資料結構報告

張凱茗

2024/10/18

目錄

解題說明

Problem 1:根據題目所給的遞迴公式實作出遞迴函式(圖 1)，並將其轉為非遞迴函式(圖 2)

Problem 2:列出輸入的元素所有可能的子集合，並用二進制來控制元素是否輸出(圖 3)

```
inline int A(int m, int n)
{
    if (m == 0)
        return n + 1;
    else if (n == 0)
        return A(m - 1, 1);
    else
        return A(m - 1, A(m, n - 1));
}
```

圖 1

```
const int MAX_M = 500;  // 假設 m 的最大值
const int MAX_N = 500;  // 假設 n 的最大值

inline int A1(int m, int n) {
    // 創建一個表格來儲存中間結果，使用固定大小的二維陣列
    int table[MAX_M + 1][MAX_N + 1] = { 0 };  // 預設所有元素初始化為 0

    // 初始化表格的基礎情況
    for (int j = 0; j <= MAX_N; j++) {
        table[0][j] = j + 1;  // 當 m = 0 時，A(0, n) = n + 1
    }

    for (int i = 1; i <= MAX_M; i++) {
        table[i][0] = table[i - 1][1];  // 當 n = 0 時，A(m, 0) = A(m-1, 1)

        for (int j = 1; j <= MAX_M; j++) {
            table[i][j] = table[i - 1][table[i][j - 1]];  // 遞迴公式：A(m, n) = A(m-1, A(m, n-1))
        }
    }

    return table[m][n];
}
```

圖 2

```
void recursive(int j,int max)
{
    if(j!=max)       //判斷是否符合條件
    {
        if (total[j] && j != max - 1)        //判斷排版
            output += sorce[j] + " ";
        else
            output += sorce[j];
        recursive(j+1,max);      //遞迴
    }
}
```

圖 3

演算法設計與實作

Problem 1: 如圖 4

```
int m, n;
cout << "輸入m,n:";
while (cin >> m >> n)
{
    cout << "A(遞迴)(" << m << "," << n << ")=" << A(m, n) << endl;
    cout << "A(非遞迴)(" << m << "," << n << ")=" << A1(m, n) << endl;
    cout << "輸入m,n:";
}
```

圖 4

Problem 2:如圖 5

```
int main()
{
    int a;
    cout << "請輸入集合S的內容:";
    cin >> sorce;
    for (int i = 0; i < pow(2, sorce.length()); i++)
    {
        total = DecToBin(i);     //找出二進制值
        if (total.length())
            recursive(0, total.length());    //呼叫遞迴開始
        output += "\n";      //排版(換行)
    }
    cout << "S可能的組合為:\n" << output;
    system("pause");
}
```

圖 5

效能分析

Problem 1:

遞迴版本 A(m, n)：

時間複雜度：

當 m == 0 時，函數直接返回 n + 1，時間複雜度為 O(1)O(1)O(1)。

當 m > 0 且 n == 0 時，會調用 A(m - 1, 1)，這樣會簡化 m 的值，但遞迴深度依然隨著 m 增加。

m > 0 且 n > 0 時，會進行兩次遞迴。對於大的 m 和 n，遞迴次數和深度會變得非常大。

空間複雜度：最壞情況下，遞迴深度等於函數的計算深度，因此空間複雜度是 **超指數級**，取決於 m 和 n 的值。

動態規劃版本 A1(m, n)：

時間複雜度： O(m×n) (在此處為 O(500×500))

空間複雜度： O(m×n)

Problem 2:

時間複雜度： 主要的迴圈遍歷了2^{sorce}次，因為每個元素都有兩種狀態。在每次迴圈中，會調用 DecToBin(i)（時間複雜度為 O(｜sorce｜)）和 recursive(0, total.length())（時間複雜度也為 O(｜sorce｜)）。因此，整個迴圈的總時間複雜度為：$O(2^{\text{sorce}\times\text{sorce}})$

空間複雜度： 主要的空間使用來自於兩個部分：

1. 用來儲存所有結果的 output，大小為 $O(2^{\text{sorce}})$。
2. 每次遞迴過程中的棧空間，深度為 sorce.length()

因此，空間複雜度 S(P)為 $O(2^{\text{sorce}}+\text{sorce})$。

測試與過程

Problem 1:

```
輸入m,n:1 1
A(遞迴)(1,1)=3
A(非遞迴)(1,1)=3
```

驗證

A(1,**1**)→A(0,**A(1,0)**) →A(0,**A(0,1)**) →A(0,**2**) →3

Problem 2:

```
請輸入集合S的內容:abc
S可能的組合為:

a
ab
b
abc
bc
ac
c
```

驗證

S={a,b,c}，可能的子集合有{}{a}{b}{c}{ab}{ac}{abc}{bc}