



聽說要會很
多 喔!!

真討厭!



推廣教育資料結構與演算法

Topic 6 動態規劃

Kuan-Teng Liao (廖冠登)

2021/06/05

大綱

概論

0-1背包問題

最長共同子
序列

動態規劃特
徵與思考方
法

動態規劃(1)

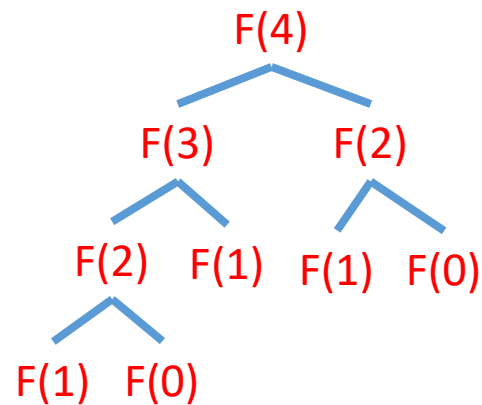
- 動態規劃是一門設計的技術，一般來說，是為了加速遞迴 (recurrence) 關係所衍生的方法。
 - ✓ 於程式面來說，可以使用遞迴 (recursion) 或
 - ✓ 迴圈或
 - ✓ 交互式函式呼叫進行實作
- Recurrence vs. Recursion
 - ✓ 都叫遞迴
 - ✓ 前一個著重在反覆出現某種行為 (e.g., 相同或交互式函式呼叫並給予不同的值)
 - ✓ 後一種著重在函式自我呼叫的程式實作方法

動態規劃(2)

- 一般來說，在討論動態規劃，基本上是討論”Recurrence”的關聯
- 為什麼要用動態規劃？
 - ✓E.g., 費式數列(fibonacci sequence)
 - $F(n) = F(n-1) + F(n-2)$, $F(0) = 0$, $F(1) = 1$

main.cpp

```
1 #include <iostream>
2
3 int fn_Fib(int i_N);
4
5 int main(){
6     int i_Num = 4;
7     std::cout << fn_Fib(i_Num);
8
9     return 0;
10 }
11
12 int fn_Fib(int i_N){
13     if(i_N == 0 || i_N == 1){
14         return i_N;
15     }
16     return fn_Fib(i_N-1) + fn_Fib(i_N-2);
17 }
```



有看到在遞迴(recurrence)的過程中，**F(2)**一直重新被算，因為來自上面的結果需要

動態規劃(2)

✓ 為了解決傳統遞迴程式實作寫法不停算同樣東西，因此**利用空間換取時間**，可以將上述結果改成以下寫法

main.cpp

```
1 #include <iostream>
2
3 void fn_Fib(int* ip_Arr, int i_N, int i_NowI);
4
5 int main(){
6     int i_Num = 4;
7     int ia_Arr[(i_Num + 1)] = {0,1};
8     fn_Fib(ia_Arr, i_Num, 2);
9     std::cout<< ia_Arr[i_Num];
10
11     return 0;
12 }
13
14 void fn_Fib(int* ip_Arr, int i_N, int i_NowI){
15     if(i_NowI <= i_N){
16         ip_Arr[i_NowI] = ip_Arr[i_NowI-1] + ip_Arr[i_NowI-2];
17         fn_Fib(ip_Arr, i_N, i_NowI + 1);
18     }
19 }
```

索引
儲存的值

0	1	2	3	4
0	1	1	2	3

有看到在遞迴(recurrence)的過程中，**F(2)**一直重新被算，因為來自上面的結果需要

動態規劃(3)

- ✓一般來說，動態規畫內所用的空間稱為搜尋表(lookup table)
- 結論，為何要使用動態規劃技術？
 - ✓利用空間儲存遞迴(recurrence)所產生的中間值，以減少運算次數
- 沒用動態規劃
 - ✓時間複雜度 $O(2^n)$
- 有用動態規劃
 - ✓時間複雜度 $O(n)$
 - 因為看到目標為止

練習6-1(1)

- 高公公最近犯下拐賣少年少女大罪被皇上抄家流放邊疆，唯一慶幸的是，高公公在山郊祖厝內偷藏了一塊 X 公克(g)重的黃金。為了變換成盤纏以便流放到邊疆能過的舒坦點，高公公欲將其 X 公克(g)的黃金熔成(或可不熔)以換取最大利益的數份(即大於等於1)，其當時黃金克數(g)對應市價如下表所示：

單位(g)	1	2	3	4	5	6	7	8
盤纏	2	5	9	10	15	16	19	26

練習6-1(2)

- 假設該黃金純度為100%且於熔開成數份時不出現任何瑕疵(即完美分離)，請問高公公應如何熔開黃金可變換最多盤纏且在熔開份數為最少份獲得最大利益？
- 輸入說明：
 - ✓ 第一列為輸入，代表高公公擁有的黃金X克數
- 輸出說明：
 - ✓ 第一列為輸出，代表高公公熔完後換取最多的盤纏值

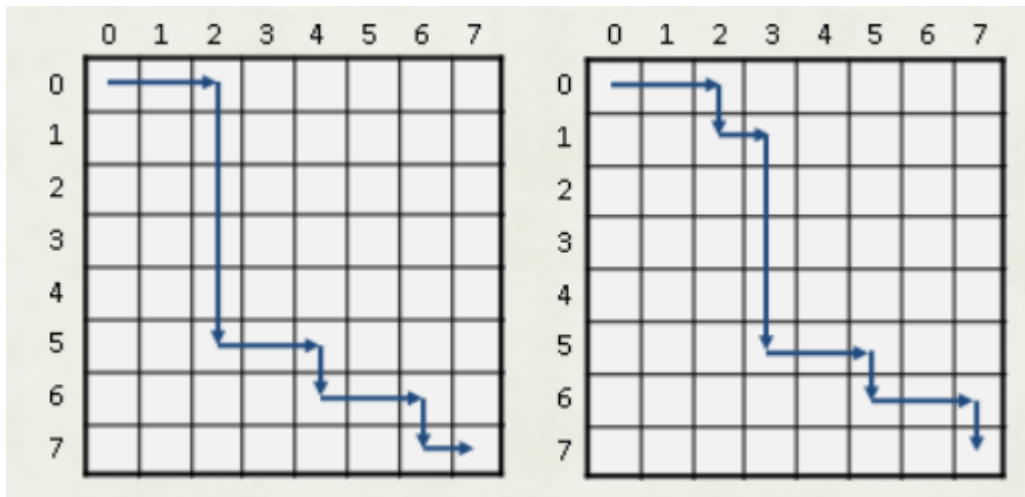
練習6-1(3)

- 輸入範例

輸入	輸出
2	5
4	11
6	18
7	20

練習6-2(1)

- 一個方格棋盤，從左上角 $(0, 0)$ 走到右下角 $(m+1, n+1)$ ，每次只能往右走一格或者往下走一格。請問有幾種走法？
- 輸入說明：
 - ✓ 第一列為輸入，代表 m 與 n 值，以造成 $m+1$ 列 $n+1$ 行大小的棋盤
- 輸出說明：
 - ✓ 輸出共有幾種方法



練習6-2(2)

- 輸入範例

輸入	輸出
2 2	6

0/1 背包問題Knapsack Problem(1)

- 0/1 Knapsack Problem

- ✓ 在有限的包包承受重量中，每種物品只會放進背包零個或一個
 - 「0/1」：一個物品要嘛整個不放進背包、要嘛整個放進背包。物品無法切割。
- ✓ 若今有以下物品要放入包包，其每樣物品重量與價錢為圖所示，請問100g的背包要如何放以下東西，其擁有最大價值。

				
Cost	400	150	150	120
Weight	85	30	50	20








- ✓ 方法

- 窮舉法(這需要講嗎？把所有組合列出來)，不過這個時間複雜度是 $O(2^n)$ ，其中 n 為物品項目

0/1 背包問題Knapsack Problem(2)

- 0/1 Knapsack Problem(cont'd)

✓若今有以下物品要放入包包，其每樣物品重量與價錢為圖所示，請問100g的背包要如何放以下東西，其擁有最大價值。

					
Cost	400	150	150	120	
Weight	85	30	50	20	100

✓方法

- 動態規劃

- 因為有四個物品，每個物品有重量及其價錢，所以除了重量與價錢外，需建立一個二維陣列(列為為物品種類，行為裝入每一克的重量)的總價值

0/1 背包問題Knapsack Problem(3)

main.cpp

```
1 #include <algorithm>
2 #include <iostream>
3
4 void fn_OLKnapsack(const int ci_IN, const int ci_TWei,
5                   int* ip_Cost, int* ip_Weight, int** ia_MCost);
6
7 int main(){
8     const int ci_IN = 4; // COM: Number of Items
9     const int ci_TWei = 100; // COM: Remainder from the bag
10    int i_Cost[ci_IN]={400, 150, 150, 120};
11    int i_Weight[ci_IN]={85, 30, 50, 20};
12
13    int** ia_MCost = new int*[(ci_IN+ 1)];
14    for(int i_Ct = 0; i_Ct <= ci_IN; i_Ct++){
15        ia_MCost[i_Ct] = new int[(ci_TWei + 1)]{};
16    }
17
18    fn_OLKnapsack(ci_IN, ci_TWei, i_Cost, i_Weight, ia_MCost);
19
20    for(int i_Ct = 0; i_Ct <= ci_IN; i_Ct++){
21        delete [] ia_MCost[i_Ct];
22    }
23    delete [] ia_MCost;
24    return 0;
25 }
```

0/1 背包問題Knapsack Problem(4)

main.cpp

```

27 void fn_OLKnapsack(const int ci_IN, const int ci_TWei,
28                   int* ip_Cost, int* ip_Weight, int** ia_MCost){
29
30     //COM: Loops for items
31     for (int i_Ct1 = 0; i_Ct1 < ci_IN; i_Ct1++){
32         //COM: Loops for weights
33         for (int i_Ct2 = 0; i_Ct2 <= ci_TWei; i_Ct2++){
34             //COM: If the weight of the bag < weight of the item
35             if (i_Ct2 - ip_Weight[i_Ct1] < 0){
36                 ia_MCost[i_Ct1+1][i_Ct2] = ia_MCost[i_Ct1][i_Ct2];
37             }
38             else{
39                 ia_MCost[i_Ct1+1][i_Ct2] = std::max(
40                     ia_MCost[i_Ct1][i_Ct2],
41                     (ia_MCost[i_Ct1][i_Ct2 - ip_Weight[i_Ct1]] +
42                      ip_Cost[i_Ct1])
43                 );
44             }
45         }
46     }
47     std::cout << "Max Value is: " << ia_MCost[ci_IN][ci_TWei];
48 }

```

```

E:\CodeWorkShop\CodeBlock\ProjectDsAlg\olKnapsack\bin\Debug\
Max Value is: 420
Process returned 0 (0x0)   execution time : 0.068 s
Press any key to continue.

```

0/1 背包問題Knapsack Problem(5)

- 沒有使用動態規劃
 - ✓時間複雜度 $O(2^n)$
- 有用動態規劃
 - ✓時間複雜度 $O(n \times w)$
- 所以當物品越多克數越少時，可以看出沒用動態規劃時間複雜度會較差

最長共同子序列(1)

- 又名 Longest Common Subsequence
- 假設今天有兩個數字串，要比較兩數字串的最長共通字串
 - ✓ E.g., 數字串：2579312與35328
 - 方法一：窮舉法，即把一數字串做組合，再去與另外數字串做比對
 - 謝謝，時間複雜度為 $O(2^n)$ ， n 為字串長度
 - 方法二：動態規劃

最長共同子序列(2)

✓E.g., 數字串：2579312與35328 (cont'd)

• 方法二：動態規劃

Index		0	1	2	3	4	5	6	7
		0	2	5	7	9	3	1	2
0	0	X 0	X 0	X 0	X 0	X 0	X 0	X 0	X 0
1	3	X 0							
2	5	X 0							
3	3	X 0							
4	2	X 0							
5	8	X 0							

最長共同子序列(3)

✓E.g., 數字串：2579312與35328 (cont'd)

- 方法二：由左至右，由上至下

Index

	0	1	2	3	4	5	6	7
	0	2	5	7	9	3	1	2
0	X 0	X 0	X 0	X 0	X 0	X 0	X 0	X 0
1	3 X 0	U 0	U 0	U 0	U 0	\ 1	-L 1	-L 1
2	5 X 0							
3	3 X 0							
4	2 X 0							
5	8 X 0							

若字元相等

- 則需記錄方向
- 個數由斜上方+1

若不相等

- 則需記錄方向
- 則需比較上面和左邊哪個數字大

最長共同子序列(4)

✓E.g., 數字串：2579312與35328 (cont'd)

- 方法二：由左至右，由上至下

Index

	0	1	2	3	4	5	6	7
	0	2	5	7	9	3	1	2
0	X 0	X 0	X 0	X 0	X 0	X 0	X 0	X 0
1	3 0	U 0	U 0	U 0	U 0	\ 1	-L 1	-L 1
2	5 0	U 0	\ 1	-L 1	-L 1	U 1	U 1	U 1
3	3 0							
4	2 0							
5	8 0							

若字元相等

- 則需記錄方向
- 個數由斜上方+1

若不相等

- 則需記錄方向
- 則需比較上面和左邊
哪個數字大

最長共同子序列(5)

✓E.g., 數字串：2579312與35328 (cont'd)

- 方法二：由左至右，由上至下

Index

		0	1	2	3	4	5	6	7
		0	2	5	7	9	3	1	2
0	0	X 0	X 0	X 0	X 0	X 0	X 0	X 0	X 0
1	3	X 0	U 0	U 0	U 0	U 0	\ 1	-L 1	-L 1
2	5	X 0	U 0	\ 1	-L 1	-L 1	U 1	U 1	U 1
3	3	X 0	U 0	U 1	U 1	U 1	\ 2	-L 2	-L 2
4	2	X 0	\ 1	U 1	U 1	U 1	U 2	-L 2	\ 3
5	8	X 0	U 1	U 1	U 1	U 1	U 2	U 2	U 3

若字元相等

- 則需記錄方向
- 個數由斜上方+1

若不相等

- 則需記錄方向
- 則需比較上面和左邊哪個數字大

最長共同子序列(6)

✓E.g., 數字串：2579312與35328 (cont'd)

- 方法二：由左至右，由上至下

Index

		0	1	2	3	4	5	6	7
		0	2	5	7	9	3	1	2
0	0	X 0	X 0	X 0	X 0	X 0	X 0	X 0	X 0
1	3	X 0	U 0	U 0	U 0	U 0	\ 1	-L 1	-L 1
2	5	X 0	U 0	\ 1	-L 1	-L 1	-L 1	-L 1	-L 1
3	3	X 0	U 0	U 1	U 1	U 1	\ 2	-L 2	-L 2
4	2	X 0	\ 1	U 1	U 1	U 1	U 2	-L 2	\ 3
5	8	X 0	U 1	U 1	U 1	U 1	U 2	U 2	U 3

開始由最後一個往後走

最長共同子序列(7)

- 方法二：時間複雜度(cont'd)
 - 為掃過整個陣列一遍 $O(n \times m)$ ， n 與 m 為別字串長度
 - 找出答案 $O(n+m)$

最長共同子序列(8)

main.cpp

```
1 #include <iostream>
2
3 void fn_LCS(int* ip_Str1 , int* ip_Str2 ,
4             int i_LenS1 , int i_LenS2 ,
5             int** ip_ConutMap , int** ip_DireMap
6             );
7 void fn_PrintLCS(int* ip_Str1 , int** ip_DireMap ,
8                  int i_IInd , int i_JInd );
```

main.cpp

```
10 int main(){
11     int i_LenS1 = 7;
12     int i_LenS2 = 5;
13     //COM: 2579312
14     int i_Str1[i_LenS1 + 1] = {0, 2, 5, 7, 9, 3, 1, 2};
15     //COM: 35328
16     int i_Str2[i_LenS2 + 1] = {0, 3, 5, 3, 2, 8};
17
18     int** ip_ConutMap = new int*[(i_LenS1 + 1)];
19     int** ip_DireMap = new int*[(i_LenS1 + 1)];
20
21     for(int i_Ct=0; i_Ct<= i_LenS1; i_Ct++){
22         ip_ConutMap[i_Ct] = new int[(i_LenS2 + 1)];
23         ip_DireMap[i_Ct] = new int[(i_LenS2 + 1)];
24     }
25
26     fn_LCS(i_Str1 , i_Str2 ,
27            i_LenS1 , i_LenS2 ,
28            ip_ConutMap , ip_DireMap );
29
30     for(int i_Ct=0; i_Ct<= i_LenS1; i_Ct++){
31         delete [] ip_ConutMap[i_Ct];
32         delete [] ip_DireMap[i_Ct];
33     }
34     delete [] ip_ConutMap;
35     delete [] ip_DireMap;
36     return 0;
37 }
```

最長共同子序列(9)

main.cpp

```

39
40 void fn_LCS(int* ip_Str1, int* ip_Str2,
41            int i_LenS1, int i_LenS2,
42            int** ip_ConutMap, int** ip_DireMap
43            ){
44
45     for (int i_Ct=0; i_Ct<=i_LenS1; i_Ct++){
46         ip_ConutMap[i_Ct][0] = 0;
47     }
48
49     for (int i_Ct=0; i_Ct<=i_LenS2; i_Ct++){
50         ip_ConutMap[0][i_Ct] = 0;
51     }
52
53     for (int i_1L=1; i_1L<=i_LenS1; i_1L++){
54         for (int i_2L=1; i_2L<=i_LenS2; i_2L++){
55             if (ip_Str1[i_1L] == ip_Str2[i_2L]){
56
57                 ip_ConutMap[i_1L][i_2L] = ip_ConutMap[i_1L-1][i_2L-1] + 1;
58                 ip_DireMap[i_1L][i_2L] = 0; //COM: Left Upper
59             }
60             else{
61                 if (ip_ConutMap[i_1L-1][i_2L] <
62                     ip_ConutMap[i_1L][i_2L-1]){
63
64                     ip_ConutMap[i_1L][i_2L] = ip_ConutMap[i_1L][i_2L-1];
65                     ip_DireMap[i_1L][i_2L] = 1; //COM: Left
66                 }
67                 else{
68                     ip_ConutMap[i_1L][i_2L] = ip_ConutMap[i_1L-1][i_2L];
69                     ip_DireMap[i_1L][i_2L] = 2; // Upper
70                 }
71             }
72         }
73     }
74 }
75 fn_PrintLCS(ip_Str1, ip_DireMap, i_LenS1, i_LenS2);
76 }

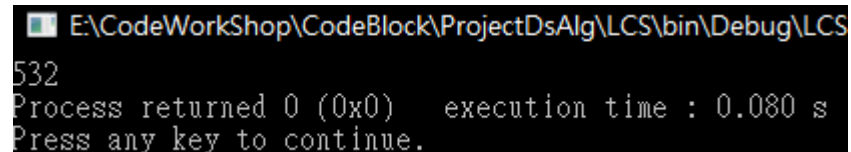
```

main.cpp

```

77 void fn_PrintLCS(int* ip_Str1, int** ip_DireMap,
78                int i_IInd, int i_JInd){
79     if (i_IInd == 0 || i_JInd == 0){
80         return;
81     }
82
83     if (ip_DireMap[i_IInd][i_JInd] == 0){//COM: Left-Upper
84         fn_PrintLCS(ip_Str1, ip_DireMap,
85                     i_IInd-1, i_JInd-1);
86
87         std::cout << ip_Str1[i_IInd];
88     }
89     else if (ip_DireMap[i_IInd][i_JInd] == 1){//COM: Left
90         fn_PrintLCS(ip_Str1, ip_DireMap,
91                     i_IInd, i_JInd-1);
92     }
93     else if (ip_DireMap[i_IInd][i_JInd] == 2){//COM: Upper
94         fn_PrintLCS(ip_Str1, ip_DireMap,
95                     i_IInd-1, i_JInd);
96     }
97 }

```



動態規劃特徵與思考方法(1)

- 特徵
 - ✓ 當要解決的問題中含有遞迴現象(recurrence)時，可將利用手動配置空間換取時間
 - ✓ 好像要窮舉所有的結果
- 思考方法
 - ✓ 分析最優解的「特徵」
 - ✓ 遞迴地定義最優解的「值」
 - ✓ 計算最優解的「值」
 - ✓ 根據計算好的資訊架構程式

動態規劃特徵與思考方法(2)

- 若以p7高公公題目為例

- ✓ 分析最優解的特徵

- 這一步的精髓是分析最優子解如何構成最優解；整個要解決的問題核心在於
 - (1)如何拆解份數，以求最大價值以及(2)如何解決若是產生多塊問題
- 想法
 - 由小塊開始，當要進一步組合出下一大塊時，將算出任意2塊小塊組合的價值是否大於1大塊，若大於，則代表該大塊最好價值應由2小塊組合而成，若價值小於，則維持該大塊價值。(即代表該大塊數字內含了可能由小塊組成或單獨1大塊組成)
 - 根據上面想法每一大塊重量可視為組合/非組合得到的最大價值

- ✓ 遞迴地定義最優解的「值」

- 畫圖思考確認每一步往上推，每個階段是否符合「分析最優解特徵的特性」

動態規劃特徵與思考方法(3)

- 若以 題目為例(cont'd)

✓計算最優解的「值」

- 公式化，若無法公式化則可以規則化
 - 假設最大價值為 $F(\cdot)$ ，假設 n 為克數所以可知

$$\begin{cases} F(1) = 1, \\ F(2) = \max(F(i) + F(2-i), F(2)), & 1 \leq i < 2 \\ \vdots \\ F(n) = \max(F(i) + F(n-i), F(n)), & 1 \leq i < n \end{cases} \quad \rightarrow \quad \begin{cases} F(1) = 1, \\ F(n) = \max(F(i) + F(n-i), F(n)), & 1 \leq i < n, n \geq 2 \end{cases}$$

✓根據計算好的資訊架構程式

- 見程式碼
 - <https://github.com/altoliaw2/DPGoldSplit>

練習6-3(1)

- ~~m 個不知死活的大學生~~在迎新郊遊要玩真心話or大冒險，並座於位置0至 $m-1$ 的上並圍成一個圈後，藉由六面骰擲出值 n 作為喊到 n 的倍數時，必須經歷真心話or大冒險的考驗，假設 $m > 6$ ，且一開始由座位0的開始喊1，並且經歷過真心話or大冒險的考驗的人需退出遊戲，請問從座位0至座位 $m-1$ 他們被排到的順序為和(從1開始算)
 - ✓輸入共一列，第一個為 m 第二個為 n 之間需隔開
 - $m\ n$
 - ✓輸出共一列，共有 m 個值，為其說真心話順序，值與值間以半形空白隔開
- 約瑟夫問題 (Josephus Problem)

練習6-3(2)

輸入	輸出
7 2	4 1 6 2 5 3 7
41 6	14 36 1 38 15 2 24 30 3 16 34 4 25 17 5 40 31 6 18 26 7 37 19 8 35 27 9 20 32 10 41 21 11 28 39 12 22 33 13 29 23

練習6-3(3)

- 提示

- ✓ 假設 $g(m, n)$ 代表 m 人組成， n 的倍數必須強迫玩遊戲的圓形結構，最後「存活者」的座位編號(座位編號是從0開始)

- ✓ 所以經過上述思考討論

- 計算最優解的「值」

- 公式化，(不細緻的寫法)

$$\begin{cases} g(1, n) = 0, \\ g(2, n) = (g(2-1, n) + n) \% m, \\ \vdots \\ g(m, n) = (g(m-1, n) + n) \% m, \end{cases}$$

練習6-4(1)

- 有幣值面額為1、3、5及7面額的硬幣無限個，今若要找錢時只許用這四種面額的幣值進行找錢，假若需要找零X元，請問最少要用幾枚硬幣，Y。

✓輸入一列

- X

✓輸出為幾枚硬幣

- Y

練習6-4(2)

輸入	輸出
15	3
18	4

練習6-5(1)

- 給予一字串集合(S)與一將要檢驗的字串(c_s)，請檢驗該將要檢驗字串中，插入若干個半形空白，是否可拆出集合中所含的字串

- ✓輸入共三列

- 第一列為集合內字串的元素個數，m
- 第二列為集合S中，每一個字串元素，共m個，每個元素之間將由半形空白隔開
- 第三列為要檢驗的字串

- ✓輸出共一列

- 若完全於集合中可以找到，請輸出TRUE；反之輸出FALSE

練習6-5(2)

輸入	輸出
12 i like sam sung Samsung mobile ice cream icecream man go mango ilikesamsung	TRUE
12 i like sam sung Samsung mobile ice cream icecream man go mango Ylikeice	FALSE