資料結構報告 HW 1-1

楊道恩

July 30, 2024

CONTENTS

CHAPTER 1 解題說明

以遞迴與非遞迴實作計算 Ackermann，已知定義如下：

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

實作參見檔案 ackermann.cpp，其遞迴函式：

```
//recursion
int ackermann(int m, int n) {
    if (m == 0) {
        return n + 1;
    }
    else if (n == 0) {
        return ackermann(m-1, 1);
    }
    else {
        return ackermann(m - 1, ackermann(m, n - 1));
    }
}
```

Figure 1.1: Recursive approach in ackermann.cpp

CHAPTER 2

演算法設計與實作

非遞迴:

```
//array size for non recursive algorithm
#define ARRAY_SIZE 1000
```

Figure 2.0 Define array size for non-recursive approach

```
//non recursion
int ackermann2(int m, int n) {
    int ack[ARRAY_SIZE][2];
    int top = 0;

    //initialize
    ack[top][0] = m;
    ack[top][1] = n;
    top++;

    //simulate each state
    while (top > 0) {
        //pop the top state
        top--;
        m = ack[top][0];
        n = ack[top][1];
```

Figure 2.1 Non-recursive approach in ackermann.cpp (1)

```cpp
        if (m == 0) {
            n = n + 1;
            if (top > 0) {
                //update the slot below the top
                ack[top - 1][1] = n;
            }
        }
        else if (n == 0) {
            //add (m-1,1)
            ack[top][0] = m - 1;
            ack[top][1] = 1;
            top++;
        }
        else if (n > 0 && m > 0) {
            //add (m-1, -1) and (m,n-1), where -1 is the marker
            ack[top][0] = m - 1;
            ack[top][1] = -1;
            top++;
            ack[top][0] = m;
            ack[top][1] = n - 1;
            top++;
        }
        else {
            //updates when n < 0
            n++;
            if (top > 0) {
                //update the slot below the top
                ack[top - 1][1] = n;
            }
        }
    }

    return n;
}
```

Figure 2.2 Non-recursive approach in ackermann.cpp (2)

```cpp
int main(){
    int x, y;
    cin >> x >> y;
    cout << "recursion: " << ackermann(x, y) << '\n';
    cout << "non-recursion: " << ackermann2(x, y) << '\n';

    return 0;
}
```

Figure 2.3 main section of ackermann.cpp

CHAPTER 3 效能分析

$$f(n) = O(n*ack)$$

ack 表示所需執行 ackermann 運算的次數

時間複雜度

$$T(P) = m$$

Ackermann 遞迴判斷條件為 m 之值

CHAPTER 4

測試與過程

```
$ g++ ackermann.cpp -o hw1-1.exe && ./hw1-1.exe
2 2
recursion: 7
non-recursion: 7
```

Figure 4.1: shell command

驗證

此函式遞迴終止條件為 m == 0
input: m = 2, n = 2

state	
ackermann(m-1, ackermann(m,n-1))	ackermann(1,ackermann(2,1))
ackermann(1,ackermann(1,ackermann(2,0)))	ackermann(1,ackermann(1,ackermann(1,1)))
ackermann(1,ackermann(1,ackermann(0,ackermann(1,0)))	ackermann(1,ackermann(1,ackermann(0,ackermann(0,1)))
ackermann(1,ackermann(1,ackermann(0,2))	ackermann(1,ackermann(1,3))
ackermann(1,ackermann(0,ackermann(1,2)))	ackermann(1,ackermann(0,ackermann(0,ackermann(1,1))))
ackermann(1,ackermann(0,ackermann(0,ackermann(0,ackermann(1,0)))))	ackermann(1,ackermann(0,ackermann(0,ackermann(0,ackermann(0,1)))))
ackermann(1,ackermann(0,ackermann(0,ackermann(0,2))))	ackermann(1,ackermann(0,ackermann(0,3)))
ackermann(1,ackermann(0,4))	ackermann(1,5)

重複以上求 ackermann 的步驟，可得 ackermann(2,2) = 7