算法设计与分析
Algorithms Design & Analysis

第八讲：动态规划

动态规划(Dynamic Programming)

- An algorithm design technique (like divide and conquer)(和分治法一样,是一种算法设计技术)
- Divide and conquer(分治法)
  - Partition the problem into independent subproblems (分割成独立的子问题)
  - Solve the subproblems recursively (递归解决子问题)
  - Combine the solutions to solve the original problem (合并求得初始问题的解)

Dynamic Programming

- Applicable when subproblems are not independent (子问题非独立)
  - Subproblems share subsubproblems (子问题求解依赖其子问题的解)
  - A divide and conquer approach would repeatedly solve the common subproblems(分治法通过递归方式解决性质相同的子问题)
  - Dynamic programming solves every subproblem just once and stores the answer in a table (动态规划每次解决一个子问题，并将结果存储在表格中)

动态规划（Dynamic Programming）

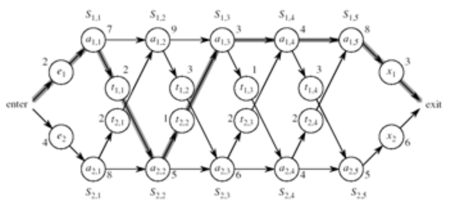
- Used for **optimization problems(适合优化问题)**
  - A set of choices must be made to get an optimal solution (通过适当的选择来获得问题的最优解)
  - Find a solution with the optimal value (minimum or maximum) (找到具有最优解决方案及其最优值：装配线排程方案以及该方案的生产时间)
  - There may be many solutions that return the optimal value: **an optimal solution (导致最优的解决方案可能不止一个)**

动态规划算法(Dynamic Programming Algorithm)

1. Characterize the structure of an optimal solution (描述最优解的结构特征)
2. Recursively define the value of an optimal solution (定义最优解决方案的递归形式)
3. Compute the value of an optimal solution in a bottom-up fashion(以自底向上的方式计算最优解决方案的值)
4. Construct an optimal solution from computed information (从计算信息构造出最优解决方案)

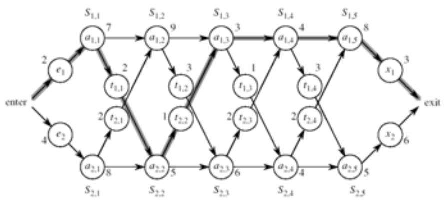
装配线排程问题 (Assembly Line Scheduling)

- Automobile factory with two assembly lines(汽车厂两条装配线)
  - Each line has n stations: $S_{1,1}, \dots, S_{1,n}$ and $S_{2,1}, \dots, S_{2,n}$(每条装配线有n个工序站台)
  - Corresponding stations $S_{1,j}$ and $S_{2,j}$ perform the same function but can take different amounts of time $a_{1,j}$ and $a_{2,j}$ (每条装配线的第j个站台的功能相同,但是效率不一致)
  - Entry times e_1 and e_2 and exit times x_1 and x_2(上线和下线时间)



装配线(Assembly Line)

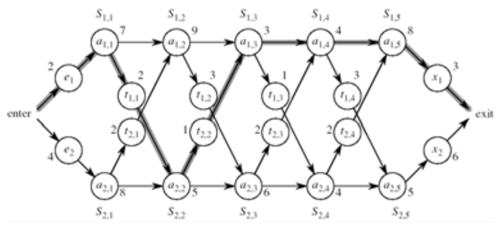
- After going through a station, can either (在一个工序台$S_{i,j}$作业完成之后，汽车可以):
  - stay on same line at no cost, or(立即(时间为0)进入本装配线的下一工序台)
  - transfer to other line: cost after $S_{i,j}$ is $t_{i,j}$, $j = 1, . . . , n - 1$(变换到另一装配线,耗时$t_{i,j}$)



7

装配线排程(Assembly Line Scheduling)

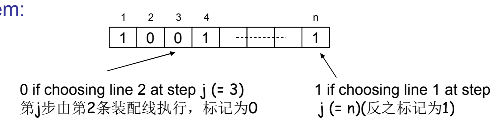
- Problem:
- what stations should be chosen from line 1 and which from line 2 in order to minimize the total time through the factory for one car?(如何充分利用两条装配线,使得组装一辆汽车的时间最短?)



8

解决方法之一(One Solution)

- Brute force(蛮力法)
  - Enumerate all possibilities of selecting stations (计算装配线排程所有可能的组合情况)
  - Compute how long it takes in each case and choose the best one (比较并选择出最短时间的组合)
- Problem:



0 if choosing line 2 at step j (= 3)
第j步由第2条装配线执行，标记为0

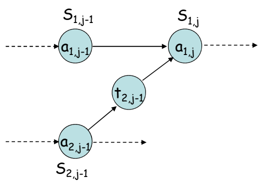
1 if choosing line 1 at step j (= n)(反之标记为1)

  - There are 2^n possible ways to choose stations(共有2^n排程方法)
  - Infeasible when n is large(如果n很大,蛮力法计算将不可接受)

9

1. 构建最优解(Structure of the Optimal Solution)

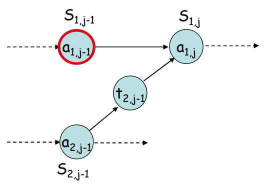
- Let's consider all possible ways to get from the starting point through station $S_{1,j}$ (考虑所有从起点到达$S_{1,j}$可能途径)
  - We have two choices of how to get to $S_{1, j}$:(两种可能)
    - Through $S_{1, j - 1}$, then directly to $S_{1, j}$ (从$S_{1, j - 1}$直接到$S_{1, j}$)
    - Through $S_{2, j - 1}$, then transfer over to $S_{1, j}$ (从$S_{2, j - 1}$ 转换到$S_{1, j}$)



10

1.构建最优解(Structure of the Optimal Solution)

- Suppose that the fastest way through $S_{1, j}$ is through $S_{1, j - 1}$ (如果到达$S_{1, j}$的最快装配路线来自$S_{1, j - 1}$)
  - We must have taken a fastest way from entry through $S_{1, j - 1}$(那么必须是从装配线起点经过$S_{1, j - 1}$的最快装配路线)
  - If there were a faster way through $S_{1, j - 1}$, we would use it instead(因为如果存在更快的经过$S_{1, j - 1}$的装配路线，则可用之替换）
- Similarly for $S_{2, j - 1}$（如果经过$S_{1, j}$的最快装配路线来自$S_{2, j - 1}$, 同样分析）



11

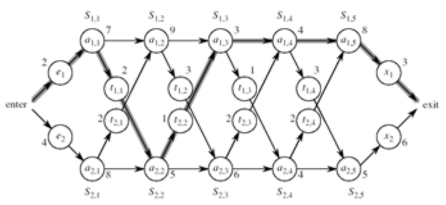
最优化解的结构（Optimal Substructure）

- **Generalization**: an optimal solution to the problem find the fastest way through $S_{1, j}$ contains within it an optimal solution to subproblems: find the fastest way through $S_{1, j - 1}$ or $S_{2, j - 1}$.（寻求从起点到达$S_{1, j}$ 最快装配路线，可分解为寻求从起点经过$S_{1, j - 1}$ or $S_{2, j - 1}$ 最快装配路线问题）
- This is referred to as the **optimal substructure** property（我们将这种具有分解递归特征的解的形式称为最优化结构特征）
- We use this property to construct an optimal solution to a problem from optimal solutions to subproblems（利用这种优化构造特征，从子问题的最优化解获得整个问题的最优化的解）

12

2. 递归解（A Recursive Solution）

- Define the value of an optimal solution in terms of the optimal solution to subproblems（利用子问题的最优解，通过递归的方式求解原问题的最优解）
- Assembly line subproblems（装配线排程子问题）
  - Finding the fastest way through station j on both lines, j = 1, 2, …, n（即是经过两条第j个工作台的最快线路问题， j = 1, 2, …, n ）

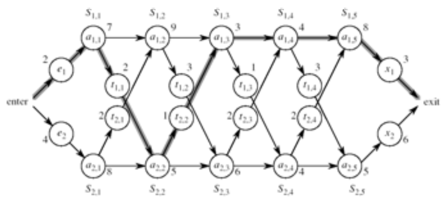


13

2.递归解（A Recursive Solution）

- f^* = the fastest time to get through the entire factory（问题最优解，完成所有装配过程的最短时间）
- $f_i[j]$ = the fastest time to get from the starting point through station $S_{i,j}$（表示从起点经过$S_{i,j}$工序的最短时间）

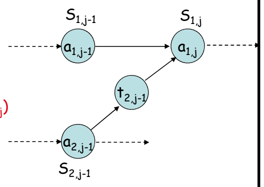
$f^* = \min(f_1[n] + x_1, f_2[n] + x_2)$



14

2.递归解（A Recursive Solution）

- Compute $f_i[j]$ for j = 2, 3, …,n, and i = 1, 2（对于i = 1, 2 和j = 2, 3, …,n ， 计算$f_i[j]$ ）
- Fastest way through $S_{1,j}$ is either:（经过$S_{1,j}$的两种情况）
  - the way through $S_{1,j-1}$ then directly through $S_{1,j}$, or（经过$S_{1,j-1}$ ）
    $f_1[j-1] + a_{1,j}$
  - the way through $S_{2,j-1}$, transfer from line 2 to line 1, then through $S_{1,j}$ （经过$S_{2,j-1}$ ）
    $f_2[j-1] + t_{2,j-1} + a_{1,j}$

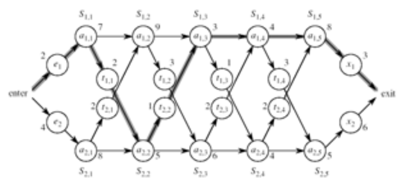


- $f_1[j] = \min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j})$

15

2.递归解（A Recursive Solution）

- $f_i[j]$ = the fastest time to get from the starting point through station $S_{i,j}$ （从起点经过$S_{i,j}$工序的最短时间）
- j = 1 (getting through station 1)（经过工作台1的最短时间）
- $f_1[1] = e_1 + a_{1,1}$
- $f_2[1] = e_2 + a_{2,1}$



16

2.递归解（A Recursive Solution）

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & \text{if } j = 1 \\ \min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}) & \text{if } j \geq 2 \end{cases}$$

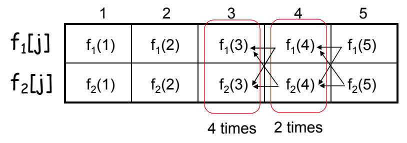
$$f_2[j] = \begin{cases} e_2 + a_{2,1} & \text{if } j = 1 \\ \min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}) & \text{if } j \geq 2 \end{cases}$$

17

3. 计算最优解（Computing the Optimal Solution）

$f^* = \min(f_1[n] + x_1, f_2[n] + x_2)$（自顶向下求最优解）

- $f_1[j] = \min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j})$



	1	2	3	4	5
$f_1[j]$	$f_1(1)$	$f_1(2)$	$f_1(3)$	$f_1(4)$	$f_1(5)$
$f_2[j]$	$f_2(1)$	$f_2(2)$	$f_2(3)$	$f_2(4)$	$f_2(5)$
			4 times	2 times	

- Solving top-down would result in exponential running time（自顶向下导致指数增长的计算时间）

18

3.计算最优解（Computing the Optimal Solution）

- For $j \geq 2$, each value $f_i[j]$ depends only on the values of $f_1[j - 1]$ and $f_2[j - 1]$（对于$j \geq 2$，计算$f_i[j]$与$f_1[j - 1]$和$f_2[j - 1]$的值相关）
- Compute the values of $f_i[j]$（如何计算$f_i[j]$?）
  - in increasing order of j（ j递增）

increasing j $\longrightarrow$

	1	2	3	4	5
$f_1[j]$					
$f_2[j]$					

- Bottom-up approach（自底向上的方法）
  - First find optimal solutions to subproblems（求解子问题的解）
  - Find an optimal solution to the problem from the subproblems（从子问题的解构造出原问题的最优解）

19

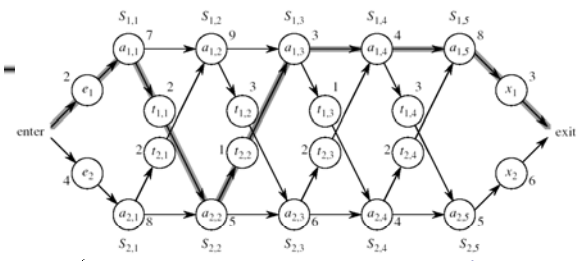
4、构造最优方案(Construct the Optimal Solution)

- We need the sequence of what line has been used at each station（表示哪条装配线哪些工序台被利用的序列）
  - $l_i[j]$ – the line number (1, 2) whose station (j - 1) has been used to get in fastest time through $S_{i,j}$, j = 2, 3, …, n（ $l_i[j]$表示经过$S_{i,j}$的最优排程序列）
  - l* - the line whose station n is used to get in the fastest way through the entire factory（ l*表示整个排程问题的最优序列）

increasing j $\longrightarrow$

	2	3	4	5
$l_1[j]$				
$l_2[j]$				

20



$$f_1[j] = \begin{cases} e_1 + a_{1,1}, & \text{if } j = 1 \\ \min(f_1[j - 1] + a_{1,j}, f_2[j - 1] + t_{2,j-1} + a_{1,j}) & \text{if } j \geq 2 \end{cases}$$

	1	2	3	4	5
$f_1[j]$	9	18[1]	20[2]	24[1]	32[1]
$f_2[j]$	12	16[1]	22[2]	25[1]	30[2]

f* = 35[1]

21

最优排程算法：FASTEST-WAY(a, t, e, x, n)

```
1.  $f_1[1] \leftarrow e_1 + a_{1,1}$
2.  $f_2[1] \leftarrow e_2 + a_{2,1}$
        } Compute initial values of f1 and f2（初值计算）
3.  for $j \leftarrow 2$ to n
4.      do if $f_1[j - 1] + a_{1,j} \leq f_2[j - 1] + t_{2,j-1} + a_{1,j}$
5.              then $f_1[j] \leftarrow f_1[j - 1] + a_{1,j}$
6.                   $l_1[j] \leftarrow 1$
7.              else $f_1[j] \leftarrow f_2[j - 1] + t_{2,j-1} + a_{1,j}$
8.                   $l_1[j] \leftarrow 2$
        } Compute the values of f1[j] and l1[j]
9.          if $f_2[j - 1] + a_{2,j} \leq f_1[j - 1] + t_{1,j-1} + a_{2,j}$
10.             then $f_2[j] \leftarrow f_2[j - 1] + a_{2,j}$
11.                  $l_2[j] \leftarrow 2$
12.             else $f_2[j] \leftarrow f_1[j - 1] + t_{1,j-1} + a_{2,j}$
13.                  $l_2[j] \leftarrow 1$
        } Compute the values of f2[j] and l2[j]
```

22

FASTEST-WAY(a, t, e, x, n) (cont.)

```
14. if $f_1[n] + x_1 \leq f_2[n] + x_2$
15.     then $f^* = f_1[n] + x_1$
16.          l* = 1
17.     else $f^* = f_2[n] + x_2$
18.          l* = 2
    } Compute the values of the fastest time through the entire factory（最终的最优解）
```

23

4. 构造最优的排程方案（Construct an Optimal Solution）

- Alg.: PRINT-STATIONS(l, n)
- i ← l*
- print “line ” i “, station ” n
- **for** j ← n **downto** 2
- **do** i ← $l_i[j]$
- print “line ” i “, station ” j - 1

line 1, station 5
line 1, station 4
line 1, station 3
line 2, station 2
line 1, station 1

	1	2	3	4	5
$f_1[j]/l_1[j]$	9	18[1]	20[2]	24[1]	32[1]
$f_2[j]/l_2[j]$	12	16[1]	22[2]	25[1]	30[2]

l* = 1

24

动态规划算法(Dynamic Programming Algorithm)

1. Characterize the structure of an optimal solution（最优解的结构特征）
   – Fastest time through a station depends on the fastest time on previous stations（比如，通过某个工作台的最快路线与通过前一工作台的最快路线相关）
2. Recursively define the value of an optimal solution（递归表示）
   – $f_1[j] = \min(f_1[j - 1] + a_{1,j}, f_2[j -1] + t_{2,j-1} + a_{1,j})$
3. Compute the value of an optimal solution in a bottom-up fashion（计算最优值，自底向上）
   – Fill in the fastest time table in increasing order of j (station #)
4. Construct an optimal solution from computed information（构造导致最优解的最佳方案，依赖求解过程的某些计算信息）
   – Use an additional table to help reconstruct the optimal solution

25

矩阵链相乘(Matrix-Chain Multiplication)

Problem: given a sequence $\langle A_1, A_2, \ldots, A_n\rangle$, compute the product:(问题：给定矩阵序列$A_1, A_2, \ldots, A_n$， 求它们的积）

$A_1 \cdot A_2 \cdots A_n$

- Matrix compatibility:（矩阵相乘的条件）

  $C = A \cdot B$

  $col_A = row_B$

  $row_C = row_A$

  $col_C = col_B$

  $A_1 \cdot A_2 \cdots A_i \cdot A_{i+1} \cdots A_n$

  $col_i = row_{i+1}$

26

矩阵链相乘(Matrix-Chain Multiplication)

- In what order should we multiply the matrices?(矩阵链相乘将按照什么顺序进行?)

  $A_1 \cdot A_2 \cdots A_n$
- Parenthesize the product to get the order in which matrices are multiplied(用括号表示出矩阵链相乘的顺序)
- E.g.: $A_1 \cdot A_2 \cdot A_3 = ((A_1 \cdot A_2) \cdot A_3)$

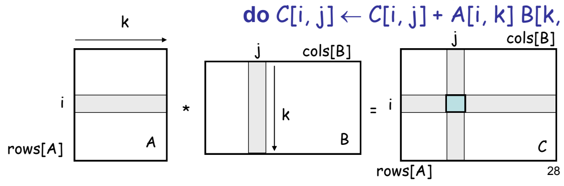
  $= (A_1 \cdot (A_2 \cdot A_3))$
- Which one of these orderings should we choose?(那种顺序最优?)
  – The order in which we multiply the matrices has a significant impact on the cost of evaluating the product(矩阵链相乘的顺序极大的影响计算的代价)

27

矩阵相乘:MATRIX-MULTIPLY(A, B)

```
if columns[A] ≠ rows[B]
  then error "incompatible dimensions"
  else for i ← 1 to rows[A]
          do for j ← 1 to columns[B]
               do C[i, j] = 0
                  for k ← 1 to columns[A]
                      do C[i, j] ← C[i, j] + A[i, k] B[k, j]
```

$\underbrace{A[i, k]\, B[k, j]}_{}$ rows[A] · cols[A] · cols[B] multiplications



28

矩阵链相乘示例(Example)

$A_1 \cdot A_2 \cdot A_3$

- A_1: 10 x 100
- A_2: 100 x 5
- A_3: 5 x 50

1. $((A_1 \cdot A_2) \cdot A_3)$: $A_1 \cdot A_2 = 10 \times 100 \times 5 = 5{,}000$ (10 x 5)

   $((A_1 \cdot A_2) \cdot A_3) = 10 \times 5 \times 50 = 2{,}500$

   Total: 7,500 scalar multiplications
2. $(A_1 \cdot (A_2 \cdot A_3))$: $A_2 \cdot A_3 = 100 \times 5 \times 50 = 25{,}000$ (100 x 50)

   $(A_1 \cdot (A_2 \cdot A_3)) = 10 \times 100 \times 50 = 50{,}000$

   Total: 75,000 scalar multiplications

one order of magnitude difference!!(数量级区别)

29

矩阵链相乘(Matrix-Chain Multiplication)

- Given a chain of matrices $\langle A_1, A_2, \ldots, A_n\rangle$, where for i = 1, 2, …, n matrix A_i has dimensions $p_{i-1} \times p_i$, fully parenthesize the product $A_1 \cdot A_2 \cdots A_n$ in a way that minimizes the number of scalar multiplications.(如何决定矩阵链相乘的顺序，即如何放置括号，使矩阵链相乘所需要的数量乘法的次数最小）

$$\begin{array}{ccccccccc} A_1 & \cdot & A_2 & \cdots & A_i & \cdot & A_{i+1} & \cdots & A_n \\ p_0 \times p_1 & & p_1 \times p_2 & & p_{i-1} \times p_i & & p_i \times p_{i+1} & & p_{n-1} \times p_n \end{array}$$

30

蛮力法(Brute Force)

- Brute force: check all possible orders?(逐一比较)
  - *P*(*n*): number of ways to multiply *n* matrices. (n长度矩阵链相乘的可能方法)

$$P(n) = \begin{cases} 1 & \text{if } n = 1, \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2. \end{cases}$$

  - $P(n) = \Omega\left(\frac{4^n}{n^{3/2}}\right)$, **exponential** in *n*.
- Any efficient solution? Dynamic programming! (高效率方法)

1. 最优矩阵链相乘顺序结构(The Structure of an Optimal Parenthesization)

- Notation:(标记$A_{i\ldots j}$)

  $A_{i\ldots j} = A_i A_{i+1} \cdots A_j$, $i \leq j$

- For i < j:

  $A_{i\ldots j} = A_i A_{i+1} \cdots A_j$

  $= A_i A_{i+1} \cdots A_k A_{k+1} \cdots A_j$

  $= A_{i\ldots k} A_{k+1\ldots j}$

- Suppose that an optimal parenthesization of $A_{i\ldots j}$ splits the product between A_k and A_{k+1}, where $i \leq k < j$(假设矩阵链$A_{i\ldots j}$相乘的最优顺序在A_k和A_{k+1}分割,即)

Optimal Substructure

$A_{i\ldots j} = A_{i\ldots k} A_{k+1\ldots j}$

- The parenthesization of the “prefix” $A_{i\ldots k}$ must be an optimal parenthesization(则$A_{i\ldots k}$必须是具有最优相乘顺序的矩阵链)
- If there were a less costly way to parenthesize $A_{i\ldots k}$, we could substitute that one in the parenthesization of $A_{i\ldots j}$ and produce a parenthesization with a lower cost than the optimum ⇒ contradiction!(否则,可以用更优顺序取代， $A_{k+1\ldots j}$同样)
- An optimal solution to an instance of the matrix-chain multiplication contains within it optimal solutions to subproblems(这样,就将原矩阵链$A_{i\ldots j}$最优相乘顺序问题转变为子矩阵序列$A_{i\ldots k}$、$A_{k+1\ldots j}$的最优相乘顺序问题)

2. 递归解(A Recursive Solution)

- Subproblem:

  determine the minimum cost of parenthesizing (求$A_{i\ldots j}$数量乘法的次数最小的运算顺序)

  $A_{i\ldots j} = A_i A_{i+1} \cdots A_j$ for $1 \leq i \leq j \leq n$

- Let m[i, j] = the minimum number of multiplications needed to compute $A_{i\ldots j}$ (利用m[i, j]标记$A_{i\ldots j}$最小的数量乘法次数)
  - Full problem ($A_{1\ldots n}$): m[1, n]
  - i = j: $A_{i\ldots i} = A_i$ ⇒ m[i, i] = 0, for i = 1, 2, …, n

2.递归解(A Recursive Solution)

- Consider the subproblem of parenthesizing(矩阵链$A_{i\ldots j}$相乘在A_k和A_{k+1}分割)

  $A_{i\ldots j} = A_i A_{i+1} \cdots A_j$ for $1 \leq i \leq j \leq n$

  $$= \overset{p_{i-1}p_kp_j}{\underbrace{A_{i\ldots k}}_{m[i,\,k]}\underbrace{A_{k+1\ldots j}}_{m[k+1,j]}} \quad \text{for } i \leq k < j$$

- Assume that the optimal parenthesization splits the product $A_i A_{i+1} \cdots A_j$ at k ($i \leq k < j$)(在A_k和A_{k+1}分割如果是最优计算顺序)

$$m[i, j] = \underbrace{m[i, k]}_{\substack{\text{min \# of multiplications} \\ \text{to compute } A_{i\ldots k}}} + \underbrace{m[k+1, j]}_{\substack{\text{min \# of multiplications} \\ \text{to compute } A_{k+1\ldots j}}} + \underbrace{p_{i-1}p_kp_j}_{\substack{\text{\# of multiplications} \\ \text{to compute } A_{i\ldots k}A_{k\ldots j}}}$$

2. A Recursive Solution (cont.)

$m[i, j] = m[i, k] + m[k+1, j] + p_{i-1}p_kp_j$

- We do not know the value of k(如何确定k?k有j – i 个选择情况)
  - There are j – i possible values for k: k = i, i+1, …, j-1
- Minimizing the cost of parenthesizing the product $A_i A_{i+1} \cdots A_j$ becomes:(最小数量乘法次数为:)

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

矩阵链相乘(Matrix-Chain Multiplication)

– $m[1, n]$: the cheapest cost to compute $A_{1..n}$.

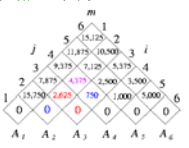
$$m[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & \text{if } i < j. \end{cases}$$

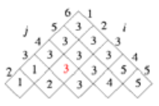
- Applicability of dynamic programming(动态规划的适用性)
  - **Optimal substructure:** an optimal solution contains within it optimal solutions to subproblems.(优化子结构)
  - **Overlapping subproblem:** a recursive algorithm revisits the same problem over and over again; only $\theta(n^2)$ subproblems.(迭代的解)

自底向上动态规划矩阵链相乘最优顺序算法(Bottom-Up DP Matrix-Chain Order)

```
Matrix-Chain-Order(p)
1. n ← length[p]-1;
2. for i ← 1 to n
3.    m[i, i] ← 0;
4. for  l ← 2 to n
5.    for i ← 1 to n – l +1
6.       j ← i + l -1;
7.       m[i, j] ← ∞;
8.       for k ← i  to j -1
9.          q ← m[i, k] + m[k+1, j] + p_{l-1}p_k p_j;
10.         if q < m[i, j]
11.            m[i, j] ← q;
12.            s[i, j] ← k;
13. return m and s
```

matrix	dimension
A_1	30 * 35
A_2	35 * 15
A_3	15 * 5
A_4	5 * 10
A_5	10 * 20
A_6	20 * 25





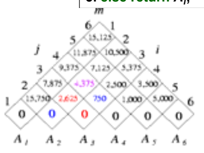
$$m[2,4] = \min \begin{cases} m[2,2] + m[3,4] + p_1p_2p_4 = 0 + 750 + 35 \times 15 \times 10 = 6000. \\ m[2,3] + m[4,4] + p_1p_3p_4 = 2625 + 0 + 35 \times 5 \times 10 = 4375. \end{cases}$$

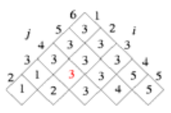
构造最优相乘顺序(Constructing an Optimal Solution)

- $s[i, j]$: value of k such that the optimal parenthesization of $A_i A_{i+1} \dots A_j$ splits between A_k and A_{k+1}.($s[i, j]$:记录了$A_i A_{i+1} \dots A_j$ 的最优分割顺序位置)
- Optimal matrix $A_{1..n}$ multiplication: $A_{1..s[1, n]}A_{s[1, n] + 1..n}$.
- **Exp:** call Matrix-Chain-Multiply(A, s, 1, 6): $((A_1 (A_2 A_3))((A_4 A_5) A_6))$.

```
Matrix-Chain-Multiply(A, s, i, j)
1. if j > i
2.   X ← Matrix-Chain-Multiply(A, s, i, s[i, j]);
3.   Y ← Matrix-Chain-Multiply(A, s, s[i, j]+1, j);
4.    return Matrix-Multiply(X, Y);
5. else return A_i;
```

matrix	dimension
A_1	30 * 35
A_2	35 * 15
A_3	15 * 5
A_4	5 * 10
A_5	10 * 20
A_6	20 * 25





最长共同子序列(Longest Common Subsequence)

- The sequence Z = (B, C, A) is a ***subsequence*** of
- X = (A, **B**, **C**, B, D, **A**, B). (子序列的概念)
- It is also a subsequence of Y = (**B**, D, **C**, **A**, B, A).
- It is a ***common subsequence*** of X and Y. (共同子序列)
- It is not a ***longest common subsequence*** (最长共同子序列) because Z' = (B, D, A, B) is a longer common subsequence.

最长共同子序列例

- **Exp:** X = <a, b, c, b, d, a, b> and
  Y = <b, d, c, a, b, a>
  LCS = <b, c, b, a> (also, LCS = <b, d, a, b>).
- Exp: DNA sequencing:
  – S1= ACCGGTCGAGATGCAG;
  – S2 = GTCGTTCGGAATGCAT;

  LCS S3 = GTCGGATGCA

蛮力法求LCS

- Brute-force method:
  – Enumerate all subsequences of X and check if they appear in Y.(穷举X的所有子序列，检查其是否在Y中出现，然后选出LCS）

  – $X = (x_1, x_2, \dots, x_m)$ 有2^m 个子序列。

LCS的最优结构（Optimal Substructure of LCS）

- Given two sequences $X = (x_1, x_2, \ldots, x_m)$ and $Y = (y_1, y_2, \ldots, y_n)$ and an LCS $Z = (z_1, z_2, \ldots, z_k)$ of X and Y.(给定两个序列$X = (x_1, x_2, \ldots, x_m)$和$Y = (y_1, y_2, \ldots, y_n)$他们的LCS $Z = (z_1, z_2, \ldots, z_k)$)
- If $x_m = y_n$, then $z_k = x_m$ andZ_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
- (如果$x_m = y_n$, 则有$z_k = x_m$, 和 Z_{k-1}是X_{m-1}和Y_{n-1}的LCS)
- If $x_m \neq y_n$, then $z_k \neq x_m$ implies Z is an LCS of X_{m-1} and Y.
- (如果$x_m \neq y_n$, 则 $z_k \neq x_m$意味Z是X_{m-1}和Y的LCS)
- If $x_m \neq y_n$, then $z_k \neq y_n$ implies Z is an LCS of X and Y_{n-1}.
- (如果$x_m \neq y_n$, 则 $z_k \neq y_n$ 意味Z是X和Y_{n-1} 的LCS)

LCS递归解(A Recursive Formulation of LCS)

- C = length of LCS of X and Y.（C： X 和Y最长共同子序列的长度）
- $c(i, j)$ = length of LCS of X_i and Y_j;（ c(i, j)：X_i 和 Y_j最长共同子序列的长度）
- that is, C = c(m, n).

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

LCS算法

- To compute $c[i, j]$, we need $c[i\text{-}1, j\text{-}1]$, $c[i\text{-}1, j]$, and $c[i, j\text{-}1]$.
- $b[i, j]$: points to the table entry w.r.t. the optimal subproblem solution chosen when computing $c[i, j]$.（ $b[i, j]$:记录求解过程信息）

```
LCS-Length(X,Y)
1. m ← length[X];
2. n ← length[Y];
3. for i ← 1 to m
4.     c[i, 0] ← 0;
5. for j ← 0 to n
6.     c[0, j] ← 0;
7. for i ← 1 to m
8.     for j ← 1 to n
9.         if x_i = y_j
10.            c[i, j] ← c[i-1, j-1]+1
11.            b[i, j] ← “ ↖”
12.        else if  c[i-1,j] ≥ c[i, j-1]
13.                c[i,j] ← c[i-1, j]
14.                b[i, j] ← `` ↑ "
15.            else c[i, j] ← c[i, j-1]
16.                b[i, j] ← “ ← “
17. return c and b
```

算法分析

- it simply fills in the table.（填表）
- Computing one table entry costs $O(1)$ time.（一个格子的计算量）
- There are $n \cdot m$ table entries.（ $n \cdot m$ 个格子）
- The cost of the algorithm is $O(nm)$.（总计算开销）

Demonstration

	j	0	1	2	3	4	5	6	7	8	9	10
i		y_j	a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0										
2	p	0										
3	a	0										
4	n	0										
5	k	0										
6	i	0										
7	n	0										
8	g	0										

Demonstration

	j	0	1	2	3	4	5	6	7	8	9	10
i		y_j	a	m	p	u	t	a	t	i	o	n
0	x_i		0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑
1	s	0	0	0	0	0	0	0	0	0	0	0
2	p	0										
3	a	0										
4	n	0										
5	k	0										
6	i	0										
7	n	0										
8	g	0										

Demonstration

	j	0	1	2	3	4	5	6	7	8	9	10
i	y_j		a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0	0	0	0	0	0	0	0	0	0	0
2	p	0	0	0	1	1	1	1	1	1	1	1
3	a	0										
4	n	0										
5	k	0										
6	i	0										
7	n	0										
8	g	0										

49

Demonstration

	j	0	1	2	3	4	5	6	7	8	9	10
i	y_j		a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0	0	0	0	0	0	0	0	0	0	0
2	p	0	0	0	1	1	1	1	1	1	1	1
3	a	0	1	1	1	1	1	2	2	2	2	2
4	n	0										
5	k	0										
6	i	0										
7	n	0										
8	g	0										

50

Demonstration

	j	0	1	2	3	4	5	6	7	8	9	10
i	y_j		a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0	0	0	0	0	0	0	0	0	0	0
2	p	0	0	0	1	1	1	1	1	1	1	1
3	a	0	1	1	1	1	1	2	2	2	2	2
4	n	0	1	1	1	1	1	2	2	2	2	3
5	k	0										
6	i	0										
7	n	0										
8	g	0										

51

Demonstration

	j	0	1	2	3	4	5	6	7	8	9	10
i	y_j		a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0	0	0	0	0	0	0	0	0	0	0
2	p	0	0	0	1	1	1	1	1	1	1	1
3	a	0	1	1	1	1	1	2	2	2	2	2
4	n	0	1	1	1	1	1	2	2	2	2	3
5	k	0	1	1	1	1	1	2	2	2	2	3
6	i	0										
7	n	0										
8	g	0										

52

Demonstration

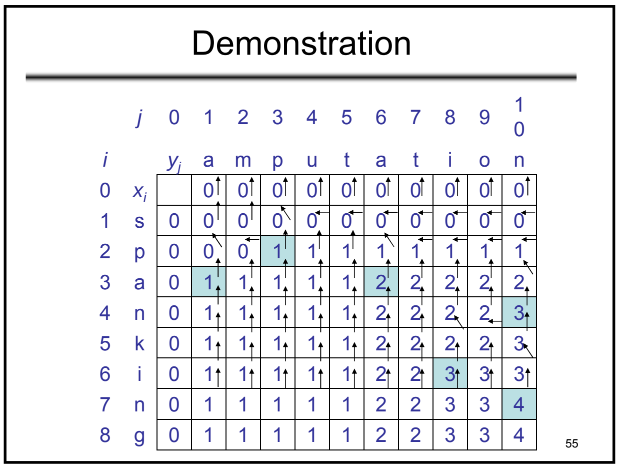
	j	0	1	2	3	4	5	6	7	8	9	10
i	y_j		a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0	0	0	0	0	0	0	0	0	0	0
2	p	0	0	0	1	1	1	1	1	1	1	1
3	a	0	1	1	1	1	1	2	2	2	2	2
4	n	0	1	1	1	1	1	2	2	2	2	3
5	k	0	1	1	1	1	1	2	2	2	2	3
6	i	0	1	1	1	1	1	2	2	3	3	3
7	n	0										
8	g	0										

53

Demonstration

	j	0	1	2	3	4	5	6	7	8	9	10
i	y_j		a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0	0	0	0	0	0	0	0	0	0	0
2	p	0	0	0	1	1	1	1	1	1	1	1
3	a	0	1	1	1	1	1	2	2	2	2	2
4	n	0	1	1	1	1	1	2	2	2	2	3
5	k	0	1	1	1	1	1	2	2	2	2	3
6	i	0	1	1	1	1	1	2	2	3	3	3
7	n	0	1	1	1	1	1	2	2	3	3	4
8	g	0										

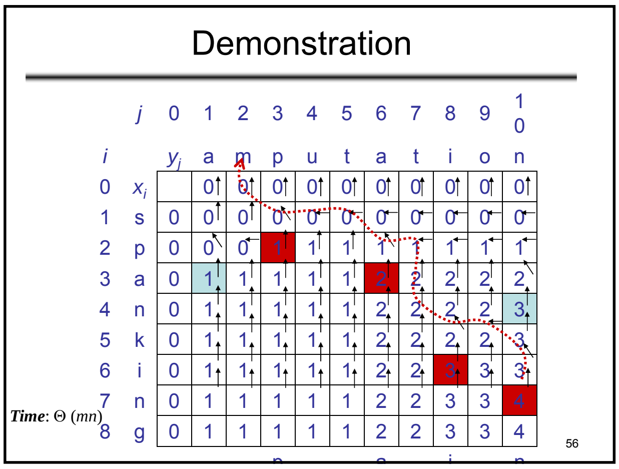
54



Demonstration

	j	0	1	2	3	4	5	6	7	8	9	10
i		y_j	a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0	0	0	0	0	0	0	0	0	0	0
2	p	0	0	0	1	1	1	1	1	1	1	1
3	a	0	1	1	1	1	1	2	2	2	2	2
4	n	0	1	1	1	1	1	2	2	2	2	3
5	k	0	1	1	1	1	1	2	2	2	2	3
6	i	0	1	1	1	1	1	2	2	3	3	3
7	n	0	1	1	1	1	1	2	2	3	3	4
8	g	0	1	1	1	1	1	2	2	3	3	4

55



Demonstration

	j	0	1	2	3	4	5	6	7	8	9	10
i		y_j	a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0	0	0	0	0	0	0	0	0	0	0
2	p	0	0	0	1	1	1	1	1	1	1	1
3	a	0	1	1	1	1	1	2	2	2	2	2
4	n	0	1	1	1	1	1	2	2	2	2	3
5	k	0	1	1	1	1	1	2	2	2	2	3
6	i	0	1	1	1	1	1	2	2	3	3	3
7	n	0	1	1	1	1	1	2	2	3	3	4
8	g	0	1	1	1	1	1	2	2	3	3	4

Time: $\Theta(mn)$

p a i n

56