#include <stdio.h>

#include <stdlib.h>

#define MAX\_TREE\_NODES 255

struct MinHeapNode {

char data;

unsigned freq;

struct MinHeapNode \*left, \*right;

};

struct MinHeap {

unsigned size;

unsigned capacity;

struct MinHeapNode\* array[MAX\_TREE\_NODES];

};

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;

}

struct MinHeap\* createMinHeap(unsigned capacity) {

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

minHeap->size = 0;

minHeap->capacity = capacity;

return minHeap;

}

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

struct MinHeapNode\* t = \*a;

\*a = \*b;

\*b = t;

}

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);

}

}

int isSizeOne(struct MinHeap\* minHeap) {

return (minHeap->size == 1);

}

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;

}

void insertMinHeap(struct MinHeap\* minHeap, struct MinHeapNode\* minHeapNode) {

int i;

++minHeap->size;

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;

}

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

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

struct MinHeap\* minHeap = createMinHeap(size);

int i;

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

insertMinHeap(minHeap, newNode(data[i], freq[i]));

while (!isSizeOne(minHeap)) {

\*counter += 1; // Increment counter for each iteration

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) {

int i;

if (root->left) {

arr[top] = 0;

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

}

if (root->right) {

arr[top] = 1;

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

}

if (!(root->left) && !(root->right)) {

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

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

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

printf("\n");

}

}

int main() {

char data[MAX\_TREE\_NODES];

int freq[MAX\_TREE\_NODES];

int size, counter = 0,top=0,i;

int arr[MAX\_TREE\_NODES];

struct MinHeapNode\* root;

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

scanf("%d", &size);

printf("Enter characters and their frequencies:\n");

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

printf("Character %d: ", i + 1);

scanf(" %c", &data[i]);

printf("Frequency %c: ", data[i]);

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

}

root = buildHuffmanTree(data, freq, size, &counter);

printCodes(root, arr, top);

printf("Time complexity: O(n log n) where n is the number of unique characters.\n");

printf("Number of iterations: %d\n", counter);

return 0;

}