

Spencer Shimko  
CMSC 441  
HW 7

1. 16.2-4.

Professor Midas drives an automobile from Newark to Reno along Interstate 80. His car's gas tank, when full, holds enough gas to travel  $n$  miles, and his map gives the distances between gas stations on his route. The professor wishes to make as few gas stops as possible along the way. Give an efficient method by which Professor Midas can determine at which gas stations he should stop, and prove that your strategy yields an optimal solution.

The professor knows the distance the automobile can travel on a full tank of gas,  $n$ . The professor starts with a full tank and drives until he can not make it to the gas station **past** the next without running out of gas. He then must stop at the next station.

Number the stations from  $0..m$ . Let  $i$  be the next gas station on the route and  $i + 1$  be the gas station after the next. He fills his gas tank at station  $i$  only when he can not make it to station  $i + 1$  without more gas.

Using this algorithm we obtain a list of gas stations at which the professor will fill up his tank,  $s_0, s_1, s_2 \dots s_k$ . Because this algorithm is greedy we claim that  $k$  is a minimum.

Proof by contradiction: assume that  $k$  is not a minimum. Then create another list of stations  $t_0, t_1, t_2 \dots t_l$  so that

$l < k$ . This must be a greedy list so that  $s_0$  can't come before  $t_0$ ,  $s_1$  can't come before  $t_1$ , and so on. Eventually we reach the case that  $s_l$  can't come before  $t_l$ . But if this is the case then station  $s_{l+1}$  can't be reached. The professor will run out of gas before reaching  $s_{l+1}$  because  $s_{l+1}$  exists after  $s_l$ . This is a contradiction and  $k$  must be a minimum.

## 2. 16.3-3.

Prove that the total cost of a tree for a code can also be computed as the sum, over all the internal nodes, of the combined frequencies of the two children of the node.

Let tree be a full binary tree with  $n$  leaves. Apply induction hypothesis on the number of leaves in  $T$ . When  $n=2$  (the case  $n=1$  is trivially true), there are two leaves  $x$  and  $y$  (say) with the same parent  $z$ , then the cost of  $T$  is

$$\begin{aligned} B(T) &= f(x)d_T(x) + f(y)d_T(y) \\ &= f[x] + f[y] && \text{since } d_T(x) = d_T(y) = 1 \\ &= f[\text{child1 of } z] + f[\text{child2 of } z]. \end{aligned}$$

Thus, the statement of theorem is true. Now suppose  $n > 2$  and also suppose that theorem is true for trees on  $n-1$  leaves.

Let  $c_1$  and  $c_2$  are two sibling leaves in  $T$  such that they have the same parent  $p$ . Letting  $T'$  be the tree obtained by deleting  $c_1$  and  $c_2$ , we know by induction that

$$\begin{aligned} B(T) &= \sum_{\text{leaves } l' \text{ in } T'} f[l']d_T(l') \\ &= \sum_{\text{internal nodes } i' \text{ in } T'} f[\text{child1 of } i'] + f[\text{child2 of } i'] \end{aligned}$$

Using this information, calculates the cost of  $T$ .

$$\begin{aligned} B(T) &= \sum_{\text{leaves } l \text{ in } T} f[l]d_T(l) \\ &= \sum_{l \neq c_1, c_2} f[l]d_T(l) + f[c_1]d_T(c_1) + f[c_2]d_T(c_2) \\ &= \sum_{\text{leaves } l' \text{ in } T'} f[l']d_{T'}(l') + f[c_1] + f[c_2] \\ &= \sum_{\text{internal nodes } i' \text{ in } T'} f[\text{child1 of } i'] + f \end{aligned}$$

$$\begin{aligned}
& [child2 \text{ of } i] + f[c1] + f[c2] \\
& = \sum_{\text{internal nodes } i \text{ in } T} f[child1 \text{ of } i] + f \\
& [child1 \text{ of } i]
\end{aligned}$$

Thus the statement is true. And this completes the proof.

The question is whether Huffman's algorithm can be generalized to handle ternary codewords, that is codewords using the symbols 0,1 and 2. Restate the question, whether or not some generalized version of Huffman's algorithm yields optimal ternary codes? Basically, the algorithm is similar to the binary code example given in the CLR-text book. That is, pick up three nodes (not two) which have the least frequency and form a new node with frequency equal to the summation of these three frequencies. Then repeat the procedure. However, when the number of nodes is an even number, a full ternary tree is not possible. So take care of this by inserting a null node with zero frequency.

Correctness

Proof is immediate from the greedy choice property and an optimal substructure property. In other words, the proof is similar to the correctness proof of Huffman's algorithm in the CLR.

### 3. 16-1.

Consider the problem of making change for  $n$  cents using the fewest number of coins. Assume that each coin's value is an integer.

a. Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution - an optimal solution having the fewest number of coins.

Start with the largest coin denomination (a quarter). Divide the total  $n$  by 25. Then take the remainder and divide it by 10 yielding the number of dimes. Then take the remainder and divide it by 5 yielding the number of nickels. Again for pennies. Numerically this looks like:

$$n / 25 = \# \text{ of quarters}$$

$$n \bmod 25 = x$$

$$x / 10 = \# \text{ of dimes}$$

$$x \bmod 10 = y$$

$$y / 5 \dots\dots$$

Once again we prove this greedy algorithm is optimal by contradiction. We note that the greedy solution can have at most 4 pennies, 1 nickel, and 2 dimes. But 2 dimes and a nickel is a quarter so this combination is not allowed. Pick a non-greedy solution for contradiction. Note that the solution **must be optimal**. We will call this non-greedy solution  $S$ .  $S$  can't have more than 4 pennies and be optimal. Nor can it have 2 nickels or even 3 dimes and still be optimal (less coinage is better in this case). If  $S$  has  $\geq 50$  cents in dimes then 5 dimes can be swapped out

for 2 quarters (sub-optimal until dimes are swapped for quarters). Using the algorithm outlined above we always obtain a greedy solution that is more optimal than  $S$ . This is the contradiction.

**b. Suppose that the available coins are in the denominations that are powers of  $c$ , i.e., the denominations are  $c^0, c^1, c^2, \dots, c^k$  for some integers  $c > 1$  and  $k \geq 1$ . Show that the greedy algorithm always yields an optimal solution.**

Once again proof by contradiction: Pick a non-greedy solution for contradiction. Note that the solution **must be optimal**. We will call this non-greedy solution  $S$ . The greedy solution yields  $c$  numbers of  $c^j$  ( $j < k$ ). Since  $S$  is non-greedy there must exist at least one "coin"  $c^j$  that is in the solution more than  $c$  times. This "coin" can be replaced by a single  $c^{(j+1)}$  and the rest remain  $c^j$ . This proves that a non-greedy solution is not optimal and thus proves by contradiction that a greedy solution is optimal.

**c. Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of  $n$ .**

Given the solution set  $\{1, 5, 10, 20, 25\}$ , it is impossible to create an optimal solution using the greedy method for 40 cents.

**d. Give an  $O(nk)$ -time algorithm that makes change for any**

**set of  $k$  different coin denominations, assuming that one of the coins is a penny.**

As proved in "c" above, greedy can't be optimal for any possible  $k$  denominations. However we can base a dynamic solution on what we developed in part "a". Instead of modulus arithmetic we will use simple subtraction to achieve similar results. Suppose the optimal solution contains coin of denomination  $d$ . Then we have  $n - d$  we must find the optimal solution for:

We create an array in which each element  $A[i]$  tells us how to optimally create  $i$  cents. We do this recursively and soon have an array that tells us how to optimally create  $n$  cents in change.

$A[i] = \min ( 1 , A[i - j] ); j$  is any possible denomination.  
This can be done in  $O(nk)$ -time.