Ackerman 函式

1.解題說明

阿克曼函數(迴圈寫法)

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

當 m = 0 return n + 1

當 n = 0 把 m – 1 並把 n = 1

如果 n > 0 則先用 ackerman 這個函式，然後再把 m – 1

阿克曼函數(遞迴寫法)

```
int ackerman(int m, int n) {

    if (m == 0)
        return n + 1;

    else if (n == 0) {
        return ackerman(m - 1, 1);
    }
    else
        return ackerman(m - 1, ackerman(m, n - 1));

}
```

如果 m = 0 傳回 n + 1
如果 n = 0 且 m > 0 遞迴 ackerman(m – 1 , 1)
當 n > 0 且 m > 0 遞迴 ackerman(m , n -1)然後把結果再傳入 ackerman(m – 1 , n)

2.效能分析

阿克曼函數會因為遞迴深度和重複計算導致效能瓶頸，特別是在 m 較大時，函數的遞迴次數會非常快速的增加，會導致記憶體溢位。
時間複雜度:
當 m = 0，O(1)
當 m = 1，O(n)
當 m = 2，$O(2^n)$
當 m = 3，$O(2^{2^n})$
當 m = 4，$O(2^{2^{2^n}})$
m 如果越來越大時間複雜度會以極快的速度增加。

3.測試和驗證

輸入:

```
輸入m n
2 3
```

輸出:

```
9
```

4.心得

如果想要讓程式更不容易溢位，可以將已經計算過的值存在記憶體裡，避免重複計算，可以提升程式的效能。

Powerset

1.解題說明

```cpp
void printSubset(string S, int bits) {
    cout << "{";
    cout << "[";
    for (int i = 0; i < S.length(); ++i) {
        // 如果 bits 的第 i 位是 1，則輸出 S 中的第 i 個元素
        if (bits & (1 << i)) {
            cout << S[i] << "";
        }
    }
    cout << "]";
    cout << "}";
}
```

```cpp
void powerset(string S, int x) {
    int n = x;
    int totalSubsets = pow(2, n);


    for (int bits = 0; bits < totalSubsets; ++bits) {
        printSubset(S, bits);
    }// 從 0 到 2^n - 1 的所有數字，並輸出對應的子集
}
```

2.效能分析

時間複雜度為全部子集數量 O(2^n)乘上每個子集的時間 O(n)，所以總時間複雜度為 O(n * 2^n)。
如果元素 n 的數量越多，那麼生成子集的數量將會為指數成長，所以這種方法比較適合用在元素較少的情況下。

3.測試和驗證

輸入:

```
輸入集合大小:
3
輸入元素:
abc
```

輸出:

```
Powerset:
{[]}{[a]}{[b]}{[ab]}{[c]}{[ac]}{[bc]}{[abc]}
```

4.心得

因為目前的程式會將所有子集合輸出出來，當元素越來越多的話，那麼子集合的數量就會成指數變多，所以我們可以設定一個條件，例如只生成子集大小為 1 的子集合，這樣可以讓程式效能變快。