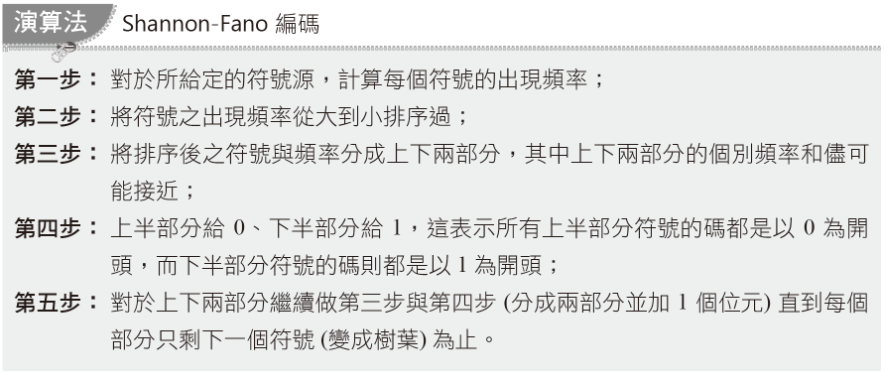
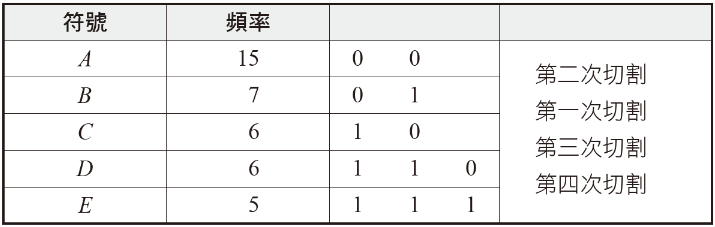
* Part 1: 入門
  + Chapter 1: 介紹
    - 1.3.1 編碼冗贅
      * 符號平均長度
        + 符號: ( = 1 ~ ); 對應機率:
        + 符號平均長度: ; : 符號長度
    - 1.4 資料壓縮之種類
      * 壓縮比(compression ratio)
* Part 3: 無失真壓縮
  + Chapter 5: 無失真資料壓縮──統計模式
    - 5.1 Shannon-Fano 編碼
      * Algorithm



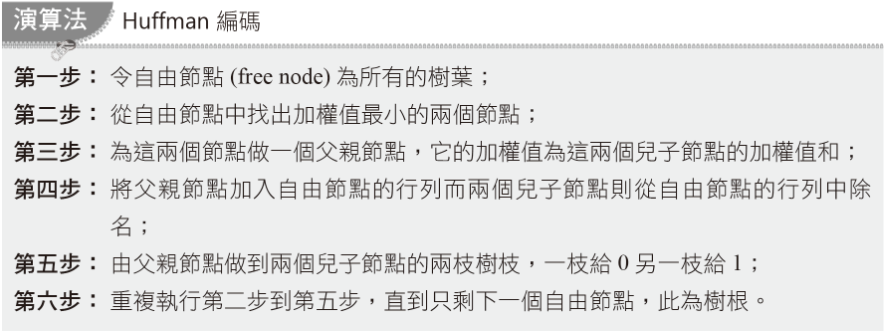
* + - * 例子



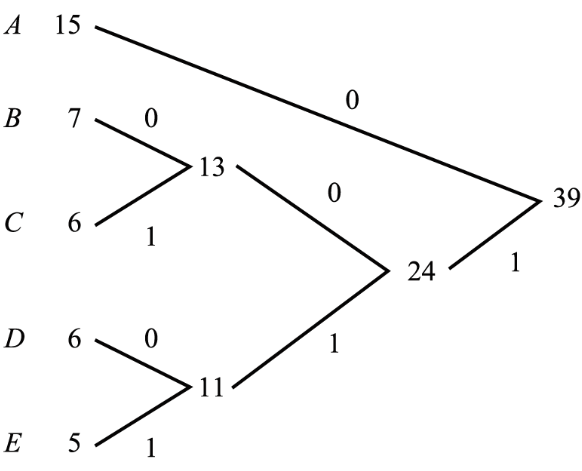
* + - * 資訊量

|  |  |  |  |
| --- | --- | --- | --- |
| 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

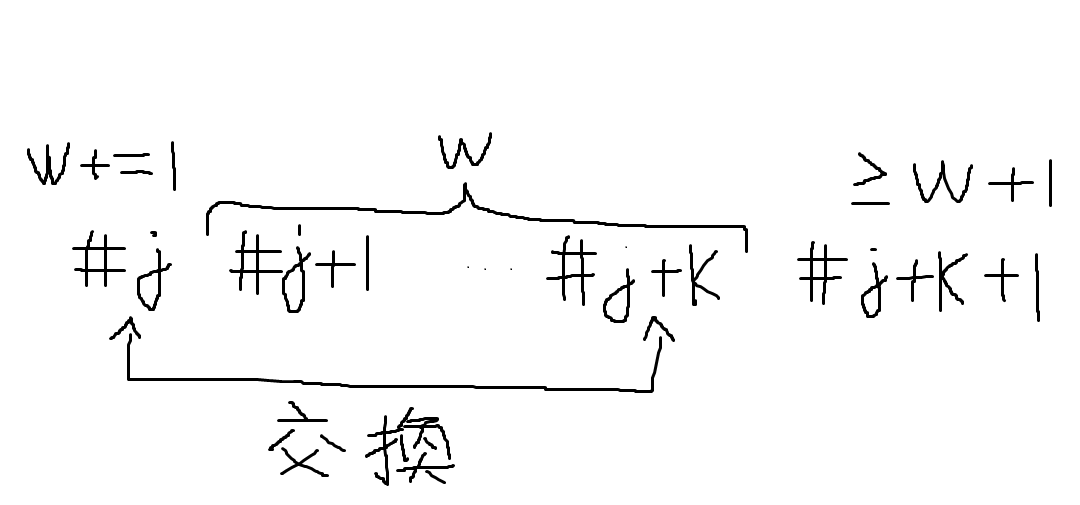
輸入 => #j (w) => #j (w+1);

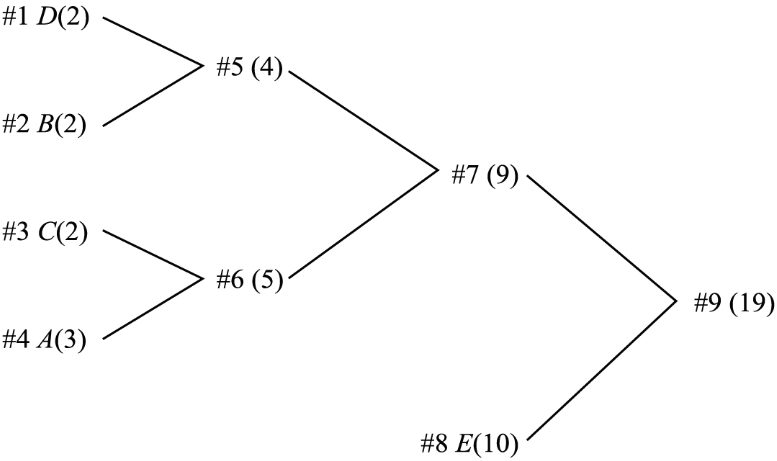
可得知 #(j+1) ~ #(j+k) 權重值為w;

找#(j+k+1) 權重值>=w+1

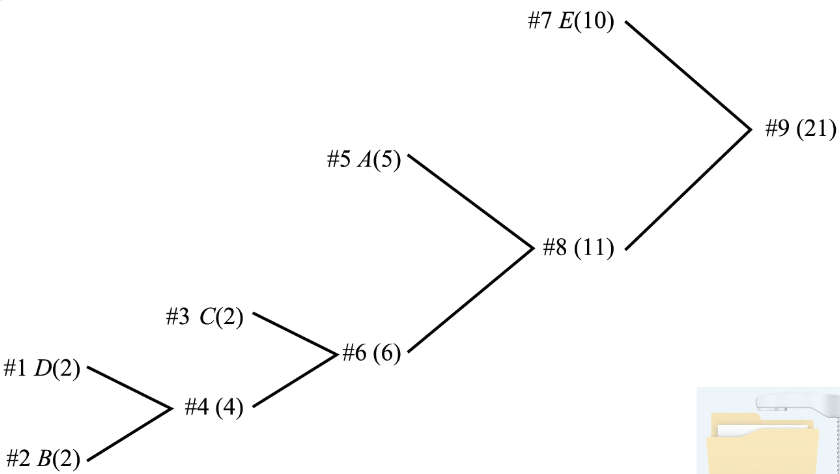
交換#j, #j+k，編號(#)不變

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