Data Structure Result Report-HW3

太空三 郭哲文 109607508

---

1. 原理

(1) Huffman Coding

一開始，所有的節點都是終端節點，節點內有三個欄位：

A.符號（Symbol）

B.權重（Weight、Probabilities、Frequency）

C.指向父節點的鏈結（Link to its parent node）

而非終端節點內有三個欄位：

A.權重（Weight、Probabilities、Frequency）

B.指向兩個子節點的鏈結（Links to two child node）

C.指向父節點的鏈結（Link to its parent node）

基本上，我們用'0'與'1'分別代表指向左子節點與右子節點，最後為完成的二元樹共有 n 個終端節點與 n-1 個非終端節點，去除了不必要的符號並產生最佳的編碼長度。

過程中，每個終端節點都包含著一個權重（Weight、Probabilities、Frequency），兩兩終端節點結合會產生一個新節點，新節點的權重是由兩個權重最小的終端節點權重之總和，並持續進行此過程直到只剩下一個節點為止。

實現霍夫曼樹的方式有很多種，可以使用優先佇列（Priority Queue）簡單達成這個過程，給與權重較低的符號較高的優先順序（Priority），演算法如下：

I.把 n 個終端節點加入優先佇列，則 n 個節點都有一個優先權 Pi，$1 \leq i \leq n$

II.如果佇列內的節點數>1，則：

a.從佇列中移除兩個最小的 Pi 節點，即連續做兩次 remove（min（Pi）, Priority Queue)

b.產生一個新節點，此節點為 a.之移除節點之父節點，而此節點的權重值為 a.兩節點之權重和

c.把 b.產生之節點加入優先佇列中

III.最後在優先佇列裡的點為樹的根節點（root）

而此演算法的時間複雜度（Time Complexity）為 O(n log n)；因為有 n 個終端節點，所以樹總共有 2n-1 個節點，使用優先佇列每個迴圈須 O(log n)。

此外，有一個更快的方式使時間複雜度降至線性時間（Linear Time）O(n)，就是使用兩個佇列（Queue）創件霍夫曼樹。第一個佇列用來儲存 n 個符號（即 n 個終端節點）的權重，第二個佇列用來儲存兩兩權重的合（即非終端節點)。此法可保證第二個佇列的前端（Front）權重永遠都是最小值，且方法如下：

I.把 n 個終端節點加入第一個佇列（依照權重大小置換，最小在前端）

II.如果佇列內的節點數>1，則：

a.從佇列前端移除兩個最低權重的節點

b.將 a.中移除的兩個節點權重相加合成一個新節點

c.加入第二個佇列

III.最後在第一個佇列的節點為根節點

雖然使用此方法比使用優先佇列的時間複雜度還低，但是注意此法的第 1 項，節點必須依照權重大小加入佇列中，如果節點加入順序不按大小，則需要經過排序，則至少花了 O(n log n)的時間複雜度計算。

但是在不同的狀況考量下，時間複雜度並非是最重要的，如果我們今天考慮英文字母的出現頻率，變數 n 就是英文字母的 26 個字母，則使用哪一種演算法時間複雜度都不會影響很大，因為 n 不是一筆龐大的數字。

霍夫曼碼樹的解壓縮就是將得到的前置碼（Prefix Huffman code）轉換回符號，通常藉由樹的追蹤（Traversal），將接收到的位元串（Bits stream）一步一步還原。但是要追蹤樹之前，必須要先重建霍夫曼樹；某些情況下，如果每個符號的權重可以被事先預測，那麼霍夫曼樹就可以預先重建，並且儲存並重複使用，否則，傳送端必須預先傳送霍夫曼樹的相關資訊給接收端。

最簡單的方式，就是預先統計各符號的權重並加入至壓縮之位元串，但是此法的運算量花費相當大，並不適合實際的應用。若是使用 Canonical encoding，則

可精準得知樹重建的資料量只占 B2^B 位元（其中 B 為每個符號的位元數（bits））。如果簡單將接收到的位元串一個位元一個位元的重建，例如：'0'表示父節點，'1'表示終端節點，若每次讀取到 1 時，下 8 個位元則會被解讀是終端節點（假設資料為 8-bit 字母），則霍夫曼樹則可被重建，以此方法，資料量的大小可能為 2~320 位元組不等。雖然還有很多方法可以重建霍夫曼樹，但因為壓縮的資料串包含"trailing bits"，所以還原時一定要考慮何時停止，不要還原到錯誤的值，如在資料壓縮時時加上每筆資料的長度等。

(2) Arithmetic Coding

待補…

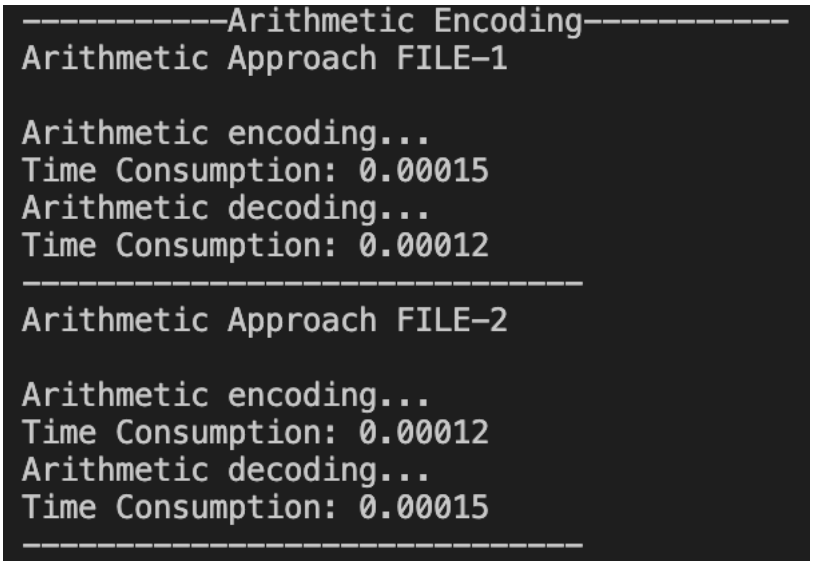
2. 編譯&執行

把一個文字檔裡的內容拆成好幾行，分別對每一行去用兩種 Coding 去 Eecoding & Decoding。

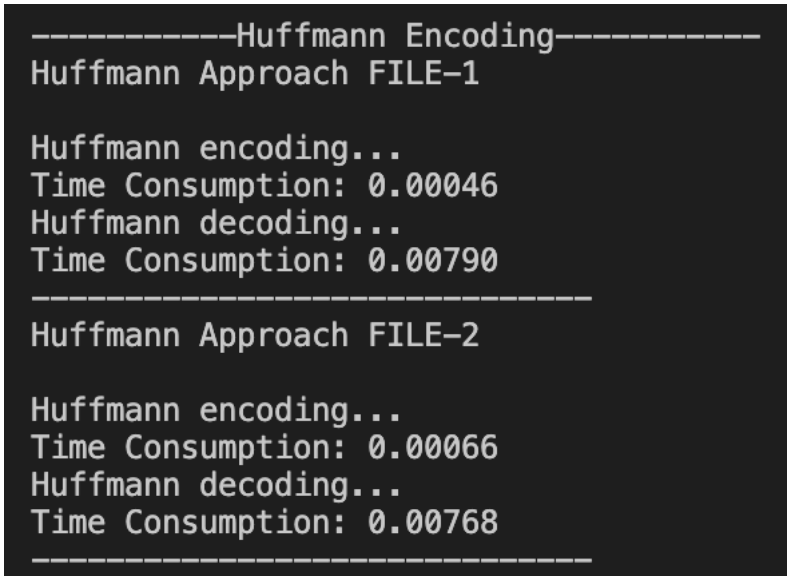
2.1 輸入 make && ./build/main.o

Arithmetic Coding 有時候會卡住，不知道為什麼 ☹

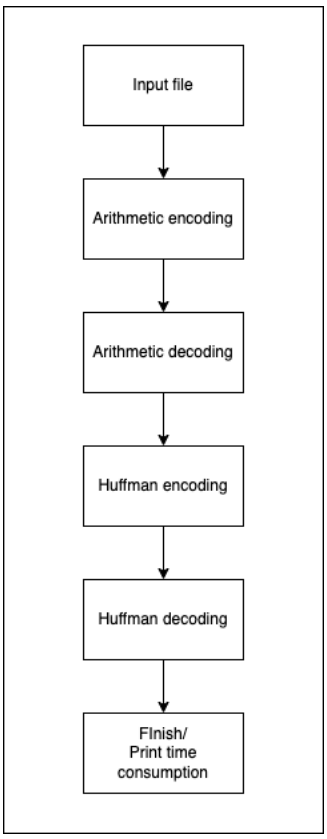
2.2 Arithmetic Coding



2.3 Huffman Coding



3. 流程



4. 分析

在 Arithmetic Coding 的時候不知道為什麼沒辦法跳入下一個檔案去執行，這來不及去探討，如果未來有機會的話會仔細去 debug。