Xiaoying Li
CS 325 -Winter 2020
Homework #4

- **Problem 1**

Here is my example:
Weight of the knapsack W = 55lb
Set of objects S = {(item1, 10lb, $60), (item2, 20lb, $100), (item3, 40lb, $80), (item4, 50lb, $150), (item5, 30lb, $120)}
Set of items' benefits per unit B/W = {(item1, $6), (item2, $5), (item3, $2), (item4, $3), (item5, $4)}
After sorting items based on decreasing benefit/weight ratios B/W = {(item1, $6), (item2, $5), (item5, $4), (item4, $3), (item3, $2)}

a.
In factional knapsack problem, a fraction of any item may be chosen.
Current weight in knapsack = 0; current benefit = 0.
Can item 1 fit? 0+10=10<55, so select it. Current benefit=0+60=60.
Can item 2 fit? 10+20=30<55, so select it. Current benefit=60+100=160.
Can item 5 fit? 30+30=60>55, so no.
We can add 55-30=25 to knapsack, so select 25/30=5/6 of item5.
Current benefit = 160+(5/6)*120=260.

So, the final greedy solution is item1, item2, and 5/6 of item5, and the benefit is $260.

Since the greedy choice keeps taking item with highest value (benefit per unit), so the final benefit is the maximum possible benefit. Therefore, the greedy solution is the optimal solution, and the greedy approach works for fractional knapsack.

b.
In 0-1 knapsack problem, a specific item is either selected or not.
Current weight in knapsack = 0; current benefit = 0.
Can item 1 fit? 0+10=10<55, so select it. Current benefit=0+60=60.
Can item 2 fit? 10+20=30<55, so select it. Current benefit=60+100=160.
Can item 5 fit? 30+30=60>55, so no.
Can item 4 fit? 30+50=80>55, so no.
Can item 3 fit? 30+40=70>55, so no.

So, the final greedy solution is item1 and item2, and the benefit is $160.

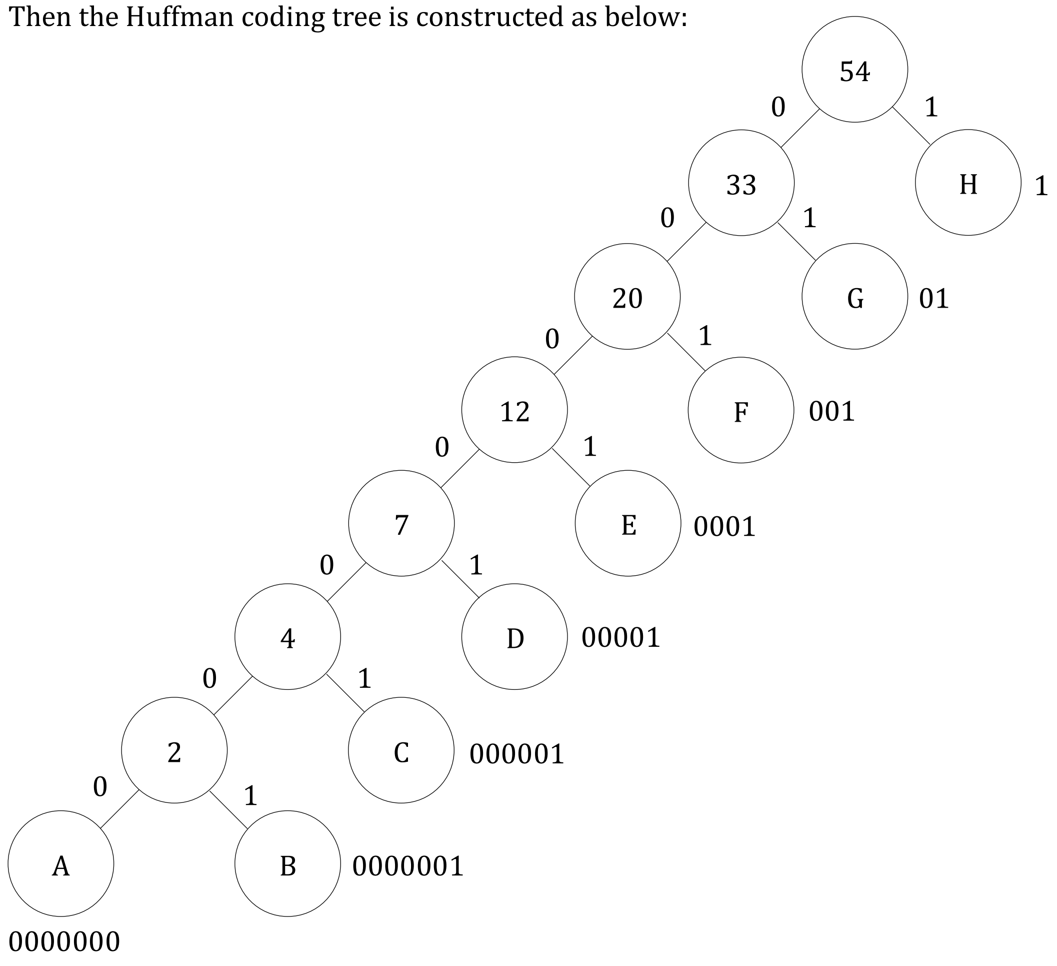
But by applying the bottom-up dynamic programming algorithm to this instance of knapsack problem, we can find that the optimal solution is item2 and item5, and the maximum possible benefit is 100+120=220. Therefore, the greedy approach may fail for 0-1 knapsack.

- **Problem 2**

a.

Step 1: A: 1, B: 1, C: 2, D: 3, E: 5, F: 8, G: 13, H: 21
    smallest numbers are 1 and 1 (A and B).
Step 2: A+B: 2, C: 2, D: 3, E: 5, F: 8, G: 13, H: 21
    smallest numbers are 2 and 2 (A+B and C).
Step 3: A+B+C: 4, D: 3, E: 5, F: 8, G: 13, H: 21
    smallest numbers are 4 and 3 (A+B+C and D).
Step 4: A+B+C+D: 7, E: 5, F: 8, G: 13, H: 21
    smallest numbers are 7 and 5 (A+B+C+D and E).
Step 5: A+B+C+D+E: 12, F: 8, G: 13, H: 21
    smallest numbers are 12 and 8 (A+B+C+D+E and F).
Step 6: A+B+C+D+E+F: 20, G: 13, H: 21
    smallest numbers are 20 and 13 (A+B+C+D+E+F and G).
Step 7: A+B+C+D+E+F+G: 33, H: 21
    smallest numbers are 33 and 21 (A+B+C+D+E+F+G and H).

Then the Huffman coding tree is constructed as below:



b.

Decode the Huffman coding tree from part a:

    A: 0000000

B: 0000001
C: 000001
D: 00001
E: 0001
F: 001
G: 01
H: 1

Thus, the code 00010010000100000000001001 can be decode as:

0001/001/00001/0000000/001/001 → EFDAFF

Therefore, the decoded string is EFDAFF.

- **Problem 3**

a.

Let the solution given by the greedy algorithm approach be $\{x_0, x_1, x_2, \cdots, x_k\}$, where x_i is the number of coins of denomination c^i. We must show that this solution is the optimal solution.

Let n be the amount of change to be made. Then $n = x_0c^0 + x_1c^1 + x_2c^2 + \cdots + x_kc^k$. Because the greedy algorithm approach picks the largest denomination first, so $x_i < c$ for every i<k. Since if $x_i \geq c$, then $x_ic^i = (x_i - c)c^i + c \cdot c^i = (x_i - c)c^i + c^{i+1}$, which contradicts the precondition that the approach picks the largest denomination first.

Then we must show that when the solution has less than c coins for each denomination c^i (except c^k), it's the optimal solution.

Assume that there exists an optimal solution that for some j<k, $x_j \geq c$. Let the optimal total number of coins be y, and $y = x_0 + x_1 + \cdots + x_j + \cdots + x_k$. Now $n = x_0c^0 + x_1c^1 + \cdots + x_jc^j + \cdots + x_kc^k$ and $x_j \geq c$, then,

$$\begin{aligned} n &= x_0c^0 + x_1c^1 + \cdots + \left(x_j - c\right)c^j + c \cdot c^j + \cdots + x_kc^k \\ &= x_0c^0 + x_1c^1 + \cdots + \left(x_j - c\right)c^j + c^{j+1} + \cdots + x_kc^k \\ &= x_0c^0 + x_1c^1 + \cdots + \left(x_j - c\right)c^j + \left(x_{j+1} + 1\right)c^{j+1} + \cdots + x_kc^k \end{aligned}$$

So, $y' = x_0 + x_1 + \cdots + (x_j - c) + (x_{j+1} + 1) + \cdots + x_k = x_0 + x_1 + \cdots + x_j + x_{j+1} + \cdots + x_k + (1 - c)$. And because c>1, 1-c<0, so $y' < x_0 + x_1 + \cdots + x_j + x_{j+1} + \cdots + x_k = y$, which contradicts the assumption that y is the optimal total number of coins. Hence, the assumption is false, and there is no such optimal solution that for some j<k, $x_j \geq c$.

Therefore, we proved that when the solution has less than c coins for each denomination c^i (except c^k), it's the optimal solution, which is exactly the solution given by the greedy algorithm approach.

b.

```
/*****************************************************************************************
** Function of implement of the greedy algorithm to make change for n cents using the
** fewest number of coins. Suppose the available coins are in the denominations that are
** powers of c, and the exponents are from 0 to k.
```

```
*********************************************************************************/
vector<int> makeChange(int c, int k, int n) {
    vector<int> solution;

    // Loop until all cents have been changed.
    while (n > 0) {
        // Denominations are powers of c, and pick the
        // largest denomination first.
        int denomination = pow(c, k);

        // If the value of left cents is no less than this
        // denomination, use this denomination to make change.
        if (n >= denomination) {
            // Add this denomination to the solution.
            solution.push_back(denomination);
            // Calculate the number of this denomination,
            // and add it to the solution.
            int cardinality = n / denomination;
            solution.push_back(cardinality);
            // Calculate the value of left cents.
            n = n - cardinality * denomination;
        }

        // Move to the next denomination.
        k--;
    }

    return solution;
}
```

In the program above, the while loop loops at most n times and the other statements all run in constant time, so the time complexity of my greedy algorithm approach is O(n).

NOTE: Since the instructor make an announcement that “we will accept two types of solutions under two assumptions” and “the time complexity of your algorithm under each option may be different”, so I chose the first one “input denominations are powers of c and the greedy approach of picking the larger denominations first” and the time complexity is not O(nk) in problem 3b.

- **Problem 5**

Given a set of symbols with their corresponding frequencies:

$$S_1: f_1, S_2: f_2, S_3: f_3, \cdots, S_n: f_n$$

Suppose the symbols are already sorted by their corresponding frequencies in ascending order, which means, $f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_n$.

If for any integer $3 \leq i \leq n$, $f_{i-2} + f_{i-1} = f_i$, aka the frequencies are in Fibonacci pattern continues, then we have the following generalizations:

(1) For the Huffman coding tree of these symbols, every none-leaf node's right child is a leaf node, and represents $S_2 \sim S_n$ from left to right, and the left-most leaf node of the tree represents S_1.

Let a none-leaf node's height be j, then its value is $f_1 + f_2 + f_3 + \cdots + f_{j+1}$. Its left child is $f_1 + f_2 + f_3 + \cdots + f_j$, and its right child is S_{j+1}.

(2) For the Huffman coding of these symbols, we can have the following forms:

$S_1: \underbrace{000 \cdots 0}_{n-1\ 0's}$

$S_2: \underbrace{000 \cdots 0}_{n-2\ 0's}1$

$S_3: \underbrace{000 \cdots 0}_{n-3\ 0's}1$

$\vdots$

$S_{n-1}: 01$

$S_n: 1$

NOTE: The generalization above is based on the announcement that "Alphabetical order left to right. For example, for A and B, A becomes the left child and B becomes the right child. Then, C will sit as a right child and so forth. Labeling: left child 0, right child 1".