

程式執行硬體環境：



$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}.$$




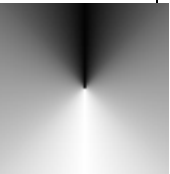


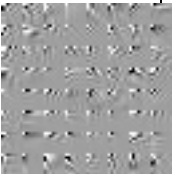
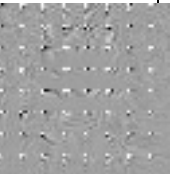
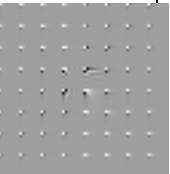




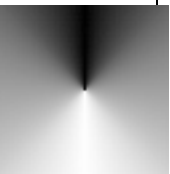

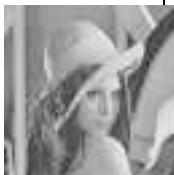




Quantization Table

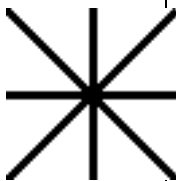




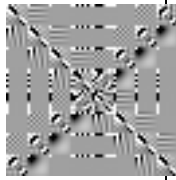
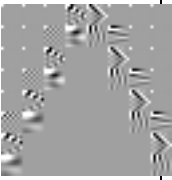

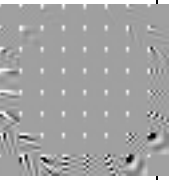

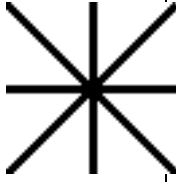




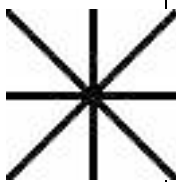



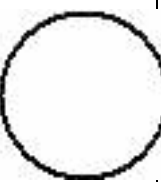
## 1. 程式說明

程式編譯完使用時將 `argv[1]`環境變數帶入要輸入的 `raw` 檔案，`argv[2]` & `argv[3]` & `argv[4]`分別是是檔案的長寬和 `Blcok` 大小，`argv[5]`則是選擇要跑 Q1-Q4 哪一題，程式就會自動輸出相應的檔案。

本程式中因為其編解碼會有同序列進行共用 `memory` 的情況，因此在此直接複寫前步驟的檔案，可以藉由 `argv[5]`來選擇要 Q1~Q4 哪一組的結果並且輸出，可以得到作業文件中所有的檔案。

## 2. 比較圖表

Name	Lena64	Pepper64	Baboon64	Gra1	Gra2
Origin Image					
1D-DCT					
1D-IDCT					
PSNR	MSE = 0	MSE = 0	MSE = 0	MSE = 0	MSE = 0
Quantization					
PSNR	29.5985	29.379	30.1116	42.5348	46.5547
Origin Size	4096	4096	4096	4096	4096
Huffman Size	2418	2616	1975	807	560
Compression Rate	1.69387	1.56553	2.07392	5.07795	7.31429
Arithmetic Size	2558	2752	2115	926	671
Compression Rate	1.60156	1.48844	1.93733	4.42572	6.1066

Name	wildcard	triangle	Circle1	Circle2	Circle3
Origin Image					
1D-DCT					
1D-IDCT					
PSNR	MSE = 0	MSE = 0	MSE = 0	MSE = 0	MSE = 0
Quantization					
PSNR	27.0902	31.5827	28.3853	30.2904	28.8672
Origin Size	4096	4096	4096	4096	4096
Huffman Size	3330	1674	2105	1625	2460
Compression Rate	1.23003	2.44683	1.94584	2.5212	1.66513
Arithmetic Size	3464	1811	2243	1764	2600
Compression Rate	1.18219	2.26205	1.82602	2.32265	1.57538

### 3. Q1 - Block 8x8 DCT IDCT

由於是 HW2 的 Q5 ( Bonus )，因此直接沿用其中我在這使用的是 Fast 版本，Block Size 由環境變數 `argv[5]` 設定，在此為 8，由於本次 Quantization & InverseQuantization 沒有正規化因此直接在 IDCT 加入  $>255$  &  $<0$  的處理。

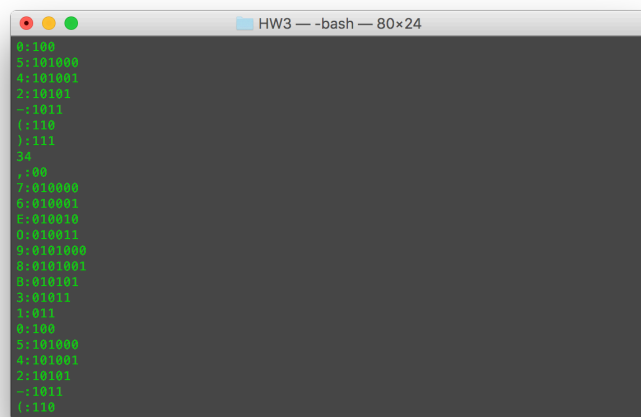
參考 2 的圖表可以看到 1D-IDCT 列的圖片與 Origin Image 列圖片在沒有做 Quantization 下是相同的，因此其  $MSE = 0$  則 PSNR 為原始圖片。

### 4. Q2~Q4 – Quantization RLC RLD Huffman Arithmetic

在 Q2 之後的每一個步驟都加入 Quantization，因為整個 Processing Stream 只有這一步影響 PSNR，因此 2 圖表只有列出一個 Quantization 後解回的 PSNR，代表 Run Length & Huffman & Arithmetic，解回後的 PSNR，可以發現對於複雜圖片此 Quantization Table 相對簡單圖片破壞較多。

Q2 RLC & RLD 輸出的.rlc 格式為（零的數量,非零值），而每一行代表一個 Block 利用字串 EOB 做為結尾。

Q3 Huffman 輸出的.huf 格式為，.rlc 的輸入，第一行是（符號 1/機率 1/ ...符號 n/機率 n/），接著就是 Huffman Tree Code（類似下圖分號後編碼），

A terminal window titled 'HW3 - bash - 80x24' displays a list of Huffman tree codes. Each line consists of a symbol followed by a binary code separated by a semicolon. The symbols are integers from 0 to 15, and the codes are binary strings of varying lengths. The window has a dark background and standard macOS window controls at the top.

```
0:100
3:101000
4:101001
2:10101
~:1011
(-:110
):111
35
~:00
7:010000
6:010001
5:010010
0:010011
9:0101000
8:0101001
0:010101
3:01011
1:011
0:100
5:101000
4:101001
2:10101
~:1011
(-:110
```

預設以 ASCII 的方式輸出方便觀察，因此計算壓縮量則是要除以 8 表示 byte 計算，註解部分則是以 8bit 的方式以 binary 輸出，解碼則是讀取第一行得到機率表格後再重新建成 Huffman Tree，然後再去搜尋讀取的編碼的符號。

Q4 Arithmetic 輸出的.ari 格式為，.rlc 的輸入，使

用 `ac.cpp` 提供的 `function`，先初始化後再將 `.rlc` 的資料以一個一個字元讀入並且編碼同時紀錄讀入數量，最終輸出 `binary` 形式的檔案，同時得到 `ari_total_byte()` 回傳值 `bit`，除以 8 得到 `byte` 數量，解碼時則是將步驟反過來即可。

## 5. 結論

首先可以觀察到用 `Block DCT` 其 `Block` 越小（相較於 `HW2 64 x 64`）似乎期能量分佈圖像會有類似於原圖的輪廓。

其次可以觀察到 `Gary1 & Gary2` 圖檔的 `Huffman & Arithmetic` 壓縮比高於其他圖檔，原因在於其圖像頻率分布很線性因此其 `RunLength` 大多數區塊可以很有效的被縮短，所以在 `VLC` 時就可以獲得較大的壓縮比。

最後因為 `Huffman & Arithmetic` 編碼原理類似，因此其壓縮比率是差不多的，而 `Huffman` 似乎比

Arithmetic 好一點。