1 2022mid1

1.1 Problem A. Task and Penalty

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int a[1000000];

int main() {
    int num;
    cin >> num;
    for (int i=0;i<num;i++) {
        cin >> a[i];
    }
    sort(a, a + num);
    int total = 0;
    int ans = 0;
    for (int i = 0; i < num; i++) {
        total = total + a[i];
        ans = ans + total;
    }
    cout << ans;
}
```

1.2 Problem B. Counting Rooms

```cpp
#include <iostream>
#include <vector>
using namespace std;

int n, m;
vector<vector<char>> grid;
vector<vector<bool>> visited;

// 移動方向：上、下、左、右
int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, -1, 1};

// 深度優先搜索（DFS）函數
void dfs(int x, int y) {
    // 標記當前節點為已訪問
    visited[x][y] = true;

    // 遍歷四個方向
    for (int i = 0; i < 4; i++) {
        int nx = x + dx[i];
        int ny = y + dy[i];

        // 檢查邊界條件和是否可以訪問
        if (nx >= 0 && nx < n && ny >= 0 && ny < m && !visited[nx][ny] && grid[nx][ny] == '.') {
            dfs(nx, ny); // 繼續訪問相鄰的 `.` 格子
        }
    }
}

int main() {
    cin >> n >> m;
    grid.resize(n, vector<char>(m));
    visited.resize(n, vector<bool>(m, false));

    // 讀取地圖
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> grid[i][j];
        }
    }

    int roomCount = 0;

    // 遍歷整個地圖
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            // 如果找到未訪問的 `.`，表示找到一個新房間
            if (grid[i][j] == '.' && !visited[i][j]) {
                dfs(i, j); // 執行 DFS
                roomCount++; // 房間計數加一
            }
        }
    }

    // 輸出房間數量
    cout << roomCount << endl;

    return 0;
}
```

1.3 Problem C. Sum of Three Values

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Number {
    int value;
    int index;
};

int main() {
    int n;
    long long x;
    cin >> n >> x;

    vector<Number> arr(n);
    for (int i = 0; i < n; ++i) {
        cin >> arr[i].value;
        arr[i].index = i + 1; // 使用 1-based index
    }

    // 按值對數組進行排序
    sort(arr.begin(), arr.end(), [](const Number &a, const Number &b) {
        return a.value < b.value;
    });

    // 嘗試固定第一個數並使用雙指針法查找另外兩個數
    for (int i = 0; i < n - 2; ++i) {
        long long target = x - arr[i].value;
        int left = i + 1, right = n - 1;

        // 使用雙指針尋找另外兩個數
        while (left < right) {
            long long sum = arr[left].value + arr[right].value;
            if (sum == target) {
                // 找到結果，將索引排序後輸出
                vector<int> result = {arr[i].index, arr[left].index, arr[right].index};
                sort(result.begin(), result.end()); // 將索引由小到大排序
                cout << result[0] << " " << result[1] << " " << result[2] << endl;
                return 0;
            } else if (sum < target) {
                ++left; // 增加左指針
            } else {
                --right; // 減少右指針
            }
        }
    }

    // 如果沒有找到任何解
    cout << "IMPOSSIBLE" << endl;
    return 0;
}
```

1.4 Problem D. LR insertion

```cpp
#include <iostream>
#include <list>
#include <string>
using namespace std;

int main() {
    int N;
    string S;

    // Read input
    cin >> N >> S;

    // Initialize list with 0
    list<int> A;
    A.push_back(0);

    // Keep track of position of i-1
    list<int>::iterator prev = A.begin();

    // Process each character in the string
    for (int i = 1; i <= N; i++) {
        if (S[i-1] == 'L') {
            // Insert to the left of previous number
            prev = A.insert(prev, i);
        } else {
            // Insert to the right of previous number
            // Need to move iterator one step forward before inserting
            list<int>::iterator next = prev;
            ++next;
            prev = A.insert(next, i);
        }
    }

    // Print the final sequence
    bool first = true;
    for (int num : A) {
        if (!first) cout << " ";
        cout << num;
        first = false;
    }
    cout << endl;

    return 0;
}
```

1.5 Problem E. Second Day in Aincrad

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
using namespace std;

// 用來生成所有可能的數字
void generateNumbers(const string &pattern, vector<int> &candidates) {
    int numX = 0;
    for (char ch : pattern) {
        if (ch == 'X') numX++;
    }

    // 當沒有 X 時，直接將原值作為候選數
    if (numX == 0) {
        candidates.push_back(stoi(pattern));
        return;
    }

    int len = pattern.size();
    int maxReplace = pow(10, numX); // 用於生成所有組合
    for (int i = 0; i < maxReplace; ++i) {
        string numStr = pattern;
        int temp = i;

        // 替換掉 X
        for (int j = len - 1; j >= 0; --j) {
            if (numStr[j] == 'X') {
                numStr[j] = '0' + (temp % 10);
                temp /= 10;
            }
        }

        // 去掉前導零的數
        if (numStr[0] != '0' || numStr == "0") {
            candidates.push_back(stoi(numStr));
        }
    }
}

int main() {
    string S_A, OP, S_B, EQ, S_C;
    cin >> S_A >> OP >> S_B >> EQ >> S_C;

    vector<int> A_candidates, B_candidates, C_candidates;

    // 生成候選值
    generateNumbers(S_A, A_candidates);
    generateNumbers(S_B, B_candidates);
    generateNumbers(S_C, C_candidates);

    // 檢查每組候選值是否滿足等式
    for (int A : A_candidates) {
        for (int B : B_candidates) {
            for (int C : C_candidates) {
                bool isValid = false;
                if (OP == "+" && A + B == C) {
                    isValid = true;
                } else if (OP == "-" && A - B == C) {
                    isValid = true;
                }

                if (isValid) {
                    cout << A << " " << B << " " << C << endl;
                    return 0;
                }
            }
        }
    }

    // 若無解，輸出空解
    cout << "IMPOSSIBLE" << endl;
    return 0;
}
```

1.6 Problem F. Circular Nearest Smaller Values

```cpp
#include <iostream>
#include <vector>
#include <stack>
using namespace std;

int main()
{
    int n;
    cin >> n;
    vector<int> A(n), result(n);
    for (int i = 0; i < n; ++i)
    {
        cin >> A[i];
    }

    // 將原陣列複製一份，形成 2n 長度的陣列
    vector<int> extendedA(2 * n);
    for (int i = 0; i < n; ++i)
    {
        extendedA[i] = extendedA[i + n] = A[i];
    }

    stack<int> st;

    // 逆向遍歷複製後的 2n 長度的陣列
    for (int i = 2 * n - 1; i >= 0; --i)
    {
        int index = i % n; // 獲取當前元素在原陣列中的位置

        // 移除棧中所有大於當前元素的索引
        while (!st.empty() && extendedA[st.top()] > extendedA[i])
        {
            st.pop();
        }

        // 設定結果為最近的較小或相等位置，如果棧為空則表示無較小值
        if (i < n)
        {

            // 只在第一次遍歷時填充結果
            result[i] = st.empty() ? 0 : st.top() % n + 1; // 加1轉為1-based index
        }

        // 將當前索引壓入棧中
        st.push(i);
    }

    // 輸出結果
    for (int i = 0; i < n; ++i)
    {
        cout << result[i] << " ";
    }
    cout << endl;

    return 0;
}
```

1.7 Problem G. Gluttony

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

typedef long long ll;

bool canAchieve(const vector<ll> &A, const vector<ll> &F, ll M, ll K) {
    ll neededTraining = 0;
    int n = A.size();

    for (int i = 0; i < n; ++i) {
        ll maxA = M / F[i]; // 每個食物能夠允許的最大消耗係數
        if (A[i] > maxA) {
            neededTraining += A[i] - maxA;
            if (neededTraining > K) return false; // 超出訓練次數
        }
    }
    return neededTraining <= K;
}

ll minScore(vector<ll> &A, vector<ll> &F, ll K) {
    sort(A.begin(), A.end());
    sort(F.rbegin(), F.rend()); // F 降序排列
    ll left = 0, right = A.back() * F.front();

    while (left < right) {
        ll mid = left + (right - left) / 2;
        if (canAchieve(A, F, mid, K)) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }
    return left;
}

int main() {
    int n;
    ll K;
    cin >> n >> K;

    vector<ll> A(n), F(n);
    for (int i = 0; i < n; ++i) cin >> A[i];
    for (int i = 0; i < n; ++i) cin >> F[i];

    cout << minScore(A, F, K) << endl;
    return 0;
}
```

1.8 Problem H. Sticks

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int n;
vector<int> sticks;
int total_length;

// 回溯法檢查能否成功組裝長度為 L 的木棍
bool canForm(int L, int current_sum, int index, int used_count) {
    if (used_count == n) return true; // 全部段已使用
    if (current_sum == L) return canForm(L, 0, 0, used_count); // 完成一根木棍，重置

    for (int i = index; i < n; ++i) {
        if (sticks[i] == 0) continue; // 已使用的段跳過

        if (current_sum + sticks[i] <= L) {
            int temp = sticks[i];
            sticks[i] = 0; // 標記已使用

            // 進行下一步回溯
            if (canForm(L, current_sum + temp, i + 1, used_count + 1)) return true;

            sticks[i] = temp; // 還原狀態

            if (current_sum == 0 || current_sum + sticks[i] == L) break; // 剪枝
        }
    }
    return false;
}

int findMinimumLength() {
    for (int L = 1; L <= total_length; ++L) {
        if (total_length % L == 0) { // 確保 L 是 total_length 的因數
            if (canForm(L, 0, 0, 0)) return L;
        }
    }
    return total_length;
}

int main() {
    while (cin >> n && n != 0) {
        sticks.resize(n);
        total_length = 0;

        for (int i = 0; i < n; ++i) {
            cin >> sticks[i];
            total_length += sticks[i];
        }

        sort(sticks.rbegin(), sticks.rend()); // 降序排列便於剪枝
        cout << findMinimumLength() << endl;
    }

    return 0;
}
```

2 w1

2.1 Apple Division

```cpp
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> apples(n);

    // 輸入蘋果重量
    for (int i = 0; i < n; i++) {
        cin >> apples[i];
    }

    long long totalWeight = 0;
    for (int i = 0; i < n; i++) {
        totalWeight += apples[i];
    }

    long long minDifference = totalWeight; // 初始化差值為總重量，理論上最大值

    // 枚舉所有可能的分組方式
    for (int subset = 0; subset < (1 << n); subset++) {
        long long group1Weight = 0;

        // 計算當前子集 (分組) 的重量
        for (int i = 0; i < n; i++) {
            if (subset & (1 << i)) {
                group1Weight += apples[i];
            }
        }

        // 計算兩組重量差
        long long group2Weight = totalWeight - group1Weight;
        long long currentDifference = abs(group1Weight - group2Weight);

        // 更新最小的重量差
        minDifference = min(minDifference, currentDifference);
    }

    // 輸出最小的重量差
    cout << minDifference << endl;
    return 0;
}
```

2.2 Chinese Rings

```cpp
#include <iostream>
#include <vector>

const int MAX_N = 20;
int n;
std::vector<bool> state(MAX_N);
int count_op = 0;

void move_out(int c) {
    std::cout << "Move ring " << c + 1 << " out" << std::endl; // Output "Move ring n out"
    state[c] = false; // Change state to indicate it's out
    count_op++;
}

void move_in(int c) {
    std::cout << "Move ring " << c + 1 << " in" << std::endl; // Output "Move ring n in"
    state[c] = true; // Change state to indicate it's in
    count_op++;
}

void solve(int n);
void rsolve(int n);

// 111 -> 000
void solve(int n) {
    if (n == 1) {
        move_out(0);
    } else if (n == 2) {
        move_out(1);
        solve(1);
    } else {
        solve(n - 2);
        move_out(n - 1);
        rsolve(n - 2);
        solve(n - 1);
    }
}

// 000 -> 111
void rsolve(int n) {
    if (n == 1) {
        move_in(0);
    } else if (n == 2) {
        rsolve(1);
        move_in(1);
    } else {
        rsolve(n - 1);
        solve(n - 2);
        move_in(n - 1);
        rsolve(n - 2);
    }
}

int main() {
    // std::cout << "Enter the number of rings: ";
    std::cin >> n;
    std::fill(state.begin(), state.begin() + n, true); // Initialize state to true (all rings in)
    solve(n);
    // std::cout << "Total moves: " << count_op << std::endl;
    return 0;
}
```

2.3 Citizen attention offices

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
#include <climits>

using namespace std;

int calculateDistance(pair<int, int> a, pair<int, int> b) {
    return abs(a.first - b.first) + abs(a.second - b.second);
}

int main() {
    int t;
    cin >> t; // 測試案例數量

    while (t--) {
        int n;
        cin >> n; // 非零人口的區域數量
        vector<pair<pair<int, int>, int>> areas(n); // 每個區域的座標及人口

        for (int i = 0; i < n; i++) {
            int r, c, p;
            cin >> r >> c >> p;
            areas[i] = {{r, c}, p};
        }

        // 所有 25 個可能的位置
        vector<pair<int, int>> allPositions;
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                allPositions.push_back({i, j});
            }
        }

        vector<int> bestOffices;
        int minTotalDistance = INT_MAX;

        // 枚舉所有選擇 5 個辦事處的位置
        vector<int> comb(25);
        fill(comb.begin(), comb.begin() + 5, 1); // 前 5 個 1 表示選擇這些位置
        do {
            vector<int> currentOffices;
            for (int i = 0; i < 25; i++) {
                if (comb[i]) {
                    currentOffices.push_back(i);
                }
            }

            // 計算當前選擇下的總距離
            int totalDistance = 0;
            for (auto area : areas) {
                int minDistance = INT_MAX;
                for (int officeIdx : currentOffices) {
                    pair<int, int> office = allPositions[officeIdx];
                    int distance = calculateDistance(area.first, office);
                    minDistance = min(minDistance, distance);
                }
                totalDistance += minDistance * area.second;
            }

            // 如果當前解更優，更新最優解
            if (totalDistance < minTotalDistance) {
                minTotalDistance = totalDistance;
                bestOffices = currentOffices;
            }

        } while (prev_permutation(comb.begin(), comb.end()));

        // 輸出結果，按遞增順序排列
        sort(bestOffices.begin(), bestOffices.end());
        for (int i = 0; i < 5; i++) {
            cout << bestOffices[i];
            if (i < 4) {
                cout << " ";
            }
        }
        cout << endl;
    }

    return 0;
}
```

2.4 Combinations

```cpp
#include <iostream>
```

```cpp
#include <cmath>
#include <vector>

using namespace std;

void generateCombinations(vector<int>& A, vector<int>& current, int start, int M) {
    // 如果當前組合長度達到 M，則輸出
    if (current.size() == M) {
        for (int i = 0; i < M; ++i) {
            if (i > 0) cout << " ";
            cout << current[i];
        }
        cout << endl;
        return;
    }

    // 遍歷所有可能的選擇
    for (int i = start; i < A.size(); ++i) {
        current.push_back(A[i]);  // 選擇當前元素
        generateCombinations(A, current, i + 1, M);  // 繼續選擇剩下的元素
        current.pop_back();  // 回溯，取消選擇當前元素
    }
}

int main() {
    int n,m;
    cin >> n >> m;
    vector<int> input(n);
    for (int i=0;i<=n-1;i++) {
        cin >> input[i];
    }
    vector<int> current;

    generateCombinations(input,current,0,m);
}
```

2.5 Creating Strings

```cpp
#include <iostream>
#include <algorithm>
#include <set>
#include <string>

using namespace std;

int main() {
    string s;
    cin >> s;
    sort(s.begin(), s.end());
    set<string> permutations;
    do {
        permutations.insert(s);
    } while (next_permutation(s.begin(), s.end()));
    cout << permutations.size() << endl;
    for (const string& perm : permutations) {
        cout << perm << endl;
    }
    return 0;
}
```

2.6 Gray code

```cpp
#include <iostream>
#include <vector>
using namespace std;

vector<string> geneGray(int n) {
    if (n == 1) {
        return {"0" , "1"};
    }
    vector<string> prev = geneGray(n-1);
    vector<string> graycode;

    for (auto code:prev) {
        graycode.push_back("0" + code);
    }
    for (int j=prev.size()-1;j>=0;j--) {
        graycode.push_back("1" + prev[j]);
    }
    return graycode;
}

int main() {
    int n;
    cin >> n;
    vector<string> ans;
    ans = geneGray(n);
    for (auto code:ans) {
        cout << code << endl;
    }
}


//#include <iostream>
//#include <vector>
//#include <string>
//
//using namespace std;
//
//vector<string> geneGrayCode(int n) {
//    if (n == 1) {
//        return {"0","1"};
//    }
//
//    vector<string> prev = geneGrayCode(n-1);
//    vector<string> graycode;
//
//    for (auto code:prev) {
//        graycode.push_back("0" + code);
//    }
//
//    for (int i=prev.size()-1 ;i>=0;i--){
//        graycode.push_back("1" + prev[i]);
//    }
////      ex
//// 00
//// 01
//// 11
//// 10
////      add "0"
//// 000
//// 001
//// 011
//// 010
////      add "1" in reverse direction
//// 110
//// 111
//// 101
//// 100
////      then it becomes...
//// 000
//// 001
//// 011
//// 010
//// 110
//// 111
//// 101
//// 100
//    return graycode;
//}
//
//int main() {
//    int n;
//    cin >> n;
//
//    vector<string> graycode = geneGrayCode(n);
//
//    for (auto code:graycode) {
//        cout << code << endl;
//    }
//}
```

2.7 Tower of Hanoi

```cpp
#include <iostream>
#include <vector>
using namespace std;

void hanoi(int n,int from,int via,int to, vector<pair<int,int>>& moves) {
    if (n == 1) {
        moves.push_back({from,to});
        return;
    }
    hanoi(n-1,from,to,via,moves);
    moves.push_back({from,to});
    hanoi(n-1,via,from,to,moves);
}

int main() {
    int n;
    cin >> n;
    vector<pair<int,int>> moves;
    hanoi(n,1,2,3,moves);
    cout << moves.size() << endl;
    for (auto ans:moves) {
        cout << ans.first << " " << ans.second << endl;
    }
}
```

3 w2

3.1 Chessboard and Queens

```cpp
#include <iostream>
#include <vector>
using namespace std;

int countWays = 0;              // 记录可行方案的数量
vector<vector<char>> board(8, vector<char>(8));  // 棋盘

// 用于记录每列、主对角线、副对角线是否被占用
bool cols[8] = {false};
bool main_diag[15] = {false};
bool anti_diag[15] = {false};

void solve(int row) {
    // 如果成功在第 8 行放置皇后，表示找到一种方案
    if (row == 8) {
        countWays++;
        return;
    }

    for (int col = 0; col < 8; col++) {
        // 如果当前格子是保留的或该列、对角线已被占用，则跳过
        if (board[row][col] == '*' || cols[col] || main_diag[row - col + 7] || anti_diag[row + col])
            continue;

        // 标记该列、主对角线、副对角线为占用状态
        cols[col] = main_diag[row - col + 7] = anti_diag[row + col] = true;

        // 递归处理下一行
        solve(row + 1);

        // 回溯：恢复状态
        cols[col] = main_diag[row - col + 7] = anti_diag[row + col] = false;
    }
}

int main() {
    // 输入棋盘
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            cin >> board[i][j];
        }
    }
    // 从第 0 行开始放置皇后
    solve(0);

    // 输出可行方案的数量
    cout << countWays << endl;

    return 0;
}
```

3.2 Grid Path

```cpp
#include <bits/stdc++.h>
#define int long long
#define double long double
#define pii pair<int, int>
#define N 200005
#define INF LONG_LONG_MAX
#define x first
#define y second
#define all(a) a.begin(),a.end()
#define IOS ios::sync_with_stdio(0),cin.tie(0)
using namespace std;
string str;
int dx[4] = {1,0,-1,0},dy[4] = {0,1,0,-1};
bool vis[9][9];
int ans = 0;

void solve(int x,int y,int s){
    if(x < 1 || x > 7 || y < 1 || y > 7 || vis[x][y])return;
    if(x == 1 && y == 7 && s < 48)return;
    if(vis[x-1][y] && vis[x+1][y] && !vis[x][y+1] && !vis[x][y-1])return;
    if(!vis[x-1][y] && !vis[x+1][y] && vis[x][y+1] && vis[x][y-1])return;
    if(s == 48){
        ans++;
        return;
    }
    vis[x][y] = 1;
    int ans = 0;
    if(str[s] == 'L')solve(x - 1, y, s + 1);
    if(str[s] == 'R')solve(x + 1, y, s + 1);
    if(str[s] == 'U')solve(x, y - 1, s + 1);
    if(str[s] == 'D')solve(x, y + 1, s + 1);
    if(str[s] == '?'){
        for(int i = 0;i < 4;i++){
            int nx = x + dx[i],ny = y + dy[i];
            solve(nx,ny,s+1);
        }
    }
    vis[x][y] = 0;
}

signed main(){
    IOS;
    cin>>str;
    memset(vis,0,sizeof(vis));
    for(int i=1;i<=7;i++){
        vis[i][0] = 1;
        vis[8][i] = 1;
        vis[i][8] = 1;
        vis[0][i] = 1;
    }
    solve(1,1,0);
    cout<<ans<<"\n";
}
```

3.3 Hive

```cpp
#include<bits/stdc++.h>
#define IOS ios::sync_with_stdio(0),cin.tie(0),cout.tie(0)
#define endl '\n'
#define pii pair<int,int>
#define F first
#define S second
using namespace std;
const int inf = 1e9+7;
int ans = inf;
int n,m;
int mp[105][105] = {},vis[105][105] = {},deg[105][105] = {};
int dx[6] = {-1,0,1,1,0,-1};
int dy[6] = {1,2,1,-1,-2,-1};
pii f(int x,int y){
  return make_pair(x,(y-1)*2+(x&1)+(x%2==0?2:0));
}
int dijkstra(int sx,int sy,int ex,int ey){
  int dis[105][105] = {},vis2[105][105] = {};
  memset(dis,0x3f3f3f3f,sizeof(dis));
  dis[sx][sy] = mp[sx][sy];
  priority_queue<pair<int,pii>,vector<pair<int,pii>>,greater<pair<int,pii>>>pq;
  pq.push({dis[sx][sy],{sx,sy}});
  while(!pq.empty()){
    auto [x,y] = pq.top().S;
    pq.pop();
    if(vis[x][y] or vis2[x][y])continue;
    vis2[x][y] = 1;
    for(int i = 0;i<6;++i){
      int nx = x+dx[i],ny = y+dy[i];
      if(nx<1 or ny<1 or nx>n or ny>2*m)continue;
      if(vis[nx][ny] or vis2[nx][ny])continue;
      if(dis[nx][ny]>dis[x][y]+mp[nx][ny]){
        dis[nx][ny] = dis[x][y]+mp[nx][ny];
        pq.push({dis[nx][ny],{nx,ny}});
      }
    }
  }
  return dis[ex][ey];
}
void dfs(int x,int y,int ex,int ey,int dis,int sx,int sy,int ex2,int ey2){
  if(dis>ans)return;
  if(x==ex and y==ey){
    ans = min(ans,dis+dijkstra(sx,sy,ex2,ey2));
    return;
  }
  for(int i = 0;i<6;++i){
    int nx = x+dx[i],ny = y+dy[i];
    if(nx<1 or ny<1 or nx>n or ny>2*m)continue;
    deg[nx][ny]++;
  }
  for(int i = 0;i<6;++i){
    int nx = x+dx[i],ny = y+dy[i];
    if(nx<1 or ny<1 or nx>n or ny>2*m)continue;
    if(vis[nx][ny])continue;
    if(deg[nx][ny]>=2)continue;
    vis[nx][ny] = 1;
    dfs(nx,ny,ex,ey,dis+mp[nx][ny],sx,sy,ex2,ey2);
    vis[nx][ny] = 0;
  }
  for(int i = 0;i<6;++i){
    int nx = x+dx[i],ny = y+dy[i];
    if(nx<1 or ny<1 or nx>n or ny>2*m)continue;
    deg[nx][ny]--;
  }
}
void solve(){
  ans = inf;
  int x[5] = {},y[5] = {};
  cin>>n>>m;
  for(int i = 1;i<=4;++i){
    cin>>x[i];
    cin>>y[i];
    auto [nx,ny] = f(x[i],y[i]);
    x[i] = nx,y[i] = ny;
  }
  for(int i = 1;i<=n;++i){
    for(int j = 1;j<=m;++j){
      auto [nx,ny] = f(i,j);
      cin>>mp[nx][ny];
    }
  }
  vis[x[1]][y[1]] = 1;
  dfs(x[1],y[1],x[2],y[2],mp[x[1]][y[1]],x[3],y[3],x[4],y[4]);
  if(ans>=inf)cout<< -1 <<endl;
  else cout<<ans<<endl;
  vis[x[1]][y[1]] = 0;
}
int main(){
  IOS;
  int t;
  cin>>t;
  while(t--){
    solve();
  }
}
```

3.4 M Queen N Rocks

```cpp
#include <iostream>
#include <vector>
#include <cstring>

using namespace std;

int N, M;
int board_size;
vector<vector<int>> board;

int place(int r, int n, int m);
int placeQ(int r, int c);
int placeR(int r, int c);

int main() {
    int testcase;
    cin >> testcase;

    for (int i = 1; i <= testcase; i++) {
        cin >> N >> M;
        board_size = N + M;
        board.assign(15, vector<int>(15, 0));

        cout << place(0, 0, 0) << endl;
    }
    return 0;
}

int placeQ(int r, int c) {
    for (int i = 1; i <= r; i++) {
        if (c - i >= 0 && board[r - i][c - i] != 0) {
            return 0;
        }
        if (c + i <= board_size - 1 && board[r - i][c + i] != 0) {
            return 0;
        }
        if (r - i >= 0 && board[r - i][c] != 0) {
            return 0;
        }
    }
    return 1;
}

int placeR(int r, int c) {
    for (int i = 1; i <= r; i++) {
        if (c - i >= 0 && board[r - i][c - i] == 1) {
            return 0;
        }
        if (c + i <= board_size - 1 && board[r - i][c + i] == 1) {
            return 0;
        }
        if (r - i >= 0 && board[r - i][c] != 0) {
            return 0;
        }
    }
    return 1;
}

int place(int r, int n, int m) {
    if (r == board_size) return 1;
    int cnt = 0;
    for (int i = 0; i < board_size; i++) {
        if (n < N && placeQ(r, i)) {
            board[r][i] = 1;
            cnt += place(r + 1, n + 1, m);
            board[r][i] = 0;
        }
        if (m < M && placeR(r, i)) {
            board[r][i] = -1;
            cnt += place(r + 1, n, m + 1);
            board[r][i] = 0;
        }
    }
    return cnt;
}
```

3.5 Square

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// Function to check if we can form a square using the sticks
bool canFormSquare(vector<int>& sticks, vector<int>& sides, int index, int sideLength) {
    // If we have assigned all sticks
    if (index == sticks.size()) {
        // Check if all four sides are of equal length
        return sides[0] == sideLength && sides[1] == sideLength && sides[2] == sideLength && sides[3] == sideLength;
    }

    // Try to place the current stick in each side
    for (int i = 0; i < 4; i++) {
        if (sides[i] + sticks[index] <= sideLength) {
            sides[i] += sticks[index]; // Place the stick
            if (canFormSquare(sticks, sides, index + 1, sideLength)) {
                return true;
            }
            sides[i] -= sticks[index]; // Backtrack
        }
    }

    return false;
}

int main() {
    int N; // Number of test cases
    cin >> N;

    while (N--) {
        int M; // Number of sticks
        cin >> M;

        vector<int> sticks(M);
        int totalLength = 0;

        // Read the stick lengths and calculate the total length
        for (int i = 0; i < M; i++) {
            cin >> sticks[i];
            totalLength += sticks[i];
        }

        // If the total length is not divisible by 4, we cannot form a square
        if (totalLength % 4 != 0) {
            cout << "no" << endl;
            continue;
        }

        int sideLength = totalLength / 4; // Length of each side of the square
        vector<int> sides(4, 0); // Initialize the 4 sides of the square

        // Sort the sticks in descending order for better performance
        sort(sticks.rbegin(), sticks.rend());

        // If the largest stick is greater than the side length, it's impossible
        if (sticks[0] > sideLength) {
            cout << "no" << endl;
            continue;
        }

        // Use backtracking to check if we can form a square
        if (canFormSquare(sticks, sides, 0, sideLength)) {
            cout << "yes" << endl;
        } else {
            cout << "no" << endl;
        }
    }

    return 0;
}
```

3.6 Sudoku

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

const int N = 9;
vector<pair<int, int>> empty_cells; // To store the positions of all empty cells

/* A utility function to print grid as a single line */
void printGrid(int grid[N][N]) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            cout << grid[i][j];
    cout << endl;
}

/* Checks whether it will be legal to assign num to the given row, col */
bool isSafe(int grid[N][N], int row, int col, int num) {
    for (int x = 0; x < N; x++)
        if (grid[row][x] == num || grid[x][col] == num)
            return false;

    int startRow = row - row % 3, startCol = col - col % 3;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (grid[i + startRow][j + startCol] == num)
                return false;

    return true;
}

/* Checks if the initial Sudoku grid is valid */
bool isValidSudoku(int grid[N][N]) {
    for (int i = 0; i < N; i++) {
        vector<bool> rowCheck(N + 1, false);
        vector<bool> colCheck(N + 1, false);
        for (int j = 0; j < N; j++) {
            if (grid[i][j] != 0) {
                if (rowCheck[grid[i][j]])
                    return false;
                rowCheck[grid[i][j]] = true;
            }
            if (grid[j][i] != 0) {
                if (colCheck[grid[j][i]])
                    return false;
                colCheck[grid[j][i]] = true;
            }
        }
    }
    for (int row = 0; row < N; row += 3) {
        for (int col = 0; col < N; col += 3) {
            vector<bool> boxCheck(N + 1, false);
            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++) {
                    int num = grid[row + i][col + j];
                    if (num != 0) {
                        if (boxCheck[num])
                            return false;
                        boxCheck[num] = true;
                    }
                }
            }
        }
    }
    return true;
}

/* Recursive function to solve the Sudoku grid */
bool solveSudoku(int grid[N][N], int index = 0) {
    if (index == empty_cells.size()) return true;

    int row = empty_cells[index].first;
    int col = empty_cells[index].second;

    // Attempt to place numbers in ascending order for lexicographically smallest solution
    for (int num = 1; num <= N; num++) {
        if (isSafe(grid, row, col, num)) {
            grid[row][col] = num;
            if (solveSudoku(grid, index + 1))
                return true;
            grid[row][col] = 0; // Backtrack if not successful
        }
    }
    return false;
}

/* Converts a single string line to a 9x9 Sudoku grid */
void parseGrid(const string& line, int grid[N][N]) {
    empty_cells.clear(); // Reset empty cells for each new grid

    for (int i = 0; i < N * N; i++) {
        int row = i / N, col = i % N;
        char ch = line[i];
        grid[row][col] = (ch == '.') ? 0 : ch - '0';

        // Store positions of empty cells for backtracking
        if (grid[row][col] == 0)
            empty_cells.emplace_back(row, col);
    }
}

int main() {
    string line;
    while (cin >> line && line != "end") {
        int grid[N][N];
        parseGrid(line, grid);

        // 檢查初始數獨是否合法
        if (!isValidSudoku(grid)) {
            cout << "No solution." << endl;
            continue;
        }

        if (solveSudoku(grid))
```

```cpp
            printGrid(grid);
        else
            cout << "No solution." << endl;
    }

    return 0;
}
```

3.7 Sum It Up

```cpp
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>

using namespace std;

int t, n;
vector<int> nums;
set<vector<int>> resultSet;

void findSums(int target, int index, vector<int>& current) {
    if (target == 0) {
        resultSet.insert(current); // Add the current sum to the result set
        return;
    }

    for (int i = index; i < n; i++) {
        if (i > index && nums[i] == nums[i - 1]) continue; // Skip duplicates
        if (nums[i] <= target) {
            current.push_back(nums[i]);
            findSums(target - nums[i], i + 1, current); // Recursively find sums
            current.pop_back(); // Backtrack
        }
    }
}

int main() {
    while (true) {
        // Read input
        cin >> t >> n;
        if (n == 0) break; // Exit condition

        nums.resize(n);
        for (int i = 0; i < n; i++) {
            cin >> nums[i];
        }

        resultSet.clear(); // Reset result set for each test case
        vector<int> current;

        findSums(t, 0, current);

        // Output results
        cout << "Sums of " << t << ":\n";
        if (resultSet.empty()) {
            cout << "NONE\n";
        } else {
            for (auto it = resultSet.rbegin(); it != resultSet.rend(); ++it) {
                for (size_t i = 0; i < it->size(); i++) {
                    if (i > 0) cout << "+";
                    cout << (*it)[i];
                }
                cout << endl;
            }
        }
    }

    return 0;
}
```

4 w3

4.1 Array Arrangement

```cpp
#include <iostream>
#include <list>
#include <unordered_map>

using namespace std;

int main() {
    int t; // Number of test cases
    cin >> t;

    while (t--) {
        int n, q; // Length of sequence and number of queries
        cin >> n >> q;

        list<int> sequence; // To store the current sequence
        unordered_map<int, list<int>::iterator> positions; // To store positions of elements

        // Initialize the sequence from 1 to n
        for (int i = 1; i <= n; ++i) {
            sequence.push_back(i);
            positions[i] = --sequence.end(); // Store iterator pointing to the element
        }

        // Process each query
        for (int i = 0; i < q; ++i) {
            char type;
            int x;
            cin >> type >> x;

            // Remove x from its current position
            if (positions.find(x) != positions.end()) {
                sequence.erase(positions[x]);
            }

            // Move x to the head or the tail
            if (type == 'H') {
                // Move x to the head
                sequence.push_front(x);
                positions[x] = sequence.begin();
            } else if (type == 'T') {
                // Move x to the tail
                sequence.push_back(x);
                positions[x] = --sequence.end();
            }
        }

        // Print the final sequence
        for (int i : sequence) {
            cout << i << " ";
        }
        cout << endl;
    }

    return 0;
}
```

4.2 Banana milk lover

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Group {
    int index;
    int people;
    long long total_banana_milk;
    int max_contribution;
    vector<int> contributions;

    bool operator<(const Group &other) const {
        if (total_banana_milk != other.total_banana_milk) {
            return total_banana_milk > other.total_banana_milk; // Descending by total banana milk
        }
        if (max_contribution != other.max_contribution) {
            return max_contribution > other.max_contribution; // Descending by max contribution
        }
        if (people != other.people) {
            return people > other.people; // Descending by number of people
        }
        return index < other.index; // Ascending by input order
    }
};

int main() {
    int T;
    cin >> T;
    while (T--) {
        int N;
        cin >> N;
        vector<Group> groups;

        for (int i = 0; i < N; ++i) {
            int K;
            cin >> K;
            Group group;
            group.index = i;
            group.people = K;
            group.total_banana_milk = 0;
            group.max_contribution = 0;
            group.contributions.resize(K);

            for (int j = 0; j < K; ++j) {
                cin >> group.contributions[j];
                group.total_banana_milk += group.contributions[j];
                group.max_contribution = max(group.max_contribution, group.contributions[j]);
            }

            groups.push_back(group);
        }

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

        for (const auto &group : groups) {
            for (int contribution : group.contributions) {
                cout << contribution << " ";
            }
            cout << endl;
        }
    }

    return 0;
}
```

4.3 Broken Keyboard (a.k.a. Beiju Text)

```cpp
#include <iostream>
#include <list>
#include <string>

```

```cpp
int main() {
    std::string line;
    while (std::getline(std::cin, line)) {
        std::list<char> text;
        auto it = text.begin();
        for (char ch : line) {
            if (ch == '[') {
                it = text.begin();
            } else if (ch == ']') {
                it = text.end();
            } else {
                text.insert(it, ch);
            }
        }
        for (char ch : text) {
            std::cout << ch;
        }
        std::cout << std::endl;
    }
    return 0;
}
```

4.4 Hunting and Distributing Meals

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <map>

using namespace std;

// 定義貓的結構體
struct Cat {
    string name;
    string position;
    int age;
};

// 定義地位優先級映射
map<string, int> priority = {
    {"elder", 1}, {"nursy", 2}, {"kit", 3},
    {"warrior", 4}, {"appentice", 5}, {"
        medicent", 6},
    {"deputy", 7}, {"leader", 8}
};

// 排序函數
bool compareCats(const Cat& a, const Cat& b)
     {
    // 根據地位優先級排序
    if (priority[a.position] != priority[b.
        position])
        return priority[a.position] <
            priority[b.position];

    // 同一地位時，按年齡排序
    if (a.position == "appentice") {
        // 對於 appentice，年齡小的優先
        if (a.age != b.age)
            return a.age < b.age;
    } else {
        // 對其他地位，年齡大的優先
        if (a.age != b.age)
            return a.age > b.age;
    }

    // 若地位和年齡相同，按名稱的字典序排序
    return a.name < b.name;
}

int main() {
    int N, M;
    // 讀取輸入，直到 EOF
    while (cin >> N >> M) {
        // 邊界檢查
        if (N <= 0 || M <= 0) {
            cout << "No valid data." << endl
                ;
            continue;
        }

        vector<Cat> cats(N);

        // 讀取每只貓的信息
        for (int i = 0; i < N; ++i) {
            cin >> cats[i].name >> cats[i].
                position >> cats[i].age;
            // 檢查地位是否在優先級映射中
            if (priority.find(cats[i].
                position) == priority.end())
                 {
                cerr << "Invalid position: "
                     << cats[i].position <<
                    endl;
                return 1;
            }
        }

        // 排序
        sort(cats.begin(), cats.end(),
             compareCats);

        // 輸出前 M 個貓的名字，避免 M 大於
            N 的情況
        for (int i = 0; i < min(M, N); ++i)
             {
            cout << cats[i].name << endl;
        }
    }

    return 0;
}
```

4.5 Problem H. Yet Another Alice and Bob

```cpp
#include <iostream>
#include <vector>
#include <set>
#include <tuple>
using namespace std;

struct Position {
    long long x, y;
    bool operator<(const Position& other)
        const {
        return tie(x, y) < tie(other.x,
            other.y);
    }
    bool operator==(const Position& other)
        const {
        return x == other.x && y == other.y;
    }
};

int main() {
    long long k;
    int n, m;
    cin >> k >> n >> m;

    vector<long long> aliceTurns(n),
        bobTurns(m);
    for (int i = 0; i < n; ++i) cin >>
        aliceTurns[i];
    for (int i = 0; i < m; ++i) cin >>
        bobTurns[i];

    set<Position> redCells, blueCells,
        purpleCells;
    Position alice = {1, 1}, bob = {1, 1};
    bool aliceRight = true, bobDown = true;
    int aliceTurnIdx = 0, bobTurnIdx = 0;

    // Process each second
    for (long long t = 1; t <= k; ++t) {
        // Check if Alice turns
        if (aliceTurnIdx < n && t ==
            aliceTurns[aliceTurnIdx]) {
            aliceRight = !aliceRight;
            aliceTurnIdx++;
        }

        // Check if Bob turns
        if (bobTurnIdx < m && t == bobTurns[
            bobTurnIdx]) {
            bobDown = !bobDown;
            bobTurnIdx++;
        }

        // Move Alice and Bob
        if (aliceRight) {
            alice.y++;
        } else {
            alice.x++;
        }

        if (bobDown) {
            bob.x++;
        } else {
            bob.y++;
        }

        // Skip (1,1) as it remains white
        if (alice.x == 1 && alice.y == 1)
            continue;
        if (bob.x == 1 && bob.y == 1)
            continue;

        // Update cell colors
        if (alice == bob) {
            // Remove cell from red and blue
                sets if it exists
            redCells.erase(alice);
            blueCells.erase(alice);
            purpleCells.insert(alice);
        } else {
            // Update Alice's position
            if (!(alice.x == 1 && alice.y ==
                1)) {
                if (!purpleCells.count(alice
                    )) {
                    blueCells.erase(alice);
                    redCells.insert(alice);
                }
            }

            // Update Bob's position
            if (!(bob.x == 1 && bob.y == 1))
                {
                if (!purpleCells.count(bob))
                     {
                    redCells.erase(bob);
                    blueCells.insert(bob);
                }
            }
        }
    }

    // Output the counts of red, blue, and
        purple cells
    cout << redCells.size() << " " <<
        blueCells.size() << " " <<
        purpleCells.size() << endl;

    return 0;
}

// 兩個人每秒走一格，過程中只會往右和往下
    走，問
// ⍰
//  秒之後他們在幾個格子相遇。

// 觀察：畫出兩個人的路徑，可以觀察到兩路徑
    相交的點就是他們相遇的點。

// 最 naive 的作法就是直接一秒一秒模擬兩人走
    路的過程，但是
// ⍰ ≤ 10 ^ 18
// ，這麼做會 TLE，所以我們需要一些優化。

// 想法一：不要每次只有一步。模擬時，一直走
    直到有人轉彎，此時檢查一下重疊狀況。

// 想法二：由於只會往右和往下移動，因此可以
    完全忽略自己左方和上方的路徑。因此模擬
    時，我們可以每次讓比較左邊的人移動。這樣
    一來，我們只需要紀錄每個人最後移動的橫線
    和直線。

// Time complexity:
// ⍰ ( ⍰ + ⍰ )
// 另外的解法：使用掃描線，配上一些資料結構，可以作到 □(□+□) log(□+□)
```

4.6 Queuing Problem

```
// #include <iostream>
// #include <vector>
// #include <algorithm>

// using namespace std;

// int where[1000005];
// vector<int> queue[1000005];

// int main() {
//     int n,m;
//     cin >> n >> m;
//     for (int i=1;i<=n;i++) {
//         where[i] = i;
//         queue[i].push_back(i);
//         //queue[i].push_back(3);
//     }
//     int ins,a,b;
//     for (int i=1;i<=m;i++) {
//         // for (int j=0;j<queue[i].size();j++) {
//         //     cout << queue[i][j];
//         // }
//         // cout << endl;
//         cin >> ins >> a >> b;
//         if (a == b) continue;
//         if (ins == 0) {
//             for (int j=0;j<queue[where[a]].size();j++) {
//                 if (queue[where[a]][j] == a) {
//                     queue[where[a]].erase(queue[where[a]].begin()+j);
//                     for (int k=0;k<queue[where[b]].size();k++) {
//                         if (queue[where[b]][k] == b) {
//                             queue[b].insert(queue[b].begin()+k+1, a);
//                         }
//                     }
//                     where[a] = where[b];
//                     break;
//                 }
//             }
//         }
//         else if (ins == 1) {
//             queue[b].insert(queue[b].end(), queue[a].begin(), queue[a].end());
//             for (int j=0;j<queue[a].size();j++) {
//                 where[queue[a][j]] = b;
//             }
//             queue[a].clear();
//         }
//     }
//     for (int i=1;i<=n;i++) {
//         cout << "#" << i << ":";
//         for(int j=0;j<queue[i].size();j++) {
//             cout << " " << queue[i][j];
//         }
//         cout << "\n";
//     }
// }
#include <iostream>
#include <vector>
#include <list>
#include <unordered_map>
using namespace std;

class QueueSystem {
private:
    vector<list<int>> queues;  // 每個隊列使用 list 儲存
    unordered_map<int, pair<int, list<int>::iterator>> personPosition; // 記錄每個人的當前位置和所屬隊列

public:
    QueueSystem(int n) {
        queues.resize(n + 1);  // 1-based indexing

        // 初始化：每個人都在自己的隊列中
        for (int i = 1; i <= n; i++) {
            queues[i].push_back(i);
            personPosition[i] = {i, queues[i].begin()}; // 記錄位置
        }
    }

    // 指令 0：將人 a 移動到人 b 後方
    void movePerson(int a, int b) {
        // 找到 a 的當前隊列並將其移除
        auto [fromQueue, posA] = personPosition[a];
        queues[fromQueue].erase(posA);

        // 找到 b 的當前隊列，並將 a 插入到 b 的後面
        int toQueue = personPosition[b].first;
        auto posB = personPosition[b].second;
        auto it = queues[toQueue].insert(next(posB), a);

        // 更新 a 的位置
        personPosition[a] = {toQueue, it};
    }

    // 指令 1：將隊列 a 合併到隊列 b
    void moveQueue(int a, int b) {
        if (a == b || queues[a].empty())
            return;

        // 將隊列 a 的所有人移動到隊列 b 的末尾
        auto &queueA = queues[a];
        auto &queueB = queues[b];

        // 將 a 的所有元素移動到 b 並更新每個人的位置
        queueB.splice(queueB.end(), queueA);
        for (auto it = queueB.begin(); it != queueB.end(); ++it) {
            personPosition[*it] = {b, it};
        }
    }

    // 輸出當前所有隊列的狀態
    void printQueues() {
        for (int i = 1; i < queues.size(); i++) {
            cout << "#" << i << ":";
            for (int person : queues[i]) {
                cout << " " << person;
            }
            cout << endl;
        }
    }
};

int main() {
    int n, m;
    cin >> n >> m;

    QueueSystem system(n);

    // 處理 m 條指令
    for (int i = 0; i < m; i++) {
        int t, a, b;
        cin >> t >> a >> b;

        if (t == 0) {
            system.movePerson(a, b);
        } else {
            system.moveQueue(a, b);
        }
    }

    system.printQueues();
    return 0;
}
```

4.7 Tasks and Deadlines

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    int n;
    cin >> n;

    vector<pair<int,int>> tasks(n);

    for (int i=0;i<n;i++) {
        cin >> tasks[i].first >> tasks[i].second ;
    }

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

    long long currentTime = 0;
    long long reward = 0;

    for (int i=0;i<n;i++) {
        currentTime += tasks[i].first;
        reward += (tasks[i].second - currentTime);
    }

    cout << reward << endl;
}

// #include <iostream>
// #include <vector>
// #include <algorithm>

// using namespace std;

// int main() {
//     int n;
//     cin >> n;

//     vector<pair<int, int>> tasks(n); // 存储任务的持续时间和截止时间

//     // 读取输入
//     for (int i = 0; i < n; i++) {
//         cin >> tasks[i].first >> tasks[i].second; // first 是持续时间 a，second 是截止时间 d
//     }

//     // 根据任务的持续时间 a 排序
//     sort(tasks.begin(), tasks.end());

//     long long currentTime = 0; // 当前完成时间
//     long long totalReward = 0; // 总奖励

//     // 处理每个任务
//     for (int i = 0; i < n; i++) {
//         currentTime += tasks[i].first; // 任务完成时间 = 前面的完成时间 + 当前任务的持续时间
//         totalReward += tasks[i].second - currentTime; // 奖励 = d - f (d 是截止时间，f 是完成时间)
//     }

//     cout << totalReward << endl;

//     return 0;
// }
```

5 w4

5.1 2022 Competitive Programming Training (I) Midterm Exam ProblemA Task and Penalty

```cpp
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>

using namespace std;
vector<int> a(1000005);

int main() {
    int n;
    cin >> n;
    for (int i=0;i<n;i++) {
        int tmp;
        cin >> tmp;
        a.at(tmp);
    }
    sort(a.begin(),a.end());
    int penalty = 0;
    int time = 0;
    for (int i=0;i<n;i++) {
        time += a[i];
        penalty += time;
    }
    cout << penalty << endl;
}
```

5.2 Array Division

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// Function to check if we can divide the array into k subarrays with a maximum sum of 'maxSum'
bool canDivide(const vector<int>& nums, int n, int k, long long maxSum) {
    int subarrayCount = 1;
    long long currentSum = 0;

    for (int num : nums) {
        if (currentSum + num > maxSum) {
            // Start a new subarray
            subarrayCount++;
            currentSum = num;
            if (subarrayCount > k) {
                return false;  // More subarrays than allowed
            }
        } else {
            currentSum += num;
        }
    }

    return true;
}

// Function to find the minimum possible maximum subarray sum
long long arrayDivision(const vector<int>& nums, int n, int k) {
    long long left = *max_element(nums.begin(), nums.end());  // Max element in the array
    long long right = 0;
    for (int num : nums) {
        right += num;  // Sum of all elements
    }

    while (left < right) {
        long long mid = left + (right - left) / 2;

        if (canDivide(nums, n, k, mid)) {
            right = mid;  // Try to find a smaller maximum subarray sum
        } else {
            left = mid + 1;  // Increase the maximum sum
        }
    }

    return left;  // This is the minimized maximum subarray sum
}

int main() {
    int n, k;
    cin >> n >> k;

    vector<int> nums(n);
    for (int i = 0; i < n; ++i) {
        cin >> nums[i];
    }

    // Output the minimized maximum subarray sum
    cout << arrayDivision(nums, n, k) << endl;

    return 0;
}
```

5.3 C. Sum of Three Values

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Element {
    int value;
    int index;
};

int main() {
    int n, x;
    cin >> n >> x; // 讀取數組大小和目標和

    vector<Element> arr(n); // 存儲數組的值和它們的索引
    for (int i = 0; i < n; ++i) {
        cin >> arr[i].value;
        arr[i].index = i + 1; // 原始位置的索引，從 1 開始
    }

    // 將數組按值進行排序
    sort(arr.begin(), arr.end(), [](const Element &a, const Element &b) {
        return a.value < b.value;
    });

    // 開始搜索三元組
    for (int i = 0; i < n - 2; ++i) {
        int target = x - arr[i].value; // 固定一個值，剩餘部分需要找到兩個數和為 target
        int left = i + 1, right = n - 1;

        while (left < right) {
            int sum = arr[left].value + arr[right].value;

            if (sum == target) {
                // 找到解決方案，輸出對應的索引
                cout << arr[i].index << " " << arr[left].index << " " << arr[right].index << endl;
                return 0;
            }
            else if (sum < target) {
                ++left; // 如果當前和小於目標，增加 left 指針
            } else {
                --right; // 如果當前和大於目標，減少 right 指針
            }
        }
    }

    // 如果找不到解決方案，輸出 "IMPOSSIBLE"
    cout << "IMPOSSIBLE" << endl;

    return 0;
}
```

5.4 Cart Racing

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
    long long S, T, A, B;
    cin >> S >> T >> A >> B;

    long long C, D, M;
    cin >> M >> C >> D;

    // Check if normal speed alone is enough to complete the track within time
    if (S <= A * T) {
        cout << "Yes" << endl;
        cout << (S + A - 1) / A << endl; // Minimum time needed at normal speed
        return 0;
    }

    // Initialize variables for simulation
    long long time = 0;
    long long distance_covered = 0;
    long long gas = M;

    while (time < T) {
        // Calculate max distance with current gas using high speed
        long long high_speed_time = min(gas / C, T - time);  // Time we can go at high speed with available gas
        distance_covered += high_speed_time * B;  // Distance covered at high speed
        gas -= high_speed_time * C;  // Gas consumed at high speed
        time += high_speed_time;

        // Check if we've completed the race
        if (distance_covered >= S) {
            cout << "Yes" << endl;
            cout << time << endl;
            return 0;
        }

        // Check if remaining time is enough to cover the remaining distance at normal speed
        long long remaining_distance = S - distance_covered;
        if (remaining_distance <= (T - time) * A) {
            cout << "Yes" << endl;
            cout << time + (remaining_distance + A - 1) / A << endl;
            return 0;
        }

        // Refill gas if time allows
        if (gas < C) {
            long long refill_time = (C - gas + D - 1) / D;
            if (time + refill_time >= T) {
                break;  // No time left to refill and use high speed
            }
            gas += refill_time * D;
            time += refill_time;
        }
    }

    // If we exit the loop without finishing, output "No" and the max reachable distance
    distance_covered += (T - time) * A;
    cout << "No" << endl;
    cout << distance_covered << endl;

    return 0;
}
```

5.5 Final Day

```
// only need to consider the best case
// if want to becocme rank k => then consider add "300" can be higher than score[k]
// if want to lower the rank => minus 300 can be lower than rank k (in the becoming rank k situation => )

#include <bits/stdc++.h>

using namespace std;
int n,k,num;
int p[100005];
int score[100005];
// bool ans[100005];

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> n >> k;
    for(int i = 0;i < n;i++)
    {
        int total = 0;
        for(int h = 0;h < 3;h++)
        {
            cin >> num;
            total += num;
        }
        p[i] = total;
        score[i] = total;// score rank
    }
    sort(score,score + n,greater<int>()); // big to small

    for(int i = 0;i < n;i++)
    {
        // if(i+1 <= k)
        // {
        //     if(p[i]-300 <= score[k-1]) cout << "Yes\n";
        //     else cout << "No\n";
        // }
        // else // need add score to be rank k
        // {
        //     if(p[i]+300 >= score[k-1]) cout << "Yes\n";
        //     else cout << "No\n";
        // }

        // can go in the top k rank?
        if(p[i] + 300 >= score[k-1]) cout << "Yes\n";
        else cout << "No\n";
    }
    return 0;
}
```

5.6 Points to Cost

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

const double eps = 1e-9;
const ll mxn = 200005;
ll x[mxn];
ll y[mxn];
ll n, c;

double f(double p) {
    double cost = 0;
    for (ll i = 0; i < n; i++) cost += (x[i] - p) * (x[i] - p);
    return cost;
}

int main() {
    (void)!scanf("%lld%lld", &n, &c);
    for (ll i = 0; i < n; i++) (void)!scanf("%lld%lld", &x[i], &y[i]);
    ll ycost = 0;
    for (ll i = 0; i < n; i++) ycost += (y[i] - c) * (y[i] - c);

    double l = *min_element(x, x + n), r = *max_element(x, x + n);
    while (r - l > eps) {
        double lm = l + (r - l) / 2;
        double rm = lm + (r - lm) / 2;
        if (f(lm) < f(rm)) r = rm;
        else l = lm;
    }
    printf("%0.15lf\n", f(l + (r - l) / 2) + ycost);
}
```

5.7 Subarray Sums

```
// #include <iostream>
// #include <unordered_map>
// #include <vector>

// using namespace std;

// int main() {
//     int n;
//     long long x;
//     cin >> n >> x;

//     vector<long long> a(n);
//     for (int i = 0; i < n; ++i) {
//         cin >> a[i];
//     }

//     unordered_map<long long, vector<int>> prefix_sums; // 用來記錄 prefix_sum 和對應的索引
//     prefix_sums[0].push_back(-1); // 處理從開頭到某一點的子數組情況，-1 表示從開頭開始

//     long long current_sum = 0;
//     long long count = 0;

//     for (int i = 0; i < n; ++i) {
//         current_sum += a[i];

//         // 檢查是否存在 current_sum - x
//         if (prefix_sums.find(current_sum - x) != prefix_sums.end()) {
//             // 找到符合條件的子數組，從每個滿足條件的 prefix_sum 出現的起始點開始
//             for (int start : prefix_sums[current_sum - x]) {
//                 // cout << "Subarray from index " << (start + 1) << " to " << i << " sums to " << x << endl;
//             }
//             count += prefix_sums[current_sum - x].size(); // 增加符合條件的子數組數量
//         }

//         // 更新 prefix_sums
//         prefix_sums[current_sum].push_back(i);
//     }
// // << "Total subarrays: "
//     cout  << count << endl;
//     return 0;
// }
#include <iostream>
#include <vector>
#include <string>

using namespace std;

vector<int> a(1000005);
vector<long long> prefix(1000005);

int main() {
    int n;
    long long sum;
    cin >> n >> sum;
    prefix[0] = 0;
    for (int i=1;i<=n;i++) {
        cin >> prefix[i];
        prefix[i] = prefix[i] + prefix[i-1];
        //cout << prefix[i] << " ";
    }
    int cnt = 0;
    for (int i=0;i<n;i++) {
        for (int j=i+1;j<=n;j++) {
            if (prefix[j] - prefix[i] == sum ) {
                cnt ++ ;
            }
        }
    }
    cout << cnt << endl;
}
```

5.8 Tian Ji horse racing

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

typedef long long ll;
const ll INF = 1e18;

struct Horse {
    ll base_speed;
    ll growth;

    // Calculate speed after M days
    ll speed_at_day(ll M) const {
        return base_speed + growth * M;
    }
};

// Check if it's possible to win K races after M days of training
bool can_win_k_races(const vector<Horse>& my_horses, const vector<ll>& opponent_horses,
                     int K, ll M) {
    int N = my_horses.size();

    // Calculate my horses' speeds after M days
    vector<ll> my_speeds(N);
    for (int i = 0; i < N; i++) {
        my_speeds[i] = my_horses[i].speed_at_day(M);
    }

    // Sort both sides' speeds in ascending order
    vector<ll> opp_speeds = opponent_horses;
    sort(my_speeds.begin(), my_speeds.end());
    sort(opp_speeds.begin(), opp_speeds.end());

    // Use two pointers to count wins
```

```cpp
    int i = 0, j = 0, wins = 0;

    while (i < N && j < N) {
        if (my_speeds[i] > opp_speeds[j]) {
            // If my horse can beat opponent's horse, count as a win
            wins++;
            j++;  // Move to next opponent's horse
        }
        i++;  // Move to next of my horses

        if (wins >= K) return true;  // Already achieved K wins
    }

    return wins >= K;
}

ll solve_test_case() {
    int N, K;
    cin >> N >> K;

    vector<Horse> my_horses(N);
    for (int i = 0; i < N; i++) {
        cin >> my_horses[i].base_speed >> my_horses[i].growth;
    }

    vector<ll> opponent_horses(N);
    for (int i = 0; i < N; i++) {
        cin >> opponent_horses[i];
    }

    // Binary search over days
    ll left = 0, right = 1e9;
    ll result = -1;

    while (left <= right) {
        ll mid = left + (right - left) / 2;

        if (can_win_k_races(my_horses, opponent_horses, K, mid)) {
            result = mid;
            right = mid - 1;  // Try to find a smaller number of days
        } else {
            left = mid + 1;
        }
    }

    return result;
}

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

    int T;
    cin >> T;

    while (T--) {
        cout << solve_test_case() << "\n";
    }

    return 0;
}
```

6 w5

6.1 Circular Sliding Window Maximum

```cpp
#include <iostream>
#include <vector>
#include <deque>

using namespace std;

vector<int> circularSlidingWindowMaximum(const vector<int>& arr, int N, int K) {
    vector<int> result(N);  // 儲存結果
    deque<int> dq;

    // 處理第一圈的窗口最大值
    for (int i = 0; i < N + K - 1; i++) {
        int idx = i % N;

        // 移除不在窗口範圍內的元素
        if (!dq.empty() && dq.front() <= i - K) {
            dq.pop_front();
        }

        // 移除隊列中小於當前元素的所有元素
        while (!dq.empty() && arr[dq.back() % N] <= arr[idx]) {
            dq.pop_back();
        }

        // 將當前索引加入隊列
        dq.push_back(i);

        // 記錄窗口最大值，從第 K - 1 個元素開始
        if (i >= K - 1 && i - K + 1 < N) {
            result[i - K + 1] = arr[dq.front() % N];
        }
    }

    return result;
}

int main() {
    int N, K;
    cin >> N >> K;

    vector<int> arr(N);
    for (int i = 0; i < N; i++) {
        cin >> arr[i];
    }

    vector<int> result = circularSlidingWindowMaximum(arr, N, K);
    for (int maxVal : result) {
        cout << maxVal << " ";
    }
    cout << endl;

    return 0;
}
```

6.2 Find the Medians

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <functional>

using namespace std;

vector<int> find_medians(const vector<int> & arr)
{
    int n = arr.size();
    vector<int> medians;

    // Max-heap to store the smaller half of numbers
    priority_queue<int> max_heap;
    // Min-heap to store the larger half of numbers
    priority_queue<int, vector<int>, greater<int>> min_heap;

    for (int i = 0; i < n; ++i)
    {
        int num = arr[i];

        // Insert the number into the appropriate heap
        if (max_heap.empty() || num <= max_heap.top())
        {
            max_heap.push(num);
        }
        else
        {
            min_heap.push(num);
        }

        // Balance the heaps: max_heap can only have at most one more element than min_heap
        if (max_heap.size() > min_heap.size() + 1)
        {
            min_heap.push(max_heap.top());
            max_heap.pop();
        }
        else if (min_heap.size() > max_heap.size())
        {
            max_heap.push(min_heap.top());
            min_heap.pop();
        }

        // Current median is the root of max_heap
        medians.push_back(max_heap.top());
    }

    return medians;
}

int main()
{
    int n;
    cin >> n;
    vector<int> arr(n);
    for (int i = 0; i < n; ++i)
    {
        cin >> arr[i];
    }

    // Get medians for each prefix
    vector<int> medians = find_medians(arr);

    // Print all medians in one line
    for (int median : medians)
    {
        cout << median << " ";
    }
    cout << endl;

    return 0;
}
```

6.3 Nearest Smaller Values

```cpp
#include <bits/stdc++.h>
using namespace std;
int a[200005];
int main() {
    int n; cin >> n;
    stack<int> stk;
    for (int i = 0; i < n; i++) {
        cin >> a[i];
        while (!stk.empty() && a[stk.top()] >= a[i]) stk.pop();
        if (stk.empty()) cout << 0;
        else cout << stk.top() + 1;
        cout << " \n"[i + 1 == n];
        stk.push(i);
    }
}
```

6.4 Nice Boat!

```cpp
#include <iostream>
#include <vector>
using namespace std;

// Function to check if a vector is safe
bool isSafe(const vector<int>& arr, int start, int len) {
    if (len == 0) return false;

    // Check prefix sums
    long long prefixSum = 0;  // Using long long to prevent overflow
    for (int i = start; i < start + len; i++) {
        prefixSum += arr[i];
        if (prefixSum < 0) return false;
    }

    // Check postfix sums
    long long postfixSum = 0;
    for (int i = start + len - 1; i >= start; i--) {
        postfixSum += arr[i];
        if (postfixSum < 0) return false;
    }

    return true;
}

// Function to find the longest safe subarray
int findLongestSafeSubarray(const vector<int>& arr) {
    int n = arr.size();
    int maxLen = 0;

    // Try all possible subarrays
    for (int i = 0; i < n; i++) {
        for (int len = 1; len <= n - i; len++) {
            // Check if current subarray is safe
            if (isSafe(arr, i, len)) {
                maxLen = max(maxLen, len);
            }
        }
    }

    return maxLen;
}

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

    int T;
    cin >> T;

    while (T--) {
        int N;
        cin >> N;

        vector<int> arr(N);
        for (int i = 0; i < N; i++) {
            cin >> arr[i];
        }

        cout << findLongestSafeSubarray(arr) << '\n';
    }

    return 0;
}
```

6.5 Paint the sticks

```cpp
#include <iostream>
#include <vector>
#include <stack>
using namespace std;

long long maxRectangle(vector<int> h)
{
    h.emplace_back(0); // Add a sentinel value to handle remaining heights
    stack<pair<int, int>> STK;
    long long ans = 0;
    for (int i = 0; i < (int)h.size(); ++i)
    {
        int corner = i;
        while (!STK.empty() && STK.top().first >= h[i])
        {
            corner = STK.top().second;
            ans = max(ans, 1LL * (i - corner) * STK.top().first);
            STK.pop();
        }
        STK.emplace(h[i], corner);
    }
    return ans;
}

int main()
{
    int n;
    while (cin >> n)
    {
        vector<int> heights(n);
        for (int i = 0; i < n; ++i)
        {
            cin >> heights[i];
        }
        cout << maxRectangle(heights) << endl;
    }
    return 0;
}
```

6.6 Sorting Queries

```cpp
#include <iostream>
#include <queue>
using namespace std;

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

    int Q;
    cin >> Q;

    queue<int> regular_queue;          // For elements in original order
    priority_queue<int, vector<int>, greater<int>> sorted_queue;  // For sorted elements

    while (Q--) {
        int type;
        cin >> type;

        if (type == 1) {
            int x;
            cin >> x;
            regular_queue.push(x);
        }
        else if (type == 2) {
            // If we have sorted elements, take from there first
            if (!sorted_queue.empty()) {
                cout << sorted_queue.top() << "\n";
                sorted_queue.pop();
            }
            // Otherwise take from regular queue
            else {
                cout << regular_queue.front() << "\n";
                regular_queue.pop();
            }
        }
        else { // type == 3
            // Move all elements from regular queue to sorted queue
            while (!regular_queue.empty()) {
                sorted_queue.push(regular_queue.front());
                regular_queue.pop();
            }
        }
    }

    return 0;
}
```

6.7 STring

```cpp
#include <iostream>
#include <string>
using namespace std;

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

    string X;
    cin >> X;

    // Count S and T separately
    int countS = 0;
    for (char c : X) {
        if (c == 'S') countS++;
    }
    int countT = X.length() - countS;

    // Keep track of unpaired S's we've seen
    int unpairedS = 0;
    int removedPairs = 0;

    // Process string from left to right
    for (char c : X) {
        if (c == 'S') {
            unpairedS++;
        } else { // c == 'T'
            if (unpairedS > 0) {
                // We can form an ST pair
                unpairedS--;
                removedPairs++;
            }
        }
    }

    // Calculate final length
    // Each removed pair reduces length by 2
    int finalLength = X.length() - (removedPairs * 2);

    cout << finalLength << endl;

    return 0;
}
```

7 w6

7.1 Counting Rooms

```cpp
#include <bits/stdc++.h>
using namespace std;
const int dx[] = {-1, 0, 1, 0};
const int dy[] = {0, 1, 0, -1};

int main() {
    int n, m; cin >> n >> m;
    vector<vector<char>> a(n + 2, vector<char>(m + 2, '#'));
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> a[i][j];
    function<void(int, int)> dfs = [&](int x, int y) {
        a[x][y] = '#';
        for (int k = 0; k < 4; k++) {
            int nx = x + dx[k], ny = y + dy[k];
            if (a[nx][ny] == '.') dfs(nx, ny);
        }
    };
    int ans = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            if (a[i][j] == '.') ans++, dfs(i, j);
    cout << ans << '\n';
}
```

7.2 Grid Maze

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <string>
#include <algorithm> // 添加此行以包含 reverse 函數

using namespace std;

struct Position {
    int x, y;
};

// 四個移動方向和對應的方向字符
const vector<Position> directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
const string dirChars = "UDLR";

bool isValid(int x, int y, int n, int m, const vector<vector<char>>& maze) {
    return x >= 0 && x < n && y >= 0 && y < m && maze[x][y] != '#';
}

int main() {
    int n, m;
    cin >> n >> m;

    vector<vector<char>> maze(n, vector<char>(m));
    Position start, end;

    // 讀取迷宮並找到起點和終點
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> maze[i][j];
            if (maze[i][j] == 'A') start = {i, j};
            if (maze[i][j] == 'B') end = {i, j};
        }
    }

    // BFS 初始化
    queue<Position> q;
    q.push(start);
    vector<vector<int>> steps(n, vector<int>(m, -1));
    vector<vector<int>> fromDirection(n, vector<int>(m, -1));
    steps[start.x][start.y] = 0;

    bool found = false;

    // BFS 遍歷
    while (!q.empty() && !found) {
        Position pos = q.front();
        q.pop();

        for (int d = 0; d < 4; d++) {
            int nx = pos.x + directions[d].x;
            int ny = pos.y + directions[d].y;

            if (isValid(nx, ny, n, m, maze) && steps[nx][ny] == -1) {
                steps[nx][ny] = steps[pos.x][pos.y] + 1;
                fromDirection[nx][ny] = d;
                q.push({nx, ny});

                // 若找到終點，結束搜尋
                if (nx == end.x && ny == end.y) {
                    found = true;
                    break;
                }
            }
        }
    }

    // 檢查結果
    if (steps[end.x][end.y] != -1) {
        cout << "YES\n";
        cout << steps[end.x][end.y] << '\n';

        // 回溯路徑
        string path;
        Position pos = end;
        while (pos.x != start.x || pos.y != start.y) {
            int d = fromDirection[pos.x][pos.y];
            path += dirChars[d];
            pos.x -= directions[d].x;
            pos.y -= directions[d].y;
        }

        reverse(path.begin(), path.end());
        cout << path << '\n';
    } else {
        cout << "NO\n";
    }

    return 0;
}
```

7.3 Monsters

```cpp
#include <algorithm>
#include <climits>
#include <cstring>
#include <iostream>
#include <queue>
#include <vector>
#define pii pair<int, int>
#define mn 1005
using namespace std;

int N, M;
queue<pii> q;
int paths[mn][mn];
pii from[mn][mn];
int oo = INT_MAX;
pii A;
string ans;
bool possible = false;

void retrace(pii node) {  // retrace from final node, adding direction from
                          // previous node to a string. This string will be
                          // backwards but will be reversed before output.
  pii origin = from[node.first][node.second];
  if (origin == pii(0, 0)) return;
  if (origin.first == node.first + 1) ans.push_back('U');
  if (origin.first == node.first - 1) ans.push_back('D');
  if (origin.second == node.second + 1) ans.push_back('L');
  if (origin.second == node.second - 1) ans.push_back('R');
  retrace(origin);
}
void check(pii origin,
           pii dest) {  // check if the considered destination may be traveled to
  int pl = paths[origin.first][origin.second];
  if (pl + 1 < paths[dest.first][dest.second]) {
    paths[dest.first][dest.second] = pl + 1;
    q.push(dest);
    from[dest.first][dest.second] = origin;
  }
}
bool mora = false;  // false if bfs for monsters, true if bfs for A
void bfs() {
  while (!q.empty()) {
    pii loc = q.front(), next;
    q.pop();
    next = loc;
    next.first++;
    check(loc, next);  // go through adjacent locations
    next = loc;
    next.first--;
    check(loc, next);
    next = loc;
    next.second++;
    check(loc, next);
    next = loc;
    next.second--;
    check(loc, next);
    if (mora &&
        (loc.first == 1 || loc.second == 1 || loc.first == N || loc.second == M)) {
      cout << "YES" << endl;
      cout << paths[loc.first][loc.second] << endl;
      retrace(loc);
      possible = true;
      return;
    }
  }
}
int main() {
  cin >> N >> M;
  for (int i = 1; i <= N; i++) {
    string s;
    cin >> s;
    for (int j = 1; j <= M; j++) {
      paths[i][j] = oo;
      if (s[j - 1] == '#') paths[i][j] = 0;
      if (s[j - 1] == 'M') {
        q.push(pii(i, j));
        paths[i][j] = 0;
      }
      if (s[j - 1] == 'A') {
        A.first = i;
        A.second = j;
      }
    }
  }
  bfs();                                 // monster bfs
  mora = true;                           // change next bfs to A bfs
  from[A.first][A.second] = pii(0, 0);  // give the retrace a terminating location
  paths[A.first][A.second] = 0;
  q.push(A);  // get ready for next bfs
  bfs();      // bfs with A
  if (possible) {
    reverse(ans.begin(), ans.end());
    cout << ans << endl;
  } else cout << "NO" << endl;
}
```

7.4 Problem A - Rush Hour Puzzle

```cpp
#include<bits/stdc++.h>
#define ll long long
#define maxn 2010
#define mod 998244353
using namespace std;
struct cv {
  int mp[6][6];
  int mov;
  friend bool operator<(cv p, cv q) {
    for (int i = 0; i < 6; i++) {
      for (int j = 0; j < 6; j++) {
        if (p.mp[i][j] != q.mp[i][j]) return p.mp[i][j] < q.mp[i][j];
      }
    }
    return p.mp[0][0] < q.mp[0][0];
  }
}dd, hf;
int to[4][2] = { 0,1,0,-1,1,0,-1,0 };
int main() {
  for (int i = 0; i < 6; i++) {
    for (int j = 0; j < 6; j++) {
      scanf("%d", &dd.mp[i][j]);
    }
  }
  dd.mov = 0;
  queue<cv>q;
  q.push(dd);
  set<cv>st;
  st.insert(dd);
  while (!q.empty()) {
    dd = q.front();
    q.pop();
    if (dd.mp[2][5] == 1) {
      int cnt = 1;
      for (int i = 4; i >= 0; i--) {
        if (dd.mp[2][i] != 1) break;
        cnt++;
      }
      int ans = (dd.mov + cnt);
      if (ans > 10) ans = -1;
      printf("%d\n", ans);
      return 0;
    }
    if (dd.mov == 10) break;
    for (int i = 0; i < 6; i++) {
      for (int j = 0; j < 6; j++) {
        if (dd.mp[i][j] == 0) {
          for (int k = 0; k < 4; k++) {
            int x = i + to[k][0], y = j + to[k][1];
            if (x < 0 || x >= 6 || y < 0 || y >= 6 || dd.mp[x][y] == 0) {
              continue;
            }
            int xx = x + to[k][0], yy = y + to[k][1];
            if (xx < 0 || xx >= 6 || yy < 0 || yy >= 6 || dd.mp[x][y] != dd.mp[xx][yy]) {
              continue;
            }
            while (xx >= 0 && xx < 6 && yy >= 0 && yy < 6 && dd.mp[x][y] == dd.mp[xx][yy]) {
              xx = xx + to[k][0], yy = yy + to[k][1];
            }
            xx = xx - to[k][0], yy = yy - to[k][1];
            hf = dd;
            hf.mov++;
            swap(hf.mp[i][j], hf.mp[xx][yy]);
            if (!st.count(hf)) {
              st.insert(hf);
              q.push(hf);
            }
          }
        }
      }
    }
  }
  printf("-1\n");
  return 0;
}
```

7.5 Rubik 2^3

```cpp
// ------------------------------------------
// SLPC2009 - Rubik 2^3 solution, verifier, and generator all-in-one :)
// Run "rubik2 < (input)" to solve, "rubik2 -g" to generate tests, or
//     "rubik2 (output) < (input)" to verify output against input
//
// Author:  Sonny Chan
// Date:    September 29, 2009
// ------------------------------------------

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <set>
#include <ctime>

using namespace std;

int g_wrap[][8] = {
    { -1, -1,  0,  1, -1, -1, -1, -1 },
    { -1, -1,  3,  2, -1, -1, -1, -1 },
    {  4,  5,  8,  9, 12, 13, 16, 17 },
    {  7,  6, 11, 10, 15, 14, 19, 18 },
    { -1, -1, 20, 21, -1, -1, -1, -1 },
    { -1, -1, 23, 22, -1, -1, -1, -1 }
};

int g_turn[3][24] = {
    { 0, 19, 16, 3, 4, 5, 6, 7, 8, 1, 2, 11, 13, 14, 15, 12, 22, 17, 18, 21, 20, 9, 10, 23 },
    { 1, 2, 3, 0, 16, 17, 6, 7, 4, 5, 10, 11, 8, 9, 14, 15, 12, 13, 18, 19, 20, 21, 22, 23 },
    { 0, 1, 15, 12, 4, 2, 3, 7, 9, 10, 11, 8, 21, 13, 14, 20, 16, 17, 18, 19, 5, 6, 22, 23 }
};

const string g_moves[] = { "X", "Y", "Z", "XXX", "YYY", "ZZZ"};
const string g_base[2] = {
    "WWWWBBBBOOOOGGGGRRRRYYYY",
    "RRRRGGGGBBBBYYYYWWWWOOOO"
};
const string g_dots = "........................";

const int g_mode = 3;   // can also do a branch factor 6 search, but it's slower

set<string> g_seen;

// function to convert an internal representation into a "wrapper map"
void output(const string &s, ostream &stream)
{
    for (int i = 0; i < 6; ++i) {
        for (int j = 0; j < 8; ++j) {
            if (g_wrap[i][j] >= 0) stream << s[g_wrap[i][j]];
            else stream << '.';
        }
        stream << endl;
    }
}

string transform(int turn, const string &s)
{
    // apply the permutation on the tiles from s to form new ordering t
    string t = g_dots;
    if (turn < 3)
        for (int i = 0; i < 24; ++i) t[i] = s[g_turn[turn][i]];
    else
        for (int i = 0; i < 24; ++i) t[g_turn[turn-3][i]] = s[i];
    return t;
}

bool solved(const string &s)
{
    // check that each of the six faces have all matching colours
    for (int i = 0; i < 6; ++i) {
        int j = i*4;
        char colour = s[j];
        for (++j; j < (i+1)*4; ++j)
            if (s[j] != colour) return false;
    }
    return true;
}

string scramble(string s, int steps = 42)
{
    for (int i = 0; i < steps; ++i)
        s = transform(rand()%6, s);
    return s;
}

bool bdfs(int depth, int limit, const string &s, string moves)
{
    // cache the configuration so we don't search it more than once
    if (g_seen.find(s) != g_seen.end())
        return false;
    g_seen.insert(s);

    // hooray!
    if (solved(s)) {
        cout << moves << endl;
        return true;
    }

    // recurse, transforming the cube and deeping the level
    if (depth < limit) {
        int m = rand() % g_mode;
        for (int i = 0; i < g_mode; ++i, m = (m+1)%g_mode)
            if (bdfs(depth+1, limit, transform(m, s), moves+g_moves[m]))
                return true;
    }

    return false;
}

string readconfig(istream &in)
{
    // read in the map and create an initial configuration from it
    string input, config = g_dots;
    for (int i = 0; i < 6; ++i) {
        in >> input;
        for (int j = 0; j < 8; ++j)
            if (g_wrap[i][j] >= 0) config[g_wrap[i][j]] = input[j];
    }
    return config;
}

int main()
{
    istream &in = cin;
    srand(time(0));

    for (;;)
    {
        // read and check for sentinel
        string initial = readconfig(in);
        if (initial == g_dots) break;

        // perform a bounded depth-first search until we're done
        for (int d = 1; ; ++d) {
            g_seen.clear();
            if (bdfs(0, d, initial, ""))
                break;
            //cout << "depth " << d << " - configurations searched: " << g_seen.size() << endl;
        }
    }

    return 0;
}
```

7.6 Swap Game

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

unordered_set<string> visited;

// Defined globally to be used in process_swap
queue<pair<string, int>> q;
int moves;
```

```cpp
string curboard;

// Processing swapping the numbers in x, y positions
void process_swap(int x, int y) {
  swap(curboard[x], curboard[y]);
  // Check whether already visited this potential board
  if (visited.find(curboard) == visited.end()) {
    q.push({curboard, moves + 1});
    visited.insert(curboard);
  }
  // Restore to original board
  swap(curboard[x], curboard[y]);
}

int main() {
  string inp;
  // Rewriting the input as a string
  for (int i = 0; i < 9; i++) {
    int a;
    cin >> a;
    inp += to_string(a - 1);
  }

  q.push({inp, 0});
  while (!q.empty()) {
    tie(curboard, moves) = q.front();
    q.pop();
    if (curboard == "012345678") {
      cout << moves << endl;
      return 0;
    }

    // Horizontal swaps
    for (int i = 0; i < 9; i += 3) {
      process_swap(i, i + 1);
      process_swap(i + 1, i + 2);
    }
    // Vertical swaps
    for (int i = 0; i < 3; i++) {
      process_swap(i, i + 3);
      process_swap(i + 3, i + 6);
    }
  }
}
```

7.7 Water Jug Puzzle

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <set>
#include <tuple>
using namespace std;

// 狀態表示：每個水壺的水量
typedef vector<int> State;

// 檢查是否已經達到目標水量
bool reachedTarget(const State& state, int Q) {
    for (int water : state) {
        if (water == Q) return true;
    }
    return false;
}

// 獲取所有可能的下一步狀態
vector<State> getNextStates(const State& current, const vector<int>& capacities) {
    vector<State> nextStates;
    int n = capacities.size();

    // 每個水壺的裝滿和清空操作
    for (int i = 0; i < n; i++) {
        // 裝滿水壺 i
        State filled = current;
        filled[i] = capacities[i];
        nextStates.push_back(filled);

        // 清空水壺 i
        State emptied = current;
        emptied[i] = 0;
        nextStates.push_back(emptied);
    }

    // 倒水操作
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i != j && current[i] > 0 && current[j] < capacities[j]) {
                State transferred = current;
                int transferAmount = min(current[i], capacities[j] - current[j]);
                transferred[i] -= transferAmount;
                transferred[j] += transferAmount;
                nextStates.push_back(transferred);
            }
        }
    }

    return nextStates;
}

// 主函數：計算達到目標水量的最小步數
int minStepsToReachTarget(const vector<int>& capacities, int Q) {
    int n = capacities.size();
    State initial(n, 0); // 初始狀態：所有水壺都是空的
    queue<pair<State, int>> q; // 狀態隊列，儲存狀態和步數
    set<State> visited; // 訪問過的狀態

    q.push({initial, 0});
    visited.insert(initial);

    while (!q.empty()) {
        State current = q.front().first;
        int steps = q.front().second;
        q.pop();

        // 檢查是否已經達到目標水量
        if (reachedTarget(current, Q)) {
            return steps;
        }

        // 產生下一步的所有可能狀態
        for (const State& next : getNextStates(current, capacities)) {
            if (visited.find(next) == visited.end()) {
                visited.insert(next);
                q.push({next, steps + 1});
            }
        }
    }

    // 如果無法達到目標水量，返回 -1
    return -1;
}

int main() {
    int N, Q;
    cin >> N;
    vector<int> capacities(N);
    for (int i = 0; i < N; i++) {
        cin >> capacities[i];
    }
    cin >> Q;

    int result = minStepsToReachTarget(capacities, Q);
    cout << result << endl;

    return 0;
}
```

8 w7

8.1 Binomial Coefficients

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 1e6 + 5;
const int mod = 1e9 + 7;
ll f[N];
ll fp(ll a, ll b) {
    ll ret = 1;
    for (; b; b >>= 1, a = a * a % mod)
        if (b & 1) ret = ret * a % mod;
    return ret;
}

ll inv(ll x) {return fp(x, mod - 2);}

int main() {
    f[0] = 1;
    for (int i = 1; i <= 1e6; i++) f[i] = f[i - 1] * i % mod;
    int q; cin >> q;
    while (q--) {
        int a, b; cin >> a >> b;
        cout << f[a] * inv(f[b] * f[a - b] % mod) % mod << '\n';
    }
}
```

8.2 Common Divisors

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

const int MAX_VAL = 1000000;

int main() {
    int n;
    cin >> n;
    vector<int> arr(n);
    vector<int> freq(MAX_VAL + 1, 0);

    // 讀入數列並計算每個數的出現次數
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
        freq[arr[i]]++;
    }

    // 計算每個因數的倍數的出現次數
    for (int i = 1; i <= MAX_VAL; ++i) {
        for (int j = i + i; j <= MAX_VAL; j += i) {
            freq[i] += freq[j];
        }
    }

    // 找到最大的能被至少兩個數整除的因數
    for (int i = MAX_VAL; i >= 1; --i) {
        if (freq[i] > 1) {
            cout << i << endl;
            return 0;
        }
    }

    return 0;
}
```

8.3 Counting Coprime Pairs

```cpp
/*
Problem Name: Counting Coprime Pairs
Problem Link: https://cses.fi/problemset/task/2417
Author: Sachin Srivastava (mrsac7)
*/
```

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

#define int long long
#define endl '\n'

const int mxN = 1e6+6;
int spf[mxN];

void sieve() {
    spf[0] = 1;
    for (int i = 1; i < mxN; i++)
        spf[i] = i;
    for (int i = 2; i*i < mxN; i++) {
        if (spf[i] == i) {
            for (int j = i*i; j < mxN; j += i) {
                if (spf[j] == j)
                    spf[j] = i;
            }
        }
    }
}

int cnt[mxN];

signed main(){
    ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0);
    #ifdef LOCAL
    freopen("input.txt", "r" , stdin);
    freopen("output.txt", "w", stdout);
    #endif

    int n; cin>>n;
    int ans = 0;
    sieve();
    for (int i = 0; i < n; i++) {
        int x; cin>>x;
        vector<int> v;
        while (x > 1) {
            int y = spf[x];
            v.push_back(y);
            while (x % y == 0)
                x /= y;
        }
        int k = v.size();
        for (int s = 1; s < (1<<k); s++) {
            int p = 1;
            for (int j = 0; j < k; j++) {
                if (s>>j&1) {
                    p *= v[j];
                }
            }
            int sgn = -1;
            if (__builtin_popcount(s)&1) sgn = 1;
            ans += sgn*cnt[p];
            cnt[p]++;
        }
    }
    cout<<n*(n-1)/2 - ans;
}
```

8.4 Exponentiation II

```cpp
#include <iostream>
using namespace std;
using ll = long long;
const int MOD = 1000000007;

// 快速冪計算 a^b % mod
ll mod_exp(ll a, ll b, ll mod) {
    ll result = 1;
    while (b > 0) {
        if (b % 2 == 1) {
            result = (result * a) % mod;
        }
        a = (a * a) % mod;
        b /= 2;
    }
    return result;
}

// 主函數
int main() {
    int n;
    cin >> n;
    while (n--) {
        ll a, b, c;
        cin >> a >> b >> c;
        // 計算 b^c % (MOD-1)，因為 MOD 是質數
        ll exp = mod_exp(b, c, MOD - 1);
        // 計算 a^exp % MOD
        ll answer = mod_exp(a, exp, MOD);
        cout << answer << endl;
    }
    return 0;
}
```

8.5 Problem D. Candies

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef tuple<ll, ll, ll> tlll;

// 擴展歐幾里得算法，用於求解 ax + by = gcd(a, b)
tlll extgcd (ll a, ll b) {
    if (b == 0) return {a, 1, 0};
    auto [g, x, y] = extgcd(b, a % b);
    return {g, y, x - a / b * y};
}

int main () {
    ll t;
    cin >> t;
    while (t--) {
        ll n, m, x, y, vx, vy;
        cin >> n >> m >> x >> y >> vx >> vy;

        // 當 vx 為 0 時，只在 x 為 0 或 n 時可能進口袋
        if (vx == 0) {
            if (x == 0 || x == n)
                cout << x << " " << (vy == 1 ? m : 0) << endl;
            else
                cout << "-1" << endl;
            continue;
        }

        // 當 vy 為 0 時，只在 y 為 0 或 m 時可能進口袋
        if (vy == 0) {
            if (y == 0 || y == m)
                cout << (vx == 1 ? n : 0) << " " << y << endl;
            else
                cout << "-1" << endl;
            continue;
        }

        // 將速度方向為負的初始坐標映射到正方向
        bool ix = false, iy = false;
        if (vx == -1) {
            ix = true;
            x = n - x;
        }
        if (vy == -1) {
            iy = true;
            y = m - y;
        }

        // 使用擴展歐幾里得算法檢查是否存在解
        auto [g, a, b] = extgcd(n, m);
        if ((x - y) % g != 0) {
            cout << "-1" << endl;
            continue;
        }

        // 轉換到可行解
        ll n2 = n / g;
        ll m2 = m / g;
        ll k = (x - y) / g;
        a *= k;
        b *= k;

        // 根據反射確定最終結果
        if (a == 0) {
            a = m2;
            b = -(-n2);
        } else {
            a = (a % m2 + m2) % m2;
            b = -((x - y) - a * n) / m;
        }

        ll ansx = (a & 1) ? n : 0;
        ll ansy = (b & 1) ? m : 0;
        if (ix) ansx = n - ansx;
        if (iy) ansy = m - ansy;

        cout << ansx << " " << ansy << endl;
    }
    return 0;
}
```

8.6 Sum of Divisors

```cpp
#include <iostream>

using std::cout;
using std::endl;

const int MOD = 1e9 + 7;
const int TWO_MOD_INV = 500000004;

/** @return The sum of all numbers in [start, end] mod MOD. */
long long total_sum(long long start, long long end) {
  return ((((end - start + 1) % MOD) * ((start + end) % MOD) % MOD) * TWO_MOD_INV % MOD);
}

int main() {
  long long n;
  std::cin >> n;

  long long total = 0;
  long long at = 1;
  while (at <= n) {
    long long add_amt = n / at;  // Our divisor to process
    // The largest number that still has the same value of q
    long long last_same = n / add_amt;

    total = (total + add_amt * total_sum(at, last_same)) % MOD;
    at = last_same + 1;
  }

  cout << total << endl;
}
```

ACM ICPC Team Reference - Angry Crow Takes Flight!

Contents

ACM ICPC Judge Test - Angry Crow Takes Flight!

C++ Resource Test

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

namespace system_test {

const size_t KB = 1024;
const size_t MB = KB * 1024;
const size_t GB = MB * 1024;

size_t block_size, bound;
void stack_size_dfs(size_t depth = 1) {
  if (depth >= bound)
    return;
  int8_t ptr[block_size]; // 若無法編譯將 block_size 改成常數
  memset(ptr, 'a', block_size);
  cout << depth << endl;
  stack_size_dfs(depth + 1);
}

void stack_size_and_runtime_error(size_t block_size, size_t bound = 1024) {
  system_test::block_size = block_size;
  system_test::bound = bound;
  stack_size_dfs();
}

double speed(int iter_num) {
  const int block_size = 1024;
  volatile int A[block_size];
  auto begin = chrono::high_resolution_clock::now();
  while (iter_num--)
    for (int j = 0; j < block_size; ++j)
      A[j] += j;
  auto end = chrono::high_resolution_clock::now();
  chrono::duration<double> diff = end - begin;
  return diff.count();
}

void runtime_error_1() {
  // Segmentation fault
  int *ptr = nullptr;
  *(ptr + 7122) = 7122;
}

void runtime_error_2() {
  // Segmentation fault
  int *ptr = (int *)memset;
  *ptr = 7122;
}

void runtime_error_3() {
  // munmap_chunk(): invalid pointer
  int *ptr = (int *)memset;
  delete ptr;
}

void runtime_error_4() {
  // free(): invalid pointer
  int *ptr = new int[7122];
  ptr += 1;
  delete[] ptr;
}

void runtime_error_5() {
  // maybe illegal instruction
  int a = 7122, b = 0;
  cout << (a / b) << endl;
}

void runtime_error_6() {
  // floating point exception
  volatile int a = 7122, b = 0;
  cout << (a / b) << endl;
}

void runtime_error_7() {
  // call to abort.
  assert(false);
}

} // namespace system_test

#include <sys/resource.h>
void print_stack_limit() { // only work in Linux
  struct rlimit l;
  getrlimit(RLIMIT_STACK, &l);
  cout << "stack_size = " << l.rlim_cur << " byte" << endl;
}
```