A *tabular* method!

Dynamic Programming

Chapter 15

用記憶體換速度

Mei-Chen Yeh

Divide-and-Conquer	Dynamic Programming	
Combines solutions of subproblems to solve the original problem		同
Disjoint subproblems	***Overlapping*** subproblems	異

小問題有可能是重複的，DP不會把它視為無關

Example

- Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, …

Recurrence:

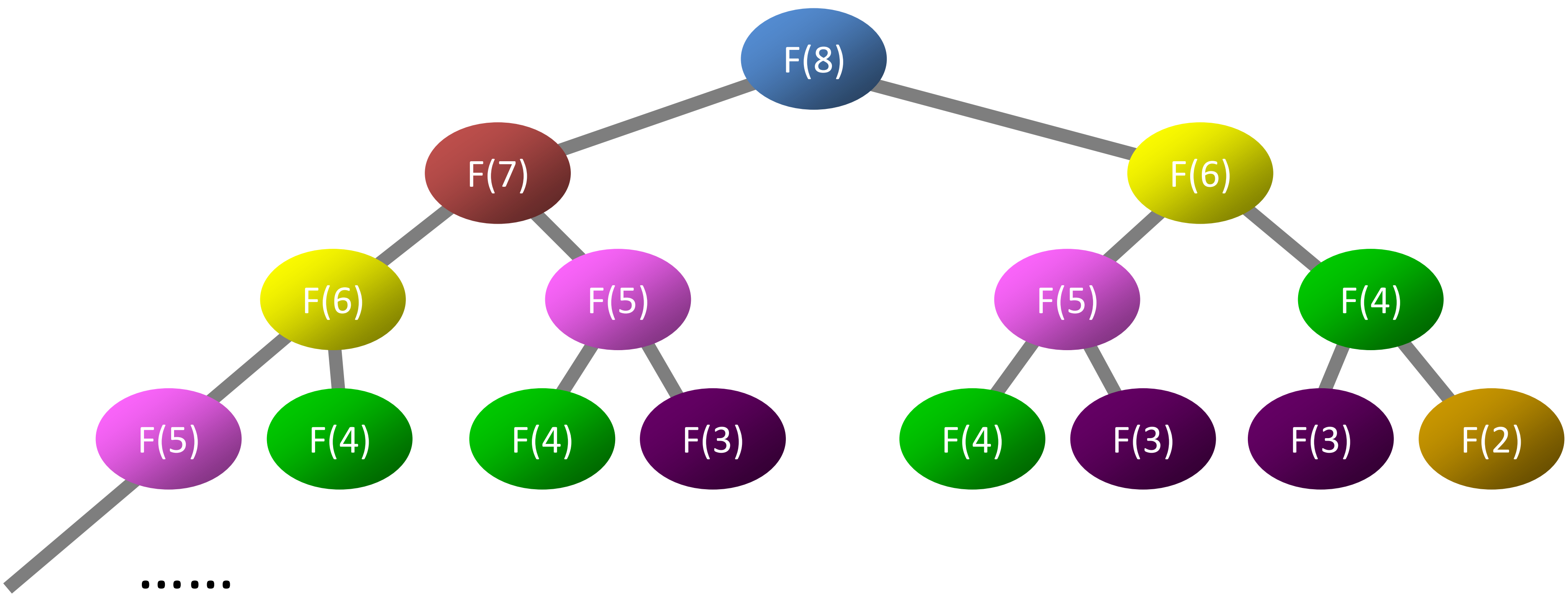
$$F(n)=F(n\text{-}1)+F(n\text{-}2) \text{ for } n>1$$

$$F(0)=0,\ F(1)=1$$

ALGORITHM $F(n)$
if $n \le 1$ **return** n
else return $F(n\text{-}1)+F(\text{n-}2)$

Inefficient!

如果用divide&conquer的話，
有很多問題會重複計算



Exponential time complexity $\Theta(2^n)$

Dynamic Programming Approach!

ALGORITHM $F(n)$

$F[0] \leftarrow 0$; $F[1] \leftarrow 1$;

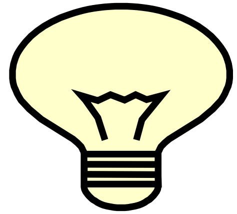
for $i \leftarrow 2$ **to** n **do** 從2跑到n，只會跑一次

$\quad F[i] \leftarrow F[i-1] + F[i-2]$;

return $F[n]$

Time: O(n)

Space: O(n)



- Footprints in the sand show where one has been

  凡走過請留下痕跡

- Use *additional memory to save computation time*

  用記憶體換速度

The rod-cutting problem

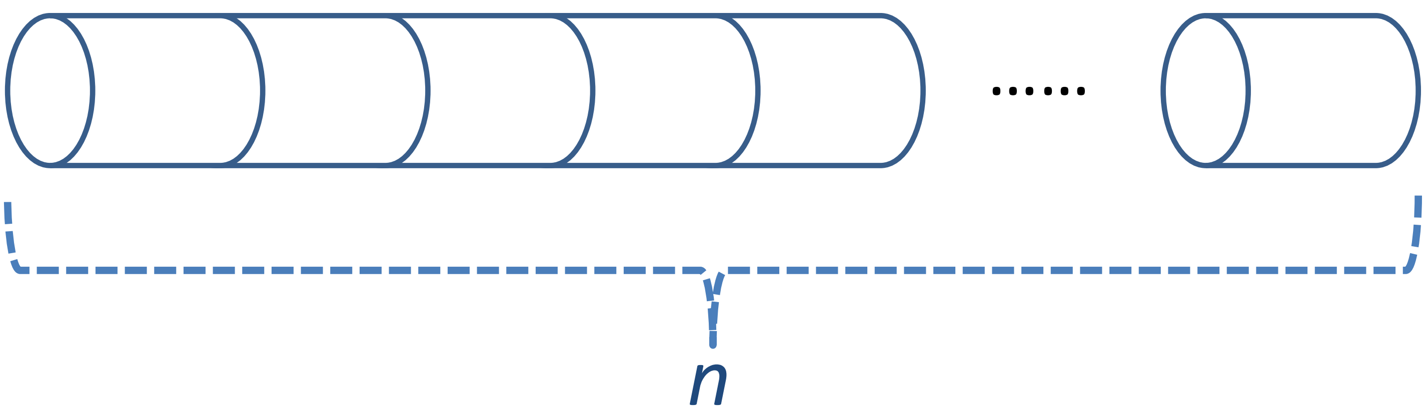
n只能切成整數

- Given a rod of length n inches and a price table, determine the maximum revenue

price table p

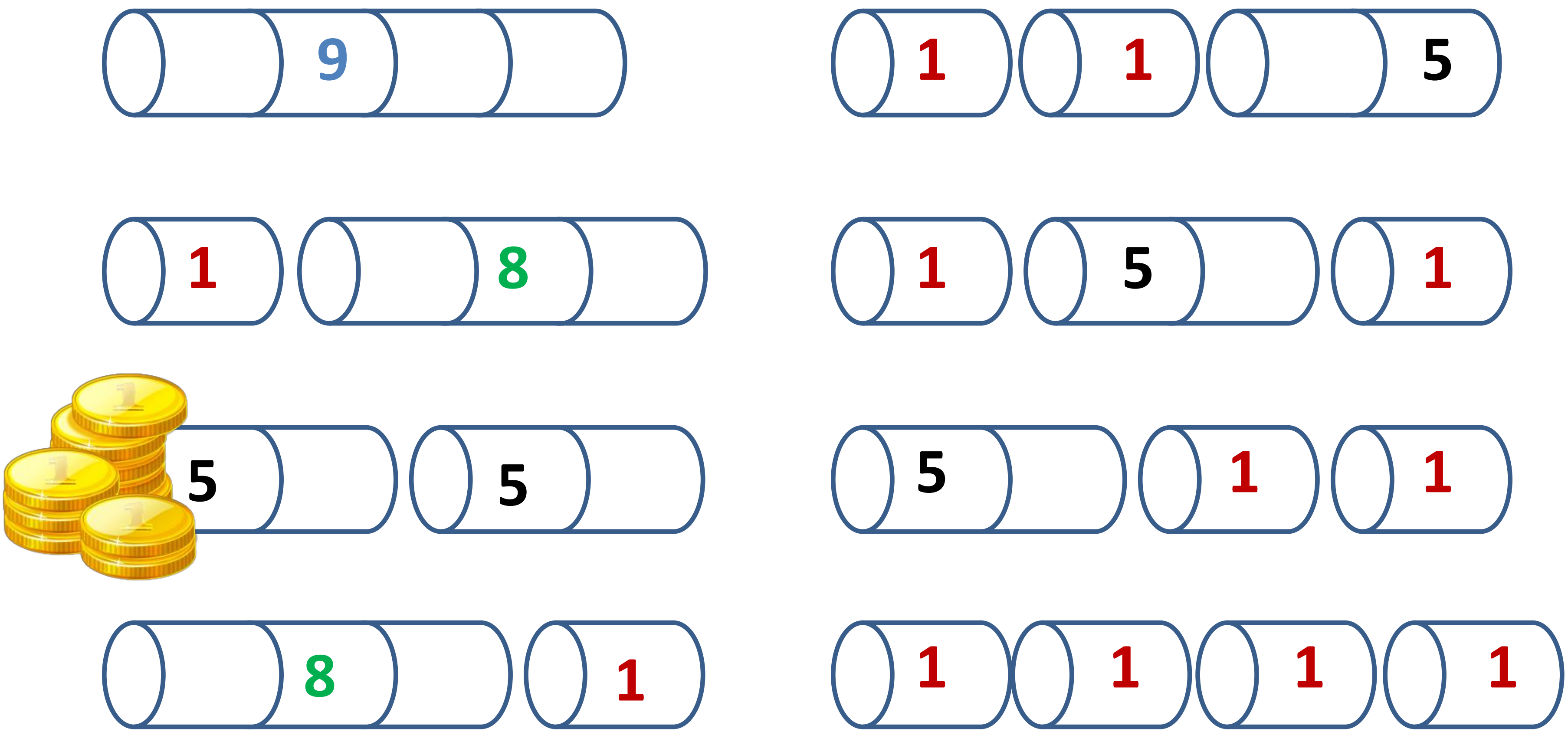
切的長度和賺的錢不是線性的

Length i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Price p_i	1	5	8	9	10	17	17	20	24	30



Example: n=4

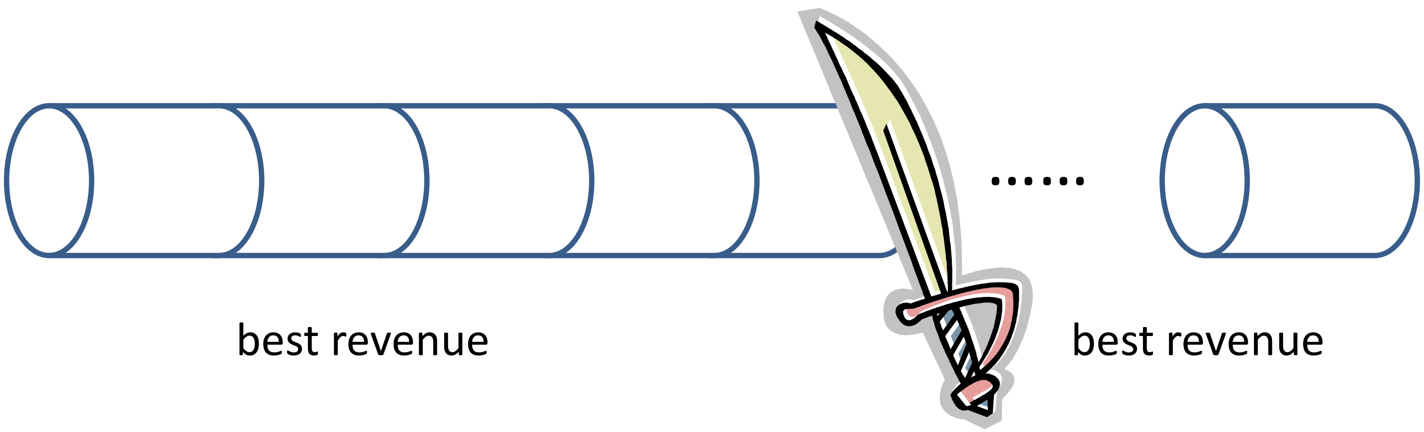
Which cut gives the maximal revenue?



Length i	1	2	3	4
Price p_i	1	5	8	9

An optimal decomposition

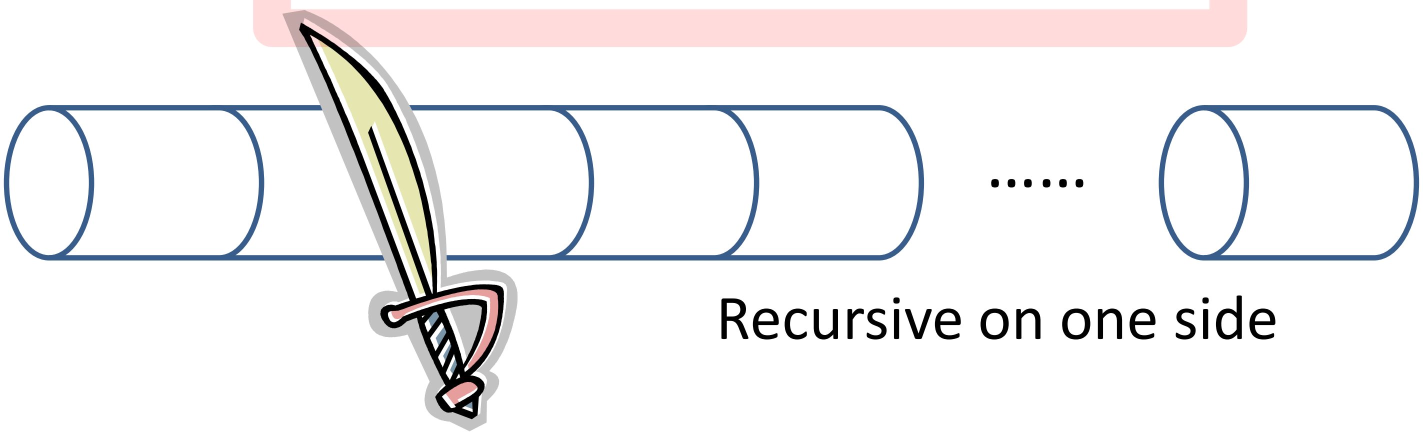
$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1)$$



A simpler way

- Cut into a piece of length *i* and a remainder of length *n-i*
- Only the remainder may be further divided

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$
$$r_0 = 0$$



A divide-and-conquer approach

price table $p[1..n]$

CUT-ROD(p, n)

rod length

1. **if** n == 0

2. **return** 0

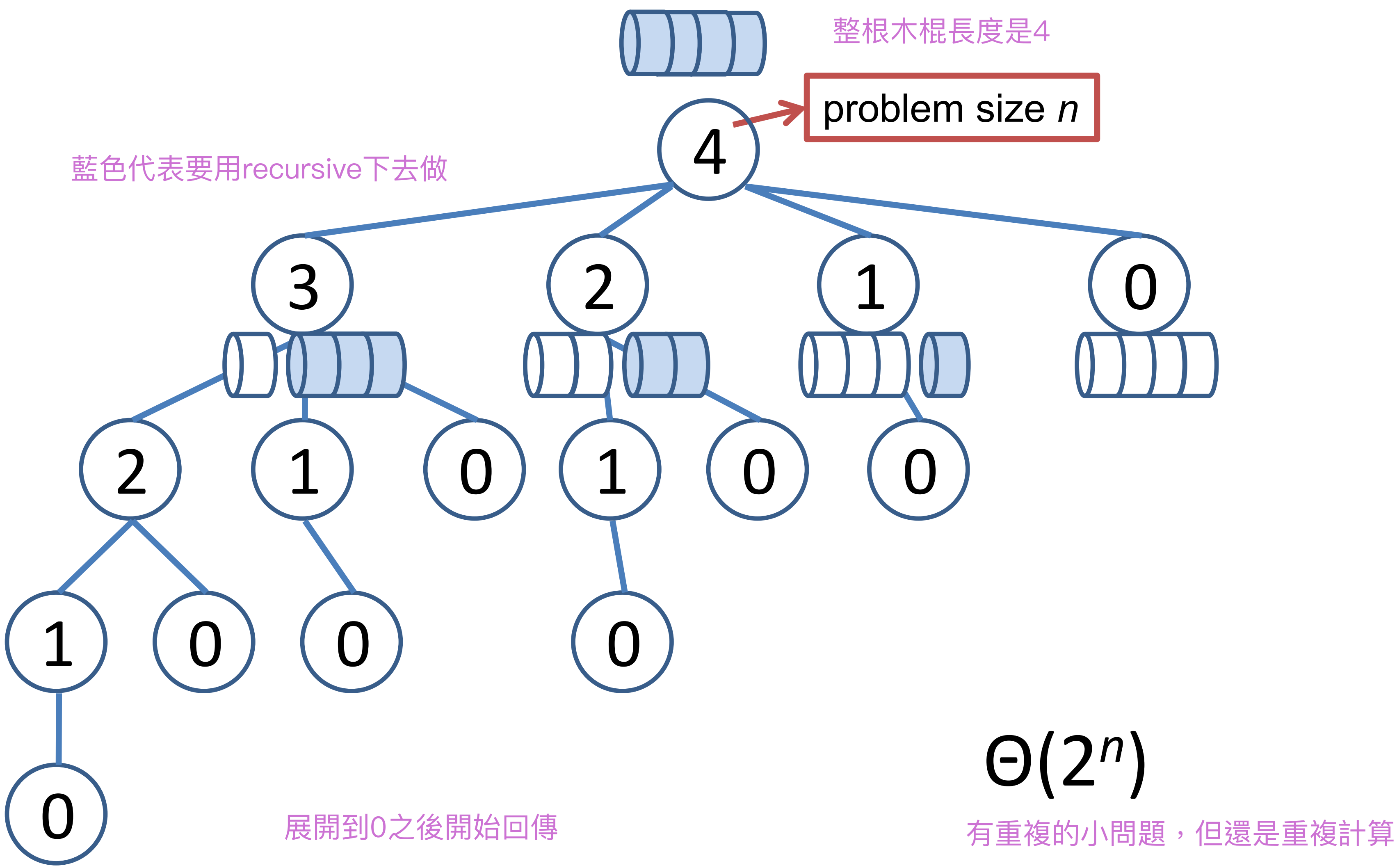
3. $q = -\infty$

4. **for** i = 1 **to** n 不確定切哪裡就只好每種組合都切切看

5. $q = \max(q, p[i] +$ CUT-ROD$(p, n-i))$

6. **return** q

CUT-ROD(p, 4)



整根木棍長度是4
problem size n
藍色代表要用recursive下去做
4
3
2
1
0
2
1
0
1
0
0
1
0
0
0
0
展開到0之後開始回傳
$\Theta(2^n)$
有重複的小問題，但還是重複計算

DP: Top-down approach

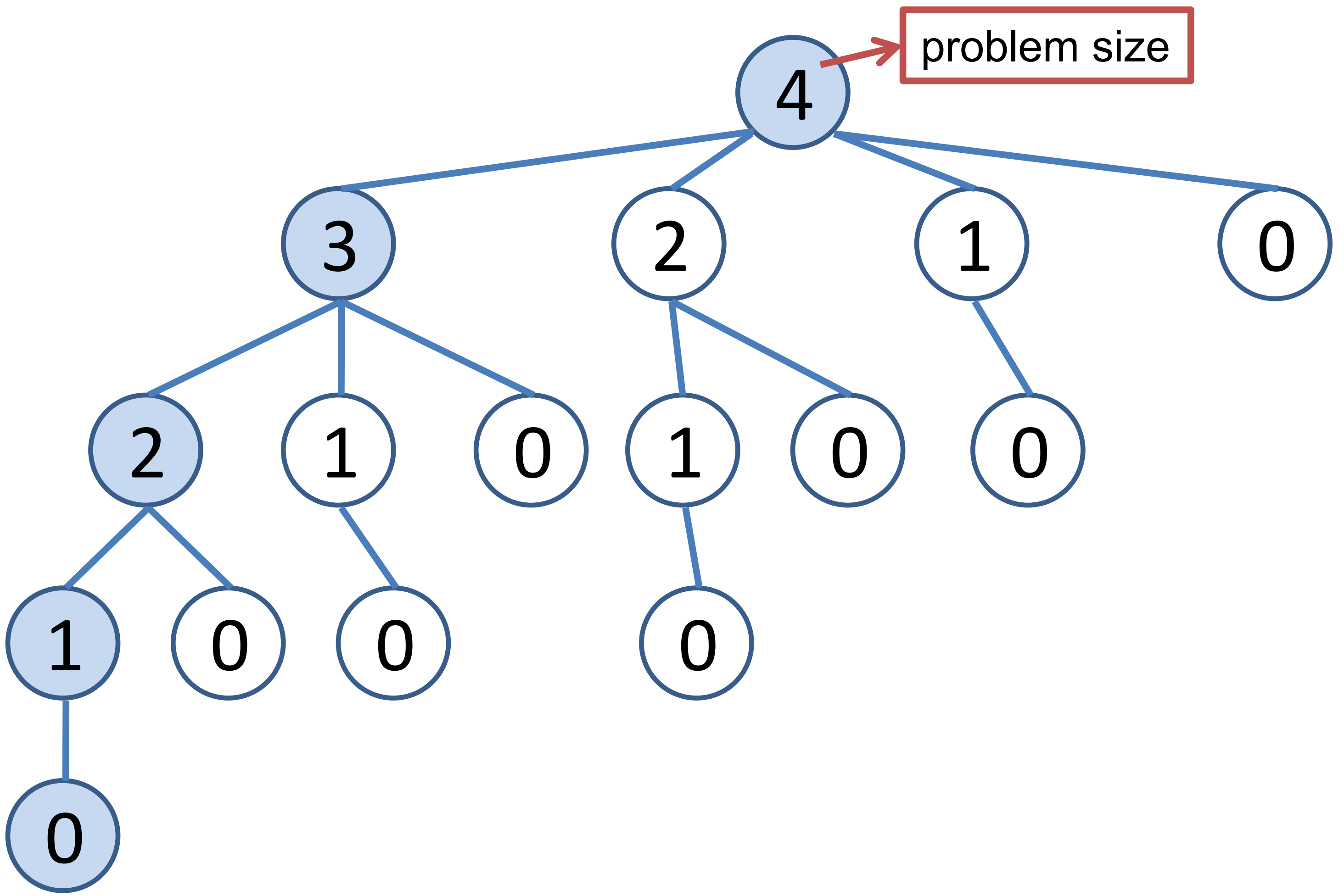
MEMOIZED-CUT-ROD(p, n)

1. let $r[0..n]$ be a new array 建立一個表格r
2. **for** i = 0 **to** n

因為是for迴圈，所以計算i = 3需要的東西，在i = 1&2 的時候都做過了

3. $r[i] = -\infty$
4. **return** MEMOIZED-CUT-ROD-AUX(p, n, r)

CUT-ROD(p, 4)



DP: Top-down approach

MEMORIZED-CUT-ROD-AUX(p, n, r) 紅色這幾行是DP才有的做法

1. **if** $r[n] \geq 0$
2. **return** $r[n]$
3. **if** $n == 0$
4. $q = 0$
5. **else** $q = -\infty$
6. **for** $i = 1$ **to** n
7. $q = \max(q, p[i] +$ MEMORIZED-CUT-ROD-AUX $(p, n\text{-}i, r))$
8. $r[n] = q$ 算好的q要存下來，下次看到重複問題就可以直接拿答案來用
9. **return** q

DP: Bottom-up approach

Length i	**1**	**2**	**3**	**4**
Price p_i	1	5	8	9

$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$
$$r_0 = 0$$

- $r_0 = 0$
- $r_1 = p_1 + r_0 = 1$
- $r_2 = \max(p_1 + r_1, p_2) = 5$ 有一個切點，可以選擇要切或不切
- $r_3 = \max(p_1 + r_2, p_2 + r_1, p_3) = 8$ 有兩個切點，可以選擇要切或不切
- $r_4 = \max(p_1 + r_3, p_2 + r_2, p_3 + r_1, p_4) = 10$
- …

```
BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] be a new array
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        if q < p[i] + r[j-i]
            q = p[i] + r[j-i]
    r[j] = q
return r[n]
```

先開一個r

把起點定下來

q是存最大值

Find max

給定長度是j，找出每個長度賺的錢的最大值，然後代換進q

stored

在j這個長度的最大值

$\boxed{\Theta(n^2)}$

兩層for loop，n跑兩次

What does this algorithm return? 回傳最多可以賣多少錢

What if we need to return the decomposition as well?

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
let r[0..n], s[0..n] be a new array
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        if q < p[i] + r[j-i]
            q = p[i] + r [j-i]
            s[j] = i    切在第 i 個地方，要把 i（切點）存下來
    r[j] = q
return r[n]
```

Call EXTENDED-BOTTOM-UP-CUT-ROD(p, 10)

We get $r[0..10]$, $s[0..10]$. Fill these two arrays.

Length *i*	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Price p_i	1	5	8	9	10	17	17	20	24	30

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$r[i]$											
$s[i]$											

Call EXTENDED-BOTTOM-UP-CUT-ROD(p, 10)

We get $r[0..10]$, $s[0..10]$. Fill these two arrays.

Length i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Price p_i	1	5	8	9	10	17	17	20	24	30

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

s 表示：為了要得到最多的錢，應該要切在哪

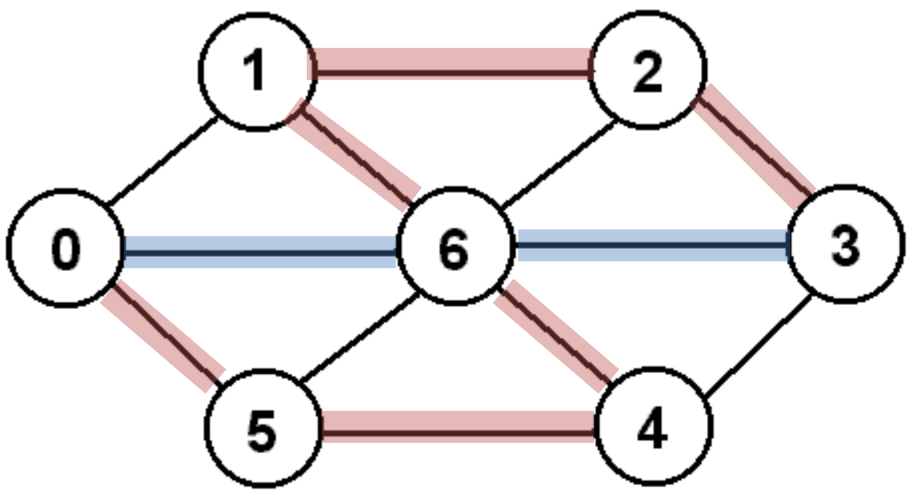
長度是9，最多賺25塊。
切在3跟(9−3)

Elements of Dynamic Programming

- Overlapping subproblems
  - Rod cutting problem
  - Fibonacci numbers
  - …
- Optimal substructure

- **Un-weighted shortest simple path**
  - find a simple path from *u* to *v* consisting of the ***fewest*** edges
- **Un-weighted longest simple path**
  - find a simple path from *u* to *v* consisting of the ***most*** edges

大問題不會用到小問題的解，
所以不能用DP



3/28(二) End

Longest Common Subsequence (LCS)

Chapter 15.4

3/30 (四)
START

The LCS Problem

“a” ***not*** “the”

- Given two sequences $x[1 \, . \, . \, m]$ and $y[1 \, . \, . \, n]$, find a longest subsequence common to them both.

要找最長「共同子序列」

這個例子中最長的共同子序列長度為4

x: A B C B D A B

y: B D C A B A

$\mathrm{LCS}(x, y)$ = BCBA or BCAB

functional notation, but ***not*** a function

在X在Y裡都有出現的序列

Brute Force Approach

- Check every subsequence of $x[1\,.\,.\,m]$ to see if it is also a subsequence of $y[1\,.\,.n]$.
  - 2^m subsequences of x 要窮舉所有組合的話，有2的m次方多種可能
  - Checking = $O(n)$ time per subsequence 掃過一次Y序列
  - Worst-case running time: $O(n2^m)$

概念上就是要先拿一個序列（無論長短）來窮舉所有可能的組合，再掃過另一個序列

Toward a better algorithm (1)

大小問題的切分方式用prefix來做

- Prefix 從頭開始算的都算
  - ABCB is a prefix of ABCBDAB
  - BCB is not a prefix of ABCBDAB 一定要從頭開始，不能從第二個B開始
- x = ABCBDAB
  - $x[1..4]$: the first four symbols in x, a prefix of x
  - $x[1..4]$ = ABCB

Toward a better algorithm (2)

- Strategy
  - Consider the ***length*** of LCS first
  - Define $|x|$ the length of a sequence x
    - |ABCBDAB|= 7
  - Define $c[i, j] = |\mathrm{LCS}(x[1\,.\,.\,i], y[1\,.\,.\,j])|$
  - Then, $c[m, n] = |\mathrm{LCS}(x, y)|$

$c[i, j]$ vs. $c[i\text{-}1, j\text{-}1]$

$c[i\text{-}1, j]$

$c[i, j\text{-}1]$

Example $c[i, j]$ vs. $c[i-1, j-1]$

$c[i-1, j]$

$c[i, j-1]$

- LCS (ABCBDAB, BDCABA) = ?

	B	D	C	A	B	A
A						
B	c[i-1, j-1]	c[i-1, j]				
C	c[i, j-1]	? LCS(ABC, BD) c[i, j]				
B						
D				? LCS(ABCBD, BDCA)		
A						
B						

記錄的是兩個prefix的長度

Examples

BCD (3) BC (2)

- LCS(ABCBD, BDCD) vs. LCS(ABCB, BDC)

  $c[i, j]$ $c[i\text{-}1, j\text{-}1]$

- LCS(ABCBD, BDCA) vs. LCS(ABCB, BDC)

  $c[i, j]$ $c[i\text{-}1, j\text{-}1]$

- LCS(ABCBD, BDCA) vs. LCS(ABCB, BDCA)

  $c[i, j]$ $c[i\text{-}1, j]$

- LCS(ABCBD, BDCA) vs. LCS(ABCBD, BDC)

  $c[i, j]$ $c[i, j\text{-}1]$

BC or BD (2) BC or BD (2)

$c[i, j]$ vs. $c[i\text{-}1, j\text{-}1]$

$c[i\text{-}1, j]$

$c[i, j\text{-}1]$

- # Theorem

左上角的格子再加1

$$c[i,j] = \begin{cases} c[i-1, j-1] + 1 & \text{if } x[i] = y[j], \\ \max\{c[i-1, j], c[i, j-1]\} & \text{otherwise.} \end{cases}$$

- # Example:

行是B，列是C，不一樣的時候要選大的，拿來加1

一開始建表格的時候，因為最左上角是空字串，所以LCS都是0

		A	B	C	B	D	A	B	m+1
	0	0	0	0	0	0	0	0	
B	0	↑0	↖1	←1	↖1	←1	←1	↖1	length of LCS(B, ABCB)
D	0	↑0	↑1	↑1	↑1	↖2	←2	←2	
C	0	↑0	↑1	↖2	←2	↑2	↑2	↑2	
A	0	↖1	↑1	↑2	↑2	↑2	↖3	←3	length of LCS(BDCA, ABCBDA)
B	0	↑1	↖2	↑2	↖3	←3	↑3	↖4	
A	0	↖1	↑2	↑2	↑3	↑3	↖4	↑4	length of LCS(X, Y)
n+1									

看三個位置，左、上、左上。
行、列字母一樣的時候一律從左上來，拿左上+1。
行、列字母不一樣的時候，取上、左之中較大的數字。

$c[ABC,\varnothing]$, $c[AB,B]$ $\max\{\underbrace{c[AB,B]}_{1}, \underbrace{c[A,BD]}_{0}\}$

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	? 0	↖ 1	← 1	↖ 1	← 1	← 1	↖ 1
D	0	? 0	↑ 1	? 1	? 1	↖ 2	← 2	← 2
C	0	? 0	↑ 1	↖ 2	← 2	? 2	? 2	? 2
A	0	↖ 1	? 1	↑ 2	? 2	↑ 2	↖ 3	← 3
B	0	↑ 1	↖ 2	? 2	↖ 3	← 3	? 3	↖ 4
A	0	↖ 1	↑ 2	? 2	↑ 3	? 3	↖ 4	? 4

? 表示 max{左邊, 上面} 取出來的值會相等，所以從哪來都一樣，可以直接設定是從上面來的。
(實務上會給定一個固定方向)

⇒ B D A B

```
LCS-LENGTH(X, Y)
let b[1..m , 1..n], c[0..m, 0..n] be new arrays
for i = 1 to m
    c[i, 0] = 0
for i = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i-1, j-1] + 1
            b[i, j] = ↖
        elseif c[i-1, j] ≥ c[i, j-1]
            c[i, j] = c[i-1, j]
            b[i, j] = ↑
        else c[i, j] = c[i, j-1]
            b[i, j] = ←
return c and b
```

記錄箭號

記錄數值

// initialize *c*

把表格填完需要的時間複雜度

Time? $O(mn)$

Space? $O(mn)$

see "↖", print!

用數字編號紀錄箭號，
看到左上角的箭號，就印出來

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	↑0	↖1	←1	↖1	←1	←1	↖1
D	0	↑0	↑1	↑1	↑1	↖2	←2	←2
C	0	↑0	↑1	↖2	←2	↑2	↑2	↑2
A	0	↖1	↑1	↑2	↑2	↑2	↖3	←3
B	0	↑1	↖2	↑2	↖3	←3	↑3	↖4
A	0	↖1	↑2	↑2	↑3	↑3	↖4	↑4

Output: B D A B

LCS的解可能不是唯一的

從右下角跟著箭頭往回走，把走過的路記下來，挑出左上角箭頭的格子（他們代表的意義是，同時加了同樣的字母進兩個序列），所以照順序寫出來，就會是最長共同序列。

```
PRINT-LCS(b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == ↖
        PRINT-LCS(b, X, i-1, j-1)
        print x_i
elseif b[i, j] == ↑
        PRINT-LCS(b, X, i-1, j)
else
        PRINT-LCS(b, X, i, j-1)
```

印出最長共同序列需要的時間複雜度（照著箭頭往回走的時間複雜度）

Time? $O(m+n)$

Practice

- What's the time and space complexity if we need to compute only the length?
  - Time: $O(mn)$
  - Space: $O(\min(m, n))$