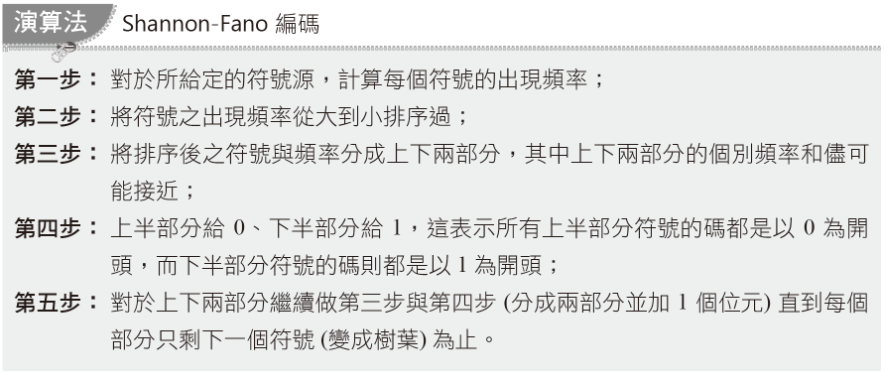
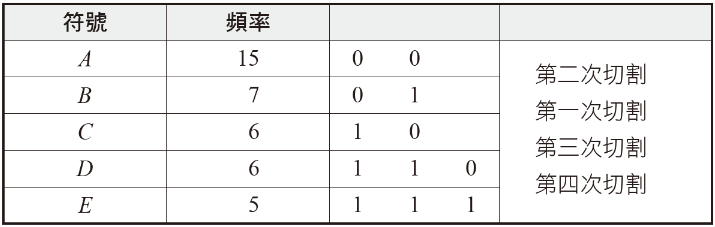
* Part 1: 入門
  + Chapter 1: 介紹
    - 1.3.1 編碼冗贅
      * 符號平均長度
        + 符號: ( = 1 ~ ); 對應機率:
        + 符號平均長度: ; : 符號長度
    - 1.4 資料壓縮之種類
      * 壓縮比(compression ratio)
* Part 3: 無失真壓縮
  + Chapter 5: 無失真資料壓縮──統計模式
    - 5.1 Shannon-Fano 編碼
      * Algorithm



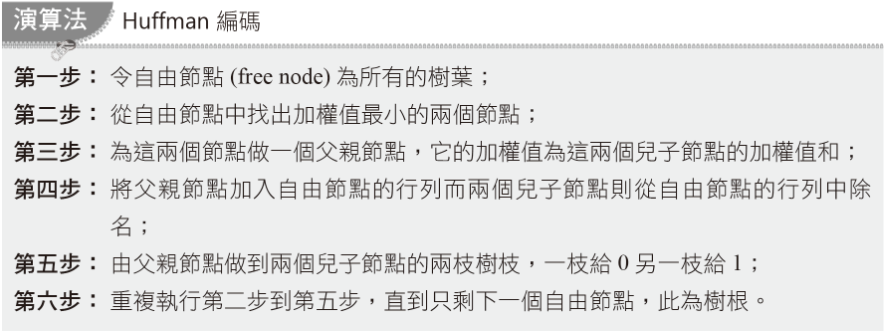
* + - * 例子



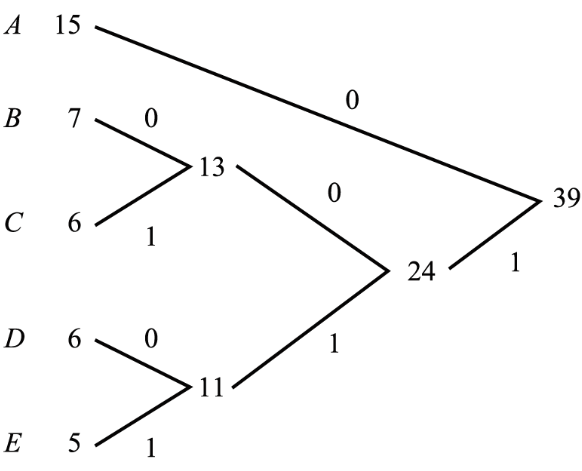
* + - * 資訊量

|  |  |  |  |
| --- | --- | --- | --- |
| 符號 | 頻率(出現次數) | 資訊量 | 總資訊量 |
| A | 15 | 1.38 | 20.68 |
| B | 7 | 2.48 | 17.35 |
| C | 6 | 2.70 | 16.20 |
| D | 6 | 2.70 | 16.20 |
| E | 5 | 2.96 | 14.82 |

* + - * 總結
        + 總資訊量: 85.25 bits
        + 改善編碼: 2(15+7+6) + 3(6+5) = 89 bits
        + 原始編碼，ASCII: 39 \* 8 = 312 bits
    - 5.2 Huffman編碼
      * Algorithm
        + 讀取數值時，資料型態為stack，由上到下



* + - * 例子



1

2

3

4

* + - * 總結
        + 總資訊量: 85.25 bits of information
        + Shannon-Fano (SF): 89 bits
        + Huffman (Huff): 87 bits

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 符號 | 頻率 | SF位元數 | SF總位元數 | Huff位元數 | Huff總位元數 |
| A | 15 | 2 | 30 | 1 | 15 |
| B | 7 | 2 | 14 | 3 | 21 |
| C | 6 | 2 | 12 | 3 | 18 |
| D | 6 | 3 | 18 | 3 | 18 |
| E | 5 | 3 | 15 | 3 | 15 |

* + - 5.3 適應性Huffman編碼
      * 適應性編碼: adaptive coding
      * 兄弟性質(Sibling property)
        + 如果一棵樹的節點可以按照加權值從小排到大而且每個節點又和自己的兄弟相鄰的話，我們便稱這棵樹具有兄弟性質
        + 定理:

一棵二元樹是 Huffman 樹 <==> (若且唯若)它遵守兄弟性質。

* + - * + 範例

w: 權重值

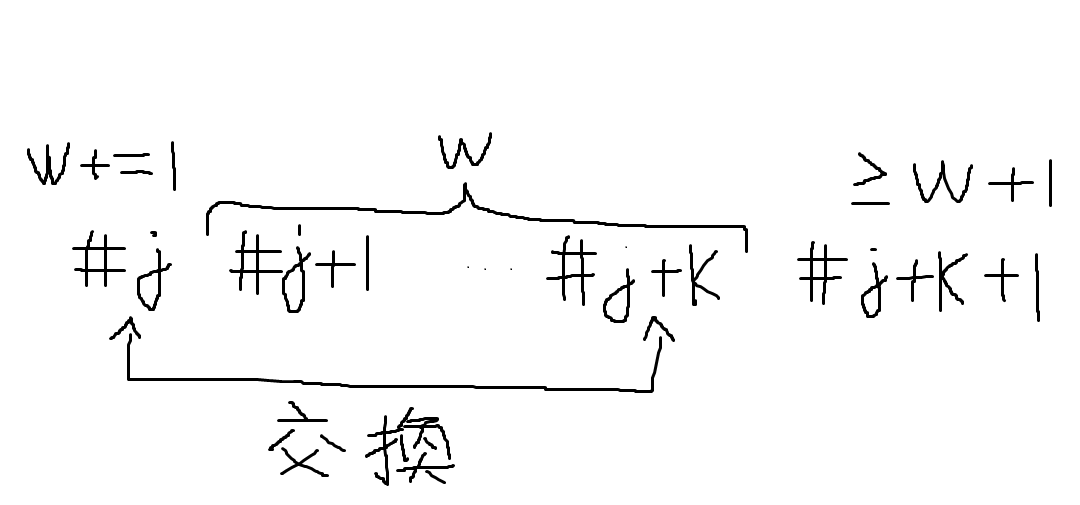
輸入 => #j (w) => #j (w+1);

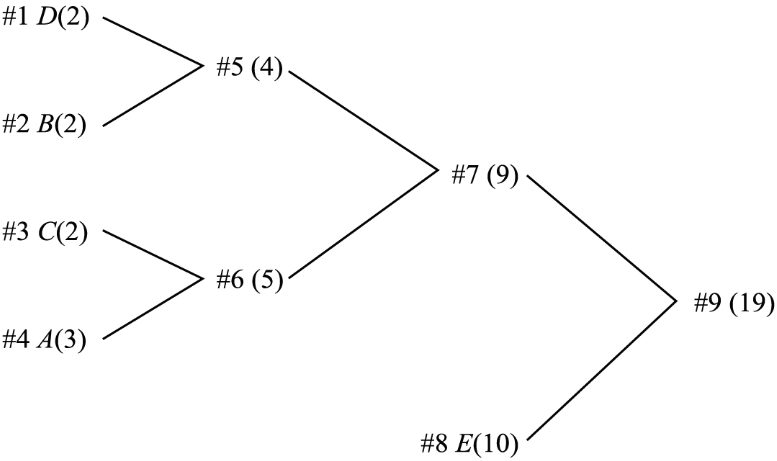
可得知 #(j+1) ~ #(j+k) 權重值為w;

找#(j+k+1) 權重值>=w+1

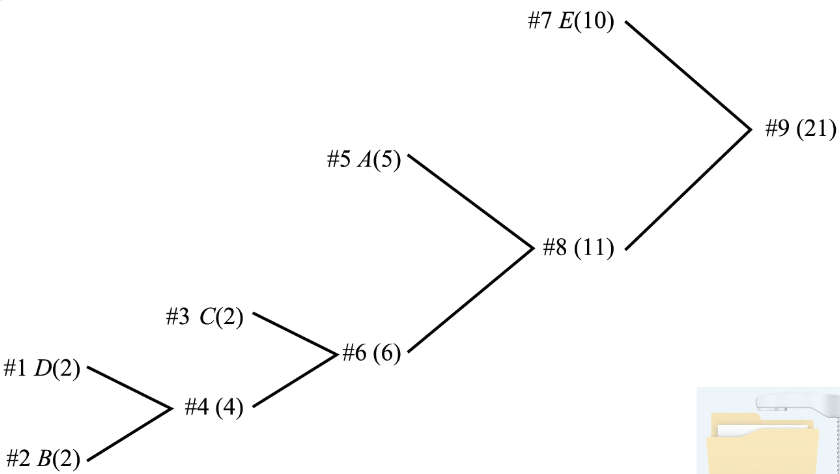
交換#j, #j+k，編號(#)不變

更新#j, #j+k 父節點





“#4 A(3)” += 2 => “A(5)”，與 “#5 (4)” 交換。



* + - * + 初始化

建立空統計表，只存有的符號，避免不存在的符號拖累效能

填入2個符號: EOF(End of File), ESC加權值為1

EOF:

檔案結束

ESC:

輸入符號不在樹裡

輸出: ESC的Huffman code

輸出: 輸入符號的ASCII

加入輸入符號

範例: 輸入16個’e’

1. 輸出9 bits:  
   ESC:1 bit,  
   e ASCII: 8bits
2. 接下來的’e’為Huffman code
   * + - * 溢位問題(The overflow problem)

加權值超過變數容納範圍

解決方法:

所有節點的加權值除以2

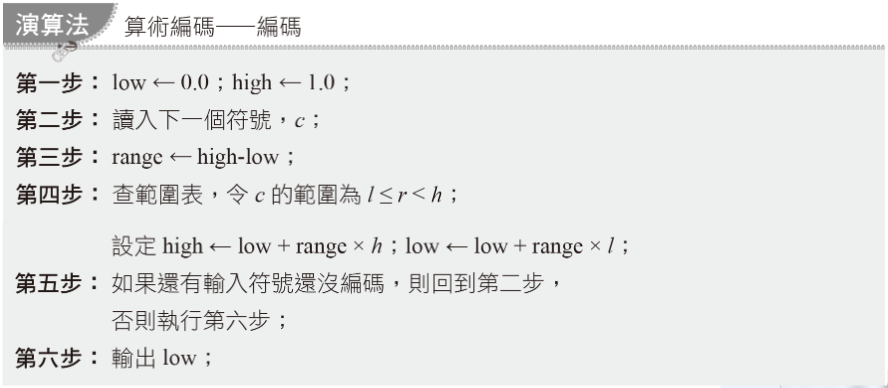
忽略小數點後位數

檢查是否遵守Sibling property

範例

|  |  |
| --- | --- |
|  |  |
| 原始圖 | 加權值除以2 |
|  | |
| 調整後 | |

* + - 5.4 算術編碼
      * Arithmetic coding
      * 避開了一個符號一個碼
      * 採取用一個實數來表示一串符號
      * Algorithm



* + - * 範例
        + 利用機率計算範圍值

|  |  |  |
| --- | --- | --- |
| 符號 | 機率 | 範圍 |
| SPACE(^) | 1/10 | 0.0 <= r < 0.1 |
| A | 1/10 | 0.1 <= r < 0.2 |
| B | 1/10 | 0.2 <= r < 0.3 |
| E | 1/10 | 0.3 <= r < 0.4 |
| G | 1/10 | 0.4 <= r < 0.5 |
| I | 1/10 | 0.5 <= r < 0.6 |
| L | 2/10 | 0.6 <= r < 0.8 |
| S | 1/10 | 0.8 <= r < 0.9 |
| T | 1/10 | 0.9 <= r < 1.0 |

* + - * + 利用範圍值計算low, high

Init(): low=0.0, high=1.0

讀取一個字元 c

range = high – low

讀取 c 的範圍

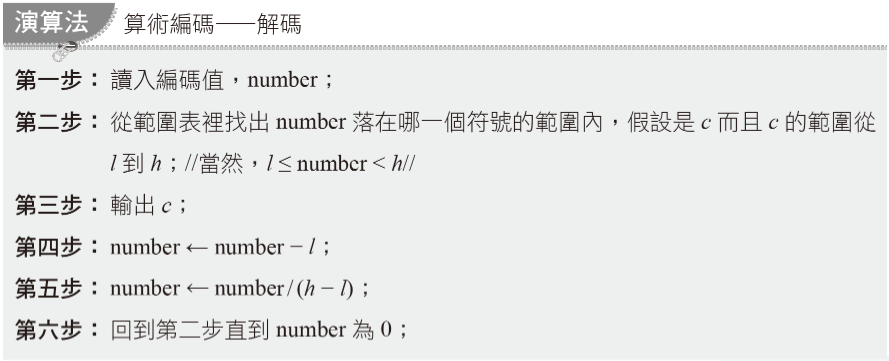
high = low + range \* ;  
low = low + range \* ;

GO TO step 2, if have next c

Print low

|  |  |  |
| --- | --- | --- |
| 新讀入符號 | low | high |
|  | 0.0 | 1.0 |
| B | 0.2 | 0.3 |
| I | 0.25 | 0.26 |
| L | 0.256 | 0.258 |
| L | 0.2572 | 0.2576 |
| ^ | 0.25720 | 0.25724 |
| G | 0.257216 | 0.257220 |
| A | 0.2572164 | 0.2572168 |
| T | 0.25721676 | 0.2572168 |
| E | 0.257216772 | 0.257216776 |
| S | 0.2572167752 | 0.2572167756 |

* + - * Algorithm解碼

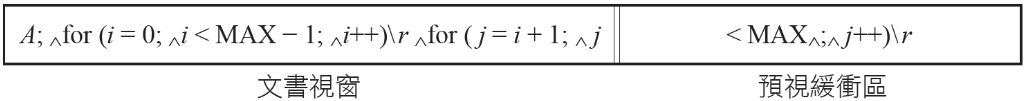


* + - * 範例
        + Encode:  
          low = low + range \* 0.2 = 0.0 + (1.0-0.0) \* 0.2
        + Decode:  
          number = (number - ) / ()  
          0.0 = (0.2 – 0.2) / (0.3 – 0.2)

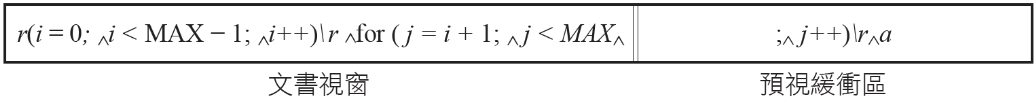
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number | 輸出符號 |  |  |  |
| 0.2572167752 | B | 0.2 | 0.3 | 0.1 |
| 0.572167752 | I | 0.5 | 0.6 | 0.1 |
| 0.72167752 | L | 0.6 | 0.8 | 0.2 |
| 0.6083876 | L | 0.6 | 0.8 | 0.2 |
| 0.041938 | ^ | 0.0 | 0.1 | 0.1 |
| 0.41938 | G | 0.4 | 0.5 | 0.1 |
| 0.1938 | A | 0.1 | 0.2 | 0.1 |
| 0.938 | T | 0.9 | 1.0 | 0.1 |
| 0.38 | E | 0.3 | 0.4 | 0.1 |
| 0.8 | S | 0.8 | 0.9 | 0.1 |
| 0.0 |  |  |  |  |

* + Chapter 6:字典基礎模式
    - Dictionary-based model
    - 6.2 Lempel Ziv壓縮法
    - 6.3 滑動視窗壓縮法
      * LZ77 輸出格式:

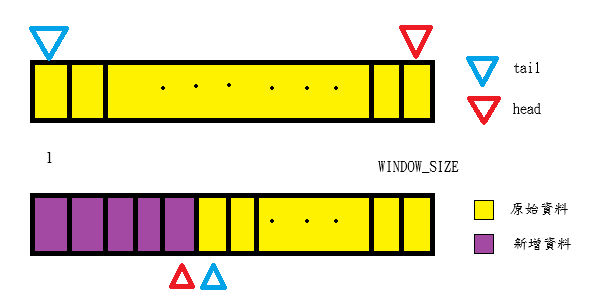
1. 文書視窗(text window)內片語的起始位置
2. 片語的長度
3. 預視緩衝區(look-ahead buffer)緊接片語的符號
   * + - 範例
         * 輸出 (14, 4, ^)



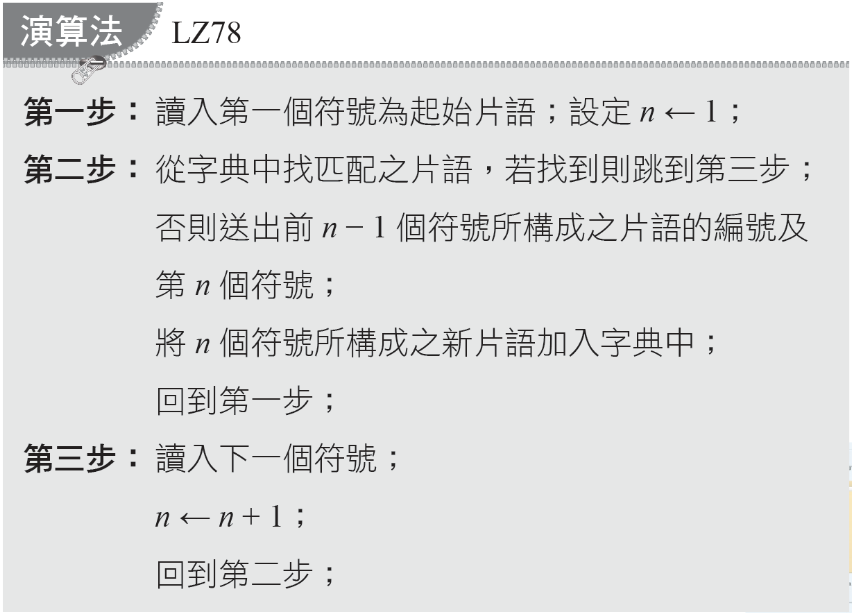
* + - * + 輸出後，text window往前移5格
        + 輸出 (33, 3, +)



* + - * + 未找到:  
          (0, 0, ‘符號’)
      * 環狀佇列(circular queue)  
        解決text window搬移資料的效能問題  
        設: 輸入5筆資料  
        head = (head + 5) mod WINDOW\_SIZE;   
        tail = (tail + 5) mod WINDOW\_SIZE;  
        資料存入(head + i) mod WINDOW\_SIZE, i = 1~5



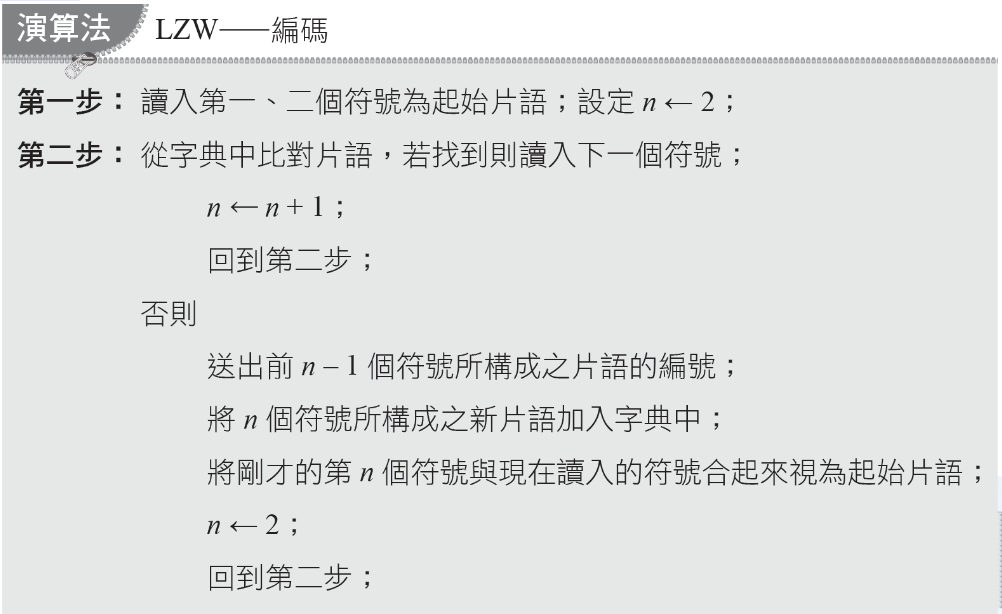
* + - 6.4 LZ78與LZW
      * 捨棄text window
      * LZ78 Algorithm



* + - * 範例
        + 輸入”DAD^DADA^DADDY^DADO…”  
          一開始 0 :

|  |  |  |  |
| --- | --- | --- | --- |
| 輸出 | | 新加入字典 | 新片語之編號 |
| 片語編號 | 字母 |
| 0 | D | D | 1 |
| 0 | A | A | 2 |
| 1 | ^ | D^ | 3 |
| 1 | A | DA | 4 |
| 4 | ^ | DA^ | 5 |
| 4 | D | DAD | 6 |
| 1 | Y | DY | 7 |
| 0 | ^ | ^ | 8 |
| 6 | O | DADO | 9 |

* + - * LZW Algorithm



* + - * 範例  
        ^WED^WE^WEE^WEB^WET…  
        0~255: alphabet

|  |  |  |
| --- | --- | --- |
| 輸入之符號 | 輸出碼 | 新片語及其編號 |
| ^WED^WE^WEE^WEB^WET… | | |
| “W” | ‘^’編碼 | 256 = ”^W” |
| ^WED^WE^WEE^WEB^WET… | | |
| “E” | ‘W’編碼 | 257 = “WE” |
| ^WED^WE^WEE^WEB^WET… | | |
| “D” | ‘E’編碼 | 258 = “ED” |
| ^WED^WE^WEE^WEB^WET… | | |
| “^” | ‘D’編碼 | 259 = “D^” |
| ^WED^WE^WEE^WEB^WET… | | |
| “WE” | 256 | 260 = “^WE” |
| ^WED^WE^WEE^WEB^WET… | | |
| “^” | ‘E’編碼 | 261 = “E^” |
| ^WED^WE^WEE^WEB^WET… | | |
| “WEE” | 260 | 262 = “^WEE” |
| ^WED^WE^WEE^WEB^WET… | | |
| “^W” | 261 | 263 = “E^W” |
| ^WED^WE^WEE^WEB^WET… | | |
| “EB” | 257 | 264 = “WEB” |
| ^WED^WE^WEE^WEB^WET… | | |
| “^” | ‘B’編碼 | 265 = “B^” |
| ^WED^WE^WEE^WEB^WET… | | |
| “WET” | 260 | 266 = “^WET” |
| ^WED^WE^WEE^WEB^WET… | | |
| “T” | ‘T’編碼 | 267 = “T^” |

* + - * LZW 解碼
        + Step 1:讀入第一個編號，從字典找出片語並輸出。  
          Step 2:讀入下一個編號，從字典找出片語並輸出；  
          {前一個片語 + 目前片語的第一個符號}，存入字典；  
          回到Step 2；
        + 範例 ^WED<256>E<260><261><257>B<260>T

|  |  |  |
| --- | --- | --- |
| 輸入編號 | 解碼所得片語(輸出) | 新加入字典中之片語 |
| ^WED<256>E<260><261><257>B<260>T | | |
| ‘^’ | “^” |  |
| ^WED<256>E<260><261><257>B<260>T | | |
| ‘W’ | “W” | 256 = “^W” |
| ^WED<256>E<260><261><257>B<260>T | | |
| ‘E’ | “E” | 257 = “WE” |
| ^WED<256>E<260><261><257>B<260>T | | |
| ‘D’ | “D” | 258 = “ED” |
| ^WED<256>E<260><261><257>B<260>T | | |
| 256 | “^W” | 259 = “D^” |
| ^WED<256>E<260><261><257>B<260>T | | |
| ‘E’ | “E” | 260 = “^WE” |
| ^WED<256>E<260><261><257>B<260>T | | |
| 260 | “^WE” | 261 = “E^” |
| ^WED<256>E<260><261><257>B<260>T | | |
| 261 | “E^” | 262 = “^WEE” |
| ^WED<256>E<260><261><257>B<260>T | | |
| 257 | “WE” | 263 = “E^W” |
| ^WED<256>E<260><261><257>B<260>T | | |
| ‘B’ | “B” | 264 = “WEB” |
| ^WED<256>E<260><261><257>B<260>T | | |
| 260 | “^WE” | 265 = “B^” |
| ^WED<256>E<260><261><257>B<260>T | | |
| ‘T’ | “T” | 266 = “^WET” |

* Midterm
  + LZ87, LZW
  + Arithmetic coding
  + Huffman tree