資料結構報告

姓名 ：莊笙禾

1 解題說明

2 演算法設計與實作

3 效能分析

4 測試與過程

第一題 ：解題說明

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.

對於不同的 m和 n有著不同的計算規則如下：

若m=0 ,則A(m,n)=n+1

若m>0且n=0,則A(m,n)=A(m-1,1)

若m>0且n>0,則A(m,n)=A(m-1,A(m,n-1))

例如：

當m=0時,A(0,n)=n+1,A(0,4)=5

當計算A(1,2)：

首先A(1,2)=A(0,A(1,1)),先計算 A(1,1)

接著A(1,1)=A(0,A(1,0)), A(1,0)=A(0,1)=2,所以A(1,1)=A(0,2)=3

最後A(1,2)=A(0,3)=4

第一題 ：演算法設計與實作

非遞迴版本

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

int ackermann_non_recursive(int m, int n) {
    stack<pair<int, int>> s; // 定義一個堆疊來存儲 (m, n)
    s.push(make_pair(m, n)); // 將初始的 (m, n) 推入堆疊
    while (!s.empty()) {
        m = s.top().first;
        n = s.top().second;
        s.pop();
        if (m == 0) {// 如果 m 等於 0，則根據定義 n+1
            n = n + 1;
            if (!s.empty()) {
                m = s.top().first;
                s.pop();
                s.push(make_pair(m, n));
            }
            else { // 如果堆疊為空，返回計算結果 n
                return n;
            }
        }
        else if (n == 0) {  // 如果 m > 0 且 n == 0，則將 (m-1, 1) 推入堆疊
            s.push(make_pair(m - 1, 1));
        }
        else {  // 如果 m > 0 且 n > 0，則推入 (m-1, -1) 和 (m, n-1)
            s.push(make_pair(m - 1, -1));  // 推入 m-1
            s.push(make_pair(m, n - 1));   // 推入 m 和 n-1
        }
    }
    return n;
}

int main() {
    int m, n;
    cout << "輸入 m 和 n : ";
    cin >> m >> n;
    int result = ackermann_non_recursive(m, n); // 執行阿克曼函數（非遞迴版本）
    cout << "Ackermann 非遞迴答案(" << m << ", " << n << ") = " << result << endl;
    return 0;
}
```

第一題 ：演算法設計與實作

遞迴版本

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

int ackermann(int m, int n) {
    if (m == 0) {
        return n + 1;
    }
    else if (m > 0 && n == 0) {
        return ackermann(m - 1, 1);
    }
    else {
        return ackermann(m - 1, ackermann(m, n - 1));
    }
}

int main() {
    int m, n;
    cout << "輸入 m 和 n 的值: ";
    cin >> m >> n;
    int result = ackermann(m, n);
    cout << "Ackermann 遞迴答案 A(" << m << ", " << n << ") = " << result << endl;
    return 0;
}
```

第一題 ：效能分析

將程式加入chrono來計算執行時間

非遞迴版本

```
#include <iostream>
#include <stack>
#include <chrono>
using namespace std;
using namespace std::chrono;

int ackermann_non_recursive(int m, int n) {
    stack<pair<int, int>> s; // 定義一個堆疊來存儲 (m, n)
    s.push(make_pair(m, n)); // 將初始的 (m, n) 推入堆疊
    while (!s.empty()) {
        m = s.top().first;
        n = s.top().second;
        s.pop();
        if (m == 0) {// 如果 m 等於 0，則根據定義 n+1
            n = n + 1;
            if (!s.empty()) {
                m = s.top().first;
                s.pop();
                s.push(make_pair(m, n));
            } else {
                // 如果堆疊為空，返回計算結果 n
                return n;
            }
        }
        else if (n == 0) {  // 如果 m > 0 且 n == 0，則將 (m-1, 1) 推入堆疊
            s.push(make_pair(m - 1, 1));
        }
        else {  // 如果 m > 0 且 n > 0，則推入 (m-1, -1) 和 (m, n-1)
            s.push(make_pair(m - 1, -1));  // 推入 m-1
            s.push(make_pair(m, n - 1));   // 推入 m 和 n-1
        }
    }
    return n;
}

int main() {
    int m, n;
    cout << "輸入 m 和 n : ";
    cin >> m >> n;
    auto start = high_resolution_clock::now();  // 記錄開始時間
    int result = ackermann_non_recursive(m, n); // 執行阿克曼函數（非遞迴版本）
    auto stop = high_resolution_clock::now();   // 記錄結束時間
    auto duration = duration_cast<milliseconds>(stop - start);// 計算持續時間，並轉換為毫秒
    cout << "Ackermann 非遞迴答案(" << m << ", " << n << ") = " << result << endl;
    cout << "執行時間: " << duration.count() << " ms" << endl;
    return 0;
}
```

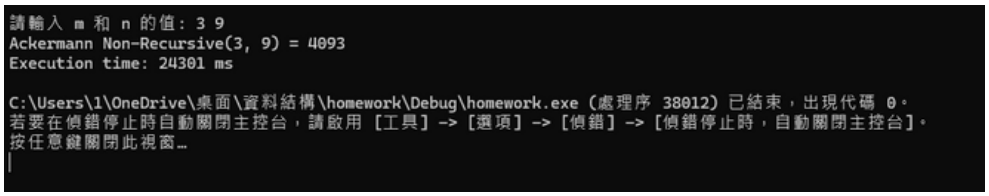
第一題 ：效能分析

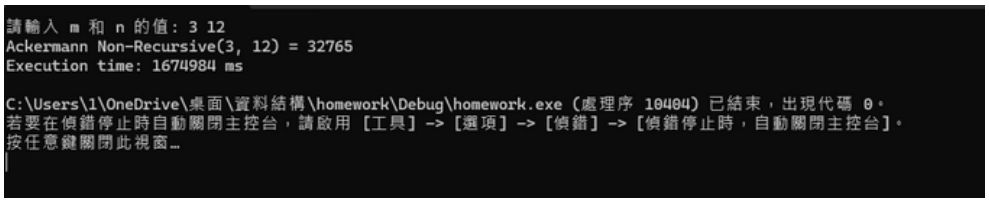
將程式加入chrono來計算執行時間

遞迴版本

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

int ackermann(int m, int n) {
    if (m == 0) {
        return n + 1;
    }
    else if (m > 0 && n == 0) {
        return ackermann(m - 1, 1);
    }
    else {
        return ackermann(m - 1, ackermann(m, n - 1));
    }
}

int main() {
    int m, n;
    cout << "輸入 m 和 n 的值: ";
    cin >> m >> n;
    auto start = high_resolution_clock::now();
    int result = ackermann(m, n);
    auto end = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(end - start);
    cout << "Ackermann 遞迴答案A(" << m << ", " << n << ") = " << result << endl;
    cout << "執行時間: " << duration.count() << "ms" << endl;

    return 0;
}
```

第一題 ：測試與過程





執行時間會依照m,n的大小增加