第四章

Dynamic Programming Algorithms

提要

Introduction to Algorithms

第15章

15.2, 15.3, 15.4, 15.5

4.1 Elements of Dynamic Programming

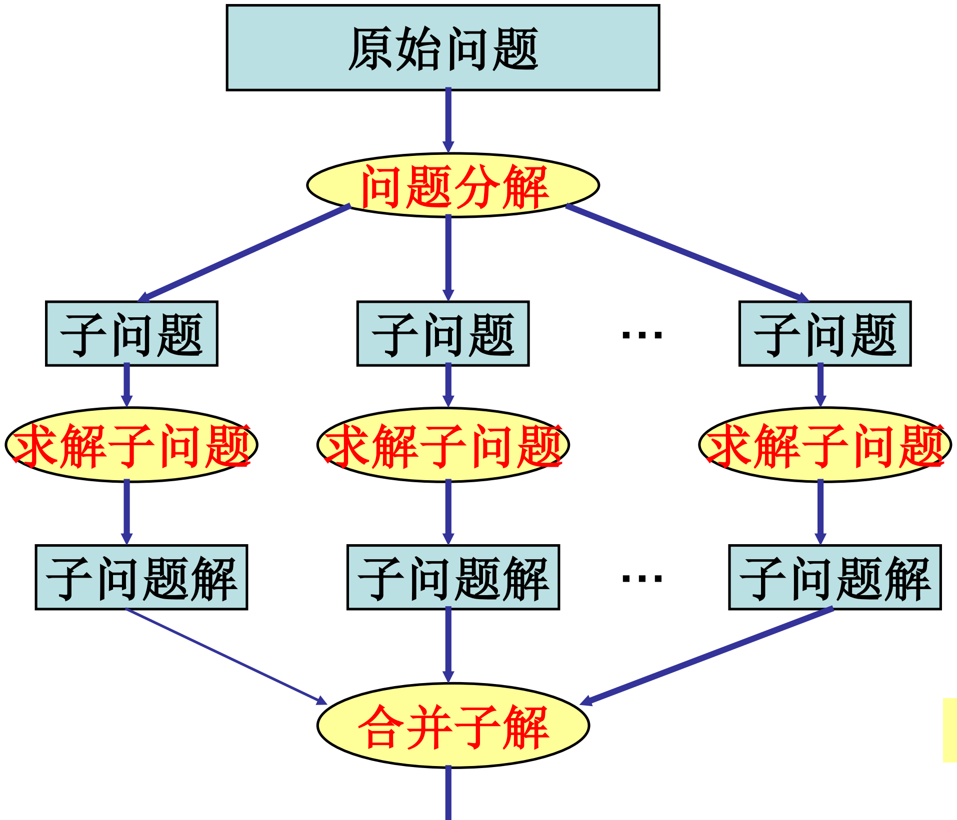
Why?

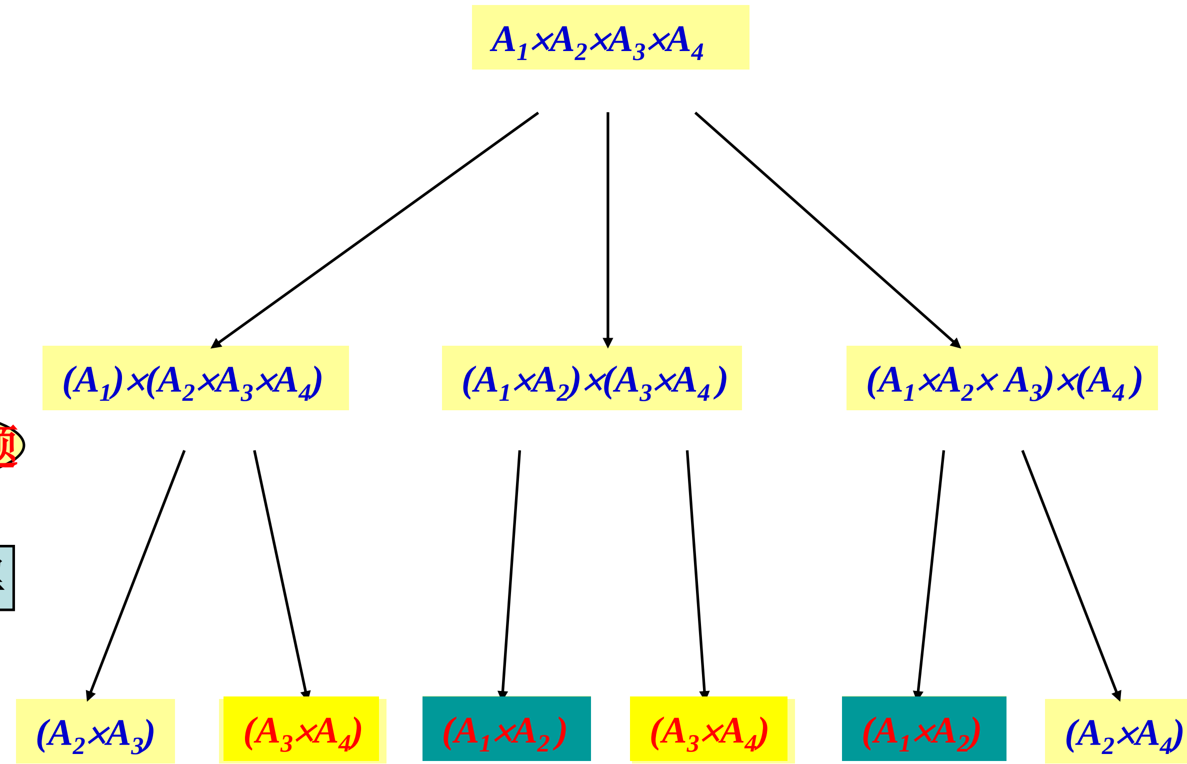
What?

How?

Why?

- 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

Why?

- 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

What?

- 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

What?

- 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

How?

- 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

How?

- 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

4.2 Matrix-chain Multiplication

问题的定义

- 输入：$<A_1, A_2, \ldots, A_n>$，A_i是$p_{i-1}\times p_i$矩阵
- 输出：计算$A_1\times A_2\times\ldots\times A_n$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$O(pqr)$

Motivation

- 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=10×100矩阵, A_2=100×5矩阵, A_3=5×50矩阵

    T((A1×A2)×A3)=10×100×5+10×5×50 =7500

    T(A1×(A2×A3))=100×5×50+10×100×50 =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

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

$p(k)$ $\quad$ $p(n-k)$

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

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

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

$(A_1) \times (A_2 \times \ldots \times A_n)$

$(A_1 \times A_2) \times (A_3 \times \ldots \times A_n)$

....

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

...

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

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

$$A_1 \times A_2 \times A_3 \times \ldots \times A_n = \begin{cases} (A_1) \times (A_2 \times \ldots \times A_n) \\ (A_1 \times A_2) \times (A_3 \times \ldots \times A_n) \\ \ldots. \\ (A_1 \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_n) \\ \ldots \\ (A_1 \times \ldots \times A_{n-1}) \times (A_n) \end{cases}$$

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

$$A_1 \times A_2 \times A_3 \times \ldots \times A_n = \begin{cases} (A_1) \times (A_2 \times \ldots \times A_n) \\ (A_1 \times A_2) \times (A_3 \times \ldots \times A_n) \\ \ldots. \\ (A_1 \times \ldots \times A_k)\times(A_{k+1} \times \ldots \times A_n) \\ \ldots \\ (A_1 \times \ldots \times A_{n-1}) \times (A_n) \end{cases}$$

$$cost_{1\sim n} = Min \begin{cases} cost_{1\sim 1} + cost_{2\sim n} + p_0 p_1 p_n \\ cost_{1\sim 2} + cost_{3\sim n} + p_0 p_2 p_n \\ \ldots. \\ cost_{1\sim k} + cost_{k+1\sim n} + p_0 p_k p_n \\ \ldots \\ cost_{1\sim n-1} + cost_{n\sim n} + p_0 p_{n-1} p_n \end{cases}$$

其中A_i是$p_{i-1} \times p_i$矩阵

分析优化解的结构

$$A_1 \times A_2 \times A_3 \times \ldots \times A_n = \begin{cases} (A_1) \times (A_2 \times \ldots \times A_n) \\ (A_1 \times A_2) \times (A_3 \times \ldots \times A_n) \\ \ldots. \\ (A_1 \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_n) \\ \ldots \\ (A_1 \times \ldots \times A_{n-1}) \times (A_n) \end{cases}$$

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$A_1 \times \ldots \times A_k$的方案必须是代价最小的方案

计算$A_{k+1} \times \ldots \times A_n$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$A_i \times \ldots \times A_j$相乘

分析优化解的结构

- 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.

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

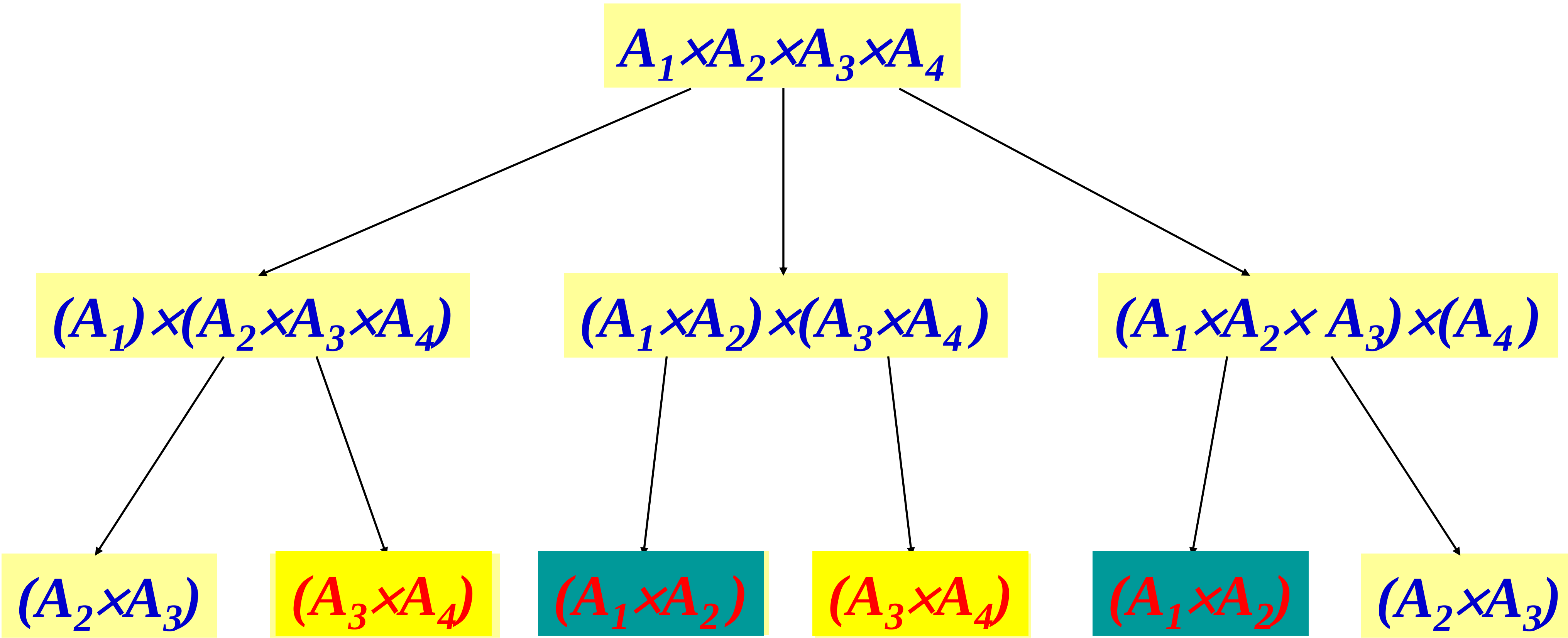
$$((A_1 \times \ldots A_i) \times (A_{i+1} \times \ldots \times A_k)) \times ((A_{k+1} \times \ldots A_j) \times (A_{j+1} \times \ldots \times A_n))$$

  - 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：

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

  - 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$A_iA_{i+1}\ldots A_j$的最优乘法方案

$$
A_i \times A_{i+1} \times \ldots \times A_j = \begin{cases} (A_i) \times (A_{i+1} \times \ldots \times A_j) \\ (A_i \times A_{i+1}) \times (A_{i+2} \times \ldots \times A_j) \\ \ldots. \\ (A_i \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_j) \\ \ldots \\ (A_i \times \ldots \times A_{j-1}) \times (A_j) \end{cases}
$$

HITWH
SE

递归地定义最优解的代价

$$A_i \times A_{i+1} \times \ldots \times A_j = \begin{cases} (A_i) \times (A_{i+1} \times \ldots \times A_j) \\ (A_i \times A_{i+1}) \times (A_{i+2} \times \ldots \times A_j) \\ \ldots. \\ (A_i \times \ldots \times A_k)\times(A_{k+1} \times \ldots \times A_j) \\ \ldots \\ (A_i \times \ldots \times A_{j-1})\times(A_j) \end{cases}$$

$$cost_{i\sim j} = Min_{i \le k \le j-1} \begin{cases} cost_{i\sim i} + cost_{i+1\sim j} + p_{i-1}p_i p_j \\ cost_{i\sim i+1} + cost_{i+2\sim j} + p_{i-1}p_{i+1} p_j \\ \ldots. \\ Cost_{i\sim k} + cost_{k+1\sim j} + p_{i-1}p_k p_j \\ \ldots \\ cost_{i\sim j-1} + cost_{j\sim j} + p_{i-1}p_{j-1}p_j \end{cases}$$

其中A_r是$p_{r-1} \times p_r$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

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

考虑到所有的k，优化解的代价方程为

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

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

递归地划分子问题

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

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

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

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

$m[i, j] = 0 \qquad\qquad\qquad\qquad if \quad i=j$

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

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]

算法描述

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

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

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

Matrix-Chain-Order(*p*)

n=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*.

获取构造最优解的信息

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

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

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]

Matrix-Chain-Order(p)

n=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 $m[i,j]=q$; $s[i,j]=k$;

Return m and s.

$S[i,j]=k$记录$A_i A_{i+1} \ldots A_j$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

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

- **Matrix-Chain-Order(*p*)**
- ***n*=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]=∞;***
-         **FOR *k←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：$O(n^3)$

构造最优解

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

IF $j=i$

THEN Print "A"$_i$;

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$A_i \ldots A_j$的最优划分处；

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

$S[S[i,j]+1, j]$记录$A_{s[i,j]+1} \ldots A_j$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

算法复杂性

- 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

问题的定义

- 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：$X = (x_1,x_2,\ldots,x_m)$，$Y = (y_1,y_2,\ldots y_n)$

输出：X与Y的最长公共子序列

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

优化子结构分析

- 第i前缀
  - 设$X=(x_1, x_2, \ldots, x_n)$是一个序列

    则$X_i=(x_1, \ldots, x_i)$是X的第i前缀

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

$$X = (x_1, x_2, \ldots, x_m)$$
$$Y = (y_1, y_2, \ldots y_n)$$

- 优化子结构的猜想

X和Y的LCS为$LCS_{XY}=(z_1, \ldots, z_k)$

If $x_m=y_n$ 则 $z_k=x_m=y_n$

$LCS_{XY}=LCS_{X_{m-1}Y_{n-1}}+ <x_m=y_n>$

If $x_m \neq y_n$,

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

- 优化子结构

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

(1) 如果$x_m=y_n$, 则$z_k=x_m=y_n$, $LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + <x_m=y_n>$,
$LCS_{X_{m-1}Y_{n-1}}$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). $X=<x_1, \ldots, x_{m-1}, x_m>$, $Y=<y_1, \ldots, y_{n-1}, x_m>$，则

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

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

$$LCS_{XY} = Z_{k-1} + <x_m=y_n>,$$

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

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

与LCS_{XY}是LCS矛盾。

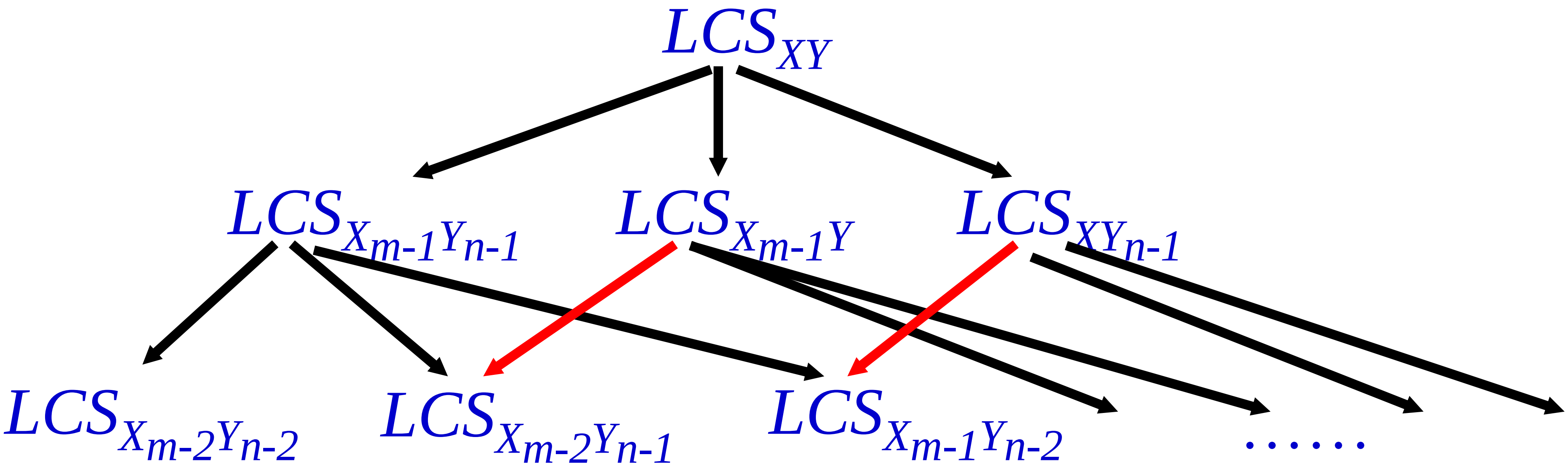
(2) $X=<x_1, \ldots, x_{m-1}, x_m>$, $Y=<y_1, \ldots, y_{n-1}, y_n>$,

$x_m \neq y_n$, $z_k \neq x_m$, 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

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

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

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

递归划分与自底向上求解

- 基本思想

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

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

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

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

自底向上计算优化解代价

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

- 递归划分子问题与自底向上求解过程

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]

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

		y_j	B	D	C	A	B	A
$i=0$	x_i	0	0	0	0	0	0	0
	A	0	0	0	0	1	1	1
	B	0	1	1	1	1	2	2
	C	0	1	1	2	2	2	2
	B	0	1	1	2	2	3	3
	D	0	1	2	2	2	3	3
	A	0	1	2	2	3	3	4
	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

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

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

LCS-length(X, Y)

m←length(X); n←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 $C[i,j] \leftarrow C[i-1,j-1]+1$;

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

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

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

        $B[i,j] \leftarrow$ "↑";

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

        $B[i,j] \leftarrow$ "←";

Return C and B.

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

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

	y_j	B	D	C	A	B	A
x_i	0	0	0	0	0	0	0
A	0	↑0	↑0	↑0	↖1	←1	↖1
B	0	↖1	←1	←1	↑1	↖2	←2
C	0	↑1	↑1	↖2	←2	↑2	↑2
B	0	↖1	↑1	↑2	↑2	↖3	←3
D	0	↑1	↖2	↑2	↑2	↑3	↑3
A	0	↑1	↑2	↑2	↖3	↑3	↖4
B	0	↖1	↑2	↑2	↑3	↖4	↑4

.AB

构造优化解

- 基本思想
  - 从$B[m, n]$开始按指针搜索
  - 若$B[i, j]$="$\boxed{\nwarrow}$"，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	B	D	C	A	B	A
x_i	0	0	0	0	0	0	0
A	0	↑0	↑0	↑0	↖1	←1	↖1
B	0	↖1	←1	←1	↑1	↖2	←2
C	0	↑1	↑1	↖2	←2	↑2	↑2
B	0	↖1	↑1	↑2	↑2	↖3	←3
D	0	↑1	↖2	↑2	↑2	↑3	↑3
A	0	↑1	↑2	↑2	↖3	↑3	↖4
B	0	↖1	↑2	↑2	↑3	↖4	↑4

Print-LCS(*B*, *X*, *i*, *j*)
If $i=0$ or $j=0$ **Then Return;**
If $B[i, j]$="$\boxed{\nwarrow}$"
Then Print-LCS(B, X, i-1, j-1); Print x_i;
Else
    If $B[i, j]$="↑"
    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=length(X)
m=length(Y)

```
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  xi = yj
        Then C[i,j]←C[i-1,j-1]+1;
               B[i,j]←"↖";
        Else If C[i-1,j]≥C[i,j-1]
               Then C[i,j]←C[i-1,j];
                      B[i,j]←"↑";
               Else C[i,j]←C[i,j-1];
                      B[i,j]←"←";
  Return C and B.
```

	y_j	B	D	C	A	B	A
x_i	0	0	0	0	0	0	0
A	0	↑0	↑0	↑0	↖1	←1	↖1
B	0	↖1	←1	←1	↑1	↖2	←2
C	0	↑1	↑1	↖2	←2	↑2	↑2
B	0	↖1	↑1	↑2	↑2	↖3	←3
D	0	↑1	↖2	↑2	↑2	↑3	↑3
A	0	↑1	↑2	↑2	↖3	↑3	↖4
B	0	↖1	↑2	↑2	↑3	↖4	↑4

算法复杂性

- 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：$(x_1, x_2, \ldots, x_n),\ x_i\in\{0, 1\}$，满足
  $\Sigma_{1\leq i\leq n} w_i x_i \leq C,\ \ \Sigma_{1\leq i\leq n} v_i x_i$ 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

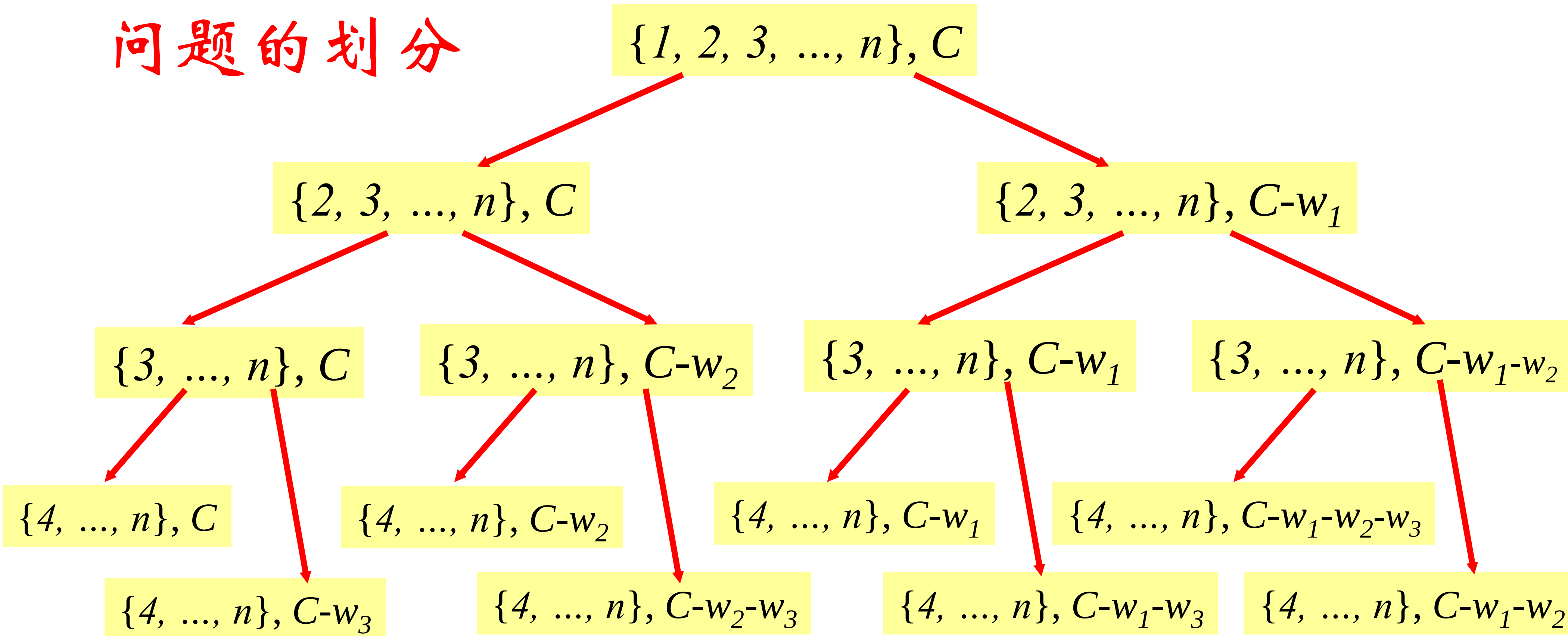
每个装取方案的计算代价为n

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

子问题重叠性

问题的划分



当w_i皆为1时，存在大量重叠子问题

问题优化子结构

$$P_2: \max \sum_{2\le i\le n} v_i x_i$$
$$\sum_{2\le i\le n} w_i x_i \le C - w_1 x_1$$
$$x_i \in \{0, 1\},\ 2\le i\le n$$

问题的划分

$\{1, 2, 3, \ldots, n\}, C$

$\{2, 3, \ldots, n\}, C$　　$\{2, 3, \ldots, n\}, C\text{-}w_1$

$\{3, \ldots, n\}, C$　　$\{3, \ldots, n\}, C\text{-}w_2$　　$\{3, \ldots, n\}, C\text{-}w_1$　　$\{3, \ldots, n\}, C\text{-}w_1\text{-}w_2$

$\{4, \ldots, n\}, C$　　$\{4, \ldots, n\}, C\text{-}w_3$　　$\{4, \ldots, n\}, C\text{-}w_2$　　$\{4, \ldots, n\}, C\text{-}w_2\text{-}w_3$　　$\{4, \ldots, n\}, C\text{-}w_1$　　$\{4, \ldots, n\}, C\text{-}w_1\text{-}w_3$　　$\{4, \ldots, n\}, C\text{-}w_1\text{-}w_2\text{-}w_3$　　$\{4, \ldots, n\}, C\text{-}w_1\text{-}w_2$

问题优化子结构

问题的划分

P_2: $max\ \Sigma_{2\le i\le n} v_i\, x_i$

$\Sigma_{2\le i\le n} w_i x_i \le C - w_1 x_1$

$x_i \in \{0, 1\},\ 2\le i\le n$

$\{1, 2, 3, \ldots, n\}, C$

$\{2, 3, \ldots, n\}, C$ $\{2, 3, \ldots, n\}, C\text{-}w_1$

$\{3, \ldots, n\}, C$ $\{3, \ldots, n\}, C\text{-}w_2$ $\{3, \ldots, n\}, C\text{-}w_1$ $\{3, \ldots, n\}, C\text{-}w_1\text{-}w_2$

$\{4, \ldots, n\}, C$ $\{4, \ldots, n\}, C\text{-}w_3$ $\{4, \ldots$ $C\text{-}w_1\text{-}w_2\text{-}w_3$ $\{4, \ldots, n\}, C\text{-}w_1\text{-}w_2$

P_i: $\{i, i+1, \ldots, n\}, C\text{-} \Sigma_{1\le k\le i-1} w_k x_k$

$max\ \Sigma_{i\le k\le n} v_k\, x_k$

$\Sigma_{i\le k\le n} w_k x_k \le C_i = C\text{-} \Sigma_{1\le k\le i-1} w_k x_k$

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

优化解结构的分析

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

$$\max \sum_{i+1\le k\le n} v_k x_k$$
$$\sum_{i+1\le k\le n} w_k x_k \le C_i - w_i y_i$$
$$x_k \in \{0, 1\},\ i+1\le k\le n$$

证明： 如果$S_{i+1}=(y_{i+1}, \ldots, y_n)$不是子问题$P_{i+1}$的优化解，则存在$S'_{i+1}=(z_{i+1}, \ldots, z_n)$是$P_{i+1}$的更优解。$S'_i=(y_i, z_{i+1}, \ldots, z_n)$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$[(i, i+1, \ldots, n), j]$的优化解$(x_i, x_{i+1}, \ldots, x_n)$的代价，$\mathbf{m(i, j)= \Sigma_{i \le k \le n} v_k x_k}$

- 形式地

  问题 $\max \Sigma_{i \le k \le n} v_k x_k$

  $\Sigma_{i \le k \le n} w_k x_k \le j$

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

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

- 递归方程：

$[\{i, i+1, \ldots, n\}, j]$

$0 \le j < w_i,$

$$m(i, j) = m(i+1, j)$$

$j \ge w_i,$

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

$[\{n\}, j]$

$0 \le j < w_n,$

$$m(n, j) = 0$$

$j \ge w_n,$

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

- 递归方程：

总结：

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

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

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

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

$j \ge w_n,$

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

- 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

$$m(i, j) = m(i+1, j),\ 0 \le j < w_i$$
$$m(i, j) = \max\{m(i+1, j),\ m(i+1, j-w_i)+v_i\},\ \ j \ge w_i$$
$$m(n, j) = 0,\quad 0 \le j < w_n$$
$$m(n, j) = v_n,\quad j \ge w_n$$

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

令n=4，w_i= 整数

$m(1,C)$

$m(2,C-w_1)$ … $m(2,C)$

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

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

$$m(i,j) = m(i+1,j),\ 0 \le j < w_i$$
$$m(i,j) = \max\{m(i+1,j),\ m(i+1,j-w_i)+v_i\},\ \ j \ge w_i$$
$$m(n,j) = 0,\ \ 0 \le j < w_n$$
$$m(n,j) = v_n,\ \ j \ge w_n$$

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

令n=4，w_i=整数

						$m(1,C)$
$m(2,0)$	$\cdots$	$m(2,w_2-1)$	$m(2,w_2)$	$\cdots$	$m(2,C-1)$	$m(2,C)$
$m(3,0)$	$\cdots$	$m(3,w_3-1)$	$m(3,w_3)$	$\cdots$	$m(3,C-1)$	$m(3,C)$
$m(4,0)$	$\cdots$	$m(4,w_4-1)$	$m(4,w_4)$	$\cdots$	$m(4,C-1)$	$m(4,C)$

- 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

HITWH
SE

$$m(i, j) = m(i+1, j),\ 0 \le j < w_i$$
$$m(i, j) = \max\{m(i+1, j),\ m(i+1, j-w_i)+v_i\},\ \ j \ge w_i$$
$$m(n, j) = 0,\quad 0 \le j < w_n$$
$$m(n, j) = v_n,\quad j \ge w_n$$

令 w_i = 整数, $n=4$

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

$m(1,C)$

$m(2, 0)$	…	$m(2,w_2-1)$	$m(2, w_2)$	…	$m(2,C-1)$	$m(2,C)$
$m(3, 0)$	…	$m(3,w_3-1)$	$m(3, w_3)$	…	$m(3,C-1)$	$m(3,C)$
$m(4, 0)$	…	$m(4,w_4-1)$	$m(4, w_4)$	…	$m(4,C-1)$	$m(4,C)$

- **算法**

(设 w_i-1<C)

$$m(i, j) = m(i+1, j),\ 0 \le j < w_i$$
$$m(i, j) = \max\{m(i+1, j),\ m(i+1, j-w_i)+v_i\},\ j \ge w_i$$
$$m(n, j) = 0,\quad 0 \le j < w_n$$
$$m(n, j) = v_n,\quad j \ge w_n$$

						$m(1,C)$
$m(2, 0)$	…	$m(2,w_2-1)$	$m(2, w_2)$	…	$m(2,C-1)$	$m(2,C)$
$m(3, 0)$	…	$m(3,w_3-1)$	$m(3, w_3)$	…	$m(3,C-1)$	$m(3,C)$
$m(4, 0)$	…	$m(4,w_4-1)$	$m(4, w_4)$	…	$m(4,C-1)$	$m(4,C)$

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

```
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**

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.

$m(1,C)$

$m(2,C-w_1)$ … $m(2,C)$

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

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

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

1. If $m(1, C) = m(2, C)$
   Then $x_1 = 0$; Else $x_1 = 1$;
2. If $x_1=0$, 由$m(2, C)$继续构造x_2;
3. If $x_1=1$, 由$m(2, C-w_1)$继续构造x_2;

……

算法复杂性

- 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
    $m[i, j] = m[i+1, j]$;
  For $j=w_i$ To C Do
    … $_i]+v_i\}$;
If …
  … $C-w_1]+v_1\}$;

这是一个伪多项式算法！

当$C=2^n$时：

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

当w_i不限定为正整数时：

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

… $_2$;

3\. If $x_1=1$，由$m(2, C-w_1)$继续构造x_2;

……

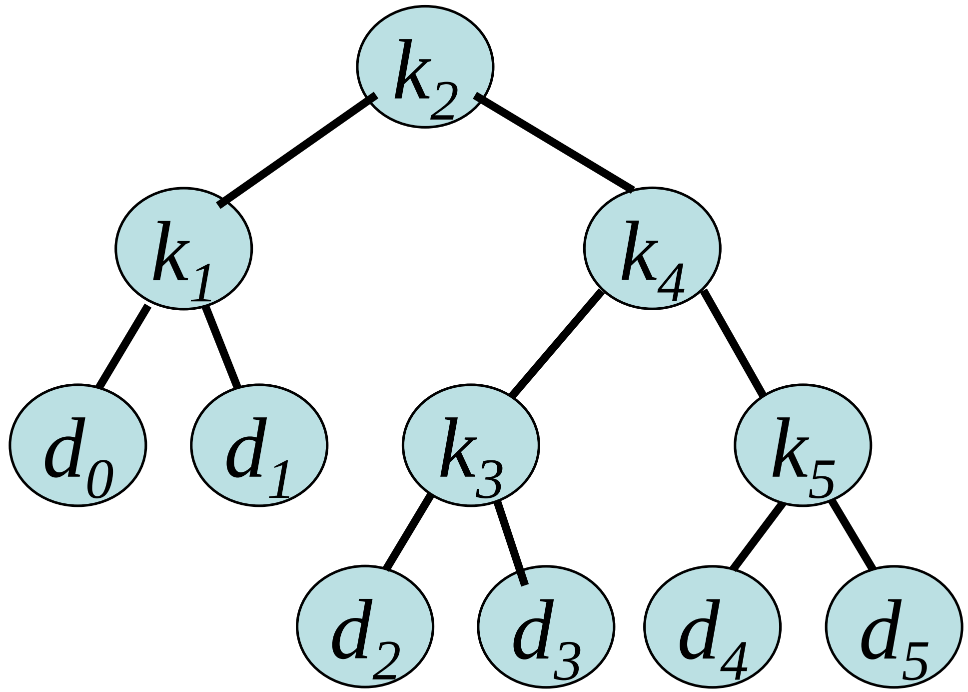
部分背包问题？

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

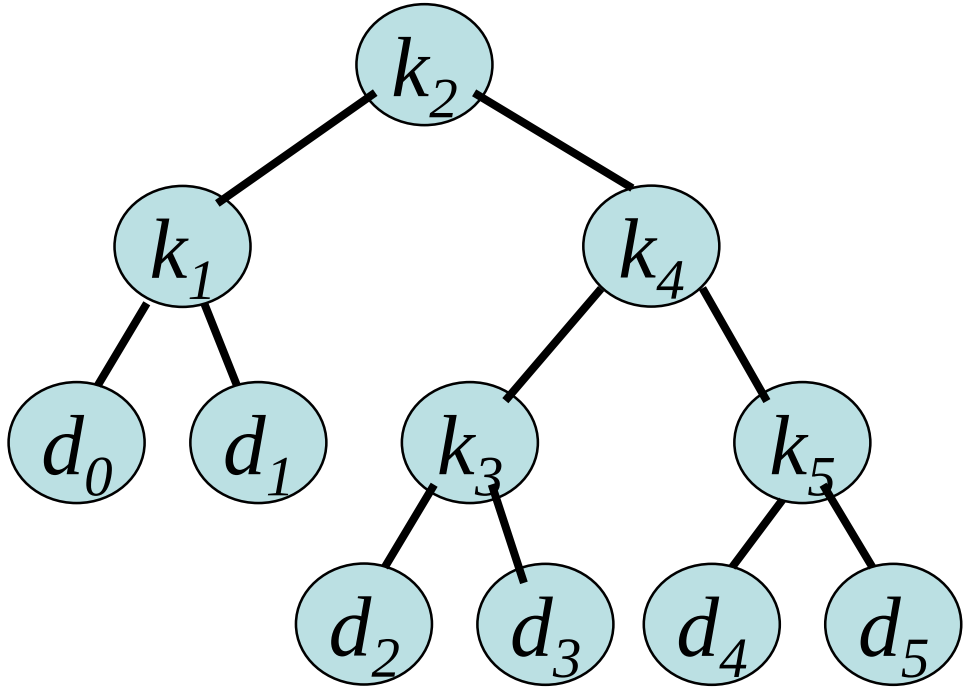
问题的定义

- 二叉搜索树T
  - 结点
    - $K=\{k_1, k_2, \ldots, k_n\}$
    - $D=\{d_0, d_1, \ldots, d_n\}$
    - 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)



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

- 问题的定义

输入：$K=\{k_1, k_2, \ldots, k_n\}, k_1< k_2< \ldots < k_n,$
$P=\{p_1, p_2, \ldots, p_n\}, p_i$为搜索$k_i$的概率
$Q=\{q_0, q_1, \ldots, q_n\}, q_i$为搜索值$d_i$的概率

输出：构造K的二叉搜索树T，最小化

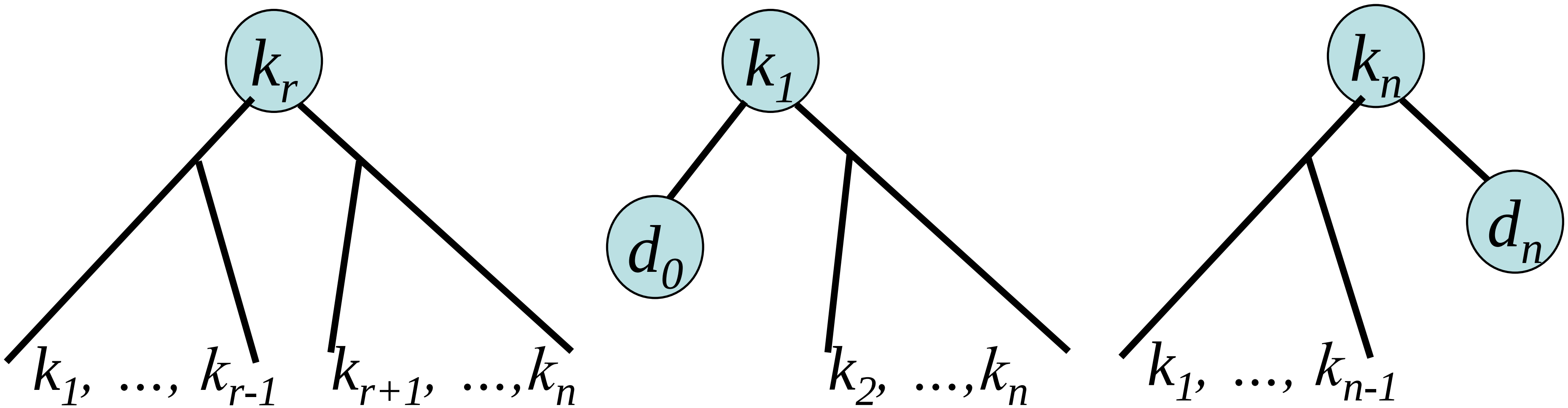
$$E(T)=\sum_{i=1}^{n}(DEP_T(k_i)+1)p_i+\sum_{j=0}^{n}(DEP_T(d_i)+1)q_i$$

- 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

优化二叉搜索树结构的分析

- 优化解的结构观察

$K=\{k_1, k_2, \ldots, k_n\}$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

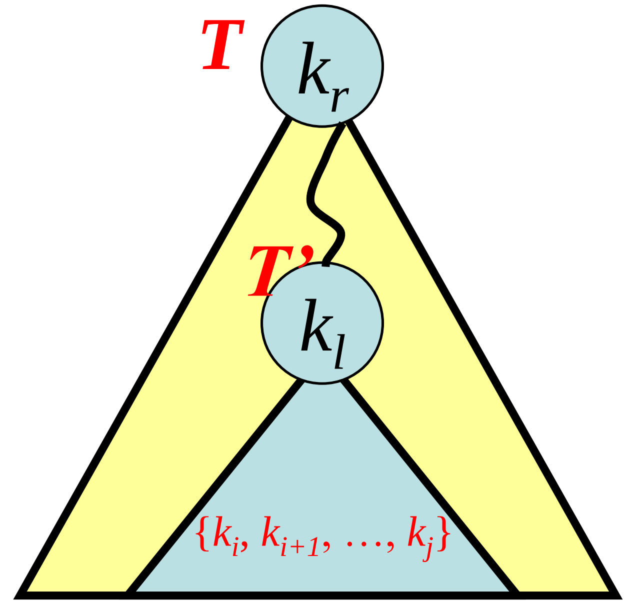
如果$r=n$, 右子树仅包含d_n

- 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$\{k_i, k_{i+1}, \ldots, k_j\}$的子树$T'$，则$T'$是关于关键字集合$\{k_i, k_{i+1}, \ldots, k_j\}$的子问题的优化解.

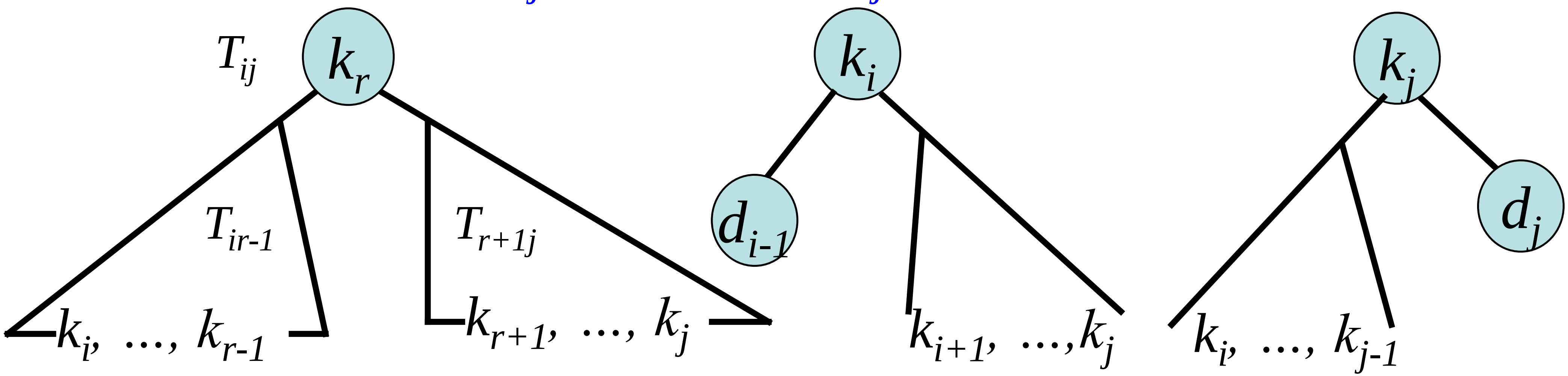
证明： 若不然，必有关键字集$\{k_i, k_{i+1}, \ldots, k_j\}$子树$T''$，$T''$的期望搜索代价低于$T'$.

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

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

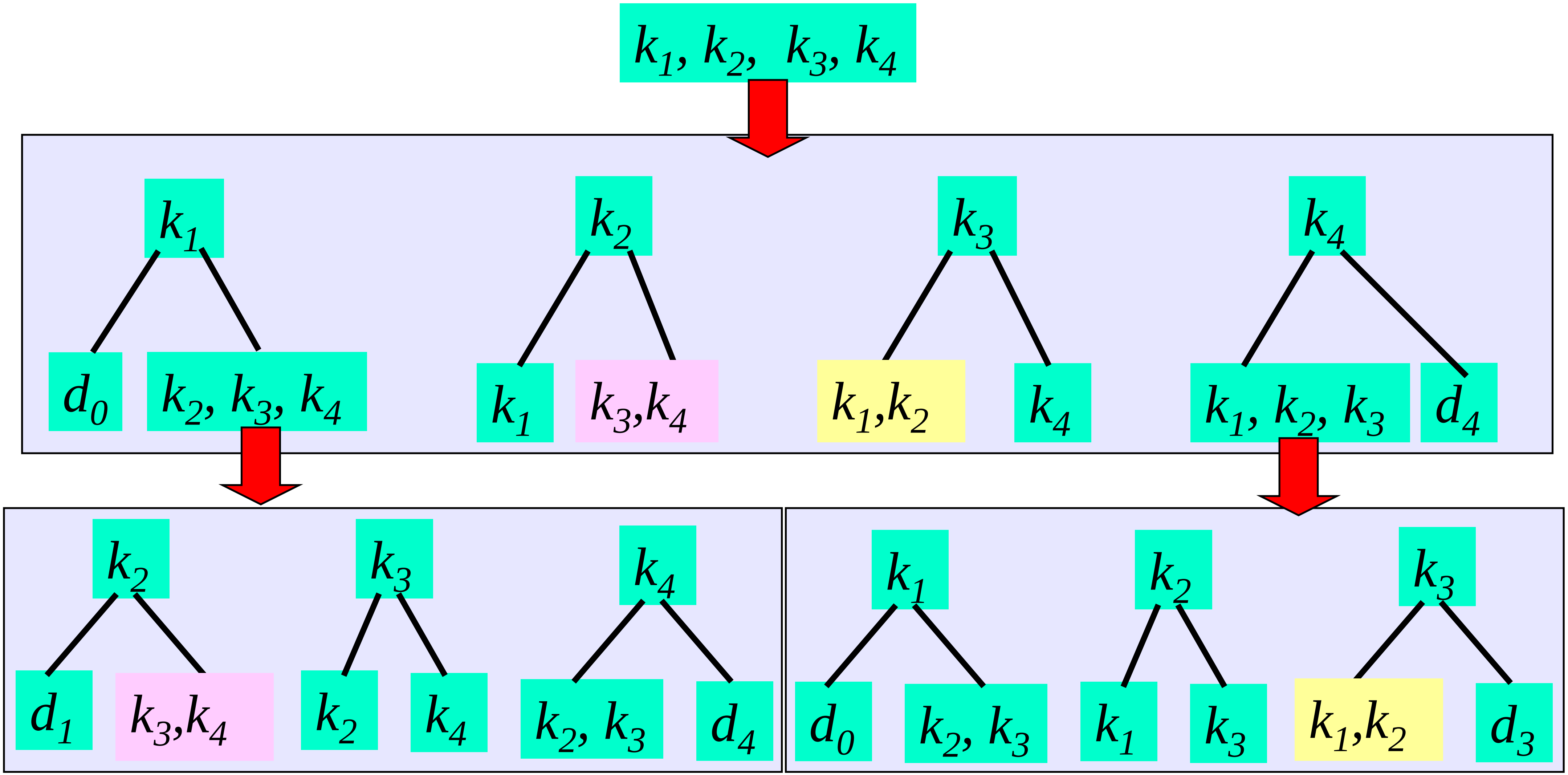


只要对于每个$k_r \in K$, 确定$\{k_i, \ldots, k_{r-1}\}$和$\{k_{r+1}, \ldots, k_j\}$的优化解, 我们就可以求出K的优化解.

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

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

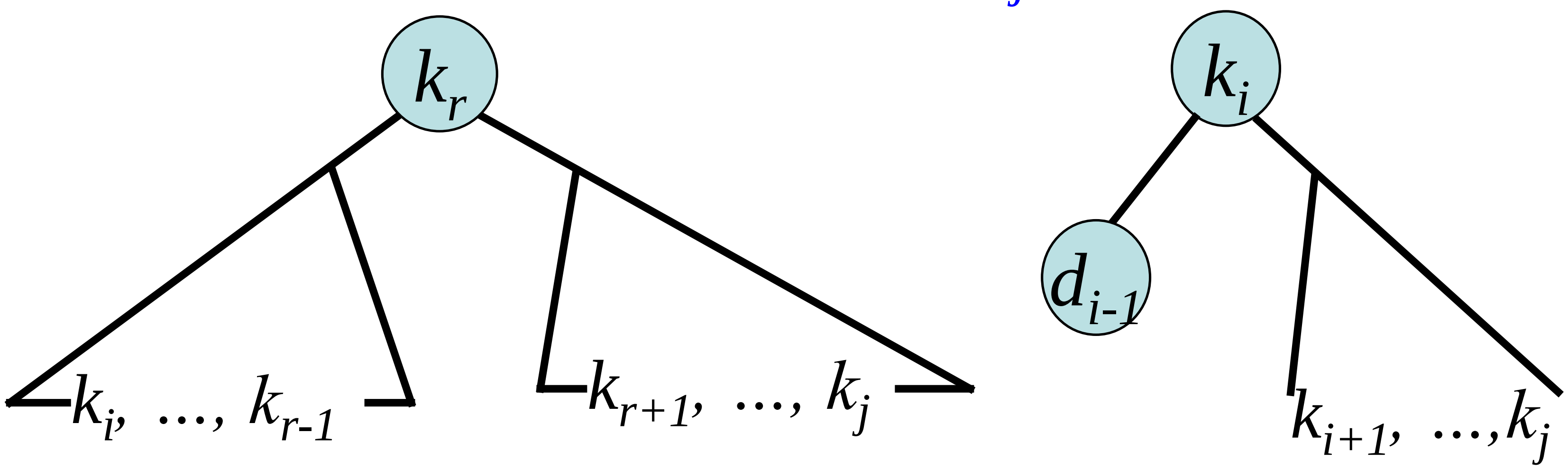
子问题重叠性


k_1, k_2, k_3, k_4
k_1
d_0
k_2, k_3, k_4
k_2
k_1
k_3,k_4
k_3
k_1,k_2
k_4
k_4
k_1, k_2, k_3
d_4
k_2
d_1
k_3,k_4
k_3
k_2
k_4
k_4
k_2, k_3
d_4
k_1
d_0
k_2, k_3
k_2
k_1
k_3
k_3
k_1,k_2
d_3

- 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

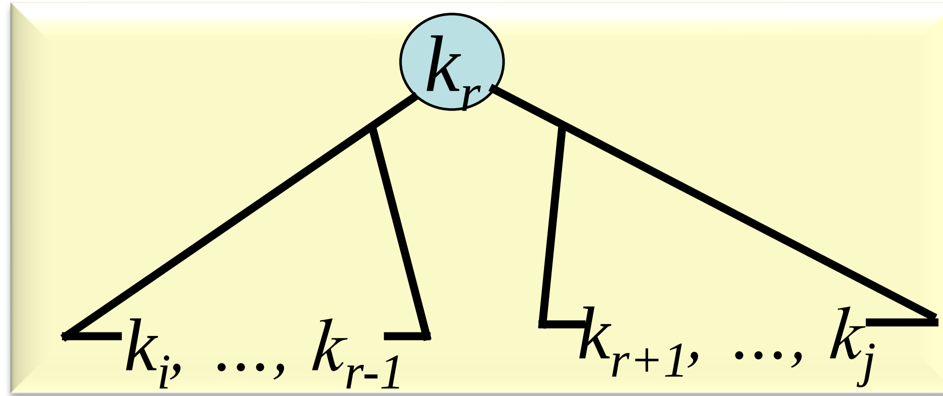
- 令$E(i, j)$为$\{k_i, \ldots, k_j\}$的优化解T_{ij}的期望搜索代价
  - 当$j=i-1$时，T_{ij}中只有叶结点d_{i-1}, $E(i, i-1)=q_{i-1}$
  - 当$j\geq i$时，选择一个$k_r\in\{k_i, \ldots, k_j\}$:



当把左右优化子树放进T_{ij}时，每个结点的深度增加1

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

HITWH
SE



- 计算$W(i, r-1)$和$W(r+1, j)$

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

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

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

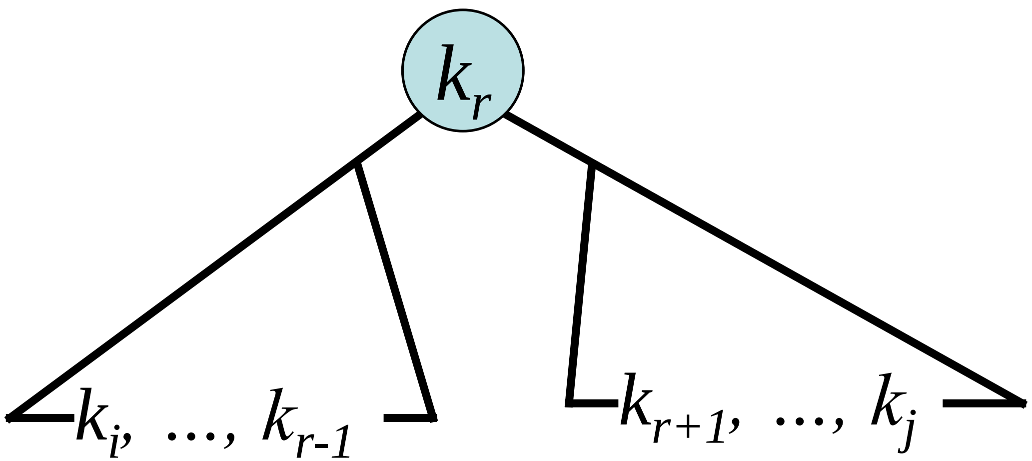
由$W(i,r-1)=\sum_{l=i}^{r-1}p_l+\sum_{l=i-1}^{r-1}q_l$ ， $W(i, i-1)=q_{i-1}$

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

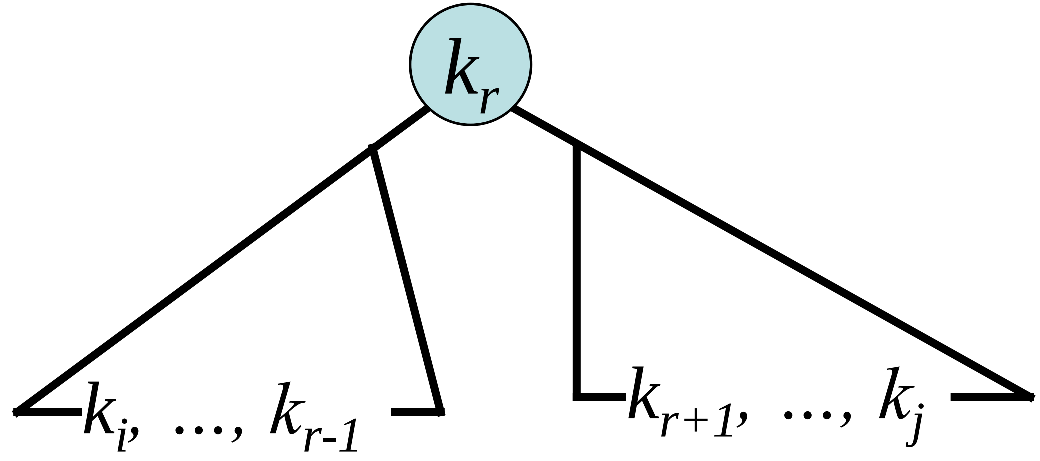
$W(i, j) = W(i, r\text{-}1) + W(r+1, j) + p_r = W(i, j\text{-}1) + p_j + q_j$

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

$E(i, j) = P_r + E(i, r\text{-}1) + W(i, r\text{-}1) + E(r+1, j) + W(r+1, j)$



$$E(i, j) = E(i, r\text{-}1) + E(r+1, j) + W(i, j)$$



总之

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

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

- 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

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

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

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

$E(i+1, j)$

$E(i+2, j)$

……

$E(j+1, j)$

- 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

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

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

$q_0=$	$E(1,0)$		$E(1,1)$	$E(1,2)$	$E(1,3)$	$E(1,4)$
		$q_1=$	$E(2,1)$	$E(2,2)$	$E(2,3)$	$E(2,4)$
			$q_2=$	$E(3,2)$	$E(3,3)$	$E(3,4)$
				$q_3=$	$E(4,3)$	$E(4,4)$
					$q_4=$	$E(5,4)$

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

$W(i, j\text{-}1)$	$W(i, j)$

$q_0 =$	$W(1,0)$	$W(1,1)$	$W(1,2)$	$W(1,3)$	$W(1,4)$
	$q_1 =$	$W(2,1)$	$W(2,2)$	$W(2,3)$	$W(2,4)$
		$q_2 =$	$W(3,2)$	$W(3,3)$	$W(3,4)$
			$q_3 =$	$W(4,3)$	$W(4,4)$
				$q_4 =$	$W(5,4)$

- 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 $\{k_i, \ldots, k_j\}$优化解的根

$$E(i, j)=q_{i-1} \quad If \;\; j=i-1$$
$$E(i, j)=min_{i\le r\le j}\{E(i,r-1)+E(r+1,j)+W(i, j)\} \quad If \; j\ge i$$

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

$E(1,0)$	$E(1,1)$	$E(1,2)$	$E(1,3)$	$E(1,4)$
	$E(2,1)$	$E(2,2)$	$E(2,3)$	$E(2,4)$
		$E(3,2)$	$E(3,3)$	$E(3,4)$
			$E(4,3)$	$E(4,4)$
				$E(5,4)$

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 $E(i, j)=t$; $Root(i, j)=r$;
Return E and $Root$

算法的复杂性

- 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)=∞;
         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; Root(i, j)=r;
 Return E and 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

- 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="snowy", y="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

优化解结构分析

- 寻找优化解析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

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

不能遗漏析分方式

优化解结构分析

- 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

$- \rightarrow U$

$O \rightarrow N$

$W \rightarrow -$

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。

优化解结构分析

- 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$X_1X_2 \dots X_i \dots X_K$的子序列，后者除$x$以外的元素都是“—”

y是$Y_1Y_2 \dots Y_i \dots Y_K$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“—”，$1 \leq i \leq K$

$X_1X_2 \dots X_i \dots X_K$称为$x$的空扩展字符串

- 满足不重叠性的编辑序列

$$\begin{array}{ccccccc} \mathcal{X}_1 & \mathcal{X}_2 & \ldots & \mathcal{X}_i & \ldots & \mathcal{X}_K \\ \downarrow & \downarrow & & \downarrow & & \downarrow \\ \mathcal{Y}_1 & \mathcal{Y}_2 & \ldots & \mathcal{Y}_i & \ldots & \mathcal{Y}_K \end{array}$$

x是$\mathcal{X}_1\mathcal{X}_2\ldots\mathcal{X}_i\ldots\mathcal{X}_K$的子序列，后者除$x$以外的元素都是“－”

y是$\mathcal{Y}_1\mathcal{Y}_2\ldots\mathcal{Y}_i\ldots\mathcal{Y}_K$的子序列，后者除$y$以外的元素都是“－”

$\mathcal{X}_i$和$\mathcal{Y}_i$不能同时等于“－”，$1 \le i \le K$

$\mathcal{X}_i = \mathcal{Y}_i$：字符$\mathcal{X}_i$原样保留

$\mathcal{X}_i =$ “－”：插入字符$\mathcal{Y}_i$

$\mathcal{Y}_i =$ “－”：删除字符$\mathcal{X}_i$

否则：字符$\mathcal{X}_i$修改为字符$\mathcal{Y}_i$

字符匹配：满足不重叠性的编辑序列图示中，字符$\mathcal{X}_i$和字符$\mathcal{Y}_i$匹配，$1 \le i \le K$

起点和终点不相同的“↓”的数量即为相应的编辑距离

优化解结构分析

- 寻找优化解析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

$$ED[m,n] = \begin{cases} ? \\ \dots \\ ? \end{cases}$$

根据$x[m]$的匹配位置分情况讨论

不能遗漏析分方式

优化解结构分析

- 在$\mathcal{Y}_1\,\mathcal{Y}_2\,\dots\mathcal{Y}_i\,\dots\mathcal{Y}_K$中，$x[m]$匹配$\mathcal{Y}_p$，$y[j]$对应$\mathcal{Y}_{q[j]}$，数组$q[j]$存储$y[1:n]$在$\mathcal{Y}$中的位置

Case 1：$p \le q[n-1]$

$ED[m,n-1]$

$$\begin{array}{cccccccccccc} \mathcal{X}_1 & \dots & \dots & \dots & \dots & \dots & \dots & X_m & - & \cdots & - & - \\ \downarrow & \dots & \dots & \dots & \dots & \dots & \dots & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \mathcal{Y}_1 & \dots & \dots & \dots & \dots & \dots & \dots & \mathcal{Y}_p & \mathcal{Y}_{p+1} & \cdots & \mathcal{Y}_{q[n-1]} & Y_n \\ & & & & & & & & & & (Y_{n-1}) & \end{array}$$

$\mathcal{X}_1\mathcal{X}_2\dots\mathcal{X}_i\dots\mathcal{X}_K$

$\mathcal{Y}_1\mathcal{Y}_2\dots\mathcal{Y}_i\dots\mathcal{Y}_K$

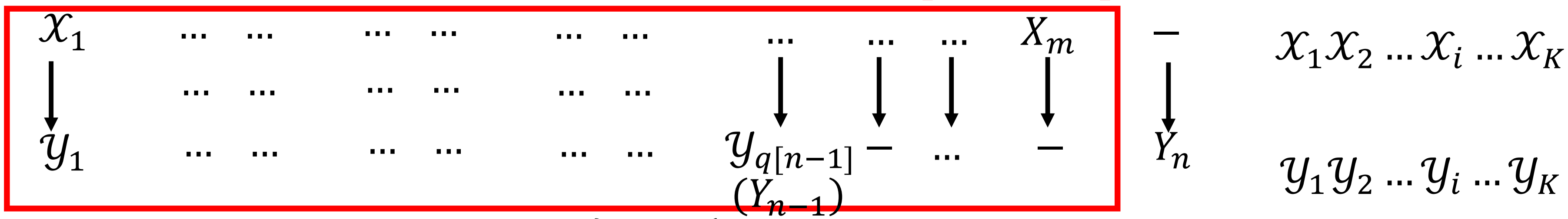
将$x[1:m]$转换成$y[1:n]$
的编辑序列示例图

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

- 在$Y_1 Y_2 \ldots Y_i \ldots Y_K$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：$q[n-1] < p < q[n]$

$ED[m, n-1]$



将$x[1:m]$转换成$y[1:n]$
的编辑序列示例图

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

- 在$\mathcal{Y}_1\mathcal{Y}_2\dots\mathcal{Y}_i\dots\mathcal{Y}_K$中，$x[m]$匹配$\mathcal{Y}_p$，$y[j]$对应$\mathcal{Y}_{q[j]}$，数组$q[j]$存储$y[1:n]$在$\mathcal{Y}$中的位置

Case 3：$p=q[n]$

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

$$\boxed{\begin{array}{cccccccccccc} x_1 & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \downarrow & \dots & \dots & \dots & \dots & \dots & \dots & \downarrow & \downarrow & \downarrow \\ y_1 & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{array}} \begin{array}{c} X_m \\ \downarrow \\ Y_n \end{array}$$

$\mathcal{X}_1\mathcal{X}_2\dots\mathcal{X}_i\dots\mathcal{X}_K$

$\mathcal{Y}_1\mathcal{Y}_2\dots\mathcal{Y}_i\dots\mathcal{Y}_K$

将$x[1:m]$转换成$y[1:n]$的编辑序列示例图

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

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

- 在$\mathcal{Y}_1\mathcal{Y}_2\ldots\mathcal{Y}_i\ldots\mathcal{Y}_K$中，$x[m]$匹配$\mathcal{Y}_p$，$y[j]$对应$\mathcal{Y}_{q[j]}$，数组$q[j]$存储$y[1:n]$在$\mathcal{Y}$中的位置

Case 4：$p > q[n]$ (隐含$p = K$)

$ED[m-1,n]$

$\mathcal{X}_1$ … … … … … … … … … X_m $\mathcal{X}_1\mathcal{X}_2\ldots\mathcal{X}_i\ldots\mathcal{X}_K$

… … … … … …

$\mathcal{Y}_1$ … … … … … … Y_n − … $\mathcal{Y}_K$ (−) $\mathcal{Y}_1\mathcal{Y}_2\ldots\mathcal{Y}_i\ldots\mathcal{Y}_K$

将$x[1:m]$转换成$y[1:n]$
的编辑序列示例图

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

优化解结构分析

- 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$Y[n]$：

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

如果$X[m]$匹配$Y[n]$之后的字符：

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

上述三种情况涵盖所有无重叠操作的编辑序列

优化子结构证明

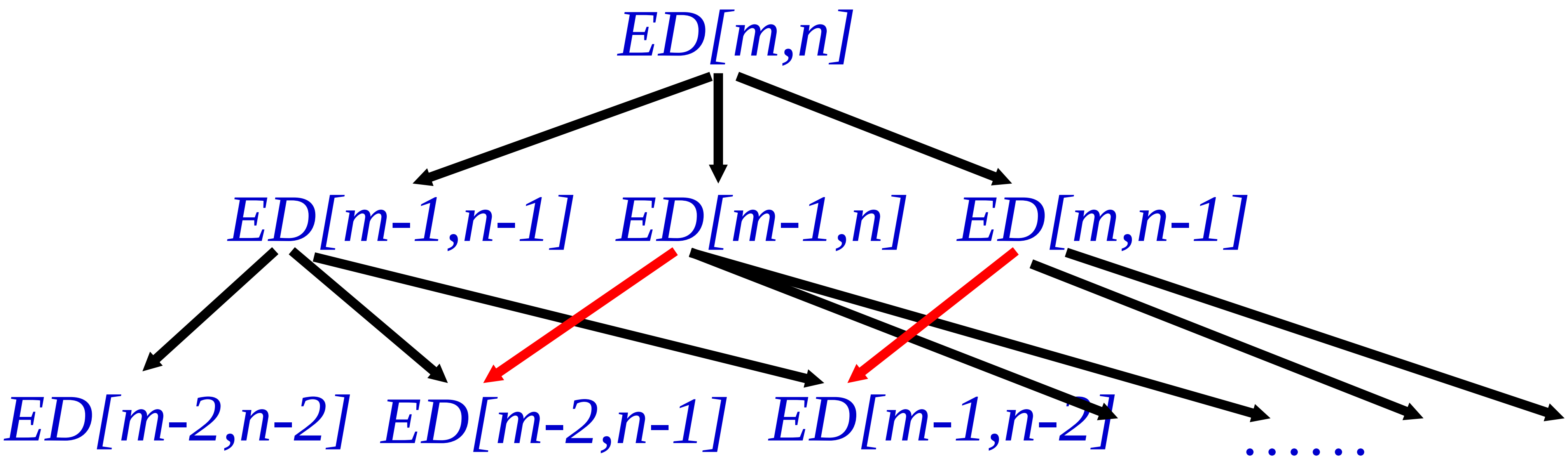
$$ED[m,n] = min\begin{cases} ED[m-1,n]+1 \\ ED[m-1,n-1]+diff(X[m],Y[n]) \\ ED[m,n-1]+1 \end{cases}$$

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$k < ED[m-1,n]$。如此，找到$k+1$次编辑操作将x转换为y，$k+1 < ED[m,n]$，与$ED[m,n]$为优化解矛盾！

情况二、情况三的证明过程与情况一类似

子问题重叠性



最小编辑距离问题具有子问题重叠性

最小编辑距离的递归方程

- 计算$X[1:i]$和$Y[1:j]$的最小编辑距离$ED[i,j]$

$$ED[i,0] = i \qquad 1 \le i \le m$$

$$ED[0,j] = j \qquad 1 \le j \le n$$

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

递归划分与自底向上求解

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

$$ED[i,0] = i \qquad 1 \le i \le m$$

$$ED[0,j] = j \qquad 1 \le j \le n$$

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

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

		y_j	R	E	N	A	T	O
i=0	x_i							
	R							
	O							
	N							
	A							
	L							
	D							
	O							
		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	*R*	*E*	*N*	*A*	*T*	*O*
i=0	x_i	0	1	2	3	4	5	6
	R	1	↖0	←1	←2	←3	←4	←5
	O	2	↑1	↖1	←2	←3	←4	↖4
	N	3	↑2	↑2	↖1	←2	←3	←4
	A	4	↑3	↑3	↑2	↖1	←2	←3
	L	5	↑4	↑4	↑3	↑2	↖2	↖3
	D	6	↑5	↑5	↑4	↑3	↖3	↖3
	O	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	R	E	N	A	T	O
$i=0$ x_i	0	1	2	3	4	5	6
R	1	↖ 0	←1	←2	←3	←4	← 5
O	2	↑ 1	↖1	←2	←3	←4	↖ 4
N	3	↑ 2	↑2	↖1	←2	←3	←4
A	4	↑ 3	↑3	↑2	↖1	←2	←3
L	5	↑ 4	↑4	↑3	↑2	↖2	↖ 3
D	6	↑ 5	↑5	↑4	↑3	↖3	↖ 3
O	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	R	E	N	A	T	O
i=0 x_i	0	1	2	3	4	5	6
R	1	↖0	←1	←2	←3	←4	←5
O	2	↑1	↖1	←2	←3	←4	↖4
N	3	↑2	↑2	↖1	←2	←3	←4
A	4	↑3	↑3	↑2	↖1	←2	←3
L	5	↑4	↑4	↑3	↑2	↖2	↖3
D	6	↑5	↑5	↑4	↑3	↖3	↖3
O	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：完成优化解的构造算法，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

在E中计算大量不必要的元素！

- 改进算法的两点启发

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

$$ED[i,0] = i \qquad 1 \le i \le m$$

$$ED[0,j] = j \qquad 1 \le j \le n$$

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

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

			R	E	N	A	T	O
i=0	x_i							
	R							
	O							
	N							
	A							
	L							
	D							
	O							

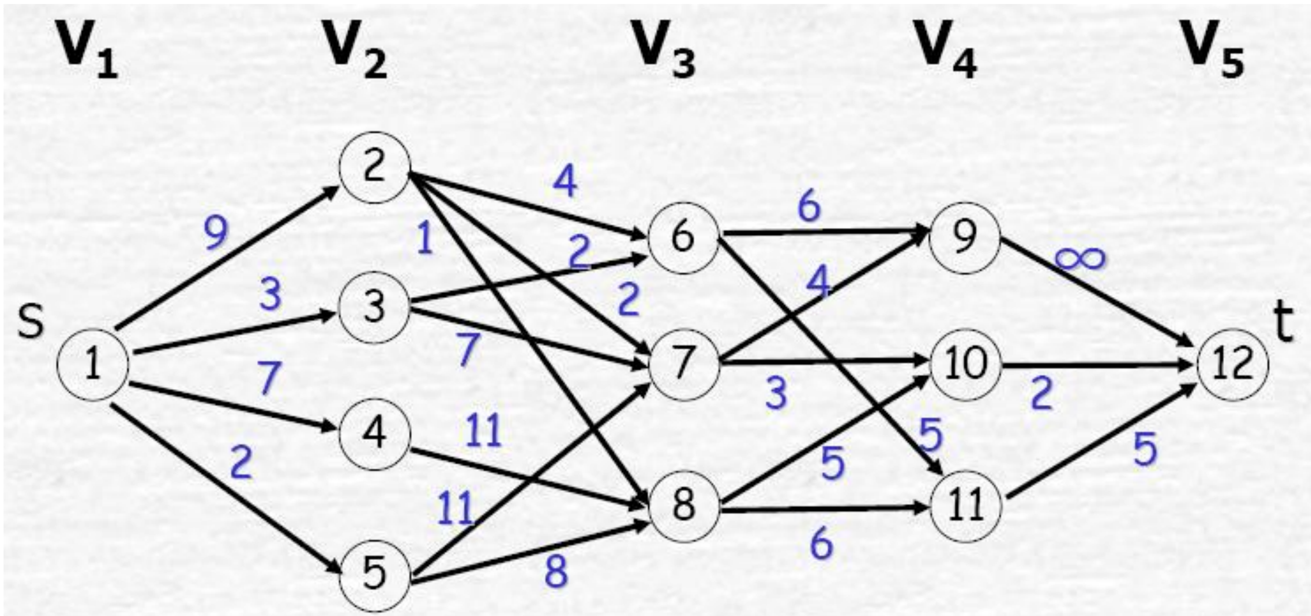
算法的复杂性

x=“RONALDO”,
y=“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.

多段图规划

- 优化解的结构分析

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

子集求和

- 问题定义

  输入：集合$S=\{n$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

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

  $$M(i, 0) = True$$
  $$M(0, j) = False \qquad \text{for } j>0$$

- 最长增长子序列问题

输入：由n个数组成的一个序列S：$a_1,a_2,\ldots,a_n$

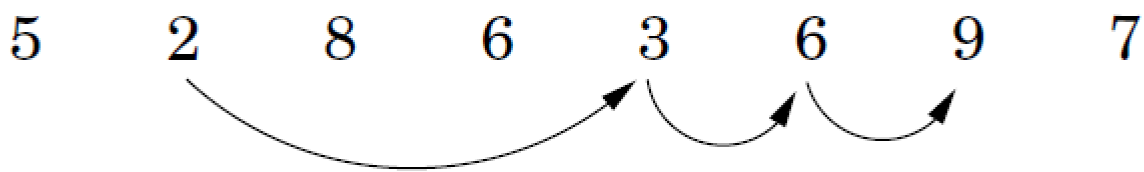
输出：子序列$S'=b_1,b_2,\ldots,b_k$，满足：

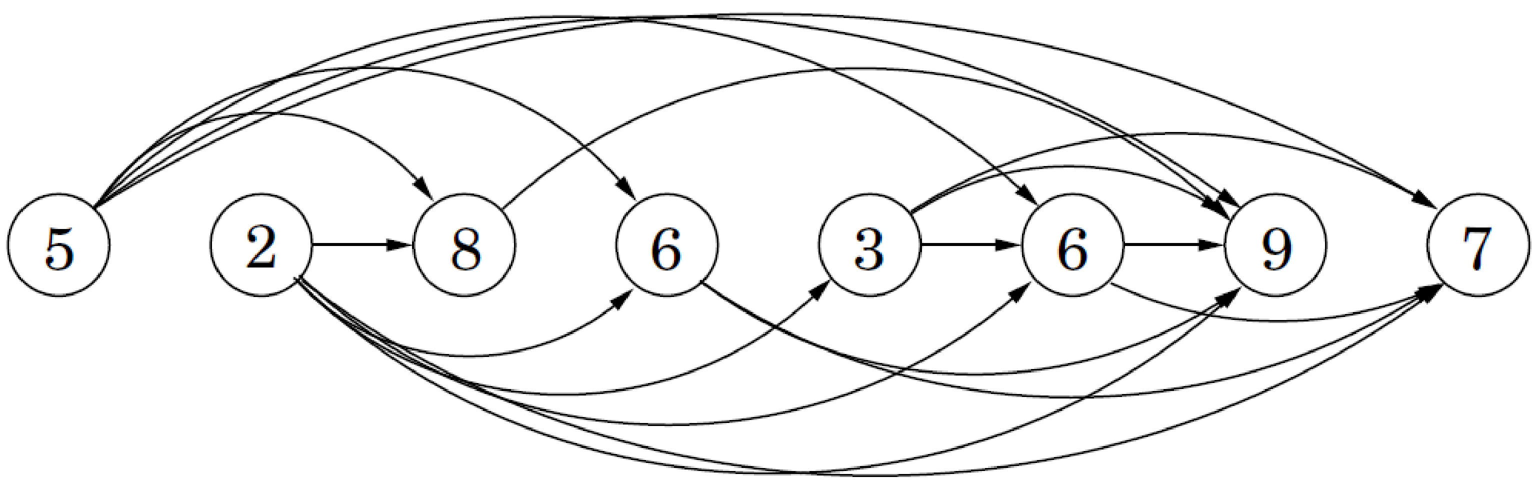
(1) $b_{i1} \le b_{i2} \le \ldots \le b_{ik}$，

(2) $|S'|$最大

5 2 8 6 3 6 9 7

- 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

总结

- 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

作业：使用递归

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

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

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

作业：使用栈

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

Construct-BST(Root)
← Root.size;
Initiate node r;
r.range ← (1, #);
Stack S ← ∅;
S.push(r);
While S ≠ ∅
    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 ← Root[i, j];
Print (;
Print-BST(Root, i, L-1);
Print) ;
Print $k_{Root[i,j]}$;
Print (;
Print-BST(Root, L+1, j);
Print) ;

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