Part 1:

Problem description:

LinkedIn is a professional networking site, people are connected with other people. The whole system appears as a giant connected graph, and when you open any user profile you can see if this user is a connection from 1st, 2nd, or 3rd connection. 1st means I'm directly connected with this user, 2nd means I'm not directly connected with this user, but I'm connected with someone who is connected with this user. 3rd means I'm not directly connected with this user, but I'm connected with someone who is connected with someone that is connected with this user. And so on.

It is required to get total number of people connected at kth vertices away from each other.

The Input will be number of people which will be represented using vertices and connections between them which will be represented as edges of the graph, then the starting vertex and value k.

The Output is the number of people that are k vertices away from starting vertex.

Algorithm:

Breadth First Search was used to get the vertices connected to the source vertex at a certain level.

Data Structures:

- Queue used for the BFS.
- Array of Booleans used to label a certain vertex as visited
- Array of Integers (N) used to store the number of vertices at a certain level away from the source vertex.
- Adjacency Matrix to form the graph.

Pseudo Code:

```
get_connected_people(g,start,k)
    let Q be Queue.
    level = 0        // Integer indicates the level that we are in away from source vertex
    Q.enqueue(start)
    mark start as visited
    while Q is not empty
        s = Q.dequeue()
        for all neighbors w of s in graph g
            if w was not visited
                Q.enqueue(w)
                mark w as visited
                increment N(level+1)
                if level+1 == k
                    increment count
    decrement N(level)
    if N(level) <= 0
        increment level
    return count
```

Running Time:

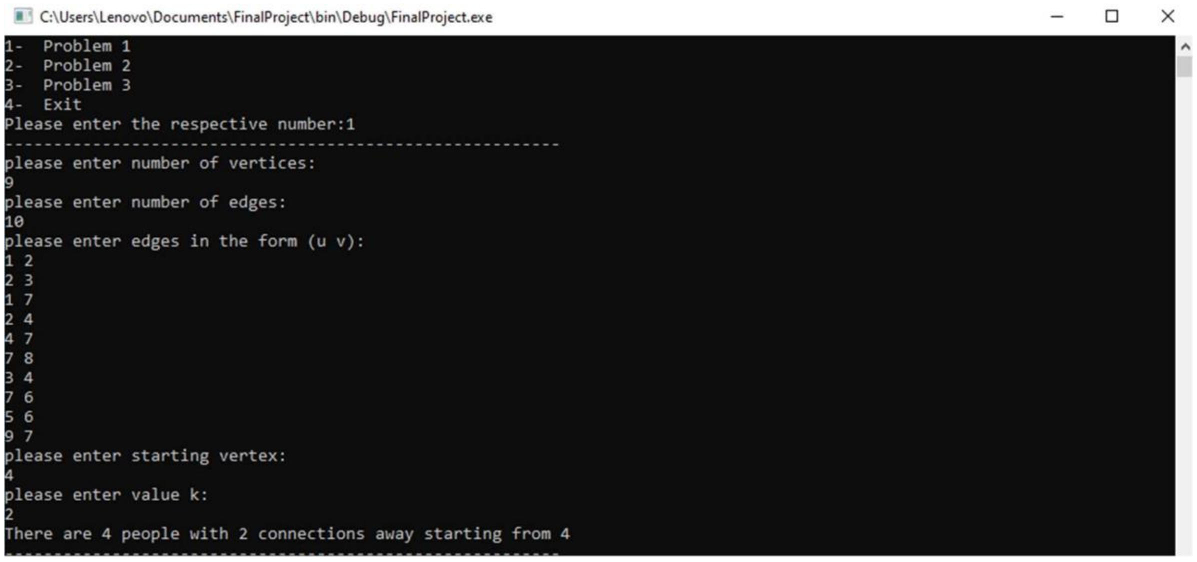
Implementation of the Graph is done using adjacency matrix so:

Time Complexity = O (V^2)

Notes:

- N(level+1) is incremented to signify that the iterations are still done on vertices of level.
- When level+1 is equal to k, that means that we just enqueued one of the required vertices that are k levels away from the source vertex.
- N(level) is decremented to signify that the iterations on one of the higher level vertices are done.
- Once N(level) = 0, that means that iterations on all the vertices in that level are done.

Sample Run:



C:\Users\Lenovo\Documents\FinalProject\bin\Debug\FinalProject.exe

```
1-  Problem 1
2-  Problem 2
3-  Problem 3
4-  Exit
Please enter the respective number:1
---------------------------------------------------------
please enter number of vertices:
9
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 connections away starting from 4
---------------------------------------------------------
```

Part 2:

Problem description:

A runner wants to escape out of a maze. The maze consists of (N x N) cells.

Some of the cells contained blocks, so, runner will avoid those cells to escape. Runner needs to know the path to follow which he can take to escape the maze.

The runner initially starts at cell (0, 0) and the exit is at cell

(N-1, N-1).

The runner can move in four directions Up, down, left or right that is if the way isn't blocked.

If his current location is (X, Y), he can move to either (X+1, Y), (X−1, Y), (X, Y+1), (X, Y−1).

The first line of input contains the size of the matrix "N". and the next N lines contains N space-separated values either 0 or 1. (0 for cells he can move through and 1 for cells he can't move through).

Output is the path that the runner can follow to escape the maze. In case no possible path found print "no path found".

Algorithm:

Breadth First Search was used to get the shortest path possible out of the maze.

Data Structures:

- Queue used for the BFS.
- A struct called Cell with 2 integers X and Y for the position of the cell, and a Pointer to another Cell called parent.
- 2D Array of Integers to signify the cells in the maze.
- 2D Array of Booleans used to label a certain cell as visited.

Pseudo Code:

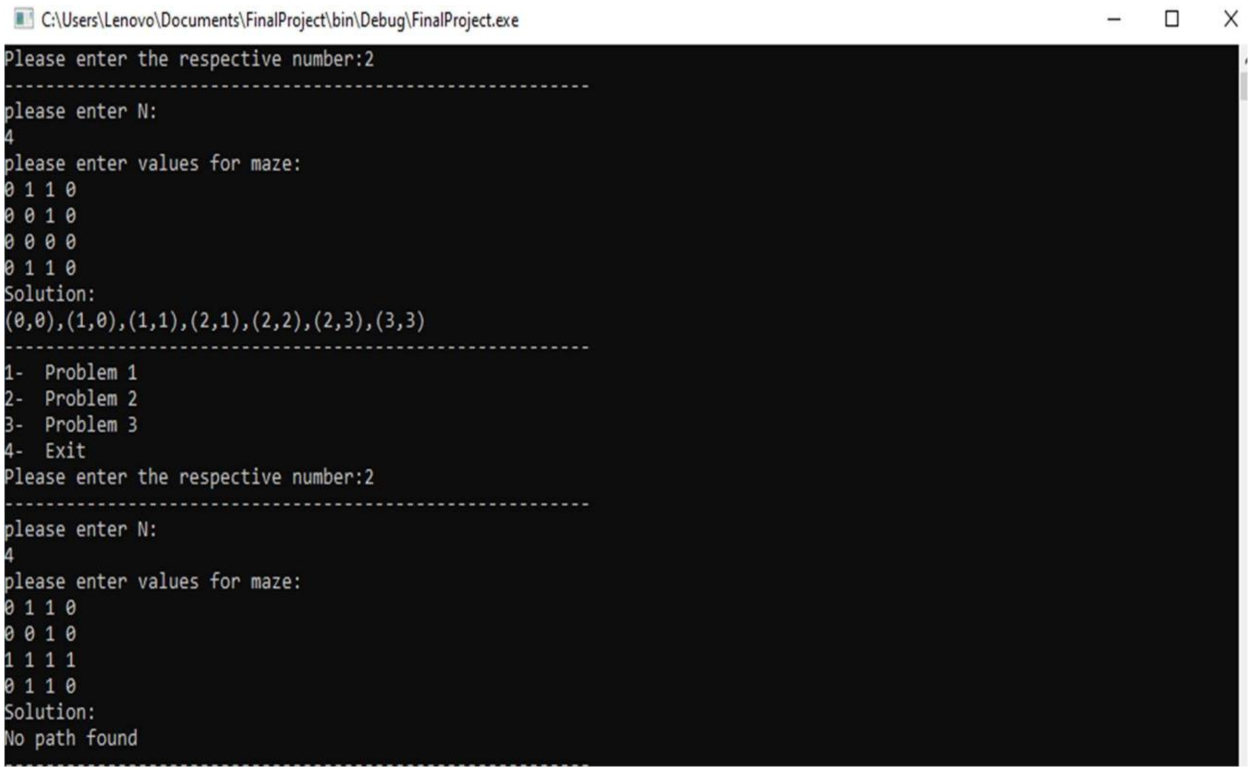
```
getPath(maze, N)
    if maze[0][0] or maze[N-1][N-1] == 1
        print "no path found"
        end
    let Q be the Queue of Cells
    make a cell with positions (X = 0, Y=0) call it starting_cell
    Q.enqueue(starting_cell)
    Mark this cell as visited using the 2D Array of Booleans
    While Q is not empty
        currentCell = Q.dequeue()
        if any of the neighboring cells is legal for movement
            initialize a tempCell with the legal cell's position
            Q.enqueue(tempCell)
            tempCell.parent = currentCell
            mark this cell as visited using the 2D array of Booleans
    if Cell with position (N-1,N-1) wasn't visited
        print "no path found"
        end
    print the path using the parent pointers starting from the cell with position (N-1,N-1)
```

Running Time:

Assume the number of 0s in the maze (number of possible cells to move through) is M.

Time Complexity = O (M)

Sample Run:



```
C:\Users\Lenovo\Documents\FinalProject\bin\Debug\FinalProject.exe
Please enter the respective number:2
---------------------------------------------------------
please enter N:
4
please enter values for maze:
0 1 1 0
0 0 1 0
0 0 0 0
0 1 1 0
Solution:
(0,0),(1,0),(1,1),(2,1),(2,2),(2,3),(3,3)
---------------------------------------------------------
1-  Problem 1
2-  Problem 2
3-  Problem 3
4-  Exit
Please enter the respective number:2
---------------------------------------------------------
please enter N:
4
please enter values for maze:
0 1 1 0
0 0 1 0
1 1 1 1
0 1 1 0
Solution:
No path found
---------------------------------------------------------
```

Part 3:

Problem description:

There are some cities and some routes connecting specific cities (not all cities are connected). For each route between two cities there is a flight with specific time and cost (same for any direction). An Employee wants to travel from city X to city Y and he needs to minimize the cost that will be paid. Every hour that the employee spends during traveling or waiting in the airport for another flight connection, he has to pay M Dollars. Assume that layover time between connecting flights is always one hour.

It’s required to get the path with the minimum cost for the employee during his journey.

Input is:

- Amount M that he will lose per hour.
- Number of cities.
- Number of existing routes.
- Cost and time for each flight between two cities.
- Source and destination Cities.

Output is:

- The route with the minimum cost with the total time and cost.

Algorithm:

Dijkstra’s shortest path algorithm was used to get the shortest path between the source and destination nodes.

Data Structures:

- Priority queue for implementing Dijkstra’s algorithm.
- Array to store the minimum cost to each node in the graph (d).
- Array to store the minimum time to each node in the graph (t).

Pseudo Code:

```
get_min_cost_route(src,dst,adjList,V,M)
    initialize priority queue PQ with the key to be a pair of d values and t values
    d[src] = 0
    d[V] = infinity for all other nodes v
    PQ.push((d[src],t[src]),src)
    while PQ is not empty do
        u = PQ.top()
        for each edge e to adjList[u]
            possibleMinCost = d[u] + e.cost + (M*e.time)
            possibleMinTime = t[u] + e.time
            if possibleMinCost < d[i] or possibleMinCost == d[i] and possibleMinTime < t[i]
                d[i] = possibleMinCost
                t[i] = possibleMinTime
                if i doesn’t equal dst
                    d[i] = d[i] + M
                    t[i] = t[i] + 1;
                PQ.push((d[i],t[i]),i))
                parent[i] = u
    print shortest using parent array.
```

Running Time:

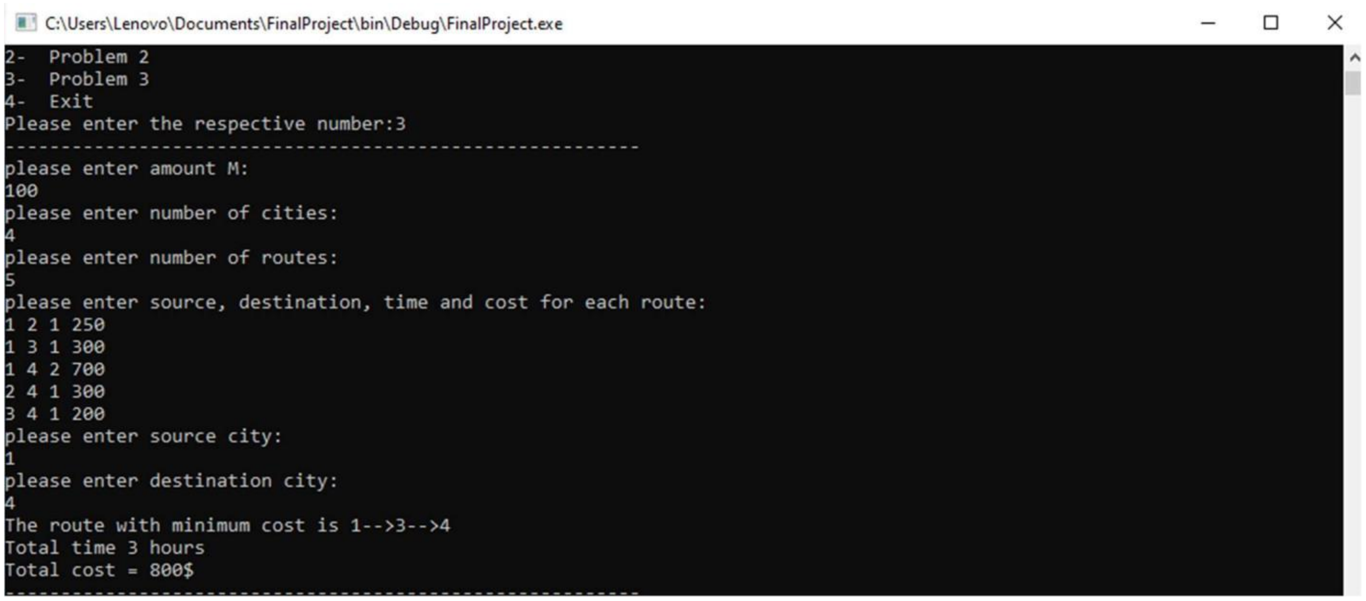
O(V) iterations of while loop

Extract-min per iteration O (V log V)

For loop has O(E) iterations

Time Complexity = O (E log V)

Sample Run:


C:\Users\Lenovo\Documents\FinalProject\bin\Debug\FinalProject.exe

```
2-  Problem 2
3-  Problem 3
4-  Exit
Please enter the respective number: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
The route with minimum cost is 1-->3-->4
Total time 3 hours
Total cost = 800$
---------------------------------------------------------
```