

資料結構作業 1-1

姓名: 侯旭昇

July 30, 2024

CONTENTS

1. 解題說明

2 . 演算法設計與實作

3 . 效能分析

4 . 測試與過程

CHAPTER 1

解題說明

Ackermann 函數 $A(m, n)$ 定義如下：

$$A(m, n) = \begin{cases} n + 1 & \text{如果 } m = 0 \\ A(m - 1, 1) & \text{如果 } m > 0 \text{ 且 } n = 0 \\ A(m - 1, A(m, n - 1)) & \text{如果 } m > 0 \text{ 且 } n > 0 \end{cases}$$

實現步驟：

1. 如果 m 是 0，返回 $n+1$ 。
2. 如果 m 大於 0 且 n 是 0，遞歸調用 $\text{ackermann}(m - 1, 1)$ 。
3. 否則，遞歸調用 $\text{ackermann}(m - 1, \text{ackermann}(m, n - 1))$ 。

實作參見檔案 `homework1-1.cpp`，其遞迴函式：

```
4  int ackermann(int m, int n) {  
5      if (m == 0) return n + 1;  
6      if (n == 0) return ackermann(m - 1, 1);  
7      return ackermann(m - 1, ackermann(m, n - 1));  
8  }
```

Figure 1.1: `homework1-1.cpp`

CHAPTER 2

演算法設計與實作

```
10  int main() {  
11      int m, n;  
12  
13  
14      m = 2; n = 3;  
15      cout << "Ackermann(" << m << ", " << n << ") = " << ackermann(m, n) << endl;  
16  
17      m = 3; n = 2;  
18      cout << "Ackermann(" << m << ", " << n << ") = " << ackermann(m, n) << endl;  
19  
20      m = 1; n = 5;  
21      cout << "Ackermann(" << m << ", " << n << ") = " << ackermann(m, n) << endl;  
22  
23      m = 0; n = 0;  
24      cout << "Ackermann(" << m << ", " << n << ") = " << ackermann(m, n) << endl;  
25  
26      return 0;  
27  }
```

Figure 2.1: homework1-1.cpp

CHAPTER 3

效能分析

時間複雜度

$$A(1, n) = O(n)$$

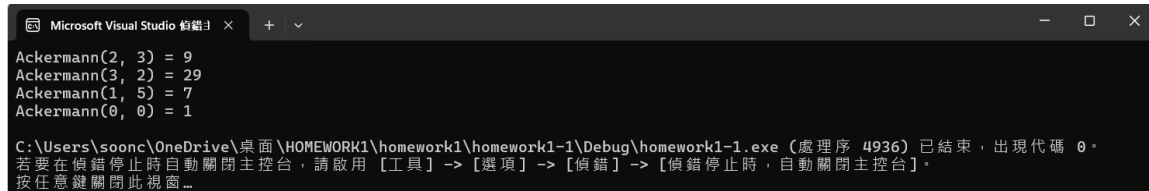
$$A(2, n) = O(2^n)$$

$$A(3, n) = O(2^{2^n})$$

$A(4, n)$ 及以上，增長速度更快

CHAPTER 4

測試與過程



```
Microsoft Visual Studio 偵錯 1 x + -
Ackermann(2, 3) = 9
Ackermann(3, 2) = 29
Ackermann(1, 5) = 7
Ackermann(0, 0) = 1
C:\Users\soonc\OneDrive\桌面\HOMEWORK1\homework1\homework1-1\Debug\homework1-1.exe (處理序 4936) 已結束，出現代碼 0。
若要在偵錯停止時自動關閉主控台，請啟用 [工具] -> [選項] -> [偵錯] -> [偵錯停止時，自動關閉主控台]。
按任意鍵關閉此視窗...
```

遞歸實現的驗證

1. $A(2, 2)$
 - $A(2, 2) = A(1, A(2, 1))$
2. 計算 $A(2, 1)$
 - $A(2, 1) = A(1, A(2, 0))$
3. 計算 $A(2, 0)$
 - $A(2, 0) = A(1, 1)$
4. 計算 $A(1, 1)$
 - $A(1, 1) = A(0, A(1, 0))$
5. 計算 $A(1, 0)$
 - $A(1, 0) = A(0, 1)$
6. 計算 $A(0, 1)$
 - $A(0, 1) = 1 + 1 = 2$
7. 回到 $A(1, 0)$
 - $A(1, 0) = 2$
8. 回到 $A(1, 1)$
 - $A(1, 1) = A(0, 2)$
 - $A(0, 2) = 2 + 1 = 3$
9. 回到 $A(2, 0)$
 - $A(2, 0) = 3$
10. 回到 $A(2, 1)$
 - $A(2, 1) = A(1, 3)$
 - $A(1, 3) = A(0, A(1, 2))$
11. 計算 $A(1, 2)$
 - $A(1, 2) = A(0, A(1, 1))$
 - $A(1, 1) = 3$
 - $A(0, 3) = 4$
12. 回到 $A(1, 3)$

- $A(1, 3)=4$
- 13. 回到 $A(2, 1)$
 - $A(2, 1)=5$
- 14. 回到 $A(2, 2)$
 - $A(2, 2)=A(1, 5)$
 - $A(1, 5)=A(0, A(1, 4))$
- 15. 計算 $A(1, 4)$
 - $A(1, 4)=A(0, A(1, 3))$
 - $A(1, 3)=5$
- 16. 回到 $A(1, 5)$
 - $A(1, 5)=6$
- 17. 回到 $A(2, 2)$
 - $A(2, 2)=7$

結論

通過逐步分析和驗證，我們可以確保遞歸實現的 Ackermann 函數在計算 $A(2, 2)A(2, 2)A(2, 2)$ 時的結果為 7。這種逐步驗證的過程可以應用於其他參數組合，以確保函數實現的正確性。

(演算法驗證及過程皆參考 GPT)