TASK-1

PART-A

We used numpy to generate a 2D array. The array was then filled using a loop. We read each line in the loop and filled the array with the data.

PART-B

We used a dictionary to track the edges and the vertices of the graph. A conditional expression is used to decide whether to write an empty line or the joined string of edges for each vertex.

TASK-2

Here we used the BFS algorithm. We use a queue and start from the source. After exploring all the child of that source we enqueue the childs and mark them as visited. Then we dequeue the queue and explore its child and mark it as visited. This procedure repeats until the queue is empty.

TASK-3

The DFS algorithm is used here. It starts from the source and visits all the nodes till the end of the path . then it returns to the previous node and checks if any other path exists or not . These steps are done recursively.

TASK-4

A list is initialized to represent an adjacency list for the graph. We used the DFS algorithm to check for cycles in the graph. A global variable is used to track whether a cycle is detected during the DFS and writes the result in the output file.

TASK-5

We made a slight modification in BFS. We kept two arrays named parent and distance. Each time it visited a node these two variables kept the track of distance and parent. We loop through the parent array from destination. Basically , we did a backtrack to find out the shortest path.

TASK-6

The problem is solved using DFS. First we check if the row and column is in the bound or there is an obstacle or it is already visited. Then we mark the path as visited and if there is a diamond , then the diamond variable increases by 1. Then recursively call the DFS for adjacent up, down, left and right. In the maximum diamond function , a variable is kept to keep track of the visited paths.