Assignment 3

Görkem Güzeler

Spring 2021

1 PART A:

Let me denote the shortest path from vertex a to b as S(a,b). Let me denote the path weight from vertex a to b as W(a,b)

The purpose is to find out quickest itinerary from Istanbul-Harem to every other city in Turkey. Therefore I will apply modified Dijkstra algorithm. I assume there is no negative edge and it is a directed graph.

Subproblems: Let shortest path from Istanbul to Adana S(Istanbul, Adana). First, we check the neighbors with the lowest route cost, let it be Bursa. By the time we arrive to Bursa, our problem becomes finding the shortest path form Bursa to Adana which is one of the subproblems. Each time we change our place, our subproblem changes as well.

Optimal substructure property: It is satisfied. Let shortest path from Istanbul to Adana S(Istanbul, Adana). Assume the first city in Shortest Path (after Istanbul) is the Ankara. Then, definitely optimal shortest path between Ankara and Adana will be satisfied by only removing Istanbul from the route. Which shows that subproblems of the problem has optimal solutions. Cut and paste is used to prove by contradiction. Because, if the route between Ankara and Adana is not the shortest path, then it implies that the path between İstanbul - Adana is not the shortest as well which contradicts with the our assumption.

Overlapping computations: There are overlapping subproblems. Let shortest path from Istanbul to Adana S(Istanbul, Adana). Assume the routes are given as: İstanbul-Bursa , İstanbul-Tekirdag, Bursa-Adana, Tekirdag-Bursa. Algorithm will first pick the shortest path to the next route (let it be Tekirdag) => S(Tekirdag to Adana) + W(Istanbul-Tekirdag). Then, assume that next shortest route is Tekirdag-Bursa. Total path becomes = S(Bursa to Adana) + W(Istanbul-Tekirdag) + W(Tekirdag-Bursa). After that, we continue with the shortest route which will be İstanbul-Bursa, therefore total path becomes S(Bursa to Adana) + W(Istanbul-Bursa). Please notice that, we have completed S(Bursa to Adana) for the second time. We will have lots of overlapping subproblems when the number of edges and vertices increases.

Recursive formulation respecting a topological ordering:

```
S(a,a) = 0
S(a,c) = min{S(a,b) + W(b+c)} where a is not equal to c.
```

I assume the graph is DAG. So, we will have a topological ordering of the cities. Assume our path is from Istanbul to Adana. There will be multiple routes such as İstanbul - Bursa - Adana, İstanbul-Ankara- Adana, İstanbul- Ankara - Antalya - Adana etc.

2 PART B:

Assume we have bus routes and train routes given as matrix in the format [[starting point,final point,cost],[starting point,final point,cost],...]

```
    for each route in bus routes:
        add key to our route dictionary with the cost value

    for each route in train routes:
        if both starting point and destination of bus route matches with train route
            take the minimum of them

    then convert dictionary to matrix routes

function naive(starting_node,goal_node, matrix routes)

    find all of the locations from matrix and store them in dictionary of vertices as key na

    d[starting_node] ← 0
    for each v  V { {starting_node}
        do d[v] ←
    visited←
    Q ← V // Q is a dictionary
    while Q
        do u ← find-min Q
        pop min element of Q
        visited ← visited  {u}
        for each v  Adj[u]
            do if d[v] > d[u] + w(u, v)
                then d[v] ← d[u] + w(u, v)

    return d[starting_node, goal_node]
```

Complexity analysis: Unfortunately, we are solving each subproblem multiple times. Assume that there are at most b routes from a city to another. Let the number of vertices be V and number of edges E. Thus; branching factor = b, vertices = V, edges = E. Time complexity will be $0(b^{|v|-1})$. Space complexity will be O(E) because of the number of edges will be stored in the matrix at most.

3 PART C:

Assume we have bus routes and train routes given as matrix in the format [[starting point,final point,cost],[starting point,final point,cost]...]

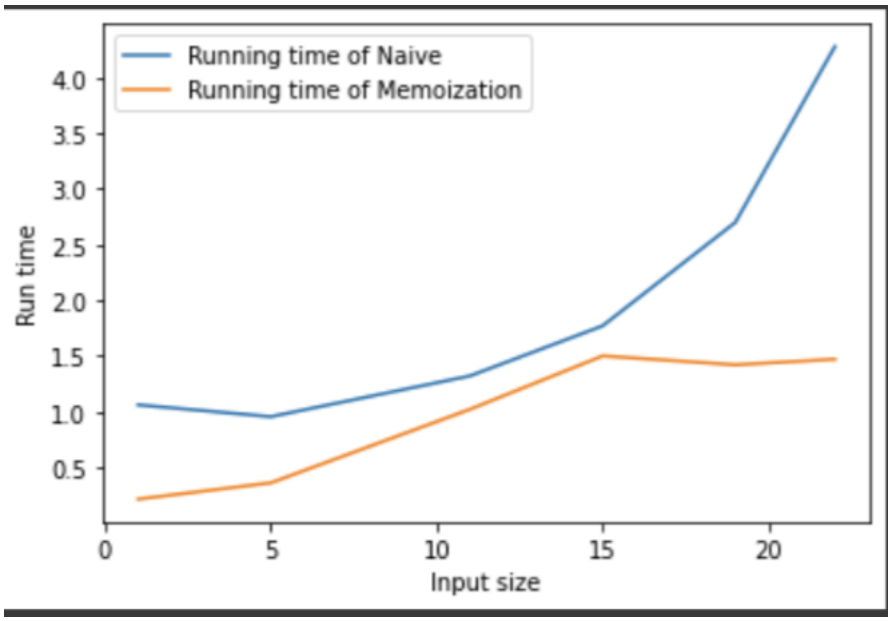
```
    for each route in bus routes:
        add key to our route dictionary with the cost value

    for each route in train routes:
        if both starting point and destination of bus route matches with train route
            take the minimum of them

    then convert dictionary to matrix routes

function memoization(starting_node,goal_node, matrix routes)

    find all of the locations from matrix and store them in dictionary of vertices as key na

    d[starting_node] ← 0
    for each v  V { {starting_node}
        do d[v] ←
    visited←
    Q ← V // Q is a priority queue maintaining V { visited
    while Q
        do u ← EXTRACT-MIN(Q)
            visited ← visited  {u}
            for each v  Adj[u]
                do if d[v] > d[u] + w(u, v)
                    then d[v] ← d[u] + w(u, v)

    return d[goal node]
```

Complexity analysis: Since we are solving each subproblem only once, it is a good algorithm. Assume that there are at most b routes from a city to another. Let the number of vertices be V and number of edges E. Thus; branching factor = b, vertices = V, edges = E. Using dictionary costs O(E) due to the number of each route. Since we have number of V vertices in our priority queue O(V).

Thus, in total time complexity becomes = 0(E+V). Space complexity will be O(E)

4 PART D:

4.1 Plot:



5 Notes:

Google Colab ipynb and py files are zipped and attached. Furthermore, you can access via link below:

https://colab.research.google.com/drive/1XGyAnFY4fm96GfZZkQJJzwDO3cwC5TVY?usp=sharing