#include <stdio.h>

#include <stdlib.h>

#define MAX\_TREE\_HT 50

struct MinHNode {

char item;

unsigned freq;

struct MinHNode \*left, \*right;

};

struct MinHeap {

unsigned size;

unsigned capacity;

struct MinHNode \*\*array;

};

// Create nodes

struct MinHNode \*newNode(char item, unsigned freq) {

struct MinHNode \*temp = (struct MinHNode \*)malloc(sizeof(struct MinHNode));

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

temp->item = item;

temp->freq = freq;

return temp;

}

// Create min heap

struct MinHeap \*createMinH(unsigned capacity) {

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

minHeap->size = 0;

minHeap->capacity = capacity;

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

return minHeap;

}

// Function to swap

void swapMinHNode(struct MinHNode \*\*a, struct MinHNode \*\*b) {

struct MinHNode \*t = \*a;

\*a = \*b;

\*b = t;

}

// Heapify

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) {

swapMinHNode(&minHeap->array[smallest], &minHeap->array[idx]);

minHeapify(minHeap, smallest);

}

}

// Check if size if 1

int checkSizeOne(struct MinHeap \*minHeap) {

return (minHeap->size == 1);

}

// Extract min

struct MinHNode \*extractMin(struct MinHeap \*minHeap) {

struct MinHNode \*temp = minHeap->array[0];

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

--minHeap->size;

minHeapify(minHeap, 0);

return temp;

}

// Insertion function

void insertMinHeap(struct MinHeap \*minHeap, struct MinHNode \*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;

int i;

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

minHeapify(minHeap, i);

}

int isLeaf(struct MinHNode \*root) {

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

}

struct MinHeap \*createAndBuildMinHeap(char item[], int freq[], int size) {

struct MinHeap \*minHeap = createMinH(size);

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

minHeap->array[i] = newNode(item[i], freq[i]);

minHeap->size = size;

buildMinHeap(minHeap);

return minHeap;

}

struct MinHNode \*buildHuffmanTree(char item[], int freq[], int size) {

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

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

while (!checkSizeOne(minHeap)) {

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 printHCodes(struct MinHNode \*root, int arr[], int top) {

if (root->left) {

arr[top] = 0;

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

}

if (root->right) {

arr[top] = 1;

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

}

if (isLeaf(root)) {

printf(" %c | ", root->item);

printArray(arr, top);

}

}

// Wrapper function

void HuffmanCodes(char item[], int freq[], int size) {

struct MinHNode \*root = buildHuffmanTree(item, freq, size);

int arr[MAX\_TREE\_HT], top = 0;

printHCodes(root, arr, top);

}

// Print the array

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

int i;

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

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

printf("\n");

}

int main() {

char arr[] = {'A', 'B', 'C', 'D'};

int freq[] = {5, 1, 6, 3};

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

printf(" Char | Huffman code ");

printf("\n--------------------\n");

HuffmanCodes(arr, freq, size);

}