1. Greedy Approach: Implementation of Huffman code  
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

typedef struct Node {

char ch;

int freq;

struct Node\* left;

struct Node\* right;

} Node;

typedef struct MinHeap {

int size;

Node\* array[MAX];

} MinHeap;

Node\* newNode(char ch, int freq) {

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

temp->ch = ch;

temp->freq = freq;

temp->left = temp->right = NULL;

return temp;

}

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

Node\* t = \*a;

\*a = \*b;

\*b = t;

}

void minHeapify(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);

}

}

Node\* extractMin(MinHeap\* heap) {

Node\* temp = heap->array[0];

heap->array[0] = heap->array[heap->size - 1];

heap->size--;

minHeapify(heap, 0);

return temp;

}

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

heap->size++;

int i = heap->size - 1;

heap->array[i] = node;

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

swapNode(&heap->array[i], &heap->array[(i-1)/2]);

i = (i-1)/2;

}

}

MinHeap\* createMinHeap(char chars[], int freq[], int n) {

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

heap->size = 0;

for (int i = 0; i < n; i++) {

heap->array[heap->size] = newNode(chars[i], freq[i]);

heap->size++;

}

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

minHeapify(heap, i);

return heap;

}

int isLeaf(Node\* node) {

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

}

void printCodesAndSize(Node\* root, char code[], int top, int\* totalSize) {

if (root->left) {

code[top] = '0';

printCodesAndSize(root->left, code, top + 1, totalSize);

}

if (root->right) {

code[top] = '1';

printCodesAndSize(root->right, code, top + 1, totalSize);

}

if (isLeaf(root)) {

code[top] = '\0';

printf("%c : %s\n", root->ch, code);

\*totalSize += root->freq \* top;

}

}

Node\* buildHuffmanTree(char chars[], int freq[], int n) {

MinHeap\* heap = createMinHeap(chars, freq, n);

while (heap->size > 1) {

Node\* left = extractMin(heap);

Node\* right = extractMin(heap);

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

top->left = left;

top->right = right;

insertHeap(heap, top);

}

return extractMin(heap);

}

int main() {

char chars[] = {'a','c','d','e','o','m','s','t','u'};

int freq[] = {20,11,2,10,15,8,10,22,2};

int n = sizeof(chars)/sizeof(chars[0]);

Node\* root = buildHuffmanTree(chars, freq, n);

char code[MAX];

int totalSize = 0;

printf("Huffman Codes:\n");

printCodesAndSize(root, code, 0, &totalSize);

printf("Total Compressed File Size (in bits) = %d\n", totalSize);

return 0;

}

Output:

