Design & Analysis of Algorithm (Lab)

Name: Ananya

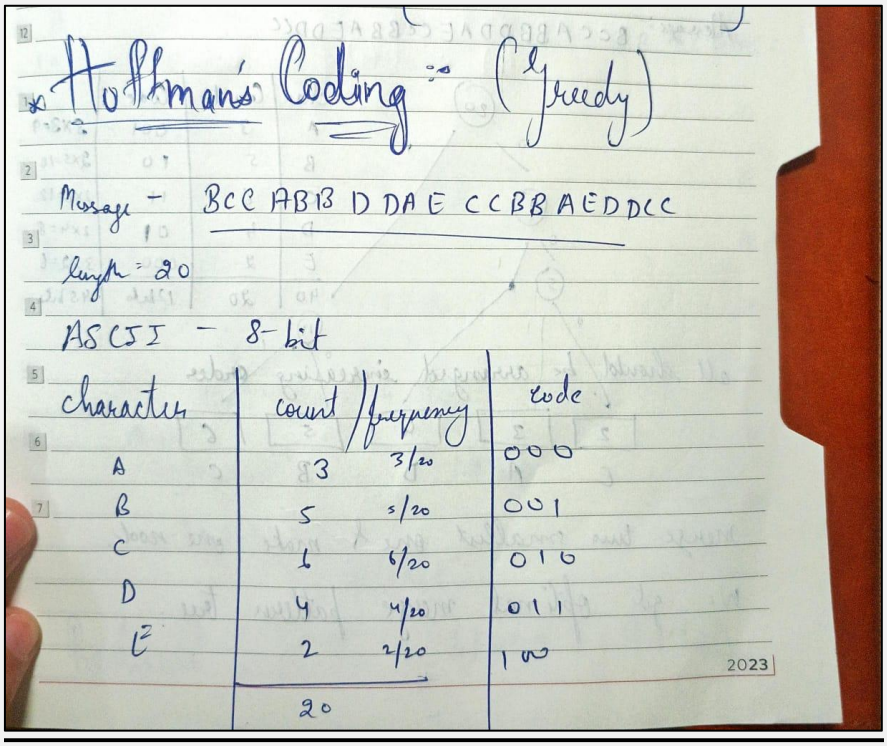
SAPID: 590013832

B-33

Submitted to: Mr.Aryan Gupta

https://github.com/ananya438/DAALAB_ANANYA-590013832

Huffman's Coding



Huffman's Coding :- (Greedy)

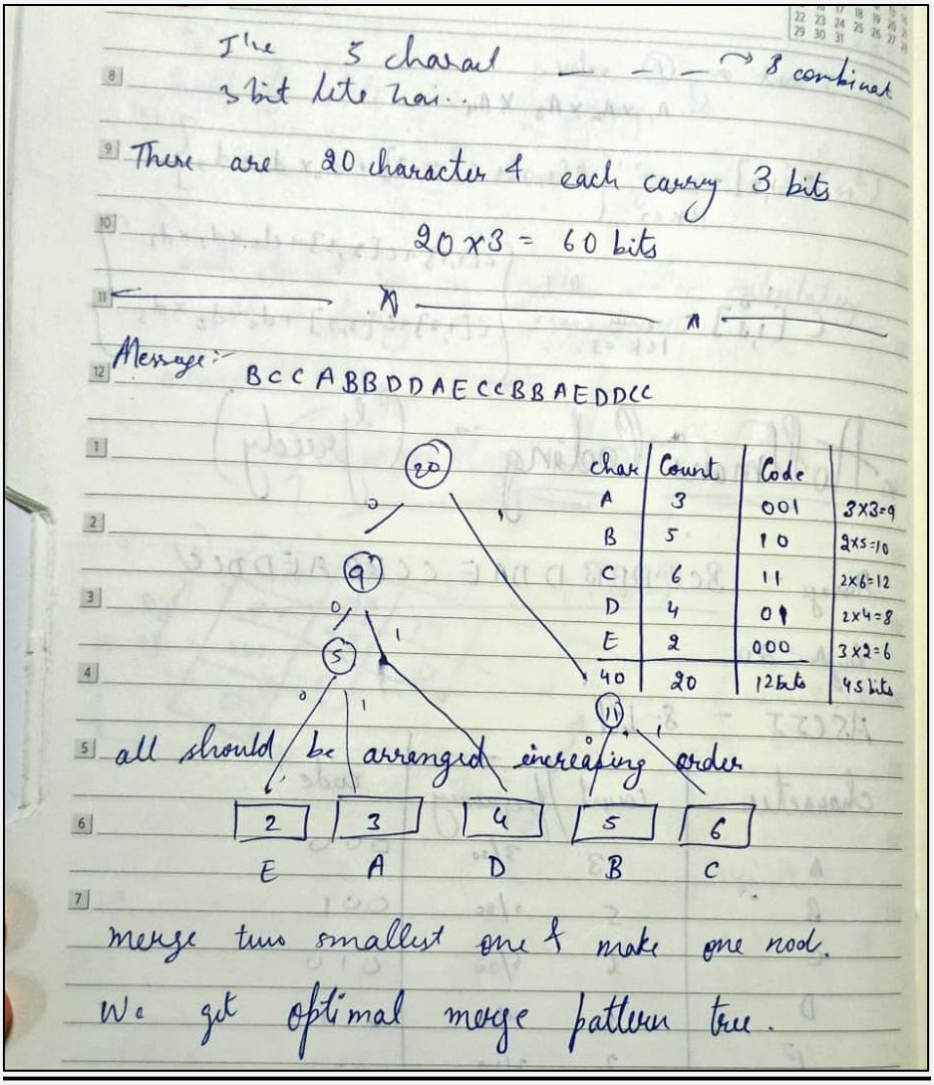
Message - BCC ABB D DA E CCBBAEDDCC

length = 20

ASCII - 8-bit

character	count / frequency		code
A	3	3/20	000
B	5	5/20	001
C	6	6/20	010
D	4	4/20	011
E	2	2/20	100
	20		

2023



There are 5 character ∴ 3 combination 3 bit lete hai..

There are 20 character & each carry 3 bits

$20 \times 3 = 60$ bits

Message: BCCABBDDAECCBBAEDDCC

char	Count	Code	
A	3	001	3×3=9
B	5	10	2×5=10
C	6	11	2×6=12
D	4	01	2×4=8
E	2	000	3×2=6
40	20	12 bits	45 bits

all should be arranged increasing order

merge two smallest one & make one node.

We get optimal merge pattern tree.

(Greedy approach)

- Mark left hand edges = 0
  right " " = 1
- From Tree find code.

Message

B C C A B B DD A E C C BB AEDDCC

10 11 11 001 10 10 01 01 ....

Size of message?

Ans 9+10+12+8+6 = **45 Bits.**

Size of Table

ASCII Code = 8 bits

Character = 5 b

So 5×8 = 40 bits

And

Code will carry = 12 bits

Total = 40+12

= **52 bits**

Reduced bits to = 45+52 = 97 bits

2023

---

23 SUNDAY

Time Complexity Analysis

1. Building Priority Queue $O(n)$
2. Extraction + Insertion $O(\log n)$
3. Performed $n \rightarrow 1$ times $\longrightarrow$ $O(n \log n)$

- Final : $O(n \log n)$
- S(n) : $O(n)$ (tree + nodes)

```
import java.util.PriorityQueue;
class Node {
    char ch;
    int freq;
    Node left, right;
    Node(char ch, int freq) {
        this.ch = ch;
        this.freq = freq;
    }
}
class HuffmanComparator implements java.util.Comparator<Node> {
    public int compare(Node n1, Node n2) {
        return n1.freq - n2.freq;
    }
}
public class HuffmanCoding {
    public static void printCodes(Node root, String code) {
        if (root == null) return;

        // If leaf node ⟶ print character and code
        if (root.left == null && root.right == null) {
            System.out.println(root.ch + " : " + code);
            return;
        }
        printCodes(root.left, code + "0");
        printCodes(root.right, code + "1");
    }
    public static void buildHuffmanTree(char[] chars, int[] freq) {
        int n = chars.length;

        // Min-heap using priority queue
        PriorityQueue<Node> pq = new PriorityQueue<>(n, new HuffmanComparator());
```

```
        // Insert all characters into priority queue
        for (int i = 0; i < n; i++) {
            pq.add(new Node(chars[i], freq[i]));
    }
        while (pq.size() > 1) {
            Node left = pq.poll();   // smallest freq
            Node right = pq.poll();  // next smallest freq
            // Create new internal node
            Node merged = new Node('-', left.freq + right.freq);
            merged.left = left;
            merged.right = right;
            pq.add(merged);
        }

        Node root = pq.poll();
        printCodes(root, "");
    }

    public static void main(String[] args) {
        char[] chars = {'a', 'b', 'c', 'd', 'e', 'f'};
        int[] freq = {5, 9, 12, 13, 16, 45};
        buildHuffmanTree(chars, freq);
    }
}
```

O/P:

```
PS C:\Users\nannu\Desktop\JAVA DSA\JAVA\First lectures\DAA> cd "c:\
.java } ; if ($?) { java HuffmanCoding }
f : 0
c : 100
d : 101
a : 1100
b : 1101
e : 111
```