**Lab3 Huffuman树压缩**

**唐凯成 PB16001695**

**一.实验要求**

1、通过Huffuman树的数据结构实现对文件的压缩

·实现Huffuman树的数据结构

·通过构建Huffuman表来对源文件转码进行压缩

·通过从压缩文件头读出的Huffuman表重构Huffuman树并解压缩

1. 实现压缩程序的GUI界面

·运用Qt（MacOS）来实现GUI界面的工程构建

·通过维护QMainWindow类来完成GUI界面与源程序的链接

**二.实验平台**

Apple LLVM version 9.1.0 (clang-902.0.39.2)

Visual Studio Code v1.29.0

Qt 5.8.0

**三.实验内容**

1、总体设计

整个实验中采用了Huffuman树作为主要的数据结构，使用Huffuman树来实现对文件二进制码的翻译工作。

2、Huffuman编码的生成

通过统计文件中不同ascll码的频数来生成不同的Huffum分支，并由此构成Huffuman编码，将该编码表存在压缩文件的开头，方便在解压缩时重新构建Huffuman树。

3、Huffuman压缩与解压缩

由生成的Huffuman树对原文件中的ASCLL编码进行转码，将原字符转为新的二进制码输出，由于字符频数的影响，原文件的大小即相对减小。

对于压缩文件的解压，即由Huffuman编码构建的原Huffuman树，由现有编码从根遍历Huffuman树即可得到对应的字符，从而完成文件的解压。

4、GUI界面的实现

由Qt内已经构建好的QMainWindow类直接调用窗口设置，并链接按键、路径到对应的函数即可实现简单的GUI界面，具体的实现界面见实验结果中。

**四.实验代码**

1、hf.h（Huffuman树与压缩的函数申明）

#ifndef HF\_H\_INCLUDED

#define HF\_H\_INCLUDED

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <limits.h>

// 字符频度的存储结点

typedef struct {

    unsigned char huf\_char;             // 无符号字符

    unsigned long weight;           // 每种字符的出现频度

} CountNode;

// 哈夫曼树结点

typedef struct {

    unsigned char huf\_char;             // 无符号字符

    unsigned long weight;               // 每种字符出现频度

    char \*code;                         // 得到哈夫曼编码（长度不固定）

    int parent,left,right;              // 父结点和左右结点

} HuffNode, \*HuffmanTree;

void Select(HuffNode \*HT, unsigned int n, int &s1, int &s2);

void CreateTree(HuffNode \*HT, unsigned int char\_kinds, unsigned int node\_num);

void HufCode(HuffNode \*HT, unsigned n,int node\_num);

bool Compress(const char \*ifname);

bool Decompress(const char \*ifname);

#endif // HF\_H\_INCLUDED

2、hf.cpp（函数具体实现文件）

#include "hf.h"

// 选择最小和次小的两个结点，建立哈夫曼树

void Select(HuffNode \*HT, unsigned int n, int &s1, int &s2)

{

    // 找最小

    unsigned int i;

    unsigned long min = ULONG\_MAX;      //将min赋值为无限大

    for(i = 0; i < n; ++i)

        if(HT[i].parent == 0 && HT[i].weight < min)

        {

            min = HT[i].weight;

            s1 = i;

        }

        HT[s1].parent=1;              // 使用parent进行标记

    // 找次小

    min=ULONG\_MAX;

    for(i = 0; i < n; ++i)

        if(HT[i].parent == 0 && HT[i].weight < min)

        {

            min = HT[i].weight;

            s2 = i;

        }

}

// 建立哈夫曼树

void CreateTree(HuffNode \*HT, unsigned int char\_kinds, unsigned int node\_num)

{

    unsigned int i;

    int s1, s2;

    for(i = char\_kinds; i < node\_num; ++i)

    {

        Select(HT, i, s1, s2);                              //合并最小的两个结点加入结点数组中

        HT[s1].parent = HT[s2].parent = i;

        HT[i].left = s1;

        HT[i].right = s2;

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

    }

}

// 生成哈夫曼编码

void HufCode(HuffNode \*HT, unsigned n,int node\_num)

{

    unsigned int i;

    int cur, next, start;

    char \*buf = (char \*)malloc(node\_num\*sizeof(char));      //创建一个缓存用以暂存编码（根据文件中字符种类动态分配）

    buf[node\_num-1] = '\0';                                 //结尾加上字符串结束符

    for(i = 0; i < n; ++i)

    {                                                       // 从叶子向根反向遍历求编码，反向保存编码到buf中

        start = node\_num-1;

        for(cur = i, next = HT[i].parent; next != 0; cur = next, next = HT[next].parent)

            if(HT[next].left == cur)

                buf[--start] = '0';                         //左结点编码0

            else

                buf[--start] = '1';                         //右结点编码0

        HT[i].code = (char \*)malloc((node\_num-start)\*sizeof(char));         // 为huf编码动态分配存储空间

        strcpy(HT[i].code, &buf[start]);                // 正向保存

    }

    free(buf);

}

// 压缩

bool Compress(const char \*ifname)

{

char \*ofname;

    CountNode node\_temp;

    HuffmanTree huf\_tree;

    unsigned int i, j;              //循环变量

    unsigned int char\_kinds;        //字符种类

    unsigned char char\_buf;         //暂存8bits字符

    unsigned long file\_len = 0;     //文件长度

    FILE \*infile, \*outfile;         //输入输出文件

    unsigned int node\_num;          //节点数

    char code\_buf[256] = "\0";      //待存编码缓冲区

    unsigned int code\_len;          //临时huf编码长度

//生出输出文件名

bool flag = false;                          //用于判断是否有路径

short PathSize = strlen(ifname);            //路径总长度

for(short i = PathSize-1; i >=0 ; i--)      //对文件路径进行拆分

{

if(ifname[i] == '\\')

{

ofname = new char[PathSize-i+3];

for(short j = 0; i < PathSize; i++, j++)

ofname[j] = ifname[i+1];

flag = true;

break;

}

}

if(flag == false)

{

ofname = new char[PathSize+4];

strcpy(ofname, ifname);

}

strcat(ofname, ".huff");

    //统计不同字符的频数

    CountNode \*tmp\_nodes =(CountNode \*)malloc(256\*sizeof(CountNode));       //生成tmp\_nodes[256]

    //初始化结点

    for(i = 0; i < 256; ++i)

    {

        tmp\_nodes[i].weight = 0;

        tmp\_nodes[i].huf\_char = (unsigned char)i;                   //当前节点内存放当前字符的ascll码

    }

//读取文件并统计字符权重

    infile = fopen(ifname, "rb");

    if (infile == NULL)                                             //判断输入文件是否存在

        return false;                                               //不存在则解压失败

    fread((char \*)&char\_buf, sizeof(unsigned char), 1, infile);    //读入一个字符

    while(!feof(infile))

    {

        tmp\_nodes[char\_buf].weight++;                               //对应字符的权重+1

        file\_len++;                                                 //文件总长度+1

        fread((char \*)&char\_buf, sizeof(unsigned char), 1, infile);

    }

    fclose(infile);

    // 对按权数排序并删去权重为零的

    for(i = 0; i < 256-1; ++i)

        for(j = i+1; j < 256; ++j)

            if(tmp\_nodes[i].weight < tmp\_nodes[j].weight)

            {

                node\_temp = tmp\_nodes[i];

                tmp\_nodes[i] = tmp\_nodes[j];

                tmp\_nodes[j] = node\_temp;

            }

    // 统计实际的字符种类（出现次数不为0）

    for(i = 0; i < 256; ++i)

        if(tmp\_nodes[i].weight == 0)

            break;

    char\_kinds = i;

    //对特殊情况的处理（字符种类太少）

    if(char\_kinds==0)

    {

        printf("error:It's a empty file!");

        return false;

    }

if(char\_kinds<=20)

printf("tips:the file is very small,the result coundl not be ideal!");

    if (char\_kinds == 1)

    {

printf("the file is too small");

        outfile = fopen(ofname, "wb");                                                  //打开压缩后将生成的文件

        fwrite((char \*)&char\_kinds, sizeof(unsigned int), 1, outfile);                  //写入字符种类

        fwrite((char \*)&tmp\_nodes[0].huf\_char, sizeof(unsigned char), 1, outfile);      //写入唯一的字符

        fwrite((char \*)&tmp\_nodes[0].weight, sizeof(unsigned long), 1, outfile);        //写入字符频度，也就是文件长度

        free(tmp\_nodes);

        fclose(outfile);

    }

    else

    {   //生成huffuman树及编码

        node\_num = 2 \* char\_kinds - 1;                                  //哈夫曼树结点数

        huf\_tree = (HuffNode \*)malloc(node\_num\*sizeof(HuffNode));       //为huffunman树分配空间

        //将频数统计的结点数据放入huffuman树阶段中

        for(i = 0; i < char\_kinds; ++i)

        {

            huf\_tree[i].huf\_char = tmp\_nodes[i].huf\_char;

            huf\_tree[i].weight = tmp\_nodes[i].weight;

            huf\_tree[i].parent = 0;

        }

        free(tmp\_nodes);                                                //释放统计结点

        // 初始化其余huffuman树结点

        for(; i < node\_num; ++i)

            huf\_tree[i].parent = 0;

        CreateTree(huf\_tree, char\_kinds, node\_num);                     //创建哈夫曼树

HufCode(huf\_tree, char\_kinds,node\_num);                         //生成哈夫曼编码

        //完成生成哈夫曼树及编码

        //向压缩文件中写入huffuman编码

        outfile = fopen(ofname, "wb");

        fwrite((char \*)&char\_kinds, sizeof(unsigned int), 1, outfile);  //写入总计字符种类数

        for(i = 0; i < char\_kinds; ++i)

        {

            fwrite((char \*)&huf\_tree[i].huf\_char, sizeof(unsigned char), 1, outfile);   //写入字符

            fwrite((char \*)&huf\_tree[i].weight, sizeof(unsigned long), 1, outfile);     //写入字符对应权重

        }

        //写入文件长度

        fwrite((char \*)&file\_len, sizeof(unsigned long), 1, outfile);       //写入文件长度

        //开始压缩(写入转换后字的字符编码)

        infile = fopen(ifname, "rb");                                       //以二进制形式打开待压缩的文件

        fread((char \*)&char\_buf, sizeof(unsigned char), 1, infile);     //每次读取1字符(8位)

        while(!feof(infile))

        {

            // 匹配字符对应编码

            for(i = 0; i < char\_kinds; ++i)

                if(char\_buf == huf\_tree[i].huf\_char)

                    strcat(code\_buf, huf\_tree[i].code);

            //将8位字符串存入一个char变量中

            while(strlen(code\_buf) >= 8)

            {

                char\_buf = '\0';                    //清空字符暂存空间，改为暂存字符对应编码

                for(i = 0; i < 8; ++i)

                {

                    char\_buf <<= 1;                 //左移一位，为下一个bit腾出位置

                    if(code\_buf[i] == '1')          //将字符串的1变为char变量中一个位的1

                        char\_buf |= 1;              //置最低位1

                }

                fwrite((char \*)&char\_buf, sizeof(unsigned char), 1, outfile);       //将1字节(8位)存入文件

                strcpy(code\_buf, code\_buf+8);       //编码buf去除前八位

            }

            fread((char \*)&char\_buf, sizeof(unsigned char), 1, infile);

        }

        // 处理最后不足8bits编码

        code\_len = strlen(code\_buf);

        if(code\_len > 0)

        {

            char\_buf = '\0';

            for(i = 0; i < code\_len; ++i)

            {

                char\_buf <<= 1;

                if(code\_buf[i] == '1')

                    char\_buf |= 1;

            }

            char\_buf <<= 8-code\_len;          //将编码字段从尾部移到字节的高位(空位补0)

            fwrite((char \*)&char\_buf, sizeof(unsigned char), 1, outfile);     //存入最后一个字节

            //完成对原文件的huffuman编码

        }

        //关闭文件

        fclose(infile);

        fclose(outfile);

        //回收内存

        for(i = 0; i < char\_kinds; ++i)

            free(huf\_tree[i].code);

        free(huf\_tree);

    }

    //压缩成功

    return true;

}

//compress

// 解压

bool Decompress(const char \*ifname)

{

    unsigned int i;

    unsigned long file\_len;

    FILE \*infile, \*outfile;

    unsigned int node\_num;

    HuffmanTree huf\_tree;

    unsigned long writen\_len = 0;   //控制文件写入长度

    unsigned int char\_kinds;        //存储字符种类

    unsigned char code\_temp;        //暂存8bits编码

    unsigned int root;              //根

    //打开解压文件

    infile = fopen(ifname, "rb");// 以二进制方式打开压缩文件

    if (infile == NULL)// 判断输入文件是否存在

        return false;

    //获取解压后的文件名

int len = strlen(ifname);

char \*ofname = new char[len-4];

for(int i = 0; i < len-5; i++)  //复制输入的文件名，去掉最后两位后缀

ofname[i] = ifname[i];

ofname[len-5] = '\0';

    // 读取压缩文件前端的字符及对应编码，用于重建哈夫曼树

    fread((char \*)&char\_kinds, sizeof(unsigned int), 1, infile);        //读取字符种类数

    if (char\_kinds == 1)

    {

        fread((char \*)&code\_temp, sizeof(unsigned char), 1, infile);    //读取唯一的字符

        fread((char \*)&file\_len, sizeof(unsigned long), 1, infile);     //读取文件长度

        outfile = fopen(ofname, "wb");                                  //打开压缩后将生成的文件

        while (file\_len--)

            fwrite((char \*)&code\_temp, sizeof(unsigned char), 1, outfile);

        fclose(infile);

        fclose(outfile);

    }

    else

    {

        node\_num = 2 \* char\_kinds - 1;                                  //根据字符种类数，计算建立哈夫曼树所需结点数

        huf\_tree = (HuffNode \*)malloc(node\_num\*sizeof(HuffNode));       //动态分配哈夫曼树结点空间

        // 读取字符及对应权重，存入哈夫曼树节点

        for(i = 0; i < char\_kinds; ++i)

        {

            fread((char \*)&huf\_tree[i].huf\_char, sizeof(unsigned char), 1, infile); //读入字符

            fread((char \*)&huf\_tree[i].weight, sizeof(unsigned long), 1, infile);   //读入字符对应权重

            huf\_tree[i].parent = 0;

        }

        //初始化

        for(; i < node\_num; ++i)

            huf\_tree[i].parent = 0;

        //重建哈夫曼树

        CreateTree(huf\_tree,char\_kinds,node\_num);

        //读取文件长度和编码

        fread((char \*)&file\_len, sizeof(unsigned long), 1, infile);     //读入文件长度

        outfile = fopen(ofname, "wb");                                  //打开压缩后将生成的文件

        root = node\_num-1;                                              //最后一个节点即为根节点

        //开始解压缩

        while(1)

        {

            fread((char \*)&code\_temp, sizeof(unsigned char), 1, infile);    //读取一个字符长度的编码

            //对该编码进行转码还原成ascll码

            for(i = 0; i < 8; ++i)

            {

                //由根向下直至叶节点正向匹配编码对应字符

                if(code\_temp & 128)                                         //检测code\_temp首位的值

                    root = huf\_tree[root].right;                            //递归生成新的根

                else

                    root = huf\_tree[root].left;

                if(root < char\_kinds)                                       //根已经到达叶处

                {

                    fwrite((char \*)&huf\_tree[root].huf\_char, sizeof(unsigned char), 1, outfile);

                    ++writen\_len;

                    if (writen\_len == file\_len) break;                      //该文件已转码完成

                    root = node\_num-1;                                      //复位根

                }

                code\_temp <<= 1;// 将编码缓存的下一位移到最高位

            }

            if (writen\_len == file\_len) break;//解压缩已完成

        }

        // 关闭文件

        fclose(infile);

        fclose(outfile);

        // 释放内存

        free(huf\_tree);

    }

    //解压成功

    return true;

}

3、GUI部分代码：

（1）hfcom.h

#ifndef HFCOM\_H

#define HFCOM\_H

#include <QtWidgets/QMainWindow>

#include "ui\_hfcom.h"

#include <qpushbutton.h>

#include <qobject.h>

#include <qstring.h>

#include <qfiledialog.h>

#include <qlineedit.h>

#include <qmessagebox.h>

#include <qstatusbar.h>

#include <qlabel.h>

#include <qicon.h>

#include "hf.h"

class hfcom : public QMainWindow

{

Q\_OBJECT

public:

hfcom(QWidget \*parent = 0);

~hfcom();

private:

QPushButton \*button\_open1;

QLineEdit \*path1;

QPushButton \*button\_compress;

QString \*compressed\_filename;

QStatusBar \*ps;

void compress();

void open\_compressed\_file();

QPushButton \*button\_open2;

QLineEdit \*path2;

QPushButton \*button\_decompress;

QString \*decompressed\_filename;

void decompress();

void open\_decompressed\_file();

};

#endif // HFCOM\_H

（2）hfcom.cpp：

#include "hfcom.h"

hfcom::hfcom(QWidget \*parent) : QMainWindow(parent)

{

this->resize(400, 247);

this->setMinimumSize(450, 278);

this->setMaximumSize(450, 278);

this->setWindowIcon(QIcon("hflog.ico"));

ps = new QStatusBar(this);

ps->setGeometry(0,241,450,37);

ps->showMessage(QString::fromLocal8Bit("ps:不支持中文路径和文件名"));

button\_open1 = new QPushButton(QString::fromLocal8Bit("打开文件"), this);

button\_open1->setGeometry(20, 40, 80, 37);

path1 = new QLineEdit(this);

path1->setReadOnly(true);

path1->setGeometry(110, 40, 200, 37);

button\_compress = new QPushButton(QString::fromLocal8Bit("压缩"), this);

button\_compress->setGeometry(350, 40, 80, 37);

button\_open2 = new QPushButton(QString::fromLocal8Bit("打开文件"), this);

button\_open2->setGeometry(20, 140, 80, 37);

path2 = new QLineEdit(this);

path2->setReadOnly(true);

path2->setGeometry(110, 140, 200, 37);

button\_decompress = new QPushButton(QString::fromLocal8Bit("解压"), this);

button\_decompress->setGeometry(350, 140, 80, 37);

QObject::connect(button\_compress, &QPushButton::clicked, this, &hfcom::compress);

QObject::connect(button\_decompress, &QPushButton::clicked, this, &hfcom::decompress);

QObject::connect(button\_open1, &QPushButton::clicked, this, &hfcom::open\_compressed\_file);

QObject::connect(button\_open2, &QPushButton::clicked, this, &hfcom::open\_decompressed\_file);

//ui.setupUi(this);

}

hfcom::~hfcom()

{

}

void hfcom::compress()

{

if( Compress((compressed\_filename->toStdString()).c\_str()) == true)

QMessageBox::information(this, tr("Information"), QString::fromLocal8Bit("压缩成功"));

else

QMessageBox::information(this, tr("Information"), QString::fromLocal8Bit("压缩失败"));

}

void hfcom::decompress()

{

if( Decompress((decompressed\_filename->toStdString()).c\_str()) == true )

QMessageBox::information(this, tr("Information"), QString::fromLocal8Bit("解压成功"));

else

QMessageBox::information(this, tr("Information"), QString::fromLocal8Bit("解压失败"));

}

void hfcom::open\_compressed\_file()

{

compressed\_filename = new QString(

QFileDialog::getOpenFileName(this, QString::fromLocal8Bit("打开要压缩的文件"), "", tr("Allfile(\*.\*)"))

);

path1->setText(\*compressed\_filename);

}

void hfcom::open\_decompressed\_file()

{

decompressed\_filename = new QString(

QFileDialog::getOpenFileName(this, QString::fromLocal8Bit("打开要解压的文件"), "", tr("hffile(\*.huff)"))

);

path2->setText(\*decompressed\_filename);

}

（3）main函数（GUI入口）：

#include "hfcom.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

hfcom w;

w.show();

return a.exec();

}

**五.实验结果**

1、GUI界面

由Hfcom工程，在Qt下构建生成的GUI界面为：

（1）进入界面：



（2）压缩结果：



（3）解压结果：



2、实验压缩结果分析



**五.实验总结**

本次实验中通过维护一个Huffuman树的结构来实现对文件的压缩与解压缩，成功将文件的大小降低了约20%，但对于非文本类文件的压缩效果还是较差。感谢老师第一次要求GUI，第一次学习了GUI的实现，感觉收获良多，在后续实验中会继续实现。