CSCE 629-601 Homework 2

Name: Adil Hamid Malla **UIN:** 425008306

1. Exercise 16.1-3 on page 422

Solution:
a) Example to show that the approach of selecting the activity of least duration among those that are compatible with previously selected activities doesn't work.

The duration is given by the formula: $d_i = f_i - s_i$ that is duration is equal to finish time minus start time. Here is the example:

i	1	2	3
s_i	0	3	4
f_i	4	5	7
$d_i = f_i - s_i$	4	2	3

Here lets see what our greedy algorithm gives us as output. Greedy output will be activity set $S = \{a_2\}$
Whereas the optimal solution of this problem is $s = \{a_1, a_3\}$

b) Example to show that the approach of selecting the compatible activity that overlaps the fewest other remaining activities doesn't work using the greedy algorithm.

Here we count the number of overlaps by considering the activity like this a$[s_i, f_i)$

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	2	2	2	7	8	10	11	11	11	13
f_i	3	8	8	8	9	11	12	16	16	16	17
No. of Overlaps	3	4	4	4	4	2	4	4	4	4	3

In this example if we apply the greedy algorithm, by selecting the that activity first whose number of overlaps is less then we end up having the only three activities as follows,

$$A = \{a_6, a_1, a_{11}\}$$

whereas the optimal solution to this problem consists of getting four activities as below,

$$A = \{a_1, a_5, a_7, a_{11}\}$$

Hence proved.

c) Example to show that the approach of always selecting the compatible remaining activity with the earliest start time as a greedy choice does not always give the optimal results.
Lets take an example to prove that above statement

i	1	2	3	4
s_i	1	0	3	8
f_i	8	11	9	11

Here according to the earliest start time as a greedy choice for selecting the activity will only result in one activity to get selected as others overlap with the first selected one: i.e.

$$A = \{a_2\}$$

Whereas, if we see the optimal solution this problem then we will get two activities which are compatible.

$$A = \{a_1, a_4\}$$

Thus we proved this strategy to be wrong as well.

The motive of this exercise is that not all greedy choices can lead to the optimal choice in general, if we are making a greedy choice we must have the global optimal in mind, all these strategies proved to not work or extend to be global optimal

2. Exercise 16.3-3 on page 436

Solution:

a) 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

Here the characters are given where the frequencies follow the Fibonacci numbers. Lets make the input table for the same with corresponding frequencies

character	a	b	c	d	e	f	g	h
freq	1	1	2	3	5	8	13	21

.

Figure 1: Huffman Code Tree for Fibonacci numbers

Here following the greedy algorithm, we can arrange the items in order of the lowest frequencies, after that we can select two lowest frequency characters(here a,b) and make them siblings of a binary tree. After that we will treat the parent node of the two characters as the standalone node and name it node ab(combination of a and b) and the frequency will be equal to sum of frequency of a and b and then we will insert back this standalone node into our list at appropriate location according to the frequency. We will repeat this procedure several times till we are left with only node, that will be our root node of tree. After that we can expand the tree by using the structure we have already saved and traverse the tree, each leaf node will represent one character and the path is defined is the codeword for the same. Moreover, if we go left from a parent node then the label of the path is 0 and if we go right we label it 1.
Following the traversal to all the leaf nodes will will get the path which in turn is the code word for that character.

Symbol	Code	bits
h	0	1
g	10	2
f	110	3
e	1110	4
d	11110	5
c	111110	6
b	1111110	7
a	1111111	7

Here we can see that the character with lowest frequency got the highest cord word. Moreover decoding of this code is very easy and straight forward, as the code generated here is prefix free code.

b) Can you generalize your answer to find the optimal code when the frequencies are the first n Fibonacci numbers?

If we check the characteristic of the above calculated code, we will see that number of the bits required for each character is increasing by one and the last two characters has the same number of bits, this tells us that the tree is skewed to right.
In generalization we can say that if we have n Fibonacci Numbers corresponding to the frequency of the n characters then we will have code words like this
Code words will be: (the last two codewords have same length)

$$\mathit{Code}\,(n) = \left\{0, 10, 110, 1110, \ldots, 1^{n-2}0, 1^{n-1}\right\} \text{ where } 1^n = n \text{ times } 1$$

This generalization is possible only because of the structure of the Fibonacci numbers. We can prove the same using the mathematical induction method.
In Fibonacci series, the seed values are F_0 and F_1 where both are equal to one. and the recurrence formula to get the next number is given by

$$F_n = F_{n-1} + F_{n-2}$$

Now we have a series of the frequencies which are always in increasing order and sum of two lowest frequencies is always greater or equal to the next element but less than rest of all the elements. This leads to a generalization that if the number of the characters are 1, then only one bit is used, if the number is two then two bits are used (one for each character), after that each new character addition will result in two characters with least frequency having same number of bits and rest each higher frequency having one less bit in its codeword. The relation can be given by formula

$$F_{n+2} = \sum_{i=0}^{n} Fi + 1$$

Where F_n represents the value of the nth number. By using the simple induction method we can say that the recurrence can be proven for the numbers 1,1,2,3. Now lets assume that the equality follows to all the number till/ smaller than F_{n+2} We can prove that it hold true for F_{n+2} as

$$F_{n+2} = F_{n+1} + F_n = \sum_{i=0}^{n-1} F_i + 1 + F_n = \sum_{i=0}^{n} F_i + 1$$

Thus we can say that F_{n+2} is chosen only and only after smaller Fibonacci numbers have been merged into a single tree.

2. Problem 16-1 on page 446-447

Solution:

a) Describe a greedy algorithm to make change consisting of quarters, dimes,nickels, and pennies. Prove that your algorithm yields an optimal solution.
The question asks as to develop a greedy strategy for getting the least number of coins to make change for n - cents. Lets make a table for the coin types and their corresponding values.

Coin Type	Coin Symbol	Value in Cents
Quarter	d_4	25
Dime	d_3	10
Nickel	d_2	5
Penny	d_1	1

Now the greedy strategy we can make here is take the coins of highest value first and then get the remainder of the amount(cents) left with the left over denominations which are lesser than those of the coin denomination selected.
The following algorithm will make it easy to understand

Algorithm 1 Minimum Coin Change Problem

```
1: procedure COIN_CHANGE($n$)          ▷ Returns the number of min coins to get the change in cents
2:     let k be number of denominations we have
3:     let $t\_count$ be total coins
4:     let $coin\_count[k]$ be a new array which stores the number of each denomination ( 4 denominations)
5:     for $i = k \to 1$ do             ▷ since we have only 4 denominations
6:         $coin\_count[i] = n/d_i$      ▷ here $d_i$ represents the denomination
7:         $n = n \mod d_i$
8:         $t\_count = t\_count + coin\_count[i]$
9:     end for
10:    return $t\_count$
11: end procedure
```

So each time we make a greedy choice of selecting the highest value of denomination, we are left with a smaller amount of money to get the change for. The total number of operations we do here is O(k). Which is order of constant time.
Example for the same will be if we need change for n = 97 cents, then we will get use quarters first. that is we will use 3 quarters and we are left with 22 cents to get the change for, then we will select the second denomination, i.e dime, for 22 cents we will use 2 dimes, we will be left with 2 cents, since nickel is greater than remaining amount, so we will go directly to pennies, so we will use 2 pennies. Thus making greedy choice at every location we used only 3+2+2 = 7 coins in total. Which is the optimal solution.
To prove that this algorithm (making a greedy choice at every time) is optimal solution for this problem we need to prove two things:
1) That we are making a choice and are left with only one subproblem to solve. This is ensured as every time we make a greedy choice of selecting the highest value denomination, we are left with a subproblem i.e coin change for smaller amount of money.

2) Second thing we need to prove is that the local optimal solution will provide the global optimal solution for this problem. This we can make sure using the following example:
lets say say that we have n cents whose change we want in fewer coins. and let c represent the optimal solution of the same. so by definition c will have following structure

$$C = \{C_q, C_d, C_n, C_p\}$$

Now lets assume that by making the greedy choice we have came up with following solution;

$$C_2 = \{C_{2q}, C_{24}, C_{2n}, C_{2p}\}$$

we say that $C < C_2$, as there is some other solution which is globally optimal as compared to the greedy choice we made.
Now any optimal solution will have these characteristics: (rulebook)
- the number of pennies will be at most 4, because if they are five then we can replace 5 pennies by one nickel.
- the number of nickels in a optimal solution will be at most 1, again if there are two we can replace by one dime.
- and there will be only at most 2 dimes, else we will replace the rest by one quarter.

Now if we follow what greedy algorithm does, is same as what the conditions for optimal solution is. Now according to assumption that $C < C_2$, then there is this condition
- $c_q < c_{2q}$: according to our greedy algorithm c_{2q} represents the number of coins which can be converted in quarters. If the given condition follows i.e. $c_q < c_{2q}$ then we can convert remaining cents into quarters ($c_{2q} - c_q$ * 25 worth of coins, doing this will never increase any other denomination. Now since we were using less number of the quarters in our greedy solution, then the remaining cents would have been distributed among dimes and nickels on our solution. Further as per the optimal solution characteristics quarter can only be represented by as 2 dimes and nickels (according to rule book). Now since we used the 25 cents to make a quarter as number of quarters in our solution we used less then we have saved 3 coins in form of 2 dimes and 1 nickel. So by default we have already reduced three coins in the optimal solution already given to us. Thus this proves the contradiction that greedy algorithm for number of quarters is not right. So $c_q = c_{2q}$ should be equal.
Following the same thought process and contradiction we can prove that $c_d = c_{2d}$, $c_n = c_{2n}$ and $c_p = c_{2p}$. In other words, the optimal solution can be obtained by making the greedy choice. which proves the optimality of the given algorithm.

b) Suppose that the available coins are in the denominations that are powers of c, i.e., the denominations are $c^0, c^1, ..., c^k$ for some integers $c > 1$ and $k >= 1$. Show that the greedy algorithm always yields an optimal solution.

Here lets take an example and then generalize for all the cases.
Let us take c = 5 and k=4, so the denomination we get is following

$$D = \{5^0, 5^1, 5^2, 5^3\}$$

So the usage of coins (rulebook) in an optimal solution is as follows

1. $5^0 = 1$ coin denomination can be used only $\left(5^1 - 5^0\right)/5^0 = 4$ coins of this denomination if it is greater than this number we can use next denomination.
2. $5^1 = 5$ coin denomination can be used only $\left(5^2 - 5^1\right)/5^1 = 4$ coins of this denomination.
3. $5^2 = 25$ coin denomination can be used only $\left(5^3 - 5^2\right)/5^0 = 4$ coins of this denomination.
4. 5^{k-1} =125 coin denomination can be used as many times as required for the problem.

Now we generalize the formula for getting the rule book of using the coins is

$$\text{Coin of Denomination } c^n = \left(c^{n+1} - c^n\right)/c^n \text{ coins, where where } n \geq 0 \text{ and } n \leq k-2$$
$$\text{And coin of denomination } c^{k-1} \text{ can be used any number of times.}$$

So the proof of optimality for this problem is same as that of above question. If we need to get a change for n cents and denomination is given by $c^0, c^1, c^3..., c^k$
lets say say that we have n cents whose change we want in fewer coins. and let c represent the optimal solution of the same. so by definition c will have following structure

$$C = \left\{nc^0, nc^1, nc^2, nc^3, \ldots, nc^{k-1}\right\}$$

Now lets assume that by making the greedy choice we have came up with following solution;

$$C_2 = \left\{nc'^0, nc'^1, nc'^2, nc'^3, \ldots, nc'^{k-1}\right\}$$

we say that $C < C_2$, as there is some other solution which is globally optimal as compared to the greedy choice we made.
Now according to assumption that $C < C_2$, then there is this condition
$nc^{k-1} < nc'^{k-1}$: according to our greedy algorithm nc'^{k-1} represents the number of coins of c^{k-1} denomination. If the given condition follows i.e. $c^{k-1} < c'^{k-1}$ then we can convert $(nc'^{k-1} - nc^{k-1}) * c^{k-1}$ worth of coins, doing this will never increase any other denomination in our optimal solution. Now since we were using less number of the quarters in our greedy solution, then the remaining cents would have been distributed among next lower denominations i.e c^{k-2} on our greedy solution. Further as per the optimal solution characteristics c^{k-1} can be represented by using 5 coins of c^{k-2} denomination (according to rule book). Now since we used the c^{k-1} cents to make a extra denomination of c^{k-1} as number of c^{k-1} in our solution, we used less then we have and thus we saved 5 coins in form of c^{k-1}. So by default we have already reduced 5 coins in the optimal solution already given to us. Thus this proves the contradiction that greedy algorithm for number of c^{k-1} denomination is not right, which is a contradiction as it is optimal solution, So this means that the assumption of $nc^{k-1} < nc'^{k-1}$ is wrong . So nc^{k-1}, nc'^{k-1} should be equal.
Following the same thought process and contradiction we can prove that for all the $c^{k-2}, c^{k-1}, c^{k-3}, ..., 1$ that it should be equal to number of coins in optimal denomination . In other words, the optimal solution can be obtained by making the greedy choice. which proves the optimality of the given algorithm.

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.
The simplest example of the denomination i can think of is if we have to get the change for n = 30 cents and the available denomination is just $D = \{1, 10, 25\}$, then our greedy algorithm will will return the the denomination of 1 coin of quarter (25 cents) and 5 coins of pennies, so making a total of 6 coins to get the change. But the optimal solution for this problem can be 3 dimes (30 cents).
Also, if the denomination is not actual US denomination then i can think of many other denominations like for n= 13 and $D = \{1, 6, 10\}$, here also the greedy algorithm will give 4 coins as answer whereas optimal solution will only have 3 coins.

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.
INPUT: In this case we need to generalize the coin change algorithm for all the set of denominations. Let us represent the denomination of k coins by set $D = \{d_1, d_2, ...d_k\}$ and we have a number n representing the cents we need change for.
OUTPUT: we want to find the number coins of each denomination used as of to decrease the total number of coins in total and it should sum up to the n cents. so output is in form like this.

$$n = \textstyle\sum_{j=1}^{k} d_i c_{d_i} \text{ and also } \sum_{i=1}^{k} c_{d_i} \text{ should be minimal}$$

The above problem can be easily divided into subproblem. lets say if we need to get the minimal change for n cents can be solved by solving the smaller subproblems. Say we have used to solve for Change(n) = 1 (coin of denomination say j) + Change (n - d_j), and we can further solve the problem using the same approach and we will store the subproblems so that we can use it in higher values. So our subproblem structure is something like this

$$s\,[i] = \min_{1 \le j \le k} s\,[i - d_j] + 1$$

The Algorithm 2 gives the details.
So doing this we will get both the number of coins used in each denomination and also will get the minimum number of coins used to get the change for given value of the n.
Time Complexity: Since we are iterating over all the possible values of n and to select the minimum number of coins we iterate over all the possible values of denomination. So time complexity of our algorithm will be $O(nk)$

Algorithm 2 Minimum Coin Change Problem

```
procedure COIN_CHANGE($n$)    ▷ Returns the number of min coins to get the change in cents
    $s[0] = 0$
    $s[1] = 1$    ▷ Since for one cent we will use a penny
    for $p = 2 \to n$ do    ▷ Using bottom up approach to calculate the min coins for each value of cent
        $min\_val = \infty$
        for $i = 1 \to k$ do
            if $p >= d_1$ then
                if $s[p - d_i] + 1 < min\_val$ then
                    $min\_val = s[p - d_i] + 1$
                    $coin[i] + +$
                end if
            end if
        end for
        $s[p] = min\_val$
    end for
    return $s[n]$
end procedure
```