Experiment Number: 7

NAME: **Kartik Banshi Katkar** ROLLNO: 36

CLASS: TY IT A BATCH: 1

DATE OF PERFORMANCE: 21/09/2023

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Question:  
  
Write a C language code to find Huffman code of the given String. Also calculate the Efficiency. COMMITTEE**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define MAX\_TREE\_HT 100

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(minHeap->capacity \* sizeof(struct MinHeapNode\*));

    return minHeap;

}

void swapMinHeapNode(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) {

        swapMinHeapNode(&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;

}

void insertMinHeap(struct MinHeap\* minHeap, struct MinHeapNode\* minHeapNode) {

    ++minHeap->size;

    int i = minHeap->size - 1;

    while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq) {

        minHeap->array[i] = minHeap->array[(i - 1) / 2];

        i = (i - 1) / 2;

    }

    minHeap->array[i] = minHeapNode;

}

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;

    int n = minHeap->size - 1;

    for (int i = (n - 1) / 2; i >= 0; --i)

        minHeapify(minHeap, i);

    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);

}

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");

    }

}

void calculateEntropyAndEfficiency(struct MinHeapNode\* root, int depth, double\* entropy, double\* averageLength, int\* freqSum) {

    if (root) {

        if (!(root->left) && !(root->right)) {

            double probability = (double)root->freq / (\*freqSum);

            \*entropy += probability \* log2(\*freqSum / (double)root->freq);

            \*averageLength += depth \* probability;

        }

        calculateEntropyAndEfficiency(root->left, depth + 1, entropy, averageLength, freqSum);

        calculateEntropyAndEfficiency(root->right, depth + 1, entropy, averageLength, freqSum);

    }

}

int main() {

      char arr[] = { 'C', 'O', 'M', 'I', 'T', 'E' };//COMMITTEE

    int freq[] = { 1, 1, 2, 1, 2, 2 };

    int size = sizeof(arr) / sizeof(arr[0]);

    // Construct Huffman Tree

    struct MinHeapNode\* root = buildHuffmanTree(arr, freq, size);

    int arrCode[MAX\_TREE\_HT], top = 0;

    // Print Huffman codes

    printf("Huffman Codes:\n");

    printCodes(root, arrCode, top);

    // Calculate entropy, average length, and efficiency

    double entropy = 0.0;

    double averageLength = 0.0;

    int freqSum = 0;

    for (int i = 0; i < size; ++i)

        freqSum += freq[i];

    calculateEntropyAndEfficiency(root, 0, &entropy, &averageLength, &freqSum);

    double efficiency = entropy / averageLength;

    // Print entropy, average length, and efficiency

    printf("Entropy: %lf bits/character\n", entropy);

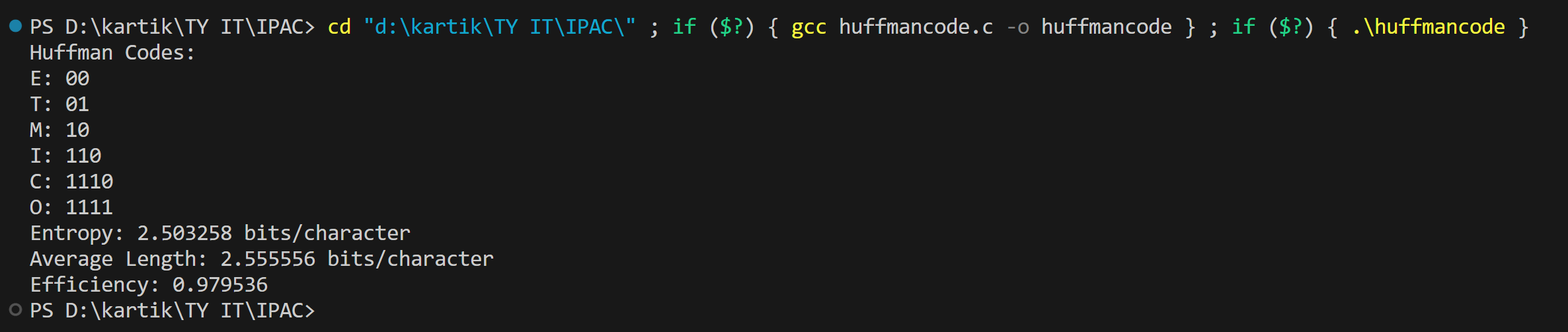
    printf("Average Length: %lf bits/character\n", averageLength);

    printf("Efficiency: %lf\n", efficiency);

    return 0;

}

**output:**

****