

# FINAL PROJECT

## Data structure II

Omar Ihab El-Hariry	5360
Ahmed Tharwat Wagdy	5336
Hossam El-kady	5446
Omar Khaled AbdelHakem	5562
Aasem Mohamed Henedy	5580

**INSTRUCTOR:**

***DR. AMR EL-MASRY***

## LinkedIn

LinkedIn site is based on connecting people according to their profiles and classify the power of the connection with numbers that identify how close people are to each other.



## CODE:

1. we scan the needed information from user.

A screenshot of a C++ IDE (Code::Blocks) showing the LinkedIn.cpp file. The code implements a graph structure and a BFS algorithm to find the number of people at a specific distance from a root vertex. The output window shows the program's execution, including prompts for vertices, edges, and the starting vertex, followed by the results of the BFS search.

```

42         visited.insert(*i);
43         queue.push(make_pair(*i, depth + 1));
44     }
45     return ans;
46 }
47 void Graph::solve()
48 {
49     int vertices, e, u, v, root, k;
50     vector<int> ans;
51     cout << "please enter number of vertices:" << "\n";
52     cin >> vertices;
53     cout << "please enter number of edges:" << "\n";
54     cin >> e;
55     cout << "please enter edges in the form (u v):" << "\n";
56     Graph* graph = new Graph(vertices + 1);
57     for (int i = 0; i < e; ++i)
58     {
59         cin >> u >> v;
60         graph->add_edge(u, v);
61     }
62     cout << "please enter starting vertex:" << "\n";
63     cin >> root;
64     cout << "please enter value k: " << "\n";
65     cin >> k;
66     ans = graph->BFS(root, k);
67     cout << "There are " << ans.size() << " People with " << k << " connection away starting from " << root << "\n";
68     cout << "And The People Are: " << "\n";
69     for (int x : ans)
70         cout << x << " ";
71 }

```

Run: Debug in graphh (compiler: GNU GCC Compiler)

Checking for existence: C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe

Process terminated with status -1073741310 (2 minutes(s), 29 second(s))

- then we create a list for the graph and add edges between vertices from both sides as it is an undirected graph.

```

1 #include<iostream>
2 #include<vector>
3 #include<set>
4 #include<queue>
5 #include"Linkedin.h"
6 using namespace std;
7 Graph::Graph(int V)
8 {
9     this->V = V;
10    adj = new list<int>[V];
11 }
12 void Graph::add_edge(int v, int w)
13 {
14     adj[v].push_back(w); // Add w to v's list.
15     adj[w].push_back(v); // Add v to w's list.
16 }
17 vector<int> Graph::BFS(int s, int k)
18 {
19     // Create a queue for BFS
20     set<int> visited;
21     queue<pair<int, int>> queue;
22
23     // Mark the current node as visited and enqueue it
24     visited.insert(s);
25     queue.push(make_pair(s, 0));
26
27     list<int>::iterator i;
28     vector<int> ans;
29     while (!queue.empty())
30     {
31         s = queue.front().first;
32         int depth = queue.front().second;
33         if (depth == k)
34         {
35             ans.push_back(s);
36         }
37         queue.pop();
38         for (i = adj[s].begin(); i != adj[s].end(); ++i)
39         {
40             if (visited.count(*i) == 0)
41             {
42                 visited.insert(*i);
43                 queue.push(make_pair(*i, depth + 1));
44             }
45         }
46     }
47     return ans;
48 }

```

Run: Debug in graph (compiler: GNU GCC Compiler)  
 Checking for existence: C:\Users\ADMIN\Documents\graph\bin\Debug\graph.exe  
 Executing: "C:\Program Files (x86)\CodeBlocks\cb\_console\_runner.exe" "C:\Users\ADMIN\Documents\graph\bin\Debug\graph.exe" (in C:\Users\ADMIN\Documents\graph\.)  
 Process terminated with status -107741510 (2 minute(s), 29 second(s))

- finally we create a queue for BFS traversal then assign the depth for each node and check if the given distance from a given node (K) is equal to the depth, then this node is a K distance away from the given node.

```

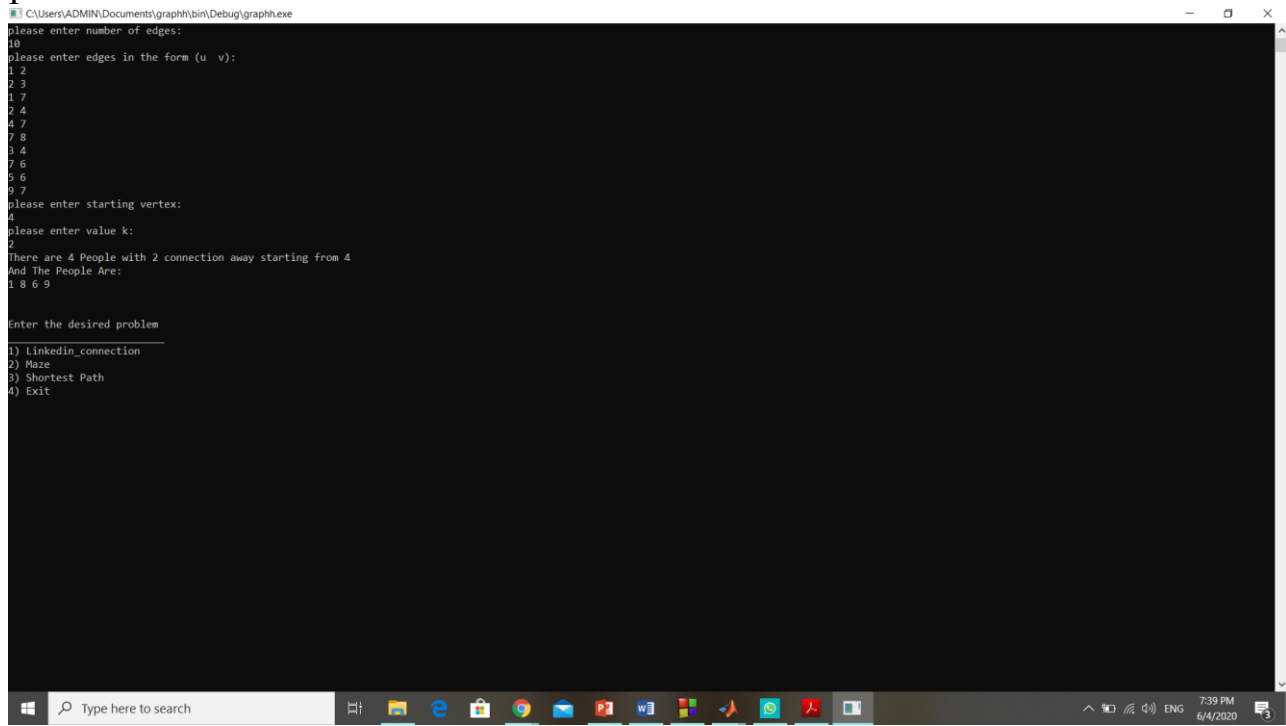
17 vector<int> Graph::BFS(int s, int k)
18 {
19     // Create a queue for BFS
20     set<int> visited;
21     queue<pair<int, int>> queue;
22
23     // Mark the current node as visited and enqueue it
24     visited.insert(s);
25     queue.push(make_pair(s, 0));
26
27     list<int>::iterator i;
28     vector<int> ans;
29     while (!queue.empty())
30     {
31         s = queue.front().first;
32         int depth = queue.front().second;
33         if (depth == k)
34         {
35             ans.push_back(s);
36         }
37         queue.pop();
38         for (i = adj[s].begin(); i != adj[s].end(); ++i)
39         {
40             if (visited.count(*i) == 0)
41             {
42                 visited.insert(*i);
43                 queue.push(make_pair(*i, depth + 1));
44             }
45         }
46     }
47     return ans;
48 }

```

Run: Debug in graph (compiler: GNU GCC Compiler)  
 Checking for existence: C:\Users\ADMIN\Documents\graph\bin\Debug\graph.exe  
 Executing: "C:\Program Files (x86)\CodeBlocks\cb\_console\_runner.exe" "C:\Users\ADMIN\Documents\graph\bin\Debug\graph.exe" (in C:\Users\ADMIN\Documents\graph\.)  
 Process terminated with status -107741510 (2 minute(s), 29 second(s))

## **OUTPUT:**

This is the output of code after run on the test case sent in the final project pdf.



```
C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe
please enter number of edges:
10
please enter edges in the form (u v):
1 2
2 3
1 7
2 4
4 7
7 8
3 4
7 6
5 6
9 7
please enter starting vertex:
4
please enter value k:
2
There are 4 People with 2 connection away starting from 4
And The People Are:
1 8 6 9

Enter the desired problem
1) LinkedIn_connection
2) Maze
3) Shortest Path
4) Exit
```

## **COMPLEXITY:**

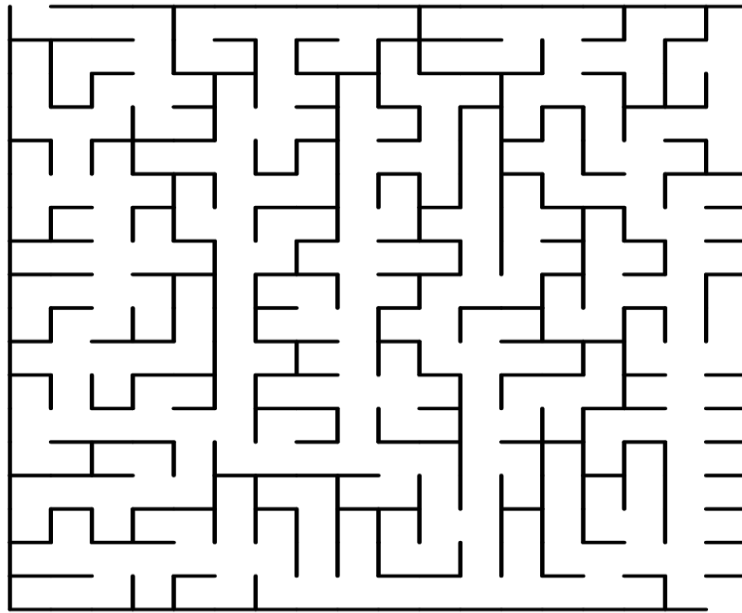
$O(|V| + |E|)$  where  $V$  is the number of vertices and  $E$  is the number of edges.

## **DATA STRUCTURES:**

1. Adjacency list
2. Vectors
3. Queues
4. sets

## Maze

Maze problem is based on entering a matrix of ones and zeros where zeros is the path that can make you get out of maze where the start should be from (0,0) and the end of maze that should be at (N-1,N-1) where N is the size of matrix.



### CODE:

1. First function is to mark the maze blocks that we pass on it as visited and to give true or false according to its status of being visited or no, while second one is to check if block is being visited before or no.

```

1 #include "maze.h"
2 #include <queue>
3 #include <iostream>
4 #include <stack>
5 //function used mark maze blocks as visited using a simple
6 //hash function to store them
7 //is a 1D boolean array with size of N*N
8 void maze::visit(int i, int j) {
9     visited_locations[i * N + j] = true;
10    visited_nodes++;
11 }
12
13 //check if a block is visited
14 bool maze::visited(int i, int j) {
15     return visited_locations[i * N + j];
16 }
17
18 //traverse the 4 neighbours of a given block by i and j
19 //also push the unvisited neighbours in a vector
20 vector<int> maze::unvisited_neighbours(int i, int j, vector<vector<int>> maze) {
21     vector<int> unvisited_neighbours;
22     if (j != N - 1 && maze[i][j + 1] == 0 && !visited(i, j + 1)) {
23         unvisited_neighbours.push_back((i) * N + j + 1);
24     }
25     if (j != 0 && maze[i][j - 1] == 0 && !visited(i, j - 1)) {
26         unvisited_neighbours.push_back((i) * N + j - 1);
27     }
28 }
29
30 //main function
31 int main() {
32     //your code here
33 }

```

- Here we traverse all the 4 neighbours of a block in a maze in case of it's not the last or first block in maze and push those unvisited blocks in a vector.

```

13 //check if a block is visited
14 bool maze::visited(int i, int j) {
15     return visited_locations[i * N + j];
16 }
17
18 //traverse the 4 neighbours of a given block by i and j
19 //and return the unvisited neighbours in a vector
20 vector<int> maze::unvisited_neighbours(int i, int j, vector<vector<int>>> maze) {
21     vector<int> unvisited_neighbours;
22     if (j != N - 1 && maze[i][j + 1] == 0 && !visited(i, j + 1)) {
23         unvisited_neighbours.push_back((i) * N + j + 1);
24     }
25     if (j != 0 && maze[i][j - 1] == 0 && !visited(i, j - 1)) {
26         unvisited_neighbours.push_back((i) * N + j - 1);
27     }
28     if (i != N - 1 && maze[i + 1][j] == 0 && !visited(i + 1, j)) {
29         unvisited_neighbours.push_back((i + 1) * N + j);
30     }
31     if (i != 0 && maze[i - 1][j] == 0 && !visited(i - 1, j)) {
32         unvisited_neighbours.push_back((i - 1) * N + j);
33     }
34     return unvisited_neighbours;
35 }
36
37 // Depth First Traversal
38 void maze::DFS() {
39     int i = 0;
40     int j = 0;
41     vector<int> s;
42 }

```

- In this part of code we iterate in maze to find all the unvisited nodes just if we did not reach the end of maze yet and then after visiting we push them in the stack, and here there are 2 types of traversing DFS and BFS doing the same idea.

## DFS

```

40 // Depth First Traversal
41 void maze::DFS() {
42     int i = 0;
43     int j = 0;
44     parents[0] = -1;
45     stack<int> s;
46     int new_node;
47     //iterate while there is unvisited nodes in the graph
48     // and the end not reached
49     while (unvisited_nodes > 0 && !visited(N-1, N-1)) {
50         visit(i, j);
51         s.push(i * N + j);
52         while (!s.empty() && !visited(N-1, N-1)) {
53             new_node = s.top();
54             // get the neighbours of the current nodes
55             vector<int> neighbours = unvisited_neighbours(new_node / N, new_node % N, input_maze);
56             // if there is unvisited neighbours Visit it and add it to stack
57             if (!neighbours.empty()) {
58                 i = neighbours[0] / N;
59                 j = neighbours[0] % N;
60                 visit(i, j);
61                 s.push(i * N + j);
62                 parents[i * N + j] = new_node;
63             }
64             else {
65                 s.pop();
66             }
67         }
68     }
69 }
70
71 }

```

## BFS

```

73 // BFS
74 // Breadth First Traversal
75 void maze::BFS() {
76     int i = 0;
77     int j = 0;
78     parents[0] = -1;
79     queue<int> q;
80     int new_node;
81     //iterate while there is unvisited nodes in the graph
82     // and the finish block not visited
83     while (unvisited_nodes > 0 && !visited(N-1, N-1)) {
84         visit(i, j);
85         q.push(i * N + j);
86         while (!q.empty() && !visited(N-1, N-1)) {
87             new_node = q.front();
88             q.pop();
89             //get all node neighbours and visit the unvisited ones
90             vector<int> neighbours = unvisited_neighbours(new_node / N, new_node % N, input_maze);
91             for (int k = 0; k < neighbours.size(); k++) {
92                 i = neighbours[k] / N;
93                 j = neighbours[k] % N;
94                 visit(i, j);
95                 q.push(i * N + j);
96                 parents[i * N + j] = new_node;
97             }
98         }
99     }
100 }
101
102
103
104

```

Run: Debug in graphh (compiler: GNU GCC Compiler)

Checking for existence: C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe  
 Executing: "C:\Program Files (x86)\CodeBlocks\cb\_console\_runner.exe" "C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe" (in C:\Users\ADMIN\Documents\graphh\.)  
 Process terminated with status -1073741510 (12 minute(s), 39 second(s))

4. Here we check if the first (0,0) and last (N-1,N-1) nodes are in the path and if the last node has a parent so there is a path and solution for the maze and then we get this path from parents array and put it in form of pair of locations (i,j) and if the last node has no parent or the path does not start from the initial node or does not reach the last node we print "No path found".

```

106 void maze::solve_maze(string traversal) {
107     stack<pair<int, int>> solution;
108     cout << "Solution" << endl;
109     //check if start and end are on the path
110     if (input_maze[N-1][N-1] == 0 && input_maze[0][0] == 0) {
111         //choose the traversal according to user input
112         if (traversal == "BFS")
113             BFS();
114         else
115             DFS();
116         //if last element have a parent
117         //then there is a path to the end
118         if (parents[N * N - 1] != -1) {
119             int i = N - 1;
120             pair<int, int> maze_block(N-1, N-1);
121             solution.push(maze_block);
122             //choose path from parents array
123             while (i > 0) {
124                 maze_block.first = parents[i] / N;
125                 maze_block.second = parents[i] % N;
126                 solution.push(maze_block);
127                 i = parents[i];
128             }
129             //printing the path
130             while (!solution.empty()) {
131                 cout << " * " << solution.top().first << " * " << solution.top().second << " * " << endl;
132                 solution.pop();
133             }
134         }
135         else {
136             cout << "NO PATH FOUND" << endl;
137         }
138     }
139     else {
140         cout << "NO PATH FOUND" << endl;
141     }
142 }

```

Run: Debug in graphh (compiler: GNU GCC Compiler)

Checking for existence: C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe  
 Executing: "C:\Program Files (x86)\CodeBlocks\cb\_console\_runner.exe" "C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe" (in C:\Users\ADMIN\Documents\graphh\.)  
 Process terminated with status -1073741510 (12 minute(s), 39 second(s))

5. At last there is a constructor of 2D vectors of maze and number of rows and columns and number of available blocks to pass through.

```

126     maze_block.second = parents[i] * N;
127     solution.push(maze_block); i = parents[i];
128
129     //printing the path
130     while (!solution.empty()) {
131         cout << "(" << solution.top().first << ", " << solution.top().second << ")" << endl;
132         solution.pop();
133     }
134
135     else {
136         cout << "NO PATH FOUND" << endl;
137     }
138
139     else {
140         cout << "NO PATH FOUND" << endl;
141     }
142 }
143 //maze constructor which take 2D vector for maze
144 // and number of rows or columns
145 // and number of valid blocks to traverse
146 maze::maze(vector<vector<int>> maze, int n, int number_of_zeros)
147 {
148     N = n;
149     vector<bool> new_visited_locations(N * N, false);
150     visited_locations = new_visited_locations;
151     input_maze = maze;
152     unvisited_nodes = number_of_zeros;
153     vector<int> new_parents(N * N, -1);
154     parents = new_parents;
155 }

```

Logs & others

```

Code::Blocks  Search results  Cocc  Build log  Build messages  CppCheck  CppCheck messages  Cscope  Debugger  DoxyBlocks  Fortran info  Closed files list  Thread search
-----
Run: Debug in graphh (compiler: GNU GCC Compiler)-----
Checking for existence: C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe
Executing: "C:\Program Files (x86)\CodeBlocks\cb_console_runner.exe" "C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe" (in C:\Users\ADMIN\Documents\graphh\.)
Process terminated with status -1073741510 (12 minute(s) - 39 second(s))

```

## OUTPUT:

This is the output of code after run on the test case sent in the final project pdf.

```

C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe
Enter the desired problem
1) LinkedIn_connection
2) Maze
3) Shortest Path
4) Exit
2
Please enter N:
4
Please enter values for maze:
0 1 0
0 0 1 0
0 0 0 0
0 1 1 0
DFS
Solution
( 0 , 0 )
( 1 , 0 )
( 1 , 1 )
( 2 , 1 )
( 2 , 2 )
( 2 , 3 )
( 3 , 3 )
Time taken by DFS: 127595 microseconds
BFS
Solution
( 0 , 0 )
( 1 , 0 )
( 1 , 1 )
( 2 , 1 )
( 2 , 2 )
( 2 , 3 )
( 3 , 3 )
Time taken by BFS: 11985 microseconds
Enter the desired problem
1) LinkedIn_connection
2) Maze
3) Shortest Path
4) Exit

```



## **COMPLEXITY:**

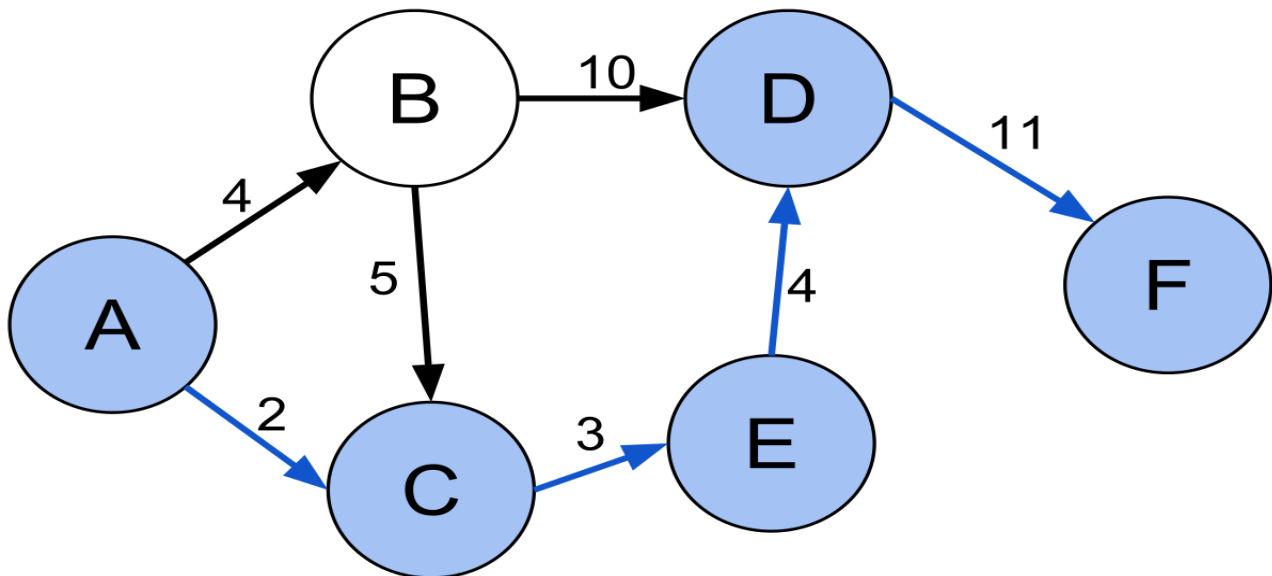
1.  $O(N^2)$  where  $N$  is the dimension of the matrix of maze.

## **DATA STRUCTURES:**

1. Queue is used for BFS traversal which is used to find the shortest path.
2. Stack is used for DFS traversal that is better to find any path but not the shortest.
3. Stack of pairs for output.
4.  $N \times N$  parents vector that we extract the output path from.
5.  $N \times N$  Boolean vector for visited function.
6.  $N \times N$  2D vector.

## Shortest Path

This project idea is based on decreasing the amount of money for the employee who should travel between cities by finding the shortest path that he could travel from.



## CODE:

1. Code start with taking input from user and make sure that they are valid under the conditions of the program.

```

10  int m, n, r, s, d, t; int N = 0, totalcost = 0, totaltime = 0;
11  std::priority_queue<node>, std::vector<node>, node_cmp> Q;
12  string shortestPath;
13
14  cout << "please enter amount M:";
15  cin >> m;
16  if (m <= 0) {
17      cout << "Invalid Data Entered";
18      exit(0);
19  }
20
21  cout << "please enter number of cities:";
22  cin >> n;
23  if (n <= 0) {
24      cout << "Invalid Data Entered";
25      exit(0);
26  }
27
28  cout << "please enter number of routes:";
29  cin >> r;
30  if (r <= 0) {
31      cout << "Invalid Data Entered";
32      exit(0);
33  }
34
35  vector<vector<int>>> input(r, vector<int>(4));
36
37  cout << "please enter source , destination , time and cost for each route:";
38  for (int i = 0; i < r; i++) {
39      for (int j = 0; j < 4; j++) {
40          cin >> input[i][j];
41      }
42  }
  
```

Logs & others

```

Run: Debug in graphh (compiler: GNU GCC Compiler)
Checking for existence: C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe
Execution: "C:\Program Files (x86)\CodeBlocks\cb_console_runner.exe" "C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe" (in C:\Users\ADMIN\Documents\graphh\)
Process terminated with status -1073741810 (1 minute(s), 23 second(s))
  
```

```

33 vector<vector<int>> input(r, vector<int>(4));
34
35
36 cout << "please enter source , destination , time and cost for each route:";
37 for (int i = 0; i < r; i++) {
38     for (int j = 0; j < 4; j++) {
39         cin >> input[i][j];
40         if (input[i][j] <= 0) {
41             cout << "Invalid Data Entered";
42             exit(0);
43         }
44     }
45 }
46
47 cout << "please enter source city: ";
48 cin >> s;
49 if (s <= 0) {
50     cout << "Invalid Data Entered";
51     exit(0);
52 }
53
54 cout << "please enter destination city:";
55 cin >> d;
56 if (d <= 0) {
57     cout << "Invalid Data Entered";
58     exit(0);
59 }
60
61 vector<vector<int>> graph(n + 1, vector<int>(n + 1, 0));
62 vector<vector<int>> time(n + 1, vector<int>(n + 1, 0));
63
64 vector<node> NodesArr(n + 2);

```

Run: Debug in graph (compiler: GNU GCC Compiler)-----  
 Checking for existence: C:\Users\ADMIN\Documents\graph\bin\Debug\graph.exe  
 Executing: "C:\Program Files (x86)\CodeBlocks\cb\_console\_runner.exe" "C:\Users\ADMIN\Documents\graph\bin\Debug\graph.exe" (in C:\Users\ADMIN\Documents\graph\.)  
 Process terminated with status -1073741510 (1 minute(s), 23 second(s))

2. We give ID to nodes that represent cities and then give nodes a distance from source by giving them time and cost according to how far they are from the source.

```

57 }
58
59 vector<vector<int>> graph(n + 1, vector<int>(n + 1, 0));
60 vector<vector<int>> time(n + 1, vector<int>(n + 1, 0));
61
62 vector<node> NodesArr(n + 2);
63
64 //giving IDs to nodes
65 for (int i = 1; i <= n; i++) {
66     NodesArr[i].ID = i;
67 }
68
69 for (int i = 0; i < r; i++) {
70     graph[input[i][0]][input[i][1]] = input[i][2] * input[i][3];
71     time[input[i][0]][input[i][1]] = input[i][2];
72 }
73
74 //Giving Nodes distance from source
75
76 for (int i = 1; i <= n; i++) {
77     if ((i != s) && (graph[s][i] != 0)) {
78         NodesArr[i].cost = graph[s][i];
79         NodesArr[i].time = time[s][i];
80         Q.push(NodesArr[i]);
81     }
82 }
83
84 //Algorithm
85 //loop works until The destination is the top of the queue
86

```

Run: Debug in graph (compiler: GNU GCC Compiler)-----  
 Checking for existence: C:\Users\ADMIN\Documents\graph\bin\Debug\graph.exe  
 Executing: "C:\Program Files (x86)\CodeBlocks\cb\_console\_runner.exe" "C:\Users\ADMIN\Documents\graph\bin\Debug\graph.exe" (in C:\Users\ADMIN\Documents\graph\.)  
 Process terminated with status -1073741510 (1 minute(s), 23 second(s))

3. Loop keep working till the destination city is the first in the queue then we assign parents and put the source as the first parent.

```

87 int f = 0; node* temp = sNodesArr[s];
88 while (!Q.empty() && (Q.top()->ID != NodesArr[d].ID))
89 {
90     //assigning parents
91     if ((f > 0) && (graph[Q.top()->ID][temp->ID] != 0))
92     {
93         Q.top()->parent = temp;
94     }
95     node* a = Q.top();
96     f++;
97     //making source as a first parent
98     if (f == 1)
99     {
100         Q.top()->parent = sNodesArr[s];
101     }
102     temp = Q.top();
103     Q.pop();
104     //relaxing
105     for (int i = 1; i <= n; i++) {
106         if ((i != temp->ID) && (graph[temp->ID][i] != 0) && (temp->parent != nullptr)) {
107             if ((graph[temp->ID][i] != 0) && ((graph[temp->parent->ID][i]) > ((graph[temp->ID][i]) + (graph[temp->parent->ID][temp->ID])))
108             {
109                 NodesArr[i].cost = graph[temp->ID][i];
110                 NodesArr[i].time = time[temp->ID][i];
111             }
112         }
113     }
114 }
115 }
116

```

Run: Debug in graphh (compiler: GNU GCC Compiler)

Checking for existence: C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe  
 Executing: "C:\Program Files (x86)\CodeBlocks\cb\_console\_runner.exe" "C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe" (in C:\Users\ADMIN\Documents\graphh\.)  
 Process terminated with status -1073741510 (1 minute(s), 23 second(s))

4. Then we extract the shortest path from the destination after that we calculate the total time and cost after finding the required path then print it.

```

115 //extracting destination node
116 if (graph[temp->ID][d] == 0) cout << "NO PATH FOUND";
117 else
118 {
119     NodesArr[d].cost = graph[temp->ID][d];
120     system("cls");
121     cout << "The route with minimum cost is:";
122     cout << "\n";
123     cout << NodesArr[s].ID;
124     cout << "\n";
125     stack<node> route;
126     route.push(*temp);
127     //printing
128     while (temp->parent != NULL)
129     {
130         route.push(*temp->parent);
131         temp = temp->parent;
132     }
133     int k = route.size();
134     while (!route.empty())
135     {
136         if ((route.top().ID == NodesArr[d].ID) || (route.top().ID == NodesArr[s].ID)) {
137             route.pop();
138             continue;
139         }
140         cout << route.top().ID;
141     }
142 }
143

```

Run: Debug in graphh (compiler: GNU GCC Compiler)

Checking for existence: C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe  
 Executing: "C:\Program Files (x86)\CodeBlocks\cb\_console\_runner.exe" "C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe" (in C:\Users\ADMIN\Documents\graphh\.)  
 Process terminated with status -1073741510 (1 minute(s), 23 second(s))

```

141     route.pop();
142     continue;
143 }
144 cout << route.top().ID;
145 if (k != 0)
146     cout << " ->";
147 k--;
148 if (route.top().ID != NodesArr[s].ID) {
149     totaltime = totaltime + route.top().time;
150     totalcost = totalcost + route.top().cost;
151 }
152 if (route.top().ID != NodesArr[d].ID) {
153     totaltime = totaltime + 1;
154 }
155 route.pop();
156 cout << NodesArr[d].ID;
157
158 totaltime = totaltime + NodesArr[d].time;
159 totalcost = totalcost + totaltime * m + NodesArr[d].cost;
160 cout << "\n";
161 cout << "Total time: ";
162 cout << totaltime;
163 cout << " hours";
164 cout << "\n";
165 cout << "Total cost: ";
166 cout << totalcost;
167 cout << "\n";
168 }
169 }
170 }

```

Run: Debug in graphh (compiler: GNU GCC Compiler)

Checking for existence: C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe  
 Executing: "C:\Program Files (x86)\CodeBlocks\bin\compiler\runner.exe" "C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe" (in C:\Users\ADMIN\Documents\graphh\.)  
 Success: Executed with status: 0 (2241516) (1 minute(s) 23 seconds(s))

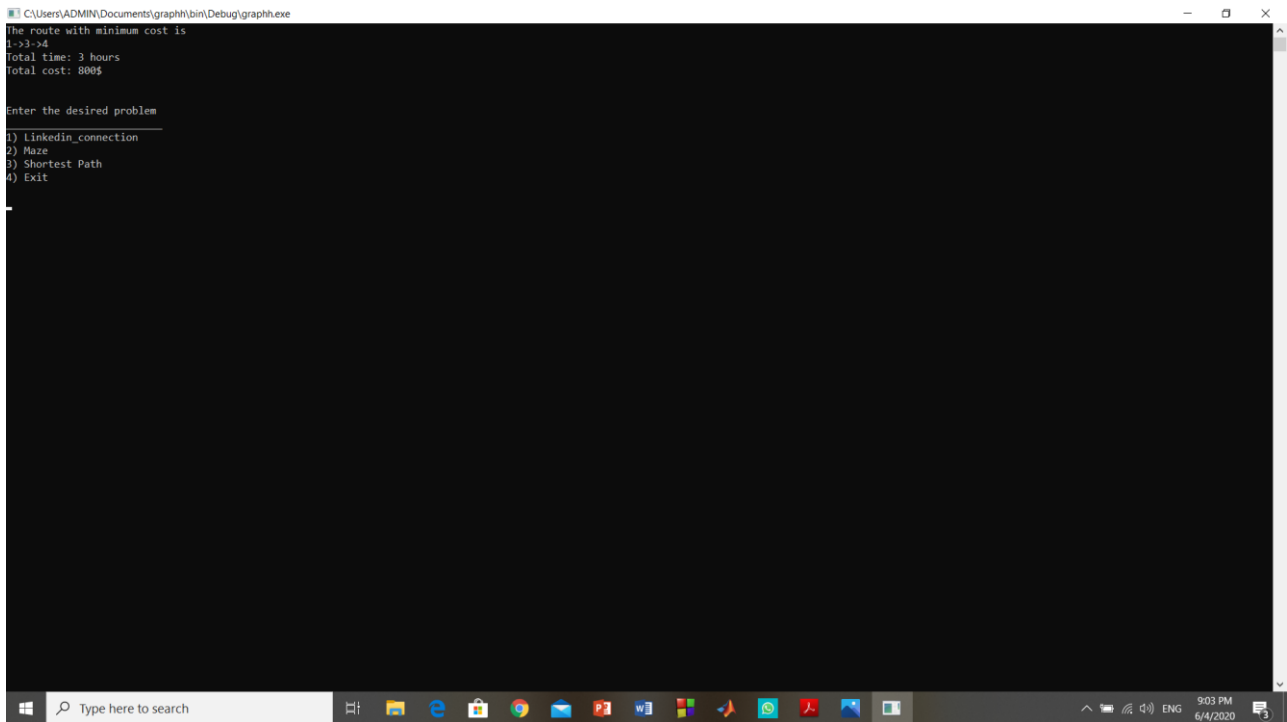
## OUTPUT:

This is the output of code after run on the test case sent in the final project pdf.

```

C:\Users\ADMIN\Documents\graphh\bin\Debug\graphh.exe
Enter the desired problem
1) LinkedIn_connection
2) Maze
3) Shortest Path
4) Exit
3
Please enter amount M:100
Please enter number of cities:4
Please enter number of routes:5
Please enter source , destination , time and cost for each route:1 2 1 250 1 3 1 300 1 4 2 700 2 4 1 300 3 4 1 200
Please enter source city: 1
Please enter destination city:4

```



```
C:\Users\ADMIN\Documents\graph\bin\Debug\graph.exe
The route with minimum cost is
1->3->4
Total time: 3 hours
Total cost: 800$

Enter the desired problem
1) LinkedIn_connection
2) Maze
3) Shortest Path
4) Exit
```

## **COMPLEXITY:**

$E \log(V)$ , Where  $V$  is the number of vertices and  $E$  is the number of edges.

## **DATA STRUCTURES:**

1. Queue.
2. Stack for printing.
3. Vectors.

## ROLES

### 1.LINKEDIN:

OMAR IHAB EL-HARIRY 5360

### 2.MAZE:

OMAR KHALED ABDELHAKEM 5562

AASEM MOHAMED HENEDY 5580

### 3.SHORTEST PATH:

AHMED THARWAT WAGDY 5336

HOSSAM EL-KADY 5446