

“数据结构”

课程设计报告

**设计题目**  哈夫曼树编码文件压缩

**姓名**  xxx

**学号** xxx

**专业**  计算机科学与技术

**班级**  21-5班

**完成日期**  2022/6/25

1. **需求和规格说明**

哈夫曼树编码文件压缩

采用哈夫曼编码思想实现文本的编码（压缩）和解码功能。

① 交互输入文本哈夫曼编码、解码。

② 文本文件的哈夫曼编码、解码。编码文本和解码文本存入文件。

③ 计算文本的压缩比。

④ 计算带权路径长度WPL。

⑤ 打印构造的哈夫曼树。

⑥ 打印哈夫曼编码表。

⑦ 对文本文件比较原文和解码文件的相同性对比。

⑧ 要求原文件的规模应不小于5KB。

**（二） 设计**

首先,根据需求，创建信息储存结构。创建结构体HTNode作为哈夫曼的的结点，结点储存的信息有元素的值（字符型），权重（双精度浮点型），父节点，左右子节点等。将char\*\*重定义为HuffmanTree，用来存放数据的哈夫曼编码，其中0位置不用。其次，根据实现的功能，设计用户界面，压缩文件、解压缩文件、打印哈夫曼树、打印哈夫曼表、计算带路径长度WPL、系统关闭功能等，代码方面用数字给各功能编号，并用switch结构筛选用户的功能选择。之后，特别的，对与文件的压缩，先遍历文件，统计各个字符ch出现的次数，放入数组arr的ch位置上。然后，根据数组长度，动态的创建哈夫曼树的大小，如果有n中不同的字符，那么哈弗曼树的结点数就为2n+1，同样的，下标为0的位置不用。创建哈夫曼树，每次从数组中选出，然后便可以根据哈夫曼树得到哈夫曼编码了。之后便可以读取单次原文件的字符，将字符换成对应的哈夫曼码写入文件，这里借助循环队列，将哈夫曼码首先存入循环队列中，当循环队列足够大的时候，便每次从队列中取出8位0,1，转换成字符型写入文件，即可。对于解压缩，读取文件，然后根据字节比较，得到对应的字符，写入文件即可。而打印输出哈夫曼树，和哈夫曼表的时候，将会在控制台打印，同时写入文件一份，用 同名但不同后缀的\*.tree 和 \*.table文件存放，便于用户操作。最后，退出系统时，释放动态分配给哈夫曼树和哈夫曼编码的那部分空间释放，系统结束。

**属性和方法定**义

|  |  |  |  |
| --- | --- | --- | --- |
|  | 成员 | 类型 | 描述 |
| Queue  （循环队列） | Data | Char | 队列存放的数据类型 |
| Front | Int | 头指针 |
| Rear | Int | 尾指针 |
| Size | Int | 队列的存放内容的大小 |

|  |  |  |  |
| --- | --- | --- | --- |
| **宏定义** | **名称** | **大小** | **描述** |
| SIZE | 128 | 参与Huffman编码的字符的的范围 |
| QUQUEMAXLEN | 100 | 循环队列的最大空间 |

|  |  |
| --- | --- |
| **全局变量** | **功能** |
| HuffmanTree HT; | 存放哈夫曼树的结点信息 |
| HuffmanCode HC; | 存放数据的哈法曼编码值 |
| string FILEPATH; | 存放文件的路径（除去后缀） |
| int PL[SIZE+2] | 用于统计的数组 |
| seqQueue \*queue | 循环队列 |

|  |  |  |  |
| --- | --- | --- | --- |
| 结构体 | **成员** | **类型** | **描述** |
| HTnode  （哈夫曼树的结点） | Data | Char | 存放文本数据，如a，1 |
| Weight | Double | 存放数据的权重 |
| Parent | Int | 存放父节点的位置 |
| Lc | Int | 存放左节点的信息，1代表有左节点，0代表无左节点 |
| **rc** | **Int** | 存放左节点的信息，1代表有右节点，0代表无右节点 |

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名称 | 函数参数 | 参数返回值 | 函数功能 |
| void Menu() | 无 | 无 | 打印系统功能菜单 |
| void YS() | 无 | 无 | 压缩文件 |
| void printTree() | 无 | 无 | 打印哈夫曼树 |
| void printTable() | 无 | 无 | 打印哈夫曼表 |
| void printWPL() | 无 | 无 | 打印路径的WPL |
| void JYS() | 无 | 无 | 解压缩 |
| void Exit() | 无 | 无 | 退出系统 |
| bool TongJi(string path, int (&arr)[size]) | Path：压缩文件的路径  arr:统计数据 | 是否统计成功，true代表成功，false代表失败 | 遍历文件，字符的出现次数 |
| string getFilePath(string path) | Path：原文件的路径 | 返回文件的除去后缀的路径 | 获取文件除去后缀的那部分 |
| void E\_In(seqQueue \*queue, char ch) | Queue，循环队列，  Ch：进入队列的字符 | 空 | 将字符类型数据，以0,1形式入队列 |
| char getC(seqQueue \*queue) | Queue：循环队列 | 返回不定长编码的字符 | 从queue中，根据哈夫曼码，弹出一个字符 |
| void getTwoMin(HuffmanTree& HT, int n, int& s1, int& s2) | HT：哈夫曼树存放结点信息的存储结构  N: HT数组有效位的大小  s1：最小值的小标  s2：第二小的值下标 | 空 | 获取HT中最小的两位数值的下标 |
| void CreateHuff(HuffmanTree& HT, int (&arr)[size]) | HT：哈夫曼树存放结点信息的存储结构  arr:统计数据 | 空 | 创建哈夫曼树，HT中存放结点的信息 |
| void HuffCoding(HuffmanTree& HT, HuffmanCode& HC, int n) | HT：哈夫曼树存放结点信息的存储结构  HC:存放哈夫曼码的存储结构  N：数组大小 | 空 | 进行哈夫曼编码，编码值放在HC中 |
| void Clear(T H, int n) | H: 指针首地址  N: 长度 | 空 | 释放内存 |
| char E\_Out(seqQueue \*queue) | Queue：循环队列 | 返回8字节编码的字符 | 每次从循环队列中弹出一个字符 |
| int isExist(HuffmanTree HT, char buf, int n) | HT：哈夫曼树存放结点信息的存储结构  Buf： 查看的字符  N：字符对应编码的下标，-1代表不存在 | 非零，代表有，大小代表位置。否则，则无 | 判断元素是否存在哈夫曼码，并返回其在HC数组中的位置 |

**（三） 用户手册**

程序运行时，会以一个菜单形式展现功能，各个功能已经按照数字编号的方式分好类，用户只需输入相应的数字，选择相应的功能。对应任何新生成文件的放置位置，默认放置文件的位置，并且同名，后缀有 \*.huf, \*.tree, \*.table，分别存放压缩后的文件，打印输出的哈夫曼树，和哈夫曼编码。例如，放置C盘的work文件夹下下的文件test.txt,路径为C:\work\test.txt,则生成的压缩文件默认也放在此work文件夹下，路径为C:\work\test.huf。

**（四） 调试及测试**

由于公司每增加一个雇员，无论他（她）是哪一类人员,其编号均是顺序加1，也就是employee类的所有派生类对象创建时，都要访问同一个employeeNo，因此将employeeNo定义为静态数据成员。

**（五） 运行实例：**







1. **进一步改进**

**答：**无大的改进。

**（七）心得体会**

答：在读写文件时，利用eof()来判断是否已经达到文件末尾，出现文件最后一位数据重复读取了一次的错误结果，在网上搜了一番，原来是在读取文件时，并没有管是否读取成功了，当在读到文件最后一个数据时，在去读取，实际上读取时失败的，而之前保存在接受体中的数据未发生变化，故又构造了一个对象，这就是看到的最后一个数据重复一次输出。seekp函数调节写指针位置后，数据的写入是覆盖的，即新写入的数据覆盖原来的数据。ASCII码：一个英文字母（不分大小写）占一个字节的空间，一个中文汉字占两个字度节的空间。UTF-8编码：一个英文字符等于一个字节，一个中文三个字节。中文标点占三个字节，英文标点占一个字节。  [哈夫曼](https://baike.baidu.com/item/%E5%93%88%E5%A4%AB%E6%9B%BC" \o "哈夫曼)编码，又称霍夫曼编码，是一种编码方式，哈夫曼编码是可变[字长](https://baike.baidu.com/item/%E5%AD%97%E9%95%BF/97660" \o "字长)编码(VLC)的一种，该方法完全依据[字符](https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6/4768913" \o "字符)出现概率来构造异字头的平均长度最短的码。 哈夫曼编码主要是是用来压缩的，我们使用的基本编码是定长的，哈夫曼编码不是定长的，定长速度快，不定长速度慢，速度慢主要是因为要重新计算，所以哈夫曼编码可以提高速度。我们知道新申请的变量必须重新分配空间，计算机以变量分配空间，如果只分配了一个变量，就占用一个空间，如果分配了n个变量就消耗了n个空间，一般用数组消耗空间较少，速度更快。假设29个字符 一个8bit，总共消耗29\*8=232bit，但是重新编码之后一个占三位，那就是29\*3=87bit，所以使用哈夫曼编码节省了空间。 路径就是编码 s消耗4位然后有5个 4\*5=20bit 然后计算出其它的，一共消耗101bit，相比于原来的消耗的空间减少。解码就是根据编码进行识别，匹配输出，只是需要解码计算，所以哈夫曼编码速度要慢一些，哈夫曼编码它主要是用来压缩的。数据在内存和磁盘数据都是满的，删掉只是标记为无效空间，比如说c盘。计算机各个地方都是存储数据，数据在内存和磁盘数据都是满的，删掉数据只是标记数据为无效数据，格式化只是重新标记了数据，所以格式化的东西是可以恢复的。新增的数据就是把数据标记为有效， 删除数据就是把数据标记为无效。

1. **对课程设计的建议**

答：无。

**（九）附录⎯⎯源程序**