Task 1 (a):

First we create an empty edge list where we stored row, col. and value as a tuple from input file and then create an empty matrix and for spacific row column we put the value on the matrix.

Task 1 (b):

Same as previous, we create edge list and then create a dictionary considering key as number of vertices from 0 to N+1 and then we iterate through the edge list and put the column value at its rows keys as considering tuple which are values of the dictionary.

And return the adjacency List.

Task 2:

Firstly, we created a visited ~~list~~ dictionary and initially make them all false as the keys are the vertices. From the input file we create a graph dictionary where keys are vertices and values are bidirectional road.

Now, We create a queue and from queue we pop an item and checked its children if not visited ~~true~~ visited and make visited dictionaries value as it's vertices as true and append in queue and same ~~the~~, again until the graph get all visited and it will be level by level.

Task 3:

~~For BFS,~~

For DFS, we do the same as previous but there we will use stack and like BFS we took a vertices and will go until there are no connection and now we backtrack the vertices and will check if any unvirited vertices are left off, if the we do same as dfs.

The difference of BFS and DFS is, in dfs we go into the last vertices and then backtrack where in BFS we check one's children and from children to it grand children.

Task 4:

For cycle detection, we will use dfs as it goes to the last of a vertices.

As the previous DFS code, now I will modify as we create a parent dictionary where keys are vertices and values initially None or Nay, same as previous if node not visited we visit it and call DFS and it will check it adjecence and if the source node is not the parent's adj and its already in stack that means there we found the cycle. Otherwise we visit the source and remove from stack and return False.

Task 5:

For shorted path we consider BFS as a graph traversal. From BFS code, here we created a queue and put the start and end value as pair and create a set of visited value. Now, we dequeue from the queue and add to the visited list. Now we iterate through the neighbor of every node of the graph, if not visited yet we will append in queue as First value is the neighbor and a list of nodes start and end values and added to the visited set. If we found node is same as the end which is provided then will return the path value. and the time will be the length of the list - 1.

Task 6:

Firstly, create a array where each rows store.

Now, we use dfs to explore the grid starting from each empty cell (.) and recursively traversing its neighbor cells: its up, down, left, right wise.

When traversing, it counts num. of diamonds when "D" found while not visited or pass through obstacles (#). max-diamond count = the all backtrack possible starting position and count it.