國立臺南大學資訊工程學系

資工三「演算法」課程

第一次作業

**題目: Fibonacci Number**

|  |  |  |
| --- | --- | --- |
| 班級 | ： | 資工三 |
| 姓名 | ： | 林晉昇 |
| 學號 | ： | S10359004 |

老師：陳宗禧

中華民國 105年10月16日

# 目錄

1. **簡介及問題描述……….……………..…………………………………………1**
   1. **簡介…………….…………………………………………………………………………3**
   2. **問題………………………….……………………………………………………………3**
2. **理論分析與演算法則及複雜度運算………………………………………4**
3. **程式設計環境架構.………………………..…………………………………10**
4. **程式.…………………………………………..………………………………11**
5. **執行結果、討論與心得.………………………..……………………………12**

參考文獻………………………………………………………….…………………14

**(一) 簡介及問題描述**

設計與實作計算Fibonacci Number，理論驗證與實驗分析該問題!

1. 簡介

求Fibonacci Number(*Fn*)! 首先定義 *Fn*為…實作與分析下列演算法in Subsection 2.5：

2. 問題

實作與分析下列演算法in Subsection 2.5：

求Fibonacci Number(*Fn*)，實作與分析下列演算法in Subsection 2.5：

* + 1. Recursive algorithm
    2. Non-recursive definition-based algorithm
    3. Explicit formula algorithm
    4. Sequential algorithm based on the following formula:



* + 1. Logarithmic algorithm based on the following formula:



**(二) 理論分析與演算法則及複雜度計算**

1. Recursive Algorithm

//Input: A positive integer n

//Output: The n-th Fibonacci number

Fib (n)

{

If (n <= 1)

Return (n);

Else

Return Fib(n-1)+Fib(n-2);

}

* **時間複雜度**

假設每跑一次遞迴函式會執行c次基本運算，

可以得到遞迴函式的recurrence relation: 。

由於我們要求的時間複雜度O是upper bound，因為T(n-2)T(n-1)，所以我們將T(n-2)用T(n-1)替代，得到新的recurrence relation: 。

接著我們使用method of backward substitutions 來計算T(n):

T(n) = 2T(n-1) + c

= 2 ( 2T(n-2) + c ) + c

= 2 ( 2 ( 2T(n-3) + c) + c) + c

=T(n-k) + c(1 + 2 + 4 + 8 …… + )

=T(n-k) + c(- 1)

=T(0) + c(- 1)

=(c+1) – c

最後我們可以得到費式數列遞迴函式的時間複雜度為O()。

* 空間複雜度

由費式數列遞迴函式的recurrence tree來看，假設n=5，則函式f(5)會呼叫f(4)，先呼叫f(4)，接著f(4)會呼叫f(3)，f(3)呼叫f(2)，f(2)呼叫f(1)，f(1)是初始條件回傳1給f(2)，因此遞迴往回走，f(1)的記憶體空間釋出，由f(0)取代，也是初始條件因此也是回傳1給f(2)，以此類推。

下圖即是最大可能記憶體空間的使用，以f(5)為條件下是占用5個記憶體空間，因為f(4)、f(3)或更小所使用的記憶體空間都不會超過f(5)。以此類推當n=k的時候，最大使用的記憶體空間為k個。

因此我們可以知道費式數列遞迴函式的空間複雜度為P(n)。

Memory stack

|  |
| --- |
|  |
|  |
|  |
| f(1) |
| f(2) |
| f(3) |
| f(4) |
| f(5) |
| main() |

2. Non-Recursive Algorithm

Fib (n)

{

If (n == 0)

Return (0);

If (n == 1)

Return (1);

Fib ← 1;

X ← 0;

For i←2 to n do

Temp = Fib;

Fib = Fib + X;

X = Temp;

Return (Fib);

}

* 時間複雜度

執行的過程只有一個迴圈，假設迴圈內執行的基本運算有c個，從i=2到i=n總共執行了n-1次迴圈的內容，所以總共執行了c(n-1)次。因此我們可以得知使用迭代法求費式數列第n項值得時間複雜度為O(n)。

* 空間複雜度

由演算法可知不論n的值為何，所使用的記憶體空間都是相同的，因此迭代法的空間複雜度為P(1)。

3. Explicit formula

藉由課本510頁所導出的公式F(n) = -，可以直接得到以下演算法。

Fib (n)

{

Return ((1/)\*[(1+)/2]^n - (1/)\*[(1-)/2]^n);

}

* 時間複雜度

由演算法可知不論n的值為何，所執行的運算量都是相同的，所以使用公式解求費式數列第n項的函式的時間複雜度為O(1)。

* 空間複雜度

由演算法可知不論n的值為何，所使用的記憶體空間都是相同的，因此公式解的空間複雜度為P(1)。

4. Sequential algorithm for computing *Fn* matrix

理論分析:

= , for n1

利用數學歸納法證明上述方程式成立

1. 首先驗證當n=1時，左式==，右式=，等式成立
2. 接著假設當n=k時等式成立，那麼當n=k+1時，左式====，

右式=，因此當n=k+1時等式亦成立

1. 故由數學歸納法得證， = , for n1

演算法:

Fib (n)

{

array A[0, 1, 1, 1];

For i←1 to n

a0 = A[1];

a1 = A[0]+A[1];

a2 = A[3];

a3 = A[2]+A[3];

return a0;

}

* 時間複雜度

雖然是矩陣的運算，但是變成類似寫成hard code的方法變成公式解後，運算的過程也變得簡單許多。跟迭代法一樣只有一個迴圈，從i=1到i=n，總共執行了n次迴圈，每次迴圈執行的計算量都相同，因此可得知時間複雜度為O(n)。

* 空間複雜度

由演算法可得知不論n的值為何，所使用的記憶體空間都相同，因此可得到空間複雜度為P(1)。

5. Logarithmic algorithm for computing *Fn* matrix

Fib (n)

{

While (true)

If n = 1 do

break;

Else if n is even do

n n / 2;

count count + 1;

Else if n is odd do

n n – 1;

n n / 2;

temp[count] 1;

count count + 1;

a0 0; a1 1; a2 1; a3 1;

While count 0

b[4] { a0, a1, a2, a3 };

a0 b[0] \* b[0] + b[1] \* b[2];

a1 b[0] \* b[1] + b[1] \* b[3];

a2 b[2] \* b[0] + b[3] \* b[2];

a3 b[2] \* b[1] + b[3] \* b[3];

if temp[count] = 1

c[4] { a0, a1, a2, a3};

a0 c[1];

a1 c[0] + c[1];

a2 c[3];

a3 c[2] + c[3];

count count + 1;

return a1;

}

* 時間複雜度

由演算法可以得知，在拆解次方項的過程中，n的值每跑完一次迴圈都會變成1/2倍，當n的值變為1的時候才結束迴圈。假設f(n)執行了T(n)次迴圈，也就是說執行了T(n)次迴圈後n的值會變成1，所以n / = 1，得到T(n) = 。

剩下的矩陣運算部分，執行次數取決於上一個拆解次方項的過程，基本上執行次數等同於T(n) = 。所以總共的執行時間大約是2T(n)也就是2。

因此可以知道時間複雜度為O()。

* 空間複雜度

不論n的值為何，所使用的記憶體空間都相同，因此空間複雜度為P(1)。

**(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 in MS Windows

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

Fn\_Prog\_S10359004.cpp

2. Input Code Format

首先輸入要使用的演算法(輸入數字1, 2, 3, 4 or 5)，或是輸入-1來結束程式。

1. Recursive 2.Nonrecursive 3.Explicit Formula 4.Sequential 5.Logarimetic

接著輸入要計算的費式數列第n項。

以下是三種範例輸入:

(1) 1 25

(2) 3 32

(3) 5 40

3. Output Code Format

以下是三種輸入所產生的輸出結果:

1. Fib(25) = 75025

Time: 0.0063938575s

(2) Fib(32) = 2178309

Time: 0.0000145402s

(3) Fib(40) = 1023341555

Time: 0.0000017106s

**(五) 執行結果、討論與心得**

1. 執行結果

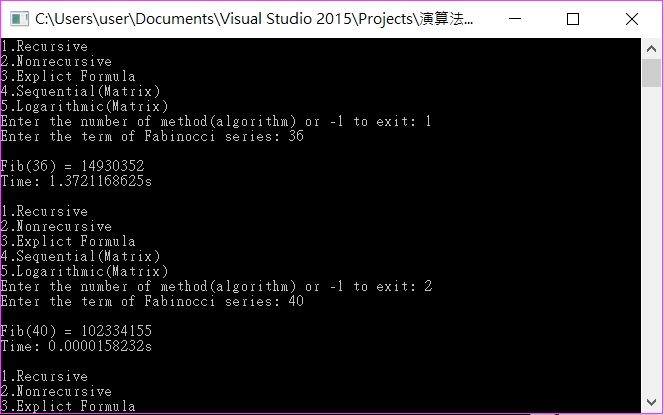
以下是程式執行過程的畫面。

第一個例子使用方法1來計算費式數列第36項的值，得到結果為:

Fib(36) = 14930352 以及執行所花費的時間為1.3721168625秒。

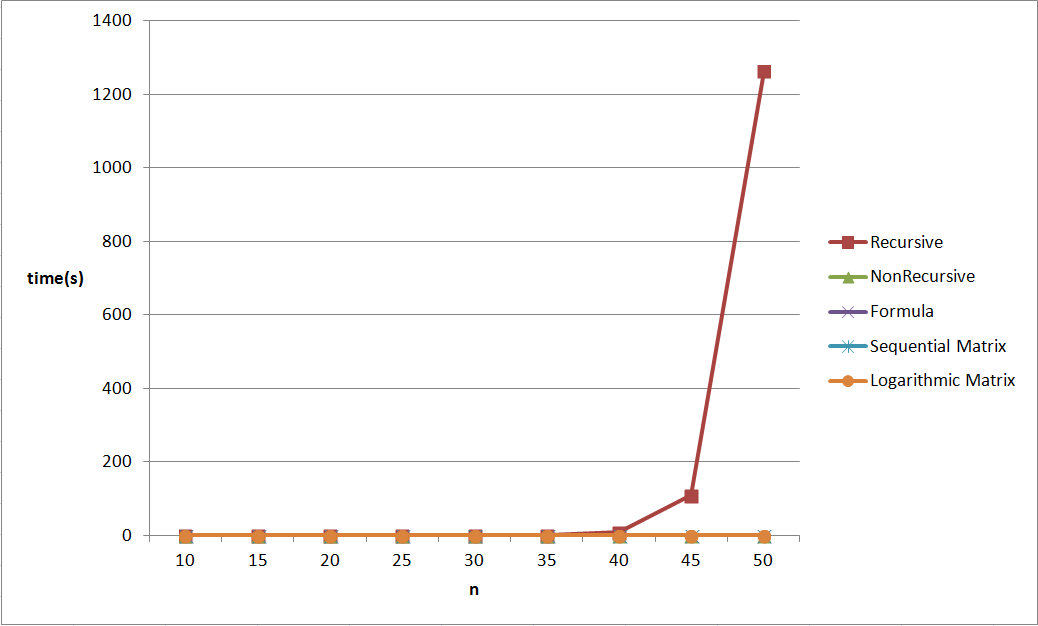
第二個例子使用方法2來計算費式數列第40項的值，得到結果為:

Fib(40) = 102334155 以及執行所花費的時間為0.0000158232秒。



2. 討論

由測試資料所產生出結果所繪成的折線圖可以觀察到，n的值從10到40的時候，五種演算法所花費的時間都差不多；但是當n從40到50的時候，遞迴演算法所花費的時間急遽上升。由這個結果我們可以知道當一個演算法的時間複雜度到達O()這樣的指數等級之後，所花費的時間會隨著n值的增加而以非常非常大的倍率增長，因此我們在撰寫程式的時候，應該盡量避免使用此種時間複雜度等級的演算法。



3. 心得

藉由這次利用五種演算法實作求費式數列第n項的程式，我對於計算演算法的時間複雜度更加的熟悉，也讓我大致理解了空間複雜度的概念，在分析程式的效能方面的能力也提升了許多，以後在程式解題的時候我應該也會更加謹慎選擇演算法。

我第一次在演算法上碰壁是在解一題找質數的題目，那時候知道的方法就只有從2~之間慢慢除，如果每一個都不能整除n的話那n就是質數，這個方法理所當然的結果是time limit exceed，後來找到別人的方法來解出這題才了解到慎選演算法的重要性。

當然這次的作業也讓我學會了如何使用excel繪製折線圖，我一直都不太會使用excel這個軟體，為了畫這張折線圖也讓我研究了好一段時間。

最後就是希望能藉著在學習演算法的過程中能提升自己程式撰寫的能力囉。

**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.