;
    fp(i, 1, k) p2[i] = 2 * p2[i - 1] % P;
    fp(i, 1, k) g[i] = mul(g[i - 1], p2[i - 1]);
    fp(i, 1, k) g[i] = mul(g[i], ifac[i]);
    g = g.inv(k + 1);
    int a = g.pre(k + 1).inv(n + 1)[n], b = g.pre(k).inv(n + 1)[n];
    printf("%lld\n", ll(a - b + P) * fac[n] % P);
}
```

3 D Distance on Tree

题意：给定大小为 n 的有根树，树上点有点权 $\{a\}_{i=1}^{n}$，边有边权。q 次询问 u 子树上任选三点满足三点权值和模 m（为一常数）为定值 k 时，两两之间距离和的最大值。$1\le n,m\le 2\times 10^3$，$1\le q\le 2\times 10^5$。

解法：对于此类子树问题，可以考虑不断将 u 的一个新子树 v 加入到子树 u 的答案中。首先考虑一个最基本的 dp：$f_{u,1,k}$ 表示子树 u 中任选一个权值模 m 等于 k 的点到 u 的最大距离的两倍（之所以用两倍是方便转移式的方便和形式统一），$f_{u,2,k}$ 表示子树 u 中任选两个点权值和模 m 为 k 的距离和加上选出的这两点到 u 的距离和的最大值，$f_{u,3,k}$ 表示 u 子树下任选三点其权值和模 m 等于 k 的约束下，两两之间距离和的最大值。

考虑从 u 的子节点 v 进行朴素转移，此时 (u,v) 边权为 c：

1. $f_{u,1,k}\leftarrow\max(f_{u,1,k},f_{v,2,k}+2c)$。
2. $f_{u,2,k}\leftarrow\max(f_{u,2,k},f_{u,1,k-j}+f_{v,1,j}+2c,f_{v,2,k}+2c)$，即新子树可以选择 1 个或两个点执行转移。
3. $f_{u,3,k}\leftarrow\max(f_{u,1,k-j}+f_{v,2,j}+2c,f_{u,2,k-j}+f_{v,1,j}+2c,f_{v,3,k})$，即还是枚举新子树加入点数目。

但是这样暴力转移的复杂度是 $\mathcal{O}(nm^2)$——$\mathcal{O}(nm)$ 个状态，每个状态需要 $\mathcal{O}(m)$ 的转移时间。

考虑如何优化这一 dp。其实不难发现，当子树大小很小时，$f_{u,1,k}$ 和 $f_{u,2,k}$ 根本达不到 $\mathcal{O}(m)$ 量级。可以考虑只从合法状态进行转移，这样就可以将状态数下降到 $\mathcal{O}(\min(|S_u|^2,m))$。考虑这样做的复杂度为什么正确，下面仅分析瓶颈部分转移也就是第三个转移式中分别从新旧子树中各取两个点和一个点的情况，以子树大小为 $\sqrt{m}$ 为分界，子树大小大于这一阈值的认为是大点，反之为小点。并约定 u 仅考虑已经合并进入自身答案部分的子树大小而非整个树上 u 子树的大小：

1. u,v 均为小点。则复杂度小于 $m\times\sqrt{m}+\sqrt{m}\times m$——$u$ 中选两个的状态数小于 m，v 中选一个的状态数小于 $\sqrt{m}$，反之同理。这种转移最多，但是单次执行复杂度小，因而这部分复杂度为 $\mathcal{O}(nm^{3/2})$。
2. u,v 均为大点。考虑使用原版暴力转移，单个点合并复杂度为 $\mathcal{O}(m^2)$，但是因为都是大点所以合并次数不超过 $\mathcal{O}\left(\frac{n}{\sqrt{m}}\right)$，因而这部分复杂度为 $\mathcal{O}(nm^{3/2})$。

3. u 为大点而 v 为小点（反过来同理）。这时 u 中选两个点 v 中选一个点的复杂度为 $m\times\sqrt{m}$，而反过来 u 选两个点 v 选一个点的复杂度为 $m\times m$。但是注意到 v 以小点身份加入 u 这一大点之后，以后关于 v 子树内部的转移就都是以大点身份进行的，因而每个节点以小点身份转移的时候，至多只会被一个大点吸收。而单次吸收的复杂度仅为 $\mathcal{O}(m)$（均摊到一个点上的贡献），因而这一部分的复杂度仅为 $\mathcal{O}(nm)$。所以这部分复杂度为 $\mathcal{O}(nm^{3/2}+nm)$。

因而可以仅从可行状态进行转移。复杂度 $\mathcal{O}(nm^{3/2})$。代码中 dp 状态的第三个参数 $0-2$ 对应上文中的 $1-3$。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 2e3 + 5;
int n, m, Q, a[N];
int f[N][N][3], ans[N][N];
vector<pair<int, int>> V[N];
pair<int, int> valid[] = {{1, 0}, {0, 1}, {0, 0}};
void dfs(int x, int fa)
{
    f[x][a[x]][0] = 0;
    vector<int> _x, _j;
    for (auto [j, c] : V[x])
    {
        if (j == fa)
            continue;
        dfs(j, x);
        for (auto [i, k] : valid)
        {
            _x.clear(), _j.clear();
            for (int w = 0; w < m; w++)
            {
                if (f[x][w][i] != -1)
                    _x.emplace_back(w);
                if (f[j][w][k] != -1)
                    _j.emplace_back(w);
            }
            for (auto w1 : _x)
                for (auto w2 : _j)
                {
                    int nw = (w1 + w2) % m;
                    f[x][nw][i + k + 1] = max(f[x][nw][i + k + 1], f[x][w1][i] + f[j][w2][k] + c * 2);
                }
        }
        for (int i = 0; i <= 2; i++)
            for (int w = 0; w < m; w++)
                if (f[j][w][i] != -1)
                    f[x][w][i] = max(f[x][w][i], f[j][w][i] + (i == 2 ? 0 : c * 2));
    }
```

```
}
int main()
{
    cin >> n >> m >> Q;
    memset(f, -1, sizeof f);
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 1; i < n; i++)
    {
        int x, y, z;
        cin >> x >> y >> z;
        V[x].push_back({y, z});
        V[y].push_back({x, z});
    }
    dfs(1, 0);
    while (Q--)
    {
        int x, w;
        cin >> x >> w;
        cout << max(0, f[x][w][2]) << endl;
    }
    return 0;
}
```

4 G-Expected Distance

题意：平面上给出两条相互垂直的线段，问在这两条线上随机各取一个点的距离期望。点横纵坐标范围 $[-10^3, 10^3]$。

解法：考虑把图通过旋转和对称操作使得两条线段都位于坐标轴上，则原式可写为：

$$\frac{1}{(b-a)(d-c)}\int_a^b\int_c^d \sqrt{x^2+y^2}\mathrm{d}x\mathrm{d}y$$

首先考虑进行拆分：$\int_a^b f(x)\mathrm{d}x = \int_a^0 f(x)\mathrm{d}x + \int_0^b f(x)\mathrm{d}x$，因而把原式化为四个形如下式的积分：

$$\int_0^b\int_0^d \sqrt{x^2+y^2}\mathrm{d}x\mathrm{d}y$$

对于此类根式积分，常见思路是三角换元或极坐标代换处理。

$$
\begin{aligned}
&\int_0^b \int_0^d \sqrt{x^2+y^2}\mathrm{d}x\mathrm{d}y \\
=&\int_0^b \int_0^{\frac{d}{b}x} \sqrt{x^2+y^2}\mathrm{d}x\mathrm{d}y + \int_0^b \int_{\frac{d}{b}x}^d \sqrt{x^2+y^2}\mathrm{d}x\mathrm{d}y \\
=&\int_0^{\arctan\frac{d}{b}} \mathrm{d}\theta \int_0^{\frac{b}{\cos\theta}} r^2\mathrm{d}r + \int_{\arctan\frac{d}{b}}^{\frac{\pi}{2}} \mathrm{d}\theta \int_0^{\frac{d}{\sin\theta}} r^2\mathrm{d}r \\
=&\frac{b^3}{3}\int_0^{\arctan\frac{d}{b}} \frac{1}{\cos^3\theta}\mathrm{d}\theta + \frac{d^3}{3}\int_{\arctan\frac{d}{b}}^{\frac{\pi}{2}} \frac{1}{\sin^3\theta}\mathrm{d}\theta \\
=&\frac{b^3}{3}\int_0^{\arctan\frac{d}{b}} \frac{1}{\cos^4\theta}\mathrm{d}\sin\theta + \frac{d^3}{3}\int_0^{\arctan\frac{b}{d}} \frac{1}{\cos^4\theta}\mathrm{d}\sin\theta
\end{aligned}
$$

考虑 $\int \frac{1}{\cos^4 x}\mathrm{d}\sin x$：

$$
\begin{aligned}
&\int \frac{1}{\cos^4 x}\mathrm{d}\sin x \\
=&\int \frac{1}{(1-u^2)^2}\mathrm{d}u \\
=&\int \frac{1}{4(u+1)} + \frac{1}{4(u+1)^2} - \frac{1}{4(u-1)} + \frac{1}{4(u-1)^2}\mathrm{d}u \\
=&\frac{1}{4}\ln(\sin x+1) - \frac{1}{4(\sin x+1)} - \frac{1}{4}\ln(\sin x-1) - \frac{1}{4(\sin x-1)}\mathrm{d}u
\end{aligned}
$$

带入上式即可。或者利用积分公式：

$$
\begin{aligned}
&\int \frac{\mathrm{d}x}{\cos^n x} = \frac{1}{n-1}\frac{\sin x}{\cos^{n-1} x} + \frac{n-2}{n-1}\int \frac{\mathrm{d}x}{\cos^{n-2} x} \\
&\int \frac{1}{\cos x}\mathrm{d}x = \ln|\sec x + \tan x| + C
\end{aligned}
$$

5 H Insert 1, Insert 2, Insert 3, ...

题意：给定长度为 n 的序列 $\{a\}_{i=1}^n$，问有多少个连续子区间 $[l,r]$ 满足这个子区间可以通过若干次依次按顺序插入 $1,2,3\cdots,k$ 的子序列构成。$1\le n\le 10^6$，$1\le a_i\le n$。

解法：显然，每个合法的区间一定是以 1 开头。考虑维护每个右端点 r 有多少个合法的左端点（即数字 1）能和他组成一个合法的区间。对于非 1 的数字 x，首先就近寻找最近的 $x-1$ 在哪里，藉此维护出为了填出当前的数字 x，最近（最靠右）的 1 在哪里。

这时在这个 1 右侧的，且不超过当前右端点的全部的 1 一定就不合法了，且随着枚举的右端点不断右移，这些被标记为不合法的 1 也永远不可能变成合法的。因而找到这个最近的 1 之后统计它左侧有多少合法的 1 即可。总时间复杂度 $\mathcal{O}(n)$。

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

using namespace std;
using ll = long long;
const int N = 1e6 + 5;
int n; ll ans;
vector<int> p[N], s;
void Solve() {
    scanf("%d", &n);
    for (int x, y, i = 1; i <= n; ++i) {
```

```
        scanf("%d", &x), --x;
        if (!x) p[1].push_back(i), s.push_back(i), ans += s.size();
        else if (p[x].empty()) s.clear();  // 注意：这里理论上要把所有的p[x]都清空，但是复杂度不允许。因而在第16行增加s.empty()判断，和这里清空所有栈的操作等价。
        else {
            y = p[x].back(), p[x].pop_back(), p[x + 1].push_back(y);
            while (!s.empty() && s.back() > y) s.pop_back();
            ans += s.size();
        }
    }
    printf("%lld\n", ans);
}
int main() {
    int t = 1;
    // scanf("%d", &t);
    while (t--) Solve();
    return 0;
}
```

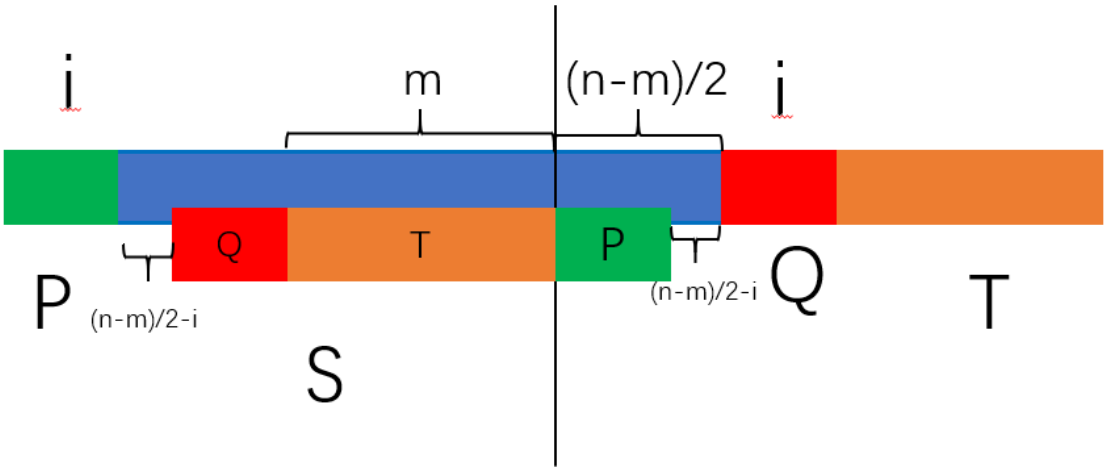
6 I Make It Square

题意：给定串 S 和 T，问有多少种长度均为 i 的串 P,Q，使得 $\text{concat}(P,S,Q,T)$ 为一个 `AA` 型的串，其中 `A` 为一任意非空字符串。可用字符集为小写字母。对 $i\in[1,m]$ 求出答案。$1\le m\le 10^6$，$1\le |S|,|T|\le 10^6$。

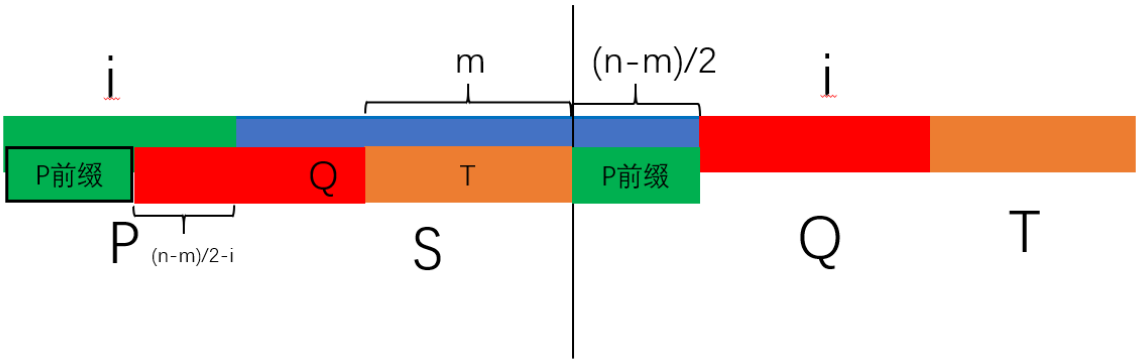
解法：如果 $|S|=|T|$，则 P 一定和 Q 匹配。只需要观察是否 $S=T$ 即可判断是否有解，有解就是 26^k 种。下记枚举的 P,Q 长度为 i。显然，如果 $|S|+|T|$ 为奇数直接无解。下面 $n+m$ 均为偶数。

如果 $|S|>|T|$：

1. 当 $i\le\dfrac{n-m}{2}$ 时，会发现 T 在 S 中匹配的位置是固定的：截取 S 后面 $\dfrac{n-m}{2}$ 个字符，然后 S' 和 T 后缀匹配。然后这时由图上所示，S 有一部分串需要自己的 border 进行匹配。因而使用 KMP 求出 S 所有的 border 即可。如果发现 border 长度满足条件，且能够匹配，则存在唯一一组解；否则无解。

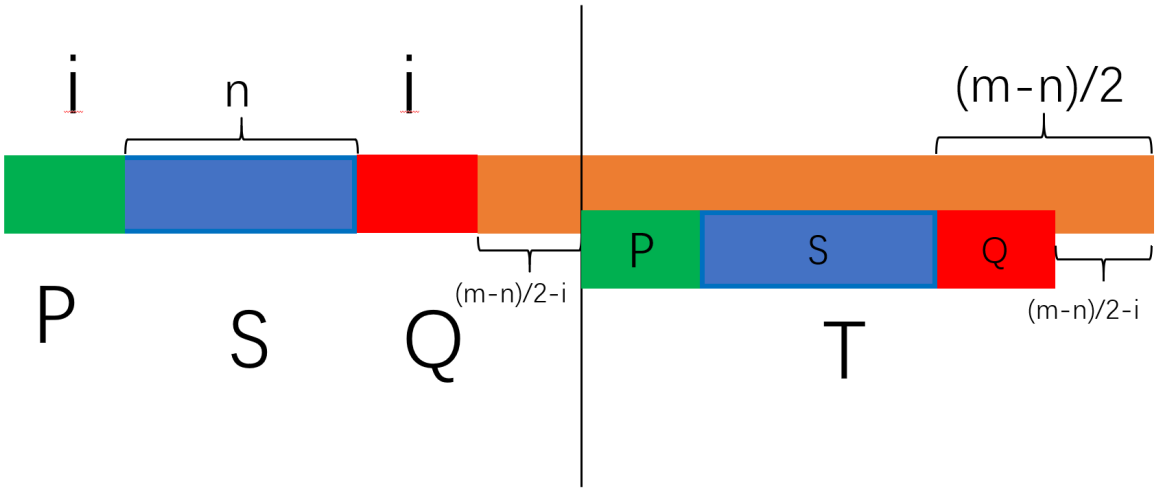


2. 当 $i>\dfrac{n-m}{2}$ 时，T 在 S 中匹配位置仍然是固定的，但是这时是 P 和 Q 串重叠，这时可选自由长度为 $\dfrac{n-m}{2}-i$。如果 S 和 T 能够匹配，则答案为 $26^{\frac{n-m}{2}-i}$。

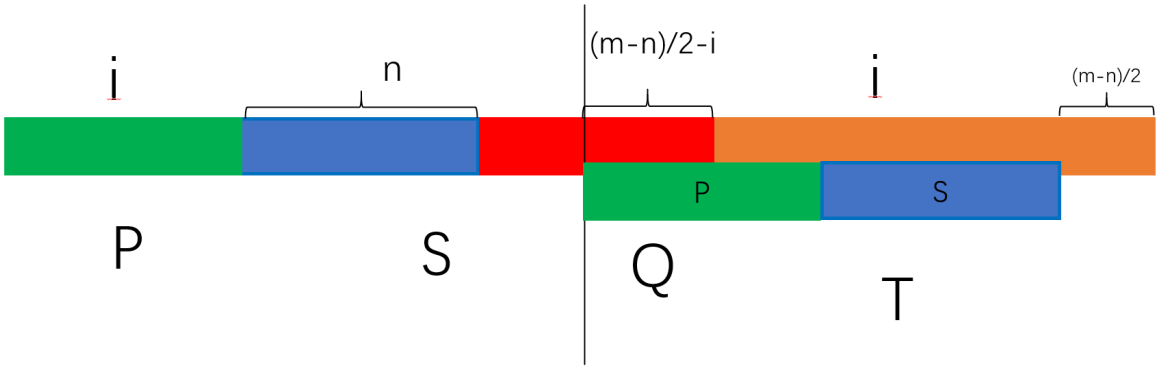


如果 $|S|<|T|$，

1. 当 $i\le\frac{m-n}{2}$ 时，情况同上：需要查看 S 是不是能在 T 固定位置匹配，以及 T 是否存在长度为 $\frac{m-n}{2}-i$ 的 border。同样使用 KMP 求解。



2. 当 $i>\frac{m-n}{2}$ 时，S 匹配段还是 T 去掉 $\frac{m-n}{2}$ 个字符的后缀处，自选段为 P 和 Q 重叠的 $\frac{m-n}{2}-i$ 的长度。



因而就这两种情况展开分类讨论即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1000000;
char s[N + 5], t[N + 5];
const int mod = 998244353;
class KMP
```

```
{
    vector<int> nx;
    string b;

public:
    KMP(string b)
    {
        this->b = b;
        int n = b.length();
        int j = 0;
        nx.resize(n);
        for (int i = 1; i < n; i++)
        {
            while (j > 0 && b[i] != b[j])
                j = nx[j - 1];
            if (b[i] == b[j])
                j++;
            nx[i] = j;
        }
    }
    int find(string &a)
    {
        int n = b.length(), m = a.length();
        int j = 0;
        long long ans = 0;
        for (int i = 0; i < m; i++)
        {
            while (j > 0 && a[i] != b[j])
                j = nx[j - 1];
            if (a[i] == b[j])
                j++;
            if (j == n)
            {
                // 匹配位点:i-n+1
                ans++;
                j = nx[j - 1];
            }
        }
        return ans;
    }
    set<int> getBorder()
    {
        set<int> s;
        int cur = nx.back();
        while (cur)
        {
            s.insert(cur);
```

```
            cur = nx[cur - 1];
        }
        s.insert(0);
        return s;
    }
};
int main()
{
    int k;
    scanf("%d%s%s", &k, s + 1, t + 1);
    int n = strlen(s + 1), m = strlen(t + 1);
    KMP S(string(s + 1)), T(string(t + 1));
    auto border1 = S.getBorder(), border2 = T.getBorder();
    if (n == m)
    {
        if (strcmp(s + 1, t + 1))
            for (int i = 1; i <= k; i++)
                printf("0 ");
        else
        {
            long long base = 1;
            for (int i = 1; i <= k; i++)
            {
                base = base * 26 % mod;
                printf("%lld ", base);
            }
        }
        return 0;
    }
    if ((n + m) % 2)
    {
        for (int i = 1; i <= k; i++)
            printf("0 ");
        return 0;
    }
    else if (n > m)
    {
        bool check = true;
        for (int i = m, j = n - (n - m) / 2; i >= 1; i--, j--)
            if (t[i] != s[j])
            {
                check = false;
                break;
            }
        if (check)
        {
            long long cur = 1;
```

```
            for (int i = 1; i <= k; i++)
            {
                if (i > (n - m) / 2)
                {
                    cur = cur * 26 % mod;
                    printf("%lld ", cur);
                }
                else if (border1.count((n - m) / 2 - i) == 0)
                    printf("0 ");
                else
                    printf("1 ");
            }
        }
        else
            for (int i = 1; i <= k; i++)
                printf("0 ");
    }
    else
    {
        bool check = true;
        for (int i = (m - n) / 2 + 1, j = 1; j <= n; i++, j++)
            if (s[j] != t[i])
            {
                check = false;
                break;
            }
        long long cur = check;
        for (int i = 1; i <= k; i++)
        {
            // 落点在t
            if (2 * i + n <= m)
                printf("%d ", check & (border2.count((m - n) / 2 - i)));
            // 落点在q
            else
            {
                cur = cur * 26 % mod;
                printf("%lld ", cur);
            }
        }
    }
    return 0;
}
```

7 J Permutation and Primes

题意：构造一个长度为 n 的排列，使得任意两个相邻数字的和或差的绝对值为一奇质数。多测，$1 \le T \le 10^5$，$1 \le n \le 10^5$。

解法：对于此类相邻差和和为质数的题，通常来说会联想到构造同余序列 $1, 1+p, 1+2p, \cdots, 1+kp, 2, 2+p, \cdots, 2+kp, \cdots, p, p+p, \cdots, p+kp$。类似题有 2023 年 CCPC 河南省赛 K 题。

所以首先可以考虑模 3 构造 $1, 4, 7, \cdots, 2, 5, 8, \cdots, 3, 6, 9, \cdots$。但是这样会导致衔接处不一定合法——如 89 下三个序列的 $1, 88, 2, 89, 3, 87$ 就无法完美衔接成一个序列。因而可以考虑模 5，试验后发现模 5 的端点都可以拼接起来了。

```cpp
#include <bits/stdc++.h>
#define fp(i, a, b) for (int i = a, i##_ = int(b); i <= i##_; ++i)
#define fd(i, a, b) for (int i = a, i##_ = int(b); i >= i##_; --i)

using namespace std;
using ll = long long;
const int N = 2e5 + 5;
int n, prime[N], vis[N], tot;
void sieve(int n)
{
    vis[1] = 1;
    fp(i, 2, n)
    {
        if (!vis[i])
            prime[++tot] = i;
        for (int j = 1; j <= tot && 1ll * prime[j] * i <= n; ++j)
        {
            int num = prime[j] * i;
            vis[num] = 1;
            if (i % prime[j] == 0)
                break;
        }
    }
    vis[2] = 1;
}
bool Chk(int x, int y)
{
    return !vis[abs(x - y)] || !vis[x + y];
}
void Solve()
{
    scanf("%d", &n);
    if (n <= 4)
    {
        fp(i, 1, n) printf("%d ", i);
        puts("");
        return;
```

```
    }
    vector<int> a[5];
    for (int i = 1; i <= n; i++)
        a[i % 5].push_back(i);
    vector<int> seq;
    if(n%5==0) seq={-3, 4, -1, 2, -5};
    /*
    5 1 2 3 4
    10 6 7 8 9
    */
    if(n%5==1) seq={-4, 3, -1, 2, -5};
    /*
    5 1 2 3 4
    10 11 7 8 9
    */
    if(n%5==2) seq={-5, 2, -4, 1, -3};
    /*
    5 1 2 3 4
    10 11 12 8 9
    */
    if(n%5==3) seq={-1, 2, -4, 3, -5};
    /*
    5 1 2 3 4
    10 11 12 13 9
    */
    if(n%5==4) seq={-2, 1, -4, 3, -5};
    /*
    5 1 2 3 4
    10 11 12 13 14
    */
    vector<int> ans;
    for(int i=0;i<5;i++) {
        int now=abs(seq[i]);
        if(now==5) now=0;
        if(seq[i]<0) {
            reverse(a[now].begin(), a[now].end());
        }
        ans.insert(ans.end(), a[now].begin(), a[now].end());
    }
    for(int v:ans) printf("%d ", v);
    puts("");
}
int main()
{
    sieve(N - 5);
    int t = 1;
    scanf("%d", &t);
```

```
    while (t--)
        Solve();
    return 0;
}
```

另一个构造是 8 个一组：

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=100007;
int T;
int N;
int main()
{

    scanf("%d",&T);
    while(T--)
    {
        scanf("%d",&N);
        int x=N%8;
        for(int i=N;i-7>0;i-=8)
            printf("%d %d %d %d %d %d %d %d ",i,i-3,i-6,i-1,i-4,i-7,i-2,i-5);
        if(x==1)
            printf("1");
        else if(x==2)
            printf("2 1");
        else if(x==3)
            printf("1 2 3");
        else if(x==4)
            printf("4 3 2 1");
        else if(x==5)
            printf("5 2 1 4 3");
        else if(x==6)
            printf("4 1 2 3 6 5");
        else if(x==7)
            printf("7 6 5 2 3 4 1");
        printf("\n");
    }
    return 0;
}
```

8 K Scheming Furry

题意：给定一 $n \times m$ 的数组 $\{a\}$，$[1, nm]$ 各出现一次。两个人先后手对数组操作，先手一次必须交换两行，后手一次必须交换两列，谁先把数组变得有序（即按行优先顺序放置数字的顺序）谁获胜。保证初始局面无序，问谁获胜。多测，$1 \le T \le 100$，$1 \le n, m \le 2 \times 100$。

解法：首先判断当前局面是否可以通过若干次行列交换变成有序——第 i 行最大值小于第 $i+1$ 行最小值对于任意 $i \in [1, n-1]$ 成立。如果不能，直接平局。

如果先手一步操作就可以成功排序，则先手必胜。

如果 $n \ge 3$ 且 $m \ge 3$，这时先后手都会达成一致：每次故意换错误的一对行或列，使得对手永远无法凭借自己的努力达成排序。因而这种情况必然平局。

考虑 $n = 2$ 且 $m \ge 3$ 的情况。这时先手必然只能交换这两行，这时如果后手通过列交换使得列有序所需的操作步数和先手对行做操作，使得行从最初状态变成行有序的操作步数**奇偶性**相同，则后手必胜，反之平局。后手执行列交换使得有序的步数必然和列置换（列大小排名视为一个置换）上各个置换子环大小减 1 之和奇偶性相同。因为一次有效的交换必然会使得置换环大小缩小 1，而无效交换必然会让置换环大小增大 1。

$n \ge 3$ 且 $m = 2$ 同理，先手对行的操作次数奇偶性和列所需操作次数奇偶性相反则先手必胜，反之平局。

当 $n = 2$ 且 $m = 2$ 的情况，由于不存在初始局面就是排好序的情况，因而当行需交换一次，列需交换一次时后手必胜，而行不需要交换，列需要交换一次时平局。行需交换一次，列不需要交换则先手必胜（第二类情况）。

```cpp
#include <bits/stdc++.h>
#define fp(i, a, b) for (int i = a, i##_ = b; i <= i##_; ++i)
#define fd(i, a, b) for (int i = a, i##_ = b; i >= i##_; --i)

using namespace std;
using ll = long long;
const int N = 200 + 5;
int n, m, a[N][N], b[N], c[N];
void p(int x) { puts(x == -1 ? "NSFW" : (x ? "FOX" : "CAT")); }
int calc(int *a, int k) {
    int cnt = 0;
    vector<int> pos(k + 1);
    fp(i, 1, k)
        while (a[i] != i)
            swap(a[i], a[a[i]]), ++cnt;
    return cnt;
}
void Solve() {
    scanf("%d%d", &n, &m);
    fp(i, 1, n) fp(j, 1, m) scanf("%d", a[i] + j);
    fp(i, 1, n) {
        int mx = 0, mn = 1e9;
        fp(j, 1, m) mx = max(a[i][j], mx), mn = min(a[i][j], mn);
        if (mx - mn != m - 1 || mn % m != 1)
            return p(-1);
        b[i] = (mn + m - 1) / m;
```

```
    }
    fp(j, 1, m) {
        int mn = 1e9;
        fp(i, 1, n) mn = min(a[i][j], mn);
        fp(i, 1, n)
            if ((a[i][j] - mn) % m)
                return p(-1);
        c[j] = mn;
    }
    int v1 = calc(b, n), v2 = calc(c, m);
    if (v1 == 1 && !v2) p(1);
    else if (n >= 3 && m >= 3) p(-1);
    else if (n == 2 && m == 2) p(v1 % 2 != v2 % 2);
    else if (n == 2) p(v1 % 2 == v2 % 2 ? 0 : -1);
    else p(v1 % 2 == v2 % 2 ? -1 : 1);
}
int main() {
    int t = 1;
    scanf("%d", &t);
    while (t--) Solve();
    return 0;
}
```