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:

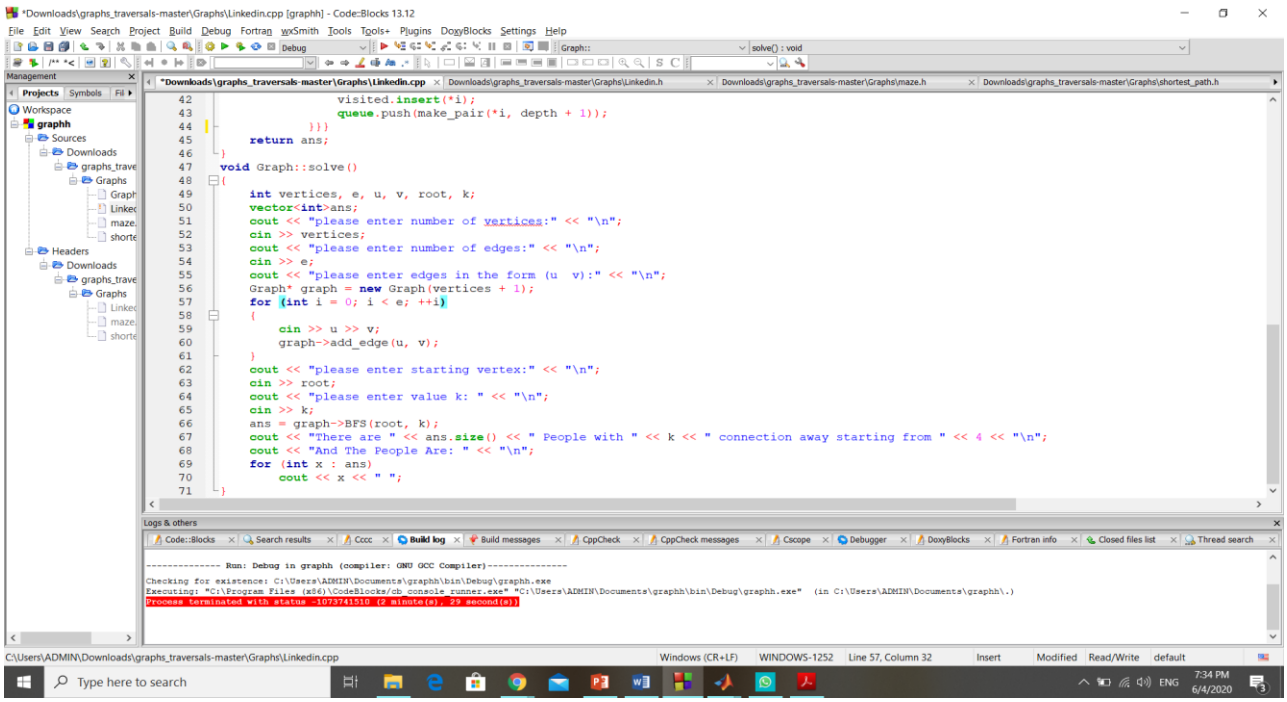
DR. AMR EL-MASRY

LinkedIn

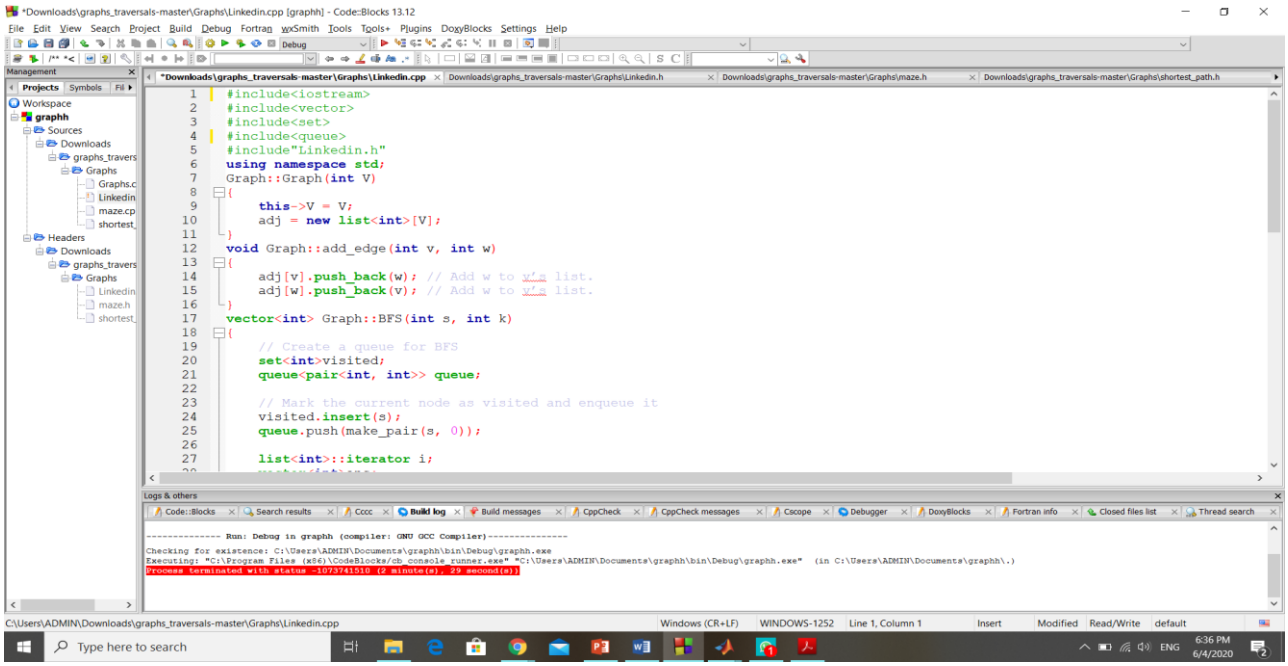
LinkedIn site is based on connecting people according to their profiles and classify the power of the connection with numbers that identify how close people are to each other.

CODE:

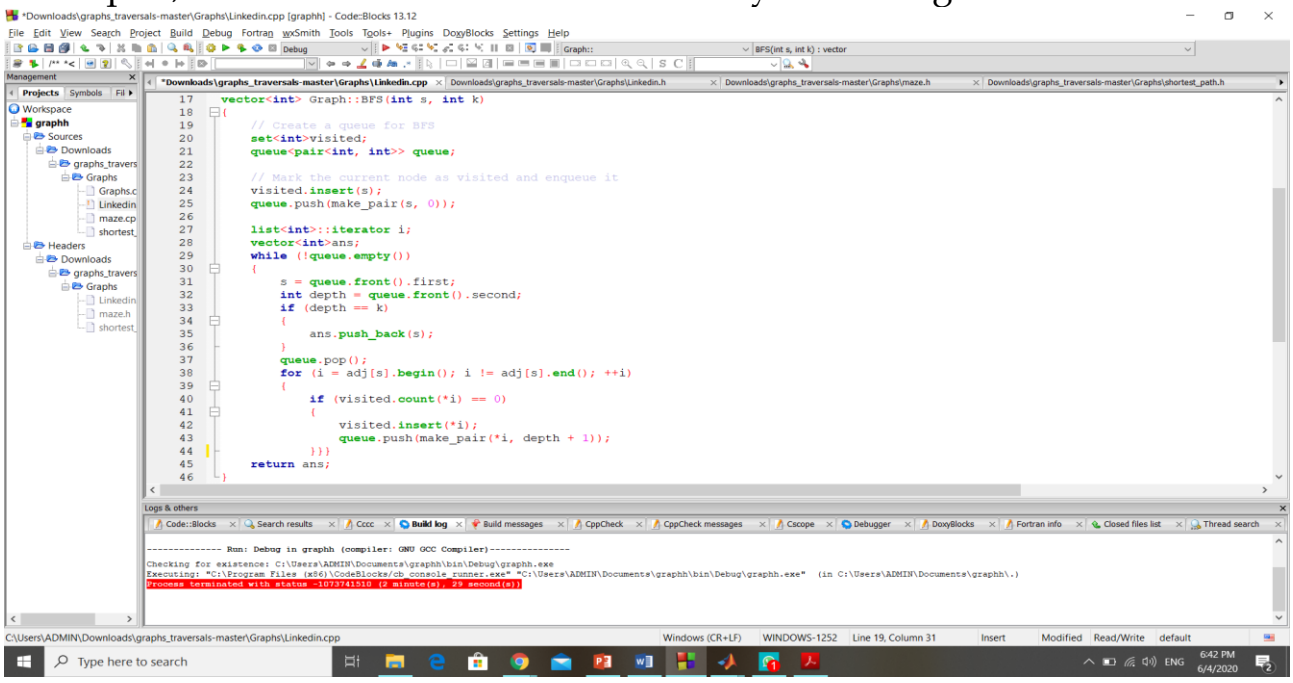
1. we scan the needed information from user.



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

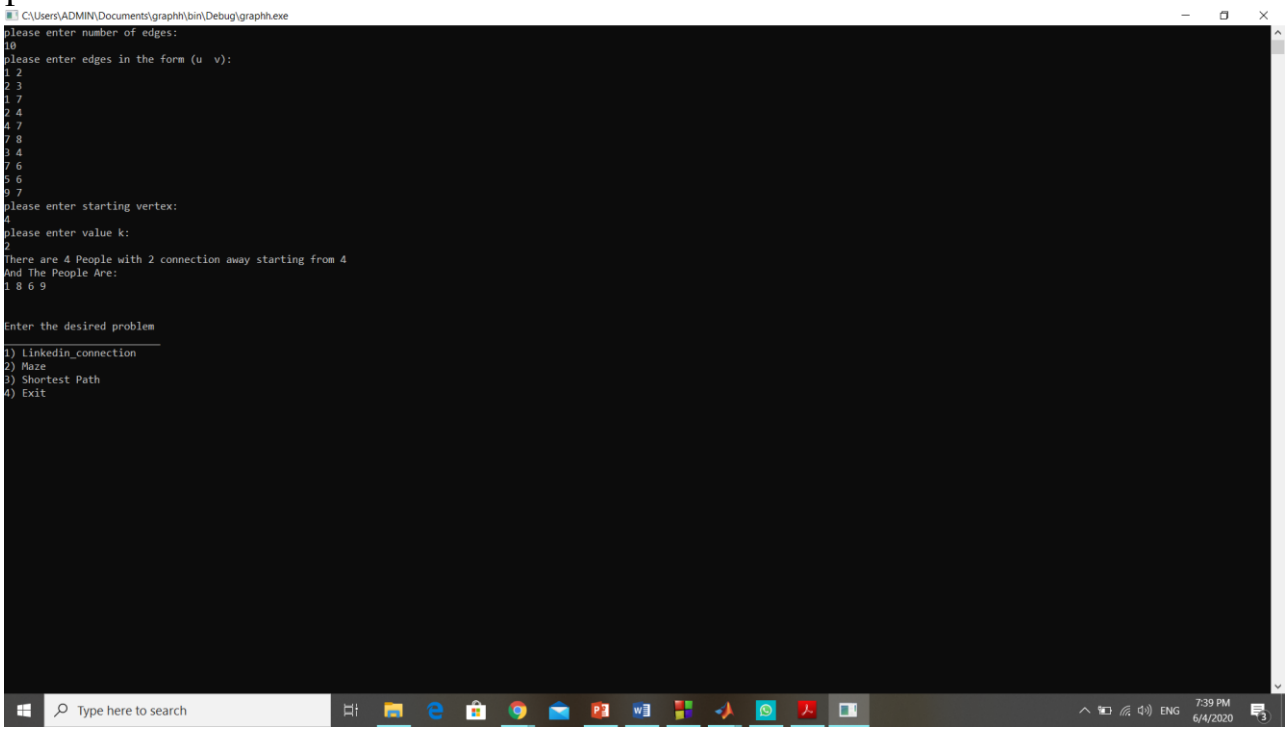


3. finally we create a queue for BFS traversal then assign the depth for each node and check if the given distance from a given node (K) is equal to the depth , then this node is a K distance away from the given node.



OUTPUT:

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



COMPLEXITY:

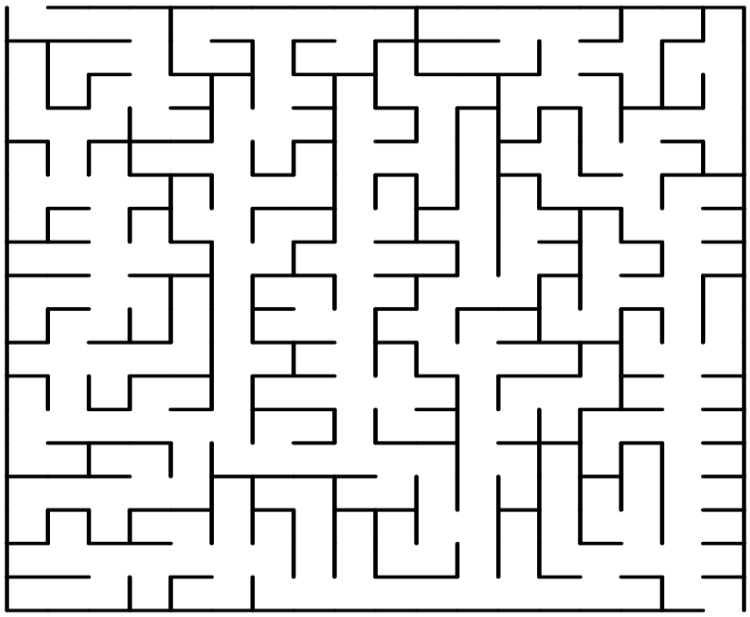
O(|V|+|E|) where V is the number of vertices and E is the number of edges.

DATA STRUCTURES:

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

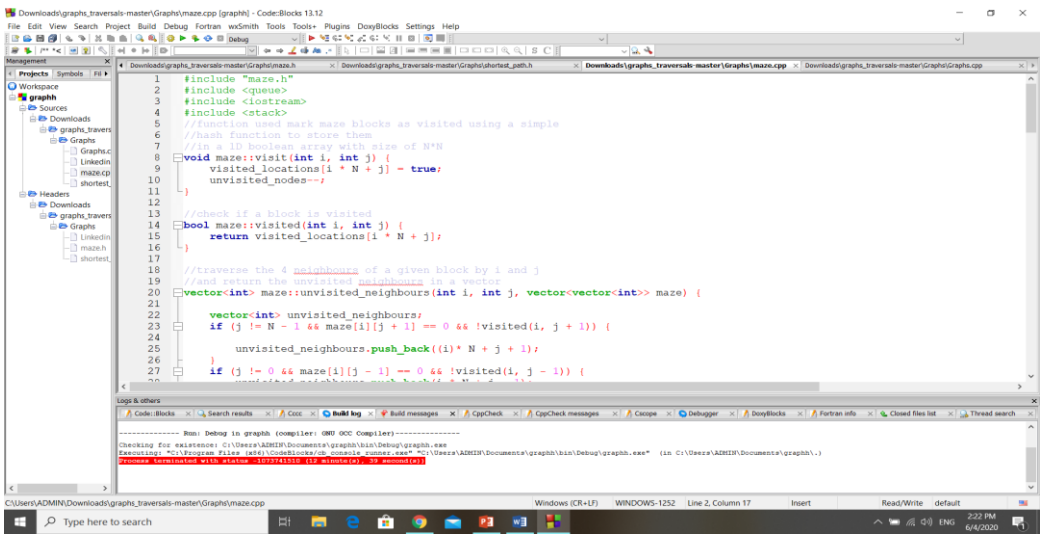
Maze

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.

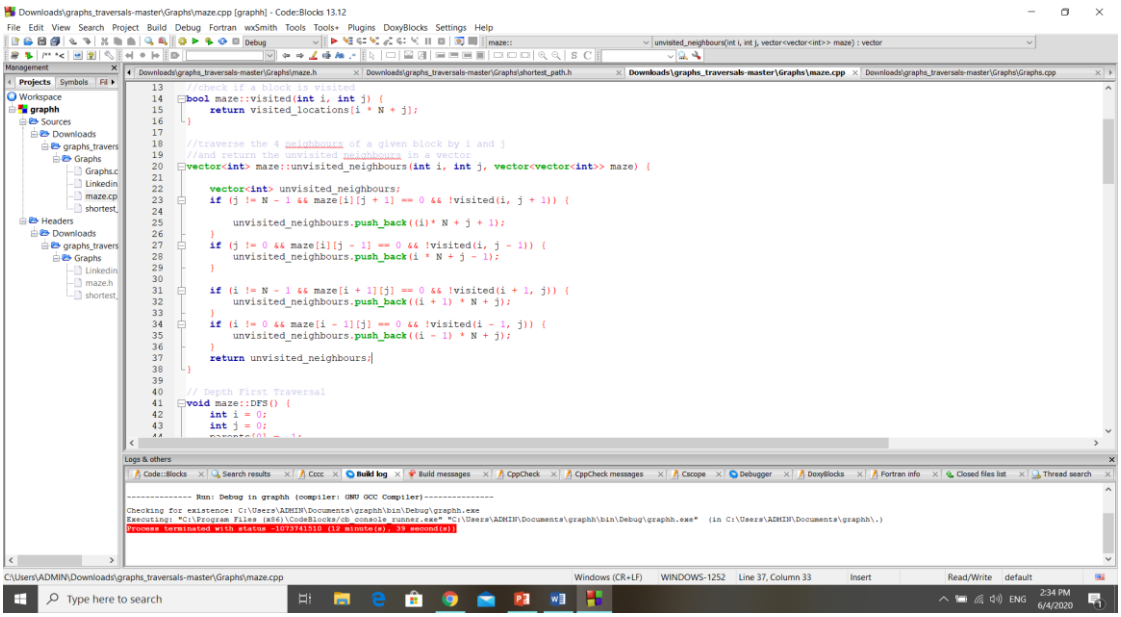


CODE:

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.



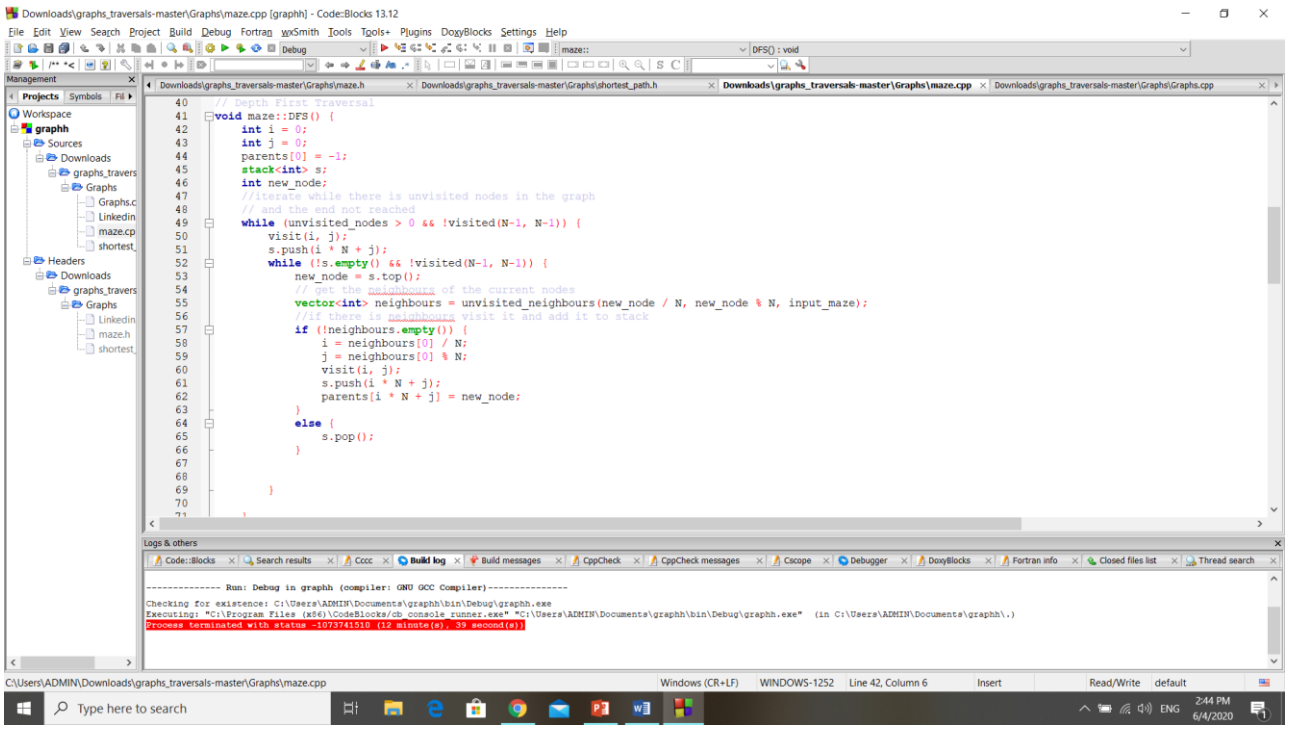
2. Here we traverse all the 4 neighbours of a block in a maze in case of it's not the last or first block in maze and push those unvisited blocks in a vector.



```
bool maze::visited(int i, int j) {
    return visited_locations[i * N + j];
}

//traverse the 4 neighbours of a given block by i and j
//and return the unvisited neighbours in a vector
vector<int> maze::unvisited_neighbours(int i, int j, vector<vector<int>> maze) {

    vector<int> unvisited_neighbours;
    if (j != N - 1 && maze[i][j + 1] == 0 && !visited(i, j + 1)) {

        unvisited_neighbours.push_back((i)* N + j + 1);
    }
    if (j != 0 && maze[i][j - 1] == 0 && !visited(i, j - 1)) {
        unvisited_neighbours.push_back(i * N + j - 1);
    }

    if (i != N - 1 && maze[i + 1][j] == 0 && !visited(i + 1, j)) {
        unvisited_neighbours.push_back((i + 1) * N + j);
    }
    if (i != 0 && maze[i - 1][j] == 0 && !visited(i - 1, j)) {
        unvisited_neighbours.push_back((i - 1) * N + j);
    }
    return unvisited_neighbours;
}

// Depth First Traversal
void maze::DFS() {
    int i = 0;
    int j = 0;
```

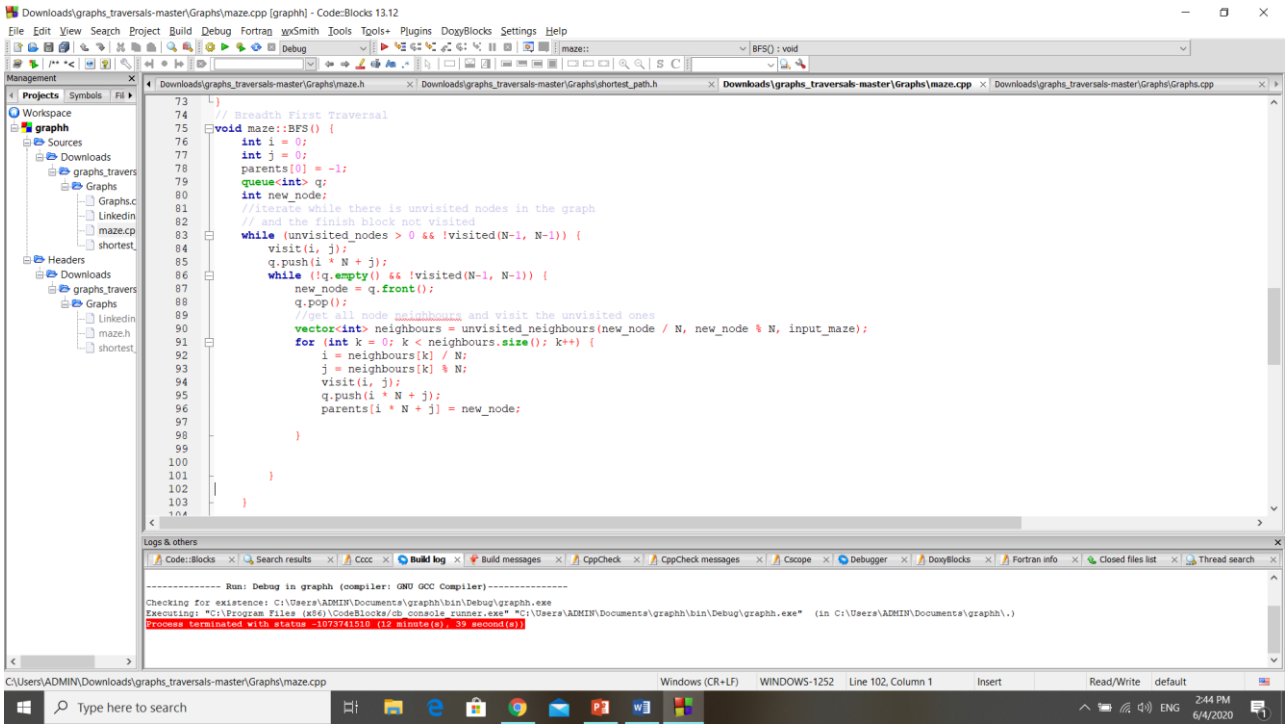
3. In this part of code we iterate in maze to find all the unvisited nodes just if we did not reach the end of maze yet and then after visiting we push them in the stack, and here there are 2 types of traversing DFS and BFS doing the same idea.

DFS

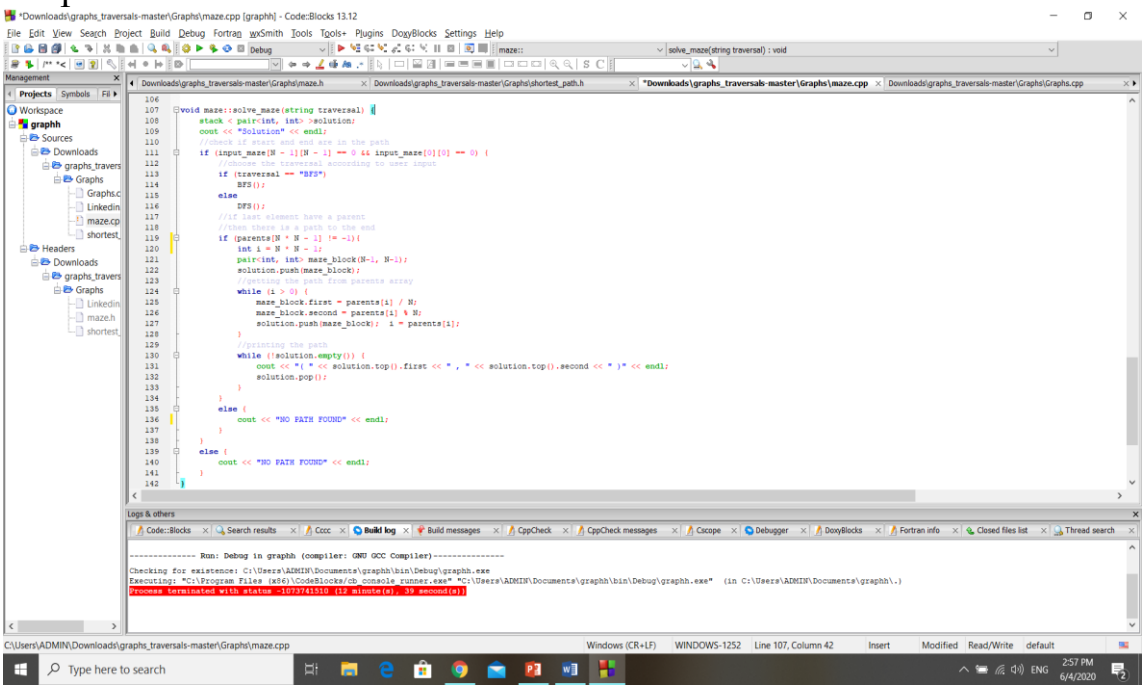


```
//Depth First Traversal
void maze::DFS() {
    int i = 0;
    int j = 0;
    parents[0] = -1;
    stack<int> s;
    int new_node;
    //iterate while there is unvisited nodes in the graph
    // and the end not reached
    while (unvisited_nodes > 0 && !visited(N-1, N-1)) {
        visit(i, j);
        s.push(i * N + j);
        while (!s.empty() && !visited(N-1, N-1)) {
            new_node = s.top();
            // get the neighbours of the current nodes
            vector<int> neighbours = unvisited_neighbours(new_node / N, new_node % N, input_maze);
            //if there is neighbours visit it and add it to stack
            if (!neighbours.empty()) {
                i = neighbours[0] / N;
                j = neighbours[0] % N;
                visit(i, j);
                s.push(i * N + j);
                parents[i * N + j] = new_node;
            }
            else {
                s.pop();
            }


        }
```

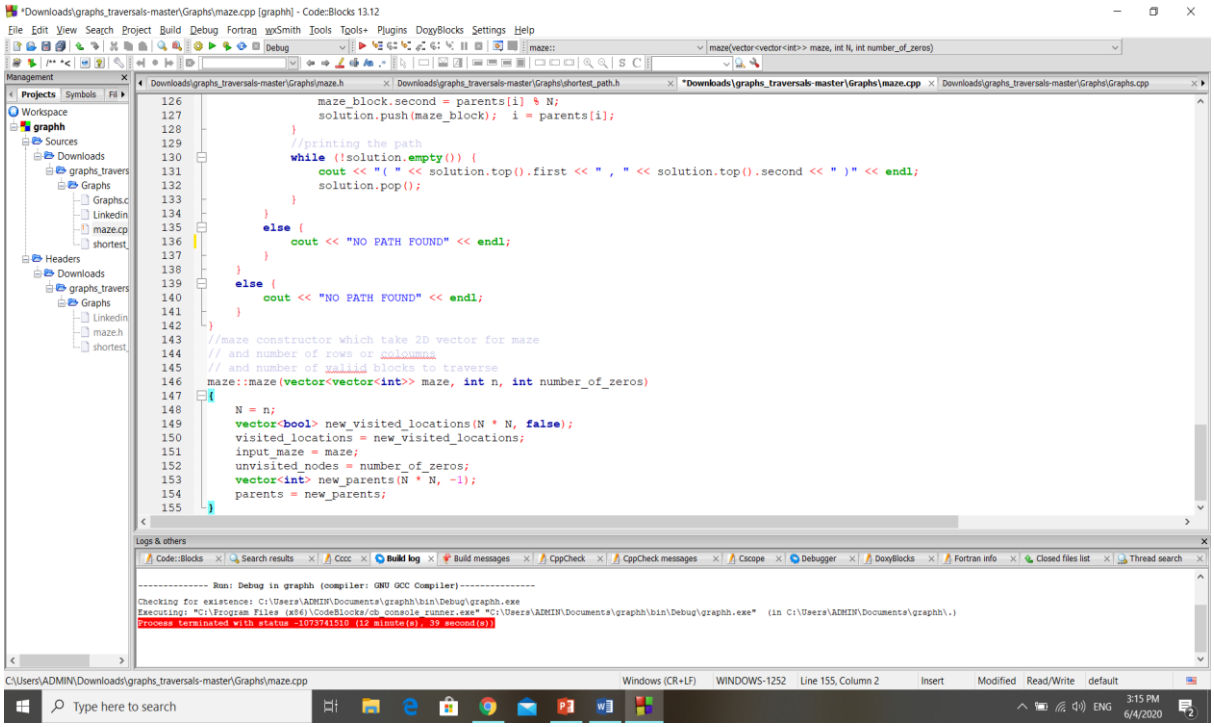
BFS



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".

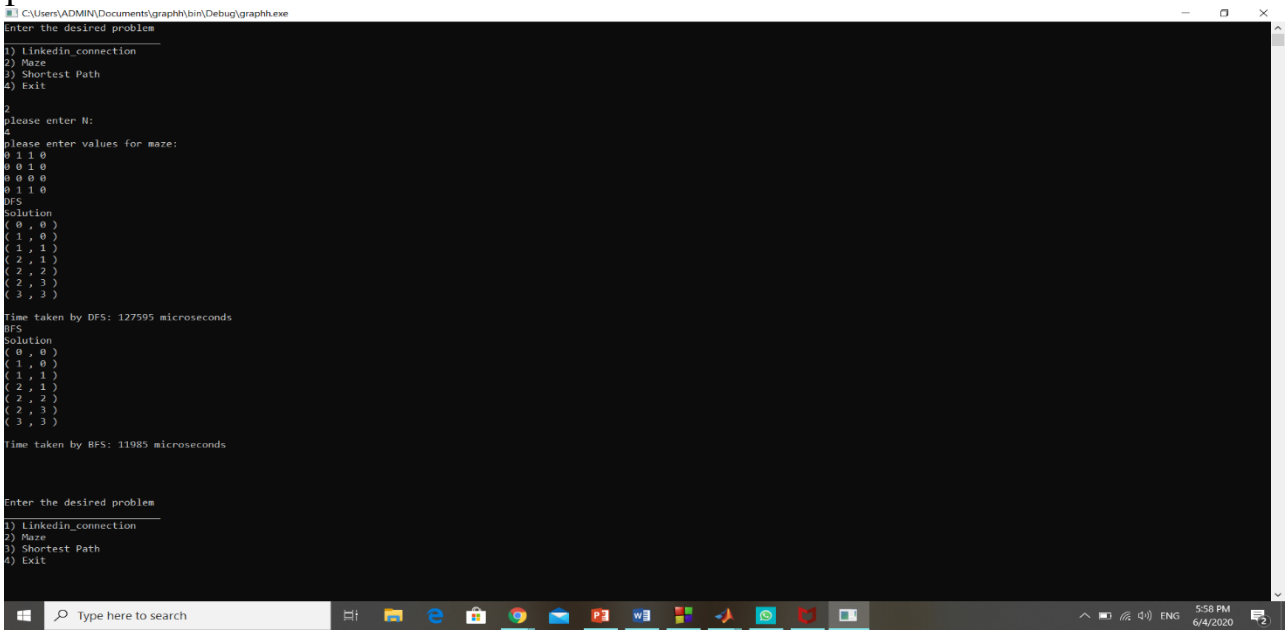


5. At last there is a constructor of 2D vectors of maze and number of rows and columns and number of available blocks to pass through.



OUTPUT:

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



COMPLEXITY:

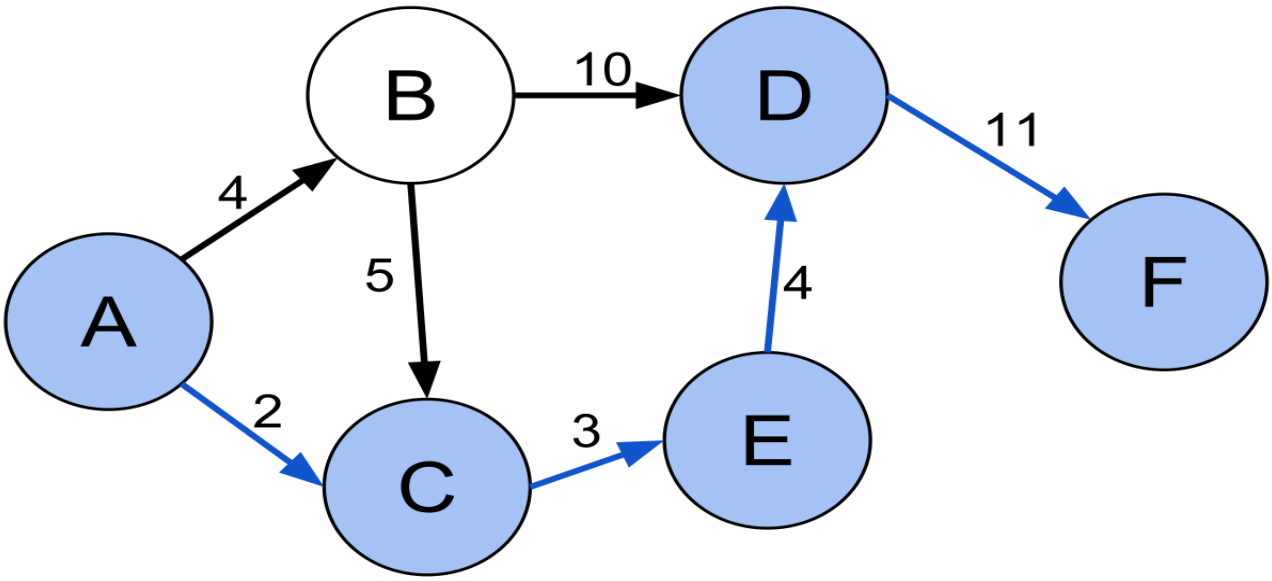
1. O(N^2) where N is the dimension of the matrix of maze.

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*N parents vector that we extract the output path from.
5. N*N Boolean vector for visited function.
6. N*N 2D vector.

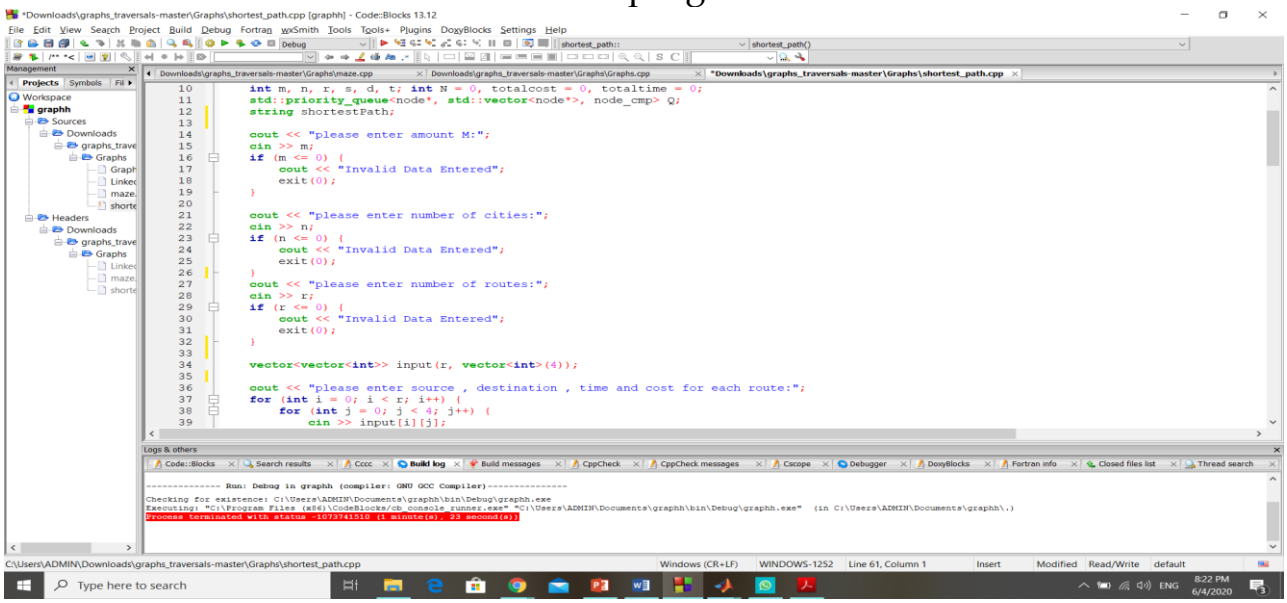
Shortest Path

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.

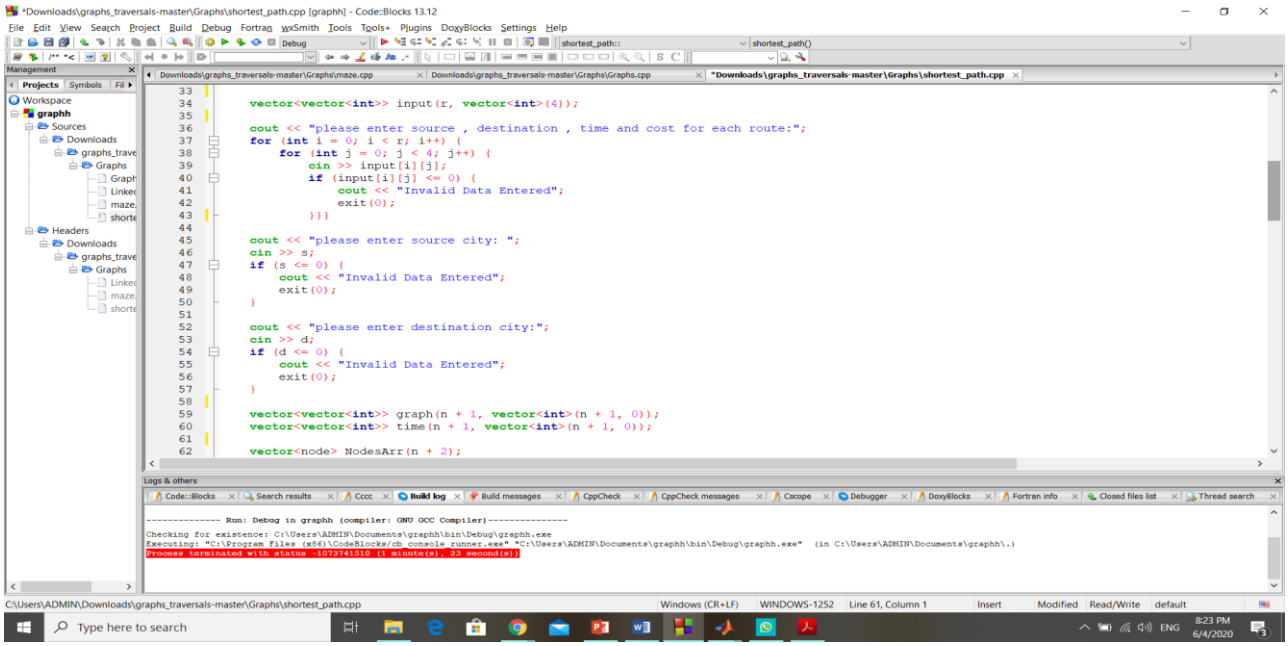


CODE:

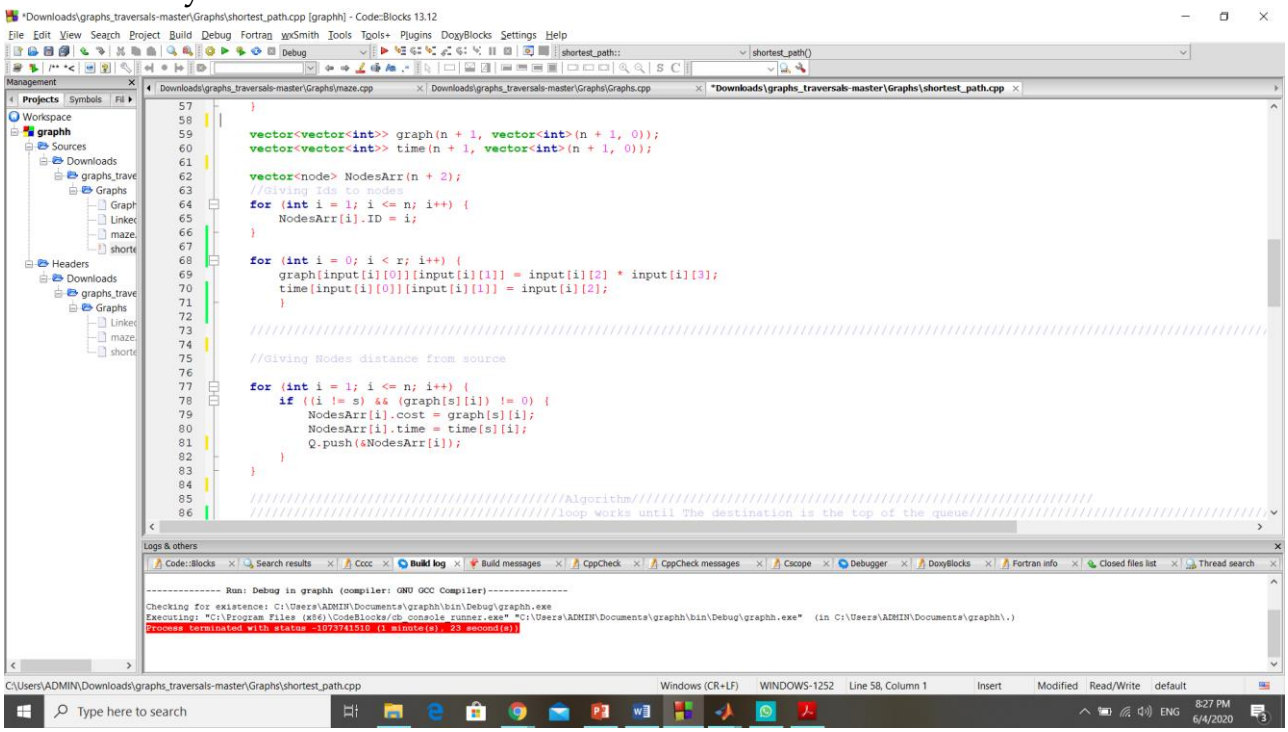
1. Code start with taking input from user and make sure that they are valid under the conditions of the program.



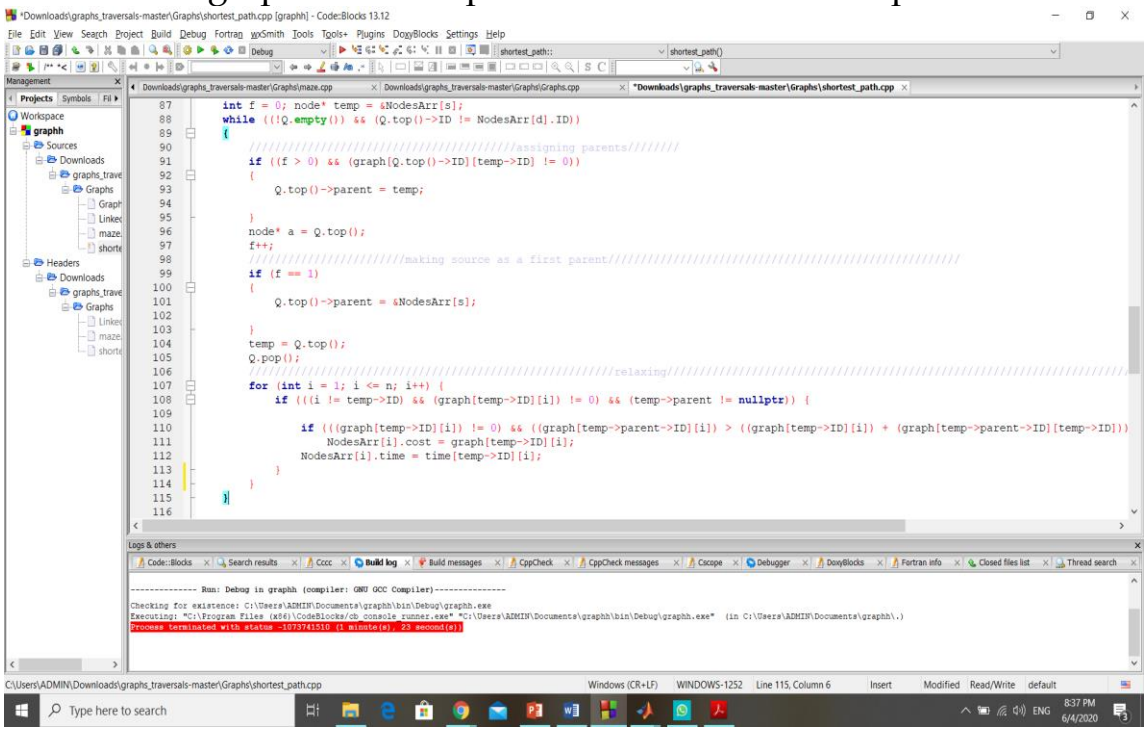
```
int m, n, r, s, d, t; int N = 0, totalcost = 0, totaltime = 0;
std::priority_queue<node*, std::vector<node*>, node_cmp> Q;
string shortestPath;

cout << "please enter amount M:";
cin >> m;
if (m <= 0) {
    cout << "Invalid Data Entered";
    exit(0);
}

cout << "please enter number of cities:";
cin >> n;
if (n <= 0) {
    cout << "Invalid Data Entered";
    exit(0);
}
cout << "please enter number of routes:";
cin >> r;
if (r <= 0) {
    cout << "Invalid Data Entered";
    exit(0);
}

vector<vector<int>> input(r, vector<int>(4));

cout << "please enter source , destination , time and cost for each route:";
for (int i = 0; i < r; i++) {
    for (int j = 0; j < 4; j++) {
        cin >> input[i][j];
```



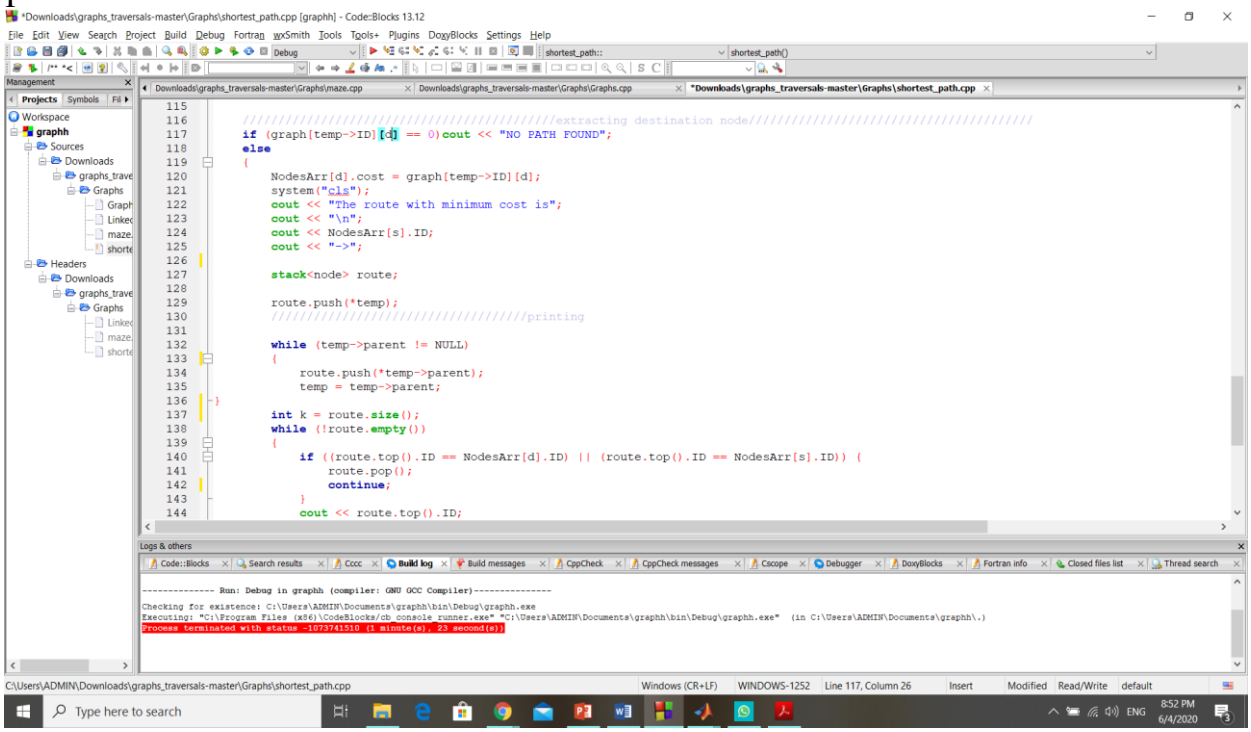
2. We give ID to nodes that represent cities and then give nodes a distance from source by giving them time and cost according to how far they are from the source.

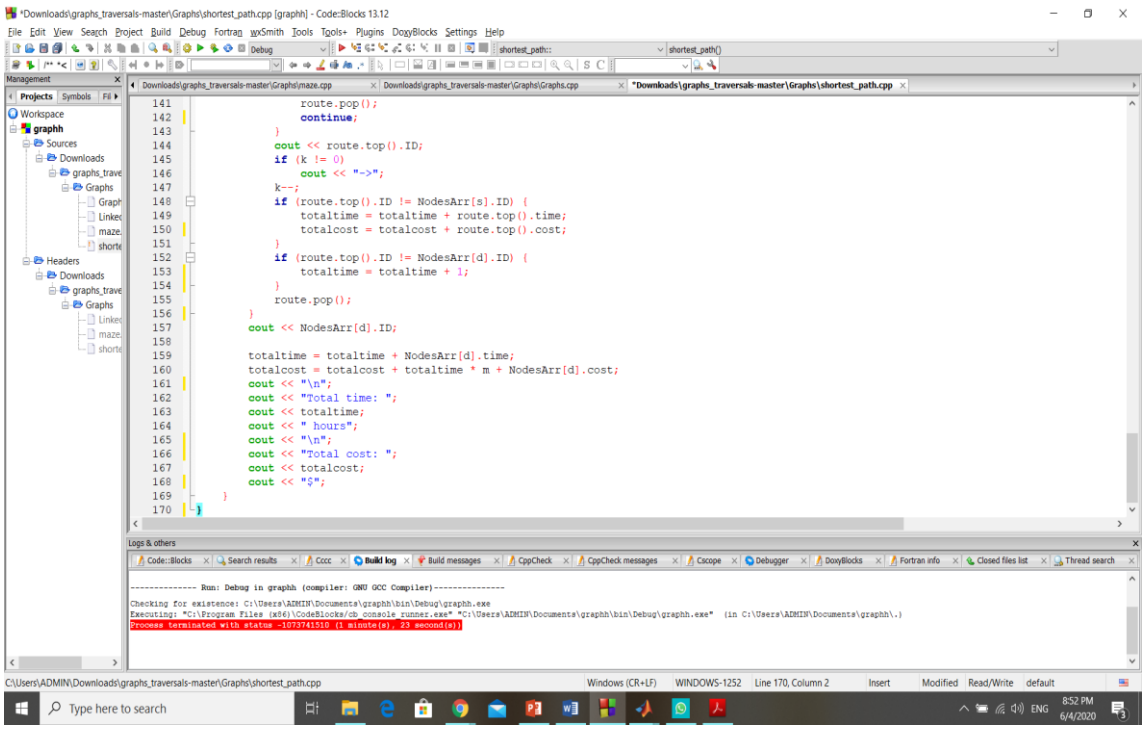


3. Loop keep working till the destination city is the first in the queue then we assign parents and put the source as the first parent.



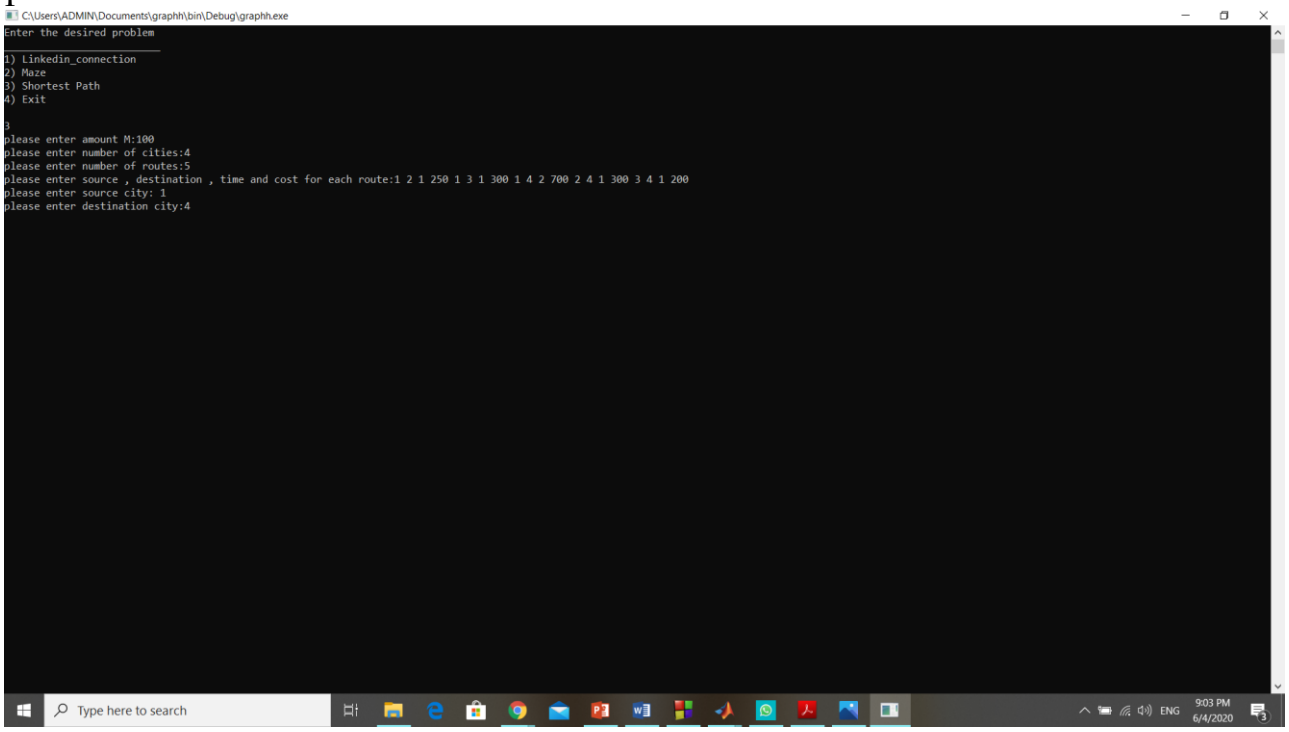
4. Then we extract the shortest path from the destination after that we calculate the total time and cost after finding the required path then print it.

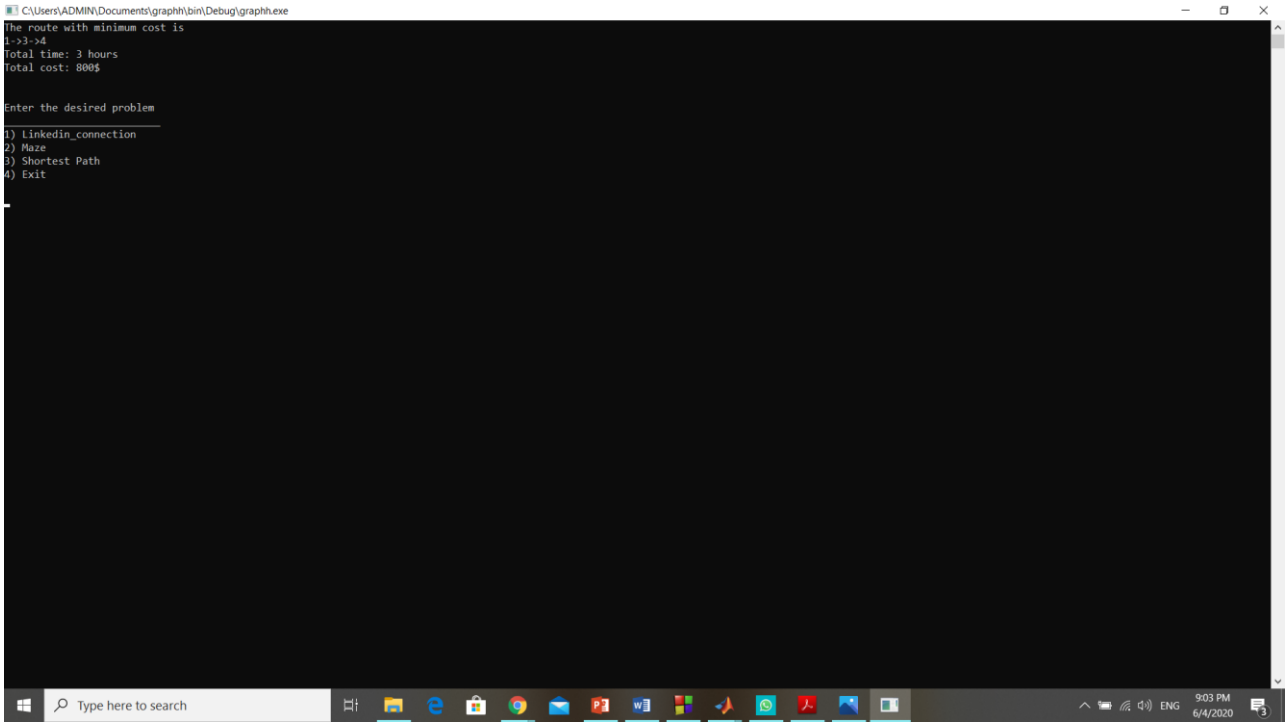




OUTPUT:

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





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