Yash Avinash Patole

Course : CS600

Homework 1

CWID: 10460520

Chapter 10

Exercise 10.5.7

Fred says that he ran the Huffman coding algorithm for the four characters, A, C, G, and T, and it gave him the code words, 0, 10, 111, 110, respectively. Give examples of four frequencies for these characters that could have resulted in these code words or argue why these code words could not possibly have been output by the Huffman coding algorithm.

Answer:

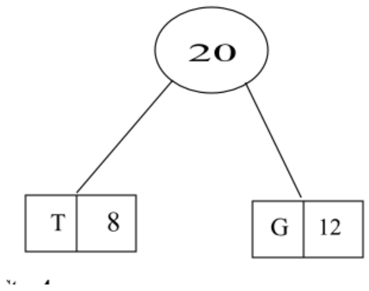
Consider frequencies A, C, G and T are as follows:

Characters	Frequencies
A	45
C	25
G	12
T	8

Constructing the Huffman Tree:

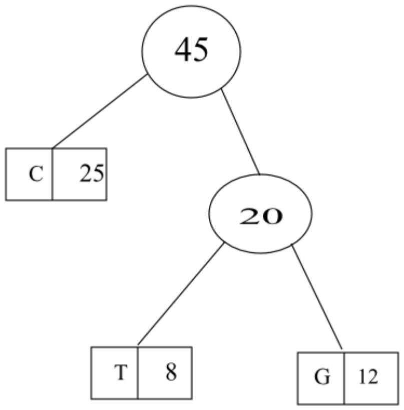
Step 1:

Select and include in the tree the two characters with the lowest frequencies. Here, G and T have the lowest frequencies, making it possible to add them to the tree. Add a parent node and update its value by incorporating T and G's frequencies.



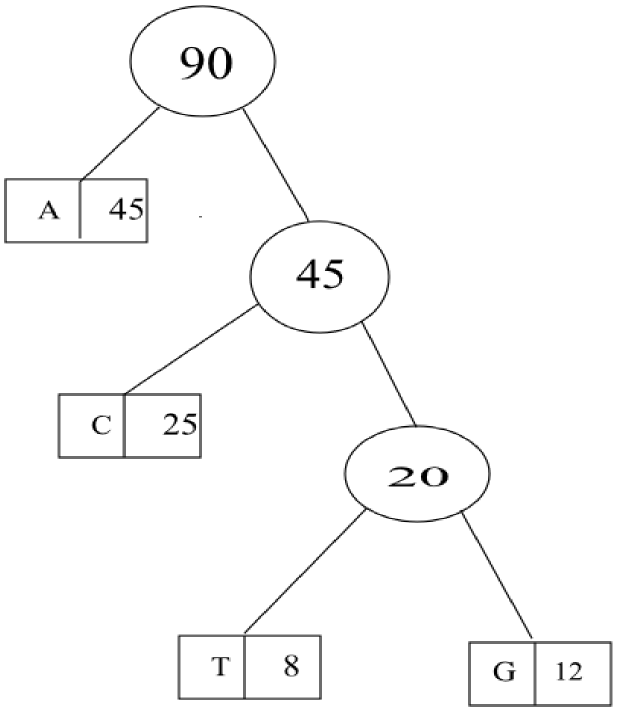
Step 2:

Add C to the tree and update the value of the new parent node by adding the frequencies of T, G, and C. Choose the character with the lowest frequency after T and G.



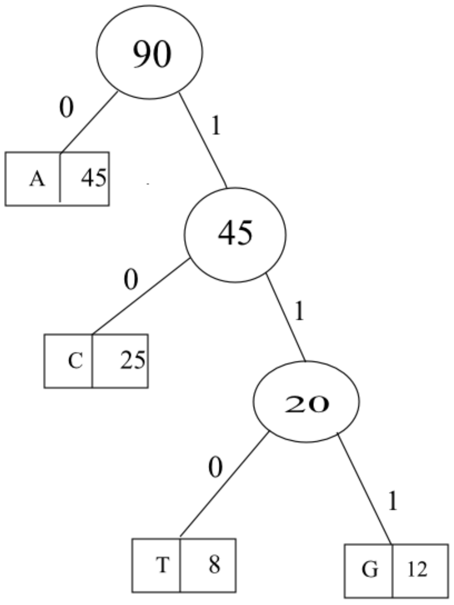
Step 3:

The highest frequency character, A, is now the only one left. Add A to the tree and update the value of the new node by including A, C, T, and G's frequencies.



Step 4:

Give the value O to all of the tree's left nodes now, and I to all of its right nodes.



Now considering the tree mentioned above the Huffman code is as follows:

Characters	Huffman Code
A	0
C	10
G	111
T	110

Exercise 10.5.16

Give an example set of denominations of coins so that a greedy change making algorithm will not use the minimum number of coins.

Answer:

The following approach can be adapted when we are looking for a solution while not looking for minimum number of coins:

The denominations can be selected randomly instead of going in an order from maximum denominations to the lowest denomination. As, in the case where the order has been followed, we will be heading for minimum number of coins. While selecting randomly we might end up with either minimum or maximum number of coins. So we might have 2,5,7 and amount is 16 while in random selection we might have 8 (2) or 2 (5), 2(3), etc.

To reduce the number of coins, a greedy algorithm selects the coin with the highest denomination first. Since all coins must have the same denomination in order to exist, we do not have any coins with higher denominations than the others. Therefore, we just need pennies, and since a penny may be used to exchange any value, the fixed denomination will be {1}.

Exercise 10.5.27

Answer:

When determining whether the string Y is an is sub-sequence of the string X.
we use a greedy approach.
Following is the algorithm:

```
Algorithm isSubsequence(X, Y)
    Input: A string X, and a string Y
    Output: True if Y is a subsequence of X, else False
    Initialize a counterj •- 1
    for i I to X.length() do
        if j <= Y.length() and X[i] == Y[j]
            j ← j+1
    if j == Y.length() + 1
        return True
    else
        return False
```

In the technique above, the for loop is repeated n times because the length of the string X is that number, and in the best situation, the increment of j is repeated m times. The algorithm's temporal complexity is therefore O(n + m).

<u>Chapter 10</u>

Exercise 11.6.1

Characterize each of the following recurrence equations using the master theorem (assuming that T (n) = c for n < d, for constants c > O and d >= 1).

Answer:

a) **$T(n) = 2T(n/2) + \log n$**

While in this case a = 2, b = 2 and f(n) = log n

According to Master Theorem $n^{\log_b a}$ = n. Thus considering in Case 1 where f(n) is O($n^{1-\varepsilon}$) for $0 < \varepsilon < 1$. This means that T(n) is Θ(n) by Master theorem.

b) T(n) = 8T(n/2) + n^2

While in this case a = 8, b = 2 and f(n) = n^2

According to Master Theorem $n^{\log_b a}$ = n^4. Thus considering in case 1 where f(n) is O($n^{1-\varepsilon}$) for ε = 1. This means that T(n) is O(n^3) by Master Theorem.

c) T (n) = 16T (n/2) + (n log n) ^{4}

While in this case a = 16, b = 2 and f(n) = $n^4 \log^4 n$

According to Master Theorem $n^{\log_b a}$ = n^4. Thus considering in Case 2 with k = 4, for f(n) is Θ(n 4 log4 n). This means that T(n) is Θ(n 4 log5 n) by Master theorem.

d) T (n) = 7T (n/3) + n

Here a = 7, b = 3 and f(n) = n

According to Master Theorem $n^{\log_b a}$ = $n^{\log_3 7}$. Thus considering in Case 1 where f(n) is O(n $^{3-\varepsilon)}$ for ε = $\log_3 7$ - 1. This means that T(n) is Θ($n^{\log_3 7}$) by Master theorem.

e) T (n) = 9T (n/3) + n^3 log n

While in this case a = 9, b = 3 and f(n) = n 3 log n

According to Master Theorem $n^{\log_b a}$ = n^2. Thus considering in Case 3, since f(n) is O($n^{2+\varepsilon}$) for ε = 1 and af(n / b) = 9((n^3 log (n/3)) / 27) = 1 / 3f(n) – log3 / 3. This means that T(n) is Θ(n^3 log n) by Master theorem.

Exercise 11.6.10

Consider the Stooge-sort algorithm, shown in Algorithm 11.6.1 , and suppose we change the assignment statement for m (on line 6) to the following: m ← max{l, In/4]}

Characterize the running time, T (n), in this case, using a recurrence equation, and use the master theorem to determine an asymptotic bound for T (n).

Answer:

$m \leftarrow max\{1, \lfloor n / 4 \rfloor\}$

Use a recurrence equation to describe the running time, T (n), in this situation, and the master theorem to find an asymptotic constraint for T. (n).

If we do the above changes in the Stooge-sort algorithm we would be solving 3/4th of the problem 3 times. Therefore, the new recurrence relation would be

$T(n) = 3T(3n/4) + O(1)$

Here $a = 3$, $b = 4 / 3$ and $f(n) = O(1)$.

According to Master Theorem $n^{\log_b a} = n^{\log_{4/3} 3}$.

Thus we are in Case 1 where f(n) is $O(n^{\log_{4/3} 3.} - \varepsilon)$ for $\varepsilon = \log_{4/3} 3$. This means that T(n) is $\Theta(n^{\log_{4/3} 3})$ which can be further by simplified to $O(n^{\log 3 / \log 1.33}) = O(n^{3.82})$.

Exercise 11.6.17

Answer:

Algorithm Skyline(S):

```
    Input: A Set S of sub intervals with n tuples ($a_i$, $b_i$, $h_i$) where $0 \leq a_i < b_i \leq 1$
    and $h_i$ is the height of the interval [ai, bi]
    Output: The skyline of S
    Sort the tuples by their x co-ordinates
    Create a BST T containing the tuples sorted by height h
    Create set O containing the output
```

```
for i = 1 to n do
    T.add(Si)
    if Si is the highest node in T then
        O.add([ai, hi])
    t ←T.remove(Si)
    if t is the highest node in T then
        t2←Highest node in T
        if t2 is not empty then
            O.add([bi, t2.h ])
        else
            O.add([bi, 0])
return O
```

The Sorting takes O(nlog n) time.

O(n) time will be required to loop through all of S's items.

It will take O(log n) time to add and remove S in T's objects.

O(1) time is required to add to the output set O.

Thus, the entire algorithm takes O(n log n) time to complete.

Chapter 12

Exercise 12.8.5

Let S ={a, b, c, d, e, f, g} be a collection of objects with benefit-weight values, a: 12,4 , b: 10,6 , c: 8,5 , d: 11,7, e:14,3 , f:7,1 , g :9,6. What is an optimal solution to the 0-1 knapsack problem for S assuming we have a sack that can hold objects with total weight 18? Show your work.

Answer:

Items	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
a(12,4)	0	0	0	0	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
b(10,6)	0	0	0	0	12	12	12	12	12	12	22	22	22	22	22	22	22	22	22
c(8,5)	0	0	0	0	12	12	12	12	12	20	22	22	22	22	22	30	30	30	30
d(11,7)	0	0	0	0	12	12	12	12	12	20	22	23	23	23	23	30	31	33	33
e(14,3)	0	0	0	14	14	14	14	26	26	26	26	26	34	36	37	37	37	37	44
1(7.1	0	7	7	14	21	21	21	26	33	33	33	34	34	41	43	44	44	44	44
g(9, 6)	0	7	7	14	21	21	21	26	33	33	33	34	34	41	43	44	44	44	44

Exercise 12.8.14

Show that we can solve the telescope scheduling problem in O(n) time even if the list of n observation requests is not given to us in sorted order, provided that start and finish times are given as integer indices in the range from 1 to n^2.

Answer:

If we use the following approach to discover the Predecessor P of I we can still solve the telescope scheduling problem in O(n) time even if the list of n observation requests is not provided to us in sorted order.

Algorithm for the same is as follows:

Algorithm P(L, i):

```
    Input: List L of n observation requests and requests i whose predecessor is to be searched

    Output: Predecessor of request i
    p ← 0
    for k = 1 to n:
        if p < endk and endk <= starti then
            p ← endk
    return p
```

Telescope scheduling algorithm can be used to find with this algorithm to find the max benefit in O(n + n) = O(n) time

```
    B[0] ← 0

    for i = 1 to n do

        B[i] ← max{B[i − 1], B[P(i)] + bi }
```

Exercise 12.8.30

Answer:

```
Algorithm: longestPalindrome(String s)
Input: String s
Output: Returning a subsequence with longest increasing subsequence
Declare a 2d array of type integer
Int dp[][] = [s.length + 1][s.length + 1];
for(i  1 to s.length)
        for(j  1 to s.length)
                if(s.charAt(i - 1) == s.charAt(s.length -i - 1)
                        dp[i][j]   1+ dp[i][j]
                else
                        dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
After calculating DP the algorithm should return a string
String reverse_string  S.reverse()
char[] temp  new char[dp[s.length][s.length]]
Index dp[s.length][s.length] - 1
i s.length
j  s.length
while(i > 0 && j > 0) {
if(S.charAt(i)== reverseString.charAt(j)) {
temp[index] = S.charAt(i);
Index–;
j–;
i–;
}
else if(dp[i - 1][j] > dp[i][j-1]
        i–;
else
        j–;
}
return String.valueOf(temp)
```

Alternative approach in python can be:
Algorithm:

```
def longestPalindrome(s: str) -> str:
```

```
m = ''  # Memory to remember a palindrome
for i in range(len(s)):  # i = start, O = n
    for j in range(len(s), i, -1):  # j = end, O = n^2
        if len(m) >= j-i:  # To reduce time
            break
        elif s[i:j] == s[i:j][::-1]:
            m = s[i:j]
            break
return m
```