CS 512: Design and Analysis of Algorithms

Assignment #3

Name: Kunal Wanikar

Roll No.: 204101070

Question 1)

Given: Two sequences X = $x_1, x_2, x_3 \ldots x_m$ and Y = $y_1, y_2, y_3 \ldots y_n$ of lengths m and n respectively.

To find: The value which OPT(m, n) computes at the end of the algorithm.

Solution:

1) Analysis: We observe that the Bellman equation is the solution for an optimization problem which tries to maximize the result. Initially we consider the maximum length matched by the two sequences is zero. This equation can be analysed by considering following cases.

1. First, we will consider the base case i.e. when the length of either of the two sequences become zero then there won't be any common part between X and Y. Hence, we can add 0 to the above recursive step as an anchor condition.
2. Now, we start by considering from the last character of both the sequences,
    a. If they match, then we increment the result by one and remove the last character and recursively check for the previous characters of both the sequences.
    b. Else we take the maximum length among the sequences which are matched earlier and store it in the intermediate result. If the maximum length matched is from the X string, then we delete the last character of X and recursively check for the new X and old Y. Similarly, if the maximum length is from Y then remove the last character of Y string and recursively check for old X and new Y.

Hence we can observe that, either adding 1 each time when we find a match in the two sequences X and Y or by taking the maximum length matched among the sequences or, we get the length of the **longest common subsequence** possible by these two strings. The maximum length of subsequence will be bounded by *min(m, n).*

2) Justification:

Base case: When either of the length of subsequence is zero, i.e. either *length(X) = 0* or *length(Y) = 0* or both are zero, then the maximum length of common part to both sequence will be null. Hence maximum length will be equal to zero. Here we get the base case as true.

Induction Hypothesis: Let us assume that *OPT(m, n)* holds true till the length of sub-sequences are k_1 and k_2 for X and Y respectively where $0 \leq k_1, k_2 \leq min(m, n)$. This means the following holds true.

$$OPT(k_1, k_2) = \begin{cases} 0 & k_1 = 0 \text{ or } k_2 = 0 \\ 1 + OPT(k_1-1, k_2-1) & X_{k1} = Y_{k2} \\ Max\{OPT(k_1, k_2-1), OPT(k_1-1, k_2)\} & Otherwise \end{cases}$$

Inductive Step: In the induction hypothesis as we have assumed that the equation holds true for k_1, k_2, hence now we need to prove for k_1+1, k_2+1. Here we look at $OPT(k_1+1, k_2+1)$. We now have two cases.

1. If the $(k_1+1)^{th}$ character matches with the $(k_2+1)^{th}$ character: Now we know that if the $(k_1+1)^{th}$ character of X matches with the $(k_2+1)^{th}$ character of Y then the length of the maximum sub-sequence which is possible by X and Y will increase by one and we will recursively check for k_1^{th} and k_2^{th} character. Hence if the last characters of X and Y matches then we get the equation as :
$$\boldsymbol{OPT(k_1+1, k_2+1) = 1 + OPT(k_1, k_2)}$$
Now, further expanding the equation, we know from the induction hypothesis that $OPT(k_1, k_2)$ computes the correct length from of the maximum subsequence possible. Thus, if we recursively get the matching again and again, at some point the equation will become either $k_1+OPT(0, y)$ or $k_2+OPT(x,0)$. In these cases, we take the anchor condition into consideration and the recursive step halts.

2. If the $(k_1+1)^{th}$ character does not match with the $(k_2+1)^{th}$ character: Here we again have two cases, of which we must consider the maximum one.

    a. We now get the equation $\boldsymbol{OPT(k_1+1, k_2+1) = OPT(k_1+1, k_2)}$ and check recursively for the $(k_1+1)^{th}$ character of X matches with k_2^{th} character of Y. If yes, then the equation reduces to $OPT(k_1, k_2-1)$, according to case 1, which by induction hypothesis calculates the correct maximum length of longest common subsequence. If no, we again come to case 2 and again recursively check for $OPT(k_1+1, k_2)$. By doing such operation recursively, we will achieve a step in which the equation will be either *OPT(0,y)* or *OPT(x,0).* This will be the base case and we will stop at that point.
    b. We now get the equation $\boldsymbol{OPT(k_1+1, k_2+1) = OPT(k_1, k_2+1)}$ and check recursively for the k_1^{th} character of X matches with $(k_2+1)^{th}$ character of Y. If yes, then the equation reduces to $OPT(k_1-1, k_2)$, according to case 1, which by induction hypothesis calculates the correct maximum length of longest

common subsequence. If no, we again come to case 2 and again recursively check for *OPT*(k_1, k_2+1). By doing such operation recursively, we will achieve a step in which the equation will be either *OPT(0, y)* or *OPT(x,0).* This is the base case, and we will stop at that point.

We have proved the Bellman equation by induction that it follows the optimal substructure and overlapping subproblems property.

Conclusion: Hence by considering cases in the above problem and providing justification of the Bellman equation *OPT(m,n)* we observe that the result gives us the length of **Longest Common Subsequence** possible between the two sequences X and Y and also the subsequence which is possible of that length can be computed by representing the above procedure in matrix form.

Question 2)

Given: Directed graph *G= (V, E)* with edge weights l_{uv} for each edge *(u, v) € G,* and an integer *k.*

To find: The value which *OPT(s, t, k)* computes at the end of the algorithm.

Solution:

1) Analysis: We observe that the Bellman equation is the solution for an optimization problem which tries to maximize the result. Initially we consider *j=0* which means we can take at most 0 edges. Hence the maximum weight will be zero. This equation can be analysed by considering following cases.

1. First, we will consider the base case i.e. when number of edges equal to zero, then the result will be zero according to the Bellman equation. Hence, we can add 0 to the above recursive step as an anchor condition.
2. Now we take the maximum of either of the two cases
    a. *OPT(u, v, j-1)* calculates the maximum weight using *(j-1)* edges recursively by again and again until the third parameter (i.e. *(j-1)*) reaches zero and, hence halting the recursion.
    b. Now we take all the outgoing edges from *'u'*. Let each edge outgoing from '*u'* reach a vertex denoted by '*w'*. We calculate the maximum weight among all the outgoing edges by adding the edge weight to the reduced recursive equation *OPT(w, v, j-1)* where '*v'* is the destination vertex. This reduced recursive equation is further reduced until the third parameter (i.e. *j*) becomes zero and we reach the anchor condition.

Hence, we can observe that the algorithm is recursively calculating the **maximum non-negative weight** for a walk in a directed graph G starting from vertex '*s'* to vertex '*t'* **using at most k edges**.

2) Justification:

Base Case: When $j=0$ i.e. $OPT(u, v, 0) = 0$. For at most 0 edges for reaching from *'u'* to *'v'*, we get the maximum weight as 0. Hence the base case is true.

Induction Hypothesis: Let us assume that $OPT(s, t, x)$ holds true for every $x \leq k$ till we walk through the directed graph having x edges. This means the following holds true.

$$OPT(u, v, x) = \begin{cases} 0 & x = 0 \\ Max\,\{OPT(u, v, x-1), max_{(u, w)\,\epsilon E}\{l_{uw} + OPT(w, v, x-1)\}\} & Otherwise \end{cases}$$

Inductive Step: In the induction hypothesis as we have assumed that the equation holds true for at most x edges, hence now if prove that it holds true for $(x+1)$ edges as well, then it will be true for at most k edges as well. We now have two cases.

1. Base case: OPT(u,v,1) = $Max\,\{OPT(u, v, 0), max_{(u,w)\epsilon E}\{l_{uw} + OPT(w, v, 0)\}\}$ .As we know that $OPT(u, v, 0) = 0$ from the basis of the induction we get $max_{(u,w)\epsilon E}\{l_{uw} + OPT(w, v, 0)\}$. Now here also we know that $OPT(w, v, 0) = 0$ from the basis. Hence, we get $OPT(u, v, 1) = l_{uv}$.
2. We must prove

   $$OPT(u, v, x+1) = Max\,\{OPT(u, v, x), max_{(u, w)\,\epsilon E}\{l_{uw} + OPT(w, v, x)\}\}$$

   From the induction hypothesis we know that $OPT(u, v, x)$ will calculate the maximum non-negative weight using at most x edges. Now considering the edges outgoing from *'u'*, we check recursively for each edge whether it forms the maximum non-negative weight pair with *'u'*. If yes, then we add the weight of the edge(l_{uw}) to the final result, else we check for the next edge recursively by again considering all the above cases possible. After all the outgoing edges are traversed, we add the edge cost of highest weight. Here the overlapping subproblem property can be observed when we store the optimal answer of computation in a matrix.

   For every reducing recursive equation we will eventually reach the base case i.e. when third parameter will be equal to zero. This guarantees that the algorithm will stop at some point.

Hence, we have proved the Bellman equation by induction that for every $x \leq k$, the equation will hold true.

Conclusion: Hence by considering cases in the above problem and providing justification of the Bellman equation $OPT(s, t, k)$ we observe that the result gives us the **maximum non-negative weight for a walk in a directed graph starting from vertex s to vertex t using at most k edges.**