code for Huffman coading

#include <stdio.h>

#include <stdlib.h>

#define MAX\_TREE\_HT 100

struct MinHeapNode {

    char data;

    unsigned freq;

    struct MinHeapNode \*left, \*right;

};

struct MinHeap {

    // Current size of min heap

    unsigned size;

    // capacity of min heap

    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;

}

// 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));

    // current size is 0

    minHeap->size = 0;

    minHeap->capacity = capacity;

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

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

    return minHeap;

}

// 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);

}

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;

}

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);

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

        top->left = left;

        top->right = right;

        insertMinHeap(minHeap, top);

    }

.

    return extractMin(minHeap);

}

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

    // it contains one of the input

    // characters, print the character

    // and its code from arr[]

    if (isLeaf(root)) {

        printf("%c: ", root->data);

        printArr(arr, top);

    }

}

void HuffmanCodes(char data[], int freq[], int size)

{

    // Construct Huffman Tree

    struct MinHeapNode\* root

        = buildHuffmanTree(data, freq, size);

    int arr[MAX\_TREE\_HT], top = 0;

    printCodes(root, arr, top);

}

// Driver code

int main()

{

    char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };

    int freq[] = { 5, 9, 12, 13, 16, 45 };

    int size = sizeof(arr) / sizeof(arr[0]);

    HuffmanCodes(arr, freq, size);

    return 0;

}