CS301 Algorithms - Homework 3

Ege Demirci - 28287

Sabanci University, Spring 2022-2023, Algorithms

1. Question - 1

1.1. Part - A

Since $L[i,j]$ refers to the length of the longest palindromic subsequence in the subsequence $A[i,j]$; i and j's range should be $1 \leq i \leq j \leq n$. As it can be understood from the inequality, from starting index to last index would define the longest subsequence so it would also define the length of the longest palindromic subsequence. So for values $i = 1$ and $j = n$ the longest palindromic subsequence defined. The answer is: $L[1,n]$

1.2. Part - B

The recurrence for the given problem is:

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

1.3. Part - C

The worst time complexity of the algorithm would be $\theta(n^2)$. The reason is simply about the 2D matrix we used in dynamic programming for the problem. We should fill every entry of the matrix using the recurrence. The number of rows and columns of this matrix should be n since we will compare the sequence with itself. Thus, we will fill the n^2 entries, and each filling will take constant time (at most we're looking at 2 other cells). So the worst time complexity of the algorithm is $\theta(n^2)$.

2. Question - 2

2.1. Part - A

Since $P[i,j]$ refers to the maximum value that can be packed into a knapsack with capacity j, we should pack every object possible to get the maximum value. Since i refers to the first i object and we're trying to pack every object; i should be equal to the number of objects so $i = n$

For the j part, the maximum capacity that we can use is W (since knapsack's capacity is W), we cannot exceed this capacity so $j = W$

The answer is: $P[n,W]$

2.2. Part - B

The recurrence for the given problem is:

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

2.3. Part - C

The worst time complexity of the algorithm would be $\theta(nW)$. The reason is simply about the 2D matrix we used in dynamic programming for the problem. We should fill every entry of the matrix using the recurrence. The number of rows should be n+1; this is because there are n objects we can put in knapsack but i starts from zero. The number of columns should be W; since this is the maximum capacity possible for our knapsack. Thus, we will fill the $(n * W) + W$ entries, and each filling will take constant time. $n * W$ is dominant term, so the worst time complexity of the algorithm is $\theta(nW)$.

3. Question - 3

3.1. Part - A

The greedy choice to pick the object to be included in our knapsack would be the choosing the object that creates maximum value/weight ratio. In other words the object o_i with the maximum v_i/w_i (it should also satisfy $w_i \leq W$).

3.2. Part - B

After picking the object o_i by the greedy choice in part (a), we will be left with a subproblem of finding the maximum value that can be packed into a knapsack with capacity $W - w_i$ using the remaining objects $o_1, o_2, \ldots, o_{i-1}, o_{i+1}, \ldots, o_n$. From the different aspect, we need to solve the knapsack problem with a reduced capacity of $W - w_i$ and a reduced set of objects excluding o_i.