计算机问题求解 – 论题3–1 - 动态规划

2014年09月3日

Fibonacci: $F_n = F_{n-1} + F_{n-2}$

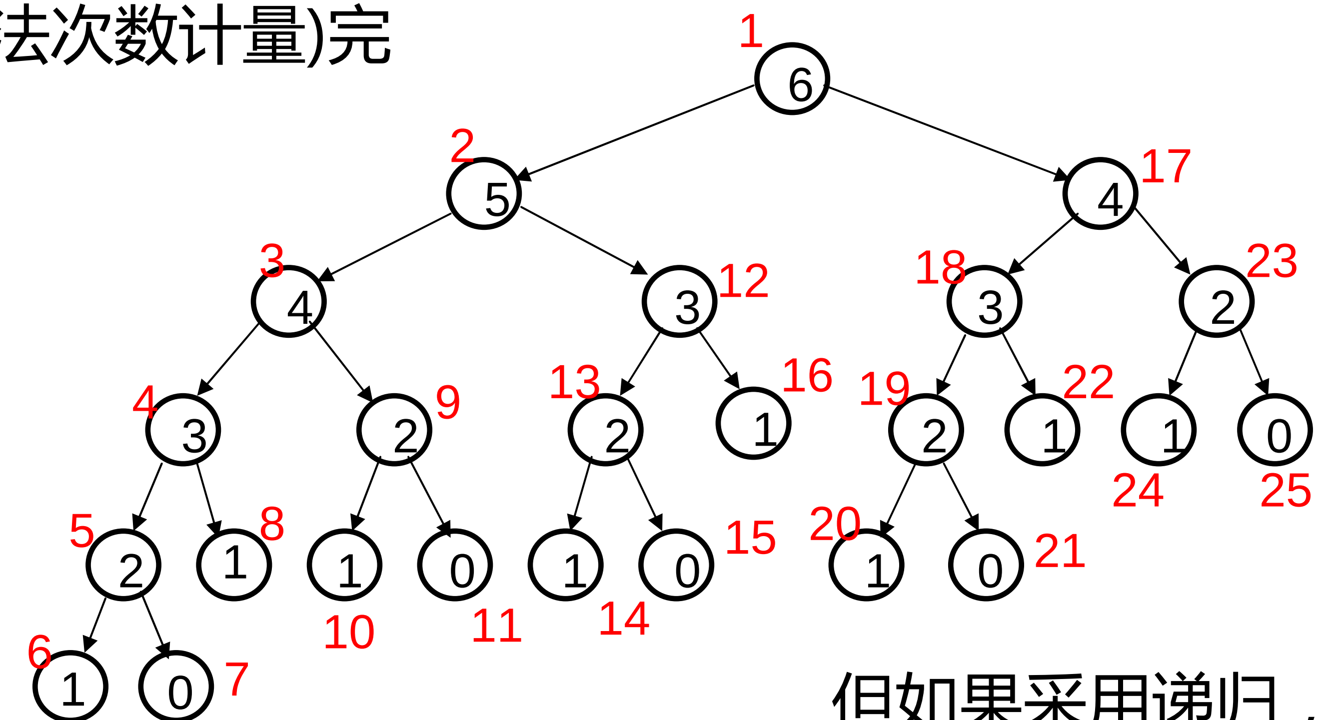
问题1:

如果要你计算第*n*个Fibonacci数，你用递归还是用循环，还是随便？为什么？

递归可能代价高昂

计算第n个Fibonacci数其实可以在线性时间内(以加法次数计量)完成。

$$F_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$



但如果采用递归，递归调用的次数是 $2F_{n+1}-1$

问题2：相比较快速排序的分治法递归，为什么上面的例子采用递归代价高昂？

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```

问题3：

我们有什么办法来应对这种情况？

Rod Cutting Problem

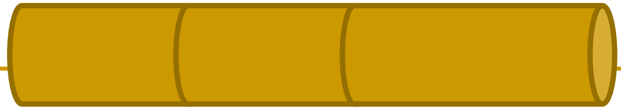
The *rod-cutting problem* is the following. Given a rod of length n inches and a table of prices p_i for $i = 1, 2, \ldots, n$, determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces.

一个样本输入及其解：

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

$r_1 = 1$ from solution $1 = 1$ (no cuts) ,
$r_2 = 5$ from solution $2 = 2$ (no cuts) ,
$r_3 = 8$ from solution $3 = 3$ (no cuts) ,
$r_4 = 10$ from solution $4 = 2 + 2$,
$r_5 = 13$ from solution $5 = 2 + 3$,
$r_6 = 17$ from solution $6 = 6$ (no cuts) ,
$r_7 = 18$ from solution $7 = 1 + 6$ or $7 = 2 + 2 + 3$,
$r_8 = 22$ from solution $8 = 2 + 6$,
$r_9 = 25$ from solution $9 = 3 + 6$,
$r_{10} = 30$ from solution $10 = 10$ (no cuts) .

r_7:



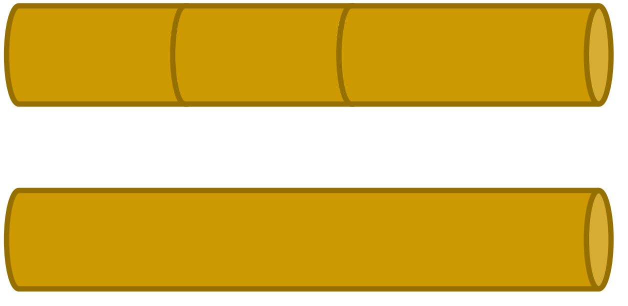
问题4：

为什么可能的割法数量是2^{n-1}？

问题5:

解决问题从那里开始?

r_7:



我们总是要切第一刀的，但是第一刀割在何?

递归的解法：扫描所有可能的割法

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

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

问题6：
左边的两个式子
有什么区别？

```
CUT-ROD(p, n)
if n == 0
    return 0
q = -∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n - i))
return q
```

最优子结构：

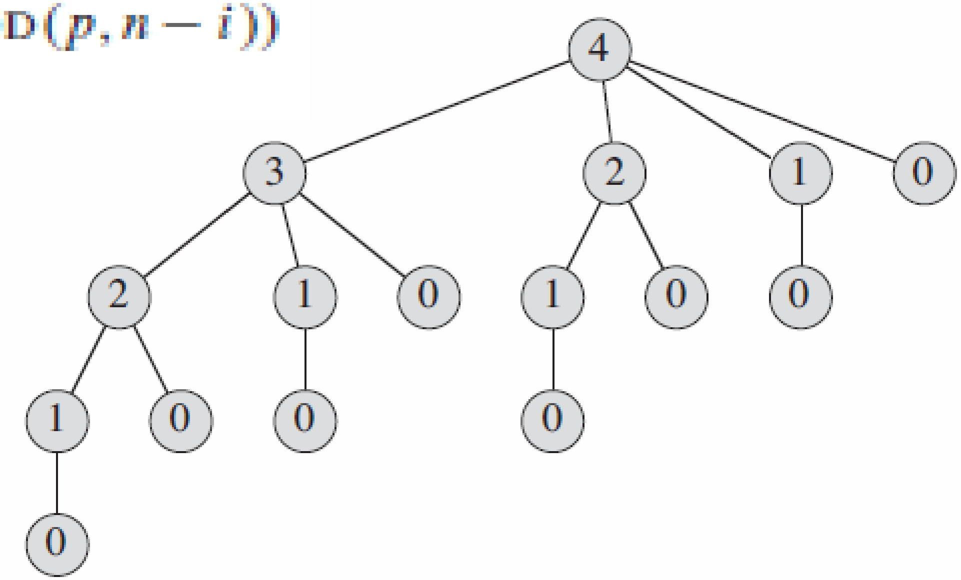
$$r_n = \max\left(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1\right)$$

问题7：

你能借助以上的式子解释一下什么是最优子结构(Optimal Substructure)？

递归的解法：

```
CUT-ROD(p, n)
if n == 0
    return 0
q = −∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n − i))
return q
```



问题8：

为什么这个算法注定是低效率的？

问题9：

用循环的方法计算第*n*个Fibonacci数效率会很高，这对你有什么启发吗？

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
if r[n] ≥ 0
    return r[n]
if n == 0
    q = 0
else q = −∞
    for i = 1 to n
        q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n − i, r))
r[n] = q
return q
```

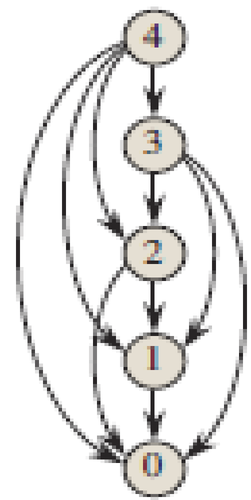
```
MEMOIZED-CUT-ROD(p, n)
let r[0..n] be a new array
for i = 0 to n
    r[i] = −∞
return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

```
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
        q = max(q, p[i] + r[j − i])
    r[j] = q
return r[n]
```

复杂度均降到平方级！

问题9：

消除重复计算，甚至完全消除递归的关键是什么？



关键是次序！

问题10：

为什么先人命名这个方法为dynamic programming？

子问题的序在动态规划算法设计中非常重要：

```
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
        q = max(q, p[i] + r[j - i])
    r[j] = q
return r[n]
```

最优值和最优解

- 最优值还不是解，我们需要得到实现最优值的“那个解”。

问题11：

我们怎么能从“值”得到“解”？

再加一个数组，跟踪最优值获得的过程。

```
PRINT-CUT-ROD-SOLUTION(p, n)
(r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
while n > 0
    print s[n]
    n = n - s[n]
```

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

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] and s[0..n] be new arrays
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
    r[j] = q
return r and s
```

问题12：

你能否用“通俗”的表达方式说说$s[j]$究竟是什么？

Matrix-Chain Multiplication Problem

- 需要完成的任务:

  求乘积: $A_1 \times A_2 \times \ldots \times A_{n-1} \times A_n$

A_i 是二维矩阵，一般不是方阵，大小符合乘法规定的要求。

- 为什么会成为问题:
  - 矩阵乘法满足结合律，因此我们可以任意指定运算顺序；
  - 而不同的计算顺序代价差别很大。
- 优化问题: **什么样的次序计算代价最小?**

矩阵乘法的代价

Let $C = A_{p\times q} \times B_{q\times r}$

$$c_{i,j} = \sum_{k=1}^{q} a_{i \ldots}$$

An example: $A_1 \times A_2 \times A_3 \times A_4$
30×1 1×40 40×10 10×25

$((A_1\times A_2)\times A_3)\times A_4$: 20700 multiplications
$A_1\times(A_2\times(A_3\times A_4))$: 11750
$(A_1\times A_2)\times(A_3\times A_4)$: 41200
$A_1\times((A_2\times A_3)\times A_4)$: 1400

C 共有 $p\times r$ 个元素

所以，总共执行乘法 pqr 次。

$$P(n)=\begin{cases}1 & \text{if } n=1\\ \sum_{k=1}^{n-1}P(k)P(n-k) & \text{if } n\geq 2\end{cases}$$

问题13:

解释上面的式子，以此说明“穷举所有加括号的方式”的解法是效率很低的？

问题14：

什么是矩阵连乘问题的“第一刀”？你能否从这一点出发讨论此问题的“最优子结构”？

最优值的递归表示

We can define $m[i,j]$ recursively as follows. If $i = j$, the problem is trivial; the chain consists of just one matrix $A_{i..i} = A_i$, so that no scalar multiplications are necessary to compute the product. Thus, $m[i,i] = 0$ for $i = 1, 2, \ldots, n$. To compute $m[i,j]$ when $i < j$, we take advantage of the structure of an optimal solution from step 1. Let us assume that to optimally parenthesize, we split the product $A_i A_{i+1} \cdots A_j$ between A_k and A_{k+1}, where $i \le k < j$. Then, $m[i,j]$ equals the minimum cost for computing the subproducts $A_{i..k}$ and $A_{k+1..j}$, plus the cost of multiplying these two matrices together. Recalling that each matrix A_i is $p_{i-1} \times p_i$, we see that computing the matrix product $A_{i..k} A_{k+1..j}$ takes $p_{i-1} p_k p_j$ scalar multiplications. Thus, we obtain

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \, .$$

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j \end{cases}$$

问题15：

如果直接用递归计算，为什么子问题一定会大量地重复被计算？

类比rod cutting problem, 递归的代价应该是指数级的，但子问题个数其实只有平方级。

问题16：

计算次序应该如何安排？

We shall implement the tabular, bottom-up method in the procedure MATRIX-CHAIN-ORDER, which appears below. This procedure assumes that matrix A_i has dimensions $p_{i-1} \times p_i$ for $i = 1, 2, \ldots, n$. Its input is a sequence $p = \langle p_0, p_1, \ldots, p_n \rangle$, where $p.length = n + 1$. The procedure uses an auxiliary table $m[1 \ldots n, 1 \ldots n]$ for storing the $m[i, j]$ costs and another auxiliary table $s[1 \ldots n-1, 2 \ldots n]$ that records which index of k achieved the optimal cost in computing $m[i, j]$. We shall use the table s to construct an optimal solution.

```
MATRIX-CHAIN-ORDER(p)
n = p.length - 1
let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                    // l is the chain length
    for i = 1 to n - l + 1
        j = i + l - 1
        m[i, j] = ∞
        for k = i to j - 1
            q = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

连乘矩阵的个数递增

连乘链起点

连乘链终点

扫描指定范围内所有子问题

记录最小值与实现点

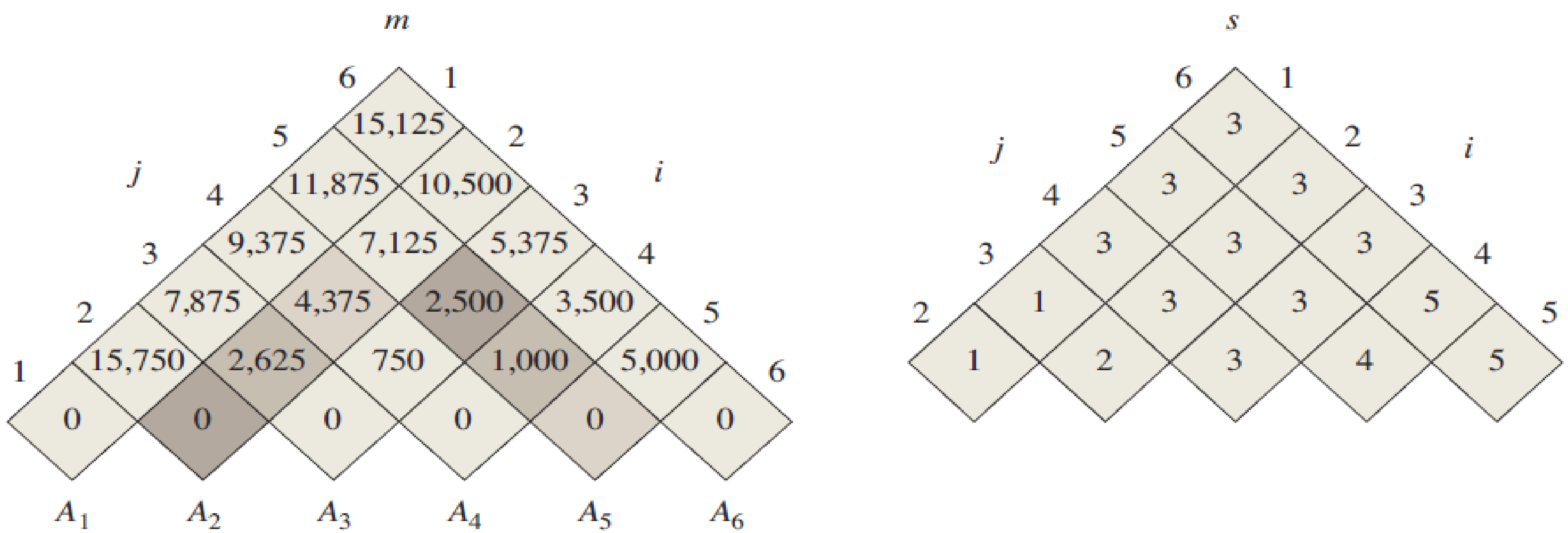


Figure 15.5 The m and s tables computed by MATRIX-CHAIN-ORDER for $n = 6$ and the following matrix dimensions:

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

The tables are rotated so that the main diagonal runs horizontally. The m table uses only the main diagonal and upper triangle, and the s table uses only the upper triangle. The minimum number of scalar multiplications to multiply the 6 matrices is $m[1,6] = 15{,}125$. Of the darker entries, the pairs that have the same shading are taken together in line 10 when computing

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13{,}000 , \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125 , \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11{,}375 \end{cases}$$
$$= 7125 .$$

问题17：数字添入顺序是怎样的？

问题18：

什么样的问题适合用动态规划解决？

问题19：

有人说动态规划能用多项式时间解决原来是指数级难度的问题，这话对吗？

课外作业

- TC pp.369-: ex.15.1-1, 15.1-3
- TC pp.378-: ex.15.2-2, 15.2-4
- TC pp.389-: ex.15.3-3, 15.3-5, 15.3-6
- TC pp.396-: ex.15.4-3, 15.4-5
- TC pp.403-: ex.15.5-1
- TC pp.404-: prob.15-4