Istanbul Technical University
Faculty of Computer and Informatics
Computer Engineering Department

BLG 336E
The LaTeX
Report

Analysis of Algorithms II, Project 3

Leminur Çelik - 150190085

May 18th, 2023

Contents

1 Description of Code

1.1 Pseudo-code

1.1.1 Best Possible Schedule for Each Place

I implement compare time function to sort places for non-decreasing ending times and the binary search function for finding available times of place. I designed a plan that maximizes the number of chairs of each place, taking into account the daily schedules and capacities of the places.

Algorithm 1 Find Maximum Chairs

```
function FindMaxChairs(*input, start, end)
    Sort places by non-decreasing ending times
    schedule ← vector of Input
    taken ← vector of bool with ending time subtract starting time, initialized to false
    array ← pointer array of int with ending time subtract starting time
    previous ← array of int with ending time subtract starting time
    array[0] ← input[0].getCapacity()
    i ← start + 1
    while i < end do
        incChair ← input[i].getCapacity()
        l ← BinarySearch(input, start, i)
        if l ≠ −1 then
            incChair+ = array[l − start]
        end if
        if incChair > array[i − start − 1] then
            array[i − start] = incChair
            taken[i − start] = true
            previous[i − start] = l − start
        end if
        array[i − start] = array[i − start − 1]
        previous[i − start] = i − start − 1
        result ← array[end − start − 1]
        return result
    end while
end function
```

1.1.2 Best Possible Plan for Tour

I implement compare date function to sort places for non-decreasing ending dates and the binary search function for finding available date of tour. I designed a plan that maximizes the number of chairs, taking into account the availability intervals of the places.

Algorithm 2 Find Maximum Revenue

```
function FindMaxRevenue(*input, end)
    Sort places by non-decreasing ending dates
    schedule ← vector of Input
    taken ← vector of bool with ending time, initialized to false
    array ← pointer array of int with ending time
    previous ← array of int with ending time
    array[0] ← input[0].getCapacity()
    i ← 1
    while i < end do
        incRevenue ← input[i].getCapacity()
        l ← BinarySearchDate(input, i)
        if l ≠ −1 then
            incRevenue+ = array[l]
        end if
        if incChair > array[i − 1] then
            array[i] = incRevenue
            taken[i] = true
            previous[i] = l
        end if
        array[i] = array[i − 1]
        previous[i] = i − 1
        revenue ← array[end − 1]
        return revenue
    end while
end function
```

1.2 Time Complexity

Time complexity of compare time and compare date functions are O(1). The time complexity of the sort function is O(nlogn). It takes O(logn) to get the previous executable for each place using the modified binary search. Therefore, as a result, this algorithm runs in O(nlogn).