|  |  |  |  |
| --- | --- | --- | --- |
| **学生学号** | 0121810880420 | **实验课成绩** |  |

**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** | 夏红夏 |
| **学生姓名** | 刘子川 |
| **学生专业班级** | 软件工程1801 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2019 | -- | 2020 | 学年 | 第 | 2 | 学期 |

实验课程名称： 数据结构与算法综合实验

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验项目名称 | 二叉树与哈夫曼图片压缩 | | | 报告成绩 |  |
| 实验者 | 严伟滔 | 专业班级 | 软件1604 | 组别 |  |
| 同组者 |  | | | 完成日期 | 2018年5月9日 |
| 第一部分：实验分析与设计（可加页）   1. 实验目的和具体内容   1.实验目的  通过“图片压缩编码”的编程实践，学习树、遍历二叉树、哈夫曼树、哈夫曼编码和他们的编程应用。  (1)掌握树的存储结构  (2)掌握二叉树的三种遍历方法  (3)掌握并理解Huffman树、Huffman编码等知识和应用  (4)掌握文件的操作  (5)使用Huffman算法实现图像压缩程序  2.实验内容  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”,压缩后保存为“Pic.bmp.huf”文件。使用VS2010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)读取原文件，统计权值：  运行程序，输入文件名。  以“Pic.bmp”文件为例。若文件存放在F盘根目录下，输入文件完整路径“F：\Pic.bmp”。按回车结束。以字节流的方式，只读打“Pic.bmp”文件。  逐字节读取文件，统计文件中256种字节重复的次数，保存到一个数组中int weight[256]中。  (2)生成Huffman树  根据（1）中统计的结果，构建Huffman树。定义一个结构体来记录每个节点的权值、父节点、左孩子和右孩子。使用结构体数组来存储这个Huffman树。  (3)生成Huffman编码  遍历（2）中生成的Huffman树，记录256个叶子节点的Huffman编码，保存在字符串数组中。  (4)压缩原文件  使用Huffman编码对原文件中的字节重新编码，获得压缩后的文件数据。  (5)保存压缩文件  将编码过的数据，保存到文件“Pic.bmp.huf”中。   1. 分析与设计  1.需求分析 **输入输出格式要求:**  输入待压缩文件路径以及名称,在同级目录输出压缩好的文件  输入待解压文件路径以及名称,在同级目录输出解压后的文件  **时空复杂性要求:**  对于一定大小的原文件,应该在不能太长的时间内完成压缩  对于一定大小的压缩文件,应该在不能太长的时间内完成解压缩  在压缩与解压过程中因注重时间与空间的协调,在耗费时间与耗费内存空间两者之间找到一个平衡点,不能走极端.  **系统需要完成的功能:**  对输入原文件进行压缩,输出压缩文件  对输入压缩文件进行解压缩,输出原文件  并且在运行过程中显示相关提示信息  **需要解决的主要问题:**  对输入文件的字符出现频率统计,构造频率表,构造哈夫曼最优二叉树,构造每个字符对应的二进制前缀码表,将原文件逐个字符转换为连续二级制码流,再转换成编码后的数据,并将对应的哈夫曼码表字典以及编码后的数据按照特定格式写入文件中,并写入相关文件信息,从而达到压缩数据的目的.  **需要的关键技术:**  按照特定方式进行原始数据元素频率统计,哈夫曼最优二叉树结点数据的构造,整棵树的构造与连接,以及最终正确生成,每个结点的内存形式以及文件存储形式及其选择以及相互转换,采用特定算法生成每个原始数据元素所对应的特定二进制前缀码,以及存储方式,利用该二进制前缀码表,将原始数据转化为二进制流,在转化为最终输出格式(即已经编码好的数据),并写入文件中,在此过程中还应记录相关数据,比如原文件大小,压缩文件中哈夫曼码表字典以及编码后的数据的偏移量,确保在解压时能够正确还原原始数据;解压时需要读入文件,并验证是否符合本程序生成的压缩文件格式,以及大致统计对数据的压缩率,还有需要对程序进行边界条件以及其他特殊情况下的稳定性测试,以及对I/O耗时进行的分析与优化,对异常的处理,错误的处理.  **关键点1:生成字符频率表,生成哈夫曼最优二叉树,生成二进制前缀码表**  **关键点2:记录压缩所需要的各种关键信息,动态二叉树在文件中的储存形式以及恢复方法** 2 数据结构设计 **特别声明: 为了保持稳定性以及避免不确定的问题,本程序处理的文件大小均在Windows平台下4GB以内,即最大为232 =** **4294967296 字节,与32位平台上unsigned int数据类型(4字节)能够表示的最大值相同.**  本程序在设计过程中尽可能使用面向对象的编程思想,既能加深我对面向对象的理解,又能将其运用到程序设计的具体应用中. 2.1 数据结构选取 **文件I/O部分:**  输入:压缩文件完整路径  输出:在与原文件同级目录中输出  生成的特定格式的压缩文件至少从包含三部分:  1.**文件头信息**  2.**哈夫曼编码字典表**  3.**压缩后的编码数据**  为使在解压时,程序能够正确识别特定的压缩格式并能够将这三部分正确区分,在文件头中应该记录相关信息,比如字典表偏移量,编码数据偏移量,字典表还应该具有特定的规律与结果,方便在解压时能够将静态数据还原为动态哈夫曼最优二叉树  考虑到本程序需要进行频繁的文件I/O操作,选用C++ STL标准模板库中fstream类进行文件操作,既能代替C的FILE结构体而操作文件,也能利用到C++面向对象的高级特性,进行相关操作会更加方便.  **哈夫曼最优二叉树结点结构:**  **在压缩与解压过程中,最重要的部分是如何保存压缩时的哈夫曼树,因为哈夫曼二叉树的可旋转性,同样的一段数据编码方式有很多种可能的情况,需要在压缩时准确保存以及在解压时准确还原,不能出错,否则压缩解压便没有了意义**  作为二叉树结点,必须存储的基本信息有:  (1).左子结点  (2).右子结点  (3).本结点存储数据  (4).本结点权值大小  考虑到本程序的具体情形,在此基础上添加以下数据  (5).双亲结点(方便从树叶回溯到根结点)  (6).本结点深度  同时应考虑到二叉树应该有两种不同的形式而分别实现两种不同的结点结构体定义:  (1).**动态二叉树**(所有结点均使用new运算符分配内存空间,并由指针相连接,内存空间中不连续,方便添加删除移动等操作,在进行字符频率统计时很方便)  (2).**静态表二叉树**(连续的内存空间,类似于顺序表存储结构)  **字符频率统计表:**  本程序采用以ASCII码为元数据进行频率统计,原因主要是ASCII码范围值为0x00~0xFF,共计256种可能状态,与文件在实际存储介质上的存储形式相一致,并且不同考虑中英文以及字符编码的问题,将文件全部看作是以单字节的组成形式,只需读取底层存储介质上的字节流即可.  实际的频率统计表以静态数组为存储结构,数组大小为ASCII码范围大小256,数组元素类型应为二叉树结点结构体类型,在这里使用指针数组,灵活性较高.  **字符二进制前缀码表:**  本程序需要从构建好的哈夫曼二叉树生成对应字符前缀码表,通过计算可知[1]单个字符前缀码是最大长度为32位的二进制数,这里存储方式选择无符号数组进行存储,再添加一个指针数组用来存储每个字符对应前缀码数组的首地址.  **特定格式压缩文件信息结构体:**  生成的压缩文件需要在头部携带相关信息,用来在解压时判断是否为特定格式的压缩文件以及使用这些信息完成解压,以及准确还原原始文件.在这里本程序添加了一些额外的信息,用于判断以及验证  该结构体包含的信息如下:  1.特定文件开头(Magical Value,用来初步判断能否被本程序识别,一个或者多个字节)  2.压缩程序版本(一个或者多个字节)  3.原始文件大小(无符号32位整数)  4.生成压缩文件大小(无符号32位整数)  5.此压缩文件中编码好的数据大小(无符号32位整数)  6.此压缩文件中编码好的数据偏移量(无符号32位整数)  7.此压缩文件中哈夫曼编码字典大小(无符号32位整数)  8.此压缩文件中哈夫曼编码字典偏移量(无符号32位整数)  9.此压缩文件中哈夫曼编码字典单条记录大小(无符号16位整数)  10.此压缩文件中哈夫曼编码字典记录数目(无符号16位整数) 2.2 数据结构存储实现 **数据类型说明:**  **为了解决int类型在不同编译器或者系统平台下的字长问题,使用以下类型来代替原始类型**  **\_\_int32 代替 int unsigned \_\_int32 代替 unsigned int**  **\_\_int16 代替 short unsigned \_\_int16 代替 unsigned short**  **其他类型不变**  **文件I/O部分:**  原文件完整路径:  std::string originalFilePath = "C:\\Users\\Asus\\Desktop\\7d.bmp";  ifstream类,使用二进制输入流模式打开文件  std::ifstream originalFile; originalFile .open(originalFilePath, ios::in | ios::binary);  输出压缩文件路径:  std::string zippedFilePath = "C:\\Users\\Asus\\Desktop\\7d.bmp-zipped";  ofstream类,使用二进制输流流模式打开(创建)文件  std::ifstream zippedFile; zippedFile.open(zippedFilePath, ios::out | ios::binary);  注意:  在这里应注意一定使用**二进制模式打开(添加ios::binary标签)**,否则在Windows平台下会按照默认的文本流模式进行输入输出,这样会产生一个问题:即读入写入的某个字节ASCII码值为0x0A(即相当于文本模式换行符'\n'),Windows会将其转换为连续两个字节的0x0D 0x0A(即相当于"\r\n"),读写就会出现问题[5].  打开文件以后应及时判断两个文件是否打开成功,如不成功应终止继续操作,并提示相关错误信息.  **哈夫曼最优二叉树结点结构体:**  /\*强制编译器使用2字节对齐\*/  #pragma pack(2)  /\*定义二叉树结点存储数据类型为char,只需存储单个字节,与ASCII表示范围相同\*/  typedef unsigned char BinTreeDataType;  /\*第1种: 动态二叉树结点结构体\*/  typedef struct BinTreeDynamicStruct {  /\*双亲结点指针,NULL表示为根结点\*/  BinTreeDynamicStruct\* par;  /\*左子树结点指针,NULL表示无左子树\*/  BinTreeDynamicStruct\* left;  /\*右子树结点指针,NULL表示无右子树\*/  BinTreeDynamicStruct\* right;  /\*本结点以及子结点数据总权重(32位无符号整数)\*/  unsigned \_\_int32 weight;  /\*本结点深度(16位有符号整数,0表示根结点所在的第0层,子树以及子树的子树依次为1,2…层)\*/  \_\_int16 depth;  /\*本结点数据类型(本程序中为unsigned char)\*/  BinTreeDataType data;  }BinTree;  /\*第2种: 静态表二叉树结点结构体\*/  typedef struct BinTreeStaticStruct {  /\*本结点双亲位置,带符号16位整数,为-1时表示根结点\*/  \_\_int16 par;  /\*本结点左子树位置,带符号16位整数,为-1时表示无左子树\*/  \_\_int16 left;  /\*本结点右子树位置,带符号16位整数,为-1时表示无右子树\*/  \_\_int16 right;  /\*本结点深度,带符号16位整数,树根为1,每深一层+1\*/  \_\_int16 depth;  /\*本结点以及子结点数据总权重(32位无符号整数)\*/  unsigned \_\_int32 weight;  /\*本结点数据类型(本程序中为unsigned char)\*/  BinTreeDataType data;  /\*以下三字节为预留位置,无实际意义,方便结构体字节对齐为16字节,提高CPU处理效率\*/  char reserve1;  char reserve2;  char reserve3;  }BinTreeTable;  /\*恢复编译器默认对齐设置\*/  #pragma pack ()  **字符频率统计表:**  BinTree\* freqList[256];  **字符二进制前缀码表:**  char\* prefixCode[256];  **特定格式压缩文件信息结构体:**  /\*强制编译器使用2字节对齐\*/  #pragma pack (2)  typedef struct ZIPFileInfoStruct {  /\*该文件特殊标识,本程序默认为WT两个字符,即16进制数0x5457\*/  unsigned \_\_int16 zipTag;  /\*压缩软件版本,第一版为0x0001\*/  unsigned \_\_int16 zipVer;  /\*原文件大小,无符号32位整数(即最大4GB)\*/  unsigned \_\_int32 oFileSize;  /\*此压缩文件大小,无符号32位整数(即最大4GB)\*/  unsigned \_\_int32 cFileSize;  /\*此压缩文件中编码好的数据大小,无符号32位整数(即最大4GB)\*/  unsigned \_\_int32 cDataSize;  /\*此压缩文件中编码好的数据偏移量,无符号32位整数\*/  unsigned \_\_int32 cDataPosi;  /\*此压缩文件中哈夫曼编码字典大小,无符号32位整数\*/  unsigned \_\_int32 cDictSize;  /\*此压缩文件中哈夫曼编码字典偏移量,无符号32位整数\*/  unsigned \_\_int32 cDictPosi;  /\*此压缩文件中哈夫曼编码字典单条记录大小,无符号16位整数\*/  unsigned \_\_int16 cDictItemSize;  /\*此压缩文件中哈夫曼编码字典记录数目,无符号16位整数\*/  unsigned \_\_int16 cDictItemNumb;  }ZIPFileInfo;  /\*恢复编译器默认对齐设置\*/  #pragma pack ()  **需要注意的问题:**  1.结构体对齐.  2.不同平台下int字长[6].  3.需要考虑每个变量的变化范围,为其选择合适的数据类型,以达到既不能溢出,也不能过于浪费存储空间的目的.  4.初始化时需要使用memset函数将这些结构体内存区域全部置为0,以防止程序由于错误地在未初始化情况下就读取这些内存区域而出现不可预知的问题.  **2.3系统的总体设计规划:**  本程序抽象结构关系如下:  **压缩编码模块:** **HuffmanEncoder 类:**  build函数,将两个字符串传入,即原文件完整路径与待输出文件完整路径传入.  init函数,初始化对象成员, freqList, prefixCode, zipFileHeadTag的内存空间,,为指针数组中的每个元素分配内存空间并将其初始化为0,将对应字符二进制前缀码表数组内存区域全部置为0xFF(初始值),同时打开build函数传入的文件路径字符串所指文件,并判断文件流是否正常打开,还需获取原文件大小,向文件头写入相关基本信息(文件标识,版本,原文件大小,单条字典记录大小,字典偏移量等)  generateFreqList函数:统计原文件中ASCII码值分别为0~255的字符出现的频率,并向对象成员字符频率表中写入相关数据  generateHFMTree函数:使用生成好的字符频率表,构造出哈夫曼最优二叉树(动态链表形式)  generatePrefixCodeTable函数,用来根据生成好的哈夫曼最优二叉树生成ASCII码值分别为0~255的字符生成二进制前缀码,并将其依次写入对象成员二进制前缀码表中.  writePrefixCodeTable函数,为generatePrefixCodeTable的具体逻辑实现函数,该函数根据动态二叉树的特点,利用函数的递归特性进行工作.  writeByteStream函数:利用构造完成的二进制前缀码表,对原文件从文件头部向文件尾部按照单字节方式进行顺序遍历得到链接完整的二进制编码流,再按照每八个二进制数转化为一个字节的规律向压缩文件中单字节输出编码后的数据,同时统计编码过程中的各种信息  encode函数,外部调用,负责将以上相关函数按照逻辑顺序的封装在一起,抽象为一个”执行操作函数”  除此以外,还有getPrefixCodeLen(获取对应字符二进制前缀码长度), prefixCodeCat(用于拼接二进制前缀码), trimPrefixCode(用于整理二进制前缀码表)  **解压解码模块:** **HuffmanDecoder 类:**  build函数: 将两个字符串传入,即压缩文件完整路径与待输出原文件完整路径传入  init函数: 初始化对象成员,压缩文件结构体内存初始化为0,打开文件操作,看其是否打开成功,然后检测输入压缩文件的有效性检测完成后再进行其他操作  checkZippedFileValidity函数:检测所打开的压缩文件是否有效,首先读入压缩文件的头结构体信息,然后依次检测各项数据是否与真实情况相符合,如果有任意一个条件不符合,说明所打开的压缩文件格式有误或者不能被识别,或者被恶意修改,如果继续进行就会有风险,所以在不符合时应该立即停止  checkBinTreeTableValidity函数:用于检测文件中存储的哈夫曼编码字典表是否有效,如果无效,解压过程终止  readByteStream函数,将压缩文件中的编码数据使用除2取余法还原为一段段的二进制流,然后遍历二进制流,使用对应的哈夫曼编码字典还原出原始数据,再一次按照单字节写入输出的原文件流中, 同时统计解码过程中的各种信息  getCharFromHfmTree函数:根据解析得到的二进制流在哈夫曼最优二叉树中查找对应叶子结点,利用了函数的递归特性实现.  decode函数: 函数,外部调用,负责将以上相关函数按照逻辑顺序的封装在一起,抽象为一个”执行操作函数”  除此以外,还有getPrefixCodeLen(获取对应字符二进制前缀码长度), prefixCodeCat(用于拼接二进制前缀码), trimPrefixCode(用于整理二进制前缀码表)    **二叉树相关操作工具以及函数:**  getBinTreeNodeNum函数: 递归遍历获取二叉树(动态与静态表)结点总数  getBinTreeLeavesNum函数: 递归遍历获取二叉树(动态与静态表)树叶总数  convertTreeToTable函数: 转换动态二叉树为静态表二叉树并返回其首元素地址  writeTable函数: convertTreeToTable的具体实现,利用了函数的递归特性遍历并写入输出的静态表二叉树以完成转换  convertTableToTree函数: 转换静态表二叉树为动态二叉树并返回其根结点首地址  writeTree函数: convertTableToTree的具体实现,利用了函数的递归特性遍历并写入输出的动态二叉树以完成转换  getBinTreeDepth函数: 顺序遍历获取动态或者静态表二叉树(最大)深度  setBinTreeDepth函数: 递归调用以正确设置动态二叉树的每一个结点深度  compareTree函数:可以比较两颗二叉树是否严格相同  destoryBinTree函数:销毁二叉树,以回收new出的内存空间,防止内存泄露  2.4系统的详细设计以及算法描述:  **压缩编码模块: HuffmanEncoder 类C++代码声明:**  class HuffmanEncoder {  /\*私有成员变量及函数\*/  private:  /\*原文件完整路径\*/  string originalFilePath;  /\*待输出压缩文件完整路径\*/  string zippedFilePath;  /\*原文件流\*/  ifstream originalFile;  /\*输出压缩文件流\*/  ofstream zippedFile;  /\*压缩文件头信息结构体\*/  ZIPFileInfo zipFileHeadTag;  /\*哈夫曼最优二叉树指针\*/  BinTree\* hfmTree;  /\*字符统计频率表\*/  BinTree\* freqList[256];  /\*原文件大小\*/  unsigned \_\_int32 oFileSize;  /\*编码好的数据大小\*/  unsigned \_\_int32 cDataSize;  /\*encoder状态码\*/  int status;  /\*字符二进制前缀码数组\*/  char\* prefixCode[256];  /\*用于输出错误信息的函数\*/  void errMsgDisplay(char\*);  /\*初始化函数\*/  void init();  /\*打开文件相关函数\*/  int openFile();  /\*生成字符频率表函数\*/  void generateFreqList();  /\*获取频率表中最小权重的元素结点函数\*/  int getMinFreqElemSeq();  /\*生成哈夫曼最优二叉树函数\*/  int generateHFMTree();  /\*写入二进制前缀码表具体实现函数\*/  void writePrefixCodeTable(BinTree\*, BinTree\*, char\*);  /\*生成各字符对应的二进制前缀码表函数\*/  int generatePrefixCodeTable();  /\*获取传入字符串中有效二进制前缀码的长度\*/  int getPrefixCodeLen(unsigned char\*);  /\*前缀码拼接函数\*/  unsigned char\* prefixCodeCat(unsigned char\*, unsigned char\*);  /\*整理前缀码表函数,用来将转换完成后的二进制前缀码清理出这个数组,然后将剩余未转换的二进制前缀码移动到最前方\*/  unsigned char\* trimPrefixCode(unsigned char\*, \_\_int32, \_\_int32);  /\*读取输入原文件并编码后将其输出到压缩文件中\*/  int writeByteStream();  /\*使用new运算符复制一个新的结点\*/  BinTree\* copyNode(BinTree\*);  /\*公有成员函数\*/  public:  /\*构建函数\*/  void build(char\*, char\*);  /\*对象的构造函数\*/  HuffmanEncoder();  /\*对外封装好的编码函数\*/  int encode();  };  **压缩编码模块: HuffmanDecoder 类C++代码声明:**  class HuffmanDecoder {  /\*私有成员变量及函数\*/  private:  /\*压缩文件完整路径\*/  string zippedFilePath;  /\*待输出原文件完整路径\*/  string originalFilePath;  /\*压缩文件流\*/  ifstream zippedFile;  /\*输出原文件流\*/  ofstream originalFile;  /\*压缩文件头信息结构体\*/  ZIPFileInfo zipFileHeadTag;  /\*哈夫曼最优二叉树(动态)指针\*/  BinTree\* hfmTree;  /\*哈夫曼最优二叉树(静态表)指针\*/  BinTreeTable\* hfmTreeTable;  /\*压缩文件大小\*/  unsigned \_\_int32 zFileSize;  /\*原文件大小\*/  unsigned \_\_int32 oDataSize;  /\*decoder状态码\*/  int status;  /\*原文件流编码好的二进制流对8取余后剩余数量\*/  unsigned char eofTag;  /\*显示错误信息\*/  void errMsgDisplay(char\*);  /\*初始化函数\*/  void init();  /\*打开文件相关函数\*/  int openFile();  /\*通过读入的压缩文件头结构体数据来判断文件是否有效\*/  int checkZippedFileValidity();  /\*检测读入的哈夫曼编码字典是否有效\*/  int checkBinTreeTableValidity();  /\*获取传入字符串中有效二进制前缀码的长度\*/  int getPrefixCodeLen(unsigned char\*);  /\*前缀码拼接函数\*/  unsigned char\* prefixCodeCat(unsigned char\*, unsigned char\*);  /\*整理前缀码表函数,用来将转换完成后的二进制前缀码清理出这个数组,然后将剩余未转换的二进制前缀码移动到最前方\*/  unsigned char\* trimPrefixCode(unsigned char\*, \_\_int32, \_\_int32);  /\*按照二进制流读取对应二叉树结点数据\*/  void getCharFromHfmTree(BinTree\*, unsigned char\*, unsigned char\*, int&, int);  /\*从编码数据中解码得到原数据\*/  int readByteStream();  /\*公有成员函数\*/  public:  /\*构建函数\*/  void build(char\*, char\*);  /\*对象的构造函数\*/  HuffmanDecoder();  /\*对外封装好的解码函数\*/  int decode();  };  **核心算法:** **[4]**  **哈夫曼树的构造(哈夫曼算法)**  **1.根据给定的n个权值{w1,w2,…,wn}构成二叉树集合F={T1,T2,…,Tn},其中每棵二叉树Ti中只有一个带权为wi的根结点,其左右子树为空.**  **2.在F中选取两棵根结点权值最小的树作为左右子树构造一棵新的二叉树,且置新的二叉树的根结点的权值为左右子树根结点的权值之和.**  **3.在F中删除这两棵树,同时将新的二叉树加入F中.**  **4.重复2、3,直到F只含有一棵树为止.(得到哈夫曼树)**  这里的F即为本程序中字符统计频率表数组,每一个元素均为指向一个结点的指针,通过对指针的重新赋值从而实现哈夫曼树的构造,这里的n最大为256,最小为0  三、主要仪器设备及耗材   1. 安装了Windows 10 2. PC机系统上安装了Microsoft Visual Studio 2015开发环境   第二部分：实验过程和结果（可加页）   1. 源代码   **关键函数1:(由字符频率表生成哈夫曼最优二叉树函数)**  int HuffmanEncoder::generateHFMTree() {  /\*用于记录频率表中最小的权值以及次小的权值的顺序\*/  int min1, min2;  /\*BinTree指针,存放交换过程中的相关结点地址\*/  BinTree\* node1, \*node2, \*temp, \*newNode;  while (true) {  /\*获取表中权重即频率最小的字符的顺序即ASCII码\*/  min1 = getMinFreqElemSeq();  /\*列表中应该至少有一个权重非0的元素,如果没有,说明出错\*/  if (min1 == -1) {  errMsgDisplay("生成树出现错误!\n");  /\*生成失败\*/  return -1;  }  /\*复制为一个全新的结点\*/  node1 = copyNode(freqList[min1]);  /\*同时将被复制过的结点状态置为已处理\*/  freqList[min1]->weight = 0;  /\*获取新表中权重即频率最小的字符的顺序即ASCII码\*/  min2 = getMinFreqElemSeq();  /\*当min2为-1时,树生成完成\*/  if (min2 == -1) {  /\*当前对象实例的hfmTree赋值为表中最后一个权值非0元素\*/  hfmTree = node1;  /\*生成成功\*/  return 0;  }  /\*将新表中权值最小的结点赋值给临时指针变量\*/  temp = freqList[min2];  /\*为node2分配新的内存空间\*/  node2 = copyNode(temp);  /\*将原表中被复制过的结点置为无效状态\*/  freqList[min2]->weight = 0;  /\*为newNode分配新的内存空间,newNode用来存储两个子结点合并成新结点的地址\*/  newNode = getNewNode();  /\*原表中权重最小的结点的双亲结点赋值为newNode\*/  node1->par = newNode;  /\*原表中权重次小的结点的双亲结点赋值为newNode\*/  node2->par = newNode;  /\*newNode左子树赋值为原表中权重最小的结点\*/  newNode->left = node1;  /\*newNode右子树赋值为原表中权重次小的结点\*/  newNode->right = node2;  /\*更改新结点权重为两个最小结点的权重之和\*/  newNode->weight = node1->weight + node2->weight;  /\*删除临时结点,回收内存空间\*/  delete temp;  /\*将新结点至于被删除的次小结点原位置上\*/  freqList[min2] = newNode;  }  return -2;  }  **关键函数2:(** **读取输入原文件并将编码好的二进制流转换为字节流,输出到压缩文件中)**  int HuffmanEncoder::writeByteStream() {  /\*当前已转换的原文件字节数\*/  unsigned \_\_int32 cLen = 0;  /\*当前已输出到压缩文件中的编码完成数据字节数\*/  unsigned \_\_int32 eLen = 0;  /\*同来存放临时二进制流为数组\*/  unsigned char tmp[256];  /\*临时数组中有效编码长度,无效编码为0xFF,有效编码为0或者1\*/  \_\_int32 tmplen;  /\*从临时数组中已转换的二进制流bit数\*/  \_\_int32 pLen;  /\*用来参与循环的辅助变量\*/  int i, j;  /\*用来存放读入的字符以及要写入的字符\*/  unsigned char a = 0, b = 0;  /\*用来在临时数组中移动的指针\*/  unsigned char\* p = tmp;  /\*两个字符指针\*/  char\* q = (char\*)&b;  char\* r = (char\*)&a;  /\*初始化临时数组内存区域为0xFF\*/  memset(tmp, 0xFF, 256);  /\*清除原文件异常状态,恢复正常状态\*/  originalFile.clear();  /\*定位文件指针到文件开头\*/  originalFile.seekg(0, ios::beg);  /\*清除输出压缩文件文件异常状态,恢复正常状态\*/  zippedFile.clear();  /\*定位文件指针到距离文件开头偏移量为数据偏移位置\*/  zippedFile.seekp(zipFileHeadTag.cDataPosi, ios::beg);  /\*对于只有一种字符的文件,直接跳转到写文件末尾的语句\*/  if (hfmTree->left == NULL && hfmTree->right == NULL && hfmTree->weight != 0)  goto label;  /\*所读取的原文件字节数小于原文件大小时循环\*/  while (cLen < oFileSize) {  /\*所读取的原文件字节数小于原文件大小且临时数组有效长度小于8时循环\*/  while (cLen < oFileSize && getPrefixCodeLen(tmp) < 8) {  /\*读入一个字节到变量q指针所指变量a中,I/O费时比较严重,这里有优化的空间\*/  originalFile.read(q, 1);  /\*将字符a对应的前缀码拼接到临时数组中\*/  prefixCodeCat(tmp, (unsigned char\*)prefixCode[b]);  /\*已读入原文件字节数加一\*/  cLen++;  }  /\*获取临时数组有效长度\*/  tmplen = getPrefixCodeLen(tmp);  /\*辅助指针p指向临时数组头部\*/  p = tmp;  /\*辅助变量p赋值为临时数组长度\*/  pLen = tmplen;  /\*临时数组中未处理的前缀码数量大于7时循环\*/  while (pLen > 7) {  /\*初始化a为0\*/  a = 0;  /\*截取临时数组中前8位二进制数,转化为一个字节,并使用位运算提高效率\*/  for (i = 7,j = 0; i > -1; i--,j++) {  /\*使用位运算,每8bit转化为一个char\*/  a += (p[i] << j);  }  /\*将转化出的这个字节写入压缩文件中,I/O费时比较严重,这里有优化的空间\*/  zippedFile.write(r, 1);  /\*已编码数据长度加一\*/  eLen++;  /\*临时数组中未处理的二进制数长度减8\*/  pLen -= 8;  /\*指针p向后移动8个字节\*/  p += 8;  }  /\*整理临时数组,使得剩余未转化的二进制数转移到头部\*/  trimPrefixCode(tmp, tmplen, pLen);  }  /\*变量b赋值为0\*/  b = 0;  /\*处理最后几个bit,不足8bit的剩余部分置为0\*/  if (pLen != 0) {  a = 0;  b = pLen;  p = tmp;  while (pLen > 0) {  pLen--;  a += (p[pLen] << (7 - pLen));  }  zippedFile.write(r, 1);  eLen++;  }  label:  /\*写入文件最后一个字节,该字节表示编码好的二进制流不足8bit的剩余部分bit大小,范围0~7\*/  zippedFile.write(q, 1);  /\*压缩后的数据大小\*/  cDataSize = eLen;  return 0;  }  **可以优化的部分:**  由存储设备以及I/O操作的相关原理[2]可知originalFile.read以及zippedFile.write这两个I/O操作,可以不直接写入存储介质,而先使用缓冲区代替,当输出完成后再刷新缓冲区将数据一次性写入存储介质,即可极大地提高程序运行效率.  在一般的压缩软件实现过程中,对于数据频率的统计并不是像本程序一样全盘统计,而是分成一定大小的块分别进行统计,根据相关原理[3]可知这样可以提高效率并增加程序运行速度,但是大部分压缩软件核心原理仍然是哈夫曼算法   1. 调试说明（调试手段、过程及结果分析）   **第1次输入输出测试:**  原文件:D:\\0.txt,内容为一段文本: 111211122232345334564  该文件相关统计信息如下:  各字符频率统计为:  1: 6次 2: 5次 3: 4次 4: 3次 5: 2次 6: 1次  总长度:21字节=总权重  当使用本程序的哈夫曼算法时,从生成的最优二叉树对这些字符进行编码的结果是:  1: 10  2: 01  3: 00  4: 110  5: 1111  6: 1110  **(这些数据编码不唯一,但是按照本程序中的算法得到的编码是唯一的)**  则原文件编码后的二进制流为:  10101001 10101001 01010001 00110111 10000110 11111110 110  每八位转换为一个字节, 最后不足8位补0处理,同时记录不足8位的位数  A9 A9 51 37 86 FE C0 (03)  生成文件位置在D:\0.txt-zipped,使用16进制编辑器打开如下:  **说明: QQ截图20171120182049**  具体说明如下:  00 h~1F h 压缩文件头信息  57 54 即 “WT”ASCII 编码  01 00 即 “01”版本号  15 00 00 00 即 原文件大小 21字节,十六进制表示为0x15  D8 00 00 00 即 压缩文件大小216字节,十六进制表示为0xD8  07 00 00 00 即 此压缩文件中编码数据大小7字节,十六进制表示为0x07  D0 00 00 00 即 此压缩文件编码数据偏移量,从文件头第208个字节开始为编码数据,十六进制表示为0xD0  B0 00 00 00 即 此压缩文件哈夫曼编码字典总大小为176字节,十六进制表示为:0xB0  20 00 00 00 即 此压缩文件哈夫曼编码字典偏移量, 从文件头第32个字节开始为编码字典数据,十六进制表示为0x20  10 00 即 哈夫曼编码字典单条记录大小为16字节,十六进制表示为0x10  0B 00 即 哈夫曼编码字典记录数目,总共11条,十六进制表示为0x0B  20 h~CF h 哈夫曼编码字典数据区域  以第一条记录为例:  FF FF 表示此结点双亲为空,即此结点为根结点  01 00 表示左子树位置为第2条记录  04 00 表示右子树位置为第5条记录  01 00 表示此结点深度为1  15 00 00 00 表示此结点权重为0x15,即21  00 表示此结点记录数据为0  CD CD CD 三个预留字节,无实际意义  ……(其余10条字典记录)  A9 A9 51 37 86 FE C0 03 编码数据区域  **数据解码:**  读取压缩文件相关信息:获取原文件大小,哈夫曼编码字典数据,编码数据  使用对应函数将读入的哈夫曼编码字典转换为动态二叉树  每一次读入单个字节,使用除2取余法,这里使用又位移运算符实现,将一个编码数据字节转换为二进制流,然后在之前生成的动态二叉树中查找,遇到0进入左子树,遇到1进入右子树,当某子树的左子树与右子树均为NULL时,查找完毕,返回对应结果  以这一段数据为例:  A9 A9 51 37 86 FE C0 (03)  最后一位03表示原文件转换为二进制流后最后一个不足8bit的部分大小  转换为二进制如下  10101001 10101001 01010001 00110111 10000110 11111110 110  进入生成好的哈夫曼二叉树查找  得到原始数据为:  111211122232345334564  可以看到与原文件完全一致  **第2次输入输出测试:**  本次我们使用图像文件:路径为D:\\5.bmp,这是一幅使用Windows画图软件所生成的500x500像素,24位位图,文件大小750,054字节(733KB),图像如下(原图为彩色,打印显示黑白)    输入程序后,运行结果如下:    压缩率为22.67%,可以看出效率还是比较高的,当然这里是因为bmp图片的特殊性以及该图片颜色不够丰富等原因压缩率比较高  使用16进制编辑器打开以后如下图(只截取了一部分,第一幅为压缩文件头部分,第二幅为编码数据开始部分)      将解压后的D:\\5.bmp-zipped-unzipped增加bmp后缀以后打开如下    与原图完全一致  最终结论:  **本程序初步达到了实验目的**  可以优化的部分:  在代码编写部分提过的originalFile.read以及zippedFile.write这两个I/O操作,可以不直接写入存储介质,而先使用缓冲区代替,当输出完成后再刷新缓冲区将数据一次性写入存储介质,即可极大地提高程序运行效率.  第三部分：实验小结、收获与体会  1.数据结构中树的应用很重要 ,哈夫曼算法有着十分重要的意义,通过本次实验学习到了数据结构与算法的很多知识,收获颇丰  2.善于运用this,在get/set类方法中遇到局部变量与类成员变量同名时可通过this来解决冲突问题  3.面向对象的思想在编程中有着重要的运用,实际是计算机发展过程中对大自然各种现象的模拟,深入理解对编程的学习很有帮助  4.编程过程中,对于边界条件需要仔细考虑,比如数组下标越界,特殊情况未考虑到等,本实验中主要体现在对不足8bit的部分需要补0处理,以及记录不足8bit的位数,方便以及提示用户输入有误,这些都是很重要的细节,在产品设计中也有着很重要的运用,做到良好的人机交互性.  5.一个工程项目中层次应该一目了然,目录以及文件名要有意义,对功能要做好恰当的划分,做好代码的可维护性,低耦合性,以及健壮性,这样也方便功能的迭代以及日后的维护  6.在开发过程中,发现对于一些文件编码出错,于是使用断点功能以及诸如\_\_FUNCTION\_\_,\_\_LINE\_\_等宏定义,排查出问题,在解码器部分的void HuffmanDecoder::getCharFromHfmTree()函数中,如果使用二进制流控制进入哈夫曼树各分支,在结点为空的情况下,仍然会访问越界区域,造成本应停止递归的函数继续进行,最后出现解码出错的问题,于是通过为该函数添加一个整数引用的参数,如果检测到访问越界,将增加后的pLen赋值为初值,并终止函数递归,最终达到解决问题的目的  7.在压缩与解压过程中,最重要的部分是如何保存压缩时的哈夫曼树,因为哈夫曼二叉树的可旋转性,同样的一段数据编码方式有很多种可能的情况,需要在压缩时准确保存以及在解压时准确还原,不能出错,否则压缩解压便没有了意义  8.I/O操作在很大程度上拖慢了本程序总体运行效率,还可以继续优化,通过查阅相关资料可知,在很多地方,I/O操作是最费时间的,一直是工程师与程序员在追求极致优化的部分  9.本软件还可以增加一个GUI界面,增加人机交互时的友好,提高用户的使用感受  10.本项目由我本人(软件1604严伟滔)完全自主编写,已开源至GitHub,项目地址:  <https://github.com/initializeZero/HuffmanZipHelper> (GitHub用户名: initializeZero)  (旧地址:  https://github.com/JackYanx/C-Project/tree/master/DataStructureLesson/HuffmanAlgorithm) 参考文献 **[1] 由本程序适用情况可知,Window平台下4GB的文件数据大小即232字节,设所能生成最优二叉树最大深度为n,则当极限条件下,有树总权重20 + 21 + 22 + 23 + … + 2n = 232字节,等比数列求和解得n整数解为32**  [2] 存储设备I/O 性能分析 http://blog.csdn.net/xygl2009/article/details/46893309  [3] ZIP压缩算法详细分析及解压实例解释 <https://www.cnblogs.com/esingchan/p/3958962.html>  [4] 严蔚敏，吴伟民. 数据结构. 北京:清华大学出版社，2013. P144  [5] Why does ofstream insert a 0x0D byte before 0x0A?(为何使用fstream写入0x0D前会增加0x0A?)  <https://stackoverflow.com/questions/5173498/why-does-ofstream-insert-a-0x0d-byte-before-0x0a>  [6] Is the size of C “int” 2 bytes or 4 bytes?(int字长为2还是4?)  https://stackoverflow.com/questions/11438794/is-the-size-of-c-int-2-bytes-or-4-bytes | | | | | |