Tên: Huỳnh Công Tính

MSSV: 3124411315

Github: congtinh06

Lớp: DCT124C6

BÀI THỰC HÀNH 06

**Câu1:**

**I. Ý TƯỞNG BÀI TOÁN**

Cho N hình tròn, mỗi hình tròn có tâm (x, y) và bán kính r.  
Người chơi có thể di chuyển giữa hai hình tròn nếu thỏa mãn điều kiện:

* Nếu khoảng cách giữa mép hai hình tròn ≤ 50 cm → có thể bước sang (loại 0).
* Nếu khoảng cách giữa mép hai hình tròn > 50 và ≤ 60 cm → có thể nhảy sang (loại 1).
* Các trường hợp khác: không thể di chuyển.

→ Mỗi hình tròn được xem là một đỉnh trong đồ thị.  
→ Các cạnh nối biểu diễn việc có thể “bước” hoặc “nhảy” giữa hai hình tròn.  
→ Mục tiêu: Tìm đường đi tốt nhất từ hình tròn S đến T sao cho:

1. Ít số lần nhảy nhất.
2. Nếu số lần nhảy bằng nhau → chọn đường đi qua ít hình tròn hơn.

**II. PHÂN TÍCH CHƯƠNG TRÌNH**

**1. Cấu trúc dữ liệu**

* Dùng struct Circle lưu tâm (x, y) và bán kính r của từng hình tròn.
* Dùng danh sách kề adj để lưu các hình tròn có thể đi được với nhau.  
  Mỗi cạnh gồm: số hiệu hình tròn và loại di chuyển (0: bước, 1: nhảy).

**2. Hàm tính khoảng cách**Hàm edgeDist(a, b) tính khoảng cách giữa mép hai hình tròn:  
√((x₁ - x₂)² + (y₁ - y₂)²) - (r₁ + r₂).  
Nếu ≤ 50 → bước, nếu ≤ 60 → nhảy.

**3. Nhập dữ liệu**

* Nhập N, S, T.
* Sau đó nhập (x, y, r) cho từng hình tròn.

**4. Tạo đồ thị**

* Duyệt qua mọi cặp (i, j):
  + Nếu khoảng cách ≤ 50 → thêm cạnh loại 0.
  + Nếu ≤ 60 → thêm cạnh loại 1.
* Cạnh được nối hai chiều.

**5. Tìm đường đi tốt nhất**

* Dùng Dijkstra để tìm đường đi có số lần nhảy ít nhất.
* Nếu số lần nhảy bằng nhau → chọn đường đi ngắn hơn.
* Mảng parent[] lưu đỉnh cha, moveType[] lưu loại di chuyển**.**

**6. Truy vết và in kết quả**

* Nếu không có đường đi → in 1.
* Nếu có:
  + In 0.
  + In số lần nhảy và số hình tròn đi qua.
  + In thứ tự các hình tròn và loại di chuyển (0: bước, 1: nhảy).

III. CODE:

#include <iostream>

#include <fstream>

#include <cmath>

#include <queue>

using namespace std;

struct *Circle* { double x, y, r; };

const int MAXN = 10005;

const int INF = 1e9;

int n, S, T;

*Circle* c[MAXN];

int adj[MAXN][MAXN]; // 0: không, 1: đi, 2: nhảy

int jumpCount[MAXN], stepCount[MAXN], pre[MAXN], moveType[MAXN];

bool inQueue[MAXN];

int main() {

*ifstream* fin("input.txt");

*ofstream* fout("output.txt");

fin >> n >> S >> T;

for (int i = 1; i <= n; i++)

fin >> c[i].x >> c[i].y >> c[i].r;

// Tạo ma trận kề

for (int i = 1; i <= n; i++) {

for (int j = i + 1; j <= n; j++) {

double d = sqrt((c[i].x - c[j].x)\*(c[i].x - c[j].x) + (c[i].y - c[j].y)\*(c[i].y - c[j].y));

double dis = d - (c[i].r + c[j].r);

if (dis <= 50) adj[i][j] = adj[j][i] = 1;

else if (dis <= 60) adj[i][j] = adj[j][i] = 2;

else adj[i][j] = adj[j][i] = 0;

}

}

for (int i = 1; i <= n; i++) {

jumpCount[i] = INF;

stepCount[i] = INF;

pre[i] = -1;

moveType[i] = -1;

inQueue[i] = false;

}

queue<int> q;

q.push(S);

jumpCount[S] = 0;

stepCount[S] = 0;

inQueue[S] = true;

while (!q.empty()) {

int u = q.front(); q.pop();

inQueue[u] = false;

for (int v = 1; v <= n; v++) {

if (adj[u][v] == 0) continue;

int addJump = (adj[u][v] == 2);

int newJump = jumpCount[u] + addJump;

int newStep = stepCount[u] + 1;

if (newJump < jumpCount[v] || (newJump == jumpCount[v] && newStep < stepCount[v])) {

jumpCount[v] = newJump;

stepCount[v] = newStep;

pre[v] = u;

moveType[v] = addJump;

if (!inQueue[v]) {

q.push(v);

inQueue[v] = true;

}

}

}

}

if (jumpCount[T] == INF) {

fout << 0;

return 0;

}

fout << 1 << "\n";

int path[MAXN], cnt = 0;

for (int cur = T; cur != -1; cur = pre[cur])

path[cnt++] = cur;

for (int i = cnt - 1; i > 0; i--)

fout << path[i] << " " << moveType[path[i - 1]] << "\n";

return 0;

}

-Input:

4 1 4

0 0 5

0 20 5

0 30 5

0 100 10

-Output:

1

1 0

3 1

**Câu 2:**

**I. Ý TƯỞNG BÀI TOÁN**

* Ta có một bảng kích thước N × M, mỗi ô chứa giá trị A[i][j].
* Chỉ được di chuyển qua các ô có giá trị khác 0, và chỉ đi kề cạnh (trên, dưới, trái, phải).
* Chi phí đường đi = tổng các giá trị trên các ô đi qua, không tính ô đầu tiên và ô cuối cùng.
* Cần tìm đường đi ngắn nhất (tổng chi phí nhỏ nhất) từ ô (XI, YI) đến (XJ, YJ).

→ Bài này là bài toán đường đi ngắn nhất trên lưới, có trọng số.  
→ Giải bằng thuật toán Dijkstra.

**II. PHÂN TÍCH CHƯƠNG TRÌNH**

**1. Cấu trúc dữ liệu**

struct Node {

int x, y, cost;

bool operator>(const Node &o) const { return cost > o.cost; }

};

Dùng struct Node để lưu tọa độ (x, y) và chi phí tạm thời (cost).

Toán tử > giúp ưu tiên chi phí nhỏ hơn khi dùng trong hàng đợi ưu tiên (priority\_queue).

**2. Nhập dữ liệu**

cin >> N >> M >> XI >> YI >> XJ >> YJ;

* Nhập kích thước bảng, và hai ô đầu – cuối.
* Sau đó nhập toàn bộ ma trận A[i][j].

Nếu **ô bắt đầu hoặc ô kết thúc có giá trị 0** → không thể đi, in 1 và dừng.

1. **Khởi tạo:**

vector<vector<int>> dist(N + 1, vector<int>(M + 1, 1e9));

vector<vector<pair<int,int>>> parent(N + 1, vector<pair<int,int>>(M + 1, {-1,-1}));

* dist[i][j]: chi phí ngắn nhất từ ô đầu đến ô (i, j).
* parent[i][j]: lưu **ô trước đó** để truy vết lại đường đi.
* Mảng dx, dy dùng để di chuyển 4 hướng.

1. **Thuật toán Dijkstra**

priority\_queue<Node, vector<Node>, greater<Node>> pq;

pq.push({XI, YI, 0});

dist[XI][YI] = 0;

1. Dùng **priority\_queue** để chọn ô có chi phí nhỏ nhất hiện tại.
2. Mỗi lần lấy ô (x, y), xét 4 ô kề (nx, ny):

* Nếu **vượt biên hoặc gặp ô 0** → bỏ qua.
* Tính **chi phí thêm**:

int add = (nx == XJ && ny == YJ) ? 0 : A[nx][ny];

→ Không tính ô cuối.

* Nếu nc < dist[nx][ny] → cập nhật chi phí và cha.

**5. Xuất kết quả**

* Nếu không có đường đi (dist[XJ][YJ] == 1e9) → in 1.
* Ngược lại:
  + In 1 (có đường đi).
  + In **độ dài ngắn nhất** = dist[XJ][YJ].
  + Truy ngược parent[][] từ ô đích về ô đầu để in ra toàn bộ đường đi.

-Code:

#include <iostream>

#include <fstream>

#include <queue>

using namespace std;

const int MAX = 105;

const int INF = 1e9;

int a[MAX][MAX], cost[MAX][MAX];

int preX[MAX][MAX], preY[MAX][MAX];

bool visited[MAX][MAX];

int n, m, xi, yi, xj, yj;

// 4 hướng di chuyển

int dx[4] = {-1, 1, 0, 0};

int dy[4] = {0, 0, -1, 1};

struct *Node* {

int x, y, w;

};

struct *cmp* {

bool operator()(*Node* *a*, *Node* *b*) {

return *a*.w > *b*.w;

}

};

int main() {

*ifstream* fin("input1.txt");

*ofstream* fout("output1.txt");

fin >> n >> m >> xi >> yi >> xj >> yj;

for (int i = 1; i <= n; i++)

for (int j = 1; j <= m; j++)

fin >> a[i][j];

// khởi tạo

for (int i = 1; i <= n; i++)

for (int j = 1; j <= m; j++) {

cost[i][j] = INF;

visited[i][j] = false;

preX[i][j] = preY[i][j] = -1;

}

priority\_queue<*Node*, vector<*Node*>, *cmp*> pq;

cost[xi][yi] = 0;

pq.push({xi, yi, 0});

while (!pq.empty()) {

*Node* u = pq.top(); pq.pop();

int x = u.x, y = u.y;

if (visited[x][y]) continue;

visited[x][y] = true;

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

int nx = x + dx[k];

int ny = y + dy[k];

if (nx < 1 || nx > n || ny < 1 || ny > m) continue;

if (a[nx][ny] == 0) continue;

int add = 0;

if (!(nx == xj && ny == yj)) add = a[nx][ny];

if (cost[x][y] + add < cost[nx][ny]) {

cost[nx][ny] = cost[x][y] + add;

preX[nx][ny] = x;

preY[nx][ny] = y;

pq.push({nx, ny, cost[nx][ny]});

}

}

}

// không có đường đi

if (cost[xj][yj] == INF) {

fout << 0;

return 0;

}

fout << 1 << "\n" << cost[xj][yj] << "\n";

// truy vết ngược đường đi

int pathX[MAX \* MAX], pathY[MAX \* MAX];

int len = 0;

int cx = xj, cy = yj;

while (cx != -1 && cy != -1) {

pathX[len] = cx;

pathY[len] = cy;

int px = preX[cx][cy];

int py = preY[cx][cy];

cx = px;

cy = py;

len++;

}

// in ngược lại

for (int i = len - 1; i >= 0; i--)

fout << pathX[i] << " " << pathY[i] << "\n";

return 0;

}

-Input:

3 4 1 1 3 4

1 2 3 4

0 3 0 2

1 2 3 4

-Ouput:

1

10

1 1

1 2

2 2

3 2

3 3

3 4