**★★★★★**

**題組：Contest Volumes**

**題號：10021: Cube in the Labirint**

**解題者：邱柏翔**

**解題日期：2025年5月15日**

**題意：**

迷宮為 X 行 Y 列，每格可有垂直或水平牆阻擋。立方體從起點 (A, B) 出發，每次可向上下左右滾動一格，翻滾會改變六面朝向。目標是從起點走到終點 (C, D)，並且到達時立方體的底面與起始相同。若能達成，輸出最少步數，否則輸出 "No solution"。

**題意範例：**

Input: 1

 10 2 (大小)

 1 1(起點)

 10 1(終點)

 v

 4 2(垂直牆1)

 6 2(垂直牆2)

 9 2(垂直牆3)

 h

 4 1(水平牆)

Output: 11(最少步數)

**解法：**

使用三維BFS。與一般迷宮問題的BFS不同在於狀態設計。傳統迷宮題中，只需要記錄每次移動後的位置 (x, y)，而這題需要使用一個陣列face[6] 來紀錄立方體六個面的編號，每次滾動一格，就透過 roll() 函數更新 face[] 中的數值，模擬六面在空間中的對應關係變化。

在搜尋過程中，紀錄當前 cube 狀態的座標位置 (x, y)、底面數字 face[1]、正面數字 face[2]，並將此三項作為唯一狀態標記，BFS 從起點狀態開始，每次嘗試向四個方向滾動，若不超出邊界且不被牆阻擋，就執行 roll() 模擬翻滾後的新狀態，並記錄是否曾經拜訪過。如果新的狀態未曾出現，則加入隊列，並更新步數。如此不斷展開直到遇到符合目標條件（位置為終點，底面與起始一樣）的狀態，該步數即為最短解。

**解法範例：**

無

**討論：**

狀態：每格最多有 24 種狀態，總共有 X×Y×24 種。

時間複雜度：O(XY)

空間複雜度：O(XY) 若 X,Y 無限制為 O(N×M×24)

**程式：**

#include <stdio.h>

#include <string.h>

#include <queue>

#include <algorithm>

using namespace std;

const int dx[] = {0, 0, 1, -1};

const int dy[] = {1, -1, 0, 0};

int g[16][16][4];

int X, Y, A, B, C, D;

struct Cube {

 int x, y, face[6]; // face[6] = {bottom, front, top, back, left, right}

 Cube(int a = 0, int b = 0):

 x(a), y(b) {

 for (int i = 0; i < 6; i++)

 face[i] = i;

 }

 Cube roll(int dir) {

 Cube r = (\*this);

 int t;

 if (dir == 0) { // 向右滾動

 t = r.face[0];

 r.face[0] = r.face[5], r.face[5] = r.face[2], r.face[2] = r.face[4];

 r.face[4] = t;

 } else if (dir == 1) { // 向左滾動

 t = r.face[0];

 r.face[0] = r.face[4], r.face[4] = r.face[2], r.face[2] = r.face[5];

 r.face[5] = t;

 } else if (dir == 2) { // 向上滾動

 t = r.face[0];

 r.face[0] = r.face[3], r.face[3] = r.face[2], r.face[2] = r.face[1];

 r.face[1] = t;

 } else if (dir == 3) { // 向下滾動

 t = r.face[0];

 r.face[0] = r.face[1], r.face[1] = r.face[2], r.face[2] = r.face[3];

 r.face[3] = t;

 }

 r.x += dx[dir], r.y += dy[dir];

 return r;

 }

};

void bfs() {

 int dist[16][16][16][16] = {};

 queue<Cube> Q; // BFS queue

 Cube u(A, B), v; // 起始狀態

 Q.push(u), dist[u.x][u.y][u.face[0]][u.face[1]] = 0;

 while (!Q.empty()) {

 u = Q.front(), Q.pop();

 // 檢查是否到達終點且底面為原本的0

 if (u.x == C && u.y == D && u.face[0] == 0) {

 printf("%d\n", dist[C][D][0][u.face[1]]);

 return ;

 }

 for (int i = 0; i < 4; i++) {

 if (g[u.x][u.y][i] == 0) {

 v = u.roll(i);

 if (v.x <= 0 || v.x > X || v.y <= 0 || v.y > Y)

 continue;

 // 若此新狀態尚未被拜訪過

 if (dist[v.x][v.y][v.face[0]][v.face[1]] == 0) {

 dist[v.x][v.y][v.face[0]][v.face[1]] = dist[u.x][u.y][u.face[0]][u.face[1]] + 1;

 Q.push(v);

 }

 }

 }

 }

 puts("No solution");

}

int main() {

 int testcase;

 char s[1024];

 scanf("%d", &testcase);

 while (testcase--) {

 memset(g, 0, sizeof(g));

 scanf("%d %d %d %d %d %d", &X, &Y, &A, &B, &C, &D);

 while (getchar() != '\n');

 int kind;

 // 根據輸入建立牆的資訊：'v' 為垂直牆，'h' 為水平牆

 while (fgets(s, sizeof(s), stdin)) {

 s[strcspn(s, "\n")] = '\0';

 if (s[0] == '\0') break;

 if (s[0] == 'v') kind = 0;

 else if (s[0] == 'h') kind = 1;

 else {

 int x, y;

 sscanf(s, "%d %d", &x, &y);

 if (kind == 0) {

 g[x][y][2] = g[x+1][y][3] = 1;

 } else {

 g[x][y][0] = g[x][y+1][1] = 1;

 }

 }

 }

 bfs();

 if (testcase)

 puts("");

 }

 return 0;

}