Faculty of Engineering and Applied Science

SOFE 377U Design and Analysis of Algorithms

Assignment 1

Group Member 1

Name: Dhanushga Lionel

Student ID: 100616831

Group Member 2

Name: Thanushan Rameswaran

Student ID: 100462146

Date: 10/13/2018

Opening Comments:

For this assignment we used our UOIT issued laptops, running Windows 10. Our code is written in C++, compiled with Visual Studio.

Pseudocode:

```
Create vector graph and distance
Enter heights of buildings into graph vector

Create a que myQue
Mark v as visited and put v into myQue
While Q is non-empty
        For (all the columns in the grid)
                For (all the rows in the grid)
                //calculate vhsquare
                Vhsquare = velocity required to make jump to current building of height vh
                If (jump is possible) enter building coordinates into que
        Remove the head u of myQue
        Mark and enqueue all (unvisited) neighbours of u
For (all columns in city grid)
        For (all rows in city grid)
        If (building can be jumped to)
                Print(number of jumps required)
        Else
                Print 'X'
```

Big-O Analysis:

Iterativley checking which buildings the trajectory passes by takes O(ax*ay). Checking for all pairs of buildings take a time of O((a(x^2))*(a*y^2))). Therefore this leads to a O(((a(x^2))*(a*y^2)))*(ax*ay)) which is equivalent to O((a(x^3))*(a*y^3))).

Code:

```
#include <cmath>
```

```
#include <cstring>
#include <tuple>
#include <vector>
#include <algorithm>
#include <iostream>
#include <queue>
#include <iomanip>
#include <math.h>
using namespace std;
#define G 9.80665
#define EPS 1e-6
#define _USE_MATH_DEFINES
#define PI 3.14159265358

bool Check(double vd, double vh, double dx, double dy, double x, double y1, double y2, double
bhnext)
{
	if (x < 0.0 || x > dx) return false;
	if (dx*y1 > x*dy) return false;  // exact
	if (dx*y2 < x*dy) return false;  // exact
	double d = hypot(x, x*dy / dx);
	double t = d / vd;
	double h = vh * t - G / 2.0 * t*t;
	return h < bhnext + EPS;
}
int main()
{
	double w, v;
	int dx, dy, lx, ly;
	while (cin >> dx >> dy >> w >> v >> lx >> ly)
	{
		vector <vector<int>> grid(dy + 1, vector<int>(dx + 1));
		vector <vector <int>> distance(dy + 1, vector<int>(dx + 1, 1000000000));
		//Getting the grid
		for (int y = 1; y <= dy; y++)
		{
			for (int x = 1; x <= dx; x++)
			{
				cin >> grid[y][x];//getting Error Here with the Java Code
			}
		}

		//Store the queue
```

```
		vector <pair<int, int>> MyQueue{ {lx,ly} };
		int current = 0;
		distance[ly][lx] = 0;
		while (!MyQueue.empty())
		{
			vector <pair<int, int>> MyQueue2;
			current++;
			for (auto CValue : MyQueue)
			{
				int x, y;
				tie(x, y) = CValue;
				for (int x2 = 1; x2 <= dx; x2++)
				{
					for (int y2 = 1; y2 <= dy; y++)
					{
						//See if you need this
						double d = (w * ((x2 - x)*(x2 - x)) + (y2 - y)*(y2 - y));
						double h = grid[y2][x2] - grid[y][x];
						double a = h * h + d * d;
						double b = -2 * h*h*v*v - G * d*d*h - d * d*v*v;
						double c = h * h*v*v*v*v + G * d*d*h*v*v + G *
G*d*d*d*d / 4;

						double discrimant = b * b - 4 * a*c;
						if (discrimant < -1e-6)
						{
							continue;
						}
						double VHSqr = (-b + sqrt(discrimant)) / 2 /
a;//Change Name

						if (VHSqr < 1e-6 || VHSqr > v*v - 1e-6)
						{
							continue;
						}
						double vd = sqrt(v*v - VHSqr), VH = sqrt(VHSqr);

						bool result = false;
						for (int x3 = min(x, x2); !result && x3 <= max(x, x2);
x3++)

						{
							for (int y3 = min(y, y2); !result && y3 <=
max(y, y2); y3++)

							{
```

```
				result |= Check(vd / w, VH, x2 - x, y2 - y, x3 + 0.5 - x, y3 - 0.5 - y, y3 + 0.5 - y, grid[y3][x3] - grid[y][x]);
				result |= Check(vd / w, VH, x2 - x, y2 - y, x3 - 0.5 - x, y3 - 0.5 - y, y3 + 0.5 - y, grid[y3][x3] - grid[y][x]);
				result |= Check(vd / w, VH, y2 - y, x2 - x, y3 + 0.5 - y, x3 - 0.5 - x, x3 + 0.5 - x, grid[y3][x3] - grid[y][x]);
				result |= Check(vd / w, VH, y2 - y, x2 - x, y3 - 0.5 - y, x3 - 0.5 - x, x3 + 0.5 - x, grid[y3][x3] - grid[y][x]);
			}
			if (result)
			{
				continue;
			}
			MyQueue2.push_back({ x2, y2 });
			distance[y2][x2];
		}
		/*
		double maxAngle = tan(v*v / (G*d));//max angle to get most height and distance
		double factor = h / d;
		if (tan(maxAngle) - G * d / (2 * v*v*cos(maxAngle)* cos(maxAngle)) < factor)// max jump doesn't get there
		{
			return false;
		}
		double low = maxAngle, high = PI / 2; //Binary search tree for finding jumping angle
		for (int i = 0; i < 100; i++)
		{
			double mid = (low + high) / 2;
			double rhs = tan(mid) - G * d / (2 * v*v*cos(mid)*cos(mid));//Change RHS Variable Name
			if (rhs > factor)//adjust low or high
			{
				low = mid;
			}
			else
			{
				high = mid;
			}
		}
		*/
		bool result = false;
```

```
                        for (int x3 = min(x, x2); !result && x3 <= max(x, x2);
x3++)
                        {
                            for (int y3 = min(y, y2); !result && y3 <=
max(y, y2); y3++)
                            {
                                result |= Check(vd / w, VH, x2 - x, y2
- y, x3 + 0.5 - x, y3 - 0.5 - y, y3 + 0.5 - y, grid[y3][x3] - grid[y][x]);
                                result |= Check(vd / w, VH, x2 - x, y2
- y, x3 - 0.5 - x, y3 - 0.5 - y, y3 + 0.5 - y, grid[y3][x3] - grid[y][x]);
                                result |= Check(vd / w, VH, y2 - y, x2
- x, y3 + 0.5 - y, x3 - 0.5 - x, x3 + 0.5 - x, grid[y3][x3] - grid[y][x]);
                                result |= Check(vd / w, VH, y2 - y, x2
- x, y3 - 0.5 - y, x3 - 0.5 - x, x3 + 0.5 - x, grid[y3][x3] - grid[y][x]);
                            }
                            if (result)
                            {
                                continue;
                            }
                            MyQueue2.push_back({ x2, y2 });
                            distance[y2][x2];
                        }
                        /*
                        double Theta = low; //Assigning jumping
angle//change variable name
                        int dx = x2 - lx;
                        int dy = y2 - ly;
                        if (dx != 0)
                        {
                            int StepX = dx / abs(dx);
                            int x3 = lx + StepX;
                            double y3 = ly + 0.5 + (y2 - ly) /
(2.0*abs(dx));

                            for (int i = 0; i < abs(dx); i++)
                            {
                                int ThisY = int(y3 + 1e-9);
                                int MustReach = grid[x3][ThisY];
                                MustReach = max(MustReach,
grid[x3 - StepX][ThisY]);

                                //Borderline case to reach height
                                if (y3 - ThisY < 1e-8)
                                {
```

```
                                                                                MustReach =
max(MustReach, grid[x3][ThisY - 1]);
                                                                                MustReach =
max(MustReach, grid[x3 - StepX][ThisY - 1]);
                                                                        }
                                                                        //Calculate horizontal and vertical
Movemen
                                                                        double HMove = w * sqrt(i + 0.5)*(i +
0.5) + (y3 - Iy - 0.5)*(y3 - Iy - 0.5);
                                                                        double T = HMove /
(v*cos(Theta));//Think this is Time value Change variable to TIme
                                                                        double MyH = v * sin(Theta)* T -
0.5*G*T*T;
                                                                        //If vertical not made
                                                                        if (MyH < MustReach, grid[Ix][Iy] +
1e-10);
                                                                        {
                                                                                return false;
                                                                        }
                                                                        x3 += StepX;
                                                                        y3 += (y2 - Iy) / (1.0 *abs(dy));
                                                                }
                                                        }
                                                        */
                                                        }
                                        }
                                        MyQueue.swap(MyQueue2);
                                }

                        }
                        for (int y = 1; y < dy; y++)
                        {
                                for (int x = 1; x < dx; x++)
                                {
                                        if (x > 1)
                                        {
                                                cout << ' ';
                                        }
                                        if (distance[y][x] > dx*dy)
                                        {
                                                cout << "X";
                                        }
                                        else
```

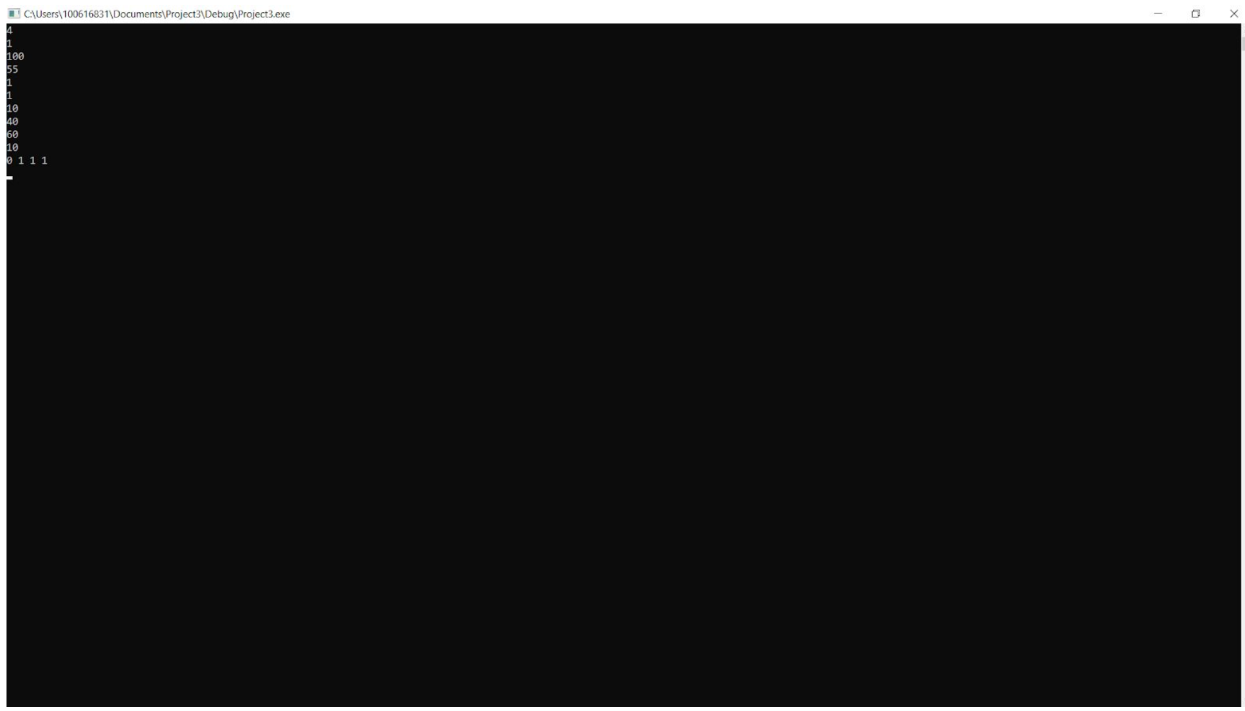
```
                                        {
                                                cout << distance[y][x];
                                        }
                                        cout << endl;
                                }
                        }
                }
}
                                                        /*
                                                        if (dy != 0)
                                                        {
                                                                int StepY = dy / abs(dy);
                                                                int y3 = ly + StepY;
                                                                double x3 = lx + 0.5 + (x2 - lx) /
(2.0*abs(dy));
                                                                for (int i = 0; i < abs(dy); i++)
                                                                {
                                                                        int ThisX = int(x3 + 1e-9);
                                                                        int MustReach = grid[ThisX][y3];//We
must reach building this high
                                                                        MustReach = max(MustReach,
grid[ThisX][y3 - StepY]);
                                                                        if (x - ThisX < 1e-8)//Another
Borderline case
                                                                        {
                                                                                MustReach =
max(MustReach, grid[ThisX - 1][y3]);
                                                                                MustReach =
max(MustReach, grid[ThisX - 1][y3 - StepY]);
                                                                        }
                                                                        //Calculating our horizontal and
vertical positions
                                                                        double HMove = w * sqrt((i + 0.5)*(i
+ 0.5) + (x3 - lx - 0.5)*(x3 - lx - 0.5));
                                                                        double T = HMove /
(v*cos(Theta));//change T variable to Time Variable
                                                                        double MyH = v * sin(Theta) * T - 0.5
* G*T*T;
                                                                        //Did not make it high enough
                                                                        if (MyH < MustReach, grid[lx][ly] =
1e-10)
                                                                        {
                                                                                return false;
```

```
                                        }
                                        x3 += (x2 - lx) / (1.0*abs(dy));
                                        y3 += StepY;

                                }
                        }
                        return true;
                        //double disc = b * b - 4 * a*c;
                        for (int j = 0; j < dy; j++)
                        {
                                for (int i = 0; i < dx; i++)
                                {
                                        if (x > 1)
                                        {
                                                cout << ' ';
                                        }
                                        if (distance[y][x] > dx*dy)
                                        {
                                                cout << "X";
                                        }
                                        else
                                        {
                                                cout << distance[y][x];
                                        }
                                }

        if (SDL_Init(SDL_INIT_VIDEO) == 0) {
                SDL_Window* window = NULL;
                SDL_Renderer* renderer = NULL;

                if (SDL_CreateWindowAndRenderer(1080, 720, 0, &window, &renderer) == 0) {
                        SDL_bool done = SDL_FALSE;

                        while (!done) {
                                SDL_Event event;

                                SDL_SetRenderDrawColor(renderer, 0, 0, 0,
SDL_ALPHA_OPAQUE);
                                SDL_RenderClear(renderer);

                                SDL_SetRenderDrawColor(renderer, 255, 255, 255,
SDL_ALPHA_OPAQUE);
```

```
            for (int y = 1; y <= dyi; y++) {
                for (int x = 1; x <= dxi; x++) {
                    SDL_RenderDrawLine(renderer, (x - 1)*w, (y -
1)*w, x*w, (y-1)*w);
                    SDL_RenderDrawLine(renderer, (x - 1)*w, (y -
1)*w, (x - 1)*w, y*w);
                    SDL_RenderDrawLine(renderer, (x - 1)*w, (y)*w,
(x)*w, y*w);
                    SDL_RenderDrawLine(renderer, (x)*w, (y - 1)*w,
(x)*w, y*w);


                }
            }



            SDL_RenderPresent(renderer);

            while (SDL_PollEvent(&event)) {
                if (event.type == SDL_QUIT) {
                    done = SDL_TRUE;
                }
            }
        }
    }

    if (renderer) {
        SDL_DestroyRenderer(renderer);
    }
    if (window) {
        SDL_DestroyWindow(window);
    }
}
SDL_Quit();
return 0;



                    }*/
```

Output:

Case 1:
Input:
4 1 100 55 1 1
10 40 60 10
Output: 0 1 1 1
Run time: .001s

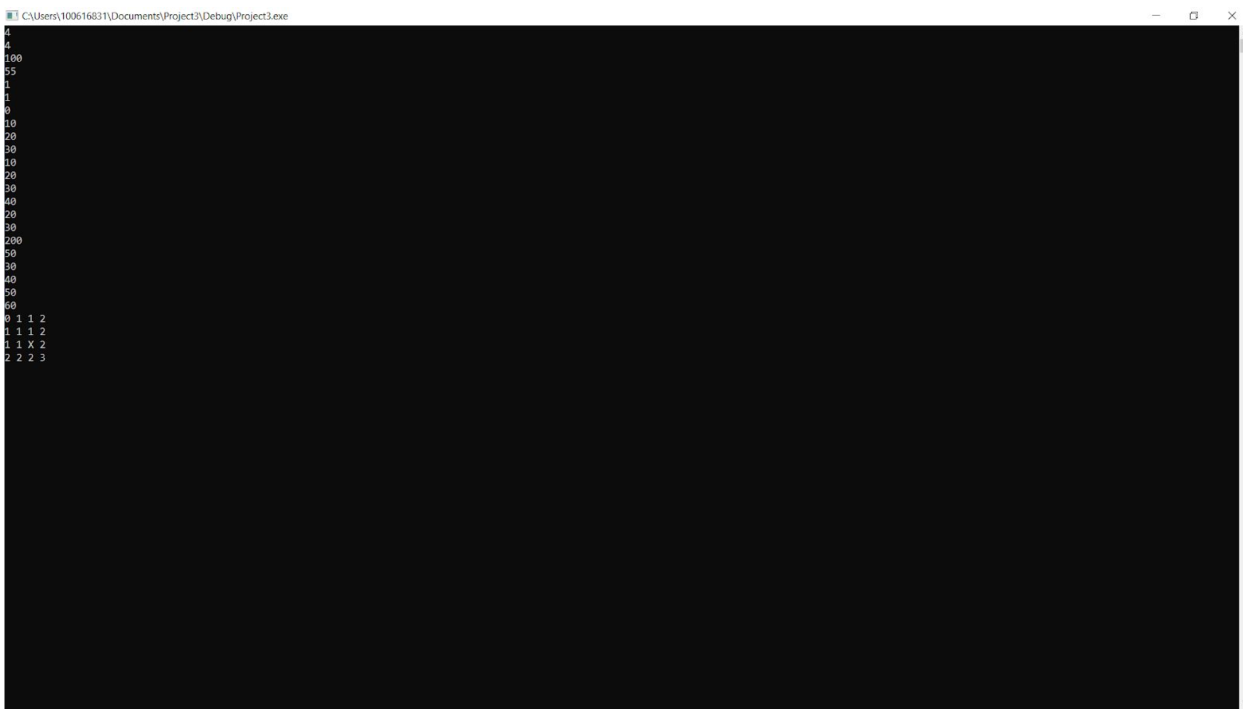


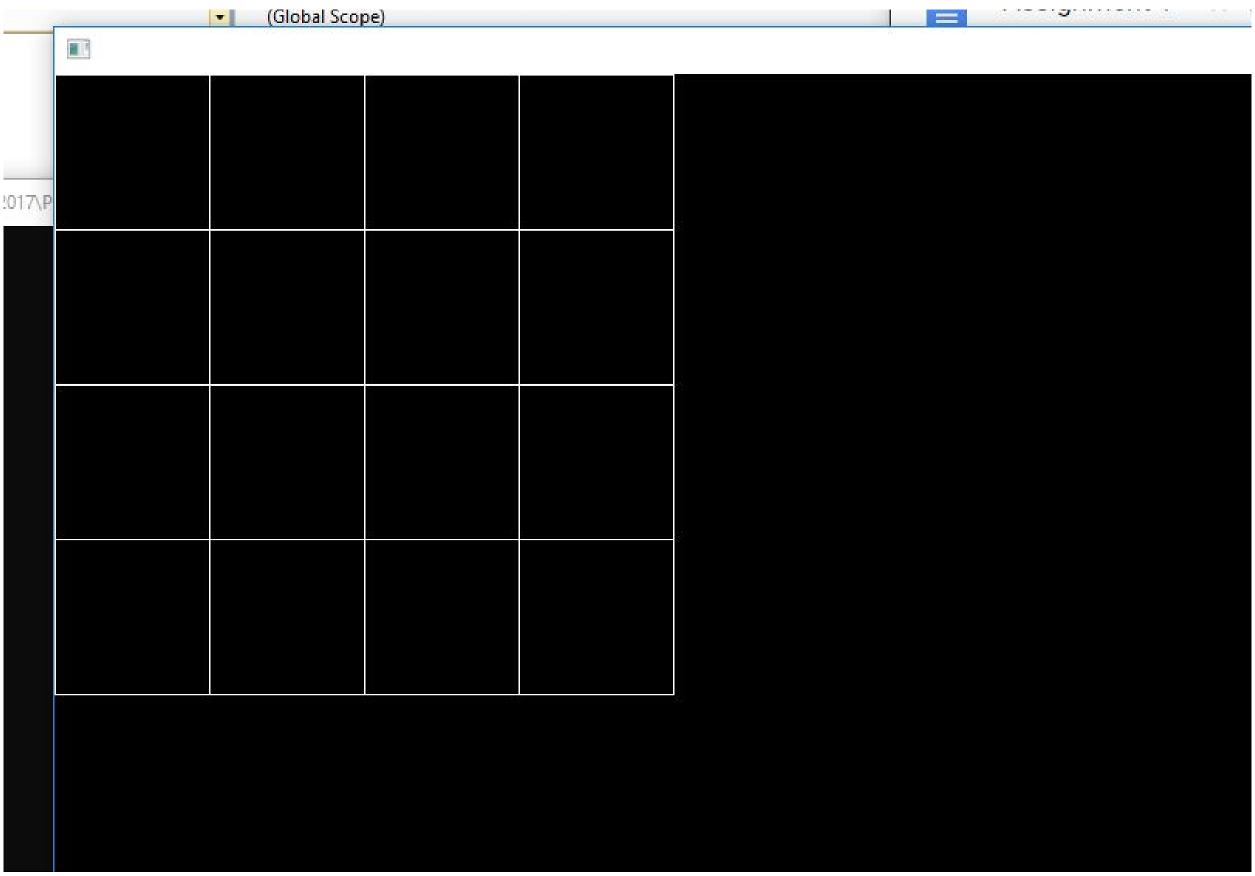
Case 2

Input:
4 4 100 55 1 1
0 10 20 30
10 20 30 40
20 30 200 50
30 40 50 60

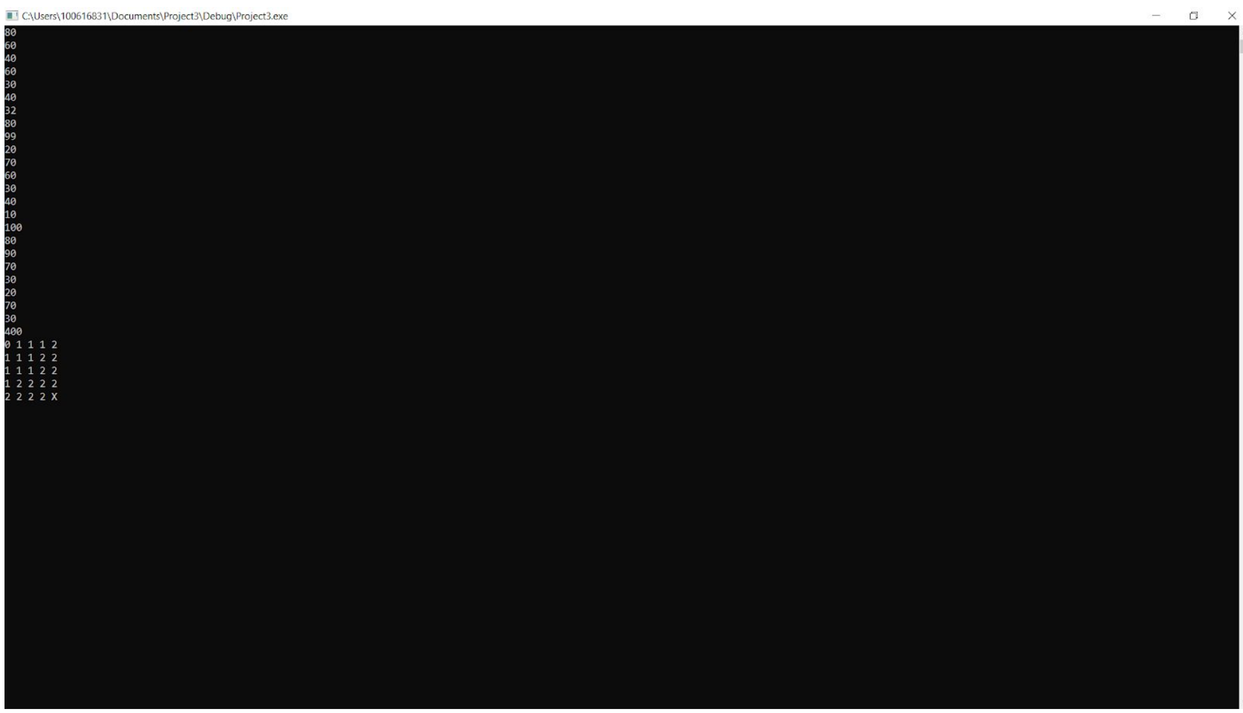
Output:
0 1 1 2
1 1 1 2
1 1 X 2
2 2 2 3

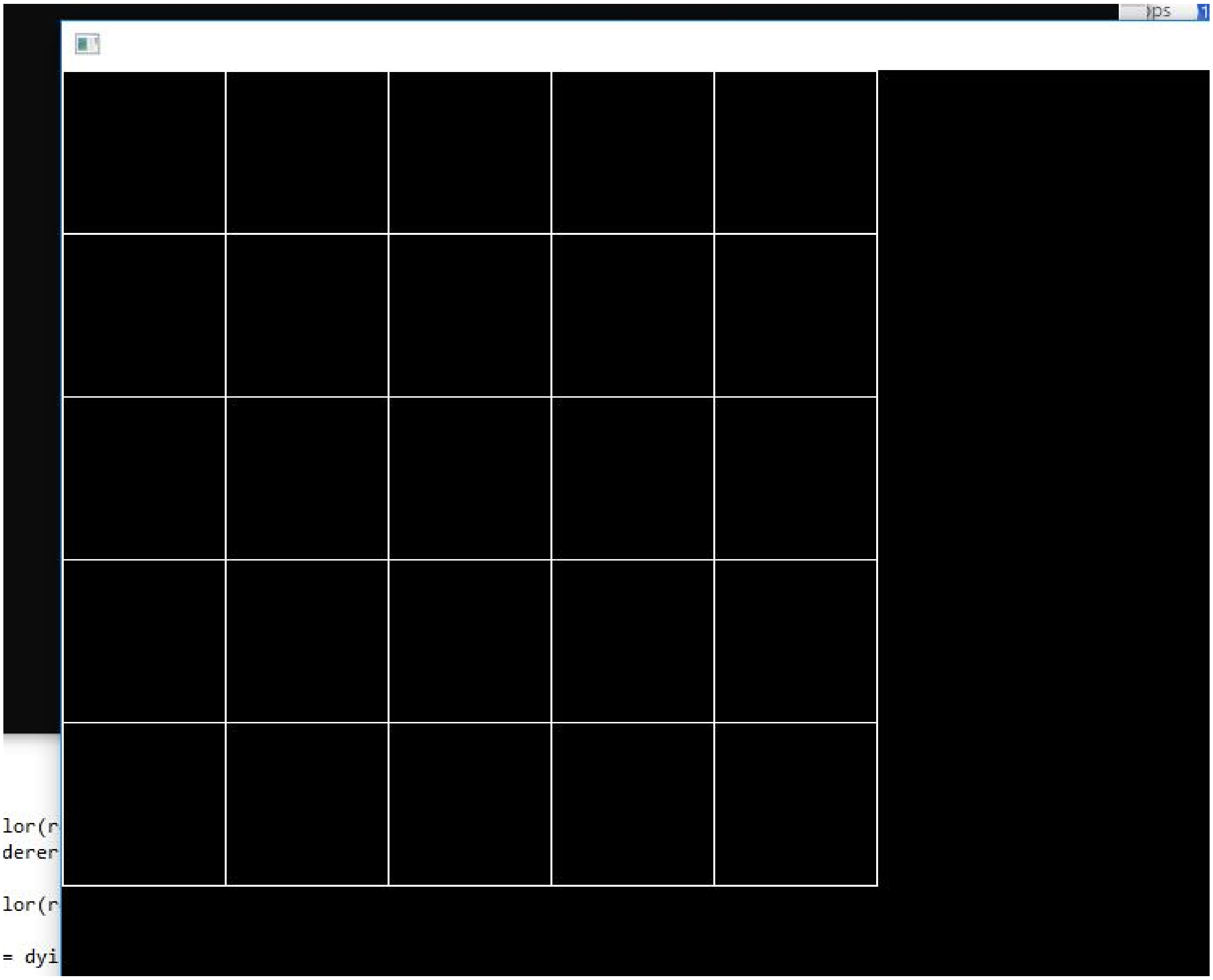
Runtime: .003s




(Global Scope)

Case 3:
Input:
5 5 100 60 1 1
20 80 60 40 80
30 40 32 80 99
20 70 60 30 40
10 100 80 90 70
30 20 70 30 40
Output:
0 1 1 1 2
1 1 1 2 2
1 1 1 2 2
1 2 2 2 2
2 2 2 2 2
Runtime: .005s




lor(r
derer
lor(r
= dyi

Case 4:
Input:
5 3 100 60 1 1
10 60 20 80 30
12 68 43 19 37
11 85 34 58 17

Output:
0 1 1 2 2
1 1 1 1 2
1 1 1 2 2

Runtime: .003s