Digital Design 2

Project 2

Andrew Nady 900184042
Akram Aziz 900184016
Mahmoud Elshinawy 900183926

Assumptions:

- We made the grid smaller (100 x100) upon the professor's approval.
- We decided to replace when a connection is established and there is some path we blocked the cells changing the value in the adjacency matrix to 0 which will be mapped as infinity in Dijkstra.
- We make the test cases have no spaces between the brackets to have an easier parsing.
- We changed some of the test cases a little bit, to fit in the new size of our grid which we scaled down to be 100x100 instead of 1000x1000.

Algorithm:

- We mainly depended on Dijkstra's shortest path finder algorithm as our main implementation
- It depends on Breadth First Search with priority queue, similar to Lee's algorithm.
- We created a class of the maze containing the adjacency matrix and a couple of functions besides Dijkstra.
- In the main program, we have a couple of functions for parsing the text files to prepare it to be sent across the maze object to the Path finder function called Lee_maze().
- We created a large graph containing 20000 nodes representing the number of cells we have half of which correspond to layer 1 and the other half to layer 2.
- We created a function called create_graph(). This function initializes and creates the graph by updating the adjacency matrix of the graph with the true costs depending on the movement direction and the layer we are in.

TestCases:

- In the submission, there is a directory called TestCases. It contains all the 8 test cases along with the output.
- We modified some of the testcases a little bit to fit our smaller grid.

Testcase3:

X (1,4,3) (1,16,17)

Y (1,20,1) (1,2,13) (1,18,25)

W (1,14,4) (1,4,21) (1,12,30)

Output:

0.

(1,4,3)

(1,5,3)

(1,6,3)

(1,7,3)

(1,8,3)

(1,9,3)

(1,10,3)

(1,11,3)

(1,12,3)

(1,13,3)

(1,14,3)

(1,15,3)

(1,16,3)

(2,16,3)

(2,16,4)

(2,16,5)

(2,16,6)

(2,16,7)

(2,16,8)

(2,16,9)

(2,16,10)

(2,16,11)

(2,16,12)

(2,16,13)

(2,16,14)

(2,16,15)

(2,16,16)

(2,16,17)

(1,16,17)

1.

(1,20,1)

(1,19,1)

(1,18,1)

(1,17,1)

(1,16,1)

(1,15,1)

(1,14,1)

(1,13,1)

(1,12,1)

(1,11,1)

(1,10,1)

(1,9,1)

(1,8,1)

(1,7,1)

(1,6,1)

(1,5,1)

(1,4,1)

(1,3,1)

(1,2,1)

(2,2,1)

(2,2,2)

(2,2,3)

(2,2,4)

(2,2,5)

(2,2,6)

(2,2,7)

(2,2,8)

(2,2,9)

(2,2,10)

(2,2,11)

(2,2,12)

(2,2,13)

(1,2,13)

(1,2,13)

(1,3,13)

(1,4,13)

(1,5,13)

(1,6,13)

(1,7,13)

(1,8,13)

(1,9,13)

(1,10,13)

(1,11,13)

(1,12,13)

(1,13,13)

(1,14,13)

(1,15,13)

(1,16,13)

(1,17,13)

(1,18,13)

(2,18,13)

(2,18,14)

(2,18,15)

(2,18,16)

(2,18,17)

(2,18,18)

(2,18,19)

(2,18,20)

(2,18,21)

(2,18,22)

(2,18,23)

(2,18,24)

(2,18,25)

(1,18,25)

2.

(1,14,4)

(1,13,4)

(1,12,4)

(1,11,4)

(1,10,4)

(1,9,4)

(1,8,4)

(1,7,4)

(1,6,4)

(1,5,4)

(1,4,4)

(2,4,4)

(2,4,5)

(2,4,6)

(2,4,7)

(2,4,8)

(2,4,9)

(2,4,10)

(2,4,11)

(2,4,12)

(2,4,13)

(2,4,14)

(2,4,15)

(2,4,16)

(2,4,17)

(2,4,18)

(2,4,19)

(2,4,20)

(2,4,21)

(1,4,21)

(1,4,21)

(1,5,21)

(1,6,21)

(1,7,21)

(1,8,21)

(1,9,21)

(1,10,21)

(1,11,21)

(1,12,21)

(2,12,21)

(2,12,22)

(2,12,23)

(2,12,24)

(2,12,25)

(2,12,26)

(2,12,27)

(2,12,28)

(2,12,29)

(2,12,30)

(1,12,30)

- In the output each line contains the next cell to go to until it reaches the final destination.

TestCase4:

A (1,0,0) (1,3,1) (1,6,1)

B (1,4,0) (1,2,1)

Output:

```
1,2,1maco@maco-VirtualBox:/media/sf_dd2/Project2$ make
g++ maze.cpp main.cpp -o maze.exe
./maze.exe
size: 2

1       0       0       1       3       1       1       6       1
TestMaze: 10
0       1       2       3       103
103     104     105     106
1       4       0       1       2       1
TestMaze: 10
4       5       6       7       107     207     206     205     204     203     202     102
this is testing for the map_real_pos function
1,2,1maco@maco-VirtualBox:/media/sf_dd2/Project2$
```

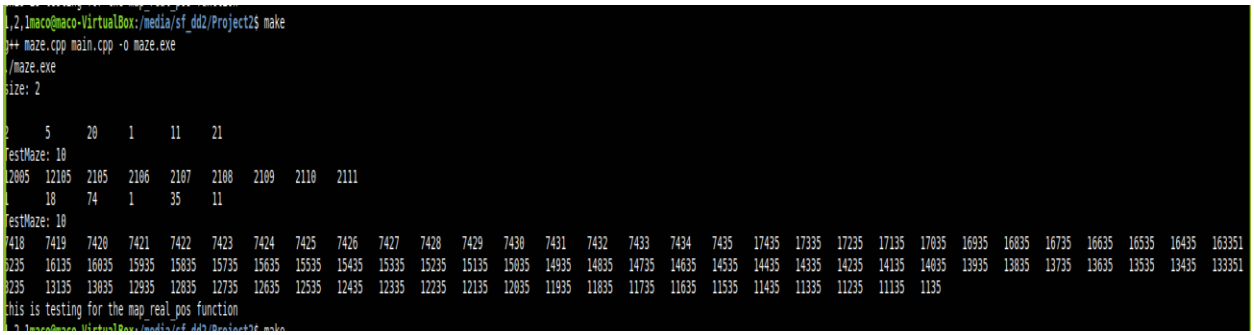
- In the previous example, the shortest path of the first wire will overlap with the second wire as we called the block_cell() function on all the nodes in the path. That is why the program chooses the next shortest path in hand.

TestCase8:

A (2,5,20) (1,11,21)

B (1,18,74) (1,35,11)

Output:



0.

(2,5,20)

(2,5,21)

(1,5,21)

(1,6,21)

(1,7,21)

(1,8,21)

(1,9,21)

(1,10,21)

(1,11,21)

1.

(1,18,74)

(1,19,74)

(1,20,74)

(1,21,74)

(1,22,74)

(1,23,74)

(1,24,74)

(1,25,74)

(1,26,74)

(1,27,74)

(1,28,74)

(1,29,74)

(1,30,74)

(1,31,74)

(1,32,74)

(1,33,74)

(1,34,74)

(1,35,74)

(2,35,74)

(2,35,73)

(2,35,72)

(2,35,71)

(2,35,70)

(2,35,69)

(2,35,68)

(2,35,67)

(2,35,66)

(2,35,65)

(2,35,64)

(2,35,63)

(2,35,62)

(2,35,61)

(2,35,60)

(2,35,59)

(2,35,58)

(2,35,57)

(2,35,56)

(2,35,55)

(2,35,54)

(2,35,53)

(2,35,52)

(2,35,51)

(2,35,50)

(2,35,49)

(2,35,48)

(2,35,47)

(2,35,46)

(2,35,45)

(2,35,44)

(2,35,43)

(2,35,42)

(2,35,41)

(2,35,40)

(2,35,39)

(2,35,38)

(2,35,37)

(2,35,36)

(2,35,35)

(2,35,34)

(2,35,33)

(2,35,32)

(2,35,31)

(2,35,30)

(2,35,29)

(2,35,28)

(2,35,27)

(2,35,26)

(2,35,25)

(2,35,24)

(2,35,23)

(2,35,22)

(2,35,21)

(2,35,20)

(2,35,19)

(2,35,18)

(2,35,17)

(2,35,16)

(2,35,15)

(2,35,14)

(2,35,13)

(2,35,12)

(2,35,11)

(1,35,11)

- We just give a couple of the test cases, for all the test cases, you can go the TestCases directory, it is going to have all the test cases along with their outputs.

Group Roles:

- Andrew:
  - Dijkstra_implementation
  - Input Parsing
  - Testing Dijkstra and Finding the right path
  - Plan the graph adjacency matrix
- Akram:
  - Dijkstra_implementation
  - Mapping_functions
  - Testcases
  - Finding the desired Path

- Mahmoud:
  - Dijkstra_implementation
  - Create_graph
  - Block_cell()
  - Output to file