



# 四川大学

## 《数据结构与算法分析》课程实验报告

课程设计题目：哈夫曼编码实现压缩与解压

设计人：任浩龙

班级：物联网工程班

学号：2018141461344

专业：物联网工程

指导教师：周欣

提交报告时间：2019 年 12 月 19 日

# 数据结构与算法分析实验课程

## 实验报告

课程名称	数据结构与算法分析实验课程	实验课时	12 课时
实验项目	哈夫曼生成树实现文件压缩	实验时间	2019. 12. 19
实验目的	<ol style="list-style-type: none"><li>1. 深入理解哈夫曼树生成应用原理</li><li>2. 了解常见文件压缩解压编码技术</li><li>3. 熟悉链表表示树及其遍历的方法</li></ol>		
实验环境	<ol style="list-style-type: none"><li>1. Visual Studio 2019</li><li>2. C++ 标准 2017 Std</li><li>3. Windows 10 x86_64</li></ol>		
实验内容	<p>问题描述：</p> <p>用哈夫曼编码设计一个压缩软件，能对输入的任何类型的文件进行哈夫曼编码，产生编码后的文件——压缩文件；也能对输入的压缩文件进行译码，生成压缩前的文件——解压文件。</p> <p>基本要求：</p> <ol style="list-style-type: none"><li>(1) 能对任何类型的文件进行压缩和解压的操作</li><li>(2) 要求编码和译码的效率尽可能高</li><li>(3) 如果时间和能力许可，可采用自适应形式的哈夫曼编码方</li></ol> <p>实验原理：</p> <p>哈夫曼树</p>		

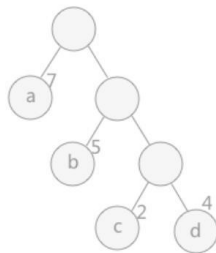
哈夫曼树又称最优树，是一类带权路径长度最短的树，权值较大的结点离根较近。

路径：在一棵树中，一个结点到另一个结点之间的通路，称为路径。

长度：在一条路径中，每经过一个结点，路径长度都要加 1

结点权：给每一个结点赋予一个新的数值，被称为这个结点的权。

结点带权长度：指的是从根结点到该结点之间的路径长度与该结点的权的乘积。



计算机中采用 0, 1 的不同排列来表示不同的字符，称为二进制编码。而哈夫曼树在数据编码中的应用，是数据的最小冗余编码问题，它是数据压缩学的基础。若每个字符出现的频率相同，则可以采用等长的二进制编码，若频率不同，则可以采用不等长的二进制编码，频率较大的采用位数较少的编码，频率较小的字符采用位数较多的编码，这样可以使字符的整体编码长度最小，这就是最小冗余编码的问题。而哈夫曼编码就是一种不等长的二进制编码，且哈夫曼树是一种最优二叉树，它的编码也是一种最优编码，在哈夫曼树中，规定往左编码为 0，往右编码为 1，则得到叶子结点编码为从根结点到叶子结点中所有路径中 0 和 1 的顺序排列，并且一个任意长的哈夫曼编码序列可以被唯一翻译为一个字符序列，这便为哈夫曼压缩打好了基础。

## 构造Huffman树的方法——Huffman算法

### ✦ 构造Huffman树步骤

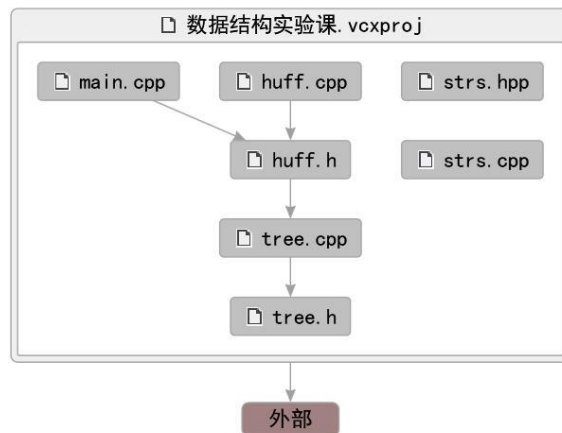
- ✦ 根据给定的 $n$ 个权值 $\{w_1, w_2, \dots, w_n\}$ ，构造 $n$ 棵只有根结点的二叉树，令其权值为 $w_j$
- ✦ 在森林中选取两棵根结点权值最小的树作左右子树，构造一棵新的二叉树，置新二叉树根结点权值为其左右子树根结点权值之和
- ✦ 在森林中删除这两棵树，同时将新得到的二叉树加入森林中
- ✦ 重复上述两步，直到只含一棵树为止，这棵树即哈夫曼树

## 文件压缩

文件压缩主要是利用哈夫曼编码来实现，但是得到编码之前需要构建哈夫曼树。因此需要统计每个字符出现的次数，用次数来构建哈夫曼树。假设小文件内容是“aaaabbbccd”，字符存在计算机中时以字节为单位的，因此我们需要将这些字符压缩成0、1表示的编码，0和1表示字节中的“位”，这样能大大降低文件的大小。

实验代码：

代码目录：



Tree: 链表树模块：

-----Tree.h-----

```

#pragma once
#include<iostream>
#define uint unsigned int
#define trtp tree<type>
template<class type>
class tree
{
public:
    type data;
    tree* lnex;
    tree* rnex;
    bool occp;           //是否存储数据
    tree();
    tree(type);
    void addc(bool, type);
    void show_fron();
    void show_midd();
    void show_back();
};
  
```

-----Tree.cpp-----

```
#include "tree.h"
template<class type>
trtp::tree() {
    this->data = *(new type());
    this->lnex = nullptr;
    this->rnex = nullptr;
}
template<class type>
trtp::tree(type inpu) {
    tree();
    data = inpu;
}
template<class type>
void trtp::addc(bool choo, type inpu) {
    tree* temp = new tree(inpu);
    if (!choo)
        this->lnex = temp;
    else
        this->rnex = temp;
}
template<class type>
void trtp::show_fron() {
    std::cout << this->data << " ";
    if (this->lnex != nullptr)
        this->lnex->show_fron();
    if (this->rnex != nullptr)
        this->rnex->show_fron();
}
template<class type>
void trtp::show_back() {
    if (this->lnex != nullptr)
        this->lnex->show_back();
    if (this->rnex != nullptr)
        this->rnex->show_back();
    std::cout << this->data << " ";
}
template<class type>
void trtp::show_midd() {
    if (this->lnex != nullptr)
        this->lnex->show_midd();
    std::cout << this->data << " ";
    if (this->rnex != nullptr)
        this->rnex->show_midd();}
```

Huff：哈夫曼树：

-----huff.h-----

```
#pragma once
#include <cstdio>
#include <iostream>
#include <vector>
```

```

#include "tree.cpp"
#pragma warning(disable:4996)
#define ulls unsigned long long
#define uchs unsigned char
#define uint unsigned int
#define trch tree<uchs>
#define vebl std::vector<bool>
class hutr :public tree<uchs> {
public:
    ulls weig;                //当前节点权重
    hutr() {
        occp = weig = data = 0;
        lnex = rnex = nullptr;};
class huff{
public:
    huff();                //初始构造函数
    FILE* file;            //存储文件指针
    FILE* ztfs;            //存储写入文件
    hutr* save;            //存储哈夫曼树
    ulls nums;             //记录字符数量
    uchs coun;             //记录统计次数
    ulls maps[256];        //记录每种概率
    uchs lens[256];        //记录编码长度
    vebl data[256];        //记录对应编码
    bool used[256];        //标记使用情况
    bool load(char*);      //统计每种概率
    void code();           //生成哈夫曼码
    uchs fmin();           //统计最小字符
    hutr* gtre(hutr*);     //递归哈夫曼树
    void gsha();           //生成速查编码
    void gatr(trch*,       //递归遍历生成
    vebl, uchs, bool);
    void gzip(char*);      //执行压缩操作
};

```

-----huff.cpp-----

```

#include "huff.h"
#include <iostream>
#include <vector>
    huff::huff() {
        for (int loop = 0; loop <= 255; loop++) {
            maps[loop] = 0; used[loop] = 0; coun = 1;
            lens[loop] = 0;
            used[loop] = false;}
        nums = 0;
        file = nullptr;}
bool huff::load(char* inpu) {
    file= fopen(inpu, "rb");
    if (inpu == nullptr) {
        std::cout << " [错误]文件无法读取 | " << (void*)file<< " | " << inpu <<
std::endl;
        return false;}
    uchs temp = fgetc(file);
    while(!feof ( file )) {
        this->maps[temp%256]++;
        nums++;
        temp = fgetc(file);}
    std::cout << " [成功]文件已经载入 | " << (void*)file<< " | " << inpu

```

```

<< std::endl;
    std::cout << " [成功]扫描文件内容 | " << (void*)maps<< " | S I Z E = "
<< nums << std::endl;
    return true;}
void huff::code() {
    uchs t1 = fmin();
    uchs t2 = fmin();
    hutr* tp = new hutr();
    hutr* t1 = new hutr();
    hutr* tr = new hutr();
    tp->weig = maps[t1] + maps[t2];
    t1->data = t1; tr->data = t2;
    t1->weig = 0; tr->weig = 0;
    t1->occp = 1; tr->occp = 1;
    tp->lnex = t1; tp->rnex = tr;
    hutr* temp=this->gtre(tp);
    this->save = temp;
    std::cout << " [成功]生成哈夫曼树 | " << (void*)save << " | M A X L = " <<
(int)temp->rnex->data << std::endl;}
hutr* huff::gtre(hutr* inpu) {
    if (coun == 255) return inpu;
    else {
        this->coun++;
        uchs next = this->fmin();
        hutr* tk = new hutr();
        tk->data = next;
        tk->weig = 0;
        tk->occp = 1;
        hutr* ts = new hutr();
        if (maps[next] > inpu->weig) {
            ts->lnex = inpu; ts->rnex = tk;}
        else {
            uchs tttp = fmin();
            if (maps[tttp] > inpu->weig) {
                ts->lnex = inpu; ts->rnex = tk;
                used[tttp] = 0;}
            else {
                coun++;
                hutr* t1 = new hutr();
                t1->weig = 0;
                t1->occp = 1;
                t1->data = tttp;
                hutr* su = new hutr();
                su->occp = 0;
                su->weig = maps[tttp] + maps[next];
                su->data = 0;
                su->lnex = t1;
                su->rnex = tk;
                ts->lnex = inpu;
                ts->rnex = su;
            }
        }
        ts->weig = maps[next] + inpu->weig;
        ts->occp = 0;
        ts->data = 0;
        this->gtre(ts);}}
uchs huff::fmin() {
    uchs temp = -1; bool flag = 0;

```

```

ulls minn =0xffffffffffffffff;
for (uchs loop = 0;; loop++) {
    if (used[loop] == false && minn >= maps[loop]) {
        temp = loop; minn = maps[loop]; flag = 1;}
    if (loop == 255) {
        if (used[255] == 0 && flag==0)
            return 255;
        else
            break;}}
    if(temp!=-1)used[temp] = true;
    return temp;}
void huff::gsha() {
    hutr* temp = this->save;
    vebl daul;
    uchs numt = 0;
    this->gatr(temp->rnex, daul, numt, 0);
    this->gatr(temp->lnex, daul, numt, 1);}
void huff::gatr(trch* inpu, vebl datp, uchs numt, bool flag) {
    datp.push_back(flag);
    numt++;
    if (inpu->occp == true) {
        lens[inpu->data] = numt;
        data[inpu->data] = datp;}
    if (inpu->lnex != nullptr)
        this->gatr(inpu->lnex, datp, numt, 1);
    if (inpu->rnex != nullptr)
        this->gatr(inpu->rnex, datp, numt, 0);}
void huff::gzip(char* path) {
    rewind(file);
    uchs temp = fgetc(file);
    vebl buff;
    while (!feof(file)) {
        vebl::iterator loop = data[temp%256].begin();
        for (;loop!= data[temp % 256].end(); ++loop){
            buff.push_back(*loop);}
        temp = fgetc(file);}
    ztfs = fopen(path, "wb+");
    uchs wrti;bool rwbl;uchs lenp = 0;
    vebl::iterator loop = buff.begin();
    for (; loop != buff.end(); ) {
        wrti = 0;
        for (lenp = 0; lenp <= 8; lenp++) {
            rwbl=buff.at(*loop);
            wrti += rwbl;
            wrti < 1;
            if (loop == buff.end())
                break;
            ++loop;}
        putc(wrti, ztfs);}
    std::cout << "[成功]写入压缩文件 | " << (void*)ztfs << " | L E N S = " <<
    buff.size() << "\n";
    fclose(ztfs);}

```




控制台输出：

Microsoft Visual Studio 调试控制台

```
[成功]文件已经载入      014F9D48      E:\CPP\PIKA-CPP-Datastruct\1.bmp.phz.phz
[成功]扫描文件内容      0113E278      S I Z E = 91025
[成功]生成哈夫曼树      01509280      M A X L = 0
[成功]写入压缩文件      014F7878      L E N S = 91025
```

文件管理器：

	1. bmp	2019/12/19 ...	BMP 文件	7,201 KB
	1. bmp. phz	2019/12/19 ...	PHZ 文件	801 KB
	1. bmp. phz. phz	2019/12/19 ...	PHZ 文件	89 KB
	1. bmp. phz. phz. phz	2019/12/19 ...	PHZ 文件	10 KB

压缩前文件：

```
2 4D 36 80 70 00 00 00 00 00 36 00 00 00 28 00
00 00 80 07 00 00 00 05 00 00 01 00 18 00 00 00
00 00 00 80 70 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 E8 A2 00 E8 A2 00 E8 A2 00 E8
A2 00 E8 A2 00 E8 A2 00 E8 A2 00 E8 A2 00 E8 A2
00 E8 A2 00 E8 A2 00 E8 A2 00 E8 A2 00 E8 A2 00
E8 A2 00 E8 A2 00 E8 A2 00 E8 A2 00 E8 A2 00 E8
A2 00 E8 A2 00 E8 A2 00 E8 A2 00 E8 A2 00 E8 A2
00 E8 A2 00 E8 A2 00 E8 A2 00 E8 A2 00 E8 A2 00
```

压缩后文件

```
DC AD B8 CA DC DF DC 98 CA DC E6 E6 53 AD CA C9
E6 59 AD DC DF CA C9 CA 15 CA C9 15 1E 4E 2F 2F
C9 C9 B7 C9 C9 C9 B7 C9 C9 C9 B7 C9 C9 C9 CE BF
03 C1 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4
C8 D0 C6 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4
C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4
C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4
C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4
```

从上图得知，实验完成，成功使用哈夫曼树将文件压缩

在我自己课程设计中，就在编写好源代码后的调试中出现了不少的错误，遇到了很多麻烦及困难，在程序设计过程中，我学到了哈夫曼编码以及其应用，同时也意识到自己还有很多不足之处，有些问题无法自己独立解决，有些时候又无法寻求到最优的算法，希望在以后的学习中可以更好地提升自我！

成绩评定：

指导教师签名：