> 有不懂的可以下方留言。

本次的4题均选自GPLT选拔赛的题目，一些本来打算出在GPLT选拔赛中的题由于难度问题或者是原题/改编题，因此经过最终考虑没有放入选拔赛，而是选择采用原创题。

4题难度界定分别为L1-L2-L2-L3，其中：

- L1(基础级): 基础语法题(简单小模拟，计算)
- L2(进阶级): 为基础算法题(前缀和/差分，简单搜索dfs/bfs)
- L3(登顶级): 为提高算法题(图论/动态规划/高级数据结构等)

具体难度界定如下：

题号	难度	知识点	期望难度
A. Orange讨厌洗衣服	L1	模拟	1100
B. 幻方	L2	搜索，暴力枚举	1400
C. Orange的失效线段树	L2	差分，前缀和，快速幂	1600
D. 图上极值和	L3	图论：最小生成树，数据结构：并查集	2100

> ps：真正选拔赛的难度不会高于这些题目，请不要有压力~热身赛的目的只是让各位熟悉oj平台，并且对各个等级的题目有大致的了解(主要是考察内容)。

A. Orange讨厌洗衣服

直接按照日程安排进行模拟即可，维护普通外套总数，印花外套总数，干净的普通外套数和干净的印花外套数。如果是去上课，则先穿普通外套，不够再考虑印花外套，如果去比赛，则只能考虑印花外套，如果不够就购买。遇到放假则重置所有的干净的普通外套数和干净的印花外套数。

一种更加聪明的做法是，用放假日来分割整个字符串，对每一段求出需要购买的衣服数量，取最大值。

```
void solve()
{
    int n, m, logo;
    string s;
    cin >> n >> m >> s;
    for (int i = 0, p = 0; i <= s.size(); i++)
    {
        if (s[i] == '0' || i == s.size())
        {
            string t = s.substr(p, i - p);
            int all = 0, lo = 0;
            for (char c : t)
                if (c == '2')
                    lo++;
            all++;
            if (lo > logo)
                logo = lo;
            if (all > m + logo)
                logo = all - m;
            p = i + 1;
        }
    }
    cout << logo << '\n';
}
```

B. 幻方

注意到数据范围很小，考虑暴力枚举，我们发现实际上对行或者列，所有的情况数量就是他们的排列数量。枚举所有行的排列和列的排列，依次验证即可。同时，因为我们每次交换的是相邻的两行或者两列，而起始状态是自然排列，那么我们到达目标状态的行排列需要的次数就是目标状态行排列的逆序对数量，列同理，对所有的合法答案取min即可。

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

int n, m, ans = 1e9;
int a[6][6], b[6][6];
int p[6] = {0, 1, 2, 3, 4, 5}, q[6] = {0, 1, 2, 3, 4, 5};

int check(int x[6][6], int y[6][6])
{
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            if (x[p[i]][q[j]] != y[i][j])
                return 0;
    return 1;
}
int main()
{
    cin.tie(0)->sync_with_stdio(0);
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> a[i][j];
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> b[i][j];
    do
    {
        do
        {
            if (not check(a, b)) // 验证
                continue;
            int num = 0;
            for (int i = 1; i <= n; i++)
                for (int j = 1; j <= n; j++)
                    num += (i < j and p[i] > p[j]); // 求逆序对的数量
            for (int i = 1; i <= m; i++)
                for (int j = 1; j <= m; j++)
                    num += (i < j and q[i] > q[j]);
            ans = min(ans, num);
        } while (next_permutation(q + 1, q + m + 1));
    } while (next_permutation(p + 1, p + n + 1));
    if (ans == 1e9)
        ans = -1;
    cout << ans << '\n';
    return 0;
}
```

C. Orange的失效线段树

> ps：本题线段树会被卡常，只有常数非常小的线段树可以通过本题

注意到存在若干次区间加法，但是最后的询问只有一次，可以采用**差分**来完成。

那么考虑计算最后全局最大值的期望，我们可以考虑枚举每一次区间加法，检查其是否会影响全局最大值。我们维护两个数组 Pre_i 和 Suf_i，分别表示前 i 项的最大值和后 i 项的最大值。

我们钦定用一个三元组 (l, r, v) 表示一次区间加法，对于一次修改，若 Pre_{l-1} 或 Suf_{r+1} 等于 Pre_n (全局最大值)则表示全局最大值在修改区间外任然存在，撤销这次区间修改并不会影响全局最大值，则可以直接将全局最大值累加入答案。假设上述条件不成立，则说明全局最大值位于区间内，撤销区间修改可能会产生新的最大值，新的最大值可能来自于 $Pre_{l-1}, Suf_{r+1}, Pre_n - v$ 这三项之一。他们的max就是最大值，累加入答案即可。

依次对所有区间加法进行上述计算，得到的最终答案，乘以修改次数m^{-1}(即逆元)即可得到最终答案。

```cpp
#include <bits/stdc++.h>
using namespace std;
using i64 = long long;
#define int long long

const int mod = 998244353;

i64 quickpow(i64 a, i64 b, i64 p) {
    i64 ans = 1;
    a %= p;
    while(b) {
        if(b & 1) ans = (ans * a) % p;
        b >>= 1;
        a = (a * a) % p;
    }
    return ans % p;
}

void solve() {
    int n, q;
    cin >> n >> q;
    vector<int> a(n + 1);
    for(int i = 1; i <= n; i++)
        cin >> a[i];
    vector<int> b(n + 2); // 差分数组
    vector<int> L(q + 1), R(q + 1), D(q + 1);
    for(int i = 1; i <= q; i++) {
        cin >> L[i] >> R[i] >> D[i];
        b[L[i]] += D[i];
        b[R[i] + 1] -= D[i];
    }
    vector<int> pre(n + 1);
    for(int i = 1; i <= n; i++) {
        b[i] += b[i - 1];
        a[i] += b[i];
        pre[i] = max(pre[i - 1], a[i]); // 求前缀max
    }
    vector<int> suf(n + 2);
    for(int i = n; i > 0; i--)
        suf[i] = max(suf[i + 1], a[i]); // 求后缀max

    int ans = 0;
    for(int i = 1; i <= q; i++) {
        int l = L[i], r = R[i], d = D[i];
        if(pre[l - 1] == pre[n] or suf[r + 1] == suf[1])  // 如果最大值在区间外
            ans += pre[n];
        else
            ans += max(pre[n] - d, max(pre[l - 1], suf[r + 1]));
        ans %= mod;
    }
    ans *= quickpow(q, mod - 2, mod);
    ans %= mod;
    cout << ans << '\n';
}

signed main()
{
    cin.tie(0) -> sync_with_stdio(0);
    int _ = 1;
    cin >> _;

    while (_--) solve();

    return 0;
}
```

D. 图上极值和

本题实际上就是一个Kruskal算法的模板变形，考察对算法原理的掌握。

由于我们要让极大+极小边权最小。我们可以考虑枚举最大的边。

将所有边预存并排序之后，依次从小到大枚举每条边。每次枚举到一条边，我们就将其所连接两点的连通块合并(考虑DSU)，当前枚举到的边就是我们图中存在的极大边权。那么如何维护最小？在每次合并连通块的时候，我们可以在DSU内维护当前连通块内所有边的最小值，即每次用边merge的时候取min。

每次枚举边，在合并连通块后，检查1和n是否联通，若联通，则当前的答案则为枚举到的极大边+其所在连通块内的极小边权。最终答案就是所有的答案的min。

```
#define int long long
void solve() {
    int n, m;
    cin >> n >> m;
    vector<int> f(n + 1), MinEdgeW(n + 1, 1e18);
    iota(f.begin(), f.end(), 0LL);

    auto find = [&](int x) -> int {
        while (x != f[x]) x = f[x] = f[f[x]];
        return x;
    };

    auto merge = [&] (int u, int v, int w) -> bool {
        u = find(u);
        v = find(v);
        f[v] = u;
        MinEdgeW[u] = min(MinEdgeW[u], MinEdgeW[v]);
        MinEdgeW[u] = min(MinEdgeW[u], w);
        return 1;
    };

    vector<array<int, 3>> edge;
    for(int i = 0, u, v, w; i < m; i++) {
        cin >> u >> v >> w;
        edge.push_back({w, u, v});
    }
    sort(edge.begin(), edge.end());

    int ans = 1e18;
    for(auto & [w, u, v] : edge) {
        merge(u, v, w);
        if(find(1) == find(n)) {
            ans = min(ans, MinEdgeW[find(1)] + w);
        }
    }

    if(ans > 1e17) ans = -1;
    cout << ans << '\n';
}
```