Sabancı University
Faculty of Engineering and Natural Sciences

CS301 – Algorithms

Homework 4

Due: April 2, 2024 @ 23.55
(upload to SUCourse)

PLEASE NOTE:

- Provide only the requested information and nothing more. Unreadable, unintelligible, and irrelevant answers will not be considered.
- Submit only a PDF file. (-20 pts penalty for any other format)
- Not every question of this homework will be graded. We will announce the question(s) that will be graded after the submission.
- You can collaborate with your `TA/INSTRUCTOR ONLY` and discuss the solutions of the problems. However, you have to write down the solutions on your own.
- Plagiarism will not be tolerated.

Late Submission Policy:

- Your homework grade will be decided by multiplying what you normally get from your answers by a "submission time factor (STF)".
- If you submit on time (i.e. before the deadline), your STF is 1. So, you don't lose anything.
- If you submit late, you will lose 0.01 of your STF for every 5 mins of delay.
- We will not accept any homework later than 500 mins after the deadline.
- SUCourse's timestamp will be used for STF computation.
- If you submit multiple times, the last submission time will be used.

Question 1

A palindrome is a sequence of characters that reads the same backward as forward. In other words, if you reverse the sequence, you get the same sequence. For example, "a", "bb", "aba", "ababa", "aabbaa" are palindromes. We also have real words which are palindromes, like "noon", "level", "rotator", etc.

We also have considered the concept of a subsequence in our lectures. Given a word A, B is a subsequence of A, if B is obtained from A by deleting some symbols in A. For example, the following are some of the subsequences of the sequence "*abbdcacdb*": "*da*", "*bcd*", "*abba*", "*abcd*", "*bcacb*", "*bbccdb*", etc.

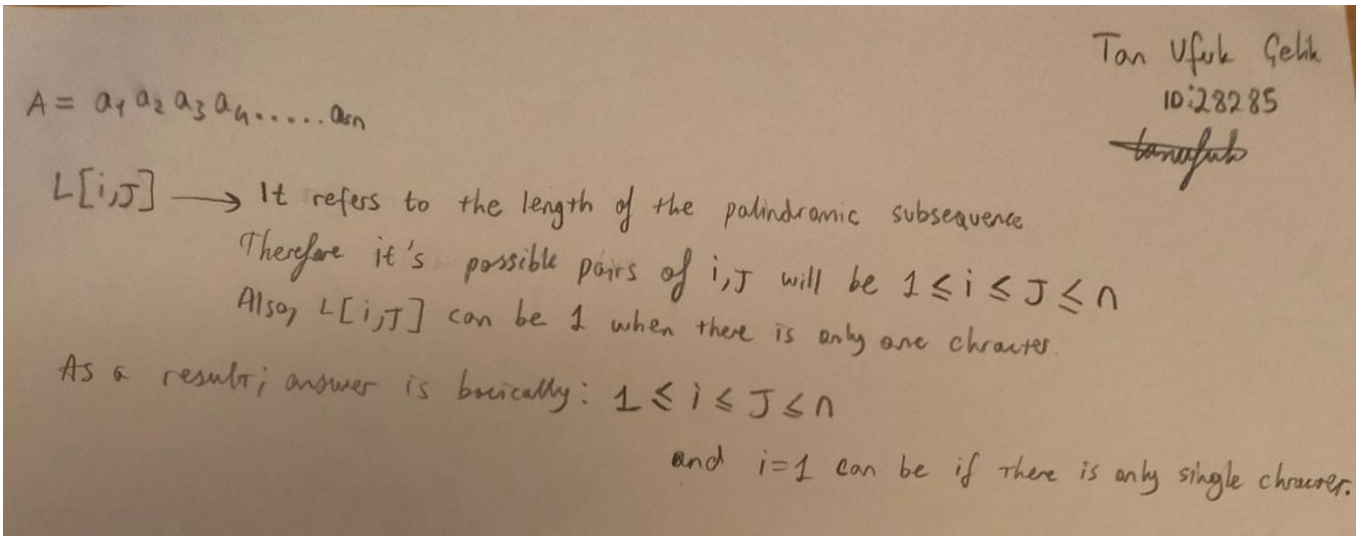
The Longest Palindromic Subsequence (LPS) problem is finding the longest subsequences of a string that is also a palindrome. For example, for the sequence "*abbdcacdb*", the longest palindromic subsequence is "*bdcacdb*", which has length 7. There is no subsequence of "*abbdcacdb*" of length 8 which is a palindrome.

One can find the length of LPS of a given sequence by using dynamic programming. As common in dynamic programming, the solution is based on a recurrence.

Given a sequence $A = a_1a_2 \dots a_n$, let $A[i, j]$ denote the sequence $a_ia_{i+1} \dots a_j$. Hence it is part of the sequence that starts with a_i and ends with a_j (including these symbols). For example, if $A = abcdef$, $A[2, 4] = bcd$, $A[1, 5] = abcde$, $A[3, 4] = cd$, etc.

For a sequence $A = a_1a_2 \dots a_n$, let us use the function $L[i, j]$ to denote the length of the longest palindromic subsequence in $A[i, j]$.

(a) If we have a sequence $A = a_1a_2 \dots a_n$, for which values of i and j, $L[i, j]$ would refer to the length of the longest palindromic subsequence in A?



(b) Write the recurrence for $L[i, j]$.

$$L[i,j] = \begin{cases} 0 & \text{if } i > j \\ \ldots\ldots\ldots\ldots\ldots\ldots & \text{if } i = j \\ \ldots\ldots\ldots\ldots\ldots\ldots & \text{if } i < j \text{ and } a_i = a_j \\ \ldots\ldots\ldots\ldots\ldots\ldots & \text{if } i < j \text{ and } a_i \neq a_j \end{cases}$$

Tan Ufuk Çelik
ID:28285

$L[i,j]=1$, when $i=j$, since each individual character forms a palindrome on its own, $i=j$ can only occur if there is only one character.

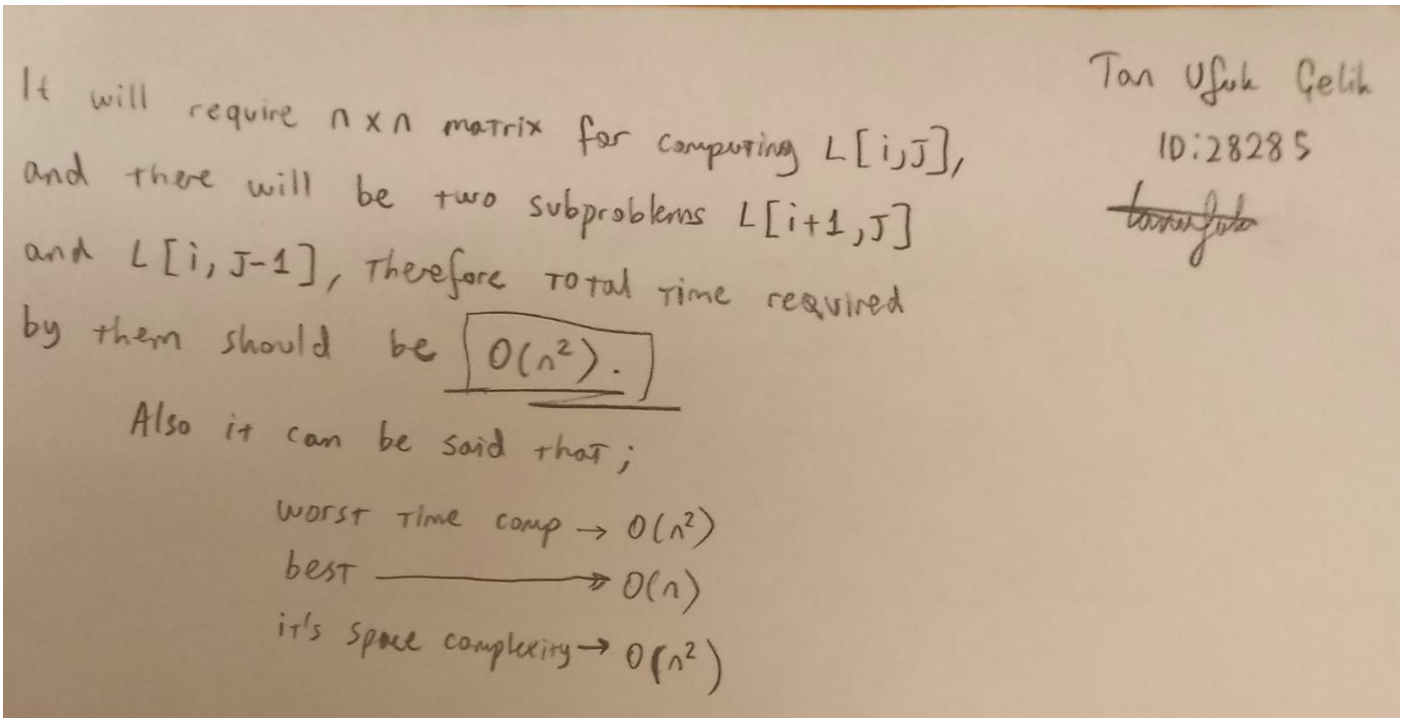
$L[i,j] = L[i+1, j-1]+2$ when $i<j$ and $a_i = a_j$, because the initial and final characters match the subsequence, we can combine them and then explore additional potential subsequences from the remaining elements

$L[i,j] = \max\{L[i+1,j], L[i,j-1]\}$ when $i<j$ and $a_i = a_j$, since the palindromic sequence must include the first or last element, we compute the maximum of these possibilities.

As a result;

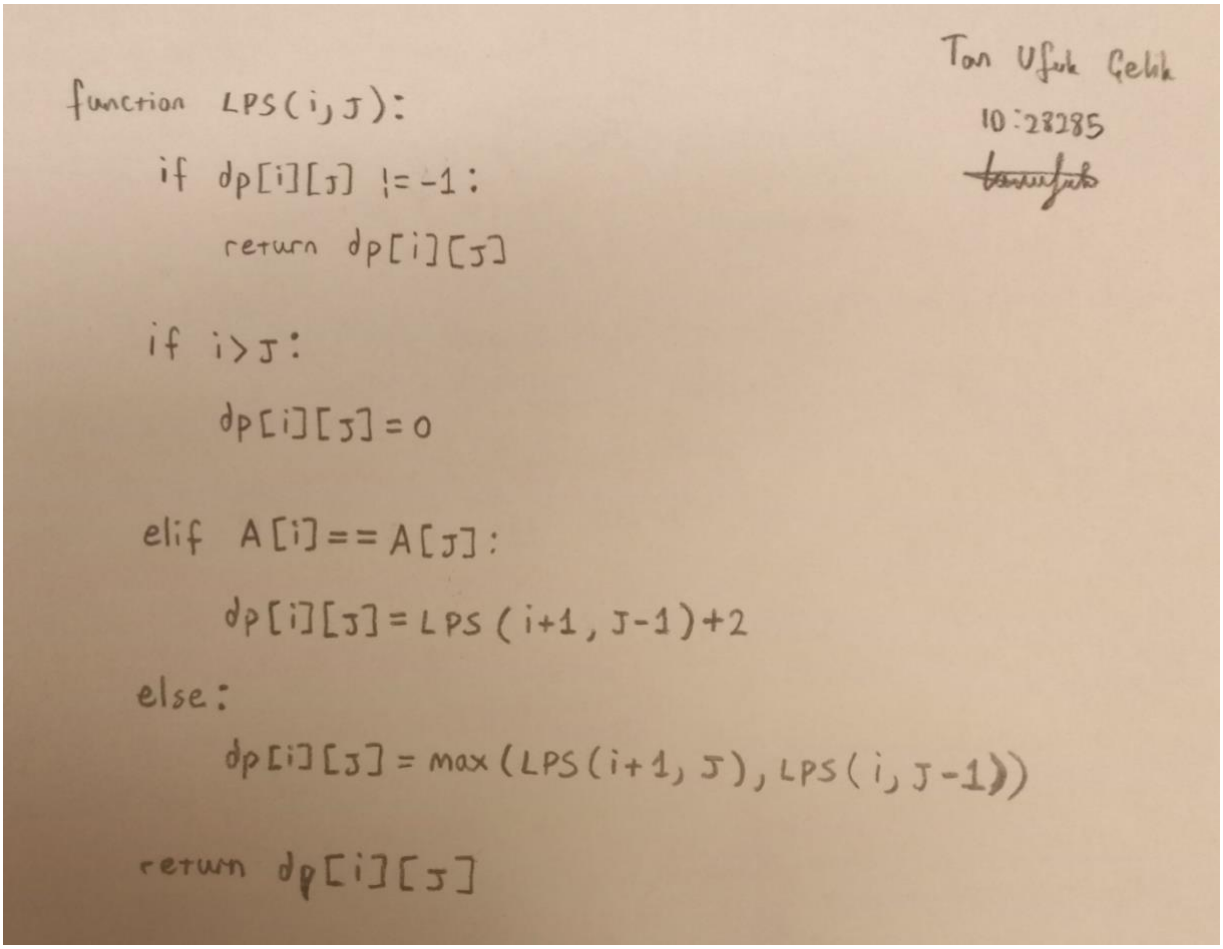
$$L[i,j] = \begin{cases} 0 & \longrightarrow \text{if } i>j \\ L[i,j]=1 & \longrightarrow \text{if } i=j \\ L[i,j] = L[i+1,j-1]+2 & \longrightarrow \text{if } i<j \text{ and } a_i=a_j \\ L[i,j] = \max\{L[i+1,j], L[i,j-1]\} & \longrightarrow \text{if } i<j \text{ and } a_i \neq a_j \end{cases}$$

(c) What would be the worst case time complexity of the algorithm in Θ notation? Why?



(d) Considering a memoization-based approach, complete the following pseudocode for the dynamic programming solution to the LPS problem. Assume the existence of a 2D array 'dp' of size $n \times n$ initialized to -1, where n is the length of sequence A, and a function 'LPS(i, j)' that computes the length of the longest palindromic subsequence in $A[i, j]$.

```
function LPS(i, j):
    if dp[i][j] != -1:
        return ...
    if i > j:
        dp[i][j] = 0
    elif i == j:
        dp[i][j] = ...
    elif A[i] == A[j]:
        dp[i][j] = ...
    else:
        dp[i][j] = max(LPS(..., j), LPS(i, ...))
    return dp[i][j]
```



Question 2

To compute the depth of a given node in an RBT in constant time, consider augmenting the depths of nodes as additional attributes in the nodes of the RBT. Please explain by an example why this augmentation increases the asymptotic time complexity of inserting a node into an RBT in the worst case.

Answer:

Step 1

Including depth information as an additional attribute for each node in a Red-Black Tree can lead to increased worst-case time complexity for node insertion operations. I will explain this situation with an example as it is wanted in the question. If I consider an RBT containing five nodes labeled A to E, with depths ranging from 1 to 5, respectively:

Tan Ufuk Gebi 10.28285

Number is not really matter, just for an example I said 5.

A (It's depth 1)
B (depth 2)
C (depth 3)
D (4)
E (5)

Step 2

We can consider the scenario where I added a new node (let's say that it is F). If we weren't tracking the depth of each node as an extra feature, the insertion process would proceed as usual, adhering to RBT rules and maintaining a worst-case time complexity of $O(\log n)$, where n represents the number of nodes of the tree.

However, if I decide to track the depth of each node, after adding node F, I also need to recalculate and update the depth information of the corresponding nodes to maintain their accuracy. In this scenario, after adding node F, the updated depths would be;

A (depth:1)
B (2)
C (3)
D (4)
E (5)
F (6)

Step 3 To revise depth values, it is necessary to traverse the tree by adjusting the depth property of each node. This process requires $O(n)$ time complexity in the worst case scenario because it requires visiting every node within the tree.

Therefore, when depth augmentation is implemented, adding a node to the RBT involves two different steps: the actual addition of the new node, which is $O(n \log n)$ in time complexity, and the tree traversal for depth updates, which is $O(n)$ in time comp. As a result, with increasing depth, the total time complexity required to add a node to the RBT increases to $O(\log n + n)$, which is basicly $O(n)$ when considering worst-case scenarios.

To summarise, in an RBT, augmenting nodes with depth information shifts the worst-case time complexity of node insertion from $O(\log n)$ to $O(n)$, so it affects the efficiency of tree operations.

Question 3

Write pseudocode for LEFT-ROTATE that operates on nodes in an interval tree and updates the max attributes in $O(1)$ time.

Answer:

Tan Ufuk Gelik
10:28285

```
LEFT-ROTATE (T, x)
    y= x.right
    x.right = y.left

    if y.left ≠ T.nil
        y.left.parent = x

    y.parent = x.parent
    if x.parent == T.nil
        T.root = y
    elseif x == x.parent.left
        x.parent.left =y
    else
        x.parent.right=y

    y.left =x
    x.parent =y

    # Update max attribute of x and y
    x.max = max(x.interval.high, x.left.max, x.right.max)
    y.max = max(y.interval.high, y.left.max, y.right.max)
```

Explanation:

1) T is the interval tree.
2) x is the node being rotated
3) x.left, x.right, x.parent, x.interval, x.max are the attributes of node x
4) Max → max endpoint of an interval in the subtree rooted at x.
5) T.nil → sentinel node used in the interval tree

– THE END –