Yunus Emre TAT 25187 Assignment 4

Problem 1

a.1)

The Longest Common Subsequence algorithm which we learnt in lecture was devised for the 2 sequences. Sq_1, Sq_2 are the 2 sequences.LCS(Sq_1, Sq_2) was defined as find the longest subsequences present in both of them.Lenght of the strings are defined as "*l*" and last of the their letter is defined *"lt.LCS algoirthm basically is checking the last index of sequences and if they are same return LCS +1 and all of the length decreased by 1 ,if they are not same,I will call the k times LCS and in every call,their length will be decremented by 1 orderly.So that our tree nodes have k leaf in every iteration when last items are not same.*

Their formula for 2 sequences is,

if $lt_1 = lt_2$ *then 1+ LCS (*l_1 -1, l_2 -1)

else max{LCS(l_1 -1, l_2) , *LCS(* l_1 , l_2 -1) }

But for now,we have $Sq_1, Sq_2, Sq_3, Sq_4, Sq_5$ …… Sq_k *k sequences.We can modify the LCS algorithm and modified one will be called LCSM.LCSM will take k sequences which called Sq instead taking 2 in LCS. LCSM(s* $Sq_1, Sq_2, Sq_3, Sq_4, Sq_5$ …… Sq_k) was defined as find the longest subsequences present in all of them. Lenght of the strings are defined as "*l*" and last of the their letter is defined *"lt".It's recursive formula is,*

if $lt_1 = lt_2 = lt_3 = lt_4 = lt_5 = \ldots.= lt_k$ *then 1+ LCSM (*l_1 -1, l_2 -1, l_3-1, l_4 -1 l_5 -1,……., l_k -1)

else max{LCSM(l_1 -1, l_2 , l_3, l_4, l_5, l_6, l_7 ,……, l_k) , *LCSM(* l_1 , l_2-1 , l_3, l_4, l_5, l_6, l_7 ,……, l_k) , *LCSM(* l_1 , l_2 , l_3-1, l_4, l_5, l_6, l_7 ,……, l_k)……., *LCSM(* l_1 , l_2 , l_3, l_4, l_5, l_6, l_7 ,……, l_k -1)}

---

a.2)

In the recursive algorithm,there are many subproblems which are solved again and again.So LCS and LCSM has overlapping substructure property.We will use top down algorithm and create matrix table for memorization.In normal LCS we can set the algorithm,

LCS(Sq_1, Sq_2)

table = it is a memorization table in 2d dimension ($l_1 * l_2$)

if $lt_1 = lt_2$ *then table*[l_1 , l_2] = 1+ *LCS* (l_1 -1, l_2 -1) return *table*[l_1 , l_2]

else table[l_1 , l_2] = *max*{*LCS*(l_1 -1, l_2) , *LCS*(l_1 , l_2 -*1*) } return *table*[l_1 , l_2]

now I will modify for the k sequences,

In LCSM,unlike using 2 dimensional,we used k dimension table and its size is multiplication of all of the length of the sequences.So that using memorization table prevent the overlapping problems.

LCSM(*Sq*$_1$, *Sq*$_2$, *Sq*$_3$, *Sq*$_4$, *Sq*$_5$, *Sq*$_6$, *Sq*$_7$, *Sq*$_8$, ……. , *Sq*$_{k-1}$, *Sq*$_k$)

table = it is a memorization table in kd dimension ($l_1 * l_2 * l_3 * l_4 * l_5 * l_6 * l_7 \ldots * l_{k-1} * l_k *$)

if $lt_1 = lt_2 = lt_3 = lt_4 = lt_5 = lt_6 = lt_7 \ldots\ldots = lt_k$ *then table*[l_1 , l_2 , l_3 , l_4 , l_5 , l_6 , l_7 …….. l_k ,] = 1+ *LCSM* (l_1 -1, l_2 -1, l_3 -1, l_4 - 1 , l_5 -1, l_6 -1, l_7 -1,….., l_K -1) return *table*[l_1 , l_2 , l_3 , l_4 , l_5 , l_6 , l_7 , l_8 , ……. l_K]

else table[l_1 , l_2 , l_3 , l_4 , l_5 , l_6 , l_7 …….. l_k ,] = *max*{*LCSM*(l_1 -1, l_2 , l_3, l_4 , l_5, l_6 , l_7 ,……, l_k) , *LCSM*(l_1 , l_2 -1 , l_3, l_4 , l_5, l_6 , l_7 ,……, l_k) , *LCSM*(l_1 , l_2 , l_3 -1, l_4 , l_5, l_6 , l_7 ,……, l_k)……., *LCSM*(l_1 , l_2 , l_3, l_4 , l_5, l_6 , l_7 ,……, l_k -1)} return *table*[l_1 , l_2 , l_3 , l_4 , l_5 , l_6 , l_7 , l_8 , ……. l_K]

---

b)

Recursive algorithm in the LCS is O(2^n*) because in algorithm it returns 2 LCS if they are not equal so it will grow 2 exponentially.But in LCSM we have k call if they all not equal so we have k branches it equals O(*k^n*) n value is the depth of the tree.In top-down algorithm we use the table and it will lead the decreasing of the complexity.In LCS with memorization complexity will be O(M.N) M is first sequence length N is the second sequence length in LCSM with memorization,time complexity will be multiplication of lengths O($l_1 * l_2 * l_3 * l_4 * l_5 * l_6 * l_7 \ldots\ldots l_k$)*

Problem 2

a)

One of the input is set of cities/towns in Turkey with a map,Second of the inputs is a set Sc of cities/town that the tourists are usually interested in visiting on their way for visiting each c in C.Shortest itinerary from Istanbul to c in C which requires visiting Sc subset members is our output.

b)

I will call the my problem as Traveler Problem.

Traveler problem is NP-Hard

Proof:

To prove that Traveler Problem is NP-hard in polynomial, I have to show that every problem in NP reduces to Traveler problem in polynomial time.There is an alternate way to prove it.The idea is to take a known NP-Hard problem and reduce it to Traveler Problem.I will select the Hamiltonian-Cycle because it is pretty similiar to the Traveler Problem.I will call Hamilton Cycle as HC. Given a graph G=(V,E) there exist a simply cycle in G that traverses every vertex exactly once and reeturns back in HC. Now to reduce HC to Traveler.Each vertex s in G can be Istanbul it does not matter because each vertices will return itself with traverses every vertex one,.Any verteices different from source vertex s can be called u and set to C.Remaining from source vertex s and u verteices which are in C,they can be in Sc.So that,from s vertex which is Istanbul need visit all vertices in Sc when it goes to C vertices and return s.Take G=(V,E), set all edge weights equal to 1.Then look if there exist a itinerary from Istanbul to C with visiting Sc with cost equals at most number of vertices.If the answer is Yes to Traveler Problem,Then Hamilton Cycle Yes.If the answer is No to Traveler Problem,Then Hamilton Cycle No.

Proof: We must show that the reduction takes polynomial time and that solutions for HC are in 1-1 correspondence with solutions to Traveler Problem using the reduction. Clearly, the reduction takes polynomial time.

For first part proof, If the answer is Yes to Traveler Problem,Then Hamilton Cycle Yes. Then there exists a simple cycle C that visits every node exactly once and return back to source vertex.It can be used in Traveler problem it starts from Istanbul to a city in C with visiting all the cities except Istanbul and city in C and return back to Istanbul.

For second part proof, If the answer is No to Traveler Problem,Then Hamilton Cycle No,if there doesnt exist a Hamiltonian cycle in Graph,Travelling all the cities exactly one from Istanbul and return the Istanbul is not possible.

We have proved that Traveler Problem is NP-Hard because HC is Np-hard .