ASSIGNMENT 5 – DIFFICULT PROBLEM
180532 & 180653

NOTE:

It can be seen that the graph we define by setting edge weights as road distance and vertices as the junctions does not follow the Optimal Subpath Property. Because if there is a shortest path from s to v passing through (x,v) as the last edge, it is not necessary that shortest path from s to x is the one included in P(s, v) because there may be another shorter path to x from s but through which we cannot go to v because tank fuel finishes before reaching from x to v. Also, there may be cycles as well, for example we cannot go from a to b because of less fuel, but we can go from a to c where c has a station and go back to a and then go to b. This means we cannot use Dijkstra's Algorithm directly here. It can be seen that the junctions where there are petrol pumps cause ambiguity otherwise, we can find optimal path (if exists) from s to all vertices if none of them have any fuel station. So, we do that only.

ALGORITHM:

First, we find the paths from s starting with c fuel and apply Dijkstra's algorithm to find shortest path to each fuel station(and destination if possible) with the amount of fuel we have **without crossing any fuel station**, i.e. we stop at the station vertices and don't take paths from there to other vertices. Now, for each vertex which has a petrol pump we repeat the same process and try to find paths from there to all other fuel stations and the destination that do not cross any fuel stations and are reachable with c fuel. After doing this we have some trees, each tree denoting the shortest distance from each fuel station to d and other stations. Now, we are interested in only the source, destination and the vertices with stations. We merge the trees to get a graph with source, destination and fuel station vertices and put edge values of distance from stations to d and other stations calculated earlier. Apply Dijkstra's algorithm on this new graph to find the shortest path from s to d.

In steps:

- First find the shortest distance from s to all other fuel station vertices (and destination if possible) with c fuel at the start such that none of the paths cross any other fuel station.
- Repeat the above process by considering each fuel station as a source and find its shortest distance to each of the other stations and destination without crossing if reachable with c fuel.
- Now we have the shortest distance from each fuel station including source, to all reachable fuel stations and the destination if reachable.
- Then we merge to form a new graph with the source, destination and fuel station vertices as the new vertices. Edges and their weight can be defined by the distance between from each station and source to all other stations and destination if reachable calculated in our previous steps.
- Apply Dijkstra's Algorithm in this new graph to find the shortest feasible distance from s to d without any constraints. Note that all edges here must be having weights less than or equal to c and if d is not reachable from s, then it means d cannot be reached from s in our original scenario.

PSEUDOCODE:

```
Shortest_Feasible_Path(G, s){

    for (each v in V) {
        L(v) = Infinity;
        F(v) = 0;
    }

    L(s) = 0
    //Build a min-heap H for storing L(v) for all vertices v.
    //Set F denotes the vertices which have fuel stations, including source s.

    for (each v in F) {
        Fuel(v) = c;
        for(each i from 0 to |V|-1){
            y = ExtractMin(H)            //vertex in heap with minimum L(v).
            move y from H to S.
            for (each (y, v) in E, with v in H and y not in F) {
                if(Fuel(y) >= w(y, v)) {
                    L(v) = min(L(v), L(y) + w(y, v));
                    if(L(v) is now L(y) + w(y, v))    Fuel(v) = Fuel(y) - w(y, v);
                }
            }
        }
        //Initialize all Fuel values for all vertices to 0.
    }

    //We have paths from each station and the source to all other stations and destination if feasible.
    //Now merge all to get the graph G’ with source, destination and stations as the vertices.

    //Initialise L values and apply Dijkstra’s now
    for(each i from 0 to |V|-1){
        y = ExtractMin(H’)            //vertex in heap with minimum L(v).
        move y from H’ to S’.
        for (each (y, v) in E’, with v in H’) {
            L(v) = min(L(v), L(y) + w(y, v));
            }
        }
    }
    if destination vertex d is not encountered above, i.e. L(d) = Infinity
        then return “NO FEASIBLE PATH”
    else return L(d)
}
```

PROOF OF CORRECTNESS:

Consider the **set of source, destination and fuel stations**. According to our algorithm, we will find the distance between source, destination and all pairs of fuel stations using the standard dijkstra algorithm. We will discard all edges between such pairs of vertices which have distance between them more than c. Then, in the graph that we get with source, destination and fuel stations, we find the shortest dijkstra distance. We have to show that this is the required distance.

Consider v_i and v_j from this set of vertices. Let d_{ij} denote the shortest path between them. Suppose, $d_{ij} > c$ for some pair i & j. Let the corresponding path be P_{ij}. Two cases arise:

- P_{ij} doesn't have any fuel stations. As the distance > c, it is not possible to reach station *j* from station *i* . Thus, this is an invalid case. There is no path from *i* to *j.*
- Only case that remains: P_{ij} must contain at least 1 fuel station, let's say k. Thus, by Principle of Optimality d_{ij} can be written as:

  $d_{ij} = d_{ik} + d_{kj}$

  This distance can be seen as sum of edge weights of pairs of stations *(i,j) & (j,k)*. Thus, we don't need an edge between *i & j* in the new graph, because that edge will never be chosen in dijkstra.

Thus, discarding stations' pairs in the given set with $d_{ij} > c$ is justified.

This gives us a graph G' in which all edges have weight <= c, thus all the paths in that graph are feasible. This graph has been obtained with the trees that were found by individual dijkstra on each fuel station. All the paths between two fuel stations are then visualized as edges in this graph with d_{ij} = Length of P_{ij} (path length obtained from the set of trees).

G' contains source s, destination d & the fuel stations. Here each edge has weight <= c. The point to be observed here is that the shortest path from s to d in G' is the same as the shortest path in the original graph G. The required path in the original graph G can be seen as including respective feasible sub-paths from source to fuel-stations to the destination. We can clearly see that all these paths should have shortest distances and should be feasible. This is exactly what is modelled by the graph G'. Since the edge weights in G' <= c, the corresponding paths in G have distance between any two fuel stations <= c, making the path feasible. While traversing between vertices in G, the required paths are those such that we get least distance between fuel stations which need to be feasible as well. This is the same as traversing a path in G' as we have already taken the shortest paths between fuel stations, and have created the graph in a way that all paths are feasible.

We can notice that if the required shortest feasible path from source to destination does not include any fuel station, this would mean that the length of this path is < c. Thus, this will be included in G' as well and because it is the shortest path, dijkstra on G' will output this path. Otherwise, if there is even a single fuel-station in the required output, this algorithm finds the correct output by taking paths from source to fuel-stations to destination. Moreover, if there exists no feasible solution then the graph G' won't have any edge and thus dijkstra algorithm on G' will give output "NO FEASIBLE PATH".

TIME COMPLEXITY:

Dijkstra's Algorithm takes O((m+n)logn) time and we repeat Dijkstra's Algorithm for each fuel stations we have and once again. So, at most (n+1) times. This means time complexity is O(n(m+n)logn). Since m can range from O(n) to O(n^2) therefore O(m+n) can be written as O(n), which means time complexity is O(mnlogn).