DP (cont.)

- DP is usually applied to optimization problems:
  - Find solution of optimal value.
  - Optimal value may not be unique.
- To design a DP algorithm:
  1. Characterize structure of optimal solution.
  2. Recursively define value of optimal solution.
  3. Compute value of optimal sol. (bottom-up).
  4. Construct actual optimal solution (e.g.: shortest path vs. shortest path length from 3.).

Matrix-Chain Multiplication

- Input: sequence $\langle A_1, A_2, \ldots, A_n \rangle$ of matrices.
- Output: an order to compute $A_1 A_2 \ldots A_n$ such that the total number of scalar operations (additions and multiplications) is minimized.
  - Order to perform matrix multiplications.
  - How to parenthesize the matrix product.

Parenthesized Products

- A fully parenthesized matrix product:
  - A single matrix or
  - Product of 2 fully parenthesized matrix products surrounded by parentheses.
- No matter the order of multiplications, final result is the same (multip. is associative).
- Example:
  - (A1(A2(A3A4)))
  - (A1((A2A3)A4))
  - ((A1A2)(A3A4))
  - ((A1(A2A3))A4)
  - (((A1A2)A3)A4)
- The way to parenthesize: big impact on *cost* of multip:
  - A_{pxq}, B_{qxr}: $AB=C_{pxr}$ requires "pqr" operations.
- A_i: $p_{i-1}xp_i$ size matrix.
- Want: fully parenthesize $A_1A_2 \ldots A_n$ s.t. minimize cost of multip.

Counting # of parenthesizations

- check all possibilities
- $P(n)$: # of parenthesizations
  - for $k = 1, 2, \dots, n-1$ do :
    - Split $A_1 \cdot A_2 \cdots A_n$ at $k, k+1$
    - parenthesize $A_1 \cdot A_2 \cdots A_k$ independently
      $A_{k+1} \cdot A_{k+2} \cdots A_n$
- Recurrence: $P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \end{cases}$
- Solution: $P(n) = C(n-1)$
  $\Downarrow$
  - sequence of Catalan numbers
  - $C(n) = \frac{1}{n+1}\binom{2n}{n} = \Omega\left(\frac{4^n}{n^{3/2}}\right)$

  $\rightarrow$ exponential

1. Structure of optimal solution

- Notation: $A_{i..j}$ $\mapsto$ matrix obtained after evaluating product $A_i \cdot A_{i+1} \cdots A_j$

- Optimal solution: split of $A_1 \cdot A_2 \cdots A_n$ at some $k, k+1$, $1 \leq k < n$:

  - first compute $A_{1..k}$, $A_{k+1..n}$

    multiply

    $A_{1..n}$

  - $A_{1..k}$, $A_{k+1..n}$ must be optimal parenthesizations

$\Longrightarrow$ optimal solution contains within it optimal solutions to subproblems

$\downarrow$

Key ingredient for applying DP!

2. Recursive solution :

- define value of optimal solution recursively
- subproblems: compute minimum cost of

$A_i A_{i+1} \cdots A_j$, $1 \le i \le j \le n$

- Let $m[i,j]$ be min # scalar multiplications to compute $A_{i..j}$

($m[1,n]$ is optimum for $A_{1..n}$)

- $i = j \Rightarrow m[i,i] = 0$
- $i < j \mapsto$ have split at A_k, A_{k+1}, $i \le k < j$

$\mapsto ((A_i A_{i+1} \cdots A_k)(A_{k+1} A_{k+2} \cdots A_j))$

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

- but we don't know k!
  - $j-i$ possible values for k:

    $i, i+1, \cdots, j-1$
  - optimal parenthesization has one of them

    $\Rightarrow$ check all

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

Note: $m[i,j]$ gives cost. To obtain actual parenthesization define $s[i,j] = k$ for optimal split of $A_i A_{i+1} \cdots A_j$

3. Computing optimal cost

- recursive algorithm to compute $m[1,n]$ $\mapsto$ exponential time!

- Key observation:
  - "few" subproblems: one for each pair i, j, $1 \le i \le j \le n$

$$\Rightarrow \binom{n}{2} + n = \Theta(n^2) \text{ subproblems}$$

- a recursive algorithm may encounter a subproblem many times

$\Longrightarrow$ overlapping subproblems

$\downarrow$

Key ingredient for applying DP!

- Perform 3rd step of DP: compute optimal cost <u>bottom-up</u>

$A_i \mapsto p_{i-1} \times p_i$

<u>Input</u>: $\langle p_0, p_1, \dots, p_n \rangle$

<u>Data structures</u>: • table $\begin{cases} m[1..n, 1..n] \text{ to store } m[i,j] \\ s[1..n, 1..n] \text{ to store } s[i,j] \end{cases}$

```
MCO (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
}
```

I.e. • first $m[i,i]$, $i=\overline{1,n}$

- next $(l=2)$: $m[i, i+1]$, $i = \overline{1, n-1}$
- next $(l=3)$: $m[i, i+2]$, $i = \overline{1, n-2}$

etc. $\mapsto$ $\boxed{m[i,j] \text{ depends on already computed } \begin{cases} m[i,k] \\ m[k+1,j] \end{cases}}$

Example:

A_1 : 30×35

A_2 : 35×15

A_3 : 15×5

A_4 : 5×10

A_5 : 10×20

A_6 : 20×25

$\downarrow$

$p_0 = 30$

$p_1 = 35$

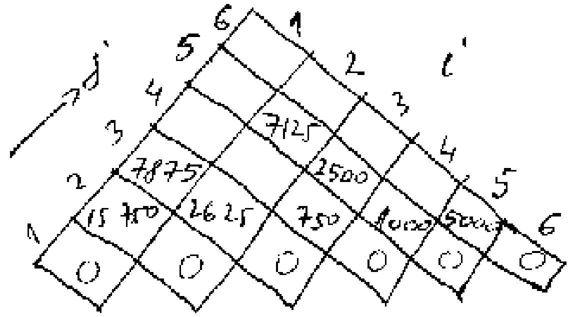
$p_2 = 15$

$p_3 = 5$

$p_4 = 10$

$p_5 = 20$

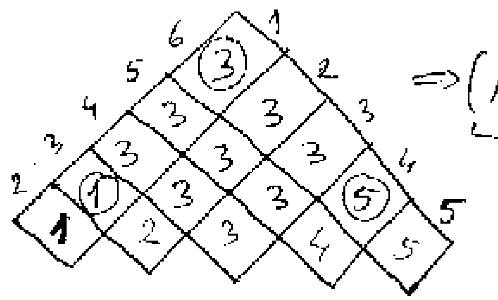
$p_6 = 25$



$m[1,3] \longrightarrow A_{1..2} \cdot A_3 \quad : 15\,750 + 30 \cdot 15 \cdot 5$

$\underline{A_1 \cdot A_{2..3}} \quad : 2625 + 30 \cdot 35 \cdot 5$

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 13000 \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 7125 \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 11\,375 \end{cases}$$



$\Rightarrow \underbrace{(A_1 (A_2 A_3))}\underbrace{((A_4 A_5) A_6)}$

4. Constructing optimal solution to have value from step 3
   - how to multiply matrices ?!
     - use $s[1..n, 1..n]$
     - $s[i,j]$ records k, $i \le k < j$, for optimal parenthesization of $A_{i..j}$
     - final multiplication in $A_{1..n}$ is $A_{1..s[1,n]} \, A_{s[1,n]+1..n}$

Elements of Dynamic Programming

→ when to look for a DP solution

- Optimal substructure
  - optimal solution contains within it optimal solutions to subproblems
  - to show optimality of subproblems
    → proof by contradiction

- Overlapping subproblems
  - # of distinct subproblems is polynomial
  - solve each subproblem once

- Next class: All pairs shortest paths.