Task 1a

To solve this task, at first made a 2D array with zero. Then used a for loop and write the weight according to their row and column ~~nu~~ numbers.

Task 1b

To solve this task we just need a dictionary and the key will be the start number and the value will be a list where the end point and weight will be stored in tupple.

Task 2

To write the bfs code, at first we need ~~to~~ two array. The first one will initialize with empty array and the second one will initialize with zero of number of nodes. Then in the first array ~~initialize~~ append zeros of number of nodes and number of nodes time using for loop. The run another for loop and mark 1 in the first array which will let us know ~~the~~ the start and end point. then call the bfs function. Start from 1, then append it in the q list. Now run a while loop until the q is not empty. Now, ~~remove the~~ pop from q and append to the out list. Run a for loop in the first array, if there's 1 and ~~it~~ in the second list that place isn't 1, then append in q and marked 1 in the second list. In the out list we will get our traversal.

Task 3

To get the dfs traversal, we need a dictionary, where start and end, also from end to start will store. Then we need two list. Call the dfs function and start with 1. At first check if that number is visited or not, if not then mark it as visited and append in out list. Then run a for loop in the dictionary of that key number and call the dfs fuction again.

Task4

To find if there is a cycle or not, at first we need a dictionary and store the start ~~as~~ value as key and all ~~ed~~ end value from that start will store as a list. Then initialized two arrays, visited and path. The call the has_cycle function, if that is not visited. In the function, at first mark that visited and path as True. If that vertex is in the dictionary, then run a for loop ~~and~~ in the values of that key. If that value is not visited then call the function again. otherwise if path of that neighbour is true and neighbour is not parent then return true.

Task 5

To find the shortest path, we need two list,. The first one is the adj. matrix and the second one is ~~the~~ for path value. Now run the bfs function. Here we create a ~~are~~ list to store the path. Then simply run queue and update the second list as the total time. And return the second and third list.

Task 6

To solve the flood fill problem, at first we need a list like adj. matrix row wise of the input. Then call the findmax function. Now, make a visited list and initiate them with false. Call the floodfill function from where we can know where will we go and how many diamonds we will get.