



SCHOOL OF  
COMPUTING

**Devadharshan.S  
CH.SC.U4CSE24113**

**Week – 7**

**Date - 19/02/2026**

**Design and Analysis of Algorithm(23CSE211)**

## 1. Huffman Coding

### Code:

```
//CH.SC.U4CSE24113
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_TREE_HT 100
#define MAX_CHAR 256
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);
    }
}

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;
}

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

int isLeaf(struct MinHeapNode* root) {
    return !(root->left) && !(root->right);
}

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 (isLeaf(root)) {
        printf(" %c | %3d | ", root->data, root->freq);
        for (int i = 0; i < top; ++i)
            printf("%d", arr[i]);
        printf("\n");
    }
}

void calculateSpaceSaving(struct MinHeapNode* root, int arr[], int top, int*
totalOriginalBits, int* totalCompressedBits) {
    if (root->left) {
        arr[top] = 0;
        calculateSpaceSaving(root->left, arr, top + 1, totalOriginalBits,
totalCompressedBits);
    }
    if (root->right) {
        arr[top] = 1;
    }
}

```

```

        calculateSpaceSaving(root->right, arr, top + 1, totalOriginalBits,
totalCompressedBits);
    }
    if (isLeaf(root)) {
        *totalOriginalBits += root->freq * 8; // Assuming 8 bits per character
        *totalCompressedBits += root->freq * top;
    }
}
void HuffmanCodes(char data[], int freq[], int size) {
    struct MinHeapNode *left, *right, *top;
    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);
    while (minHeap->size != 1) {
        left = extractMin(minHeap);
        right = extractMin(minHeap);
        top = newNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        insertMinHeap(minHeap, top);
    }
    int arr[MAX_TREE_HT], topIdx = 0;
    printf("\nChar | Freq | Huffman Code\n");
    printf("-----\n");
    struct MinHeapNode* root = extractMin(minHeap);
    printCodes(root, arr, topIdx);
    int totalOriginalBits = 0, totalCompressedBits = 0;
    calculateSpaceSaving(root, arr, 0, &totalOriginalBits,
&totalCompressedBits);
    printf("\nSpace Saving Analysis:\n");
    printf("-----\n");
    printf("Original size (8 bits per character): %d bits\n",
totalOriginalBits);
    printf("Compressed size (Huffman coding): %d bits\n", totalCompressedBits);
    printf("Space saved: %d bits (%.2f%%)\n",
        totalOriginalBits - totalCompressedBits,
        ((float)(totalOriginalBits - totalCompressedBits) /
totalOriginalBits) * 100);
}
int main() {
    printf("CH.SC.U4CCSE24113\n");
    char str[1000];
    int freq[MAX_CHAR] = {0};
    printf("Enter the string: ");
    scanf("%s", str);
    for (int i = 0; str[i] != '\0'; i++) {
        freq[(unsigned char)str[i]]++;
    }
    char unique_chars[MAX_CHAR];
    int unique_freqs[MAX_CHAR];
    int n = 0;
    for (int i = 0; i < MAX_CHAR; i++) {

```

```

        if (freq[i] > 0) {
            unique_chars[n] = (char)i;
            unique_freqs[n] = freq[i];
            n++;
        }
    }
HuffmanCodes(unique_chars, unique_freqs, n);
return 0;
}

```

## Output:

```

devadharshan@devadharshan-HP-Pavilion-Laptop-15-eg3xxx:/media/devadharshan/New V
olume/DAA/Lab/Week-7$ gcc Huffman_Coding.c -o Huffman
devadharshan@devadharshan-HP-Pavilion-Laptop-15-eg3xxx:/media/devadharshan/New V
olume/DAA/Lab/Week-7$ ./Huffman
CH.SC.U4CCSE24113
Enter the string: DATAANALYTICSANDINTELLIGENCELABORATORY

Char | Freq | Huffman Code
-----
N   |   4   | 000
L   |   4   | 001
O   |   2   | 0100
R   |   2   | 0101
C   |   2   | 0110
B   |   1   | 01110
G   |   1   | 01111
T   |   4   | 100
Y   |   2   | 1010
S   |   1   | 10110
D   |   2   | 10111
E   |   3   | 1100
I   |   3   | 1101
A   |   7   | 111

Space Saving Analysis:
-----
Original size (8 bits per character): 304 bits
Compressed size (Huffman coding): 138 bits
Space saved: 166 bits (54.61%)
devadharshan@devadharshan-HP-Pavilion-Laptop-15-eg3xxx:/media/devadharshan/New V
olume/DAA/Lab/Week-7$ 

```

**Time Complexity:**  $O(n \log n)$

**Justification:** The algorithm builds a min-heap in  $O(n)$  time. It then performs  $n-1$  iterations to extract the two smallest nodes and insert a new internal node. Each heap operation (insertion and extraction) takes  $O(\log n)$  time, making the total construction time  $O(n \log n)$ . Tree traversal for printing codes is linear,  $O(n)$ .

**Space Complexity:**  $O(n)$

**Justification:** The space is primarily determined by the Huffman tree and the min-heap, both of which store  $O(n)$  nodes. Additionally, the recursion stack for generating codes depends on the tree height, which is  $O(n)$  in the worst-case (highly skewed tree) and  $O(\log n)$  in the average case. No other significant auxiliary structures are used.