Problem 1

Question:

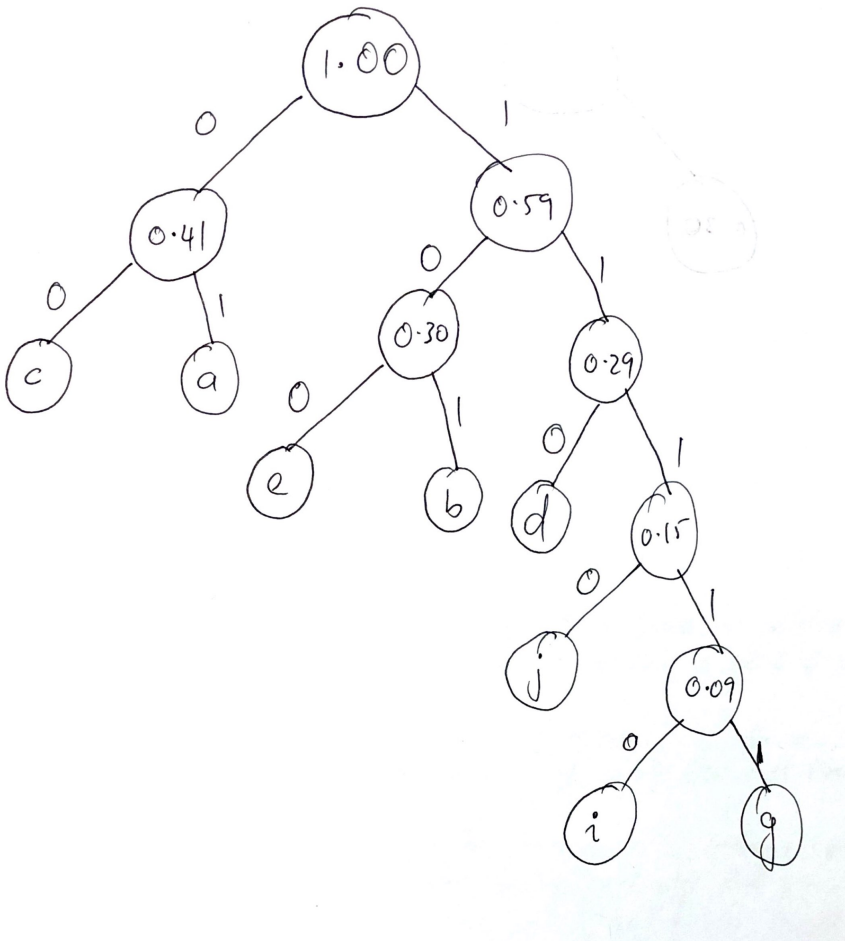
Suppose $S = \{a, b, c, d, e, g, i, j\}$, where the frequencies are $fa = 0.25, fb = 0.15, fc = 0.16, fd = 0.14, fe = 0.15, fg = 0.05, fi = 0.04, fj = 0.06$. Build an optimal prefix code for this data.

Answer:

Consider sorting this data by increasing frequency so we get the following Char, Frequency and Prefix Codes by Huffman Algorithm:

Char	Frequency	Prefix Code
i	0.04	$\gamma(i) = 11110$
g	0.05	$\gamma(g) = 11111$
j	0.06	$\gamma(j) = 1110$
d	0.14	$\gamma(d) = 110$
e	0.15	$\gamma(e) = 100$
b	0.15	$\gamma(b) = 101$
c	0.16	$\gamma(c) = 00$
a	0.25	$\gamma(a) = 01$

This is the tree we get from the huffman algorithm



Problem 2

Question:

For each of the following recurrence relations find the general pattern formula and find the running time $T(n)$ of the algorithm given by the recurrence relations:

Answer:

(a)

$$\begin{aligned}
T(n) &\leq 3T(n/2) + Cn \\
&\leq 9T(n/4) + 2Cn \\
&\leq 27T(n/8) + 3Cn \\
&\leq 3^k T(n/2^k) + kCn \qquad n/2^k = 1 \text{ at termination } \implies k = \log_2(n) \text{ and } T(1) = 1 \\
&\leq 3^{\log_2(n)} + Cn \cdot \log_2(n) \\
&\leq n^{\log_2(3)} + Cn \cdot \log_2(n) \\
T(n) &= \mathcal{O}(n^{1.585})
\end{aligned}$$

(b)

$$\begin{aligned}
T(n) &\leq 2T(n/3) + C \\
&\leq 4T(n/9) + 2C \\
&\leq 8T(n/27) + 3C \\
&\leq 2^k T(n/3^k) + kC \qquad n/3^k = 1 \text{ at termination } \implies k = \log_3(n) \text{ and } T(1) = 1 \\
&\leq 2^{\log_3(n)} + C \cdot \log_3(n) \\
&\leq n^{\log_3(2)} + C \cdot \log_3(n) \\
T(n) &= \mathcal{O}(n^{0.63})
\end{aligned}$$

(c)

$$\begin{aligned}
T(n) &\leq 5T(n/4) + Cn \\
&\leq 25T(n/16) + 2Cn \\
&\leq 125T(n/64) + 3Cn \\
&\leq 5^k T(n/4^k) + kCn \qquad n/4^k = 1 \text{ at termination } \implies k = \log_4(n) \text{ and } T(1) = 1 \\
&\leq 5^{\log_4(n)} + Cn \cdot \log_4(n) \\
&\leq n^{\log_4(5)} + Cn \cdot \log_4(n) \\
T(n) &= \mathcal{O}(n^{1.16})
\end{aligned}$$

Problem 2 continued on next page...

(d)

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + C \\
&\leq 4T(n/4) + 2C \\
&\leq 8T(n/8) + 3C \\
&\leq 2^k T(n/2^k) + kC \qquad n/2^k = 1 \text{ at termination } \implies k = \log_2(n) \text{ and } T(1) = 1 \\
&\leq 2^{\log_2(n)} + C \cdot \log_2(n) \\
&\leq n + C \cdot \log_2(n) \\
T(n) &= \mathcal{O}(n)
\end{aligned}
$$

(e)

$$
\begin{aligned}
T(n) &\leq 9T(n/3) + Cn^2 \\
&\leq 81T(n/9) + 2Cn^2 \\
&\leq 729T(n/27) + 3Cn^2 \\
&\leq 9^k T(n/3^k) + kCn^2 \qquad n/3^k = 1 \text{ at termination } \implies k = \log_3(n) \text{ and } T(1) = 1 \\
&\leq 9^{\log_3(n)} + Cn^2 \cdot \log_3(n) \\
&\leq n^{\log_3(9)} + Cn^2 \cdot \log_3(n) \\
&\leq n^2 + Cn^2 \cdot \log_3(n) \\
T(n) &= \mathcal{O}(n^2 \log_3(n)) = \mathcal{O}(n^2 \log(n))
\end{aligned}
$$

(f)

$$
\begin{aligned}
T(n) &\leq 8T(n/2) + Cn^3 \\
&\leq 64T(n/4) + 2Cn^3 \\
&\leq 512T(n/8) + 3Cn^3 \\
&\leq 8^k T(n/2^k) + kCn^3 \qquad n/2^k = 1 \text{ at termination } \implies k = \log_2(n) \text{ and } T(1) = 1 \\
&\leq 8^{\log_2(n)} + Cn^3 \cdot \log_2(n) \\
&\leq n^{\log_2(8)} + Cn^3 \cdot \log_2(n) \\
&\leq n^3 + Cn^3 \cdot \log_3(n) \\
T(n) &= \mathcal{O}(n^3 \log_3(n)) = \mathcal{O}(n^3 \log(n))
\end{aligned}
$$

Problem 3

Question:

Say that an array A with n elements has a majority element if more than half of its entries are the same. The elements of the arrays are not necessarily comparable (that is you cant ask the questions of the type $A(i) \leq A(j)$?). However, assume that you can ask the questions of the type $A(i) = A(j)$ in constant time.

Design an algorithm that tells us if A has a majority element. Your algorithm should run in $\mathcal{O}(n \cdot \log(n))$ time.

Answer:

The whole idea of the algorithm is to Divide until you reach the base case and then Conquer up towards the required answer :

Split the array recursively into halves, at the bottom layer where there are only 1 element in the array. Return the element into the layer above where a comparison is made with the other 1 element half.

The comparison is as follow:

Case 1: if both halves returns an element (true) , if the elements are the same, then return this element up to the higher layer as the majority element combined would also be a majority element in the combined array. If the elements are not the same, count the occurrence of each of the element in the array at the current layer. Return the element that has a higher count, if both count equals then return no majority.

Case 2: If exactly 1 half returns an element (true) and the other returns no majority (false), then count the occurrence of the element, if it is greater or equal to $n/2 + 1$ where n is the total number of elements of the array at the layer, then the array would have a majority element else it does not.

Case 3: if both halves returns no majority (false) , then the combine array would also not have a majority element.

Recursively run the algorithm up, and if an element is returned at the end, then the full array would have a majority element otherwise it does not. Note that String can be used as return type as it can represent both characters and integer values. By using this tree structure, the run-time of the algorithm will be O(nlogn).

Problem 3 continued on next page...

Problem 4

Question:

Solve Exercises 1 and 2 in Lecture Note 14.

Answer:

Exercise 1: Show that applying the greedy algorithm developed for the interval scheduling problem does not solve the weighted interval scheduling problem.

Let v_i represent the weight of a request r_i

Consider this schedule

r_1, $v_1 = 1000$

r_2, $v_2 = 3$ r_3, $v_3 = 4$

Using the greedy algorithm to select requests, the greedy algorithm would choose at each step/iteration the request that finishes first.

In the counter example above, the greedy algorithm would choose r_2 as it has the earliest finishing time (and eliminate r_1 since it is overlapping). Then, it would choose r_3 as that is the request that finishes next and it does not overlap with r_2.

The algorithm/schedule would end with a total weight of 7.

The optimal solution would be choosing r_1 where the total weight would be 1000. Therefore, the greedy algorithm does not always produce the optimal solution.

Excercise 2: Consider the sequence $p(1), p(2), ..., p(n)$.

1. Give examples where this sequence consist of 0s only.

In words: r_k is not compatible with $r_{k-1}, ..., r_1$

This usually happens when each request overlaps with all the requests before it for example:

$p(1) = 0$

$p(2) = 0$

$p(3) = 0$

Or when all of the requests overlap (same start time and finish time):

$p(1) = 0$

$p(2) = 0$

$p(3) = 0$

Problem 4 continued on next page...

2. Give examples where the sequence alternates between 0 and 1.

No overlapping with previous index request (pairwise compatible)

Alternates between overlapping with the first request and not overlapping the first request.

$p(1) = 0$

$p(2) = 1$

$p(3) = 0$

$p(4) = 1$

$p(5) = 0$

$p(6) = 1$

$p(7) = 0$

3. Give examples where the sequence has the form $0, 1, ..., n-1$

$p(1) = 0$

$p(2) = 1$

$p(3) = 2$

$p(4) = 3$

And so forth, and we will get that $p(n) = n - 1$.

Another example is (in order) :

$p(1) = 0$ $p(2) = 1$ $p(3) = 2$ $p(4) = 3$

and so forth...