Discrete Structures for Computer Science (CS F222)
Assignment - 1
First Semester 2023-2024

Group 13

Varun Sai Sajja 2021B3A71054P

Sai Rithvik Padma 2021A7PS1632P

Yamsani Hari Charan 2022A7PS0042P

Q1.

The question given is to find out the number of junctions in the graph and a junction is a station that is at least connected to 4 different stations.

We can find the number of stations one particular station is connected to by adding up all the values in the corresponding row in the adjacency matrix. We do this for every Station index and the keep increasing the junction count if the sum adds up to 4 or more.

Q2.

In the second question we had to find out if the graph has a tour which starts at a particular station, goes through every track exactly once, and ends at the same station or a different one. That is, we basically have to find out if the graph has a Euler path or a Euler circuit.

We can test this by checking the number of odd degree vertices the graph has. If this count is 0 or 2 then there is a Euler path and there is a Euler circuit is there are no odd degree vertices. We find out the degrees of vertices by adding up their respective rows or columns in the adjacency matrix and check if it is even or odd.

Q3.

The question is to find the number of impossible to travel city pairs. To find this we take the help of the transitive closure of the adjacency matrix and see if the entry corresponding to the city pair in this matrix is zero. The transitive closure of the matrix is calculated using the Warshall's algorithm in the function warshall(). In this function we set the value corresponding to (a,c) to 1 in the matrix if (a,b) and (b,c) are 1 for some b, and we repeat this process as many times as the number of vertices. The transitive closure matrix has the entry (a,b) to 1 if a and b are connected. So if (a,b) is 0 they are not connected.

Q4.

In this question we need to find the number of vital tracks. We do this by removing the track or edge, say (a,b), from the adjacency matrix and the see if the same entry of (a,b) is 1 in the transitive closure of the modified matrix. If the value is still 1, that means that the track is not vital and removing it maintains the same connectivity.

Q5.

The question to determine if it is possible to upgrade the railway stations in a way that satisfies the constraints, we can use a greedy approach. We start by assigning a restaurant to the first station and a maintenance depot to the second station. Then, for each remaining station, we check if it is connected to a station with a restaurant or a maintenance depot. If it is, we assign the opposite upgrade to the current station. If not, it is not possible to upgrade the stations and we return -1.

Q6.

To find the distance between two stations we us Breadth First Search traversal. We first assign all the distances as -1 in a new array also keep a track of the visited stations using another array. We then start at the given city_x and the by breadth traversal we go through it connected vertices and keep updating the distances. We also us a makeshift queue made of an array and two variables front and rear that point in the array. When ever we find the connected vertices of the vertex, we are in we add them up to the queue and then we remove the first element of the queue to explore it next. We keep updating the distance array on every iteration and return it at the end.

Q7.

In this question we need to find out the Railway Capital which is the station that has minimum possible sum of the distances from all the other stations. For this we loop through each and every station pair and find the sum of distances from other stations for all the stations and the we return the index of one of the stations that has the minimum sum of them all.

Q8.

Given a source we have to find is there is a path from the source so that it ends at the source, does not visit the same station twice and also does not travel a track two times or more.

To determine if it is possible to have a such a path starting from a given source city, we can use a modified version of Depth-First Search (DFS). We start from the source city and recursively explore all possible paths, making sure not to visit the same city twice. If we reach the source city again and have visited all other cities, we return true. Otherwise, we return false.