沈阳师范大学 2025 团体程序设计天梯赛(GPLT) 校内选拔赛

概览

题目编号	题目名称	题目分值		题目编号	题目名称	题目分值
L1-1	请遵守规则	5		L2-1	蝴蝶变化	25
L1-2	虚数计算	5		L2-2	瓦学弟的救赎	25
L1-3	实数取模	10		L2-3	消灭星星	25
L1-4	容斥原理	10		L2-4	分类网络	25
L1-5	图像放大	15		L3-1	图上极差	30
L1-6	Orange与他的水果朋友们	15		L3-2	哨兵划分	30
L1-7	摩斯电码	20		L3-3	分类网络(HardVersion)	30
L1-8	解方程	20				

题解报告

L1-1 请遵守规则

直接输出即可：

```
print("I understand the serious consequences of cheating and promise to participate with integrity!")
```

L1-2 虚数计算

直接计算即可。

$$(a+bi)(c+di) = ac + adi + bci + bdi^2 = ac - bd + (ad+bc)i$$

$$\text{ans} = \sqrt{(ac-bd)^2 + (ad+bc)^2}$$

```
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
using i64 = long long;

signed main()
{
    cin.tie(0) -> sync_with_stdio(0);
    i64 a, b, c, d;
    cin >> a >> b >> c >> d;
    i64 u = a * c - b * d;
    i64 v = a * d + b * c;
    cout << fixed << setprecision(3) << sqrt(u * u + v * v) << '\n';
    return 0;
}
```

L1-3 实数取模

观察到题目对余数的定义，实际上最大的 $k = \lfloor \frac{a}{b} \rfloor$，那么余数 $c = a - kb$。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
using i64 = long long;

signed main()
{
    cin.tie(0) -> sync_with_stdio(0);
    double a, b;
    cin >> a >> b;
    int k = a / b;
    cout << a - b * k << '\n';
    return 0;
}
```

L1-4 容斥原理

直接暴力枚举，统计答案即可。虽然本题叫做容斥原理，但是实际上并不满足容斥原理的条件，因为并没有保证 x, y, z 互质，请不要聪明反被聪明误(xD)。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
using i64 = long long;

signed main()
{
    IOS;
    int n, x, y, z;
    cin >> n >> x >> y >> z;
    int cnt = 0;
    for (int i = 1; i <= n; i++)
        if (i % x != 0 and i % y != 0 and i % z != 0)
            cnt++;
    cout << cnt;
}
```

L1-5 图像放大

依次枚举每一行，重复处理 k 次，每次处理时枚举每个字符重复输出 k 次即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0); cin.tie(0);cout.tie(0);

signed main()
{
    IOS;
    int n, m, k;
    cin >> n >> m >> k;
    vector<string> ss(n);
    for (int i = 0; i < n; i++)
        cin >> ss[i];
    for (int i = 0; i < n; i++)
        for (int l = 0; l < k; l++) // 每一行重复处理 k 次
        {
            for (int j = 0; j < m; j++)
                for (int o = 0; o < k; o++) // 每个字符重复输出 k 次
                    cout << ss[i][j];
            cout << endl;
        }
    return 0;
}
```

L1-6 Orange与他的水果朋友们

本题主要考察对模拟和映射的理解。

每次，编号为 i 的水果跳到平台 `0` 所在的平台上，实际上可以看成是编号 i 所在的平台与 `0` 所在的平台交换了位置。那么，我们只需要维护一个映射 pos，记录每个水果所在的平台编号即可。每次，我们只需要将编号为 i 的水果所在的平台编号与 `0` 所在的平台编号交换即可。

最后，将所有的映射还原回去即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
using i64 = long long;

signed main()
{
    IOS;
    int n;
    cin >> n;
    vector<int> a(n + 1), pos(n + 1);
    for(int i = 1; i <= n; i ++)
        cin >> a[i], pos[a[i]] = i;
    for(int i = 1; i < n; i ++) swap(pos[i], pos[0]);
    for(int i = 0; i < n; i ++) a[pos[i]] = i;
    for(int i = 1; i <= n; i ++)
        cout << a[i] << ' ';
    return 0;
}
```

Bonus

如果你的注意力惊人，不难发现，最终的答案序列就是 $(a_i + 1) \bmod n$。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
using i64 = long long;
signed main()
{
    IOS;
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        int x;
        cin >> x;
        cout << (x + 1) % n << ' ';
    }
    return 0;
}
```

L1-7 摩斯电码

纯纯打表题，没有任何技巧。

当然，对于这样空格隔开的字符串可以考虑用 `while(cin >> s)` 进行读入，这样就自动达到了分割字符串的效果，映射可以考虑用 `map` 容器。

> GPLT就喜欢这样的题不是吗？xD

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
using i64 = long long;
signed main () {
    IOS;
    map<string, string> mp;
    mp["01"] = "A";
    mp["1000"] = "B";
    mp["1010"] = "C";
    mp["100"] = "D";
    mp["0"] = "E";
    mp["0010"] = "F";
    mp["110"] = "G";
    mp["0000"] = "H";
    mp["00"] = "I";
    mp["0111"] = "J";
    mp["101"] = "K";
    mp["0100"] = "L";
    mp["11"] = "M";
    mp["10"] = "N";
    mp["111"] = "O";
    mp["0110"] = "P";
    mp["1101"] = "Q";
    mp["010"] = "R";
    mp["000"] = "S";
    mp["1"] = "T";
    mp["001"] = "U";
    mp["0001"] = "V";
    mp["011"] = "W";
    mp["1001"] = "X";
    mp["1011"] = "Y";
    mp["1100"] = "Z";
    mp["01111"] = "1";
    mp["00111"] = "2";
    mp["00011"] = "3";
    mp["00001"] = "4";
    mp["00000"] = "5";
    mp["10000"] = "6";
    mp["11000"] = "7";
    mp["11100"] = "8";
    mp["11110"] = "9";
    mp["11111"] = "0";
    mp["001100"] = "?";
    mp["10010"] = "/";
    mp["101101"] = "()";
    mp["100001"] = "-";
    mp["010101"] = ".";

    string s;    while(cin >> s) cout << mp[s];
    return 0;
}
```

L1-8 解方程

首先，本题的第一个难点在于解析字符串，当然，如果你记得C语言的 `scanf` 的话，你可以很轻松的使用 `scanf("%dx%d=%d", &a, &b, &c)` 把方程的各个参数提取出来。

当你解析完方程的各个参数后，由于题目保证了方程一定有解，且一定为正整数，因此根据等式性质 $x_i = \frac{c_i - b_i}{a_i}$ 。

接下来，对于如何统计在 $L \leq x \leq R$ 的范围内，有多少个正整数 x 满足 x 是其中至少一个方程的解，这一步可以直接暴力求解，直接暴力枚举 L 到 R 的所有整数就行，之前求解得到的 x 值，可以用 map 之类的容器进行标记，这样方便验证答案。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

signed main()
{
    int n, q;
    cin >> n >> q;
    map<int, int> mp;
    for(int i = 1; i <= n; i ++)
    {
        int a, b, c;
        scanf("%dx%d=%d",&a,&b,&c);
        mp[(c - b) / a] = 1;
    }
    while(q --)
    {
        int l, r;
        cin >> l >> r;
        int ans = 0;
        for(int i = l; i <= r; i ++) if(mp[i]) ans ++;
        cout << ans << '\n';
    }
    return 0;
}
```

Bonus

本题原来是作为下周 CEIT 周赛的题的，临时决定改到选拔赛。本题原始的数据范围均为 2×10^5，但是想到作为L1的题，不应包含任何算法知识，遂将范围改小，允许暴力通过。

这里给出一种适用于原始数据范围的做法，同样考虑求出所有的 x_i，然后我们要验证的是有**多少种不同的** x_i 满足 $L \leq x_i \leq R$。可以考虑先将所有的 x_i 排序然后去重，在 x_i 中二分查找 L 和 R，即可将每次询问优化到 $O(\log n)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

signed main()
{
    int n, q;
    cin >> n >> q;
    vector<int> x;
    for(int i = 1; i <= n; i ++)
    {
        int a, b, c;
        scanf("%dx%d=%d",&a,&b,&c);
        x.push_back((c - b) / a);
    }
    sort(x.begin(), x.end());
    x.erase(unique(x.begin(), x.end()), x.end());
    while(q --)
    {
        int l, r;
        cin >> l >> r;
        int ans = 0;
        auto L = lower_bound(x.begin(), x.end(), l);
        auto R = upper_bound(x.begin(), x.end(), r);
        cout << R - L << '\n';
    }
    return 0;
}
```

L2-1 蝴蝶变换

模拟题，可以思考形如归并排序的方法，依次递归每个区间，将奇与偶分别存到一个数组，再向下递归，直到数组元素大小为1，此时直接输出答案即可。

每层都会将数组的大小除以二，因此最多不超过 $\log n$ 层，因此复杂度为 $O(n \log n)$。

```
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

void dfs(vector<int> a)
{
    if(a.size() == 1)
    {
        cout << a[0] << ' ';
        return;
    }
    vector<int> even, odd;
    for(int i = 0; i < a.size(); i ++)
        (i % 2 == 0 ? odd : even).push_back(a[i]);
    dfs(odd);
    dfs(even);
}

signed main()
{
    int n;
    cin >> n;
    vector<int> a(n);
    for(int i = 0; i < n; i ++) a[i] = i;
    dfs(a);
    return 0;
}
```

Bonus

没错，本题也有故事！如果你注意力惊人：不难发现，**每个数的最终位置其实就是他的二进制数的反转**。

例如：

原数字	0	1	2	3	4	5	6	7
原数二进制	000	001	010	011	100	101	110	111
原数二进制反转	000	100	010	110	001	101	011	111
原数二进制反转对应的数	0	4	2	6	1	5	3	7

因此，我们可以考虑动态规划，设 $rev[i]$ 表示第 i 个数的二进制反转，则有转移方程：

$$rev[i] = rev[i >> 1] >> 1 \mid (i \,\&\, 1) << k;$$

其中 k 表示最高位二进制的长度，可以自行思考这个转移方程的含义。

因此综上所述，我们可以得到一个 $O(n)$ 的递推做法。所以本题最多承受的极限数据范围可以到 10^8。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
signed main()
{
    IOS;
    int n;
    cin >> n;
    vector<int> rev(n);
    int k = __builtin_ctz(n) - 1;
    for (int i = 0; i < n; i++) rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
    for(int i = 0; i < n; i ++) cout << rev[i] << ' ';
    cout << '\n';
    return 0;
}
```

L2-2 瓦学弟的救赎

仔细读完题目，很明显能发现这是一道非常简单模拟题，我们只需要模拟8个枪线上的k发子弹的伤害，然后输出max即可。只是在模拟上需要一点点的特判，和一点点的细心。

对于每一发子弹我们能够从题意得出：

- `T`：不可穿过
- `W`：直接穿过
- `O`：对墙造成伤害，子弹基础伤害减少10点
- `C`：子弹不可穿过，要对墙体造成伤害，击穿的一发会带着溢出伤害前进
- `B`：造成伤害，基础伤害减少10点前进
- `H`：造成双倍伤害，基础伤害减少10点前进

为了方便记录每颗子弹射击后的血量剩余，可以先遍历整个地图，将可击碎墙体和人体的血量记录下来再开始模拟。

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, k, v, res;
char g[510][510]; // 存地图
int hp[510][510]; // 存墙和人物的hp

int dx[] = {-1, -1, -1, 0, 1, 1, 1, 0};
int dy[] = {-1, 0, 1, 1, 1, 0, -1, -1};
int sea(int det, int x, int y, int atk)
{
   int rey = 0;
   while (atk > 0)
   {
        x += dx[det]; // 加上位移偏移方向
        y += dy[det];
        if (x < 1 || x > n || y < 1 || y > n)
                // 子弹超出地图停止模拟
                break;
        if (g[x][y] == 'T')
                // 以下按照题意编写
                break;
        else if (g[x][y] == 'W')
                continue;
        else if (g[x][y] == 'O')
                atk -= 10;
        else if (g[x][y] == 'C')
        {
                if (hp[x][y] <= 0)
                        continue;
                else if (hp[x][y] >= atk)
                {
                        hp[x][y] -= atk;
                        atk = 0;
                }
                else
                {
                        atk -= hp[x][y];
                        hp[x][y] = 0;
                }
        }
        else if (g[x][y] == 'B')
        {
                int c = 0;
                if (hp[x][y] != 0)
                        c = -10;
                rey += min(hp[x][y], atk);
                hp[x][y] = max(0, hp[x][y] - atk);
                atk += c;
        }
        else if (g[x][y] == 'H')
        {
                int c = 0;
                if (hp[x][y] != 0)
                        c = -10;
                rey += min(hp[x][y], atk * 2);
                hp[x][y] = max(0, hp[x][y] - atk * 2);
                atk += c;
        }
   }
   return rey;
}
int main()
{
   cin >> n >> k >> v;
   int stx, sty;
   for (int i = 1; i <= n; i++)
   {
        for (int j = 1; j <= n; j++)
```

```
        {
                cin >> g[i][j];
                if (g[i][j] == 'B' || g[i][j] == 'H')
                        hp[i][j] = 150;
                if (g[i][j] == 'C')
                        hp[i][j] = 100;
                if (g[i][j] == 'X')
                        stx = i, sty = j;
        }
    }
    int maxn = 0;
    for (int i = 0; i <= 7; i++)
    {
        res = 0;
        for (int j = 0; j < k; j++)
                // 发射k发子弹
                res += sea(i, stx, sty, v); // i方位第1发子弹;
        maxn = max(res, maxn);
    }
    cout << maxn << endl;
}
```

L2-3 消灭星星

同样没有什么算法，考虑每个连通块的消除顺序。对消除顺序做 `dfs` ，先 `bfs` 搜索每个连通块，对于其大小大于3的，就尝试将其消除，然后对所有连通块模拟下落，再向下搜索。代码实现细节可以看下面的方法。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 10, INF = 1e9;
int n, m;
using pii = pair<int, int>;
int dx[] = {1, -1, 0, 0}, dy[] = {0, 0, -1, 1};
int ans = 0;
vector<pii> bfs(vector<vector<int>> g, pii p, vector<vector<int>> &st) // 以坐标 p 为起点，搜索连通块，把所有搜到的连通块加入到 st 中
{
    queue<pii> q;
    q.push(p);
    st[p.first][p.second] = g[p.first][p.second];
    vector<pii> res;
    res.push_back(p);
    while (q.size())
    {
        auto t = q.front();
        q.pop();
        for (int i = 0; i < 4; i++)
        {
            int px = t.first + dx[i], py = t.second + dy[i];
            if (px > 0 && py > 0 && px <= n && py <= m && !st[px][py] && g[px][py] == g[p.first][p.second])
            {
                st[px][py] = g[p.first][p.second];
                q.push({px, py});
                res.push_back({px, py});
            }
        }
    }
    return res;
}

void work(vector<vector<int>> &g, vector<pii> pos) //模拟下落
{
    for (auto [x, y] : pos)
    {
        g[x][y] = -1;
    }
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
        {
            if (g[i][j] == -1)
            {
                for(int k = i; k > 1; k--)
                {
                    swap(g[k][j],g[k-1][j]);
                }
            }
        }
}

void dfs(vector<vector<int>> g, int res) // g表示当前的矩阵状态，res表示结果
{
    vector<vector<int>> st = vector<vector<int>>(n + 1, vector<int>(m + 1));
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
        {
            if (!st[i][j] and g[i][j] != -1)
            {
                auto stat = bfs(g, {i, j}, st);
                if(stat.size() >= 3)
                {
                    auto ig = g;
                    work(ig, stat);
                    ans = max(ans, res + (int)stat.size()); // 每次都更新答案
                    dfs(ig, res + (int)stat.size());
                }
            }
```

```
        }
    }

    int main()
    {
        cin >> n >> m;
        vector<vector<int>> g = vector<vector<int>>(n + 1, vector<int>(m + 1));
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++)
                cin >> g[i][j];
        dfs(g, 0);
        cout << ans << '\n';
        return 0;
    }
```

L2-4 分类网络

这里引入一个关键的概念，**映射**。

我们假定输入的向量为 $(A_1, A_2, A_3, A_4, A_5)^T$，输出向量为 $(A_{c_1}, A_{c_2}, A_{c_3}, A_{c_4}, A_{c_5})^T$。则称 C 是对输出向量顺序到输出向量顺序的一个映射。同时，映射具有如下性质：

- 可叠加性质
  假设 $C_1 = [1, 2, 3, 5, 4]$ 是一个映射，$C_2 = [1, 3, 2, 4, 5]$ 是另一个映射，则 $C_1 \times C_2 = C_{1C_{2_i}} = [1, 3, 2, 4, 5]$。
- 可逆性
  假设 $C_1 = [1, 2, 3, 5, 4]$ 是一个映射，那么他的逆映射为 $C_1^{-1} = [1, 2, 3, 5, 4]$，即 $C_{C^{-1}} = [1, 2, 3, 4, 5]$ (自然映射)。

具体的证明会在下方给出。

通过上述性质，我们发现，每次交换两个位置，实际上就是一组交换对应位置的映射，我们可对映射做出叠加前缀和，那么 $C_{pre}[r]$ 则代表 $1 \sim r$ 层映射叠加之后的映射，又因为映射可逆，因此我们求出 $C_{pre}[l-1]$ 的逆映射，再映射到原映射上，就可以得到最终叠加了第 $l \sim r$ 层的映射了，再把这个叠加映射映射到输入的 a_i 上，最终 a_C 就是最终答案。

关于映射性质的证明

可叠加性的证明

对于一个向量:

$$\vec{A} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

假设我们要交换第一行和第二行，实际上通过矩阵的行变换，我们实际上是左乘了一个单位矩阵对应变换的置换矩阵 P

$$E = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix} \xrightarrow{\text{交换第一和第二行}} P = \begin{pmatrix} & 1 & & \\ 1 & & & \\ & & 1 & \\ & & & 1 \end{pmatrix}$$

例如我们要交换 $\vec{A}$ 的第一行和第二行，只需要左乘一个 P 矩阵即可：

$$P \times A = \begin{pmatrix} b \\ a \\ c \\ d \end{pmatrix}$$

接下来，我们要交换第三行和第四行，对应的置换矩阵 P'

$$E = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix} \xrightarrow{\text{交换第三和第四行}} P' = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & & 1 \\ & & 1 & \end{pmatrix}$$

假设我们现在要依次进行如上两次操作，实际上相当于向量 $\vec{A}$ 依次左乘 P 和 P'，即 $P \times P' \times \vec{A}$，根据矩乘法具有的结合律，我们可以先计算左边的置换矩阵：$(P \times P') \times \vec{A}$，结果不发生改变。

综上所述，我们的操作若干次，每次操作都对应向量 $\vec{A}$ 左乘一个置换矩阵 P_i，那么我们使用结合律先计算出 P_i 矩阵的前缀乘积 P_{Pre}，即可快速求出前 i 次操作的叠加。

可逆性的证明

对于上述置换矩阵 P，不难发现，对于单个置换矩阵 P，其一定是一个**正交矩阵**，因为 $P^T = P^{-1}$，可以自行验证。

又根据矩阵乘法的性质：**正交矩阵的乘积一定也正交**，那么对于上述若干次操作的叠加矩阵 P_{pre}，其一定也满足正交，由此可以推出该矩阵一定可逆，且其逆矩阵一定满足： $P_{pre}^{-1} = P_{pre}^T$。至此我们证明了操作的可逆性。

矩阵乘法的优化

我们知道，常规的矩阵乘法复杂度是 $O(n^3)$ 的，如果放到本题，有可能会超时。

我们再考虑置换矩阵的性质，实际上左乘置换矩阵，主要的效果为：其每一行 1 的位置，仅决定 A_i 在向量中的行下标。

那么根据这个性质，我们并不需要真的去做矩阵和向量对应行列的卷积，我们只需要用一个映射 C 来记录 A_i 左乘置换矩阵后的位置即可。例如：

$$P = \begin{pmatrix} & 1 & & \\ 1 & & & \\ & & & 1 \\ & & 1 & \end{pmatrix} \leftrightarrow C = \begin{bmatrix} 2 \\ 1 \\ 3 \\ 4 \end{bmatrix}$$

那么我们在对 $\vec{A}$ 做矩阵乘法实际上就优化成了：$A'_i = A_{C_i}$。对逆矩阵的操作也同理，即 $A'_{C_i} = A_i$。

这样，我们就将矩阵乘法优化到了 $O(n)$ 的复杂度，可以接受。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

signed main()
{
    IOS;
    int n, m, q;
    cin >> n >> m >> q;
    vector<vector<int>> pre(1, vector<int>(n));
    for(int i = 0; i < n; i ++) pre[0][i] = i;
    while(m--)
    {
        int u, v;
        cin >> u >> v;
        u --, v --;
        pre.push_back(pre.back());
        swap(pre.back()[u], pre.back()[v]);
    }
    while(q--)
    {
        int l, r;
        cin >> l >> r;
        vector<int> a(n);
        for(int i = 0; i < n; i ++) cin >> a[i];
        vector<int> pos(n);
        for(int i = 0; i < n; i ++) pos[pre[l-1][i]] = i;
        for(int i = 0; i < n; i ++) cout << a[pos[pre[r][i]]] << ' ';
        cout << '\n';
    }
    return 0;
}
```

L3-1 图上极差

> 本题为了防止骗分，所以改成了图联通，这意味着一定有解xD

> update(25/04/06) 本题的测试数据已经加强，现在 $O(n^2 \log_k)$ 的解决方法会超出时限，请使用下方的 $O(n^2)$ 的做法。

如果你之前做过周赛的T4，你可能会得到本题的启发。

考虑相似的做法，枚举最大边权。因为我们求的是极差，因此本题转化成，如何求解在当前状态下**最大化经过的最小边权**。

此时，你应该意识到，经过的最小边权一定具有**单调性**！假设对于某个阈值 p，断开所有边权小于 p 的边后，图不再联通，此时 p 就是我们能达到的最大的最小边权。那么你一定已经意识到了，我们可以对最小边权进行二分答案，每次用bfs验证1到n是否联通即可。每次求出答案之后，取所有答案的最小值即可。

这样，每次二分check复杂度为 $O(n)$，二分本身复杂度 $O(\log K)$，枚举所有边复杂度为 $O(m)$，因此总复杂度为 $O(n^2 \log K)$。

一些对于过程的优化：

- 可以考虑用DSU维护S到T的连通性，只有联通才进行二分答案。
- 在bfs进行check时，搜到答案后立马返回。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
using pii = pair<int, int>;

struct DSU {
    std::vector<int> f, siz;

    DSU() {}
    DSU(int n) {
        init(n);
    }

    void init(int n) {
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
    }

    int find(int x) {
        while (x != f[x]) {
            x = f[x] = f[f[x]];
        }
        return x;
    }

    bool same(int x, int y) {
        return find(x) == find(y);
    }

    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int size(int x) {
        return siz[find(x)];
    }
};

signed main()
{
    IOS;
    int n, m, s, t;
    cin >> n >> m >> s >> t;
    vector<array<int, 3>> edges;
    while(m --)
    {
        int u, v , w;
        cin >> u >> v >> w;
        edges.push_back({u, v, w});
    }
    // 将边按照升序排序
    sort(edges.begin(), edges.end(), [&](auto p1, auto p2){return p1[2] < p2[2];});
    vector<vector<pii>> edge(n + 1);
    DSU dsu(n + 1);
    auto bfs = [&](int x) // check连通性
    {
        queue<int> q;
        q.push(s);
        vector<int> vis(n + 1);
        vis[s] = 1;
```

```
        while (q.size())
        {
            auto u = q.front();
            q.pop();
            for(auto [v, w] : edge[u])
            {
                if(vis[v]) continue;
                if(w >= x)
                {
                    // 搜到答案立马返回
                    if(v == t) return 1;
                    q.push(v);
                    vis[v] = 1;
                }
            }
        }
        return 0;
    };
    int ans = 1e9;
    for(auto[u, v, w] : edges)
    {
        edge[u].push_back({v, w});
        edge[v].push_back({u, w});
        dsu.merge(u, v);
        // 所有s和t联通才搜答案
        if(dsu.same(s, t))
        {
            int l = 0, r = w;
            while(l < r) // 二分最小边权
            {
                int mid = l + r + 1 >> 1;
                if(bfs(mid)) l = mid;
                else r = mid - 1;
            }
            ans = min(ans, w - r); // 更新答案
        }
    }
    cout << ans << '\n';
    return 0;
}
```

Bonus

当然，本题还有复杂度更优的做法，可以做到完全 $O(n^2)$，由验题人CangshuV提供：

利用Kruskal算法的原理，同样枚举最大边，然后再次枚举最小边，每次利用Kruskal算法验证最小边是否能够联通1和n，可以做到完全$O(n^2)$ 的复杂度。

当然，本题并没有卡掉$O(n^2 \log K)$的做法，因此不必担心。

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

const int N = 1e4 + 5;
vector<pair<int, int>> g[N];
void add(int f, int t, int w) {
    g[f].push_back({t, w});
    g[t].push_back({f, w});
}

template <typename T = size_t> struct __DSU {
    vector<T> pa;
    explicit __DSU(T size) : pa(size + 1) {}

    void init(T size) { iota(pa.begin(), pa.end(), 0); }

    T find(T x) { return pa[x] == x ? x : pa[x] = find(pa[x]); }

    void merge(T x, T y) { pa[find(x)] = find(y); }

    int query(T x, T y) { return (find(x) == find(y)); }
};

int n, m, s, t;
// <w, f, t>
vector<tuple<int, int, int>> war;
__DSU<> dsu(N);
int kruskal(int id) {
    dsu.init(n);
    for (; id < m; ++id) {
        dsu.merge(get<1>(war[id]), get<2>(war[id]));
        if (dsu.query(s, t))
            return id;
    }
    return -1;
}

int main() {
    cin.tie(nullptr)->sync_with_stdio(false);
    cin >> n >> m >> s >> t;
    war.resize(m);
    for (int i = 0; i < m; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        add(u, v, w);
        war[i] = {w, u, v};
    }

    sort(war.begin(), war.end());

    int ans = 2e9 + 1;
    for (int i = 0; i < m; ++i) {
        int id = kruskal(i);
        if (id != -1)
            ans = min(ans, get<0>(war[id]) - get<0>(war[i]));
    }

    cout << ((ans == 2e9 + 1) ? -1 : ans) << '\n';
    return 0;
}
```

L3-2 哨兵划分

定义一个排列中的哨兵 P_i，满足 $\forall_{j<i} P_j < P_i$ and $\forall_{j>i} P_j > P_i$。

假设 P_i 是哨兵，我们不难发现一些性质：

- P_i 的位置一定是其自然排列所在的对应位置，例如，假设 4 是哨兵，那么的位置一定也是 4 。因为这样才能满足其前面的数一定小于它，且后面的数一定大于他。
- 假设 P_i 是哨兵，要求其左边的数 $[1,...,P_i-1]$ 的排列中不存在任何哨兵，右边 $[P_{i+1},...,n]$ 也是同理。那么实际上，不难发现，数的大小已经没有意义了。

考虑动态规划：$dp(i)$ 表示的是长度为 i 的所有排列中，不含任何哨兵的排列的数量。那么对于我们的答案有： $f(i)=dp(i-1)\times dp(n-i)$

那么 $dp(i)$ 如何状态转移呢？很简单。考虑 n 的排列总数为 $n!$，其减去存在至少一个哨兵的排列数就是我们要求的 $dp(i)$：

$$dp(i)=i!-\sum_{j=1}^{i}(j-1)!\cdot dp(i-j)$$

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
using i64 = long long;
constexpr int mod = 1e9 + 7;

signed main()
{
    IOS;
    int n;
    cin >> n;
    vector<i64> fact(n + 1);
    fact[0] = 1;
    // 预处理阶乘
    for(int i = 1; i <= n; i ++) fact[i] = fact[i - 1] * i % mod;
    vector<i64> f(n + 1);
    f[0] = 1;
    for(int i = 1; i <= n; i ++)
    {
        i64 sum = 0;
        for(int j = 1; j <= i; j ++)
        {
            sum += fact[j - 1] * f[i - j] % mod;
            sum %= mod;
        }
        f[i] = (fact[i] - sum + mod) % mod;
    }
    // 计算答案
    for(int i = 1; i <= n; i ++) cout << f[i - 1] * f[n - i] % mod << ' ';
    return 0;
}
```

L3-3 分类网络(HardVersion)

本题的主要难点在于NormalVersion的操作可叠加性证明和可逆证明还有矩阵乘法优化。

在之前的证明下，我们观察本题的本质：

- 实际上本题的单次映射实际上就是一组映射的叠加
- 映射之间任然可以叠加
- 修改某一组映射为新的映射，实际上相当于单点修改

综上所述，我们可以考虑用线段树为维护每次修改，每次查询则取出对应的区间进行合并，合并操作本质就是映射的叠加。

朴素做法下，线段树维护矩阵乘法的合并是 $O(n^3)$ 的，但是基于我们之前证明的结论，可以利用映射的性质将其优化到 $O(n)$ 即可通过本题。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
using i64 = long long;

struct Info
{
    int n;
    vector<int> c;
    Info(i64 n = 0) : n(n) { }
    Info(vector<int> a, i64 n = 0) : n(n), c(a) { }
    friend Info operator+(Info a, Info b)
    {
        Info c;
        c.n = a.n;
        c.c = vector<int>(c.n);
        vector<int> t1(a.n), t2(a.n);
        for(int i = 0; i < a.n; i ++) t1[b.c[i]] = i;
        for(int i = 0; i < a.n; i ++) t2[t1[i]] = a.c[i];
        for(int i = 0; i < a.n; i ++) c.c[i] = t2[i];
        return c;
    }
};

struct segmentTree
{
    #define Lson(u) (u << 1)
    #define Rson(u) (u << 1 | 1)
    int N; // N = n + 1
    vector<int> L, R;
    vector<Info> ar;
    vector<Info> node;
    segmentTree(int n) : N(n + 1), L(4 * N), R(4 * N), node(4 * N) // range 1~n
    {
        ar = vector<Info>(N);
        build(1, 1, n);
        ar.clear();
    }
    segmentTree(vector<Info> &a) : ar(a), N(a.size()), L(4 * N), R(4 * N), node(4 * N) // a.size = n + 1, range 1~n
    {
        build(1, 1, a.size() - 1);
        ar.clear();
    }
    void build(int u,int l,int r)
    {
        L[u] = l, R[u] = r;
        if(l == r)
        {
            node[u] = ar[r];
            return;
        }
        int mid = l + r >> 1;
        build(Lson(u), l, mid);
        build(Rson(u), mid + 1, r);
        node[u] = node[Lson(u)] + node[Rson(u)];
    }
    Info query(int u, int l, int r)
    {
        assert(l <= r);
        assert(l >= 1);
        assert(r < N);
        if(l <= L[u] and R[u] <= r)
            return node[u];
        int mid = L[u] + R[u] >> 1;
        if(r <= mid) return query(Lson(u), l, r);
        else if(l > mid) return query(Rson(u), l, r);
        else return query(Lson(u), l, r) + query(Rson(u), l, r);
    }
```

```
    Info query(int l, int r) {return query(1, l, r);}
    void modify(int u,int pos, Info v)
    {
        assert(pos >= 1);
        assert(pos < N);
        if(L[u] == R[u] and L[u] == pos)
        {
            node[u] = v;
            return;
        }
        int mid = L[u] + R[u] >> 1;
        if(pos <= mid) modify(Lson(u), pos, v);
        else modify(Rson(u), pos, v);
        node[u] = node[Lson(u)] + node[Rson(u)];
    }
    void modify(int pos, Info v) { modify(1, pos, v); }
    template<class F>
    int findfirst(int u,int l,int r,F &&check)
    {
        if(l > R[u] or r < L[u]) return -1;
        if(!check(node[u])) return -1;
        if(L[u] == R[u]) return L[u];
        int mid = L[u] + R[u] >> 1;
        int res = findfirst(Lson(u), l, r, check);
        if(~res) return res;
        return findfirst(Rson(u), l, r, check);
    }
    template<class F>
    int findfirst(int l,int r,F &&check) {return findfirst(1, l, r, check);}
    template<class F>
    int findlast(int u,int l,int r,F &&check)
    {
        if(l > R[u] or r < L[u]) return -1;
        if(!check(node[u])) return -1;
        if(L[u] == R[u]) return L[u];
        int mid = L[u] + R[u] >> 1;
        int res = findfirst(Rson(u), l, r, check);
        if(~res) return res;
        return findfirst(Lson(u), l, r, check);
    }
    template<class F>
    int findlast(int l,int r,F &&check) {return findlast(1, l, r, check);}
    #undef Lson(u)
    #undef Rson(u)
};

signed main()
{
    IOS;
    int n, m, q;
    cin >> n >> m >> q;
    vector<Info> a(m + 1);
    for(int i = 1; i <= m; i ++)
    {
        vector<int> c(n);
        for(int j = 0; j < n; j ++) cin >> c[j], c[j] -= 1;
        a[i] = Info(c, n);
    }
    segmentTree sgtr(a);
    while(q--)
    {
        int op;
        cin >> op;
        if(op == 1)
        {
            int l, r;
            cin >> l >> r;
            vector<int> a(n);
            for(int i = 0; i < n; i ++) cin >> a[i];
```

```
            auto res = sgtr.query(l, r);
            for(int i = 0; i < n; i ++) cout << a[res.c[i]] << ' ';
            cout << '\n';
        }
        else
        {
            int p;
            cin >> p;
            vector<int> c(n);
            for(int j = 0; j < n; j ++) cin >> c[j], c[j] -= 1;
            sgtr.modify(p, Info(c, n));
        }
    }
    return 0;
}
```