Explanation

Name: Md. Mehedi Hasan Tanvir
ID: 2210107

CSE 221
Lab section: 14

#Task 1A: creates 2D array with numpy. Extracted row and collum no from the input along with the weight for this undirected graph. Presented the 2D array as a matrix at last.

#Task 1B: Here I've used a dictionary to create the Adjacency list. The keys of this dictionary are vertices and it stores what other vertices it is connected to along with the edge weight.

#Task 2: For this BFS traversal, I've taken an array as queue, another one as a way of keeping track of the already visited vertices, alongside with the Adjacency list, which is actually a dictionary. We start from 1, make it lebeled visited and check its vertices connected to it, if they aren't visited they are queued and lebeled visited. Lastly the parent is popped.

Scanned with CamScanner

#Task 3 : I took a dictionary to keep track of visited nodes. We start from 1 and we put 1 inside that dictionary's key 1 to label visited. Then check it's neighbour one by one, but if we find unvisited we recursively do the same process.

#Task 4 : Here Ive used DFS to find out if the graphs provided have cycles in them. I'm keeping a boolean to track if we ever reen it into a cycle. I'm running DFS from each vertices one by one and keeping a DFS array that resets after each iteration. However it checks if any later vertex matches it's first value on current iteration. If it does that means there is a cycle.

#Task 5 : Here I'm using good old BFS but with a parent array added to it. It registers each vertices's parent while running BFS. After finishing BFS simply backtracking the parents and then reversing them gives us the shortest path from one node to another.

Scanned with CamScanner

#Task 6 : This is a flood fill problem with some added complexity. I took the whole input as a matrix. Then used flood fill on each node one by one. I copied the original matrix each time I did this. On this new matrix along with implementing flood fill on basis of '.' and 'D', i counted total diamond reached/collected in that iteration. compared that with the previously stored max.

#Task 7 : Here using the ideas of DFS first I found out the furthest node from node 1. Then found out the furthest node from that node. These two nodes are the the answer.

#Task 8 : Here Ive used the techniques of bipartite graph and BFS. Made vampires black colored and lykans lime colored. No two ~~colors~~ node of same colors are connected to each other. counted the number of vampires and lykans, compared and maximum number is the answer.

Scanned with CamScanner