演算法導論 HW5

110511254 徐煜絨

一、檔案差別

兩個 cpp 檔中，_multiple 表示能輸出所有最小切割次數的方法且不會重複（如(1, 9)和(9, 1)不會同時出現）；_one 則表示只能輸出一種切割方法。

二、程式架構

這次作業使用 dynamic programming，解題的方式是將問題拆分成更容易處理的子問題，在解決子問題後，就可以將答案整理並得到原本問題的解。在這次作業中，要嘗試適當分割 rod，以得到最佳利潤。比較特別的是要輸出切割次數最小的方法，所以要對講義提供的 pseudo code 稍作修改。

實作 DP 的方法有兩種，分別是 Top-down 和 Bottom-up。

Top-down 是在過程中不斷切割大問題成數個子問題，並且藉由 vector（程式中的 r，r[i]指長度 i 的最大獲利）把答案儲存下來，使在遇到重複的子問題時，可以直接從 vector 中取得，以免重複計算已經得到結果的答案。

Top-down 的實現常伴隨遞迴（如下圖 pseudo code），例如我需要知道 length=5 時可以獲得的最大值，那我就會分別計算 length=4, 3, 2, 1, 0 時的最大值，這就是遞迴的應用。這種方法適用在子問題會重複出現或 size 較小的題目中，前者是因為 top-down 允許快速提供算過的值（return r[n]），後者是因為 size 過大可能造成 stack overflow。

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
1  if r[n] ≥ 0
2      return r[n]
3  if n == 0
4      q = 0
5  else q = −∞
6      for i = 1 to n
7          q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n − i, r))
8  r[n] = q
9  return q
```

Bottom-up 則是從最小子問題逐步建構，最後得到大問題的解。其和 top-down 的差異在於 top-down 常用遞迴，但 bottom-up 常用的卻是迭代，以 for 迴圈實現，並儲存 r[i]的解，即先解出 r[1]，再依序得到 r[2], r[3], …。

兩者的時間複雜度雖然都是$\theta(n^2)$，但除非有 prune 等特殊情況，不然在時間或空間複雜度上，bottom-up 通常比 top-down 表現更好。原因包含 bottom-up 避免了遞迴，以消除和 call stack 相關的 overhead，而且迭代相較於遞迴也更 cache friendly，最後則是 bottom-up 能避免 call stack 所產生的額外空間。

三、程式碼解釋

1. 輸出所有最小切割數的組合與如何避免重複

最小切割次數的計算是用 bottom-up 的 function 做協助，這邊以 bu2 函式為例，bu1 概念相同，並且只描述與_one 檔案有差別之處。

```
else if(q == v[i]+r[j-i]){
    q = v[i]+r[j-i];
    int ori_c = c[j];
    int new_c = c[j-i] + 1;
    if(ori_c > new_c){
        s[j].clear();
        s[j].push_back(i);
        c[j] = new_c;
    }
    else if(ori_c == new_c){
        if(i <= (j/2)){
            s[j].push_back(i);
        }
    }
}
```

當 q 和 v[i]+r[j-i]相等時，代表多了一種切割方法，原本的切割次數設為 old_c，新的切割次數設為 new_c。new_c 的計算方法為，假設現在 j=5, i=2，且當 5 由 2 和 3 組成時，會得到最大值。那 new_c 就會是 2 的切割次數+1（分為 2 和 3 時要多一刀）。又設 c[0]會等於-1，這樣如果不用切割時可以得最大值，那 new_c 就會是 0。

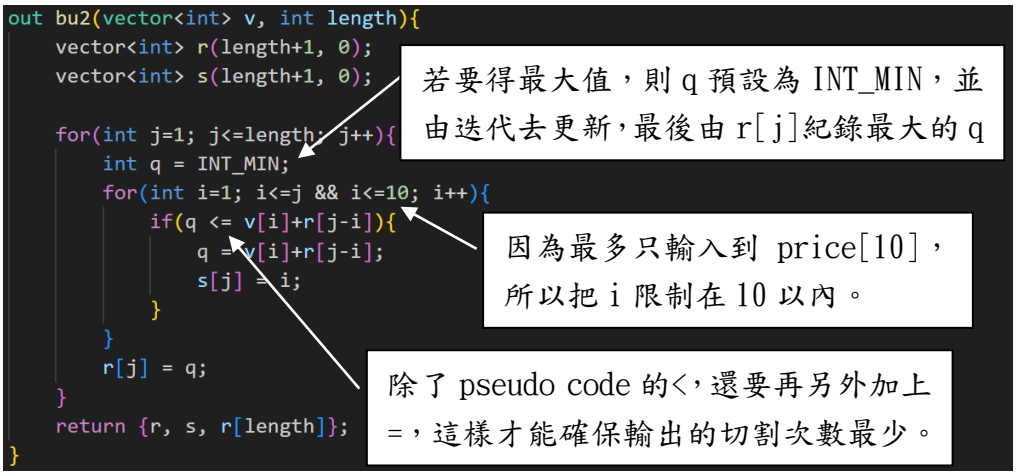
如果 ori_c>new_c，代表新的切割方法比較好（切割次數較少），此時把 s[j]更新，先 clear 再存入最新的 i；如果 ori_c==new_c，則保留 s，並加入 i，裡面的判斷式式為了不要出現重複的切割方法，如(9, 1)和(1, 9)，加上該判斷式會使切割方法剩(1, 9)。

組合各種切割方法的函式是 deal。假如今天 5 是由 2 和 3 組成，3 又是由 1 和 2 組成，那經過 deal 後就會得知 5 是由 1, 1, 2 組成。詳細解釋寫在程式碼中的註解。

然而，在經過初步處理後可能會造成重複分割方法的出現（當 length 1=0 時，或 price[i+k]<price[i]時，或 length>10 時），所以會先用 merge sort 把每一種組合由短至長做排序（如可以切成(3, 7, 5, 5)，則排序為(3, 5, 5, 7)），接著再用”compare”去剔除相同切法的（(3, 7, 5, 5)和(5, 7, 5, 3)視為同一種）。

```
void compare(vector<vector<int>> &a){
    for(int i=0; i<a.size(); i++){
        vector<int> temp = a[i];  //拿a[i]和a[k]做比較（i>=k）
        int j = i;
        while(j<a.size()){
            if(j==i){  // j必須從i開始增加，因為不保證a.size() > 1
                j++;  // 往下一個做比較
                continue;
            }
            else{
                if(a[j] == temp){  // 重複 -> 刪掉，注意j要維持
                    a.erase(a.begin() + j);
                }
                else{
                    j++;  // 不重複，往下一個比較
                }
            }
        }
    }
}
```

2. 得最大值，並確保 cut times 最小（以 bottom-up 為例）



在第三個說明上可以舉一個例子，若今天有一個範例如下：

Length	1	2	3	4	5	6	7	8	9	10
price	1	2	3	4	5	6	7	8	9	10

假設今天輸入要購買的 length 是 5，如果在判斷式中只有'<'，則會輸出的切割分法會是長度分別為 1, 1, 1, 1, 1 的 pieces，總共有 5 個 pieces、切 4 刀，但在加上'='後，輸出的切割長度則為 5，共 1 個 pieces、切 0 刀，很明顯後者的切割次數較少。

如果今天要得到最小值，則把 q 預設為 INT_MAX，並將<=改成>=。

輸出方式在下一頁。

四、輸出

1. _multiple 版本

```
Please enter the length of the rod: 30
price p1: 1
price p2: 2
price p3: 3
price p4: 4
price p5: 5
price p6: 5
price p7: 5
price p8: 9
price p9: 10
price p10: 10

Top Down:
total length: 30
maximum price: 33
All kinds of combinations that have minimum pieces and have maximum price:
(3, 9, 9, 9)
(4, 8, 9, 9)
(5, 8, 8, 9)
number of pieces: 4
minimum price: 22
All kinds of combinations that have minimum pieces and have minimum price:
(2, 7, 7, 7, 7)
number of pieces: 5
Bottom up:
total length: 30
maximum price: 33
All kinds of combinations that have minimum pieces and have maximum price:
(3, 9, 9, 9)
(4, 8, 9, 9)
(5, 8, 8, 9)
number of pieces: 4
minimum price: 22
All kinds of combinations that have minimum pieces and have minimum price:
(2, 7, 7, 7, 7)
number of pieces: 5
```

2. _one 版本

```
Please enter the length of the rod: 10
price p1: 1
price p2: 2
price p3: 3
price p4: 4
price p5: 5
price p6: 5
price p7: 5
price p8: 9
price p9: 10
price p10: 10
Top Down:
total length: 10
maximum price: 11
9 1
number of pieces: 2

minimum price: 8
7 3
number of pieces: 2

Bottom Up:
total length: 10
maximum price: 11
9 1
number of pieces: 2

minimum price: 8
7 3
number of pieces: 2
```