# 第3章 动态规划

# 提　纲

- 引例
- 动态规划
- 矩阵连乘问题
- 最长公共子序列
- 最短路径的Floyd算法
- 0-1背包问题
- 最优二叉搜索树
- 总结

# 引　例

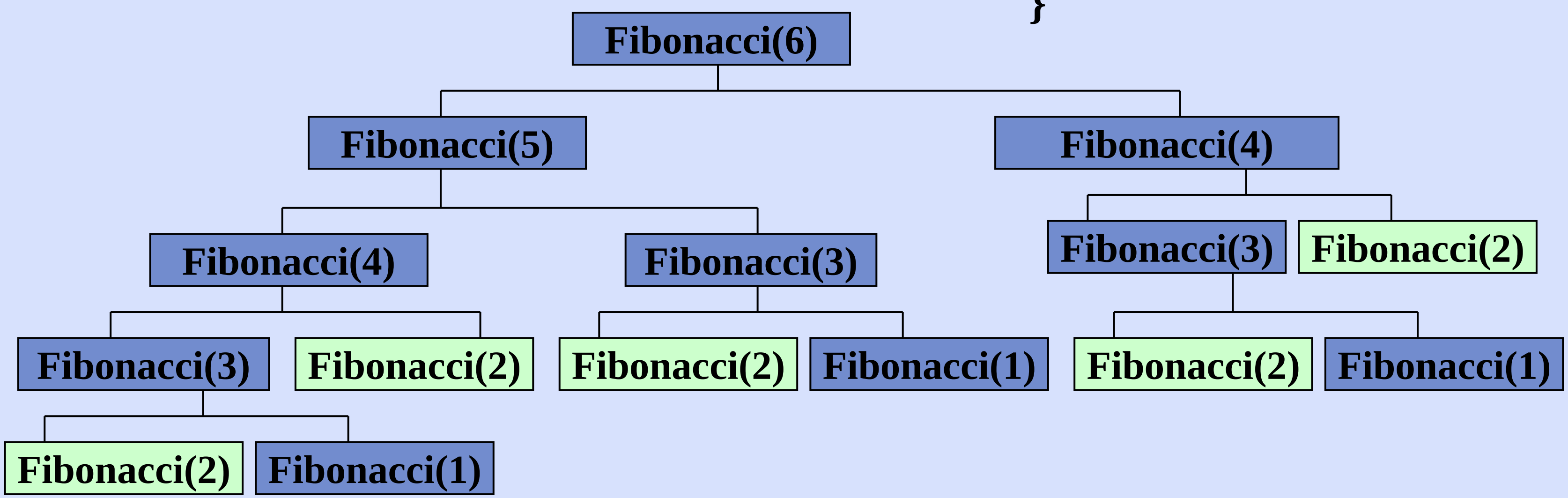
- **Fibonacci序列：1,1,2,3,5,8,13, ….。**

  递归定义为：

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

- **分治算法（递归）：**

```
int f(n){
 if(n==1)||(n==2) {
   return 1;
 }else{
   return( f(n-1)+f(n-2) );
 }
}
```



## ❖递归方程

$$T(n)=\begin{cases} 1 & n=1\text{或}n=2 \\ T(n-1)+T(n-2)+1 & n>=3 \end{cases}$$

$$T(n)\approx O(\phi^{n}),\text{其中}\phi=1.618$$

原因：对函数重复调用，并且重复调用数量巨大，导致 $T(n)$ 为指数阶

不是有效的算法！

$$\Phi^{43}\approx 2^{30}\approx 10^{9}\ flo=1\sec$$

$$\Phi^{67}\approx 10^{14}flo=10^{5}\sec=1day$$

$\Phi^{92}\approx ?$　三百年

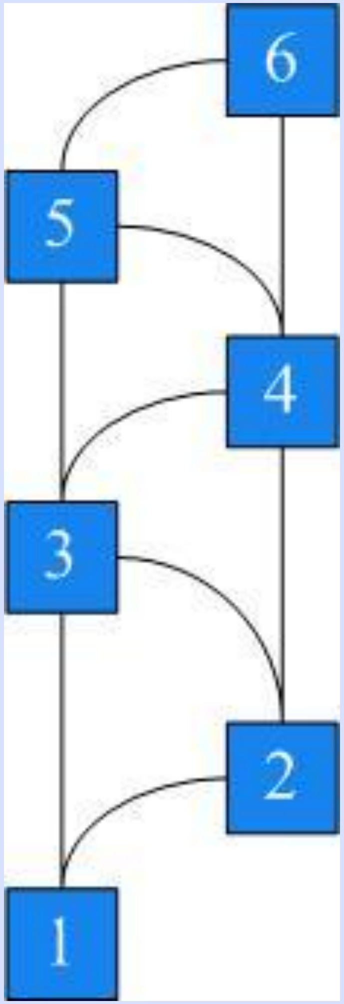
## 两种改进的方法

改进思路：子问题不重复计算，每个子问题只需计算一次。

### 备忘录方法

将已经计算过实例的结果制表备查

| Fib(1) | Fib(2) | Fib(3) | Fib(4) | Fib(5) | Fib(6) | …….. |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 5 | -1 | -1 |




### 动态规范

颠倒计算方向：由自顶向下递归，为自底向上迭代

## ❖O(n) 阶的算法

输入：n　　输出：f(n)

s[1]=1; s[2]=1;

for(i=3;i<=n;i++) s[i]=s[i-1]+s[i-2];

输出s[n];

特点：

(1) 给出原问题解和子问题解之间的关系

(2)首先解决子问题，解决问题的过程是自底向上的

(3)为了在后面计算中利用,子问题的解需要记录

## ❖O(logn) 阶的算法

定理：设{Fn}为Fibonacci数构成的数列，那么

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

证明：对n进行归纳

当n=1时，结果显然。

假设对于n定理成立，即：$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$

那么，当n+1时

因$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} F_{n+1}+F_n & F_{n+1} \\ F_n+F_{n-1} & F_n \end{bmatrix} = \begin{bmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{bmatrix}$

由假设$\begin{bmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n+1}$

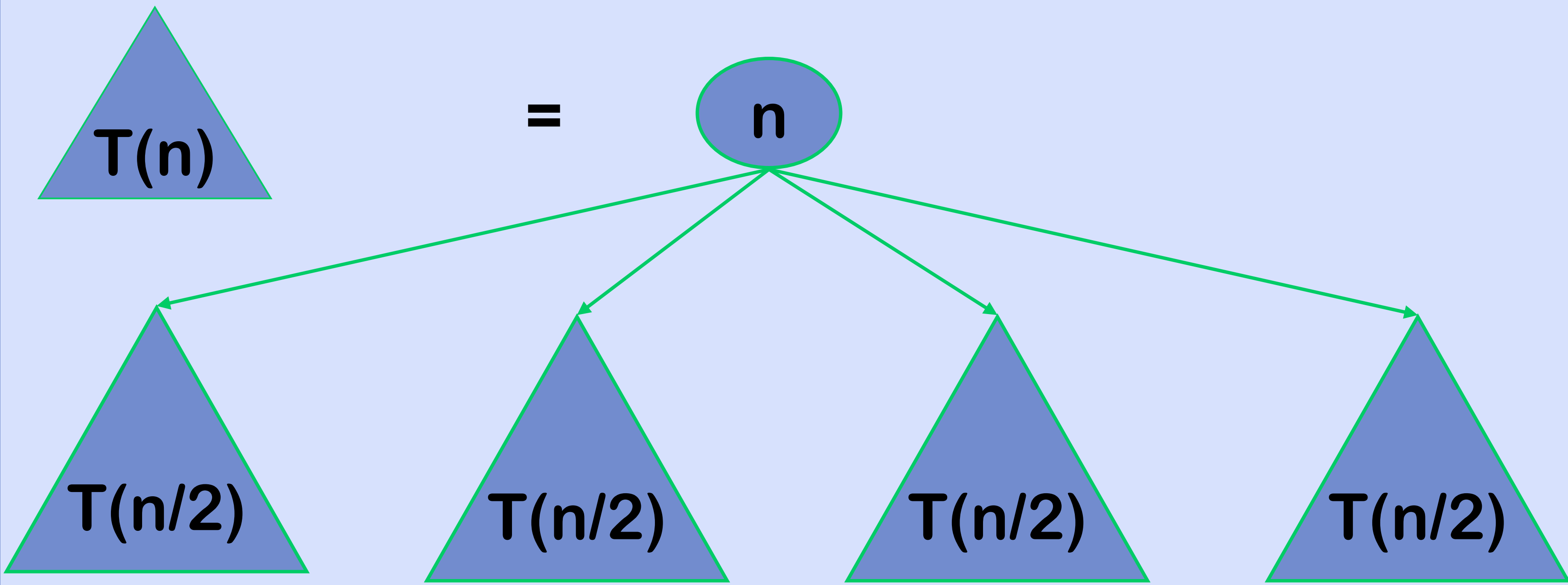
定理得证。

# 提 纲

- 引例
- 动态规划
- 矩阵连乘问题
- 最长公共子序列
- 图像压缩
- 最短路径的Floyd算法
- 0-1背包问题
- 最优二叉搜索树
- 总结

# 动态规划

- 动态规划(Dynamic Programming)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



❖动态规划的分解

–经分解得到的子问题往往不是互相独立的

引例：

$$f(n)=f(n-1)+f(n-2)$$

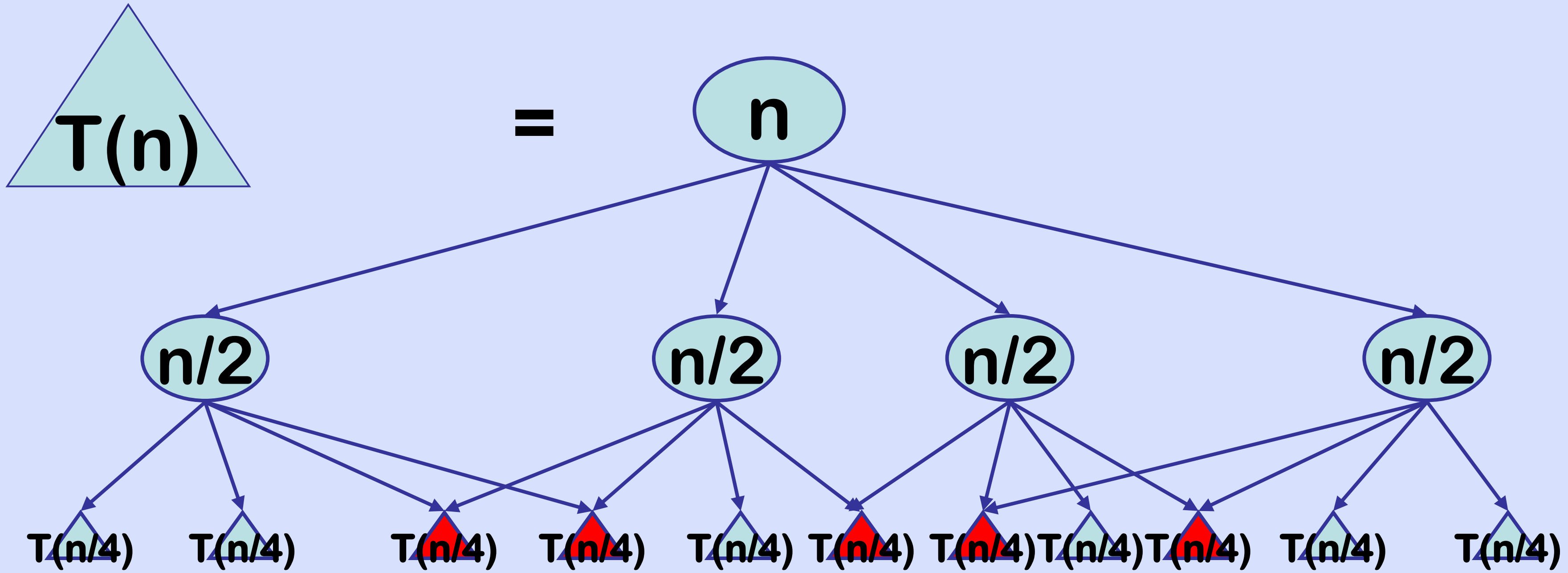
在用分治法求解时，有些子问题被重复计算了许多次

–不同子问题的数目常常只有多项式数量级

引例：n个子问题

## ❖基本思想

- 分治，子问题之间存在依赖关系
- 保存已解决的子问题的答案，以备后面求解，利用已得到的子问题的答案，构造待求解的大规模问题的答案
- 可以避免大量重复计算，从而得到多项式时间算法



## ❖动态规划的基本步骤

动态规划算法适用于解最优化问题（多个可行解解，每个解有一个值，找最优值）

例：**0-1**背包问题，**x1,x2….xn**，可行解，最优解，最优值

- 找出最优解的性质，并刻划其结构特征（最优子结构）
- 递归地定义最优值（问题最优值和子问题最优值的关系）
- 以自底向上的方式计算出最优值（填表）
- 根据计算最优值时得到的信息，构造最优解

# 提　纲

- 引例
- 动态规划
- 矩阵连乘问题
- 最长公共子序列
- 图像压缩
- 最短路径的Floyd算法
- 0-1背包问题
- 最优二叉搜索树
- 总结

# 矩阵连乘问题

## ❖ 引例

利用标准的矩阵乘法计算矩阵$M_1(2\times10)$, $M_2(10\times2)$, $M_3$ $(2\times10)$的乘积

(1) $(M_1M_2)M_3$: $2\times10\times2+2\times2\times10=80$ 次乘法

(2) $M_1(M_2M_3)$: $2\times10\times10+10\times2\times10=400$ 次乘法

**结论：不同的乘法执行顺序，乘法次数相差很大！**

## ❖ 矩阵连乘问题

**给定$n$个矩阵$\{A_1, A_2, \ldots, A_n\}$，其中$A_i$和$A_{i+1}$可乘, $i=1, 2, \ldots, n-1$, 确定这$n$个矩阵乘积的计算次序，使得所需乘法次数最少**

**说明：**

- 矩阵乘法满足结合律，**连乘的计算次序可由加括号方式确定**
- 计算次序完全确定——完全加括号——按此次序进行2个矩阵相乘

- 穷举搜索法

设不同加括号（计算次序）的方式为$P(n)$, 前$k$个矩阵有$P(k)$种加括号方式，对每一个$k$，有$P(k)P(n-k)$种加括号方式

$$(A_1A_2\cdots A_k)\times(A_{k+1}A_{k+2}\cdots A_n)$$

$$P(n)=\begin{cases}1 & n=1\\ \sum_{k=1}^{n-1}P(k)P(n-k) & n>1\end{cases}$$

$$P(n)=\frac{1}{n}C_{n-1}^{2n-2}=\frac{(2n-2)!}{n((n-1)!)^2}\approx\frac{4^n}{4\sqrt{\pi}n^{1.5}}$$

**1,1,2,5,14,42,132,429,1430,4862,16796, …**

**$P(n)$随$n$的增长呈指数增长！**

# 动态规划的方法

- 找出最优解的性质，并刻划其结构特征

用记号$A[i:j]$表示矩阵乘积链$A_iA_{i+1}\ldots A_j$求值，设最后一次计算是在$A_k$和$A_{k+1}$处将矩阵链断开$i\le k<j$，考虑最优计算次序：

$$A_iA_{i+1}\ldots A_j \longrightarrow (A_iA_{i+1}\ldots A_k)\times(A_{k+1}A_{k+2}\ldots A_j)$$

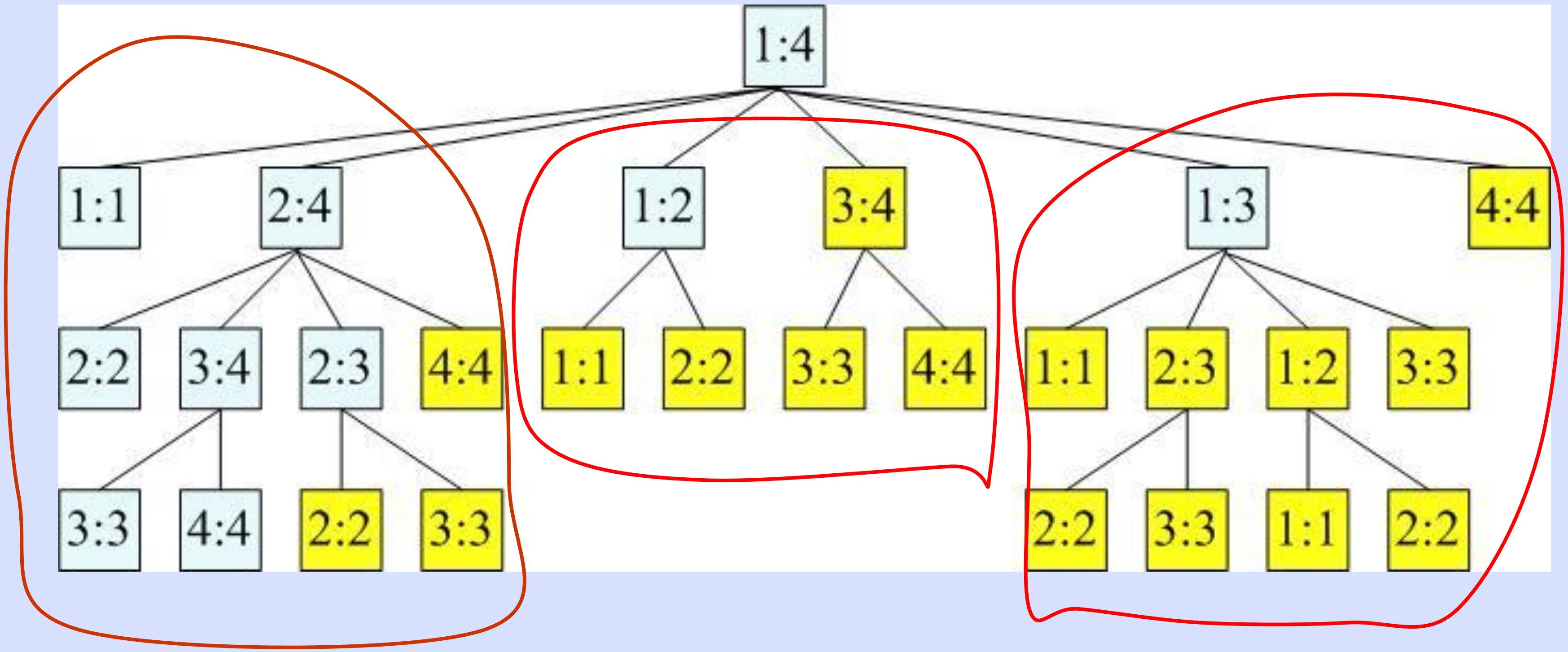
**最优子结构性质**

**计算$A[i:j]$的最优次序所包含的计算矩阵子链 $A[i:k]$和$A[k+1:j]$的次序也是最优的**

**重叠子问题性质?**

例：$A_1 \times A_2 \times A_3 \times A_4$

$k=1,2,3$




子问题：对于$A_1A_2\ldots A_n$的矩阵链，其包含的所有子问题是

$i, j$的不同组合(i<=j): $O(n^2)$

## • 递归地定义最优值

对应于计算$A[i:j]$的最优加括号计算次序，用一个**最优值**$m[i:j]$来表示该计算次序下的乘法次数， $A_i$的维数为$p_{i-1}\times p_i$， $A_1A_2\ldots A_n$的维数表示为$p_0p_1\ldots p_n$。

**$m[i,j]$的递推关系式:**

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

$$m[1,n]=\min_{1\le k<n}\{m[1,k]+m[k+1,n]+p_0p_kp_n\}$$

例： 若$n$=6, $m$(2,5)为以下三个耗费的最小值：

- $m(2,2)+m(3,5)+p_1 \times p_2 \times p_5$
- $m(2,3)+m(4,5)+p_1 \times p_3 \times p_5$
- $m(2,4)+m(5,5)+p_1 \times p_4 \times p_5$

| | | | | | |
|---|---|---|---|---|---|
| m(1,1) | m(1,2) | m(1,3) | m(1,4) | m(1,5) | m(1,6) |
| | m(2,2) | m(2,3) | m(2,4) | **m(2,5)** | m(2,6) |
| | | m(3,3) | m(3,4) | m(3,5) | m(3,6) |
| | | | m(4,4) | m(4,5) | m(4,6) |
| | | | | m(5,5) | m(5,6) |
| | | | | | m(6,6) |

- 考虑两个方向：
  - $m(i, i) \rightarrow m(i, j\text{-}1)$
  - $m(i+1, j) \rightarrow m(j, j)$
- 计算：
  - $m(i, i)+m(i+1, j)$ 到 $m(i, j\text{-}1)+ m(j, j)$，共j-i个
- 决策：
  $\min\{m(i, j)\}, i \le k < j$

## ❖依据其递归式以自底向上的方式计算出最优值（填表）

```
public static void matrixChain(int p [ ],int m [ ][ ],int s[ ][ ])
 {
   int n=p.length-1;
   for(int i=1;i<=n;i++) m[i][i]=0;
   for(int r=2;r<=n;r++)  //r是子问题的长度, 按子序列的长度递增的次序
     for(int i=1;i<=n-r+1;i++){//这个循环计算所有长度为r矩阵链的最优值
       int j=i+r-1;
       m[i][j]=m[i+1][j]+p[i-1]*p[i]*p[j];
       s[i][j]=i;    //记录和最优值对应的k，为第4步构造最优解做准备
       for(int k=i+1;k<j;k++){
         int t=m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j];
         if(t<m[i][j]){
           m[i][j]=t;
           s[i][j]=k;
         }
       }
     }
 }
```

$A_i(A_{i+1}\ldots A_j)$

$(A_i\ldots A_k)(A_{k+1}\ldots A_j)$, $i<k<j$

例：

| $A_1$ | $A_2$ | $A_3$ |
|---|---|---|
| 30×35 | 35×15 | 15×5 |

| $A_4$ | $A_5$ | $A_6$ |
|---|---|---|
| 5×10 | 10×20 | 20×25 |

$\{m(i, j)\}$

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 15750 | 7875 | 9375 | 11875 | 15125 |
| 2 | | 0 | 2625 | 4375 | 7125 | 10500 |
| 3 | | | 0 | 750 | 2500 | 5375 |
| 4 | | | | 0 | 1000 | 3500 |
| 5 | | | | | 0 | 5000 |
| 6 | | | | | | 0 |

- **算法的时间：**

设一次乘法的代价为$c$. 那么 $T(n)=\sum_{r=2}^{n}\sum_{i=1}^{n-r+1}\sum_{k=i}^{j-1} c = O(n^3)$

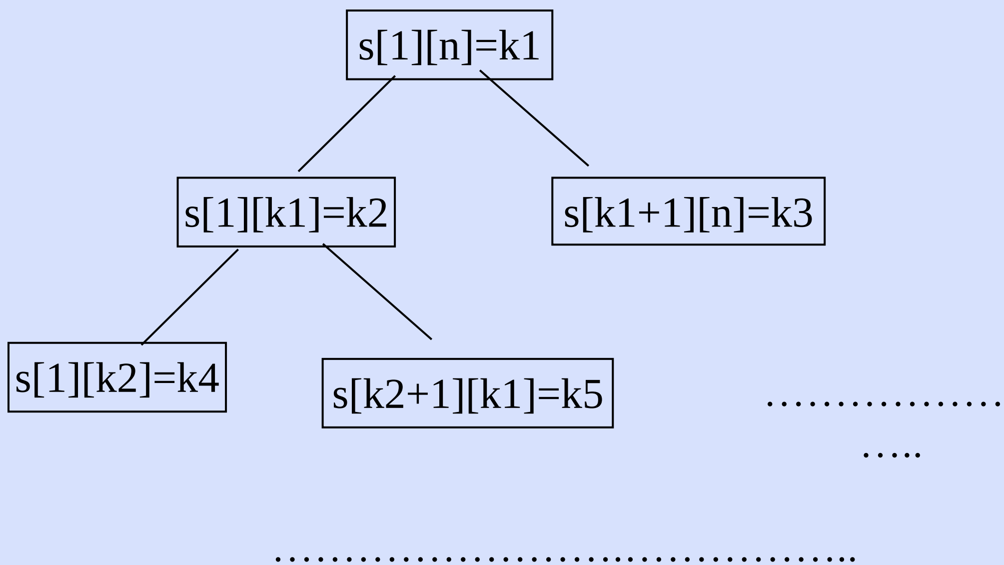
- **所需空间：** $O(n^2)$

注意：这个n是矩阵的个数，通过这样一个工作确定矩阵连乘的计算次序是非常上算的！！

# 根据计算最优值时得到的信息，构造最优解

- *m[i:j]*中计算了矩阵链乘积所需的最优乘法次数，但是，没有直接说明如何相乘。
- *s[i: j]*中已经记录了构造完全加括号的全部信息（递归计算）

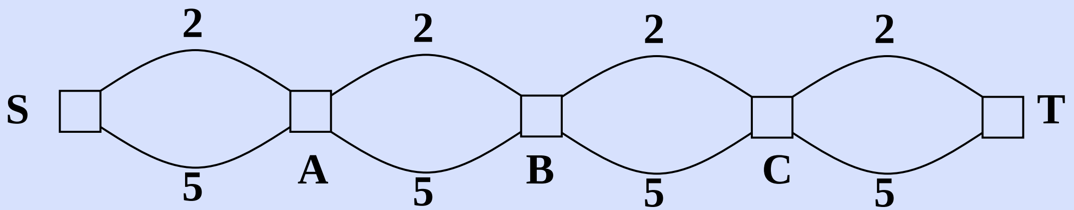
| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 15750 | 7875 | 9375 | 11875 | 15125 |
| 2 | | 0 | 2625 | 4375 | 7125 | 10500 |
| 3 | | | 0 | 750 | 2500 | 5375 |
| 4 | | | | 0 | 1000 | 3500 |
| 5 | | | | | 0 | 5000 |
| 6 | | | | | | 0 |



## ❖动态规划的适用条件

- 最优子结构
  - 问题的最优解包含了其子问题的最优解
  - 最优子结构是问题能用动态规划算法求解的前提
  - 利用问题的最优子结构性质，以自底向上的方式递归地从子问题的最优值逐步构造出整个问题的最优值（自顶向下得到最优解）

例：多段图




考虑：从S到T的最短路径？

从S到T的总长模10的最短路径？

## ❖动态规划的适用条件

- 重叠子问题
  - 通过分治产生的子问题并不总是新问题，有些子问题被反复计算多次
  - 对每一个子问题只解一次，自底向上递归求值，并把中间结果存储起来以便以后用来计算所需要的解
  - 通常不同的子问题个数随问题的大小呈多项式增长（多项式时间）

## ❖备忘录方法

备忘录方法是动态规划方法的变形，与动态规划算法不同的是，备忘录方法采用和分治递归相同的自顶向下的方式，而动态规划算法则是自底向上的。

**方法：**

为每个子问题建立记录项，初始存入一个特殊值，自顶向下解决问题的过程中，如果子问题已经解决（记录项中非特殊值），直接引用其结果；如未解决，计算该子问题的解，并保留。

**对比：**

- 算法控制结构不同
- 所有子问题都需要算，用动态规划。有的子问题可以不必算，用备忘录方法。

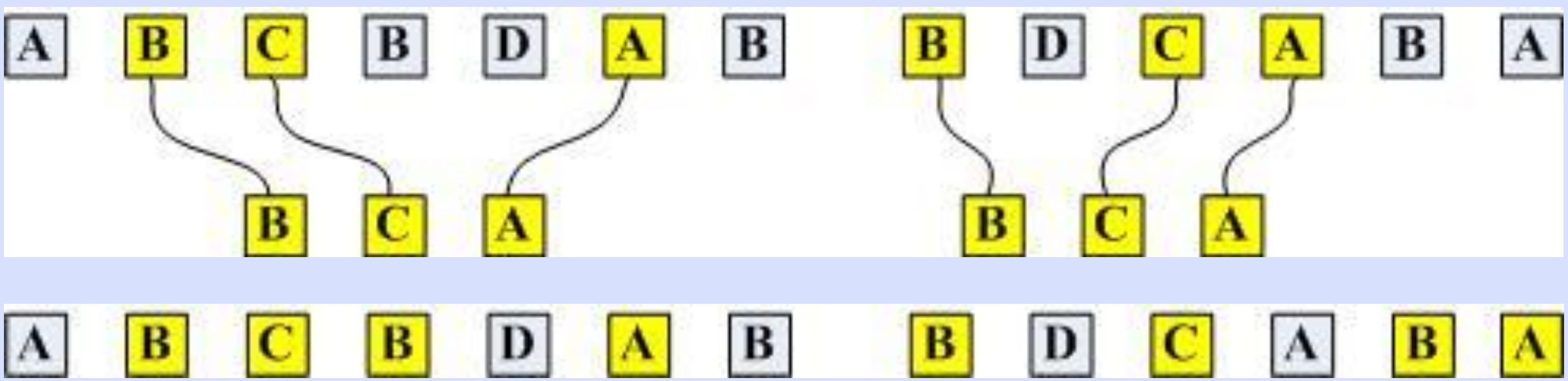
# 提　纲

- 引例
- 动态规划
- 矩阵连乘问题
- 最长公共子序列
- 图像压缩
- 最短路径的Floyd算法
- 0-1背包问题
- 最优二叉搜索树
- 总结

# 最长公共子序列

- **LCS** **(Longest Common Sequences)**
- 引例

子序列：由序列中若干字符，按原相对次序构成




公共子序列

如果∑={*A*,*B*,*C*,*D*}, *X*=*ABCBDAB*, *Y*=*BDCABA*, 序列*BCA*是*X*和*Y*的公共子序列；*BCBA*是最长公共子序列，唯一吗？

# 最长公共子序列

## ❖ 概念

- 给定序列$X=\{x_1, x_2, \ldots, x_m\}$, 序列$Z=\{z_1, z_2, \ldots, z_k\}$是$X$的子序列，如果：存在一个严格递增下标序列$\{i_1, i_2, \ldots, i_k\}$（在序列X中），使得对于所有$j=1,2, \ldots, k$有$z_j=x_{i_j}$
- 序列$Z$同时是$X$和$Y$的子序列，称$Z$是序列$X$和$Y$的公共子序列

## ❖ 问题

给定2个序列$X=\{x_1, x_2, \ldots, x_m\}$和$Y=\{y_1, y_2, \ldots, y_n\}$，找出$X$和$Y$的最长公共子序列

# 最长公共子序列

## 穷举搜索法

- 若$|X|=m$，那么$X$的子序列有$2^m$个，检查每个子序列是否也是$Y$的子序列

- 计算时间至少为$\Theta(2^m)$——指数级

# 动态规划

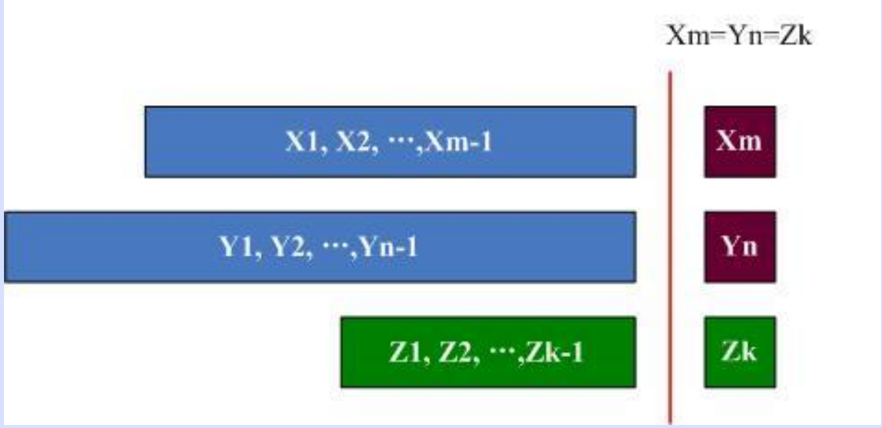
- 最优子结构性质

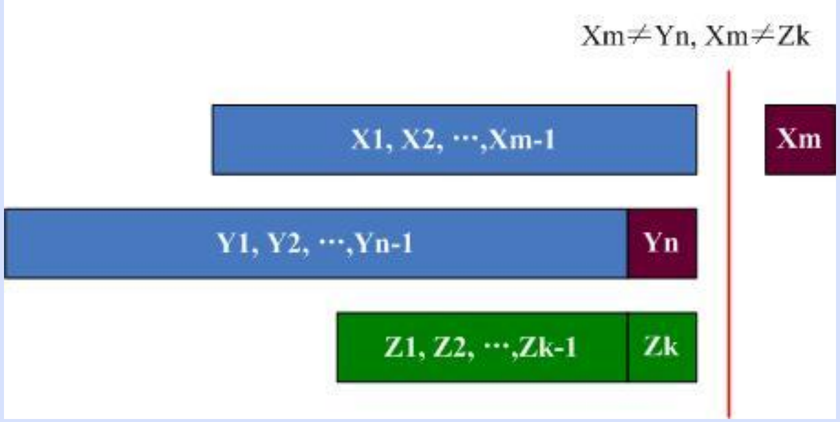
- $X=\{x_1, x_2, \ldots, x_m\}$, $Y=\{y_1, y_2, \ldots, y_n\}$, 假设 $Z=\{z_1, z_2, \ldots, z_k\}$是X和Y的最长公共子序列

(1) 若$x_m=y_n$，则$z_k=x_m=y_n$，且$Z_{k-1}$是$X_{m-1}$和$Y_{n-1}$的最长公共子序列。

(2) 若$x_m \neq y_n$且$z_k \neq x_m$，则$Z$是$X_{m-1}$和$Y$的最长公共子序列。

(3) 若$x_m \neq y_n$且$z_k \neq y_n$，则$Z$是$X$和$Y_{n-1}$的最长公共子序列。




- **结论**：

两个序列的最长公共子序列，包含了这两个序列的前缀的最长公共子序列（即原问题的最优解包含子问题的最优解，具有最优子结构性质）

- 建立递归关系
  - 求序列X和Y的最长公共子序列Z
    (1) 若$X$或$Y$为空，则不存在公共子序列 递归基
    (2) 若$x_m=y_n$，则$Z$=LCS($X_{m\text{-}1}$,$Y_{n\text{-}1}$)+{$x_m$} 分治法
    (3) 若$x_m \neq y_n$，则$Z$=max{LCS($X_{m\text{-}1}$,$Y$)，LCS($X$,$Y_{n\text{-}1}$)} 分治法

- 递推公式， $c\,[i, j]$为$Xi$和$Yj$的最长公共子序列长度

$$c[i,j]=\begin{cases}0 & i=0\text{或}j=0\\ c[i-1,j-1]+1 & i>0,j>0,x_i=y_j\\ \max\{c[i,j-1],c[i-1,j]\} & i>0,j>0,x_i\neq y_j\end{cases}$$

## 考虑计算次序

| | | 0 | 1 | …. | n-1 | n |
|---|---|---|---|---|---|---|
| | | | | | | **Y** |
| | 0 | | | | | |
| | 1 | | | | | |
| | 2 | | | | | |
| | …. | | | | | |
| | | | | | | |
| | m-1 | | | | **c[m-1,n-1]** | **c[m-1,n]** |
| **X** | m | | | | **c[m,n-1]** | **c[m,n]** |

自上而下，从左到右

## ❖计算最优值

b[i,j]记录c[i,j]的来源

<u>Algorithm lcsLength(x[], y[], b[][])</u>

```
m=x.length-1;
n=y.length-1;
c[i][0]=0;  c[0][i]=0;
for(int i=1;i<=m;i++)
  for(int j=1;j<=n;j++)
    if(x[i]==y[j])
      c[i][j]=c[i-1][j-1]+1;
      b[i][j]='↖';
    else if(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]='←';
```

- 计算时间：*O*(*m*n* )
- *b*中为标志，可将*b*省去
- 如何根据最优值得到最优解

- 构造最优解

例： X=xyxxzxyzxy, Y=zxzyyzxxyxxz, 求最长公共子序列长度及公共子序列。

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X | x | y | x | x | z | x | y | z | x | y | | |
| Y | z | x | z | y | y | z | x | x | y | x | x | z |

Y

X

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 ← | 1 ← | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 1 | 2 | 2 ← | 2 ← | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 3 | 3 ← | 3 | 3 | 3 | 3 |
| 4 | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 3 | 4 | 4 | 4 | 4 | 4 |
| 5 | 0 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 4 | 4 | 4 ↑ | 4 | 5 |
| 6 | 0 | 1 | 2 | 2 | 2 | 2 | 3 | 4 | 4 | 4 | 5 | 5 | 5 |
| 7 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 5 | 5 | 5 ↑ | 5 |
| 8 | 0 | 1 | 2 | 3 | 3 | 3 | 4 | 4 | 4 | 5 | 5 | 5 | 6 |
| 9 | 0 | 1 | 2 | 3 | 3 | 3 | 4 | 5 | 5 | 5 | 6 | 6 | 6 ↑ |
| 10 | 0 | 1 | 2 | 3 | 4 | 4 | 4 | 5 | 5 | 6 | 6 | 6 | 6 ↑ |

- 长度: 6
- **xyxxxz**和**zxyzxy**均为*X*和*Y*的最长公共子序列
- 最长公共子序列不唯一

# 用备忘录方法求解LCS

$$c[i,j]=\begin{cases}0 & i=0\text{或}j=0\\ c[i-1,j-1]+1 & i>0,j>0,x_i=y_j\\ \max\{c[i,j-1],c[i-1,j]\} & i>0,j>0,x_i\neq y_j\end{cases}$$

有什么问题？

Y

| | 0 | 1 | …. | n-1 | n |
|---|---|---|---|---|---|
| 0 | | | | | |
| 1 | | | | | |
| 2 | | | | | |
| …. | | | | | |
| | | | | | |
| m-1 | | | | **c[m-1,n-1]** | **c[m-1,n]** |
| m | | | | **c[m,n-1]** | **c[m,n]** |

X

如果Xm=Yn

# 提　纲

- 引例
- 动态规划
- 矩阵连乘问题
- 最长公共子序列
- 图像压缩
- 最短路径的Floyd算法
- 0-1背包问题
- 最优二叉搜索树
- 总结

# 图像压缩

❖ 动机

图像由像素点构成，分辨率高，像素多，图像处理需要大的存储空间和高的处理速度，解决办法---图像压缩

❖ 考虑灰度图像

像素点灰度值：0~255，用8位二进制数表示

图像描述为像素点灰度值序列：$< p_1, p_2, \ldots, p_n >$

❖ 压缩方法

前提：图片中连续区域中像素点的灰度值是接近的

方法：分段存储，值小，位数少，值大，位数多

## ❖ 变位压缩存储格式

将< $p_1, p_2, \ldots, p_n$ >分割 $m$ 段 $S_1, S_2, \ldots, S_m$，设第$i$段有 $l[i]$ 个像素，每个像素用 $b[i]$ 位二进制描述，$h_i$ 为该段最大像素的位数

$$h_i \le b[i] \le 8, \quad h_i = \left\lceil \log(\max_{p_k \in S_i} p_k + 1) \right\rceil$$

约束条件：每段像素个数 $l[i] \le 256$

段头11位： $b[i]$的二进制表示(3 位) + $l[i]$的二进制表示(8位)

第i 段占用空间：$b[i] \times l[i] + 11$

例：灰度值序列 $P$=<10,12,15,255,1,2,1,1,2,2,1,1>

分法1： $S_1$=<10,12,15>，$S_2$=<255>, $S_3$=<1,2,1,1,2,2,1,1>

分法2： $S_1$=<10,12,15,255,1,2,1,1,2,2,1,1>

分法3： 分成12组，每组一个数

存储空间

分法1：(4*3+11)+(8*1+11)+(2*8+11)=69

分法2：11×1+8×12=107

分法3：11×12+4×3+8×1+1×5+2×3=163

结论：分法1是其中最优的分法。怎么获得最优分段？

**问题**

给定像素序列< $p_1, p_2, \ldots, p_n$>，**确定最优分段**，使得依次分段所需的**存储空间最小**，即

$$\min_T\{\sum_{i=1}^{j}(b[i]\times l[i]+11)\},\quad T=\{S_1,S_2,\ldots,S_j\}\text{为分段}$$

**最优子结构性质**

设*l*[i]，b[i]，是{p1,p2,…,pn}的最优分段。

显而易见，*l*[1]，b[1]是{$p_1$,…,$p_{l[1]}$}的最优分段，且*l*[i]，b[i]，是{$p_{l[1]+1}$,…,$p_n$}的最优分段。即图象压缩问题满足最优子结构性质。

## 递归定义最优值

设 $s[i]$ 是像素序列 $\langle p_1, p_2, \ldots, p_i \rangle$ 的最优分段所需存储位数

$$S[1] = b[1,1] + 11$$

$$S[i] = \min_{1 \le j \le \min\{i,256\}} \{S[i-j] + j \times b[i-j+1, i]\} + 11$$

$$b[i-j+1, j] = \left\lceil \log\left(\max_{p_k \in S_m} p_k + 1\right) \right\rceil \le 8$$

$P_1$ $P_2$ … $P_{i-j}$ | $P_{i-j+1}$ … $P_i$

$S[i-j]$位     $j$ 个灰度

$j * b\max(i-j+1, i)$

# 实例

$P=\langle 10, 12, 15, 255, 1, 2\rangle$.

$S[1]=15$, $S[2]=19$, $S[3]=23$, $S[4]=42$, $S[5]=50$，

$l[1]=1$, $l[2]=2$, $l[3]=3$, $l[4]=1$, $l[5]=2$

$b[1]=4$, $b[2]=4$, $b[3]=4$, $b[4]=8$, $b[5]=1$, $b[6]=2$

| | | | | | |
|---|---|---|---|---|---|
| 10 | 12 | 15 | 255 | 1 | 2 |

$S[5]=50$ $1\times 2+11$

| | | | | | |
|---|---|---|---|---|---|
| 10 | 12 | 15 | 255 | 1 | 2 |

$S[4]=42$ $2\times 2+11$

| | | | | | |
|---|---|---|---|---|---|
| 10 | 12 | 15 | 255 | 1 | 2 |

$S[3]=23$ $3\times 8+11$

| | | | | | |
|---|---|---|---|---|---|
| 10 | 12 | 15 | 255 | 1 | 2 |

$S[2]=19$ $4\times 8+11$

| | | | | | |
|---|---|---|---|---|---|
| 10 | 12 | 15 | 255 | 1 | 2 |

$S[1]=15$ $5\times 8+11$

| | | | | | |
|---|---|---|---|---|---|
| 10 | 12 | 15 | 255 | 1 | 2 |

$6\times 8+11$

# 计算最优值

**算法 Compress ($P,n$)** //计算最小位数S[n]

1. $Lmax \leftarrow 256$; $header \leftarrow 11$; $S[0] \leftarrow 0$ //最大段长Lmax, 头header
2. for $i \leftarrow 1$ to $n$ do
3. $\quad b[i] \leftarrow length(P[i])$ //b[i]是第i个灰度P[i]的二进制位数
4. $\quad bmax \leftarrow b[i]$ //3-6行分法的最后一段只有P[i]自己
5. $\quad S[i] \leftarrow S[i-1]+bmax$
6. $\quad l[i] \leftarrow 1$
7. $\quad$ for $j \leftarrow 2$ to $\min\{i, Lmax\}$ do //最后段含 j 个像素，最多执行256次
8. $\qquad$ if $bmax < b[i-j+1]$ //统一段内表示像素的二进制位数
9. $\qquad\quad$ then $bmax \leftarrow b[i-j+1]$
10. $\qquad$ if $S[i] > S[i-j]+j*bmax$
11. $\qquad\quad$ then $S[i] \leftarrow S[i-j]+j*bmax$
12. $\qquad\qquad l[i] \leftarrow j$
13. $S[i] \leftarrow S[i]+header$

时间复杂度 $T(n)=O(n)$

# 构造最优解

**算法 Traceback($n$,$l$)**

输入：数组$l$

输出：数组$C$ // $C[j]$是从后向前追踪的第 $j$段的长度

1. $j \leftarrow 1$ // $j$为正在追踪的段数$C[1]$为最后一段的长度
2. while $n \neq 0$ do
3. $\quad C[j] \leftarrow l[n]$
4. $\quad n \leftarrow n - l[n]$
5. $\quad j \leftarrow j+1$

时间复杂度：$O(n)$

# 提　纲

- 引例
- 动态规划
- 矩阵连乘问题
- 最长公共子序列
- 图像压缩
- 最短路径的Floyd算法
- 0-1背包问题
- 最优二叉搜索树
- 总结

# Floyd 算法（任意顶点对的最短路径）

- **问题**

  寻找带权图中任意顶点对之间的最短路径

- **带权图的邻接矩阵**

  $W(i, j)=0$ 如果 $i=j$

  $W(i, j)=\infty$ 如果 $i$ 和 $j$ 之间没有边

  $W(i, j)=$权值

*Dijkstra* 算法?

- # 子问题

设 $G=(V,E)$, 定义顶点序列$V=\{v_1, v_2, \ldots, v_n\}$

定义一个$n\times n$的矩阵D， $D[i, j]$表示两个顶点之间的“最短路径”

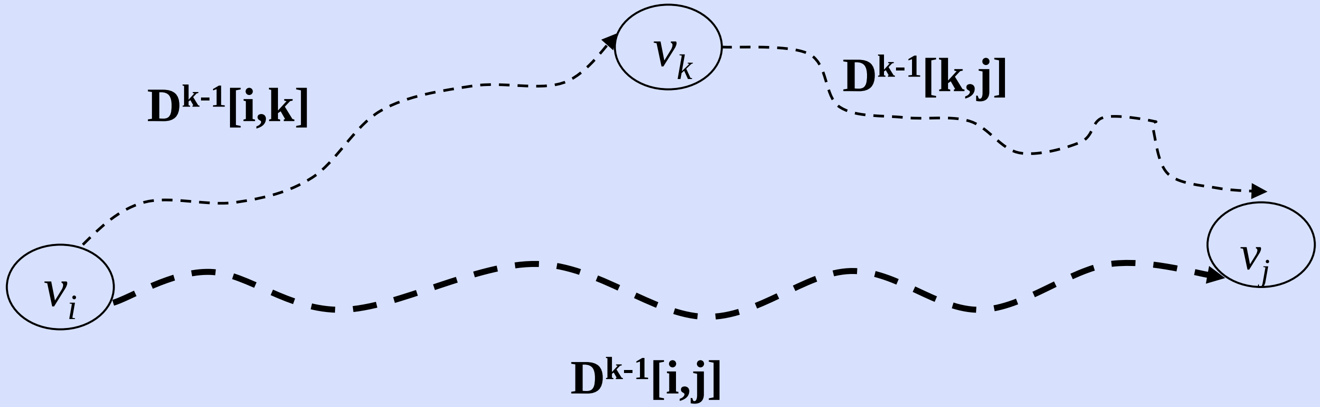
$D^0[i, j]=W[i, j]$ 两个顶点之间不经过其它顶点的最短路径

## 按顶点序列加入顶点进行试探

$V=\{v_1, v_2, \ldots, v_n\}$ 多阶段决策

$D^{(k-1)}[i, j]$两个顶点之间中间顶点序号不大于$k$-1的最短路径

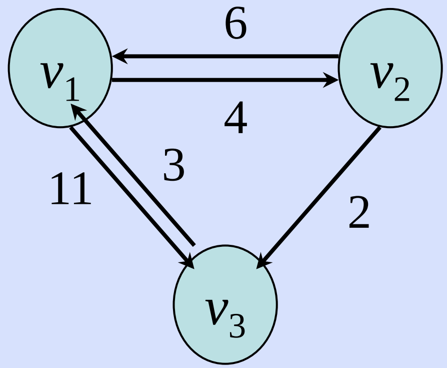
试探加入顶点$v_k$



❖建立递归关系

$$D_{i,j}^{k} = \begin{cases} W[i,j] & 若k=0 \\ \min\left\{D_{i,j}^{k-1}, D_{i,k}^{k-1} + D_{k,j}^{k-1}\right\} & 若1 \le k \le n \end{cases}$$

# 例子




| | 1 | 2 | 3 |
|---|---|---|---|
| **1** | **0** | **4** | **11** |
| **2** | **6** | **0** | **2** |
| **3** | **3** | $\infty$ | **0** |

**注：第*k*次迭代中第*k*行和第*k*列不变！**

- $D(k)$ 和迭代:

| $D^{(0)}$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 11 |
| 2 | 6 | 0 | 2 |
| 3 | 3 | $\infty$ | 0 |

| $D^{(1)}$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 11 |
| 2 | 6 | 0 | 2 |
| 3 | 3 | 7 | 0 |

| $D^{(2)}$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 6 |
| 2 | 6 | 0 | 2 |
| 3 | 3 | 7 | 0 |

| $D^{(3)}$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 6 |
| 2 | 5 | 0 | 2 |
| 3 | 3 | 7 | 0 |

结果

## ❖计算最优值

```
void ShortestPath_FLOYD(MGraph G, DistancMartrix &D){
 for(i=1; i<=n; ++i)
  for(j=1; j<=n; ++j){
   D[i][j]=G [i][j];
   }\\for
 for(k=1; k<=n; ++k)
  for(i=1; i<=n; ++i)
   for(j=1; j<=n; ++j)
       if(D[i][k]+D[k][j]<D[i][j]){
       D[i][j]= D[i][k]+D[k][j];
        }\\if
}
```

时间: $O(n^3)$

空间: $O(n^2)$

# 提　纲

- 引例
- 动态规划
- 矩阵连乘问题
- 最长公共子序列
- 图像压缩
- 最短路径的Floyd算法
- 0-1背包问题
- 最优二叉搜索树
- 总结

# 0-1背包问题

❖ 问题：

- 给定$n$种物品和一背包。物品 $i$的重量是$w_i$，其价值为$v_i$，背包的容量为$C$。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大?
- 特殊的整数规划问题：求一个$n$元0-1向量$\{x_1, x_2, \ldots, x_n\}$

目标： $$\max \sum_{i=1}^{n} v_i x_i$$

约束条件： $$\begin{cases} \sum_{i=1}^{n} w_i x_i \le C \\ x_i \in \{0,1\}, 1 \le i \le n \end{cases}$$

- 穷举搜索法

  - 每个$x_i$可能为0或1，因此向量$\{x_1, x_2, \ldots, x_n\}$有$2^n$个不同的0，1值序列

  - 计算时间为$\Theta(2^n)$——指数级

## 动态规划

- 最优子结构性质

  设$(x_1, x_2, \ldots, x_n)$是所给0/1背包问题的一个最优解，则$(x_1, \ldots, x_{n-1})$是下面一个子问题的最优解：

$$\max \sum_{i=1}^{n-1} v_i x_i \qquad \begin{cases} \sum_{i=1}^{n-1} w_i x_i \le C - w_n x_n \\ x_i \in \{0,1\} \quad (1 \le i \le n-1) \end{cases}$$

## ❖建立递归关系

- 子问题

  - 设$m(i, j)$为背包容量为$j$时，考虑装入1~$i$种物品的最大价值

例：背包容量$C$=9， 4种物品，重量$w_i$={2,3,4,5}，价值$v_i$={3,4,5,7}

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| w | 2 | 3 | 4 | 5 |
| v | 3 | 4 | 5 | 7 |

i=0~4，j=0~9

$m(i, 0)=0$， $m(0, j)=0$

$m(1,1)=0$ 容量为1时，考虑装入第一种物品能获得的最大值

$m(1,2)=3$ $m(1,3)=3$ ….. $m(1,9)=3$

$m(2,1)=0$ $m(2,2)=3$ $m(2,3)=4$ $m(2,5)=7$ ……….

$m(4,9)$=？

所有子问题个数： (n+1)×(C+1)

# 考虑物品 i，放还是不放入背包

m(i, j)是下面两个量的最大值：

(1) m(i–1, j): 在容量为j的背包中装入1~i–1中的物品，不装入物品 i，价值最大

(2) m(i–1, j–$w_i$)+$v_i$ :必装入物品 i，在容量为j–$w_i$的背包中装入1~i–1中的物品的最大价值，再加上物品 i的价值$v_i$ (j≥$w_i$)


❖ 递推式：

$$m(i,j)=\begin{cases}\max\{m(i-1,j),m(i-1,j-w_i)+v_i\} & j\ge w_i\\ m(i-1,j) & 0\le j<w_i\\ 0 & i=0\text{或}j=0\end{cases}$$

## ❖计算最优值

- 用一个$(n+1)\times(C+1)$的矩阵(表)来计算$m(i, j)$, 逐行填表
- 算法: knapsack

**Input:**

$n$种物品的重量和价值:

$\{w_1, w_2, \ldots, w_n\}$,

$\{v_1, v_2, \ldots, v_n\}$;

背包容量$C$

**Output:** $m(n, C)$

计算时间：$O(nC)$

```
for i=0 to n do
 m[i,0]=0
end for
for j=0 to C do
 m[0,j]=0
end for
for i=1 to n do \\考察物品
 for j=1 to C do
  m[i,j]=m[i-1,j]
  if wi≤j then  \\第i个物品可放
   m[i,j]=max{m[i,j],m[i-1,j-wi]+vi}
  end if
 end for
end for
return m[n,C]
```

当背包容量c很大时，算法需要的计算时间较多。例如，当$c>2^n$时，算法需要$\Omega(n2^n)$计算时间。

- 例： 如果背包容量$C$=9, 4种物品的重量和价值分别为{2,3,4,5}和{3,4,5,7}, 尽可能将物品装入背包，并使总价值最大。

5×10的表：行是物品$i$，列是容量$j$

| | *1* | *2* | *3* | *4* |
|---|---|---|---|---|
| w | 2 | 3 | 4 | 5 |
| v | 3 | 4 | 5 | 7 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | **3** | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 2 | 0 | 0 | 3 | 4 | 4 | **7** | 7 | 7 | 7 | 7 |
| 3 | 0 | 0 | 3 | 4 | 5 | 7 | 8 | 9 | 9 | **12** |
| 4 | 0 | 0 | 3 | 4 | 5 | 7 | 8 | 10 | 11 | **12** |

$x_1$=1

$x_2$=1

$x_3$=1

$x_4$=0

最优值：最大价值为 12

最优解：装入物品1,2,3; 装入物品3,4

$m(i, j)$的计算只与$m(i–1,0)$~$m(i–1, j)$的值相关

- 改进

定义序偶（$V_i,W_j$）:表示当背包重量为$W_j$时，其价值为$V_i$，每个序偶表示一种可行解;

序偶对的集合$S^i$，如： $S^0=\{(0,0)\}$

它是第$i$步决策时所有可行解的集合（$S^i$表示考虑$1\sim i$中物品放入背包的所有可行解）

$S^{i-1}$ 考虑第$i$个物品，放或不放！！

不放 $S^{i-1}$ 放 $S^{i-1} \longrightarrow S_1^i$

$S^i=S^{i-1}\oplus S_1^i$

支配原则：有2个序偶对$(V_i,W_j)$和$(V_k,W_l)$，如果有$W_j>=W_l$，$V_i<=V_k$，则可舍弃$(V_i,W_j)$

例： 如果背包容量$C$=9, 4种物品的重量和价值分别为{2,3,4,5}和{3,4,5,7}, 尽可能将物品装入背包，并使总价值最大。

$$S^0 = \{(0,0)\}$$

| | *1* | *2* | *3* | *4* |
|---|---|---|---|---|
| w | 2 | 3 | 4 | 5 |
| v | 3 | 4 | 5 | 7 |

*i*=1..*n*，从*i*=1开始，通过*n*步决策获得最优解

$S_1^1 = \{(3,2)\}$  $x_1$=1  $S^1 = S^0 \oplus S_1^1 = \{(0,0),(3,2)\}$

$S_1^2 = \{(4,3),(7,5)\}$  $S^2 = \{(0,0),(3,2),(4,3),(7,5)\}$

$S_1^3 = \{(5,4),(8,6),(9,7),(12,9)\}$  $S^3 = \{(0,0),(3,2),(4,3),(5,4),(7,5),(8,6),(9,7),(12,9)\}$

$$S_1^4 = \{(7,5),(10,7),(11,8),(12,9),(14,10),(15,11),(16,12),(19,14)\}$$

$$S^4 = \{(0,0),(3,2),(4,3),(5,4),(7,5),(8,6),(10,7),(11,8),(12,9)\}$$

$O(min(nc,2^n))$

# 提　纲

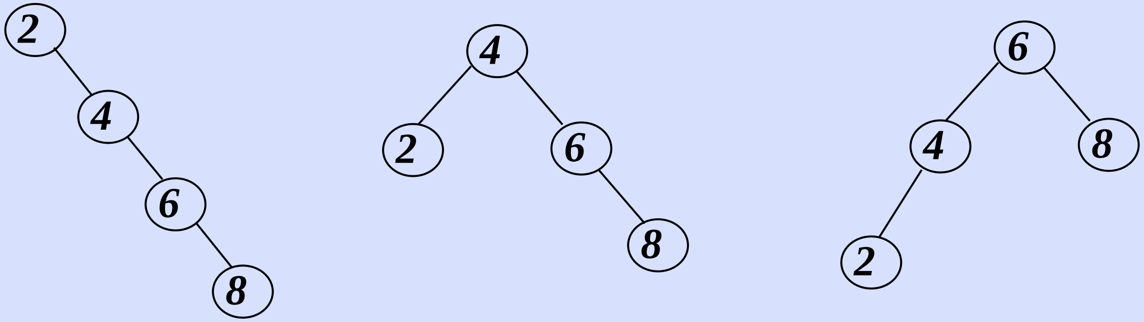
- 引例
- 动态规划
- 矩阵连乘问题
- 最长公共子序列
- 图像压缩
- 最短路径的Floyd算法
- 0-1背包问题
- 最优二叉搜索树
- 总结

# 最优二叉搜索树

❖ **问题：**

基于二叉树进行搜索。 什么是最优二叉搜索树？

**引例**：有序集合S={2, 4, 6, 8}的查找概率是{0.1, 0.2, 0.4, 0.3}




(a) (b) (c)

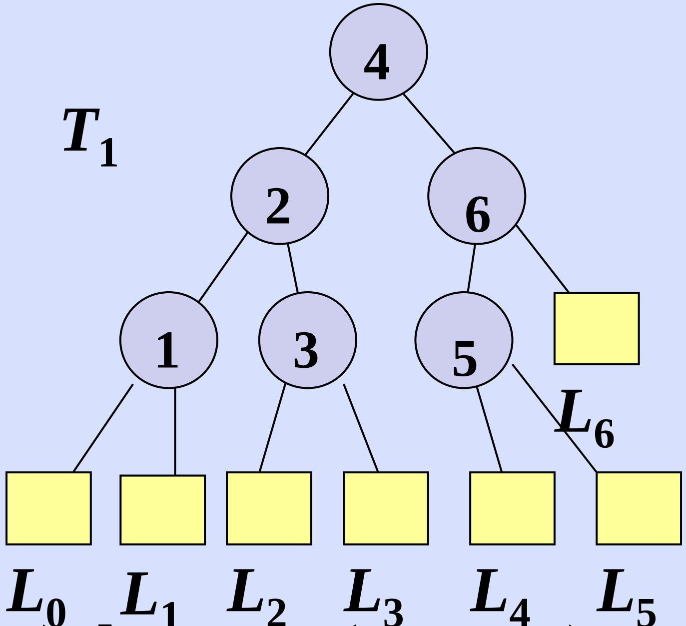
二叉搜索树示例

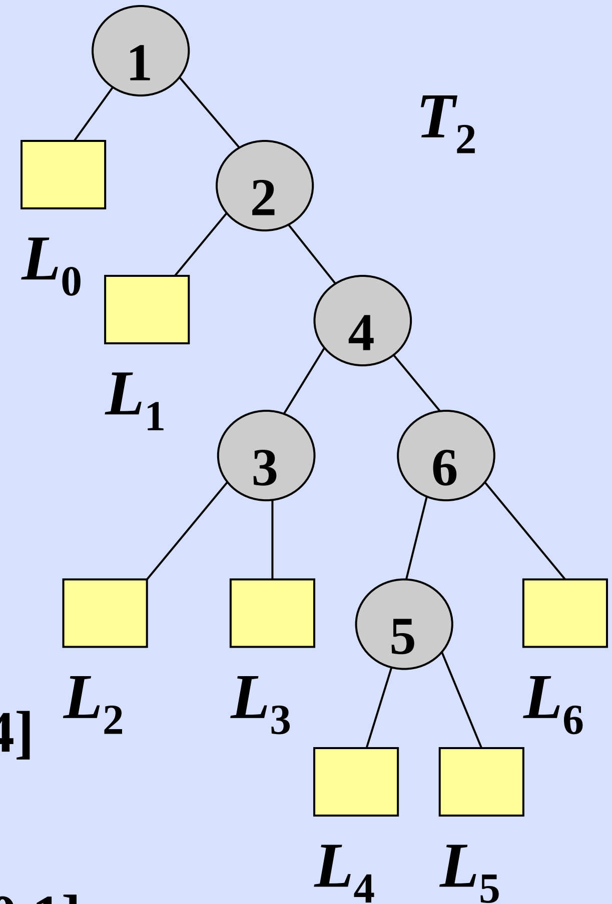
**ASLa=1*0.1+2*0.2+3*0.4+4*0.3=2.9**

**ASLc=1*0.4+2*0.2+2*0.3+3*0.1=1.7**

$S=\{1,2,3,4,5,6\}$

$P=<0.04, 0.1, 0.01, 0.2, 0.05, 0.2, 0.02, 0.1, 0.02, 0.1, 0.07, 0.05, 0.04>$



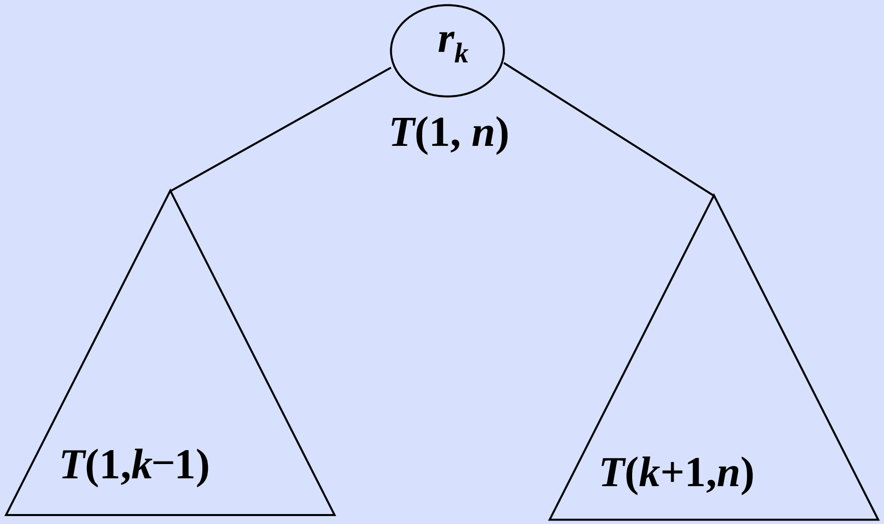




$m(T_1)=[1*0.1+2*(0.2+0.05)+3*(0.1+0.2+0.1)]$
$+[3*(0.04+0.01+0.05+0.02+0.02+0.07)+2*0.04]$
$= 1.8+0.71=2.51$

$m(T_2)= [1*0.1+2*0.2+3*0.1+4*(0.2+0.05)+5*0.1]$
$+[1*0.04+2*0.01+4*(0.05+0.02+0.04)+5*(0.02+0.07)]$
$= 2.3+0.95=3.25$

实例

## ❖动态规划




以$r_k$为根的二叉搜索树

将由$\{r_1, r_2, \ldots, r_n\}$构成的二叉搜索树记为$T(1, n)$，其中$r_k$（$1 \le k \le n$）是$T(1, n)$的根结点，则其左子树$T(1, k-1)$由$\{r_1, \ldots, r_{k-1}\}$构成，其右子树$T(k+1, n)$由$\{r_{k+1}, \ldots, r_n\}$构成

## 类似矩阵连乘问题

- 子问题？重叠子问题性质？
- 递推关系？怎样计算最优值及最优解？

# 总结

- 动态规划的基本思想、适用条件，及所解决问题的主要特征
- 动态规划方法解决问题的一般方法和步骤、多阶段决策问题的特征和最优化原理
- 动态规划的重要算法实例：
  - 矩阵连乘问题的动态规划算法
  - 最长公共子序列的动态规划算法
  - 任意结点对间最短路径的Floyd动态规划算法
  - 0-1背包问题的动态归划算法
  - 最优二叉搜索树的动态规划算法


谢谢大家！