《用哈夫曼编码实现文件压缩》

实 验 报 告

课程名称 《数据结构 B》

实验学期 2017 至 2018 学年 第 1 学期

学生所在院部 计算机学院

年级 2016 专业班级 网络 B16-2

学生姓名 朱彦东 学号 201607024205

任课教师 朱冬梅

实验成绩

一、实验题目：

用哈夫曼编码实现文件压缩

二、实验目的：

1、了解文件的概念。
2、掌握线性链表的插入、删除等算法。
3、掌握 Huffman 树的概念及构造方法。
4、掌握二叉树的存储结构及遍历算法。
5、利用 Huffman 树及 Huffman 编码，掌握实现文件压缩的一般原理。

三、实验设备与环境：

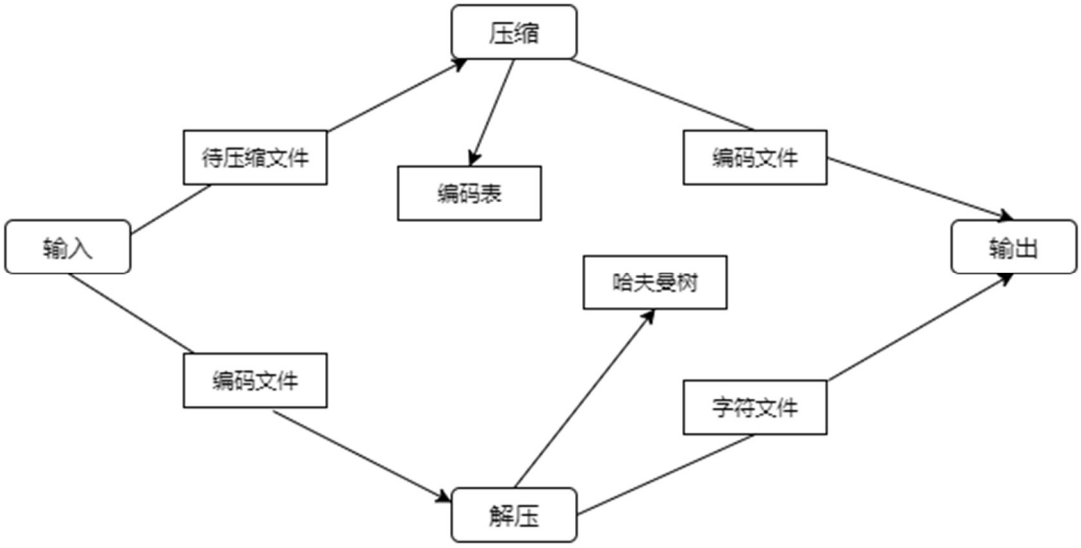
微型计算机、Windows 系列操作系统 、Visual C++6.0 软件

四、实验内容：

根据 ASCII 码文件中各 ASCII 字符出现的频率情况创建 Haffman 树，再将各字符对应的哈夫曼编码写入文件中，实现文件压缩。

五、概要设计：

1、原理分析图



2、函数实现

```
1. 两个重要的结点结构体
// 统计字符频度的临时结点
typedef struct {
  unsigned char uch;          // 以8bits为单元的无符号字符
  unsigned long weight;       // 每类（以二进制编码区分）字符出现频度
} TmpNode;
```

```
// 哈夫曼树结点
typedef struct {
  unsigned char uch;                  // 以8bits为单元的无符号字符
  unsigned long weight;                  // 每类（以二进制编码区分）字符出现
频度
  char *code;                         // 字符对应的哈夫曼编码（动态分配存储空
间）
  int parent, lchild, rchild;         // 定义双亲和左右孩子
} HufNode, *HufTree;
```

2. 用于建立哈夫曼树和生成哈夫曼编码

```
  // 选择最小和次小的两个结点，建立哈夫曼树调用
  void select(HufNode *huf_tree, unsigned int n, int *s1, int *s2)
  // 建立哈夫曼树
  void CreateTree(HufNode *huf_tree, unsigned int char_kinds, unsigned
int node_num)
  // 生成哈夫曼编码
  void HufCode(HufNode *huf_tree, unsigned char_kinds)
```

说明：Select 函数供 CreateTree 函数调用，找两个最小的结点，找到第一个后需要将其 parent 设为‘1’（初始化后为‘0’）表明此结点已被选中。建立哈夫曼树过程中，每次用 select()函数找两个最小结点。

3. 主要函数——压缩解压函数。

```
  // 压缩函数
  int compress(char *ifname, char *ofname)
  // 解压函数
  int extract(char *ifname, char *ofname)
```

3、文件压缩思路

为了建立哈夫曼树，首先扫描源文件，统计每类字符出现的频度（出现的次数），然后根据字符频度建立哈夫曼树，接着根据哈夫曼树生成哈夫曼编码。再次扫描文件，每次读取 8bits，根据“字符—编码”表，匹配编码，并将编码存入压缩文件，同时存入编码表。

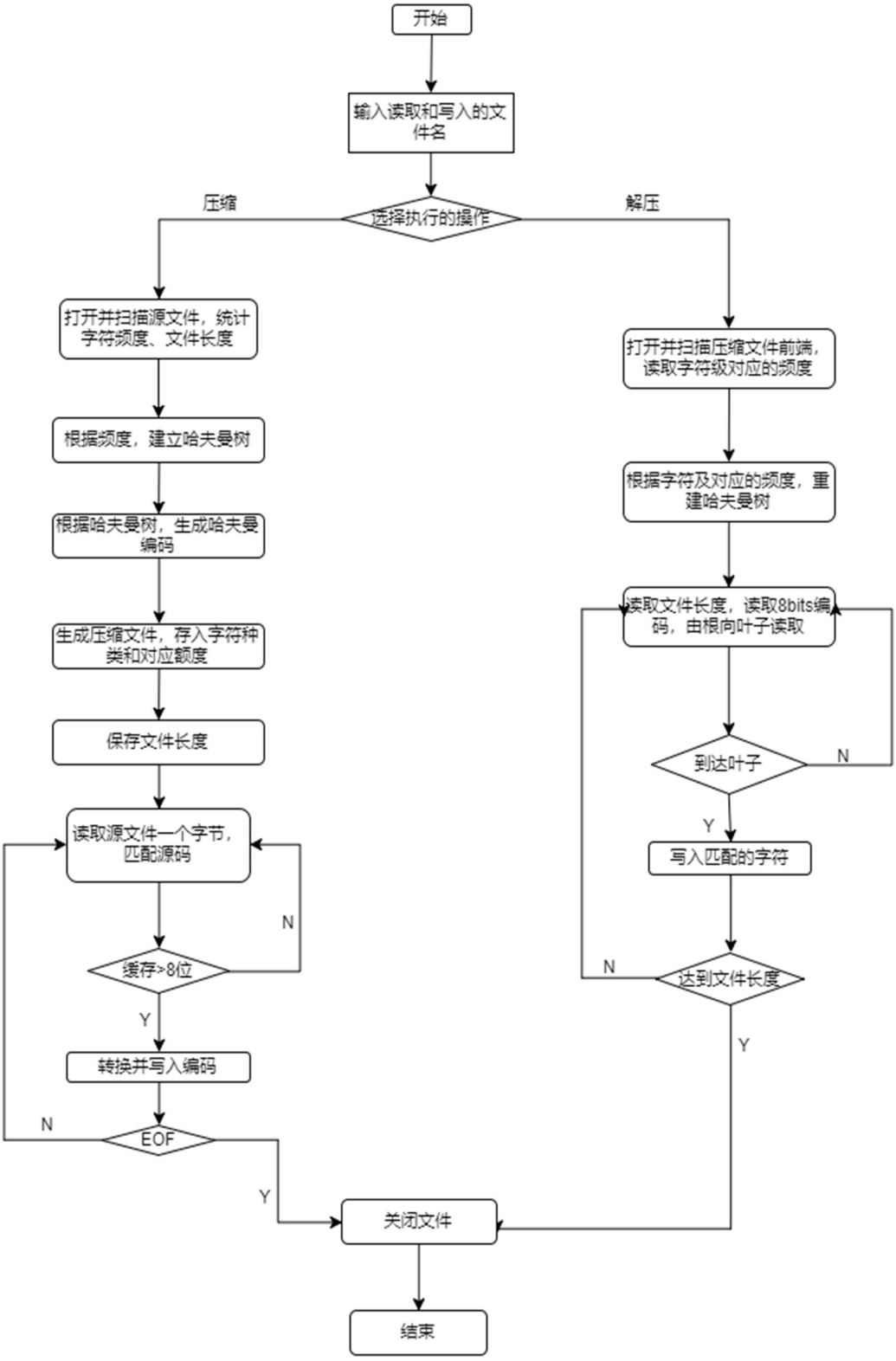
4、文件解压思路

读取编码表，然后读取编码匹配编码表找到对应字符，存入文件，完成解压。

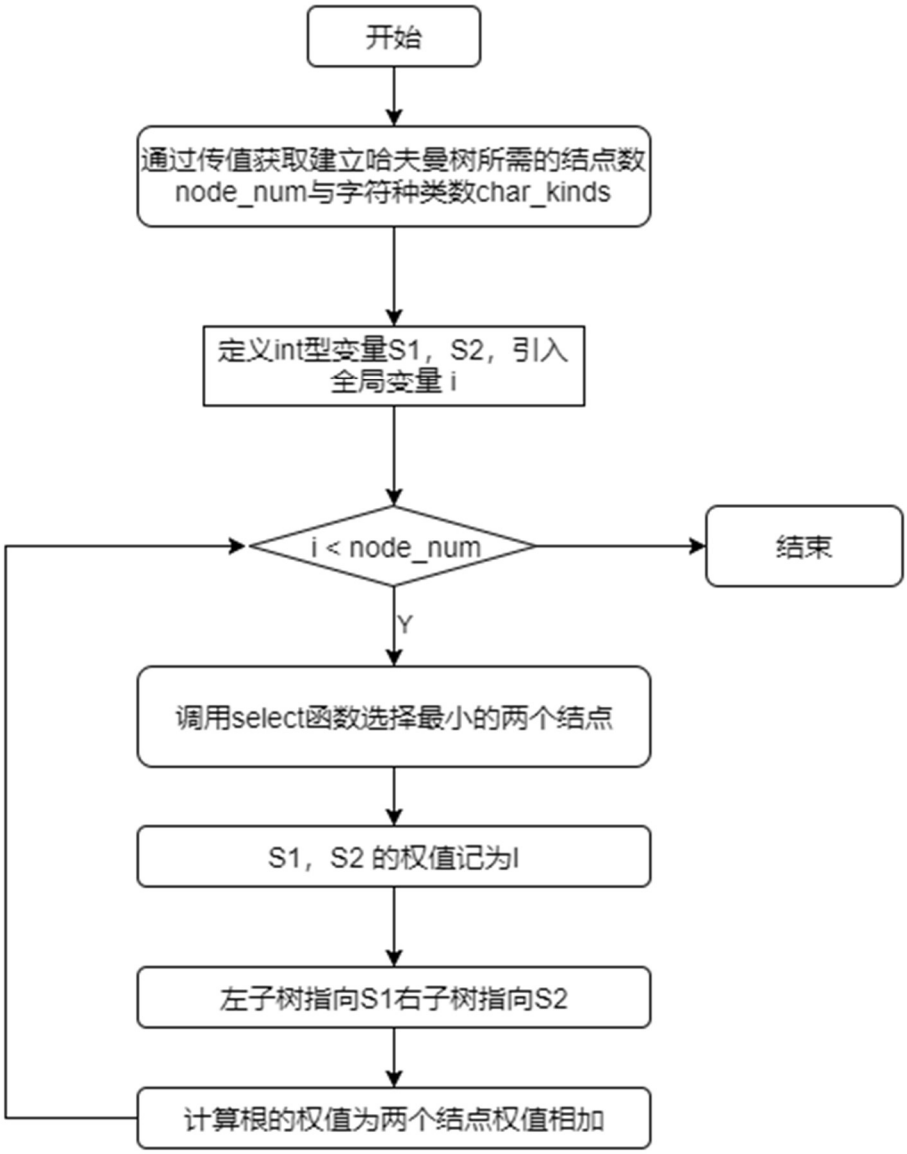
压缩解压的第一步就是读取文件，为了能够处理任何格式的文件，采用二进制方式读写文件。以一个无符号字符（unsigned char）的长度 8 位为处理单元，最多有 256（0~255）种组合，即 256 类字符

六、详细设计：

1、程序整体运行流程图



3、 构造哈夫曼树流程图



对应函数为：`void CreateTree(HufNode *huf_tree, unsigned int char_kinds, unsigned int node_num)`

分析：哈夫曼树为二叉树，树结点含有权重（在这里为字符频度，同时也要把频度相关联的字符保存在结点中）、左右孩子、双亲等信息。

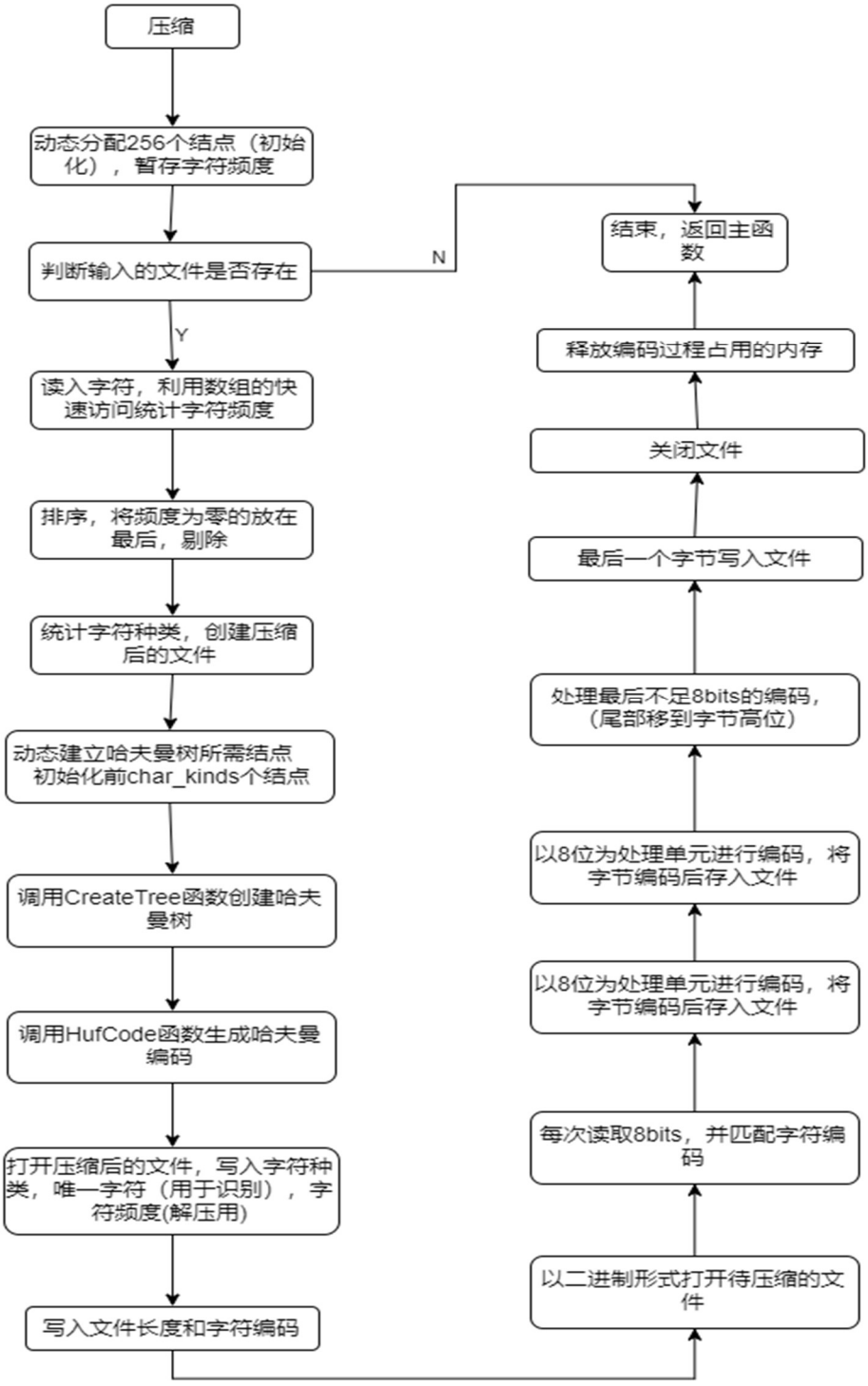
考虑到建立哈夫曼树所需结点会比较多，也比较大，如果静态分配，会浪费很大空间，故我们打算用动态分配的方法，并且，为了利用数组的随机访问特性，也将所需的所有树节点一次性动态分配，保证其内存的连续性。另外，结点中存储编码的域，由于长度不定，也动态分配内存。

4、 生成哈夫曼编码

对应函数为：`void HufCode(HufNode *huf_tree, unsigned char_kinds)`

分析：每类字符对应一串编码，故从叶子结点（字符所在结点）由下往上生成每类字符对应的编码，左‘0’，右‘1’。为了得到正向的编码，设置一个编码缓存数组，从后往前保存，然后从前往后拷贝到叶子结点对应编码域中，根据上面“建立哈夫曼树的协商”的约定，需要根据得到的编码长度为编码域分配空间。对于缓存数组的大小，由于字符种类最多为 256 种，构建的哈夫曼树最多有 256 个叶子结点，树的深度最大为 255，故编码最长为 255，所以分配 256 个空间，最后一位用于保存结束标志。

5、 压缩文件



对应函数：int compress(char *ifname, char *ofname)

分析：首先将字符及种类和编码（编码表）存储于压缩文件中，供解压时使用。

然后以二进制打开源文件，每次读取一个 8 位的无符号字符，循环扫描匹配存储于哈夫曼树节点中的编码信息。

由于编码长度不定，故需要一个编码缓存，待编码满足 8 位时才写入，文件结束时缓存中可能不足 8 位，在后面补 0，凑足 8 位写入，并将编码的长度随后存入文件。

在哈夫曼树节点中，编码的每一位都是以字符形式保存的，占用空间很大，不可以直接写入压缩文件，故需要转为二进制形式写入；至于如何实现，可以定义一个函数，将保存编码的字符数组转为二进制，但是比较麻烦，效率也不高；正好，可以利用 C 语言提供的位操作（与、或、移位）来实现，每匹配一位，用"或"操作存入低位，并左移一位，为下一位腾出空间，依次循环，满足 8 位就写入一次。

6、解压文件：

对应函数：int extract(char *ifname, char *ofname)

分析：以二进制方式打开压缩文件，首先将文件前端的字符种类数读取出来，读取数据重建哈夫曼树（双重循环，每个循环的次数最大为 511），据此动态分配足够空间，从树根到叶子对比编码，只要一次遍历就可以找到编码对应的存于叶子结点中的字符，极大提高了效率。

压缩文件为二进制文件，feof 在这里无法正确判断结束，故用一个死循环处理编码，以压缩时存储的文件长度来控制循环的结束。每当 root 小于 char_kinds，就匹配到了一个字符，是因为字符的下标范围是 0~char_kinds-1。

7、主函数

对应函数：int main()

分析：程序的入口，获取输入的文件名和压缩后的文件名，并调用各函数实现压缩解压的功能，并提供用户界面，与用户进行交互、

为防止程序出错，在 main 函数中加入压缩解压函数是否异常退出的判断：

使用说明：

压缩和解压时要求输入源文件和目标文件，可以输入完整的路径名加文件名，也可以仅输入一个文件名（默认在当前运行目录下寻找），如果不小心输错源文件名或源文件不存在，将提示出错，然后可以再次输入。

注意：输入文件名时请加上后缀名，否则无法识别。

七、测试结果及分析：

1、文件压缩：

```
请选择要执行的功能:
 1: 压缩
 2: 解压
 3: 退出
1
请输入要执行的文件名: 001.txt
请输入要输出的文件名: 003.zip
Compressing……
 操作完成!
```

2、文件解压

```
请选择要执行的功能:
 1: 压缩
 2: 解压
 3: 退出
2
请输入要执行的文件名: 003.zip
请输入要输出的文件名: 002.txt
Extracting……
 操作完成!
```

3、总结体会

实现哈夫曼树编码的过程中，首先构建哈夫曼树，并用动态分配数组存储，也用动态分配数组存储哈夫曼编码表。

在编写程序时也遇到一些问题，开始统计字符串中出现的各种字符及其次数时，将字母存放数组中，但是考虑到字母出现的不同，完全初始化再统计其出现的个数，不仅占用空间并且时间复杂度高。后来我在初步编码时，发现一些问题：解码后无法得到完全正确的源文件，经过排查，发现以 EOF 判断压缩文件的结束不可取，因为压缩文件是二进制文件，而 EOF 一般用来判断非二进制文件的结束，所以我们加入了文件长度来控制。

哈夫曼编码对文本文件，一般可以达到大约 2:1 的压缩比，特别是有规律的文本文件，可以达到高于 2:1 的压缩比，而对于图像等特殊文件压缩比几乎为 1:1，效果不理想。

通过这一个压缩和解压程序的设计，我学习了 UML 的使用，提升了编码能力，提升了调试能力，总之，受益匪浅。

八、教师评语：

教师评价	评定项目	A	B	C	D	评定项目	A	B	C	D
	算法正确					界面美观，布局合理				
	程序结构合理					操作熟练				
	语法、语义正确					解析完整				
	实验结果正确					文字流畅				
	报告规范					题解正确				
	其他： 评价教师签名： 年 月 日									