

学生学号	0121710880414	实验课成绩	
------	---------------	-------	--

武汉理工大学 学 生 实 验 报 告 书

实验课程名称	数据结构与算法综合实验
开 课 学 院	计算机科学与技术学院
指导教师姓名	夏红霞
学 生 姓 名	穆逸诚
学生专业班级	软件 1704 班

2018 -- 2019 学 年 第 二 学 期

实验教学管理基本规范

实验是培养学生动手能力、分析解决问题能力的重要环节；实验报告是反映实验教学水平与质量的重要依据。为加强实验过程管理，改革实验成绩考核方法，改善实验教学效果，提高学生质量，特制定实验教学管理基本规范。

- 1、本规范适用于理工科类专业实验课程，文、经、管、计算机类实验课程可根据具体情况参照执行或暂不执行。
- 2、每门实验课程一般会包括许多实验项目，除非常简单的验证演示性实验项目可以不写实验报告外，其他实验项目均应按本格式完成实验报告。
- 3、实验报告应由实验预习、实验过程、结果分析三大部分组成。每部分均在实验成绩中占一定比例。各部分成绩的观测点、考核目标、所占比例可参考附表执行。各专业也可以根据具体情况，调整考核内容和评分标准。
- 4、学生必须在完成实验预习内容的前提下进行实验。教师要在实验过程中抽查学生预习情况，在学生离开实验室前，检查学生实验操作和记录情况，并在实验报告第二部分教师签字栏签名，以确保实验记录的真实性。
- 5、教师应及时评阅学生的实验报告并给出各实验项目成绩，完整保存实验报告。在完成所有实验项目后，教师应按学生姓名将批改好的各实验项目实验报告装订成册，构成该实验课程总报告，按班级交课程承担单位（实验中心或实验室）保管存档。
- 6、实验课程成绩按其类型采取百分制或优、良、中、及格和不及格五级评定。

附表：实验考核参考内容及标准

	观测点	考核目标	成绩组成
实验预习	1. 预习报告 2. 提问 3. 对于设计型实验，着重考查设计方案的科学性、可行性和创新性	对实验目的和基本原理的认识程度，对实验方案的设计能力	20%
实验过程	1. 是否按时参加实验 2. 对实验过程的熟悉程度 3. 对基本操作的规范程度 4. 对突发事件的应急处理能力 5. 实验原始记录的完整程度 6. 同学之间的团结协作精神	着重考查学生的实验态度、基本操作技能；严谨的治学态度、团结协作精神	30%
结果分析	1. 所分析结果是否用原始记录数据 2. 计算结果是否正确 3. 实验结果分析是否合理 4. 对于综合实验，各项内容之间是否有分析、比较与判断等	考查学生对实验数据处理和现象分析的能力；对专业知识的综合应用能力；事实求实的精神	50%

实验课程名称： 算法设计与分析实验

实验项目名称	二叉树与赫夫曼图片压缩			实验成绩	
实 验 者	穆逸诚	专业班级	软件 1704	组 别	
同 组 者				实验日期	2019 年 4 月 22 日

第一部分：实验分析与设计

一、实验目的和要求

(1) 实验目的：

- ①掌握树的存储结构；
- ②掌握二叉树的三种遍历方法；
- ③理解 Huffman 树；
- ④理解 Huffman 编码；
- ⑤掌握文件的操作；
- ⑥5 使用 Huffman 算法实现图像压缩程序。

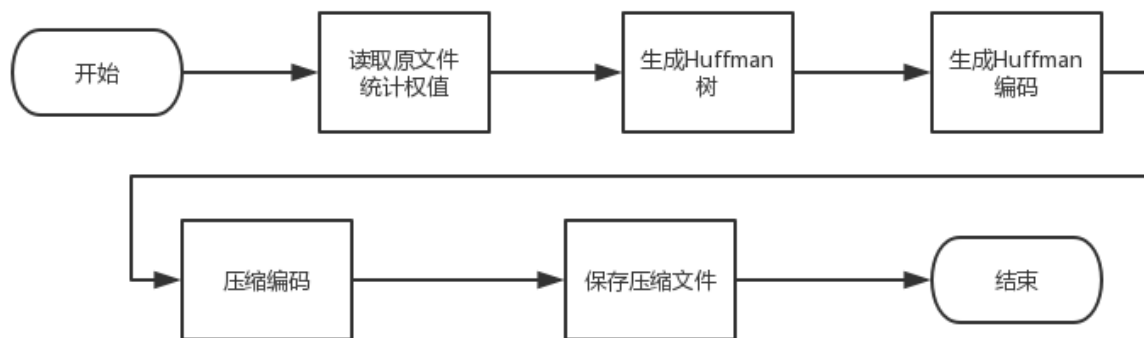
(2) 实验要求：

使用 Huffman 压缩算法，对一幅 BMP 格式的图片文件进行压缩。图片文件名为“Pic.bmp”，内容自定。压缩后保存为“Pic.bmp.huf”文件。使用 VS 作为开发工具，开发一个控制台程序，使用 Huffman 压缩算法对图片文件“Pic.bmp”进行压缩，具体要求如下：

- ①读取源文件，统计权值；
- ②生成 Huffman 树；
- ③生成 Huffman 编码；
- ④压缩源文件；
- ⑤保存压缩文件。

二、实验分析与设计

程序流程图：



1、数据结构设计

(1) 定义存储结构

定义一个结构体 HTNode 来表示二叉树的叶子节点，记录每个节点的权值、父节点、左孩子和右孩子。动态分配数组存储 Huffman 树。

```

typedef struct
{
    int weight;    //权值
    int parent;    //父节点

```

```

    int lchild;//左孩子
    int rchild;//右孩子
}HTNode, *HuffmanTree;

```

(2) 遍历算法

遍历二叉树，需根据叶子节点的路径生成 Huffman 编码表。本程序采用先序遍历的方式查找二叉树上所有的叶子节点，生成对应的 Huffman 编码。

(3) 压缩编码

由于 Huffman 编码表是以字符数组的形式保存的，重新编码后的数据将是一个很长的字符串。定义 Str2byte 函数，将形如“01010101”字符串转换成字节，才能得到最终的编码。将其保存到“.huf”文件中，即实现了文件的压缩。

(4) 程序设计

使用 VS2017 开发工具，创建一个空的控制台工程（Win32 Console Application）。利用树的存储结构和 Huffman 编码方法，使用 C++ 语言开发图像压缩编码程序，工程名为 HfmCompressCPro。

- ①创建 Main.cpp 文件，定义 int main（）函数，作为程序的入口函数；
- ②创建 Huffman.h 文件与 Huffman.cpp 文件，实现 Huffman 算法相关的函数；
- ③创建 Compress.h 与 Compress.cpp 文件，实现文件的压缩算法。

2、头文件函数展示

(1) Huffman.h

```

#ifndef _HUFFMAN_H_
#define _HUFFMAN_H_
#define ERROR 0
#define OK 1
#define SIZE 256
#define _CRT_SECURE_NO_WARNINGS
//Huffman 树节点
struct HTNode {
    int weight;//权值
    int parent;//父节点
    int lchild;//左孩子
    int rchild;//右孩子
};
//Huffman 树
typedef HTNode *HTree;//动态分配数组存储 Huffman 树
//Huffman 编码表
typedef char **HCode;//动态分配数组存储 Huffman 编码表
int HuffmanTree(HTree &pHT, int *w, int n);//用于生成 Huffman 树
int HuffmanCoding(HCode &pHC, HTree &pHT);//用于生成 Huffman 编码
int Select(HTree pHT, int nSize);
void CreatHC(HCode &HC, HTree &HT, int n);
#endif // !_HUFFMAN_H_

```

(2) Compress.h

```

#ifndef _COMPRESS_H_

```

```

#define _COMPRESS_H_
#define _CRT_SECURE_NO_WARNINGS
#include "Huffman.h"
//文件头
struct HEAD {
    char type[4];//文件类型
    int length;//原文件长度
    int weight[256];//权值数值
};
//缓冲区
typedef char *BUFFER;
int Compress(const char *pFilename);//用于实现文件压缩
char Str2byte(const char *pBinStr);//用于将“01010101”形式的字符串转化为字节
int Encode(const char *pFilename, const HCode pHc, BUFFER &pBuffer, const int nSize);//利用 Huffman 编码实现文件压缩
int InitHead(const char *pFilename, HEAD &sHead);//读取原文件，初始化文件头数据信息
int WriteFile(const char *pFilename, const HEAD sHead, const BUFFER pBuffer, const int nSize);//用于将压缩后的数据写入新的文件
#endif // !_COMPRESS_H_

```

3、核心算法展示

(1) 创建 Huffman 树

```

int HuffmanTree(HTree &pHT, int *w, int n) {
    //开辟空间
    int m = 2 * SIZE - 1;
    pHT = (HTree)malloc((m + 1) * sizeof(HTNode));
    if (!pHT) {
        cerr << "内存分配失败" << endl;
        return ERROR;
    }
    //初始化树
    HTree p = pHT + 1;//0 号单元未使用
    for (int i = 0; i < m; i++) {
        if (i < n) {
            p->weight;
        }
        else {
            p->weight = 0;
        }
        p->parent = 0;
        p->lchild = 0;
        p->rchild = 0;
        p++;
    }
    //创建 Huffman 树
}

```

```

    for (int i = SIZE + 1; i <= m; i++) {
        //第 1 个最小元素
        int s1 = Select(pHT, i - 1);
        pHT[s1].parent = i;
        //第 2 个最小元素
        int s2 = Select(pHT, i - 1);
        pHT[s2].parent = i;
        pHT[i].weight = pHT[s1].weight + pHT[s2].weight;
        pHT[i].lchild = s1;
        pHT[i].rchild = s2;
    }
    return 0;
}

(2) 生成 Huffman 编码
int HuffmanCoding(HCode &pHC, HTree &pHT) {
    //无栈非递归遍历 Huffman 树, 求 Huffman 编码
    char cd[256] = { '\0' }; //记录访问路径
    int cdlen = 0; //记录当前路径长度
    for (int i = 1; i < 512; i++) {
        pHT[i].weight = 0; //遍历 Huffman 树时用作节点的状态标志
    }
    int p = 511; //根节点
    while (p != 0) {
        if (pHT[p].weight == 0) { //向左
            pHT[p].weight = 1;
            if (pHT[p].lchild != 0) {
                p = pHT[p].lchild;
                cd[cdlen++] = '0';
            }
            else if (pHT[p].rchild == 0) { //登记叶子节点的字符的编码
                pHC[p] = (char*)malloc((cdlen + 1) * sizeof(char));
                cd[cdlen] = '\0';
                strcpy(pHC[p], cd); //复制代码
            }
        }
        else if (pHT[p].weight == 1) { //向右
            pHT[p].weight = 2;
            if (pHT[p].rchild != 0) { //右孩子为叶子节点
                p = pHT[p].rchild;
                cd[cdlen++] = '1';
            }
        }
        else { //退回父节点, 编码长度减 1
            pHT[p].weight = 0;

```

```

        p = pHT[p].parent;
        cdlen--;
    }
}
return OK;
}
(3) 压缩编码算法
int Encode(const char *pFilename, const HCode pHC, BUFFER &pBuffer, const int nSize) {
    //以二进制流形式打开文件
    FILE *in = fopen(pFilename, "rb");
    if (!in) {
        cerr << "打开源文件失败" << endl;
    }
    //开辟缓冲区
    pBuffer = (char*)malloc(nSize * sizeof(char));
    if (!pBuffer) {
        cerr << "开辟缓冲区失败" << endl;
        return ERROR;
    }
    char cd[SIZE] = { 0 }; //工作区
    int pos = 0; //缓冲区指针
    int ch;
    //扫描文件，根据 Huffman 编码表对其进行压缩，压缩结果暂存到缓冲区中
    while ((ch = fgetc(in)) != EOF) {
        strcat(cd, pHC[ch+1]); //从 HC 复制编码到 cd
        //压缩编码
        while (strlen(cd) >= 8) {
            //截取字符串左边的 8 个字符，编码成字节
            pBuffer[pos++] = Str2byte(cd);
            //字符串整体左移 8 字节
            for (int i = 0; i < SIZE - 8; i++) {
                cd[i] = cd[i + 8];
            }
        }
    }
    if (strlen(cd) > 0) {
        pBuffer[pos++] = Str2byte(cd);
    }
    fclose(in);
}

```

三、主要仪器设备及耗材

1. PC 机
2. 开发环境：VS2017

第二部分：实验调试与结果分析

一、调试过程（包括调试方法描述、实验数据记录，实验现象记录，实验过程发现的问题等）

1. 调试方法描述

- ① 输入 C++ 程序，并保存；
- ② 编译 C++ 程序，找出程序的语法错误并改正；
- ③ 输入测试数据（除每区域普遍值外包括一些特殊临界值），运行 C++ 程序，若有错，查找并修改程序的逻辑错误；
- ④ 重复②-③步，直到得到正确的运行结果

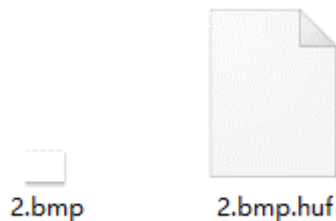
2. 实验输入/输出数据记录

—>★具体实验操作截图如下：

（1）输入 IDCard.bmp 文件



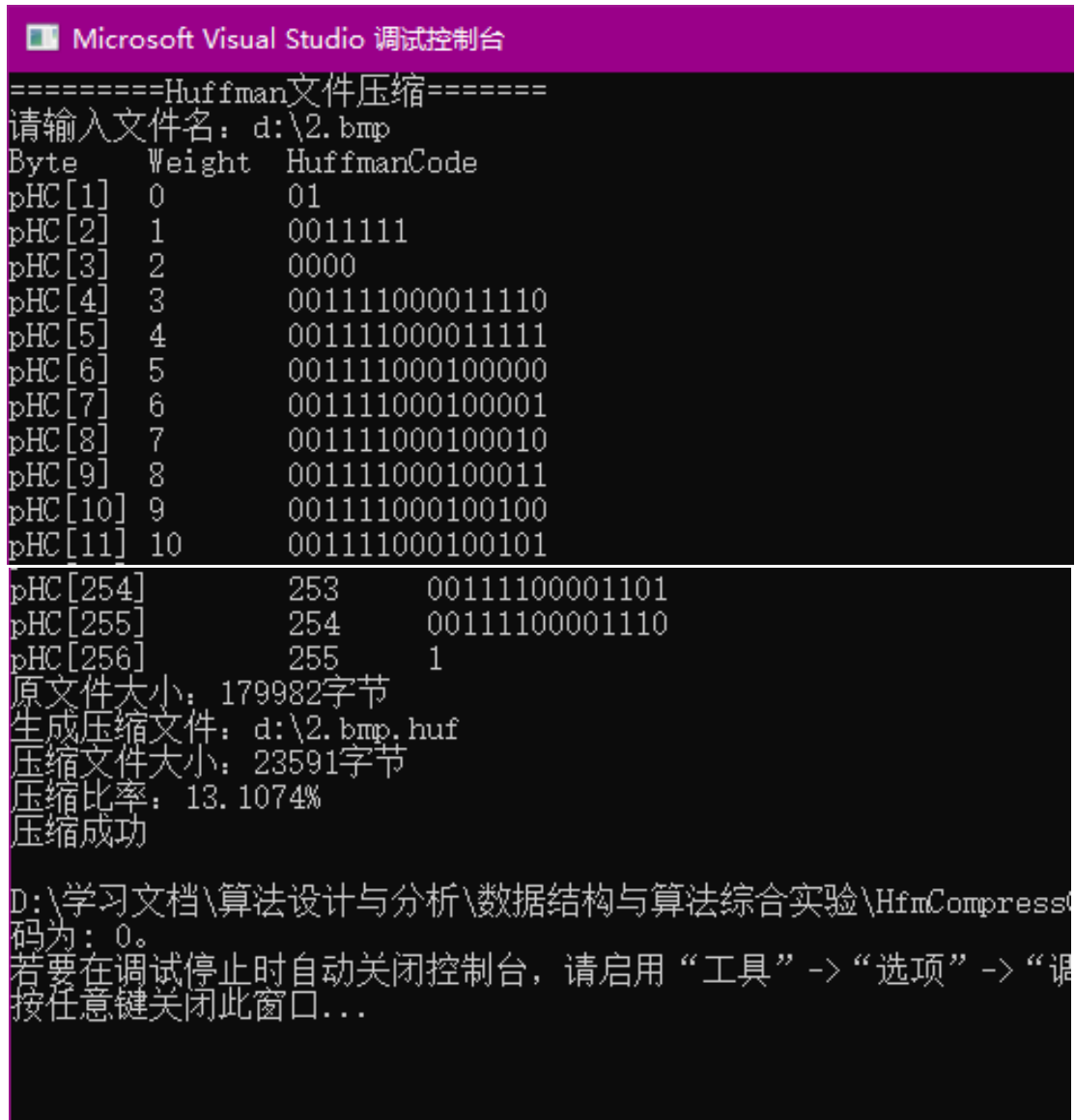
（2）输出 IDCard.bmp.huf 文件



（3）控制台输入

```
D:\学习文档\算法设计与分析\数据结构与算法综合实验\HfmCompressCPro2\Debug\HfmCompressCPro2.exe
=====Huffman文件压缩=====
请输入文件名: d:\2.bmp
```


(4) 控制台输出哈夫曼编码与相关输出信息



```
Microsoft Visual Studio 调试控制台
=====Huffman文件压缩=====
请输入文件名: d:\2.bmp
Byte    Weight  HuffmanCode
pHC[1]   0      01
pHC[2]   1      0011111
pHC[3]   2      0000
pHC[4]   3      0011110000011110
pHC[5]   4      0011110000011111
pHC[6]   5      001111000100000
pHC[7]   6      001111000100001
pHC[8]   7      001111000100010
pHC[9]   8      001111000100011
pHC[10]  9      001111000100100
pHC[11] 10      001111000100101
pHC[254] 253     00111100001101
pHC[255] 254     00111100001110
pHC[256] 255     1
原文件大小: 179982字节
生成压缩文件: d:\2.bmp.huf
压缩文件大小: 23591字节
压缩比率: 13.1074%
压缩成功

D:\学习文档\算法设计与分析\数据结构与算法综合实验\HfmCompress
码为: 0。
若要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调
按任意键关闭此窗口...
```

3. 实验过程发现的问题

(1) 本次实验中涉及到了多处文件操作与指针设计的地方, 这两方面涉及到的知识点较多交杂, 因而在编程过程中遇到了格式不熟悉, 方法不明确的问题, 如文件打开与关闭的方法 (fopen 括号内的内容)、指针 “*” 符号的运用 (与 “&” 的含义、命名规则等)。这些问题最后通过查阅书籍与上网查找相关资料得以解决;

(2) 在选取二叉树的存储结构时, 可以使用顺序结构和链式结构。经过分析与考虑, 我发现 Huffman 树的叶子节点数为 256, 其存储空间是固定的, 适合使用顺序结构来表示。因此, 我最后采用了结构体数组来存储 Huffman 树;

(3) 在遍历 Huffman 树时, 可以采用先序遍历、中序遍历或者后序遍历三种方式。我选择了先序遍历 Huffman 树的方法, 但我发现要有记录访问每个叶子节点的路径, 因此需要对先序遍历算法进行一些改进。经过分析, 我定义了一个二位字符串数组来存放所有叶子节点的编码, 并记录访问路径 (左孩子为 “0”, 右孩子为 “1”), 同时只记录叶子节点, 访问到叶子节点时, 保存路径。

第三部分 实验小结、建议及体会

本次实验主要围绕二叉树与赫夫曼图片压缩展开，实验难度适中，涉及到了文件操作、压缩、遍历算法与二叉树方面的知识，考验我们的综合编程能力与解决问题的能力。压缩软件是利用特定算法来压缩数据的工具，压缩后生成的文件称为压缩包（Archive）。利用压缩软件对文件中重复的数据进行压缩，可以减小文件中的字节总数，使文件更加精简。使用 Huffman 编码对原文件中的字节重复编码。重复次数较多的字节，Huffman 编码的长度较短。原文件中每个字节的编码长度都是 8 位，而重复次数较多的字节的 Huffman 编码，长度比 8 位短，因而可以实现压缩。

实验过程中，难免遇到许多问题，如指针运用不熟，文件操作失误等。指针是 C++ 语言中一个重要的内容，通过指针，可以简化一些 C++ 编程任务的执行，还有一些任务，比如动态内存分配，没有指针时不能执行的。而文件操作是本次实验的必要环节，文件操作的失误将直接导致程序无法执行。通过此次实验，我强化了对指针的运用，掌握了文件操作的方法，使得程序更加精简，结构更加清晰，并且能正确运行。

同时，对于二叉树的存储结构与遍历算法，我也进行了一定的分析。在选取二叉树的存储结构时，可以使用顺序结构和链式结构。经过分析与考虑，我发现 Huffman 树的叶子节点数为 256，其存储空间是固定的，适合使用顺序结构来表示。因此，我最后采用了结构体数组来存储 Huffman 树。在遍历 Huffman 树时，可以采用先序遍历、中序遍历或者后序遍历三种方式。我选择了先序遍历 Huffman 树的方法，但我发现要有记录访问每个叶子节点的路径，因此需要对先序遍历算法进行一些改进。经过分析，我定义了一个二位字符串数组来存放所有叶子结点的编码，并记录访问路径（左孩子为“0”，右孩子为“1”），同时只记录叶子节点，访问到叶子节点时，保存路径。

通过本次实验，我学习了有关二叉树与 Huffman 图片压缩的知识，了解了压缩的原理，明白了指针的一些运用与文件的相关操作，掌握了树的存储结构与二叉树的三种遍历方法，最终使用 Huffman 算法实现了图像压缩程序，并且可以正确运行。总体而言，本次实验意义鲜明，收获颇丰。

教师签字_____