**ITC Tutorial - 09 Home Assignment**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Name: Chandana Ramesh Galgali Roll No.: 16010422234 Batch: B-3**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Concept chosen from syllabus: Module - 2 : Huffman Coding**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Program:**

#include <stdio.h>

#include <stdlib.h>

// A Huffman tree node

struct MinHeapNode {

char data; // One of the input characters

unsigned freq; // Frequency of the character

struct MinHeapNode \*left, \*right; // Left and right child

};

// A Min Heap: Collection of min-heap (or Huffman tree) nodes

struct MinHeap {

unsigned size; // Current size of min heap

unsigned capacity; // Capacity of min heap

struct MinHeapNode\*\* array; // Array of minheap node pointers

};

// A utility function to create a new min heap node

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;

}

// A utility function to create a min heap of given capacity

struct MinHeap\* createMinHeap(unsigned capacity) {

struct MinHeap\* minHeap = (struct MinHeap\*)malloc(sizeof(struct MinHeap));

minHeap->size = 0; // current size is 0

minHeap->capacity = capacity;

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

return minHeap;

}

// A utility function to swap two min heap nodes

void swapMinHeapNode(struct MinHeapNode\*\* a, struct MinHeapNode\*\* b) {

struct MinHeapNode\* t = \*a;

\*a = \*b;

\*b = t;

}

// The standard minHeapify function.

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);

}

}

// A utility function to check if size of heap is 1 or not

int isSizeOne(struct MinHeap\* minHeap) {

return (minHeap->size == 1);

}

// A standard function to extract minimum value node from heap

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;

}

// A utility function to insert a new node to Min Heap

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;

}

// A standard function to build min heap

void buildMinHeap(struct MinHeap\* minHeap) {

int n = minHeap->size - 1;

int i;

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

minHeapify(minHeap, i);

}

// A utility function to print an array of size n

void printArr(int arr[], int n) {

int i;

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

printf("%d", arr[i]);

printf("\n");

}

// Utility function to check if this node is leaf

int isLeaf(struct MinHeapNode\* root) {

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

}

// Creates a min heap of capacity equal to size and inserts all character of data[] in min heap. Initially size of min heap is equal to capacity

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;

buildMinHeap(minHeap);

return minHeap;

}

// The main function that builds Huffman tree

struct MinHeapNode\* buildHuffmanTree(char data[], int freq[], int size) {

struct MinHeapNode \*left, \*right, \*top;

// Step 1: Create a min heap of capacity equal to size. Initially, there are modes equal to size.

struct MinHeap\* minHeap = createAndBuildMinHeap(data, freq, size);

// Iterate while size of heap doesn't become 1

while (!isSizeOne(minHeap)) {

// Step 2: Extract the two minimum freq items from min heap

left = extractMin(minHeap);

right = extractMin(minHeap);

// Step 3: Create a new internal node with frequency equal to the sum of the two nodes frequencies. Make the two extracted node as left and right children of this new node. Add this node to the min heap

// '$' is a special value for internal nodes, not used

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

top->left = left;

top->right = right;

insertMinHeap(minHeap, top);

}

// Step 4: The remaining node is the root node and the tree is complete.

return extractMin(minHeap);

}

// Prints Huffman codes using the Huffman tree built above

void printCodes(struct MinHeapNode\* root, int arr[], int top) {

// Assign 0 to left edge and recur

if (root->left) {

arr[top] = 0;

printCodes(root->left, arr, top + 1);

}

// Assign 1 to right edge and recur

if (root->right) {

arr[top] = 1;

printCodes(root->right, arr, top + 1);

}

// If this is a leaf node, then print the character and its code from arr[]

if (isLeaf(root)) {

printf("%c: ", root->data);

printArr(arr, top);

}

}

// The main function that builds a Huffman Tree and print codes by traversing the built Huffman Tree

int main() {

int size;

printf("Enter the number of characters: ");

scanf("%d", &size);

char arr[size];

int freq[size];

// Taking input from user

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

printf("Enter character and frequency respectively for element %d: ", i + 1);

scanf(" %c %d", &arr[i], &freq[i]);

}

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

int codes[size], top = 0;

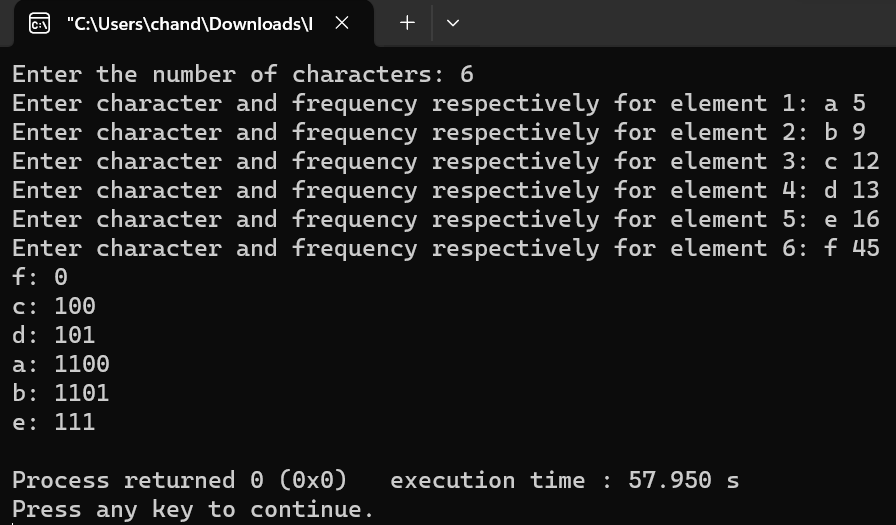
printCodes(root, codes, top);

return 0;

}

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

**Output:**

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**