# CS 260 – Homework #3

**Chris Kasper**

**January 26, 2018**

## Problem 1

Include with this portion of your homework a copy of your two Fibonacci programs, including your memoisation function, from the implementation section. Analyze the time complexity (big-Oh notation) of your memoisation function and also of the two Fibonacci programs. You don't need to include output from running them, just the python code itself. Using this information, explain why the two (or three) programs take different amounts of time to run.

## Solution

```
def fib(num):
    if num == 0: O(1)
        return 1 O(1)
    elif num == 1: O(1)
        return 1 O(1)
    else:
        return fib(num-1) + fib(num-2) O(2^n)
```

T(n) is $O(2^n)$ for fib(num)

```
def fib_memo(num):
    if num in memo: O(1)
        return memo[num] O(n)
    elif num == 0: O(1)
        return 1 O(1)
    elif num == 1: O(1)
        return 1 O(1)
    else:
```

value = fib_memo(num-1) + fib_memo(num-2)
^ T(n-1) ^ will get computed from memoisation (c)
memo[num] = value O(1)
return value O(1)

T(n)= T(n-1) + c ; c is the time needed to compute n numbers, therefore…
T(n) is $O(n^2)$ for fib_memo(num)

The reason the two programs run at different speeds is because fib_memo has the benefit of memoization. This allows for the function to not have to do any duplicate work that the regular fib() would have to do.

# Problem 2

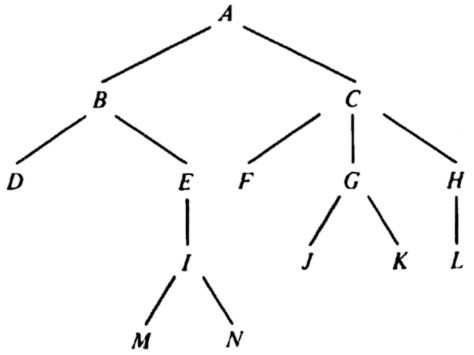
What if you had to write a memoisation function that could not count on a bounded input size. (In the implementation, recall, we told you that you would never see inputs larger than 100.) What would change? What would its running time be then?

## Solution

If the memoisation function couldn't count on a bounded input size, this would decrease the speed of the program by a lot. This is because the function would have to start appending the list, rather than modify it. Modifying a list only takes O(1), while appending one would be O(n). Accounting for this in the new run time, it would then $O(n^3)$.

## Problem 3 (3.1 from text)

Answer the following questions about the tree of Fig. 3.31.




a. Which nodes are leaves?

b. Which node is the root?

c. What is the parent of node *C*?

d. Which nodes are children of *C*?

e. Which nodes are ancestors of *E*?

f. Which nodes are descendants of *E*?

g. What are the right siblings of *D* and *E*?

h. Which nodes are to the left and to the right of *G*?

i. What is the depth of node *C*?

j. What is the height of node *C*

## Solution

a. D, M, N, J, K, L

b. A

c. A

d. F, G, H

e. B, A

f. I, M, N

g. None

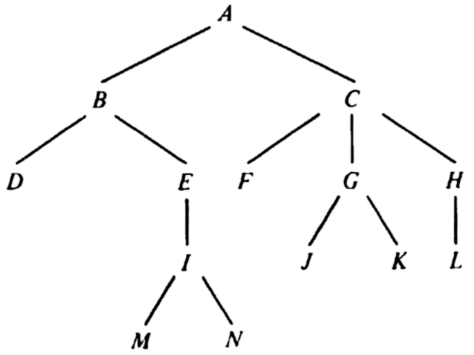
h. J (left), K (right)

i. 1

j. 2

## Problem 4 (3.2 from text)

In the tree of Fig. 3.31 how many different paths of length three are there?




## Solution

1. A, B, E, I
2. B, E, I, M
3. B, E, I, N
4. A, C, G, J
5. A, C, G, K
6. A, C, H, L

Total number of paths of length three are 6.

## Problem 5 (3.6 from text)

Place a check in row $i$ and column $j$ if the two conditions represented by row $i$ and column $j$ can occur simultaneously.

| | *preorder(n)* < *preorder(m)* | *inorder(n)* < *inorder(m)* | *postorder(n)* < *postorder(m)* |
|---|---|---|---|
| *n* is to the left of *m* | | | |
| *n* is to the right of *m* | | | |
| *n* is a proper ancestor of *m* | | | |
| *n* is a proper descendant of *m* | | | |

### Solution

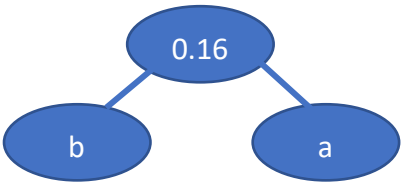
| | *preorder(n)* < *preorder(m)* | *inorder(n)* < *inorder(m)* | *postorder(n)* < *postorder(m)* |
|---|---|---|---|
| *n* is to the left of *m* | ✓ | ✓ | ✓ |
| *n* is to the right of *m* | | | |
| *n* is a proper ancestor of *m* | ✓ | ✓ | |
| *n* is a proper descendant of *m* | | ✓ | ✓ |

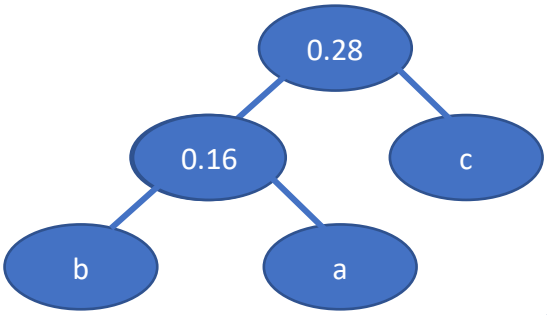
## Problem 6 (3.20 from text)

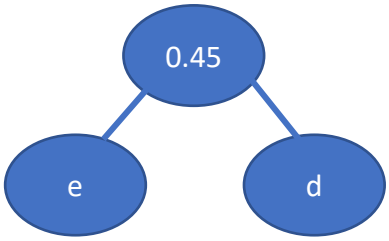
Suppose characters $a$, $b$, $c$, $d$, $e$, $f$ have probabilities .07, .09, .12, .22, .23, .27, respectively. Find an optimal Huffman code and draw the Huffman tree. What is the average code length?

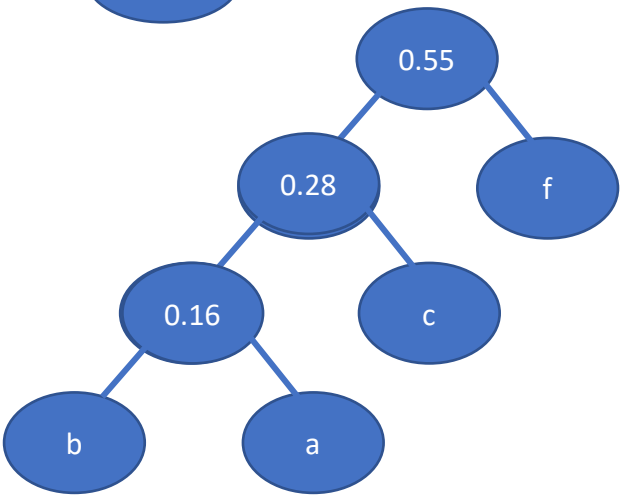
### Solution

| Characters (c) | P(c) | Code | Length(code) |
|---|---|---|---|
| a | 0.07 | 0001 | 4 |
| b | 0.09 | 0000 | 4 |
| c | 0.12 | 001 | 3 |
| d | 0.22 | 11 | 2 |
| e | 0.23 | 10 | 2 |
| f | 0.27 | 01 | 2 |

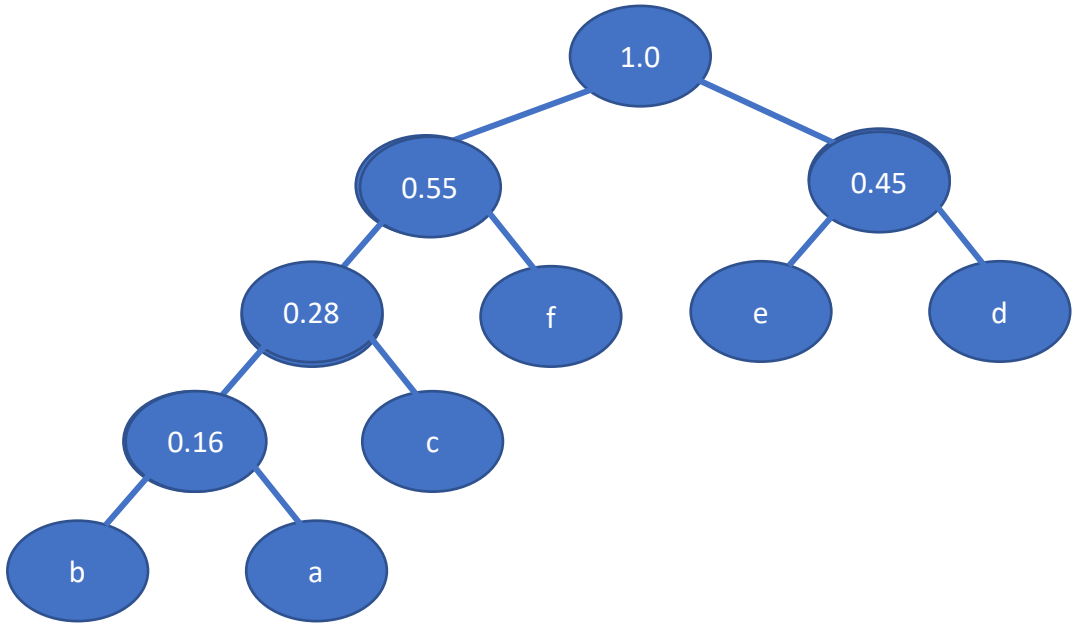


Huffman Tree:



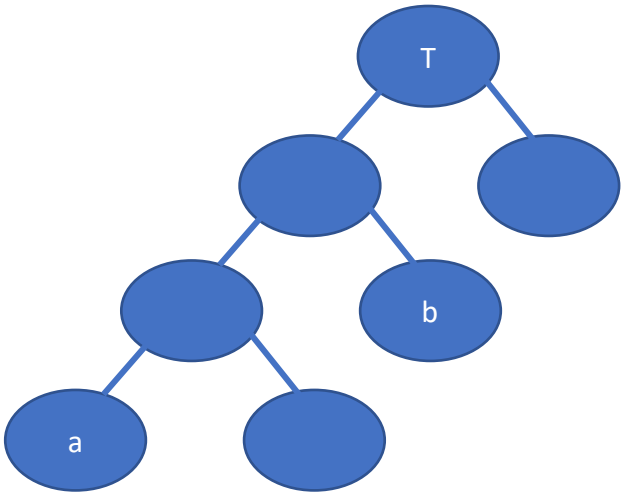

Average code length = 2.83 ≈ 3

## Problem 7 (3.21 from text)

Suppose $T$ is a Huffman tree, and that the leaf for symbol $a$ has greater depth than the leaf for symbol $b$. Prove that the probability of symbol $b$ is no less than that of $a$.

### Solution

If depth(a) > depth(b), prove that P(b) is no less than P(a)…




Visually, it can be seen that number of edges need to travel to reach leaf b is less than that of leaf a from the root, T. Because depth(a) > depth(b), it can be assumed that depth(a) is at least depth(b)+1. Therefore, P(b) >= P(a) because leaf a will have more edges in its path then leaf b.