# 实习一 BMP灰度图像压缩

## 一、实习目的与要求

**【问题描述】**

灰度图像的像素值范围在[0，255]之间，如果采用一个像素一个字节的存储方式，势必会造成空间的浪费。如果采用一定的无损压缩算法，可以大大提高减小文件大小，减少存储空间。本课题要求针对提供的256色（8位）位图数据，采用教材上第15章动态规划中图像压缩算法（图像分段合并的思想），设计一个类，实现灰度位图数据的压缩和解压过程。

**【基本要求】**

一个完整的灰度图像类应具有以下功能：

（1）对8位位图数据的读功能，提供ReadBitmap方法。

ReadBitmap方法有一个参数为输入位图文件名(\*.bmp)，它能解析8位位图文件格式，获取位图BITMAPINFOHEADER信息和每个像素的数据信息，放入内存中。

（2）对8位位图数据的写功能，提供WriteBitmap方法。

WriteBitmap方法有一个参数为输出位图文件名(\*.bmp)，它能将内存中的位图文件信息，按照位图格式，写到位图文件中保存。

（3）灰度图像压缩功能，提供Compress方法。

Compress方法有一个参数为输出压缩文件名(\*.img) ，它能将已经装入到内存中的8位位图信息，进行压缩，形成段标题和以变长格式存储的像素的二进制串，写入到文件中（注意：Img文件格式自行定义）。

（4）灰度图像解压功能，提供UnCompress方法。

UnCompress方法有一个参数为输入压缩文件名(\*.img)，它能解析Img文件格式，将其在内存中解压缩为8位位图信息，以便输出为位图文件。

（5）以上是该灰度图像类基本的四个方法，在实现时可根据需要扩充其他方法。在设计时，要使用面向对象的思想，考虑各个成员的访问权限。

**【提高要求】**

（1）基于Windows对话框界面，可选择输入/输出文件名，有压缩进度条显示。

（2）采用不同的数据集，比较其压缩比，采用最有效的压缩方式。

**【测试数据】**

数字化.bmp，636\*455\*8

纹理.bmp， 512\*512\*8

**【测试用例】**

类的测试用例如下：

CCompressImage Test;

Test. ReadBitmap(“数字化.bmp”); 读原始位图

Test. Compress(“Out.img”); 压缩

Test. UnCompress(“Out.img”); 解压

Test. WriteBitmap(“Out.bmp”); 还原位图信息

测试结果：

可以使用ACDSee打开两幅位图，比较其属性信息及文件大小，验证你所实现的灰度图像类是否做到了无损压缩。

**【实现提示】**

有关8位的位图格式可以参考MSDN中BITMAPINFOHEADER结构的说明文档，注意其中biBitCount=8的说明。

#### 二、分析与设计

1. **问题分析与类设计**

**问题分析：**

**给定一个bmp图片文件，要求按照规定的动态规划算法进行压缩，并且能够实现解压操作。在做好底层算法后还要对界面进行处理。**

**类设计：**

**压缩类：**

**(1)设计一个压缩类。主要包含读文件，压缩，解压三个成员函数。**

**(2)读文件：根据所给路径，读出bmp文件的三个文件头，之后开始按照蛇形读出像素值。并将这些值存入成员函数中**

**(3)压缩：1.先做预处理，根据上一步得到的像素值，来计算这些像素的有效长度，存入到数组中。**

**2.进行主要的算法，动态规划进行合并，最终得到一个存储分割点的数组。**

**3.进行压缩，首先将之前读取的三个文件头写入文件。之后根据每一段的首尾位置，先后存储，段数(8位)，段长(3位)，这一段的所有像素(段长个字)。**

**(3)解压: 1.从压缩文件中读取三个文件头，并写入解压文件。并且得到像素的个数。**

**2.与压缩对应，先后按照 8字，3字，以及这段个数的像素，分别读出，存储到临时的数组中。**

**当数组写满时，将其按照蛇形写入解压文件。当读出所有像素后，结束。**

**(4)成员变量：三个文件头的结构体。 存储像素的数组。**

**界面类：**

(1)主要的文本框：传递需要压缩或解压的文件路径。同时实现拖拽获取文件路径。

(2)设置选择按钮：通过可视化直接选择所需的压缩与解压的文件。

(3)按钮:通过判断文件类型进行压缩或解压。

1. **出现的问题及解决办法**

**(1)读文件时对于文件头数据读取错误。**

解决: 之前按照网上查找而定义的结构体的顺序可能出现错误，因此通过调用系统自带的三个bmp文件头直接进行读取。

**(2)对于处理有效长度时，应该注意长度为0-7，而在进行计算时长度为1-8，加一。**

**(3)动态规划时，对于存储分割点的数组，存储数据较多，因此会导致系统内存不足。**

**解决:注意到一段长度最多为256，因此可以只保留当前256个数组。将之后利用不到的数组及时进行清理。**

**(4)压缩并且解压后图片与原图片不相符。**

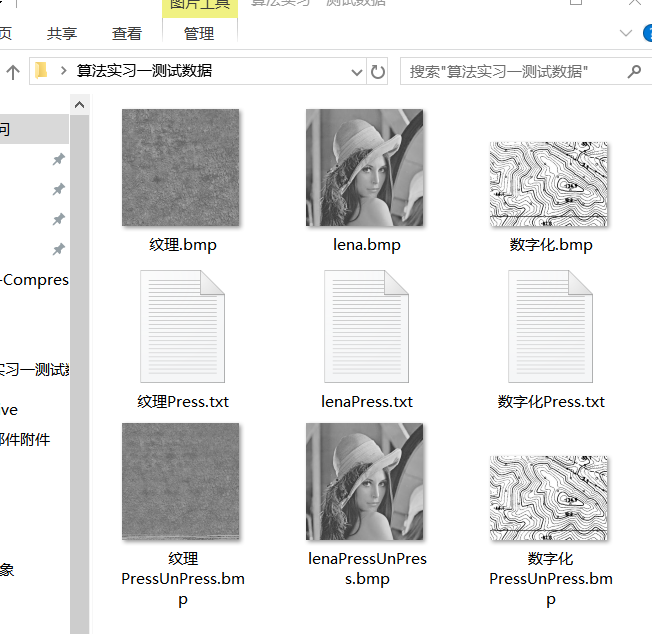
**解决:错误出现在压缩处，对于写入的数据，如果写入一次后，需要进行附零处理，否则之后进行的位操作会出现错误。**

**(5)需要注意如果一个数据左移或右移它本身个长度时，并没有任何改变。**

**(6)对于算法出现许多错误，导致处理的像素出现混乱。**

#### 【源代码】

#### 【运行结果】



## 四、实习小结

通过本次实习，对于动态规划算法有了一定的练习。在处理较大数据时可以看出，动态规划的效率还是较高的，当然这是一种空间换时间的做法，申请的空间不能过多，因此在处理问题时也需要对空间利用上也作出一定的优化。对于位运算与处理也能够进一步的掌握。当然由于之前做过Huffman压缩，因此对于位处理还是较为熟悉的。同时本次实习也了解了bmp文件的格式与预处理。

当然本次实习也是存在一些不足的。主要是在压缩的方面，时间消耗可能过大。对于申请的空间过大，进行动态删除可能不太合适，应该是只是申请一维的数组，通过不断维护数组最终得到结果。对于获取像素的有效位与比较某一段的最大长度时时间消耗过大，需要进一步改进。另外，对于写入文件来说，最好能够申请一个缓存空间，之后一次写入一大段，减少io操作。最后还是代码整体判断较多，需要减少判断，合并多个相同的情况，从而使代码效率得到进一步提升。