Problem Set 4

CSCI 3104 Spring 2014
Cristobal Salazar
07/22
Partner: Alex Tsankov

Problem 1

a) We are assuming the edges on our constructed Huffman binary tree are interpreted as 0 for left, and 1 for right. We also construct the tree from the bottom up, with letters of higher frequency to the right. With these rules in mind our codes for each letter are as follows:

a : 0000000	b : 0000001	c : 000001	d : 00001	e : 0001	f : 001	g : 01	h : 1

b) In the first three elements of the list, we have $a:1, b:1, c:2$. This means that adding the $a:1$ and $b:1$ we will get $ab:2$ which is equal to $c:2$. Because these are equal, we can switch them around and still get the same amount of compression. There are $2^2 = 4$ combinations of Optimal Huffman Codes depending on how we tie-break. Then we can do the same procedure, but mirror the tree. This gives us 1 more set of optimal codes, then we can do the same permutation on $a:1, b:1$, and $c:2$ on the mirrored tree to get 4 more sets of optimal code, and including our own set of codes, we get 9 total sets of the optimal codes.

c) The optimal Huffman code for an array of n Fibonacci numbers is: The base case code for a, which we will call k, is $n-1$ "0"s, for b,$k+1$, it is $n-2$ "0"s, followed by a 1. As $k \to \infty$, we can simply do a left bit-shift on $k+1$ equalling difference with $k+1$, while dropping any extraneous zeros on the end.

Problem 2

a)

```
final = [] \\number of coins in array ordered by val (25,10,5,1)
coins = [25,10,5,1]

int count(val,sum,k){
    int i = sum/val
    sum = sum % val
    if (k <= 2)
        final[k+1] = count(coins[k+1],sum,k+1)
    return i
}

final[0] = count(coins[0],sum,0)
```

b)We can show that this is optimal becuase of the greedy choice property. This means the algorithm will uses the largest denomination of coin possible, before using the next largest, then the next, until we have made change. If $2^j = 2 * 2^i$, then the algorithm will use one 2^j coins, rather than two 2^i coins. This is demonstrated below:

$$n = \frac{n}{25} = d_1 = \text{amount of quarters}$$
$$n_1 = n - d_1 * 25$$
$$n_2 = n_1 - (10 * d_2)$$
$$n_3 = n_2 - (5 * d_3)$$
$$n_4 = d_4$$

c) If we have the following set $\{1, 10, 25\} \in C$ our greedy algorithm will not yield the optimal solution because all denominations will require at most $k\%10$ pennies for every k in the one's position of a number For example, a worst case scenario of making 9 cents will require 9 coins, as opposed to 5 with the normal system. As a rule of thumb, we can always use second smallest coin denomination, C_2 in our set as our $k\%C_2$ for our maximum worst case penny scenario

Problem 3

This algorithm needs two sorted arrays, and a counter that will be used to find the median. The alogorithm works in $O(logn)$ by constantly checking to see if its conditions are met, and then recursing on a reduced problem size.

```
//There are n elements in each list
int findMedian(A,B,k){
    if(((k < n) && (B[n-k] <= A[k] <= B[n-k+1])) || (A[k] <= B[n-k+1]))
         return A[k]
    else if(((k < n) && (A[n-k] <= B[k] <= A[n-k+1])) || (B[k] <= A[n-k+1]))
         return B[k]
    else{
         if(A[k] < B[n-k])
              return findMedian(A,B,k+1)
         if(A[k] > B[n-k+1])
              return findMedian(A,B,k-1)
         if(B[k] < A[n-k])
              return findMedian(A,B,k+1)
         if(B[k] > A[n-k+1])
              return findMedian(A,B,k-1)
    }
}

A = []  //both A and B have the same length
B = []
findMedian(A,B,(A.length/2))
```