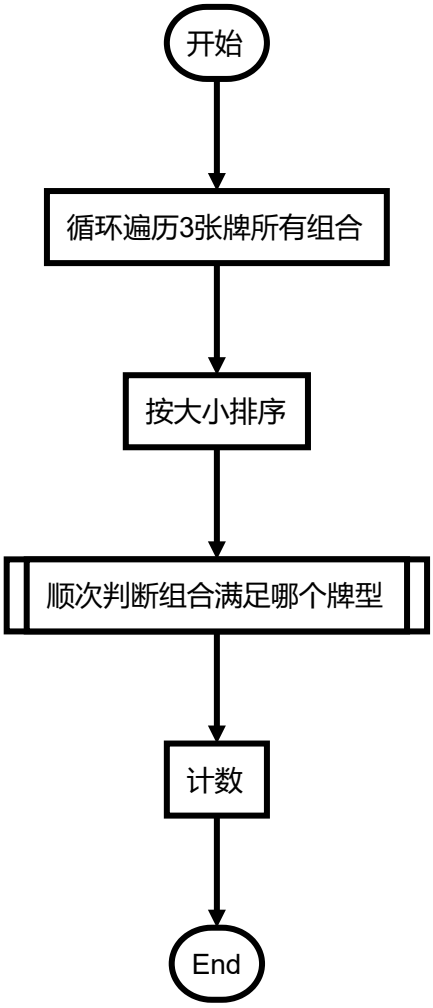
# 一、复杂模拟题

## 1. 代码流程




## 2. 提交代码

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

int A, B;
int a1, b1, a2, b2;
int Cnt[10];
int Sort[5];      // 排序大小

struct card {
    int x, y;

    // bool operator == (card &ano) const {
    //     return (x == ano.x && y == ano.y);
    // }
}C[6];

bool f2 () {
    // 炸弹
    int tmp = 0, res = 0;
    for (int i = 2; i <= 5; i++) {
        if (C[1].x == C[i].x) tmp++;
    }
    res = max(res, tmp);
    tmp = 0;
    for (int i = 4; i >= 1; i--) {
        if (C[5].x == C[i].x) tmp++;
    }
    res = max(res, tmp);

    if (res >= 3) return true;
    return false;
}

bool f3() {
    // 三带二
    int res = 0;
    for (int i = 0; i < 4; i++) {
        if (Sort[i] == Sort[i + 1]) res++;
    }

    if (res == 3) return true;
    return false;
}

bool f4() {
    int res = 0;
    for (int i = 4; i >= 1; i--) {
        if (C[5].y == C[i].y) res++;
    }
```

```
    if (res == 4) return true;
    return false;
}

bool f5() {
    // 顺子
    int tmp[5], ok = true;

    for (int i = 0; i < 5; i++) {
        tmp[i] = C[i + 1].x;
    }

    sort(tmp, tmp + 5);

    for (int i = 0; i < 4 ; i++) {
        if (tmp[i] + 1 != tmp[i + 1]) {
            ok = false;
            break;
        }
    }

    return ok;
}

bool f1 () {
    // 同花顺
    if (f4() && f5()) return true;
    return false;
}

bool f6() {
    bool ok = false;

    for (int i = 0; i < 4; i++) {
        if (Sort[i] == Sort[i + 1]) {
            if (ok == true) return true;
            ok = true;
        }
        else {
            ok = false;
        }
    }

    return false;
}

bool f7() {
    int res = 0;
    for (int i = 0; i < 4; i++) {
        if (Sort[i] == Sort[i + 1])
```

```
            res++;
    }

    if (res == 2) return true;
    return false;
}

bool f8() {
    int res = 0;
    for (int i = 0; i < 4; i++) {
        if (Sort[i] == Sort[i + 1])
            return true;
    }

    return false;
}

void count(int a, int b, int c) {
    C[3].x = a / B;
    C[3].y = a % B;
    C[4].x = b / B;
    C[4].y = b % B;
    C[5].x = c / B;
    C[5].y = c % B;

    for (int i = 0; i < 5; i++) {
        Sort[i] = C[i + 1].x;
    }

    // 排序后可通过判断相邻相等关系判断某些牌型
    sort(Sort, Sort + 5);

    if (f1()) Cnt[1]++;
    else if (f2()) Cnt[2]++;
    else if (f3()) Cnt[3]++;
    else if (f4()) Cnt[4]++;
    else if (f5()) Cnt[5]++;
    else if (f6()) Cnt[6]++;
    else if (f7()) Cnt[7]++;
    else if (f8()) Cnt[8]++;
    else Cnt[9]++;
}

void solve() {
    cin >> A >> B;
    cin >> a1 >> b1 >> a2 >> b2;

    C[1].x = a1;
    C[1].y = b1;
    C[2].x = a2;
    C[2].y = b2;
```

```
    // C_98^3
    // 通过约束3张牌的大小关系来选取组合
    for (int i = 0; i < A * B; i++) {       // % B
        if (a1 * B + b1 == i || a2 * B + b2 == i)
            continue;
        for (int j = i + 1; j < A * B; j++) {
            if (a1 * B + b1 == j || a2 * B + b2 == j)
                continue;
            for (int k = j + 1; k < A * B; k++) {
                if (a1 * B + b1 == k || a2 * B + b2 == k)
                    continue;
                count(i, j, k);
            }
        }
    }


    for (int i = 1; i <= 9; i++) {
        cout << Cnt[i] << " ";
    }
    cout << '\n';
}

int main (){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    solve();
    return 0;
}
```

### 3. 解法的时间复杂度

循环遍历3张牌所有组合复杂度为 $O(C^3_{A*B-2})$，之后的操作对象只有5张牌和数据规模无关，因此算法的时间复杂度为 $O(C^3_{A*B-2})$。

# 二、第三周模测

# 切蛋糕

#### 题目大意

对一个正方体，可以用和表面平行的平面切割，问每一次切割后最大长方的体积。

#### 解法

可以转化成维护 $x/y/z$ 三个方向上的最长线段，它们对应最大长方体的长宽高。
数据较小，每次切割之后，排序，差分，取最大值。

**时间复杂度**

一共切割 $m$ 次，每次排序复杂度为 $O(mlogm)$， 差分取最值为 $O(m)$ ，总复杂度为 $O(m^2logm)$

**代码**

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

char Cut[305];
ll mX, mY, mZ;
// 维护三个方向上最大长度
vector<int> X, Y, Z;

void solve() {
    int n, m;
    cin >> n >> m;
    mX = mY = mZ = n;

    for (int i = 1; i <= m; i++) {
        cin >> Cut[i];
    }

    X.push_back(0);
    X.push_back(n);
    Y.push_back(0);
    Y.push_back(n);
    Z.push_back(0);
    Z.push_back(n);

    int a, Max;
    for (int i = 1; i <= m; i++) {
        Max = 0;
        cin >> a;
        if (Cut[i] == 'x') {
            X.push_back(a);
            sort(X.begin(), X.end());
            for (int j = 1; j < X.size(); j++) {
                if (X[j] - X[j - 1] > Max) Max = X[j] - X[j - 1];
            }
            mX = Max;
        }
        else if (Cut[i] == 'y') {
            Y.push_back(a);
            sort(Y.begin(), Y.end());
            for (int j = 1; j < Y.size(); j++) {
                if (Y[j] - Y[j - 1] > Max) Max = Y[j] - Y[j - 1];
            }
            mY = Max;
        }
        else {
            Z.push_back(a);
            sort(Z.begin(), Z.end());
            for (int j = 1; j < Z.size(); j++) {
                if (Z[j] - Z[j - 1] > Max) Max = Z[j] - Z[j - 1];
```

```
            }
            mZ = Max;
        }
        // cout << mX << " " << mY << " " << mZ << '\n';
        cout << 1ll * mX * mY * mZ << '\n';
    }
}
```

# 摘星星

### 题目大意

给定一个 $n * m$ 的地图，地图上某些坐标有星星，给定一个动作序列，问能收集到多少星星。

### 解法

使用 $set$ 维护星星的坐标，每个运动指令视为一次“跳跃”，从当前位置 $(cX, cY)$ 到 $(cX + Step[D][0] * step, cY + Step[D][1] * step)$，二分搜索可以搜索集到的星星，统计。

### 时间复杂度

共有 $k$ 个星星，至多每个星星摘一次，时间复杂度为 $O(k)$

### 代码

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

set<int> SetX[10005], SetY[10005];
bool Reach[10005][10005];

void solve() {
    int n, m, k;
    cin >> n >> m;
    cin >> k;

    int x, y, ans = 0;
    for (int i = 0; i < k; i++) {
        cin >> x >> y;
        SetX[x].insert(y);
        SetY[y].insert(x);
    }

    int t;
    cin >> t;

    int cX = 1, cY = 1;
    if (SetX[1].count(1)) {
        ans++;
        SetX[1].erase(1);
        SetY[1].erase(1);
    }

    char Dir;
    int step, D;

    while (t--) {
        cin >> Dir >> step;

        // (cX, cY) -> (cX + Step[D][0] * step, cY + Step[D][1] * step)
        int nX = cX, nY = cY;
        set<int>::iterator A, B, Clr;

        if (Dir == 'W') {
            nX -= step;
            // set上二分，lower_bound(x)返回第一个大于等于x的元素下标
            A = SetY[cY].lower_bound(nX);
            B = SetY[cY].lower_bound(cX);

            if (A == SetY[cY].end() || B == SetY[cY].begin())
                goto TAG;
            B--;
            if (*A > *B) goto TAG;
            while (A != B) {
```

```
            if (!Reach[*A][cY]) {
               Reach[*A][cY] = 1;
               ans++;
            }
            Clr = A;
            A++;
            SetY[cY].erase(Clr);

        }
        if (!Reach[*A][cY]) {
            Reach[*A][cY] = 1;
            ans++;
        }
    }
    else if (Dir == 'A') {
        nY -= step;
        A = SetX[cX].lower_bound(nY);
        B = SetX[cX].lower_bound(cY);

        if (A == SetX[cX].end() || B == SetX[cX].begin())
            goto TAG;
        B--;
        if (*A > *B) goto TAG;
        while (A != B) {
            if (!Reach[cX][*A]) {
               Reach[cX][*A] = 1;
               ans++;
            }
            Clr = A;
            A++;
            SetX[cX].erase(Clr);
        }
        if (!Reach[cX][*A]) {
            Reach[cX][*A] = 1;
            ans++;
        }
    }
    else if (Dir == 'S') {
        nX += step;
        A = SetY[cY].upper_bound(cX);
        B = SetY[cY].upper_bound(nX);

        if (A == SetY[cY].end() || B == SetY[cY].begin())
            goto TAG;
        B--;
        if (*A > *B) goto TAG;
        while (A != B) {
            if (!Reach[*A][cY]) {
               Reach[*A][cY] = 1;
               ans++;
            }
```

```
                Clr = A;
                A++;
                SetY[cY].erase(Clr);
            }
            if (!Reach[*A][cY]) {
                Reach[*A][cY] = 1;
                ans++;
            }
        }
        else if (Dir == 'D') {
            nY += step;
            A = SetX[cX].upper_bound(cY);
            B = SetX[cX].upper_bound(nY);

            if (A == SetX[cX].end() || B == SetX[cX].begin())
                goto TAG;
            B--;
            if (*A > *B) goto TAG;
            while (A != B) {
                if (!Reach[cX][*A]) {
                   Reach[cX][*A] = 1;
                   ans++;
                }
                Clr = A;
                A++;
                SetX[cX].erase(Clr);
            }
            if (!Reach[cX][*A]) {
                Reach[cX][*A] = 1;
                ans++;
            }
        }
        TAG: cX = nX, cY = nY;
    }

    cout << ans << '\n';
}
```