CS345 Assignment 5

Chinmay Goyal(180206) and Supreeth Baliga(180801)

1 Adventurous drive in Thar Desert:

Solution:

We approached this solution through various angles. So we considered a smaller instance of the problem saying suppose there was a fuel station at each junction, the problem would be reduced to general Dijkstra, just that we first remove all the edges with weights greater than c(where c is the distance the motorcycle can drive with full fuel tank). So based on this approach, we present our algorithm as follows:

Algorithm:
As the question has stated, we can model this problem as an undirected graph where the nodes are junctions and the edges are the roads. Edge weights denote the length of the road. Let us call this graph G(V,E). $\mid V \mid = n$, $\mid E \mid = m$

Firstly, the question does not mention if the graph is connected. So, we take in the following case as well. If s and d are not in the same connected component, then it is anyways not possible to reach d from s. So, our algorithm returns in this case as no required walk is possible. If s and d belong to the same connected component, we remove all the vertices and edges which do not belong to the same connected component as that of s and d. Note that to check if s and d belong to the same connected component, it is sufficient to see if d is reachable from s and to do this, a simple DFS traversal without considering the fuel capacity limit can be done. So, we now need to basically see the case where our graph G is connected with say n' vertices and m' edges.

Modified Dijkstra:
We will first define modified Dijkstra's Algorithm, here we keep a parent array which stores the parent for every vertex in the shortest path tree(the default value of the parent array is -1): Let p be the source vertex in the Dijkstra Algorithm, then while finding the shortest distance for any node, say the current node in consideration is X(that is the node that we are finding the shortest distance for in the current iteration), then if X is a fuel station other than the source vertex p then do not update the distance array for the neighbors of X else update the parent value of all the neighbors(whenever applicable that is when distance is updated) and the distance array for the neighbors of X just like in normal Dijkstra's Algorithm.

Let us maintain a matrix D which stores the distances (meaning length of the shortest path) between all pair of fuel station vertices. Now, we find the shortest distance between each pair of fuel station points by running Modified Dijkstra's algorithm at most n' times (n' is number of vertices, the worst case being when all the vertices are fuel station vertices), each time taking a different fuel station vertex as the source. Note that for each of the node, we also store the shortest path tree that we get when we apply Modified Dijkstra's algorithm

with that node as the source. We can easily store this by storing a parent's array for every Modified Dijkstra algorithm that we perform(parent of the source vertex in the Modified Dijkstra will be -1). The shortest distances are stored in the distances matrix D which we talked about above. Once, we are done with it, distances in the matrix with value greater than c have to be set to infinity. Here also note that in Dijkstra's Algorithm the default distance of all the nodes which do not have an edge from source vertex is infinity.

After this, we construct a graph G‘ with the nodes as all the junctions with fuel stations in G and *d*. (Note that G‘ also consists of *s* as it had a fuel station). Now, edges in this graph between nodes say (u,v) have weight which is basically the distances that we calculated previously i.e. D(u,v) in this case. So for each pair of vertices (u,v) where $u \neq v$, if $D(u, v) \neq \infty$, then we add an edge of length D(u,v) between u and v and if $D(u, v) == \infty$, then we do not add an edge between them.

Now, we simply run Dijkstra's algorithm on this new graph G‘ without any restrictions with s as the source. This will give us a shortest path in G‘ from s to d from which we can obtain the shortest route from s to d in G as follows:

Say the shortest path in G' from s to d was s, x_1, x_2, x_3, ..., x_k, d. So, we first find the shortest path from s to x_1 in G (say this is sp(s,x_1)) by using the shortest path tree with s as the source, then we find the shortest path from x_1 to x_2 in G (say this is sp(x_1,x_2)) by using the shortest path tree with x_1 as source, and so on. At the end, we concatenate these paths. So our required path is given by sp(s,x_1) + sp(x_1,x_2) + ... + sp(x_k,d) where '+' denotes concatenation of the paths.

Note that this route will be valid (meaning the person won't run out of fuel in between). We will prove that this algorithm works in proof of correctness. For now, let us look at the pseudo code.

Psuedo-Code:

```
FindShortestRoute(G) {
    if (s and d not in same connected component) {
        return "ROUTE NOT POSSIBLE";
    }
    else {
        Remove nodes and edges in G which are
        not in connected component of u and v;
    }
    // now G is a connected graph
    for(each u,v ∈ V) {
        if(u ≠ v) d(u,v) = ∞;
        else d(u,v) = 0;
    }
    for (each u ∈ V such that u is a fuel station node) {
        T(u) = ModifiedDijkstra(G,u);
        // T(u) stores the shortest path tree with u as source in G
    }
    for(each u,v ∈ V) {if(d(u,v) > c) d(u,v) = ∞;}
    Construct G‘ as explained in the above algorithm;
```

```
    T‘(s) = Dijkstra(G′,s); // T′(s) stores the shortest path
                           // tree with s as source in G′
    // from this we get path from s to d as
    // s, x_1, ... x_k, d. Say this path is P
    // Array of strings sp which stores the shortest paths
    // in G between adjacent vertices of P
    finalroute <- NULL;
    for(i from 0 to |P|-2) {
        sp[i] <- Path from P[i] to P[i+1] in G
        finalroute <- finalroute + sp[i]
        // '+' denotes concatenation
    }
    return finalroute;
}
```

Proof Of Correctness:

We will prove that a shortest route in G that we get in our algorithm will correspond to a shortest path that we obtain in G'. Consider any shortest route in G (Say R). We will first prove that this route corresponds to a valid path in G'.

So, now consider the fuel stations that you come across in R. Let they be s, x_1, x_2, x_3,..., x_k in this order. Now, firstly, all of these vertices are distinct. If not, there will be at least 2 appearances of at least 1 of these vertices (say x_i). We can collapse the route between the two appearances of x_i because it is just a cycle and it won't affect any fuel constraints. So, now we get a route shorter than R. This is a contradiction. Thus, each of s, x_1, ..., x_k are distinct. Now, note that take any pair of adjacent vertices here, say x_i and x_{i+1} (Note: can be s and x_1 too), note that the route from x_i to x_{i+1} will be a path (All the vertices will be distinct). Suppose it is not a path, we will simply be traversing extra non-fuel nodes with no benefits since we won't be coming across any fuel stations in between. So, the route will consist of k+1 paths which are s to x_1, x_1 to x_2,....., x_{k-1} to x_k and x_k to d. Now, each of these paths should be of length less than or equal to c, else it won't be possible because we will run out of fuel in the middle and there are no fuel stations between the two endpoints of any of the above paths. So, that means, there will be an edge from s to x_1, x_1 to x_2,....., x_k to d in G' since the distances between each of these pairs is less than or equal to c. Thus, s x_1 x_2 .... x_k d, will be a valid path in G'.

Now, we need to prove that the path that we got in G' i.e. s x_1 x_2 .... x_k d is a shortest path from s to d in G'. Now, suppose there is another shorter path from s to d in G' say s y_1 y_2 ... y_l d, then note that this means

$$d(s, x_1) + d(x_1, x_2) + ... + d(x_k, d) > d(s, y_1) + d(y_1, y_2) + ... + d(y_l, d)$$

This means we have a shorter route R' from s to d with fuel stations s, y_1, y_2, ..., y_l which has length shorter than that of R. This is a contradiction. Thus, s x_1 x_2 .... x_k d is a shortest path from s to d in G'.

We can easily see that for any path R' from s to d in G', we will get a corresponding route R from s to d in G. Now we claim that if R' is a shortest path from s to d in G' then its, corresponding route R in G will also be a shortest possible route in G. We prove this by contradiction. Suppose there is a shorter feasible route X in G, and its corresponding path

in G' is X'. Then we know that according to our assumption(here length of path or route is the sum of all the edges in that path/route):

$$\text{Length}(X') = \text{Length}(X) < \text{Length}(R) = \text{Length}(R')$$

But this is a contradiction because we know that R' is the shortest path from s to d in G'. Hence we prove that if R' is the shortest path from s to d in G' then its, corresponding route R in G will also be the shortest possible route in G.

Now, in our algorithm, once we get the shortest path in G', we are obtaining the shortest route in G. Now, as explained above each of s to x_1, x_1 to x_2, ... , x_k to d is a path in G. To get the shortest route from s to d in G', we need to find the shortest path from s to x_1, x_1 to x_2,..., x_k to d in G and then concatenate them as explained above by the use of the shortest path trees which we got while application of multiple instances of modified dijkstra's algorithm on G.

This proves that our proposed algorithm correctly finds a shortest route from s to d in G as required. This completes our proof.

Time Complexity:

First, we apply a DFS traversal to check if s and d belong to the same connected component. This takes time O(m+n). Now, after removal of the unrequired nodes and edges, say the remaining nodes are n' and edges are m' (Clearly $n' \leq n$ and $m' \leq m$). Since the graph is now connected, we have m' to be at least n'-1.

Now, each execution of modified dijkstra's algorithm will take time O(m' log(n')) and we do this n' times on G. So, total time for this = O($m'n'log(n')$). Now, for constructing G', we can find the relevant nodes in O(n') time and then I am considering an edge between each pair of the relevant nodes and this will take time O(n'^2) since the number of vertices in G' will be less than G. Now, applying dijkstra's algorithm on G' with s as source, the time taken will be O($n'^2 log(n')$) since number of edges in G' is O(n'^2). Now, for obtaining the shortest route in G from the shortest path in G', it will take time O(n'^2) since for each of the adjacent nodes in the obtained path in G', I will take time O(n') time to find a corresponding shortest route between those two nodes in G and there are O(n') such adjacent nodes in the obtained shortest path. And concatenating these obtained shortest routes will again take time of O(n'^2). So, the total time is:

$$Time(FindShortestRoute(G)) = O(m+n)+O(m'n'log(n'))+O(n')+3O(n'^2)+O(m'log(n'))$$

We have that $m' \geq n' - 1$, $n' \leq n$ and $m' \leq m$. So using these and simiplifying the above expression, we get

$$Time(FindShortestRoute(G)) = O(mnlog(n))$$