#include <iostream>

#include <vector>

#include <queue>

#include <cmath>

#include <climits>

using namespace std;

struct Point {

double x, y;

double distanceTo(const Point& other) const {

return sqrt(pow(x - other.x, 2) + pow(y - other.y, 2));

}

};

struct Edge {

int to;

double weight;

};

struct QueueEntry {

int vertex;

double distance;

bool operator>(const QueueEntry& other) const {

return distance > other.distance;

}

};

vector<double> dijkstra(const vector<Point>& points, const vector<vector<Edge>>& graph, int src) {

int V = points.size();

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

vector<double> distances(V, INT\_MAX);

distances[src] = 0;

pq.push({src, 0});

while (!pq.empty()) {

QueueEntry current = pq.top();

pq.pop();

int u = current.vertex;

for (const Edge& edge : graph[u]) {

int v = edge.to;

double weight = points[u].distanceTo(points[v]);

if (distances[v] > distances[u] + weight) {

distances[v] = distances[u] + weight;

pq.push({v, distances[v]});

}

}

}

return distances;

}

int main() {

vector<Point> points = {{0, 0}, {1, 1}, {2, 2}, {3, 3}};

vector<vector<Edge>> graph = {

{{1, 0}, {2, 0}},

{{2, 0}, {3, 0}},

{{3, 0}},

{}

};

vector<double> distances = dijkstra(points, graph, 0);

for (int i = 0; i < distances.size(); ++i) {

cout << "Distance from source to vertex " << i << " is " << distances[i] << endl;

}

return 0;

}