Ackermann 遞迴與非遞迴函式報告

41243103 林采儀

2024/10/18

1. 解題說明

Ackermann 函數是一個典型的遞迴函數，其特點在於對兩個非負整數進行計算，並產生極其快速增長的結果。

其遞迴關係如下：

若 m = 0，則 A(m, n) = n + 1

若 n = 0，則 A(m, n) = A(m - 1, 1)

否則，A(m, n) = A(m - 1, A(m, n - 1))

遞迴:

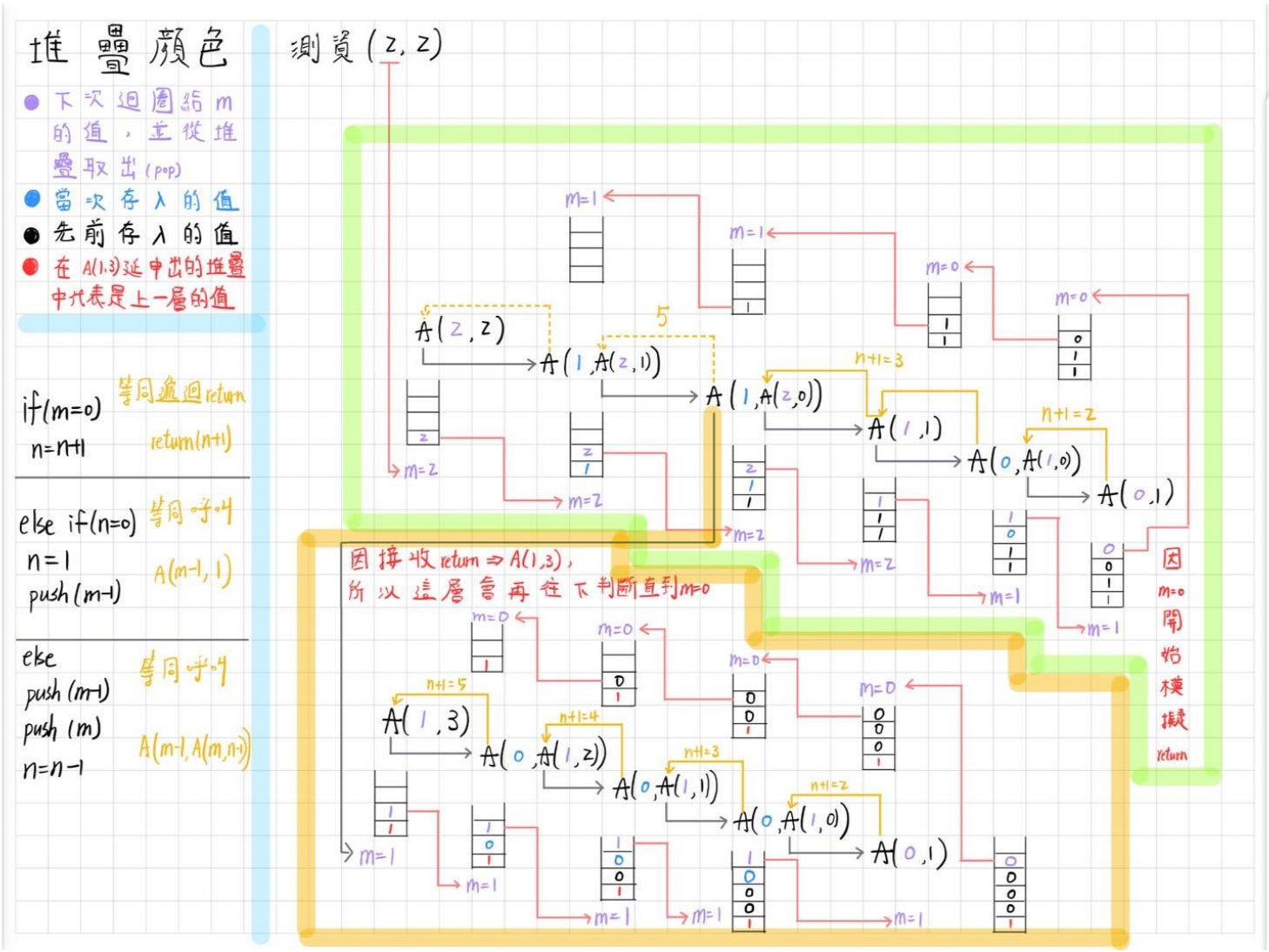
由於題目提供了遞迴關係，因此我首先從遞迴的寫法著手。遞迴方法簡單明瞭，但在計算較大值時，可能會導致堆疊溢出。

非遞迴:

參照遞迴的堆疊特性，我的非遞迴實現最初是導入 stack 函式庫，使用現成的堆疊來模擬遞迴逐層計算和返回結果的過程。由於不能使用 STL，我選擇了整數陣列來手刻模擬堆疊。

在手刻堆疊的過程中，我查了相關資料，發現許多資源都是以雙堆疊的方式呈現。由於當時的邏輯和理解程度不足，我無法掌握雙堆疊的寫法。因此，我決定使用單堆疊，將 n 視為程式中的最終答案，它會隨著計算過程不斷變動，而 m 則用來紀錄狀態，需要保留以進行判斷。

經過手寫推演的過程（後面附有手稿），我對這一邏輯有了更清晰的理解，並進一步嘗試將單堆疊的程式改為雙堆疊的實現。


堆疊顏色
下次迴圈給 m 的值，並從堆疊取出 (pop)
當次存入的值
先前存入的值
在 A(1,3) 延申出的堆疊中代表是上一層的值
if(m=0) n=n+1 等同遞迴return return(n+1)
else if(n=0) n=1 push(m-1) 等同呼叫 A(m-1, 1)
else push(m-1) push(m) n=n-1 等同呼叫 A(m-1, A(m, n-1))
測資 (2, 2)
A(2,2)
A(1,A(2,1))
A(1,A(2,0))
A(1,1)
A(0,A(1,0))
A(0,1)
5
n+1=3
n+1=2
m=2
m=1
m=0
因接收return ⇒ A(1,3)，所以這層會再往下判斷直到m=0
因m=0開始模擬return
A(1,3)
A(0,A(1,2))
A(0,A(1,1))
A(0,A(1,0))
A(0,1)
n+1=5
n+1=4

(非遞迴單堆疊手稿)

2. 實現方法

遞迴實現：利用遞迴特性，逐層深入計算，再逐步返回結果。

- 優點在於程式簡潔
- 缺點是當 m 和 n 值較大時，會導致遞迴深度過大，產生堆疊溢出。

非遞迴實現：為了解決遞迴深度問題，我寫了兩種非遞迴版本：

單堆疊版本：通過使用一個堆疊模擬遞迴過程中的每一層計算，依序處理 m 和 n 的變化。

- **堆疊存儲**：堆疊（陣列）主要用來存儲 m 值，並且在計算的每一步中，會將當前的 m 值推入堆疊。
- **m 和 n 的變化**：在每次取出 m 值進行計算後，n 的值可能會根據條件進行修改。n 值會隨著計算的進展而更新。
- **優點**：
    - **簡化實現**：只需管理一個堆疊，使得實現過程更為簡單。
    - **節省空間**：相較於使用多個堆疊，單堆疊能減少記憶體使用，尤其在 n 動態變化且無需歷史記錄時。
    - **避免堆疊溢出**：可有效避免因遞迴深度過大而導致的堆疊溢出問題。
- **缺點**：
    - **狀態管理困難**：n 的歷史值無法直接保留，需謹慎管理其變化，增加了邏輯錯誤的風險。
    - **性能限制**：在需要頻繁回溯或多個值計算的情況下，單堆疊可能導致不必要的計算負擔。
    - **狀態保存不足**：單堆疊無法同時追蹤多個變量的變化，對需要同時記錄多個狀態的複雜問題不夠靈活。

雙堆疊版本：在雙堆疊版本中，我們使用兩個堆疊來分別存儲 m 和 n 的值，這樣可以更靈活地管理計算過程中的狀態。

- **堆疊存儲：**
  - **第一個堆疊（stack_m）：**用於存儲 m 值。在每一步計算中，當需要進入下一層遞迴時，會將當前的 m 值推入此堆疊。
  - **第二個堆疊（stack_n）：**用於存儲 n 值。類似於第一個堆疊，當進入下一層遞迴時，會將當前的 n 值推入此堆疊。

- **m 和 n 的變化：**在每次從堆疊取出 m 和 n 值進行計算後，根據 Ackermann 函數的遞迴關係，n 的值會根據條件進行相應的修改。當計算完成後，可以通過第二個堆疊的值來更新結果，這樣不僅可以保持對 m 值的追蹤，還可以隨時獲取當前的 n 值。

- **優點：**
  - **清晰分離 m 和 n 的狀態**：兩個堆疊分別存儲 m 和 n 的值，使遞迴過程更清晰，數據不易混淆。
  - **減少數據錯亂**：雙堆疊能分開記錄 m 和 n 的變化，確保數據處理過程中不會發生混淆，計算更穩定。
  - **模擬遞迴結構更接近原始邏輯**：模擬過程接近遞迴本質，遞迴層次更直觀。

- **缺點：**
  - **佔用更多記憶體**：需要為 m 和 n 各自開闢空間，較單堆疊耗費更多記憶體。
  - **實現較為複雜**：操作雙堆疊邏輯繁瑣，增加程式設計複雜度。
  - **執行效能相對較低**：每次操作兩個堆疊可能導致執行效率下降，大的數據情況下更明顯。

3. 效能分析

效能測試基於輸入的不同數值組合 (m, n)，並比較各版本在同樣環境下的表現：

- **時間複雜度：**極高，增長超越 O(2^n)，具體取決於 m 和 n。

- **空間複雜度：**O(m)，非遞迴版本可能會稍微減少堆疊空間使用。

- **遞迴版本：**遞迴版本寫法簡單，但隨著輸入值的增大，遞迴深度迅速上升，導致系統超出遞迴深度限制。在測試中，遞迴最大成功處理 (3,9)，耗時 0.210857 秒。但一旦數值超過這範圍，比如 (3,10)，系統就會因為遞迴層數過多而無法計算，顯示出遞迴在處理大數據時的劣勢。

- **單堆疊非遞迴版本：**它通過一個堆疊模擬遞迴過程。測試顯示它能成功處理 (3,14)，但耗時相對較長，達到了 12.3976 秒。這種方法在較大的數據範圍內仍然能運行，不過堆疊空間需求也變大。當數值增長到 (3,15) 時，堆疊空間不足，導致計算失敗。

- **雙堆疊非遞迴版本：**雙堆疊版本理論上應該更靈活，因為它能分別管理 m 和 n 值。但實際上，這種方法內存需求更大。在測試中，雙堆疊最大成功處理的數據範圍是 (3,12)，耗時 1.25805 秒。當數據增長到 (3,13) 時，雙堆疊所需的內存超出系統負荷，無法完成計算。

4. 測試與驗證

- 遞迴版本測試結果：
  - 測資（3， 9）成功；執行時間：0.210857 秒
  - 測資（3，10）失敗（超過遞迴深度）

```
3 9
Ackermann Recursive: 4093
Execution time: 0.210857 seconds
```

- 單堆疊非遞迴測試結果(STACK_SIZE 堆疊大小對應可承受測資)：
  - 測資（3，10）成功；執行時間：0.0556813 秒
  - 測資（3，11）成功；執行時間：0.19451 秒
  - 測資（3，12）成功；執行時間：0.772036 秒
  - 測資（3，13）成功；執行時間：3.0535 秒 ；堆疊 [100000]
  - 測資（3，14）成功；執行時間：12.3976 秒；堆疊 [200000]
  - 測資（4， 1）成功；執行時間：3.05755 秒；堆疊 [100000]
  - 測資（3，15）失敗（堆疊空間不足）
  - 能正常執行最大堆疊 [250000]

```
3 10
Ackermann Single Stack: 8189
Execution time: 0.0556813 seconds
3 11
Ackermann Single Stack: 16381
Execution time: 0.19451 seconds
3 12
Ackermann Single Stack: 32765
Execution time: 0.772036 seconds
3 13
Ackermann Single Stack: 65533
Execution time: 3.0535 seconds
3 14
Ackermann Single Stack: 131069
Execution time: 12.3976 seconds
4 1
Ackermann Single Stack: 65533
Execution time: 3.05755 seconds
```

- 雙堆疊非遞迴測試結果(STACK_SIZE 堆疊大小對應可承受測資)：
  - 測資（3，10）成功；執行時間：0.0792792 秒
  - 測資（3，11）成功；執行時間：0.316088 秒
  - 測資（3，12）成功；執行時間：1.25805 秒；堆疊 [50000]
  - 測資（3，13）失敗（堆疊空間不足）
  - 能正常執行最大堆疊 [50000]

```
3 10
Ackermann Double Stack: 8189
Execution time: 0.0792792 seconds
3 11
Ackermann Double Stack: 16381
Execution time: 0.316088 seconds
3 12
Ackermann Double Stack: 32765
Execution time: 1.25805 seconds
```

5. 申論及心得

Ackermann 函數的快速增長特性讓我在這次作業中實際體驗到了遞迴與非遞迴方法的效能差異。我嘗試了不同的實現方式，並根據測試結果分析了它們的優劣。

在程式編寫中，在需要反覆執行的部分，遞迴通常是自然選擇。但這次作業讓我看到遞迴的弱點：雖然它結構簡潔，但系統的遞迴深度限制使其在處理較大數值時容易溢出。例如，遞迴版本在 (3,9) 下耗時 0.210857 秒能夠完成，但在 (3,10) 時就發生了溢出，顯示出遞迴的局限。

單堆疊非遞迴方法內存佔用小，雖然在處理 (3,14) 時耗時 12.3976 秒，但它能成功執行並避免溢出問題。然而在處理 (3,15) 時遇到了內存不足，顯示出它的極限。

雙堆疊方法理論上應該更靈活，因為能同時管理 m 和 n 的變化。但實際上，處理 (3,13) 就遇到了失敗。這是因為雙堆疊需要更多記憶體來管理遞迴過程中的多個狀態，尤其是在數據範圍較大時，內存需求比單堆疊更高。

這次實驗讓我學到了在處理不同規模的數據時，選擇合適的實現方式對效能有很大影響。單堆疊雖然簡單，但對記憶體要求更低，而雙堆疊邏輯清晰，但在大數據下內存消耗過多。這讓我更加理解內存管理的重要性，特別是在數據量增加時，如何有效利用資源變得至關重要。