实习报告：实验五

树及其应用——哈夫曼编码

班级：17计算机一班 姓名：陈新朋 学号：1725111006 完成日期：2019.1.2

1. 需求分析
2. 以二叉链表为存储结构来存储哈夫曼树，内容包括字符和权重。
3. 用户可以选择从终端中输入字符和权重来建立哈夫曼树或者从文件中读取数据进行
4. 用户界面设计为了菜单方式，输入菜单命令I、初始化 E、编码 D、译码 P、印代码文件 T、印哈夫曼树 Q、退出

输入字符选择功能：

1. 初始化（Initialization）：从终端或文件读入字符集大小n,以及n个字符和n个权值，建立哈夫曼树，并将它存于文件hfmTree中。
2. 编码（Encoding）：利用已建好的哈夫曼树（如不在内存，则从文件hfmTree中读入），对文件ToBeTran中的正文进行编码，然后将结果存入文件CodeFile中。
3. 译码（Decoding）：利用已建好的哈夫曼树将文件CodeFile中的代码进行译码，结果存入文件TextFile中。
4. 印代码文件（Print）：将文件CodeFile以紧凑格式显示在终端上，每行50个代码。同时将此字符形式的编码文件写入文件CodePrin中。
5. 印哈夫曼树（Tree printing）：将已在内存中的哈夫曼树以直观的方式（树或凹入表形式）显示在终端上，同时将此字符形式的哈夫曼树写入文件TreePrint中。
6. 退出：输入Q，程序退出。
7. 用户可以以文件的形式读入哈夫曼树的信息，在”hfmTree.txt”文件中第一行的数据为节点个数，之后每一行分别为字符以及字符的权重。
8. 测试数据：见上机指导书Ｐ149测试数据。
9. 程序执行的命令为
10. 判断用户输入，执行相应初始化、编码、译码、印代码文件和印哈夫曼树的功能

二、概要设计

1、设定树节点的抽象数据类型定义：

ADT Node

{

Node \*parent, \*lchild, \*rchild;

int weight,mark;

char ch;

};

2、设定哈夫曼树的抽象数据类型定义：

ADT huffman

{

数据对象D={ai |ai∈Node,i=1,2,…,n,n>=0}

数据关系：R1={<ai-1 ,ai >|ai-1 ,ai ∈D,i=2,…,n}

基本操作

void init();

操作结果：从终端或文件读入字符集大小n,以及n个字符和n个权值，建立哈夫曼树，并将它存于文件hfmTree中。

void Encoding();

初始条件：哈夫曼树已存在

操作结果：利用已建好的哈夫曼树（如不在内存，则从文件hfmTree中读入），对文件ToBeTran中的正文进行编码，然后将结果存入文件CodeFile中。

void Decoding()

初始条件：哈夫曼树已存在

操作结果：利用已建好的哈夫曼树将文件CodeFile中的代码进行译码，结果存入文件TextFile中。

void PrintCode();

初始条件：至少执行过一次编码操作

操作结果：将文件CodeFile以紧凑格式显示在终端上，每行50个代码。同时将此字符形式的编码文件写入文件CodePrin中。

void print()

初始条件：哈夫曼树已存在

操作结果：将已在内存中的哈夫曼树以凹入表形式显示在终端上，同时将此字符形式的哈夫曼树写入文件TreePrint中。

void printHelp(Node\* p, string ss)

初始条件：根节点p存在

操作结果：凹入表打印函数，为print函数内部调用。

void printHFMCode()

初始条件：哈夫曼树已存在

操作结果：在终端显示所有字符以及其相应的哈夫曼编码。

void CreatHuffmanCode()

初始条件：哈夫曼树已存在

操作结果：从根节点开始创建哈夫曼编码，使其存在于内存中。

void select(Node T[], int n, int &min, int &sub)

操作结果：通过两次遍历，找到节点数组T中权重最小和次小的值的下标。

huffman()

操作结果：构造函数，对哈夫曼树进行初始化。

};

本程序包含四个模块

1. 主程序模块

int main()

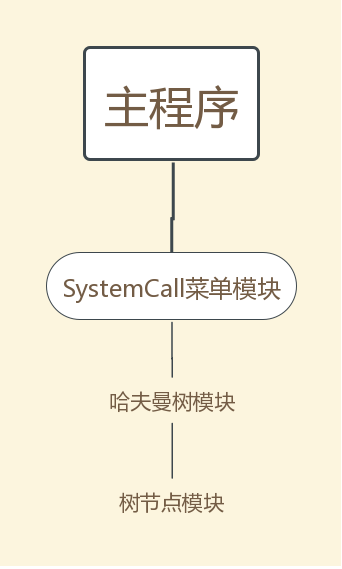
{

SystemCall();

return ();

}

1. SystemCall菜单模块——实现用户交互
2. 哈夫曼树模块——实现哈夫曼树的存储
3. 树节点模块——实现具体信息的存储



三、详细设计

1、哈夫曼树.cpp主文件

// Huffman.cpp: 定义控制台应用程序的入口点。

//

#include "stdafx.h"

#include "huffman.h"

#include <iostream>

using namespace std;

void SystemCall()

{

huffman tree;

char cmd;

cout << "请输入命令：I、初始化 E、编码 D、译码 P、印代码文件 T、印哈夫曼树 Q、退出" << endl;

while (cin >> cmd)

{

if (cmd == 'Q')

break;

switch (cmd)

{

case 'I':

tree.init();

break;

case 'E':

tree.Encoding();

break;

case 'D':

tree.Decoding();

break;

case'P':

tree.PrintCode();

break;

case'T':

tree.print();

break;

}

cout << "请输入命令：I、初始化 E、编码 D、译码 P、印代码文件 T、印哈夫曼树 Q、退出" << endl;

}

}

int main()

{

SystemCall();

return 0;

}

2、huffman.h文件

#ifndef \_HUFFMAN\_H\_

#define \_HUFFMAN\_H\_

#include <iostream>

using namespace std;

struct Node

{

Node \*parent, \*lchild, \*rchild;

int weight,mark;

char ch;

Node(char ch = '0', int mark = 0, Node \*parent = NULL, Node \*lchild = NULL, Node \*rchild = NULL, int weight = -1)

{

this->parent = parent;

this->lchild = lchild;

this->rchild = rchild;

this->weight = weight;

this->mark = mark;

this->ch = ch;

}

};

class huffman

{

public:

int NodeNum;

char \*\*HuffmanCode;

huffman()

{

NodeNum = 0;

header = 0;

HuffmanCode = NULL;

SumCode = NULL;

}

void init();//初始化（Initialization）。从终端或文件读入字符集大小n,以及n个字符和n个权值，建立哈夫曼树，并将它存于文件hfmTree中。

void print();//印哈夫曼树（Tree printing）。将已在内存中的哈夫曼树以直观的方式（树或凹入表形式）显示在终端上，同时将此字符形式的哈夫曼树写入文件TreePrint中。

void printHFMCode();

void Encoding();//编码（Encoding）。利用已建好的哈夫曼树（如不在内存，则从文件hfmTree中读入），对文件ToBeTran中的正文进行编码，然后将结果存入文件CodeFile中。

void Decoding();//译码（Decoding）。利用已建好的哈夫曼树将文件CodeFile中的代码进行译码，结果存入文件TextFile中。

void PrintCode();//印代码文件（Print）。将文件CodeFile以紧凑格式显示在终端上，每行50个代码。同时将此字符形式的编码文件写入文件CodePrin中。

private:

Node \* header;

char \* SumCode;

void CreatHuffmanCode();

void printHelp(Node\* p, string ss);

void select(Node T[], int n, int &min, int &sub);

};

#endif //\_HUFFMAN\_H\_

3、huffman.cpp文件

#include "stdafx.h"

#include "Huffman.h"

#include <string>

#include <fstream>

void huffman::select(Node T[], int n, int &min, int &sub)

{

int i;

for (i = 0; i<n&&T[i].parent!=NULL; i++){}

min = i;

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

{

if (T[i].parent != NULL)

continue;

else if (T[min].weight>T[i].weight)

min = i;

}

T[min].parent = &T[n];

for (i = 0; i<n&&T[i].parent!=NULL; i++){}

sub = i;

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

{

if (T[i].parent !=NULL)

continue;

else if (T[sub].weight>T[i].weight&&i!=min)

sub = i;

}

T[sub].parent = &T[n];

}

void huffman::init()

{

int cmd, i;

cout << "请选择初始化方式：1.新建一个哈夫曼树 2.从文件中读取" << endl;

cin >> cmd;

if (cmd != 1 && cmd != 2)

cout << "输入有误";

else if (cmd == 1)

{

cout << "请输入想要加入的节点个数：";

cin >> NodeNum;

if (NodeNum <= 0)

cout << "输入错误" << endl;

else//新建一个霍夫曼树

{

fstream out("hfmTree.txt");

out << NodeNum << endl;

Node \*temp = new Node[2 \* NodeNum - 1];

cout << "输入n次（节点字符，节点权重）";

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

{

cin >> temp[i].ch >> temp[i].weight;

out << temp[i].ch << " " << temp[i].weight << endl;

}

out.close();

for (i = NodeNum; i < 2 \* NodeNum - 1; i++)

{

int s1, s2;

select(temp, i, s1, s2);

temp[i].lchild = &temp[s1];

temp[i].rchild = &temp[s2];

temp[i].weight = temp[s1].weight + temp[s2].weight;

}

header = &temp[i - 1];//将头节点赋给头指针

CreatHuffmanCode();

}

}

else

{

fstream in("hfmTree.txt");

in >> NodeNum;

if (NodeNum <= 0)

cout << "输入错误" << endl;

else//建一个霍夫曼树

{

Node \*temp = new Node[2 \* NodeNum - 1];

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

{

in >> temp[i].ch >> temp[i].weight;

}

in.close();

for (i = NodeNum; i < 2 \* NodeNum - 1; i++)

{

int s1, s2;

select(temp, i, s1, s2);

temp[i].lchild = &temp[s1];

temp[i].rchild = &temp[s2];

temp[i].weight = temp[s1].weight + temp[s2].weight;

}

header = &temp[i - 1];//将头节点赋给头指针

CreatHuffmanCode();

}

}

}

void huffman::CreatHuffmanCode()

{

int i;

//初始化

char \*cd = new char[NodeNum + 1];

SumCode = new char[NodeNum];

int cdlen = 0;

HuffmanCode = new char\*[NodeNum];

int hfmcodelen = 0;

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

{

HuffmanCode[i] = new char[NodeNum];

}

//初始化end

Node \*p = header;

while (p)

{

if (p->mark == 0)

{

p->mark = -1;

if (p->lchild!=NULL)

{

p = p->lchild;

cd[cdlen++] = '0';

}

else if (p->rchild == NULL)

{

cd[cdlen] = '\0';

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

HuffmanCode[hfmcodelen][i] = cd[i];

SumCode[hfmcodelen]=p->ch;

hfmcodelen++;

}

}

else if (p->mark == -1)

{

p->mark = 1;

if (p->rchild != NULL)

{

p = p->rchild;

cd[cdlen++] = '1';

}

}

else

{

p = p->parent;

cd[cdlen--] = '\0';

}

}

delete cd;

}

void huffman::printHelp(Node\* p, string ss)

{

if (p == NULL)

return;

ss += " ";

printHelp(p->rchild, ss);

ofstream out("TreePrint.txt",ios::app);

cout << ss << p->ch << " " << p->weight << endl;

out << ss << p->ch << " " << p->weight << endl;

out.close();

printHelp(p->lchild, ss);

}

void huffman::print()

{

ofstream out("TreePrint.txt", ios::trunc);//清空文件

out.close();

cout << "对哈夫曼树进行凹入表打印，并写入文件TreePrint中" << endl;

string ss = " ";

printHelp(header, ss);

cout << endl << "印哈夫曼树成功" << endl;

}

void huffman::printHFMCode()

{

int i;

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

cout <<SumCode[i]<<" "<< HuffmanCode[i] << endl;

}

void huffman::Encoding()

{

char temp;

int i;

cout << "对文件ToBeTran进行编码" << endl;

ifstream in("ToBeTran.txt");

ofstream out("CodeFile.txt");

while (!in.eof())

{

in >> temp;

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

if (temp == SumCode[i])

break;

out << HuffmanCode[i];

}

in.close();

out.close();

cout << "编码成功" << endl;

}

void huffman::Decoding()

{

char temp;

Node \*p = header;

cout << "对文件CodeFile进行解码" << endl;

ifstream in("CodeFile.txt");

ofstream out("TextFile.txt");

while (!in.eof())

{

in >> temp;

if (temp == '0')

{

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

{

out << p->ch;

cout << p->ch;

p = header->lchild;

}

else

{

p = p->lchild;

}

}

else if(temp == '1')

{

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

{

out << p->ch;

cout << p->ch;

p = header->rchild;

}

else

{

p = p->rchild;

}

}

}

in.close();

out.close();

cout <<endl<< "解码成功" << endl;

}

void huffman::PrintCode()

{

char temp;

int count=0;

cout << "对文件CodeFile进行印代码操作" << endl;

ifstream in("CodeFile.txt");

ofstream out("CodePrin.txt");

while (!in.eof())

{

in >> temp;

if (count == 50)

{

out << endl;

cout << endl;

count = 0;

}

out << temp;

cout << temp;

count++;

}

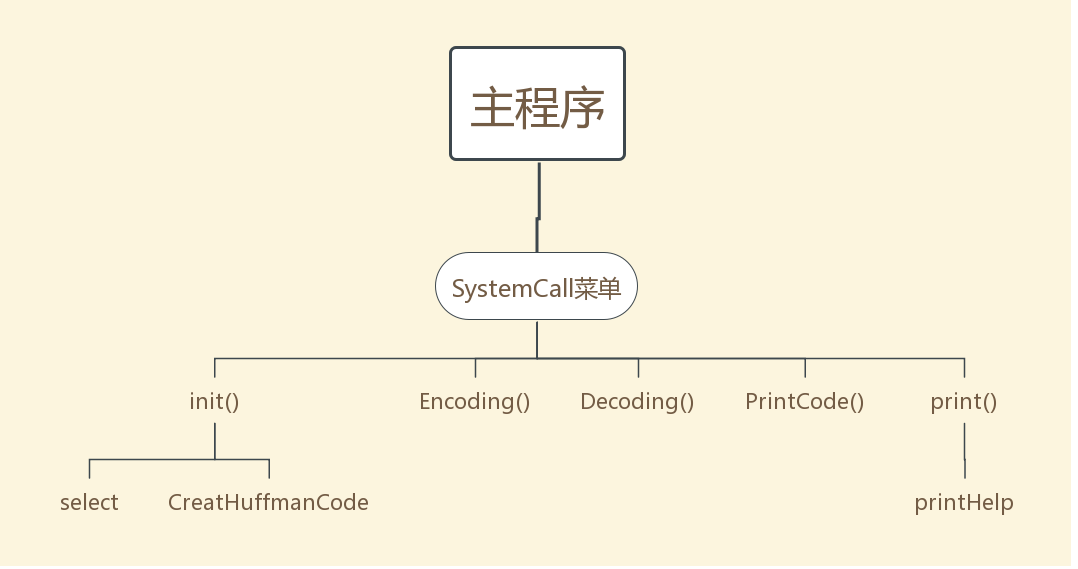
in.close();

out.close();

cout << endl << "印代码成功" << endl;

}

4、函数调用关系图：



四、调试分析

1、此次作业使用到了文件的操作，因此更需要注意‘\0’的使用，不然会出现输入乱码的情况。

2、解码部分需要注意，当到达叶子节点时，要把头指针的左孩子或者右孩子重新赋值给指针p。

3、本题中查建立哈夫曼树算法init的时间复杂度为O（n2），建立哈夫曼编码的算法CreatHuffmanCode的时间复杂度为O（n）。

3、经验体会：写代码需要一步一步加功能，验证完一个功能后再开始做下一个功能，否则前面的没写好，后面的也会乱掉。

五、用户手册

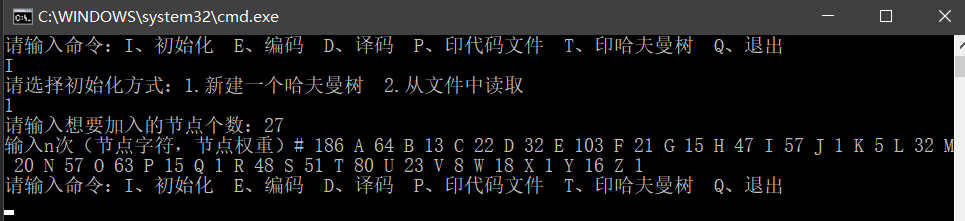
用户界面设计为了菜单方式，输入菜单命令I、初始化 E、编码 D、译码 P、印代码文件 T、印哈夫曼树 Q、退出

输入字符选择功能：

* 1. 初始化（Initialization）：从终端或文件读入字符集大小n,以及n个字符和n个权值，建立哈夫曼树，并将它存于文件hfmTree中。
  2. 编码（Encoding）：利用已建好的哈夫曼树（如不在内存，则从文件hfmTree中读入），对文件ToBeTran中的正文进行编码，然后将结果存入文件CodeFile中。
  3. 译码（Decoding）：利用已建好的哈夫曼树将文件CodeFile中的代码进行译码，结果存入文件TextFile中。
  4. 印代码文件（Print）：将文件CodeFile以紧凑格式显示在终端上，每行50个代码。同时将此字符形式的编码文件写入文件CodePrin中。
  5. 印哈夫曼树（Tree printing）：将已在内存中的哈夫曼树以直观的方式（树或凹入表形式）显示在终端上，同时将此字符形式的哈夫曼树写入文件TreePrint中。
  6. 退出：输入Q，程序退出。

六：测试结果：

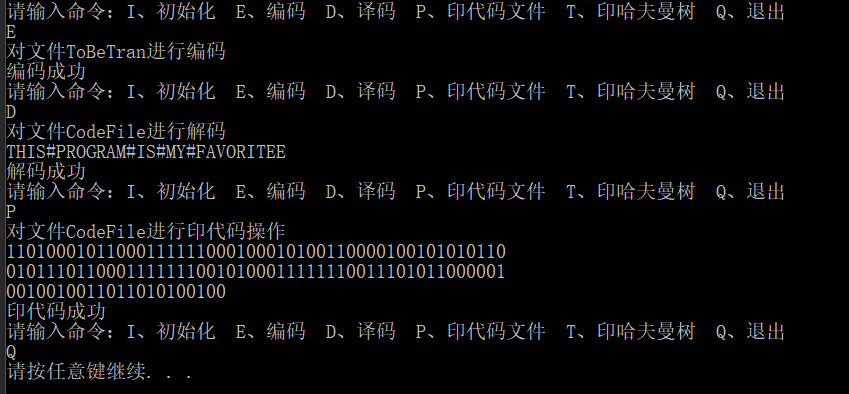
* + 1. 显示菜单，新建一个哈夫曼树并进行初始化：



从文件中读取哈夫曼树并使用凹入表打印



* + 1. 编码、解码、印代码、退出操作



七、附录

源程序文件名清单：

哈夫曼树.Cpp //主程序

huffman.h //哈夫曼树头文件

huffman.c //哈夫曼树源文件