动态规划

算法设计与分析

武汉大学
国家网络安全学院
李雨晴

第二部分 基于递归的技术

- 递归的概念
- 分治
- 动态规划算法*

引例：费氏数列

- 费氏数列是由13世纪的意大利数学家、来自Pisa的 Leonado Fibnacci发现。
- 费氏数列是由0，1开始，之后的每一项等于前两项之和：

  0，1，1，2，3，5，8，13，21，34，55，89，144...... 。

引例：费氏数列

- 这个数列有如下一些特性：
  - 前2个数相加等于第3个数
  - 前1个数除以后一个数越往后越无限接近于0.618 (黄金分割)
  - 相邻的两个比率必是一个小于0.618一个大于0.618
  - 后1个数除以前一个数越往后越无限接近于1.618
  - …

$$f(n)=\begin{cases}1 & , \ if \ n=1,2 \\ f(n-1)+f(n-2), & if \ n\geq 3\end{cases}$$

引例：费氏数列

$$f(n)=\begin{cases}1 & ,\ if\ n=1,2 \\ f(n-1)+f(n-2) & ,\ if\ n\geq 3\end{cases}$$

递归形式的算法：

```
procedure fib(n)
   if n=1 or n=2 then return 1
   else return fib(n-1)+fib(n-2)
```

优点：→ 简洁，容易书写以及调试。

缺点：→ 效率低下 。

为何效率低下？

- 使用时间复杂性的方式分析

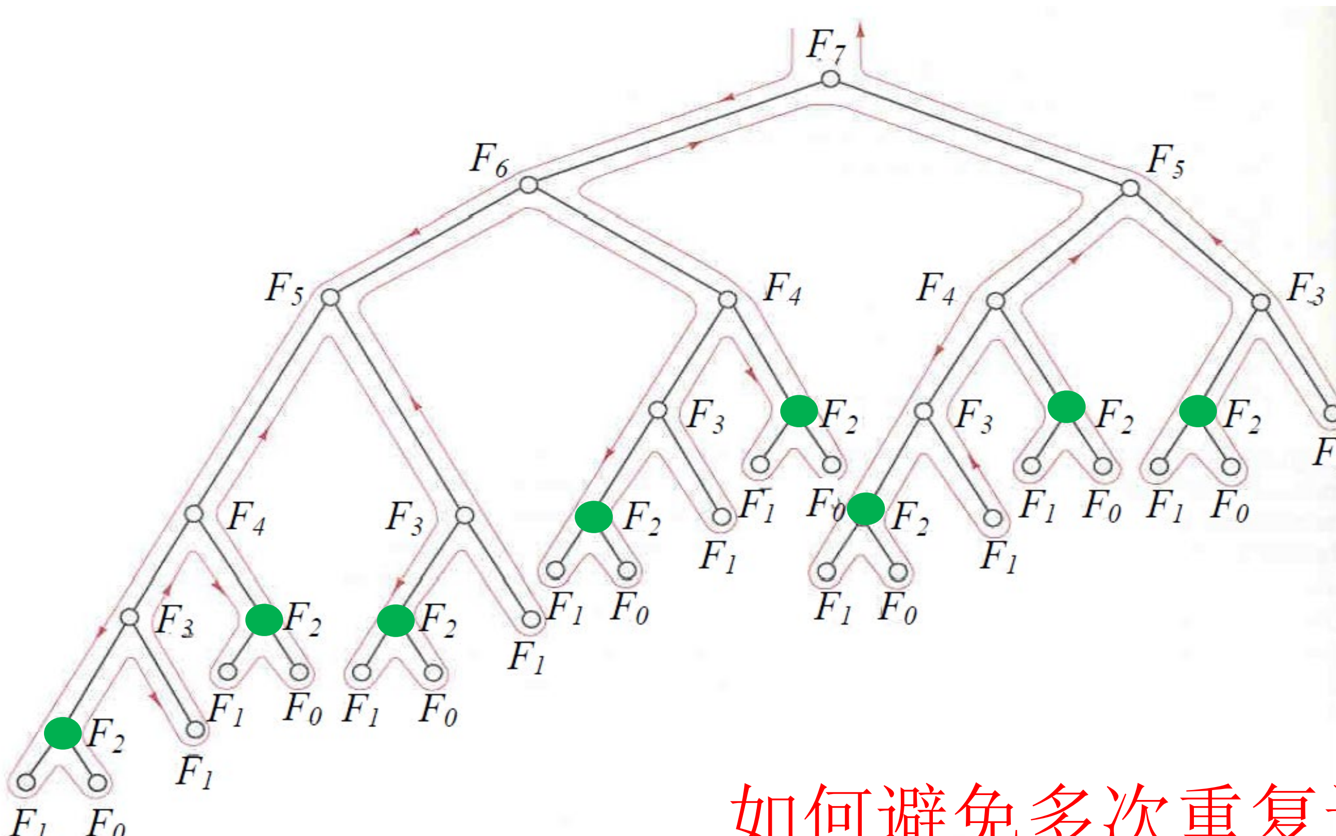
$$T(n)=\begin{cases}1 & if\ \ n=1,2\\ T(n-1)+T(n-2) & if\ \ n\geq 3\end{cases}$$

$$\longrightarrow \quad T(n)\approx\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n\approx 0.447(1.618)^n$$

即时间复杂度为输入规模的指数形式。当n=100时，用递归求解的时间T(100)≈3.53×10^{20}, 若每秒计算10^8次，需111,935年!

为何效率低下？

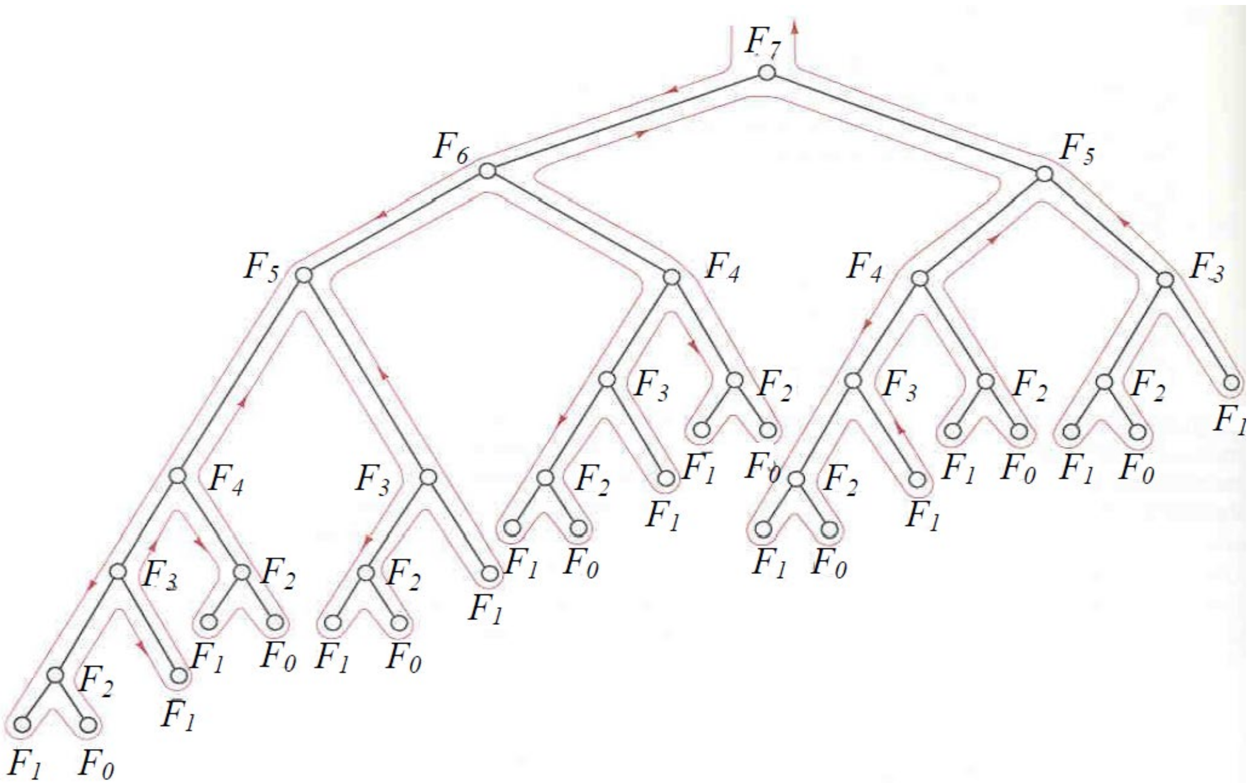
- 使用直观的方式分析 → 存在大量重复计算



如何避免多次重复计算?

解决方法

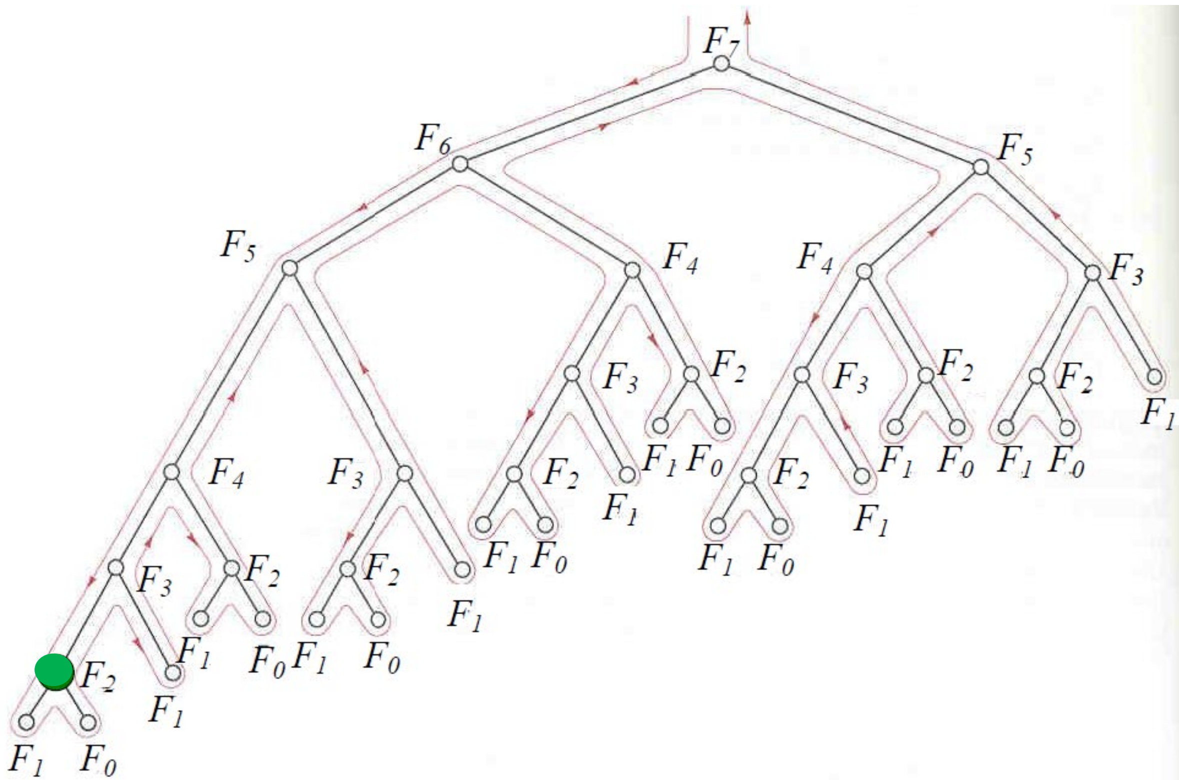
- 用表(通常用数组)记录子问题的解，以便保存和以后的检索



v[0]	1
v[1]	1
v[2]	-1
v[3]	-1
v[4]	-1
v[5]	-1
v[6]	-1
v[7]	-1

解决方法

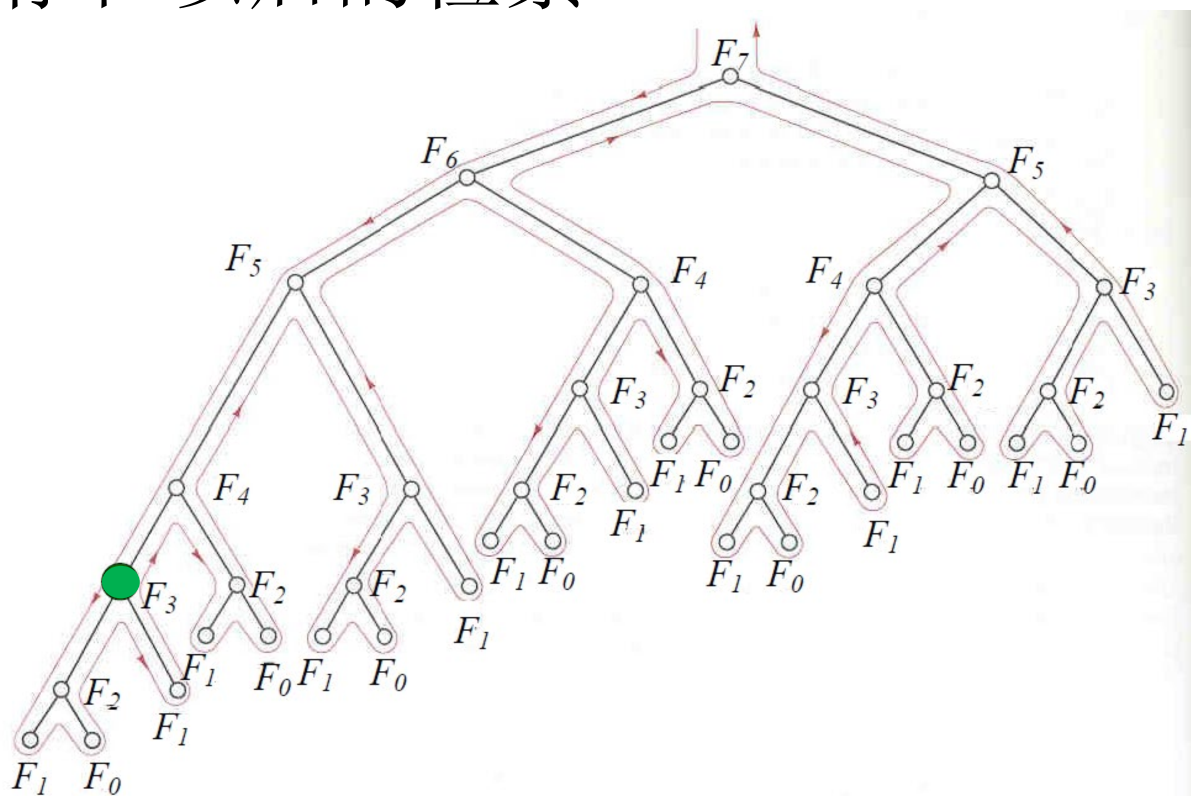
- 用表(通常用数组)记录子问题的解，以便保存和以后的检索



v[0]	1
v[1]	1
v[2]	2
v[3]	-1
v[4]	-1
v[5]	-1
v[6]	-1
v[7]	-1

解决方法

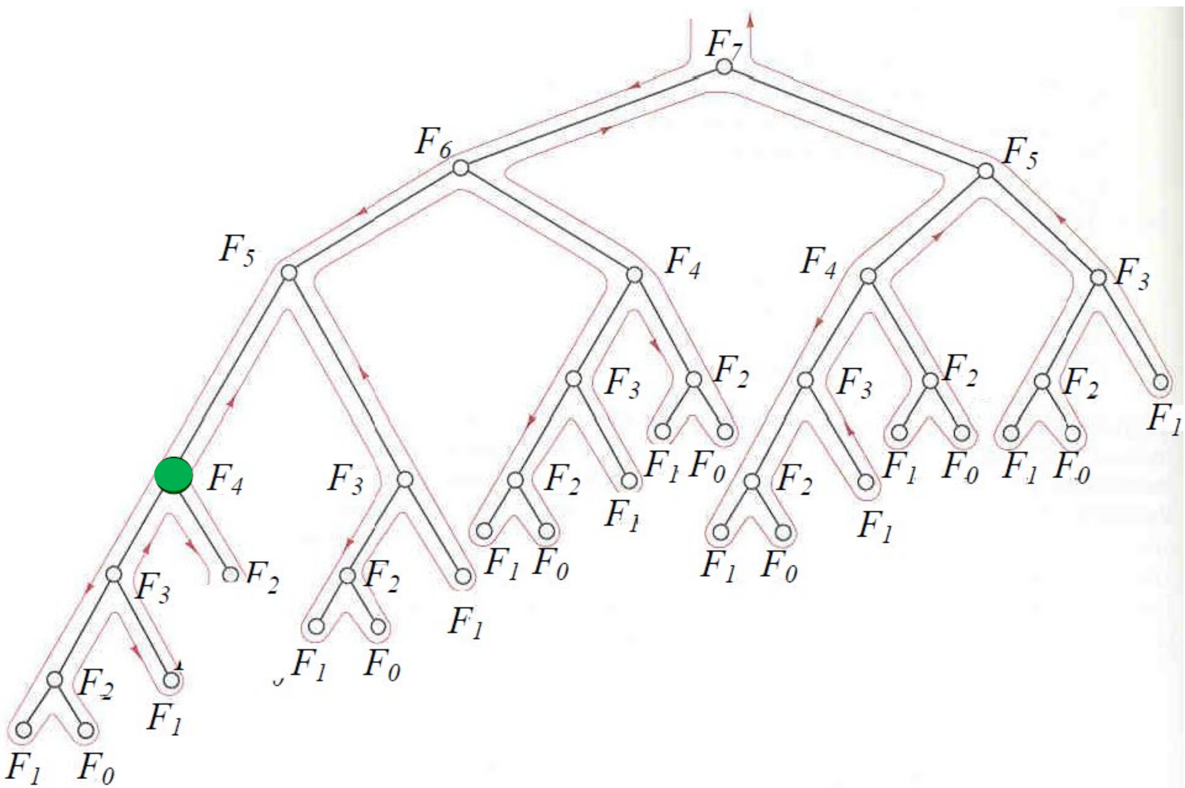
- 用表(通常用数组)记录子问题的解，以便保存和以后的检索



v[0]	1
v[1]	1
v[2]	2
v[3]	3
v[4]	-1
v[5]	-1
v[6]	-1
v[7]	-1

解决方法

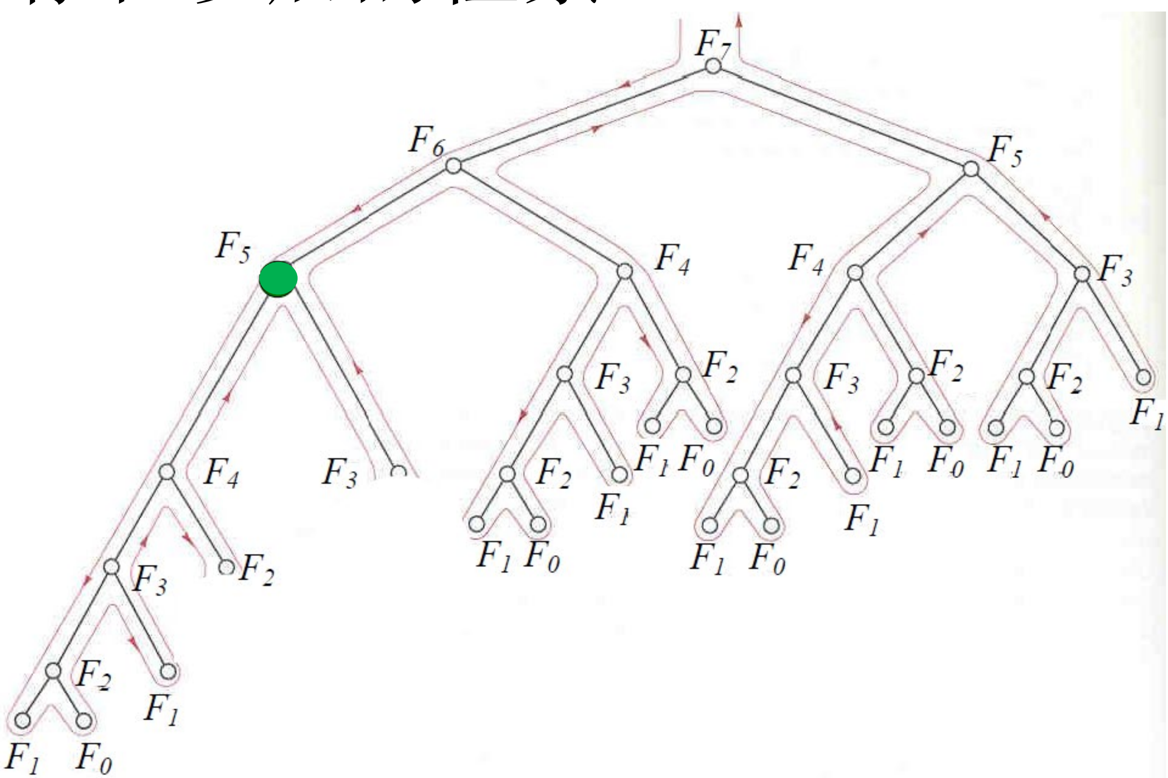
- 用表(通常用数组)记录子问题的解，以便保存和以后的检索



$v[0]$	1
$v[1]$	1
$v[2]$	2
$v[3]$	3
$v[4]$	**5**
$v[5]$	-1
$v[6]$	-1
$v[7]$	-1

解决方法

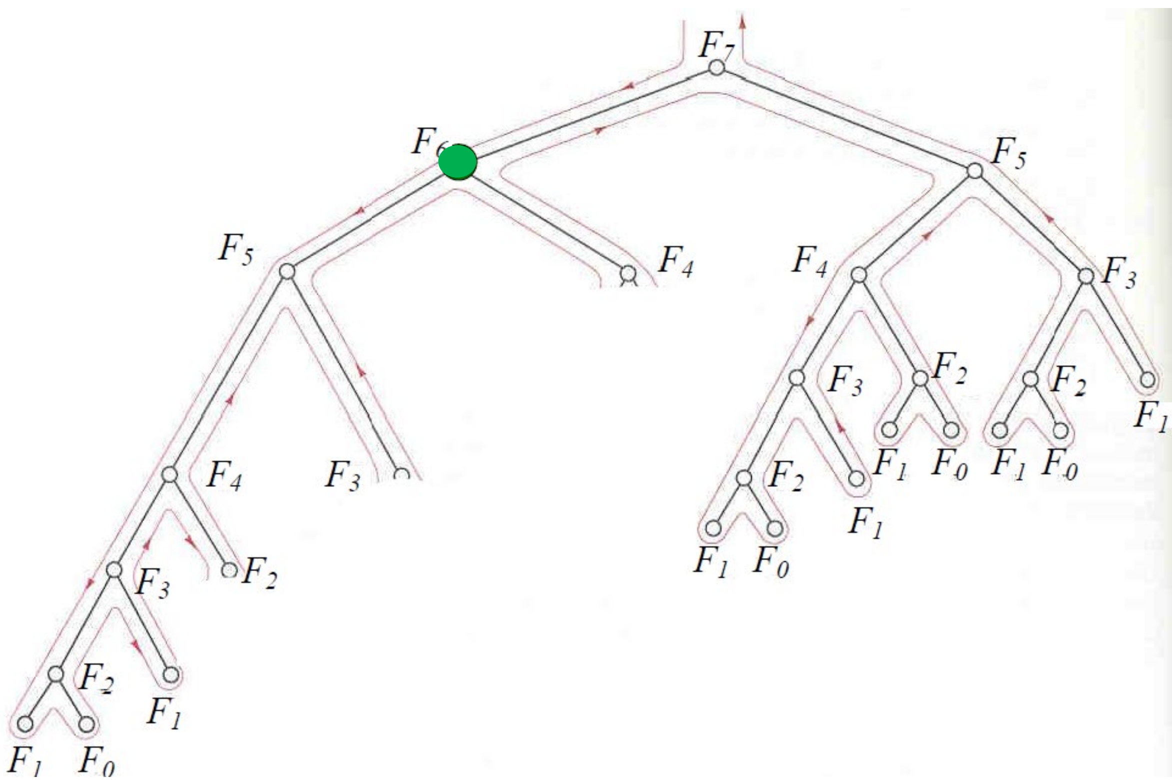
- 用表(通常用数组)记录子问题的解，以便保存和以后的检索



v[0]	1
v[1]	1
v[2]	2
v[3]	3
v[4]	5
v[5]	**8**
v[6]	-1
v[7]	-1

解决方法

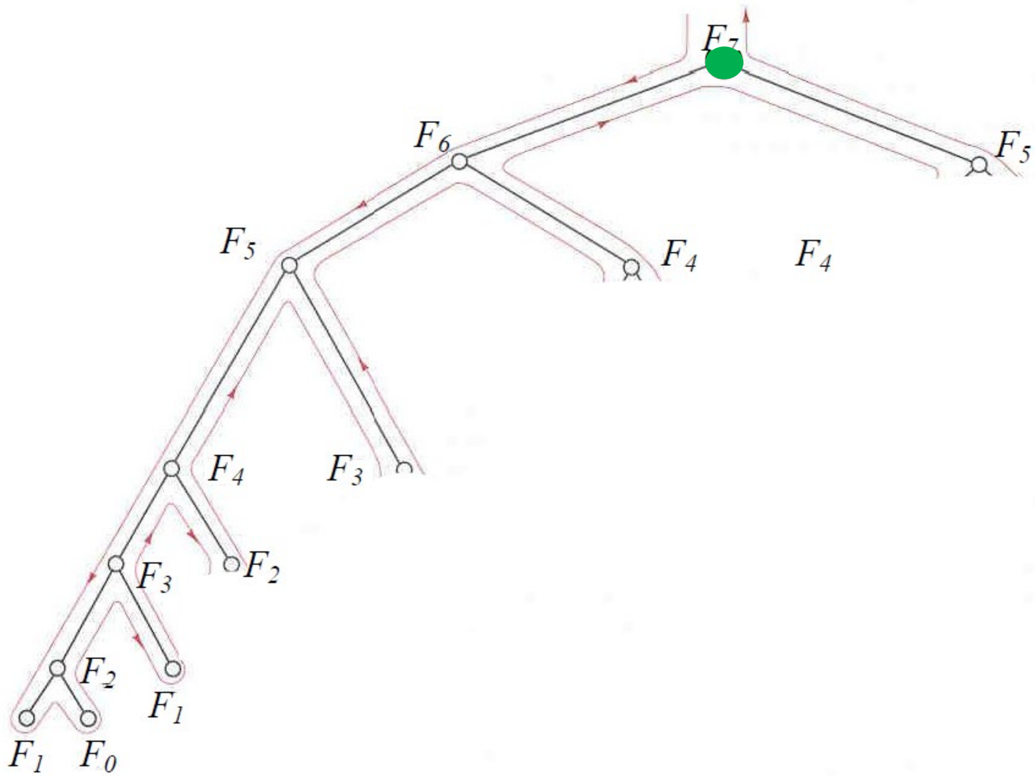
- 用表(通常用数组)记录子问题的解，以便保存和以后的检索



v[0]	1
v[1]	1
v[2]	2
v[3]	3
v[4]	5
v[5]	8
v[6]	**13**
v[7]	-1

解决方法

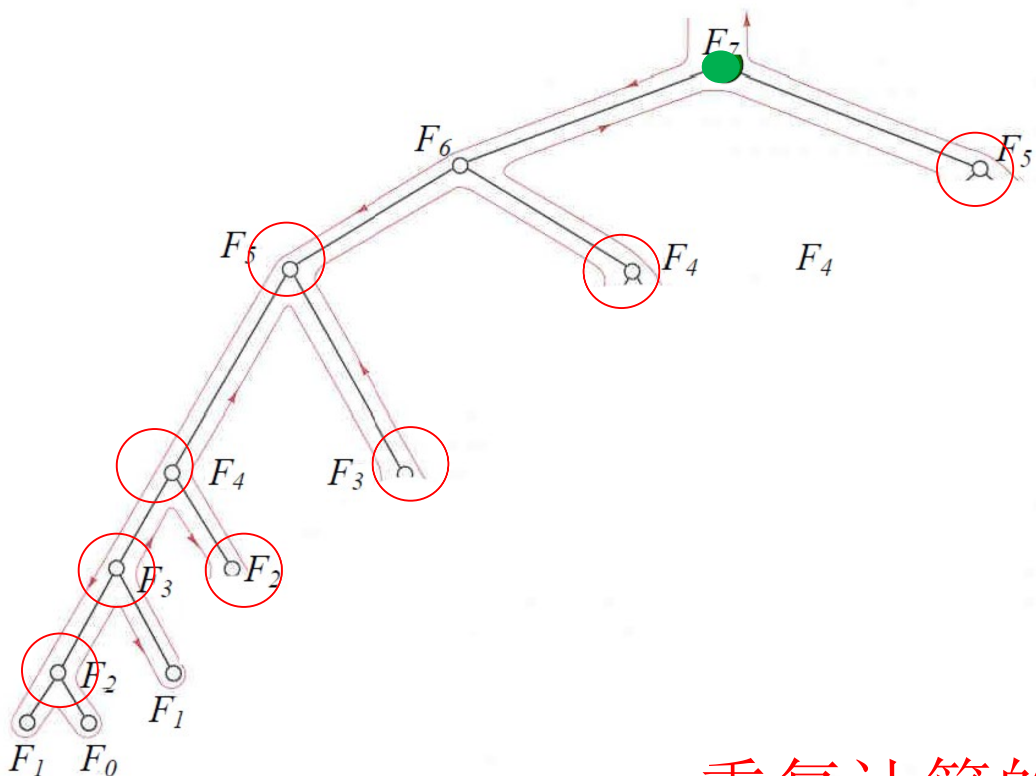
- 用表(通常用数组)记录子问题的解，以便保存和以后的检索



v[0]	1
v[1]	1
v[2]	2
v[3]	3
v[4]	5
v[5]	8
v[6]	13
v[7]	**21**

解决方法

- 还有没有更好的改进?



重复计算的函数调用

解决方法

- 自顶向下 Top-down：递归
- 自底向上 Bottom-up：递推，节约大量无用递归调用

递推形式的算法 F(n)：

$A[1] = A[2] \leftarrow 1$
for i $\leftarrow$ 3 to n do
    $A[i] \leftarrow A[i-1] + A[i-2]$
return A[n]

$$T(n) = \Theta(n)$$

临时变量 →

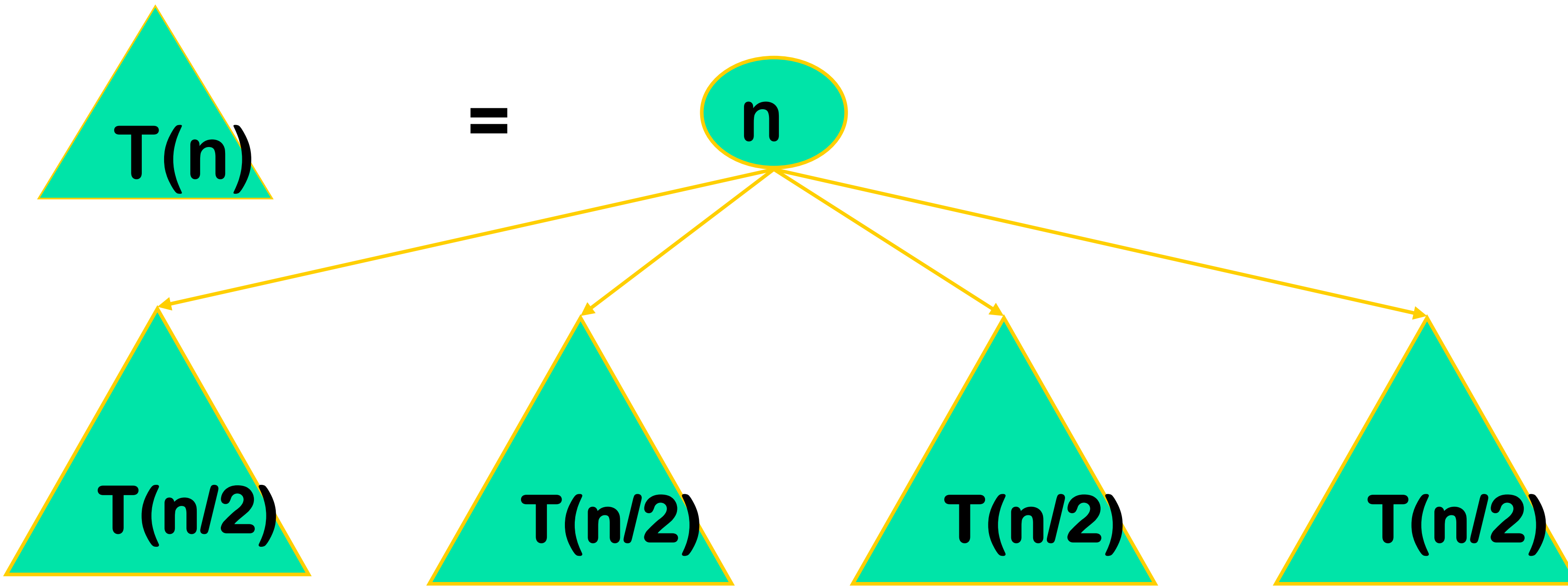
$f_1 \leftarrow 1$
$f_2 \leftarrow 1$
for i $\leftarrow$ 3 to n
    result $\leftarrow f_1+f_2$
    $f_1 \leftarrow f_2$
    $f_2 \leftarrow$ result
end for
return result

动态规划

- 动态规划(dynamic programming)是运筹学的一个分支，20世纪50年代初美国数学家R.E.Bellman等人在研究多阶段决策过程的优化问题时，提出了著名的最优化原理，把多阶段过程转化为一系列单阶段问题，利用各阶段之间的关系，逐个求解，创立了解决这类过程优化问题的新方法--动态规划。
- 动态规划主要用于求解以时间划分阶段的动态过程的优化问题，但一些与时间无关的静态规划(如线性规划、非线性规划)，只要人为地引进时间因素，把它视为多阶段决策过程，也可以用动态规划方法方便地求解。
- 动态规划问世以来，在经济管理、生产调度、工程技术和最优控制等方面得到了广泛应用。例如最短路线、资源分配、设备更新等问题，用动态规划比用其它方法求解更为方便。

算法总体思想

- 动态规划算法与分治法类似，其基本思想也是将待求解问题分解成若干个子问题，先求解子问题，再从子问题的解得到原问题的解。

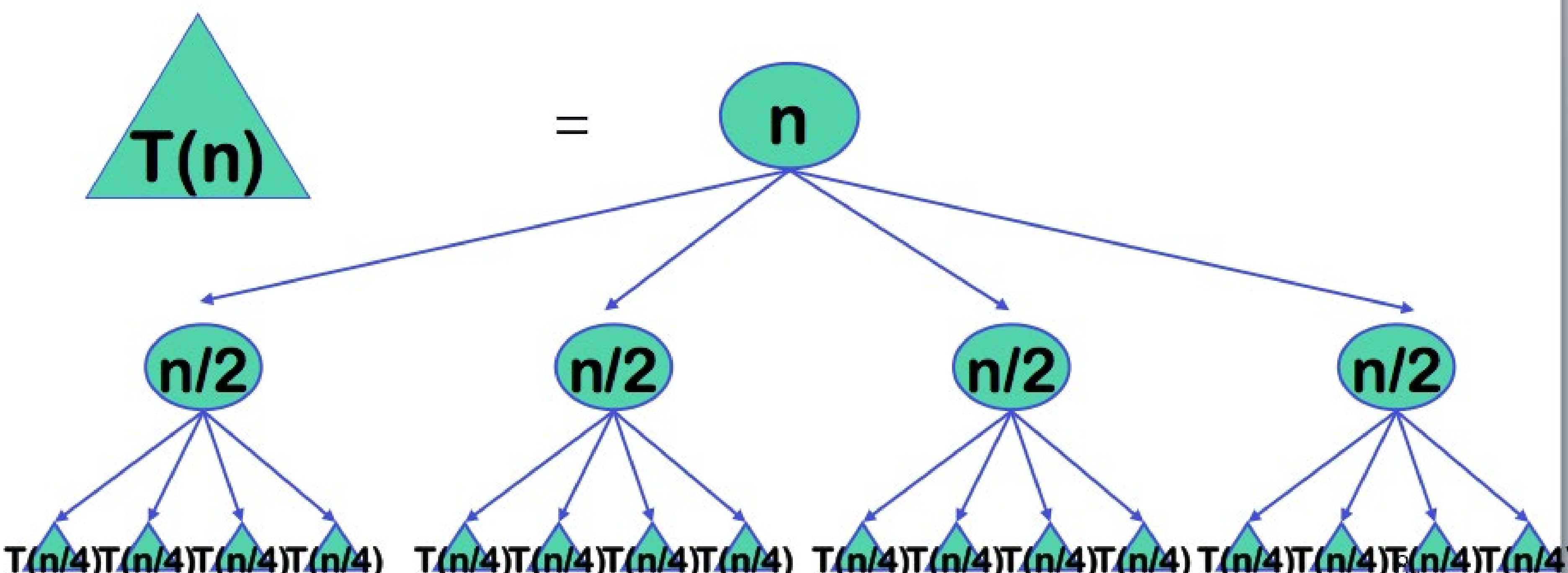


算法总体思想

- 和分治法的区别
  - 主要用于优化问题（求最优解）
  - 子问题并不独立，即子问题是可能重复的
    - 重复的子问题，不需要重复计算

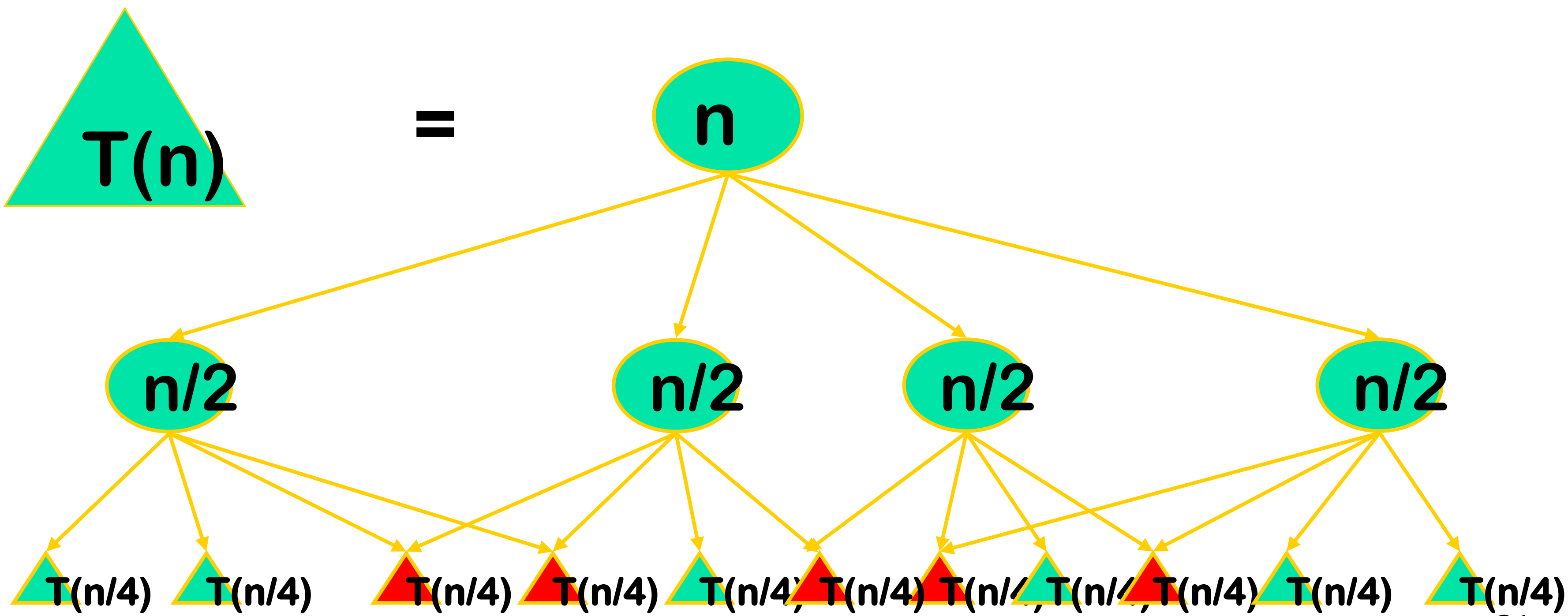
算法总体思想

- 但是经分解得到的子问题往往不是互相独立的。不同子问题的数目常常只有多项式量级。在用分治法求解时，有些子问题被重复计算了许多次。



算法总体思想

- 如果能够保存已解决的子问题的答案，而在需要时再找出已求得的答案，就可以避免大量重复计算，从而得到多项式时间算法。



动态规划的基本思想

- 动态规划的实质是分治和消除冗余，是一种将问题实例分解为更小的、相似的子问题，并存储子问题的解以避免计算重复的子问题，来解决最优化问题的算法策略。

动态规划的基本步骤

- 找出最优解的性质，并刻画其结构特征
- 递归地定义最优值
- 以自底向上的方式计算出最优值
- 根据计算最优值时得到的信息，构造最优解

- 步骤①-③是动态规划的基本步骤。如果只需要求出最优值的情形，步骤④可以省略。
- 若需要求出问题的一个最优解，则必须执行步骤④，步骤③中记录的信息是构造最优解的基础；

动态规划实例

- 最长公共子序列
- 矩阵链相乘
- 平面凸多边形最优三角划分
- 背包问题

最长公共子序列问题

- 给定两个定义在字符集∑上的字符串A和B，长度分别为n和m，现在要求它们的最长公共子序列的长度值(最优值)，以及对应的子序列(最优解)。

最长公共子序列问题

- 子序列

$A = a_1 a_2 \cdots a_n$ 的一个子序列是形如下式的一个字符串：$a_{i_1} a_{i_2} \cdots a_{i_k}$,其中 $1 \le i_1 < i_2 < \cdots < i_k \le n$

例如：$A = zxy$ 子序列可以是 : “”, z, x, y, zx, zy, xy, zxy。

但是xz, yz, xyz不是它的子序列。

$$C_n^0 + C_n^1 + \ldots + C_n^n = 2^n$$

- 最长公共子序列

例如：A=zxyxyz, B=xxyyzx, 最长公共子序列是 xyyz

最长公共子序列问题

- 穷举法(Brute-Force)：
  - 找出A字符串所有可能的子序列(2^n);
  - 对于A的每一个子序列， 判断其是否是B的一个子序列,需要的时间为$\Theta(m)$;
  - 求max， 总的时间为Θ ($m\ 2^n$).

最长公共子序列问题

- 动态规划法：
  - 对于问题$\boldsymbol{A} =<A_1A_2\ldots A_n>$，$\boldsymbol{B} =<B_1B_2\ldots B_m>$，不直接求解最长公共子序列，而是获取它的长度；获取长度后再扩张算法以求解最长公共子序列LCS
  - 定义$L[i,j]$表示子序列$<A_1A_2\ldots A_i>$和$<B_1B_2\ldots B_j>$的最长公共子序列长度，则$L[i,0] = L[0,j] = 0$
  - 递归求解原始序列$\boldsymbol{A}$和$\boldsymbol{B}$的最长公共子序列长度$L[n,m]$（目标）
  - 记录求解每个最长公共子序列长度的状态，用于推导LCS

最长公共子序列问题

- 动态规划法：
  - 对于任意的$L[i,j]$，当$i > 0$且$j > 0$，我们有

如果$A_i = B_j$， $L[i,j] = L[i-1,j-1] + 1$

$$\boxed{\begin{array}{c} <A_1A_2\ldots A_{i-1}A_i> \\ <B_1B_2\ldots B_{j-1}A_i> \end{array}}$$

如果$A_i \neq B_j$， $L[i,j] = \max\{L[i,j-1], L[i-1,j]\}$

$$\boxed{\begin{array}{c} <A_1A_2\ldots A_{i-1}A_i> \\ <B_1B_2\ldots B_{j-1}B_j> \end{array}} \Rightarrow \boxed{\begin{array}{c} <A_1A_2\ldots A_{i-1}A_i> \\ <B_1B_2\ldots B_{j-1}> \end{array}} \quad \boxed{\begin{array}{c} <A_1A_2\ldots A_{i-1}> \\ <B_1B_2\ldots B_{j-1}B_j> \end{array}}$$

最长公共子序列问题

- $\boldsymbol{A}$ =<$A_1A_2\ldots A_n$>， $\boldsymbol{B}$ =<$B_1B_2\ldots B_m$>

$$L[i,j]=\begin{cases}0 & 若\ i=0\ 或\ j=0\\ L[i-1,j-1]+1 & 若\ i>0,j>0\ 和\ a_i=b_j\\ \max\{L[i,j-1],L[i-1,j]\} & 若\ i>0,j>0\ 和\ a_i\neq b_j\end{cases}$$

i \ *j*	**0**	**1**	**2**	**3**	**…**	***m*-1**	***m***
0	0	0	0	0	0	0	0
1	0						
2	0						
…	0						
***n*-1**	0						
n	0						??

最长公共子序列问题

- $\boldsymbol{A} =<A_1A_2\ldots A_n>$，$\boldsymbol{B} =<B_1B_2\ldots B_m>$

$$L[i,j]=\begin{cases}0 & \text{若 } i=0 \text{ 或 } j=0\\ L[i-1,j-1]+1 & \text{若 } i>0,j>0 \text{ 和 } a_i=b_j\\ \max\{L[i,j-1],L[i-1,j]\} & \text{若 } i>0,j>0 \text{ 和 } a_i\neq b_j\end{cases}$$

i \ *j*	0	1	2	3	…	*m*-1	*m*
0	0	0	0	0	0	0	0
1	0	$\triangle$	$\triangle$				
2	0	$\triangle$	$\triangle$				
…	0						
n-1	0						
n	0						??

$$L[i,j]=\begin{cases}0 & \text{若 } i=0 \text{ 或 } j=0\\ L[i-1,j-1]+1 & \text{若 } i>0,j>0 \text{ 和 } a_i=b_j\\ \max\{L[i,j-1],L[i-1,j]\} & \text{若 } i>0,j>0 \text{ 和 } a_i\neq b_j\end{cases}$$

- 例子

$\boldsymbol{A}=<A,B,C,B,D,A,B>$

$\boldsymbol{B}=<B,D,C,A,B,A>$

j		0	1	2	3	4	5	6
i		B_j	B	D	C	A	B	A
0	A_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

算法 7.1　LCS

输入： 字母表Σ上的两个字符串 A 和 B，长度分别为 n 和 m。

输出： A 和 B 最长公共子序列的长度。

```
for i ← 0 to n
    L[i,0] ← 0
end for
for j ← 0 to m
    L[0,j] ← 0
end for
for i ← 1 to n
    for j ← 1 to m
        if a_i = b_j then L[i,j] ← L[i-1,j-1] + 1
        else L[i,j] ← max{L[i,j-1], L[i-1,j]}
        end if
    end for
end for
return L[n,m]
```

初始化

$$T(n) = \Theta(nm)$$

自底向上计算

最长公共子序列问题

- 问题$\boldsymbol{A}$ =<$A_1A_2\ldots A_n$>，$\boldsymbol{B}$ =<$B_1B_2\ldots B_m$>
- 空间复杂度：能否只使用一维数组？
- 如何从长度得到最长公共子序列？

$$L[i,j]=\begin{cases}0 & \text{若 } i=0 \text{ 或 } j=0\\ L[i-1,j-1]+1 & \text{若 } i>0, j>0 \text{ 和 } a_i=b_j\\ \max\{L[i,j-1],L[i-1,j]\} & \text{若 } i>0, j>0 \text{ 和 } a_i\neq b_j\end{cases}$$

- 空间复杂度：能否只使用一维数组?

	j	0	1	2	3	…	m-1	m
i	0	0	0	0	0	0	0	0
	1	0	▲	▲				
	2	0		▲				
	3	0						
	…	0						
	n-1	0						
	n	0						??

最长公共子序列问题

```
LCS( )
{  Q[1,2,…., n] = 0;
   for(j=1; j≤m; j++)
   {     if(B_j=A_1)   T=1;
         else   T = Q[1];
         for(i=2; i≤n; i++)
         {   if(B_j=A_i){ S = Q[i-1]+1; Q[i-1] = T;  T = S;}
             else { Q[i-1] = T; T =  max{T, Q[i]};}
         }
         Q[n] = T;
   }
   return Q[n];
}
```

$$L[i,j]=\begin{cases}0 & \text{若 } i=0 \text{ 或 } j=0\\ L[i-1,j-1]+1 & \text{若 } i>0,j>0 \text{ 和 } a_i=b_j\\ \max\{L[i,j-1],L[i-1,j]\} & \text{若 } i>0,j>0 \text{ 和 } a_i\neq b_j\end{cases}$$

- 如何从长度得到最长公共子序列？

j		0	1	2	3	4	5	6
i		B_j	B	D	C	A	B	A
0	A_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

完全加括号的矩阵连乘积

- 完全加括号的矩阵连乘积可递归地定义为：

（1）单个矩阵是完全加括号的；
（2）矩阵连乘积A是完全加括号的，则A可表示为2个完全加括号的矩阵连乘积B和C的乘积并加括号，即 $A=(BC)$

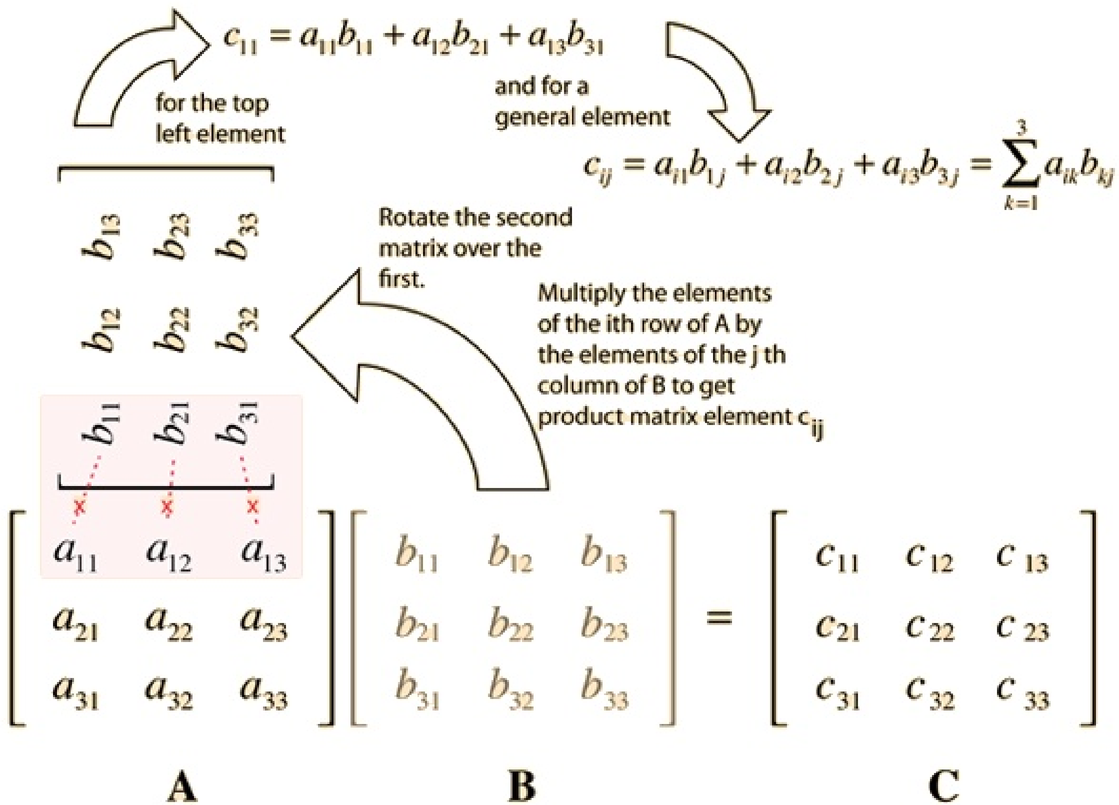
完全加括号的矩阵连乘积

- 设有四个矩阵 A, B, C, D
- 总共有五种完全加括号的方式：

$$(A((BC)D)) \qquad (A(B(CD))) \qquad ((AB)(CD))$$

$$(((AB)C)D) \qquad ((A(BC))D)$$

矩阵乘法


$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$
for the top left element
and for a general element
$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j} = \sum_{k=1}^{3} a_{ik}b_{kj}$
Rotate the second matrix over the first.
Multiply the elements of the ith row of A by the elements of the j th column of B to get product matrix element c_{ij}
$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$
A
B
C

Example

$$A = \quad \underset{10 \times 20}{A_1} \quad \underset{20 \times 50}{A_2} \quad \underset{50 \times 1}{A_3} \quad \underset{1 \times 100}{A_4}$$

- Order 1

$$A_1 \times (A_2 \times (A_3 \times A_4))$$
$$\text{Cost}(A_3 \times A_4) = 50 \times 1 \times 100$$
$$\text{Cost}(A_2 \times (A_3 \times A_4)) = 20 \times 50 \times 100$$
$$\text{Cost}(A_1 \times (A_2 \times (A_3 \times A_4))) = 10 \times 20 \times 100$$

$$\text{Total Cost} = 125000$$

- Order 2

$$(A_1 \times (A_2 \times A_3)) \times A_4$$
$$\mathrm{Cost}(A_2 \times A_3) = 20 \times 50 \times 1$$
$$\mathrm{Cost}(A_1 \times (A_2 \times A_3)) = 10 \times 20 \times 1$$
$$\mathrm{Cost}((A_1 \times (A_2 \times A_3)) \times A_4) = 10 \times 1 \times 100$$
$$\text{Total Cost} = 2200$$

矩阵链相乘

- 给定n个连乘的矩阵$A_1 \cdot A_2 \ldots A_{n-1} \cdot A_n$，问：所需要的最小乘法次数(最优值)是多少次？对应此最小乘法次数，矩阵是按照什么结合方式相乘(最优解)的？

$(A)_{p\times q} \cdot (B)_{q\times r}$ 所需要的乘法次数为：$p\times q\times r$

观察结论：多个矩阵连乘时，相乘的结合方式不同，所需要的乘法次数大不相同。

矩阵链相乘

$A_1 \cdot A_2 \ldots A_{n-1} \cdot A_n$ 按照何种结合方式相乘，所需要的乘法次数最少？

穷举(蛮力)法：

1.找出所有可能的相乘结合方式;
2.计算每种相乘结合方式所需要的乘法次数;
3.求min;

算法复杂度

$f(n)$ 表示 n 个矩阵连乘所有可能的结合方式,下面设法求出其解析解。

$$\underbrace{(A_1 \cdot A_2 \cdot A_3 \ldots A_k)}_{f(k)} \underbrace{(A_{k+1} \ldots A_n)}_{f(n-k)}$$

$$f(n) = \sum_{k=1}^{n-1} f(k) \cdot f(n-k)$$

$$f(1)=1, f(2)=1, f(3)=2 \longrightarrow f(n) = \frac{1}{n} C_{2n-2}^{n-1} \xrightarrow{n! \approx \sqrt{2\pi n}(n/2)^n}$$

$$f(n) = \frac{(2n-2)!}{n((n-1)!)^2} \approx \frac{4^n}{4\sqrt{\pi} n^{1.5}} \longrightarrow f(n) = \Omega\left(\frac{4^n}{n^{1.5}}\right)$$

结论:

穷举法时间复杂度太高。

$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16\ 796, \cdots$

矩阵链相乘

◆动态规划

将矩阵连乘积 $A_iA_{i+1}\ldots A_j$ 简记为$A[i:j]$ ，这里$i\leq j$

数组$r[1,2,\ldots,n+1]$中存储每个矩阵的行数和列数

矩阵A_i 的维度为 $r_i\times r_{i+1}$；矩阵A_n 的维度为 $r_n\times r_{n+1}$

矩阵链相乘

考察计算$A[i:j]$的最优计算次序。设这个计算次序在矩阵A_{k-1}和A_k之间将矩阵链断开，$i < k \le j$，则其相应完全加括号方式为$(A_i A_{i+1} \ldots A_k)(A_{k+1} A_{k+2} \ldots A_j)$，这两个子矩阵乘也必须是最优的

计算量：$A[i:k-1]$的计算量加上$A[k:j]$的计算量，再加上$A[i:k-1]$和$A[k:j]$相乘的计算量

分析最优解的结构

- 特征：计算$A[i:j]$的最优次序所包含的计算矩阵子链 $A[i:k-1]$和$A[k:j]$的次序也是最优的。
- 矩阵连乘计算次序问题的最优解包含着其子问题的最优解。这种性质称为**最优子结构性质**。问题的最优子结构性质是该问题可用动态规划算法求解的显著特征。

建立递归关系

- 计算$A[i:j]$，$1 \le i \le j \le n$，所需要的最少数乘次数$C[i,j]$，则原问题的最优值为$C[1,n]$
- 当$i=j$时，$A[i:j] = A_i$，因此，$C[i,i]=0$，$i=1,2,\ldots,n$
- 当$i<j$时，设k为最优断开点

$$C[i,j] = C[i,k-1] + C[k,j] + r_i r_k r_{j+1}$$

矩阵A_i的维度为$r_i \times r_{i+1}$

- $(A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5 \quad A_6)$

C[1,3] C[4,6]

$(A_1 \quad A_2 \quad A_3) \; (A_4 \quad A_5 \quad A_6)$

建立递归关系

- 可以递归地定义$C[i,j]$为：

$$C[i,j]=\min_{i<k\leq j}\{C[i,k-1]+C[k,j]+r_i r_k r_{j+1}\}$$

k的位置只有 $j-i$ 种可能

计算最优值

- 如果通过简单的递归来求解此问题，其复杂度为

$$T(n) \geq 1 + \sum_{1 \leq k < n} (T(k) + T(n-k) + 1) = \Omega(2^n), \text{ for } n > 1$$

计算最优值

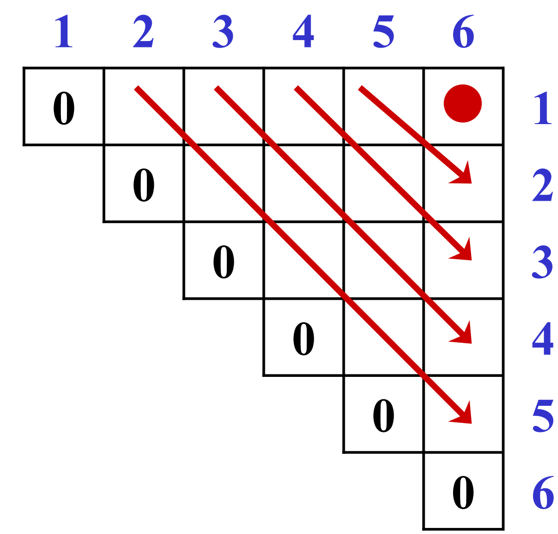
- 对于1≤i≤j≤n不同的有序对(i,j)对应于不同的子问题。因此，不同子问题的个数最多只有

$$\binom{n}{2}+n=\Theta(n^2)$$

- 由此可见，在递归计算时，**许多子问题被重复计算多次**。这也是该问题可用动态规划算法求解的又一显著特征。
- 用动态规划算法解此问题，可依据其递归式以自底向上的方式进行计算。在计算过程中，保存已解决的子问题答案。每个子问题只计算一次，而在后面需要时只要简单查一下，从而避免大量的重复计算，最终得到多项式时间的算法

计算最优值

- 计算所有的 ***C[i,j]***:
  --Start by setting ***C[i,i]***=0 for ***i*** = 1,…,n.
  --Then compute ***C***[1,2], ***C***[2,3],…,***C***[*n-1*,*n*].
  --Then ***C***[1,3], ***C***[2,4],…,***C***[*n-2*,*n*],…
  --… so on till we can compute ***C***[1,*n*].



$$C[i,j]=\min_{i<k\leq j}\{C[i,k-1]+C[k,j]+r_i r_k r_{j+1}\}$$

矩阵连乘问题

输入： r[1..n+1]，表示n个矩阵规模的n+1个整数.

输出： n个矩阵连乘的最小乘法次数.

for i←1 to n {填充对角线d_0}
  C[i,i] ←0
end for
for d←1 to n-1 {填充对角线d_1到d_{n-1}}
  for i←1 to n-d {填充对角线d_i的每个项目}
    j←i+d {该对角线上j,i满足的关系}
    C[i,j] ←∞
    for k←i +1 to j
      C[i,j] ←min{ C[i,j], C[i,k-1]+ C[k,j]+ $r_i \times r_k \times r_{j+1}$}
    end for
  end for
end for
return C[1,n]

武漢：$C[i,j]=\min\limits_{i<k\leq j}\{C[i,k-1]+C[k,j]+r_i r_k r_{j+1}\}$

$(M_1)_{5\times10}\cdot(M_2)_{10\times4}\cdot(M_3)_{4\times6}\cdot(M_4)_{6\times10}\cdot(M_5)_{10\times2}$ $r_1=5,r_2=10,r_3=4,r_4=6,r_5=10,r_6=2$

$d=0$ $d=1$ $d=2$ $d=3$ $d=4$

C[1,1]=0 (M_1)	C[1,2]=200 $(M_1 M_2)$	C[1,3]=320	C[1,4]=620	C[1,5]=348
	C[2,2]=0 (M_2)	C[2,3]=240 $(M_2 M_3)$	C[2,4]=640 $(M_2)(M_3M_4)$	C[2,5]=248
		C[3,3]=0 (M_3)	C[3,4]=240 (M_3M_4)	C[3,5]=168
			C[4,4]=0 (M_4)	C[4,5]=120 $(M_4 M_5)$
				C[5,5]=0 (M_5)

$$C[2,4]=\min_{2<k\leq4}\{C[2,k-1]+C[k,4]+r_2\cdot r_k\cdot r_{4+1}\}$$

$$\left.\begin{array}{l} k=3 \rightarrow C[2,4]=C[2,2]+C[3,4]+r_2\cdot r_3\cdot r_{4+1}=0+240+10\cdot4\cdot10=640\rightarrow(M_2)\cdot(M_3\cdot M_4) \\ k=4 \rightarrow C[2,4]=C[2,3]+C[4,4]+r_2\cdot r_4\cdot r_{4+1}=240+0+10\cdot6\cdot10=840\rightarrow(M_2\cdot M_3)\cdot(M_4) \end{array}\right\}\min$$

算法复杂度

$$T(n)=\sum_{d=1}^{n-1}\sum_{i=1}^{n-d}\sum_{k=i+1}^{j}c=\sum_{d=1}^{n-1}\sum_{i=1}^{n-d}\sum_{k=i+1}^{i+d}c=\sum_{d=1}^{n-1}\sum_{i=1}^{n-d}\sum_{k=1}^{d}c=\Theta(n^3)$$

$$T(n)=\sum_{d=1}^{n-1}\sum_{i=1}^{n-d}\sum_{k=i+1}^{j}c=\sum_{d=1}^{n-1}\sum_{i=1}^{n-d}\sum_{k=i+1}^{i+d}c=\sum_{d=1}^{n-1}\sum_{i=1}^{n-d}\sum_{k=1}^{d}c$$

$$=\sum_{d=1}^{n-1}\sum_{i=1}^{n-d}cd=c\sum_{d=1}^{n-1}\sum_{i=1}^{n-d}d$$

$$=c\left(\sum_{i=1}^{n-1}1+\sum_{i=1}^{n-2}2+\sum_{i=1}^{n-3}3+\ldots+\sum_{i=1}^{n-(n-1)}(n-1)\right)$$

$$=c\big((n-1)\cdot 1+(n-2)\cdot 2+(n-3)\cdot 3+\ldots+(n-(n-1))\cdot(n-1)\big)$$

$$=c\big(n\cdot 1+n\cdot 2+n\cdot 3+\ldots+n\cdot(n-1)-1\cdot 1-2\cdot 2-(n-1)\cdot(n-1)\big)$$

$$=c\left(n\big(1+2+3+\ldots+(n-1)\big)-\sum_{k=1}^{n-1}k^2\right)$$

$$=c\left(n\cdot\frac{n(n-1)}{2}-\frac{1}{6}(n-1)n(2n-1)\right)$$

$$=\frac{1}{6}\left(cn^3-cn\right)\qquad=\Theta\left(n^3\right)$$

蛮力方法：

$$T(n)=\Omega\left(\frac{4^n}{n^{1.5}}\right)$$