2023 年牛客多校第四场题解

出题人：南京大学

1 A Bobo String Construction

题意：给定一个 01 字符串 t，构造一个长度为 n 的 01 串 s，使得 t 在 $\text{concat}(t, s, t)$ 中仅出现两次。多测，$1 \le T \le 10^3$，$1 \le n, |t| \le 10^3$。

解法：结论是全 0 或全 1 串一定可行。

首先如果 t 就是全 0 或全 1，那显然构造全 1 或全 0 串一定可行。

如果 t 是 01 混杂，考虑以下两种情况：

1. 首先 s 串内部肯定不会出现 t。
2. 考虑 $\text{concat}(t, s)$ 和 $\text{concat}(s, t)$ 部分。显然只需要考虑 t 的 border（最长公共前后缀）和 s 的拼接部分即可。如果 border 部分 01 混杂那显然交叠部分不会出现。如果 border 只有 0，那就构造全 1，反之亦然。则 s 的交叠部分末端（可能出现匹配的首段是 t 的尾端 border）一定无法出现 t 的 border，也就不会出现匹配。

所以枚举到底是全 0 还是全 1，然后使用 KMP 算法计算 $\text{concat}(t + s + t)$ 中 t 是否只出现两次即可。复杂度 $\mathcal{O}(T(n + |t|))$。

```cpp
#include <bits/stdc++.h>
using namespace std;
// KMP template
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) // a中出现多少次b
```

```
    {
        int n = b.length(), m = a.length();
        int j = 0;
        int 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)
            {
                ans++;
                j = nx[j - 1];
            }
        }
        return ans;
    }
};
void Solve()
{
    int n;
    string t;
    cin >> n >> t;
    string s0, s1;
    for (int i = 0; i < n; i++)
    {
        s0 += "0";
        s1 += "1";
    }
    KMP solve(t);
    if (solve.find(t + s0 + t) == 2)
        cout << s0 << "\n";
    else if (solve.find(t + s1 + t) == 2)
        cout << s1 << "\n";
    else
        cout << "-1\n";
}
int main()
{
    cin.tie(0)->sync_with_stdio(0);
    cin.exceptions(cin.failbit);
    cin.tie(NULL);
    cout.tie(NULL);
    int t;
    cin >> t;
    while (t--)
```

```
        Solve();
    return 0;
}
```

2 F Election of the King

题意：给定长度为 n 的数列 $\{a\}_{i=1}^{n}$，最开始每个数字都存在。持续进行 $n-1$ 轮下述操作：

1. 当前剩下来的每个数，选择距离它最远（绝对值最大）的数进行投票，如果最远距离相等选择大的。
2. 当前被投票数最多的数在本轮删掉，平票则选择最大的数字删掉。

问最后是哪个数字留下来。$1 \leq n \leq 10^6$，$1 \leq a_i \leq 10^9$。

解法：考虑维护数列的中位数，并观察它的投票情况。因为如果中位数投最大，它左侧也一定投最大；中位数投最小，它右侧也一定投最小。因而哪怕是两侧投票数势均力敌也是由中位数定胜负。因而时刻维持中位数投票情况以决定淘汰的数字是谁，同时同步移动中位数即可。整体复杂度为 $\mathcal{O}(n \log n + n)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1000000;
pair<int, int> a[N + 5];
int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
    {
        scanf("%d", &a[i].first);
        a[i].second = i;
    }
    sort(a + 1, a + n + 1);
    int l = 1, r = n;
    // [l, r] 区间表示存活的数字
    for (int i = n, j = (n + 1) / 2; i >= 2; i--)
    {
        int midl = (a[r].first - a[j].first >= a[j].first - a[l].first);
        if (i % 2 == 0) // 偶数要考虑中位数相邻两个
        {
            int midr = (a[r].first - a[j + 1].first >= a[j + 1].first - a[l].first);
            if (midl || midr) // 票死大的
                r--;
            else
            {
                l++;
                j++;
            }
```

```c
        }
        else
        {
            if (!midl) // 票死小的
                l++;
            else
            {
                r--;
                j--;
            }
        }
    }
    printf("%d", a[l].second);
    return 0;
}
```

3 G Famished Felbat

题意：给定长度为 n 的数列 $\{a\}_{i=1}^{n}$，和长度为 m 的数列 $\{b\}_{i=1}^{m}$。第 i 轮从 $\{b\}$ 数列中任意选择一个数字 b_j，然后执行 $a_i \leftarrow a_i + b_j$，然后将 b_j 从 $\{b\}$ 数列中删去。n 轮操作后求 $\sum\limits_{i=1}^{n} f(a_i)$ 的期望，其中：

$$f(x) = \frac{1}{L}\sum_{i=1}^{L}\left\lceil \frac{x}{i} \right\rceil$$

L 为一已知常数。$1 \leq n \leq m \leq 10^3$，$1 \leq L, a_i, b_j \leq 2 \times 10^9$。

解法：首先由期望的线性性，每个 b_i 都会等概率加到每个 a_j 上。同时由 $\left\lceil \frac{x}{i} \right\rceil = \left\lfloor \frac{x+i-1}{i} \right\rfloor = \left\lfloor \frac{x-1}{i} \right\rfloor + 1$，将上取整转化到常用的下取整。因而本质是求：

$$n + \frac{1}{mL}\sum_{k=1}^{L}\sum_{i=1}^{n}\sum_{j=1}^{m}\left\lfloor \frac{a_i+b_j-1}{k} \right\rfloor$$

仅考虑求和部分式子的计算，下面所有的枚举都是建立在 L 充分大的情况，严格的式子都需要对 L 取 min。首先最朴素的想法是进行整除分块，对每个 a_i+b_j 进行整除分块，但这样的时间复杂度为 $\mathcal{O}\left(nm\sqrt{L}\right)$，显然不能通过。这时可以注意到一个性质：

$$\left\lfloor \frac{x+y}{i} \right\rfloor = \left\lfloor \frac{x}{i} \right\rfloor + \left\lfloor \frac{y}{i} \right\rfloor + [x \bmod i + y \bmod i \geq i]$$

即，两个被加数本身整除的部分，再检查余数之和是否能够再凑一个 i。那么对于 k 比较小的情况，显然就可以枚举 k，然后先单独计算完 $m\sum\limits_{k=1}^{B}\left\lfloor \frac{a_i}{k} \right\rfloor$ 和 $n\sum\limits_{k=1}^{B}\left\lfloor \frac{b_j}{k} \right\rfloor$，以及各自的余数，再通过枚举每个 b_j 的余数，检查 $\{a\}$ 中余数大于等于 $k - b_j \bmod k$ 的个数有多少即可。这样这部分复杂度就是 $\mathcal{O}\left(B(n+m) + Bm\log n\right)$。

考虑如果当 k 很大会怎么样。这时由于枚举的量太大，显然无法承受。但是结合整除分块的性质——在根号以下，自变量变化小，但值域变化大；在根号以上，自变量变化大，但值域变化小。因而对于 k 大的情况，不难考虑通过枚举整除得到的值有多少自变量区间对应来求解。因而有 k 较大（$k \geq B$，B 为阈值）部分的转化求和式：

$$\sum_{i=1}^{\left\lfloor \frac{L}{B} \right\rfloor} \max\left(0, \min\left(\left\lfloor \frac{x}{i} \right\rfloor, L\right) - B\right)$$

其中 i 是枚举的整除值，$-B$ 操作表示挖去自变量较小的部分的贡献，上限对 L 取 min 以限制分母的范围。

考虑把 min 函数去掉以快速计算，那么问题转化为 $a_i + b_j \geq iL$ 的二维偏序问题，这个可以用树状数组快速解决。这部分复杂度为 $O\left(\left\lfloor \frac{L}{B} \right\rfloor m \log n\right)$

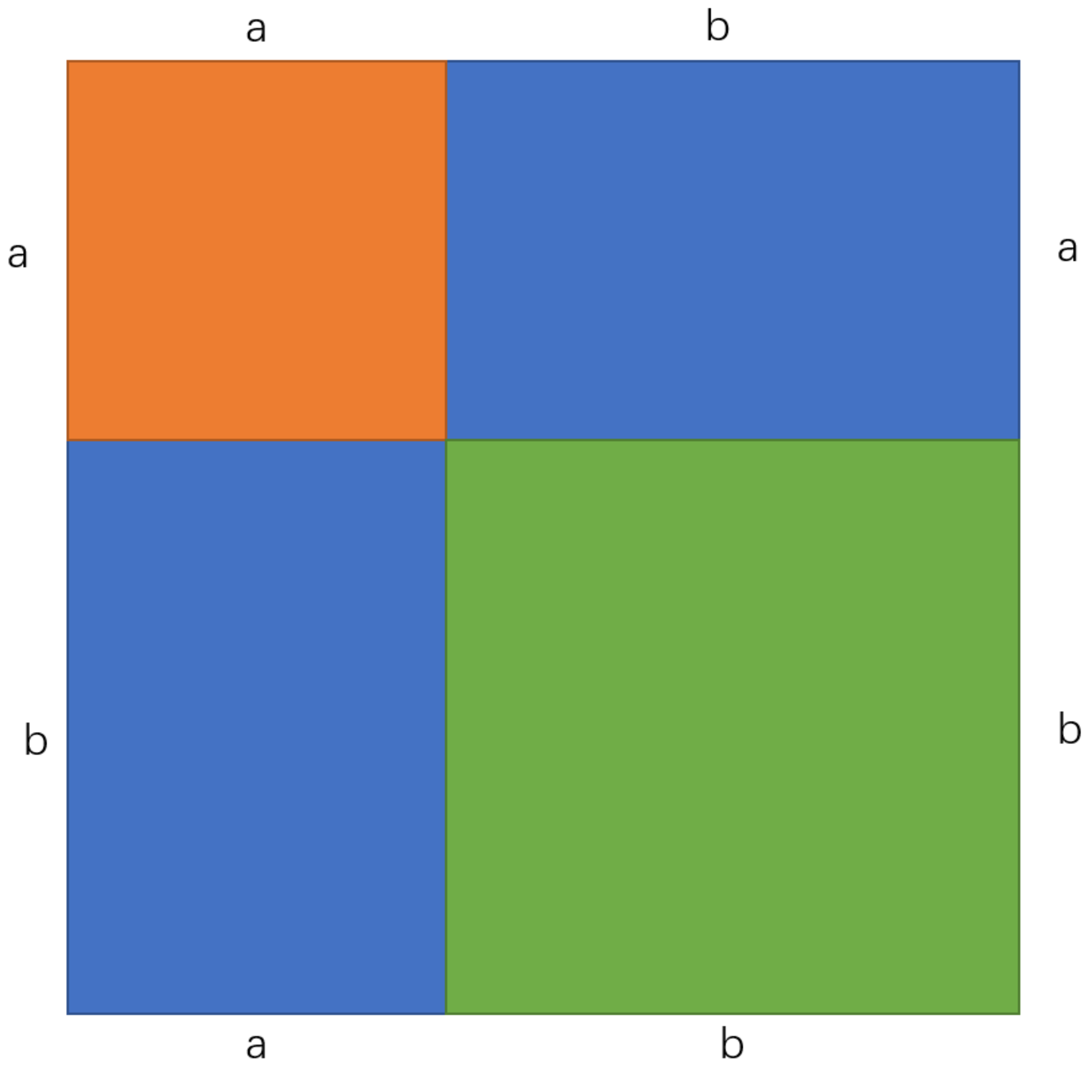
考虑取一个合适的阈值以平衡二者，因而取 $B = \sqrt{L}$，最终复杂度为 $O\left(\sqrt{L} m \log n\right)$。

4 H Merge the squares!

题意：给定 $n \times n$ 个 1×1 组成的正方形，每次可以合并相邻不超过 50 个正方形变成一个大正方形。问如何通过合并得到一个大的 $n \times n$ 的大正方形，不限次数。$1 \leq n \leq 10^3$。

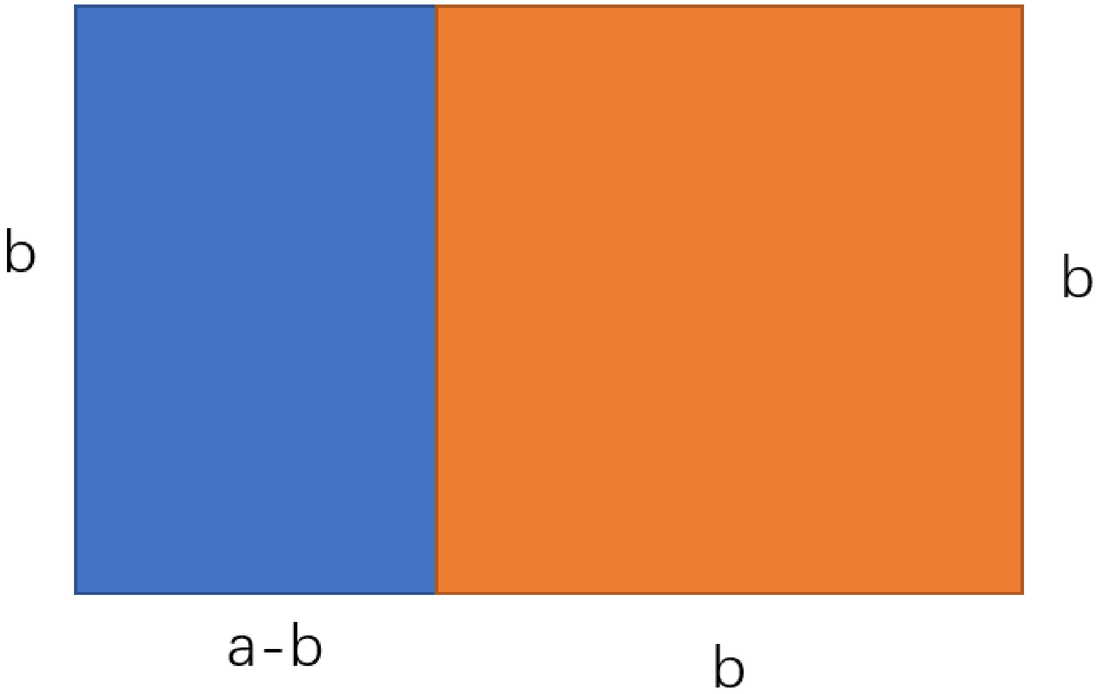
解法：考虑 $7^2 \leq 50$，所以如果边长 x 是 $[2, 7]$ 的倍数，可以考虑直接先拆分成 $d \times d$ 个 $\frac{x}{d} \times \frac{x}{d}$ 个正方形求解。

最棘手的问题在于大质数。显然质数不能按照这种乘除法的倍数拆分，因而考虑加减法。注意到完全平方和公式：$(a+b)^2 = a^2 + 2ab + b^2$，构造下面的图形：



即 $a \times a$ 和 $b \times b$ 的正方形，和两个 $a \times b$ 的矩形。正方形可以递归下去构造，考虑矩形如何尽可能少的构造。

不妨令 $a > b$，一个贪心的想法是，每次构造一个 $b \times b$ 的正方形，然后留下一个 $(a-b, b)$ 的矩形递归下去构造，即类似辗转相减法：



每次我们都找了一个最大的正方形，这样做整体个数不会太劣。考虑它会拆分到多少个正方形：$\left\lfloor \frac{a}{b} \right\rfloor$ 个 $b \times b$ 的正方形（横向放置），然后再加上 $(a \bmod b, b)$ 的答案。因而可以用欧几里得算法求得它的答案：

```
int gcd(int x, int y)
{
    if (x == y)
        return 1; // 正方形
    if (x < y)
        swap(x, y);
    return x / y + gcd(x % y, y); // 先横向拆分，再递归到子矩形中
}
```

因而回到整体大正方形拆分，可以考虑枚举这样的 a，求出这样拆分的矩形 $(a, x-a)$ 需要包含多少个小正方形，如果不超过 24 个就可以视为一个合法的拆分。这是因为 $24 \times 2 + 2 = 50$，$a \times a$ 和 $b \times b$ 的正方形视为一个，剩下的 48 个均分给两个矩形构造。

```
#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 = 1e3 + 5;
```

```
int n, f[N];
vector<array<int, 3>> ans;
int check(int a, int b) {
    if (!b) return a <= 7;
    int cnt = 1, c;
    while (b) {
        cnt += a / b;
        c = a % b, a = b, b = c;
    }
    return cnt <= 25;
}
void dfs(int, int, int);
void calcC(int, int, int, int);
void calcR(int x, int y, int r, int c) { // c = a * r + b
    if (r <= 1) return;
    int a = c / r;
    fp(i, 0, a - 1) dfs(x, y + i * r, r);
    calcC(x, y + a * r, r, c % r);
}
void calcC(int x, int y, int r, int c) { // r = a * c + b
    if (c <= 1) return;
    int a = r / c;
    fp(i, 0, a - 1) dfs(x + i * c, y, c);
    calcR(x + a * c, y, r % c, c);
}
void dfs(int x, int y, int k) {
    if (k == 1) return;
    ans.push_back({x, y, k});
    // printf("%d %d %d\n", x, y, k);
    if (!f[k]) return;
    int a = k - f[k], b = f[k];
    calcR(x + a, y, b, a), calcC(x, y + a, a, b);
    dfs(x, y, a), dfs(x + a, y + a, b);
}
void Solve() {
    scanf("%d", &n);
    // freopen("s.out", "w", stdout);
    // printf("%d\n", n);
    memset(f, -1, sizeof f);
    f[1] = 0;
    fp(i, 2, n) {
        fp(j, 0, i / 2) {
            if (check(i - j, j)) {
                f[i] = j;
                break;
            }
        }
```

```
    }
    dfs(1, 1, n);
    printf("%llu\n", ans.size());
    reverse(ans.begin(), ans.end());
    for (auto [x, y, k] : ans) printf("%d %d %d\n", x, y, k);
}
int main() {
    int t = 1;
    while (t--) Solve();
    return 0;
}

```

5 I Portal 3

题意：n 个点的有向图，给定其邻接矩阵 G。现在沿着一条长度为 k 的路径 $\{v\}_{i=1}^{k}$（给定 k 个路径点依次到达），并可以合并两个点 u, v（$G_{u,v} = G_{v,u} = 0$），问合并后最短路径长。$1 \le n \le 500$，$0 \le G_{i,j} \le 10^9$，$1 \le k \le 10^6$。

解法：首先 Floyd 跑出任意两点之间的最短路 $\{d\}_{(i,j)=(1,1)}^{(n,n)}$。然后路径本身可以转化到统计经过两点 (s,t) 的次数 $c(s,t)$。考虑合并 u, v 两个点会发生什么：显然有些 (s,t) 会考虑绕道 (u,v) 以拉近最短路。由于不指定 u, v 顺序，因而可以认为一定是 $s \to u \to v \to t$。则绕道后会节省（贡献）$d(s,t) - d(s,u) - d(v,t)$。因而一个简易的暴力算法流程如下：

```
long long maxSaved = 0;
for (int u = 1; u <= n; u++)
    for (int v = 1; v <= n; v++)
    {
        long long curSaved = 0;
        for (int s = 1; s <= n; s++)
            for (int t = 1; t <= n; t++)
                curSaved += max(0ll, c[s][t] * (d[s][t] - d[s][u] - d[v][t]));
        maxSaved = max(maxSaved, curSaved);
    }
```

即固定枚举是合并哪两个点，然后考虑路径上每一对 (s,t) 对这一对 (u,v) 的贡献。但是这样计算复杂度是 $O(n^4)$。下面给出两种做法：

5.1 $O(n^3 \log n)$

考虑首先枚举 u, t，这时可以首先枚举所有的 s，固定 $d(s,t) - d(s,u)$。对 s 按 $d(s,t) - d(s,u)$ 项排序。当按排序后 s 的顺序枚举时，$d(s,t) - d(s,u)$ 项单增，这时如果 v 按 $d(v,t)$ 单增的顺序排列，就可以考虑双指针去快速找到每个 s 下贡献最大的 v 是什么。复杂度 $\mathcal{O}(n^3 \log n + n^3 + n^2 \log n + k)$。

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

using namespace std;
```

```
using ll = long long;
const int N = 505;
int n, k, d[N][N], c[N][N];
ll ans, len, w[N][N];
vector<pair<int, int>> val, nv[N];
void Solve() {
    scanf("%d%d", &n, &k);
    fp(i, 1, n) fp(j, 1, n) scanf("%d", d[i] + j);
    fp(k, 1, n) fp(i, 1, n) fp(j, 1, n)
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
    {
        int u, v;
        scanf("%d", &u);
        for (k--; k--;) scanf("%d", &v), ++c[u][v], len += d[u][v], u = v;
    }
    fp(t, 1, n) {
        fp(v, 1, n) nv[t].push_back({d[v][t], v});
        sort(nv[t].begin(), nv[t].end());
    }
    fp(u, 1, n) fp(t, 1, n) {
        val.clear();
        fp(s, 1, n) if (c[s][t] && d[s][t] > d[s][u])
            val.push_back({d[s][t] - d[s][u], c[s][t]});
        sort(val.begin(), val.end());
        ll tot = 0, cnt = 0, i = 0;
        for (auto [d, c] : val) tot += (ll)d * c, cnt += c;
        for (auto [d, v] : nv[t]) {
            while (i < val.size() && d >= val[i].first)
                tot -= (ll)val[i].first * val[i].second, cnt -= val[i].second, ++i;
            w[u][v] += tot - cnt * d;
        }
    }
    ans = len;
    fp(u, 1, n) fp(v, u, n) ans = min(ans, len - w[u][v] - w[v][u]);
    printf("%lld\n", ans);
}
int main() {
    int t = 1;
    while (t--) Solve();
    return 0;
}
```

5.2 $O(n^3)$

转变维护思路。考虑维护一个 v 数组，其中第 i 项表示当前要合并的点是 $(i,j), j \in [1,v]$ 时整个经过路径最大缩短量。因而这个时候可以考虑枚举每一对 (s,t)，考虑这一对 (s,t) 会对这个数组产生什么影响。下面固定 (s,t)。

观察 $d(s,t)-d(s,u)-d(v,t)$，这时第一项已经固定。再枚举 u，不难注意到 $d(s,t)-d(s,u)$ 都已经确定，这时满足 $d(v,t) \le d(s,t)-d(s,u)$ 都会更新。因而可以考虑将所有的 v 按 $d(v,t)$ 顺序排列，这时按 $d(s,u)$ 递增的顺序枚举 u 的时候，更新的 v 就是连续的一段，可以考虑差分和前缀和维护。等到 u 一轮更新完，再对 v 恢复顺序。这样复杂度为 $\mathcal{O}(n^3+2n^2\log n+k)$。

6 J Qu'est-ce Que C'est?

题意：给定长度为 n 的数列 $\{a\}_{i=1}^{n}$，要求每个数都在 $[-m,m]$ 范围，且任意长度大于等于 2 的区间和都大于等于 0，问方案数。$1 \le n,m \le 5\times 10^3$。

解法：下面给出两种 dp 状态设计。

6.1 法一

考虑 $f_{i,j}$ 表示填了 i 个数字，当前最小后缀和为 j 的方案数。显然 $j\in[-m,m]$。

维护转移：

1. 填入正数，此时 $j \ge 0$。$f_{i,j} \leftarrow \sum_{k=-j}^{m} f_{i-1,k}$，即填入一个数字使得这一位和上一位加起来得大于等于 0。
2. 填入一个负数。枚举填了什么数字 j，这时上一位必须满足最小后缀和得大于等于 $-j$，否则拼接起来会小于 0。因而 $f_{i,j} \leftarrow \sum_{k=-j}^{m} f_{i-1,k}$。

```
#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 = 5e3 + 5, P = 998244353;
int n, m, f[2][2 * N], suf[2 * N];
void Solve() {
    scanf("%d%d", &n, &m);
    int q = 0, ans = 0;
    fp(i, -m, m) f[0][N + i] = 1;
    fp(i, 2, n) {
        q ^= 1;
        fd(x, N + m, N -m) suf[x] = (suf[x + 1] + f[q ^ 1][x]) % P;
        fp(x, 0, m) f[q][N + x] = suf[N - m + x];
        fp(x, 1, m) f[q][N - x] = suf[N + x];
    }
    fp(i, -m, m) ans = (ans + f[q][N + i]) % P;
    printf("%d\n", ans);
```

```
}
int main() {
    int t = 1;
    while (t--) Solve();
    return 0;
}
```

整体复杂度 $\mathcal{O}(n^2)$。

6.2 法二

除了最后一个数字，其余的负数一定是可以和非负数绑定的。例如，考虑如下的正负数列可以被划分为：

负正/正/正/负正/负正/正/正/正/负正/

将负数和后面紧邻的正数绑定成为一个完整块，一起填充。考虑 $f_{i,j}$ 表示前 i 个数，填的一个完整块的和为 j 的方案数。考虑如下几种情况的转移：

- 当前填非负数。$f_{i,j} \leftarrow \sum_{k=0}^{m} f_{i-1,k}$
- 当前准备带负数的完整块。$f_{i,j} \leftarrow \sum_{k=0}^{m} \sum_{l=-k}^{-1} [1 \le j-l \le m] f_{i-2,k}$，即 l 枚举负数范围为 $[-k,-1]$，正数需要和满足 j 条件下，仍然在 $[0,m]$ 范围。因而有转移 $f_{i,j} \leftarrow \sum_{k=0}^{m} \min(k, j-m) f_{i-2,k}$。

基于这些转移，可以写出这样的暴力代码：

```
#include <bitsdc++.h>
using namespace std;
const int N = 5000, P = 998244353;
int f[N + 5][N + 5], g[N + 5][N + 5];
int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    f[0][0] = 1;
    for (int i = 0; i <= m; i++)
        f[1][i] = 1;
    for (int i = 0; i <= m; i++)
    {
        for (int j = 0; j <= m; j++)
            f[2][i] = (f[2][i] + f[1][j]) % P;
        for (int k = -m; k <= -1; k++)
        {
            int res = i - k;
            if (res <= m && res >= 0)
                f[2][i] = (f[2][i] + 1) % P;
        }
    }
    for (int i = 3; i <= n; i++)
```

```
    {
        for (int j = 0; j <= m; j++)
        {
            for (int k = 0; k <= m; k++)
                f[i][j] = (f[i][j] + f[i - 1][k]) % P;
            for (int k = 0; k <= m; k++)
                for (int l = -k; l <= -1; l++) // 枚举负数
                {
                    int res = j - l;
                    if (res <= m)
                        f[i][j] = (f[i][j] + f[i - 2][k]) % P;
                }
        }
    }
    int ans = 0;
    // 最后一个数字可以填负数，需要单独考虑
    for (int i = 0; i <= m; i++)
        ans = (ans + f[n][i] + (long long)f[n - 1][i] * i) % P;
    printf("%d", ans);
    return 0;
}
```

不难发现只需要维护 $f_{i,j}$ 的前缀和和 $jf_{i,j}$ 的前缀和即可快速计算。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 5000, P = 998244353;
int f[N + 5][N + 5], g[N + 5][N + 5];
// f表示直接的前缀和，g表示i*f的前缀和
int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    if (n == 1)
    {
        printf("%d", 2 * m + 1);
        return 0;
    }
    f[0][0] = 1;
    for (int i = 0; i <= m; i++)
    {
        f[1][i] = i + 1;
        if (i)
            g[1][i] = (g[1][i - 1] + i) % P;
    }
    for (int i = 0; i <= m; i++)
    {
```

```
        f[2][i] = 2 * m - i + 1;
        if (i)
        {
            g[2][i] = (g[2][i - 1] + (long long)f[2][i] * i) % P;
            f[2][i] = (f[2][i - 1] + f[2][i]) % P;
        }
    }
    for (int i = 3; i <= n; i++)
    {
        for (int j = 0; j <= m; j++)
            f[i][j] = (f[i - 1][m] + g[i - 2][m - j] + (long long)(m - j) * (f[i - 2][m] - f[i - 2][m - j] + P) % P) % P;
        for (int j = 1; j <= m; j++)
        {
            g[i][j] = (g[i][j - 1] + (long long)f[i][j] * j) % P;
            f[i][j] = (f[i][j - 1] + f[i][j]) % P;
        }
    }
    // 最后一位特判
    long long ans = (f[n][m] + g[n - 1][m]) % P;
    printf("%lld", ans);
    return 0;
}
```

整体复杂度 $\mathcal{O}(n^2)$。

7 L We are the Lights

题意：$n \times m$ 的灯阵，初始全灭。一次操作可以执行：第 i 行或列全灭或全亮。问执行完全部 q 条操作亮着的灯有多少。$1 \le n, m, q \le 10^6$。

解法：首先为了防止后面的操作对前面有影响，显然是倒序执行所有的操作。对于一次行操作，只需要维护列中在后续操作中确定会灭或亮的灯数（确定亮的灯在之前行操作中已经计数过了），列同理。因而使用四个变量维护行、列中确定亮、灭的个数即可。整体复杂度 $\mathcal{O}(q)$。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1000000;
int st[2][N + 5], cnt[2][2];
struct node
{
    int dir;
    int id;
    int op;
    node(int _dir, int _id, int _op) : dir(_dir), id(_id), op(_op) {}
};
```

```cpp
char s[50], t[50];
int main()
{
    memset(st, -1, sizeof(st));
    int n[2], q, x;
    long long ans = 0;
    scanf("%d%d%d", &n[0], &n[1], &q);
    vector<node> que;
    while (q--)
    {
        scanf("%s%d%s", s, &x, t);
        int dir = (s[0] == 'c'), op = (t[1] == 'n');
        que.emplace_back(dir, x, op);
    }
    reverse(que.begin(), que.end());
    for (auto [dir, x, op] : que)
    {
        if (st[dir][x] != -1)
            continue;
        if (op)
            ans += n[dir ^ 1] - cnt[dir ^ 1][0] - cnt[dir ^ 1][1];
        st[dir][x] = op;
        cnt[dir][op]++;
    }
    printf("%lld", ans);
    return 0;
}
```