#include<math.h>

#define EPSILON 2

struct node {

int x\_coord;

int y\_coord;

struct node \*parent\_node;

struct node \*next\_in\_list;

} NODE;

struct tree\_nodes {

int no\_of\_nodes;

node \*list\_head;

} TREE\_NODES;

int dist\_between(NODE \*p1,NODE \*p2) {

x1 = p1->x\_coord;

x2 = p2->x\_coord;

y1 = p1->y\_coord;

y2 = p2->y\_coord;

dist = sqrt(pow((x1-x2),2)+pow((y1-y2),2));

dist = (int)dist;

return dist;

}

NODE\* find\_nearest\_node(TREE\_NODES \*tn, int new\_point\_x, int new\_point\_y) {

n = tn->no\_of\_nodes;

curr\_node = tn->list\_head;

NODE \*temp = (NODE\*)malloc(sizeof(NODE));

temp->x\_coord = new\_point\_x;

temp->y\_coord = new\_point\_y;

int min\_dist = dist\_between(curr\_node, temp);

NODE \*nearest\_node = curr\_node;

curr\_node = curr\_node->next\_in\_list;

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

if min\_dist>dist\_between(curr\_node, temp)

nearest\_node = curr\_node;

curr\_node = curr\_node->next\_in\_list;

return nearest\_node;

}

init\_tree(tn, start\_x, start\_y);

choose\_point(tn);

adjust\_len(int \*newnode, nearest) {

}

int path\_clear(NODE \*nearest, intnewnode, obst\_list) {

int x1 = nearest->x\_coord;

int y1 = nearest->y\_coord;

}

void add\_to\_tree(NODE \*nearest, int \*newnode, TREE \*tn, NODE \*added\_node) {

NODE \*temp = (NODE\*)malloc(sizeof(NODE));

temp->x\_coord = newnode[0];

temp->y\_coord = newnode[1];

temp->parent\_node = nearest;

temp->next\_in\_list = nearest->next\_in\_list;

nearest->next\_in\_list = temp;

added\_node = temp;

}

int not\_equal(NODE \*added\_node, int end\_x, int end\_y) {

return ((added\_node->x\_coord == end\_x)(added\_node->y\_coord == end\_y));

}

int main() {

int n\_cols;

int n\_rows;

scanf("%d", n\_cols);

scanf("%d", n\_rows);

int start\_x;

int start\_y;

scanf("%d", start\_x);

scanf("%d", start\_y);

int end\_x;

int end\_y;

scanf("%d", end\_x);

scanf("%d", end\_y);

int no\_obst;

scanf("%d", no\_obst);

int obst\_list[no\_obst][2];

for (int i=0; i<no\_obst; i++) {

scanf("%d", obst\_list[i][0]);

scanf("%d", obst\_list[i][1]);

}

TREE\_NODES \*tn;

init\_tree(tn, start\_x, start\_y);

int newnode[2];

NODE\* nearest;

int iter = 0;

NODE\* added\_node = NULL;

do {

newnode = choose\_point(tn);

nearest = find\_nearest\_node(tn, newnode[0], newnode[1]);

newnode = adjust\_len(newnode, nearest);

if (path\_clear(nearest, newnode, obst\_list))

add\_to\_tree(nearest, newnode, tn, added\_node);

iter+=1;

} while(not\_equal(added\_node, end\_x, end\_y) && iter<=10000);

return 0;

}

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define EPSILON 2

struct node {

int x\_coord;

int y\_coord;

struct node \*parent\_node;

struct node \*next\_in\_list;

} NODE;

struct tree\_nodes {

int no\_of\_nodes;

NODE \*list\_head;

} TREE\_NODES;

int dist\_between(NODE \*p1,NODE \*p2) {

x1 = p1->x\_coord;

x2 = p2->x\_coord;

y1 = p1->y\_coord;

y2 = p2->y\_coord;

dist = sqrt(pow((x1-x2),2)+pow((y1-y2),2));

dist = (int)dist;

return dist;

}

NODE\* find\_nearest\_node(TREE\_NODES \*tn, int new\_point\_x, int new\_point\_y) {

n = tn->no\_of\_nodes;

curr\_node = tn->list\_head;

NODE \*temp = (NODE\*)malloc(sizeof(NODE));

temp->x\_coord = new\_point\_x;

temp->y\_coord = new\_point\_y;

int min\_dist = dist\_between(curr\_node, temp);

NODE \*nearest\_node = curr\_node;

curr\_node = curr\_node->next\_in\_list;

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

if min\_dist>dist\_between(curr\_node, temp)

nearest\_node = curr\_node;

curr\_node = curr\_node->next\_in\_list;

}

return nearest\_node;

}

void init\_tree(TREE\_NODES \*tn, int \*start\_x, int \*start\_y) {

NODE \*temp = (NODE\*)malloc(sizeof(NODE));

temp->x\_coord = start\_x;

temp->y\_coord = start\_y;

temp->parent\_node = NULL;

temp->next\_in\_list = NULL;

tn->no\_of\_nodes = 1;

tn->list\_head = temp;

}

int choose\_point(int n\_cols, int n\_rows, TREE\_NODES \*tn, int \*newnode) {

int x, y, iter = 0;

int isnotallowed = 1;

while (isnotallowed) {

iter++;

if (iter>10000)

return 1;

isnotallowed = 0

x = rand() % (n\_cols+1);

y = rand() % (n\_rows+1);

curr\_node = tn->list\_head;

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

if is\_equal(curr\_node, x, y) {

isnotallowed = 1;

break;

}

curr\_node = curr\_node->next\_in\_list;

}

}

newnode[0] = x;

newnode[1] = y;

return 0;

}

void adjust\_len(int \*newnode, nearest, int \*x, int \*y) { // \*\*CHANGE\*\*

NODE \*temp = (NODE\*)malloc(sizeof(NODE));

temp->x\_coord = newnode[0];

temp->y\_coord = newnode[1];

if (dist\_between(nearest, temp)<= EPSILON)

return;

return;

}

int path\_clear(NODE \*nearest, int newnode, int no\_obst, int obst\_list[no\_obst][2]) {

int x1 = nearest->x\_coord;

int y1 = nearest->y\_coord;

int x2 = newnode[0];

int y2 = newnode[1];

int yt = y1-y2;

int xt = x1-x2;

double coeff = yt/xt;

double lhs,rhs;

for (int i=0; i<no\_obst; i++) {

lhs = obst\_list[i][0];

rhs = coeff\*obst\_list[i][1];

if ((lhs<=rhs+0.01)&&(lhs>=rhs-0.01))

return 0;

}

return 1;

}

void add\_to\_tree(NODE \*nearest, int \*newnode, TREE \*tn, NODE \*added\_node) {

NODE \*temp = (NODE\*)malloc(sizeof(NODE));

temp->x\_coord = newnode[0];

temp->y\_coord = newnode[1];

temp->parent\_node = nearest;

temp->next\_in\_list = nearest->next\_in\_list;

nearest->next\_in\_list = temp;

added\_node = temp;

tn->no\_of\_nodes += 1;

}

int is\_equal(NODE \*added\_node, int end\_x, int end\_y) {

return ((added\_node->x\_coord == end\_x)&&(added\_node->y\_coord == end\_y));

}

int main() {

int n\_cols;

int n\_rows;

scanf("%d", &n\_cols);

scanf("%d", &n\_rows);

int start\_x;

int start\_y;

scanf("%d", &start\_x);

scanf("%d", &start\_y);

int end\_x;

int end\_y;

scanf("%d", &end\_x);

scanf("%d", &end\_y);

int no\_obst;

scanf("%d", &no\_obst);

int obst\_list[no\_obst][2];

for (int i=0; i<no\_obst; i++) {

scanf("%d", &(obst\_list[i][0]));

scanf("%d", &(obst\_list[i][1]));

}

TREE\_NODES \*tn;

init\_tree(tn, start\_x, start\_y);

int newnode[2];

NODE\* nearest;

int iter = 0;

NODE\* added\_node = NULL;

int failure = 0;

int x;

int y;

do {

failure = retchoose\_point(n\_cols, n\_rows, tn, newnode);

if (failure) {

printf("DOWN");

return 0;

}

nearest = find\_nearest\_node(tn, newnode[0], newnode[1]);

adjust\_len(newnode, nearest, &x, &y);

newnode[0] = x;

newnode[1] = y;

if (path\_clear(nearest, newnode, no\_obst, obst\_list))

add\_to\_tree(nearest, newnode, tn, added\_node);

iter+=1;

} while(!(is\_equal(added\_node, end\_x, end\_y)) && iter<=10000);

return 0;

}