Design and Analysis of Algorithms

Dynamic Programming

Reference:
CLRS Chapter 15

Topics:

- **Dynamic Programming (DP) paradigm**
- **Assembly-Line Scheduling**
- **Matrix-Chain Multiplication**

Optimization Problems

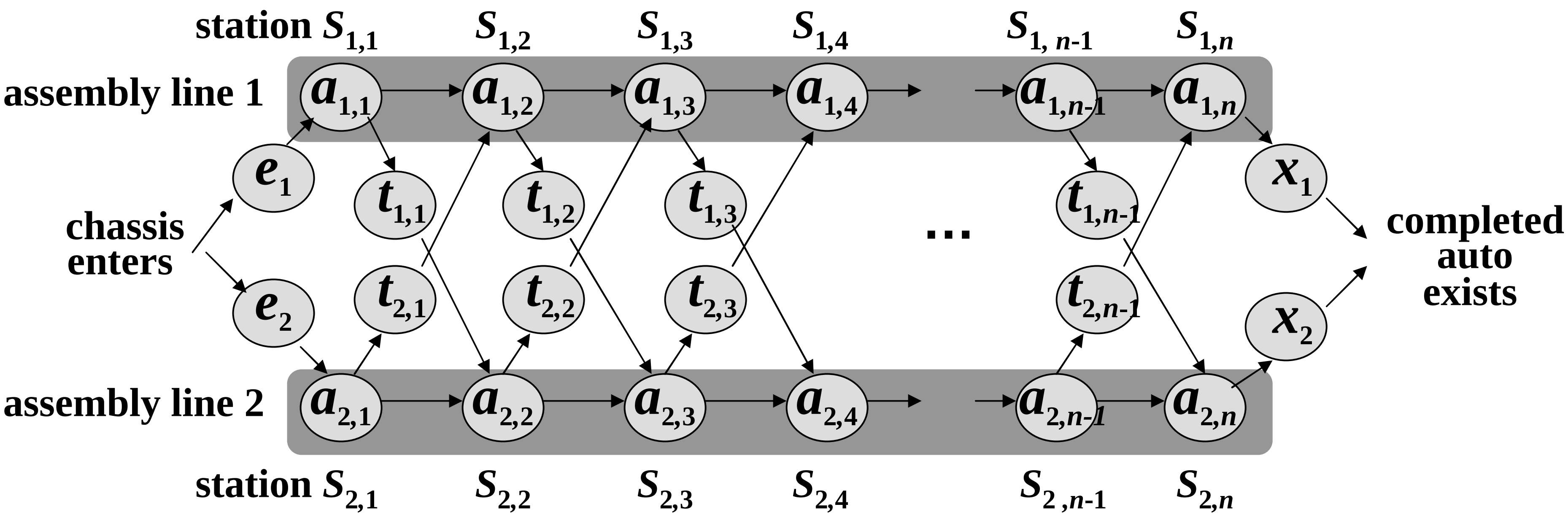
- A design technique, like divide-and-conquer.
- Works bottom-up rather then top-down.
- Useful for optimization problems.

- **Four-step method:**
  1. Characterize the structure of the optimal solution.
  2. Recursively define the value of the optimal solution.
  3. Compute the value of the solution in a bottom-up fashion.
  4. Construct the optimal solution using the computed information.

Manufacturing problem

- **Assembly-Line Scheduling**
  - Two parallel assembly lines in a factory, lines 1 and 2
  - Each line has n stations $S_{i,1} \ldots S_{i,n}$, $i = 1, 2$
  - For each j, $S_{1,j}$ does the same thing as $S_{2,j}$, but it may take a different amount of assembly time $a_{i,j}$
  - Transferring away from line i after stage j costs $t_{i,j}$, $i = 1, 2$ and $j = 1, 2, \ldots, n-1$
  - Also entry time e_i and exit time x_i at beginning and end

Assembly Lines



- n stations on each line: $S_{1,1}, \ldots, S_{1,n}$ and $S_{2,1}, \ldots, S_{2,n}$.
- $a_{i,j}$: time required on line i at station j.
- Entry and exit times: e_1, e_2, x_1, x_2.
- Transfer times: $t_{i,j}$.
- Goal: Find the fastest path through the factory.
- (Trying all possibilities is not tractable.)

Optimal Substructure

- Properties of the optimal solution: consider the fastest way of exiting station $S_{1,j}$.
  - if $j = 1$, then there's only one way,
  - if $j \geq 2$, then in order to exit station j, we must have
    - » 1. either gone through station $S_{1,j-1}$, or
    - » 2. gone through station $S_{2,j-1}$ and transferred to $S_{1,j}$.
  - In the first case, we must have used the fastest way of exiting $S_{1,j-1}$.
  - In the second case, we must have used the fastest way of exiting $S_{2,j-1}$.
- Key observation. An optimal solution to the problem (fastest way through $S_{1,j}$) contains within it an optimal solution to subproblems (fastest way through $S_{1,j-1}$ or $S_{2,j-1}$). [Optimal substructure]
- It's easy to write a recursive solution to the problem now.

Recursive Formula

- $f_i[j]$ = fastest way to exit station $S_{i,j}$.
- f^* = fastest way to exit the assembly-line.
- Clearly $f^* = \min(f_1[n] + x_1, \; f_2[n] + x_2)$ and our observations about the optimal solution lead to

$$f_1[1] = e_1 + a_{1,1}$$
$$f_2[1] = e_2 + a_{2,1}$$

- and for $j \geq 2$,

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

- To construct the optimal solution we keep track of $l_i[j]$, the line (1 or 2) whose station $j - 1$ is used in the fastest way through $S_{i,j}$ and l, the line whose station n is used.

Recursive Formula

- **Let $f_i[j]$ denote the fastest possible time (which is the values of optimal solution, optimal substructure) to get the chassis through $S_{i,j}$**

- **Have the following formulas:**

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

Using symmetric reasoning, we can get the fastest way through station $S_{2,j}$

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

- **Total time:**

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

Running time of the recursive solution

- $r_i(j)$ = running time to compute $f_i[j]$.
- Then, for $i = 1, 2$,

  $r_i(j) \geq r_1(j - 1) + r_2(j - 1)$ with $r_1(1) = r_2(1) = 1$.

- **Claim.** $r_i(j) \geq 2^{j-1}$.
- **Proof.** By induction on j. For the basis, we have $r_i(1) = 1$. By the induction hypothesis,

$$r_i(j) \geq 2^{j-2} + 2^{j-2} = 2^{j-1}.$$

- This works because $f_i[j]$ depends only on $f_1[j\text{-}1]$ and $f_2[j\text{-}1]$. (Start by computing $f_1[1]$ and $f_2[1]$.) It's easy to see that we compute f is linear time.

A better computation

- We can do much better if we compute the $f_i[j]$ values in a different order from the recursive way.
  - By computing the $f_i[j]$ values in order of increasing station numbers j – left to right in Fig.15.2(b)
  - It's running time is linear in n, that is, $\Theta(n)$.

j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

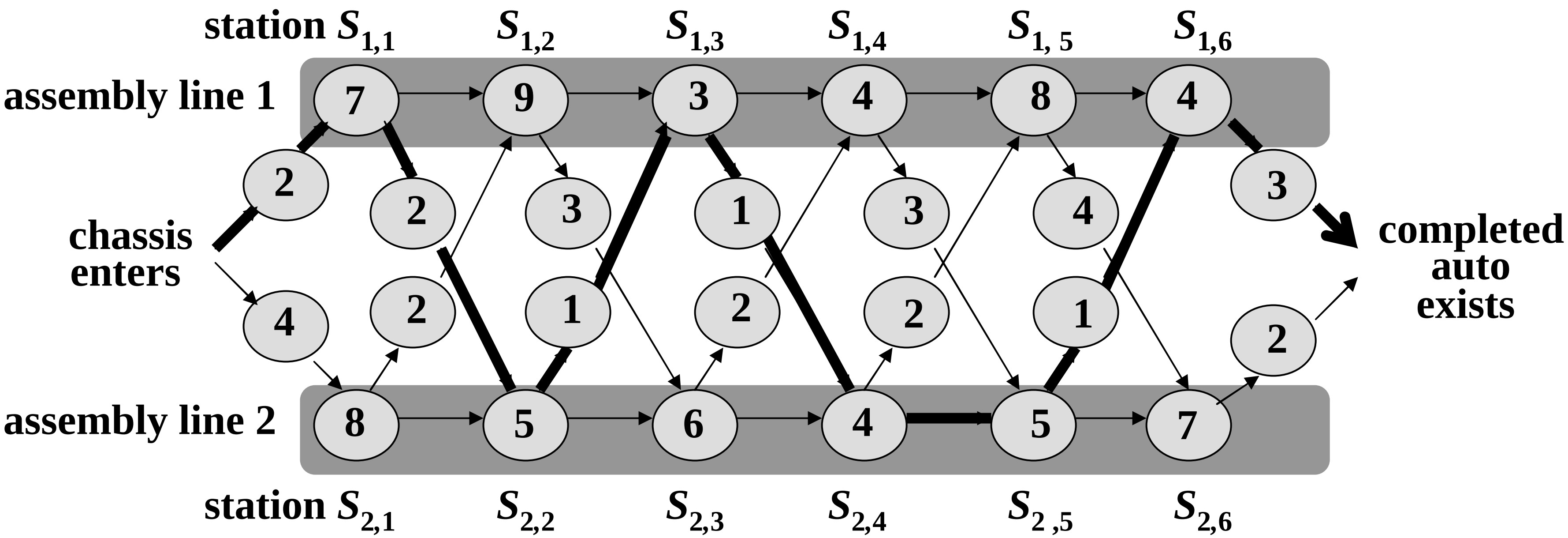
ASSEMBLY-LINE SCHEDULING ALGORITHM

```
FASTEST-WAY(a,t,e,x,n)
f_1[1] ← e_1 + a_{1,1}
f_2[1] ← e_2 + a_{2,1}
for j ← 2 to n
  do if f_1[j-1]+a_{1,j} ≤ f_2[j-1]+t_{2,j-1}+a_{1,j}
         then f_1[j] ← f_1[j-1]+a_{1,j}
              l_1[j] ← 1
         else f_1[j] ← f_2[j-1]+t_{2,j-1}+a_{1,j}
              l_1[j] ← 2
     if f_2[j-1]+a_{2,j} ≤ f_1[j-1] +t_{1,j-1}+a_{2,j}
         then f_2[j] ← f_2[j-1]+a_{2,j}
              l_2[j] ← 2
         else f_2[j] ← f_1[j-1]+t_{1,j-1}+a_{2,j}
              l_2[j] ← 1
if f_1[n] + x_1 ≤ f_2[n] + x_2
   then f* = f_1[n] + x_1
        l* = 1
   else f* = f_2[n] + x_2
        l* = 2
```

Construct an optimal solution

- To keep track of how to construct an optimal solution,
  - Define $l_i[j]$ to be the line number 1 or 2, whose station $j-1$ is used in a fastest way through station $S_{i,j}$, for $i = 1, 2$ and $j = 2, 3, \ldots, n$.
  - Define l^* to be the line whose station n is used in a fastest way through the entire factory.

Construct an optimal solution



j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

Constructing the fastest way

PRINT STATION

```
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
```

- In the example described above, PRINT-STATIONS would produce the output

  line 1, station 6

  line 2, station 5

  line 2, station 4

  line 1, station 3

  line 2, station 2

  line1, station 1

RECURSIVE PRINT STATION

```
RECURSIVE-PRINT-STATIONS(l,i,j)
if j = 0 then return
RECURSIVE-PRINT-STATIONS(l,l_i[j],j-1)
print "line" i ", station" j

Note: To print out all the stations,
   call RECURSIVE-PRINT-STATIONS(l,l*,n)
```

Matrix-chain multiplication

- Goal. Given a sequence of matrices $A_1, A_2, \ldots, A_n$, find an optimal order of multiplication.

- Multiplying two matrices of dimension $p \times q$ and $q \times r$, takes time which is dominated by the number of scalar multiplication, which is pqr.

- Generally. A_i has dimension $p_{i-1} \times p_i$ and we'd like to minimize the total number of scalar multiplications.

Example

- $A = \quad A_1 \qquad\qquad A_2 \qquad\qquad A_3 \qquad\qquad A_4$

  $\quad 10 \times 20 \qquad 20 \times 50 \qquad 50 \times 1 \qquad 1 \times 100$

- **Order 1** $A_1 \times (A_2 \times (A_3 \times A_4))$

  $\text{Cost}(A_3 \times A_4) = 50 \times 1 \times 100$

  $\text{Cost}(A_2 \times (A_3 \times A_4)) = 20 \times 50 \times 100$

  $\text{Cost}(A_1 \times (A_2 \times (A_3 \times A_4))) = 10 \times 20 \times 100$

  Total Cost = 125000

- **Order 2** $(A_1 \times (A_2 \times A_3)) \times A_4$

  $\text{Cost}(A_2 \times A_3) = 20 \times 50 \times 1$

  $\text{Cost}(A_1 \times (A_2 \times A_3)) = 10 \times 20 \times 1$

  $\text{Cost}((A_1 \times (A_2 \times A_3)) \times A_4) = 10 \times 1 \times 100$

  Total Cost = 2200

Brute force method

- What if we check all possible ways of multiplying? How many ways of parenthesizing are there?
- $P(n)$: number of way of parenthesizing. Then $P(1) = 1$ and for $n \geq 2$,

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

- Fact

$$P(n) = \frac{1}{2n+1}\binom{2n}{n} = \Omega(4^n/n^{1.5})$$

- These numbers are called Catalan numbers. There are about 65 combinatorial interpretations in Stanley, *Enumerative Combinatorics*, *Vol*. 2.

Optimal substructure

- Notation. $A_{i..j}$ represents $A_i \ldots A_j$.
- Any parenthesization of $A_{i..j}$ where $i < j$ must split into two products of the form $A_{i..k}$ and $A_{k+1..j}$.

- Optimal substructure. If the optimal parenthesization splits the product as $A_{i..k}$ and $A_{k+1..j}$, then parenthesizations within $A_{i..k}$ and $A_{k+1..j}$ must each be optimal.
  - We apply cut-and-paste argument to prove the optimal substructure property.

An optimal parenthesization's structure

- If the optimal parenthesization of $A_1 \times A_2 \times \ldots \times A_n$ is split between A_k and A_{k+1}, then

optimal parenthesization for $A_1 \times A_2 \times \ldots \times A_n$

$\left\{ \begin{array}{l} \text{optimal parenthesization for } A_1 \times \ldots \times A_k \\ \text{optimal parenthesization for } A_{k+1} \times \ldots \times A_n \end{array} \right.$

- The only uncertainty is the value of k
  - Try all possible values of k. The one that returns the minimum is the right choice.

A recursive solution

- Define $m[i, j]$ as the minimum number of scalar multiplications needed to compute the matrix product $A_{i..j}$. (We want the value of $m[1, n]$.)
  - If $i = j$, there is nothing to do, so that $m[i, i] = 0$.
  - Otherwise, suppose that the optimal parenthesization split the product as $A_{i..k}$ and $A_{k+1..j}$.

- $(A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5 \quad A_6)$

  $m[1,3]$ $\qquad$ $m[4,6]$

  $(A_1 \quad A_2 \quad A_3) \quad (A_4 \quad A_5 \quad A_6)$

  $p_0 \times p_3$ $\qquad$ $p_3 \times p_6$

  $m[1,3] + m[4,6] + p_0 p_3 p_6$

A recursive formulation

- We would like to find the split that uses the minimum number of multiplications. Thus,

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

  - $m[i, k]$ = optimal cost for $A_i \times \ldots \times A_k$
  - $m[k+1, j]$ = optimal cost for $A_{k+1} \times \ldots \times A_j$
  - $p_{i-1} p_k p_j$ = cost for $(A_i \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_j)$

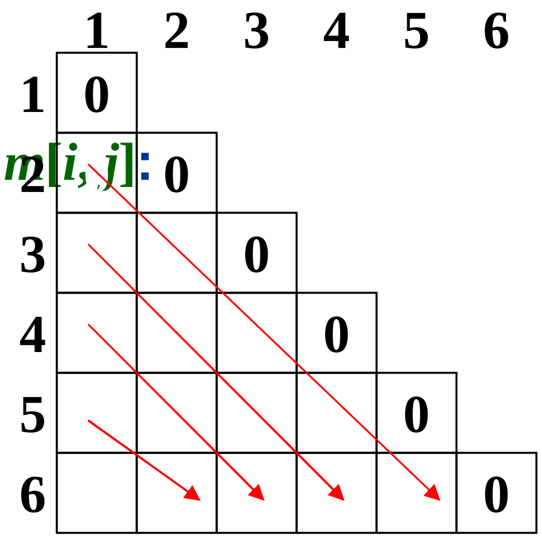
- To obtain the actual parenthesization, keep track of the optimal k for each pair (i, j) as $s[i, j]$.

Computing the Optimal Costs

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

  $T(n) \ge 1 + \sum\limits_{1 \le k < n} (T(k) + T(n-k) + 1) = \Omega(2^n)$, for $n > 1$

  – The recursive solution takes exponential time. (Easy proof by induction.)



- Instead, use a dynamic program to fill in a table $m[i, j]$:
  - Start by setting $m[i, i] = 0$ for $i = 1, \ldots, n$.
  - Then compute $m[1, 2], m[2, 3], \ldots, m[n-1, n]$.
  - Then $m[1, 3], m[2, 4], \ldots, m[n-2, n], \ldots$
  - . . . so on till we can compute $m[1, n]$.
- The input a sequence $p = \langle p_0, p_1, \ldots, p_n \rangle$, we use an auxiliary table $s[1 .. n, 1 .. n]$ that records which index of k achieved the optimal cost in computing $m[i, j]$.

Matrix-Chain Multiplication DP Algo.

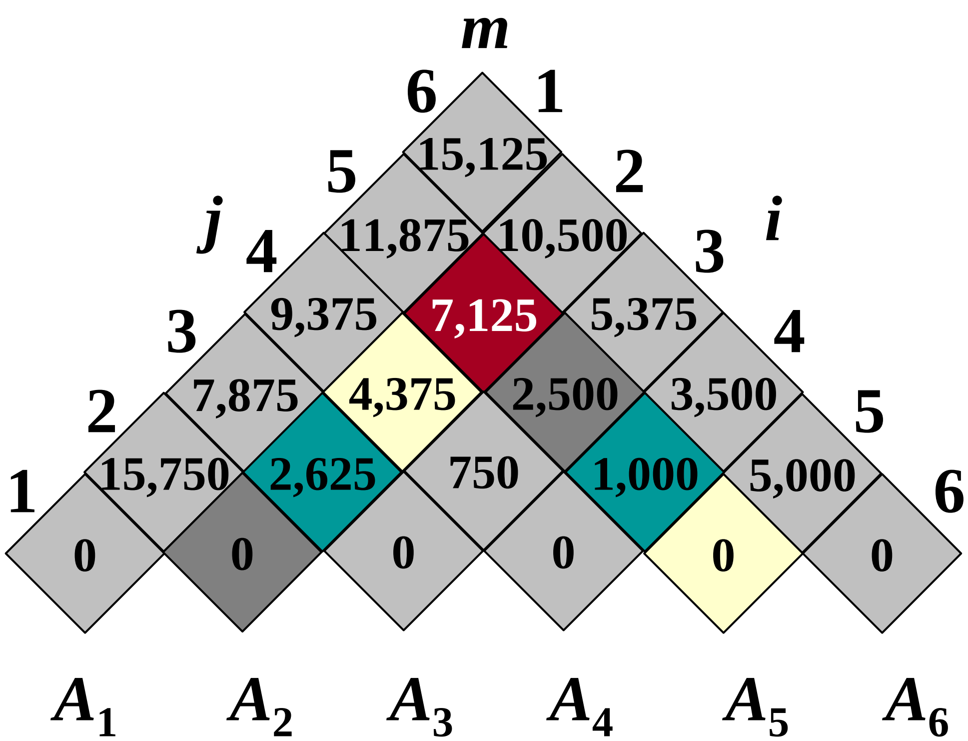
MATRIX-CHAIN MULTIPLICATION DP

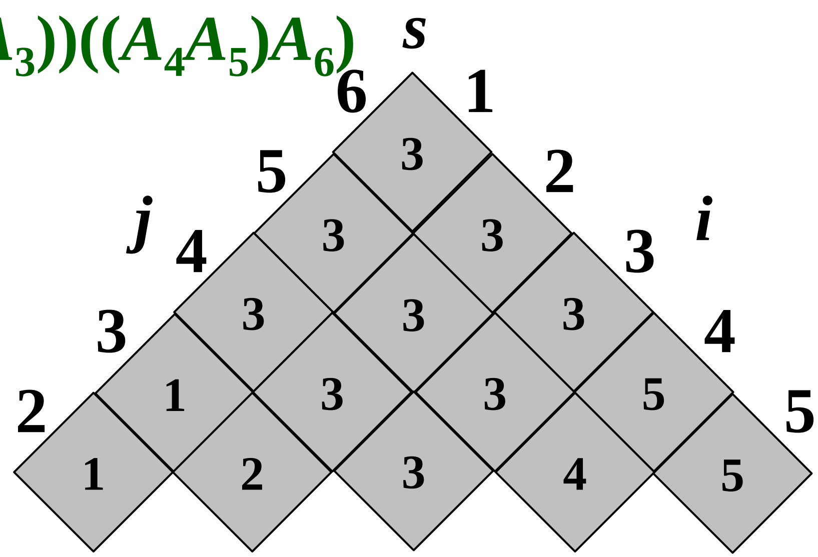
```
MATRIX-CHAIN-ORDER(p)
n ← length[p]-1
for i ← 1 to n
    do m[i,i] ← 0
for l ← 2 to n
     do for i ← 1 to n–l +1
             do j ← i + l – 1
                 m[i,j] ← ∞
                 for k ← i to j-1
                      do q ← m[i,k] + m[k+1,j] + p_{i-1} p_k p_j
                          if q < m[i,j]
                             then m[i,j] ← q
                                  s[i,j] ← k
return m and s
```

- $O(n^3)$

Example: DP for CMM

- **The optimal solution is $((A_1(A_2A_3))((A_4A_5)A_6))$**





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

Construct an Optimal Solution

- The final matrix multiplication in computing $A_{1..n}$ optimally is $A_{1..s[1,n]} A_{s[1,n]+1..n}$. $s[1, s[1, n]]$ determines the last matrix multiplication in computing $A_{1..s[1,n]}$ and $s[s[1, n]+1, n]$ determines the last matrix multiplication in computing $A_{s[1,n]+1..n}$.

PRINT-OPTIMAL-PARENS

```
PRINT-OPTIMAL-PARENS(s,i,j)
if i=j
  then print "A"i
  else print "( "
       PRINT-OPTIMAL-PARENS(s,i,s[i,j])
       PRINT-OPTIMAL-PARENS(s,s[i,j]+1,j)
       print ") "
```