

HW1

Ackermann's function

1. 解題說明

利用遞迴計算出輸入的阿克曼函數，計算公式如下：

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

檔案為 ack.cpp 檔，遞迴規則

```
int ack(int m,int n){
    if(m==0) //如果m是0，回傳n=n+1;
        return n+=1;

    else if(n==0) //如果m不是0，且n是0，回傳遞迴m-1，n=1;
        return ack(m-1,1);

    else //如果都不是0，回傳遞迴m-1，n=ack(m,n-1);
        return ack(m-1,ack(m,n-1));
}
```

主程式

```
int main(){
    int m,n;
    cout<<"輸入m,n"<<endl;
    while(cin>>m>>n){ //當我有輸入時，就一直做
        cout<<"ack("<<m<<","<<n<<")="<<ack(m,n)<<endl;//輸出
    }

    return 0;
}
```

2. 效能分析

```
輸入 m, n  
3 12  
ack(3, 12)=32765
```

$m=3, n=12$ 做了 2 的 15-3 次方用了 5.52 秒

00:05.52

00:05.52

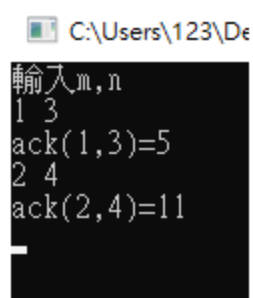


時間複雜度為 M 固定時為 $O(n^2)$

1. 若 $m = 0$ ，函數直接回傳 $n + 1$ ，時間複雜度是 $O(1)$ 。
2. 若 $m = 1$ ，時間複雜度約為 $O(n)$ 。
3. 若 $m = 2$ ，時間複雜度約為 $O(2n)$ 。
4. 若 $m = 3$ ，時間複雜度約為 $O(2^{2^n})$ ，這是指數級的成長。
5. 若 $m = 4$ ，時間複雜度約為 $O(2^{2^{2^n}})$ ，這比指數級更快，是雙重指數的成長。

3.測試與過程

測試過程



```
C:\Users\123\Desktop  
輸入 m, n  
1 3  
ack(1,3)=5  
2 4  
ack(2,4)=11  
_
```

4.心得

最簡單的遞迴寫出來的程式，當資料太大時就無法運算，還有待改進的地方，好像可以吧最好的數儲存，下次就不用再遞迴。

Powerset

1. 解題說明

用遞迴將陣列所有的值輸出成子集合

檔案為 powerset.cpp，遞迴規則

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

// 輔助函數，將子集合結果輸出
void printSubset(char subset[], int subsetSize) {
    cout << "{" << " ";
    for (int i = 0; i < subsetSize; i++) {
        cout << subset[i] << " ";
    }
    cout << "}" << endl;
}

// 使用遞迴產生所有子集合
void generateSubsets(char number[], char subset[], int index, int subsetIndex, int n) {
    if (index == n) { // 當遍歷完所有元素後，輸出當前的子集合
        printSubset(subset, subsetIndex);
        return;
    }

    // 不選擇當前元素，繼續遞迴
    generateSubsets(number, subset, index + 1, subsetIndex, n);

    // 選擇當前元素，加入到當前子集合中，然後遞迴
    subset[subsetIndex] = number[index];
    generateSubsets(number, subset, index + 1, subsetIndex + 1, n);
}
```

主程式

```

int main() {
    int n;

    // 輸入元素數量
    cout << "請輸入元素個數: ";
    cin >> n;

    char number[n]; // 根據輸入的大小來建立陣列
    char subset[n]; // 用來存放當前子集合的陣列

    // 輸入元素
    cout << "請輸入 " << n << " 個元素: ";
    for (int i = 0; i < n; i++) {
        cin >> number[i];
    }

    cout << "所有子集合: " << endl;
    generateSubsets(number, subset, 0, 0, n);

    return 0;
}

```

2.效能分析

```

請輸入元素個數: 13
請輸入 13 個元素: 1 2 3 4 5 6 7 8 9 10 11 12 13
所有子集合:
{ }
{ 1 }

```

輸出 2 的 13 次方個，耗時 15.33 秒

視頻6:35 | 1.0KB/s 已

非 4G 65%



00:15.33



時間複雜度為: $O(2^n)$ (2 的 n 次方)

3.測試過程

```
請輸入元素個數: 3
請輸入 3 個元素: 1 2 3
所有子集合:
{ }
{ 3 }
{ 2 }
{ 2 3 }
{ 1 }
{ 1 3 }
{ 1 2 }
{ 1 2 3 }
```

C:\Users\123\Desktop\HW1\scr\code\powerset.exe

請輸入元素個數: 5
請輸入 5 個元素: 1 2 3 4 5
所有子集合:

```
{ }  
{ 5 }  
{ 4 }  
{ 4 5 }  
{ 3 }  
{ 3 5 }  
{ 3 4 }  
{ 3 4 5 }  
{ 2 }  
{ 2 5 }  
{ 2 4 }  
{ 2 4 5 }  
{ 2 3 }  
{ 2 3 5 }  
{ 2 3 4 }  
{ 2 3 4 5 }  
{ 1 }  
{ 1 5 }  
{ 1 4 }  
{ 1 4 5 }  
{ 1 3 }  
{ 1 3 5 }  
{ 1 3 4 }  
{ 1 3 4 5 }  
{ 1 2 }  
{ 1 2 5 }  
{ 1 2 4 }  
{ 1 2 4 5 }  
{ 1 2 3 }  
{ 1 2 3 5 }  
{ 1 2 3 4 }  
{ 1 2 3 4 5 }
```

4.心得

要先創兩個陣列，一個大小一個子集合，再分別用兩個陣列的位置去看要不要加目前的數，並且位置加一，最後在迴圈輸出陣列