LAB TASK – 8

1. Greedy Approach: Implementation of Huffman code

**Code :**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

typedef struct Node {

    char letter;

    int count;

    struct Node\* left;

    struct Node\* right;

} Node;

typedef struct Heap {

    int length;

    Node\* nodes[MAX];

} Heap;

Node\* makeNode(char letter, int count) {

    Node\* node = (Node\*)malloc(sizeof(Node));

    node->letter = letter;

    node->count = count;

    node->left = node->right = NULL;

    return node;

}

void swap(Node\*\* a, Node\*\* b) {

    Node\* temp = \*a;

    \*a = \*b;

    \*b = temp;

}

void heapify(Heap\* heap, int idx) {

    int min = idx;

    int left = 2 \* idx + 1;

    int right = 2 \* idx + 2;

    if (left < heap->length && heap->nodes[left]->count < heap->nodes[min]->count)

        min = left;

    if (right < heap->length && heap->nodes[right]->count < heap->nodes[min]->count)

        min = right;

    if (min != idx) {

        swap(&heap->nodes[min], &heap->nodes[idx]);

        heapify(heap, min);

    }

}

Node\* getMin(Heap\* heap) {

    Node\* temp = heap->nodes[0];

    heap->nodes[0] = heap->nodes[heap->length - 1];

    heap->length--;

    heapify(heap, 0);

    return temp;

}

void insertHeap(Heap\* heap, Node\* node) {

    heap->length++;

    int i = heap->length - 1;

    heap->nodes[i] = node;

    while (i && heap->nodes[i]->count < heap->nodes[(i - 1) / 2]->count) {

        swap(&heap->nodes[i], &heap->nodes[(i - 1) / 2]);

        i = (i - 1) / 2;

    }

}

Heap\* buildHeap(char letters[], int freq[], int n) {

    Heap\* heap = (Heap\*)malloc(sizeof(Heap));

    heap->length = 0;

    for (int i = 0; i < n; i++) {

        heap->nodes[heap->length] = makeNode(letters[i], freq[i]);

        heap->length++;

    }

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

        heapify(heap, i);

    return heap;

}

int isLeaf(Node\* node) {

    return !(node->left) && !(node->right);

}

void display(Node\* root, char buffer[], int depth, int\* size) {

    if (root->left) {

        buffer[depth] = '0';

        display(root->left, buffer, depth + 1, size);

    }

    if (root->right) {

        buffer[depth] = '1';

        display(root->right, buffer, depth + 1, size);

    }

    if (isLeaf(root)) {

        buffer[depth] = '\0';

        printf("%c : %s\n", root->letter, buffer);

        \*size += root->count \* depth;

    }

}

Node\* constructHuffman(char symbols[], int freq[], int n) {

    Heap\* heap = buildHeap(symbols, freq, n);

    while (heap->length > 1) {

        Node\* left = getMin(heap);

        Node\* right = getMin(heap);

        Node\* merged = makeNode('$', left->count + right->count);

        merged->left = left;

        merged->right = right;

        insertHeap(heap, merged);

    }

    return getMin(heap);

}

int main() {

    int n;

    printf("Enter number of characters: ");

    scanf("%d", &n);

    char letters[n];

    int freq[n];

    printf("Enter characters: ");

    for (int i = 0; i < n; i++) {

        scanf(" %c", &letters[i]);

    }

    printf("Enter their frequencies: ");

    for (int i = 0; i < n; i++) {

        scanf("%d", &freq[i]);

    }

    Node\* root = constructHuffman(letters, freq, n);

    char c[MAX];

    int total = 0;

    printf("Huffman Codes:\n");

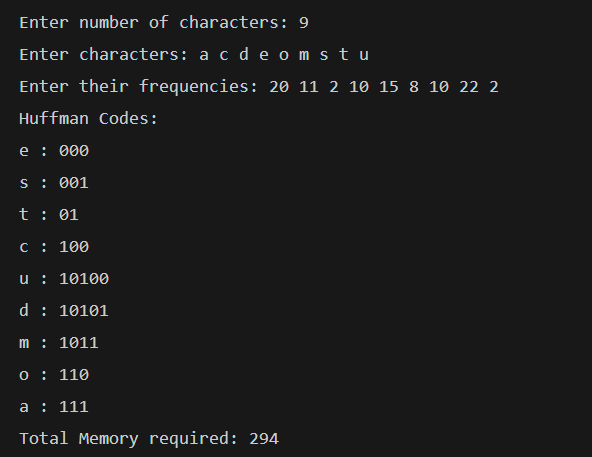
    display(root,c, 0, &total);

    printf("Total Memory required: %d\n", total);

    return 0;

}

**Output :**

****