MEDIUM

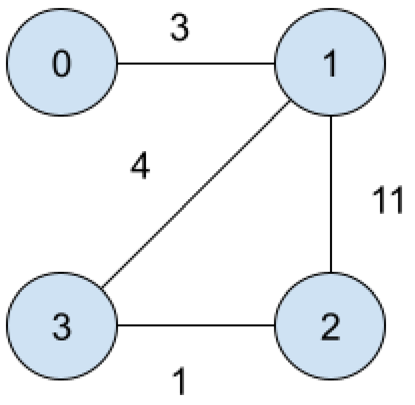
Find City with Smallest Number of Neighbors at Threshold Distance

Intuition

[From , to , weight] , threshold and n are given

Eg.

Threshold = 4



N = 4

// under the threshold we can reach
0 : 1, 2
1 : 0, 3, 2
2 : 3, 1, 0
3 : 1, 2

Therefore maximum can be gone using 1 2 ie. 3

The lowest number of cities can be traversed through 0 and 3 and 3 is greater so we have to return 3.

We can solve it using the Floyd Warshall Algorithm

We will apply the floyd warshall over it and get the distance and will generate a 2-dimensional matrix that will contain the shortest distances from one node to other ie. from one city to the other city and then we will use it to mark the cities that will be under the threshold

Eg.

0	3	4	5
3	0	1	2
4	1	0	1
5	2	1	0

We can say that

0 : 3-1 : 2 // equal to minimum update minimumcity
1 : 4-1 : 3 // greater move forward
2 : 4-1 : 3 // greater move forward
3 : 3-1 : 2 // equal update city

Approach

- Create a distance vector and initialize all with infinity
- Traverse through edges :
    - Mark reachable cities with their respective distances
- Mark the node to node distances as 0
- Perform the Floyd Warshall algorithm to find the minimum distance to reach
- Create a variable to store the citycount of minimum number reachable
- Create a variable to store the city having citycount
- Traverse through the distance vector and perform a count of the threshold or less valued cities :
    - If citycount<counter :
        - Update citycount with counter
        - Update city with current countercity
    - If citycount == counter :
        - Update city with current countercity
- Return city

Function Code

```
int findCity(int n, int m, vector<vector<int>>& edges,
                 int distanceThreshold) {


    // creating a distance vector initialized with infinity
    vector<vector<int>>distance(n,vector<int>(n,INT_MAX));
    // traversing through the edges and marking them on distance vector
    for(auto it:edges)
    {
        int from = it[0];
        int to = it[1];
        int weight = it[2];
        distance[from][to] = weight;
        distance[to][from] = weight;
    }
    // marking the node to node distance as 0
    for(int i=0;i<n;i++)
    {
        distance[i][i]=0;
    }
    // Performing the floyd warshall algorithm
    for(int via=0;via<n;via++)
    {
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                // continue if not reachable
                if(distance[i][via]==INT_MAX ||
distance[via][j]==INT_MAX)continue;
                distance[i][j] = min(distance[i][j], distance[i][via] +
distance[via][j]);

            }
        }
    }
    // Creating variables to store city count and city number
    int citycount  = n;
    int citynumber = -1;
    for(int i=0;i<n;i++)
    {
        int counter = 0;
        for(int j=0;j<n;j++)
        {

            if(distance[i][j]<=distanceThreshold)
            {
                counter+=1;
            }
        }
        if(counter<citycount)
        {
            citycount = counter;
            citynumber = i;
        }
        else if(counter==citycount)
        {
            citynumber=i;
        }
    }
    return citynumber;

}
```

Time Complexity

O(N^3)