## Work Integrated Learning Programme

## Division M.Tech (Data Science and Engineering)

**(S1-21\_DSECCZG519)**

## (Data Structures and Algorithms Design) Academic Year 2021-2022

**Assignment 2 – PS01 - [Hospital Distance] - [Weightage 13%]**

**Group# G025**

|  |  |  |
| --- | --- | --- |
| Roll Number | Name | Contribution (%) |
| **2021C104029** | **SUBHAJIT PAUL** | **100.00 %** |
| **2021C104035** | **GOURAB BANERJEE** | **100.00 %** |
| **2021C104023** | **RISHABH CHOWDHURY** | **100.00 %** |

**1. Problem Statement:**

# A township has houses and other infrastructure like schools, colleges, hospitals, shopping centers and places of worship, all of these have been built in a grid layout. In a grid graph the nodes are arranged in a lattice or grid, and the edges form a regular pattern.

# Assume that the layout is a square grid, and each cell contains an integer that represents the time to reach from one grid to next grid.

**2. Assumptions:**

# The Grid must be square grid (n x n matrix)

# Each cell must has positive integer number

# The source and destination cell must have “0” value

# Cannot move is diagonal direction between cells of the grid

**3. Requirements:**

# The requirement is to design efficient algorithm to find out the best possible route with the shortest time to reach from the source (House) to destination (Hospital)

# The total time taken to travel from source to destination

**4. Solution Design:**

**4.1 Input File Format:**

Data inputs need to provide in predefined format in a file “**inputPS01.txt**”. The input details should follow below format:

* + Grid/Co-ordinate details input format: [Row#, Column#]
  + Time/Cost need to provide against each co-ordinate separated by “**:**”

E.g., [x,y] : Time/Cost

[1, 2] : 5

[1, 3] :11

* + Source Node/Co-Ordinate:

Source Node: [1,1]

* + Destination Node/Co-Ordinate:

Destination Node: [10,10]

**Pseudocode:**

*Class Graph()*

*constructor*

*Open* ***inputPS07.txt*** *in READ mode*

*Read each line to get co-ordinate(grid) and time associated against each co-ordinate*

*Create a graph with that grid co-ordinate and its’ corresponding time*

*Get Source and Destination Grid*

*Close* ***inputPS07.txt*** *file*

*# function to check whether direction is valid or not*

*Function isValid(x,y,visited)*

*if ((x >= 0 and y >= 0) and*

*(x < self.V and y < self.V) and ((x, y) not in visited)):*

*return True*

*else:*

*return False*

*# function to find diagonals of a given index*

*Function isDiagonal(index)*

*for i,j in <diagonal\_directions> loop:*

*new\_x, new\_y = (index[0]+i), (index[1]+j)*

*if((index[0] >= 0 and index[1] >= 0) and (index[0] < self.V and index[1] < self.V)):*

*index\_dia.add((new\_x, new\_y))*

*return index\_dia*

*# function to find neighbours of a given index*

*Function isNeighbor(index)*

*for i,j in <adjacent\_directions> loop:*

*new\_x, new\_y = (index[0]+i), (index[1]+j)*

*if((index[0] >= 0 and index[1] >= 0) and (index[0] < self.V and index[1] < self.V)):*

*index\_dir.add((new\_x, new\_y))*

*return index\_dir*

*# implementing breadth first search to find the shortest path*

*Function bfs(self,graph,src,dest)*

*visited = set() # visited set to hold indexes for visited cells*

*visited.add(tuple(src))*

*shortest\_path = cols.deque() # queue set to hold indexes shortest path route*

*shortest\_path.append(tuple(src))*

*q = cols.deque() # queue to hold indexes to explore using bfs*

*dirs = [up, down, left, right]*

*x, y = <source index>*

*q.append((<source index value>,(x,y)))*

*while q: # while loop will run as long as there are cells to explore and q is not empty*

*cost, temp = q.popleft()*

*x,y = temp[0],temp[1]*

*if(x,y = <destination\_index>):*

*return shortest\_path*

*min\_cost=infinity*

*index=tuple()*

*neighbour\_set=set()*

*for i,j in dirs:*

*new\_x, new\_y = (x+i), (y+j) # unexplored directions*

*if self.isValid(new\_x, new\_y, visited):*

*# checking whether whether obtained directions as vaild*

*q.append((graph[new\_x][new\_y],(new\_x, new\_y))) # if valid append to q*

*visited.add((new\_x, new\_y)) # if valid add to visited set*

*neighbour\_set.add((new\_x, new\_y)) # if valid add to neighbours set*

*if(graph[new\_x][new\_y]<min\_cost):*

*# checking which direction has min cost*

*min\_cost=graph[new\_x][new\_y]*

*index=(new\_x, new\_y)*

*if(index != () and index not in shortest\_path):*

*neighbour\_set.remove(index)*

*if(index not in self.isDiagonal(shortest\_path[-1])*

*and index in self.isNeighbor(shortest\_path[-1])):*

*shortest\_path.append(index)*

*# adding to shortest path only if cell have min cost and*

*# it is a neighbour of previous index*

*for i in neighbour\_set:*

*if(i != () and i in shortest\_path):*

*shortest\_path.remove(i) # removing any outliers from shortest path*

*Function shortestPath()*

*Call bsf() to obtain shortest path*

*Write output to “outputPS01.txt”*

*g = object of class Graph*

*g.shortestPath()*

**Time Complexity Analysis:**

Time complexity of above code (bfs time complecxity is O(rows x cols))

O(n2)

**4.3 Output File Format:**

Data output would be printed on “**outputPS01.txt”** file.

The output details should follow below format:

Shortest route from the House to reach the Hospital is:

[1,1]

[1,2] ……

[10,9]

[10,10]

Minimum time: 175