Fundamental Algorithms - Spring 2018
Homework 8

Daniel Rivera Ruiz
Department of Computer Science
New York University
drr342@nyu.edu

1. The following is the pseudocode for the algorithm:

```
1    R[0] = 0
2    for j = 1 to N^2:
3        R[j] = 0
4        for k = 1 to W:
5            TEMP = PRICE[k] + R[j - k]
6            if (R[j] < TEMP):
7                R[j] = TEMP
8            end if
9        end for
10   end for
11   return R[N^2]
```

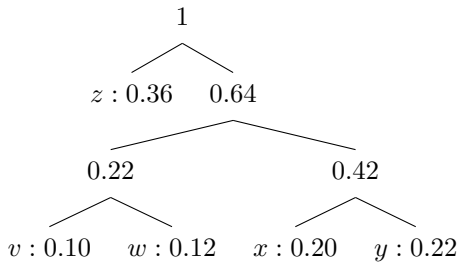
   All the operations executed in the inner `for` loop of the algorithm take constant time, so its time complexity is $O(W)$. The outer `for` loop is executed N^2 times, so the overall time complexity of the algorithm is:

$$T(N) = N^2 \cdot O(W)$$
$$T(N) = O(N^2 W)$$
$$T(N) = O(N^2 \left\lfloor \sqrt{N} \right\rfloor)$$
$$T(N) = O(N^{\frac{5}{2}})$$

   The algorithm works as follows:

   - On line 1 we establish the base case $R[0] = 0$.
   - On lines 2 through 10, we iterate over all the possible values of j to get the revenue values R.
   - Line 3 initializes $R[j]$ to zero (assuming that it is the minimum value the revenue can take).
   - Lines 4 through 9 define the position of the first cut in the rod based in the conditions of the problem: 1) the rod lengths must be at most W (hence the `for` loop from 1 to W) and 2) we want to maximize the total revenue. So while we traverse the loop, we compute the revenue produced by making the first cut at position k which is equals to $P[k] + R[j - k]$. If this value is bigger than the current value of $R[j]$ we update it.
   - Finally, on line 11 we return the desired value $R[N^2]$.

2. Lets suppose that the frequency for z is less than $\frac{1}{3}$ and z has an encoding of length 1. This means that z must not merge with any other node until the end of the Huffman algorithm. If we look at the last stage where there are only tree elements left, the other two elements (lets call them a and b) must merge to form another node so that z still corresponds to a codeword of length 1. At this point we have $f(z) + f(a) + f(b) = 1$ and $f(z) < \frac{1}{3}$, which implies $f(a) + f(b) > \frac{2}{3}$.
   The last inequality entails that at least one of $f(a)$ or $f(b)$ must be greater than $\frac{1}{3}$, but in that case according to the Huffman algorithm they wouldn't merge, since choosing z and $argmin(f(a), f(b))$ would yield a better solution. This leads to a contradiction of the original assumption, which means that for any character to be encoded in length one, it must have frequency at least $\frac{1}{3}$.
   The following tree is an example for $f(z) = 0.36$ with z encoded as 0:



3. Selecting the last activity to start that is compatible with all previously selected activities will traverse the time available for the activities in a "backward" manner, appending to the optimal set the activity that leaves more time available for other activities at the beginning of the time interval. This algorithm is optimal because the intuition behind it is the same as for the original one, in the sense that they both try to maximize the time interval left after selecting an activity (the original algorithm selects the activity that ends first, leaving more time at the end of the interval).
The following is the pseudocode for the modified algorithm, where s is an array containing the starting times of the activities (in monotonically decreasing order) and f is the corresponding array with the finishing times:

```
Greedy-Activity-Selector-2(s, f):
    n = s.length
    A = {a_1}
    k = 1
    for m = 2 to n:
        if f[m] ≤ s[k]:
            A = A ∪ {a_m}
            k = m
        end if
    end for
    return A
```

4. (a) The following figure shows a counterexample for a small problem with four activities: selecting the shortest activity first would yield a_1, which only leaves room for a_4. This is clearly not optimal, since the set $A = \{a_2, a_3, a_4\}$ is also feasible and has more activities.

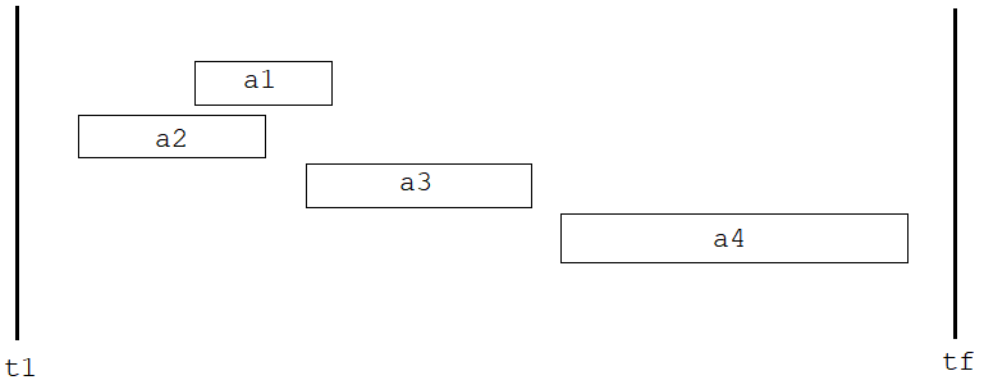


Figure 1: Counterexample for shortest activity selection.

(b)

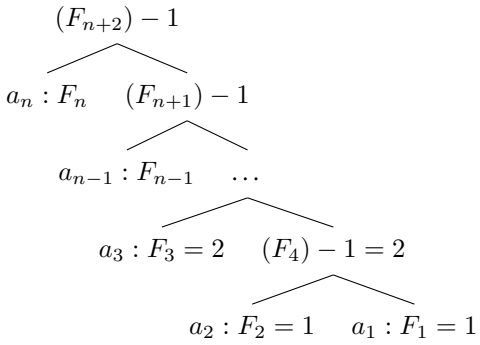
(c) Counterexample: the activity a_i with the earliest starting time has the longest duration of all the remaining activities. In such a case, selecting a_i would not yield an optimal solution because a lot of time would be allocated to only that activity. Since the optimization problem is concerned with the amount of activities selected, it would be better to select more activities with shorter duration times.

5. (a) The following table shows an optimal Huffman code for the eight characters associated to the first eight Fibonacci numbers:

i	a_i	F_i	Encoding
1	a	1	1111111
2	b	1	1111110
3	c	2	111110
4	d	3	11110
5	e	5	1110
6	f	8	110
7	g	13	10
8	h	21	0

(b) The generalization of the previous exercise to n letters with frequencies equal to the first n Fibonacci numbers is as follows: *the first element is encoded as $(n-i)$ ones, and for all other values of i, the i^{th} element is encoded as $(n-i)$ ones plus a trailing zero.*
The following table and tree show this generalization:

i	a_i	F_i	Encoding
1	a_1	1	$\overbrace{1\ldots1}^{\text{n - 1}}$
2	a_2	1	$\overbrace{1\ldots1}^{\text{n - 2}}0$
3	a_3	2	$\overbrace{1\ldots1}^{\text{n - 3}}0$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
$n-1$	a_{n-1}	F_{n-1}	10
n	a_n	F_n	0



6. If we used the "standard" method to implement the Huffman code, instead of using a Min-Heap we would have to keep a sorted array with the frequencies of the characters. The time complexity analysis would be as follows:

(a) The original sorting of the frequencies would take $O(n \log n)$ time.

(b) The two `Extract-Min` steps would take constant time $O(1)$.

(c) The `Insert` step would take linear time $O(n-i)$, since we would have to traverse the array to find the correct position for the new frequency.

Steps (b) and (c) are executed inside a for loop with values of i from 1 to $n-1$, and therefore the overall time complexity becomes:

$$T(n) = O(n \log n) + O\left(\sum_{i=1}^{n-1} (1 + n - i)\right)$$
$$T(n) = O(n \log n) + O\left((n-1) + (n-1)(n) - \frac{(n-1)(n)}{2}\right)$$
$$T(n) = O(n^2)$$