Name: Mauro Vargas Jr

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 3A (55 points) **Summer 2020, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solution:

- The solutions **should be typed**, we cannot accept hand-written solutions. Here's a short intro to **Latex**.
- In this homework we denote the asymptomatic *Big-O* notation by $\mathcal{O}$ and *Small-O* notation is represented as o.
- We recommend using online Latex editor **Overleaf**. Download the **.tex** file from Canvas and upload it on overleaf to edit.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Canvas if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.

Name: Mauro Vargas Jr

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 3A (55 points) **Summer 2020, CU-Boulder**

Piazza threads for hints and further discussion

Piazza Threads
Question 1
Question 2
Question 3
Question 4

Recommended reading:

Greedy Algorithms: Ch. 16 16.1, 16.2, 16.3; Ch. 2 2.1, 2.2

Name: Mauro Vargas Jr

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

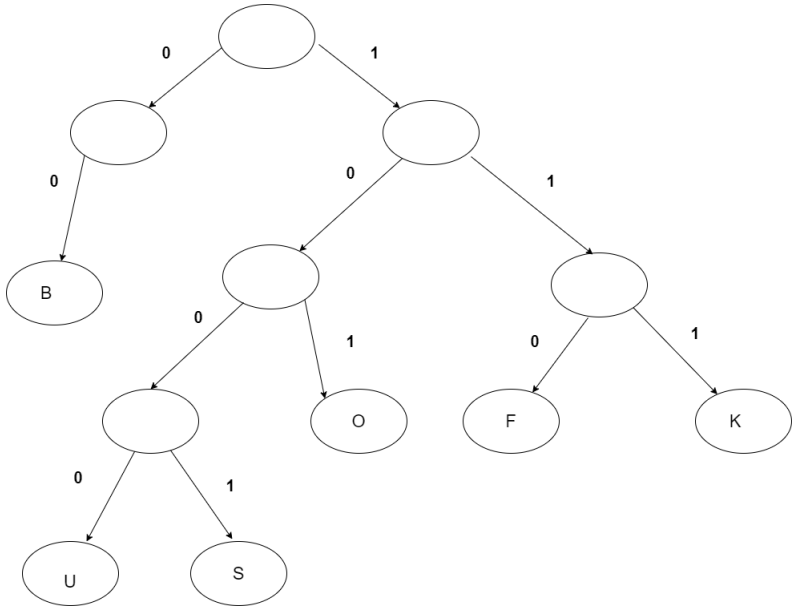
Homework 3A (55 points) **Summer 2020, CU-Boulder**

1. *(5 pts) Assume you run your Huffman tree algorithm and you produce the following pre-fix codes. Describe why there must be an error in your algorithm.*

```
S = 11
c = 10
i = 110
e = 100
n = 010
```

When using Huffman tree algorithm the characters in a message are converted to the **code word** a unique binary string. In order for the binary string to be decoded properly it must have not prefix of another character. Looking at the binary code you can see that **S** is a prefix of **i** and **c** is a prefix of **e**, Therefor this causes an error in the Huffman algorithm.

2. *(5 × 2 = 10 pts) Consider the given Huffman Tree.*



(a) *Decode the string* ***"10011111010010001101101001"*** *encoded using the above Huffman encoding tree.*

S_1001 K_111 O_101 B_00 U_1000 F_110 F_110 S_1001 SKOBUFFS

(b) Decode the string ***"1011110010001001"*** *encoded using the above Huffman encoding tree.*

O_101 K_111 B_00 U_1000 S_1001 OKBUS

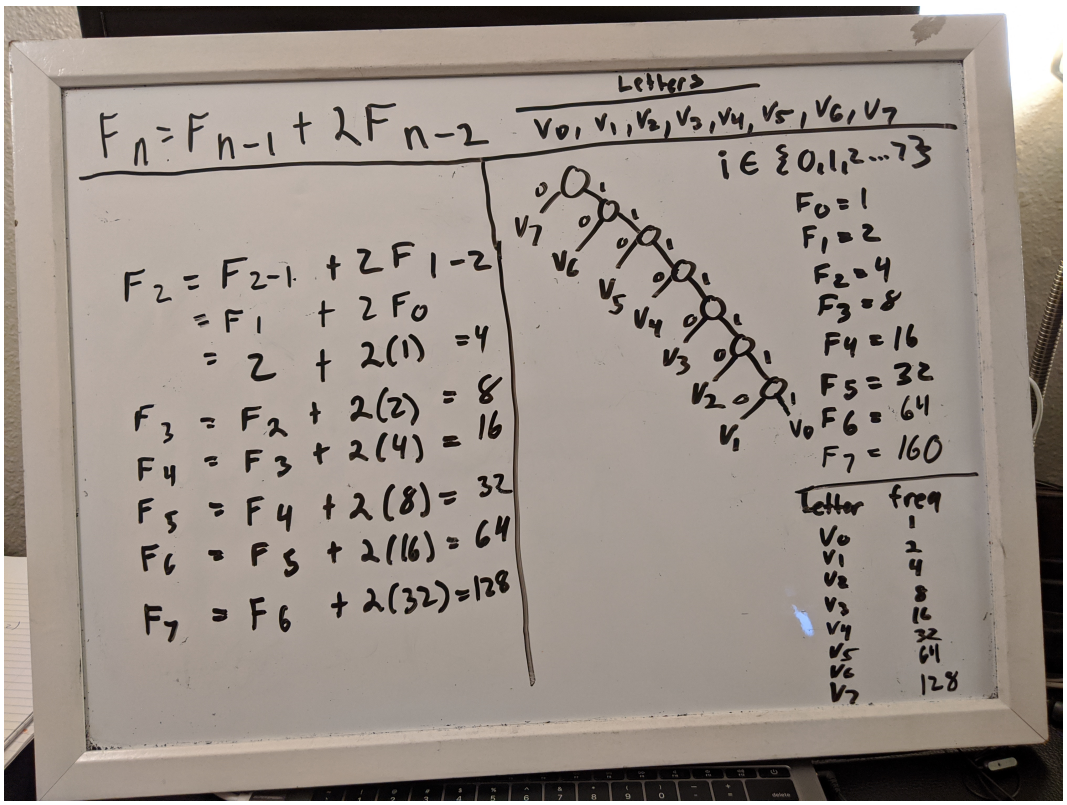
Name: Mauro Vargas Jr

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 3A (55 points) **Summer 2020, CU-Boulder**

3. *(15 pts) Consider the recurrence* $F_n = F_{n-1} + 2F_{n-2}$*, with the base cases* $F_0 = 1$ *and* $F_1 = 2$*. Suppose we have letters* $v_0, \ldots, v_7$*; the frequency of* v_i *is given by* F_i*, where* $i \in \{0, 1, 2...6, 7\}$*. Draw a Huffman tree for* $v_0, \ldots, v_7$*.*



4. *(25 pts) In this question we consider the changemaking problem, of making change for n cents using the smallest number of coins. Suppose we have coins with denominations of $v_1 > v_2 > ...v_r$ for r coins types, where each coin's value v_i is a positive integer. Your goal is to determine how many coins of each denomination d_i are required to reach the sum n, such that the total number of coins used is minimized (i.e the value of $\sum_{i=1}^{r} d_i = k$ is minimized).*

   Note that the sum of all included coins should be n. (i.e $\sum_{i=1}^{r} d_i v_i = n$)

   (a) *(15 pts) A greedy algorithm for making change is the **cashier's algorithm**. In cashier's algorithm at each iteration we add a coin of highest value ensuring that it does not take us past the value n. The above step is repeated until the sum reaches n. If it is not possible to reach the sum n the algorithm returns no solution as the answer. Consider the following pseudocode meant to implement the cashier's algorithm where n is the amount of money to make change for and v is a vector of the coin denominations sorted in descending order and the vector d should hold the count of number of coins in each denomination.*

```
get_change(n, v, r):
{
    d[1...r] = 1
    while(n>0)
    {
        k=r
        while(k>0 and v[k]>n)
        {
            k++
        }
        if(k <= 0)
        {
            return 'no solution'
        }
        else
        {
            n = n-v[k]
        }
    }
    return d
}
```

Name: Mauro Vargas Jr
ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 3A (55 points) **Summer 2020, CU-Boulder**

This code has bugs. Identify the bugs and explain why each would cause the algorithm to fail.

1st Bug: lines 3 turn array **d** into an integer, this array should be filled with zero to keep the count of the denotation of coin types that were used.
2nd Bug: is on line 6 k = r should be k = 1 since our array **v** is in descending order the highest denomination will be at the front of the array when k = 1. We want the greedy algorithm to to subtract the highest denomination if it does not go over **n**.
3rd Bug: is on line 7 inside the while loop condition "k ¿ 0" should be "k ¡ r", this will tell us when the program has iterated through all the denominations and there is no solution because.
4th Bug: is on line 11 "$k <= 0$" should be "$k >= r$" to tell us that there is no solution for our **n** amount of cents that can be satisfied by our denomination of coin types.
5th Bug: is caused on line 20 **d** array is suppose to hold the denomination count in the array, but no where in our algorithm do we increment our count. This should be done after line 17 "$d[k] = d[k] + 1$".

(b) (10 *pts*) *Assume that a country has the following coin denominations.* $\{\$1, \$6, \$10, \$15\}$. *Provide two examples for which the greedy algorithm does not yield an optimal solution for making change. In both the examples also show the optimal solution adds that adds up to the same value using smaller set of coins.*

Example 1: n = 50
After the first three iteration **n** will equal 5 "50 - 3(15) = 5" and then we will need to iterate 5 more times to get n = 0 "5 - 5(1) = 0" total count will be 7. If we only use the denomination of **10 dollars** we will have a count of **5**. Showing that it is not an optimal solution.
Example 2: n = 12
After the first 1 iteration **n** will be equal to 2 "12-10=2" and then we need to iterate two more times to make **n**= 0 "2 - 2 = 0" are count will be 3. If we use the denomination of only **6 dollars** then we only need a count of two. Showing that it is not an optimal solution.

5. ***Extra Credit (5% of total homework grade)*** *For this extra credit question, please refer the leetcode link provided below or click here. Multiple solutions exist to this question ranging from brute force to the most optimal one. Points will be provided based on Time and Space Complexities relative to that of the most optimal solution.*

   Please provide your solution with proper comments which carries points as well.

   https://leetcode.com/problems/jump-game/

   1. Iterate from right to left
   2. Check for a potential jump that reaches a good index
   3. The program will iteration through deincrement i by 1 from the back of the array
   4. If we reach a good index then our position is itself is a good
   5. Once i in our for loop is equal to 0 it will pass r = 0 and the bool will return statement which will be true because 0 == 0 is a true statement, if we iterate through our array and the last i is any integer ¿ 0 the statement will be false and we will know that we got a bad position and return false for that list.

```
class Solution {
public:
bool canJump(vector<int>& nums) {
int r = nums.size() - 1;
for(int i = nums.size() - 1; i >= 0; i--){
    if(i + nums[i] >= r){
        r = i;
    }
}
return r == 0;
}
};
```