# 课程回顾

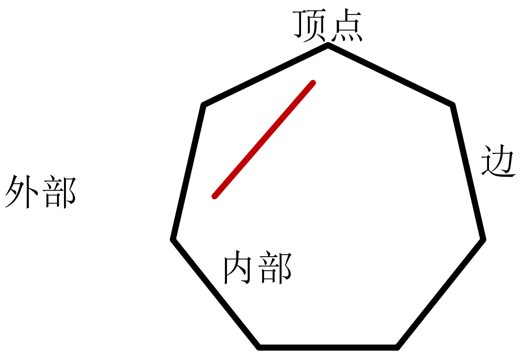
- 动态规划概述
  - 算法步骤：刻画一个最优解的结构特征、递归地定义最优解的值、计算最优解的值，通常采用自底向上的方法、利用计算出的信息构造一个最优解
  - 实现方法：带备忘的自顶向下法、自底向上法
- 动态规划问题：钢条切割、矩阵链乘法的最优括号化

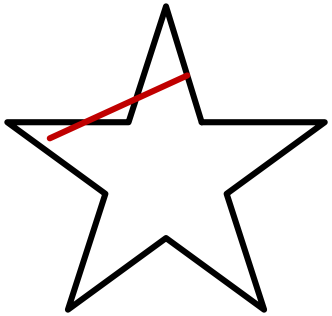
# 动态规划问题

- 钢条切割
- 矩阵链乘法的最优括号化
- 多边形的最佳三角剖分
- 最长公共子序列
- 最优二叉搜索树
- 0-1背包

# 多边形的最佳三角剖分

- **凸多边形的最佳三角剖分类似于矩阵链乘问题**
- **凸多边形**
  - 多边形：无交叉边
  - 凸多边形：边界上或内部的任意两点连线上的点均在边界上或内部



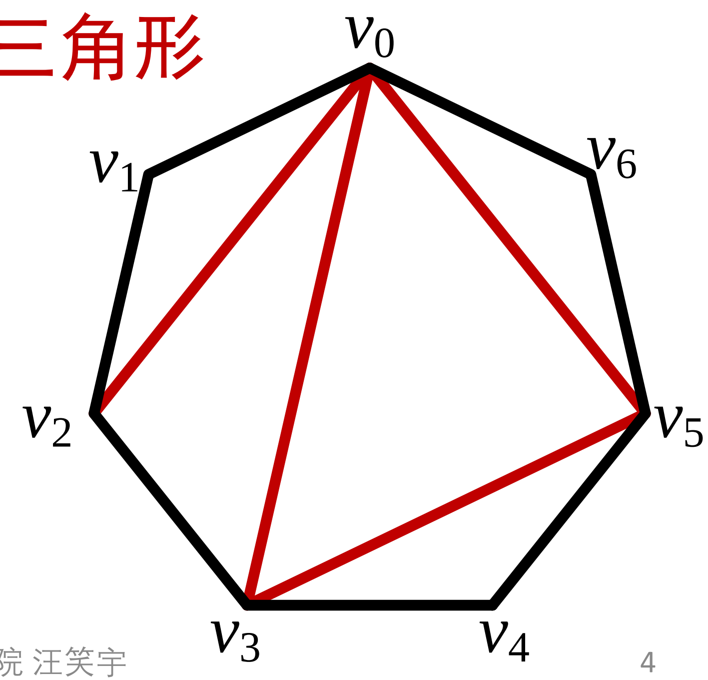



- **表示**
  - 顶点逆时针列表：$P_n$=<$v_0$, $v_1$, …, $v_{n\text{-}1}$>
  - $n$条边：$\overline{v_0v_1}, \overline{v_1v_2}, \ldots, \overline{v_{n-1}v_n}, \quad v_0 = v_n$
  - 顶点号是顶点数模除结果

以下均讨论凸多边形！

# 多边形的最佳三角剖分 (续)

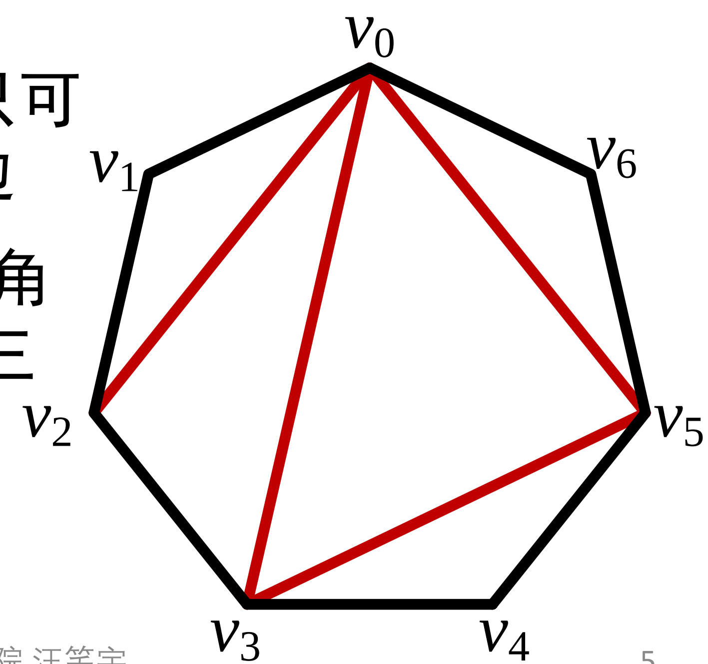
- 弦：若$v_i$ , $v_j$是不相邻点，则线段 $\overline{v_i v_j}$ 是一条弦，它将多边形划分为两个多边形：
  $\langle v_i, v_{i+1}, \ldots, v_j \rangle$ 和 $\langle v_j, v_{j+1}, \ldots, v_i \rangle$
- 多边形的三角剖分：多边形的一个弦集T，将多边形划分为若干个不相交的三角形



# 多边形的最佳三角剖分 (续)

■一些结论：

- 在一个三角剖分中，没有弦在除端点外的位置相交且弦集合T最大（每条不在T中的弦与T中的某弦相交于端点外的位置）
- 三角剖分产生的三角形的边只可能是剖分中的弦或多边形的边
- $n$个顶点的凸多边形的每个三角剖分有$n$-3条弦，划分所得的三角形个数为$n$-2



# 多边形的最佳三角剖分 (续)

■最优的三角剖分

给定多边形及三角形的权值函数$W$，找到权值之和最小的三角剖分

- 三角形权值：视具体问题而定，例如：

$$W(\Delta v_i v_j v_k) = |v_i v_j| + |v_j v_k| + |v_k v_i|$$

$|v_i v_j|$是$v_i$和$v_j$的欧氏距离

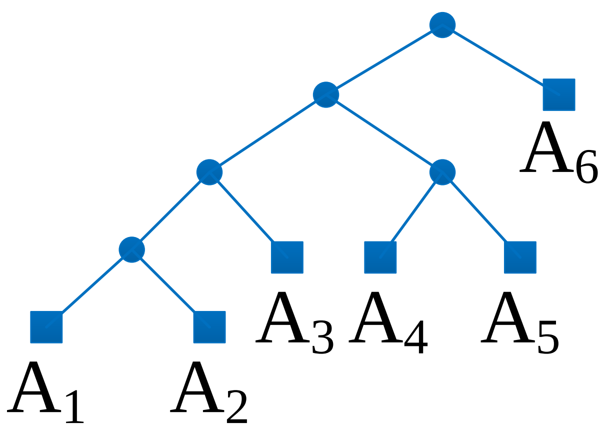
- 三角剖分的权：各三角形权值之和
- 最优三角剖分：权值之和最小的三角剖分

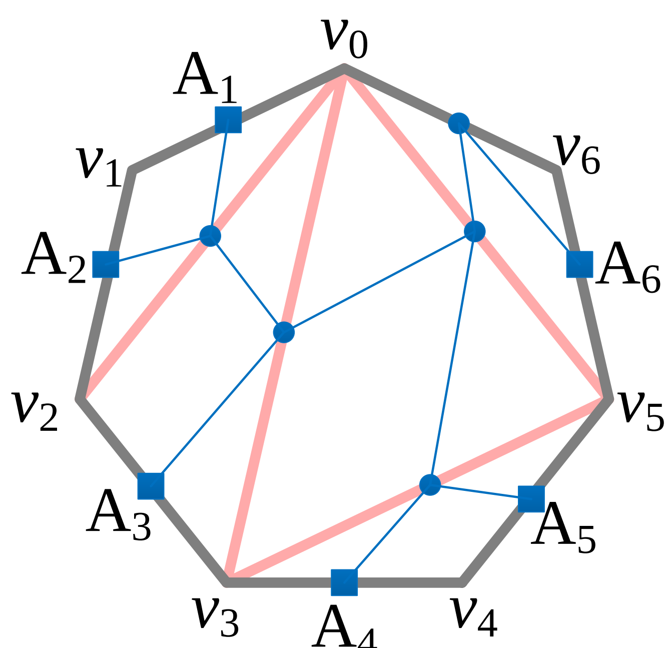
# 多边形的最佳三角剖分 (续)

■与矩阵链乘法的括号化一致

通过树来解释三角剖分与表达式括号化的一致性

$(((A_1A_2)A_3)(A_4A_5))(A_6)$

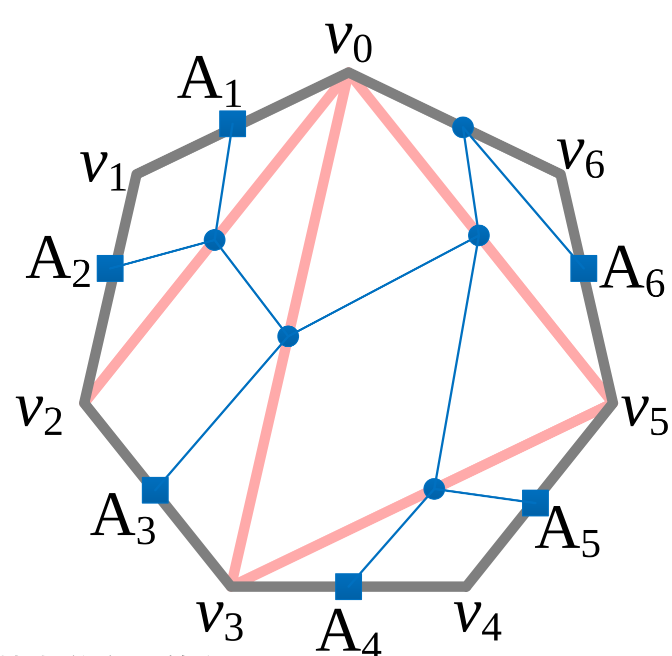


# 多边形的最佳三角剖分 (续)

■ $n$个矩阵的完全括号化
⇔ $n$个叶子的解析树 (parse tree)
⇔ $n$+1个顶点的多边形的三角剖分

- ➢矩阵$A_i$ (大小为$p_{i-1} \times p_i$)对应边 $\overline{v_{i-1}v_i}$ ($i$=1, 2, ..., $n$-1)
- ➢矩阵积$A_{i+1..j}$对应弦 $\overline{v_iv_j}$



# 多边形的最佳三角剖分 (续)

- 最优矩阵链乘可看作是最优三角剖分的特例：
  - 定义三角剖分的权函数：$W(\Delta v_i v_j v_k) = p_i p_j p_k$
  - 该权函数下的$P_{n+1}$的最优三角剖分就给出了$A_1A_2 \cdots A_n$的一个最优括号化
- 求解最优三角剖分问题
  - 用$v_0, v_1, \ldots, v_n$代替$p_0, p_1, \ldots, p_n$
  - 修改MATRIX_CHAIN_ORDER中$q$计算公式为：$q \leftarrow m[i, k] + m[k+1, j] + W(\triangle v_{i-1}v_kv_j)$

$m[1, n]$包含了$P_{n+1}$的一个最优三角剖分的权值，为什么？

$n$个矩阵进行链乘：$n$-1次标准矩阵乘法；$n$+1边形三角剖分：划分出$n$-1个三角形

# 多边形的最佳三角剖分 (续)

1. 刻画一个最优解的结构特征——最优三角剖分的子结构

  - 设多边形$P_{n+1} = <v_0, v_1, \ldots, v_n>$的一个最佳三角剖分包括某个三角形$\triangle v_0 v_k v_n$，则权值

$$W(P_{n+1}) = W(\triangle v_0 v_k v_n) + W(<v_0, v_1, \ldots, v_k>) + W(<v_k, v_{k+1}, \ldots, v_n>)$$

  - $W(P_{n+1})$最优就要求子多边形$<v_0, v_1, \ldots, v_k>$和$<v_k, v_{k+1}, \ldots, v_n>$的三角剖分也是最优的

# 多边形的最佳三角剖分 (续)

2. 递归地定义最优解的值

- 设$t[i, j](1 \le i \le j \le n)$是多边形$<v_{i-1}, v_i, \ldots, v_j>$的一个最优解的值（即最优三角剖分的权），则多边形$P_{n+1}$的最优三角剖分的权为$t[1, n]$。
- $t[i, j]$的递归定义如下：
  - 若$i=j$，多边形退化为$<v_{i-1}, v_i>$，可定义权为0 即$t[i, i]=0, 1 \le i \le n$
  - 若$i<j$, 则多边形$<v_{i-1}, v_i, \ldots, v_j>$至少有3个顶点，设最佳剖分点为$k$ $(i \le k < j)$，则剖分结果为：
    $\triangle v_{i-1} v_k v_j$ $<v_{i-1}, v_i, \ldots, v_k>$ $<v_k, v_{k+1}, \ldots, v_j>$

# 多边形的最佳三角剖分 (续)

2. 递归地定义最优解的值

- $t[i, j]$的递归定义如下：

$$t[i,j]=\begin{cases}0, & i=j,\\ \min\limits_{i\le k<j}\{t[i,k]+t[k+1,j]+W(\Delta v_{i-1}v_kv_j)\}, & i<j.\end{cases}$$

3. 计算最优解的值 & 4. 构造最优解

- 与矩阵链完全一致，仅权函数不同
- 时间复杂度：$O(n^3)$；空间复杂度：$O(n^2)$

# 动态规划原理

- ■最优子结构
- ■注意事项
- ■重叠子问题
- ■重构最优解
- ■备忘

# 最优子结构

- 最优子结构性质：一个问题的最优解包含其子问题的最优解
  - 例：矩阵$A_iA_{i+1}\cdot\cdot\cdot A_j$的最优括号化问题蕴含着两个子问题$A_i\cdot\cdot\cdot A_k$和$A_{k+1}\cdot\cdot\cdot A_j$的解也必须是最优的
- 具有最优子结构性质的问题可能会使用动态规划
- 在动态规划中，可用子问题的最优解来构造原问题的最优解

# 最优子结构 (续)

- 如何发现最优子结构？
  - 说明问题的解必须进行某种选择，这种选择导致一个或多个待解的子问题
  - 对一给定问题，假定导致最优解的选择已给定，即无须关心如何做出选择，只须假定它已给出
  - 给定选择后，决定由此产生哪些子问题，如何最好地描述子问题空间
  - 证明用在问题最优解内的子问题的解也必须是最优的。方法是“剪切-粘贴”(cut-and-paste)技术和反证法。假定在最优解对应的子问题的解非最优，删去它换上最优解，得到原问题的解非最优，矛盾！

# 最优子结构 (续)

- 如何描述子问题空间（不同的子问题个数）：尽可能使其简单，然后再考虑有没有必要扩展
  - 钢条切割：对每个$i$值，长度为$i$钢条的最优切割
  - 矩阵链乘法：$A_{i..j}$
- 最优子结构有关的两方面问题
  - 原问题最优解中涉及多少个子问题
  - 用在最优解中的子问题有多少种选择
    - 钢条切割：一个子问题，$n$种选择 $\boxed{r_n} = \max\limits_{1 \le i \le n}(p_i + \boxed{r_{n-i}})$
    - 矩阵链乘法：两个子问题，$j$-$i$种选择

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

# 最优子结构 (续)

- 动态规划算法的运行时间
  - 子问题总数
  - 对每个子问题涉及多少种选择
- 例：矩阵链乘共要解$\Theta(n^2)$个子问题: $1\leq i\leq j\leq n$，求解每个子问题至多有$n$-1种选择，最终的运行时间为$\Theta(n^3)$
- 动态规划求解方式
  - 自底向上

# 注意事项

- 注意问题是否具有最优子结构性质！
- 例：给定有向无权图$G$=($V$, $E$)和顶点$u$, $v \in V$，求顶点$u$到$v$的最短/最长路径的问题（指简单路径）
  - 无权最短路径问题具有最优子结构性质
    设从$u$到$v$的最短路径是$P$，并设中间点为$w$，则

$$u \xrightarrow{P} v \qquad \Rightarrow \qquad u \xrightarrow{P_1} w \xrightarrow{P_2} v$$

    显然$P_1$和$P_2$也必须是最优(短)的

# 注意事项 (续)

- 最长路径不具有最优子结构

  设$P$是从$u$到$v$的最长路径，$w$是中间某点，则

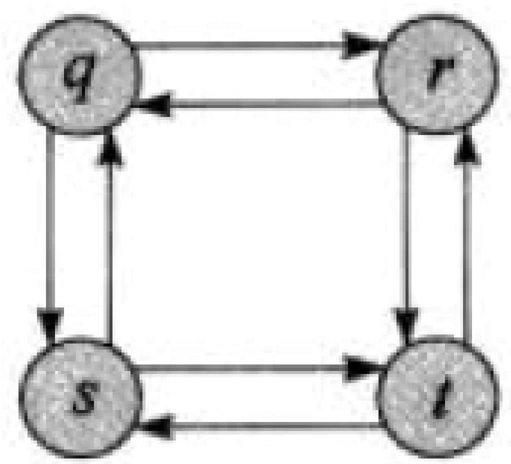
$$u \xrightarrow{P} v \qquad \Rightarrow \qquad u \xrightarrow{P_1} w \xrightarrow{P_2} v$$

  但$P_1$不一定是从$u$到$w$的最长路径，$P_2$也不一定是从$w$到$v$的最长路径

- 例：考虑一条最长路径：$q \rightarrow r \rightarrow t$

  但$q$到$r$的最长路径是：$q \rightarrow s \rightarrow t \rightarrow r$

  $r$到$t$的最长路径是：$r \rightarrow q \rightarrow s \rightarrow t$



# 注意事项 (续)

- 为什么两问题有差别？
  - 最长路径的子问题不是独立的
    所谓独立指一个子问题的解不能影响另一个子问题的解
    但第一个子问题中使用了$s$和$t$，第二个子问题又使用了，使得产生的路径不再是简单路径
    从另一个角度看：一个子问题求解时使用的资源(顶点)不能在另一个子问题中再使用
  - 最短路径问题中，两子问题没有共享资源，可用反证法证明
- 再如矩阵链乘法：$A_{i..j} \Rightarrow A_{i..k} \cdot A_{k+1..j}$
  显然两子链不相交，是相互独立的两子问题

# 重叠子问题

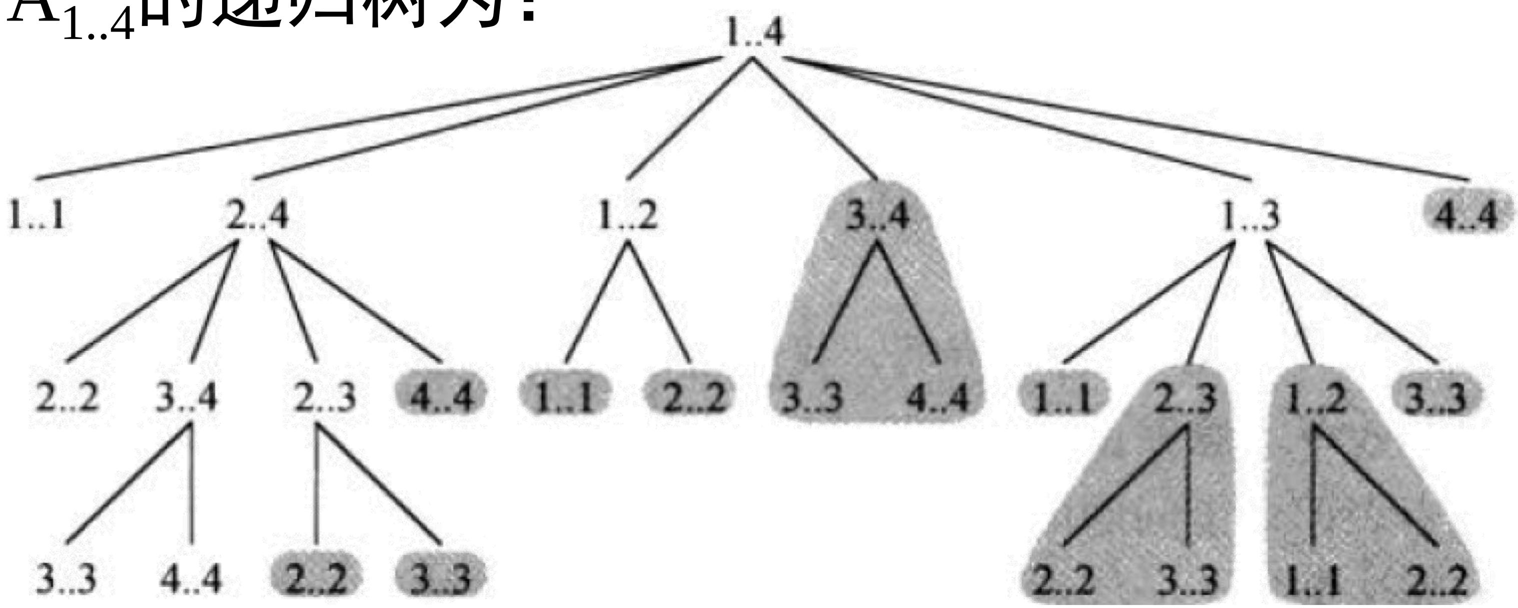
- 子问题空间足够小：用递归算法解某问题时重复计算同一子问题
- 分治法与动态规划的比较
  - 当递归每一步产生一个新的子问题时，适合使用分治法
  - 当递归中较多出现重叠子问题时，适合使用动态规划，即对重叠子问题只求解一次，然后存储在表中，当需要使用时常数时间内查表
  - 若子问题规模是多项式阶的，动态规划特别有效

# 重叠子问题 (续)

■例：用自然递归算法求解矩阵链乘法（算法见教材p219）

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

$A_{1..4}$的递归树为：



# 重叠子问题 (续)

■阴影部分是重叠子问题，递归算法须重复计算

$$m[1,n]=\begin{cases}0, & n=1,\\ \min\limits_{1\le k<n}\{m[1,k]+m[k+1,n]+p_0p_kp_n\}, & n>1.\end{cases}$$

■递归式时间： $\begin{cases}T(1)\ge 1,\\ T(n)\ge 1+\sum\limits_{k=1}^{n-1}(T(k)+T(n-k)+1), \; n>1.\end{cases}$

代入法得$T(n)=\Omega(2^n)$

■结论：当自然递归算法是指数阶，但实际不同的子问题数目是多项式阶时，可用动态规划来获得高效算法

# 重构最优解

- ■用附加表保存中间选择结果能节省重构最优解的时间
  - ➢例：矩阵链乘法利用表$s$记录最优分割位置

# 备忘

- 动态规划：分析是自顶向下，实现是自底向上
- 可采用备忘（记忆）型版本：采用自顶向下实现，是一个记忆型递归算法
- 将子问题的解记录在一个表中：
  - 每个子问题的解对应一表项
  - 每个表项初值为一个特殊值，表示尚未填入
  - 在递归算法执行过程中第一次遇某子问题时，计算其解并填入表中，以后再遇此子问题时，将表中值简单地返回（不重复计算），截断递归

# 备忘 (续)

```
MEMOIZED_MATRIX_CHAIN(p)
n ← p.length - 1
let m[1..n, 1..n] be a new table
for i ← 1 to n do
    for j ← 1 to n do
        m[i, j] ← ∞
return LOOKUP_CHAIN(m, p, 1, n)
```

$\Theta(n^2)$

```
LOOKUP_CHAIN(m, p, i, j)
if m[i, j] < ∞
    return m[i, j]
if i = j
    m[i, j] ← 0
else for k ← i to j-1
        q ← LOOKUP_CHAIN(m, p, i, k)
              + LOOKUP_CHAIN(m, p, k+1, j) + p_{i-1} p_k p_j
        if q < m[i, j]
            m[i, j] ← q
return m[i, j]
```

共$\Theta(n^2)$项

计算每个表项需$O(n)$

算法总时间$O(n^3)$

# 备忘 (续)

- 若所有子问题须至少解一次，自底向上的动态规划时间常数因子较优（不需要递归开销，维护表的开销较小）
- 若子问题空间有些不需要计算，则备忘型递归具有只需计算需要的子问题的优点

# 动态规划问题

- 钢条切割
- 矩阵链乘法的最优括号化
- 多边形的最佳三角剖分
- **最长公共子序列**
- 最优二叉搜索树
- 0-1背包

# 最长公共子序列LCS

■子序列：将给定序列中零个或多个元素去掉之后得到的结果

- 形式化定义：给定一个序列$X=\langle x_1, x_2, \ldots, x_m\rangle$，另一个序列$Z=\langle z_1, z_2, \ldots, z_k\rangle$满足如下条件时称为$X$的子序列：存在一个严格递增的$X$的下标序列$\langle i_1, i_2, \ldots, i_k\rangle$，对所有$j=1, 2, \ldots, k$，满足$x_{i_j}=z_j$
- 例：X=<A, B, C, B, D, A, B>
  Z=<    B, C,    D,    B>为X的子序列，对应下标序列为<2, 3, 5, 7>

**子序列不一定是由原序列连续元素构成的序列！**

# 最长公共子序列LCS (续)

- 公共子序列 (common subsequence)：给定两个序列$X$和$Y$，如果$Z$既是$X$的子序列，也是$Y$的子序列，则称$Z$是$X$和$Y$的公共子序列
- 最长公共子序列问题 (longest-common-subsequence problem)：求两个序列公共子序列中最长的一个
- 求解两个给定序列的LCS
  1. 刻画LCS结构特征
  2. 递归解
  3. 计算LCS长度
  4. 构造LCS

# 最长公共子序列LCS (续)

1. 刻画LCS特征

- 穷举法：穷举$X$的所有子序列，检查是否也是$Y$的子序列。若$|X|=m$，则子序列共$2^m$个，穷举法为指数阶下界
- LCS具有最优子结构性质
  前缀：给定一个序列$X=<x_1, x_2, ..., x_m>$，对$i$=0, 1, ..., $m$，定义$X$的第$i$前缀为$X_i=<x_1, x_2, ..., x_i>$
- **定理15.1** 令$X=<x_1, x_2, ..., x_m>$和$Y=<y_1, y_2, ..., y_n>$为两个序列，$Z=<z_1, z_2, ..., z_k>$为$X$和$Y$的任意LCS
  1. 若$x_m=y_n$，则$z_k=x_m=y_n$且$Z_{k-1}$是$X_{m-1}$和$Y_{n-1}$的一个LCS；
  2. 若$x_m \neq y_n$，则$z_k \neq x_m$意味着$Z$是$X_{m-1}$和$Y$的一个LCS；
  3. 若$x_m \neq y_n$，则$z_k \neq y_n$意味着$Z$是$X$和$Y_{n-1}$的一个LCS。

# 最长公共子序列LCS (续)

1. 刻画LCS特征

- **定理15.1** 证明（反证法）：

  1\. (1) $z_k=x_m=y_n$：若$z_k \neq x_m$，则可将$x_m=y_n$追加到$Z$的末尾，得到$X$和$Y$的一个长度为$k$+1的公共子序列，矛盾！

  (2) $Z_{k-1}$是$X_{m-1}$和$Y_{n-1}$的一个LCS：若$X_{m-1}$和$Y_{n-1}$存在长度大于$k$-1的公共子序列$W$，则可将$x_m=y_n$追加到$W$末尾，得到$X$和$Y$的一个长度大于$k$的公共子序列，矛盾！

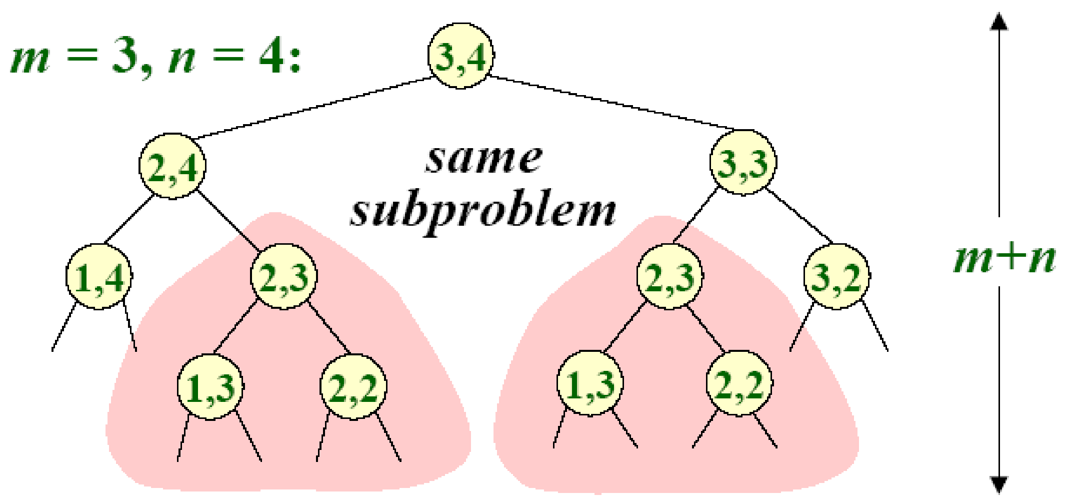
  2\. 因为$x_m \neq y_n$，所以$X_{m-1}$和$Y$的LCS与$X$和$Y$的LCS相同。若存在$X_{m-1}$和$Y$长度大于$k$的公共子序列$W$，则$W$也是$X$和$Y$的公共子序列，长度大于$k$，矛盾！

  3\. 与2对称

# 最长公共子序列LCS (续)

1. 刻画LCS特征

- 最优子结构性质：两个序列的一个LCS包含了两个序列的前缀子序列的一个LCS
    - 蕴含的选择：当$x_m \neq y_n$时，我们事先并不知道Z的长度，只能在$X_{m-1}$和$Y$的LCS以及在$X$和$Y_{n-1}$的LCS中取最大者
- 重叠子问题性质



# 最长公共子序列LCS (续)

2. 递归解

- 由定理15.1：求$X=<x_1, x_2, \ldots, x_m>$和$Y=<y_1, y_2, \ldots, y_n>$的一个LCS时，需求解一个或两个子问题：
  - 若$x_m=y_n$，则求解$X_{m-1}$和$Y_{n-1}$的一个LCS，将$x_m=y_n$追加到这个LCS的末尾
  - 若$x_m\neq y_n$，(1) 求解$X_{m-1}$和$Y$的LCS；(2) 求解$X$和$Y_{n-1}$的LCS。求两者长度最大者

# 最长公共子序列LCS (续)

## 2. 递归解

- $c[i, j]$：$X_i$和$Y_j$的LCS长度（$0 \le i \le m$, $0 \le j \le n$）

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

- 限定了需求解哪些子问题，并非所有子问题都要求解

# 最长公共子序列LCS (续)

3. 计算LCS长度

- 不同子问题个数：$\Theta(mn)$
- 输入：序列$X$=<$x_1, x_2, \ldots, x_m$>和$Y$=<$y_1, y_2, \ldots, y_n$>
- 输出：$c[0..m, 0..n]$：按行主次序计算LCS长度
  $b[1..m, 1..n]$：辅助构造最优解子序列

$$b[i,j] = \begin{cases} \nwarrow, & c[i,j] = c[i-1,j-1]+1, \\ \uparrow, & c[i,j] = c[i-1,j], \\ \leftarrow, & c[i,j] = c[i,j-1]. \end{cases}$$

- 构造解时，从$b[m, n]$出发，根据箭头方向上溯至$i$=0或$j$=0为止，当$b[i, j]$包含“↖”时打印出$x_i$即可