|  |  |  |  |
| --- | --- | --- | --- |
| **学生学号** |  | **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** | 李晓红 |
| **学生姓名** | 彭群凯 |
| **学生专业班级** | 软件工程zy1902 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 | 彭群凯 | 专业班级 | 软件zy1902 | 组别 |  |
| 同组者 |  | | | 完成日期 | 2021年 3 月 5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. 数据结构的设计   本次实验主要使用了Huffman树作为主要的结构，来进行数据的存储。   * + - 1. 核心算法设计   本次实验的核心算法是huffman压缩算法，首先通过以二进制流的方式，只读打开文件，统计文件中256种字节的重复次数，再以此为权值构建huffman树，同时得到huffman编码，最后通过Strbyte()函数来将八个字符转成一个字符来进行压缩。  三、主要仪器设备及耗材   1. 安装了Windows XP或Windows 7或其它版本的Windows操作系统的PC机1台 2. PC机系统上安装了Microsoft Visual Studio开发环境   第二部分：实验过程和结果（可加页）   1. 源代码     /\*  函数名：Compress（）  功能：实现文件压缩  参数：const char\* pFilename-文件路径  返回值：int:FALSE-失败；TRUE-成功；  \*/  Compress.h  #ifndef COMPRESS\_H  #define COMPRESS\_H  //typedef char \*\*HuffmanCode;  //文件头  struct HEAD  {  char type[4];  int length;  int weight[256];  };  //实现文件压缩  /\*  函数名：Compress（）  功能：实现文件压缩  参数：const char\* pFilename-文件路径  返回值：int:FALSE-失败；TRUE-成功；  \*/  int Compress(const char \*pFilename);  //读取源文件和初始化头文件的信息  /\*  函数名：InitHead（）  功能：读取源文件和初始化头文件的信息  参数：const char \* pFilname, HEAD & sHead  返回值：int:FALSE-失败；TRUE-成功；  \*/  int InitHead(const char \* pFilname, HEAD & sHead);  //利用Huffman编码 实现压缩编码  /\*  函数名：Encode（）  功能：利用Huffman编码 实现压缩编码  参数：const char \*, char\*\*, char \*, const int  返回值：int:FALSE-失败；TRUE-成功；  \*/  int Encode(const char \*, char\*\*, char \*, const int);  //int Encode(const char \*pFilname, const HuffmanCode pHC, char \*pBuffer, const int nSize);  //将二进制字符串转换成字节  /\*  函数名：Str2byte（）  功能：将二进制字符串转换成字节  参数：const char \* pBinStr  返回值：char b = 0x00;  \*/  char Str2byte(const char \* pBinStr);  //生成压缩文件  /\*  函数名：WriteFile（）  功能：生成压缩文件  参数：const char \* pFilename, const HEAD sHead, const char \* pBuffer, const int nSize  返回值：int:FALSE-失败；TRUE-成功；  \*/  int WriteFile(const char \* pFilename, const HEAD sHead, const char \* pBuffer, const int nSize);  #endif  Compress.cpp  #define \_CRT\_SECURE\_NO\_WARNINGS  #include "pch.h"  #include <iostream>  #include <stdlib.h>  #include "Compress.h"  #include"Huffman.h"  #include<iomanip>  using namespace std;  #define OK 1  #define ERROR 0  const int SIZE = 256;  //扫描文件和初始化头文件的信息  int InitHead(const char \* pFilname, HEAD & sHead)  {  strcpy(sHead.type, "HUF"); //文件类型  sHead.length = 0; //源文件长度  for (int i = 0; i < SIZE; i++)  {  sHead.weight[i] = 0; //权值  }  //以二进制流形式打开文件  FILE \*in = fopen(pFilname, "rb");  //扫描文件，获得权重  int ch;  while ((ch = fgetc(in)) != EOF)  {  sHead.weight[ch]++;  sHead.length++;  }  //关闭文件  fclose(in);  in = NULL;  return OK;  }  //得到编码文件  int Compress(const char \* pFilename)  {  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  //打开并扫描文件  cout << "正在读取文件……" << endl;  int weight[256] = { 0 }; //打开文件，获取权重  FILE\* in = fopen(pFilename, "rb");  int ch;  while ((ch = getc(in)) != EOF)  {  weight[ch]++;  }  fclose(in);  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  //将编码生成Huffman树  int i;  int n = 256; //Huffman树共有n个叶子节点  int m = 2 \* n - 1; //那么就有2n+1个节点  HuffmanTree pHT = new HTNode[m + 1]; //定义Huffman树  CreateHuffmanTree(pHT, weight, n);  //生成Huffman编码  char\*\* pHC = new char\*[n + 1]; //编码  for (int i = 1; i <= n; i++)  pHT[i].weight = weight[i - 1];  HuffmanCoding(pHC, pHT);  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  //计算编码缓冲区大小  int nSize = 0;  for (int i = 0; i < 256; i++)  nSize += weight[i] \* strlen(pHC[i + 1]);  nSize = (nSize % 8) ? nSize / 8 + 1 : nSize / 8;  //对编码文件进行压缩  char \*pBuffer = NULL;  pBuffer = new char[nSize];  memset(pBuffer, 0, (nSize) \* sizeof(char));  Encode(pFilename, pHC, pBuffer, nSize);  if (!pBuffer) {  return ERROR;  }  HEAD sHead;  InitHead(pFilename, sHead);  cout << "文件大小：" << sHead.length << endl;  int afterlen = WriteFile(pFilename, sHead, pBuffer, nSize);  cout << "压缩大小：" << afterlen << endl;  cout << "压缩率：" << (double)afterlen \* 100 / sHead.length << "%" << endl;  delete pHT; delete[] pHC; delete pBuffer;  return OK;  }  //实现·压缩编码  int Encode(const char \* pFilname, const HuffmanCode pHC, char \* pBuffer, const int nSize)  {  //打开文件  FILE \*in = fopen(pFilname, "rb");  //开辟缓冲区ni  pBuffer = (char \*)malloc(nSize \* sizeof(char));  char cd[SIZE] = { 0 }; //工作区  int pos = 0; //缓冲区指针  int ch;  //扫描文件  while ((ch = fgetc(in)) != EOF)  {  strcat(cd, pHC[ch + 1]);  //压缩编码  while (strlen(cd) >= 8)  {  pBuffer[pos++] = Str2byte(cd);  for (int i = 0; i < SIZE - 8; i++)  {  cd[i] = cd[i + 8];  }  }  }  if (strlen(cd) > 0)  {  pBuffer[pos++] = Str2byte(cd);  }  fclose(in); //关闭文件  return OK;  }  //生成压缩文件  int WriteFile(const char \* pFilename, const HEAD sHead, const char \* pBuffer, const int nSize)  {  //生成文件名  char filename[256] = { 0 };  strcpy(filename, pFilename);  strcat(filename, ".huf");  //以二进制流形式打开文件  FILE \* out = fopen(filename, "wb");  //写文件  fwrite(&sHead, sizeof(HEAD), 1, out);  //写压缩后的编码  fwrite(pBuffer, sizeof(char), nSize, out);  //关闭文件，释放文件指针  fclose(out);  out = NULL;  cout << "生成压缩文件：" << filename << endl;  int len = sizeof(HEAD) + strlen(pFilename) + 1 + nSize;  return len;  }  //将字符串转换成字节  char Str2byte(const char \* pBinStr)  {  char b = 0x00;  for (int i = 0; i < 8; i++)  {  b = b << 1; //左移一位  if (pBinStr[i] == '1')  {  b = b | 0x01;  }  }  return b;  Huffman.h  #ifndef HUFFMAN\_H  #define HUFFMAN\_H  //Huffman's tree node  typedef struct {  int weight;  int parent;  int lchild;  int rchild;  }HTNode, \*HuffmanTree;  typedef char \*\*HuffmanCode;//Huffman Code  //create huffman tree  /\*  函数名：CreateHuffmanTree（）  功能：create huffman tree  参数：HuffmanTree pHT, int weight[], int n  返回值：int:FALSE-失败；TRUE-成功；  \*/  int CreateHuffmanTree(HuffmanTree pHT, int weight[], int n);  /\*  函数名：Select（）  功能：查找Huffman树节点数组中权值最小的节点  参数：HuffmanTree &pHT, int i, int &s1, int &s2  返回值：void  \*/  void Select(HuffmanTree &pHT, int i, int &s1, int &s2);  /\*  函数名：TestHuTree（）  功能：测试huffman树  参数：HuffmanTree pHT  返回值：int  \*/  int TestHuTree(HuffmanTree pHT);  /\*  函数名：HuffmanCoding（）  功能：生成Huffman编码  参数：HuffmanCode &pHC, HuffmanTree &pHT  返回值：int  \*/  int HuffmanCoding(HuffmanCode &pHC, HuffmanTree &pHT);  /\*  函数名：TestHufCode（）  功能：测试生成Huffman编码  参数：int root, HuffmanTree &pHT, HuffmanCode &pHC  返回值：void  \*/  void TestHufCode(int root, HuffmanTree &pHT, HuffmanCode &pHC);  #endif // !HUFFMAN\_H  Huffman.cpp  #define \_CRT\_SECURE\_NO\_WARNINGS  #include"Huffman.h"  #include<iostream>  #include<malloc.h>  #define OK 1  #define ERROR 0  using namespace std;  //create Huffman tree  int CreateHuffmanTree(HuffmanTree pHT, int weight[], int n)  {  int s1, s2, i;  int m = 2 \* n - 1;    //initialization  for (i = 1; i <= n; i++) {  pHT[i].weight = weight[i - 1];  pHT[i].lchild = 0;  pHT[i].rchild = 0;  pHT[i].parent = 0;  }  for (i = n + 1; i <= m; i++) {  pHT[i].weight = 0;  pHT[i].lchild = 0;  pHT[i].rchild = 0;  pHT[i].parent = 0;  }  for (i = n + 1; i <= m; i++)  {  //从pHT[1...i-1]中选择parent为0且weight最小的两个结点，其序号分别s1,s2  Select(pHT, i - 1,s1,s2);  //修改s1,s2结点的父指针parent  pHT[s1].parent = i;  pHT[s2].parent = i;  //修改i结点的左右孩子指针  pHT[i].lchild = s1;  pHT[i].rchild = s2;  pHT[i].weight = pHT[s1].weight + pHT[s2].weight;//修改权值  }  return OK;  }  // 查找Huffman树节点数组中权值最小的节点  void Select(HuffmanTree &pHT, int i, int &s1, int &s2) {  int minValue = 0x7FFFFFFF;  //找到最小的一个权值  for (int j = 1; j <= i; j++) {  if (pHT[j].parent == 0 && pHT[j].weight < minValue) {  minValue = pHT[j].weight;  s1 = j;  }  }  minValue = 0x7FFFFFFF;  //找到倒数第二小的权值  for (int j = 1; j <= i; j++) {  if (j != s1 && pHT[j].parent == 0 && pHT[j].weight < minValue) {  minValue = pHT[j].weight;  s2 = j;  }  }  }  int TestHuTree(HuffmanTree pHT) {  cout << "huffmanTree的每个结点的信息为" << endl;  cout << "Byte\t\tWeight\tParent\tLchild\tRchild\n";  for (int i = 1; i < 512; i++) {  //判断语句为了对齐格式  if(i<=99)  cout << "pHT[" << i << "]\t\t" << pHT[i].weight << "\t" << pHT[i].parent << "\t" << pHT[i].lchild << "\t" << pHT[i].rchild << endl;  else  cout << "pHT[" << i << "]\t" << pHT[i].weight << "\t" << pHT[i].parent << "\t" << pHT[i].lchild << "\t" << pHT[i].rchild << endl;  }  return OK;  }  //生成Huffman编码  int HuffmanCoding(HuffmanCode &pHC, HuffmanTree &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树时用做节点的状态标志  int p = 511; //根节点  while (p != 0) {  //向左  if (pHT[p].weight == 0) {  pHT[p].weight = 1;  if (pHT[p].lchild != 0) {  p = pHT[p].lchild;  cd[cdlen++] = '0';  }  //登记叶子节点的字符的编码  else if (pHT[p].rchild == 0) {  pHC[p] = (char\*)malloc((cdlen + 1) \* sizeof(char));  cd[cdlen] = '\0';  strcpy(pHC[p], cd);//复制编码  }  }  //向右  else if (pHT[p].weight == 1) {  pHT[p].weight = 2;  //右孩子为叶子节点  if (pHT[p].rchild != 0) {  p = pHT[p].rchild;  cd[cdlen++] = '1';  }  }  //退回父节点，编码长度减一  else {  pHT[p].weight = 0;  p = pHT[p].parent;  cdlen--;  }  }  return OK;  }  void TestHufCode(int root, HuffmanTree &pHT, HuffmanCode &pHC) {  if (root <= 1) return;  if (pHT[root].lchild == 0 && pHT[root].rchild == 0)  printf("0x%02X\t%s\n", root - 1, pHC[root - 1]);  if (pHT[root].lchild)//访问左孩子  TestHufCode(pHT[root].lchild, pHT, pHC);  if (pHT[root].rchild)//访问右孩子  TestHufCode(pHT[root].rchild, pHT, pHC);  }  Main.cpp  #include "pch.h"  #include<iomanip>  #include <iostream>  #include "Compress.h"  #include"Huffman.h"  using namespace std;  int main()  {  cout << "=========Huffman文件压缩=======" << endl;  cout << "请输入文件名：";  char filename[256];  cin >> filename;  if (Compress(filename) == 1) {  cout << "\n完成！" << endl;  }  else {  cout << "\n失败" << endl;  }  return 0;  }  Main.h  #pragma once  int main();  pch.h  #ifndef PCH\_H  #define PCH\_H  // TODO: 添加要在此处预编译的标头  #endif //PCH\_H  Pch.cpp  #include "pch.h"  // 一般情况下，忽略此文件，但如果你使用的是预编译标头，请保留它。   1. 调试说明（调试手段、过程及结果分析）    * + 1. 按 Ctrl+F5，编译运行程序。        2. 输入文件名      * + - 1. 压缩文件 对文件进行压缩，控制台输出文件长度及压缩比。压缩后生成“Pic.bmp.huf”文件，保存在与 Pic.bmp 相同的目录下。     第三部分：实验小结、收获与体会  通过本次实验我了解到了Huffman压缩算法的使用以及具体的代码实现，也学习到了通过文件中字符的重复为权值构建huffman树，利用二叉树的特点，遍历二叉树可以得到所有叶子节 点的编码。利用结构体数组的特点，反过来从叶子节点寻找父节点，逆向记 录访问路径，生成 Huffman 编码。 | | | | | |
|  | | | | | |