



國立臺南大學資訊工程學系

資工三「演算法」課程  
第三次作業

題目: String Matching


班級：資工三

姓名：呂益銓

學號：S10659013

老師：陳宗禧

中華民國 108 年 12 月 4 日



# 目錄

(一) 簡介及問題描述.....	3
1. 簡介.....	3
2. 問題.....	3
(二) 理論分析.....	4
(三) 演算法則.....	5
(四) 程式設計環境架構.....	9
(五) 程式.....	10
(六) 執行結果、討論與心得.....	16
參考文獻.....	19

# (一) 簡介及問題描述

## 1. 簡介

給定兩個字串(or 檔案) Text (長度  $n$ )與 Pattern(長度  $m$ )。請設計與實作 String matching 問題。

## 2. 問題

- i 解 String matching 問題，Text 為一較長文章(可為新聞網站擷取其內容、DNA sequences 等)，Pattern 為我們擬尋找的字串，請實作下列三個演算法，最後分析並比較三個演算法所尋找的時間、頻率、輸入字串的長度( $n$  and  $m$ )間關係等。最後須比較這三種方法的效率，找到或沒有找到 Pattern 除一起比較外，應再個別分開討論。若有多個 pattern，皆找出來，並且計算 Frequencies! Note: 輸入字串需有中文、英文、DNA sequences (參考 Exercises 7.2 第二題作業)
  - a. 實作 Brute-Force Algorithm
  - b. 實作 Horspool's Algorithm
  - c. 實作 Boyer-Moore Algorithm
- ii 找 System log events in log files (多個檔案)!

## (二) 理論分析

### i. Brute-Force Algorithm

從頭到尾 text 每個字都需要比對一次 pattern 有沒有符合，如果符合就跳過 pattern 的長度，繼續比剩下的字，否則就從下一個字繼續比較，直到比到剩餘的長度小於 pattern 的長度的時候就不用繼續比下去了

### ii. Horspool's Algorithm

先對需要比對的 pattern 做一次 preconstruct 建立出 table，完成之後再去比較 text 如果比對錯誤的話就去 table 查詢和 pattern 最後一個字比較的 text 的字不論這個字是比對正確還是錯誤，下次比對的起始點就是現在的起始點再加上查詢到的結果。

#### 1. create shift table

從 pattern 的起始點開始往後，如果遇到不在 table 裡面的字就把它加進 table 並把 value 設為  $\text{length} - i$  (位移量)，如果已經在裡面的字就更新為新的 value，其餘沒在裡面的字的 value 都設為 length。

Example: BARBER

character $c$	A	B	C	D	E	F	...	R	...	Z	_
shift $t(c)$	4	2	6	6	1	6	6	3	6	6	6

#### 2. 根據 table 來判斷比對失敗的 pattern 位移的距離 $t(c)$

如果比對失敗就用比對的最後一個字下去查表，如果查不到就位移 length

$$t(c) = \begin{cases} \text{distance from } c\text{'s rightmost occurrence in pattern} \\ \text{among its first } m-1 \text{ characters to its right end} \\ \text{pattern's length } m, \text{ otherwise} \end{cases}$$

Example:

```

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P
B A R B E R                B A R B E R
      B A R B E R          B A R B E R
          B A R B E R      B A R B E R
```

### iii. Boyer-Moore Algorithm

先對要比對的字串先做兩次 preconstruct 創出兩個 table，一個為 bad shift table，另外一個為 good suffix table，bad shift table 的建立和 Horspool's algorithm 一樣，建立完 table 後再利用 table 下去決定位移的距離，位移的距離由 pattern 和 text 比較後連續正確的字數量(k)來決定，如果  $k=0$  則去 bad table 找最右邊比對錯誤的字，位移距離就是  $\max\{t(c) - k, 1\}$ ，(t(c)為查到的 value)，如果  $k>0$ ，d1 為剛剛所說的方法找出來的距離，d2 為 good suffix table 查表找出來的距離，位移距離為  $\max\{d1, d2\}$ ，重複一直下去直到 text 結尾。

#### 1. Create bad shift table(和 horspool's algorithm 的 table 一樣)

從 pattern 的起始點開始往後，如果遇到不在 table 裡面的字就把它加進 table 並把 value 設為  $\text{length} - i$  (位移量)，如果已經在裡面的字就更新為新的 value，其餘沒在裡面的字的 value 都設為 length。

Example: BAOBAB

C	B	A	0	other
value	5	1	3	6

#### 2. Create good suffix table

從 pattern 的最右邊的一個字開始向左比對如果遇到相同的字就加到 table 裡面，下次就從右邊兩個字開始向左比對，如果比到 pattern 最前面不夠的比的時候只要比對字的後半部分相同就可以，前半部分比不到的地方不用理它，所以 table 總共有  $\text{length} - 1$  個 value

Example:BAOBAB

k	pattern	$d_2$
1	BAOBAB	2
2	BAOBAB	5
3	BAOBAB	5
4	BAOBAB	5
5	BAOBAB	5

， $k$  為連續正確

d2 為利用連續正確的字數 k 下去查表找到的 value

$$d = \begin{cases} d_1 & \text{if } k = 0 \\ \max\{d_1, d_2\} & \text{if } k > 0 \end{cases}$$

B E S S \_ K N E W \_ A B O U T \_ B A O B A B S  
 B A O B A B  
 $d_1 = t_1(K) - 0 = 6$   
 B A O B A B  
 $d_1 = t_1(-) - 2 = 4$  B A O B A B  
 $d_2 = 5$   $d_1 = t_1(-) - 1 = 5$   
 $d = \max\{4, 5\} = 5$   $d_2 = 2$   
 $d = \max\{5, 2\} = 5$   
 B A O B A B

### (三) 演算法則

#### 1. Brute-Force Algorithm

##### Algorithm

Pattern: a string of  $m$  characters to search for

Text: a string of  $n$  characters to search in

Step1: Align pattern at beginning of text

Step2: Moving from left to right, compare each character of pattern to the corresponding character in text until either all characters are found to match (successful search) or a mismatch is detected

Step 3: While a mismatch is detected and the text is not yet exhausted, realign pattern one position to the right and repeat Step 2 if all character of pattern are found to match realign pattern  $m$  position to the right and repeat Step2

時間複雜度(time complexity)

Worst-case:  $O(mn)$

#### 2. Horspool's Algorithm

##### Algorithm Shift table( $P[0..m-1]$ )

// Fill the shift table used by Horspool's and Boyer-Moore algorithms

// Input: Pattern  $P[0..m-1]$  and an alphabet of possible characters

// Output: Table[0..size-1] indexed by the alphabet's characters and

// filled with shift sizes computed by formula (7.1)

for  $i \leftarrow 0$  to size - 1 do Table[i]  $\leftarrow m$

for  $j \leftarrow 0$  to  $m - 2$  do Table[P[j]]  $\leftarrow m - 1 - j$  return Table

## Algorithm HorspoolMatching( $P[0..m-1], T[0..n-1]$ )

//Input: Pattern  $P[0..m-1]$  and text  $T[0..n-1]$   
//Output: The index of the left end of the first matching substring  
// or -1 if there are no matches  
ShiftTable( $P[0..m-1]$ );  
 $i \leftarrow m - 1$   
    while  $i \leq n - 1$  do  
         $k \leftarrow 0$   
        while  $k \leq m - 1$  and  $P[m-1-k] = T[i-k]$  do  
             $k \leftarrow k + 1$   
        if  $k = m$   
            return  $i - m + 1$   
        else  $i \leftarrow i + \text{Table}[T[i]]$   
    return -1

時間複雜度(time complexity)

Worst-case:  $O(mn)$  average:  $O(m+n)$

空間複雜度(space complexity)

Using link list implement  $O(m)$

## 1. Boyer-Moore algorithm

### Algorithm Bad Shift table( $P[0..m-1]$ )

// Fill the shift table used by Horspool's and Boyer-Moore algorithms  
// Input: Pattern  $P[0..m-1]$  and an alphabet of possible characters  
// Output: Table[0..size-1] indexed by the alphabet's characters and  
// filled with shift sizes computed by formula (7.1)  
for  $i \leftarrow 0$  to size - 1 do Table[i]  $\leftarrow m$   
for  $j \leftarrow 0$  to  $m - 2$  do Table[P[j]]  $\leftarrow m - 1 - j$  return Table



## Algorithm Good Suffix Table(P[0..m-1])

```
// Input: Pattern P[0..m-1] and an alphabet of possible characters
// Output: Table[0..size-1] size is pattern size
// Shift_table[0..size-1] size is pattern size
// m is pattern size
Shift_table[0] ← m
Table[m-1] ← m
for i ← m-2 to 0 do
    correct ← 0
    for j ← 0 to i do
        if pattern[i-j] ≠ pattern[m-1-j]
            break
        else correct++
    Shift_table[(m-1)-i] ← correct
    Table[i] ← m
for i ← m-1 to 0 do
    if shift_table[i] == m - i
        for j ← m-i to m-1 do
            Table[j] = i

for i ← m-1 to 1 do
    Table[shift_table[i]] ← i
```

## Boyer-Moore Algorithm

Step 1 Fill in the bad-symbol shift table

Step 2 Fill in the good-suffix shift table

Step 3 Align the pattern against the beginning of the text

Step 4 Repeat until a matching substring is found or text ends:


Compare the corresponding characters right to left. If no characters match, retrieve entry  $t1(c)$  from the bad-symbol table for the text's character  $c$  causing the mismatch and shift the pattern to the right by  $t1(c)$ . If  $0 < k < m$  characters are matched, retrieve entry  $t1(c)$  from the bad-symbol table for the text's character  $c$  causing the mismatch and entry  $d2(k)$  from the good-suffix table and shift the pattern to the right by  $d = \max \{d1, d2\}$  where  $d1 = \max \{t1(c) - k, 1\}$ .

時間複雜度(time complexity)

$O(m + n)$

空間複雜度(space complexity)

$O(m)$



## (四) 程式設計環境架構

程式設計語言、工具、環境與電腦硬體等規格說明...

### 1. 程式語言

C in MS Windows

### 2. 程式開發工具

Visual Studio Code + MinGw64

### 3. 電腦硬體

CPU: Intel i5-9400f

Main Memory: 16GB

Disk: 1T HDD

500G SSD

250G SSD

OS: W10 x64

## (五) 程式 (含 source code, input code, and output code)

程式含 source code, input code, and output code 等...

### String Matching

#### 1. 主程式

1. 先輸入檔名(檔案必須為 big5 編碼方式)

```
D:\algorithm\hw3\test\Project1\hw3.exe
Enter file name.....CBS.log
```

2. 會先輸出檔案大小(BYTE)之後輸入是否要印出檔案  
如果要印出就打 Y 如果不要就打 N(如果檔案很大建議不要印出來會印很久)

```
D:\algorithm\hw3\test\Project1\hw3.exe
Enter file name.....CBS.log
file size 10371029
Do you want to print out the log file(Y or N)N
```

3. 輸入需要尋找的字串(ex: 2019-11-22 20:56:20, Info  
Last boot time: 2019-11-15 23:38:38.039)

CBS      TI:

```
D:\algorithm\hw3\test\Project1\hw3.exe
Enter file name.....CBS.log
file size 10371029
Do you want to print out the log file(Y or N)N
Enter search word.....2019-11-22 20:56:20, Info      CBS      TI: Last boot time: 2019-11-15 23:38:38.039
```

4. 輸出字串出現位置和出現次數和各種演算法比較的次數和時間還有 bad table 和 good suffix table (table 太長這裡不貼上來)

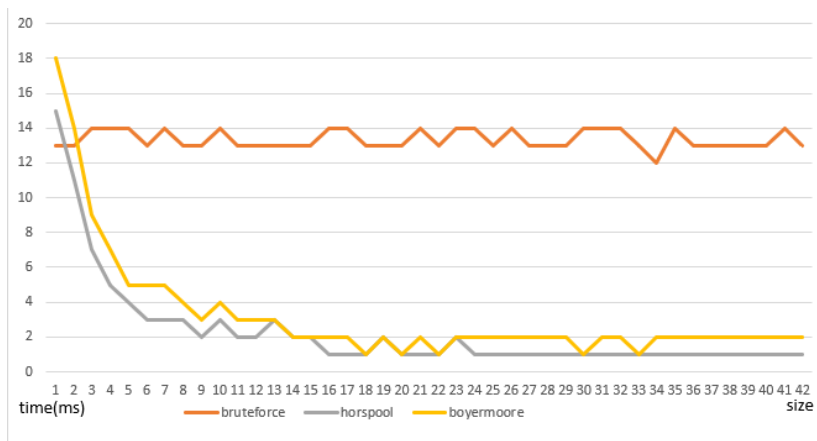
```
Brute force 11001120 comparision time 15 answer size 1
95
Horspool 257791 comparision time 4 answer size 1
95
Boyer_Moore 256240 comparision time 4 answer size 1
95
```

## (五) 執行結果、討論與心得

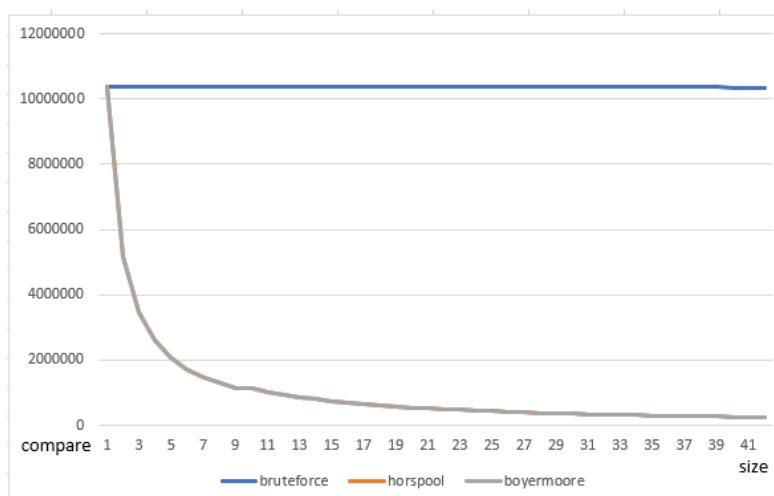
執行結果與討論 (執行時間、problem  $n$  的大小等問題討論)等...

### 1. 執行結果

1. 當比對字串數字增加時間和比較次數變化圖(pattern 皆為不在 text 裡面)



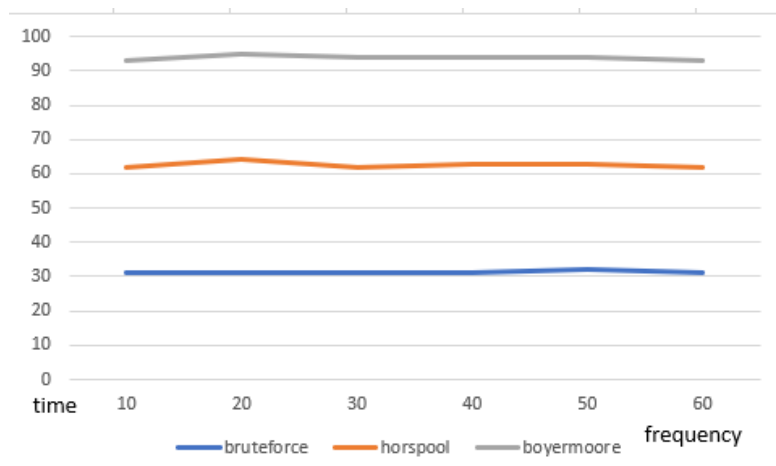
時間(us)-Size 與 Pattern size 關係圖



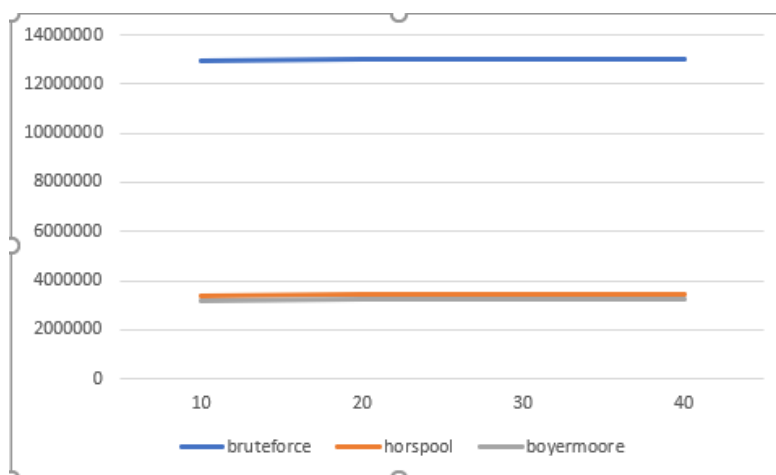
Compare 次數與 Pattern Size 關係圖

比對的字串比較少的时候 brute force 效率比較好是因為相對於 horspool 和 boyer moore, brute force 的 overhead 較少所以跑起來速度快, 而且字串少的时候位移的量也不會太大, compare 次數就和 brute force 差不多可是會多一些額外的判斷式, 當字串變多的時候 brute force 就不會降低還是一樣可是 horspool 和 boyer moore 會因為字串變多位移量增加 compare 次數就大幅減少。

2. 當比對字串出現頻率與時間變化圖(字串出現頻率增加 text 和 pattern 大小不變)



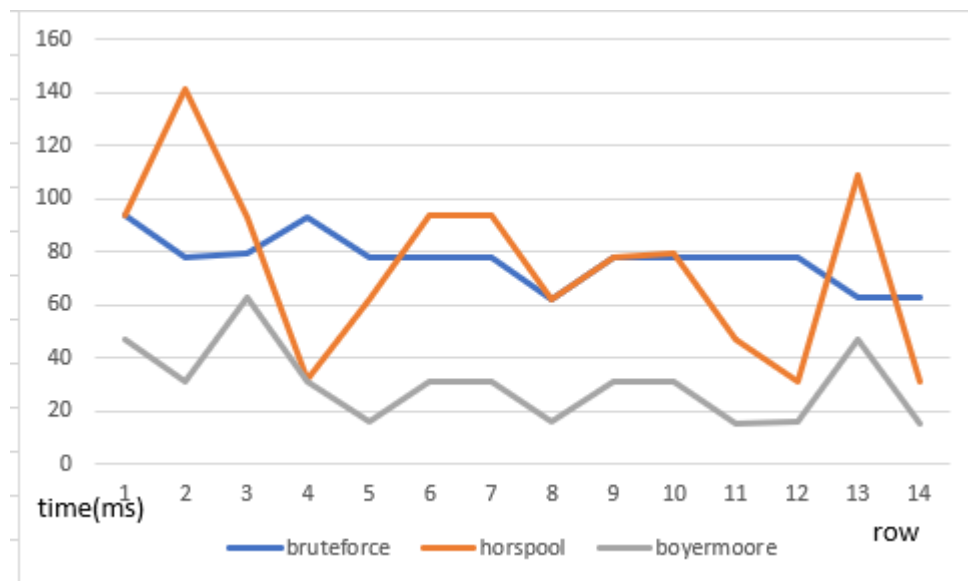
時間(us)-Size 與 frequency 關係圖(time 是累加上去)



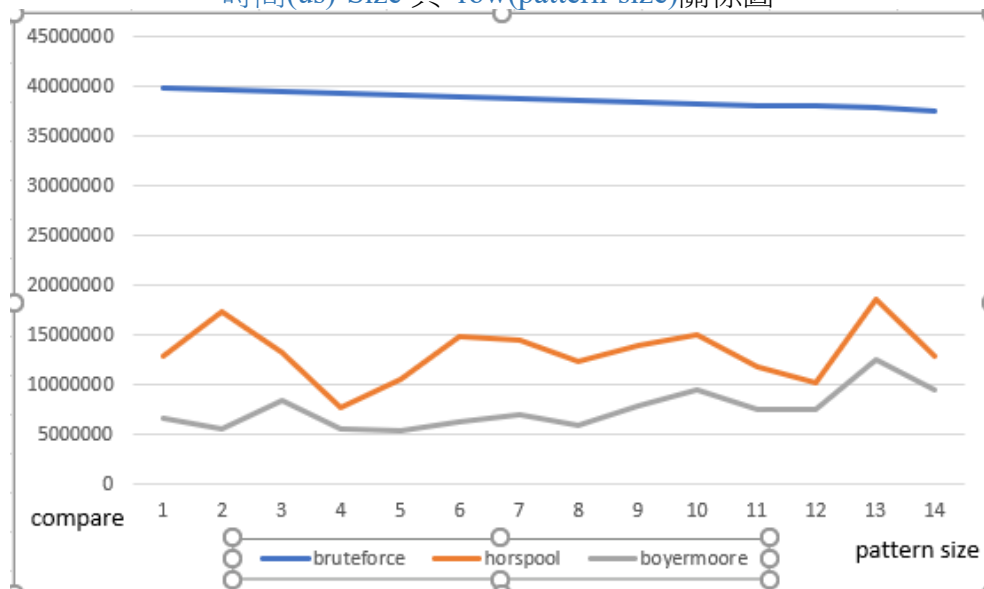
compare 與 frequency 關係圖

對於 pattern 出現次數的多寡並不會影響到字串搜尋時間，所以搜尋字串他出現次數並和演算法並沒有太大關係，因為搜尋 pattern 的關係所以三個演算法跑出來的時間差不多，雖然時間差不多可是 compare 的次數相對於 horspool 和 boyer moore，brute Force 明顯比較多，如果以比較次數當基準那 brute force 就是跑最久的演算法

3. 當比對字串數字增加時間和比較次數變化圖(pattern 在 text 裡面出現的次數不變)



時間(us)-Size 與 row(pattern-size)關係圖



compare 與 pattern-size 關係圖

當 pattern-size 增加時所花費時間會漸漸下降但是因為每次比對字串 size 都會增加所以 compare 次數就不一定會下降，但是對於 boyer moore 和 horspool 來當字串越長時說兩者時間都比 brute force 時間來的快，所以不管字串有沒有在文章裡只要字串越長就越適合使用 boyer moore 和 horspool，但是如果字串很短那 brute force 的方法會比較快因為 brute force overhead 少。

## 參考文獻

[1]

[1] 台南大學 E-course 演算法課程 Algorithms Report Format