

# 第3章 动态规划

## (Dynamic Programming)

3.1 动态规划的设计思想

3.2 动态规划的设计要素

3.3 动态规划算法的典型应用

3.3.1 投资问题

3.3.2 背包问题

3.3.3 最长公共子序列LCS

3.3.4 图像压缩

3.3.5 最大子段和

3.3.6 最优二分检索树

3.3.7 生物信息学中的动态规划算法

## 3.1 动态规划的设计思想

**例3.1** 求从始点到终点的最短路径

输入：起点集合  $\{S_1, S_2, \dots, S_m\}$  ,

终点集合  $\{T_1, T_2, \dots, T_m\}$  ,

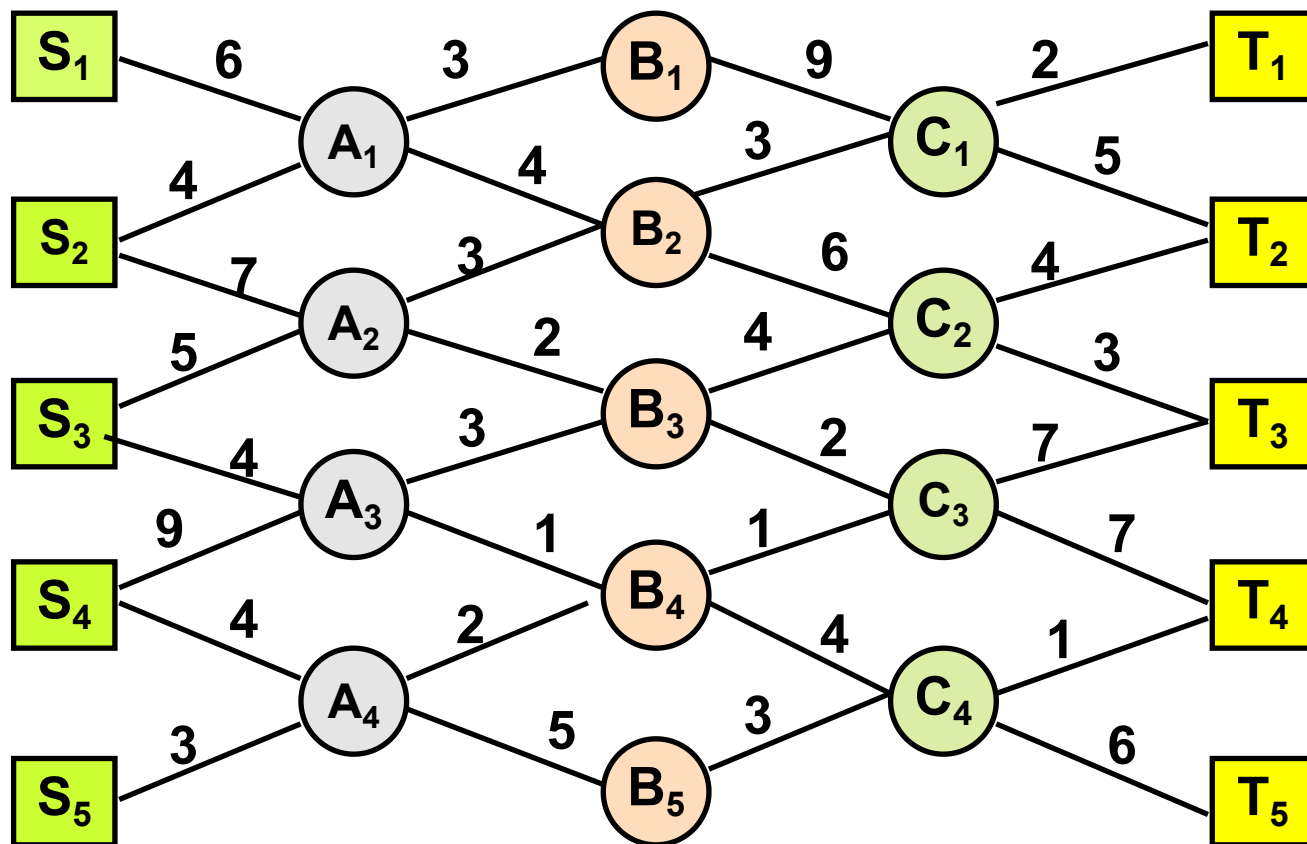
中间结点集,

边集  $E$ , 对于任意边  $e$  有长度

输出：一条从起点到终点的最短路

# 实例

例3.1 求从始点到终点的最短路径



# 思路分析

## 蛮力算法:

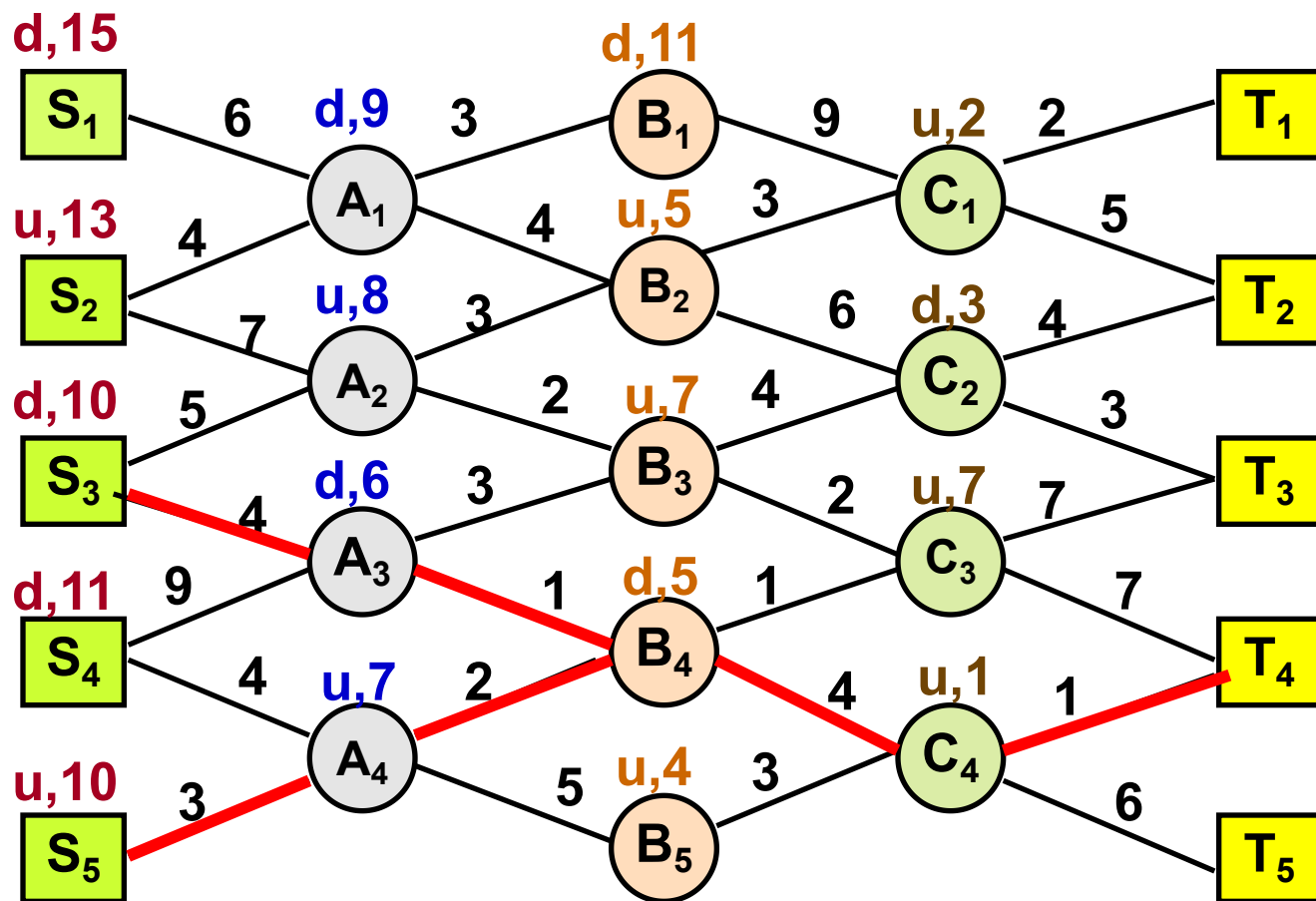
- 穷举每一个起点到每一个终点所有可能的路径，计算每条路径长度，从其中找出最短路径。
- 令 $m$ 为起点或终点个数，如果网络的层数为 $n$ ，那么路径条数大致达到 $O(m2^n)$ ，其中2指的是多数节点都有2条路可选。

## 动态规划算法:

- 从终点向起点回推;
- 每步对应子问题的终点不变，但起点前移，使得前步已经求解的问题恰好是后面新问题的子问题;
- 最后一步最大的子问题即为原始问题。

# 动态规划求解

例3.1 求从始点到终点的最短路径



# 动态规划的基本思想

解：判断序列

后边界不变, 前边界前移

$$F(C_l) = \min_m \{C_l T_m\}$$

 决策 1

$$F(B_k) = \min_l \{B_k C_l + F(C_l)\}$$

  决策 2

$$F(A_j) = \min_k \{A_j B_k + F(B_k)\}$$

   决策 3

$$F(S_i) = \min_j \{S_i A_j + F(A_j)\}$$

    决策 4

$S_i$     $A_j$     $B_k$     $C_l$     $T_m$

时间复杂度为:  $O(mn)$

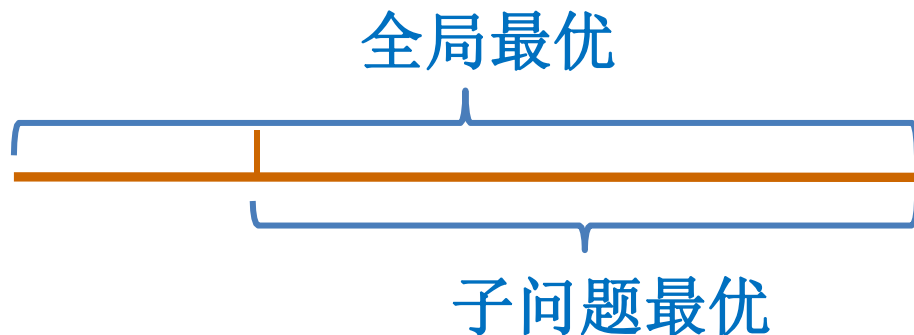
**优化函数的特点：**任何最短路径的子路径都是相对于子路径始点和终点的最短路径

**求解步骤：**确定子问题的边界、从最小的子问题开始进行多步判断

# 使用动态规划技术的条件:优化原则

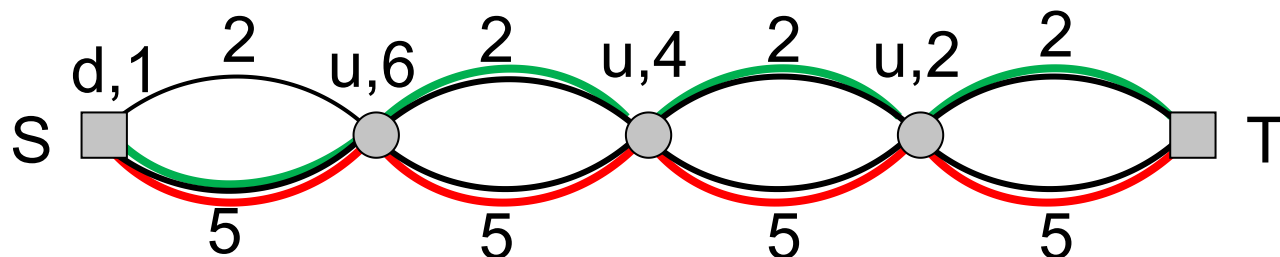
**动态规划初步总结：**求解过程是多阶段决策（优化）问题；求解过程是多步判断，从小到大依次求解子问题，最后求解的子问题就是原问题。

**优化原则：**一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优的决策序列。



# 优化原则反例

例3.2 求总长模10的最小路径（反例）



最优解：下、下、下、下（20模10为0）

动态规划算法的解：下、上、上、上

不满足优化原则，不能使用动态规划设计技术



## 3.2 动态规划算法设计要素

**例3.3** 矩阵乘法： 设 $A_1, A_2, \dots, A_n$ 为矩阵序列， $A_i$ 为 $P_{i-1} \times P_i$ 阶矩阵， $i = 1, 2, \dots, n$ . 确定乘法顺序使得元素相乘的总次数最少.

输入： 向量  $P = \langle P_0, P_1, \dots, P_n \rangle$ ,  $n$ 个矩阵的行数、列数

输出： 矩阵链乘法加括号的位置

## 3.2 动态规划算法设计要素

矩阵 $A$ :  $i$  行  $j$  列,  $B$ :  $j$  行  $k$  列, 以元素相乘作基本运算, 计算  $AB$  的工作量为  $ijk$

$$\begin{bmatrix} \dots \\ a_{t1} & a_{t2} & \dots & a_{tj} \\ \dots \end{bmatrix} \begin{bmatrix} b_{1s} \\ b_{2s} \\ \vdots \\ b_{js} \end{bmatrix} = \begin{bmatrix} c_{ts} \end{bmatrix}$$

$C$ 为 $i$ 行 $k$ 列, 计算每个元素需 $j$ 次乘法:  
$$c_{ts} = a_{t1} b_{1s} + a_{t2} b_{2s} + \dots + a_{tj} b_{js}$$

实例:  $P = \langle 10, 100, 5, 50 \rangle$

$A_1: 10 \times 100, A_2: 100 \times 5, A_3: 5 \times 50,$

乘法次序

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

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

# 搜索空间的规模

枚举算法：加 $n$ 对括号的方法有  $\frac{1}{n+1}\binom{2n}{n}$  种，是一个 Catalan 数，是指数级别（用到 Stirling 公式）

$$\begin{aligned} W(n) &= \Omega\left(\frac{1}{n+1} \frac{(2n)!}{n!n!}\right) = \Omega\left(\frac{1}{n+1} \frac{\sqrt{2\pi 2n} \left(\frac{2n}{e}\right)^{2n}}{\sqrt{2\pi n} \left(\frac{n}{e}\right)^n \sqrt{2\pi n} \left(\frac{n}{e}\right)^n}\right) \\ &= \Omega\left(\frac{1}{n+1} \frac{n^{\frac{1}{2}} 2^{2n} n^{2n} e^n e^n}{e^{2n} n^{\frac{1}{2}} n^n n^{\frac{1}{2}} n^n}\right) = \Omega(2^{2n} / n^{\frac{3}{2}}) \end{aligned}$$

# 搜索空间的规模

枚举算法：加 $n$ 对括号的方法有  $\frac{1}{n+1}\binom{2n}{n}$  种，是一个 Catalan 数，是指数级别（用到 Stirling 公式）

$$\begin{aligned} W(n) &= \Omega\left(\frac{1}{n+1} \frac{(2n)!}{n!n!}\right) = \Omega\left(\frac{1}{n+1} \frac{\sqrt{2\pi 2n} \left(\frac{2n}{e}\right)^{2n}}{\sqrt{2\pi n} \left(\frac{n}{e}\right)^n \sqrt{2\pi n} \left(\frac{n}{e}\right)^n}\right) \\ &= \Omega\left(\frac{1}{n+1} \frac{n^{\frac{1}{2}} 2^{2n} n^{2n} e^n e^n}{e^{2n} n^{\frac{1}{2}} n^n n^{\frac{1}{2}} n^n}\right) = \Omega(2^{2n} / n^{\frac{3}{2}}) \end{aligned}$$

# 动态规划算法

由  $i$  和  $j$  确定子问题的边界：输入  $P = \langle P_0, P_1, \dots, P_n \rangle$ ,  $A_{i..j}$  表示乘积  $A_i A_{i+1} \dots A_j$  的结果 ( $i=1, \dots, n$ )，其最后一次相乘是

$$A_{i..j} = A_{i..k} A_{k+1..j}$$

确定优化函数和递推方程：

$m[i, j]$  表示得到  $A_{i..j}$  的最少的相乘次数，则递推方程和初值

$$m[i, j] = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} \{m[i, k] + m[k+1, j] + P_{i-1} * P_k * P_j\} & i < j \end{cases}$$

设立标记函数：为了确定加括号的次序，设计表  $s[i, j]$ ，记录求得最优时最后一次运算的划分位置

# 算法：递归实现

## 算法3.1 RecurMatrixChain ( $P, i, j$ )

1.  $m[i, j] \leftarrow \infty$
2.  $s[i, j] \leftarrow i$
3. for  $k \leftarrow i$  to  $j-1$  do
4.      $q \leftarrow \text{RecurMatrixChain}(P, i, k)$   
           $+ \text{RecurMatrixChain}(P, k+1, j) + p_{i-1}p_kp_j$
5.     if  $q < m[i, j]$
6.         then  $m[i, j] \leftarrow q$
7.          $s[i, j] \leftarrow k$
8. return  $m[i, j]$

这里没有写出算法的全部描述（进入递归调用的初值等）

# 递归实现的复杂性

复杂性满足递推关系

$$T(n) \geq \begin{cases} O(1) & n = 1 \\ \sum_{k=1}^{n-1} (T(k) + T(n-k) + O(1)) & n > 1 \end{cases}$$

$$T(n) \geq \Theta(n) + \sum_{k=1}^{n-1} T(k) + \sum_{k=1}^{n-1} T(n-k) = \Theta(n) + 2 \sum_{k=1}^{n-1} T(k)$$

**定理：** 对于  $n > 1$ ,  $T(n) = \Omega(2^{n-1})$




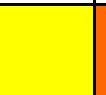
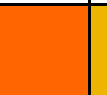


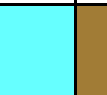
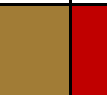


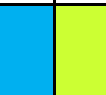



归纳法证明.  $n=2$ ,  $T(2) \geq c = c_1 2^{2-1}$ ,  $c_1 = c/2$  为某个正数

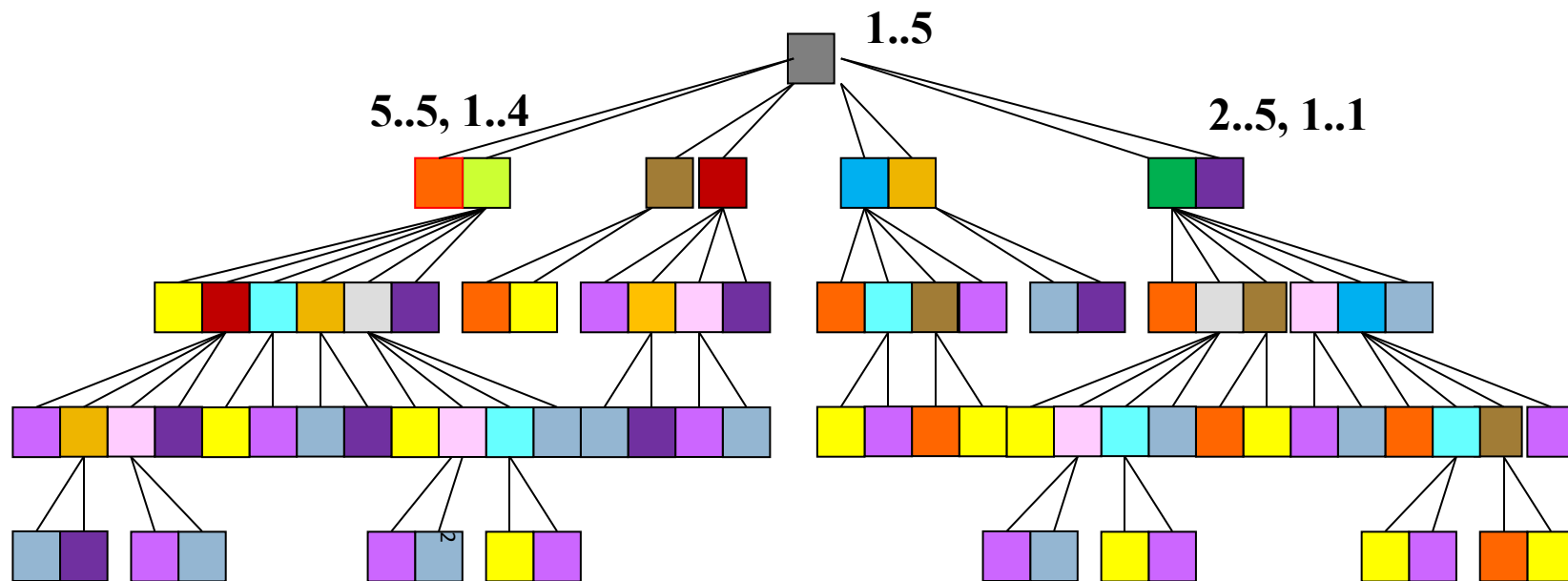
假设对于任何小于  $n$  的  $k$  命题为真, 则存在  $c'$  使得

$$\begin{aligned} T(n) &\geq c' n + 2 \sum_{k=1}^{n-1} T(k) \geq c' n + 2 \sum_{k=2}^{n-1} c_1 2^{k-1} \\ &= c' n + c_1 (2^n - 4) \geq c_1 2^{n-1} \end{aligned}$$

# 复杂性高的原因：子问题重复计算

$n=5$ ，计算子问题：81个；不同的子问题：15个

子问题	1-1	2-2	3-3	4-4	5-5	1-2	2-3	3-4	4-5	1-3	2-4	3-5	1-4	2-5	1-5
															
数	8	12	14	12	8	4	5	5	4	2	2	2	1	1	1

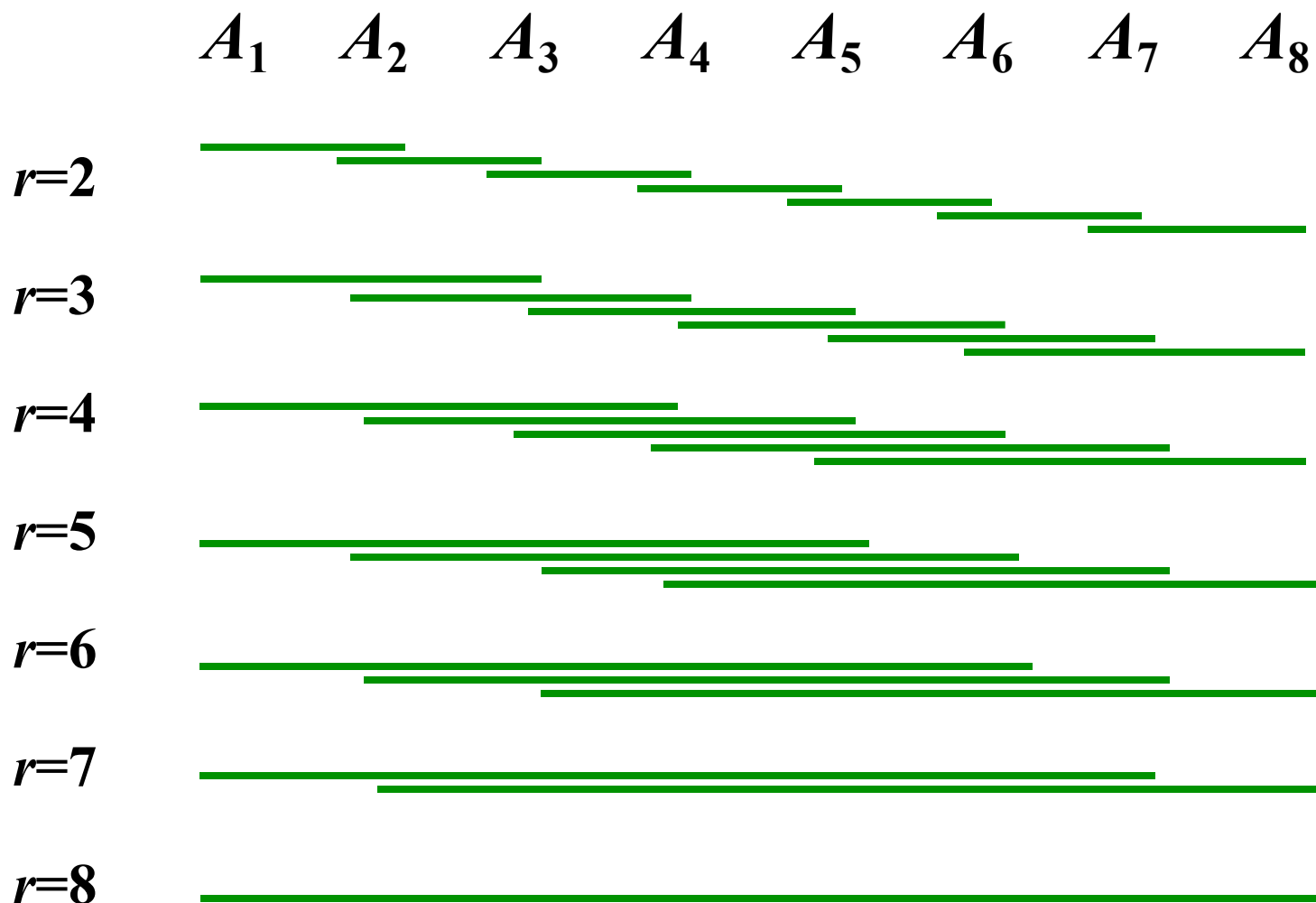




# 算法：迭代实现

- 目的：每个子问题只计算一次
  - 迭代过程
    - 从最小的子问题算起
    - 考虑计算顺序，以保证后面用到的值前面已经计算好
    - 存储结构保存计算结果——备忘录
  - 解的追踪
    - 设计标记函数标记每步的决策
    - 考虑根据标记函数追踪解的算法
- 长度1：只含1个矩阵，有 $n$ 个子问题
- 长度2：含2个矩阵， $n-1$ 个子问题
- 长度3：含3个矩阵， $n-2$ 个子问题
- ...
- 长度 $n-1$ ：含 $n-1$ 个矩阵，2个子问题
- 长度  $n$ ：原始问题，只有1个

# $n=8$ 的迭代过程



# 迭代实现的伪码

## 算法3.2 MatrixChain( $P, n$ )

1. 令所有的  $m[i, i]$  初值为0       $1 \leq i \leq n$
2. for  $r \leftarrow 2$  to  $n$  do      //  $r$  为计算的矩阵链长度
3.    for  $i \leftarrow 1$  to  $n-r+1$  do      //  $n-r+1$  为最后  $r$  链的始位置
4.      $j \leftarrow i+r-1$       // 计算链  $i-j$
5.      $m[i, j] \leftarrow m[i+1, j] + p_{i-1} * p_i * p_j$       //  $A_i(A_{i+1} .. A_j)$
6.      $s[i, j] \leftarrow i$       // 记录分割位置
7.     for  $k \leftarrow i+1$  to  $j-1$  do
8.         $t \leftarrow m[i, k] + m[k+1, j] + p_{i-1} * p_k * p_j$       //  $(A_i .. A_k)(A_{k+1} .. A_j)$
9.        if  $t < m[i, j]$
10.        then  $m[i, j] \leftarrow t$
11.         $s[i, j] \leftarrow k$

复杂性：行2,3,7循环进行都是  $O(n)$ ，循环内为  $O(1)$   
 $W(n) = O(n^3)$

# 实例

输入  $P = \langle 30, 35, 15, 5, 10, 20 \rangle$ ,  $n=5$ ,

矩阵链:  $A_1 A_2 A_3 A_4 A_5$ , 其中

$A_1: 30 \times 35$ ,  $A_2: 35 \times 15$ ,  $A_3: 15 \times 5$ ,  $A_4: 5 \times 10$ ,  $A_5: 10 \times 20$

## 备忘录

$r=1$	$m[1,1]=0$	$m[2,2]=0$	$m[3,3]=0$	$m[4,4]=0$	$m[5,5]=0$
$r=2$	$m[1,2]=15750$	$m[2,3]=2625$	$m[3,4]=750$	$m[4,5]=1000$	
$r=3$	$m[1,3]=7875$	$m[2,4]=4375$	$m[3,5]=2500$		
$r=4$	$m[1,4]=9375$	$m[2,5]=7125$			
$r=5$	$m[1,5]=11875$				

$r=2$	$s[1,2]=1$	$s[2,3]=2$	$s[3,4]=3$	$s[4,5]=4$	
$r=3$	$s[1,3]=1$	$s[2,4]=3$	$s[3,5]=3$		
$r=4$	$s[1,4]=3$	$s[2,5]=3$			
$r=5$	$s[1,5]=3$				

解的追踪:  $s[1,5]=3 \Rightarrow (A_1 A_2 A_3)(A_4 A_5)$ ;  $s[1,3]=1 \Rightarrow A_1(A_2 A_3)$

输出计算顺序:  $(A_1(A_2 A_3))(A_4 A_5)$ ; 乘法次数:  $m[1,5]=11875$

# 两种实现的比较

递归实现：时间复杂性高，空间较小

迭代实现：时间复杂性较低，空间消耗多

时间复杂性不同的原因：

- 递归实现
  - 子问题被多次重复计算
  - 子问题计算次数呈指数增长
- 迭代实现
  - 每个子问题只计算一次
  - 子问题的计算随问题规模成多项式增长

# 动态规划算法设计步骤

## (1) 划分子问题

用参数表达子问题的边界，将问题求解转变成多步判断的过程。

## (2) 确定优化函数

以该函数的极大(或极小)作为判断的依据，确定是否满足优化原则。

## (3) 列出关于优化函数的递推方程 (或不等式)和边界条件

## (4) 考虑是否需要设立标记函数

## (5) 自底向上计算，以备忘录方法 (表格)存储中间结果

## 3.3.1 投资问题

**例3.4**  $m$ 元钱,  $n$ 项投资,  $f_i(x)$ : 将  $x$  元投入第  $i$  个项目的效益

目标函数  $\max \{f_1(x_1) + f_2(x_2) + \dots + f_n(x_n)\}$

约束条件  $x_1 + x_2 + \dots + x_n = m, x_i \in \mathbb{N}$

实例: 5万元钱, 4个项目, 效益函数如下表所示

$x$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	0	0	0
1	11	0	2	20
2	12	5	10	21
3	13	10	30	22
4	14	15	32	23
5	15	20	40	24

# 子问题划分和优化函数

设 $F_k(x)$  表示  $x$ 元钱投给前 $k$  个项目的最大效益

多步判断:

假设知道  $p$  元钱 ( $p \leq x$ ) 投给前  $k-1$ 个项目的最大效益 $F_{k-1}(p)$  , 决定  $x$  元钱投给前  $k$  个项目的分配方案

递推方程和边界条件:

$$F_k(x) = \max_{0 \leq x_k \leq x} \{f_k(x_k) + F_{k-1}(x - x_k)\} \quad k > 1$$

$$F_1(x) = f_1(x)$$



$x$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	0	0	0
1	11	0	2	20
2	12	5	10	21
3	13	10	30	22
4	14	15	32	23
5	15	20	40	24

## k=1时实例的计算

$k=1$ 为初值

$F_1(1)=11, F_1(2)=12, F_1(3)=13,$

$F_1(4)=14, F_1(5)=15,$

## k=2时实例的计算

方案: (1,0), (0,1)

$$F_2(1) = \max\{f_1(1), f_2(1)\} = 11$$

方案: (2,0), (1,1), (0,2)

$$F_2(2) = \max\{f_2(2), F_1(1)+f_2(1), F_1(2)\} = 12$$

方案: (3,0), (2,1), (1,2), (0,3)

$$F_2(3) = \max\{f_2(3), F_1(1)+f_2(2), F_1(2)+f_2(1), F_1(3)\} = 16$$

类似地计算

$$F_2(4) = 21, F_2(5) = 26$$

# 实例的计算

$x$	$F_1(x)$ $x_1(x)$	$F_2(x)$ $x_2(x)$	$F_3(x)$ $x_3(x)$	$F_4(x)$ $x_4(x)$
1	11 1	11 0	11 0	20 1
2	12 2	12 0	13 1	31 1
3	13 3	16 2	30 3	33 1
4	14 4	21 3	41 3	50 1
5	15 5	26 4	43 4	61 1

$x_k(x)$ 为标记函数，即 $F_k(x)$ 取得最大值时应分给第 $k$ 个项目的钱数.

解  $x_1=1, x_2=0, x_3=3, x_4=1$   $F_4(5)=61$

# 算法的复杂度分析

表中有  $m$  行  $n$  列, 共计  $mn$  项

$$F_k(x) = \max_{0 \leq x_k \leq x} \{f_k(x_k) + F_{k-1}(x - x_k)\} \quad k > 1$$

$$F_1(x) = f_1(x)$$

对第  $F_k(x)$  项 ( $2 \leq k \leq n, 1 \leq x \leq m$ ) 要  $x+1$  次加法,  $x$  次比较

加法次数  $\sum_{k=2}^n \sum_{x=1}^m (x+1) = \frac{1}{2}(n-1)m(m+3)$

比较次数  $\sum_{k=2}^n \sum_{x=1}^m x = \frac{1}{2}(n-1)m(m+1)$

$$W(n) = O(nm^2)$$

## 3.3.2 背包问题 (Knapsack Problem)

**例3.5** 一个旅行者准备随身携带一个背包. 可以放入背包的物品有 $n$ 种, 每种物品的重量和价值分别为 $w_j, v_j$ . 如果背包的最大重量限制是 $b$ , 怎样选择放入背包的物品以使得背包的价值最大?

目标函数 
$$\max \sum_{j=1}^n v_j x_j$$

约束条件 
$$\sum_{j=1}^n w_j x_j \leq b,$$
$$x_j \in \mathbf{N}$$

### 线性规划问题

由线性条件约束的线性函数取最大或最小的问题

**整数规划问题** 线性规划问题的变量 $x_j$ 都是非负整数

# 子问题划分、优化函数、标记函数

$F_k(y)$ : 装前  $k$  种物品, 总重不超过  $y$ , 背包的最大价值

$i_k(y)$ : 装前  $k$  种物品, 总重不超过  $y$ , 背包达最大价值时  
装入物品的最大标号

类似投资问题:  $F_k(y) = \max_{0 \leq x_k \leq \lfloor y/w_k \rfloor} \{F_{k-1}(y - x_k w_k) + x_k v_k\}$

复杂性较高, 有  
没更好的方式?

递推方程、边界条件、标记函数:

$$F_k(y) = \max\{F_{k-1}(y), F_k(y - w_k) + v_k\}$$

$$F_0(y) = 0, \quad 0 \leq y \leq b, \quad F_k(0) = 0, \quad 0 \leq k \leq n$$

$$F_1(y) = \left\lfloor \frac{y}{w_1} \right\rfloor v_1$$

$$F_k(y) = -\infty \quad y < 0$$

$$i_k(y) = \begin{cases} i_{k-1}(y) & F_{k-1}(y) > F_k(y - w_k) + v_k \\ k & F_{k-1}(y) \leq F_k(y - w_k) + v_k \end{cases}$$

# 实例计算

$$v_1 = 1, \quad v_2 = 3, \quad v_3 = 5, \quad v_4 = 9,$$

$$w_1 = 2, \quad w_2 = 3, \quad w_3 = 4, \quad w_4 = 7,$$

$$b = 10$$

$F_k(y)$  的计算表如下:

$k \backslash y$	1	2	3	4	5	6	7	8	9	10
1	0	1	1	2	2	3	3	4	4	5
2	0	1	3	3	4	6	6	7	9	9
3	0	1	3	5	5	6	8	10	10	11
4	0	1	3	5	5	6	9	10	10	12

# 实例计算

$v_1 = 1, v_2 = 3, v_3 = 5, v_4 = 9,$   
 $w_1 = 2, w_2 = 3, w_3 = 4, w_4 = 7,$   
 $b = 10$

$$F_k(y) = \max\{F_{k-1}(y), F_k(y - w_k) + v_k\}$$

$$F_k(y) = \max_{0 \leq x_k \leq \lfloor y/w_k \rfloor} \{F_{k-1}(y - x_k w_k) + x_k v_k\}$$

$F_k(y)$  的计算表如下:

$k \backslash y$	1	2	3	4	5	6	7	8	9	10
1	0	1	1	2	2	3	3	4	4	5
2	0	1	3	3	4	6	6	7	9	9
3	0	1	3	5	5	6	8	10	10	11
4	0	1	3	5	5	6	9	10	10	12

# 用标记函数追踪问题的解

$k \backslash y$	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	1	1	1
2	0	1	2	2	2	2	2	2	2	2
3	0	1	2	3	3	3	3	3	3	3
4	0	1	2	3	3	3	4	3	4	4

在上例中, 求得

$$i_4(10)=4 \Rightarrow x_4 \geq 1$$

$$i_4(10-w_4)=i_4(3)=2 \Rightarrow x_2 \geq 1, x_4=1, x_3=0$$

$$i_2(3-w_2)=i_2(0)=0 \Rightarrow x_2=1, x_1=0$$

解  $x_1=0, x_2=1, x_3=0, x_4=1$



# 追踪解 $x_j$ 的算法

## 算法3.3 TrackSolution

输入:  $i_k(y)$ 表, 其中 $k=1,2,\dots,n$ ,  $y=1,2,\dots,b$

输出:  $x_1, x_2, \dots, x_n$ ,  $n$ 种物品的装入量

1. for  $j=1$  to  $n$  do
2.      $x_j \leftarrow 0$
3.      $y \leftarrow b, j \leftarrow n$
4.      $j \leftarrow i_j(y)$
5.      $x_j \leftarrow 1$
6.      $y \leftarrow y - w_j$
7.     while  $i_j(y) = j$  do
8.          $y \leftarrow y - w_j$
9.          $x_j \leftarrow x_j + 1$
10.    if  $i_j(y) \neq 0$  then goto 4

# 用标记函数追踪问题的解

$k \backslash y$	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	1	1	1
2	0	1	2	2	2	2	2	2	2	2
3	0	1	2	3	3	3	3	3	3	3
4	0	1	2	3	3	3	4	3	4	4

在上例中, 求得

$$i_4(10)=4 \Rightarrow x_4 \geq 1$$

$$i_4(10-w_4)=i_4(3)=2 \Rightarrow x_2 \geq 1, x_4=1, x_3=0$$

$$i_2(3-w_2)=i_2(0)=0 \Rightarrow x_2=1, x_1=0$$

解  $x_1=0, x_2=1, x_3=0, x_4=1$

# 时间复杂度

根据公式

$$F_k(y) = \max\{F_{k-1}(y), F_k(y - w_k) + v_k\}$$

备忘录需计算  $nb$  项，每项常数时间，**计算时间为  $O(nb)$** 。

**\*伪多项式时间算法：**时间为参数  $b$  和  $n$  的多项式，不是输入规模的多项式。参数  $b$  是整数，表达  $b$  需要  $\log b$  位，输入规模是  $\log b$ 。

# 背包问题的推广

物品数受限：第  $i$  种物品最多用  $n_i$  个.

0-1背包问题：  $x_i = 0, 1, i = 1, 2, \dots, n$ .

多背包问题：  $m$  个背包，背包  $j$  装入最大重量  $B_j, j = 1, 2, \dots, m$ .

二维背包问题：每件物品有重量  $w_i$  和体积  $t_i, i = 1, 2, \dots, n$ ，背包总重不超过  $b$ ，体积不超过  $V$ ，如何选择物品以得到最大价值.

## 3.3.3 最长公共子序列 LCS

### 相关概念

**定义**  $X$  的子序列  $Z$  : 设序列  $X, Z$ ,

$$X = \langle x_1, x_2, \dots, x_m \rangle$$

$$Z = \langle z_1, z_2, \dots, z_k \rangle$$

若存在  $X$  的元素构成的下标严格递增序列  $\langle x_{i_1}, x_{i_2}, \dots, x_{i_k} \rangle$

使得  $z_j = x_{i_j}, j = 1, 2, \dots, k$ , 则称  $Z$  是  $X$  的**子序列**

$X$  与  $Y$  的**公共子序列**  $Z$  :  $Z$  是  $X$  和  $Y$  的子序列

**子序列的长度**: 子序列的元素个数

# 问题描述

**例3.6** 给定序列  $X=\langle x_1, x_2, \dots, x_m \rangle$ ,  $Y=\langle y_1, y_2, \dots, y_n \rangle$   
求  $X$  和  $Y$  的最长公共子序列

实例

$X$ :    **A**    **B**    **C**    **B**    **D**    **A**    **B**

$Y$ :    **B**    **D**    **C**    **A**    **B**    **A**

一个最长公共子序列: **B**    **C**    **B**    **A**

蛮力算法:  $X$  有  $2^m$  个子序列,  $Y$  有  $2^n$  个子序列. 检查  $X$  的每个子序列在  $Y$  的子序列中出现是  $O(2^n)$  时间,  $X$  有  $2^m$  个子序列, 最坏情况下时间复杂度:  $O(2^{m+n})$

# 子问题划分及依赖关系

子问题边界：  $X$ 和 $Y$ 起始位置为1，  $X$ 的终止位置是  $i$ ，  $Y$ 的终止位置是  $j$ ， 记作

$$X_i = \langle x_1, x_2, \dots, x_i \rangle, \quad Y_j = \langle y_1, y_2, \dots, y_j \rangle$$

依赖关系：

$X = \langle x_1, x_2, \dots, x_m \rangle$ ,  $Y = \langle y_1, y_2, \dots, y_n \rangle$ ,  $Z = \langle z_1, z_2, \dots, z_k \rangle$ ,  
 $Z$  为  $X$  和  $Y$  的 LCS， 那么

- (1) 若  $x_m = y_n \Rightarrow z_k = x_m = y_n$ , 且  $Z_{k-1}$  是  $X_{m-1}$  与  $Y_{n-1}$  的 LCS;
- (2) 若  $x_m \neq y_n$ ,  $z_k \neq x_m \Rightarrow Z$  是  $X_{m-1}$  与  $Y$  的 LCS;
- (3) 若  $x_m \neq y_n$ ,  $z_k \neq y_n \Rightarrow Z$  是  $X$  与  $Y_{n-1}$  的 LCS.

满足优化原则和子问题重叠性

# 递推方程、标记函数

令  $X$  与  $Y$  的子序列

$$X_i = \langle x_1, x_2, \dots, x_i \rangle, \quad Y_j = \langle y_1, y_2, \dots, y_j \rangle$$

$C[i, j]$ :  $X_i$  与  $Y_j$  的 LCS 的长度

递推方程

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

标记函数:  $B[i, j]$ , 其值为字符  $\nwarrow$ 、 $\leftarrow$ 、 $\uparrow$ , 分别表示  $C[i, j]$  取得最大值时的三种情况



# 动态规划算法

## 算法3.4 LCS( $X, Y, m, n$ )

1. for  $i \leftarrow 1$  to  $m$  do //行1-4, 边界情况
2.      $C[i, 0] \leftarrow 0$
3. for  $i \leftarrow 1$  to  $n$  do
4.      $C[0, i] \leftarrow 0$
5. for  $i \leftarrow 1$  to  $m$  do
6.     for  $j \leftarrow 1$  to  $n$  do
7.         if  $X[i] = Y[j]$
8.             then  $C[i, j] \leftarrow C[i-1, j-1] + 1$
9.              $B[i, j] \leftarrow \text{“}\nwarrow\text{”}$
10.         else if  $C[i-1, j] \geq C[i, j-1]$
11.             then  $C[i, j] \leftarrow C[i-1, j]$
12.              $B[i, j] \leftarrow \text{“}\uparrow\text{”}$
13.         else  $C[i, j] \leftarrow C[i, j-1]$
14.              $B[i, j] \leftarrow \text{“}\leftarrow\text{”}$

# 利用标记函数构造解

## 算法3.5 Structure Sequence ( $B, i, j$ )

输入:  $B[i, j]$

输出:  $X$  与  $Y$  的最长公共子序列

1. if  $i = 0$  or  $j = 0$  then return //一个序列为空
2. if  $B[i, j] = “↖”$
3. then 输出  $X[i]$
4.     Structure Sequence ( $B, i-1, j-1$ )
5. else if  $B[i, j] = “↑”$  then Structure Sequence ( $B, i-1, j$ )
6.     else Structure Sequence ( $B, i, j-1$ )

# 实例

输入：  $X = \langle A, B, C, B, D, A, B \rangle$ ,  $Y = \langle B, D, C, A, B, A \rangle$ ,

标记函数：

	1	2	3	4	5	6
1	$B[1,1] = \uparrow$	$B[1,2] = \uparrow$	$B[1,3] = \uparrow$	$B[1,4] = \nearrow$	$B[1,5] = \leftarrow$	$B[1,6] = \nearrow$
2	$B[2,1] = \nearrow$	$B[2,2] = \leftarrow$	$B[2,3] = \leftarrow$	$B[2,4] = \uparrow$	$B[2,5] = \nearrow$	$B[2,6] = \leftarrow$
3	$B[3,1] = \uparrow$	$B[3,2] = \uparrow$	$B[3,3] = \nearrow$	$B[3,4] = \leftarrow$	$B[3,5] = \uparrow$	$B[3,6] = \uparrow$
4	$B[4,1] = \nearrow$	$B[4,2] = \uparrow$	$B[4,3] = \uparrow$	$B[4,4] = \uparrow$	$B[4,5] = \nearrow$	$B[4,6] = \leftarrow$
5	$B[5,1] = \uparrow$	$B[5,2] = \nearrow$	$B[5,3] = \uparrow$	$B[5,4] = \uparrow$	$B[5,5] = \uparrow$	$B[5,6] = \uparrow$
6	$B[6,1] = \uparrow$	$B[6,2] = \uparrow$	$B[6,3] = \uparrow$	$B[6,4] = \nearrow$	$B[6,5] = \uparrow$	$B[6,6] = \nearrow$
7	$B[7,1] = \nearrow$	$B[7,2] = \uparrow$	$B[7,3] = \uparrow$	$B[7,4] = \uparrow$	$B[7,5] = \nearrow$	$B[7,6] = \uparrow$

解：  $X[2], X[3], X[4], X[6]$ , 即  $B, C, B, A$

# 复杂度分析

## 算法3.4 LCS( $X, Y, m, n$ )

```
1. for  $i \leftarrow 1$  to  $m$  do
2.    $C[i, 0] \leftarrow 0$ 
3. for  $i \leftarrow 1$  to  $n$  do
4.    $C[0, i] \leftarrow 0$ 
5. for  $i \leftarrow 1$  to  $m$  do
6.   for  $j \leftarrow 1$  to  $n$  do
7.     if  $X[i] = Y[j]$ 
8.       then  $C[i, j] \leftarrow C[i-1, j-1] + 1$ 
9.        $B[i, j] \leftarrow \nwarrow$ 
10.    else if  $C[i-1, j] \geq C[i, j-1]$ 
11.      then  $C[i, j] \leftarrow C[i-1, j]$ 
12.       $B[i, j] \leftarrow \uparrow$ 
13.    else  $C[i, j] \leftarrow C[i, j-1]$ 
14.       $B[i, j] \leftarrow \leftarrow$ 
```

## 算法3.5 Structure Sequence ( $B, i, j$ )

输入:  $B[i, j]$

输出:  $X$  与  $Y$  的最长公共子序列

1. if  $i = 0$  or  $j = 0$  then return //一个序列为空
2. if  $B[i, j] = \nwarrow$
3. then 输出  $X[i]$
4. Structure Sequence ( $B, i-1, j-1$ )
5. else if  $B[i, j] = \uparrow$  then Structure Sequence ( $B, i-1, j$ )
6. else Structure Sequence ( $B, i, j-1$ )

## 算法的计算复杂度

计算优化函数和标记函数: 时间为  $\Theta(mn)$

构造解: 每一步至少缩小  $X$  或  $Y$  的长度, 时间  $\Theta(m+n)$

空间:  $\Theta(mn)$

# LCS问题小结

- 最长公共子序列问题的建模
  - 子问题边界的界定
  - 递推方程及初值，优化原则判定
  - 伪码
  - 标记函数与解的追踪
  - 时间复杂度从蛮力算法 $O(2^{m+n})$ 提升到 $\Theta(mn)$
- 有一个缺点：动态规划求得的解只有一个最长公共子序列，蛮力算法可以求得全部最长公共子序列

## 3.3.4 图像压缩

### 例3.7

像素点灰度值：0~255，表示为8位二进制数

像素点灰度值序列： $\langle p_1, p_2, \dots, p_n \rangle$ ， $p_i$ 为第  $i$  个像素点灰度值。

变位压缩存储格式：将  $\langle p_1, p_2, \dots, p_n \rangle$  分割  $m$  段  $S_1, S_2, \dots, S_m$   
 $i$  段有  $l[i]$  个像素，每个像素  $b[i]$  位， $h_i$  为该段最大像素的位数

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

约束条件：每段像素个数  $l[i] \leq 256$

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

$i$  段占用空间： $b[i] \times l[i] + 11$

问题：给定像素序列  $\langle p_1, p_2, \dots, p_n \rangle$ ，确定最优分段，即

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

# 实例

灰度值序列  $P = \langle 10, 12, 15, 255, 1, 2, 1, 1, 2, 2, 1, 1 \rangle$

分法 1:  $S_1 = \langle 10, 12, 15 \rangle$ ,  $S_2 = \langle 255 \rangle$ ,  $S_3 = \langle 1, 2, 1, 1, 2, 2, 1, 1 \rangle$

分法 2:  $S_1 = \langle 10, 12, 15, 255, 1, 2, 1, 1, 2, 2, 1, 1 \rangle$

分法 3: 分成12组, 每组一个数

段头需存储本段长度、像素位数: 规定每段最长255个像素, 需8位存储; 每像素值范围0~255, 最多8位, 需3位存储。

存储空间

分法 1:  $11 \times 3 + 4 \times 3 + 8 \times 1 + 2 \times 8 = 69$

分法 2:  $11 \times 1 + 8 \times 12 = 107$

分法 3:  $11 \times 12 + 4 \times 3 + 8 \times 1 + 1 \times 5 + 2 \times 3 = 163$

结论: 分法 1 是其中最优的分法

# 算法设计

## 递推方程

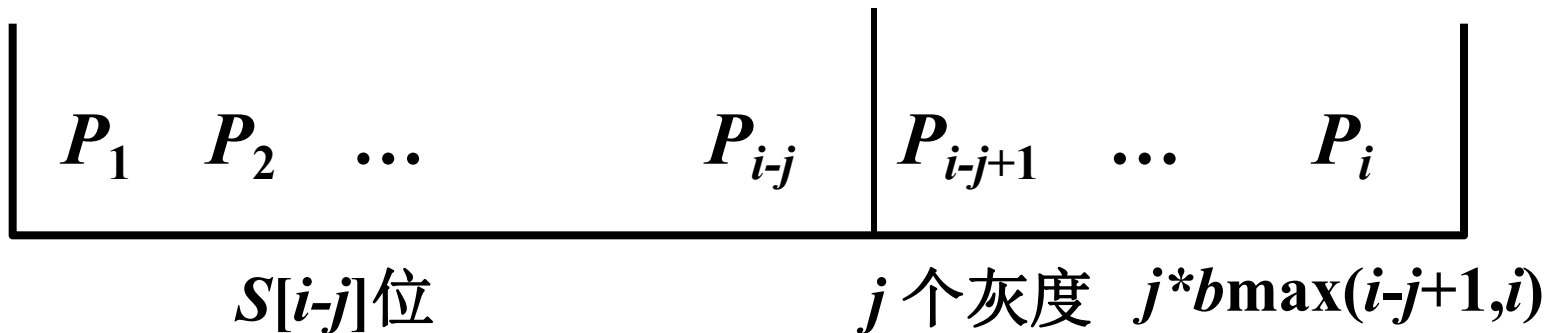
设  $S[i]$  是像素序列  $\langle p_1, p_2, \dots, p_i \rangle$  的最优分段所需存储位数.  
 $j$  为最后一段的长度.

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

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

$$S[0] = 0$$

最后一段像素集记为  $S_m$





# 算法

## 算法3.6 Compress ( $P, n$ )

//计算最小位数  $S[n]$

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

3-6行相当于给 $j=1$ 赋初值,  
即将 $S[i]$ 拆为 $S[i-1]$ 和 $P[i]$

注意第7行 $j$ 从2到 $\min\{l, Lmax\}$ 循环,最多循环256次,不随 $n$ 增大,为常数级别

时间复杂度  $T(n) = O(n)$

# 实例

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

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

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

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

$S[5]=50$

$1 \times 2 + 11$

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

$S[4]=42$

$2 \times 2 + 11$

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

$S[3]=23$

$3 \times 8 + 11$

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

$S[2]=19$

$4 \times 8 + 11$

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

$S[1]=15$

$5 \times 8 + 11$

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

$6 \times 8 + 11$

# 追踪解

## 算法3.7 Traceback( $n, l$ )

输入：数组  $l$            //  $l[i]$  是  $P[i]$  取到最优时最后一段的长度

输出：数组  $C$            //  $C[j]$  是从后向前追踪的第  $j$  段的长度

1.  $j \leftarrow 1$            //  $j$  为正在追踪的段数

2. while  $n \neq 0$  do

3.    $C[j] \leftarrow l[n]$

4.    $n \leftarrow n - l[n]$

5.    $J \leftarrow j + 1$

时间复杂度：  $O(n)$

## 3.3.5 最大子段和

**例3.8** 给定 $n$ 个整数（可以为负数）的序列

$$\langle a_1, a_2, \dots, a_n \rangle$$

求  $\max\{ 0, \max_{1 \leq i \leq j \leq n} \sum_{k=i}^j a_k \}$

实例:  $\langle -2, 11, -4, 13, -5, -2 \rangle$

解: 最大子段和  $a_2 + a_3 + a_4 = 20$

算法1---顺序求和+比较

算法2---分治策略

算法3---动态规划

# 算法1 顺序求和+比较

## 算法3.8 Enumerate

输入：数组  $A[1..n]$

输出：  $sum, first, last$

1.  $sum \leftarrow 0$
2. for  $i \leftarrow 1$  to  $n$  do     //  $i$ 为当前和的首位置
3.   for  $j \leftarrow i$  to  $n$  do     //  $j$ 为当前和的末位置
4.      $thisum \leftarrow 0$      //  $thisum$ 为 $A[i]$ 到 $A[j]$ 之和
5.     for  $k \leftarrow i$  to  $j$  do
6.        $thisum \leftarrow thisum + A[k]$
7.     if  $thisum > sum$
8.       then  $sum \leftarrow thisum$
9.        $first \leftarrow i$      // 记录最大和的首位置
10.       $last \leftarrow j$      // 记录最大和的末位置

时间复杂度：  $O(n^3)$

## 算法2 分治策略

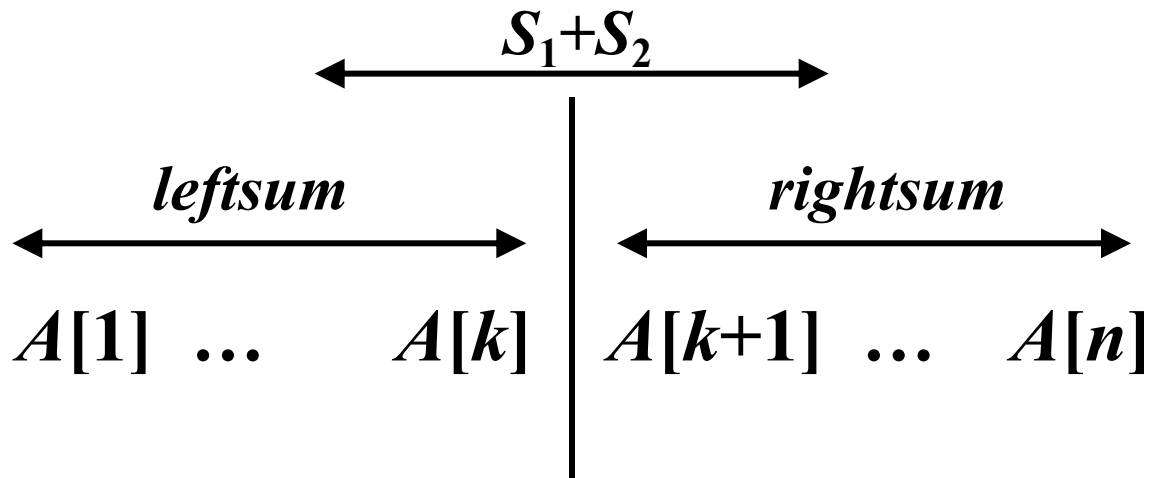
将序列分成左右两半，中间分点 $center$

递归计算左段最大子段和  $leftsum$

递归计算右段最大子段和  $rightsum$

$a_{center} \rightarrow a_1$ 的最大和  $S_1$ ,  $a_{center+1} \rightarrow a_n$ 的最大和  $S_2$

$\max \{ leftsum, rightsum, S_1+S_2 \}$



# 分治算法

## 算法3.9 MaxSubSum( $A, left, right$ )

输入：数组 $A$ ,  $left$ ,  $right$ 分别是 $A$ 的左、右边界

输出： $A$ 的最大子段和 $sum$ 及其子段的前后边界

1. if  $|A| = 1$  then 输出元素值（当值为负时输出0）
2.  $center \leftarrow \lfloor (left+right)/2 \rfloor$
3.  $leftsum \leftarrow \text{MaxSubSum}(A, left, center)$  //子问题 $A_1$
4.  $rightsum \leftarrow \text{MaxSubSum}(A, center+1, right)$  //子问题 $A_2$
5.  $S_1 \leftarrow A_1[left, center]$  //从 $center$ 向左的最大和
6.  $S_2 \leftarrow A_2[center+1, right]$  //从 $center+1$ 向右的最大和
7.  $sum \leftarrow S_1 + S_2$
8. if  $leftsum > sum$  then  $sum \leftarrow leftsum$
9. if  $rightsum > sum$  then  $sum \leftarrow rightsum$

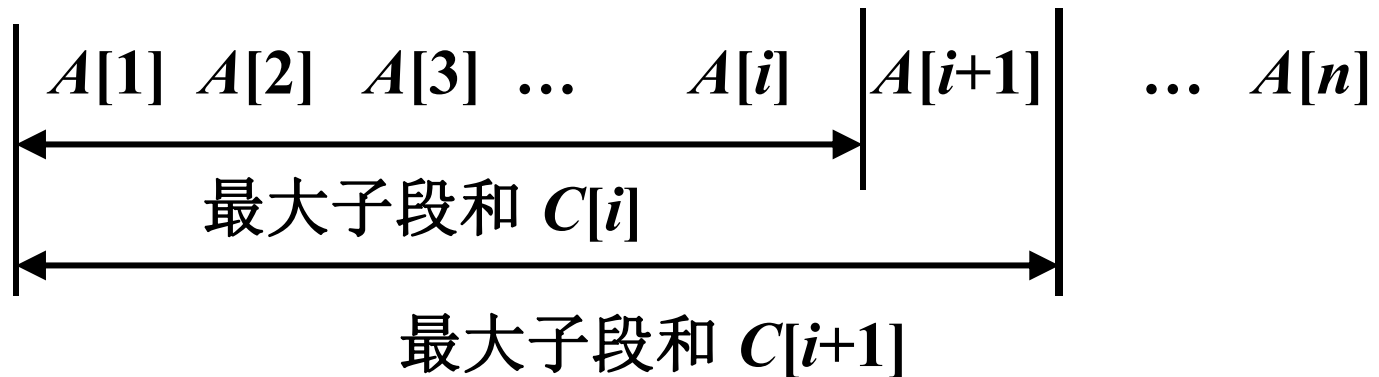
时间： $T(n) = 2T(n/2) + O(n)$ ,  $T(c) = O(1)$

$T(n) = O(n \log n)$

# 算法3： 动态规划

令  $C[i]$  是  $A[1..i]$  中必须包含元素  $A[i]$  的最大子段和

$$C[i] = \max_{1 \leq k \leq i} \left\{ \sum_{j=k}^i A[j] \right\}$$



递推方程:  $C[i+1] = \max\{C[i] + A[i+1], A[i+1]\} \quad i = 1, 2, \dots, n$   
 $C[1] = A[1]$

解:

$$\text{OPT}(A) = \max_{1 \leq i \leq n} \{C[i]\}$$



# 算法 MaxSum

## 算法3.10 MaxSum ( $A, n$ )

输入：数组  $A$

输出：最大子段和  $sum$ , 子段的最后位置  $c$

1.  $sum \leftarrow 0$
2.  $b \leftarrow 0$  //  $b$ 是前一个最大子段和
3. for  $i \leftarrow 1$  to  $n$  do
4.   if  $b > 0$
5.   then  $b \leftarrow b + A[i]$
6.   else  $b \leftarrow A[i]$
7.   if  $b > sum$
8.   then  $sum \leftarrow b$
9.      $c \leftarrow i$  //记录最大和的末项标号
10. return  $sum, c$

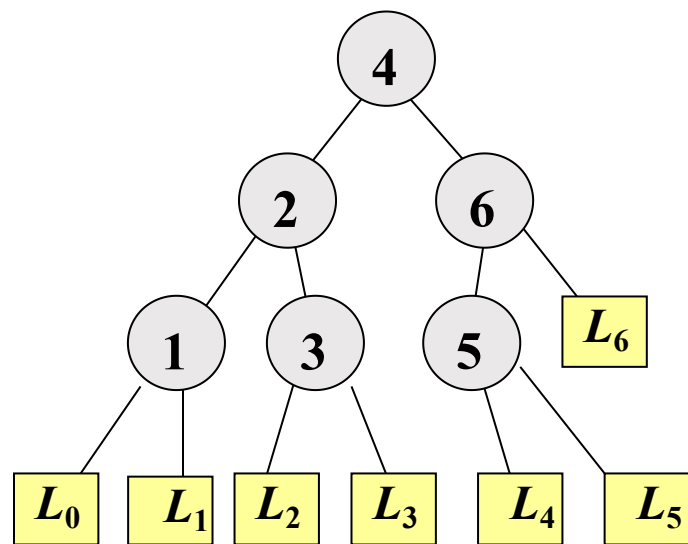
时间复杂度：  $O(n)$ , 空间复杂度：  $O(n)$

## 3.3.6 最优二叉检索树

设集合  $S$  为排序的  $n$  个元素  $x_1 < x_2 < \dots < x_n$ ，将这些元素存储在一棵二叉树的结点上，以查找  $x$  是否在这些数中。如果  $x$  不在，确定  $x$  在哪个空隙。

检索方法：

1. 初始， $x$  与根元素比较；
2.  $x <$  根元素，递归进入左子树；
3.  $x >$  根元素，递归进入右子树；
4.  $x =$  根元素，算法停止，输出  $x$ ；
5.  $x$  到达叶结点，算法停止，输出  $x$  不在数组中。



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

# 存取概率不等情况

空隙:  $(x_0, x_1), (x_1, x_2), \dots, (x_{n-1}, x_n), (x_n, x_{n+1})$ ,  
 $x_0 = -\infty, x_{n+1} = \infty$

给定序列  $S = \langle x_1, x_2, \dots, x_n \rangle$ ,

$x$  在  $x_i$  的概率为  $b_i$ ,  $x$  在  $(x_i, x_{i+1})$  的概率为  $a_i$ ,

$S$  的存取概率分布如下:

$$P = \langle a_0, b_1, a_1, b_2, a_2, \dots, b_n, a_n \rangle$$

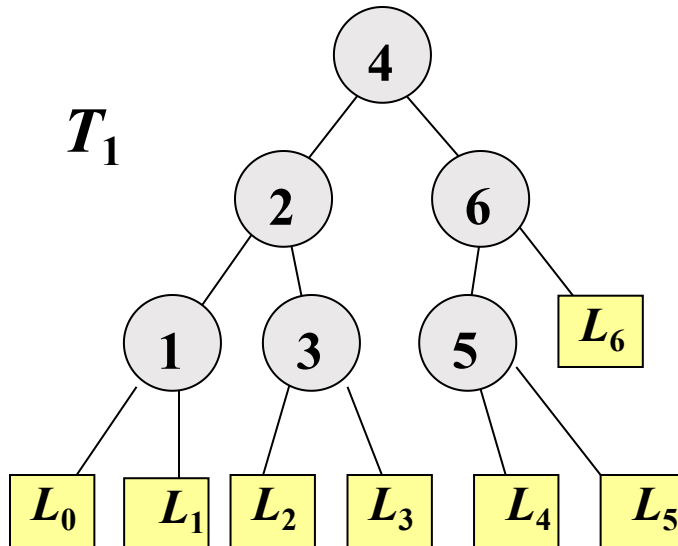
实例

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

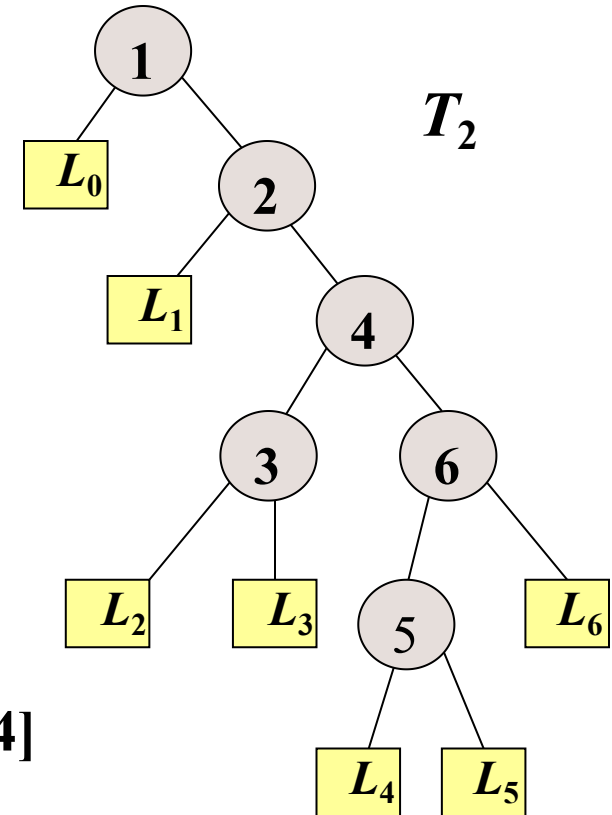
$$P = \langle 0.04, 0.1, 0.01, 0.2, 0.05, 0.2, 0.02, 0.1, 0.02, 0.1, 0.07, 0.05, 0.04 \rangle$$

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

$P=<0.04, \mathbf{0.1}, 0.01, \mathbf{0.2}, 0.05, \mathbf{0.2}, 0.02, \mathbf{0.1}, 0.02, \mathbf{0.1}, 0.07, \mathbf{0.05}, 0.04>$



$$\begin{aligned}
 m(T_1) &= [1*0.1 + 2*(0.2+0.05) + 3*(0.1+0.2+0.1)] \\
 &\quad + [3*(0.04+0.01+0.05+0.02+0.02+0.07) + 2*0.04] \\
 &= 1.8 + 0.71 = \mathbf{2.51}
 \end{aligned}$$



$$\begin{aligned}
 m(T_2) &= [1*0.1 + 2*0.2 + 3*0.1 + 4*(0.2+0.05) + 5*0.1] \\
 &\quad + [1*0.04 + 2*0.01 + 4*(0.05+0.02+0.04) + 5*(0.02+0.07)] \\
 &= 2.3 + 0.95 = \mathbf{3.25}
 \end{aligned}$$

**实例**

# 问 题

**例3.9** 数据集  $S = \langle x_1, x_2, \dots, x_n \rangle$

存取概率分布  $P = \langle a_0, b_1, a_1, b_2, \dots, a_i, b_{i+1}, \dots, b_n, a_n \rangle$

结点  $x_i$  在  $T$  中的深度是  $d(x_i)$ ,  $i = 1, 2, \dots, n$ ,

空隙  $L_j$  的深度为  $d(L_j)$ ,  $j = 0, 1, \dots, n$ ,

平均比较次数为:

$$t = \sum_{i=1}^n b_i (1 + d(x_i)) + \sum_{j=0}^n a_j d(L_j)$$

问题: 给定数据集  $S$  和相关存取概率分布  $P$ , 求一棵最优的 (即平均比较次数最少的) 二分检索树.

# 算法设计:子问题划分

$S[i, j] = \langle x_i, x_{i+1}, \dots, x_j \rangle$  是  $S$  以  $i$  和  $j$  作为边界的子数据集

$P[i, j] = \langle a_{i-1}, b_i, a_i, b_{i+1}, \dots, b_j, a_j \rangle$  是对应  $S[i, j]$  存取概率分布

例:  $S = \langle A, B, C, D, E \rangle$

$P = \langle 0.04, \underline{0.1}, 0.02, \underline{0.3}, 0.02, \underline{0.1}, 0.05, \underline{0.2}, 0.06, \underline{0.1}, 0.01 \rangle$

$S[2, 4] = \langle B, C, D \rangle$

$P[2, 4] = \langle 0.02, \underline{0.3}, 0.02, \underline{0.1}, 0.05, \underline{0.2}, 0.06 \rangle$

子问题划分: 以  $x_k$  作为根, 划分成两个子问题:

$S[i, k-1], P[i, k-1]$

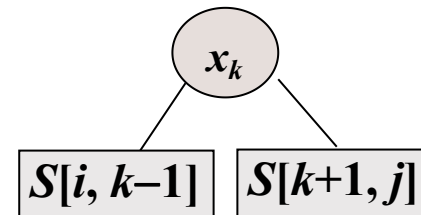
$S[k+1, j], P[k+1, j]$

例: 以  $B$  为根, 划分成以下子问题:

$S[1, 1] = \langle A \rangle, P[1, 1] = \langle 0.04, \underline{0.1}, 0.02 \rangle$

$S[3, 5] = \langle C, D, E \rangle, P[3, 5] = \langle 0.02, \underline{0.1}, 0.05, \underline{0.2}, 0.06, \underline{0.1}, 0.01 \rangle$

# 递推方程



设  $m[i, j]$  是相对于输入  $S[i, j]$  和  $P[i, j]$  的最优二叉搜索树的平均比较次数，令

$$w[i, j] = \sum_{p=i-1}^j a_p + \sum_{q=i}^j b_q$$

是  $P[i, j]$  中所有概率（包括数据元素与空隙）之和

递推方程：

$$m[i, j] = \min_{i \leq k \leq j} \{m[i, k-1] + m[k+1, j] + w[i, j]\}, \quad 1 \leq i \leq j \leq n$$

$$m[i, i-1] = 0, \quad i = 1, 2, \dots, n$$

# 证 明

$m[i, j]_k$ : 根为  $x_k$  时的二分检索树平均比较次数的最小值

$$m[i, j]_k$$

$$= (m[i, k-1] + w[i, k-1]) + (m[k+1, j] + w[k+1, j]) + 1 \times b_k$$

$$= (m[i, k-1] + m[k+1, j]) + (w[i, k-1] + b_k + w[k+1, j])$$

$$= (m[i, k-1] + m[k+1, j]) + \left( \sum_{p=i-1}^{k-1} a_p + \sum_{q=i}^{k-1} b_q \right) + b_k + \left( \sum_{p=k}^j a_p + \sum_{q=k+1}^j b_q \right)$$

$$= (m[i, k-1] + m[k+1, j]) + \sum_{p=i-1}^j a_p + \sum_{q=i}^j b_q$$

$$= m[i, k-1] + m[k+1, j] + w[i, j]$$

平均比较次数: 在所有  $k$  的情况下  $m[i, j]_k$  的最小值,

$$m[i, j] = \min \{ m[i, j]_k \mid i \leq k \leq j \}$$



# 实例

$$m[i, j] = \min_{i \leq k \leq j} \{m[i, k-1] + m[k+1, j] + w[i, j]\} \quad 1 \leq i \leq j \leq n$$

$$m[i, i-1] = 0 \quad i = 1, 2, \dots, n$$

$$m[1, 1] = 0.16, \quad m[3, 5] = 0.88$$

$$m[1, 5]$$

$$= \min_{k=1,2,3,4,5} \{m[1, k-1] + m[k+1, 5]\} + w(1, 5)$$

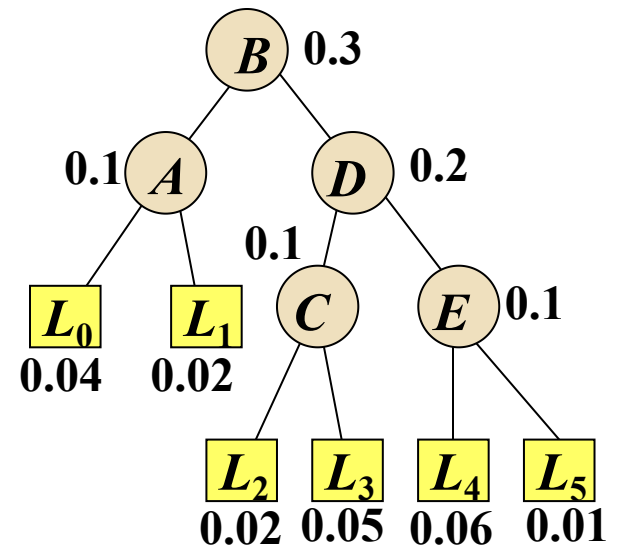
$$= \min\{m[2, 5], m[1, 1] + m[3, 5], m[1, 2] + m[2, 3], m[1, 3] + m[4, 5], m[1, 4]\} + 1$$

$$= \{0.16 + 0.88\}_{k=2} + 1$$

$$= 2.04$$

复杂性估计:

$$T(n) = O(n^3) \quad S(n) = O(n^2)$$



# 3.3.7生物信息学中的动态规划算法

## RNA二级结构预测

一级结构：由字母  $A, C, G, U$  标记的核苷酸构成的一条链

二级结构：核苷酸相互匹配构成二级结构（平面图）

匹配原则：

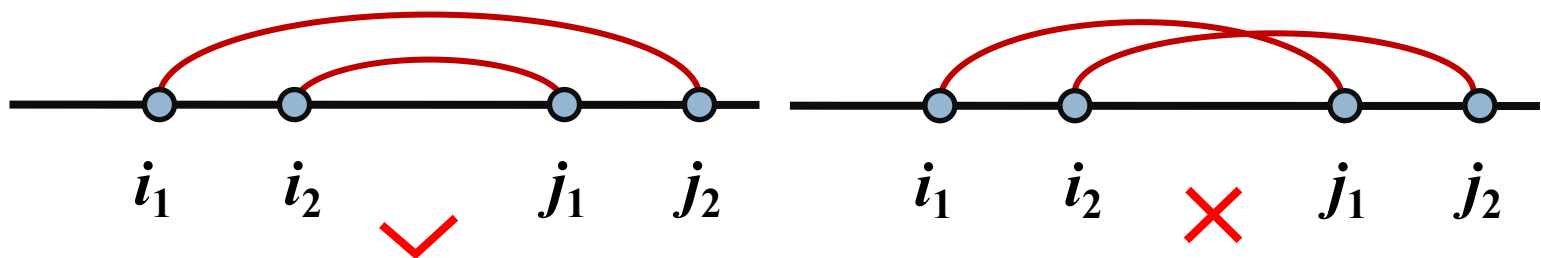
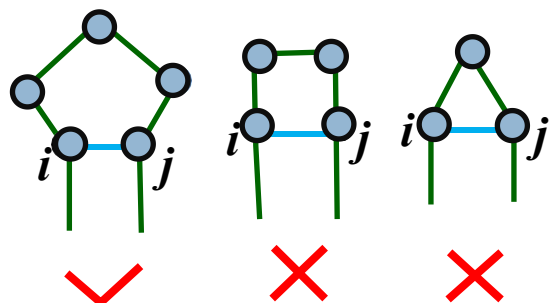
(1) 配对  $U-A, C-G$ ;

(2) 末端不出现“尖角”，位置

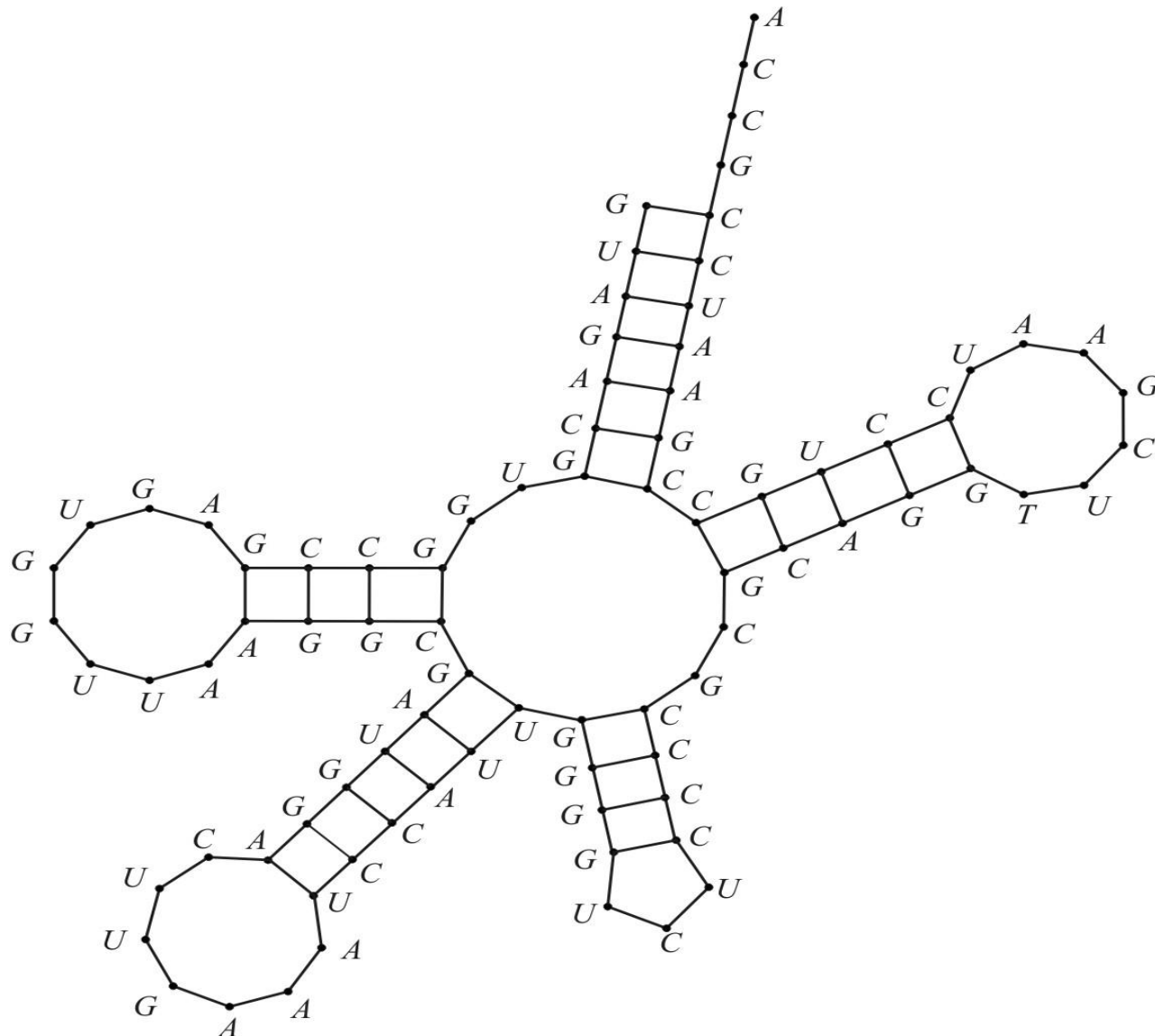
$i-j$  配对，则  $i \leq j-4$ ;

(3) 每个核苷酸只能参加一个配对;

(4) 不允许交叉，即如果位置  $i_1, i_2, j_1, j_2$  满足  $i_1 < i_2 < j_1 < j_2$ ，  
不允许  $i_1-j_1, i_2-j_2$  配对. 但可以允许  $i_1-j_2, i_2-j_1$  配对.

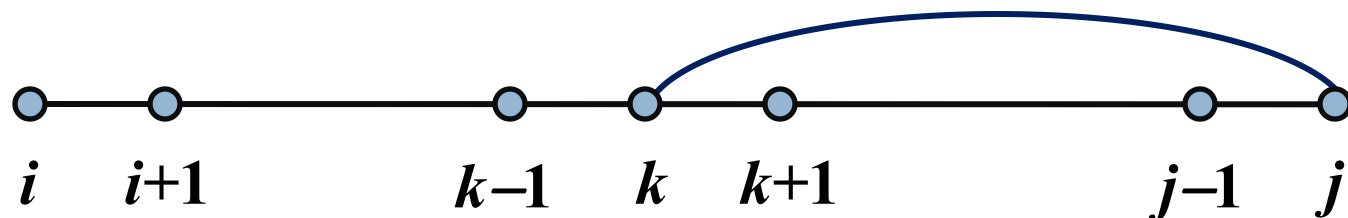


# 实例：4sRNA的二级结构



# 问题与算法设计

**例3.10** 给定RNA的一级结构：由 $A, U, C, G$  构成的长为  $n$  的序列，寻找具有最大匹配对数的二级结构.



令  $C[i, j]$  是序列  $S[i..j]$  的最大匹配对数

$$C[i, j] = \max \{ C[i, j-1], \max_{i \leq k < j-4} \{ 1 + C[i, k-1] + C[k+1, j-1] \} \}$$

$$C[i, j] = 0 \quad j - i < 5$$

算法时间复杂度是  $O(n^3)$

# 序列比对

**例3.11** 编辑距离：给定两个序列  $S_1$  和  $S_2$ ，通过一系列字符编辑（插入、删除、替换）等操作，将  $S_1$  转变成  $S_2$ . 完成这种转换所需要的最少的编辑操作个数称为  $S_1$  和  $S_2$  的编辑距离.

实例：vintner 转变成 writers，编辑距离  $\leq 6$ ：

vintner

删除v: -intner

插入w: **w**intner

插入r: **w****r**intner

删除n: **wri**-tner

删除n: **writ**-er

插入s: **writers**

# 算法设计

$S_1[1..n]$  和  $S_2[1..m]$  表示两个序列

子问题划分:  $S_1[1..i]$  和  $S_2[1..j]$

$C[i, j]$ :  $S_1[1..i]$  和  $S_2[1..j]$  的编辑距离

操作	归约子问题	编辑距离
删除 $S_1[i]$	$(i-1, j)$	+1
$S_1[i]$ 后插入 $S_2[j]$	$(i, j-1)$	+1
$S_1[i]$ 替换为 $S_2[j]$	$(i-1, j-1)$	+1
$S_1[i]=S_2[j]$	$(i-1, j-1)$	+0

$$C[i, j] = \min\{C[i-1, j] + 1, C[i, j-1] + 1, C[i-1, j-1] + t[i, j]\}$$

$$t[i, j] = \begin{cases} 0 & S_1[i] = S_2[j] \\ 1 & S_1[i] \neq S_2[j] \end{cases} \quad i = 1, 2, \dots, n, \quad j = 1, 2, \dots, m$$

$$C[0, j] = j, \quad j = 1, 2, \dots, m$$

$$C[i, 0] = i, \quad i = 1, 2, \dots, n$$

算法的时间复杂度是 $O(nm)$

# 小结

- (1) 引入参数来界定子问题的边界.
- (2) 判断该优化问题是否满足优化原则.
- (3) 注意子问题的重叠程度.
- (4) 给出带边界参数的优化函数定义与优化函数的递推关系  
考虑标记函数. 找到递推关系的初值.
- (5) 采用自底向上的实现技术, 从最小的子问题开始迭代计算, 计算中用备忘录保留优化函数和标记函数的值.
- (6) 利用备忘录和标记函数通过回溯得到最优解.
- (7) 动态规划算法的时间复杂度是对所有子问题的计算工作量求和.
- (8) 动态规划算法一般使用较多的存储空间, 这往往成为限制动态规划算法使用的瓶颈因素.