0511105 李頤

Algorithm Homework 1 Report

A. Flow chart or Pseudo Code

i. Recursion

```
Max_Value ( area, crop_area, crop_value, crop_index )
    if crop_index < 0 or area <= 0
        return 0
    sub_area = area - crop_area[crop_index]
    if sub_area >= 0
        return max(
            Max_Value (sub_area, crop_area, crop_value, crop_index)
            + crop_value[crop_index],
            Max_Value (area, crop_area, crop_value, crop_index-1)
        )
    else
        return Max_Value (area, crop_area, crop_value, crop_index-1)
```

ii. Recursion + Memoize

```
// add table for calculated numbers
Max_Value ( area, crop_area, crop_value, crop_index, calculated )
    if crop_index < 0 or area <= 0 // base case
        return 0
    // 確認是否在 table 裡面
    if calculated[ area ][ crop_index ] return calculated[ area ][ crop_index ]
```

```
sub_area = area - crop_area[crop_index]
if sub_area >= 0
    t1 =  Max_Value ( sub_area, crop_area, crop_value, crop_index,
                      calculated )
        + crop_value[crop_index]
    t2 = Max_Value ( area, crop_area, crop_value, crop_index-1,
                     calculated)

    calculated[area][crop_index] = max( t1, t2 )
else
    calculated[area][crop_index] =
        Max_Value ( area, crop_area, crop_value, crop_index-1,
                    calculated)
    return  calculated[area][crop_index]
```

iii. Dynamic Programming

```
Max_Value ( area, crop_area, crop_value, crop_index )
    // dp table initialization
    for i to calculated.size
        calculated[i] = 0
    for i = 0  to area
        for j = 0 to crop_index
            if i - crop_area[j] >= 0
                calculated[i] = max( calculated[i],
                                     calculated[ i-crop_area[j] ]+crop_value[j] )
    return calculated[area]
```

B. Design Logic

recursion 分割問題的方式為：要與不要選擇某種作物，選擇某作物時，加上此作物價值，面積減少，往下遞迴。不選擇某作物時，面積不變，以相同面積往下遞迴，但不會再選擇到此作物。這樣可以確保每種可能性被跑到。兩種情況都跑完，比較大小，就可以得到最佳解

recursion mem 相同方式，只是多加上table，可以根據面積與作物查詢計算過的數值，這樣可以避免重複運算相同的東西，運算速度也會有所提升

dp也是用選與不選的邏輯，只是dp是從面積零逐一往上計算每一面積的最大值

C. Comparison

recursion_memoize 會比 recursion 快上許多因為可以避免進入同樣數值的遞迴。在面積小的情況下 dp 省去 function call 的時間所以比較快

case	recursion	recursion+mem	dp
1	120 ms	37 ms	6 ms
2	297739 ms	106 ms	29 ms

D. Discussion

在建構table時覺得要開一大塊memory這個方法有點沒效率，所以我有想過利用 hash map，因為查找為O(1)，而且不會像陣列一樣，可能會有很多沒用到的空格。本來用的是C++的 unordered_map，但除了工作站上面不給compile之外，自己實測速度也比array直接查找（畢竟還是需要計算hash）慢許多，所以後來還是用陣列的方式儲存計算過的數值（希望memory不要炸r）

E. Bonus

	1	1-2	1-3
recursive	120(ms)	124	122
dp	6	54	579
memo	35 (18)	139 (19)	973 (20)

	2	2-2	2-3
recursive	294405	289826	284894
dp	29	268	28513
memo	101 (70)	316 (70)	23684 (70)

recursion 為窮舉，總共有2^N種可能，每種O(N)所以複雜度為 O(N*2^N)。

dp 複雜度為 O(AN) (area, num_of_crop) ，空間複雜度為 O(A)。

case1、 case2 數值帶入可以算出與實驗值相近的比例，可驗證時間複雜度為O(AN)

另外從數據也可以看到時間複雜度與面積呈正比的現象。

memoize的方式應該不會受到面積影響，後來發現是在allocate table上花太多時間了。recursion本身時間是沒有變的（仍為70ms）。

memoize 空間複雜度為所開的陣列 所以是 O(A*N)

(II) DP Algorithm with same time complexity

我的方式為求某個作物數量的最佳解，逐一增加作物數量直到目標最佳解。以 case1 來說，只選擇一種作物，對應面積最佳解就是

area 3 4 6 10 19 21 31

value 5 7 11 20 37 39 69

選擇兩種作物時，將原本作物資料加上目前最佳解，如果遇到相同面積時視情況更新數值，這樣就可以得到選取兩個作物時，每種面積組合的最佳解。

area 3 4 6 7 8 9 10 …19 …21 …31

value 5 7 11 12 14 16 20 …37 …39 …69

選三種選四種以此類推，一直求解直到最小面積能填滿目標面積，這樣就可以得到最佳解，這個方法很慢，但不會被直接被面積影響（作物面積與目標面積比例相同時）

code在cpp裡這遍就不詳細寫了