資料結構作業 1-2

姓名:侯旭昇

July 30, 2024

CONTENTS

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+1n + 1n+1。
2. 如果 m 大於 0 且 nnn 是 0，遞歸調用 ackermann(m - 1, 1)。
3. 否則，遞歸調用 ackermann(m - 1, ackermann(m, n - 1))。

實作參見檔案 homework1-1.cpp，其遞迴函式：

```
int ackermann_non_recursive(int m, int n) {
    stack<pair<int, int>> stk;
    stk.push(make_pair(m, n));

    while (!stk.empty()) {
        pair<int, int> p = stk.top();
        m = p.first;
        n = p.second;
        stk.pop();
        if (m == 0) {
            n = n + 1;
        }
        else if (n == 0) {
            stk.push(make_pair(m - 1, 1));
        }
        else {
            stk.push(make_pair(m - 1, -1)); // Mark position to continue with this m
            stk.push(make_pair(m, n - 1));
            continue;
        }
```

```
        if (!stk.empty() && stk.top().second == -1) {
            stk.pop();
            n = stk.top().first;
            stk.pop();
            stk.push(make_pair(m, n));
        }
        else if (stk.empty()) {
            return n;
        }
    }
    return n;
}
```

Figure 1.1: homework1-2.cpp

Figure 1.2: homework1-2.cpp

CHAPTER 2

演算法設計與實作

```cpp
int main() {
    int m, n;


    m = 2; n = 3;
    cout << "Ackermann(" << m << ", " << n << ") = " << ackermann(m, n) << endl;

    m = 3; n = 2;
    cout << "Ackermann(" << m << ", " << n << ") = " << ackermann(m, n) << endl;

    m = 1; n = 5;
    cout << "Ackermann(" << m << ", " << n << ") = " << ackermann(m, n) << endl;

    m = 0; n = 0;
    cout << "Ackermann(" << m << ", " << n << ") = " << ackermann(m, n) << endl;

    return 0;
}
```

Figure 2.1: homework1-2.cpp

CHAPTER 3

效能分析

時間複雜度

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

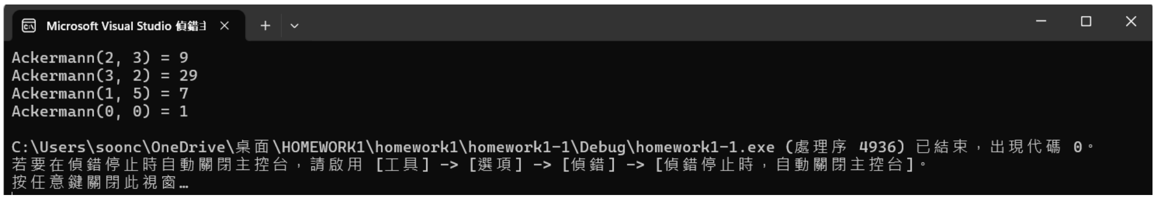
$$A(2, n) = O(2^n)$$

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

$A(4, n)$ 及以上，增長速度更快

CHAPTER 4

測試與過程



非遞歸實現的驗證

- 初始狀態：
    - 堆疊：{(2,2)}

處理 (2,2)：

- m=2,n=2
- A(2,2)=A(1,A(2,1))
- 堆疊：{(1,−1),(2,1)}

處理 (2,1)：

- m=2,n=1
- A(2,1)=A(1,A(2,0))A(2, 1) = A(1, A(2, 0))
- 堆疊：{(1,−1),(1,−1),(2,0)}

處理 (2,0)：

- m=2,n=0
- A(2,0)=A(1,1)
- 堆疊：{(1,−1),(1,−1),(1,1)}

處理 (1,1)：

- m=1,n=1
- A(1,1)=A(0,A(1,0))
- 堆疊：{(1,−1),(1,−1),(0,−1),(1,0)}

處理 (1,0)：

- m=1,n=0
- A(1,0)=A(0,1)
- 堆疊：{(1,−1),(1,−1),(0,−1),(0,1)}

處理 (0,1)：

- m=0,n=1
- A(0,1)=2
- 回到 (1,0)，將結果 2 代入
- 堆疊：{(1,−1),(1,−1),(0,−1)}

處理 (0,−1)：

- m=1,n=2m = 1, n = 2m=1,n=2
- 堆疊：{(1,−1),(1,−1)}

處理 (1,−1)：

- m=0,n=3m = 0, n = 3m=0,n=3
- 堆疊：{(1,−1)}

處理 (1,−1)

- m=1,n=3
- 堆疊：{}

處理完畢，結果 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。

(演算法設計、驗證過程皆參考 GPT)