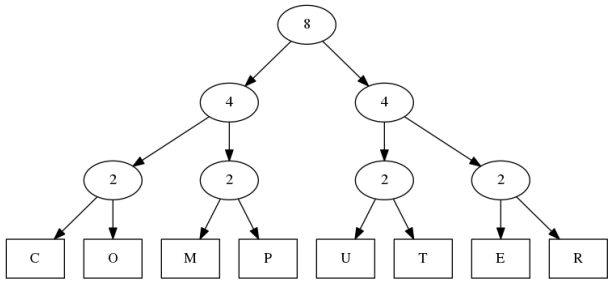
CS 600 Advanced Algorithms

Homework 5 Solutions

1 R-10.6 Example Huffman tree that is a complete binary tree

Lemma 10.4 states that the two lowest-frequency characters have maximum depth in a Huffman tree. Using this, we can conclude that any set of 8 characters where each character has exactly the same frequency produces a Huffman tree that is a complete binary tree. An example for the character set $\{C, O, M, P, U, T, E, R\}$ is shown below.



2 C-10.5 Greedy algorithm for making change

We have a set of coins $C = \{25, 10, 5, 1\}$ and we have to make change for a specific value, A, using the minimum number of coins. We start by picking the largest value, $c_{max} \in C$. If $A - c_{max}$ is positive, we record the choice c_{max} and recurse with $A \leftarrow A - c_{max}$ and the set of coins unchanged. If $A - c_{max}$ is negative, we don't record anything and then recurse with A unchanged and $C \leftarrow C \setminus \{c_{max}\}$. We are done when $A = 0$. Essentially, we list out as many quarters as possible, then as many dimes as possible, then as many nickles as possible and so on.

Algorithm MAKE-CHANGE(A, C, r)

Input: An amount A, a set of coins C, and the current list of coins chosen, r.
Output: A list of coins, r, that make change for A.

if $A = 0$ **then**
    return r
else if $A - c_{max} > 0$ **then**
    add c_{max} to r
    MAKE-CHANGE$(A - c_{max}, C, r)$
else
    $C \leftarrow C \setminus \{c_{max}\}$
    MAKE-CHANGE(A, C, r)

---

3 A-10.1 Art Gallery Guarding

We start by noting that one guard can protect a distance of length 2 in total. The greedy algorithm starts with x_0 and covers all the points within distance 2 of x_0 using a single guard. If x_i is the next uncovered point, then we repeat this same covering step starting from x_i using another guard. We repeat this process until we have covered all the points in X.

Algorithm GUARD-POSITIONS(X)

Input: A set $X = \{x_0, x_1, \ldots, x_{n-1}\}$ of positions
Output: A set G describing the placement of guards

$G \leftarrow \{\}$
for $x \in X$ **do**
  if x is not guarded **then**
    $G \leftarrow G \cup \{x\}$
return G

4 R-11.1 Characterizing Recurrence Equations

4.1 $T(n) = 2T(n/2) + \log n$

In this case we have $n^{\log_b a} = n^{\log_2 2} = n$. Therefore, we are in case 1 of the Master Theorem where $f(n) = \log n \in O(n^{1-\epsilon})$ for a small constant $\epsilon > 0$. We can choose $\epsilon = 0.5$ and we have $T(n) \in \Theta(n)$.

4.2 $T(n) = 8T(n/2) + n^2$

In this case we have $n^{\log_b a} = n^{\log_2 8} = n^3$. Therefore, we are in case 1 of the Master Theorem where $f(n) = n^2 \in O(n^{3-\epsilon})$ for $\epsilon > 0$. We can choose $\epsilon = 1$ and we have $T(n) \in \Theta(n^3)$.

4.3 $T(n) = 16T(n/2) + (n \log n)^4$

In this case we have $n^{\log_b a} = n^{\log_2 16} = n^4$. Thus we are in case 2 of the Master Theorem where $f(n) = (n \log n)^4 \in \Theta(n^4 \log^k n)$ for $k = 4$. By the theorem, we have $T(n) \in \Theta(n^4 \log^5 n)$.

4.4 $T(n) = 7T(n/3) + n$

In this case we have $n^{\log_b a} = n^{\log_3 7}$. Thus we are in case 1 of the Master Theorem where $f(n) = n \in O(n^{\log_3 7 - \epsilon})$ for $\epsilon = 0.77$. By the theorem, we have $T(n) \in \Theta(n^{\log_3 7})$.

4.5 $T(n) = 9T(n/3) + n^3 \log n$

In this case we have $n^{\log_b a} = n^{\log_3 9} = n^2$. Thus we are in case 3 of the Master Theorem where $f(n) = n^3 \log n \in \Omega(n^{2+\epsilon})$ for $\epsilon = 1$. Also we have $af(n/b) = 9(n/3)^3(\log(n/3)) \leq \delta f(n)$ for $\delta = 1/3$. By the theorem, $T(n) \in \Theta(n^3 \log n)$.

5 C-11.3 Correctness of STOOGE-SORT

The correctness of STOOGE-SORT can be estasblished by a simple induction on the length of the array n. The case for $n = 1 \vee n = 2$ is trivially correct, since nothing is done when $n = 1$ and only a swap is performed if $n = 2$ and $A[0] > A[1]$. We assume correctness for size n. The correctness for the $n + 1$ case follows from the fact that the algorithm divides the problem into three recursive calls of size $2(n + 1)/3$, where each is of size less than n.

The recurrence is $T(n) = 3T(2n/3) + bn$. Using case 1 of the Master Theorem with $\epsilon = 1$, we have $T(n) \in \Theta(n^{\log_{3/2} 3})$, which is roughly $\Theta(n^{2.7})$

6 A-11.1 Bounding Box Construction

The goal is to design a divide-and-conquer algorithm for finding the minimum and maximum of n numbers using no more than $3n/2$ comparisons. We start by dividing the set of numbers into $n/2$ groups of two elements each. With $n/2$ comparisons we can determine the minimum and maximum of each group. We separate the minimums and maximums, and for each, we use simply scan through the collections to find the minimum and maximum. These additional scans take $n/2$ comparisons each. In total, we have no more than $3n/2$ comparisons.

7 R-12.9 Internet Auction Optimization Characterization

This is a knapsack problem where the weight of the sack is n, and each bid i corresponds to an item of weight k_i and value d_i. If each bidder i is unwilling to accept fewer than k_i widgets, then this is a 0-1 knapsack problem. On the other hand, if the bidders are willing to accept partial lots, then this is a fractional knapsack problem.

8 A-12.3 Uppercase letters to English Words

We have a string, S of n uppercase letters and a function $\mathsf{valid}(s)$ which can check whether a character string, s, is a valid English word in $O(1)$ time. The goal is to break S into a sequence of valid English words.

We create an array D of n elements. $D[j]$ is set to true if an only if the first j characters of S can be divided into valid words. We set $D[0]$ to true and define $D[j]$ recursively. For each $i < j$, we check whether $\mathsf{valid}(S[i : j])$ and whether $D[i - 1]$ is true. If this is the case, we have found a complete word and we set $D[j]$ to true. In general, it takes $O(n)$ time to check if $D[i]$ is true, given $D[0 : i - 1]$. The total running time of this algorithm, therefore is $O(n^2)$.