

資料結構作業 1

學號:41043244

姓名:賴明賢

題目一：

Ackermann's function $A(m, n)$ is defined as follows:

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

This function is studied because it grows very fast for small values of m and n . Write a recursive function for computing this function. Then write a nonrecursive algorithm for computing Ackermann's function.

1. 解題說明：

若 $m=0$, 則 $A(m, n)=n+1$ 。

若 $m>0$ 並 $n=0$, 則 $A(m, n)=A(m-1, 1)$ 。

若 $m=0$ 並 $n>0$, 則 $A(m, n)=A(m-1, A(m, n-1))$ 。

2. 演算法設計與實作：

演算法：

```
function ack(m, n)

    while m  $\neq$  0

        if n = 0

            n := 1

        else

            n := ack(m, n-1)

        m := m - 1

    return n+1
```

遞迴：

```
int Ackermann(long long int m, long long int n) {
    if (m == 0) // m = 0 時， n + 1
        return n + 1;
    else if (n == 0) // n = 0 時， m - 1 以及 n + 1
        return Ackermann(m - 1, 1);
    else
        return Ackermann(m - 1, Ackermann(m, n - 1)); // 套用 Ackermann 函數的規則，開始遞迴
}
```

非遞迴：

```
int non_Ackermann(long long int m, long long int n) {
    long long num; // Ackermann函數的答案
    if (m == 0) return num = n + 1; // 透過Ackermann函數的建表，推斷 m 和 n 的關係式
    if (m == 1) return num = n + 2;
    if (m == 2) return num = 2 * (n + 3) - 3;
    if (m == 3) return num = pow(2, (n + 3)) - 3; // pow(2,n) = 2 ^ n
}
```

```

/*
41043244
賴明賢
*/
#include<iostream>
#include<math.h>
using namespace std;

int Ackermann(long long int m, long long int n) {
    if (m == 0) // m = 0 時， n + 1
        return n + 1;
    else if (n == 0) // n = 0 時， m - 1 以及 n + 1
        return Ackermann(m - 1, 1);
    else
        return Ackermann(m - 1, Ackermann(m, n - 1)); // 套用 Ackermann 函數的規則，開始遞迴
}

int non_Ackermann(long long int m, long long int n) {
    long long num; // Ackermann函數的答案
    if (m == 0) return num = n + 1; // 透過Ackermann函數的建表，推斷 m 和 n 的關係式
    if (m == 1) return num = n + 2;
    if (m == 2) return num = 2 * (n + 3) - 3;
    if (m == 3) return num = pow(2, (n + 3)) - 3; // pow(2,n) = 2 ^ n
}

int main() {
    long long int m, n; //宣告變數 m 和 n
    while (cout << "輸入 m 和 n: " && cin >> m >> n) { //輸入 m 和 n
        cout << "遞迴: " << Ackermann(m, n) << " " << endl;
        cout << "非遞迴: " << non_Ackermann(m, n) << endl; // 輸出
    }
}

```

3. 效能分析：

時間複雜度(遞迴)：

遞迴 Ackermann 函數的時間複雜度極高。該函數的增長速度比任何原始遞迴函數都快，可以描述為非初等的，這意味著它的增長速度比任何固定的指數堆棧都快。

對於 $m = 0$ ，時間複雜度是 $O(1)$ ，因為它僅返回 $n + 1$ 。

對於 $m = 1$ ，時間複雜度是 $O(n)$ ，因為它執行 $n + 1$ 次加法。

對於 $m = 2$ ，時間複雜度是 $O(2^n)$ ，因為它執行指數運算。

對於 $m \geq 3$ ，時間複雜度變得極高。

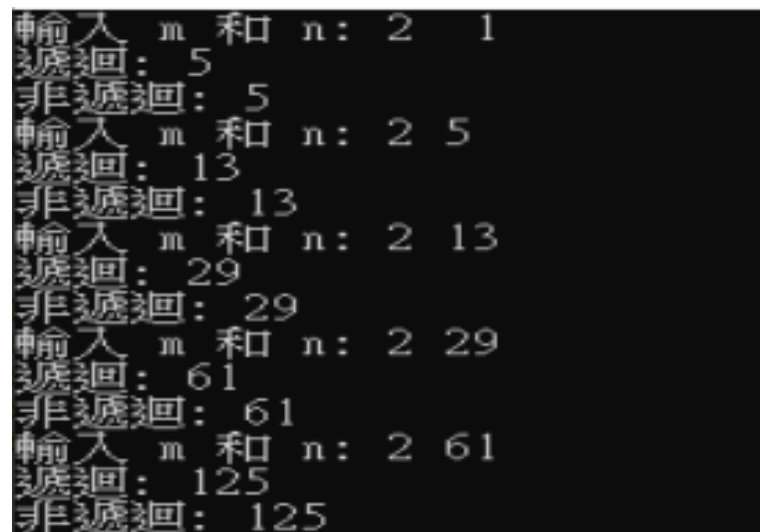
空間複雜度(遞迴)：空間複雜度也非常高，這是由於深層遞迴造成的。每次遞迴調用都需要在調用堆棧中佔用空間。對於較大的 m 和 n 值，遞迴的深度可能變得非常大，導致大量的空間需求。

時間複雜度(非遞迴): 非遞迴函數的時間複雜度要低得多，因為它基於預計算的公式直接計算結果：

- 對於 $m = 0$ ， $O(1)$
- 對於 $m = 1$ ， $O(1)$
- 對於 $m = 2$ ， $O(1)$
- 對於 $m = 3$ ， $O(\log(n))$ ，由於 `pow` 函數以對數時間計算指數。

空間複雜度(非遞迴): 空間複雜度是 $O(1)$ ，因為它只使用了固定數量的空間來進行計算，且不涉及任何遞迴或額外的數據結構。

4. 測試與過程



```
輸入 m 和 n: 2 1
遞迴: 5
非遞迴: 5
輸入 m 和 n: 2 5
遞迴: 13
非遞迴: 13
輸入 m 和 n: 2 13
遞迴: 29
非遞迴: 29
輸入 m 和 n: 2 29
遞迴: 61
非遞迴: 61
輸入 m 和 n: 2 61
遞迴: 125
非遞迴: 125
```

題目二：

If S is a set of n elements, the *powerset* of S is the set of all possible subsets of S . For example, if $S = (a, b, c)$, then $\text{powerset}(S) = \{(), (a), (b), (c), (a, b), (a, c), (b, c), (a, b, c)\}$. Write a recursive function to compute $\text{powerset}(S)$.

1. 解題說明：

生成集合的所有子集。

2. 演算法設計與實作：

如果目前 `index` 等於集合的大小 `size`，表示已經考慮了所有元素。此時，輸出目前的子集 `current` 並傳回。

遞迴處理：

不包含目前元素：呼叫 `Powerset` 函數處理下一個元素，保持 `current` 不變，即不包含目前元素。

包含目前元素：呼叫 `Powerset` 函數處理下一個元素，同時將目前元素追加到 `current` 中，形成包含目前元素的新子集。

```

#include <iostream>
using namespace std;

void Powerset(char* set, int size, int index, string current) {
    if (index == size) {
        cout << "{ " << current << "}" << endl;
        return;
    }

    // 不包含當前元素並移動到下一個
    Powerset(set, size, index + 1, current);

    // 包含當前元素並移動到下一個
    Powerset(set, size, index + 1, current + set[index] + " ");
}

int main() {
    int n = 0;
    while (true) {
        cout << "請輸入set有幾個elements:" << endl;
        cin >> n;
        if (n < 0) break;

        char* S = new char[n];
        cout << "請輸入元素:" << endl;
        for (int i = 0; i < n; i++) {
            cin >> S[i];
        }

        cout << "Powerset:" << endl;
        Powerset(S, n, 0, "");

        delete[] S;
    }

    return 0;
}

```

3. 效能分析：

時間複雜度： $O(2^n)$

空間複雜度： $O(n)$

5. 測試與過程

請輸入set有幾個elements:

5

請輸入元素:

a

b

c

d

e

Powerset:

{ }

{ e }

{ d }

{ d e }

{ c }

{ c e }

{ c d }

{ c d e }

{ b }

{ b e }

{ b d }

{ b d e }

{ b c }

{ b c e }

{ b c d }

{ b c d e }

{ a }

{ a e }

{ a d }

{ a d e }

{ a c }

{ a c e }

{ a c d }

{ a c d e }

{ a b }

{ a b e }

{ a b d }

{ a b d e }

{ a b c }

{ a b c e }

{ a b c d }

{ a b c d e }

請輸入set有幾個elements:

3

請輸入元素:

a

b

c

Powerset:

{ }

{ c }

{ b }

{ b c }

{ a }

{ a c }

{ a b }

{ a b c }

請輸入set有幾個elements:

-1