



实习一 动态规划算法及应用

【题目一】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)，它可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)，它可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】

lena.bmp, 512*512*8

【测试用例】

类的测试用例如下：

```
CCompressImage Test;
```

```
Test.ReadBitmap("lena.bmp");    读原始位图
```



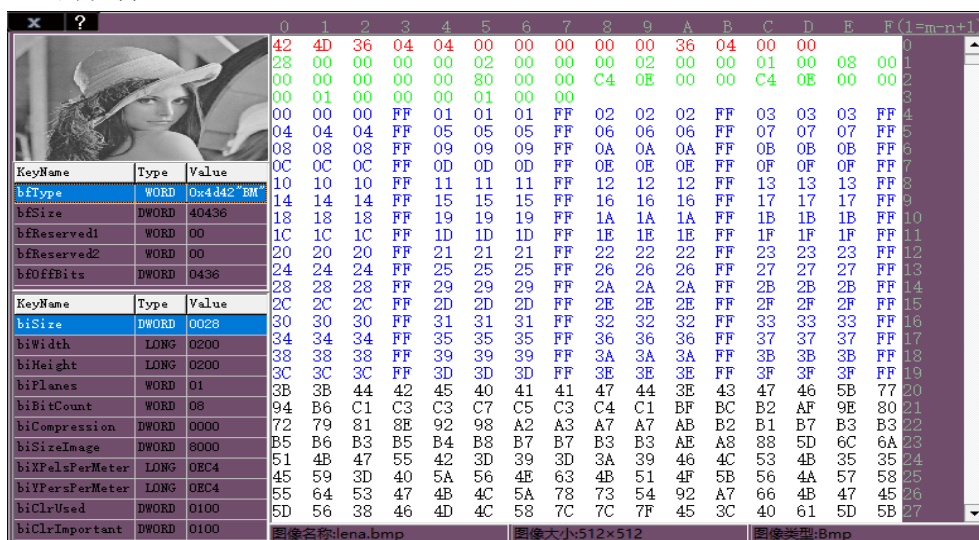
Test. Compress("Out.img"); 压缩
Test. UnCompress("Out.img"); 解压
Test. WriteBitmap("Out.bmp"); 还原位图信息

【测试结果】

可以使用 MD5 比较解压后的图与原图是否一样，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。



变量名	地址偏移	大小	作用
bfType	0000h	2 bytes	说明文件的类型，可取值为： • BM - Windows 3.1x, 95, NT, ... • RA - OS/2 Bitmap Array • CI - OS/2 Color Icon • CP - OS/2 Color Pointer • IC - OS/2 Icon • PT - OS/2 Pointer
bfSize	0002h	4 bytes	说明该位图文件的大小，用字节为单位
bfReserved1	0006h	2 bytes	保留，必须设置为0
bfReserved2	0008h	2 bytes	保留，必须设置为0
biOffBits	000Ah	4 bytes	说明从文件头开始到实际的图象数据之间的字节的偏移量。 这个参数是非常有用的，因为位图信息头和调色板的长度会根据不同情况而变化，所以我们可以用这个偏移值迅速的从文件中读取到位图数据。



变量名	地址偏移	大小	作用
biSize	0000h	4 bytes	BITMAPINFOHEADER结构所需要的字数。
biWidth	0012h	4 bytes	说明图像的宽度，以像素为单位。
biHeight	0016h	4 bytes	说明图像的高度，以像素为单位。 注：这个值除了用于描述图像的高度之外，它还有另一个用处，就是指明该图像是纵向的位图，还是正向的位图。 如果该值是一个正数，说明图像是纵向的，如果该值是一个负数，则说明图像是正向的。 大多数的BMP文件都是纵向的位图，也就是高度值是一个正数。
biPlanes	001Ah	2 bytes	为目标设备说明颜色平面数，其值将总是被设为1。
biBitCount	001Ch	2 bytes	说明比特数/像素，其值为1、4、8、16、24或32。
biCompression	001Eh	4 bytes	说明图像数据压缩的类型。取值范围： 0 BI_RGB 不压缩（最常用） 1 BI_RLE8 8比特游程编码（RLE），只用于8位位图 2 BI_RLE4 4比特游程编码（RLE），只用于4位位图 3 BI_BITFIELDS 比特域，用于16/32位位图 4 BI_JPEG JPEG 位图含JPEG图像（仅用于打印机） 5 BI_PNG PNG 位图含PNG图像（仅用于打印机）
biSizeImage	0022h	4 bytes	说明图像的大小，以字节为单位。当用BI_RGB格式时，可设置为0。
biXPelsPerMeter	0026h	4 bytes	说明水平分辨率，用像素/米表示，有符号整数。
biYPelsPerMeter	002Ah	4 bytes	说明垂直分辨率，用像素/米表示，有符号整数。
biClrUsed	002Eh	4 bytes	说明位图实际使用的调色表中的颜色索引数（设为0的话，则说明使用所有调色板项）。
biClrImportant	0032h	4 bytes	说明对图像显示有重要影响的颜色索引的数目。如果是0，表示不重要。

二、分析与设计

1.需求分析与类设计

1.1 需求分析

1.压缩：

- 1) 能通过对对话框选择要压缩的图片（只能选择 BMP 格式的图片）；
- 2) 压缩后的文件能以另一个格式存储在原图相同目录下；
- 3) 压缩过程中能在窗口上实时显示出当前状态以及压缩文件所在位置。
- 4) 进度条实时显示压缩状态。

2.解压：

- 1) 通过对对话框选择要解压的图片（只能选择特定的格式，这里是 CBMP）；
- 2) 解压后自动在原压缩文件同目录下生成一张 BMP 格式的图片。
- 3) 窗口上实时显示当前解压状态及解压出来的图片所在位置。
- 4) 进度条实时显示解压状态。

1.2 类设计

1.定义一个 CCompressImage 类来执行解压及压缩的相关操作。基本内容有：

- 1) 压缩：ReadBitmap() 读取位图文件数据；Compress() 对读取到的数据进行压缩处理并写入到压缩文件。
- 2) 解压：UnCompress() 读取压缩文件的数据并对数据进行解压处理；WriteBitmap() 将数据写入到位图。
- 3) 其他成员变量：位图包括有文件头、信息头、调色板、像素点等数据，在设计 CCompressImage 的时候，需要添加上这些成员变量。文件头、信息头、调色板需要先定义相关的结构体，像素点数据就是要数组存储起来即可。除此之外，为了能在界面上显示相关的信息，还要加上文件大小



（压缩前/压缩后/解压前/解压后），文件路径（压缩后/解压后）等成员变量。

4) 其他辅助函数: 获取路径（压缩后/解压后），获取文件大小（压缩前/压缩后/解压前/解压后）。

length (): 获取一个数用 2 进制表示时最少需要多少位。用于压缩时对像素点数据进行压缩处理。

reset(): 重置类中的像素点数据，防止对即将要读取的数据造成影响。用于压缩前或者解压前。

具体设计如下:

名称	属性	描述
path	成员变量(私有)	要压缩的文件或者要解压的文件的的路径
data	成员变量(私有)	二维数组，存储像素点信息
dataline	成员变量(私有)	一维数组，将 data 的数据蛇形转储到 dataline
b	成员变量(私有)	一维数组，表示 dataline 中每个数据的位数
l	成员变量(私有)	一维数组，分段信息，表示每个段内的像素个数
s	成员变量(私有)	一维数组，s[i]表示像素序列 {p1, p2... pi} 的最优分段所需的总存储位数
head	成员变量(私有)	BitHead 类型，存储位图的文件头
Inf	成员变量(私有)	BitInf 类型，存储位图的信息头
tag	成员变量(私有)	tagRGB*类型，一维数组，表示位图的调色板
tagnum	成员变量(私有)	Int，表示调色板的个数
beforecompressedsize	成员变量(私有)	Int，表示压缩前文件大小
compressedsize	成员变量(私有)	Int，表示压缩后文件大小
beforeuncompressedsize	成员变量(私有)	Int，表示解压前文件大小
uncompressedsize	成员变量(私有)	Int，表示解压后文件大小
cpath	成员变量(私有)	CString，表示压缩后文件路径
upath	成员变量(私有)	CString，表示解压后文件路径
CCompress()	成员函数(公有)	构造函数
~CCompress()	成员函数(公有)	析构函数
reset	成员函数(公有)	重置 data 和 dataline
setpath	成员函数(公有)	设置路径
ReadBitmap	成员函数(公有)	读取位图数据
WriteBitmap	成员函数(公有)	写位图数据
Compress	成员函数(公有)	对数据进行压缩处理
UnCompress	成员函数(公有)	对数据进行解压处理
length	成员函数(公有)	计算一个数用二进制表示最少需要多少位
getsize	成员函数(公有)	获取压缩前文件大小
getcompresssize	成员函数(公有)	获取压缩后文件大小
getbefore	成员函数(公有)	获取解压前大小
getafter	成员函数(公有)	获取解压后大小
getcpath	成员函数(公有)	获取压缩后文件路径
getupath	成员函数(公有)	获取解压后文件路径



2. 算法设计与分析

压缩算法:

对像素点数据进行分段, 每一段像素个数不超过 256 个, 段内每一个像素都可以用若干位的二进制数来表示 (1 到 8 位)。最后存储时, 像素个数都减一, 就可以用 8 位的二进制数来表示, 段内像素的长度也减一, 就可以用 3 位的二进制数来表示。各像素点的数据用该段像素点的长度大小的二进制数来表示, 就可以少存储一些没用的 ‘0’。从而达到压缩的效果。因此, 主要的内容就在于寻找最优分段。寻找最优分段我是用了书上的算法。但书上的算法有点错误而且递归时会内存不足, 因此对书上的算法改了一部分。具体如下:

//压缩过程

```
int t = Inf.biHeight * Inf.biWidth;
s = new int[t+1]; //前n各像素点最优分段存储所需要的总位数
b = new int[t+1]; //每个像素用二进制表示时最少需要多少位
l = new int[t+1]; //像素点所在段每个像素需要的最大位数
s[0] = 0;
b[0] = 0;
l[0] = 0;
for (int i = 1; i <= t; i++) {
    b[i] = length(dataline[i - 1]); //计算像素本身需要的最大位数
    int bmax = b[i]; //记录下当前像素点位数
    s[i] = s[i - 1] + bmax; //加上b的位数
    l[i] = 1; //段长为1(初始化)
    for (int j = 2; j <= i && j <= Lmax; j++) { //向前查找是否存在更优的分段, 长度不超过Lmax(256)
        if (bmax < b[i - j + 1]) { //当前像素点位数小于前面分段的像素点位数, 更新bmax
            bmax = b[i - j + 1];
        }
        if (s[i] > s[i - j] + j * bmax) {
            //当前i个像素点分段的大小大于前i-j个像素点分段大小加上j*bmax
            //说明存在更优的分段
            s[i] = s[i - j] + j * bmax; //更新大小
            l[i] = j; //更新段长数据
        }
    }
    s[i] += header; //加上“头”的大小(11位)
}
//构造最优解, 需要由后往前算, 所以使用栈
stack<int> L; //像素段含有的像素数 (0-255)
int index = Inf.biWidth*Inf.biHeight;
while (index > 0) {
    L.push(l[index]-1); //从后向前存入段长信息
    index = index - l[index];
}
```

//文件名

利用栈先进后出的特性可以从后往前存入段长信息, 书上使用的递归方法容易造成程序崩溃, 使用这种方法可以避免程序崩溃的情况。

段内元素需要的位数我是在写入数据时再来计算获取, 所以这里没有体现。

获取段内元素需要的位数: 知道段长后, 查找段内元素所需位数的最大值, 即为段内元素需要的位数。

写像素点数据到压缩文件: 设置一个缓冲区 buffer 存储要写入到压缩文件的数据, 当要完 buffer



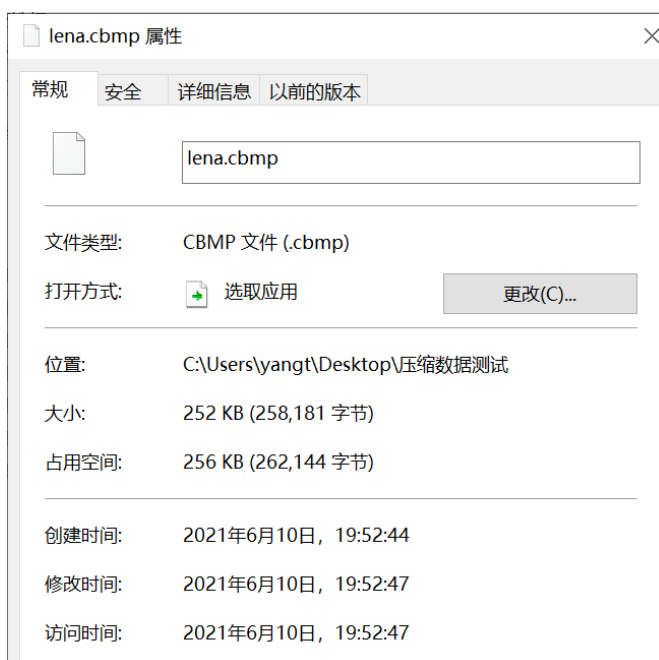
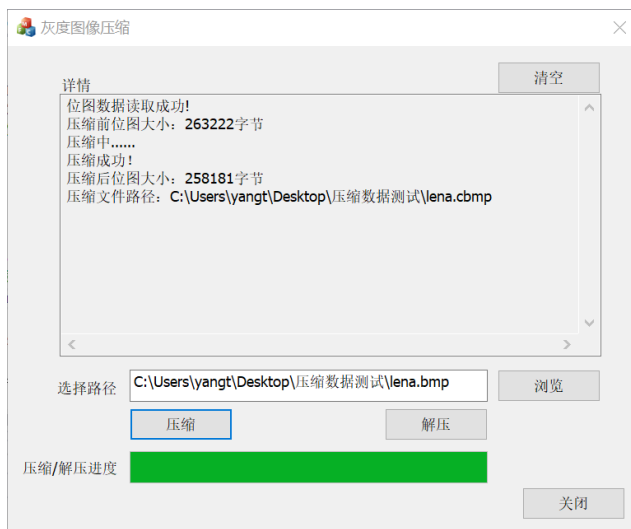
中加入一个 n 位数据 num 时就先将 $buffer$ 向前移 n 位，再与 num 取位于操作将 num 加到 $buffer$ 的后 n 位。当 $buffer$ 长度大于 8 位时，就将 $buffer$ 前 8 位存入(使用位移操作)，然后再将 $buffer$ 的前 8 位置 0，并更新 $buffer$ 的长度。具体代码见源码 CCompressImage.cpp 下的 Compress 函数相关内容。

从压缩文件读取数据并进行解压操作于上述类似，篇幅有限，就不赘述。详见源码 CCompressImage.cpp 下的 Uncompress 函数。

3、功能测试与改进

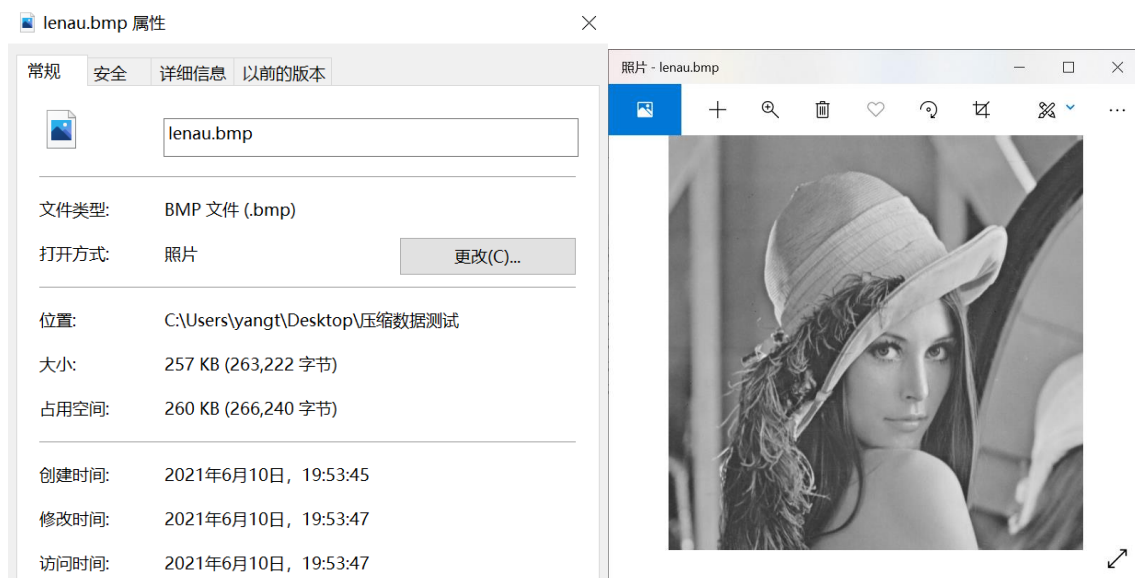
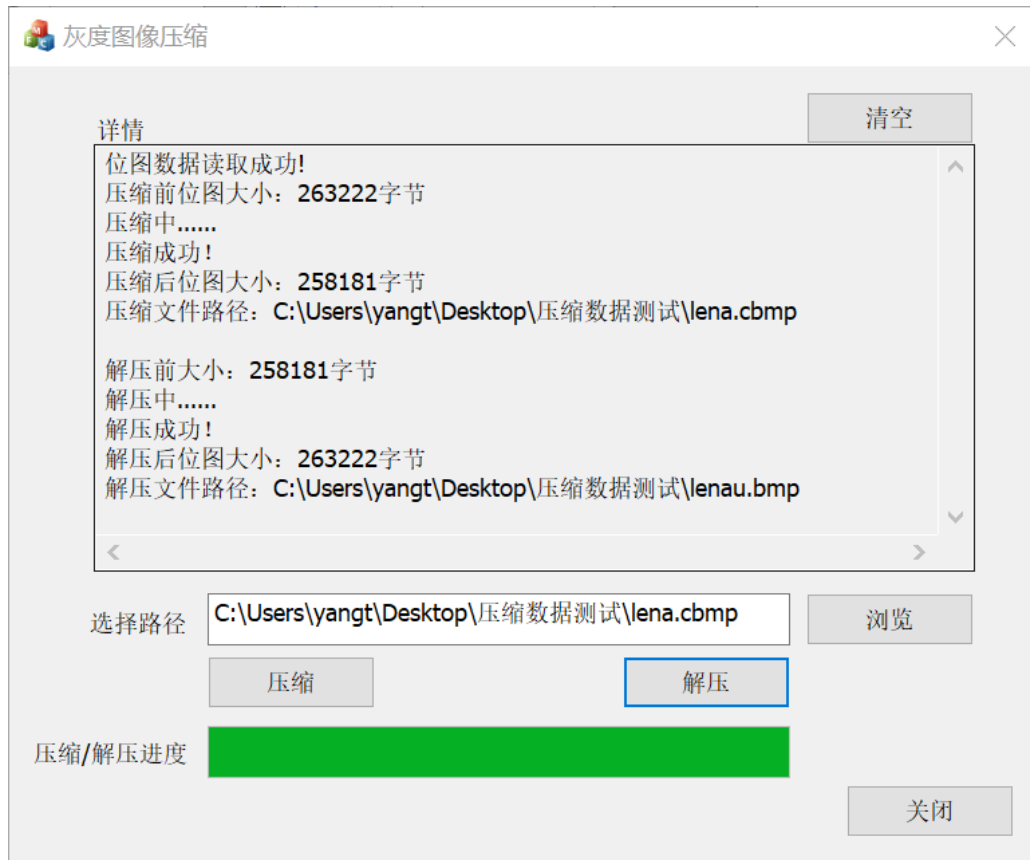
测试：

压缩 lena.bmp:




















解压 lean.bmp:





其他测试用例：

 纹理u.bmp	2021/6/10 19:58	BMP 文件	257 KB
 数字化u.bmp	2021/6/10 19:58	BMP 文件	284 KB
 lenalightu.bmp	2021/6/10 19:57	BMP 文件	258 KB
 lenadarku.bmp	2021/6/10 19:57	BMP 文件	258 KB
 纹理.cbmp	2021/6/10 19:57	CBMP 文件	224 KB
 数字化.cbmp	2021/6/10 19:57	CBMP 文件	278 KB
 lenalight.cbmp	2021/6/10 19:57	CBMP 文件	259 KB
 lenadark.cbmp	2021/6/10 19:56	CBMP 文件	164 KB
 lenau.bmp	2021/6/10 19:53	BMP 文件	258 KB
 lena.cbmp	2021/6/10 19:52	CBMP 文件	253 KB
 lenadark.bmp	2021/6/3 18:10	BMP 文件	258 KB
 lenalight.bmp	2021/6/3 18:10	BMP 文件	258 KB
 数字化.bmp	2019/3/18 17:12	BMP 文件	284 KB
 lena.bmp	2018/3/26 22:55	BMP 文件	258 KB
 纹理.bmp	2002/5/29 19:30	BMP 文件	257 KB

lena.bmp, lenalight.bmp, lenadark.bmp, 纹理.bmp, 数字化.bmp 是原测试文件。

lena.cbmp, lenalight.cbmp, lenadark.cbmp, 纹理.cbmp, 数字化.cbmp 是压缩后的文件。

lenau.bmp, lenalightu.bmp, lenadarku.bmp, 纹理 u.bmp, 数字化 u.bmp 是解压后的文件。

改进：

1. 进度条与压缩/解压过程的匹配。我这里由于压缩/解压的相关函数都封装到类中，虽然使用了多线程，但也只做到了窗口上信息的实时显示，试了几种方法还是无法使进度条与压缩/解压过程匹配。
2. 采用不同的数据集比较压缩比，选择最高效的压缩方式。

三、实习小结

这次实习使用到了动态规划的算法，让我体会到了动态规划的好用之处。同时，这次实习也涉及到文件读写以及数字位移、位或、位与等操作，通过这次的实习对这些内容有了进一步的理解。对于这次实习的结果我觉得还存在可以优化的地方，比如进度条与过程的匹配，以及耗费时间的显示等等。

成绩评定：

教师签名：

批改日期：