

**数据结构课程实验报告**

**（3）**

**姓名：顾宬树**

**学号：2135061913**

**专业：通信工程**

**完成日期：2023.4.3**

**目录**

[目录 - 1 -](#_Toc434170429)

[1 设计要求 - 2 -](#_Toc434170430)

[2 程序功能框图 - 2 -](#_Toc434170431)

[3 数据结构说明 - 2 -](#_Toc434170432)

[4 重要算法核心代码 - 2 -](#_Toc434170433)

[5 测试运行界面 - 2 -](#_Toc434170434)

[6 完整源程序 - 3 -](#_Toc434170435)

**数据结构实验三（二叉树）**

**<“哈夫曼树”的设计与实现> 设计要求**

【问题描述】

假设有一段电文由字符集 {A, B, C, D, E, F, G, H} 中的字符组成，各字符在电文中出现的频率由对应次数集 {5，29，7，8，14，23，3，11} 中的数字表示，请设计各字符的哈夫曼编码。

【基本要求】

应包含以下几方面的功能：

1. 设计哈夫曼树。具体构造方法如下：以字符集{A, B, C, D, E, F, G, H} 中的字符作为叶子结点，以各字符在次数集 {5，29，7，8，14，23，3，11} 中对应的次数作为各叶子结点的权值构造一棵哈夫曼树。
2. 设计哈夫曼编码。按照构造出来的哈夫曼树，规定哈夫曼树的左分支为0，右分支为1，则从根结点到每个叶子结点所经过的分支对应的0和1组成的序列便为该结点对应字符的哈夫曼编码。
3. 依次求出每个字符的哈夫曼编码并输出。

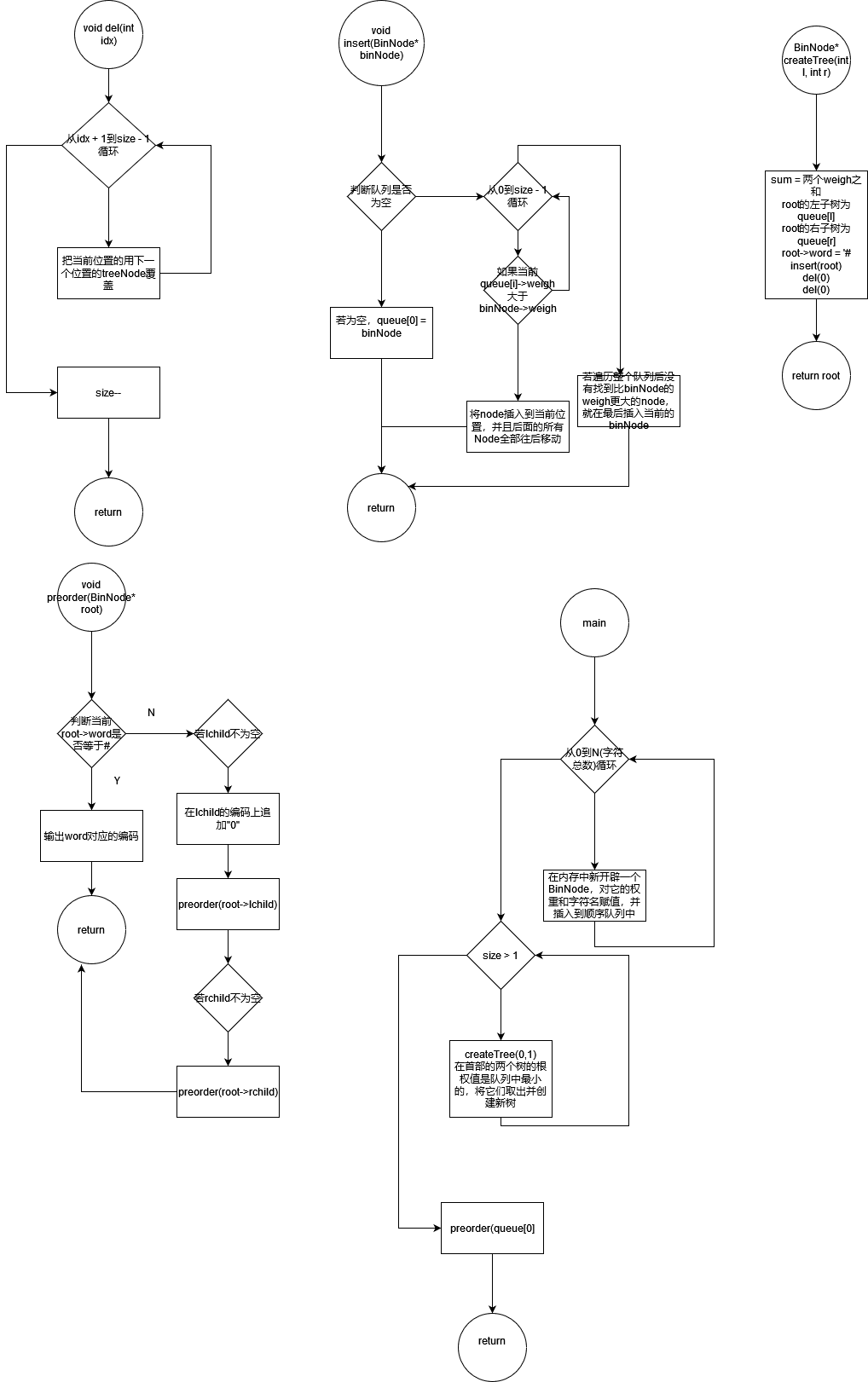
【扩展功能】

从键盘上分别输入哈夫曼编码字符的个数以及每个字符对应的权值，程序执行中请一步一步依次显示出哈夫曼树的构造过程，最后输出每个权值对应的哈夫曼编码。

【实现方法】

选择合适的存储结构，要便于从叶子结点找到双亲结点。

【程序功能框图】



【数据结构的说明】

typedef struct BinNode\_ {

char word;

int weigh;

char\* code;

struct BinNode\_\* lchild;

struct BinNode\_\* rchild;

}BinNode;

BinNode结构体的成员包括：

* word：一个字符变量，表示存储在该节点中的数据。
* weigh：一个整数变量，表示该节点的权重或频率，通常在哈夫曼编码中使用。
* code：一个字符指针，可以用来存储与该节点相关联的哈夫曼编码。
* lchild和rchild：分别指向该节点的左子节点和右子节点的指针。

【重要算法核心代码】

void del(int idx) {

for (int i = idx + 1; i <= size - 1; i++) {

queue[i - 1] = queue[i];

}

size--;

}

void insert(BinNode\* binNode) {

if (size == 0) {

queue[size] = binNode;

size++;

return;

}

for (int i = 0; i <= size - 1; i++) {

if (binNode->weigh < queue[i]->weigh) {

for (int j = size; j >= i + 1; j--) {

queue[j] = queue[j - 1];

}

queue[i] = binNode;

size++;

return;

}

}

queue[size] = binNode;

size++;

}

//将queue中索引为l和r的两颗树取出并合并成一棵新的树

BinNode\* createTree(int l, int r) {

int sum = queue[l]->weigh + queue[r]->weigh;

BinNode\* root = (BinNode\*)malloc(sizeof(BinNode));

init(root);

root->lchild = queue[l];

root->rchild = queue[r];

root->weigh = sum;

root->word = '#';

insert(root);

del(0);

del(0);

return root;

}

void preorder(BinNode\* root) {

if (root->word != '#') {

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

return;

}

if (root->lchild != NULL) {

strcat(root->lchild->code, root->code);

strcat(root->lchild->code, "0");

preorder(root->lchild);

}

if (root->rchild != NULL) {

strcat(root->rchild->code, root->code);

strcat(root->rchild->code, "1");

preorder(root->rchild);

}

}

【测试运行界面】



【完整源程序】

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct BinNode\_ {

char word;

int weigh;

char\* code;

struct BinNode\_\* lchild;

struct BinNode\_\* rchild;

}BinNode;

BinNode\* queue[0xFF];

int size = 0;

void init(BinNode\* node) {

node->lchild = NULL;

node->rchild = NULL;

node->code = (char\*)malloc(sizeof(char) \* 0xFF);

strcpy(node->code, "");

}

void del(int idx) {

for (int i = idx + 1; i <= size - 1; i++) {

queue[i - 1] = queue[i];

}

size--;

}

void insert(BinNode\* binNode) {

if (size == 0) {

queue[size] = binNode;

size++;

return;

}

for (int i = 0; i <= size - 1; i++) {

if (binNode->weigh < queue[i]->weigh) {

for (int j = size; j >= i + 1; j--) {

queue[j] = queue[j - 1];

}

queue[i] = binNode;

size++;

return;

}

}

queue[size] = binNode;

size++;

}

//将queue中索引为l和r的两颗树取出并合并成一棵新的树

BinNode\* createTree(int l, int r) {

int sum = queue[l]->weigh + queue[r]->weigh;

BinNode\* root = (BinNode\*)malloc(sizeof(BinNode));

init(root);

root->lchild = queue[l];

root->rchild = queue[r];

root->weigh = sum;

root->word = '#';

insert(root);

del(0);

del(0);

return root;

}

void preorder(BinNode\* root) {

if (root->word != '#') {

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

return;

}

if (root->lchild != NULL) {

strcat(root->lchild->code, root->code);

strcat(root->lchild->code, "0");

preorder(root->lchild);

}

if (root->rchild != NULL) {

strcat(root->rchild->code, root->code);

strcat(root->rchild->code, "1");

preorder(root->rchild);

}

}

int main() {

//create the tree

char word[100];

int freq[100];

int N = 0;

scanf("%d", &N);

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

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

}

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

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

}

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

BinNode\* node = (BinNode\*)malloc(sizeof(BinNode));

init(node);

node->weigh = freq[i];

node->word = word[i];

insert(node);

}

while (size > 1) {

createTree(0, 1);

}

BinNode\* haffmanRoot = queue[0];

preorder(haffmanRoot);

}