**Final Project**

Tesla makes electric cars. But they have a problem in that the car can go only m miles on a charge and specialized charging stations are needed to quickly recharge the battery back to full capacity. To make the cars more appealing to long distance drivers, Tesla is planning to place charging stations along one major highway connecting Los Angeles and San Francisco. They still have the problem of drivers worrying what will happen if their cars run out of charge on the highway in-between two charging stations. Drivers don't want to stop at every charging station either. So Tesla wants you to come with an algorithm that, given the distance from a starting location on the highway to the first charging station, the distance between all intermediate charging stations, and the distance between the last charging station before destination and the destination, will tell the driver which stations to stop at so that he or she makes the minimum number of stops from among the n charging stations between the driver's starting location and destination.

1. Come up with a recursive characterization of the solution. Then explain why it has optimal substructure. Both these should be included in your submission

The Problem can be recursively characterized as follows

Let stops[i , j] be the minimum number of stops needed to stop from station i to destination point j

We can define stops [i , j] as follows, when i=j then the problem is trivial and minimum number of stops required between the same stations is zero. If i<j then stops required between i and j can be characterized by splitting it to sub problems. Find the minimum number of stops required if stopped at first station, if stopped at second station and so on up to last station and find the minimum of those options . minimum number of stops required to reach destination if stopped at first station is equal to 1 plus minimum number of stations it needed to stop before the first station plus minimum number it is needed to stop after the first station, applying the same characterization to the whole problem we get

stops[i ,j] = stops[i, k] +stops[k+1,j]+1 where i<= k < j

//since j is the destination point we have no chance to stop at j

Thus..

stops[i ,j] = 0 if i=j

stops[i ,j] = min i<=k<j  {stops[ i , k] +stops[k+1, j] +1} if i< j

we use s[i,j] to store the station numbers at which we should stop so that total number of stops between i and j is minimum

**Optimal Substructure:**

This problem has an optimal substructure because to find the minimum number of stops between stations I and j we divide it in to sub problems and then find the minimum number of stops for each of its sub problems. Thus for the solution to be minimum each of its sub problems should be solved minimally which says this problem has an optimal sub structure.

2. Design a corresponding non-recursive bottom-up table lookup algorithm to solve the problem. This should be included in your submission

MINUMUM-STOPS ( m,n,d[1…n+1)

1 let m,n be the mileage of the car and toal number number of stations

2 d [1…n+1] // an array containing distance from start to first station ,first station to second station …nth station to destination

3 let stops [1..n +1, 1..n+1] and s[1…n +1,1…n+1] be two new tables

4 **for** i =1 to d.length

5 if d[i] > m then

6 exit and print ”No valid solution for the given distances”

7 **for** i 1 **to** n+1

8 stops [i,i]= 0 // minimum number of stops between the same stations is 0

9 **for** l = 2 **to** d.length **//** l is number of number of possible stations

10 **for** i = l **to** n – l +1

11 j = i+l – 1

12 stops[ i ,j ] = ∞

13 **for** k = i **to** j - 1

14 q = stops[ i ,k ] + stops [k+1 ,j] +1

15 **if** q <= stops[ i , j] then

16 if j - i = 1 and d[i] + d[j] < m

17                                                                                           stops[i, j] = 0

18                                                                           else if j - i = 1 and d[i] + d[j] = m

19                                                                                           stops[i, j] = 0

20                                                                                          s[i,j] = j

21                                                                           else if j - i = 1 and d[i] + d[j] > m

22                                                                                           stops[ i,j] = 1

23                                                                                           s[ i,j] = i

24                                                           else

25                                                                                           stops[i,j] = q

26                                                                                           s[i, j] = k

27 return stops[1,n], s

The above algorithm returns the minimum number of stops between station 1 and station n infact it finds the minimum number of stops between any two stations and stores them in a table ,It also returns a table containing the stop numbers that the car should stops between the respective stations to get the minimum possible stops .The algorithm below prints the those stops in a recursive way

PRINT-Stops(s, d ,n)

// let n be the minimum number of stops returned by the above MINUMUM-STOPS algorithm and and s be the table containing minimum stops between respective stations, initially d is the total distances available

1 **if** n > 0

2 n= n-1

3 PRINT-Stops(s,s[0][d],n)

4 print(s[0][d])

5 **else** print “destination reached”

**Complexity Calculation:**

In the MINUMUM-STOPS Algorithm

1. steps 1,2 and 3 take constant time hence complexity is O(1)
2. step 4 executes O(n+2) times since it is for loop running from 1 to n+1 ,i.e size of d
3. steps 5 and 6 each executes O(n+1) times
4. steps 7 and 8 execute O(n+2) and O(n+1) times
5. step 9 executes O(n+1) times
6. step 10 executes O(n2 ) times // since nested for loop
7. steps 11,12,each execute O(n2 ) times
8. step 13 executes O(n3 ) times //since nested inside two for loops each of complexity O(n)
9. similarly steps 14 to 26 have complexity O(n3 )
10. step 27 has constant complexity O(1)

From the above steps we can say that approximate complexity of algorithm is O(n3 )

References:

1. Introduction to algorithms 3rd Edition
2. Used the Algorithm of MATRIX-CHAIN-ORDER() to build my algorithm, it mostly look the same