Data Compression Project 1

Author

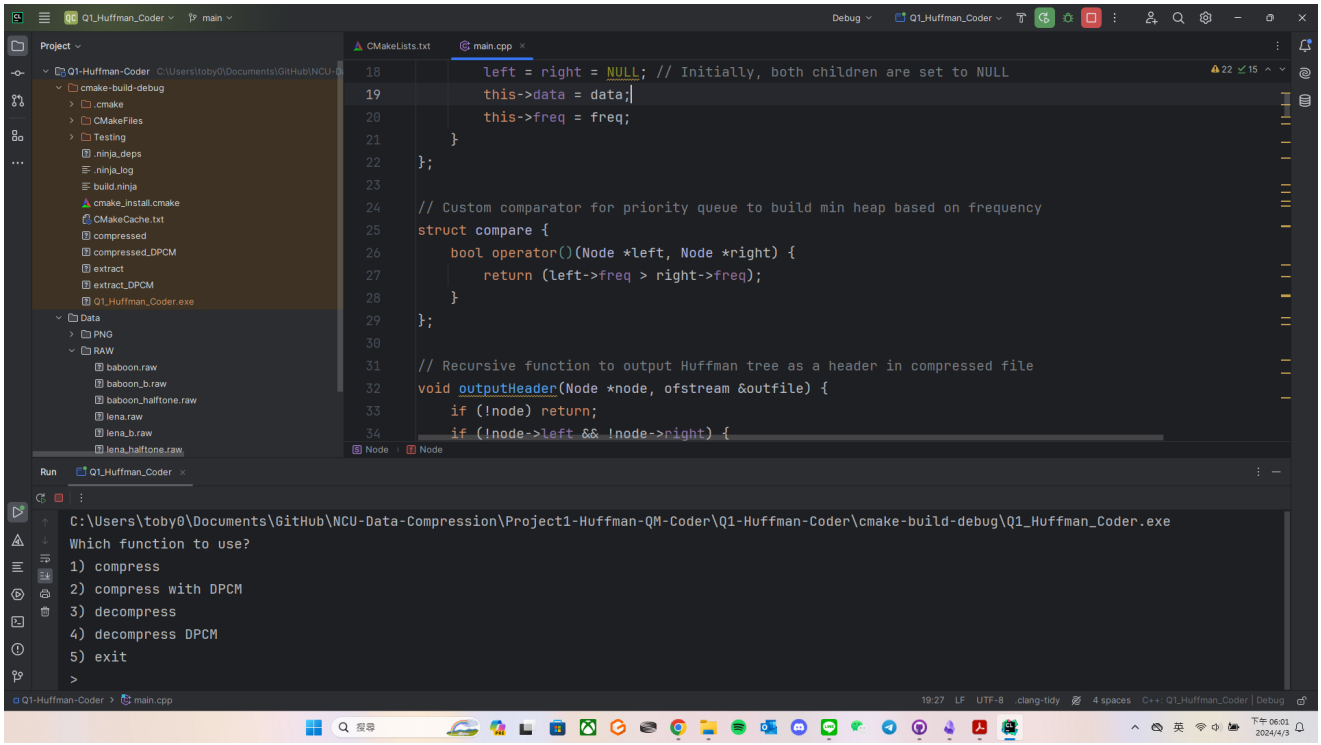
- 112522083
- 中央資工碩一
- 鄧祺文

Environment

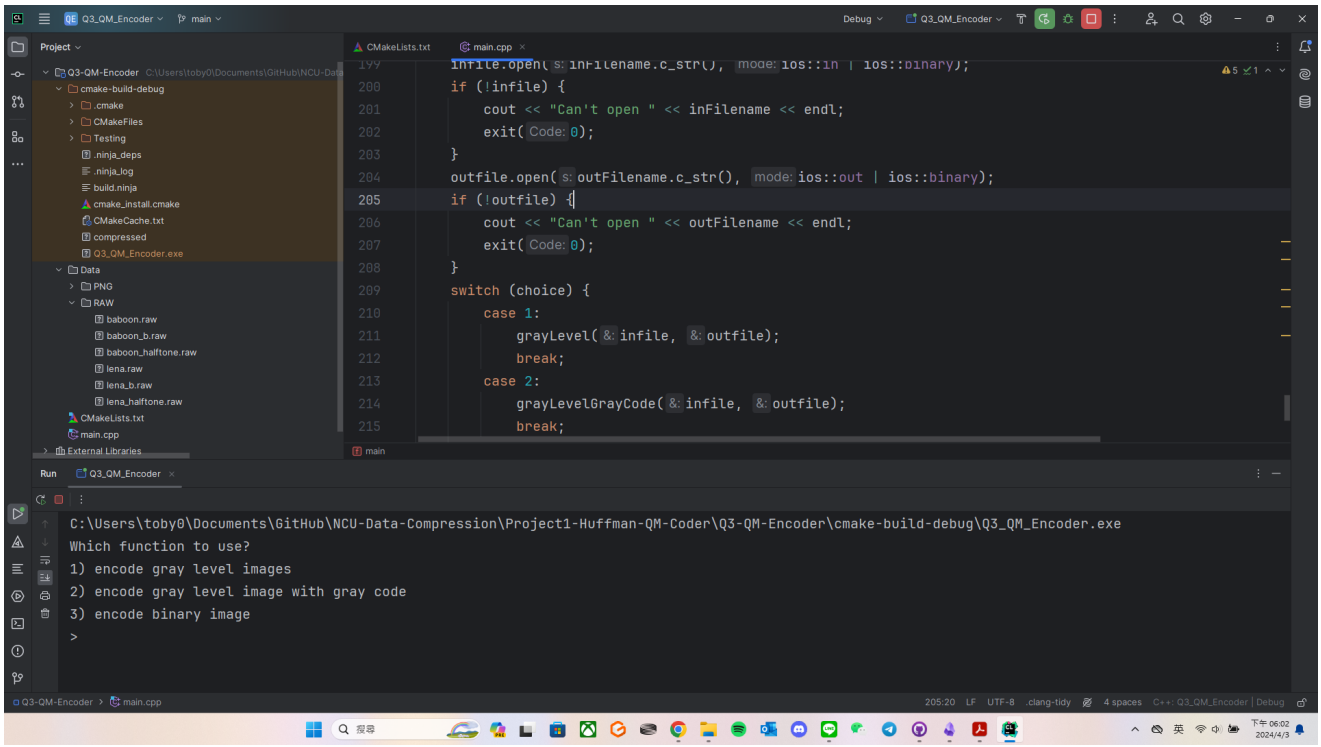
- Operating System: Windows 11 Professional
- C/C++ Compiler: MinGW LLVM
- Project IDE: JetBrains CLion with CMake

開啟 JetBrains CLion 並完成 C/C++ Compiler 配置後，開啟 Q1-Huffman-Coder 和 Q3-QM-Encoder 兩個 Project 的資料夾，即可執行並於 Terminal 進入功能選單。

Q1-Huffman-Coder Function Menu



Q3-QM-Encoder Function Menu



Q1: Huffman Coder

Part 1: Compression

Original Data

- 主要的 `Structure` 採用 `C++` 的 `map` 來記錄原始資料各數值出現的次數，並使用 `priority_queue` 來實作 `Heap`，以構成所需的 `Huffman Tree`
- 完成 `Huffman Tree` 的創建之後使用 `DFS (Depth First Search)` 進行遍歷，找出各資料所需的編碼，接著便開始壓縮工作
- 然後在執行壓縮 `Process` 的時候多用一個 `Buffer Counter` 來記錄現在裡面有多少個 `Bit`，只要塞滿八個 `Bit` 就寫進壓縮檔案內

ORIG baboon.raw	ORIG lena.raw
[+] Compressing... [+] Compress complete!! [+] Original file size = 65536 byte [+] Compressed file size = 60142 byte [+] Compress rate = 91.7694 % [+] Entropy = 7.24065	[+] Compressing... [+] Compress complete!! [+] Original file size = 65536 byte [+] Compressed file size = 63169 byte [+] Compress rate = 96.3882 % [+] Entropy = 7.59536

DCPM Data

- DPCM 是採用圖片中相鄰的兩個 Pixel 所保存的資料相對接近這一概念來進行操作，也就是說壓縮的概念是基於相鄰的資料來進行操作，而不是直接使用 Original Data 來壓縮
- 作業代碼裡面採用的運算方法是以當前資料左側的數據來做預測，因為要拿左邊的資料來進行預測，所以需要事先知道圖片的長寬，否則一不小心拿到最右和最左的資料就會導致計算差值大的不合理

Info

代碼概念設計如下：

1. 使用當前資料減去左側資料，得到差值落在 (-255) ~ 255 之間
2. 發現會多浪費一個 Bit (需要用到九個)，選擇除以二來重新讓區間落在 (-127.5) ~ 127.5 內，接著將區間加上 128 使差值結果皆為正，變成處在 0 ~ 255 之間，最後只要把差值減掉 128 再乘以 2 再搭配左邊的資料就能 Rebuild
3. 而 Rebuild 的時候會沒有原始資料，所以在計算差值的時候是拿當前的原始資料和左邊重建的資料來算差值，至於最左邊會因為沒有參考的 Data，這邊選擇拿 128 當作基準

```
for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
                if (j == 0) {
                        errBuf.push_back((infile.get() - 128) / 2 + 128);
                        rebuildBuf.push_back(128 + (errBuf.back() - 128) *
2);
                } else {
                        errBuf.push_back((infile.get() -
rebuildBuf.back()) / 2 + 128);
                        rebuildBuf.push_back(rebuildBuf.back() +
(errBuf.back() - 128) * 2);
                }
        }
}
```

Hint

DPCM 的優勢在於只要有差值就能 Rebuild，直接記錄所有差值並對差值結果進行壓縮即可

DPCM baboon.raw	DPCM lena.raw
[+] Compressing... [+] Compress complete!! [+] Original file size = 65536 byte [+] Compressed file size = 44113 byte [+] Compress rate = 67.3111 % [+] Entropy = 5.30714	[+] Compressing... [+] Compress complete!! [+] Original file size = 65536 byte [+] Compressed file size = 40159 byte [+] Compress rate = 61.2778 % [+] Entropy = 4.80015

Part 2: Extraction

Original Data Codewords Transfer

- 最開始要先記錄壓縮的檔案一共使用了多少 Data，舉個例子，如果壓縮檔案會用到 A, B 和 C，就將三記錄下來，這樣才能分辨哪些是用來建立 Huffman Tree 而哪些又是真正的檔案資料
- 接著一樣走 DFS 去對 Huffman Tree 進行遍歷，碰到 Internal Node 就輸出 0，而碰到 Leaf Node 就輸出 1 並同時 Output 該 Node 的資料，如此 Decoder 才有辦法判斷說 0 是 Internal Node，需要繼續往下遍歷；而 1 就代表該 Node 是 Leaf，要在這個 Node 填上 1 後面的資料
- 經過前面的操作使 Decoder 和 Encoder 擁有相同的 Huffman Tree ，即可解壓縮

DPCM Data Codewords Transfer

- 由於 DPCM 需要使用相鄰的資料來進行預測，必須事先知道原始圖片的長寬，再來就跟 Original Data 的操作沒什麼區別，一樣把 Huffman Tree 的資料送去 Decoder 進行重建即可解壓縮
- 因為 DPCM 是拿相鄰資料相減的結果去做壓縮，所以解壓縮出來的資料會是兩兩相鄰的計算差值，只要把所有差值按照長寬排好，減掉 128 再乘以 2，最後加上左邊那筆重建後的資料就能完成還原

Part 3: Final Results

- 影像皆使用 .raw 進行處理與比較，原始大小皆為 65535 (Byte)

	File	Compressed Size (Byte)	Compressed Ratio	Entropy
ORIG	baboon	60142	91.7694	7.24065
ORIG	lena	63169	96.3882	7.59536
	File	Compressed Size (Byte)	Compressed Ratio	Entropy
DPCM	baboon	44113	67.3111	5.30714
DPCM	lena	40159	61.2778	4.80015

> Summary
>
> 1. 從實驗結果我們可以發現使用 Original Data 進行壓縮的效果並不明顯，整體的 Entropy 也維持在較高的水平
> 2. 相較之下，使用 DPCM Data 來進行壓縮可以明顯地看到優勢，同時 Entropy 對照 Original Data 也有顯著下降

Q3: QM Encoder

> Info
>
> 包含下列三部分：
>
> 1. Gray Level Image 根據 Bit-Plane 進行壓縮
> 2. Gray Level Image 先進行 Gray Code 運算後之後再根據 Bit-Plane 去壓縮
> 3. Binary Image 直接進行壓縮

Gray Level Image (Pure)

- Gray Level Image 直接就是灰階資料，每一個 Pixel 的 Value 只會落在 0 ~ 255 之間，256 個數字可以用 8 Bit 進行表示，所以由高到低區分成 8 個 Bit-Plane，然後為了可以分成不同的 Bit-Plane 進行探討，會需要先把所有的 Data 都吃進來再基於不同Bit-Plane 進行壓縮
- 按照每筆資料的 MSB 到 LSB 的前後依序存取 Bit Stream 並送入 Encoder 中 (通過 Left Shift 持續讀取最高位)，接著基於 QM Encoder 的定義去設計即可，同樣的這裡也需要一個 Buffer Counter 來記錄 Encoder 輸出的 Bit，然後八個八個的寫入到壓縮檔案內 (用一個 bitLength 來統計 Bit-Plane 的總長，當 Encoder 進行輸出時就將bitLength 加一)

Test Result

GLI baboon.raw	GLI lena.raw
7 bit-plane's bit-stream length = 51485 6 bit-plane's bit-stream length = 51260 5 bit-plane's bit-stream length = 60211 4 bit-plane's bit-stream length = 61023 3 bit-plane's bit-stream length = 61208 2 bit-plane's bit-stream length = 61478 1 bit-plane's bit-stream length = 61294 0 bit-plane's bit-stream length = 61336 [+] Encode complete!!	7 bit-plane's bit-stream length = 35292 6 bit-plane's bit-stream length = 49954 5 bit-plane's bit-stream length = 51972 4 bit-plane's bit-stream length = 58358 3 bit-plane's bit-stream length = 60576 2 bit-plane's bit-stream length = 61314 1 bit-plane's bit-stream length = 61338 0 bit-plane's bit-stream length = 61388 [+] Encode complete!!

Gray Level Image (with Gray Code)

- 跟前一部分的操作基本一致，只是需要另外對每一筆 Data 進行 Gray Code 操作後才把資料丟到 Encoder 去
- Gray Code 的想法是把每一筆 Data 跟當前 Data 除以 2 的結果去執行 Exclusive OR 運算，得到的值就是 Gray Code，獲得 Gray Code 之後再輸入到 Encoder 裡面進行壓縮

Test Result

GLIGC baboon.raw	GLIGC lena.raw
7 bit-plane's bit-stream length = 51485 6 bit-plane's bit-stream length = 64536 5 bit-plane's bit-stream length = 51199 4 bit-plane's bit-stream length = 54932 3 bit-plane's bit-stream length = 60435 2 bit-plane's bit-stream length = 61232 1 bit-plane's bit-stream length = 61462 0 bit-plane's bit-stream length = 61324 [+] Encode complete!!	7 bit-plane's bit-stream length = 35292 6 bit-plane's bit-stream length = 57308 5 bit-plane's bit-stream length = 50412 4 bit-plane's bit-stream length = 52854 3 bit-plane's bit-stream length = 57303 2 bit-plane's bit-stream length = 60518 1 bit-plane's bit-stream length = 61403 0 bit-plane's bit-stream length = 61454 [+] Encode complete!!

Binary Image

- 由於 Binary Image 直接就是 0 或 1 的資料，直接傳送到 Encoder 即可，沒有額外的操作需要實現，不過會需要注意一下 Binary Data 的排列方式，視情況會需要進行 Left Shift (Bit Manipulation) 來確保輸入 Encoder 的 Data

---