CS141, Programming IV Report **Due: 23:59, Wed 11/09, 2022**

Name: Majd Kawak / NetID: mkawa025 / CodeForces ID: mkawa025 / Student ID: 862273310

A. Longest decreasing subsequence

Submission#: 180250359

In this program, firstly I declared an *int* that would hold array size, then I declare an *array* with *int size*. I move on to input all values and store them inside my *array*. I declare a function called *compute_LDS* that takes my *array* and it's *size* as parameters. Inside *compute_LDS* I declare a *vector* that would hold *LDS* value at each index, initialize all *vector* elements to 1 (its longest length at each index value). Then I go through my *array* elements (*ints*) using double *for* loop, computing each *LDS* value in my *vector* iteratively using Recurrence Relation below:

$$l_i = max\begin{cases} 1 \\ \max\limits_{0<j<i,\, a_j>a_i} \Big\{(l_j+1) \end{cases}$$

Lastly, I display max value in my *vector* which refers to max *LDS* value. Total time complexity $= O(n^2)$.

B. Weighted Edit Distance

Submission#: 180261248

In this program, I start with declaring an two *int* to hold my string A and B sizes. I move on to declaring two *vector* with string sizes(A and B respectively). After inputing both strings, I create a DP Matrix table to hold computed values. Then I declare a function called *compute_DP* that takes both my strings(A and B) with their sizes and my DP matrix as parameters. Inside *compute_DP* I compute all DP values iteratively using recursion below: Recurrence Relation below:

$$DP[i][j] = \begin{cases} max\{i,j\} & (\text{when } i=0 \lor j=0) \\ DP[i-1][j-1] & (\text{when } i>0 \land j>0 \land x_i = y_j) \\ min\begin{cases} DP[i][j-1] + cost\_of\_inserting \\ DP[i-1][j] + cost\_of\_deleting \\ DP[i-1][j-1] + cost\_of\_replacing \end{cases} & (\text{when } i>0 \land j>0 \land x_i \neq y_j) \end{cases}$$

Where:
$cost\_of\_inserting = y_j \ value \ at \ B[j]$
$cost\_of\_deleting = x_i \ value \ at \ A[i]$
$cost\_of\_replacing = |x_i - y_j|$

Lastly, I display last element computed at $DP[A.size][B.size]$ that refers to our min cost to transform A to B. Total time complexity $= O(m*n)$ where($m = A.size$ and $n = B.size$).

C. Longest Unimodal Subsequence

Submission#: 180263692

In this program, firstly I declared an *int* that would hold array size, then I declare an *array* with *int size*. I move on to input all values and store them inside my *array*. Then I declare a function called *compute_LUS* that takes my *array* and it's *size* as parameters. Inside *compute_LUS* I compute *LIS* on all my array elements from LEFT → RIGHT iteratively, saving each index *LIS* value inside a *vector* called *lis_Index_Val* using Recurrence Relation below:

$$l_i = max \begin{cases} 1 \\ \underset{0<j<i,\, a_j<a_i}{max} & \Big\{(l_j+1) \end{cases}$$

Then I compute *LDS* on all my array elements from RIGHT → LEFT iteratively, saving each index *LDS* value inside a *vector* called *lds_Index_Val* using Recurrence Relation below:

$$l_i = max \begin{cases} 1 \\ \underset{0<j<i,\, a_j>a_i}{max} & \Big\{(l_j+1) \end{cases}$$

Lastly I compute *LUS* max value using Longest Bitonic Subsequence logic, declare an *array* called *lus_Index_Val* of *size* identical to my original *array*. Iterate through all indices using Recurrence Relation below:

$$lus\_Index\_Val[i] = (lis\_Index\_Val[i] + lds\_Index\_Val[i])) - 1$$

(where we subtract 1 to remove index double counted in LIS and LDS)

Lastly, I display max value in my *array lus_Index_Val* which refers to max *LUS* value. Total time complexity $= O(n^2) + O(n^2) + O(n) = O(n^2)$.

References

[1] geeksforgeeks *Longest Decreasing Subsequence* https://www.geeksforgeeks.org/longest-decreasing-subsequence/

[2] Sun, Yihan. *"Lecture - Dynamic Programming"* CS 141, University of California Riverside. pdf presentation.

[3] geeksforgeeks *Longest Bitonic Subsequence* https://www.geeksforgeeks.org/longest-bitonic-subsequence-dp-15/