Exercises in Algorithms No 2

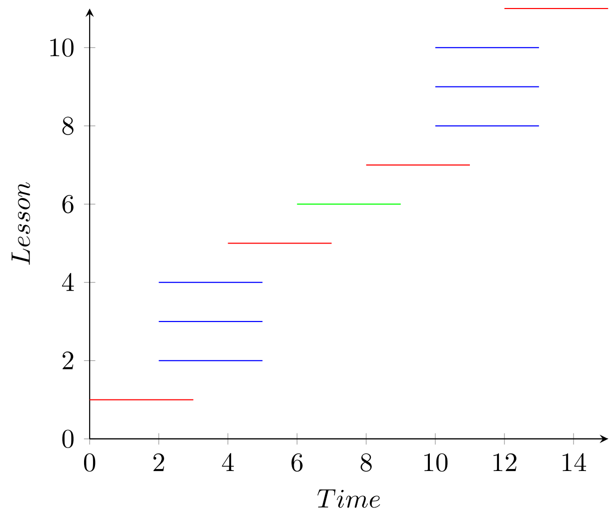
Alexandros Kyriakakis (el12163)

November 2018

1 Courses' Selection and Completion

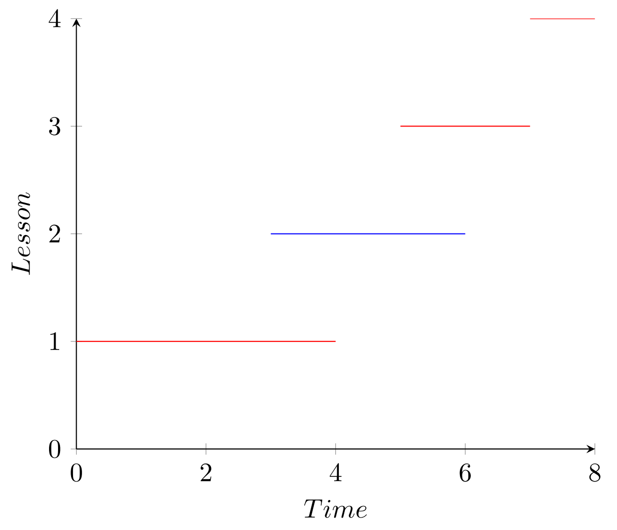
1.1 Greedy Courses' Selection

1.1.1 Less Overlaps



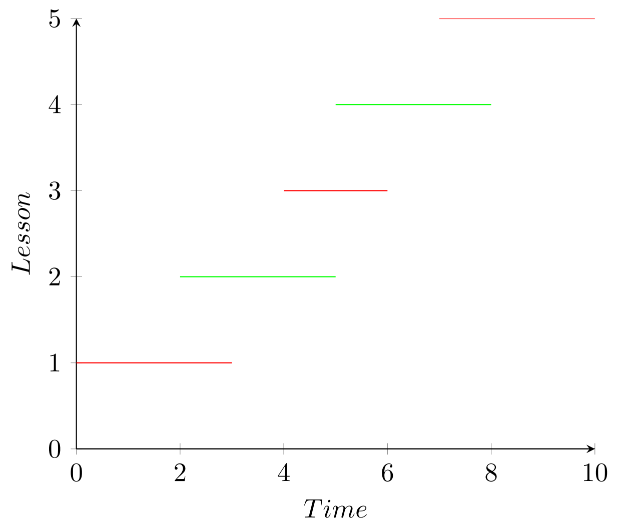
At the example above we see that lesson "6" has the least amount of overlaps but if we follow the given algorithm we will remove lessons "5" and "7" which are part of the optimal solution. Given algorithm leads to lessons "1", "6" and "11" while optimal (red) leads to lessons "1", "5", "7" and "11".

1.1.2 Most duration



At the example above the lesson "2" overlaps the lessons "1" and "3". If we remove the lesson "2" we have three non-overlapping lessons. If we follow the algorithm of the most duration we remove at first lesson "1" which have the most duration but we still have overlap between lessons "2" and "3" so we remove the lesson 2. At the end of this algorithm the the leftover lessons will be "3" and "4". So this algorithm leads to less lessons than the example we gave by removing only lesson "2" so it's wrong.

1.1.3 Most overlaps



At the example above lessons "1" and "5" have one overlap each and lessons

"2", "3" and "4" have two overlaps each. Then if the algorithm choose to remove the lesson "3" ends up with only two lessons. But if we remove lessons "2" and "4" we make a session with three lessons. So the un-determination of the algorithm can lead to wrong answer.

1.2 ECTS

This problem will be solved with dynamic programming. At first we sort the lessons ascending the finishing moments. We allocate a matrix $ECTS[n][W_{max}]$ to save the optimal solutions of the sub-problems for reuse. Then there are to options for the n lesson. First to be part of the optimal solution, and second not being part of it. This is shown by the following recursive equation:

$$ECTS[n][w_{max}] = max \begin{cases} ECTS[n-1][W] \\ w_n + ECTS[n-1][W - w_n] \end{cases}$$

The above dynamic programming time complexity is $O(n^2)$.

1.3 Fast info transport

Generally main part of the algorithm is that you have to pass the minimum once from each lesson.

Algorithm Sort for less worries

```
$Struct\ Class:\ int\ x,\ int*\ y$
$Class\ F[n].x \leftarrow f_i,\ S[n].x \leftarrow s_i$
$GoneThere[n]$
$j \leftarrow 0$
for $i = 0 \rightarrow n$ do
    $GoneThere[i] \leftarrow 0$
    $F[i].y\ =\ GoneThere\ +\ i$
    $S[i].y\ =\ GoneThere\ +\ i$
end for
$Sort\ F[n].x$
$SortS[n].x$
for $i = 0 \rightarrow n$ do
    if $GoneThere$ then
        continue
    else
        $GoneThere[i] = 1$
        while $F[i].x > S[j].x$ do
            if GoneThere[j] then
                $j++$
            else
                $GoneThere[j] = 1$
                $j++$
            end if
        end while
    end if
end for
```

So if we sort the finishing time of all the lessons. We go to the first sorted lesson's finish moment and check which lessons have already started so we go also there. Then we mark all the lessons that we've already gone as "GoneThere". We continue to the next available lesson and do the same. The complexity of this algorithm is $O(n \log n)$ because sorting costs $2 * O(n \log n)$ for each price of starting time and finish time and the process costs $2 * n$ as we run the times the arrays.

2 Video Game

Our first goal is to earn the biggest amount of money. We accomplish that by hitting the boxes, starting from the one that includes the most money and then continue to the second one box includes the most money and go on. We stop when there are no boxes or no hammers, or hammers with less power than the power we need to open the box with the less power needed. So the optimal way to do it is by hitting the box with the most money with the hammer forces the less power to open it.

Algorithm Earn the most Lose the Least

```
for $i = 1 \to n$ do
    $Sort\ Boxes\ U[i]$
end for
for $j = 1 \to n$ do
    $Sort\ Hammers\ F[j]\ to\ i\ teams\ with\ Ui\ as\ pivot\ in\ each\ team\ using\ AVL$
end for
for $i = n \to 0$ do
    $Bsearch\ Pi\ in\ F[j]$
    $Use\ the\ Hammer\ to\ open\ the\ Box.\ (destroy\ the\ hammer)$
end for
```

Lines 1-3 need $O(i \log i)$ time. Also Lines 4-6 need $O(j \log i)$ which exists because we proved it to the previous exercise series No1. The last "for loop" in Lines 7-10 needs $O(i \log i)$ time because we search in i teams where we need $logi$ time to go, we repeat this i times for each box. So generally we need $O(i \log i)$ time to solve the whole problem if $i > j$ or $O(j \log i)$ if not.

3 Souvenirs

Our goal is to buy the souvenirs with the most sentimental value from each country with total price C. This is a classic DP problem.

Algorithm Love Traveling

```
1: $Pleasure[n][C][k_{max}]$
2: $Cost[n][C][k_{max}]$
3: for $i = 0 \rightarrow n$ do
4:     $Pleasure[i, 0, 0] \leftarrow 0$
5:     $Cost[i, 0, 0] \leftarrow 0$
6: end for
7: for $i = 0 \rightarrow max_K$ do
8:     $Pleasure[0, 0, i] \leftarrow 0$
9:     $Cost[0, 0, i] \leftarrow 0$
10: end for
11: for $i = 1 \rightarrow n$ do
12:     $MinSum \leftarrow MinSum + Min\ cost\ souvenir\ of\ the\ country\ i$
13:     for $j = 1 \rightarrow k_i$ do
14:         for $c = 0 \rightarrow C$ do
15:             if $c - c_{ij} \geq MinSum$ then
16:                 $currentP \leftarrow Pleasure[\text{i-1}, c - c_{ij}, k_{i-1}] + p_{ij}$
17:                 $currentC \leftarrow Cost[i - 1, c - c_{ij}, k_{i-1}] + c_{ij}$
18:             else
19:                 $currentP \leftarrow 0$
20:             end if
21:             if $Pleasure[i, c,\text{j-1}] \geq current$ then
22:                 $Pleasure[i, c, j] \leftarrow Pleasure[i, c,\text{j-1}]$
23:                 $Cost[i, c, j] \leftarrow Cost[i, c, j - 1]$
24:             else
25:                 $Pleasure[i, c, j] \leftarrow currentP$
26:                 $Cost[i, c, j] \leftarrow currentC$
27:             end if
28:         end for
29:     end for
30: end for
31: return $Pleasure[n, C, k_{max}]$
```

At first we create two matrices (lines 1 $\rightarrow$ 2) where we will save all the results of the process for reuse. Also we initialize to 0 the parts of the matrices where the $c = 0$ (lines 3 $\rightarrow$ 10) where probably there's no result to extract. Then for every souvenir from every country we find the optimal choice while c is increasing and we store the optimal solutions to the matrix $Pleasure[]$ to use them again. When these "For Loop" ends we'll have the total optimal result. Specifically, (at lines 15 $\rightarrow$ 20) if the element ij cost is less than the current value of c deducted by the value of the less expensive elements from the previous countries then save the "Pleasure" and the "Cost" of that choice of elements to the "currentP" and "currentC" variables respectively, else save 0. If the "Pleasure" of the current elements is bigger than the previous choice of elements (lines 21 $\rightarrow$ 27) save the values of current "Cost" and "Pleasure" at the matrices, else save the last best choice. Continue the process until you check all the elements of the value C of money. Return the value of the last element $Pleasure[n-1, C-1, k_n - 1]$. So the time complexity is $O(n * C * k_{max})$

4 Chocolates

This problem is another implementation of Dynamic Programming. The following algorithm has been made according to this idea. We have a left-meter (i) and a right-meter (j) that counts how long have gone right and left from the starting point. At a specific point ij we can go with four ways. If we go to i we can go from i-1 or j and if we go to j we can go from j-1 or i so we are looking the most amount of chocolates for the least amount of steps. This leads to $O(n^2)$ time complexity solution where we use a nXn matrix to save what we count.

Algorithm 1 Bedtime Happiness

```
$Quantity[n][n]$
$Time[n][n]$
$Quantity[0,0] \leftarrow 0$
$Time[0,0] \leftarrow 0$
for $i = 0 \rightarrow n$ do
    $Quantity[i,0] \leftarrow 0$
    $Quantity[0,i] \leftarrow 0$
    $Time[i,0] \leftarrow i$
    $Time[0,i] \leftarrow i$
end for
for $i = 0 \rightarrow n/2$ do
    for $j = 0 \rightarrow n/2$ do
        if $q_i \geq q_{i-1}$ and $t_i \neq t_{i-1}$ then
            $CurrentQ1 = q_i + Quantity[i-1][j]$
            $CurrentT1 = 1 + Time[i-1][j]$
        else
            $CurrentQ1 = 0$
            $CurrentT1 = \infty$
        end if
        if $q_j \geq q_{j-1}$ and $t_j \neq t_{j-1}$ then
            $CurrentQ2 = q_j + Quantity[i][j-1]$
            $CurrentT2 = 1 + Time[i][j-1]$
        else
            $CurrentQ2 = 0$
            $CurrentT2 = \infty$
        end if
        if $q_i \geq q_j$ and $t_i \neq t_j$ then
            $CurrentQ3 = q_i + Quantity[i-1][j]$
            $CurrentT3 = j + i - 1 + Time[i-1][j]$
        else
            $CurrentQ3 = 0$
            $CurrentT3 = \infty$
        end if
        if $q_j \geq q_i$ and $t_i \neq t_j$ then
            $CurrentQ4 = q_j + Quantity[i][j-1]$
            $CurrentT4 = j + i - 1 + Time[i][j-1]$
        else
            $CurrentQ4 = 0$
            $CurrentT4 = \infty$
        end if
        $Quantity[i][j] = max(CurrentQ1,\ CurrentQ2,\ CurrentQ3,\ CurrentQ4)$
        if Two high prices are the same then
            Keep the one with the least time
        end if
    end for
end for
```

5 Transmitters and Receivers

At this problem we have a general purpose, which is to find a way to minimize the energy consumption. We can solve this problem by using this argument: "Use as Receiver the antenna with the highest price of $T_j - R_j$"

Algorithm Green Wallet

```
1:  $Transmitter[2n] \leftarrow T_i$
2:  $Receiver[2n] \leftarrow R_j$
3:  Vector: $ArgumentTree[2n]\ (x, y)$
4:  for $i = 0 \rightarrow 2n$ do
5:      $ArgumentTree[i].x = Transmitter[i] - Receiver[i]$
6:      $ArgumentTree[i].y = i$
7:  end for
8:  $Sort\ ArgumentTree.x\ and\ Save\ it\ as\ AVL$
9:  for $i = 0 \rightarrow 2n$ do
10:     if $*p = BSearch(Transmitter[i] - Receiver[i],\ ArgumentTree.x) \neq NULL$ then
11:         $Remove\ *p\ from\ ArgumentTree$
12:         return $(i,\ (*min = MinSearch(ArgumentTree.x)).y)$
13:         $Remove\ *min\ Element\ from\ ArgumentTree$
14:     else
15:         $Continue$
16:     end if
17: end for
```

First of all we save all the prices of the Antennas at two Arrays of $2n$ size *Transmitter* for T_i's and *Receiver* for R_j (Lines 1 $\rightarrow$ 2). Then we create a Tree data structure of Vectors (x,y) (at line 3). At lines 4 $\rightarrow$ 7 we apply a "For Loop" to initialize the prices for every x value to the vectors as the price of the i^{th} Antenna energy cost for transmitting minus the cost of receiving. Then at the y value of these Vectors we save the original position of the element this costs $O(n)$. Then we Sort the tree according to the x values of the vectors, so we transpose the tree to an AVL tree with time cost $O(nlog(n))$. Now according to the fact that the first element is always a Transmitter, we run a "For Loop" which matches the antenna with the current least $T_i - R_j$ with the current first element. After every loop we remove the used elements from the Tree. Specifically, while i runs from 0 to 2n, at first we check if the element i already exists. We do it by B-Searching the current T_i at the tree. If the B-Search returns a value different from NULL it means that the i element already exists and also means that its the first because after we find it at line 10 we remove it at line 11 so we've removed all the elements before that i, so this element will be a Transmitter. Now as for the receiver we take the element with the least value $T_i - r_j$ and we return the index of each element at line 12. Then we remove the antenna that we used as a current receiver we loop again. We remove the *p before Min-Search because it could resolve to the same element. So this costs $O(log(n))$ for B-Search, $O(log(n))$ for every Element-Remove, $O(log(n))$ also for Min-Search function because we use an AVL tree and we repeat these 2n times so Total time complexity for that algorithm is $O(log(n))$.

We prove these with the Exchange Argument. Assume that P is the greedy solution that we've made with $E(P)$ power consumption and P^* is the optimal solution of this problem with $E(P^*)$ energy consumption. If we exchange a couple of antennas from P^* taken as a transmitter and receiver with a couple of the antennas we've chosen with our greedy criterion we will show that $E(P) \leq E(P^*)$. Lets exchange the first couple of antennas so we take as a receiver an antenna with $T_i - R_i \geq T_i^* - R_i^* \;\Rightarrow\; T_i - T_i^* \geq R_i - R_i^*$ this means that if we exchange the optimal receiver with our greedy receiver then amount of energy that will add will be equal or less than the amount of energy would be added if this was a transmitter. So if we exchange all k^{th} antennas with the greedy selected the total anount of energy will be equal or less than the optimal. So our solution is the optimal.