AMRITA VISHWA VIDYAPEETHAM

CHENNAI CAMPUS

Second Year. B.TECH

(Computer Science and Engineering)

(DAA Lab Work)

Name: M CYNTHIA SHREE

RollNo.: CH.SC.U4CSE24110

Department: CSE-B

Academic Year: 2025-26

1. Huffman Coding

CODE:

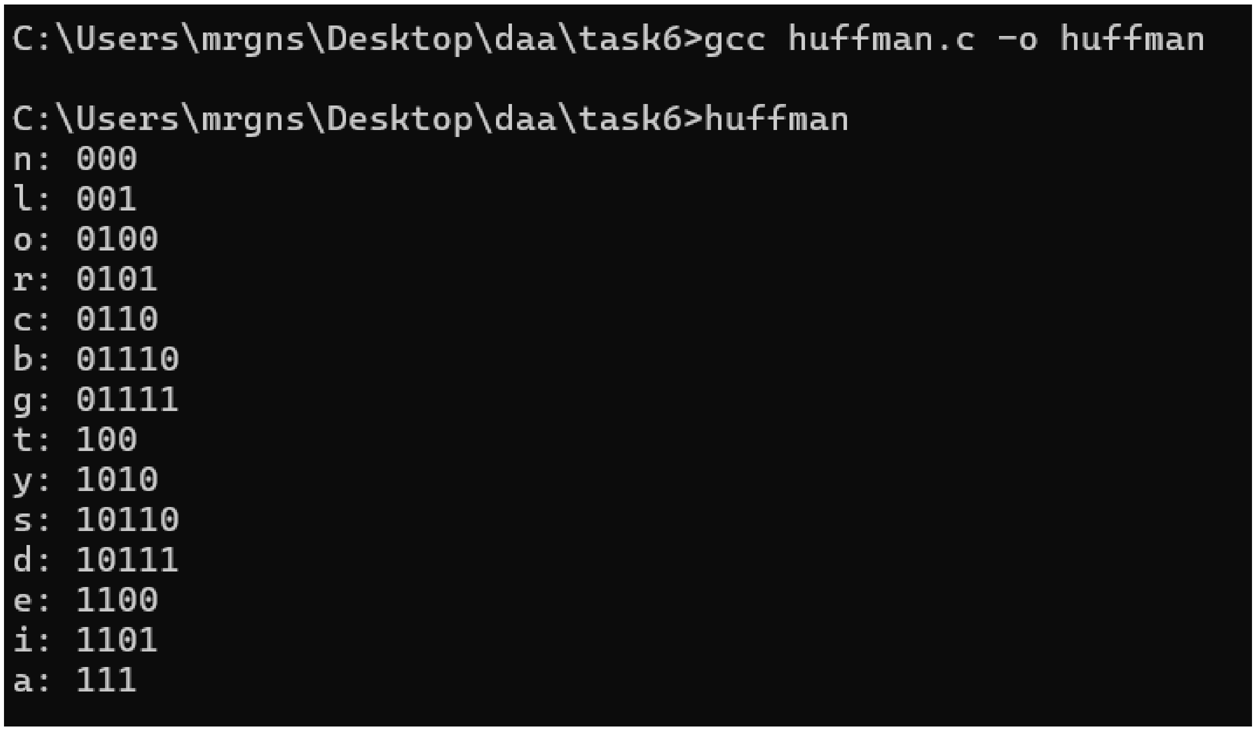
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_TREE_HT 100
#define SIZE 14

char data[SIZE] = {'a','b','c','d','e','g','i','l','n','o','r','s','t','y'};
int freq[SIZE] = {7,1,2,2,3,1,3,4,4,2,2,1,4,2};

struct MinHeapNode {
    char data;
    unsigned freq;
    struct MinHeapNode *left, *right;
};

struct MinHeap {
    unsigned size;
    unsigned capacity;
    struct MinHeapNode** array;
};

struct MinHeapNode* newNode(char data, unsigned freq) {
    struct MinHeapNode* temp =
        (struct MinHeapNode*)malloc(sizeof(struct MinHeapNode));
    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;
    return temp;
}

struct MinHeap* createMinHeap(unsigned capacity) {
    struct MinHeap* minHeap =
        (struct MinHeap*)malloc(sizeof(struct MinHeap));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array =
        (struct MinHeapNode**)malloc(capacity * sizeof(struct MinHeapNode*));
    return minHeap;
}
```

```
void swapNode(struct MinHeapNode** a, struct MinHeapNode** b) {
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}

void minHeapify(struct MinHeap* minHeap, int idx) {
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;

    if (left < minHeap->size &&
        minHeap->array[left]->freq < minHeap->array[smallest]->freq)
        smallest = left;

    if (right < minHeap->size &&
        minHeap->array[right]->freq < minHeap->array[smallest]->freq)
        smallest = right;

    if (smallest != idx) {
        swapNode(&minHeap->array[smallest], &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
}

int isSizeOne(struct MinHeap* minHeap) {
    return (minHeap->size == 1);
}

struct MinHeapNode* extractMin(struct MinHeap* minHeap) {
    struct MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];
    --minHeap->size;
    minHeapify(minHeap, 0);
    return temp;
}
```

```c
void insertMinHeap(struct MinHeap* minHeap, struct MinHeapNode* node) {
    ++minHeap->size;
    int i = minHeap->size - 1;
    while (i && node->freq < minHeap->array[(i - 1) / 2]->freq) {
        minHeap->array[i] = minHeap->array[(i - 1) / 2];
        i = (i - 1) / 2;
    }
    minHeap->array[i] = node;
}

void buildMinHeap(struct MinHeap* minHeap) {
    int n = minHeap->size - 1;
    for (int i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}

struct MinHeap* createAndBuildMinHeap(char data[], int freq[], int size) {
    struct MinHeap* minHeap = createMinHeap(size);
    for (int i = 0; i < size; ++i)
        minHeap->array[i] = newNode(data[i], freq[i]);
    minHeap->size = size;
    buildMinHeap(minHeap);
    return minHeap;
}

struct MinHeapNode* buildHuffmanTree(char data[], int freq[], int size) {
    struct MinHeapNode *left, *right, *top;
    struct MinHeap* minHeap = createAndBuildMinHeap(data, freq, size);

    while (!isSizeOne(minHeap)) {
        left = extractMin(minHeap);
        right = extractMin(minHeap);

        top = newNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;

        insertMinHeap(minHeap, top);
    }
    return extractMin(minHeap);
}
```

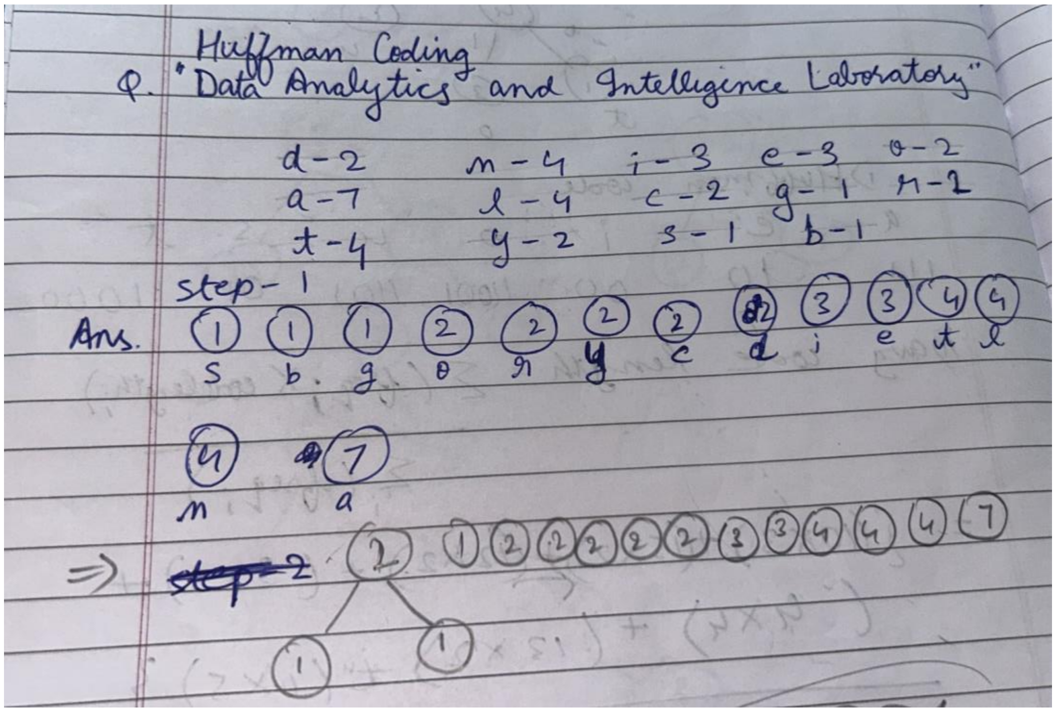
```c
void printCodes(struct MinHeapNode* root, int arr[], int top) {
    if (root->left) {
        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }

    if (root->right) {
        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }

    if (!root->left && !root->right) {
        printf("%c: ", root->data);
        for (int i = 0; i < top; ++i)
            printf("%d", arr[i]);
        printf("\n");
    }
}

int main() {
    struct MinHeapNode* root =
        buildHuffmanTree(data, freq, SIZE);

    int arr[MAX_TREE_HT], top = 0;
    printCodes(root, arr, top);

    return 0;
}
```

OUTPUT:



```
C:\Users\mrgns\Desktop\daa\task6>gcc huffman.c -o huffman

C:\Users\mrgns\Desktop\daa\task6>huffman
n: 000
l: 001
o: 0100
r: 0101
c: 0110
b: 01110
g: 01111
t: 100
y: 1010
s: 10110
d: 10111
e: 1100
i: 1101
a: 111
```

WORKING:



Huffman Coding

Q. "Data Analytics and Intelligence Laboratory"

d - 2	n - 4	i - 3	e - 3	o - 2
a - 7	l - 4	c - 2	g - 1	r - 2
t - 4	y - 2	s - 1	b - 1	

step - 1

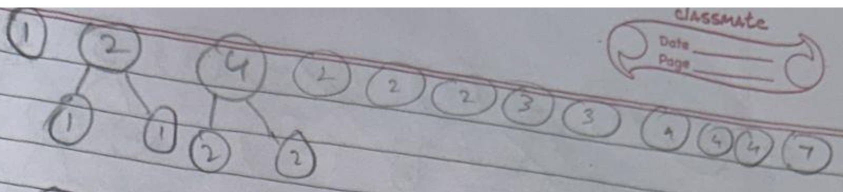
Ans. 1 1 1 2 2 2 2 2 3 3 4 4
s b g o r y c d i e t l

4 7
n a

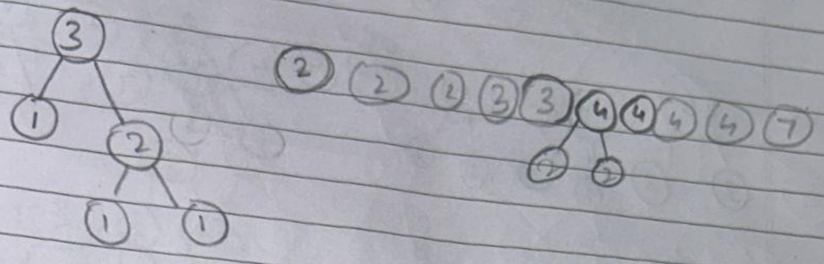
=> ~~step-2~~ 2 1 2 2 2 2 2 3 3 4 4 4 7

$$\begin{tikzcd}
 & 2 \arrow[dl, no head] \arrow[dr, no head] & \\
1 & & 1
\end{tikzcd}$$

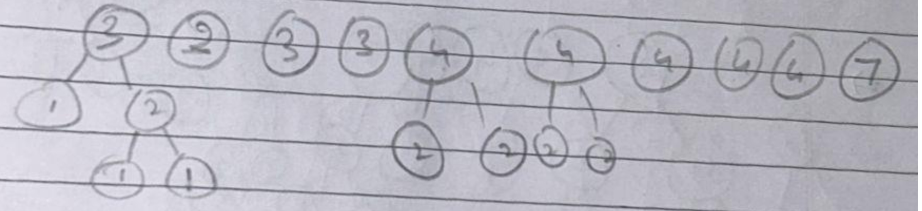
=>



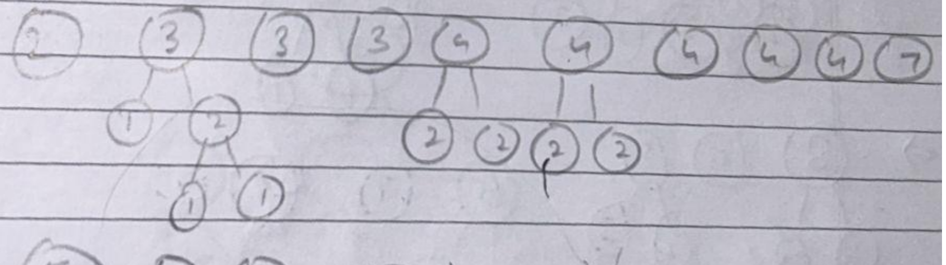
=>



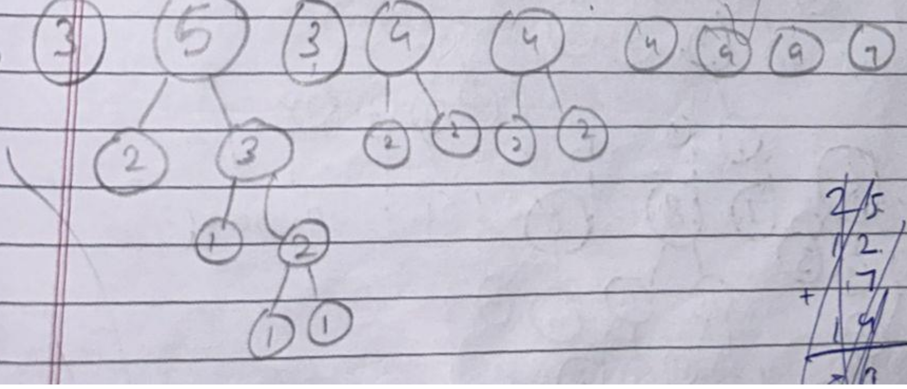
=>



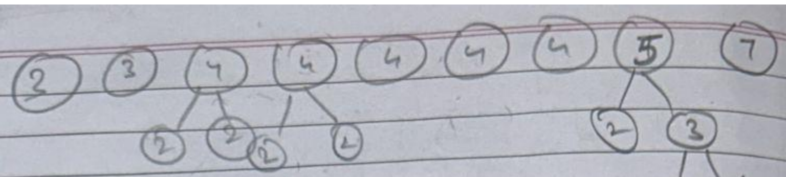
=>

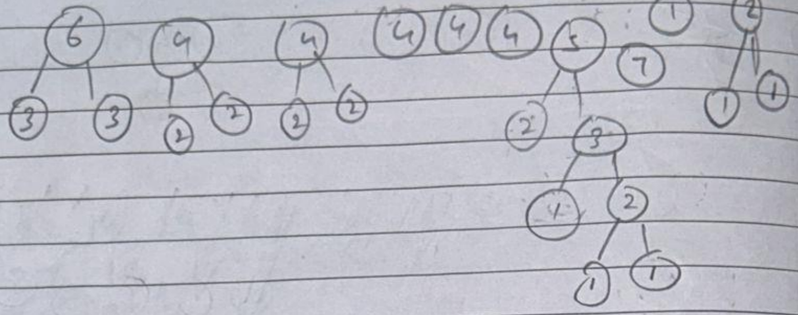


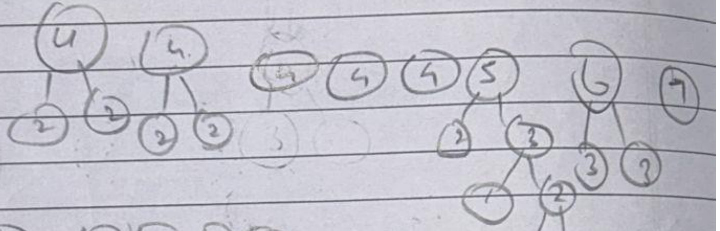
=>

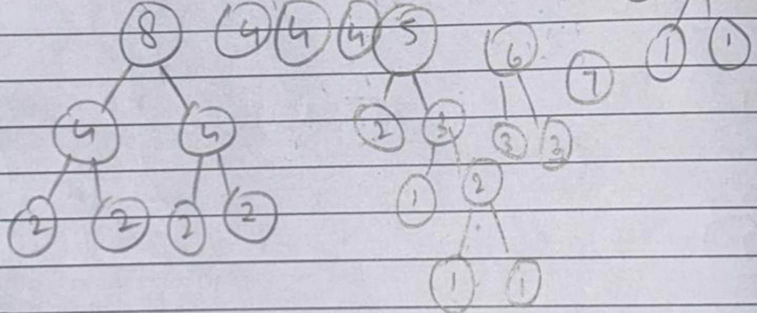


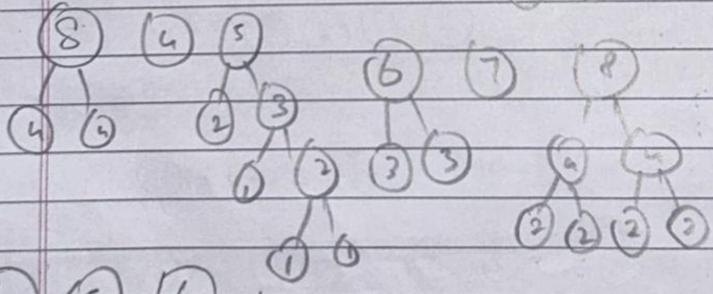
~~25~~
~~12~~
~~17~~
~~+ 19~~

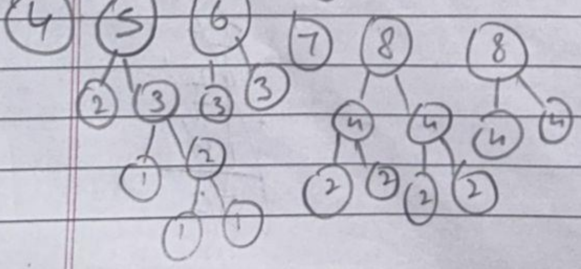
=>


=>


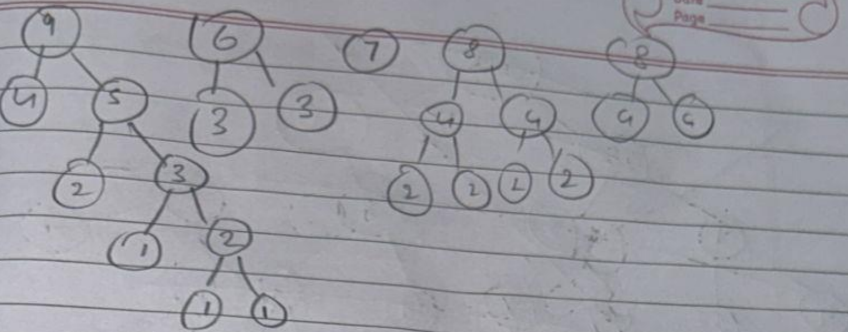
=>


=>


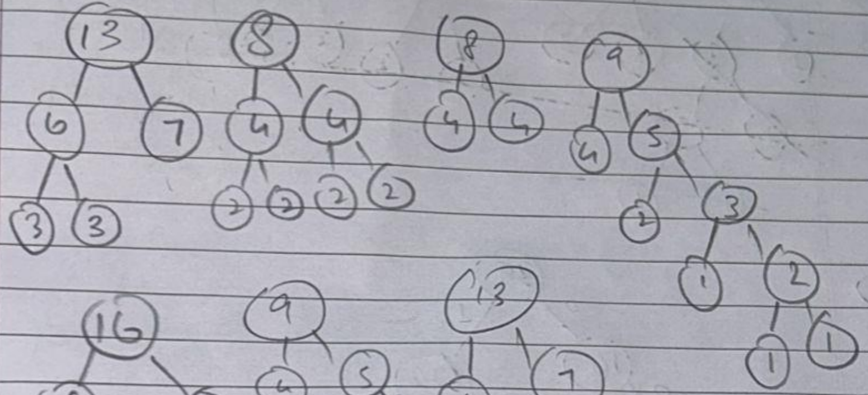
=>


=>


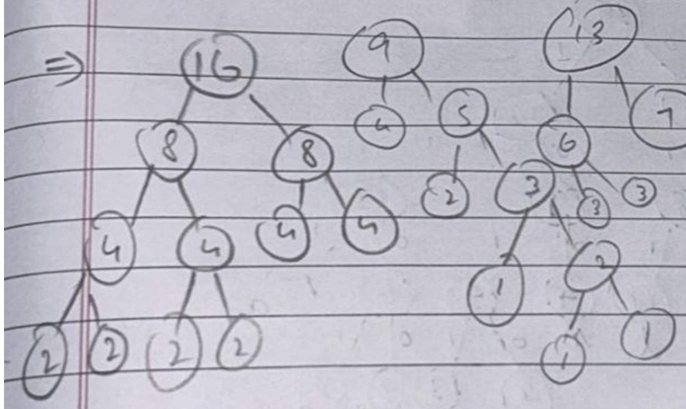
=>



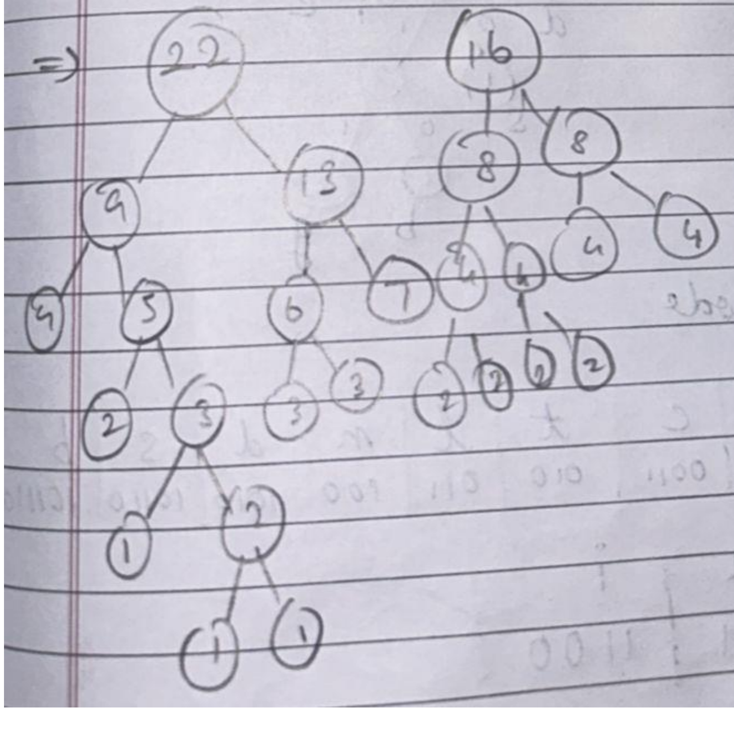
=>



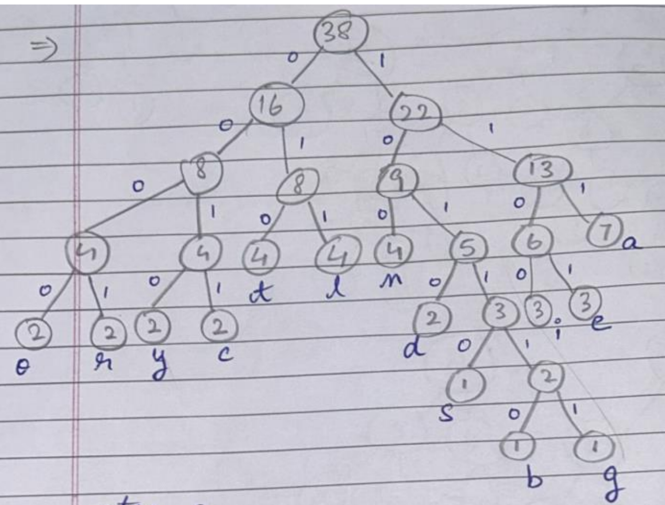
=>



=>



⇒



step 2
Huffman code

θ	r	y	c	t	l	m	d	s	b
0000	0001	0010	0011	010	011	100	1010	10110	101110

g	e	a	i
101111	1101	111	1100

Avg code length :

$$= \frac{\Sigma(freq_i \times codelength_i)}{\Sigma(freq_i)}$$

$$= \frac{(2\times4)+(2\times4)+(2\times4)+(2\times4)+(4\times3)+(4\times3)+(4\times3)+(2\times4)+(1\times5)+(1\times6)+(1\times6)+(3\times4)+(7\times3)+(3\times4)}{2+2+2+2+4+4+4+2+1+1+1+3+3+7}$$

$$= \frac{138}{38} = 3.6315$$

Length of huffman encoded msg :

$$38 \times 3.6315 = \underline{\underline{138}}$$

TIME COMPLEXITY: O(nlogn), n min heap operations and each heap operation takes O(logn) time.

SPACE COMPLEXITY: O(n), from n leaf nodes and n-1 internal nodes.