DESIGN ANALYSIS OF ALGORITHM

NAME: RIYA KUMARI

SAP ID: 590016221

BATCH: 34

COURSE: B.Tech. CSE (Sem III)

SUBJECT: DAA Lab

DATE: 17 October 2025

SUBMITTED TO: VIRENDRA KUMAR

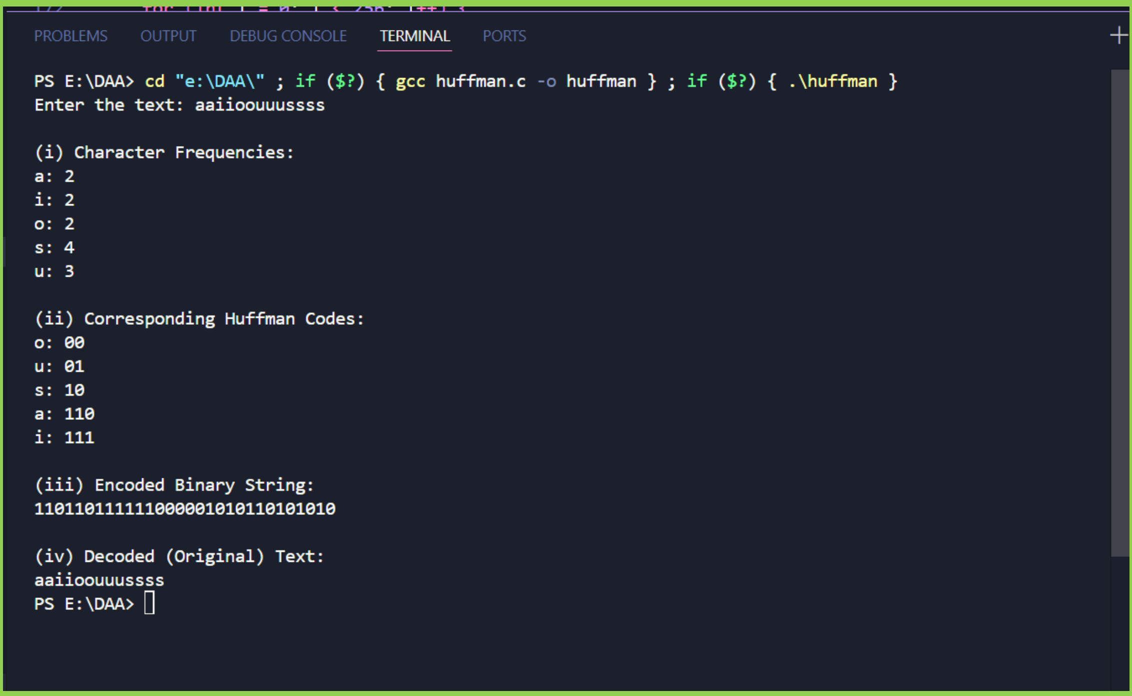
REPOSITORY LINK: https://github.com/Riyakumari1314/DAA-2nd-year.git

1) Objective: To implement Huffman Coding - a data compression algorithm - using the concept of greedy algorithms in C. The program should generate variable-length codes for characters based on their frequencies and demonstrate encoding and decoding of a given input text.
Problem Statement: Write a program that takes an input string. Calculates the frequency of each character. Builds a Huffman Tree based on these frequencies. Generates Huffman Codes (binary codes) for each character. Output: i. Character frequencies ii. Corresponding Huffman codes iii. Encoded binary string iv. Decoded (original) text

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_HT 100
struct HuffNode {
    char ch;
    unsigned freq;
    struct HuffNode *left, *right;
};
struct MinHeap {
    unsigned size;
    unsigned capacity;
    struct HuffNode **array;
};
struct HuffNode* newNode(char ch, unsigned freq) {
    struct HuffNode* temp = (struct HuffNode*)malloc(sizeof(struct HuffNode));
    temp->left = temp->right = NULL;
    temp->ch = ch;
    temp->freq = freq;
    return temp;
}

struct MinHeap* createMinHeap(unsigned capacity) {
    struct MinHeap* heap = (struct MinHeap*)malloc(sizeof(struct MinHeap));
    heap->size = 0;
    heap->capacity = capacity;
    heap->array = (struct HuffNode**)malloc(heap->capacity * sizeof(struct HuffNode*));
    return heap;
}
void swapNode(struct HuffNode** a, struct HuffNode** b) {
    struct HuffNode* t = *a;
    *a = *b;
    *b = t;
}
void minHeapify(struct MinHeap* heap, int idx) {
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;
    if (left < heap->size && heap->array[left]->freq < heap->array[smallest]->freq)
        smallest = left;
    if (right < heap->size && heap->array[right]->freq < heap->array[smallest]->freq)
        smallest = right;
    if (smallest != idx) {
        swapNode(&heap->array[smallest], &heap->array[idx]);
        minHeapify(heap, smallest);
    }
}
int isSizeOne(struct MinHeap* heap) {
    return (heap->size == 1);
}
struct HuffNode* extractMin(struct MinHeap* heap) {
    struct HuffNode* temp = heap->array[0];
    heap->array[0] = heap->array[heap->size - 1];
    --heap->size;
    minHeapify(heap, 0);
    return temp;
}

void insertMinHeap(struct MinHeap* heap, struct HuffNode* node) {
    ++heap->size;
    int i = heap->size - 1;
    while (i && node->freq < heap->array[(i - 1) / 2]->freq) {
        heap->array[i] = heap->array[(i - 1) / 2];
        i = (i - 1) / 2;
    }
    heap->array[i] = node;
}
void buildMinHeap(struct MinHeap* heap) {
    int n = heap->size - 1;
    for (int i = (n - 1) / 2; i >= 0; --i)
        minHeapify(heap, i);
}
int isLeaf(struct HuffNode* root) {
    return !(root->left) && !(root->right);
}
struct MinHeap* createAndBuildMinHeap(char ch[], int freq[], int size) {
    struct MinHeap* heap = createMinHeap(size);
    for (int i = 0; i < size; ++i)
        heap->array[i] = newNode(ch[i], freq[i]);
    heap->size = size;
    buildMinHeap(heap);
    return heap;
}
struct HuffNode* buildHuffTree(char ch[], int freq[], int size) {
    struct HuffNode *left, *right, *top;
    struct MinHeap* heap = createAndBuildMinHeap(ch, freq, size);
    while (!isSizeOne(heap)) {
        left = extractMin(heap);
        right = extractMin(heap);
        top = newNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        insertMinHeap(heap, top);
    }
    return extractMin(heap);
}
void printCodes(struct HuffNode* root, int arr[], int top, char codes[256][MAX_HT]) {
    if (root->left) {
        arr[top] = 0;
        printCodes(root->left, arr, top + 1, codes);
    }
    if (root->right) {
        arr[top] = 1;
        printCodes(root->right, arr, top + 1, codes);
    }
    if (isLeaf(root)) {
        printf("%c: ", root->ch);
        for (int i = 0; i < top; i++)
            printf("%d", arr[i]);
        printf("\n");

        for (int i = 0; i < top; i++)
            codes[(unsigned char)root->ch][i] = arr[i] + '0';
        codes[(unsigned char)root->ch][top] = '\0';
    }
}
void HuffmanCodes(char ch[], int freq[], int size, char codes[256][MAX_HT]) {
    struct HuffNode* root = buildHuffTree(ch, freq, size);
    int arr[MAX_HT], top = 0;
    printCodes(root, arr, top, codes);
}
int main() {
    char text[100];
    printf("Enter the text: ");
    fgets(text, sizeof(text), stdin);
    text[strcspn(text, "\n")] = '\0';

    int freq[256] = {0};
    for (int i = 0; text[i] != '\0'; i++)
```

```
        freq[(unsigned char)text[i]]++;
    char ch[256];
    int frequency[256], size = 0;
    printf("\n(i) Character Frequencies:\n");
    for (int i = 0; i < 256; i++) {
        if (freq[i]) {
            ch[size] = (char)i;
            frequency[size] = freq[i];
            printf("%c: %d\n", i, freq[i]);
            size++;
        }
    }
    char codes[256][MAX_HT] = {0};
    printf("\n(ii) Corresponding Huffman Codes:\n");
    HuffmanCodes(ch, frequency, size, codes);
    printf("\n(iii) Encoded Binary String:\n");
    for (int i = 0; text[i] != '\0'; i++)
        printf("%s", codes[(unsigned char)text[i]]);
    printf("\n");
    printf("\n(iv) Decoded (Original) Text:\n%s\n", text);
    return 0;
}
```



```
PS E:\DAA> cd "e:\DAA\" ; if ($?) { gcc huffman.c -o huffman } ; if ($?) { .\huffman }
Enter the text: aaiioouuussss

(i) Character Frequencies:
a: 2
i: 2
o: 2
s: 4
u: 3

(ii) Corresponding Huffman Codes:
o: 00
u: 01
s: 10
a: 110
i: 111

(iii) Encoded Binary String:
110110111111000001010110101010

(iv) Decoded (Original) Text:
aaiioouuussss
PS E:\DAA> 
```