資料結構作業1-1
Ackermann函式

41243111 王家偉

Oct. 22, 2024

1.解題說明

Ackermann函式的基本規則

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

實作參見檔案
遞迴版本 Ackermann_recursion.cpp
非遞迴版本 Ackermann_nonrecursive.cpp

2.演算法設計與實作

遞迴版本:
套用遞迴規則直接實作

```
// 遞迴 Ackermann 直接套用規則
int Ackermann(int m,int n) {
    if (m==0)
        return n + 1;
    else if (n==0)
        return Ackermann(m - 1, 1);
    return Ackermann(m - 1, Ackermann(m, n - 1));
}
```

非遞迴版本:
使用自定義的堆疊類型來模擬函示展開

```
// 非遞迴 Ackermann
int Ackermann(int m, int n) {
    Stack stack;  // 使用自定義堆疊
    stack.push(m, n);

    while (!stack.Empty()) {
        m = stack.getM();  // 取得最上層 m 值
        n = stack.getN();  // 取得最上層 n 值
        stack.pop();  // 取值後消除堆疊最上層

        if (m == 0) {  // n+1, if m=0
            n += 1;
            if (!stack.Empty()) {
                int temp = stack.getM();
                stack.pop();          // 每次計算到 n+1 需要返回上一層
                stack.push(temp, n);  // m 值不變, 更新 n 的值
            }
        } else if (n == 0) {
            stack.push(m-1, 1);  // A(m-1, 1), if n=0
        } else {                 // A(m-1, A(m, n-1)), otherwise
            stack.push(m-1, -1); // 分解上式為 A(m-1, -1) 和 A(m, n-1),
            stack.push(m, n-1);  // 以模擬函式展開處理, 其中 -1 代表A(m, n-1)展開值
        }
    }

    return n;  //回傳最終值
}
```

3.效能分析

1. 時間複雜度

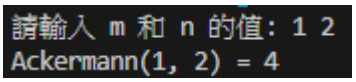
$$T(m,n)=O(Ackermann(m,n))$$

2. 空間複雜度

$$S(m,n)=O(深度)$$

4.測試與過程

1.驗證



帶入A(1, 2)
一般展開為:
A(1, 2)
= A(0, A(1, 1))
= A(0, A(0, A(1, 0)))
= A(0, A(0, A(0, 1)))
= A(0, A(0, 2))
= A(0, 3)
= 4

程式內堆疊m值n值情況:
stack = {(1, 2)}
= {(0, -1), (1, 1)}
= {(0, -1), (0, -1), (1, 0)}
= {(0, -1), (0, -1), (0, 1)}
= {(0, -1), (0, 2)}
= {(0, 3)}
= {}, 此時堆疊外n值為 4, 輸出答案 4