Homework 9

Siddharth Rajagopalan | sraja21@illinois.edu

Sai Aitha | sa60@illinois.edu

Rahul Kasibhatla | rahulk8@illinois.edu

Question 1

1 a.)

Let graph G' = (V', E'). We define V' as the product of the following.

```
V = Vertices of originalG
E = {Bought, NotBought} representing if we have bought an empanada
G = {Filled, NotFilled} representing if we have filled gas or not
```

Edge Construction:

For this component, we construct a helper function named Weight2 which specifies the weight of the edges in G'.

First we run dijkstras algorithm on s (the starting node in the original graph G). From here we can assume we have a dictionary data structure in which each key is a node and its value is its minimum distance from s. We will call this dictionary data structure minDfromS. We do this to see where {u, E, NotFilled} can point to. For each u -> v in G , if minDFromS[v] > D, then we do not add any edges from {u, e, NotFilled} where e can be Bought or Not Bought. If minDFromS[u] <= D, G' would have the edge

```
{u, e, NotFilled} -> {v, e, NotFilled} where e is in {NotBought, Bought}
and weight2({{u,E,NotFilled},{u,E, NotFilled}}) = w({u,v}) and {u,v} is an
edge in G
```

However if G = Filled, then we keep all of the edges the same as we assume we have an infinite amount of milleage in our tank. Therefore the edges coming from {u, E, Filled} would look like the following

```
    {u, e, Filled} -> {v, e, Filled} where e is in {NotBought, Bought} and
weight2({{u,E,NotFilled},{u,E, NotFilled}}) = w({u,v}) and {u,v} is an
edge in G
```

Now we have to connect the different components of our layers. We have to consider the cases a u is a gas station or u is an empananada store in our graph G. The edges in G' would look like the following

if u is an empanada store in G, G' has the following edges,

```
{u, NotBought, g} -> {u, Bought, g} given that g is a member of G (defined
above). Weight2({{u, NotBought, g},{u, Bought, g}}) = 0
```

Additionally, if u is a Gas Station, G' has the following edges,

```
{u, E, NotFilled} -> {u, E, Filled} given that e is a member of E (defined
above). Weight2({{u, E, NotFilled},{u, E, Filled}}) = 0.
```

From here, we run Dijkstra's algorithm on G' starting from {s, NotBought, NotFilled} to {t, Bought, g} where g is a member of G and return the shortest path. If {t, Bought, g} cannot be reached we simply return the value infinity.

Runtime Analysis:

```
We have 2 * 2 * n nodes where n = |V|. Our edges in G' remain the same.
Therefore our runtime is O(m + nlog(n)). We get this runtime from running
Dijkstra's algorithm from {s,NotBought, NotFilled}
```

1 b.)

Let G' = (V', E'). We define V' as the product of the following:

```
E = {Bought, NotBought} // referencing that we have bought an empanada

Therefore, V' = V x E
```

Edge Construction:

for each u -> v in G, G' have the edges

```
{u, e} -> {v,e} such that e is a member of E where Weight{{u, e} -> {v,e}
} = Weight({u, v})
```

Additionally if u is an empanada store then G' have the edges

```
{u, NotBought} -> {u, Bought} such that Weight({u, NotBought} -> {u,
Bought}) = 0
```

Now our algorithm can be modeled utilizing the following pseudocode:

```
FindMinDistance(G, s):
    G' = initalizing G' by the rules above
    //assuming dijkstra's algorithm returns a list in which each node
represents the min distance from {s, Not Bought}
    if (djistrasFromS'[{t, Bought}] <= R)
        return dijstrasFromS'[{t, Bought}]
    else:
        for every {u, NotBought} such that u is a gas station
        and dijkstrasFromS'[{u, NotBought}] <= R:
            run dikstra's on each u as a start node and see if any one of
them reach {t,Bought} and store the distances in an array + distance(s,
u). If it cannot reach, then run dijkstra's on each of u's reachable gas
stations such that the distance does not excede R and keep continuing the
process. We return the min distance to {t, bought} + the min of all the
incoming distances. We keep going until every gas station was processed.

    if no value was returned
        return infinity
```

For this algorithm we run Dijkstra's on {s, NotBought} can reach {t, Bought} and return its min distance. If this is not possible, then we run djikstras on each of the gas stations reachable from {s, NotBought}. Let u be reachable gas stations from s. We store the distance from every {u, NotBought} to {t, Bought} + Distance({s, NotBought},{u, Bought}). If this cannot reach, we go to every unvisitied gas station that every u can reach that are unvisited and repeat the process of finding its min distance(Djikstra's) to {t,Bought} + distance from {u, NotBought} to {u's gastation, NotBought}.

Run time Analysis:

```
Since we run Dijkstras on {S, NotBought} and check if it can reach {t,E}
and at worst case repeat for every single gas station, we know that the
runtime must be (n(m+nlogn)) where n is the number of nodes and m is the
number of edges.
```