JPEG 壓縮

Github 網址(內有程式碼):

https://github.com/ChristopherChen070535/JPEG-Compression

撰寫人:

國立台北大學通訊工程學系三年級陳品鈞

- **專題簡介**
  - encoder.c ： 此程式將輸入的圖片檔在使用方法 0 時，把圖片讀入，並讀取輸入圖片之 RGB 數據；在使用方法 1 時，將讀入的 RGB 數據進行 2 維餘弦轉換並進行量化。
  - decoder.c ： 此程式主要是利用將 encoder 的做法反著做的方式以及 linux 的指令去比對輸出圖片和原圖去印證在 encoder 的處理是否正確。
  - Makefile ： 用來將上述程式一鍵執行以得到結果，demo0 表示執行 encoder0, 以及 decoder0，並利用 cmp 指令去比對輸出圖檔是否和原圖相同，demo1 表示執行 encoder1, 以及 decoder1，並利用 cmp 指令去比對輸出圖檔再加上誤差值前後和原圖的差異

- **執行成果錄影**

  https://youtu.be/9NX6zMFOj5w

- **程式原理:**

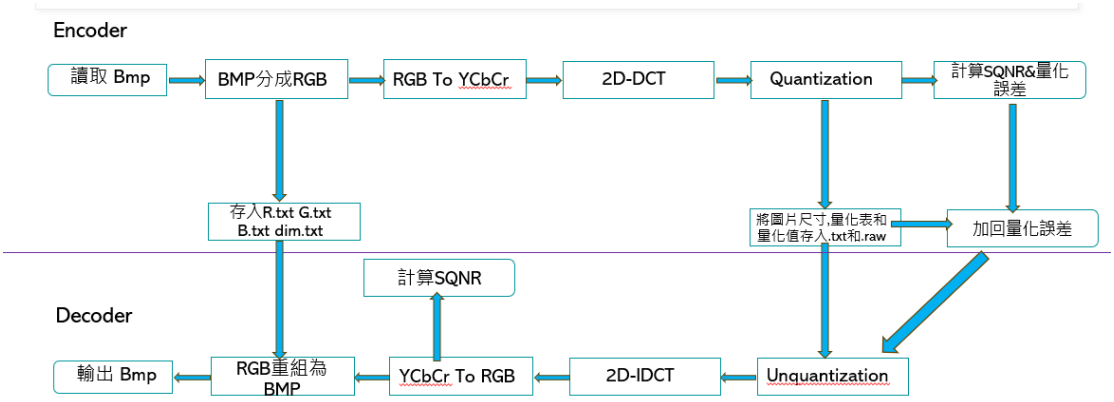

圖 1:此作品之 block diagram

根據上圖 1 之流程此程式我主要將步驟分為讀取 BMP 中的 RGB 數據、RGB to YcbCr、2D-DCT、Quantization，並且會計算其誤差值以及 SQNR

1. RGB to YcbCr: RGB 轉換到 YcbCr 主要是將 RGB 色彩空間中的每個像素的顏色信息轉換為 Y（亮度）、Cb（藍色色度）和 Cr（紅色色度）三個分量。這個轉換是基於人眼對於亮度和色彩的感知方式不同的原理，因此對圖像進行這種轉換後，可以更好地保留人眼對圖像的感知品質，轉換公式如下圖 2。

```
// 根據YCbCr的定義，計算Y（亮度）、Cb（藍色色度分量）和Cr（紅色色度分量
ycbcrData[i][j].Y = 0.299 * R + 0.587 * G + 0.114 * B;
ycbcrData[i][j].Cb = 128 - 0.168736 * R - 0.331264 * G + 0.5 * B;
ycbcrData[i][j].Cr = 128 + 0.5 * R - 0.418688 * G - 0.081312 * B;
```

圖 2: RGB to YcbCr 之公式

2. 2D-DCT: 二維離散餘弦變換（2D-DCT）用於將二維數據轉換為一組不同頻率的餘弦波。在 JPEG 壓縮中，2D-DCT 應用於圖像壓縮，以將圖像轉換為一組係數，以便更有效地表示圖像。以下是 2D-DCT 的基本原理：
   (1)區塊劃分：首先，圖像被分成大小為 N×N 的小區塊（在此專題中 N 為 8），每個小區塊都會獨立進行 2D-DCT 轉換。
   (2)水平和垂直變換：對於每個小區塊，首先對其行（水平方向）進行一維離散餘弦變換（1D-DCT），然後對其列（垂直方向）進行 1D-DCT。這樣就得到了小區塊的 2D-DCT 係數。
   (3)2D-DCT 的計算：根據下圖 3 之公式對每個小區塊的行和列都進行 1D-DCT 的計算。即對圖像中的每個小區塊應用兩次 1D-DCT，將圖像之時域訊號轉為頻域。

$$\begin{cases} F[u,v] = \sqrt{\frac{4}{HW}} \sum_{r=0}^{H-1} \sum_{c=0}^{W-1} \beta[u]\beta[v] f[r,c] cos(\frac{\pi u(2r+1)}{2H}) cos(\frac{\pi v(2c+1)}{2W}) \\ f[r,c] = \sqrt{\frac{4}{HW}} \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} \beta[u]\beta[v] F[u,v] cos(\frac{\pi u(2r+1)}{2H}) cos(\frac{\pi v(2c+1)}{2W}) \end{cases}$$

圖 3:DCT 運算之公式

   (4)係數獲取：最後，得到每個小區塊的 2D-DCT 係數，這些係數表示了圖像中不同頻率的餘弦波成分的貢獻程度。

3. 量化: 量化是 JPEG 壓縮中的一個關鍵步驟，它用於減少圖像的數據量，進一步壓縮圖像而不會對人眼的視覺感知品質產生過大的影響。量化通常應用於 2D-DCT 的係數，即將 2D-DCT 轉換後得到的係數進行量化處理。以下是量化的基本原理：
   (1)量化矩陣：量化過程中使用的量化矩陣是一個固定的二維矩陣，量

化矩陣中的每個元素對應於 2D-DCT 係數矩陣中的一個位置，，其元素值決定了量化的程度。下圖 4,5 為此專題中使用的量化矩陣，其中通道 CbCr 之矩陣相同:

```
// 定義 Y, Cb, Cr 的量化矩陣
int Q_Y[8][8] = {
    {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}};
```

圖 4:通道 Y 之矩陣

```
int Q_Cb[8][8] = {
    {17, 18, 24, 47, 99, 99, 99, 99},
    {18, 21, 26, 66, 99, 99, 99, 99},
    {24, 26, 56, 99, 99, 99, 99, 99},
    {47, 66, 99, 99, 99, 99, 99, 99},
    {99, 99, 99, 99, 99, 99, 99, 99},
    {99, 99, 99, 99, 99, 99, 99, 99},
    {99, 99, 99, 99, 99, 99, 99, 99},
    {99, 99, 99, 99, 99, 99, 99, 99}};
```

圖 5:通道 CbCr 之矩陣

(2)量化過程：量化過程是通過將 2D-DCT 係數除以對應位置上的量化矩陣元素來實現的，亦即對於每個係數，將其除以量化矩陣中對應位置的元素，然後對結果進行四捨五入或者直接截斷取整，最終得到量化後的係數。

(3)量化細節：由於量化過程中的除法和取整操作，會導致一些係數的信息丟失。一般來說，量化矩陣中的元素值越大，量化的程度就越高，壓縮後的數據量就越小，但同時也會導致更多的信息丟失。因此，量化過程中量化矩陣的選擇是影響壓縮品質的一個重要因素。

(4)逆量化：在 JPEG 解壓縮過程中，量化過的係數需要經過逆量化處理才能恢復原始的 2D-DCT 係數。逆量化的過程就是將量化後的係數乘以量化矩陣中對應位置的元素，以恢復原始的 2D-DCT 係數。

➢ 程式使用說明

- **使用方法 0:**

1. Encoder 指令:

   encoder 0 Kimberly.bmp R.txt G.txt B.txt dim.txt

   輸入:

   (1) 0: 代表第零個使用方法

(2) Kimberly.bmp: 代表輸入的 bmp 檔

輸出:

(1) R.txt: 以 ascii 儲存的 channel R 的值，Kimberly.bmp 裏面 channel R 的每一個 row 的值存成一個 R.txt 的一個 row。

(2) G.txt: 以 ascii 儲存的 channel G 的值，Kimberly.bmp 裏面 channel G 的每一個 row 的值存成一個 G.txt 的一個 row。

(3) B.txt: 以 ascii 儲存的 channel B 的值，Kimberly.bmp 裏面 channel B 的每一個 row 的值存成一個 B.txt 的一個 row。

(4) dim.txt: 裡面儲存圖的 size

2. Decoder 指令:

decoder 0 ResKimberly.bmp R.txt G.txt B.txt dim.txt

輸入:

(1) 0: 代表第零個使用方法

(2) R.txt: 由 encoder 使用第零個使用方法以 ascii 儲存的 channel R 的值。

(3) G.txt: 由 encoder 使用第零個使用方法以 ascii 儲存的 channel G 的值。

(4) B.txt: 由 encoder 使用第零個使用方法以 ascii 儲存的 channel B 的值。

(5) dim.txt: 由 encoder 使用第零個使用方法得到圖的 size

輸出:

(1) ResKimberly.bmp，是從輸入的 R/G/B.txt 以及 dim.txt 讀取資料，然後再儲存成 bmp 格，此輸出應該要和輸入圖檔相同

- **使用方法 1:**用於計算以及檢證 RGB t0 YCbCr、2D-DCT、Quantization 之後是否動作正確並且計算出計算量化誤差。

1. Encoder 指令:

encoder 1 Kimberly.bmp Qt_Y.txt Qt_Cb.txt Qt_Cr.txt dim.txt qF_Y.raw qF_Cb.raw qF_Cr.raw eF_Y.raw eF_Cb.raw eF_Cr.raw

輸入 :

(1)代表第壹個使用方法

(2) Kimberly.bmp: 代表輸入的 bmp 檔

輸出:

(1)Qt_Y.txt: 以 ascii 儲存的 channel Y 的 quantization table，8x8 quantization table 的每一個 row 的值存成一個 Qt_Y.txt 的一個 row，方便人觀察數值。

(2)Qt_Cb.txt: 以 ascii 儲存的 channel Cb 的 quantization table，儲存方法同 Qt_Y.txt。

(3)Qt_Cr.txt: 以 ascii 儲存的 channel Cr 的 quantization table，儲存方法同 Qt_Y.txt。

(4)dim.txt: 圖的 size

(5)qF_Y.raw: 以 binary 儲存的 channel Y 經過量化後的值，也就是 qF_Y=round(F_Y/Qt_Y)， 其中 F_Y 就是經過 2D-DCT 轉換後的頻率軸值，而 Qt_Y 就是量化表裡的數值，因為圖檔有 504 x 378 個 8x8 的 block，每一個 block 有 64 個值，請依 row-major order (https://en.wikipedia.org/wiki/Row-_and_column-major_order)順序將每一個量化值以 short 資料型態儲存。

(6)qF_Cb.raw: 以 binary 儲存的 channel Cb 經過量化後的值，儲存方式如同 qF_Y.raw。

(7)qF_Cr.raw: 以 binary 儲存的 channel Cr 經過量化後的值，儲存方式如同 qF_Y.raw。

(8)eF_Y.raw: 以 binary 儲存的 Channel Y 量化誤差值，也就是 eF_Y = F_Y - qF_Y * Qt_Y，每一個誤差值以 float 資料型態儲存，儲存順序如同 qF_Y.raw。

(9)eF_Cb.raw: 以 binary 儲存的 channel Cb 量化誤差值，儲存順序與方法如同 eF_Y.raw。

(10)eF_Cr.raw: 以 binary 儲存的 channel Cr 量化誤差值，儲存順序與方法如同 eF_Y.raw。

(11)螢幕列印出以 ascii 表示的 3x64 頻率軸上的 SQNR (dB)，其中 3 代表 Y/Cb/Cr，64 代表 8x8 的頻率。

2. Decoder 指令:

1(a):輸出的還原圖為沒加上量化誤差，故和原圖不同

decoder 1 QResKimberly.bmp Kimberly.bmp Qt_Y.txt Qt_Cb.txt Qt_Cr.txt dim.txt qF_Y.raw qF_Cb.raw qF_Cr.raw

輸入 :

(1)1:代表第壹個使用方法

(2)Kimberly.bmp: 原本的圖檔，將用來計算以 pixel 為單位的分 R/G/B 三個 channel 的 SQNR (dB)

(3)Qt_Y.txt: 以 ascii 儲存的 channel Y 的 quantization table。

(4)Qt_Cb.txt: 以 ascii 儲存的 channel Cb 的 quantization table。

(5)Qt_Cr.txt: 以 ascii 儲存的 channel Cr 的 quantization table。

(6)dim.txt: encoder 輸出的圖 size

(7)qF_Y.raw: 以 binary 儲存的 channel Y 經過量化後的值。

(8)qF_Cb.raw: 以 binary 儲存的 channel Cb 經過量化後的值。

(9)qF_Cr.raw: 以 binary 儲存的 channel Cr 經過量化後的值。

輸出:

(1)QResKimberly.bmp，是由 Qt_Y/Cb/Cr.txt 以及 qF_Y/Cb.Cr.raw 重建的 bmp 檔，會和原本的 Kimberly.bmp 有誤差。

(2)螢幕列印出以 ascii 表示的 3x1 pixel 為單位的 SQNR，其中 3 代表 R/G/B 的三個 channel 。也就是用 QResKimberly.bmp 和 Kimberly.bmp 之間的差值來計算。

1(b):還原出的原圖加上量化誤差故和輸入圖片相同

decoder 1 ResKimberly.bmp Qt_Y.txt Qt_Cb.txt Qt_Cr.txt dim.txt qF_Y.raw qF_Cb.raw qF_Cr.raw eF_Y.raw eF_Cb.raw eF_Cr.raw

輸入:

(1)1: 代表第壹個使用方法

(2)Qt_Y.txt: 以 ascii 儲存的 channel Y 的 quantization

table。

(3)Qt_Cb.txt: 以 ascii 儲存的 channel Cb 的 quantization table。

(4)Qt_Cr.txt: 以 ascii 儲存的 channel Cr 的 quantization table。

(5)dim.txt: encoder 輸出的圖 size

(6)qF_Y.raw: 以 binary 儲存的 channel Y 經過量化後的值。

(7)qF_Cb.raw: 以 binary 儲存的 channel Cb 經過量化後的值。

(8)qF_Cr.raw: 以 binary 儲存的 channel Cr 經過量化後的值。

(9)eF_Y.raw: 以 binary 儲存的 Channel Y 量化誤差值。

(10)eF_Cb.raw: 以 binary 儲存的 channel Cb 量化誤差值。

(11)eF_Cr.raw: 以 binary 儲存的 channel Cr 量化誤差值。

輸出:

(1)ResKimberly.bmp，是由 Qt_Y/Cb/Cr.txt、qF_Y/Cb/Cr.raw、以及 eF_Y/Cb/Cr.raw，重建的 bmp 檔，應該要和原本的 Kimberly.bmp 一模一樣。

- **MAKEFILE:**

  make:編譯 encoder 以及 decoder

  make demo0:執行使用方法 0

  make demo1:執行使用方法 1

  makeclean:清除產生之檔案，以利下一次執行

- **心得與反思:**

這次作業十分的有挑戰性，本來有排定進度，但是在寫程式的過程中，發生了太多不可預期的 bug，像是做到 IDCT，和 Unqutization 時，才發現我在 RGB 轉 YCbCr 時，資料存取的形式就發生了錯誤，才導致後面還原出來的圖片出現錯誤，或是在 decoder0 還原 bmp 時才發現 header 有錯誤……等，非常可惜的是我對於 JPEG 的運算細節有疏忽，所以在 Debug 上面就花費非常多時間，如果下次還有機會接觸到這種較為大型的程式，我一定會更早開始去了解題目的根本。 在這學期的課程中，非常感謝教授以及助教的教導以及解惑，因為我對於程式的理解沒有很敏捷，所以時常有很多問

題需要去詢問，非常謝謝教授以及助教不厭其煩地幫我解答。