CAVIT CAKIR 23657

1)a) Recursive Algorithm;

The algorithm that we discussed in class was for 2 strings. To find an LCS for k sequences we should modify recursive formula as follows;

$$LCS(X_i, Y_j, ..., K_k) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \ldots \text{or } k = 0 \\ LCS(X_{i-1}, Y_{j-1}, ..., K_{k-1}) + 1 & \text{if } i, j, ..., k > 0 \text{ and } x_i = y_j = ... = k_k \\ \text{MAX}\{LCS(X_{i-1}, Y_j, ..., K_k), LCS(X_i, Y_{j-1}, ..., K_k) ... LCS(X_i, Y_j, ..., K_{k-1})\} & \text{otherwise.} \end{cases}$$

We make recursive call with k sequence and if their last characters are all same we add 1 and make recursive call. If one of last character is not same as others then we call k recursive calls and maximize them.

Top-Down Algorithm;

We see in class that to solve 2 string LCS problem with a matrix. We filled it according to algorithm in slides. Now we need to fill an k-dimensional matrix. Code follows:

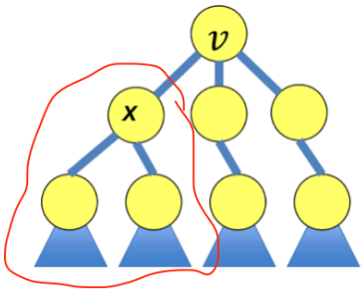
```
LCS(x, y, … , k, i, j, … , z)
    IF c[i, j, … , k] is NIL then
        IF x[i] = y[j] = … = k[z] then
        c[i, j, … , k] ← LCS(x, y, … , k, i–1, j–1 , … , z-1) + 1
        ENDIF
        ELSE c[i, j, … , k] ← MAX{ LCS(x, y, … , k, i–1, j, … , z ),
                                   LCS(x, y, … , k, i, j, … , z-1) }
        ENDELSE

    ENDIF
```

b) Complexity analysis:

N is maximum string length and **k** is # of strings.
To analysis **recursive algorithm**; We should consider the recursive tree, there will be k child and height of tree will be nk. So the complexity will be **$O(K^{(nk)})$**.
To analysis **Top-Down algorithm**; It was **$O(N^2)$** for 2 dimensional matrix and now we should fill k-dimensional matrix. So our complexity will be **$O(N^{(k)})$**.

2)



a)Design of Algorithm:

Subproblems: If we consider the three above. Our subproblems will be the smallest vertex cover in subtree rooted at V.

Optimal Structure: If we always choose smallest vertex cover when we are dealing with subproblems, overall will be optimal.

Recursive Solution:

Case 1 : Calculate size of vertex cover when root **is** part of it. This means that all children are covered. Then we will consider childrens minimum vertex cover.

Case 2 : Calculate size of vertex cover when root **is not** part of it. We have to pick all the children to be connected vertex cover. Then we will consider childrens minimum vertex cover.

Psuedo :

```
vCover(root)
{
   if tree is empty
      return 0
   if there is only one node
      return 0

   if vertex cover for this node is already evaluated
      return calculated

   // Calculate size of vertex cover when root is part of it
   including = 1 + vCover(root->left) + vCover(root->right)

   // Calculate size of vertex cover when root is not part of it
   excluding = 0
   if (root->left)
     excluding += 1 + vCover(root->left->left) + vCover(root->left->right)
   if (root->right)
     excluding += 1 + vCover(root->right->left) + vCover(root->right->right)

   vc = min(including,excluding)
   store vc

   return vc
}
```

b) Time complexity:

Number of subproblems = V —> memoization
Time of each subproblem = O(V)
Overall = V * O(V) = $O(V^2)$

c) If a graph is given:

Analysis: Vertex Cover Problem is a known NP-Complete problem. There is no polynomial time solution.

Algortithm:

```
C = ∅
while there is uncovered edge (u,v)
  add vertex u into C
  add vertex v into C
return C
```

3) a) Minimum Leaf Spanning Tree:

Input : Graph G
Output : A spanning tree in G, whose number of leaves is minimum.

b) Real-world application : Assume that you have a network provider company; you want to spread fiber internet cables to provide network to your customers; and the fiber cable company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your customers with a minimum total cost. It should be a minimum leaf spanning tree to maintain minimum cost.

c) NP-complete : To prove MLST is NP-Complete, first we should prove that MLST is NP problem, afterwards we can reduce it to some other NP-Complete problems. Our aim is reduce MLST to Hamilton Path Problem. We can see clearly that MLST is NP problem and Hamilton Path Problem is special case of MLST which number of leaves is 2. Our problem includes when number of leaves is 0,1,2,3… so our problem is at hard at least as Hamilton Path Problem. So we can say that MLST is NP-Complete.