Sabancı University
Faculty of Engineering and Natural Sciences

CS301 – Algorithms

Homework 2

Due: April 26, 2023 @ 23.55
(Upload to SUCourse)

PLEASE NOTE:

- Provide only the requested information and nothing more. Unreadable, unintelligible and irrelevant answers will not be considered.
- 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	Points	Score
1	40	
2	40	
3	20	
Total:	100	

Question 1 [40 points]

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 $A = a_1a_2 \ldots a_n$, let $A[i, j]$ denote the sequence $a_ia_{i+1} \ldots a_j$. 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 $A = a_1a_2 \ldots a_n$, let us use the function $L[i, j]$ to denote the length of the longest palindromic subsequence in $A[i, j]$.

(a) [10 points] If we have a sequence $A = a_1a_2 \ldots a_n$, for which values of i and j, $L[i, j]$ would refer to the length of the longest palindromic subsequence in A?

For i = 1, j = n. Since the entire sequence is given as a1a2...an, L[1,n] would give the longest palindromic subsequence.

(b) [20 points] Write the recurrence for $L[i, j]$.

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

If i == j, single character is always considered as a palindrome.

If i<j and ai = aj, since the characters at the position i and j included in the palindrome, we need to find the remaining sequence between i+1 and j-1. Therefore, 2 + L[i+1, j-1]

If i < j and ai != aj, we need to find the longest palindromic sequence by not including the character at the position i or character at the position j. Therefore, max(L[i+1,j],L[i,j-1])

(c) [10 points] What would be the worst case time complexity of the algorithm in Θ notation? Why?

> Θ(n^2) We need to compute every possible sequence from single character to the whole sequence
> Therefore, we need to find L[i,j] for every possible combination of the i and j. Which is Θ(n^2)

Question 2 [40 points]

Consider the 0–1 knapsack problem, where we have a set of n objects $(o_1, o_2, \ldots, o_n)$, each with a weight $(w_1, w_2, \ldots, w_n)$ and a value $(v_1, v_2, \ldots, v_n)$. Here, the object o_i has the weight w_i and the value v_i. Suppose that W is the capacity of our knapsack. We would like to compute the maximum value that we can pack into our backpack.

Let $P[i, j]$ denote the maximum value that can be packed into a knapsack with capacity j, if we consider only the first i objects, i.e. $o_1, o_2, \ldots, o_i$.

(a) [10 points] For which values of i and j, $P[i, j]$ would refer to the maximum value that can be packed into our knapsack of capacity W if we consider all n items?

> P[n,W] i=n, j=W since we want to consider all n items and use the full capacity.

(b) [20 points] Write the recurrence for $P[i, j]$.

$$P[i,j] = \begin{cases} \text{0} & \text{if } i = 0 \\ \text{P[i-1,j]} & \text{if } i > 0 \text{ and } j < w_i \\ \text{max(P[i-1,j], vi + P[i-1, j-wi])} & \text{if } i > 0 \text{ and } j \geq w_i \end{cases}$$

> If i = 0, no items to select, therefore P[i,j] = 0
>
> If i > 0 and j < wi, item i is not included, so we can only use i-1 items with capacity j. Therefore, P[i,j] = P[i-1,j]
>
> If i > 0 and j >= wi, there are two possibilities, whether to include item i or not, so we need to choose the maximum. If we exclude the item, we can use first i-1 items to find the maximum value. If included, value of item i plus the maximum value with capacity j-wi using only the first i-1 items.
> Therefore, max(P[i-1,j], vi + P[i-1, j-wi])

(c) [10 points] What would be the worst case time complexity of the algorithm in Θ notation? Why?

> Θ(nW), because we need to compute each combination of items and capacities, from 1 to n, 1 to W.
> Therefore, there are total of n*W combinations.

Question 3 [20 points]

Consider again the 0–1 Knapsack problem. This time, instead of developing a dynamic programming solution, we would like to suggest a greedy solution.

If we have a set of n objects $(o_1, o_2, \ldots, o_n)$, each with a weight $(w_1, w_2, \ldots, w_n)$ and a value $(v_1, v_2, \ldots, v_n)$, and if we have a capacity W:

(a) [10 points] What would be a greedy choice to pick the object to be included in our knapsack at this point?

(b) [10 points] If we pick the object o_i by the greedy choice in part (a), what would be the subproblem that we will be left with, after this choice of object?