HW3 Theory

Alexander Kazantsev

February 9, 2016

Problem 1

Fib-Normal

```
def fib(n):
    if n < 2:
        return 1
    return fib(n-1) + fib(n-2)
```

Fib-Memo

```
memoSize = 100
memo = [None] * memoSize
def FibMemo(n):
    global memo
    memo = [None] * memoSize
    return FibMemoAlgo(n)
def FibMemoAlgo(n):
    if memo[n] != None:
        return memo[n]
    if n <= 1:
        return 1
    val = FibMemoAlgo(n-1) + FibMemoAlgo(n-2)
    memo[n] = val
    return val
```

The time complexity of the normal Fibonacci algorithm is O(2^n). This is because it forms a binary tree, and at minimum will have a height of $n-1$ from the root of the tree, therefore it will have on average 2^n nodes.

The time complexity of the memoization Fibonacci algorithm is O(n), or linear. The reason it is linear and more efficient than the standard Fibonacci algorithm is because it doesn't have to visit each node in the tree. The Fiboncci algorithm visits nodes in postfix form (Left Right Current), which means the most left branch will be generated "first". One branch is usually generated in linear time. Once the left most branch is generated, all of the Fibonacci values are available from $[0, n-1]$ and must only be added up with each left most nodes right sibling, which again is linear time. This is a significantly faster algorithm, even without permanent storage.

Problem 2

If the size of the memo list was unbounded, the time complexity would still be linear. If memo list is of current size m and an input of n was given where $n > m$, also assuming n is os significant size larger, then the memo list would have to grow by a size of $n - m$ once. Since $n - m \in \mathrm{O}(n)$ the operation is linear, and since it is performed only once the rules of additivity can be envoked. The algorithm is still linear.

Problem 3.1

Part a

All the nodes are leaves

Part b

A is the root

Part c

A is the parent node of C

Part d

F, G, and H are children of C

Part e

B and A are ancestors of E

Part f

I, M, and N are decendants of E

Part g

E is D's right sibling, but E has no right sibling

Part h

J and K

Part i

C has a depth of 2

Part j

C has a height of 2

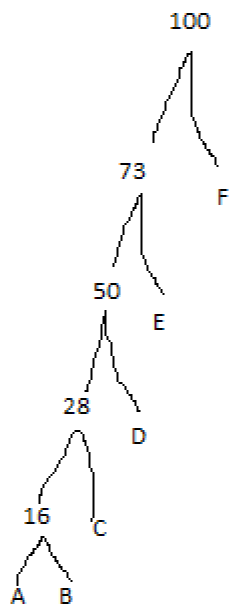
Problem 3.2

6 paths with a length of 3

Problem 3.6

	pre(n) < pre(m)	in(n) < in(m)	post(n) < post(m)
n left of m		✓	✓
n right of m		✓	✓
n ancestor m	✓	✓	
n decendant m		✓	✓

Problem 3.20



1 is left 0 is right

A: 11111 B: 11110 C: 1110 D: 110 E: 10 F: 0

Average bits = .27 + 2*.23 + 3*.22 + 4* .12 + 5* .09 + 6* .07 = 2.74 bits/letter

Problem 3.21

Math and stuff like that... It's 3 AM and I'm tired, and this is a star problem