

二叉树的应用

郑悟强 PB22051082

2023.11.7

1 问题描述

用 huffman 压缩技术实现对任意文件的压缩和解压缩处理

要求对所有的文件类型进行压缩，压缩之后的文件后缀名为 huff。同时，可以对所有后缀名为 huff 的压缩文件进行解压缩。

具体要求：

1. 题目 2 中以 1 个字节 (8bit) 为单位进行 huffman 编码
2. 对任意文件进行压缩后可以输出一个后缀名为 huff 的单文件，并且可以对任意一个后缀名为 huff 的单文件进行解压还原出原文件。
3. 群内将提供 5 不同种类的文件包括文档、图片、视频、可执行文件等进行压缩测试，要求可以完成压缩和解压的步骤，并且解压出来的文件没有任何损失。

2 算法的描述

2.1 数据结构的描述

采用 HuffmanNode 结构体存储每一个结点，采用 HuffmanTree 类来存储一个 Huffman 树并封装 huffman 树的基本操作。

```
typedef struct huffnode {
    int left;
    int right;
    int parent;
    unsigned long long weight;
    unsigned char content; // 存储这个结点对应的原始内容
    string huffcode; // 存储这个字符的huffman编码
}huffnode;

class HuffmanTree {
public:
    huffnode huff[512]; // 所有结点

    int size = 0; // 目前用到的所有结点

    void CreateTree(); // 将结点连成树

    void EnCode(); // 产生huffman编码

    void select(int length, int& no1, int& no2); // 找到权重最小的两个结点

    void Input(); // 手动输入元素构建初始huffman树(debug用)

    void print(); // 打印huffman树
};
```

其中核心操作的具体代码如下：

```
void HuffmanTree::CreateTree() { //建立huffman树
    if (size < 3) {
        return;
    }
    //初始化一下这个树的每一个结点
    for (int t = 1; t < 2 * size; t++) {
        huff[t].left = 0;
        huff[t].right = 0;
        huff[t].parent = 0;
    }

    int min1, min2;
    //将树连接起来
    for (int i = size + 1; i < 2 * size; i++) {
        select(i-1, min1, min2); //找到权重最小且无双亲结点的两个结点
        huff[i].left = min1;
        huff[i].right = min2;
        huff[min1].parent = i;
        huff[min2].parent = i;
        huff[i].weight = huff[min1].weight + huff[min2].weight;
    }
}

void HuffmanTree::EnCode() { //根据huffman树生成huffman编码
    for (int i = 1; i <= size; i++) { //依次遍历所有元素
        int p = huff[i].parent;
        int cur = i;
        while (p != 0) {
            if (huff[p].left == cur) {
                huff[i].huffcode.insert(huff[i].huffcode.begin(), '0');
            }
            else {
                huff[i].huffcode.insert(huff[i].huffcode.begin(), '1');
            }
            cur = p;
            p = huff[cur].parent;
        }
    }
}
```

2.2 程序结构的描述

1. 用 compress 类封装核心功能：

```
#pragma once
#include "Huffmantree.h"
#include <fstream>

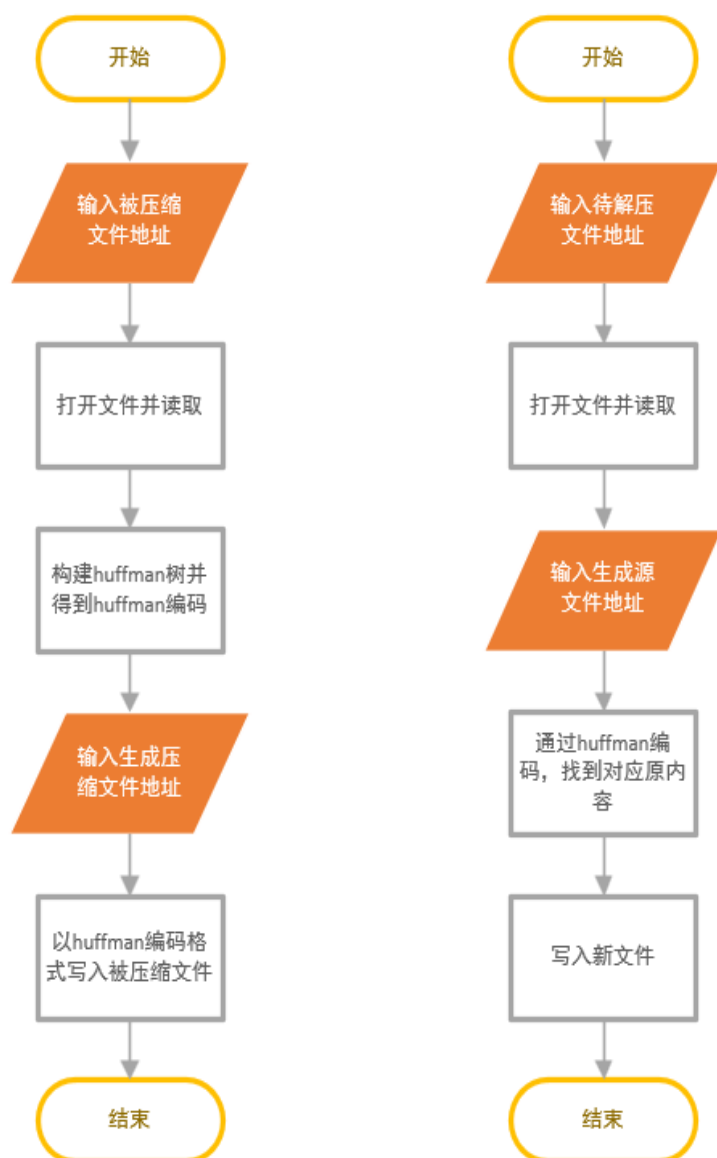
class Compress {
private:
    string loca; // 存储被压缩文件的位置
    string dest; // 存储压缩到的文件的地址
    HuffmanTree HuffT; // Huffman树

public:
    void compress(); // 对文件进行压缩

    void uncompress(); // 对文件进行解压

    void GetLine(FILE* FIn, string& strContent); // 一行一行读文件
};
```

两种核心功能流程图如下：



2. 压缩功能:

(1) 将被压缩文件以二进制格式读入, 并存储每一种字符的个数。

```
void Compress::compress() {
    cout << "请输入文件路径: ";
    cin >> loca;

    //step1:将目的文件读入, 并存储在数组中
    FILE* fp = fopen(loca.c_str(), "rb");//打开文件
    if (fp == nullptr) {
        cout << "文件路径不存在" << endl;
        return;
    }
    //读取文件内容, 采用循环, 一次读取1024个字节
    unsigned char readbuff[1024]; //设置缓冲区
    unsigned long long int fullweight[256] = { 0 };
    unsigned char content[256];
    unsigned long long int weight[256] = { 0 };
    while (1) {
        //fread函数的返回值, 表示成功读取的字节数量
        //用read_size来接收成功读取的字节数
        unsigned long long read_size = fread(readbuff, 1, 1024, fp);
        if (read_size == 0) {
            break; //read_size = 0表示已经读到了文件末尾
        }
        //统计出现字符的次数
        for (unsigned long long i = 0; i < read_size; ++i) {
            //利用字符的ASCII码值作为数组的下标统计次数
            fullweight[readbuff[i]]++;
        }
    }
    int charsize = 0; //表示一共出现的字符类型数量
    for (int i = 0; i < 256; i++) {
        if (fullweight[i]) {
            content[charsize] = i;
            weight[charsize] = fullweight[i];
            charsize++;
        }
    }
}
```

(2) 根据存储的权数生成 huffman 树并得到 huffman 编码。

```
//step2:根据权数和字符数构建huffman树
for (int t = 0; t <= charsize; t++) {
    HuffT.huff[t + 1].content = content[t];
    HuffT.huff[t + 1].weight = weight[t];
}
HuffT.size = charsize;
HuffT.CreateTree();
HuffT.EnCode();
```

(3) 根据 huffman 编码重写文件。

```
//step3:生成压缩后的文件
cout << "输入需要解压缩文件后的文件名";
cin >> dest;
FILE* fl = fopen(dest.c_str(), "wb");//f1文件指针指向打开的文件2.txt
fseek(fp, 0, SEEK_SET);//使用rewind(fp)函数也行
unsigned char ch = 0;//记录按位或后的字符
int bitcount = 0;//记录按位或的比特位长度
//采用循环读文件内容
while (1) {
    //fread函数的返回值，表示成功读取的字节数量
    //用read_size来接收成功读取的字节数
    unsigned long long read_size = fread(readbuff, 1, 1024, fp);
    if (read_size == 0) {
        break;//read_size = 0表示已经读到了文件末尾
    }

    for (unsigned long long i = 0; i < read_size; ++i) {
        //获取对应的中的编码
        string strcode;
        //找到对应的编码
        for (int t = 1; t <= HuffT.size; t++) {
            if (HuffT.huff[t].content == readbuff[i]) {
                strcode.append(HuffT.huff[t].huffcode);
                break;
            }
        }
        for (size_t j = 0; j < strcode.size(); j++) {
            ch <<= 1;//先将ch左移一位
            if (strcode[j] == '1') { //再让strcode的每个比特位与1进行或操作
                ch |= 1;
            }
            bitcount++; //每按位或一次就对bitcount++
            if (bitcount == 8) { //一个字节放满8个比特位
                fputc(ch, fl); //往文件中进行写操作
                bitcount = 0; //比特位计数置0
            }
        }
    }
    //检测ch放置的比特位个数，不够8个继续往文件中写
    if (bitcount > 0 && bitcount < 8) {
        ch <<= (8 - bitcount);
        fputc(ch, fl);
    }

    int fpSize = fpSize = ftell(fl);
    int outSize = outSize = ftell(fp);
    cout << "\n*****\n" << endl;
    cout << "压缩前: " << outSize << "KB" << endl;
    cout << "压缩后: " << fpSize << "KB" << endl;
    cout << "*****" << endl;
    fclose(fp); //关闭文件
    fclose(fl);
    cout << "压缩成功" << endl;
}
```

3. 解压缩功能:

(1) 读取待解压文件。

```

void Compress::uncompress() {
    //step1:打开待解压文件并读取内容
    cout << "请输入要解压的文件的地址: ";
    cin >> loca;
    if (loca.size() < 5 || loca.compare(loca.size() - 5, loca.size() - 1, ".huff")) {
        //只能解压缩后缀名为.huff的文件
        cout << "只能解压缩后缀名为.huff的文件"<<endl;
        return;
    }
    FILE* FIn = fopen(loca.c_str(), "rb");//以读的方式打开文件
    if (FIn == nullptr) { //如果文件为空或者没有文件
        perror("fopen");
        return;
    }

    cout << "请输入解压缩到的地址及文件名: ";
    cin >> dest;
    cout << dest << endl;
    FILE* FOut = fopen(dest.c_str(), "wb");//以写的方式打开文件

```

(2) 按 huffman 树找到源码并写入。






```

//step2:按huffman树找到对应源码进行写入
unsigned char readbuff[1024]; //写文件缓冲, 一次写1024bits
unsigned char bitcount = 0;
int cur = HuffT.size * 2 - 1; //cur指向哈夫曼树的根节点(2n-1)
const int FileSize = HuffT.huff[cur].weight; //根节点的权值即位原文件的大小
int CompressSize = 0; //记录压缩后的大小
while (1) {
    size_t read_size = fread(readbuff, 1, 1024, FIn);
    if (read_size == 0) {
        break;
    }
    for (size_t i = 0; i < read_size; i++) {
        unsigned char ch = readbuff[i]; //ch来保存压缩后的每一个字节
        bitcount = 0;
        while (bitcount < 8) {
            if (ch & 0x80) { //每个比特位与1000 0000按位与
                cur = HuffT.huff[cur].right;
            }
            else {
                cur = HuffT.huff[cur].left;
            }
            if (HuffT.huff[cur].left == 0 && HuffT.huff[cur].right == 0) { //走到了叶子节点的位置
                fputc(HuffT.huff[cur].content, FOut);
                cur = HuffT.size * 2 - 1; //回到根节点的位置
                CompressSize++;
                if (CompressSize == FileSize) { //判断解压缩后的结果与原文件大小是否相同
                    break;
                }
            }
            bitcount++;
            ch <<= 1;
        }
    }
}






```

3 调试分析

用于测试的被压缩文件:

 2_1	2023/11/2 16:19	文本文档	1 KB
 2_2	2023/11/2 16:19	文本文档	1 KB
 2_3	2023/11/2 16:40	BMP 文件	142 KB
 2_4	2023/11/2 16:43	MP4 文件	128 KB
 2_5	2023/11/2 16:56	应用程序	53 KB

解压得到的.huff 文件

 2_1.huff	2023/11/7 19:46	HUFF 文件	1 KB
 2_2.huff	2023/11/7 19:48	HUFF 文件	1 KB
 2_3.huff	2023/11/7 19:49	HUFF 文件	34 KB
 2_4.huff	2023/11/7 19:50	HUFF 文件	127 KB
 2_5.huff	2023/11/7 19:52	HUFF 文件	33 KB

解压缩得到的文件

 2.3(1)	2023/11/7 19:50	BMP 文件	142 KB
 2_3	2023/11/2 16:40	BMP 文件	142 KB
 2_4(1)	2023/11/7 19:52	MP4 文件	128 KB
 2_4	2023/11/2 16:43	MP4 文件	128 KB
 2_1(1)	2023/11/7 19:47	文本文档	1 KB
 2_1	2023/11/2 16:19	文本文档	1 KB
 2_2(1)	2023/11/7 19:48	文本文档	1 KB
 2_2	2023/11/2 16:19	文本文档	1 KB
 2_5(1)	2023/11/7 19:52	应用程序	53 KB
 2_5	2023/11/2 16:56	应用程序	53 KB

- 2_1 压缩前 515B，压缩后 218B
- 2_2 压缩前 528B，压缩后 369B
- 2_3 压缩前 144594B，压缩后 34286B
- 2_4 压缩前 130160B，压缩后 129599B
- 2_5 压缩前 54236B，压缩后 32832B

4 算法时空分析

用 m 表示使用的字符种类（最多 256 种）， n 表示文件的字符数量

4.1 时间复杂度

构造 huffman 树复杂度为 $O(m)$ ，得到 huffman 编码的复杂度为 $O(m^2)$ ，读写文件的复杂度为 $O(n)$ 。

4.2 空间复杂度

构造 huffman 树复杂度为 $O(m)$ ，得到 huffman 编码的复杂度为 $O(m)$ ，读写文件的复杂度为 $O(n)$ 。

5 实验体会与分析

通过本次实验，我学会了二叉树的基本功能的实现和使用，学会特殊的 huffman 二叉树的应用，并成功编写了 huffman 编码压缩文件的程序，掌握了文件二进制读写的操作。