19201089
Section: 9

Task 1A

Here, we get the number of nodes and vertices from the input file and build an empty matrix based on them. Then we iterate through the rest of the input file to get first node, second node and the cost between them. Then this cost is set in the matrix for the two nodes in this way mat[node1][node2] = cost

Task 1B

Like the previous task, we build an empty dictionary consisting of empty lists according to the number of nodes and vertices from the input file. For every configuration in the input file, the array of the node 1 in the dictionary is updated with value of node 2 and cost to go to node2 from node 1.

Task 2

We are given undirected unweighted graph to find path traversal. Inside the BFS function, we have a visited array and a queue array, path array. The visited array is updated with a node when we come across a node that had not been visited before. The queue has the newly visited nodes. While the queue is not empty, we pop a node from it and push it to the path array. We then push the neighbours of the node into the visited array or and queue if it is not visited. This continuous until the whole graph has been traversed.

Task 3

The DFS function is a recursive function that for every unvisited neighbour, calls DFS again, marks the node as visited and pushes it into the path array. This way, the graph is traversed branchwise.

Task 4

The isCycle function calls the DFS function for every node in the graph. The DFS ~~graph~~ function maintains two arrays - visited (the overall visited nodes) and the stack (nodes found in current traversal). For every neighbour of the node, the DFS function is recursively called if it is not visited. This is to check if a cycle exists and if it possible to go back to the original node from another node. After each node is checked, the node is ~~rem~~ not included in the stack. If a cycle is detected, the DFS function returns True, which causes the isCycle function to also return True.

Task 5

Here, we find every path possible to take from the starting point and check if the end node is reached. There are two arrays - pathList (holds all possible paths), ~~pre~~ visited (holds previously visited nodes to prevent backtracking). For every pathlist, we take the last node, get all its children and check if the end node is among there. If not, and if the node is unvisited, a new path is added to pathList containing the previous path and this node added. It is also set as visited. In this way, we check paths till we find the shortest one. Time is just number ~~of~~ of nodes - 1.

Task 6

We use recursive DFS to explore the jungle. It starts from a cell and traverses the neighbour cells in all directions. The encountered diamonds are added to a diamond count variable. The visited cells are marked to prevent backtracking. # are obstacles and are avoided. The countDiamonds function iterates through all cells, calling DFSFloodFill for all empty (.) cells to find the maximum diamonds. The maximum of this result is stored.