Introduction to Algorithms
Assignment 2

Sourabh S. Shenoy
(UIN:225009050)

September 19, 2016

1. Question 1

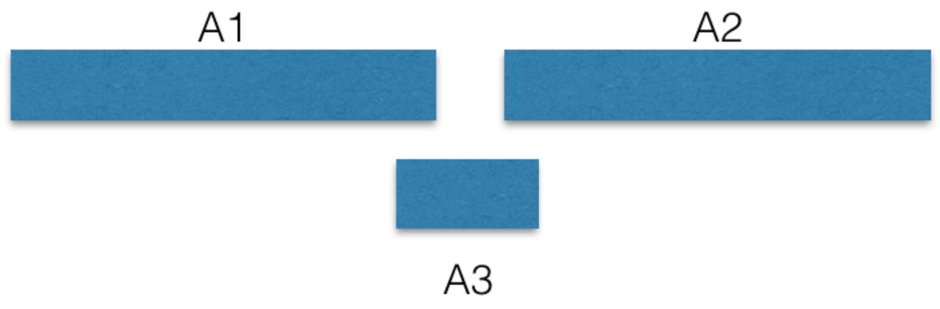
1.1 Greedy Activity Selection

Not just any greedy approach to the activity-selection problem produces a maximum-size set of mutually compatible activities. Give an example to show that the approach of selecting the activity of least duration from among those that are compatible with previously selected activities does not work. Do the same for the approaches of always selecting the compatible activity that overlaps the fewest other remaining activities and always selecting the compatible remaining activity with the earliest start time.

1.2 Solution

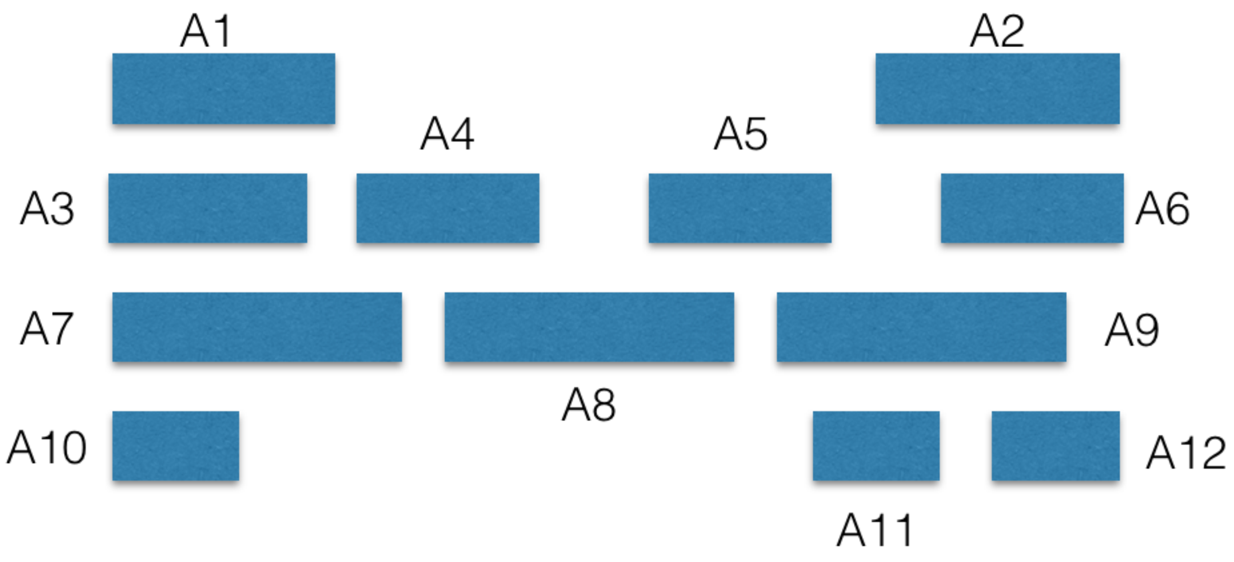
Following is a counterexample which proves that:

a) Selecting the activity of least duration from among those that are compatible with previously selected activities does not work:



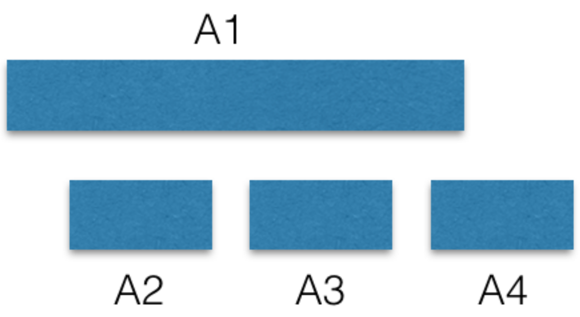
In the above figure, we see that Activity A3 has the least duration and hence, is picked first. However, we can not select any other activity after that. This is not the optimal solution, as we can pick up the activity set {A1,A2} which is the required optimal way of selecting activities.

b) Selecting the compatible activity that overlaps the fewest other remaining activities does not work:



In the above figure, activity A8 has the least overlap with other activities. However, if we select A8, then we can select {A8,A11 and A1/A10/A6/A12/}. None of these 4 possible combinations give the optimal solution. The optimal solution would be selecting the activities A1/A3,A4,A5,A6/A12}.

c) Selecting the activity with earliest start time does not work:



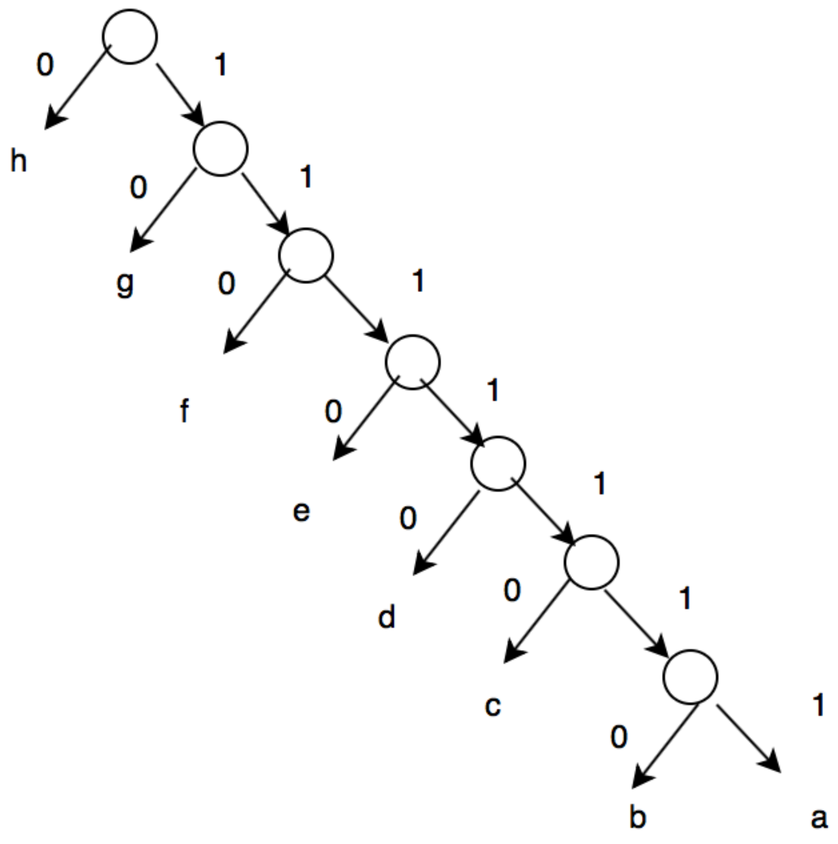
In the above figure, selecting the activity with earliest start time would mean that we only get to select activity A1, where as the optimal way of selecting activities would be to select {A2,A3,A4}

2. Question 2

2.1 Huffman Code

What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers? a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21
Can you generalize your answer to find the optimal code when the frequencies are the first n Fibonacci numbers?

2.2 Solution



The optimal codes assigned are as follows:
a:1111111
b:1111110
c:111110
d:11110
e:1110
f:110
g:10
h:0

In general, for the first *n* fibonacci numbers, we have:
f_1: 0
f_2: 10
f_3: 110
.
.
.
f_{n-1}: $(1..1)_{n-2}0$
f_n: $(1..1)_{n-1}$

3. Question 3

3.1 Coin Changing

Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer:
a. Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.
b. Suppose that the available coins are in the denominations that are powers of c, show that the greedy algorithm always yields an optimal solution.
c. Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of n.
d. Give an O(nk) time algorithm that makes change for any set of k different coin denominations, assuming that one of the coins is a penny.

3.2 Solution

a)

Idea:
In the greedy strategy, we choose the coin which has the largest denomination. In this case, it is 25¢. We then repeat by subtracting the largest denomination possible from the amount to get the lowest possible number of coins.

Pseudocode:

Input: N, the amount for which change has to be given. D={d_1,d_2...d_n}, the set of valid denominations.

Output: K coins, where K is the least number of coins whose value adds up to N.

Pseudocode:

```
    return 0 if N <= 0. Else:      for i = k to 1
        if N >= $d_i$
            increment ct[i] (Counter for the i^th denomination)
            N=N-$d_i$
    repeat
```

Proof of Correctness:

To prove the correctness of the algorithm, we have to first prove that the greedy approach is correct. This implies, that the choice of selecting the largest denomination every time recursively holds true.

We have 2 possibilities. If our optimal solution has the largest denomination, then our algorithm is correct. If our optimal solution does not have the largest denomination 'c', then:

Suppose N is between 1 and 5, then the solution can have only 1s, and hence the approach is correct.

Suppose N is between 5 and 10, then suppose we do not use the greedy approach, we would have at least 5 pennies. These can be replaced with a nickel to get a more optimal solution.

Suppose N is between 10 and 25, then suppose we do not use the greedy approach, we would have only pennies and nickels. Some coins among these would add up to 10. These can be replaced with a dime to get a more optimal solution.

The same explanation holds good for N > 25 as well, where combinations of pennies, dimes and nickels would give a quarter and hence improve the solution to make it more optimal.

Hence, our assumption about the presence of the highest possible solution in the optimal solution should be true and hence greedy approach works well.

Time Complexity:
The algorithm chooses one coin at a time and runs recursively. Hence, the run time is O(k), where k is the no of coins used for the optimal solution. Hence, it takes linear time to run.

b.

For any value of i, we know that the number of coins of denomination c_i in the optimal solution, can never exceed c. Because if it did, then it could be replaced with a single coin of denomination c_{i+1}. In our greedy algorithm, we pick the coin of denomination c_i, such that $c_i<=N<=c_{i+1}$. After subtracting such a denomination from N, we apply the same algorithm recursively to obtain the optimal solution.

To prove the validity of the above statement, consider that there exists no such i which gives the optimal solution. Then, we have to choose k coins such that the highest power of c is not used. In the coins that we have chosen, there would exist values c_a and c_b, such that c_a+c_b would equal to c_i where c_i is the highest power of c <= N. Replacing the coins with a coin of c_i would give the optimal solution. Thus, greedy approach gives the optimal solution.

As an example, consider that c=3 and our denominations are {1,3,9,27..} etc. Suppose we had to choose coins for N=10, our greedy approach would pick 9,1. Suppose we had not picked 9, we could have picked {3,3,3,1}. but then the three 3s can be replaced with a single 9 to form the optimal solution. Thus, greedy approach works best in these cases.

c. Greedy Approach does not work optimally for all denominations. An example is if the denominations of the coins are 1, 6, 15. Suppose we need the optimal change for 20¢, then the greedy approach would pick 1 15¢and 5 pennies, giving us 6 coins. However, the optimal solution would be to pick up 3 6¢coins and 2 pennies, thus using only 5 coins.

d.
Idea: We can solve the coin problem using Dynamic Programming approach. For a given value N, we can find the minimum number of coins needed to get $N\text{-}d_i$ coins for all values of i. We can store all the values in an array and retrieve it whenever needed.

Pseudocode:

```
return 0 if N <= 0. Else:
Create and Initialize array O[N] to 0
for i = 1 to k
    Calculate and store in 'x' the minimum of T[i-$d_j$] for j between 1 and k
        T[i]=1 + x
repeat
```

The optimal solution will be located at array location T[N].

Time Complexity: We fill in N values in the array and for each value, we look for k values to find minimum. Hence, the complexity is O(N*k). Space Complexity is O(N) since we store N values.