

## 一、代码设计思路

整体工作与冈萨雷斯第八章例 8.17 类似，先介绍编码器工作过程：

1. 对灰度图像做-128 灰度级移位。
2. 将图像分割为 8\*8 大小的分块，对每一分块做如下处理：
  - a) 对该分块做二维 DCT，之后进行量化。量化矩阵采用教材图 6.13(a)如下：  

```
Quant_matrix=[16 11 10 16 24 40 51 61
               12 12 14 19 26 58 60 55
               14 13 16 24 40 57 69 56
               14 17 22 29 51 87 80 62
               18 22 37 56 68 109 103 77
               24 35 55 64 81 104 113 92
               49 64 78 87 103 121 120 101
               72 92 95 98 112 100 103 99]
```
  - 考虑到可以设置不同的压缩系数  $g$ ，将不同的压缩系数与量化矩阵相乘可以得到对应的新量化矩阵，起到缩小矩阵元素大小，增大压缩比的目的。  
将 DCT 后的 8\*8 矩阵与量化矩阵对位相除即可得到量化后的分块。  
该部分操作由函数 DCT\_Quant()完成。
  - b) 将分块根据教材图 6.12 做之字形扫描，由 8\*8 矩阵变为 1\*64 行向量。该部分操作由函数 Zigzag()完成，采用直接查表方式实现。
  - c) 每个分块的第一个元素为直流分量，应由 DPCM 方式进行编码。由函数 codeDC()完成。
    - i. 根据两相邻分块的 DC 分量差的 Amplitude 查找对应的 Size，由 find\_size\_DC()函数完成。
    - ii. 根据所得(size, amplitude)进行编码，由 code\_size\_DC()、code\_amp()函数完成，采用冈萨雷斯书附录表 A.4 实现。
  - d) 处理 AC 分量。由函数 codeAC()完成。
    - i. 找到行程 Runlength、Amplitude、Size，由 find\_size\_AC()、code\_size\_AC()、code\_amp()完成，采用冈萨雷斯书附录表 A.5 实现。
  - e) 将所得 DC 码及 AC 码拼在一起，并加在已编码部分的后面。更新变量 last\_DC 为当前分块的直流分量。
3. 扫描整个图像后得到一个码流 jpeg\_code。将图像的长宽信息转为 10 位二进制数，压缩系数转为 5 位二进制数，附在码流开头。不足 10/5 位的部分在二进制数前补零。意味着该情况下最大能编码 1023\*1023 的图像文件，压缩比不超过 15。可通过改变编码位数进行调整。
4. 扫描整个图像后得到一个码流，根据码流长度计算压缩比与压缩效率。
5. 将所得码流以字符串形式写入 data.dat 文件。

解码器工作思路类似，也主要以查表方式实现：

1. 读取 data.dat 文件，得到码流。
2. 码流前 5 位为压缩系数  $g$ ，转为 10 进制数保存。第 6-15 位为图像长度，第 16-25 位为图像宽度，同样转为 10 进制数保存。
3. 根据所得的长宽信息，按顺序写入 8\*8 分块：
  - a) 解码 DC 分量，由 decodeDC()函数完成。
  - b) 解码剩下 63 个 AC 分量，由 decodeAC()函数完成。
  - c) 得到 1\*64 向量 zigzag\_stream，使用 Izigzag()函数转为 8\*8 矩阵，利用

Iquant\_IDCT()函数做反量化与反 DCT，写入结果图像 jpeg\_result。

4. 将结果图像写入文件。

最后使用一个测试脚本进行绘图及 PSNR 计算。

## 二、实验结果

将不同压缩系数下的压缩比、压缩效率以及 PSNR 列表如下。

压缩系数	压缩比	压缩效率/(bit/pixel)	PSNR/dB
1	7.25	1.10	34.00
2	11.18	0.72	31.06
4	17.69	0.45	28.77
8	28.28	0.28	26.51

显然，随着压缩系数的增大，压缩比也增大，相应的压缩效率提高，PSNR 下降。这一结果体现了 JPEG 是一种有损压缩算法，信息损失主要源自量化操作，重构图像的质量与压缩比之间存在矛盾。



## 三、学习体会

尽管 JPEG 的编解码看起来是一项庞杂的工程，但是按照 PPT 及课本例子，可以将其拆解为一个顺序执行的流程，从顶层分解为一项项具体工作，并采用函数对每一项具体工作进行实现。

但是本次作业中存在大量的查表操作，对运行速度造成了很大的影响。自己也需要进一步学习一些 MATLAB 编程技巧以提高运算效率。查阅资料后发现实际使用的 JPEG 压缩算法更为复杂，其在解码过程中采用哈夫曼树可大大提高计算速度。有机会可以尝试进一步学习，实现更为复杂的编解码算法。