四資工二乙 張大軒 41143229

Problem1

Ackermann’s Function(阿克曼函數)

[介紹]

阿克曼函數是非原始遞迴函數的例子；它需要兩個自然數作為輸入值，輸出一個自然數。它的輸出值增長速度非常高。

定義:

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

函數值表:

$A(m, n)$ 的值

m\\n	0	1	2	3	4	n
0	1	2	3	4	5	$n+1$
1	2	3	4	5	6	$n+2$
2	3	5	7	9	11	$2\cdot(n+3)-3$
3	5	13	29	61	125	$2^{(n+3)}-3$
4	13	65533	$2^{65536}-3$	$A(3, 2^{65536}-3)$	$A(3, A(4, 3))$	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}} -3$ （n+3個數字2）
5	65533	$A(4, 65533)$	$A(4, A(5, 1))$	$A(4, A(5, 2))$	$A(4, A(5, 3))$	
6	$A(5, 1)$	$A(5, A(5, 1))$	$A(5, A(6, 1))$	$A(5, A(6, 2))$	$A(5, A(6, 3))$	

[程式實作]

```cpp
#include <iostream>
using namespace std;

// 阿克曼函數迴圈
int ackermann(int m, int n) {
    if (m == 0) {
        return n + 1;
    }
    else if (m > 0 && n == 0) {
        return ackermann(m - 1, 1);
    }
    else if (m > 0 && n > 0) {
        return ackermann(m - 1, ackermann(m, n - 1));
    }
    return 0;
}
//主程式
int main() {
    int m, n;
    cout << "輸入m與n的值 ";
    cin >> m >> n;
    int result = ackermann(m, n);
    cout << "Ackermann(" << m << ", " << n << ") = " << result << endl;

    return 0;
}
```

[程式執行畫面]

```
Microsoft Visual Studio 偵錯主
輸入m與n的值 1 3
Ackermann(1, 3) = 5

C:\Users\DaXuan\source\repos\Ack
若要在偵錯停止時自動關閉主控台，
按任意鍵關閉此視窗…
```

[時間複雜度]

阿克曼函數的時間複雜度會隨著 m 與 n 的值增加，變得非常高。

例如:

當 m = 0，O(1)。

當 m = 1，O(n)。

當 m = 2，O(2^n)。

當 m = 3，O(2^2^n)。

因此此函數的時間複雜度極高，無法用常見的大 O 記號來描述。

[空間複雜度]

此函數的空間複雜度也會隨者 m 與 n 的值增加，而迅速增加。

例如:

當 m = 0，O(1)。

當 m = 1，O(n)。

當 m = 2，O(n)。

當 m = 3，O(n)。

Problem2

[介紹]

S 是 n 個元素的集合，powerset(S)是所有 S 可能的子集合的冪集合。

例如:S=(a,b,c)，則 powerset(s)= {(), (a), (b), (c),(a,b), (a,c), (b,c), (a,b,c)}。

[程式實作]

```
#include <iostream>
#include <vector>
using namespace std;

void generatePowerset(const vector<int>& S, vector<vector<int>>& powerset, vector<int>& subset, int index) {
    if (index == S.size()) {
        powerset.push_back(subset);
        return;
    }

    // 不包含當前元素
    generatePowerset(S, powerset, subset, index + 1);

    // 包含當前元素
    subset.push_back(S[index]);
    generatePowerset(S, powerset, subset, index + 1);

    // 回溯，移除當前元素以進行下一次迭代
    subset.pop_back();
}


/*  初始化一個空的冪集和子集，
    調用 generatePowerset 開始遞歸。   */

vector<vector<int>> powerset(const vector<int>& S) {
    vector<vector<int>> powerset;
    vector<int> subset;
    generatePowerset(S, powerset, subset, 0);
    return powerset;
}
```

```cpp
//主程式
int main() {
    vector<int> S = { 4, 5, 6 }; // 定義示範集合
    vector<vector<int>> result = powerset(S);

    cout << "{ ";
    for (const auto& subset : result) {
        cout << "{";
        for (int elem : subset) {
            cout << elem << " ";
        }
        cout << "} ";
    }
    cout << "}" << endl;

    return 0;
}
```

[程式執行畫面]

```
Microsoft Visual Studio 偵錯主
{ {} {6 } {5 } {5 6 } {4 } {4 6 } {4 5 } {4 5 6 } }

D:\code\problem2\problem2\Debug\problem2.exe (處理序
若要在偵錯停止時自動關閉主控台，請啟用 [工具] -> [選
按任意鍵關閉此視窗…
```

[時間複雜度]

遞歸生成冪集時，每個元素都有兩種選擇：包含或不包含。因此，對於包含 n 個元素的集合，總共有 2^n個子集。

時間複雜度=O(2^n)

[空間複雜度]

空間複雜度包括遞歸調用堆疊和存儲所有子集所需的空間。

遞歸調用堆疊:

- 最大遞歸深度為 n
- 每層遞歸調用需要常數空間
- 因此遞歸調用堆疊的空間複雜度為 O(n)

存儲子集:

- 每個子集平均大小為 O(n)，因為每個元素最多可以出現在所有子集中
- 有 2^n 個子集
- 因此存儲所有子集所需的空間為 O(n*2^n)

結合上述，空間複雜度為 O(n*2^n)。

[心得]

這次實作讓我知道阿克曼函數是一個看似簡單的遞歸定義，卻能造成極高的時間與空間複雜度，這也是為何此函數在計算理論中常被用來測試系統性能和理

解遞歸行為的原因。而後面的 problem2，讓我更加清楚遞迴的原理以及其用法，能讓我在未來 coding 的路上更輕鬆。