CSE 321 Introduction to Algorithm Design
Homework 5 Report

Ahmet Yuşa Telli
151044092

January 2020

1 Dynamic Programming - Location

We need to find minimum cost when we travel between cities. Each cities have their own cost and we have to pay minimum money to cities. But we consider the moving cost. When we go to the city to another city, we check the moving cost. If the next city is more expensive than the moving cost, we should not go to the city. For example these are our cities costs:
$NY1 = [1, 3, 20, 30]$
$SF1 = [50, 20, 2, 4]$
When we chance the city, we should check the next cost. We have one loop for this calculation from 1 to length of the cities cost array. Then, the time complexity is $T(n) \in O(n)$
All cases are the same result is $O(n)$

2 Greedy Algorithm - Sessions

A student wants to join as much sessions as possible. We need to find optimal list of sessions with the maximum number of sessions. We should design the greedy algorithm for this problem. The student can be only one session at the same time. We should design the algorithm for one student. And the student can not leave any sessions before it is over.

In the algorithm, we have 2 integer arrays, one is start time and other one is finish time of sessions.These are our arrays:
$start = [1, 3, 0, 5, 8, 5]$
$finish = [2, 4, 6, 7, 9, 9]$
Firstly, we sort the start array. While we sort the start array ,we need to change the finish indexes. When we sort, we use Insertion Sort. The Insertion sort worst case is $T(n) \in O(n^2)$.

After sort, we try to all session combinations. We have 2 for loop, one is start from 0 to length of start, other one is start from 1 to length of start.

Then, we compare previous session's finish time and next session's start time. Then, we add a list all list of sessions and find the list with most elements. The function's worst case's time complexity is
$T(n) = n \times (n-1) + n^2$
$T(n) = n^2 - n + n^2$
$T(n) = 2 \times n^2 - n$
$\Rightarrow T(n) \in \theta(n^2)$

3 Subset with Total Sum is 0

We have a unsorted, mixed array. And we need to find a subset which sum of elements equal to zero. In this algorithm, We search all subsets and check the sum of elements. After find subsets, we find sum, if we find "0" then return it. This is our array: $array = [10, 3, 9, -5, 4, 2, -7, 8]$
After the running we find this subset: $Subset : [-5, 4, -7, 8]$
When we finding the subsets, we have a recursive call and its time complexity is $T(n) \in O(2^n)$

4 Alignment Sequence

We have two string sequences. We create a 2D array and its size is length of these sequences(M+1 x N+1). We fill the zeros. On this array, we compare our sequences and if one character is the same we add two to array. When adding, we check the before characters equalization. If match some chars, we add to other array. If there is mist match, we add another array and we minus one from the array. After that we have this array:
$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$
$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$
$[0, 0, 2, 1, 0, 0, 0, 0, 0, 0]$
$[0, 0, 1, 3, 2, 1, 0, 0, 0, 0]$
$[0, 0, 0, 2, 1, 0, 2, 1, 0, 0]$
$[0, 0, 0, 1, 0, 0, 1, 3, 2, 1]$

And, we check the elements of this, we find the match, gap characters. Time complexity is $T(n) \in O(m \times n)$. m = length of sequence1, n = length of sequence2. All cases are the same result.

5 Greedy - Operations

We have an array and we need to find sum of elements with minimum operations. We use subarrays. We take three elements of array and we find minimum two elements in this subarray. Then, we calculate sum and operation. The maximum

one will go to the next subarray. At the end, if we have one or two elements, before we add the minimum element and add the other one.
We have a loop for check length is greater than 3. This time complexity is $T(n) \in O(n-3) \Rightarrow T(n) \in O(n)$