题意解释

在 k 维空间里有 n 个障碍点。问从$(0,0,\cdots,0)$出发，到达$(s_1,s_2,\cdots,s_k)$，不经过任何障碍点的方案数。每一步只能选择一个维度$+1$。

知识点提炼

容斥，组合计数，动态规划

核心解题思路

思路1：$k=2,m\le 5000$

也即二维的情况。可以直接暴力DP。用$f_{i,j}$ 表示从起点出发，走到(i,j)，且不经过任何障碍点的方案数。

对于非障碍点，转移 $f[i][j]=f[i-1][j]+f[i][j-1]$，边界以外的DP值认为是 0。

对于障碍点，转移 $f[i][j]=0$。

复杂度 $O(n^2)$，期望得分 30 分

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

using namespace std;

const int P = 998244353;

int main()
{
    ios::sync_with_stdio(false);

    int k, n;
    cin >> k >> n;
    assert(k == 2);
    int X, Y;
    cin >> X >> Y;
    int M = max(X, Y);
    vector<vector<int>> ban(M + 1, vector<int>(M + 1, 0));
    vector<vector<int>> f(M + 1, vector<int>(M + 1, 0));

    for (int i = 0; i < n; ++i) {
        int x, y;
        cin >> x >> y;
        ban[x][y] = 1;
    }

    f[0][0] = 1;
    for (int i = 0; i <= M; ++i)
        for (int j = 0; j <= M; ++j)
            if (!ban[i][j]) {                         // 障碍点跳过，其余点转移
                if (i + 1 <= M) (f[i + 1][j] += f[i][j]) %= P;
                if (j + 1 <= M) (f[i][j + 1] += f[i][j]) %= P;
            }
```

```
    cout << f[X][Y] << endl;
    return 0;
}
```

思路2：$n = 0$

也即没有障碍点的情况。此时可以直接利用组合数计数。

二维情况下，一共走 $s_1 + s_2$ 步到达终点，其中有 s_1 步选择 $x \to x+1$，有 s_2 步选择 $y \to y+1$，所以方案数就是$s_1 + s_2$步中选择s_1步向右走的方案数，也即 $C(s_1 + s_2, s_1) = \frac{(s_1+s_2)!}{s_1!s_2!}$。

三维的情况是类似的，根据组合意义，答案是多重组合数。也可以理解成先从 $s_1 + s_2 + s_3$步中选择 s_1 步沿 x 走，再从剩下 $s_2 + s_3$ 步中选择 s_2 步沿 y 走

$$\binom{s_1+s_2+s_3}{s_1, s_2, s_3} = \binom{s_1+s_2+s_3}{s_1}\binom{s_2+s_3}{s_2}\binom{s_3}{s_3} = \frac{(s_1+s_2+s_3)!}{s_1!s_2!s_3!}$$

更高维的情况同理。

可以$O(n)$预处理所有阶乘和阶乘的逆元，进而$O(1)$计算答案。复杂度 $O(m)$，m是坐标的范围。 期望得分 40 分。

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <array>
const int N = 5010, M = 1000000, mod = 998244353;
int m, n;
long long fac[M], facinv[M], inv[M];
struct point {
    std::array<int, 10> x;
    long long get_c(const point &r) {
        for (int k = 0; k < m; k++)
            if (r.x[k] < x[k]) return 0ll;
        int sum = 0;
        for (int k = 0; k < m; k++)
            sum += r.x[k] - x[k];
        long long ans = fac[sum];
        for (int k = 0; k < m; k++)
            (ans *= facinv[r.x[k] - x[k]]) %= mod;
        return ans;                                    // 多重组合数
    }
}a[N];
inline long long sub(long long x, long long y) {
    return x - y > 0? x - y: x - y + mod;
}
int main() {
    scanf("%d%d", &m, &n);
    int maxx = 0;
    for (int k = 0; k < m; k++)
        scanf("%d", &a[n + 1].x[k]), maxx = std::max(maxx, a[n + 1].x[k]);
    for (int k = 1; k <= n; k++)
        for (int i = 0; i < m; i++)
```

```
            scanf("%d", &a[k].x[i]);
    maxx *= m;
    fac[0] = 1;
    for (int k = 1; k <= maxx; k++)
        fac[k] = (fac[k - 1] * k) % mod;
    inv[1] = 1;
    for (int k = 2; k <= maxx; k++)
        inv[k] = (mod - mod / k) * inv[mod % k] % mod;
    facinv[0] = 1;
    for (int k = 1; k <= maxx; k++)
        facinv[k] = facinv[k - 1] * inv[k] % mod;       // 预处理阶乘，以及阶乘的逆元
    printf("%lld\n", a[0].get_c(a[n + 1]));
}
```

思路3：容斥，动态规划

考虑有限制的情况，即有一些点不能经过，可以发现不能经过的点是不多的，可以作为突破口。我们设 $f[i]$ 表示中途不经过任何障碍点，最终到达第 i 个障碍点的合法方案数。我们把$(s_1, s_2, \cdots, s_k)$ 钦定为第 $n+1$ 个障碍点，那么答案就是 $f[n+1]$。

可以通过一个简单的容斥来计算 $f[i]$ 。答案总的方案数是从原点到第 i 个点的方案数（允许经过障碍点），然后要减去中途经过障碍点的方案数。不合法的方案可能会经过很多个障碍点，为了避免重复计算，我们可以枚举第一个经过的障碍点 j ，然后要减去的方案数就是 $f[j]$ 乘上 j 到 i 不考虑障碍点的方案数。

为了确保 dp 的顺序是正确的，即在计算点 i 时，所有可以到达 i 的点 j 都在之前已经计算过了，我们可以对所有点进行排序，优先按照第一维，然后第二维，依次类推。这样就可以确保转移的顺序时正确的。

时间复杂度 $O(n^2)$。

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <array>
const int N = 5010, M = 1000000, mod = 998244353;
int m, n;
long long f[N], fac[M], facinv[M], inv[M];
struct point {
    std::array<int, 10> x;

    bool operator < (const point &r) const {
        for (int k = 0; k < m; k++)
            if (x[k] ^ r.x[k]) return x[k] < r.x[k];
        return 0;
    }                                                   // 优先按照第一维，然后第二
维，依次类推，排序

    long long get_c(const point &r) {
        for (int k = 0; k < m; k++)
```

```
            if (r.x[k] < x[k]) return 0ll;
        int sum = 0;
        for (int k = 0; k < m; k++)
            sum += r.x[k] - x[k];
        long long ans = fac[sum];
        for (int k = 0; k < m; k++)
            (ans *= facinv[r.x[k] - x[k]]) %= mod;
        return ans;                                 // 多重组合数
    }
}a[N];
inline long long sub(long long x, long long y) {
    return x - y > 0? x - y: x - y + mod;
}
int main() {
    scanf("%d%d", &m, &n);
    int maxx = 0;
    for (int k = 0; k < m; k++)
        scanf("%d", &a[n + 1].x[k]), maxx = std::max(maxx, a[n + 1].x[k]);
    for (int k = 1; k <= n; k++)
        for (int i = 0; i < m; i++)
            scanf("%d", &a[k].x[i]);
    maxx *= m;
    fac[0] = 1;
    for (int k = 1; k <= maxx; k++)
        fac[k] = (fac[k - 1] * k) % mod;
    inv[1] = 1;
    for (int k = 2; k <= maxx; k++)
        inv[k] = (mod - mod / k) * inv[mod % k] % mod;
    facinv[0] = 1;
    for (int k = 1; k <= maxx; k++)
        facinv[k] = facinv[k - 1] * inv[k] % mod;       // 预处理阶乘，以及阶乘的逆元
    std::sort(a + 1, a + 1 + n);                         // 将所有点排序，保证转移时有序
    for (int k = 1; k <= n + 1; k ++) {
        f[k] = a[0].get_c(a[k]);
        for (int i = 1; i < k; i++)
            f[k] = sub(f[k], f[i] * a[i].get_c(a[k]) % mod);  // 容斥，get_c函数计算两点间不考虑障碍点的路径数目
    }
    printf("%lld\n", f[n + 1]);
}
```

本题易错点

- 转移时没有考虑到先后顺序
- 容斥时重复统计贡献