



HITWH  
SE

## 第四章

# Dynamic Programming Algorithms



- 4.1 Elements of Dynamic Programming
- 4.2 Matrix-chain multiplication
- 4.3 Longest Common Subsequence
- 4.4 0/1 Knapsack Problem
- 4.5 The Optimal binary search trees



## Introduction to Algorithms

### 第15章

15.2, 15.3, 15.4, 15.5



## 4.1 Elements of Dynamic Programming

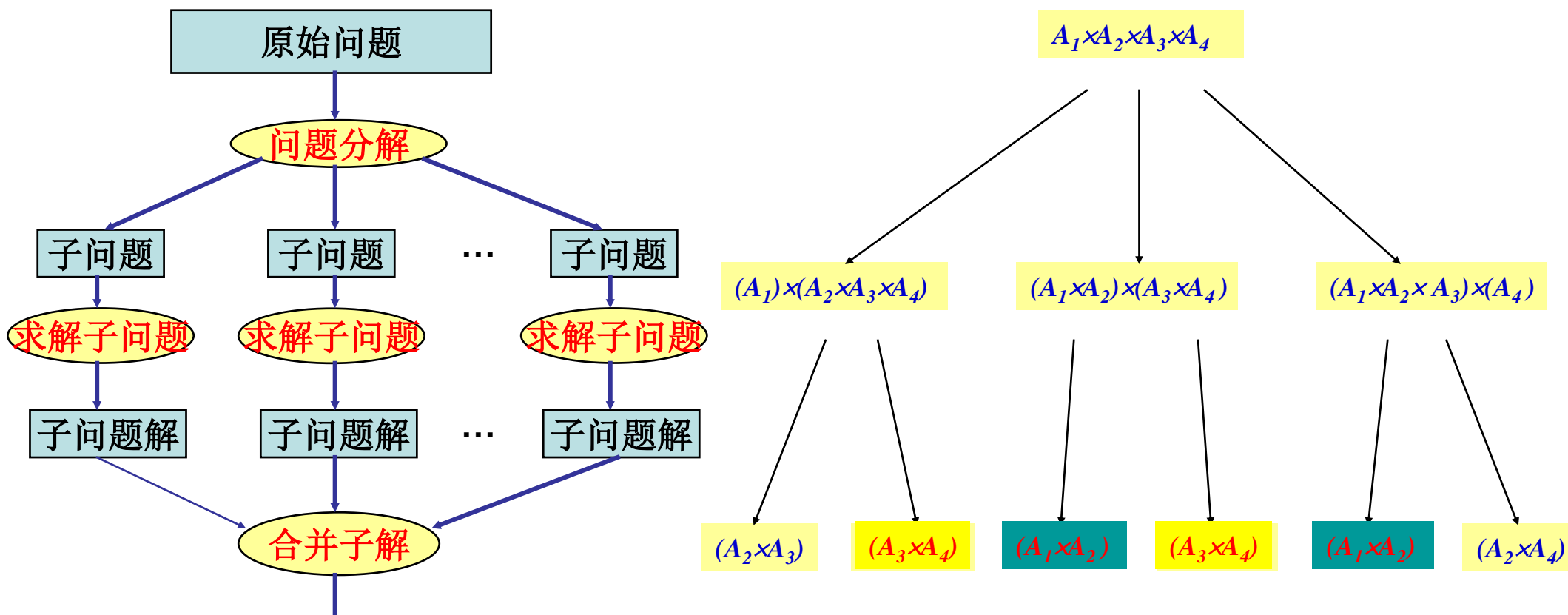
**Why?**

**What?**

**How?**



- Divide-and-Conquer方法的问题



**问题:** 如果子问题不是相互独立的, 分治方法将重复计算公共子问题, 效率很低



## • 优化问题

- 问题可能有很多解，每个可能的解都对应有一个值，这个值通常称为代价
- 优化问题是要在该问题所有可能的解中找到具有最优值(最大/最小)的解，即问题的一个最优解
- 一个问题的最优解不一定是唯一的
- 举例：最短路径，旅行商、任务调度等问题
- 因此我们也可以说：优化问题就是给定一个代价函数，在问题的解空间中搜索具有最小或最大代价的优化解

动态规划是解决优化问题的一种常见方法



- Dynamic Programming历史

- 动态规划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 (Multistep decision process) 的优化问题时，提出了著名的 **最优性原理**，把多阶段过程转化为一系列单阶段问题，逐个求解，创立了解决这类过程优化问题的新方法----动态规划自底向上地求解子问题
- **多阶段决策问题**：求解的问题可以划分为一系列相互联系的阶段，在每个阶段都需要做出决策，且一个阶段决策的选择会影响下一个阶段的决策，从而影响整个过程的活动路线，求解的目标是选择各个阶段的决策是整个过程达到最优



- Dynamic Programming
  - 把原始问题划分成一系列子问题
    - 不同子问题的数目常常只有多项式量级
  - 求解每个子问题仅一次，并将其结果保存在一个表中，以后用到时直接存取，不重复计算，节省计算时间
  - 自底向上地求解子问题
- 适用范围
  - 一类优化问题：可分为多个相关子问题，子问题的解被重复使用





- 使用Dynamic Programming的条件
  - 优化子结构
    - 当一个问题的优化解包含了子问题的优化解时，我们说这个问题具有优化子结构。
  - 重叠子问题
    - 在问题的求解过程中，很多子问题的解将被多次使用



- Dynamic Programming算法的设计步骤
  - 分析优化解的结构
  - 递归地定义最优解的代价
  - 递归地划分子问题，直至不可分
  - 自底向上地求解各个子问题
    - 计算优化解的代价并保存之
    - 获取构造最优解的信息
  - 根据构造最优解的信息构造优化解



**HITWH**  
**SE**

## 4.2 Matrix-chain Multiplication



# 问题的定义

- 输入:  $\langle A_1, A_2, \dots, A_n \rangle$ ,  $A_i$  是  $p_{i-1} \times p_i$  矩阵
- 输出: 计算  $A_1 \times A_2 \times \dots \times A_n$  的最小代价方法

矩阵乘法的代价/复杂性: 乘法的次数

若  $A$  是  $p \times q$  矩阵,  $B$  是  $q \times r$  矩阵, 则  $A \times B$  的代价是  $O(pqr)$



- 矩阵链乘法

- 矩阵乘法满足结合率。
- 计算一个矩阵链的乘法可有多种方法：

例如,  $(A_1 \times A_2 \times A_3 \times A_4)$

$$= (A_1 \times (A_2 \times (A_3 \times A_4)))$$

$$= ((A_1 \times A_2) \times (A_3 \times A_4))$$

....

$$= ((A_1 \times A_2) \times A_3) \times A_4$$



- 矩阵链乘法的代价与计算顺序的关系

— 设  $A_1=10 \times 100$  矩阵,  $A_2=100 \times 5$  矩阵,  $A_3=5 \times 50$  矩阵

$$T((A_1 \times A_2) \times A_3) = 10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$$

$$T(A_1 \times (A_2 \times A_3)) = 100 \times 5 \times 50 + 10 \times 100 \times 50 = 75000$$

结论: 不同计算顺序有不同的代价



- 矩阵链乘法优化问题的解空间

- 设  $p(n)$  = 计算  $n$  个矩阵乘积的方法数

- $p(n)$  的递归方程

$$p(n) = 1 \quad \text{if } n=1$$

$$p(n) = \sum_{k=1}^{n-1} p(k)p(n-k) \quad \text{if } n>1$$

$$P(n) = \Omega(2^n)$$

$$(A_1 \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_{n-1} \times A_n)$$

$$p(k)$$

$$p(n-k)$$

$$(A_1) \quad \times (A_2 \times \dots \times A_n)$$

$$(A_1 \times A_2) \quad \times (A_3 \times \dots \times A_n)$$

....

$$(A_1 \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_n)$$

...

$$(A_1 \times \dots \times A_{n-1}) \times (A_n)$$

如此之大的解空间是无法用枚举方法  
求出最优解的！



# 下边开始设计求解矩阵链乘法问题的 Dynamic Programming 算法

- 分析优化解的结构
- 递归地定义最优解的代价
- 递归地划分子问题，直至不可分
- 自底向上地求解各个子问题
  - 计算优化解的代价并保存之
  - 获取构造最优解的信息
- 根据构造最优解的信息构造优化解





# 分析优化解的结构

$$A_1 \times A_2 \times A_3 \times \dots \times A_n = \left\{ \begin{array}{l} (A_1) \quad \times (A_2 \times \dots \times A_n) \\ (A_1 \times A_2) \quad \times (A_3 \times \dots \times A_n) \\ \dots \\ (A_1 \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_n) \\ \dots \\ (A_1 \times \dots \times A_{n-1}) \times (A_n) \end{array} \right.$$

如果等式右端所有子问题的最优乘法方案的代价均已知，则

根据等式右端组合子问题的解，取组合方案代价的最小值即可获得解



# 分析优化解的结构

$$A_1 \times A_2 \times A_3 \times \dots \times A_n = \left\{ \begin{array}{l} (A_1) \quad \times (A_2 \times \dots \times A_n) \\ (A_1 \times A_2) \quad \times (A_3 \times \dots \times A_n) \\ \dots \\ (A_1 \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_n) \\ \dots \\ (A_1 \times \dots \times A_{n-1}) \times (A_n) \end{array} \right.$$

$$cost_{1 \sim n} = \text{Min} \left\{ \begin{array}{l} cost_{1 \sim 1} + cost_{2 \sim n} + p_0 p_1 p_n \\ cost_{1 \sim 2} + cost_{3 \sim n} + p_0 p_2 p_n \\ \dots \\ cost_{1 \sim k} + cost_{k+1 \sim n} + p_0 p_k p_n \\ \dots \\ cost_{1 \sim n-1} + cost_{n \sim n} + p_0 p_{n-1} p_n \end{array} \right.$$

其中  $A_i$  是  $p_{i-1} \times p_i$  矩阵



# 分析优化解的结构

$$A_1 \times A_2 \times A_3 \times \dots \times A_n = \left\{ \begin{array}{l} (A_1) \quad \times (A_2 \times \dots \times A_n) \\ (A_1 \times A_2) \quad \times (A_3 \times \dots \times A_n) \\ \dots \\ (A_1 \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_n) \\ \dots \\ (A_1 \times \dots \times A_{n-1}) \times (A_n) \end{array} \right.$$

优化子结构：

如果红色方案是代价最小的方案，则该方案中  
计算 $A_1 \times \dots \times A_k$ 的方案必须是代价最小的方案  
计算 $A_{k+1} \times \dots \times A_n$ 的方案必须是代价最小的方案

下面用 $A_{i \sim j}$ 表示矩阵链 $A_i \times \dots \times A_j$ 相乘



## • 优化解的结构

- 若计算 $A_{1\sim n}$ 的优化顺序在 $k$ 处断开矩阵链, 即 $A_{1\sim n} = A_{1\sim k} \times A_{k+1\sim n}$ , 则在 $A_{1\sim n}$ 的优化顺序中, 对应于子问题 $A_{1\sim k}$ 的解必须是 $A_{1\sim k}$ 的优化解, 对应于子问题 $A_{k+1\sim n}$ 的解必须是 $A_{k+1\sim n}$ 的优化解.

$$(A_1 \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_{n-1} \times A_n)$$

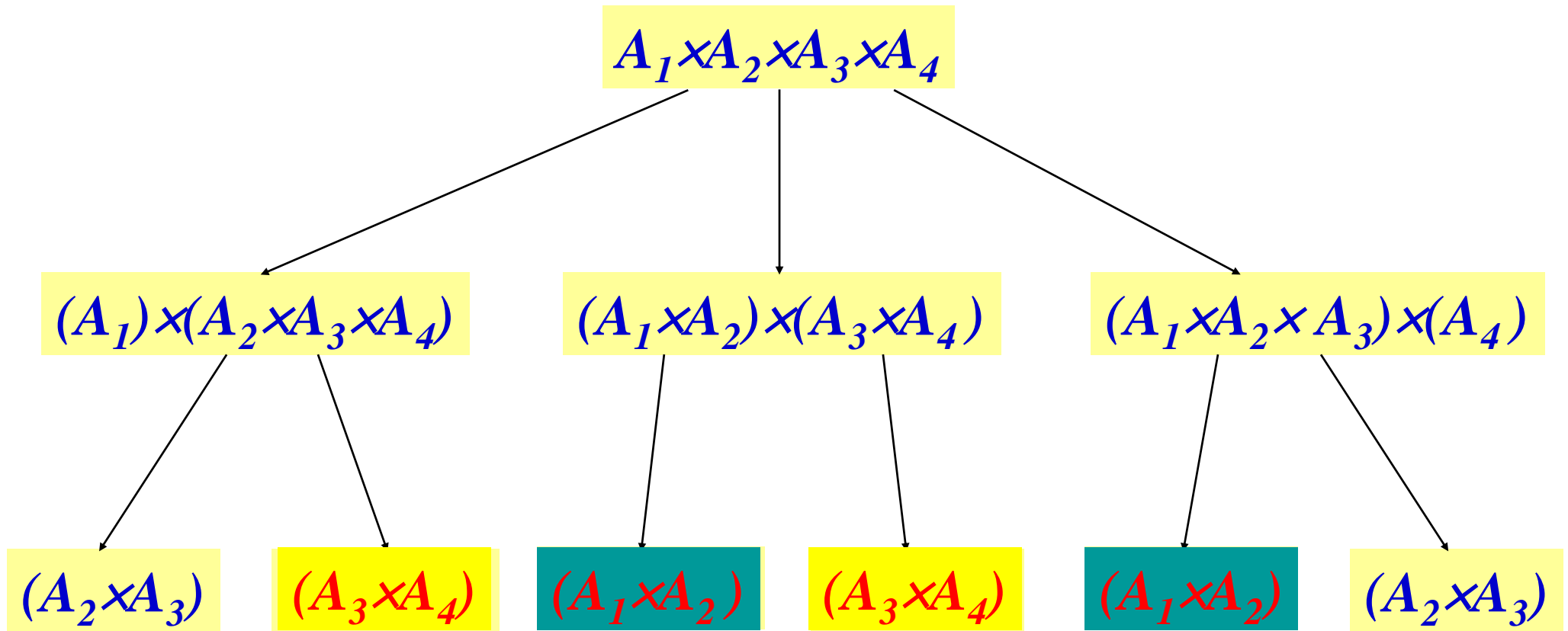
$$((A_1 \times \dots \times A_i) \times (A_{i+1} \times \dots \times A_k)) \times ((A_{k+1} \times \dots \times A_j) \times (A_{j+1} \times \dots \times A_n))$$

- 否则, 若优化解中给出的子问题 $A_{1\sim k}$ 的计算顺序不是 $A_{1\sim k}$ 的优化顺序, 则一定存在 $A_{1\sim k}$ 的一个计算代价更小的优化顺序:

$$((A_1 \times \dots \times A_r) \times (A_{r+1} \times \dots \times A_k))$$

- 用其替代 $A_{1\sim n}$ 的优化解中 $A_{1\sim k}$ 的计算顺序, 将会得到一个计算代价更小的解, 则与 $A_{1\sim n} = A_{1\sim k} \times A_{k+1\sim n}$ 是优化顺序相矛盾了.
- 对子问题 $A_{k+1\sim n}$ 亦同理.

- 子问题重叠性



具有子问题重叠性



# 递归地定义最优解的代价

- 递归求解过程

— 一般化表示：计算子链 $A_i A_{i+1} \dots A_j$ 的最优乘法方案

$$A_i \times A_{i+1} \times \dots \times A_j = \left\{ \begin{array}{l} (A_i) \quad \times (A_{i+1} \times \dots \times A_j) \\ (A_i \times A_{i+1}) \quad \times (A_{i+2} \times \dots \times A_j) \\ \dots \\ (A_i \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_j) \\ \dots \\ (A_i \times \dots \times A_{j-1}) \times (A_j) \end{array} \right.$$



# 递归地定义最优解的代价

$$A_i \times A_{i+1} \times \dots \times A_j = \left\{ \begin{array}{l} (A_i) \quad \times (A_{i+1} \times \dots \times A_j) \\ (A_i \times A_{i+1}) \quad \times (A_{i+2} \times \dots \times A_j) \\ \dots \\ (A_i \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_j) \\ \dots \\ (A_i \times \dots \times A_{j-1}) \times (A_j) \end{array} \right.$$

$$cost_{i \sim j} = \text{Min}_{i \leq k \leq j-1} \left\{ \begin{array}{l} cost_{i \sim i} + cost_{i+1 \sim j} + p_{i-1} p_i p_j \\ cost_{i \sim i+1} + cost_{i+2 \sim j} + p_{i-1} p_{i+1} p_j \\ \dots \\ Cost_{i \sim k} + cost_{k+1 \sim j} + p_{i-1} p_k p_j \\ \dots \\ cost_{i \sim j-1} + cost_{j \sim j} + p_{i-1} p_{j-1} p_j \end{array} \right. \quad \text{其中 } A_r \text{ 是 } p_{r-1} \times p_r \text{ 矩阵}$$



# 递归地定义最优解的代价

- 假设

- $m[i, j]$  = 计算  $A_{i \sim j}$  的最小乘法数

- $m[1, n]$  = 计算  $A_{1 \sim n}$  的最小乘法数

$$(A_i \dots A_k)(A_{k+1} \dots A_j)$$

考虑到所有的  $k$ , 优化解的代价方程为

$$m[i, j] = 0 \quad \text{if } i = j$$

$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \} \quad \text{if } i < j$$





# 递归地划分子问题

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

$m[i, i]$

$m[i, i+1]$

.....

$m[i, j-1]$

$m[i, j]$

$m[i+1, j]$

$m[i+2, j]$

.....

$m[j, j]$



# 递归地划分子问题

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

$m[1,1]$	$m[1,2]$	$m[1,3]$	$m[1,4]$	$m[1,5]$
	$m[2,2]$	$m[2,3]$	$m[2,4]$	$m[2,5]$
		$m[3,3]$	$m[3,4]$	$m[3,5]$
			$m[4,4]$	$m[4,5]$
				$m[5,5]$

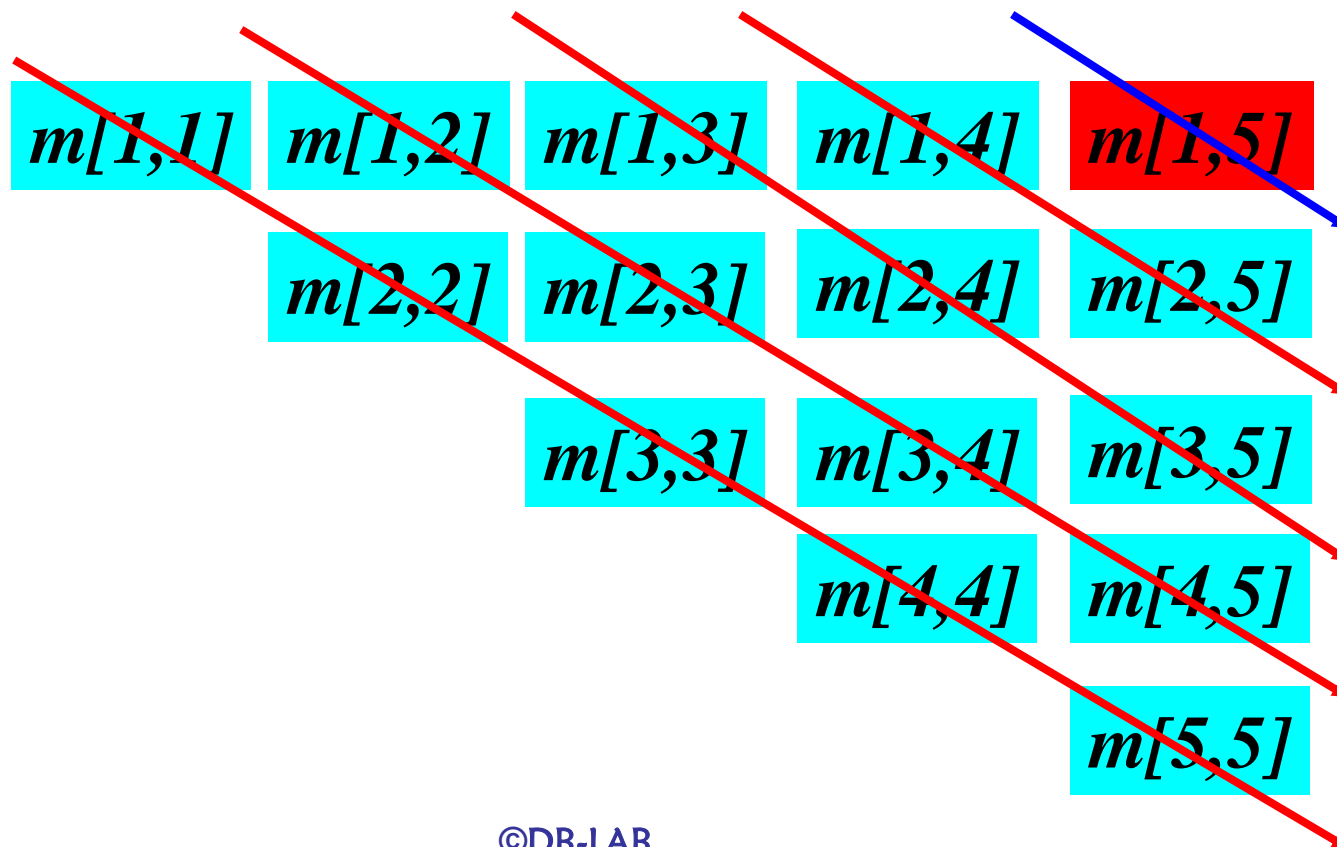


# 自底向上计算优化解的代价

$$m[i, j] = 0$$

if  $i = j$

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



# 算法描述

$$m[i, j] = 0 \quad \text{if } i = j$$

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

**Matrix-Chain-Order( $p$ )**

$n = \text{length}(p) - 1;$

**FOR**  $i = 1$  **TO**  $n$  **DO**

$m[i, i] = 0;$

**FOR**  $l = 2$  **TO**  $n$  **DO** /\* 计算对角线 \*/

**FOR**  $i = 1$  **TO**  $n - l + 1$  **DO**

$j = i + l - 1;$

$m[i, j] = \infty;$

**FOR**  $k \leftarrow i$  **TO**  $j - 1$  **DO** /\* 计算  $m[i, j]$  \*/

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

**IF**  $q < m[i, j]$  **THEN**  $m[i, j] = q;$

**Return**  $m.$

$m[1,1]$

$m[1,2]$

$m[1,3]$

$m[1,4]$

$m[1,5]$

$m[2,2]$

$m[2,3]$

$m[2,4]$

$m[2,5]$

$m[3,3]$

$m[3,4]$

$m[3,5]$

$m[4,4]$

$m[4,5]$

$m[5,5]$

# 获取构造最优解的信息

$$m[i, j] = 0 \quad \text{if } i = j$$

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

Matrix-Chain-Order( $p$ )

$n = \text{length}(p) - 1;$

FOR  $i = 1$  TO  $n$  DO

$m[i, i] = 0;$

FOR  $l = 2$  TO  $n$  DO /\* 计算对角线 \*/

    FOR  $i = 1$  TO  $n - l + 1$  DO

$j = i + l - 1;$

$m[i, j] = \infty;$

        FOR  $k \leftarrow i$  TO  $j - 1$  DO /\* 计算  $m[i, j]$  \*/

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

            IF  $q < m[i, j]$  THEN  $m[i, j] = q; \quad s[i, j] = k;$

Return  $m$  and  $s$ .

$m[1,1]$	$m[1,2]$	$m[1,3]$	$m[1,4]$	$m[1,5]$
	$m[2,2]$	$m[2,3]$	$m[2,4]$	$m[2,5]$
		$m[3,3]$	$m[3,4]$	$m[3,5]$
			$m[4,4]$	$m[4,5]$
				$m[5,5]$

$S[i, j] = k$  记录  $A_i A_{i+1} \dots A_j$  的最优划分处是在  $A_k$  与  $A_{k+1}$  之间



# 获取构造最优解的信息

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

- **Matrix-Chain-Order( $p$ )**
- $n = \text{length}(p) - 1$ ;
- **FOR**  $i=1$  **TO**  $n$  **DO**
- $m[i, i] = 0$ ;
- **FOR**  $l=2$  **TO**  $n$  **DO**
- **FOR**  $i=1$  **TO**  $n-l+1$  **DO**
- $j = i+l-1$ ;
- $m[i, j] = \infty$ ;
- **FOR**  $k \leftarrow i$  **TO**  $j-1$  **DO**
- $q = m[i, k] + m[k+1, j] + p_{i-1} \times p_k \times p_j$
- **IF**  $q < m[i, j]$  **THEN**  $m[i, j] = q$ ,  $s[i, j] = k$ ;
- **Return**  $m$  and  $s$ .

时间复杂性:  $O(n^3)$



Print-Optimal-Parens( $s, i, j$ )

IF  $j=i$

THEN Print “A” <sub>$i$</sub> ;

ELSE Print “(”

Print-Optimal-Parens( $s, i, s[i, j]$ )

Print-Optimal-Parens( $s, s[i, j]+1, j$ )

Print “)”

$S[i, j]$ 记录 $A_i \dots A_j$ 的最优划分处;

$S[i, S[i, j]]$ 记录 $A_i \dots A_{s[i, j]}$ 的最优划分处;

$S[S[i, j]+1, j]$ 记录 $A_{s[i, j]+1} \dots A_j$ 的最优划分处.

调用Print-Optimal-Parens( $s, 1, n$ )

即可输出 $A_{1 \sim n}$ 的优化计算顺序



- 时间复杂性
  - 计算代价的时间
    - $(l, i, k)$  三层循环, 每层至多  $n-1$  步
    - $O(n^3)$
  - 构造最优解的时间:  $O(n)$
  - 总时间复杂性为:  $O(n^3)$
- 空间复杂性
  - 使用数组  $m$  和  $S$
  - 需要空间  $O(n^2)$

*Hu, TC; Shing, MT (1982). "Computation of Matrix Chain Products, Part I"*  
*Hu, TC; Shing, MT (1984). "Computation of Matrix Chain Products, Part II"*





## 4.3 Longest Common Susequence

- 问题定义
- 问题求解
  - 优化解的结构分析
  - 建立优化解的代价递归方程
  - 递归地划分子问题
  - 自底向上计算优化解的代价  
记录优化解的构造信息
  - 构造优化解



- 子序列

- $X=(A, B, C, B, D, B)$

- $W=(B, D, A)$  是  $X$  的子序列?

- $Z=(B, C, D, B)$  是  $X$  的子序列?

- 公共子序列

- $Z$  是序列  $X$  与  $Y$  的公共子序列如果  $Z$  是  $X$  的子序列也是  $Y$  的子序列。



## 最长公共子序列 (LCS) 问题

输入:  $X = (x_1, x_2, \dots, x_m)$ ,  $Y = (y_1, y_2, \dots, y_n)$

输出:  $X$ 与 $Y$ 的最长公共子序列

$$Z = (z_1, z_2, \dots, z_k)$$



- 第 $i$ 前缀

— 设 $X=(x_1, x_2, \dots, x_n)$ 是一个序列

则 $X_i=(x_1, \dots, x_i)$ 是 $X$ 的第 $i$ 前缀

例.  $X=(A, B, D, C, A)$ ,  $X_1=(A)$ ,  $X_2=(A, B)$ ,  $X_3=(A, B, D)$



$$X = (x_1, x_2, \dots, x_m)$$

$$Y = (y_1, y_2, \dots, y_n)$$

## • 优化子结构的猜想

$X$ 和 $Y$ 的LCS为 $LCS_{XY} = (z_1, \dots, z_k)$

If  $x_m = y_n$       则  $z_k = x_m = y_n$

$$LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + \langle x_m = y_n \rangle$$

If  $x_m \neq y_n$ ,

$$\left. \begin{array}{l} z_k \neq x_m \quad LCS_{XY} = LCS_{X_{m-1}Y} \\ z_k \neq y_n \quad LCS_{XY} = LCS_{XY_{n-1}} \end{array} \right\} LCS_{XY} = \max \{ LCS_{X_{m-1}Y}, LCS_{XY_{n-1}} \}$$



- 优化子结构

**定理1 (优化子结构)** 设  $X=(x_1, \dots, x_m)$ ,  $Y=(y_1, \dots, y_n)$  是两个序列,  $LCS_{XY}=(z_1, \dots, z_k)$  是  $X$  与  $Y$  的  $LCS$ , 我们有:

(1) 如果  $x_m=y_n$ , 则  $z_k=x_m=y_n$ ,  $LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + \langle x_m=y_n \rangle$ ,

$LCS_{X_{m-1}Y_{n-1}}$  是  $X_{m-1}$  和  $Y_{n-1}$  的  $LCS$ .

(2) 如果  $x_m \neq y_n$ , 且  $z_k \neq x_m$ , 则  $LCS_{XY}$  是  $X_{m-1}$  和  $Y$  的  $LCS$ , 即

$$LCS_{XY} = LCS_{X_{m-1}Y}$$

(3) 如果  $x_m \neq y_n$ , 且  $z_k \neq y_n$ , 则  $LCS_{XY}$  是  $X$  与  $Y_{n-1}$  的  $LCS$ , 即

$$LCS_{XY} = LCS_{XY_{n-1}}$$



## 证明:

(1).  $X = \langle x_1, \dots, x_{m-1}, x_m \rangle$ ,  $Y = \langle y_1, \dots, y_{n-1}, x_m \rangle$ , 则

$$z_k = x_m = y_n \text{ 且 } LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + \langle x_m = y_n \rangle.$$

设  $z_k \neq x_m$ , 则可加  $x_m = y_n$  到  $Z$ , 得到一个长为  $k+1$  的  $X$  与  $Y$  的公共序列, 与  $Z$  是  $X$  和  $Y$  的  $LCS$  矛盾。于是,  $z_k = x_m = y_n$ 。

设存在  $X_{m-1}$  与  $Y_{n-1}$  的非最长公共子序列  $Z_{k-1}$ , 使得

$$LCS_{XY} = Z_{k-1} + \langle x_m = y_n \rangle,$$

则由于  $|Z_{k-1}| < |LCS_{X_{m-1}Y_{n-1}}|$ ,

$$|LCS_{XY} = Z_{k-1} + \langle x_m = y_n \rangle| < |LCS_{X_{m-1}Y_{n-1}} + \langle x_m = y_n \rangle|,$$

与  $LCS_{XY}$  是  $LCS$  矛盾。



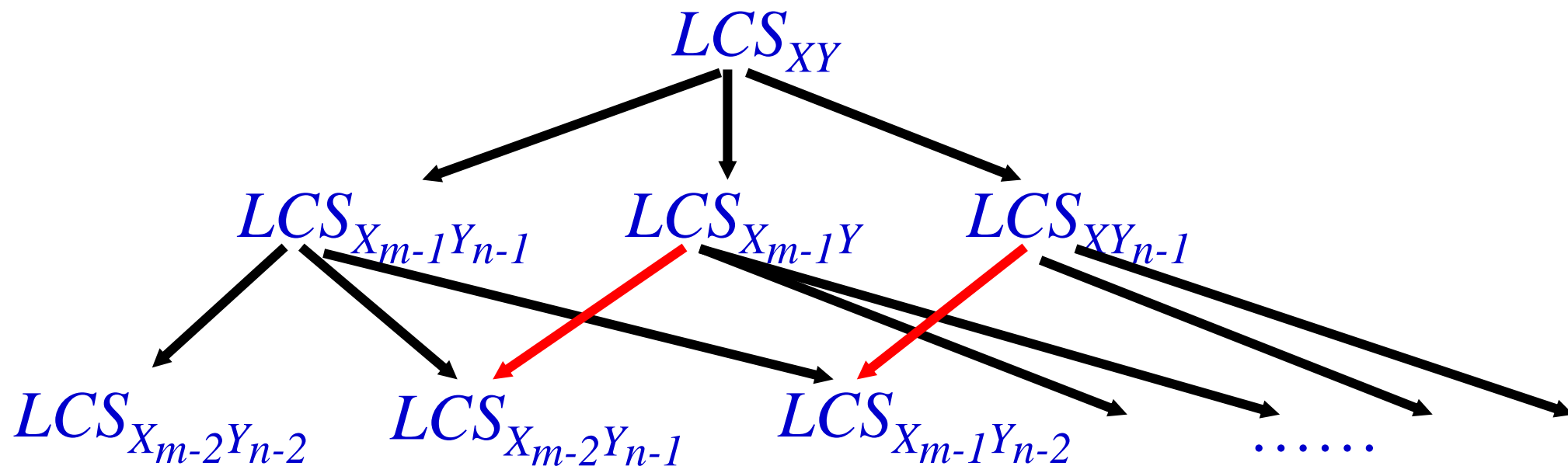
(2)  $X = \langle x_1, \dots, x_{m-1}, x_m \rangle$ ,  $Y = \langle y_1, \dots, y_{n-1}, y_n \rangle$ ,  
 $x_m \neq y_n$ ,  $z_k \neq x_m$ , 则  $LCS_{XY} = LCS_{X_{m-1}Y}$

由于  $z_k \neq x_m$ ,  $Z = LCS_{XY}$  是  $X_{m-1}$  与  $Y$  的公共子序列。  
我们来证  $Z$  是  $X_{m-1}$  与  $Y$  的  $LCS$ 。设  $X_{m-1}$  与  $Y$  有一个公共子序列  $W$ ,  $W$  的长大于  $k$ , 则  $W$  也是  $X$  与  $Y$  的公共子序列, 与  $Z$  是  $LCS$  矛盾。

(3) 证明同(2)。



- 子问题重叠性



**LCS问题具有子问题重叠性**

**递归方式需要处理多少个子问题？**



# 建立LCS长度的递归方程

- $C[i, j]$  =  $X_i$ 与 $Y_j$ 的LCS的长度
- LCS长度的递归方程

$$C[i, j] = 0 \quad \text{if } i=0 \text{ 或 } j=0$$

$$C[i, j] = C[i-1, j-1] + 1 \quad \text{if } i, j > 0 \text{ and } x_i = y_j$$

$$C[i, j] = \text{Max}(C[i, j-1], C[i-1, j]) \quad \text{if } i, j > 0 \text{ and } x_i \neq y_j$$



# 递归划分与自底向上求解

- 基本思想

$$C[i, j] = 0, \text{ if } i=0 \text{ 或 } j=0$$

$$C[i, j] = C[i-1, j-1] + 1 \quad \text{if } i, j > 0 \text{ and } x_i = y_j$$

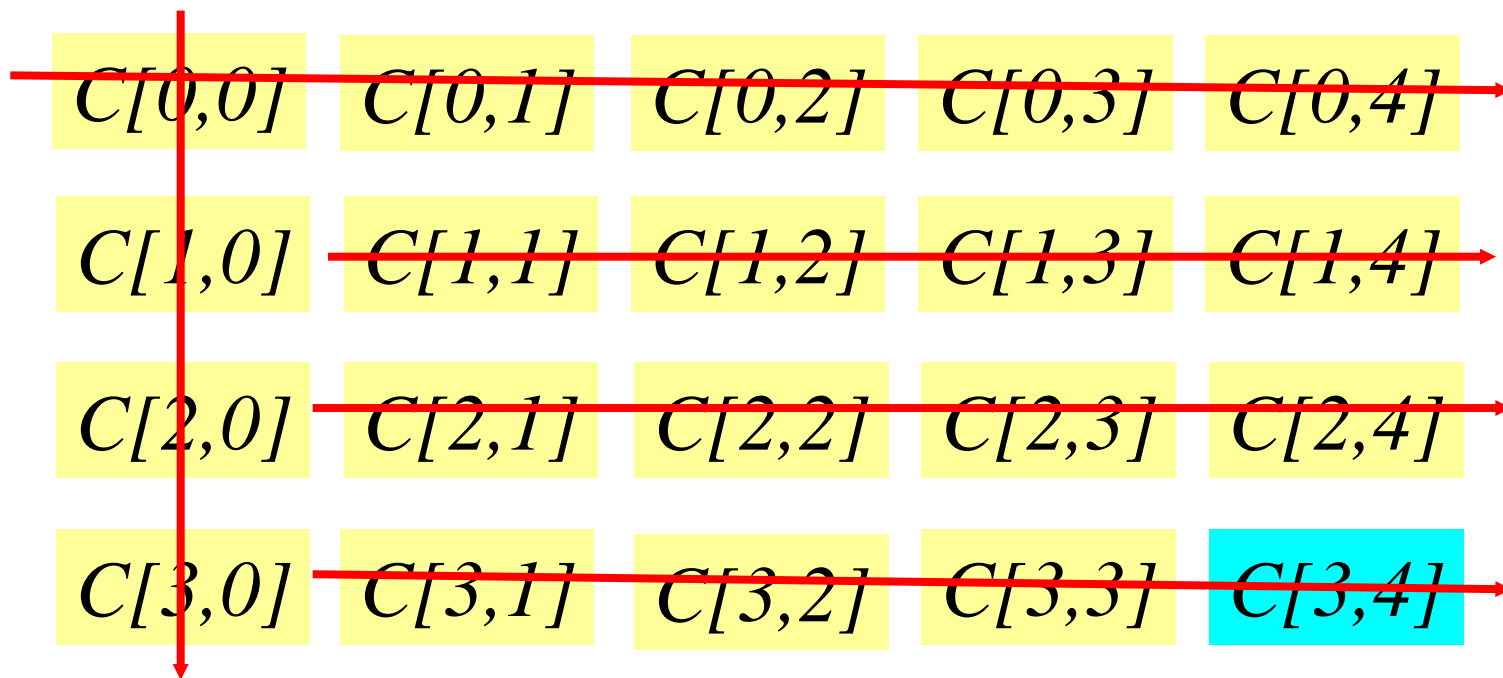
$$C[i, j] = \text{Max}(C[i, j-1], C[i-1, j]) \quad \text{if } i, j > 0 \text{ and } x_i \neq y_j$$

	$C[i-1, j-1]$	$C[i-1, j]$	
	$C[i, j-1]$	$C[i, j]$	

$C[i-1, j-1]$	$C[i-1, j]$
$C[i, j-1]$	$C[i, j]$

## 自底向上计算优化代价

- 递归划分分子问题与自底向上求解过程



- $C[i, j] = 0, i=0$  或  $j=0$
- $C[i, j] = C[i-1, j-1] + 1, x_i = y_j$
- $C[i, j] = \text{Max}(C[i, j-1], C[i-1, j]), x_i \neq y_j$

		$y_j$	<b>B</b>	<b>D</b>	<b>C</b>	<b>A</b>	<b>B</b>	<b>A</b>
$i=0$	$x_i$	0	0	0	0	0	0	0
	<b>A</b>	0	0	0	0	1	1	1
	<b>B</b>	0	1	1	1	1	2	2
	<b>C</b>	0	1	1	2	2	2	2
	<b>B</b>	0	1	1	2	2	3	3
	<b>D</b>	0	1	2	2	2	3	3
	<b>A</b>	0	1	2	2	3	3	4
	<b>B</b>	0	1	2	2	3	4	4
		$j=0$						



- 计算LCS长度的算法
  - 数据结构

$C[0:m, 0:n]$ :  $C[i, j]$  是  $X_i$  与  $Y_j$  的LCS的长度

$B[1:m, 1:n]$ :  $B[i, j]$  记录优化解的信息

记录优化解信息

- $C[i, j] = 0, i=0 \text{ 或 } j=0$
- $C[i, j] = C[i-1, j-1] + 1, x_i = y_j$
- $C[i, j] = \text{Max}(C[i, j-1], C[i-1, j]), x_i \neq y_j$

		$y_j$	<i>B</i>	<i>D</i>	<i>C</i>	<i>A</i>	<i>B</i>	<i>A</i>
$i=0$	$x_i$	0	0	0	0	0	0	0
	<i>A</i>	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
	<i>B</i>	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
	<i>C</i>	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
	<i>B</i>	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
	<i>D</i>	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
	<i>A</i>	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
	<i>B</i>	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4
		$j=0$						

LCS-length( $X, Y$ )

$m \leftarrow \text{length}(X); n \leftarrow \text{length}(Y);$

For  $i \leftarrow 0$  To  $m$  Do

$C[i,0] \leftarrow 0;$

For  $j \leftarrow 1$  To  $n$  Do

$C[0,j] \leftarrow 0;$

For  $i \leftarrow 1$  To  $m$  Do

For  $j \leftarrow 1$  To  $n$  Do

If  $x_i = y_j$

Then  $C[i,j] \leftarrow C[i-1,j-1] + 1;$

$B[i,j] \leftarrow "\nwarrow";$

Else If  $C[i-1,j] \geq C[i,j-1]$

Then  $C[i,j] \leftarrow C[i-1,j];$

$B[i,j] \leftarrow "\uparrow";$

Else  $C[i,j] \leftarrow C[i,j-1];$

$B[i,j] \leftarrow "\leftarrow";$

Return  $C$  and  $B$ .

- $C[i, j] = 0, i=0$  或  $j=0$
- $C[i, j] = C[i-1, j-1] + 1, x_i = y_j$
- $C[i, j] = \text{Max}(C[i, j-1], C[i-1, j]), x_i \neq y_j$

$C[0,0]$	$C[0,1]$	$C[0,2]$	$C[0,3]$	$C[0,4]$
$C[1,0]$	$C[1,1]$	$C[1,2]$	$C[1,3]$	$C[1,4]$
$C[2,0]$	$C[2,1]$	$C[2,2]$	$C[2,3]$	$C[2,4]$
$C[3,0]$	$C[3,1]$	$C[3,2]$	$C[3,3]$	$C[3,4]$

	$y_j$	<i>B</i>	<i>D</i>	<i>C</i>	A	<i>B</i>	<i>A</i>
$x_i$	0	0	0	0	0	0	0
A	0	↑0	↑0	↑0	↖1	←1	↖1
<i>B</i>	0	↖1	←1	←1	↑1	↖2	←2
<i>C</i>	0	↑1	↑1	↖2	←2	↑2	↑2
<i>B</i>	0	↖1	↑1	↑2	↑2	↖3	←3
<i>D</i>	0	↑1	↖2	↑2	↑2	↑3	↑3
<i>A</i>	0	↑1	↑2	↑2	↖3	↑3	↖4
<i>B</i>	0	↖1	↑2	↑2	↑3	↖4	↑4

AB





- 基本思想

- 从  $B[m, n]$  开始按指针搜索
- 若  $B[i, j] = “\nwarrow”$ , 则  $x_i = y_j$  是  $LCS$  的一个元素
- 如此找到的 “ $LCS$ ” 是  $X$  与  $Y$  的  $LCS$  的 *Inverse*



	$y_j$	<i>B</i>	<i>D</i>	<i>C</i>	<i>A</i>	<i>B</i>	<i>A</i>
$x_i$	0	0	0	0	0	0	0
<i>A</i>	0	↑0	↑0	↑0	↖1	←1	↖1
<i>B</i>	0	↖1	←1	←1	↑1	↖2	←2
<i>C</i>	0	↑1	↑1	↖2	←2	↑2	↑2
<i>B</i>	0	↖1	↑1	↑2	↑2	↖3	←3
<i>D</i>	0	↑1	↖2	↑2	↑2	↑3	↑3
<i>A</i>	0	↑1	↑2	↑2	↖3	↑3	↖4
<i>B</i>	0	↖1	↑2	↑2	↑3	↖4	↑4

**Print-LCS(*B*, *X*, *i*, *j*)**

**If  $i=0$  or  $j=0$  Then Return;**

**If  $B[i, j]=“\nwarrow”$**

**Then Print-LCS(*B*, *X*,  $i-1$ ,  $j-1$ ); Print  $x_i$ ;**

**Else**

**If  $B[i, j]=“\uparrow”$**

**Then Print-LCS(*B*, *X*,  $i-1$ , *j*);**

**Else Print-LCS(*B*, *X*, *i*,  $j-1$ ).**

**Print-LCS(*B*, *X*, *n*, *m*)**

可打印出*X*与*Y*的LCS

$n=\text{length}(X)$

$m=\text{length}(Y)$

# 算法复杂性

```

LCS-length(X, Y)
m ← length(X); n ← length(Y);
For i ← 0 To m Do
    C[i, 0] ← 0;
For j ← 1 To n Do
    C[0, j] ← 0;
For i ← 1 To m Do
    For j ← 1 To n Do
        If  $x_i = y_j$ 
            Then  $C[i, j] \leftarrow C[i-1, j-1] + 1$ ;
             $B[i, j] \leftarrow \nwarrow$ ;
        Else If  $C[i-1, j] \geq C[i, j-1]$ 
            Then  $C[i, j] \leftarrow C[i-1, j]$ ;
             $B[i, j] \leftarrow \uparrow$ ;
        Else  $C[i, j] \leftarrow C[i, j-1]$ ;
             $B[i, j] \leftarrow \leftarrow$ ;
Return C and B.
    
```

	$y_j$	B	D	C	A	B	A
$x_i$	0	0	0	0	0	0	0
A	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\leftarrow 1$	$\nwarrow 1$
B	0	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$
C	0	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\uparrow 2$	$\uparrow 2$
B	0	$\nwarrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\leftarrow 3$
D	0	$\uparrow 1$	$\nwarrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\uparrow 3$
A	0	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\uparrow 3$	$\nwarrow 4$
B	0	$\nwarrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\nwarrow 4$	$\uparrow 4$

## • 时间复杂性

— 计算代价的时间

•  $(i, j)$  两层循环

•  $O(mn)$

— 构造最优解的时间:  $O(m+n)$

— 总时间复杂性为:  $O(mn)$

## • 空间复杂性

— 使用数组  $C$  和  $B$

— 需要空间  $O(mn)$

## • 空间优化策略



## 4.4 0/1 Knapsack Problem

- 问题定义
- 问题求解
  - 优化解的结构分析
  - 建立优化解代价的递归方程
  - 递归地划分子问题
  - 自底向上计算优化解的代价  
记录优化解的构造信息
  - 构造优化解



# 问题的定义

给定 $n$ 种物品和一个背包，物品 $i$ 的重量是 $w_i$ ，价值 $v_i$ ，背包承重为 $C$ ，问如何选择装入背包的物品，使装入背包中的物品的总价值最大？

对于每种物品只能选择完全装入或不装入，一个物品至多装入一次。



# 问题的定义

- 输入:  $C > 0, w_i > 0, v_i > 0, 1 \leq i \leq n$
- 输出:  $(x_1, x_2, \dots, x_n), x_i \in \{0, 1\}$ , 满足
$$\sum_{1 \leq i \leq n} w_i x_i \leq C, \sum_{1 \leq i \leq n} v_i x_i \text{ 最大}$$

## Naïve方法:

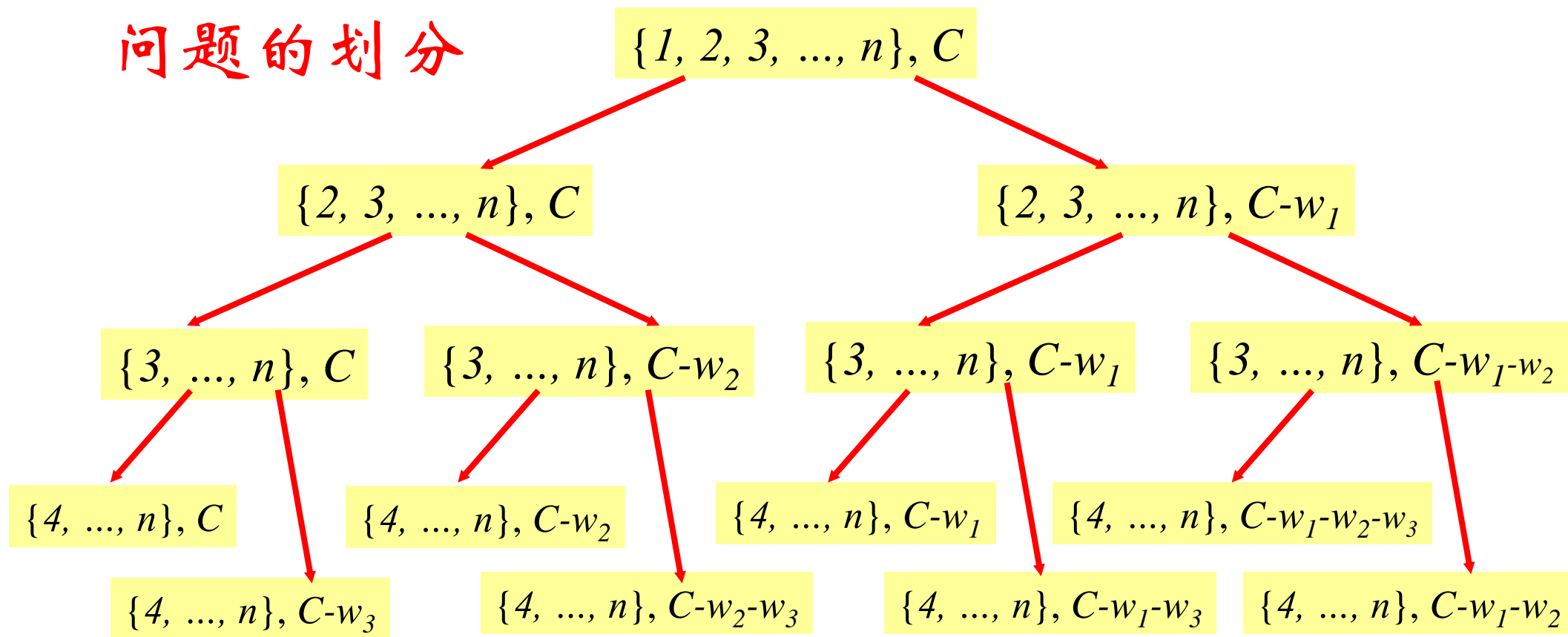
每个物品有两种选择: 1(装)或0(不装)  
 $n$ 个物品共 $2^n$ 个装取方案  
每个装取方案的计算代价为 $n$   
总计算代价为 $O(n2^n)$



- 优化解的结构分析
- 建立优化解代价的递归方程
- 递归地划分子问题
- 自底向上计算优化解的代价  
记录优化解的构造信息
- 构造优化解



## 问题的划分



当 $w_i$ 皆为1时, 存在大量重叠子问题







# 优化解结构的分析

**定理** 如果  $S_i = (y_i, y_{i+1}, \dots, y_n)$  是 0-1 背包子问题  $P_i = [\{i, i+1, \dots, n\}, C_i = C - \sum_{1 \leq k \leq i-1} w_k y_k]$  的优化解, 则  $(y_{i+1}, \dots, y_n)$  是如下子问题  $P_{i+1}$  的优化解:

$$\begin{aligned} \max & \sum_{i+1 \leq k \leq n} v_k x_k \\ \sum_{i+1 \leq k \leq n} w_k x_k & \leq C_i - w_i y_i \\ x_k & \in \{0, 1\}, \quad i+1 \leq k \leq n \end{aligned}$$

**证明:** 如果  $S_{i+1} = (y_{i+1}, \dots, y_n)$  不是子问题  $P_{i+1}$  的优化解, 则存在  $S'_{i+1} = (z_{i+1}, \dots, z_n)$  是  $P_{i+1}$  的更优解。  $S'_i = (y_i, z_{i+1}, \dots, z_n)$  是问题  $P_i$  之比  $S_i$  更优的解, 与  $S_i$  优化矛盾。



- 优化解的结构分析
- 建立优化解代价的递归方程
- 递归地划分子问题
- 自底向上计算优化解的代价  
记录优化解的构造信息
- 构造优化解



- 定义代价矩阵  $m$

矩阵元素  $m(i, j)$  表示子问题  $[(i, i+1, \dots, n), j]$  的优化解  $(x_i, x_{i+1}, \dots, x_n)$  的代价,  $m(i, j) = \sum_{i \leq k \leq n} v_k x_k$

- 形式地

问题

$$\max \sum_{i \leq k \leq n} v_k x_k$$

$$\sum_{i \leq k \leq n} w_k x_k \leq j$$

$$x_k \in \{0, 1\}, \quad i \leq k \leq n$$

的最优解代价为  $m(i, j) = \sum_{i \leq k \leq n} v_k x_k$ .



• 递归方程:

总结:

$$m(n, j) = 0, \quad 0 \leq j < w_n$$

$$m(n, j) = v_n, \quad j \geq w_n$$

$$m(i, j) = m(i+1, j), \quad 0 \leq j < w_i$$

$$m(i, j) = \max\{m(i+1, j), m(i+1, j-w_i) + v_i\}, \quad j \geq w_i$$

$$m(n, j) = v_n$$



- 优化解的结构分析
- 建立优化解代价的递归方程
- 递归地划分子问题
- 自底向上计算优化解的代价  
记录优化解的构造信息
- 构造优化解



$$m(i, j) = m(i+1, j), 0 \leq j < w_i$$

$$m(i, j) = \max\{m(i+1, j), m(i+1, j-w_i)+v_i\}, j \geq w_i$$

$$m(n, j) = 0, 0 \leq j < w_n$$

$$m(n, j) = v_n, j \geq w_n$$

$$\begin{array}{c|c} & m(i, j) \\ \hline m(i+1, j-w_i) & m(i+1, j) \end{array}$$

令  $n=4$ ,  $w_i = \text{整数}$

$$m(1, C)$$

$$m(2, C-w_1)$$

...

$$m(2, C)$$

...

$$m(3, C-w_1-w_2)$$

...

$$m(3, C-w_1)$$

...

$$m(3, C-w_2)$$

...

$$m(3, C)$$

...

$$m(4, C-w_2-w_3)$$

...

$$m(4, C-w_2)$$

...

$$m(4, C-w_3)$$

...

$$m(4, C)$$



$$m(i, j) = m(i+1, j), 0 \leq j < w_i$$

$$m(i, j) = \max\{m(i+1, j), m(i+1, j-w_i)+v_i\}, j \geq w_i$$

$$m(n, j) = 0, 0 \leq j < w_n$$

$$m(n, j) = v_n, j \geq w_n$$

	$m(i, j)$
$m(i+1, j-w_i)$	$m(i+1, j)$

令  $n=4$ ,  $w_i = \text{整数}$

$m(1, C)$

$m(2, 0)$	$\dots$	$m(2, w_2-1)$	$m(2, w_2)$	$\dots$	$m(2, C-1)$	$m(2, C)$
-----------	---------	---------------	-------------	---------	-------------	-----------

$m(3, 0)$	$\dots$	$m(3, w_3-1)$	$m(3, w_3)$	$\dots$	$m(3, C-1)$	$m(3, C)$
-----------	---------	---------------	-------------	---------	-------------	-----------

$m(4, 0)$	$\dots$	$m(4, w_4-1)$	$m(4, w_4)$	$\dots$	$m(4, C-1)$	$m(4, C)$
-----------	---------	---------------	-------------	---------	-------------	-----------





- 优化解的结构分析
- 建立优化解代价的递归方程
- 递归地划分子问题
- 自底向上计算优化解的代价  
记录优化解的构造信息
- 构造优化解



$$m(i, j) = m(i+1, j), 0 \leq j < w_i$$

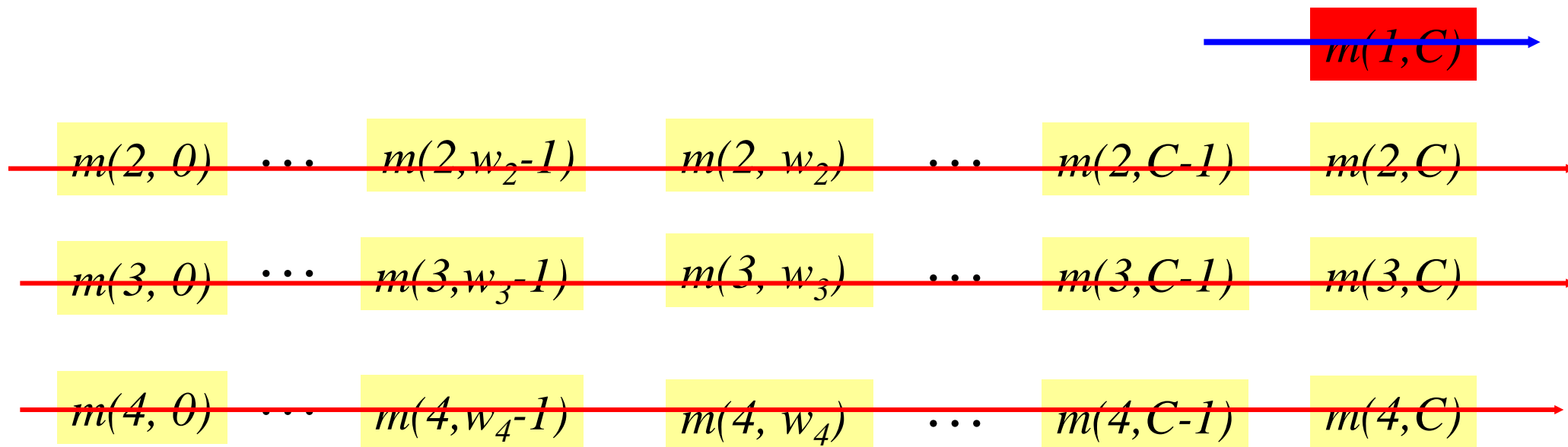
$$m(i, j) = \max\{m(i+1, j), m(i+1, j-w_i)+v_i\}, j \geq w_i$$

$$m(n, j) = 0, 0 \leq j < w_n$$

$$m(n, j) = v_n, j \geq w_n$$

令  $w_i = \text{整数}, n=4$

$$\begin{array}{c|c} & m(i, j) \\ \hline m(i+1, j-w_i) & m(i+1, j) \end{array}$$



# • 算法

(设  $w_i - 1 < C$ )

$$m(i, j) = m(i+1, j), 0 \leq j < w_i$$

$$m(i, j) = \max\{m(i+1, j), m(i+1, j-w_i)+v_i\}, j \geq w_i$$

$$m(n, j) = 0, 0 \leq j < w_n$$

$$m(n, j) = v_n, j \geq w_n$$

For  $j=0$  To  $w_n-1$  Do

$$m[n, j] = 0;$$

For  $j=w_n$  To  $C$  Do

$$m[n, j] = v_n;$$

For  $i=n-1$  To  $2$  Do

For  $j=0$  To  $w_i-1$  Do

$$m[i, j] = m[i+1, j];$$

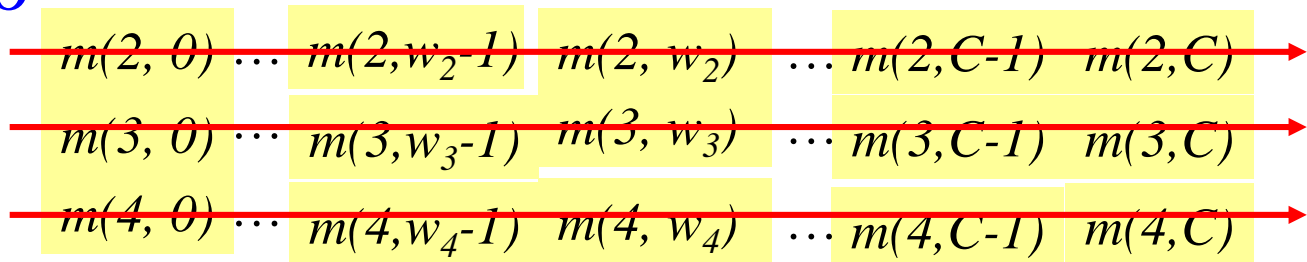
For  $j=w_i$  To  $C$  Do

$$m[i, j] = \max\{m[i+1, j], m[i+1, j-w_i]+v_i\};$$

If  $C < w_1$  Then  $m[1, C] = m[2, C];$

Else  $m[1, C] = \max\{m[2, C], m[2, C-w_1]+v_1\};$

$m(1, C)$





```
For  $j=0$  To  $\min(w_n-1, C)$  Do
     $m[n, j] = 0$ ;
For  $j=w_n$  To  $C$  Do
     $m[n, j] = v_n$ ;
For  $i=n-1$  To  $2$  Do
    For  $j=0$  To  $\min(w_i-1, C)$  Do
         $m[i, j] = m[i+1, j]$ ;
    For  $j=w_i$  To  $C$  Do
         $m[i, j] = \max\{m[i+1, j], m[i+1, j-w_i] + v_i\}$ ;
If  $C < w_1$  Then  $m[1, C] = m[2, C]$ ;
    Else  $m[1, C] = \max\{m[2, C], m[2, C-w_1] + v_1\}$ ;
```



- 优化解的结构分析
- 建立优化解代价的递归方程
- 递归地划分子问题
- 自底向上计算优化解的代价  
记录优化解的构造信息
- 构造优化解



1.  $m(1, C)$ 是最优解代价值，相应解计算如下：

If  $m(1, C) = m(2, C)$

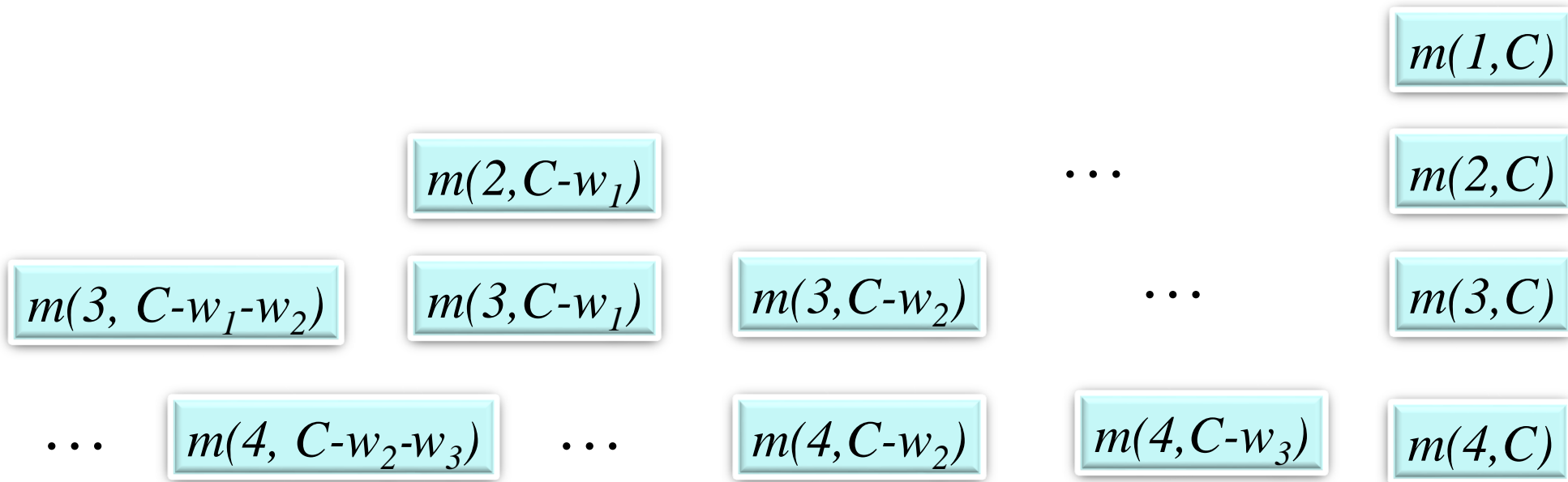
Then  $x_1 = 0$ ;

Else  $x_1 = 1$ ;

	$m(i, j)$
$m(i+1, j-w_i)$	$m(i+1, j)$

2. 如果  $x_1=0$ ，由  $m(2, C)$ 继续构造最优解；

3. 如果  $x_1=1$ ，由  $m(2, C-w_1)$ 继续构造最优解。





## • 时间复杂性

— 计算代价时间

•  $O(Cn)$

— 构造最优解时间:

$O(n)$

— 总时间复杂性为:

$O(Cn)$

## • 空间复杂性

— 使用数组  $m$

— 需要空间  $O(Cn)$

```
For j=0 To min( $w_n-1$ , C) Do
     $m[n, j] = 0$ ;
For j= $w_n$  To C Do
     $m[n, j] = v_n$ ;
For i=n-1 To 2 Do
    For j=0 To min( $w_i-1$ , C) Do
         $m[i, j] = m[i+1, j];$ 
```

这是一个伪多项式算法!

If 当  $C=2^n$  时:

$$T(n) = O(n2^n)$$

当  $w_i$  不限定为正整数时:

$$T(n) = O(2^n)$$

3. If  $x_1=1$ , 由  $m(2, C-w_1)$  继续构造  $x_2$ ;

.....



HITWH  
SE

部分背包问题?





## 4.5 The Optimal Binary Search Trees

- 问题定义
- 问题求解
  - 优化解的结构分析
  - 建立优化解代价的递归方程
  - 递归地划分子问题
  - 自底向上计算优化解的代价  
记录优化解的构造信息
  - 构造优化解



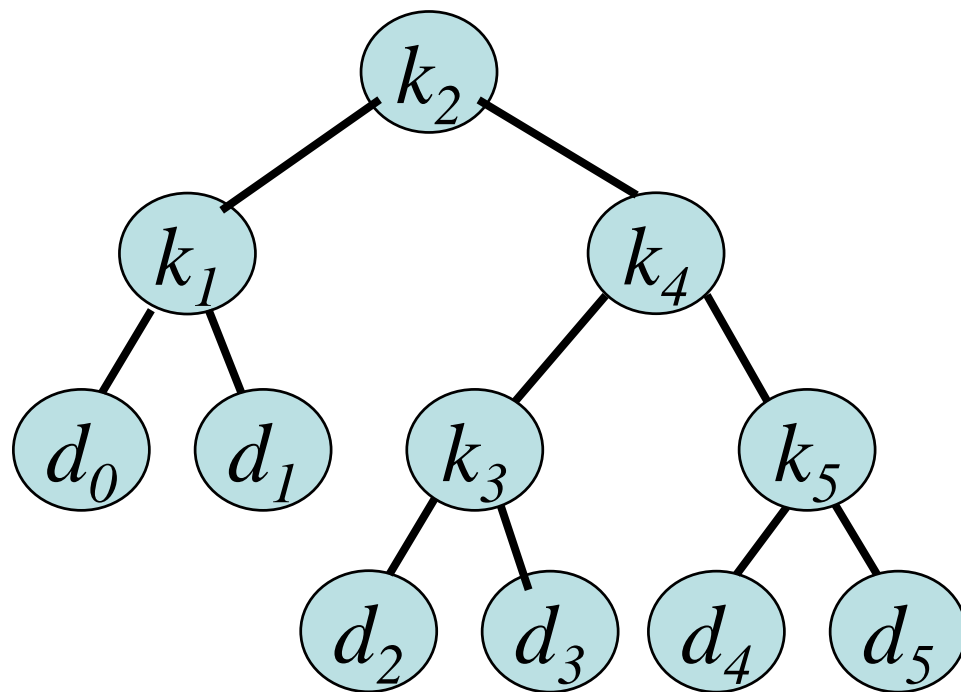
## • 二叉搜索树 $T$

### — 结点

- $K = \{k_1, k_2, \dots, k_n\}$
- $D = \{d_0, d_1, \dots, d_n\}$
- $d_i$  对应区间  $(k_i, k_{i+1})$   
 $d_0$  对应区间  $(-\infty, k_1)$   
 $d_n$  对应区间  $(k_n, +\infty)$

### — 附加信息

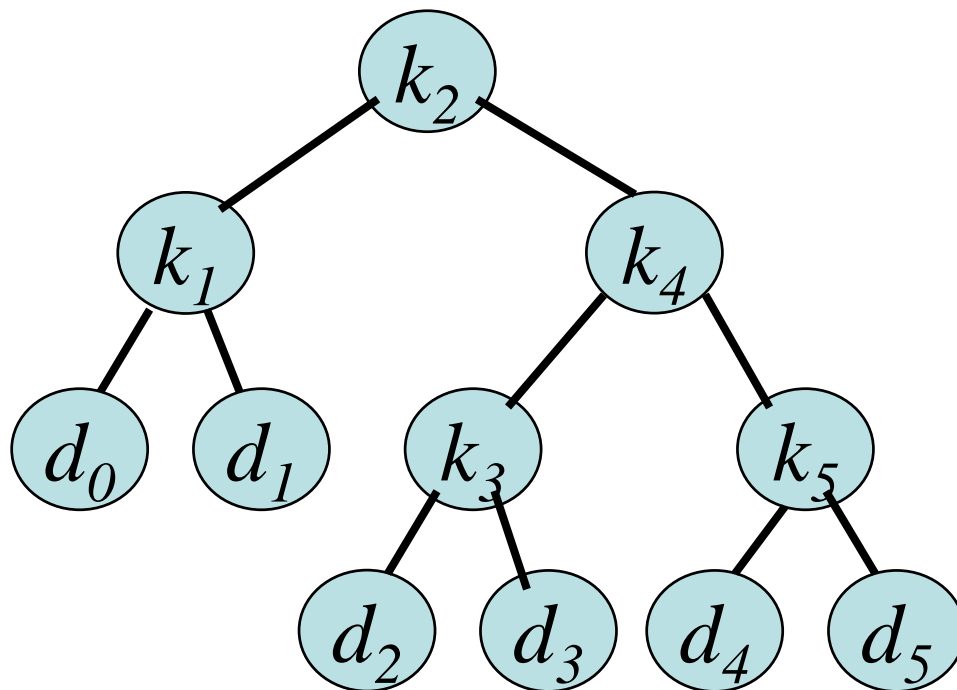
- 搜索  $k_i$  的概率为  $p_i$
- 搜索  $d_i$  的概率为  $q_i$



$$\sum_{i=1}^n p_i + \sum_{j=0}^n q_j = 1$$



- 搜索树的期望代价(用检查树顶点的个数衡量)



$$E(T) = \sum_{i=1}^n (DEP_T(k_i) + 1) p_i + \sum_{j=0}^n (DEP_T(d_j) + 1) q_j$$



## • 问题的定义

输入:  $K=\{k_1, k_2, \dots, k_n\}, k_1 < k_2 < \dots < k_n,$

$P=\{p_1, p_2, \dots, p_n\}, p_i$  为搜索  $k_i$  的概率

$Q=\{q_0, q_1, \dots, q_n\}, q_i$  为搜索值  $d_i$  的概率

输出: 构造  $K$  的二叉搜索树  $T$ , 最小化

$$E(T) = \sum_{i=1}^n (DEP_T(k_i) + 1) p_i + \sum_{j=0}^n (DEP_T(d_j) + 1) q_j$$



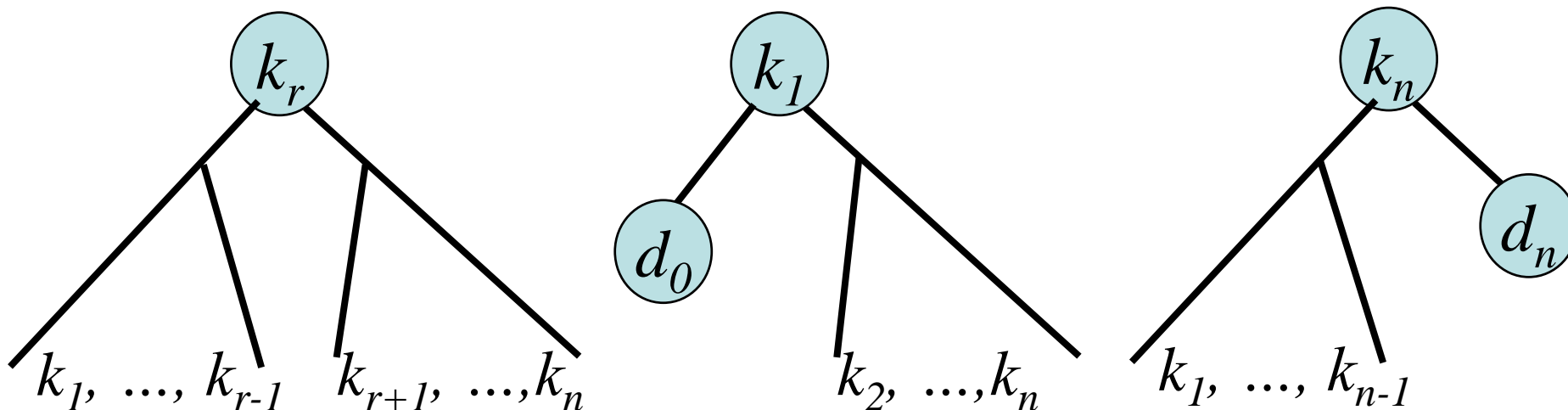
- 优化解的结构分析
- 建立优化解代价的递归方程
- 递归地划分子问题
- 自底向上计算优化解的代价  
记录优化解的构造信息
- 构造优化解



# 优化二叉搜索树结构的分析

- 优化解的结构观察

$K=\{k_1, k_2, \dots, k_n\}$  的优化解的根必为  $K$  中某个  $k_r$



如果  $r=1$ , 左子树仅包含  $d_0$

如果  $r=n$ , 右子树仅包含  $d_n$

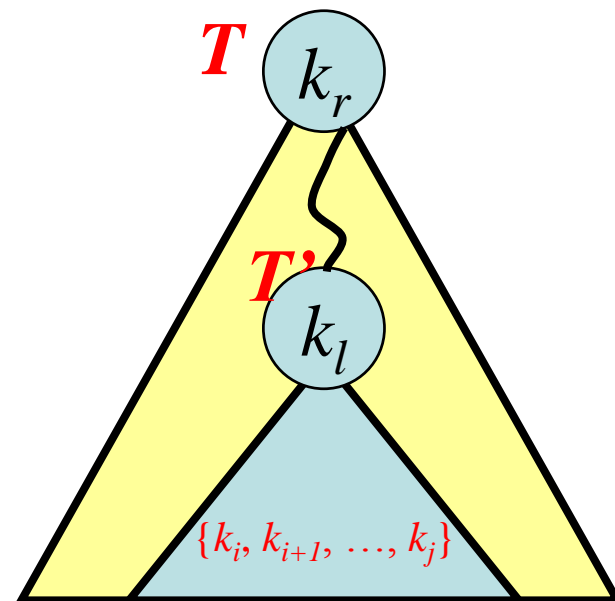


## • 优化子结构

**定理.** 如果优化二叉搜索树 $T$ 具有包含关键字集合 $\{k_i, k_{i+1}, \dots, k_j\}$ 的子树 $T'$ , 则 $T'$ 是关于关键字集合 $\{k_i, k_{i+1}, \dots, k_j\}$ 的子问题的优化解.

**证明:** 若不然, 必有关键字集 $\{k_i, k_{i+1}, \dots, k_j\}$ 子树 $T''$ ,  $T''$ 的期望搜索代价低于 $T'$ .

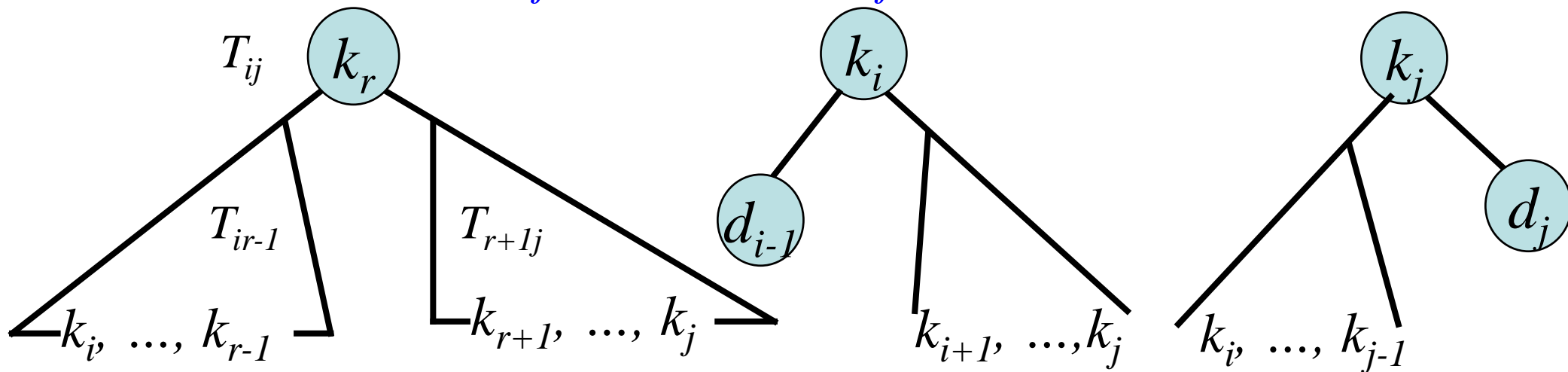
用 $T''$ 替换 $T$ 中的 $T'$ , 可以得到一个期望搜索代价比 $T$ 小的原始问题的二叉搜索树。与 $T$ 是最优解矛盾.





- 用优化子结构从子问题优化解构造优化解

$K = \{k_i, k_{i+1}, \dots, k_j\}$  的优化解  $T_{ij}$  的根必为  $K$  中某个  $k_r$



只要对于每个  $k_r \in K$ , 确定  $\{k_i, \dots, k_{r-1}\}$  和  $\{k_{r+1}, \dots, k_j\}$  的优化解, 我们就可以求出  $K$  的优化解.

如果  $r=i$ , 左子树  $T_{ii-1} = \{k_i, \dots, k_{i-1}\}$  仅包含  $d_{i-1}$

如果  $r=j$ , 右子树  $T_{j+1j} = \{k_{j+1}, \dots, k_j\}$  仅包含  $d_j$

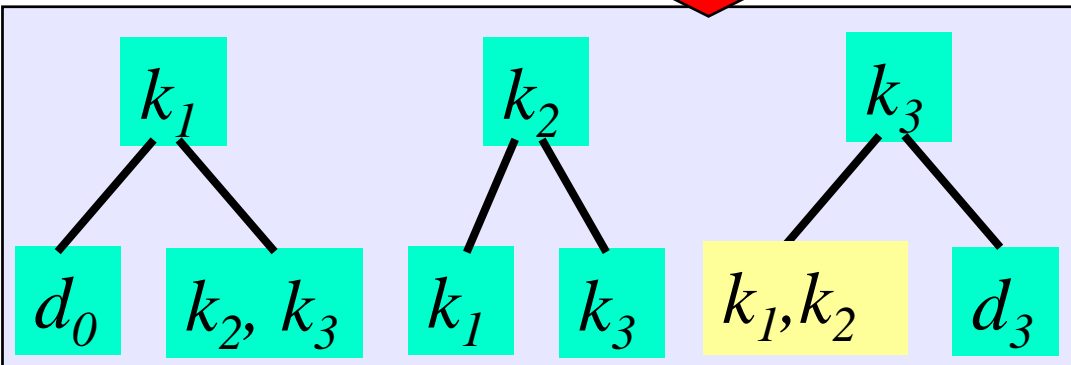
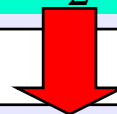
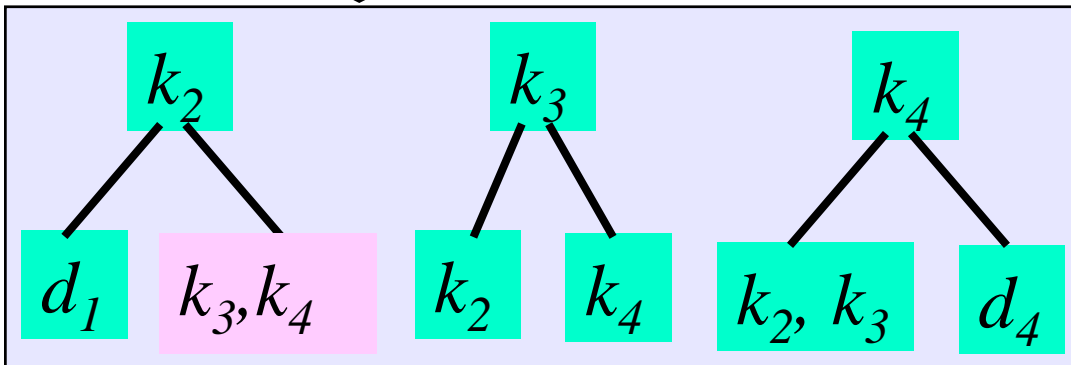
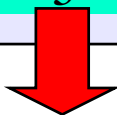
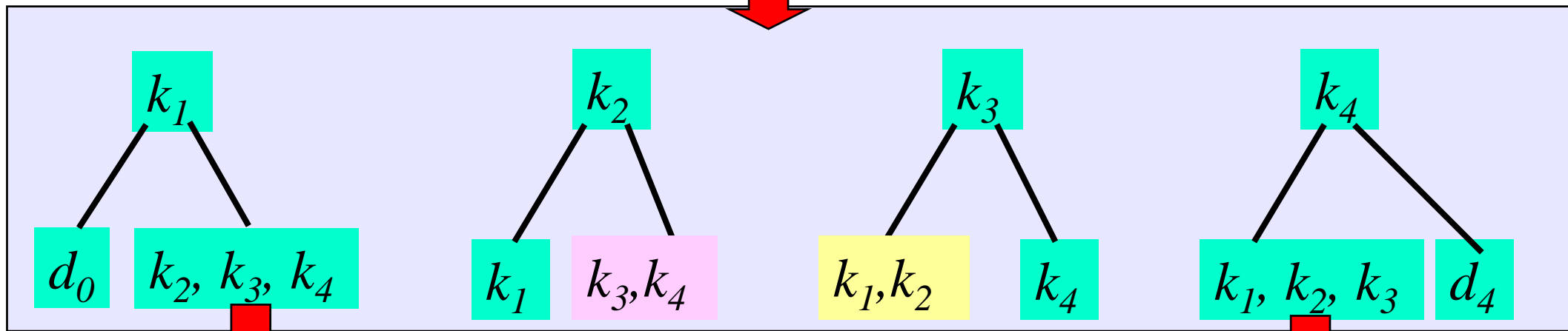
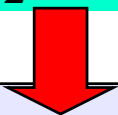




# 子问题重叠性



$k_1, k_2, k_3, k_4$



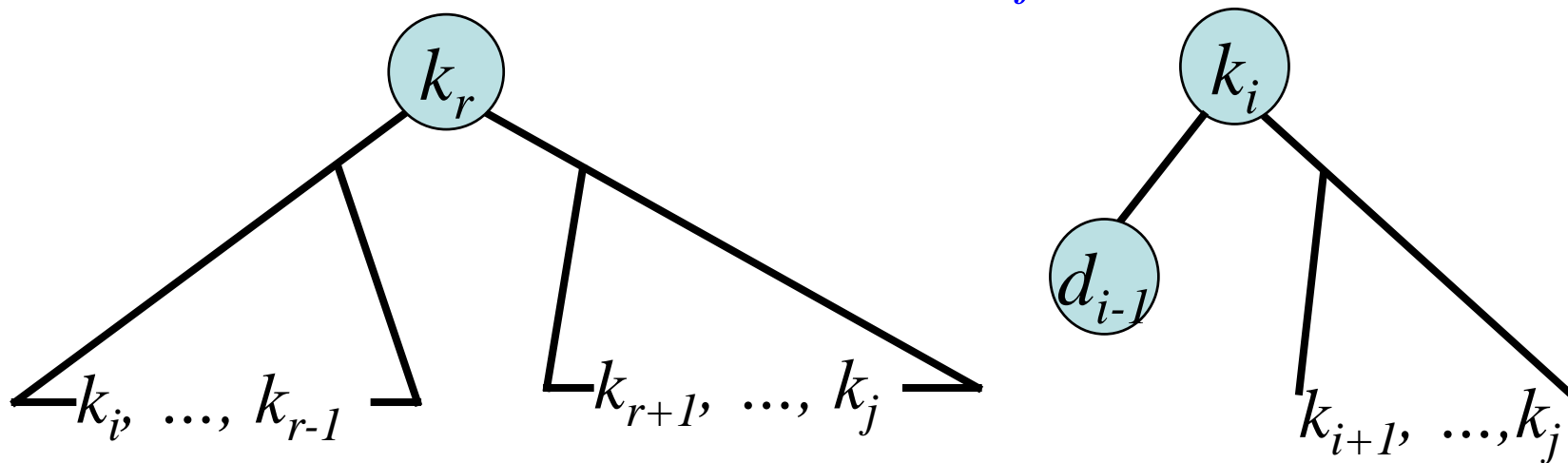


- 优化解的结构分析
- 建立优化解代价的递归方程
- 递归地划分子问题
- 自底向上计算优化解的代价  
记录优化解的构造信息
- 构造优化解



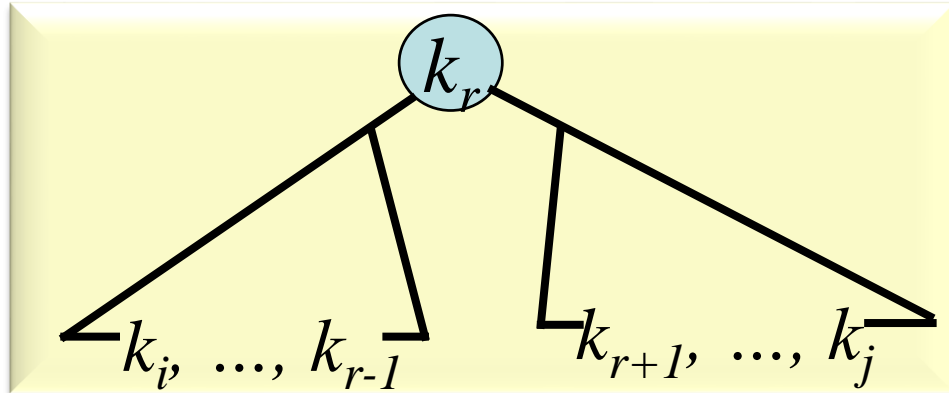
# 建立优化解的搜索代价递归方程

- 令  $E(i, j)$  为  $\{k_i, \dots, k_j\}$  的优化解  $T_{ij}$  的期望搜索代价
  - 当  $j=i-1$  时,  $T_{ij}$  中只有叶结点  $d_{i-1}$ ,  $E(i, i-1)=q_{i-1}$
  - 当  $j \geq i$  时, 选择一个  $k_r \in \{k_i, \dots, k_j\}$ :



当把左右优化子树放进  $T_{ij}$  时, 每个结点的深度增加 1

$$E(i, j) = P_r + E(i, r-1) + W(i, r-1) + E(r+1, j) + W(r+1, j)$$



- 计算  $W(i, r-1)$  和  $W(r+1, j)$

由  $E(LT+1) = \sum_{l=i}^{r-1} (DEP_{\text{左}}(k_l) + 2)p_l + \sum_{l=i-1}^{r-1} (DEP_{\text{左}}(d_l) + 2)q_l$

$$E(LT) = \sum_{l=i}^{r-1} (DEP_{\text{左}}(k_l) + 1)p_l + \sum_{l=i-1}^{r-1} (DEP_{\text{左}}(d_l) + 1)q_l$$

知  $W(i, r-1) = E(LT+1) - E(LT) = \sum_{l=i}^{r-1} p_l + \sum_{l=i-1}^{r-1} q_l$

同理,  $W(r+1, j) = \sum_{l=r+1}^j p_l + \sum_{l=r}^j q_l$

令  $W(i, j) = W(i, r-1) + W(r+1, j) + p_r = \sum_{l=i}^j p_l + \sum_{l=i-1}^j q_l = W(i, j-1) + p_j + q_j$

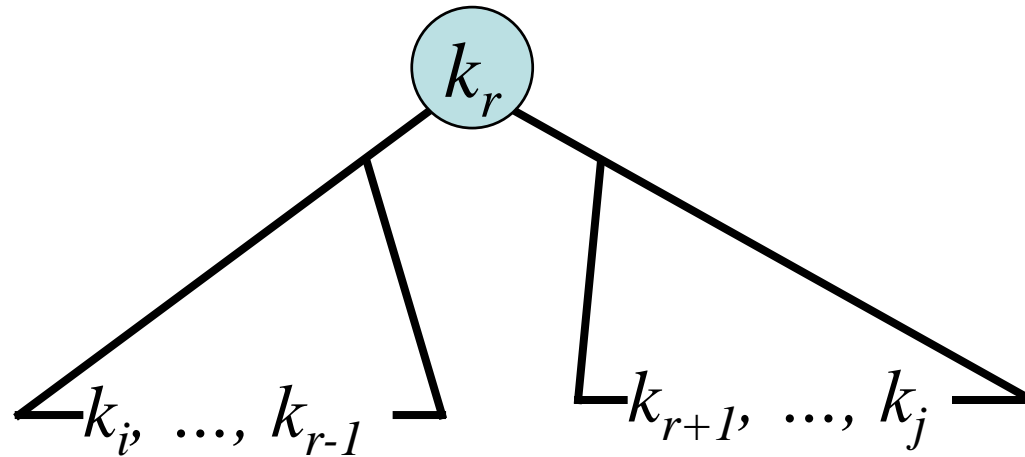
由  $W(i, r-1) = \sum_{l=i}^{r-1} p_l + \sum_{l=i-1}^{r-1} q_l$ ,  $W(i, i-1) = q_{i-1}$

$$W(i, i-1) = q_{i-1}$$

$$W(i, j) = W(i, r-1) + W(r+1, j) + p_r = W(i, j-1) + p_j + q_j$$

$$E(i, i-1) = q_{i-1}$$

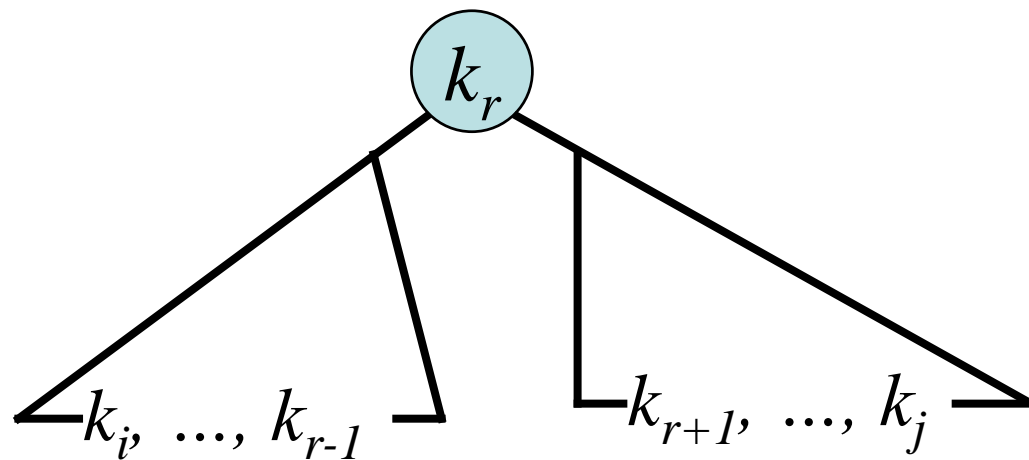
$$E(i, j) = P_r + E(i, r-1) + W(i, r-1) + E(r+1, j) + W(r+1, j)$$



$$E(i, j) = E(i, r-1) + E(r+1, j) + W(i, j)$$



总之



$$W(i, i-1) = q_{i-1},$$

$$W(i, j) = W(i, j-1) + p_j + q_j$$

$$E(i, i-1) = q_{i-1}$$

$$E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\} \quad \text{If } j \geq i$$



- 优化解的结构分析
- 建立优化解代价的递归方程
- 递归地划分子问题
- 自底向上计算优化解的代价  
记录优化解的构造信息
- 构造优化解



# 递归划分子问题

$$E(i, j) = q_{i-1} \quad \text{If } j = i-1$$

$$E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\} \quad \text{If } j \geq i$$

$$E(i, i-1)$$

$$E(i, i)$$

...

$$E(i, j-1)$$

$$E(i, j)$$

$$E(i+1, j)$$

$$E(i+2, j)$$

...

$$E(j+1, j)$$





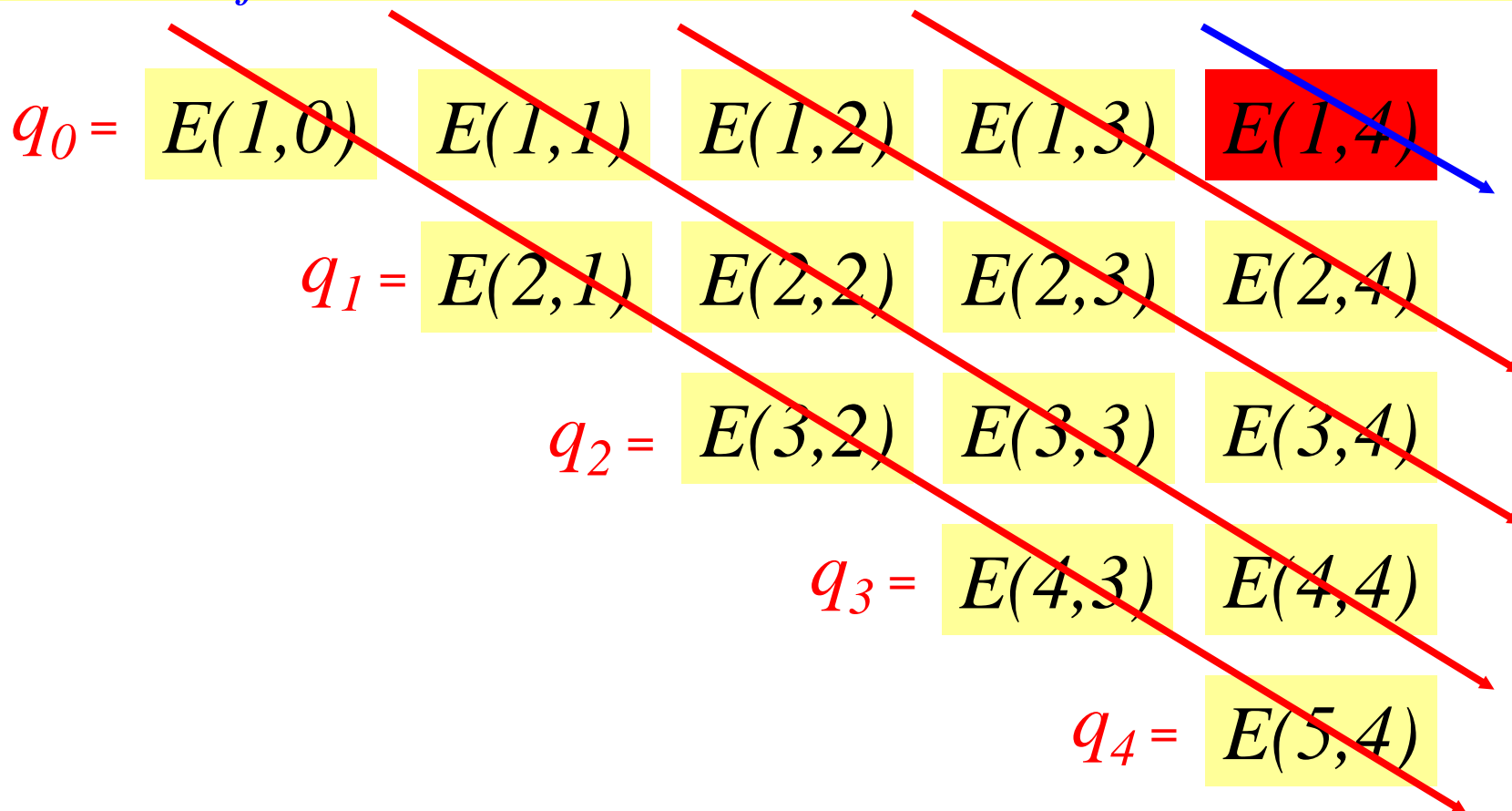
- 优化解的结构分析
- 建立优化解代价的递归方程
- 递归地划分子问题
- 自底向上计算优化解的代价  
记录优化解的构造信息
- 构造优化解



# 自下而上计算优化解的代价

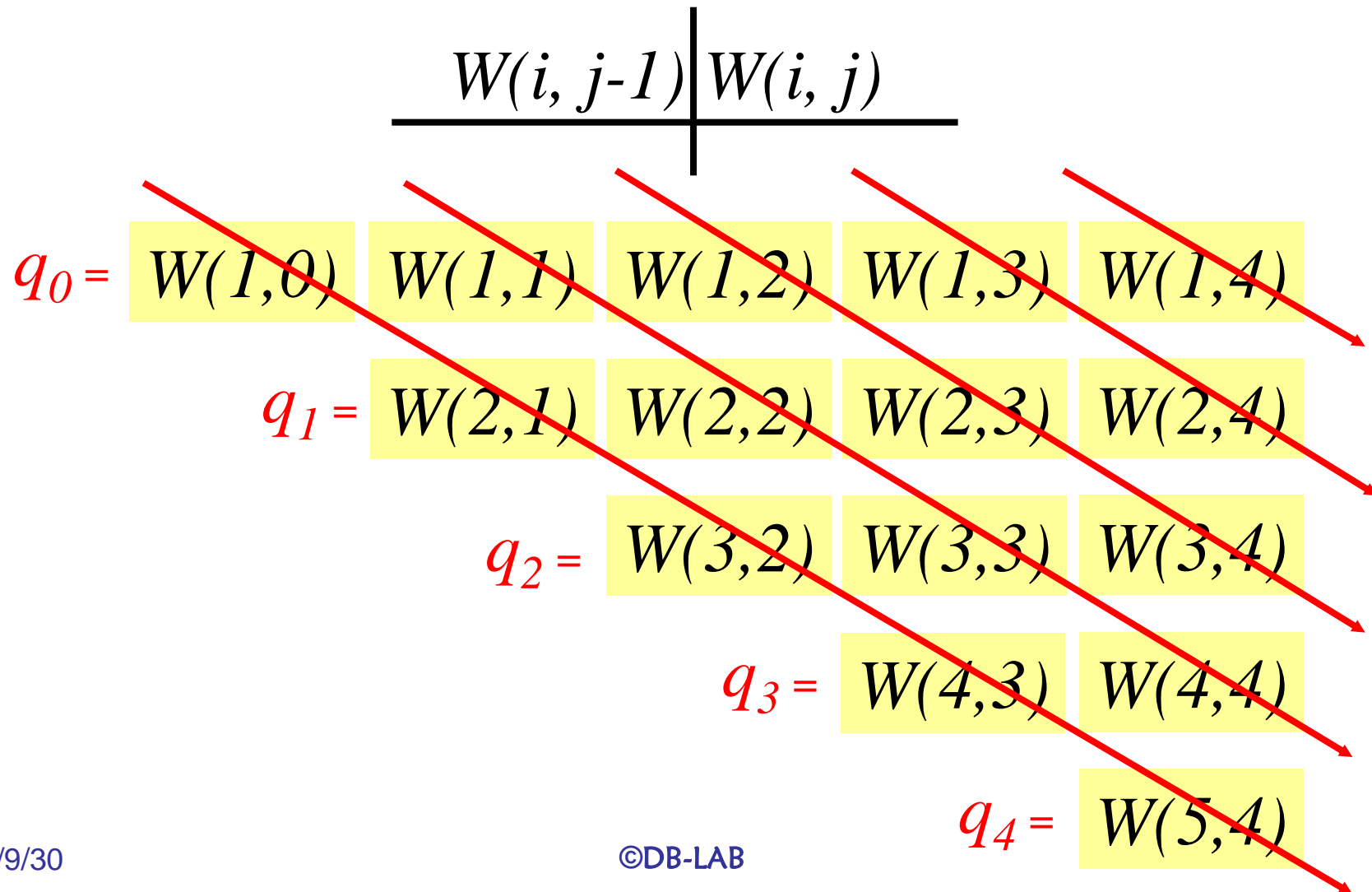
$$E(i, j) = q_{i-1} \quad \text{If } j = i - 1$$

$$E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\} \quad \text{If } j \geq i$$





- $W(i, i-1) = q_{i-1}, \quad W(i, j) = W(i, j-1) + p_j + q_j$





## • 算法

### • 数据结构

- $E[1:n+1; 0:n]$ : 存储优化解搜索代价
- $W[1:n+1; 0:n]$ : 存储代价增量
- $Root[1:n; 1:n]$ :  $Root(i, j)$ 记录子问题  $\{k_i, \dots, k_j\}$  优化解的根



$$E(i, j) = q_{i-1} \quad \text{If } j = i - 1$$

$$E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\} \quad \text{If } j \geq i$$

$$W(i, i-1) = q_{i-1}, \quad W(i, j) = W(i, j-1) + p_j + q_j$$

Optimal-BST( $p, q, n$ )

For  $i=1$  To  $n+1$  Do

$$E(i, i-1) = q_{i-1}; \quad W(i, i-1) = q_{i-1};$$

For  $l=1$  To  $n$  Do

For  $i=1$  To  $n-l+1$  Do

$$j = i + l - 1;$$

$$E(i, j) = \infty;$$

$$W(i, j) = W(i, j-1) + p_j + q_j;$$

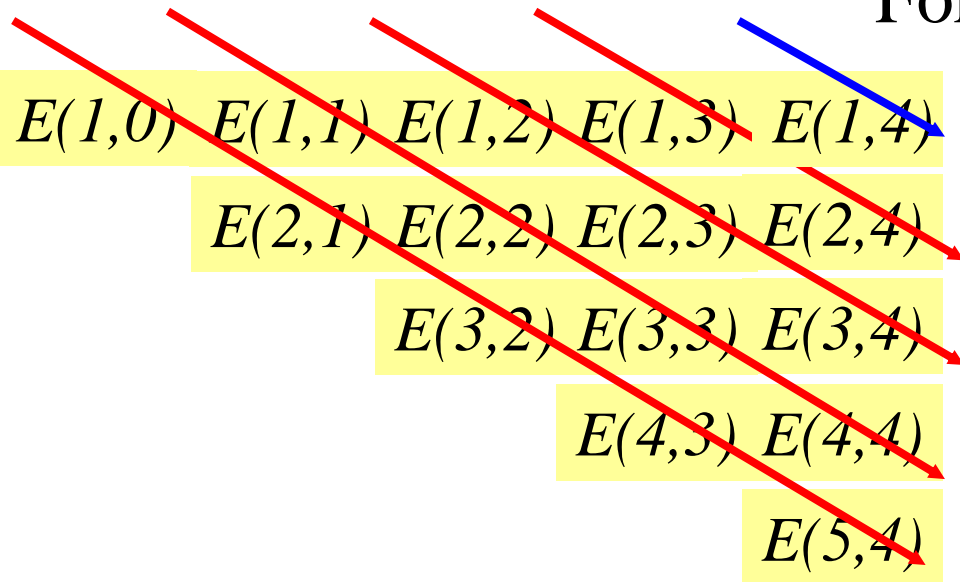
For  $r=i$  To  $j$  Do

$$t = E(i, r-1) + E(r+1, j) + W(i, j);$$

$$\text{If } t < E(i, j)$$

$$\text{Then } E(i, j) = t; \quad \text{Root}(i, j) = r;$$

Return  $E$  and  $\text{Root}$





- 时间复杂性
  - $(l, i, r)$  三层循环，每层循环至多  $n$  步
  - 时间复杂性为  $O(n^3)$
- 空间复杂性
  - 二个  $(n+1) \times (n+1)$  数组，一个  $n \times n$  数组
  - $O(n^2)$



- 优化解的结构分析
- 建立优化解代价的递归方程
- 递归地划分子问题
- 自底向上计算优化解的代价
- 记录优化解的构造信息
- 构造优化解



# 随堂测试：完成优化解的构造算法

Optimal-BST( $p, q, n$ )

For  $i=1$  To  $n+1$  Do

$E(i, i-1) = q_{i-1}; W(i, i-1) = q_{i-1};$

For  $l=1$  To  $n$  Do

For  $i=1$  To  $n-l+1$  Do

$j=i+l-1;$

$E(i, j)=\infty;$

$W(i, j)=W(i, j-1)+p_j+q_j;$

For  $r=i$  To  $j$  Do

$t=E(i, r-1)+E(r+1, j)+W(i, j);$

If  $t < E(i, j)$

Then  $E(i, j)=t; \text{Root}(i, j)=r;$

Return  $E$  and  $\text{Root}$





## 4.6 The Minimum Edit Distance

- 问题定义
- 问题求解
  - 优化解的结构分析
  - 建立优化代价的递归方程
  - 递归地划分子问题
  - 自底向上计算优化解的代价  
记录优化解的构造信息
  - 构造优化解



- 最小编辑距离

输入：两个字符串 $x[1..m]$ 和 $y[1..n]$

输出：将 $x[1..m]$ 转换为 $y[1..n]$ 所需要的最少操作数.

操作：插入一个符号，或者  
删除一个符号，或者  
替换一个符号

例如： $x=\text{"snowy"}$ ,  $y=\text{"sunny"}$

S	—	N	O	W	Y
S	U	N	N	—	Y

Cost: 3

—	S	N	O	W	—	Y
S	U	N	—	—	N	Y

Cost: 5



- 寻找优化解拆分成子问题解的方式

$ED[i, j]$ : 字符串  $x[1:i]$  和  $y[1:j]$  的编辑距离

$$ED[i, j] = \begin{cases} ? \\ \dots \\ ? \end{cases}$$

**\*不能遗漏拆分方式\***



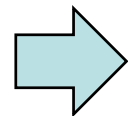
- 观察将SNOWY转变为SUNNY的编辑操作

编辑操作1: 插入U (S和N之间)

编辑操作2: 将O修改为N

编辑操作3: 删除W

— → U



O → N

W → —

只包含修改的编辑序列  
，引入空字符“—”

上述编辑序列的示例图

S	—	N	O	W	Y
	↓		↓	↓	
S	U	N	N	—	Y

同一字符发生多次编辑操作，  
则需要多层示例图，才能表示  
编辑过程。



- 最小编辑距离对应编辑序列的性质

**不重叠性：**任一字符只执行一次编辑操作（插入、修改、删除之一）

**顺序无关性：**将编辑序列中的操作顺序任意重排列，不改变编辑结果

**对称性：**将 $s$ 编辑为 $t$ 的最短序列与将 $t$ 编辑为 $s$ 的最短序列等长



- 满足不重叠性的编辑序列

**提问：**具有不重叠性的编辑序列一定生成最小编辑距离吗？

**例子：**删除 $x$ 中所有字符，在插入 $y$ 中所有字符，编辑距离不一定最小。

满足不重叠性的编辑序列 ( $x \rightarrow y$ ) 如下图所示

$$\begin{array}{ccccccc} x_1 & x_2 & \dots & x_i & \dots & x_K \\ \downarrow & \downarrow & & \downarrow & & \downarrow \\ y_1 & y_2 & \dots & y_i & \dots & y_K \end{array}$$

$x$ 是 $x_1 x_2 \dots x_i \dots x_K$ 的子序列，后者除 $x$ 以外的元素都是“—”

$y$ 是 $y_1 y_2 \dots y_i \dots y_K$ 的子序列，后者除 $y$ 以外的元素都是“—”

$x_i$ 和 $y_i$ 不能同时等于“—”， $1 \leq i \leq K$



将 $x[1:m]$ 转换成 $y[1:n]$

- 满足不重叠性的编辑序列

$$\begin{array}{ccccccc} x_1 & x_2 & \dots & x_i & \dots & x_K \\ \downarrow & \downarrow & & \downarrow & & \downarrow \\ y_1 & y_2 & \dots & y_i & \dots & y_K \end{array}$$

$x$ 是 $x_1 x_2 \dots x_i \dots x_K$ 的子序列，后者除 $x$ 以外的元素都是“—”

$y$ 是 $y_1 y_2 \dots y_i \dots y_K$ 的子序列，后者除 $y$ 以外的元素都是“—”

$x_i$ 和 $y_i$ 不能同时等于“—”， $1 \leq i \leq K$

$x_i = y_i$ ：字符 $x_i$ 原样保留

$x_i = \text{“—”}$ ：插入字符 $y_i$

$y_i = \text{“—”}$ ：删除字符 $x_i$

否则：字符 $x_i$ 修改为字符 $y_i$

字符匹配：满足不重叠性的编辑序列图示中，字符 $x_i$ 和字符 $y_i$ 匹配， $1 \leq i \leq K$

起点和终点不相同的“↓”的数量即为相应的编辑距离



- 寻找优化解拆分成子问题解的方式

$ED[i, j]$ : 字符串  $x[1:i]$  和  $y[1:j]$  的编辑距离

$$ED[m, n] = \begin{cases} ? \\ \dots \\ ? \end{cases} \quad \begin{array}{l} \nearrow \\ \rightarrow \\ \searrow \end{array} \quad \begin{array}{l} \text{根据 } x[m] \text{ 的匹配} \\ \text{位置分情况讨论} \end{array}$$

**\*不能遗漏拆分方式\***

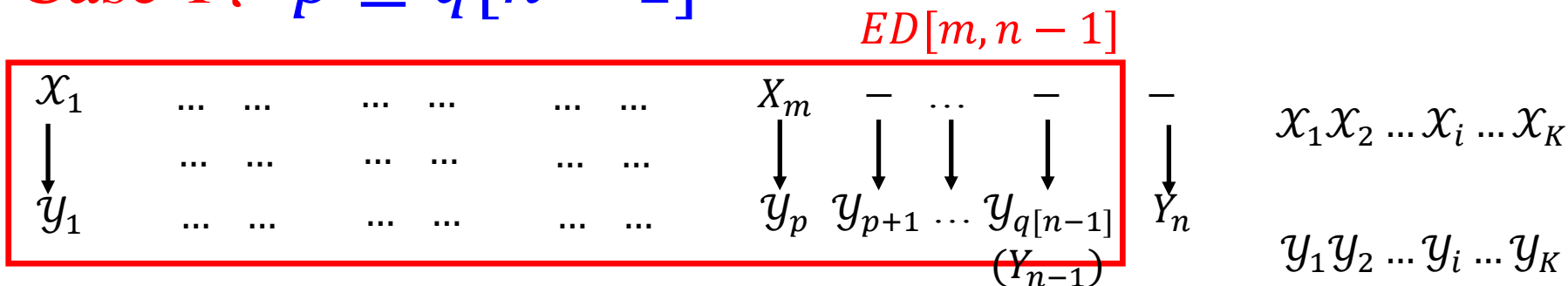




将 $x[1:m]$ 转换成 $y[1:n]$

- 在 $y_1 y_2 \dots y_i \dots y_K$  中,  $x[m]$  匹配 $y_p$ ,  $y[j]$ 对应 $y_{q[j]}$ , 数组 $q[j]$ 存储 $y[1:n]$ 在 $Y$ 中的位置

Case 1:  $p \leq q[n-1]$



将 $x[1:m]$ 转换成 $y[1:n]$   
的编辑序列示例图

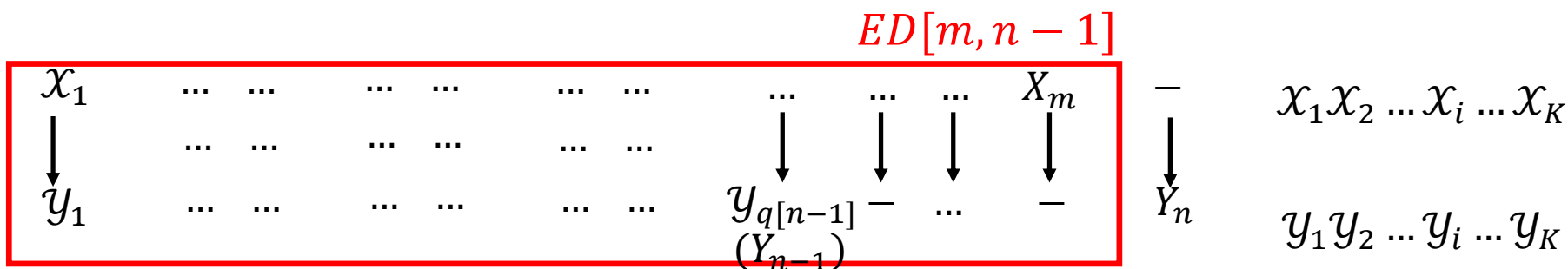
$$ED[m, n] = ED[m, n-1] + 1$$



将 $x[1:m]$ 转换成 $y[1:n]$

- 在 $y_1 y_2 \dots y_i \dots y_K$  中,  $x[m]$  匹配 $y_p$ ,  $y[j]$ 对应 $y_{q[j]}$ , 数组 $q[j]$ 存储 $y[1:n]$ 在 $Y$ 中的位置

Case 2:  $q[n-1] < p < q[n]$



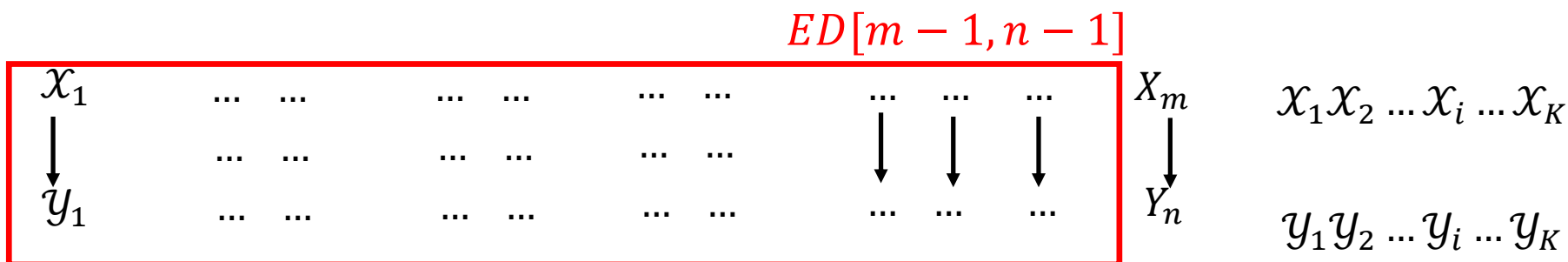
将 $x[1:m]$ 转换成 $y[1:n]$   
的编辑序列示例图

$$ED[m, n] = ED[m, n-1] + 1$$

将 $x[1:m]$ 转换成 $y[1:n]$ 

## 优化解结构分析

- 在 $y_1 y_2 \dots y_i \dots y_K$  中,  $x[m]$  匹配 $y_p$ ,  $y[j]$ 对应 $y_{q[j]}$ , 数组 $q[j]$ 存储 $y[1:n]$ 在 $Y$ 中的位置

Case 3:  $p = q[n]$ 将 $x[1:m]$ 转换成 $y[1:n]$ 

的编辑序列示例图

$$ED[m, n] = ED[m-1, n-1] + \text{diff}(X_m, Y_n)$$

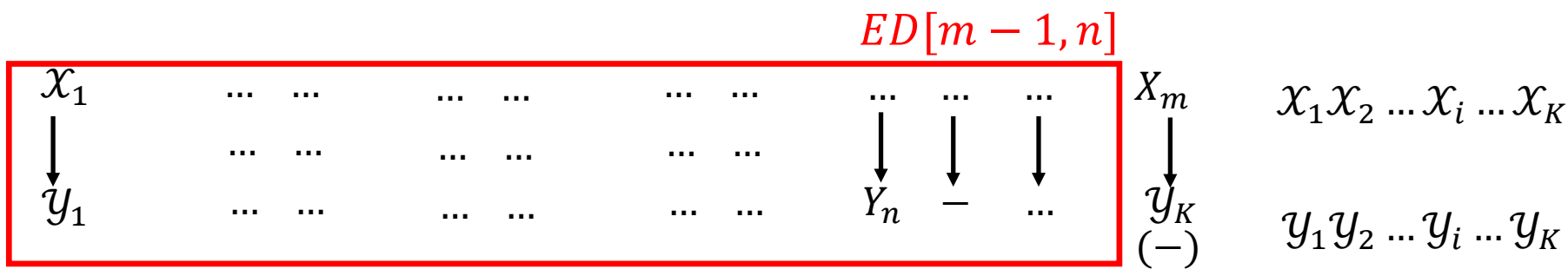
$$\text{diff}(x, y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases}$$



将 $x[1:m]$ 转换成 $y[1:n]$

- 在 $y_1 y_2 \dots y_i \dots y_K$  中,  $x[m]$  匹配 $y_p$ ,  $y[j]$ 对应 $y_{q[j]}$ , 数组 $q[j]$ 存储 $y[1:n]$ 在 $\mathcal{Y}$ 中的位置

Case 4:  $p > q[n]$  (隐含 $p = K$ )



将 $x[1:m]$ 转换成 $y[1:n]$   
的编辑序列示例图

$$ED[m, n] = ED[m-1, n] + 1$$



- 总结上述情况

如果  $X[m]$  匹配  $Y[n]$  之前的字符:

$$ED[m, n] = ED[m, n - 1] + 1$$

如果  $X[m]$  匹配  $Y[n]$  :

$$ED[m, n] = ED[m - 1, n - 1] + \text{diff}(X[m], Y[n])$$

如果  $X[m]$  匹配  $Y[n]$  之后的字符:

$$ED[m, n] = ED[m - 1, n] + 1$$

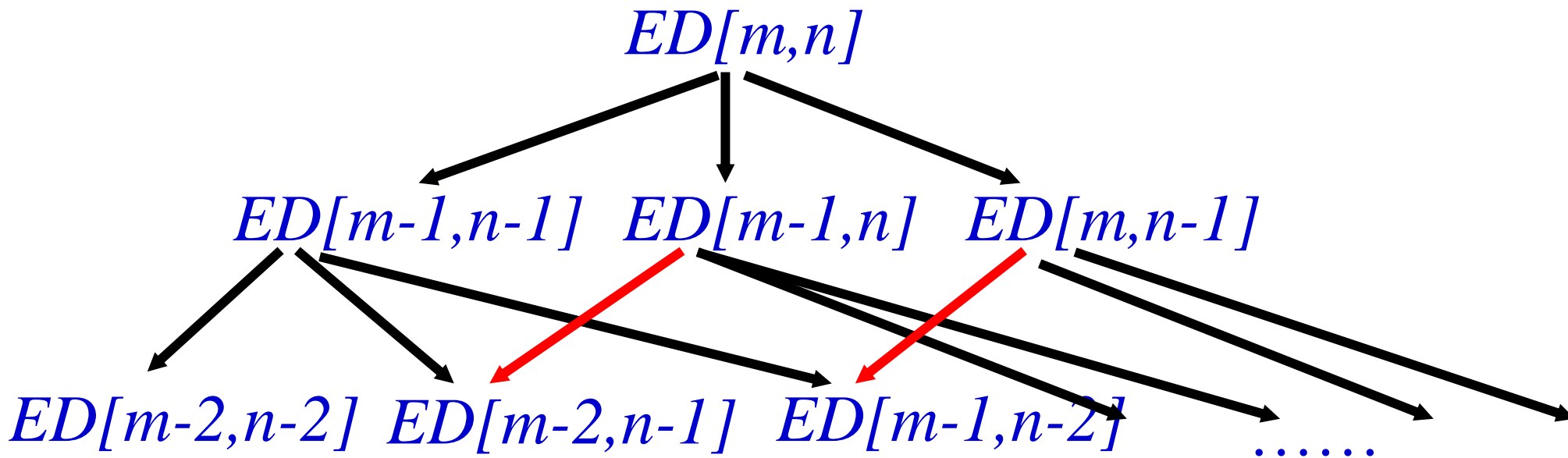


$$ED[m, n] = \min \begin{cases} ED[m-1, n] + 1 \\ ED[m-1, n-1] + \text{diff}(X[m], Y[n]) \\ ED[m, n-1] + 1 \end{cases}$$

证明：

情况一：  $ED[m, n] = ED[m-1, n] + 1$  并取得优化解，则  $ED[m-1, n]$  必为  $x[1:m-1]$  和  $y[1:n]$  的最小编辑距离。否则，将存在一组编辑操作将  $x[1:m-1]$  在  $k$  步转换为  $y[1:n]$ ，且  $k < ED[m-1, n]$ 。

情况二、情况三的证明过程与情况一类似



最小编辑距离问题具有子问题重叠性



• 计算  $X[1:i]$  和  $Y[1:j]$  的最小编辑距离  $ED[i, j]$

$$ED[i, 0] = i \quad 1 \leq i \leq m$$

$$ED[0, j] = j \quad 1 \leq j \leq n$$

$$ED[i, j] = \min \begin{cases} ED[i-1, j] + 1 \\ ED[i-1, j-1] + \text{diff}(X[i], Y[j]) \\ ED[i, j-1] + 1 \end{cases}$$





# 递归划分与自底向上求解

$$ED[i, j] = \min \begin{cases} ED[i-1, j] + 1 \\ ED[i-1, j-1] + \text{diff}(X[i], Y[j]) \\ ED[i, j-1] + 1 \end{cases}$$

$$ED[i, 0] = i \quad 1 \leq i \leq m$$

$$ED[0, j] = j \quad 1 \leq j \leq n$$

	$ED[i-1, j-1]$	$ED[i-1, j]$	
	$ED[i, j-1]$	$ED[i, j]$	



$$ED[i, j] = \min \begin{cases} ED[i-1, j] + 1 \\ ED[i-1, j-1] + \text{diff}(X[i], Y[j]) \\ ED[i, j-1] + 1 \end{cases}$$

		$y_j$	<i>R</i>	<i>E</i>	<i>N</i>	<i>A</i>	<i>T</i>	<i>O</i>
$i=0$	$x_i$							
	<i>R</i>							
	<i>O</i>							
	<i>N</i>							
	<i>A</i>							
	<i>L</i>							
	<i>D</i>							
	<i>O</i>							
		$j=0$						

记录优化解信息

$$ED[i,j] = \min \begin{cases} ED[i-1,j] + 1 \\ ED[i-1,j-1] + diff(X[i],Y[j]) \\ ED[i,j-1] + 1 \end{cases}$$

		$y_j$	<i>R</i>	<i>E</i>	<i>N</i>	<i>A</i>	<i>T</i>	<i>O</i>
$i=0$	$x_i$	0	1	2	3	4	5	6
	<i>R</i>	1	↖ 0	← 1	← 2	← 3	← 4	← 5
	<i>O</i>	2	↑ 1	↖ 1	← 2	← 3	← 4	↖ 4
	<i>N</i>	3	↑ 2	↑ 2	↖ 1	← 2	← 3	← 4
	<i>A</i>	4	↑ 3	↑ 3	↑ 2	↖ 1	← 2	← 3
	<i>L</i>	5	↑ 4	↑ 4	↑ 3	↑ 2	↖ 2	↖ 3
	<i>D</i>	6	↑ 5	↑ 5	↑ 4	↑ 3	↖ 3	↖ 3
	<i>O</i>	7	↑ 6	↑ 6	↑ 5	↑ 4	↖ 4	↖ 4
		$j=0$						

记录优化解信息

$$ED[i,j] = \min \begin{cases} ED[i-1,j] + 1 \\ ED[i-1,j-1] + diff(X[i],Y[j]) \\ ED[i,j-1] + 1 \end{cases}$$

		$y_j$	<i>R</i>	<i>E</i>	<i>N</i>	<i>A</i>	<i>T</i>	<i>O</i>
$i=0$	$x_i$	0	1	2	3	4	5	6
	<i>R</i>	1	↖ 0	← 1	← 2	← 3	← 4	← 5
	<i>O</i>	2	↑ 1	↖ 1	← 2	← 3	← 4	↘ 4
	<i>N</i>	3	↑ 2	↑ 2	↖ 1	← 2	← 3	← 4
	<i>A</i>	4	↑ 3	↑ 3	↑ 2	↖ 1	← 2	← 3
	<i>L</i>	5	↑ 4	↑ 4	↑ 3	↑ 2	↘ 2	↘ 3
	<i>D</i>	6	↑ 5	↑ 5	↑ 4	↑ 3	↖ 3	↘ 3
	<i>O</i>	7	↑ 6	↑ 6	↑ 5	↑ 4	↘ 4	↖ 4
		$j=0$						

```

MinimumED(X, Y)
M ← length(X); n ← length(Y);
For i ← 0 To m Do
    E[i, 0] ← i;
For j ← 1 To n Do
    E[0, j] ← j;
For i ← 1 To m Do
    For j ← 1 To n Do
        If  $x_i = y_j$ 
            Then  $E[i, j] ← E[i-1, j-1]$ ;
        Else
             $E[i, j] ← E[i-1, j-1] + 1$ ;
            B[i, j] ← “↖”;
        If  $E[i, j] > E[i-1, j] + 1$ 
            Then  $E[i, j] = E[i-1, j] + 1$ ;
            B[i, j] ← “↑”;
        If  $E[i, j] > E[i, j-1] + 1$ 
            Then  $E[i, j] = E[i, j-1] + 1$ ;
            B[i, j] ← “←”;
Return E and B.

```

# 算法和算法复杂性

## • 时间复杂性

- $(i, j)$  两层层循环，每层循环至多  $m$  和  $n$  步
- 时间复杂性为  $O(mn)$

## • 空间复杂性

- 一个  $(m+1) \times (n+1)$  数组，一个  $n \times n$  数组
- $O(mn)$
- $B$  可以省去

# 构造优化编辑序列

$$ED[i,j] = \min \begin{cases} ED[i-1,j] + 1 \\ ED[i-1,j-1] + diff(X[i],Y[j]) \\ ED[i,j-1] + 1 \end{cases}$$

		$y_j$	<i>R</i>	<i>E</i>	<i>N</i>	<i>A</i>	<i>T</i>	<i>O</i>
$i=0$	$x_i$	0	1	2	3	4	5	6
<i>R</i>	1	↘ 0	← 1	← 2	← 3	← 4	← 5	
<i>O</i>	2	↑ 1	↘ 1	← 2	← 3	← 4	↘ 4	
<i>N</i>	3	↑ 2	↑ 2	↘ 1	← 2	← 3	← 4	
<i>A</i>	4	↑ 3	↑ 3	↑ 2	↘ 1	← 2	← 3	
<i>L</i>	5	↑ 4	↑ 4	↑ 3	↑ 2	↘ 2	↘ 3	
<i>D</i>	6	↑ 5	↑ 5	↑ 4	↑ 3	↘ 3	↘ 3	
<i>O</i>	7	↑ 6	↑ 6	↑ 5	↑ 4	↘ 4	↘ 4	
		$j=0$						

边界条件  $E[i, 0]$ : 删除  $x[1:i]$

边界条件  $E[0, j]$ : 在  $x[1]$  前插入  $y[1:j]$

↘ 将  $x[1:i-1]$  修改为  $y[1:j-1]$  后，将  $x[i]$  按需修改为  $y[j]$

← 将  $x[1:i]$  修改为  $y[1:j-1]$  后，在末尾插入  $y[j]$

↑ 将  $x[1:i-1]$  修改为  $y[1:j]$  后，删除末尾符号（原  $x[i]$ ）



**作业1:** 输入为CRONALDO和RENATO, 填写矩阵 $E$ 和 $B$ ;

**作业2:** 完成优化的构造算法, 输入 $x$ 和 $y$ , 输出基本操作序列, 将 $x$ 转换为 $y$

基本操作:

(1) Insert(pos, a)

(2) Modify(pos, b)

(3) Delete(pos)

(伪代码、时间复杂性、空间复杂性)



- 最小编辑距离判别问题

输入：两个字符串 $x[1..m]$ 和 $y[1..n]$ ，整数 $t$

输出：*True*，如果 $x$ 和 $y$ 的最小编辑距离不大于 $t$   
*False*，如果 $x$ 和 $y$ 的最小编辑距离大于 $t$

输入：RONALDO, RENATO, 4, 输出：*True*

输入：RONALDO, RENATO, 2, 输出：*False*





- 最小编辑距离判别问题

Step 1. 计算输入字符串的最小编辑距离

Step 2. 与阈值比较

- 改进算法的两点启发

在E中计算大量不必要的元素!

$$ED[i, j] = \min \begin{cases} ED[i-1, j] + 1 \\ ED[i-1, j-1] + \text{diff}(X[i], Y[j]) \\ ED[i, j-1] + 1 \end{cases}$$
$$ED[i, 0] = i \quad 1 \leq i \leq m$$
$$ED[0, j] = j \quad 1 \leq j \leq n$$

字符串x和y的最小编辑距离不小于二者长度之差的绝对值



HITWH  
SE

$x = \text{“RONALDO”}$ ,  $y = \text{“RENATO”}$ ,  
编辑距离阈值  $t = 2$

		<i>R</i>	<i>E</i>	<i>N</i>	<i>A</i>	<i>T</i>	<i>O</i>
$i=0$	$x_i$						
	<i>R</i>						
	<i>O</i>						
	<i>N</i>						
	<i>A</i>						
	<i>L</i>						
	<i>D</i>						
	<i>O</i>						



# 算法的复杂性

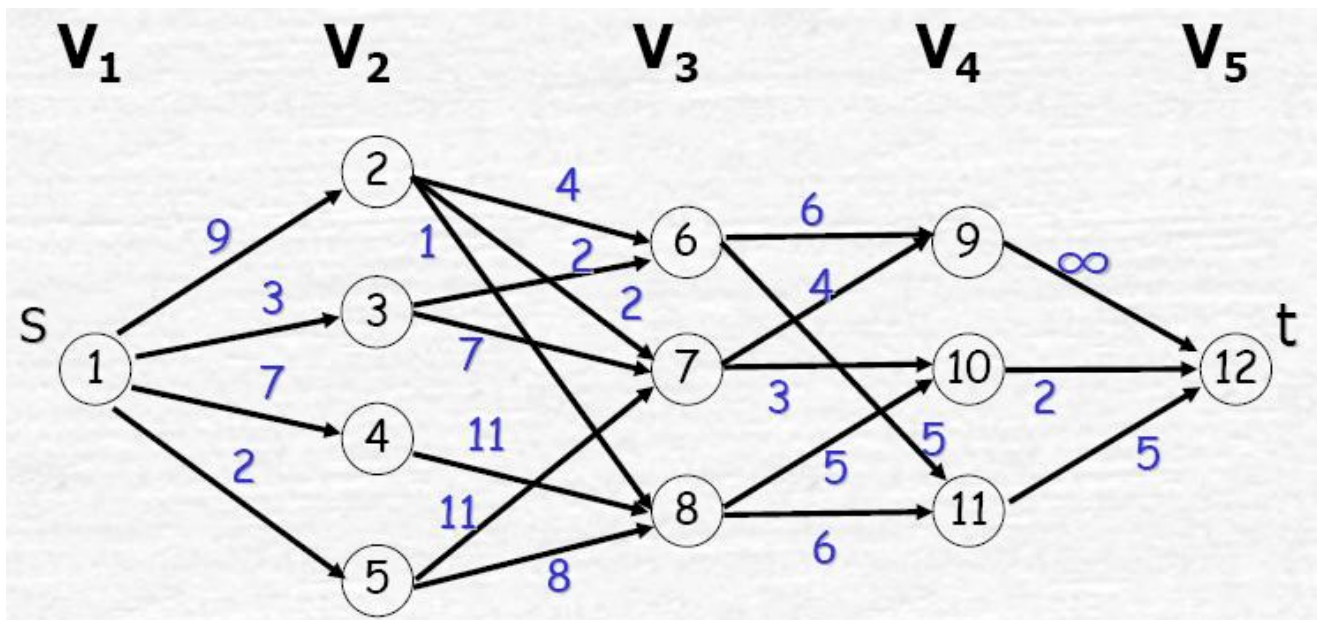
$x = \text{"RONALDO"} ,$   
 $y = \text{"RENATO"} ,$   
编辑距离阈值  $t = 2$

		R E N A T O						
$i=0$	$x_i$	0	1	2				
	R	1	0	1	2			
	O	2	1	1	2	3		
	N		2	2	1	2	3	
	A			3	2	1	2	3
	L				3	2	2	3
	D					3	3	3
	O							

- 时间复杂性
  - 每行、列最多计算  $2t + 1$
  - 时间复杂性为  $O(\min\{m, n\}t)$
- 空间复杂性
  - $O(\min\{m, n\}t)$
  - 可优化为  $O(t)$



# 多段图规划



求从s到t的最短路径.



- 优化解的结构分析

设  $s, \dots, v_{ij}, \dots, v_{ik}, \dots, t$  是一条由  $s$  到  $t$  的最短路径，  
则  $v_{ij}, \dots, v_{ik}, \dots, t$  也是由  $v_{ij}$  到  $t$  的最短路径



- 问题定义

输入：集合  $S = \{n \text{ 个正整数}\}$ ，正整数  $P$

输出：True，若存在子集合  $S'$ ，使得  $P = S'$  中所有数之和

False，否则



- 优化解结构分析

$M(i, j) : = True$ , 当且仅当在  $S$  的前  $i$  个数中, 存在一个子集合, 使其数据之和为  $j$ .

$M(n, P)$  即为原始问题

$$M(i, j) = M(i-1, j-S[i]) \vee M(i-1, j)$$

$$M(i, 0) = True$$

$$M(0, j) = False \quad \text{for } j > 0$$



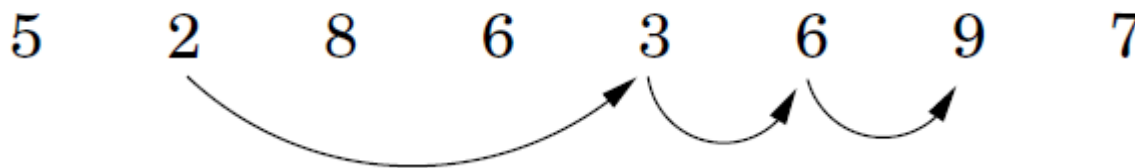
- 最长增长子序列问题

输入：由 $n$ 个数组成的一个序列 $S: a_1, a_2, \dots, a_n$

输出：子序列 $S'=b_1, b_2, \dots, b_k$ ，满足：

(1)  $b_{i1} \leq b_{i2} \leq \dots \leq b_{ik}$ ,

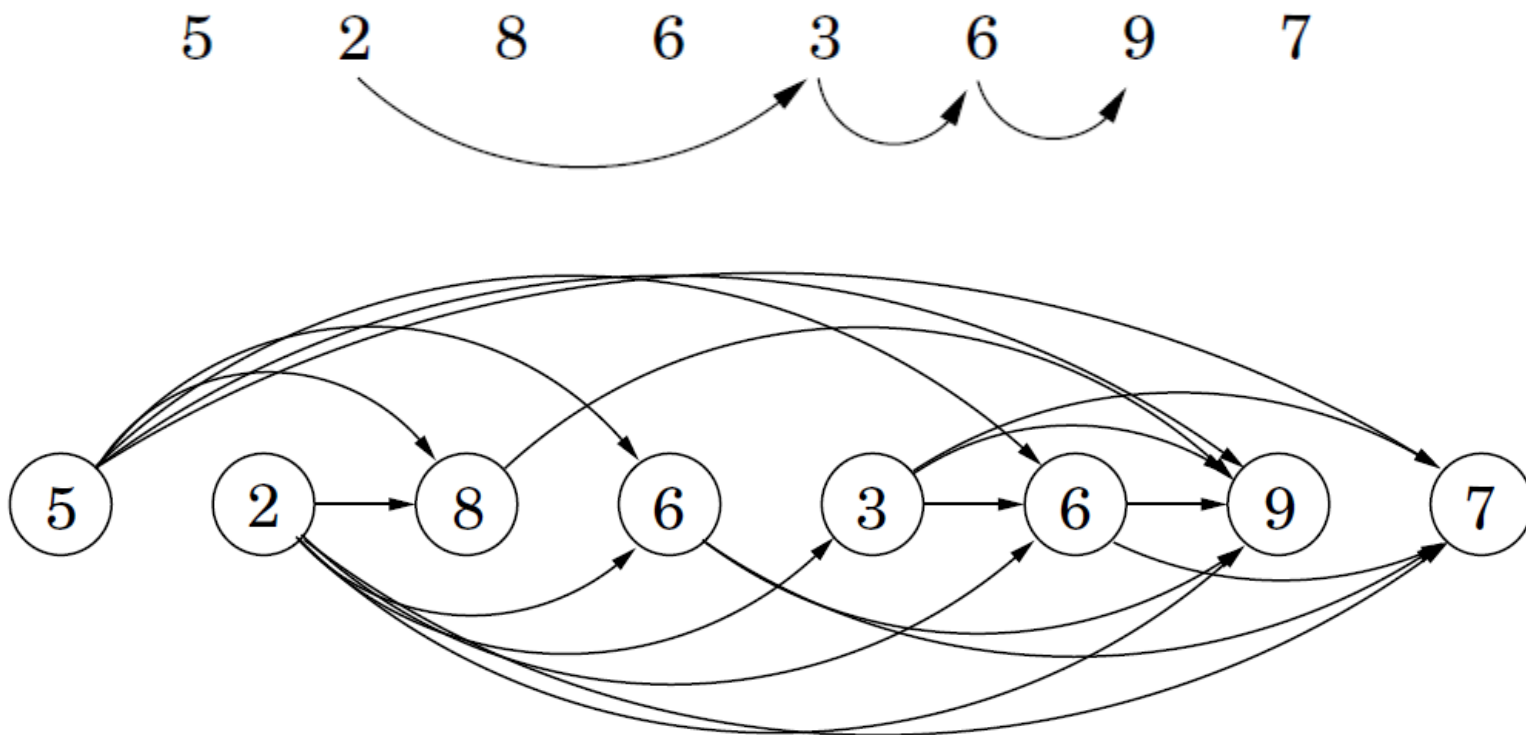
(2)  $|S'|$ 最大







## • 最长增长子序列问题





# 总结

- 原始问题可以划分成一系列子问题，子问题之间不是相互独立的
- 不同子问题的数目常常只有多项式量级
- 优化子结构



- 优化解的结构分析
- 建立优化解代价的递归方程
- 递归地划分子问题
- 自底向上计算优化解的代价  
记录优化解的构造信息
- 构造优化解



```
Node{  
  range;    // range of keys  
  label;    //  $k_i$  or  $d_j$   
  leftChild;  
  rightChild; }  
}
```

## 作业：使用递归

```
Extend(Root, n)  
(i, j)  $\leftarrow$  n.range;  
If i > j  
    n.label  $\leftarrow d_j$ ;  
    Return;  
n.Label  $\leftarrow k_{Root[i,j]}$ ;  
L  $\leftarrow$  Root[i, j];  
Initiate node lc;  
lc.range  $\leftarrow$  (i, L-1);  
n.leftChild  $\leftarrow$  lc;  
Initiate node rc;  
rc.range  $\leftarrow$  (L+1, j);  
n.rightChild  $\leftarrow$  rc;  
Extend(Root, lc);  
Extend(Root, rc);
```

```
Construct-BST(Root)  
Initiate an empty Tree T;  
#  $\leftarrow$  Root.size;  
Initiate node r;  
r.range  $\leftarrow$  (1, #);  
T.Root  $\leftarrow$  r;  
Extend(Root, r);  
Return T;
```



```
Node{  
  range;    // range of keys  
  label;    //  $k_i$  or  $d_j$   
  leftChild;  
  rightChild; }  
}
```

## 作业：使用栈

### Construct-BST(Root)

```
# ← Root.size;  
Initiate node r;  
r.range ← (1, #);  
Stack S ←  $\emptyset$ ;  
S.push(r);  
While S  $\neq \emptyset$   
  n = S.pop();  
  (i, j) ← n.range;  
  If i > j  
    | n.label ←  $d_j$ ;  
    | continue;  
  L ← Root[i, j];  
  n.Label ←  $k_L$ ;  
  Initiate node lc;
```

```
lc.range ← (i, L-1);  
n.leftChild ← lc;  
Initiate node rc;  
rc.range ← (L+1, j);  
n.rightChild ← rc;  
S.push(lc);  
S.push(rc);  
Initiate an empty Tree T;  
T.root ← r;  
Return T;
```



# 作业：打印T

```
Print-BST(Root, i, j)
If  $i > j$ 
    Print  $d_j$ ;
    Return;
 $L \leftarrow \text{Root}[i, j]$ ;
Print ( ;
Print-BST(Root, i, L-1);
Print ) ;
Print  $k_{\text{Root}[i,j]}$ ;
Print ( ;
Print-BST(Root, L+1, j);
Print ) ;
```

打印BST的语句：  
**Print-BST**(Root, 1, Root.size);