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 | ： | 林晉昇 |
| 學號 | ： | S10359004 |

老師：陳宗禧

中華民國 105年12月10日

# 目錄

1. **簡介及問題描述……….……………..…………………………………………3**
   1. **簡介…………….…………………………………………………………………………3**
   2. **問題………………………….……………………………………………………………3**
2. **理論分析與演算法則…………..………………………………………4**
3. **程式設計環境架構.………………………..…………………………………8**
4. **程式.…………………………………………..………………………………9**
5. **執行結果、討論與心得.………………………..……………………………11**

參考文獻………………………………………………………….…………………13

**(一) 簡介及問題描述**

設計與實作計算String Matching，理論驗證與實驗分析該問題!

1. 簡介

實作三種演算法來解決字串配對的問題。

2. 問題

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：

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。

a. 實作 Brute-Force Algorithm

b. 實作 Horspool’s Algorithm

c. 實作 Boyer-Moore Algorithm

Note: 輸入字串需有中文、英文、DNA sequences (參考 Exercises 7.2 第二題作業)。 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。

**(二) 理論分析與演算法則**

1. BruteForce Algorithm

暴力法，非常直觀的作法。將pattern的第一個位置跟text第一個位置作比較，若相同則繼續比，若pattern與text的片段完全吻合，就找到了。若比對不相同則將向pattern後位移一個單位後繼續比較。

以課本3-2的例子來說，text=”NOBODY NOTICED HIM”，pattern=”NOT”

NOBODY\_NOTICED\_HIM //紅色代表有進行比對的動作

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | O | B | O | D | Y | \_ | N | O | T | I | C | E | D | \_ | H | I | M |
| N | O | T |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | N | O | T |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | N | O | T |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | N | O | T |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | N | O | T |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | N | O | T |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | N | O | T |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | N | O | T |  |  |  |  |  |  |  |  |

演算法pseudo code如下:

**ALGORITHM** BruteForceStringMatch(T[0..n-1], P[0..m-1])

//Implements brute-force string matching

//Input: An array T[0..n-1] of n characters representing a text and

// an array P[0..m-1] of m characters representing a pattern

//Output: The index of the first character in the text that starts a

// Matching substring or -1 if the search is unsuccessful

**For** i ←0 to n – m **do**

ｊ←0

**While** j < m and P[j] = T[i+j] **do**

j ←j+1

**if** j=m **return** i

**return** -1

**時間複雜度Time Complexity**

假設text長度為n，pattern長度為m，以worst case來說暴力法每次在位移之前都要進行m次的比較，最多會位移n-m+1次，所以說總共會進行m(n-m+1)次的比較，所以我們可以說暴力法的時間複雜度是O(nm)。

**空間複雜度Space Complexity**

由於不論n與m的值如何改變，都不會消耗額外的記憶體空間，因此暴力法的空間複雜度Space Complexity為P(1)。

2. Horspool’s Algorithm

Horspool演算法是BoyerMoore的簡化版，用一張Bad Symbol Table就解決了。

Bad Symbol Table的演算法如下:

**ALGORITHM** BadSymbolTable(P[0..m-1])

//Files 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 t(c)

**For** i←0 to size-1 **do** Table[i] ←m //將所有值先設為m (m為pattern的長度)

**For** j←0 to m-2 **do** Table[P[j]] ←m-1-j //對所有出現在pattern中的字元，將其table中的值

**Return** Table //設為此字元與末位的距離

＊第二個迴圈因為是由左至右，所以較後出現的字元c的會覆蓋掉前面出現的table的值

Table內的值可能有兩種

t(c)=

▲Bad Symbol Table Example

以字元ABCCBA為例，Shift Table如下:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Character | A | B | C | D | E | F | …… |
| Shift | 5 | 1 | 2 | 6 | 6 | 6 | 6 |

最終Horspool的演算法pseudo code如下:

**ALGORITHM** HorspoolMatching(P[0..m-1], T(0..n-1))

//Implements Horspool’s algorithm for string matching

//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

BadSymbolTable(P[0..m-1]) //產生Shift table

i←m-1 //Pattern 最右手邊的位置

**while** i n-1 **do**

k ← 0 //已經比較過的字元數

**while** k m-1 **and** P[m-1-k] = T[i-k] **do**

k←k+1

**if** k=m

**return** i-m+1

**else**

i←i+Table[T[i]] //i為pattern向右移動的單位數

**return** -1

**時間複雜度Time Complexity**

Worst case 時間複雜度為O(nm)。

**空間複雜度Space Complexity**

空間複雜度為P()，為Bad Symbol Table的Size。

3. BoyerMoore formula

演算法:

Step1 根據text以及pattern的字元建立一個Bad Symbol Table

Step2 根據pattern建立一個Good Suffix Table

Step3 將pattern與text對齊後開始比較，從pattern的末位開始比，看有幾位是相同的， 若k代表相同的位數數量，則如果k=0，則移動max{(c) – k, 1}個位元，如果k>0， 代表至少一位相同，則移動max{d1, d2}位元。重複此步驟直到完全匹配或者比較 至text長度結束為止。

▲Good Suffix Table就是依據字尾匹配成功的字元數量來判斷要位移的多寡，比起Bad Symbol Table只根據最後一個位元來判斷位移多寡，能夠一次位移的長度通常較大。以下舉pattern=”ABCBAB”為例實作Good Sufix Table:

|  |  |  |
| --- | --- | --- |
| k | pattern | d2 |
| 1 | ABCBAB | 2 |
| 2 | ABCBAB | 4 |
| 3 | ABCBAB | 4 |
| 4 | ABCBAB | 4 |
| 5 | ABCBAB | 4 |

**☆Boyer-Moore Algorithm判斷公式**

d= ，where =max{}

時間複雜度Time Complexity

O(m+n)

空間複雜度Space Complexity

P()

**(三) 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++

2. 程式開發工具

Visual Studio C++ 2015 Community

3. 電腦硬體

處理器: Intel Core i7-4500U

RAM: 8GB

系統類型:64位元作業系統，x64處理器**(四) 程式 (含source code, input code, and output code)**

1. 主程式

String\_Matching\_Prog\_S10359004.cpp

2. Input Code Format

Three of examples for input use are in below….

輸入1、2、3選擇English, Chinese, or DNA，接著輸入要尋找的pattern

1. 1

mainstream

1. 2

夢一場

1. 3

TGAGC

3. Output Code Format

輸出為三種演算法所找到pattern在text中的位置，若沒找到則輸出-1。

輸出包含使用三種演算法所花費的時間。

(1) BruteForce: 330

Time: 0.0009857429s

Horspool: 330

Time: 0.0004135416s

BoyerMoore: 330

Time: 0.0017692053s

(2) BruteForce: 48

Time: 0.0012384865s

Horspool: 48

Time: 0.0003074833s

BoyerMoore: 48

Time: 0.0003776186s

(3) BruteForce: -1

Time: 0.0013090494s

Horspool: -1

Time: 0.0002634350s

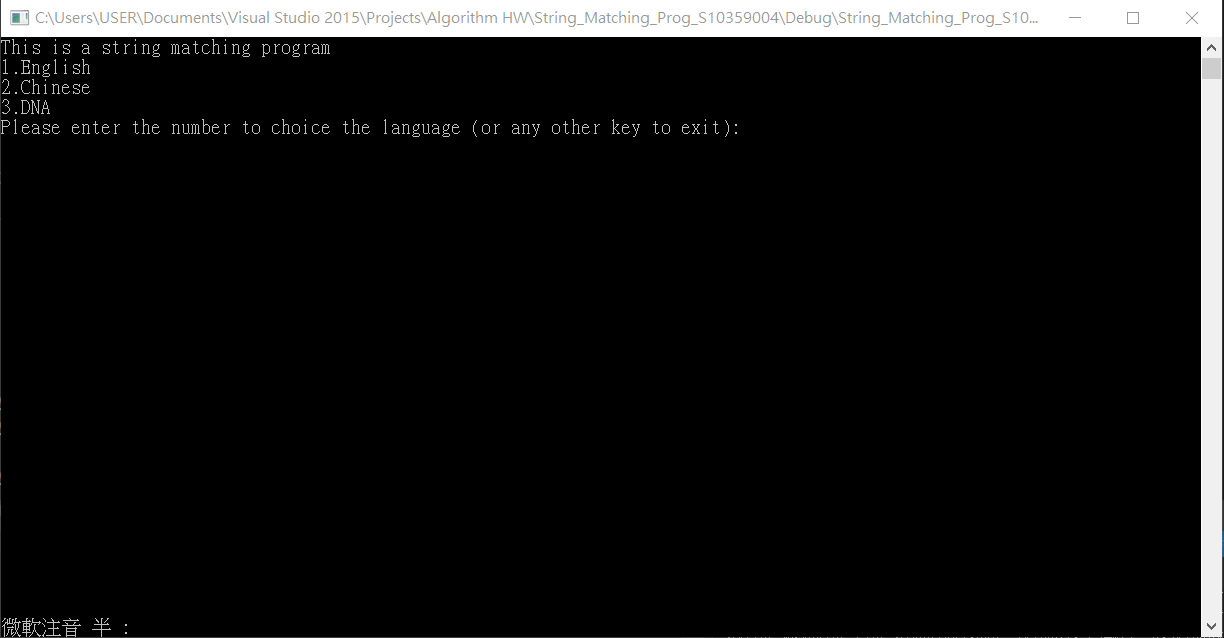
BoyerMoore: -1

Time: 0.0004862428s

**(五) 執行結果、討論與心得**

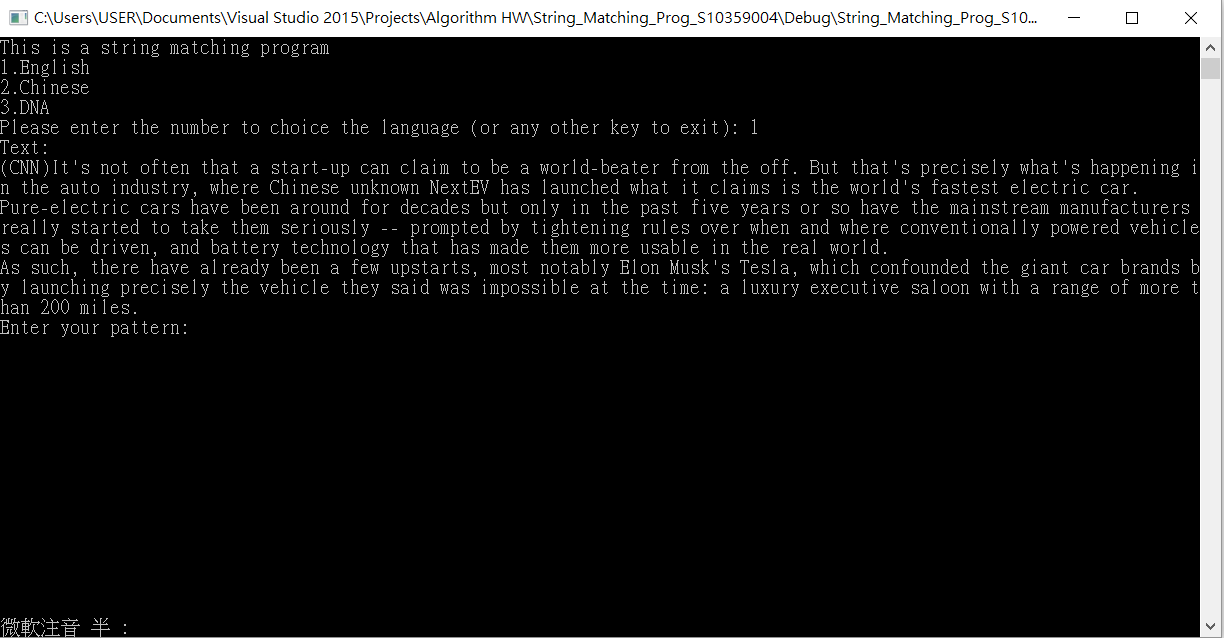
1. 執行結果

(1)程式執行時的狀況



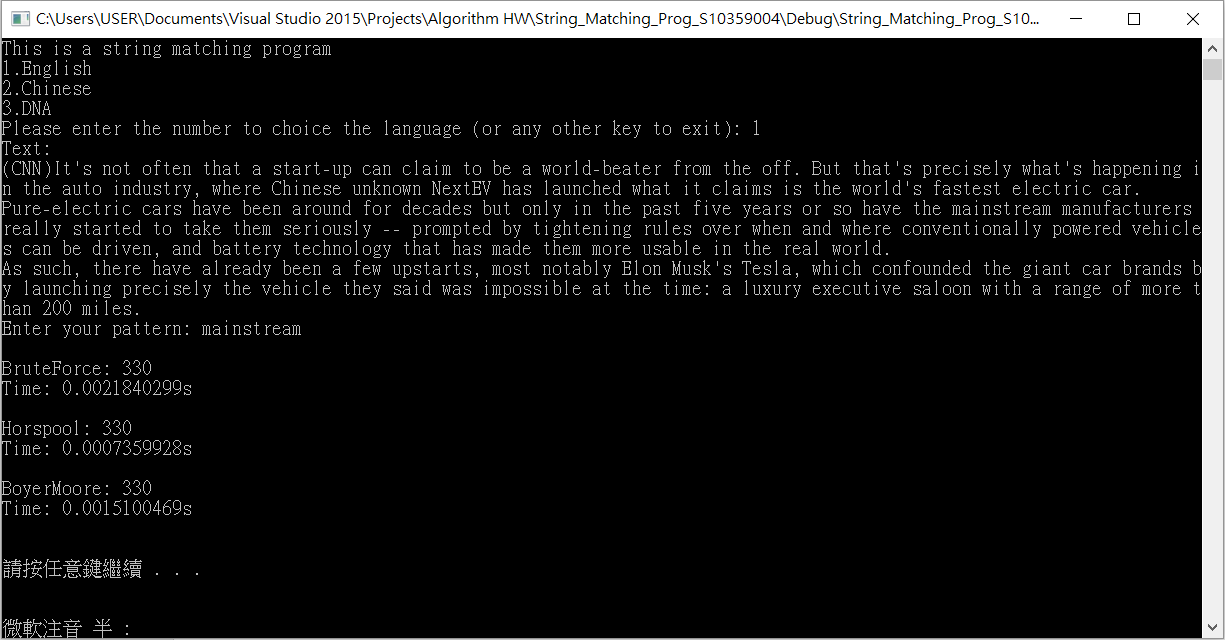
(2)輸入: 1 (選擇英文)

輸出Text



(3)輸入pattern: mainstream

輸出三種演算法計算後的結果



2. 程式限制

(1) 英文text可為半形英文字、標點符號及空格，英文pattern限制輸入字型為半形英文字，盡量不要輸入標點符號或空白，容易造成程式錯誤。

(2) 中文text及pattern限制為全型中文字及標點符號，輸入半形字會導致程式錯誤。

(3) DNA比對text及pattern都必須為半型英文字母A、T、C、G，若有其他輸入則會造成程式錯誤。

3. 心得

不知不覺已經來到第三次作業了，感覺程式碼似乎越來越長了，這次的作業真的是苦工題，三種輸入三個演算法總共要寫9個函式，不過在下還是咬著消夜盡力的趕出來了。但在下相信自己又朝著演算法大師之路再稍微往前邁進一點了，期待下次作業能夠再創巔峰，謝謝各位。

**參考文獻**

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, "Introduction to Algorithms," Second Edition, The MIT Press, 2001.
2. Anany V. Levitin, "Introduction to the Design and Analysis of Algorithms," 3nd Edition, Addison Wesley, 2012.
3. http://www-igm.univ-mlv.fr/~lecroq/string/node1.html