**資料結構 課堂作業**

**HW1 Problem 1 & Problem 2**

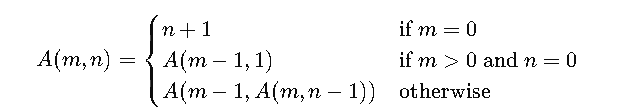
**姓名: 廖章竹**

**日期: 10/22**

**CHAPTER 1: 解題說明**

**Problem 1: Ackermann Function**

此問題要求實作 Ackermann 函數，分為遞迴 (Recursive) 及非遞迴 (Non-Recursive) 版本。  
Ackermann 函數是一個典型的遞迴函數，其定義如下：



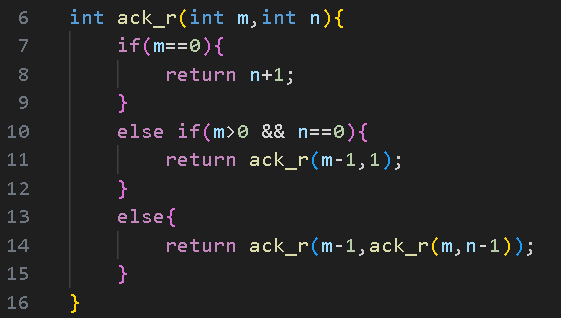
**Problem 2: Powerset**

此問題要求生成集合的冪集，利用遞迴來實作所有子集的產生過程。冪集是所有可能的子集組合，包括空集與全集。

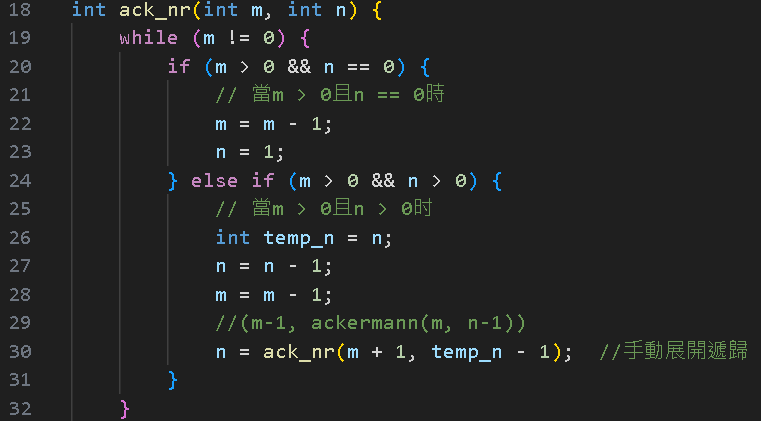
**CHAPTER 2: 演算法設計與實作**

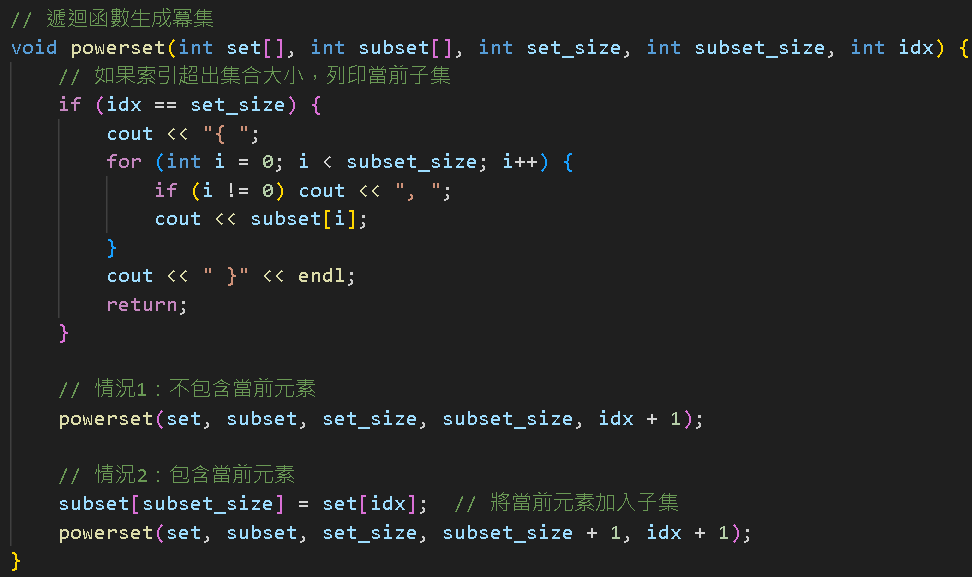
**Problem 1: Ackermann Function**

**Recursive**



Non-recursive



**Problem 2: Powerset**

**CHAPTER 3: 效能分析**

**Problem 1: Ackermann Function**

**時間複雜度:**

* 由於遞迴深度可能非常深，Ackermann 函數的時間複雜度極高，尤其在 m 增大時，時間複雜度近似於指數增長。

**空間複雜度:**

* 遞迴版本：需要依賴系統堆疊，空間複雜度為 O(m)O(m)O(m)。
* 非遞迴版本：雖然避免了系統堆疊溢出問題，但運算過程中仍需額外儲存部分狀態。

**時間複雜度:**

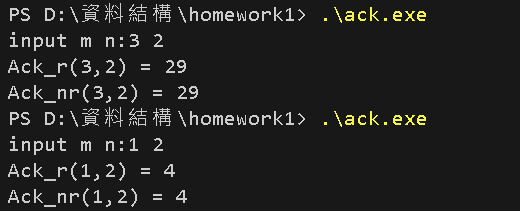
* 生成冪集的時間複雜度為 O(2n) O(2^n) O(2n)，因為每個元素有兩種狀態（包含或不包含），需要遍歷所有可能的子集。

**空間複雜度:**

* 空間複雜度為 O(n) O(n) O(n)，用於存儲當前子集。

**CHAPTER 4: 測試與過程**

**Problem 1 測試**



**Problem 2 測試**

****

**測試結果驗證：**

* **Ackermann 函數的兩種實作結果相同，驗證了非遞迴版本的正確性。**
* **Powerset 程式正確地生成了集合 {1, 2, 3} 的所有子集。**

**Ackermann 函數的實作，通過遞迴與非遞迴兩種方式進行了實現。在測試過程中，兩種實作方式的輸出結果一致，說明非遞迴版本成功模擬了遞迴運算，並有效解決了遞迴深度可能過深的問題。**

**Powerset 的生成部分，遞迴方法在處理這類問題上較直觀與便利。**