**同济大学计算机系**

**高级程序设计大作业报告**

**I**

****

**学 号 1950000**

**姓 名 一二三**

**专 业 计算机类**

**班 级 18级计算机二班**

**授课老师 陈宇飞**

**完成日期 2019/4/29**

文件压缩小程序大作业

一、实验内容

ALPD公司(爱乐普第)名下有一个网站 (ALPDOJ, 爱乐普第Orange Juice) 用于在线预约橙汁。该公司的橙汁特别好喝而且十分畅销，导致网站访问量特别大，每天都有上百人登录网站预约橙汁，所以导致公司的日志记录非常的长。公司负责人wws在归档网站日志的时候发现公司服务器硬盘实在是太小导致不够存了，所以在考虑怎么压缩数据以保存这些一点也不重要的数据。可是wws实在是太菜了，思来想去也不知道这份文件该怎么压缩。随着日子渐渐推移服务器的硬盘空间越来越小，一旦空间到了0那么整个网站都会崩溃。这可急坏了wws，于是他找到了你，希望你能帮帮他压缩这些文件。

**二、设计思路与功能描述**

**2.1 项目思路说明**

**2.1.1 压缩算法的选择**

通过对OJ网站上给出内容与要求的分析可知，本次大作业的主要目的是设计一套压缩与解压程序，实现文本内容（即题目给出ser.log）的有效压缩。而正如OJ平台的介绍所述，压缩分为无损压缩与有损压缩，但考虑到有损压缩一般多用于视频、图片的压缩处理上。而且在正常情况下，人们对文本中缺字少字的敏感度要远高于对图片像素降低的敏感度。

所以基于上述的综合考量，本次大作业的目标可以细化为：使用某种无损压缩方式实现ser.log的压缩与解压。

而OJ平台上给出的示例程序采取的是有损压缩的方式，所以在本次大作业的编写过程中不对平台给出代码进行参考。通过在网络上查阅相关资料可以得知，目前常见的无损压缩方法主要有如下几类：

①哈夫曼编码；

②Lempel-Ziv压缩算法；

③算术编码；

哈夫曼编码主要是基于各个文字在待压缩文本中的出现频度，构建起一颗huffman树，该huffman树的主要特点是每一个叶结点都是一个文字，而每一个中间结点一定不包含文字，所以当我们想要表示某个文字时，只需要说明该文字是从根节点出发，每到一个新节点是向左还是向右即可——而这种表示方式可以用简单的bool变量0-1来完成。同时huffman树能够保证出现频度高的字母能够对应较短的编码，从而使压缩后的文本尽可能小。

LZ算法主要分为LZ77算法及其变种与LZ78算法及其变种。LZ算法的共同特征就是用前面出现过的文本来替换之后再次出现文本。而替换时只需要记录替换位置即可，从而达到压缩文本的目的。同时，由于被替换的内容一定可以从之前已经生成的文本中找到，所以不用像哈夫曼编码一样需要给出字母表。而LZ系列算法中的不同算法则给出了不同的替换方式，其中不乏有著名的LZM、LZO等应用广泛的压缩算法

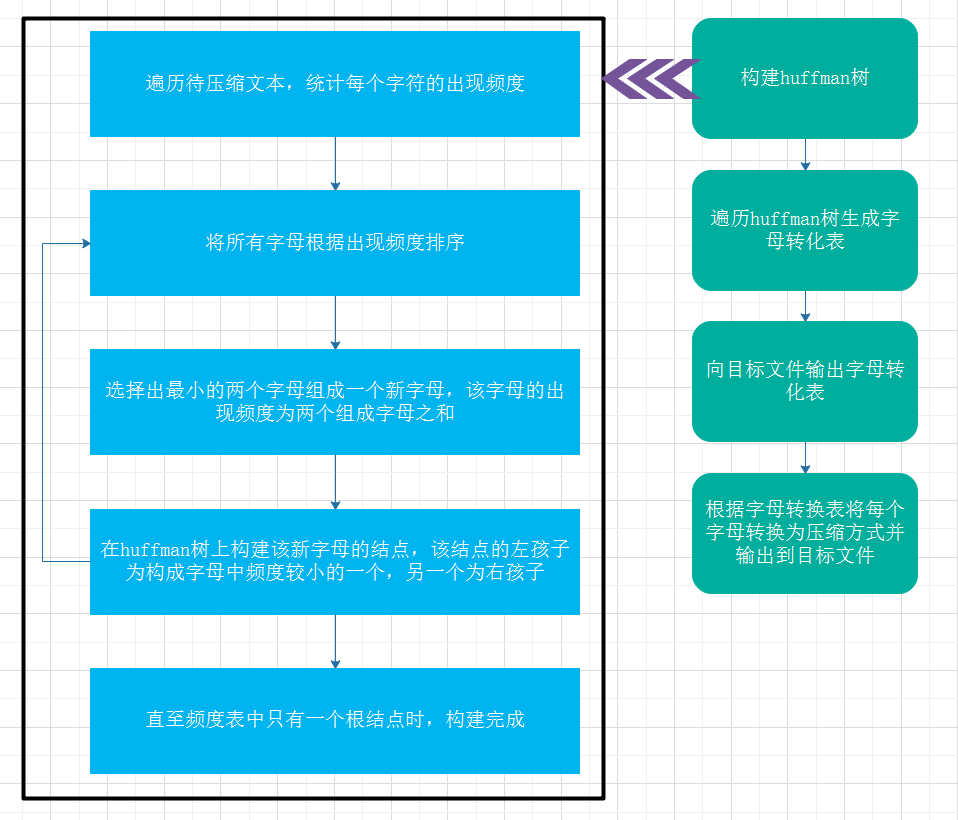
算术编码的主要思想就是将文本文件转换为0-1的二进制编码，然后用一个一个相互独立的实数来表示0和1之间的距离，随着消息的逐渐变长，从而逐步形成更为紧密的压缩文本。

而在本次大作业中，考虑到本人在上学期的《数据结构》课程上已经对哈夫曼编码和LZ77编码有所了解掌握，故在本次大作业中，将选用这两种无损编码方式来进行ser.log的压缩处理。

**2.1.2 哈夫曼编码的思路说明**

**2.1.2.1 哈夫曼编码压缩文件**

哈夫曼编码算法压缩文件的主要思路如下图所示：



正如上图所示，huffman编码压缩文件过程主要有四步，以下将对这四步内容进行详细说明：

1）构建huffman树

为了让出现频度较高的字母能够对应较短的编码，huffman树在构建时就选择从出现频度低的叶结点开始构建，从而使得这些出现频度低的叶结点占据的层次较低，把层次较高的叶子留给出现频度高的叶结点。

所以在构建时，首先要获取一张所有字符的出现频度表，然后选取出频度表中出现次数最小的两个字符组合为一个新的“字符”，而该新“字符”的出现频度即为两个子字符的出现频度之和，再将该新“字符”放回到频度表中参与后续比较与运算。

与此同时，每产生一个新“字符”，生成一个新“字符”对应的结点，使得该结点的左孩子指向出现频度较小的子字符对应的叶结点、右孩子指向另一个子字符对应的叶结点。该生成过程随着频度表的更新操作往复循环，直至频度表中只剩一个字符，即所有节点都汇聚在了一个根结点上时，说明huffman树构建完成，退出操作。

2）遍历huffman树生成字母转化表

huffman树的本质其实就是一个较为特殊的二叉树，所以huffman树的遍历可以直接套用一般二叉树的遍历规则，每当遍历到一个叶结点时，就在转换表中加入该叶结点所表示的字母，其对应的转换方式即为从根结点走到该结点的每一步是向左还是向右的组合（在本次大作业中用0表示向左，1表示向右）。

考虑到这里对到达叶结点的路径较为关心，所以采用深度优先遍历的方式对huffman树进行搜索（在本次大作业中选择了DLR的方式）。

而对于字母表的生成，考虑到log.src文件中的字符均在ASCII码的表示范围内，所以在本题构建了一个长度为256的字符串集合，用以对应每一个ASCII码的转换关系。

3）向目标文件输出字母转化表

由于huffman编码方式不具备让编码文本自我解码的能力，所以会在压缩文本的开始先输入字母转化表，以便解压时使用。而在本次大作业中，通过预先编码发现，log.src文本中，最长的字母转化方式长度为17，所以这里采用int类型4个字节（32个bit位）来存储每一个字母的转化方式。

而考虑到待编译字符都在ASCII码表示范围内，所以采用char类型1个字节（8个bit位）来存储每一个字母。同时，由于字母表读取时，0和1都是有效位，所以还需要额外注明该段转化关系是前多少位，该位数的注明采用char类型1个字节。

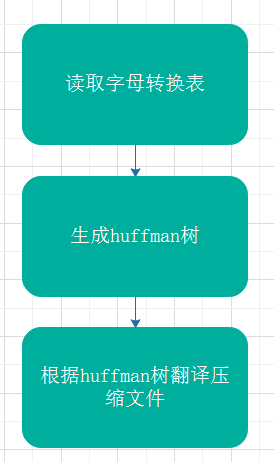
4）根据字母转化表输出压缩文件

再次读取待转化文本，将每一个字母转化为字母表中对应的转化方式再输出到目标文件。

在这里要特别指出的是由于huffman编码能够保证任意两种转化方式一定互不为对方从第一位开始的子串，所以在压缩时不用考虑字母与字母之间分隔符的问题，直接输出即可。

**2.1.2.2 哈夫曼编码解压文件**

相交于压缩文件来说，huffman编码解压文件就变得较为简单。其主要流程为：



如图所示，huffman编码解压文件主要有上述三步，以下将对这三个步骤进行详细说明：

1）读取字母转换表

根据之前压缩时商定的方式（即1个字节的字母+1个字节的编码长度+4个字节的编码方式），可以识别文本中的每一对字母转换表。

值得一提的是，为了在解压时能够知晓每一步的读取到哪里结束，本次大作业在设计压缩算法时，在文本的最开始额外输入了两个数据，分别代表字母转换表的长度与文本的总字符。前者为char型占据1个字节，后者为int型占据4个字节。

这样在解压时，就可以根据这两个数据的限制，完成字母转换表与文本内容的分割以及文本翻译的结束判断。

2）生成huffman树：

该步骤其实是与第一个步骤一并进行的，其具体方式为从根结点开始，按照0和1的指引转换到左孩子或是右孩子，如果当前结点没有要指向走向的孩子，那就临时生成一个孩子并转移到该孩子继续转移。

当完成该字母对应的所有转移后，当前所在的结点即为该字母对应的叶结点。在huffman树上对该叶结点进行标记，一遍后续的使用。

3）根据huffman树翻译压缩文件

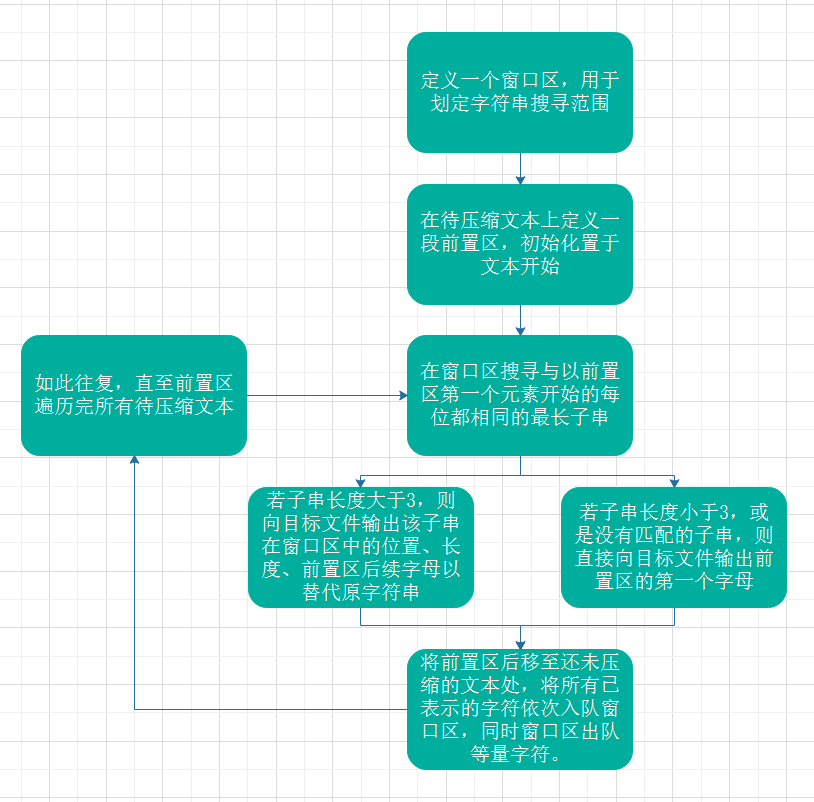
该过程即为解压原压缩文本，其方式是让工作指针从根结点开始，逐一读取压缩文件中的每一个bit位，0表示工作指针转移到左孩子，1表示工作指针指向右孩子。

如此循环读入bit，直至工作指针指向了一个叶结点，则立即输出该叶结点，并将工作指针重新指向根结点翻译下一个字母。从而可以逐字母地解压得到解压后的原始文件。

**2.1.3 lz77编码的思路说明**

**2.1.3.1 lz77编码压缩文件**

lz77编码压缩文件的主要思路如下：



正如上图所示，lz77编码压缩文件的主要步骤都集中在一个不同变化的循环内，一下将对该循环中的内容以及两个区域的定义进行详细说明。

1）窗口区与前置区的建立

窗口区相当于是“暂存”了指定长度的已经被表示过的文本内容，划定了重复子串的搜索范围；而前置区则用于指定当前要进行表示的文本位置，同时划定了重复子串的最大长度。

2）重复子串搜索

正如2.1所述，lz系列压缩算法的核心就是在不借助字母表的情况下，使用已经解压完成的文本来进行当前文本的解压。而对于lz77算法来说，这种复用的方式即体现在搜索当前要表示的字符（串）是否在前述文本中已经被表述过。

为了让压缩结果最好，自然要找寻能够让所有所有更多的字符被一同表示的重复子串，但同时，又为了避免因为搜索范围过大而导致时间复杂度升级的情况，只考虑窗口区范围内的文本内容即可。

3）目标文件输出

由于每一个调用前序子串的操作都需要指定调用位置、调用长度以及后缀的一位待压缩文本（都是char类型，共计3个字节），所以只有当最长的重复子串长度大于等于3时，才有必要使用调用前序子串的方式来生成压缩文件。而在其他情况下，只需要直接将前置区的第一位文本输出即可。

但在输出到目标文件时要特别注意的是，在解压文件的时候，两种不同的输出方式对应的解压方式也有所不同，所以在本次大作业中，采用了前置“标志位”的方式来告诉解压程序这里是直接输出还是调用前序子串，本次大作业的标志位为char类型，占据1个字节。当它为‘0’时，说明采用调用前序子串的方式输出；为‘1’时说明采用直接输出的方式。

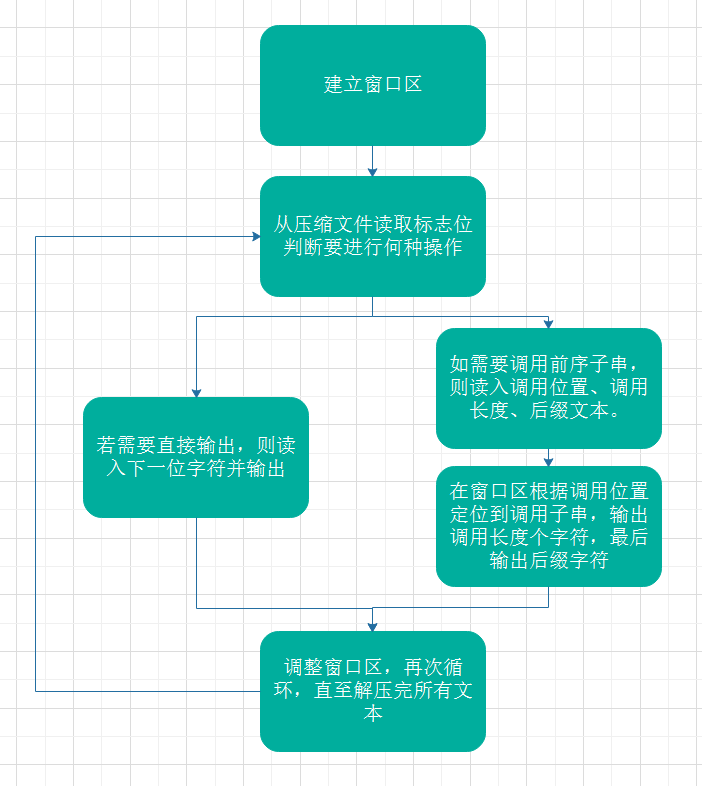
4）窗口区与前置区后移

当完成当前字符或字符串的编码之后，就应当将前置区移动到下一个要进行编码的位置，而这些刚刚编码完成的字符则应当要按顺序进入到窗口区的队列中去（先进后出）。但同时，为了保证窗口区的长度保持不变，就需要让窗口区队列出队等量的字符。

至此便完整地完成了一段的文本的压缩，并已经调整好了窗口区和前置区的位置以便于下一步操作。故可以返回至第（2）步，循环往复，直至所有的待压缩文本都被压缩编码输出到了目标文件。

**2.1.3.2 lz77编码解压文件**

使用lz77编码方式解压文件的基本思路为：

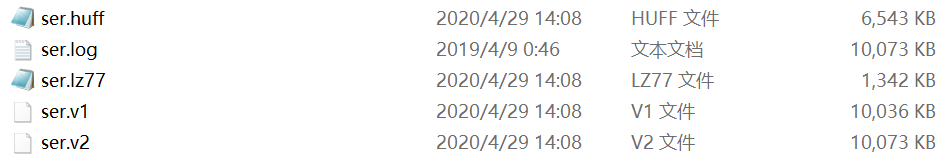


正如上图所示，lz77编码解压文件的主要步骤即为压缩文件的“倒置”，只需要进入循环并逐个步骤进行操作即可。其过程较为简单，故不在此进行赘述。

**2.2 功能描述**

**2.2.1 压缩文件生成说明**

调用本次大作业所设计的程序对ser.log文件进行压缩，可以得到生成的文件目录如下：



其中：

ser.log文件是OJ上提供的待压缩文件；

ser.huff文件是使用huffman编码压缩得到的文件；

ser.v1文件是使用huffman编码解压ser.huff得到的文件；

ser.lz77文件是使用lz77编码压缩得到的文件；

ser.v2文件是使用lz77编码解压ser.lz77得到的文件。

在这里要特别指出的是，通过截图不难发现ser.v1的文件大小并不是严格等于源文件ser.log。通过人工对文本文件的反复比对，发现两者在文本内容、空格等可能会影响阅读体验的方面均完全一样，推测可能是由于换行符的不同或是其他不可见字符（例如超过ASCII表示范围的字符）的影响导致的。

由于这里的错误完全不会影响阅读体验，故在此可认为这种纰漏是可以被允许的。

**2.2.2 命令行窗口说明**

由于本次大作业在OJ平台上标明了有运行时间、压缩比的要求，故将这些信息一并在命令行窗口进行展示，同时加入了人性化提示。

当使用本程序压缩ser.log时，命令行窗口的显示内容如下：



通过本例对比不难看出，huffman编码方式和lz77编码方式相比各有利弊。huffman算法的压缩时间与解压时间均短于lz77算法；但lz77的压缩效率却要优于huffman算法。

三、项目亮点与情况说明

**3.1 项目亮点**

①本次大作业的所有内容均是在本人浏览学习了网络介绍资料后，独自开发的，除了<vector>这一允许的头文件外，没有使用任何未学习的内容，自认为达到了高程群中所谓的“生写”标准。

②本次大作业同时完成了两种不同压缩算法的实现，以便于相互比较，体现各自优劣。

③两种算法的具体实现都进行了一定优化，保证两种算法的运行时间都远低于OJ平台要求的15秒，同时均有不错的压缩比。

④本次大作业实现的huffman编码和lz77编码都是目前使用较为广泛的压缩算法，自认为其符合OJ要求的“高阶的核心算法”；同时，由于本次大作业实现的是无损压缩，压缩方式与文本内容本身无关，所以该算法的适用范围应该比有损压缩要广。

⑤自认为命令行窗口的内容弹出设计得较为人性化，能够清晰地比对两种压缩算法的特点。

**3.2 诸多情况说明**

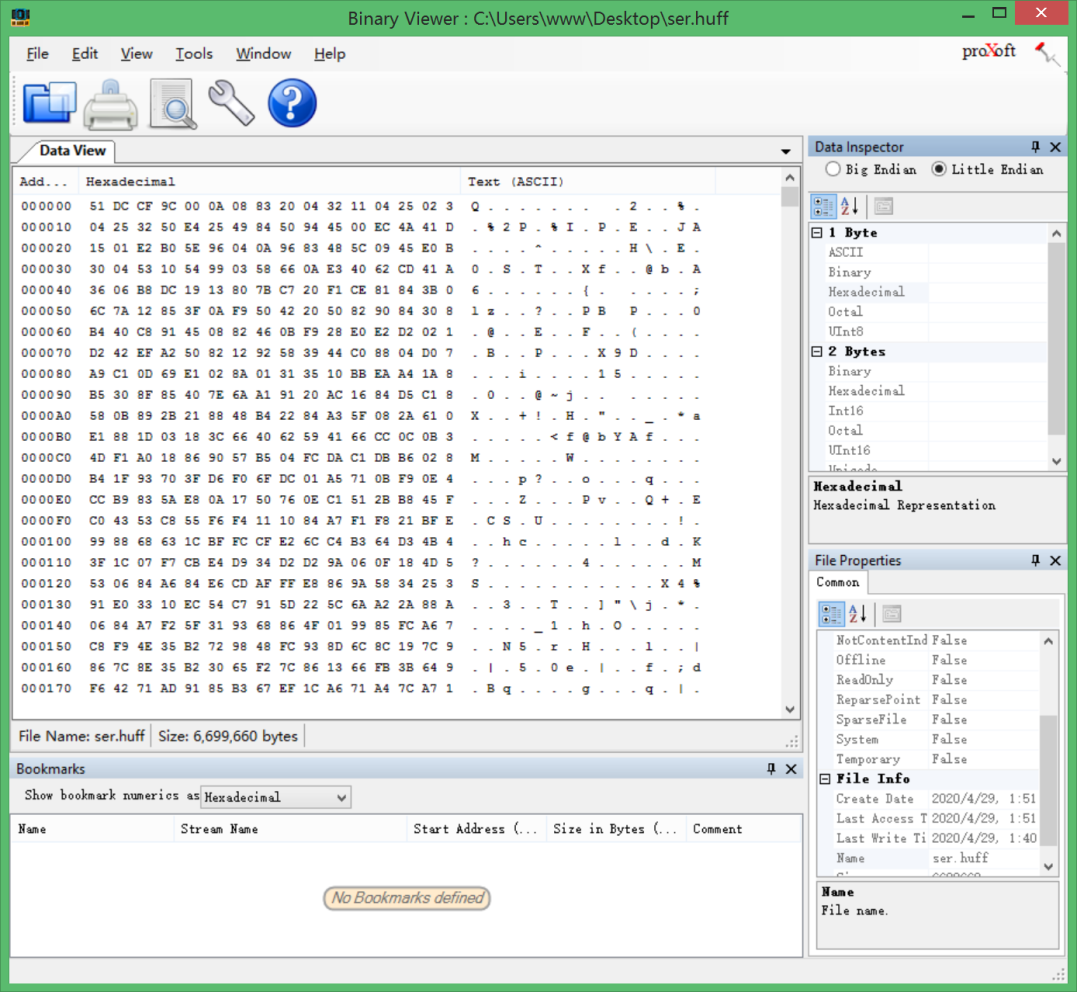
四、遇到的问题与解决方法

**4.1 遇到的问题**

在刚开始进行huffman编码的过程中，为了节约内存，我选择了将所有的内容都按照比特位相连接，然后每8位输出一次的方式来进行压缩文件的生成。但在我进行调试时，发现解压程序总会在压缩文件中识别乱码，从而导致.huff压缩文件无法正确解压。

**4.2 解决方法**

下图为我错误输出文件的二进制查看结果：



五、心得体会

六、源代码

#include <iostream>

#include <fstream>

#include <vector>

#include <ctime>

using namespace std;

//首先定义huffman压缩方法

struct huffnode {

bool flag = true;

char c = '0';

int value = 0;

huffnode \*left = NULL;

huffnode \*right = NULL;

}\*head\_for\_huffman, \*head;

vector<huffnode> vec\_for\_huffman;

char trans\_for\_huffman[256][256];

char transnum\_for\_huffman = 0;

int wordnum\_for\_huffman = 0;

vector<bool> trans\_temp\_for\_huffman;

char buf\_for\_huffman = 0;

short bufnum\_for\_huffman = 0;

int findmin() {

int min, mini;

for (unsigned int i = 0; i < vec\_for\_huffman.size(); i++) {

if (i == 0) {

min = vec\_for\_huffman[i].value;

mini = 0;

}

else {

if (min > vec\_for\_huffman[i].value) {

min = vec\_for\_huffman[i].value;

mini = i;

}

}

}

return mini;

}

void transcode(huffnode \*p) {

if (!p->flag) {

for (unsigned int i = 0; i < trans\_temp\_for\_huffman.size(); i++) {

if (trans\_temp\_for\_huffman[i]) {

trans\_for\_huffman[(int)(p->c)][i] = '1';

}

else {

trans\_for\_huffman[(int)(p->c)][i] = '0';

}

}

transnum\_for\_huffman++;

trans\_for\_huffman[(int)(p->c)][trans\_temp\_for\_huffman.size()] = '\0';

return;

}

else {

trans\_temp\_for\_huffman.push\_back(false);

transcode(p->left);

trans\_temp\_for\_huffman.pop\_back();

trans\_temp\_for\_huffman.push\_back(true);

transcode(p->right);

trans\_temp\_for\_huffman.pop\_back();

}

}

void huffmaketree(const char add[]) {

int alpha[256], o1;

char temp;

fstream fp;

fp.open(add, ios::in);

huffnode \*p = NULL, \*uu = NULL;

if (!fp.is\_open()) {

cerr << "这个文件打不开！！！";

exit(-1);

}

for (int i = 0; i < 256; i++) {

alpha[i] = 0;

}

for (;;) {

fp >> noskipws >> temp;

wordnum\_for\_huffman++;

alpha[(int)temp]++;

if (fp.eof()) {

break;

}

}

fp.close();

for (int i = 0; i < 256; i++) {

if (alpha[i] != 0) {

p = new huffnode;

p->c = (char)i;

p->flag = false;

p->value = alpha[i];

vec\_for\_huffman.push\_back(\*p);

}

}

for (;;) {

if (vec\_for\_huffman.size() == 1) {

head\_for\_huffman = &vec\_for\_huffman[0];

vec\_for\_huffman.clear();

break;

}

uu = new huffnode;

p = new huffnode;

o1 = findmin();

p->c = vec\_for\_huffman[o1].c;

p->flag = vec\_for\_huffman[o1].flag;

p->value = vec\_for\_huffman[o1].value;

p->left = vec\_for\_huffman[o1].left;

p->right = vec\_for\_huffman[o1].right;

vec\_for\_huffman.erase(vec\_for\_huffman.begin() + o1);

uu->left = p;

p = new huffnode;

o1 = findmin();

p->c = vec\_for\_huffman[o1].c;

p->flag = vec\_for\_huffman[o1].flag;

p->value = vec\_for\_huffman[o1].value;

p->left = vec\_for\_huffman[o1].left;

p->right = vec\_for\_huffman[o1].right;

vec\_for\_huffman.erase(vec\_for\_huffman.begin() + o1);

uu->right = p;

uu->flag = true;

uu->value = uu->left->value + uu->right->value;

vec\_for\_huffman.push\_back(\*uu);

}

transcode(head\_for\_huffman);

}

void huffouttree(const char addin[], const char addout[]) {

fstream inn, outt;

char temp, tt, lil;

inn.open(addin, ios::in | ios::binary);

int tool;

if (!inn.is\_open()) {

cerr << "文件打不开！！！！！！";

exit(-1);

}

outt.open(addout, ios::out | ios::binary);

if (!outt.is\_open()) {

cerr << "文件打不开！！！！！！";

exit(-1);

}

outt.write((char\*)&transnum\_for\_huffman, sizeof(transnum\_for\_huffman));

outt.write((char\*)&wordnum\_for\_huffman, sizeof(wordnum\_for\_huffman));

for (int i = 0; i < 256; i++) {

if (trans\_for\_huffman[i][0] != '1'&&trans\_for\_huffman[i][0] != '0') {

continue;

}

temp = i;

for (int k = 0; k < 8; k++) {

tt = (temp >> (7 - k)) % 2;

buf\_for\_huffman = buf\_for\_huffman \* (char)2 + tt;

bufnum\_for\_huffman++;

if (bufnum\_for\_huffman == 8) {

outt.write((char\*)&buf\_for\_huffman, sizeof

(buf\_for\_huffman));

bufnum\_for\_huffman = 0;

buf\_for\_huffman = 0;

}

}

for (lil = 0; trans\_for\_huffman[i][lil] != '\0'; lil++);

for (int k = 0; k < 8; k++) {

tt = (lil >> (7 - k)) % 2;

buf\_for\_huffman = buf\_for\_huffman \* (char)2 + tt;

bufnum\_for\_huffman++;

if (bufnum\_for\_huffman == 8) {

outt.write((char\*)&buf\_for\_huffman, sizeof

(buf\_for\_huffman));

bufnum\_for\_huffman = 0;

buf\_for\_huffman = 0;

}

}

tool = 0;

for (int j = 0; trans\_for\_huffman[i][j] != '\0'; j++) {

tool = tool \* 2 + trans\_for\_huffman[i][j] - '0';

}

outt.write((char\*)&tool, sizeof(tool));

}

for (;;) {

inn >> noskipws >> temp;

for (int i = 0; trans\_for\_huffman[(int)temp][i] != '\0'; i++) {

buf\_for\_huffman = buf\_for\_huffman \* (char)2 + (trans\_for\_huffman

[(int)temp][i] - '0');

bufnum\_for\_huffman++;

if (bufnum\_for\_huffman == 8) {

outt.write((char\*)&buf\_for\_huffman, sizeof

(buf\_for\_huffman));

bufnum\_for\_huffman = 0;

buf\_for\_huffman = 0;

}

}

if (inn.eof()) {

for (;;) {

if (bufnum\_for\_huffman == 0) {

break;

}

buf\_for\_huffman = buf\_for\_huffman \* (char)2;

bufnum\_for\_huffman++;

if (bufnum\_for\_huffman == 8) {

outt.write((char\*)&buf\_for\_huffman, sizeof

(buf\_for\_huffman));

bufnum\_for\_huffman = 0;

buf\_for\_huffman = 0;

}

}

break;

}

}

inn.close();

outt.close();

}

void huffman(const char add1[], const char add2[]) {

huffmaketree(add1);

huffouttree(add1, add2);

}

//这里给出huffman的解码方式

char buf\_for\_dehuffman;

short bufnum\_for\_dehuffman = 8;

fstream fin\_for\_dehuffman, fout\_for\_dehuffman;

char dehuffman\_get() {

if (bufnum\_for\_dehuffman == 8) {

fin\_for\_dehuffman.read((char\*)&buf\_for\_dehuffman, sizeof(buf\_for\_dehuffman));

bufnum\_for\_dehuffman = 0;

}

bufnum\_for\_dehuffman++;

return (buf\_for\_dehuffman >> (8 - bufnum\_for\_dehuffman)) % 2;

}

void dehuffman(const char inn[], const char outt[]) {

char transnum;

int wordnum;

char wordc, too, haha;

int len, tool;

huffnode \*head = NULL, \*p = NULL, \*q = NULL;

fin\_for\_dehuffman.open(inn, ios::in | ios::binary);

if (!fin\_for\_dehuffman.is\_open()) {

cerr << "文件打不开！！！！！！";

exit(-1);

}

fout\_for\_dehuffman.open(outt, ios::out | ios::binary);

if (!fout\_for\_dehuffman.is\_open()) {

cerr << "文件打不开！！！！！！";

exit(-1);

}

fin\_for\_dehuffman.read((char\*)&transnum, sizeof(transnum));

fin\_for\_dehuffman.read((char\*)&wordnum, sizeof(wordnum));

for (int i = 0; i < transnum; i++) {

wordc = 0;

for (int uhi = 0; uhi < 8; uhi++) {

if (bufnum\_for\_dehuffman == 8) {

fin\_for\_dehuffman.read((char\*)&buf\_for\_dehuffman, sizeof

(buf\_for\_dehuffman));

bufnum\_for\_dehuffman = 0;

}

bufnum\_for\_dehuffman++;

haha = (buf\_for\_dehuffman >> (8 - bufnum\_for\_dehuffman)) % 2;

wordc = wordc \* 2 + haha;

}

len = 0;

for (int uhi = 0; uhi < 8; uhi++) {

if (bufnum\_for\_dehuffman == 8) {

fin\_for\_dehuffman.read((char\*)&buf\_for\_dehuffman, sizeof

(buf\_for\_dehuffman));

bufnum\_for\_dehuffman = 0;

}

bufnum\_for\_dehuffman++;

haha = ((buf\_for\_dehuffman >> (8 - bufnum\_for\_dehuffman)) % 2);

len = len \* 2 + haha;

}

if (i == 0) {

head = new huffnode;

}

tool = 0;

p = head;

fin\_for\_dehuffman.read((char\*)&tool, sizeof(tool));

for (int j = 0; j < len; j++) {

too = (tool >> (len - j - 1)) % 2;

if (too == 0) {

if (p->left == NULL) {

q = new huffnode;

p->left = q;

}

p = p->left;

}

else {

if (p->right == NULL) {

q = new huffnode;

p->right = q;

}

p = p->right;

}

if (j == len - 1) {

p->flag = false;

p->c = wordc;

}

}

}

for (int i = 0; i < wordnum; i++) {

p = head;

for (;;) {

if (p->flag == false) {

break;

}

if (bufnum\_for\_dehuffman == 8) {

fin\_for\_dehuffman.read((char\*)&buf\_for\_dehuffman, sizeof

(buf\_for\_dehuffman));

bufnum\_for\_dehuffman = 0;

}

bufnum\_for\_dehuffman++;

haha = (buf\_for\_dehuffman >> (8 - bufnum\_for\_dehuffman)) % 2;

too = haha;

if (too == 0) {

p = p->left;

}

else {

p = p->right;

}

}

fout\_for\_dehuffman.write((char\*)&(p->c), sizeof(p->c));

}

fin\_for\_dehuffman.close();

fout\_for\_dehuffman.close();

}

//这里给出lz77的编码方式

const int WINDOW\_for\_lz77 = 8192;

vector <char> window\_for\_lz77;

const int ZOOM\_for\_lz77 = 100;

vector <char> zoom\_for\_lz77;

int getlen(int wi, int zo) {

int ans = 0, window\_for\_lz77i = wi, zoom\_for\_lz77i = zo;

for (;;) {

if (window\_for\_lz77[window\_for\_lz77i] == zoom\_for\_lz77[zoom\_for\_lz77i]) {

window\_for\_lz77i++;

zoom\_for\_lz77i++;

ans++;

if (window\_for\_lz77i == window\_for\_lz77.size() || zoom\_for\_lz77i == zoom\_for\_lz77.size()) {

return ans;

}

}

else {

return ans;

}

}

}

void lz77(const char inn[], const char outt[]) {

fstream fin, fout;

char temp;

int zoom\_for\_lz77i, window\_for\_lz77i;

bool flag = true;

int len = 0, lenmax, leni;

char buf;

short buf2;

fin.open(inn, ios::in | ios::binary);

if (!fin.is\_open()) {

cerr << "文件打不开！！！！！";

exit(-1);

}

fout.open(outt, ios::out | ios::binary);

if (!fout.is\_open()) {

cerr << "文件打不开！！！！！";

exit(-1);

}

for (int i = 0; i < ZOOM\_for\_lz77; i++) {

fin >> noskipws >> temp;

zoom\_for\_lz77.push\_back(temp);

}

for (int i = 0; i < WINDOW\_for\_lz77; i++) {

window\_for\_lz77.push\_back('\0');

}

for (;;) {

if (zoom\_for\_lz77.empty()) {

break;

}

lenmax = -1;

zoom\_for\_lz77i = window\_for\_lz77i = 0;

for (int i = 0; i < WINDOW\_for\_lz77; i++) {

if (window\_for\_lz77[i] == zoom\_for\_lz77[0]) {

len = getlen(i, 0);

if (len > lenmax) {

lenmax = len;

leni = i;

}

}

}

if (lenmax <= 3) {

buf = '0';

fout.write((char\*)&buf, sizeof(buf));

buf = zoom\_for\_lz77[0];

fout.write((char\*)&buf, sizeof(buf));

window\_for\_lz77.erase(window\_for\_lz77.begin());

window\_for\_lz77.push\_back(zoom\_for\_lz77[0]);

zoom\_for\_lz77.erase(zoom\_for\_lz77.begin());

if (zoom\_for\_lz77.empty()) {

break;

}

if (flag) {

fin >> noskipws >> temp;

zoom\_for\_lz77.push\_back(temp);

}

if (fin.eof()) {

flag = false;

}

}

else {

buf = '1';

fout.write((char\*)&buf, sizeof(buf));

buf2 = leni;

fout.write((char\*)&buf2, sizeof(buf2));

buf2 = lenmax;

fout.write((char\*)&buf2, sizeof(buf2));

for (int uhi = 0; uhi < lenmax; uhi++) {

window\_for\_lz77.erase(window\_for\_lz77.begin());

window\_for\_lz77.push\_back(zoom\_for\_lz77[0]);

zoom\_for\_lz77.erase(zoom\_for\_lz77.begin());

if (zoom\_for\_lz77.empty()) {

break;

}

if (flag) {

fin >> noskipws >> temp;

zoom\_for\_lz77.push\_back(temp);

}

if (fin.eof()) {

flag = false;

}

}

buf = zoom\_for\_lz77[0];

fout.write((char\*)&buf, sizeof(buf));

window\_for\_lz77.erase(window\_for\_lz77.begin());

window\_for\_lz77.push\_back(zoom\_for\_lz77[0]);

zoom\_for\_lz77.erase(zoom\_for\_lz77.begin());

if (zoom\_for\_lz77.empty()) {

break;

}

if (flag) {

fin >> noskipws >> temp;

zoom\_for\_lz77.push\_back(temp);

}

if (fin.eof()) {

flag = false;

}

}

}

fin.close();

fout.close();

}

//这里给出lz77的解码方式

const int WINDOW\_for\_delz77 = 8192;

vector <char> window\_for\_delz77;

void delz77(const char inn[], const char outt[]) {

fstream fin, fout;

fin.open(inn, ios::in | ios::binary);

char temp, buf;

short buf2, tool;

if (!fin.is\_open()) {

cerr << "文件打不开！！！！！";

exit(-1);

}

fout.open(outt, ios::out | ios::binary);

if (!fout.is\_open()) {

cerr << "文件打不开！！！！！";

exit(-1);

}

for (int i = 0; i < WINDOW\_for\_delz77; i++) {

window\_for\_delz77.push\_back('\0');

}

for (;;) {

if (fin.eof()) {

break;

}

fin.read((char\*)&buf, sizeof(buf));

if (fin.eof()) {

break;

}

if (buf == '0') {

fin.read((char\*)&buf, sizeof(buf));

fout.write((char\*)&buf, sizeof(buf));

window\_for\_delz77.erase(window\_for\_delz77.begin());

window\_for\_delz77.push\_back(buf);

}

else {

fin.read((char\*)&buf2, sizeof(buf2));

fin.read((char\*)&tool, sizeof(tool));

for (int i = 0; i < tool; i++) {

if (buf2 >= window\_for\_delz77.size()) {

continue;

}

temp = window\_for\_delz77[buf2];

if (temp == '\0') {

continue;

}

fout.write((char\*)&temp, sizeof(temp));

window\_for\_delz77.erase(window\_for\_delz77.begin());

window\_for\_delz77.push\_back(temp);

}

fin.read((char\*)&buf, sizeof(buf));

fout.write((char\*)&buf, sizeof(buf));

window\_for\_delz77.erase(window\_for\_delz77.begin());

window\_for\_delz77.push\_back(buf);

}

}

fin.close();

fout.close();

}

int main() {

fstream fp;

clock\_t ta, tb, tb1;

int size0, size1, size2;

double t1, t2, t3, t4;

fp.open("ser.log", ios::app);

if (fp.is\_open()) {

fp.seekg(0, ios::end);

size0 = fp.tellg();

fp.close();

}

else {

cerr << "这文件打不开！！！！";

exit(-1);

}

cout << "原ser.log大小为：" << size0 << "字节。" << endl << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << " 以下演示huffman压缩 " << endl;

ta = clock();

huffman("ser.log", "ser.huff");

cout << "huffman压缩成功！" << endl;

tb = clock();

cout << "huffman压缩文件用时：" << (t1 = ((double)tb - (double)ta)) / CLOCKS\_PER\_SEC << "秒" << endl;

ta = clock();

dehuffman("ser.huff", "ser.v1");

tb = clock();

cout << "huffman解压文件用时：" << (t3 = ((double)tb - (double)ta)) / CLOCKS\_PER\_SEC << "秒" << endl;

cout << "------------------------------------------------" << endl;

fp.open("ser.huff", ios::app);

if (fp.is\_open()) {

fp.seekg(0, ios::end);

size1 = fp.tellg();

fp.close();

}

else {

cerr << "这文件打不开！！！！";

exit(-1);

}

cout << "huffman压缩后ser.huff大小为：" << size1 << "字节。" << endl;

cout << "压缩比为：" << 100.00\*size1 / (double)size0 << "%。" << endl << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << " 以下演示lz77压缩 " << endl;

ta = clock();

lz77("ser.log", "ser.lz77");

cout << "lz77压缩成功！" << endl;

tb = clock();

cout << "lz77压缩文件用时：" << (t1 = ((double)tb - (double)ta)) / CLOCKS\_PER\_SEC << "秒" << endl;

ta = clock();

delz77("ser.lz77", "ser.v2");

tb = clock();

cout << "lz77解压文件用时：" << (t3 = ((double)tb - (double)ta)) / CLOCKS\_PER\_SEC << "秒" << endl;

cout << "------------------------------------------------" << endl;

fp.open("ser.lz77", ios::app);

if (fp.is\_open()) {

fp.seekg(0, ios::end);

size1 = fp.tellg();

fp.close();

}

else {

cerr << "这文件打不开！！！！";

exit(-1);

}

cout << "lz77压缩后ser.lz77大小为：" << size1 << "字节。" << endl;

cout << "压缩比为：" << 100.00\*size1 / (double)size0 << "%。" << endl << endl;

return 0;

}