1. 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:

The code words provided in the question are indeed generated by the Huffman coding algorithm.

Let us consider one example for which we can generate a valid Huffman Tree.

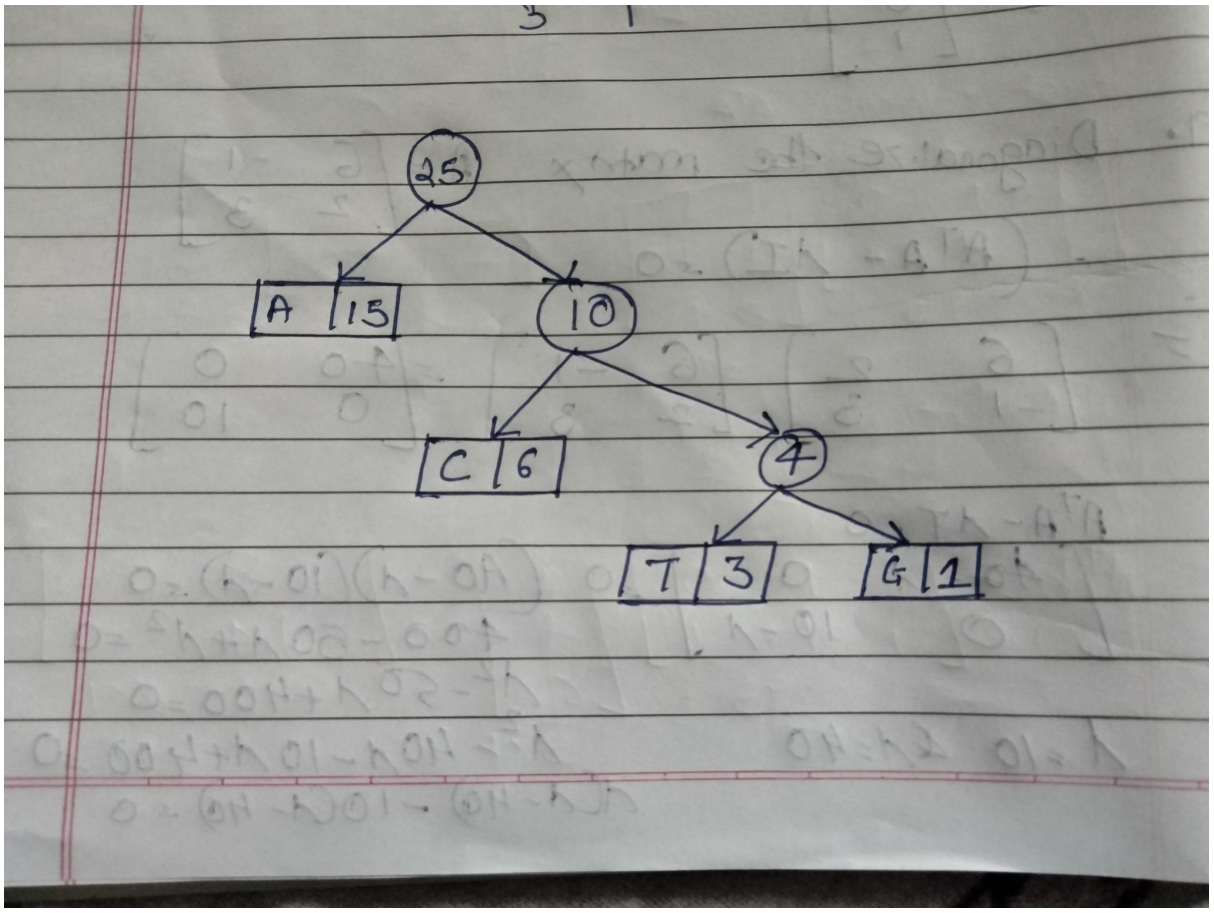
Consider the frequencies of the letters as follows
A→15 C → 6 G → 3 T →1

‘0’ representation in the Huffman Coding Algorithm states that we traverse left and ‘1’ represents to traverse right.

In order to generate a Huffman tree we first select two frequencies with minimum value. Here in this case G and T. Add a parent for G & T which stores the total frequency of T & G.

Next we select C and follow the same process. And finally we will select A.

So the tree will look something like this.



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

Answer:

Let us consider an example S = {10,6,9,1} where S is the set that contains denominations of coins and let v = 15 dollars where v represents the value.

In the above example we need to have coins with a total value of 15 dollars.

If we follow the greedy approach we will consider one 10 dollar coin and five 1dollar coins. This will lead us to the value 15 dollars

However, this is not an optimal solution
The correct solution will be to pick one 6 dollar coin and one 9 dollar coin. This will lead us to the value of 15 dollars and the minimum number of coins.

3.
When data is transmitted across a noisy channel, information can be lost during the transmission. For example, a message that is sent through a noisy channel as "WHO PARKED ON HARRY POTTER'S SPOT?"could be received as the message, "HOP ON POP"That is, some characters could be lost during the transmission, so that only a selected portion of the original message is received. We can model this phenomenon using character strings, where, given a string $X=x_1x_2 \ldots x_n$, we say that a string $Y=y_1y_2 \ldots y_m$ is a ***subsequence*** of X if there are a set of indices $i_1,i_2,\ldots,i_k$, such that $y_1=x_{i1}$, $y_2=x_{i2}$, …, $y_k=x_{ik}$, and $i_j<i_{j+1}$, for $j=1,2,\ldots,k-1$. In the case of transmission along a noisy channel, it could be useful to know if our transcription of a received message is indeed a subsequence of the message sent. Therefore, describe an O(n+m)-time method for determining if a given string,Y, of length mis a subsequence of a given string, X, of length n.

Answer:
We will not be using any data structure to solve this problem
Algorithm: **checkValidSub**(X, Y)
Input: Two strings X and Y
Output: Returning true if Y is a subsequence of X else false

```
i ← 0
j ←  0

if(Y.length is greater than X.length) then
        return false

while(i < X.length and j < Y.length):
        if(X.charAt(i) == Y.charAt(i)) then
                increment j
        Increment i
if(j is equal to Y.length) then
        return true
else
        return false
```

The time complexity of the above algorithm is O(n +m) where n is the number of elements in string X and m is the number of elements for string Y.

4.

Characterise each of the following recurrence equations using the master theorem (assuming that T(n)=c for n<d, for constants c>0 and d≥1).

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

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

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

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

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

Solution:
According to the master's theorem

If $T(n) = a\ T(n / b) + O(n^d)$(for constants $a > 0$, $b > 1$ and $d >= 0$) then

$T(n) = O(n^d)$ if $d > \log_b a$
$= O(n^d \log n)$ if $d = \log_b a$
$= O(n^{\log_b a})$ if $d < \log_b a$

a) $T(n)=2T(n/2)+\log n$
Here $a = 2$, $b = 2$ and $d = 0$
Since $d < \log_b a$
$T(n) = O(n)$

b) $T(n)=8T(n/2)+n^2$
Here $a = 8$, $b = 2$ and $d = 0$
Since $d < \log_b a$
$T(n) = O(n^3)$

c) $T(n) = 16T(n/2) + (n\log n)^4$
Here $a = 16$, $b = 2$ and $d = 4$ since $d = \log_b a$
$T(n) = O(n^4 \log^5 n)$

d) $T(n) = 7T(n/3)+n$
Here $a = 7$, $b = 3$ and $d = 1$
Since $d < \log_b a$
$T(n) = O(n^{1.77})$

e) $T(n) = 9T(n/3) + n^3 \log n$
Here $a = 9$ $b = 3$ and $d = 3$

Since d > $\log_b a$

$T(n) = O(n^3 \log n)$
Resource:
https://www.youtube.com/watch?v=2H0GKdrIowU

5.
Consider the Stooge-sort algorithm,and suppose we change the assignment statement for m(on line 6) to the following:m←max1,n/4 Characterise 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:

We know that the standard recurrence relation for stooge sorting algorithm is given as T(n) = 3T(2N/3) + O(1)

In this case if we set m ← max{1, n / 4}

We get relation as
T(N) = 3T(3N/4) + O(1)

Let us calculate the complexity using masters theorem

We have a = 3 b = 4/3 = 1.25 and d = 0

Since d < $\log_b a$

Then

T(n) =O(n raised to ($\log_b a$)) = O(n raised to $\log_{1.33} 3$) = O($n^{3.852}$)

6.

We will use the similar technique which is used in the mergeSort

We will use the divide and conquer technique. We will divide the array until we have one element. Once we have divided the array in left and right half we will merge the array and return it.

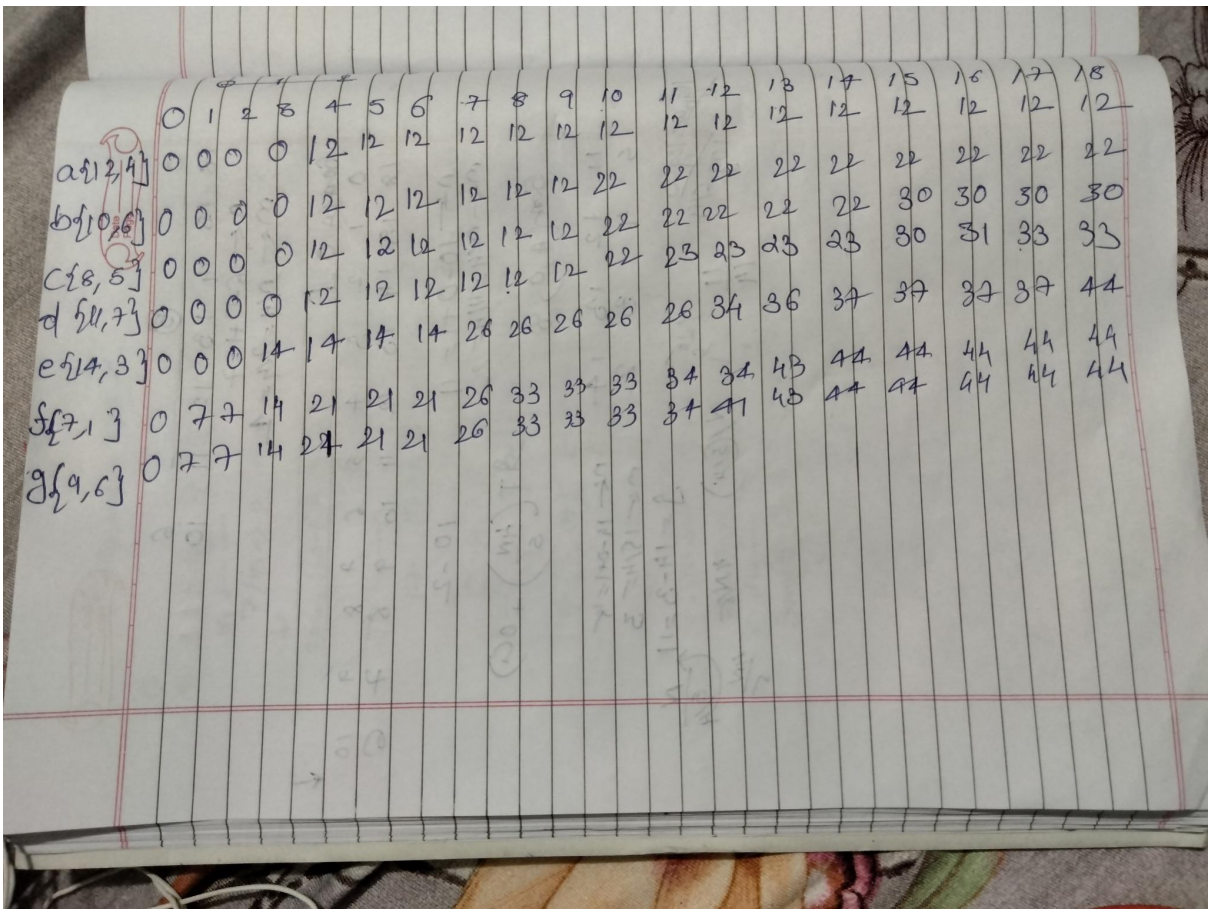
```
findSkyline(S,l,r)
if(l == r) then
        Declare a list that stores pairs
        list.append(store the starting point i.e S.get(l).get(0), height)
        list.append(store the ending point i.e S.get(r).get(1), 0)
```

```
mid ← (l + r) / 2

skyLine1 = findSkyline(S,l,mid)
skyLine2 = findSkyline(S,mid + 1,r);
skyLine result = mergeSkyLine(skyLine1, skyLine2, l, mid ,r)

mergeSkyLine(skyLine1, skyLine2, l, mid, r)

h1←0
h2←0
Declare a list result of size (r - l + 1)

i←0
j←0

while(i < skyLine1.size() && j < skyLine2.size())
        if(skyLine1.get(i).get(0) < skyLine2.get(j).get(0)) then
                temp ← skyLine1.get(i).get(0)
                h1 ← skyLine1.get(i).get(1)
                maxh ← max(h1, h2)
                result.append(temp, maxh)
                i ← i + 1
        Else
                temp ← skyLine2.get(j).get(0)
                h2 ← skyLine2.get(j).get(1)
                maxh ← max(h1, h2)
                result.append(temp, maxh)
                j ← j + 1

while(i < skyLine1.size())
        result.append(skyLine1.get(i))
        i ← i + 1
while(j < skyLine2.size())
        result.append(skyLine2.get(j)
        j ← j + 1
```

Resources:
https://www.geeksforgeeks.org/the-skyline-problem-using-divide-and-conquer-algorithm/

7. 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:



Thus the maximum profit can be obtained if we consider {a,b,c,e}

8.
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 discover the Predecessor P of I with the following approach in O(n) time, we can solve the telescope scheduling issue even if the list of n observation requests is not provided to us in sorted order.

Algorithm *P(L, i):*

Input: List *L* of *n* observation requests and request *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*

With this algorithm we can still use the telescope scheduling 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* }

9.

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])
```

Once we have calculated dp table we have to return the 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)
```

Resource: https://www.geeksforgeeks.org/word-break-problem-dp-32/