

基于 Huffman Code 的文件压缩工具开发文档

算法实现思路：

PART ONE 压缩文件：首先以二进制读取的方式，遍历整个被压缩文件，获取各个节点出现的频率，存入数组 w ；接下来根据各节点出现频率构造 Huffman Tree（这里为加快生成速度，在当前最小权重节点的选取中应用了最小堆）；然后前序遍历 Huffman Tree，生成各个节点对应的 Huffman Code 并用额外数组存储。创建新的压缩文件，写入原文件的基本信息（最重要的是各节点的出现频率）。然后读取原文件，根据生成的 Huffman Code 数组把原文件中的节点转化为二进制数，暂时存储于缓冲区字符数组 $buff$ 内，直到 $buff$ 长度超过 8。以长度 8 为一个单位，截取 $buff$ 中 2 进制字符串，转化为 10 进制数写入压缩文件，直到 $buff$ 中元素不足 8 个。最后若 $buff$ 中剩余元素小于 8，以 0 补全后转换为 10 进制数写入压缩文件。

PART TWO 解压缩文件：读取压缩文件，获得原文件的基本信息以及各节点出现的频次，存入数组 w ，借以生成 Huffman Tree，同样生成各个节点对应的 Huffman Code。读取压缩文件中的 10 进制数，每读一个就转化为 2 进制并存放于缓冲区字符数组 $buff$ 内，直到 $buff$ 长度大于有效节点数（即出现的频次非零的节点数，表征了 Huffman Code 的最大长度）。根据 $buff$ 内数据，从 Huffman Tree 的根节点开始，为 0 就走向左子树，为 1 就走向右子树，直到走到叶子节点，然后把该叶子节点对应的 10 进制字符转化成 2 进制写入解压缩文件，直到读取完整个压缩文件。结束时如果出现 $buff$ 内出现 2 个以上的连续 0，说明是人为补上的不用写入解压缩文件。

注：本次压缩工具开发实质上只实现了利用 Huffman Code 对文件进行压缩，但实际上对文件夹的压缩处理原理是相同的，但自己实现复杂度太高，一般常见解决方法是使用 zlib 和 minizip 库来实现。在课后和陈碧欢老师交流以后，由于期末时间精力有限，陈老师同意我们实现对文件的压缩即可。

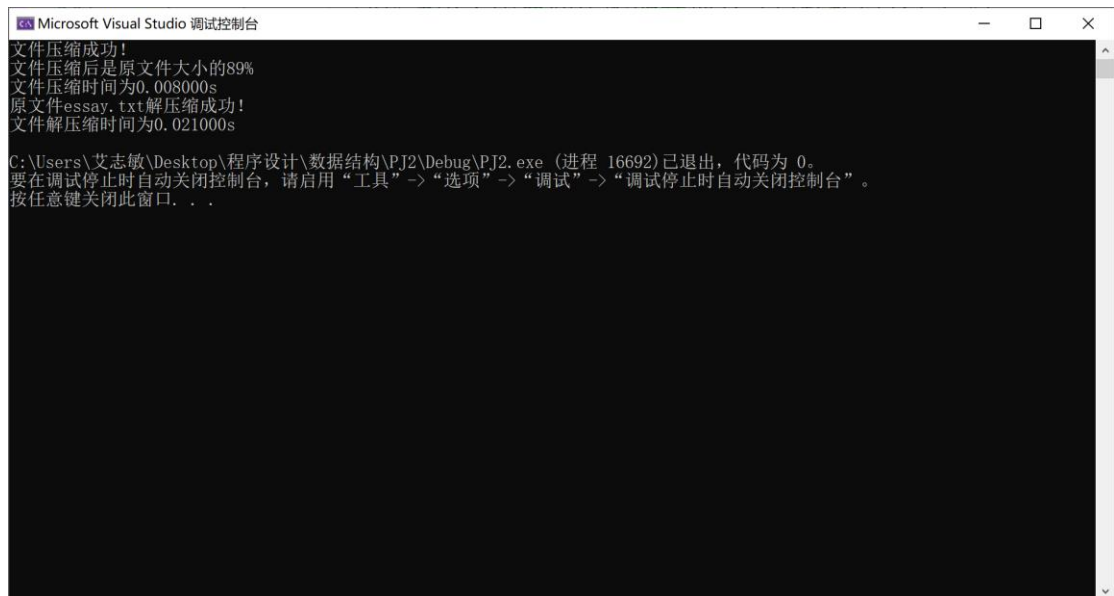
开发当中遇到的问题：

- 1、 构造 Huffman Tree 时使用的 Minheap.h 头文件是需要进行特殊改造的，以 Huffman Node 中节点出现频次作为判断大小的依据，这种做法可以提升 Huffman Tree 的生成速度，但是也造成了 bug 的出现。原因是应该生成 Huffman Node 指针为堆中元素的最小堆，而不是以 Huffman Node 节点为堆中元素的最小堆，否则会导致 Huffman Tree 中某些节点的孩子指针指向自己，造成矛盾。
- 2、 在压缩时获取每个有效节点的 Huffman Code 也造成了一些问题，最后采用了递归法前序遍历每一个节点，并且借当前节点的父节点已生成的 Huffman Code 来生成编码，最终解决了问题。
- 3、 获取文件大小时，自带的 ftell 函数效果并不理想，因为编译器把 long 型看作和 int 型一样的 4 位，最后重写了 fsize 函数。
- 4、 在从文件读取一个字节的字符和写入一个字节的字符时，我一开始采用了 fputc 和 fgetc 函数，但他们获得的是有符号的 char 型数据，导致压缩和解压缩时存在溢出和损失，解压后文件改变。最后将所有用到节点对应字符的地方都换成了 unsigned char，方才解决了问题。
- 5、 将十进制 unsigned char 转化为二进制字符串的函数一开始没有记得在字符串前部补 0 到八位，导致数据的丢失。

测试用例评估：

- txt

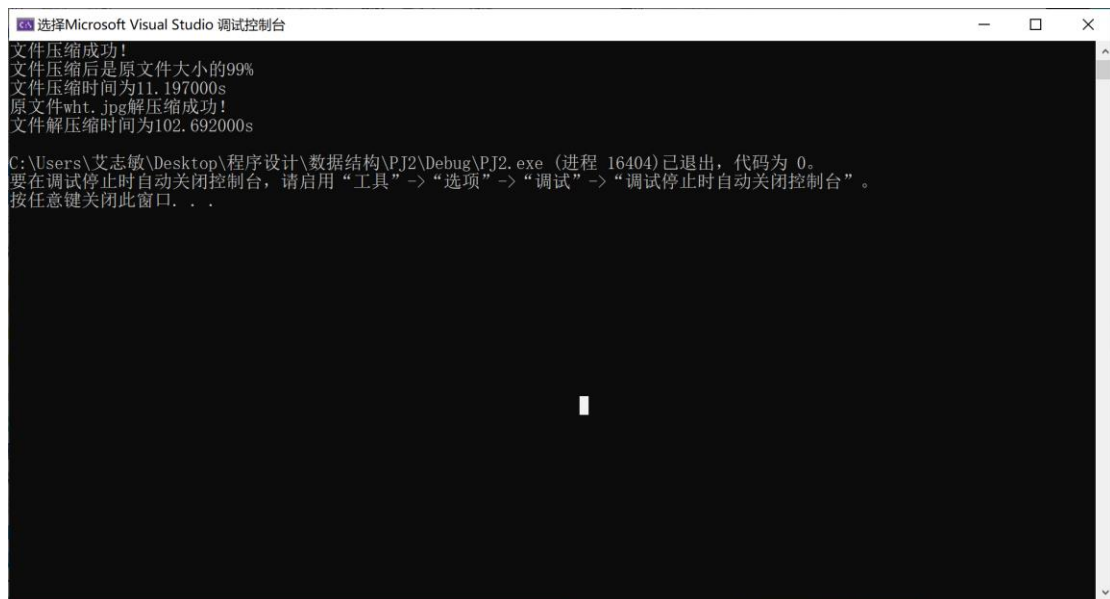
essay.txt 原文件大小 409 字节



由于压缩文件中用了一定空间来存储各个节点的出现频次, 所以对于较小的文件压缩效率偏低

- jpg

picture.jpg 原文件大小 1.77 MB



```
选择Microsoft Visual Studio 调试控制台
文件压缩成功!
文件压缩后是原文件大小的99%
文件压缩时间为11.197000s
原文件wht.jpg解压缩成功!
文件解压缩时间为102.692000s

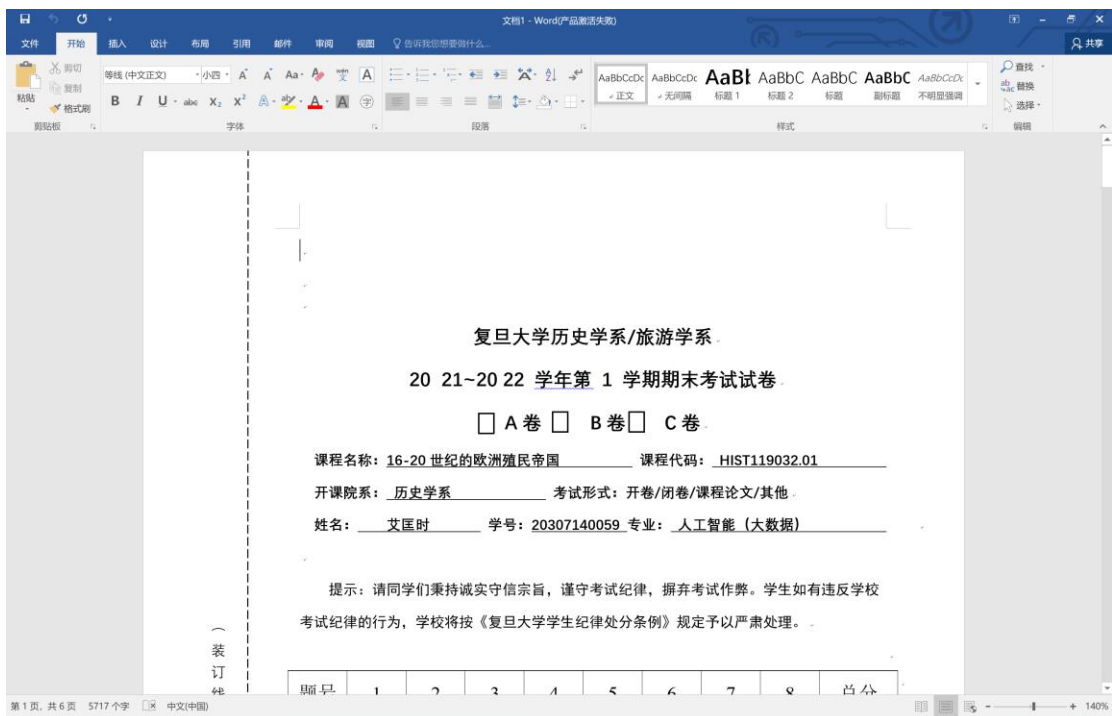
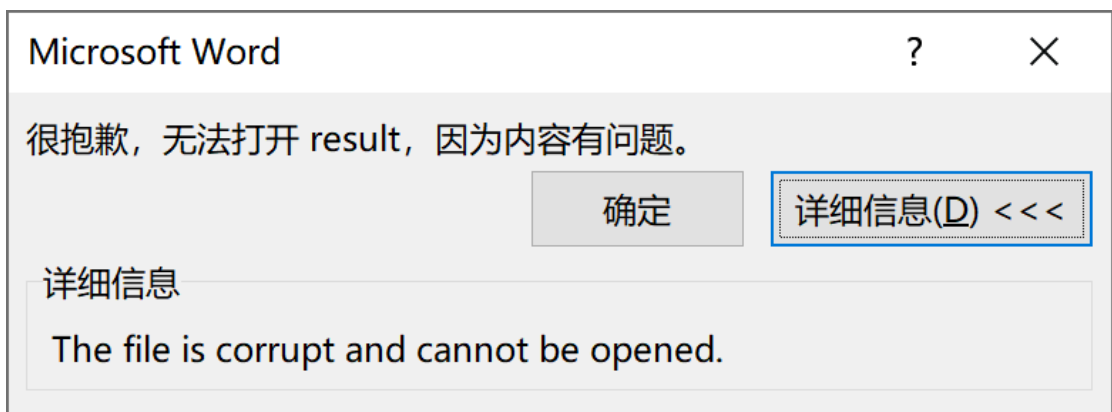
C:\Users\艾志敏\Desktop\程序设计\数据结构\PJ2\Debug\PJ2.exe (进程 16404) 已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

可以发现 jpg 型的文件解压缩时间较长, 通过增加一个缓冲区, 字符积累到一定数量以后再做输出应该会有所改观

jpg 格式压缩后大小几乎没有变化, 查资料得知这是因为 jpg 格式原本就用 Huffman 编码压缩过了, 因而看到原文件、压缩文件、解压缩文件都是相同的。

- doc

一米的宽度: 滇越铁路、云南与帝国殖民边缘.docx 原文件大小 38.5 KB 采用的是我这个学期七模课的期末论文

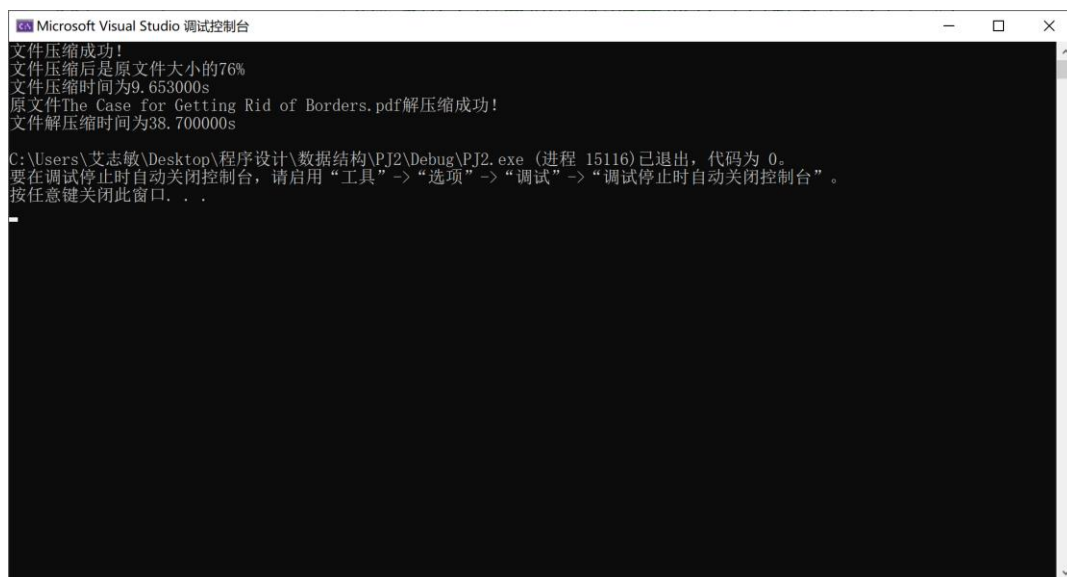


发现 word 在压缩以后反而还变大了, 而且一开始打开压缩后的文件, 会提

示文件损坏，但是修复以后可以打开，或许解压缩和压缩造成了几个字节的损失

- pdf

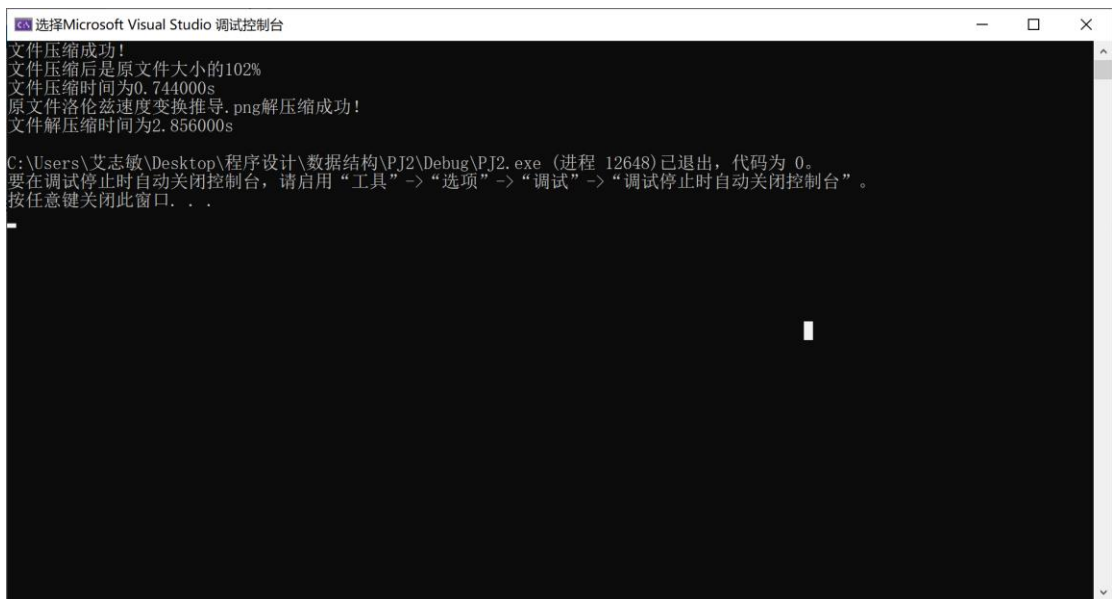
The Case for Getting Rid of Borders.pdf 原文件大小 1501KB



同样是解压缩时间太长，性能有待提升；但是相比 jpg 格式而言，pdf 是有进一步压缩的空间的

- png

原文件洛伦兹速度变换推导.png 原文件大小 65.7 KB



```
选择Microsoft Visual Studio 调试控制台
文件压缩成功!
文件压缩后是原文件大小的102%
文件压缩时间为0.744000s
原文件洛伦兹速度变换推导.png解压缩成功!
文件解压缩时间为2.856000s

C:\Users\艾志敏\Desktop\程序设计\数据结构\PJ2\Debug\PJ2.exe (进程 12648) 已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...
```

与 jpg 相似, png 也是本来就有过压缩的文件类型

- 综上其实发现 Huffman 编码文件压缩整体速度还是偏慢, 写入速度低下可以用缓冲区的方法改进, 但是生成 Huffman Tree 的时间复杂度就是由算法本身决定的了, 已经采用最小堆进行优化。了解到 jpg 文件已经用 Huffman Code 优化过, 故压缩效率不是很理想, 但对于大多数类型的文件, 在 Huffman 算法下都是有压缩空间的。Huffman 算法优点也是很明显的, 压缩后再解压, 文件的质量不会改变。