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	

please proceed to the next page ⟶

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 "*bcacb*", which has length 5. There is no subsequence of "*abbdcacdb*" of length 6 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?

To find the length of the longest palindromic subsequence of the whole sequence A = a1a2 ...an, we would use L[1,n], which refers to the length of the longest palindromic subsequence in the subsequence A[1,n], i.e., the entire sequence A.

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

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

If i=j then it means that there is only one letter i the word such as "a" and thus 1 s the length of the palindrom i.e L[i,j] = 1

If i < j and ai = aj, A[i,j] starts and ends with the same symbol. Any subsequence consisting of only two identical symbols is itself a palindrome.A[i,j] must contain both ai and aj, since they are equal and form a palindrome of length 2. Thus, we can set L[i,j] = 2.

If i < j and ai ≠ aj,means the first and last characters are different, the length of the longest palindromic subsequence is the maximum of the length of the longest palindromic subsequence in the subsequence A[i+1, j] and the subsequence A[i, j-1] so we can etiher include ai or aj since they are different

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

Worst case complexity is O(n^2)because the algorithm fills up an n x n matrix with L[i,j] values, where each entry takes constant time to compute. Therefore, the total time taken is proportional to the number of entries in the matrix, which is 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?

```
If we consider all n items and the capacity of the knapsack is W, then we have i = n and
j = W. Therefore, P[n,W] would refer to the maximum value that can be packed into our
knapsack of capacity W if we consider all n items.
```

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

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

```
If there are no items to consider (i = 0) or the capacity of the knapsack is 0 (j = 0),
then we cannot pack any value, so P[i,j] = 0.

If the weight of the current item (w[i]) is greater than the remaining capacity of the
knapsack (j < w[i]), then we cannot include the current item, so the maximum value we
can pack is the same as the maximum value we can pack without the current item, i.e.
P[i-1,j].

 If we can include the current item (j ≥ w[i]), then we have two options: either include
the current item and the maximum value becomes P[i-1,j-w[i]] + v[i], or do not include
the current item and the maximum value remains P[i-1,j]. We choose the maximum of these
two options as the value for P[i,j].
```

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

```
Worst case complexity is O(nW) because we need to fill up a table of size (n+1) x (W+1)
to store the maximum value that can be packed for all possible combinations of items and
knapsack capacities.
```

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?

One possible greedy algorithm is to compute the value-to-weight ratio for each item, at each step of the algorithm, we compute the value-to-weight ratio for each remaining item i, defined as ri = vi/wi. We then pick the item with the highest ratio that can still fit in the knapsack, i.e. the item with the maximum ri such that wi ≤ W.This greedy criterion is based on the intuition that we want to maximize the value we can pack in the knapsack per unit of weight, since the weight capacity is limited. By picking the item with the highest value-to-weight ratio, we are prioritizing items that are most valuable relative to their weight.

(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?

A reduced capacity W - wi, since we have added an item of weight wi to the knapsack. A reduced set of items, consisting of all the remaining items except oi.
In other words, we need to solve the subproblem of finding the maximum value that can be packed into a knapsack with capacity W - wi, using only the remaining n-1 items (excluding oi). Formally, the subproblem can be defined as finding the maximum value that can be obtained by selecting a subset of items {o1, o2, ..., oi-1, oi+1, ..., on} with total weight at most W - wi. We can use a similar approach as in part (a) to solve this subproblem, either recursively or iteratively, by applying the same greedy criterion of maximizing the value-to-weight ratio at each step.

ZEYNEP KURTULUS
29045