Data Analysis and Algorithm

Practical 6

Write a program to implement Huffman's code algorithm.

Date.: 11-10-21

Name – Yash Vasudeo Prajapati

Rollno - 022

MSc. Computer Science

6) Write a program to implement Huffman's coding.

Theory :- Huffman Coding is a technique of compressing data to reduce its size without losing any of the details. It was first developed by David Huffman.

Example :-

Consider a string "this is an example"

Char	Freq	code	size	
s	2	000	2*3	6
i	2	001	2*3	6
e	2	010	2*3	6
a	2	011	2*3	6
m	1	1000	1*4	4
x	1	1001	1*4	4
l	1	1010	1*4	4
f	1	1011	1*4	4
t	1	1100	1*4	4
n	1	11010	1*4	4
h	1	11011	1*4	4
' '	3	111	3*3	9
12*8 = 96 bits				61 bits

We notice that the string without encoding would take

216 bits

And after encoding

$96 + 18 + 61 = 175$ bits

which is not much but with larger strings or strings with more frequency of same characters a clear difference is seen.

7 steps :-

Let see another example

'afsdfasfasdafasda'

1) Calculate the frequency.

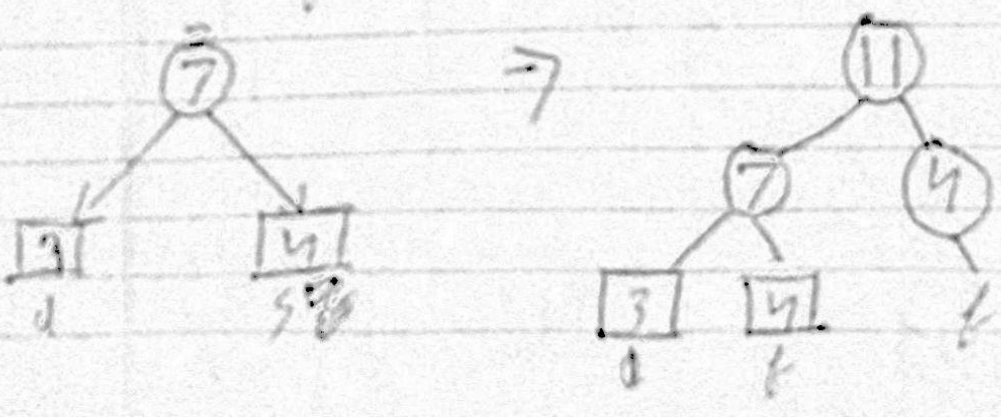
a	f	s	d
↓	↓	↓	↓
6	4	4	3

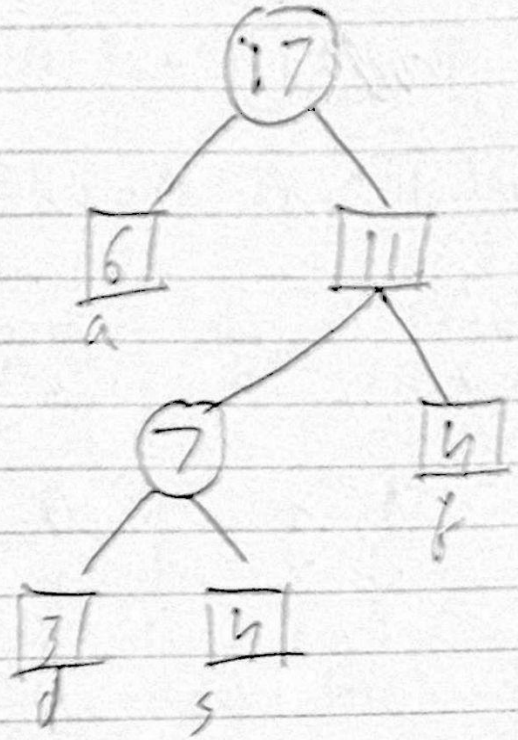
2) Sort in increasing order of frequency

d	s	f	a
3	4	4	6

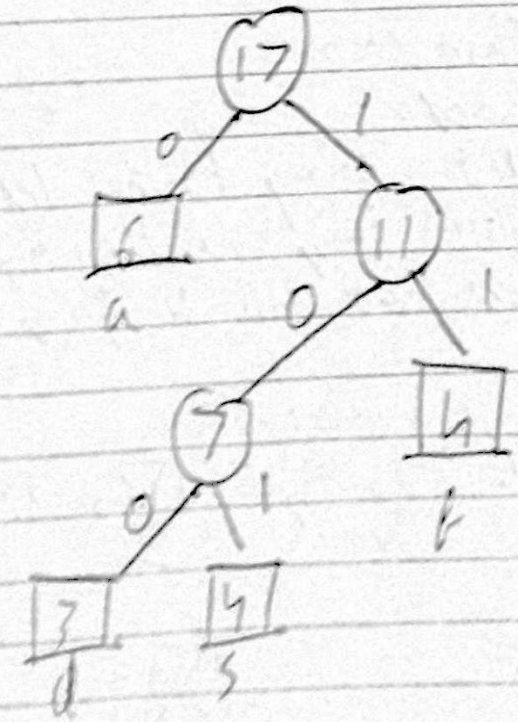
3) Make each character a leaf node, remove the first two nodes & replace with an empty node whose value is the sum of the freq of left & right child & repeat till all nodes are replaced.

7	4	6		11	6
z	f	a		z	a





4) Using 0 to the left & 1 to right of non-leaf node edge.



We get each code :

a → 0
d → 100
f → 11
s → 101

Algorithm

1) Creat a list with all unique characters
2) Calculate the frequency of each unique characters
3) Repeat for all characters.
   a) create a new node
   b) assign the smallest freq to the left
   c) assign the smallest freq to the right.
   d) Sum of the above two will be freq of new node.
   e) insert new node in tree.
4) Assign 0 to left & 1 to right nodes recursively
5) return code

Complexity Analysis:

To create a min heap tree it takes $\theta(n)$

For every node we follow two steps

1) find to 2 freq $\Rightarrow \log(n) + \log(n)$
since we are using heap tree.

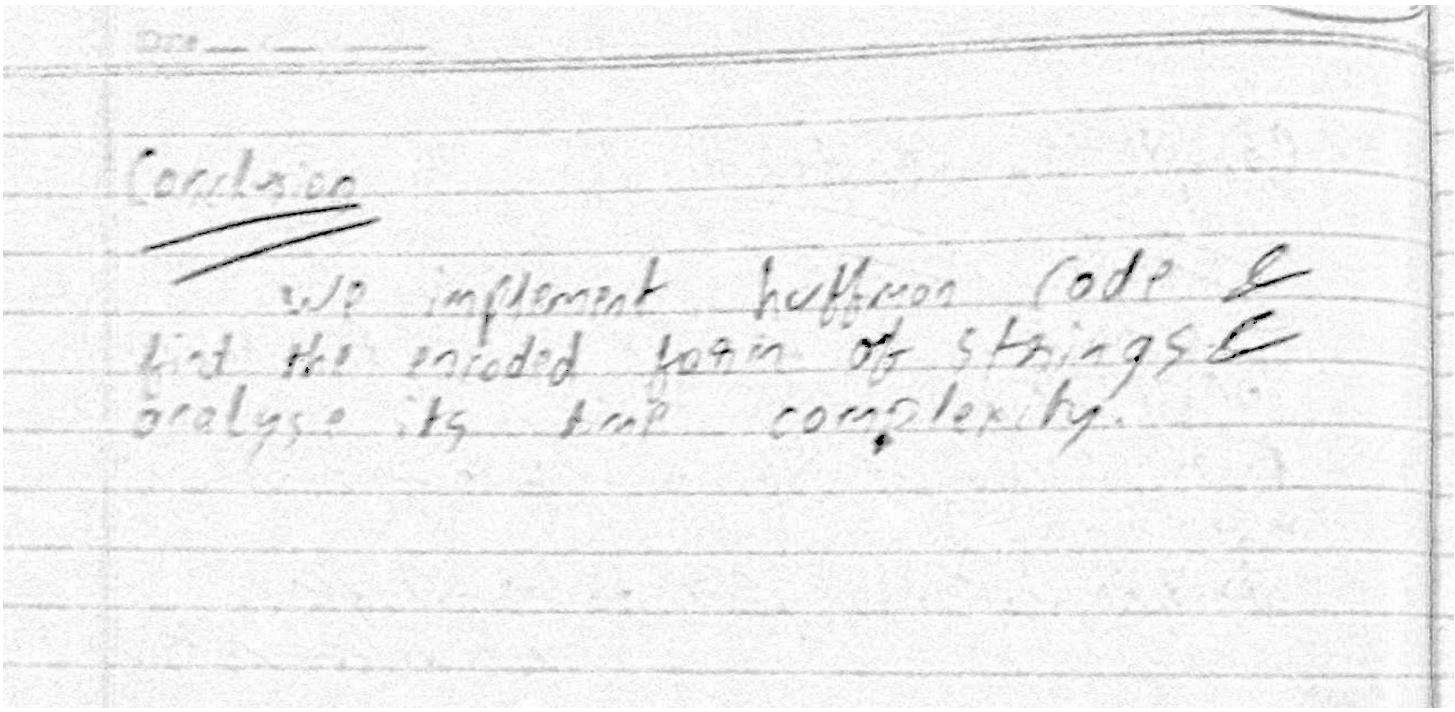
2) merge $\Rightarrow \log(n)$
$\Rightarrow$ create a new node & merge 2
place the new node

Thus giving us $\log(n) + \log(n) + \log(n)$

$\Rightarrow 3\log(n) \approx \log(n)$

Then we create tree for all nodes which is one less the total node
$(n-1) \approx n$

Then we can say the total time taken will be
$O(n \log(n))$



Program

```
class NodeTree(object):
    def __init__(self, left=None, right=None):
        self.left = left
        self.right = right

    def nodes(self):
        return (self.left, self.right)

# Main function implementing huffman coding
def huffman_code_tree(node, left=True, binString=''):
    if type(node) is str:
        return {node: binString}
    (l, r) = node.nodes()
    d = dict()
    d.update(huffman_code_tree(l, True, binString + '0'))
    d.update(huffman_code_tree(r, False, binString + '1'))
    return d


# Calculating frequency
string = 'ACDSVDASCASDAW'
freq = {}
for c in string:
    if c in freq:
        freq[c] += 1
    else:
        freq[c] = 1

freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

nodes = freq

while len(nodes) > 1:
    (key1, c1) = nodes[-1]
    (key2, c2) = nodes[-2]
    nodes = nodes[:-2]
    node = NodeTree(key1, key2)
    nodes.append((node, c1 + c2))
```

```
    nodes = sorted(nodes, key=lambda x: x[1], reverse=True)

huffmanCode = huffman_code_tree(nodes[0][0])
print(huffmanCode)
```

Coding.py

```
{'S': '00', 'D': '01', 'W': '1000', 'V': '1001', 'C': '101', 'A': '11'}
```