

哈夫曼树实验报告

课程名称：数据结构与算法

学 院： 计算机与信息安全学院

专业班级：19003603

学 号：1900301517

姓 名：陆洪业

报告日期：2021年 01 月 10 日

目录

**[1. 利用队列实现密码加密解密实验目的](#_Toc1831915961_WPSOffice_Level1)** **[1](#_Toc1831915961_WPSOffice_Level1)**

[1.1. 选用储存结构](#_Toc647397264_WPSOffice_Level2) [1](#_Toc647397264_WPSOffice_Level2)

[1.2. 实验代码](#_Toc1130979933_WPSOffice_Level2) [1](#_Toc1130979933_WPSOffice_Level2)

[1.3. 系统测试与结果](#_Toc1068613576_WPSOffice_Level2) [7](#_Toc1068613576_WPSOffice_Level2)

[1.4. 实验小结](#_Toc1598252537_WPSOffice_Level2) [8](#_Toc1598252537_WPSOffice_Level2)

# 哈夫曼树编码的应用

编写能对给定n个叶子结点，构建哈夫曼树，给出每个叶子结点对应编码的程序。

## 选用储存结构

树

## 实验代码

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct \_Node{

int data;

struct \_Node \*lchild;

struct \_Node \*rchild;

int huffmanCode[10],pos;//保存编码,在数组中从后往前存储，最多10位

} Node,\*Tree;

/\* 排序算法,从大到小 \*/

void Sort(int a[],int low,int high)

{

int i;

int temp;

if(low == high)return;

for(i=low; i<=high; i++){

if(a[i] > a[low]){

temp = a[low];

a[low] = a[i];

a[i] = temp;

}

}

Sort(a,low+1,high);

}

/\* 将一个字符型数组 char a[]转化为 Tree型数组，而且已经从大到小排序

\* 为构建哈夫曼树做准备\*/

Tree\* InitHuffmanTree(int a[],int count)

{

int i,j;

Tree T ;

Sort(a,0,count-1);//先排序

Tree \*HuffmanTrees = (Tree \*)malloc(sizeof(Tree)\*20);

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

T = (Node \*)malloc(sizeof(Node));

T->data = a[i];

T->lchild = T->rchild = NULL;

T->pos = 10;

HuffmanTrees[i] = T;

}

return HuffmanTrees;

}

/\* 给一棵子树的节点赋哈夫曼编码 \*/

void Encoding(Tree head,int code)

{

if(!head)return;

head->huffmanCode[--(head->pos)] = code;

Encoding(head->lchild,code);

Encoding(head->rchild,code);

}

/\* 构建哈夫曼树，打印哈夫曼编码

\* huffmanTrees[]中存储的是所有huffman树的根，

\* 而且已经从大到小排序\*/

Tree CreateHuffmanTree(Tree \*huffmanTrees,int count)

{

int i;

if(count <= 1)return huffmanTrees[0];

Node \*newTree = NULL;

if(!(newTree=(Node \*)malloc(sizeof(Node))))printf("overflow!\n");

newTree->data = huffmanTrees[count-1]->data + huffmanTrees[count-2]->data;

newTree->lchild = huffmanTrees[count-1];

Encoding(huffmanTrees[count-1],0);//修改子树的哈夫曼编码

newTree->rchild = huffmanTrees[count-2];

Encoding(huffmanTrees[count-2],1);

newTree->pos = 10;

count -= 2;

/\*插入合适位置（从大到小顺序）\*/

for(i=count; i>0 && newTree->data > huffmanTrees[i-1]->data; i--)

{

huffmanTrees[i] = huffmanTrees[i-1];

}

huffmanTrees[i] = newTree;

count++;

return CreateHuffmanTree(huffmanTrees,count);

}

/\* 输出哈夫曼编码 : 左0右1\*/

float printHuffmanCode(Tree head)

{

int i;

static int length = 0,TotLen=0,num=0;

if(!head)return length;

if (!head->lchild && !head->rchild)

{

printf("%d ",head->data);

for(i=head->pos; i<10; i++)

{

printf("%d",head->huffmanCode[i]);

}

printf("\n");

// printf("%d\n",head->data);

TotLen += head->data \* (10-head->pos);

num += head->data;

}

printHuffmanCode(head->lchild);

printHuffmanCode(head->rchild);

length = TotLen/num;

return length;

}

int main(int argc, char \*argv[])

{

int a[100] = {0},length;

int i=0,j=0,n;

scanf("%d",&n);

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

{

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

}

int count = n;

Tree \*huffmanTrees = InitHuffmanTree(a,count);

Tree huffmanTree = CreateHuffmanTree(huffmanTrees,count);

// printf("Here is the HuffmanCode :\n\n");

length = printHuffmanCode(huffmanTree);

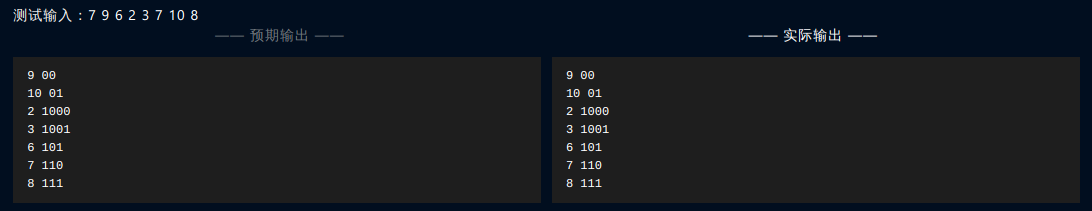
// printf("\nAverage length of the code : %.2f\n",length);

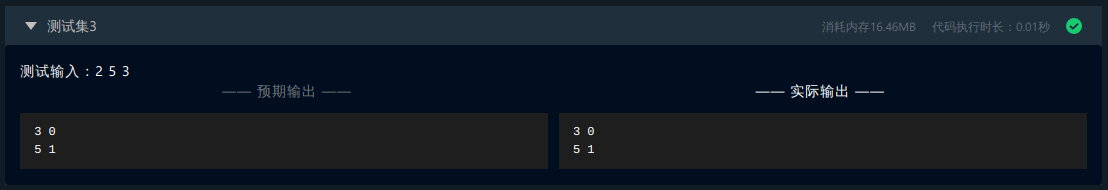
getchar();

return 0;

}

## 系统测试与结果





上述测试及结果证明了系统测试通过,各项操作函数能正常执行,并达到了实验要求的目标,实现了哈夫曼树的构建, 并给叶节点进行了哈夫曼编码.

## 实验小结

通过本次实验, 我学会了哈夫曼树的构建和哈夫曼编码的生成.