Matrix-Chain Multiplication

- Let A be an n by m matrix, let B be an m by p matrix, then $C = AB$ is an n by p matrix.
- $C = AB$ can be computed in $O(nmp)$ time, using traditional matrix multiplication.

- Suppose I want to compute $A_1A_2A_3A_4$.
- Matrix Multiplication is associative, so I can do the multiplication in several different orders.

Example:

- A_1 is 10 by 100 matrix
- A_2 is 100 by 5 matrix
- A_3 is 5 by 50 matrix
- A_4 is 50 by 1 matrix
- $A_1A_2A_3A_4$ is a 10 by 1 matrix

Matrix-Chain Multiplication

- Let A be an n by m matrix, let B be an m by p matrix, then $C = AB$ is an n by p matrix.
- $C = AB$ can be computed in $O(nmp)$ time, using traditional matrix multiplication.

- Suppose I want to compute $A_1A_2A_3A_4$.
- Matrix Multiplication is associative, so I can do the multiplication in several different orders.

Example:

- A_1 is 10 by 100 matrix
- A_2 is 100 by 5 matrix
- A_3 is 5 by 50 matrix
- A_4 is 50 by 1 matrix
- $A_1A_2A_3A_4$ is a 10 by 1 matrix

Example

- A_1 is 10 by 100 matrix
- A_2 is 100 by 5 matrix
- A_3 is 5 by 50 matrix
- A_4 is 50 by 1 matrix
- $A_1 A_2 A_3 A_4$ is a 10 by 1 matrix

5 different orderings = 5 different parenthesizations

- $(A_1(A_2(A_3A_4)))$
- $((A_1A_2)(A_3A_4))$
- $(((A_1A_2)A_3)A_4)$
- $((A_1(A_2A_3))A_4)$
- $(A_1((A_2A_3)A_4))$

Each parenthesization is a different number of mults

Let $A_{ij} = A_i \cdots A_j$

Example

- A_1 is 10 by 100 matrix, A_2 is 100 by 5 matrix, A_3 is 5 by 50 matrix, A_4 is 50 by 1 matrix, $A_1A_2A_3A_4$ is a 10 by 1 matrix.
- $(A_1\boxed{(A_2(A_3A_4))})$
  - $A_{34} = A_3A_4$, 250 mults, result is 5 by 1
  - $A_{24} = A_2A_{34}$, 500 mults, result is 100 by 1
  - $A_{14} = A_1A_{24}$, 1000 mults, result is 10 by 1
  - Total is 1750
- $(\overbrace{(A_1A_2)}\overbrace{(A_3A_4)})$
  - $A_{12} = A_1A_2$, 5000 mults, result is 10 by 5
  - $A_{34} = A_3A_4$, 250 mults, result is 5 by 1
  - $A_{14} = A_{12}A_{34})$, 50 mults, result is 10 by 1
  - Total is 5300
- $((\overbrace{(A_1A_2)A_3})A_4)$
  - $A_{12} = A_1A_2$, 5000 mults, result is 10 by 5
  - $A_{13} = A_{12}A_3$, 2500 mults, result is 10 by 50
  - $A_{14} = A_{13}A_4$, 500 mults, results is 10 by 1
  - Total is 8000

Example

- A_1 is 10 by 100 matrix, A_2 is 100 by 5 matrix, A_3 is 5 by 50 matrix, A_4 is 50 by 1 matrix, $A_1A_2A_3A_4$ is a 10 by 1 matrix.
- $(\overbrace{(A_1(A_2A_3))}A_4)$
  - $A_{23} = A_2A_3$, **25000 mults, result is 100 by 50**
  - $A_{13} = A_1A_{23}$, **50000 mults, result is 10 by 50**
  - $A_{14} = A_{13}A_4$, **500 mults, results is 10 by**
  - **Total is 75500**
- $(A_1\boxed{((A_2A_3)A_4)})$
  - $A_{23} = A_2A_3$, **25000 mults, result is 100 by 50**
  - $A_{24} = A_{23}A_4$, **5000 mults, result is 100 by 1**
  - $A_{14} = A_1A_{24}$, **1000 mults, result is 10 by 1**
  - **Total is 31000**

Conclusion **Order of operations makes a huge difference. How do we compute the minimum?**

One approach

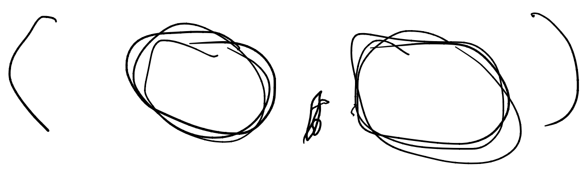
Parenthesization A product of matrices is fully parenthesized if it is either

- a single matrix, or
- a product of two fully parenthesized matrices, surrounded by parentheses

Each parenthesization defines a set of n-1 matrix multiplications. We just need to pick the parenthesization that corresponds to the best ordering.

How many parenthesizations are there?

$$(((A_1 A_2) A_3)(A_4 A_5))$$



One approach

Parenthesization A product of matrices is **fully parenthesized** if it is either

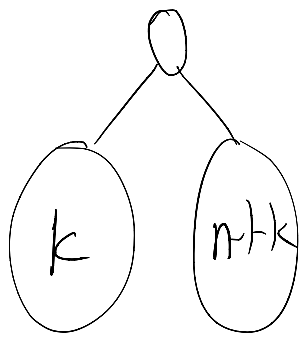
- a single matrix, or
- a product of two fully parenthesized matrices, surrounded by parentheses

Each parenthesization defines a set of **n-1** matrix multiplications. We just need to pick the parenthesization that corresponds to the best ordering.

How many parenthesizations are there?

Let **P(n)** be the number of ways to parenthesize **n** matrices.

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



This recurrence is related to the Catalan numbers, and solves to

$$P(n) = \Omega(4^n/n^{3/2}).$$

Conclusion Trying all possible parenthesizations is a bad idea.

Use dynamic programming

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution bottom-up
4. Construct an optimal solution from the computed information

Structure of an optimal solution If the outermost parenthesization is

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

then the optimal solution consists of solving A_{1i} and $A_{i+1,n}$ optimally and then combining the solutions.

Proof

Structure of an optimal solution If the outermost parenthesization is

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

then the optimal solution consists of solving A_{1i} and $A_{i+1,n}$ optimally and then combining the solutions.

Proof: Consider an optimal algorithm that does not solve A_{1i} optimally. Let x be the number of multiplications it does to solve A_{1i}, y be the number of multiplications it does to solve $A_{i+1,n}$, and z be the number of multiplications it does in the final step. The total number of multiplications is therefore

$$x + y + z.$$

But since it is not solving A_{1i} optimally, there is a way to solve A_{1i} using $x' < x$ multiplications. If we used this optimal algorithm instead of our current one for A_{1i}, we would do

$$x' + y + z < x + y + z$$

multiplications and therefore have a better algorithm, contradicting the fact that our algorithms is optimal.

Proof

Proof: Consider an optimal algorithm that does not solve A_{1i} optimally. Let x be the number of multiplications it does to solve A_{1i}, y be the number of multiplications it does to solve $A_{i+1,n}$, and z be the number of multiplications it does in the final step. The total number of multiplications is therefore

$$x + y + z.$$

But since it is not solving A_{1i} optimally, there is a way to solve A_{1i} using $x' < x$ multiplications. If we used this optimal algorithm instead of our current one for A_{1i}, we would do

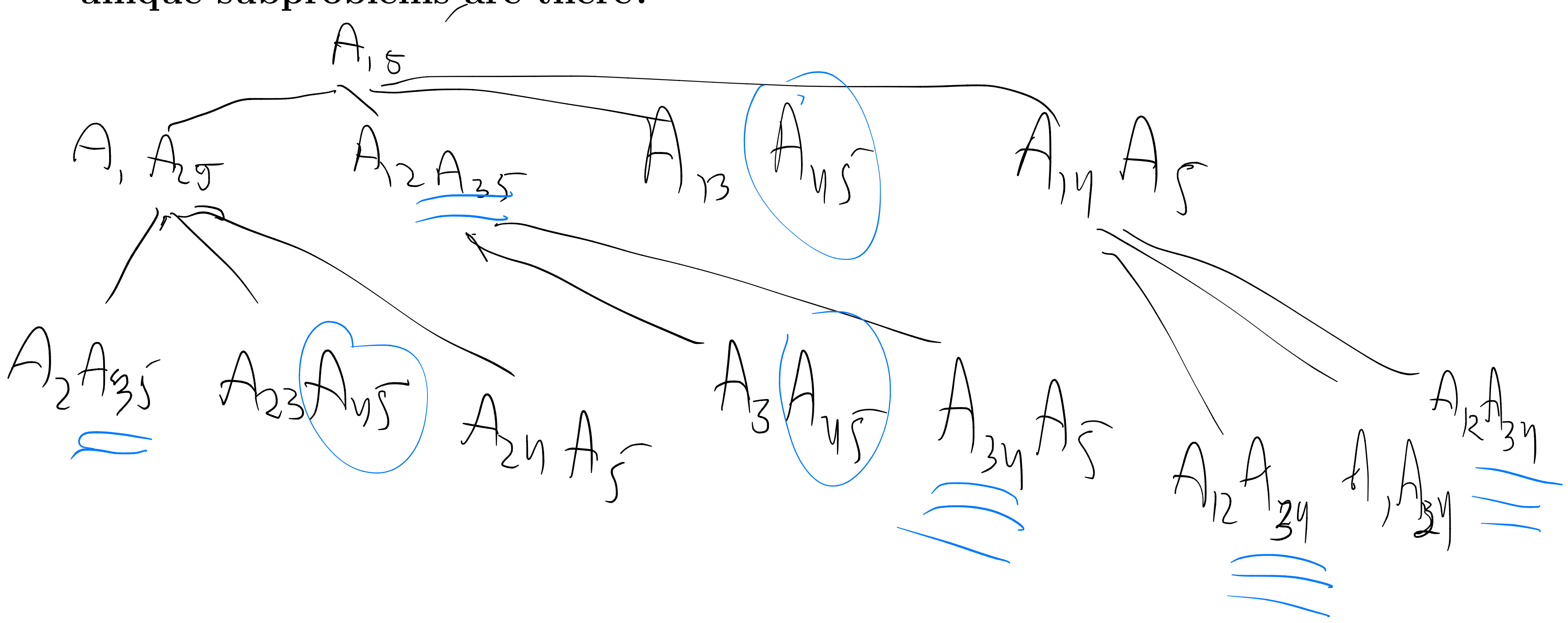
$$x' + y + z < x + y + z$$

multiplications and therefore have a better algorithm, contradicting the fact that our algorithms is optimal.

Meta-proof that is not a correct proof Our problem consists of subproblems, assume we didn't solve the subproblems optimally, then we could just replace them with an optimal subproblem solution and have a better solution.

Recursive solution

In the enumeration of the $P(n) = \Omega(4^n/n^{3/2})$ subproblems, how many unique subproblems are there?



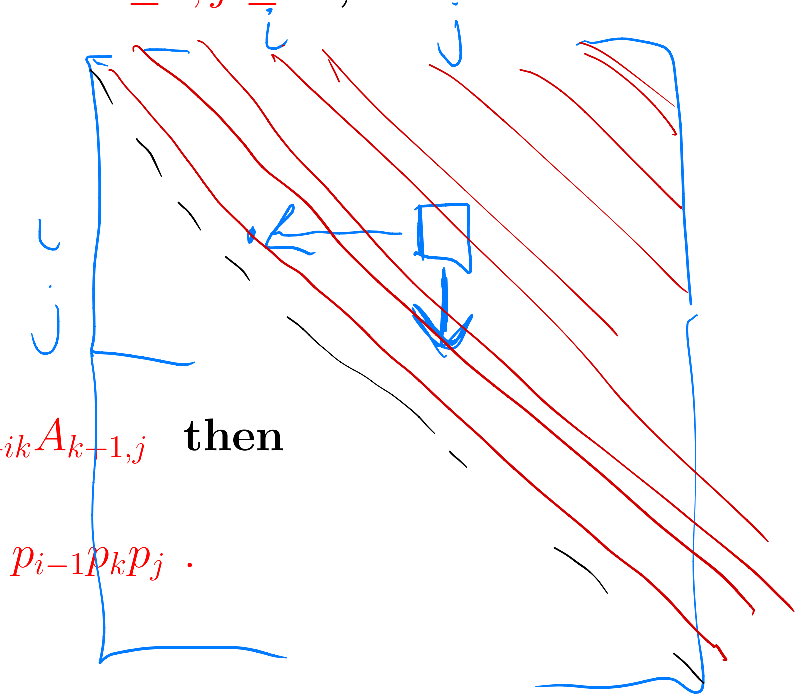
Recursive solution

In the enumeration of the $P(n) = \Omega(4^n/n^{3/2})$ subproblems, how many unique subproblems are there?

Answer: A subproblem is of the form A_{ij} with $1 \le i, j \le n$, so there are $O(n^2)$ subproblems!

Notation

- Let A_i be p_{i-1} by p_i.
- Let $m[i, j]$ be the cost of computing A_{ij}



If the final multiplication for A_{ij} is $A_{ij} = A_{ik}A_{k+1,j}$ then

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

We don't know k a priori, so we take the minimum

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

Direct recursion on this does not work! We must use the fact that there are at most $O(n^2)$ different calls. What is the order?

The final code

$A_{1,100}$

$\sim 100^2/2 \approx 5000$

4100

Matrix-Chain-Order(p)

```
$n \leftarrow length[p] - 1$
for $i \leftarrow 1$ to $n$
        do $m[i,i] \leftarrow 0$
for $l \leftarrow 2$ to $n$                    $\triangleright$ $l$ is the chain length.
        do for $i \leftarrow 1$ to $n - l + 1$
                do $j \leftarrow i + l - 1$
                    $m[i,j] \leftarrow \infty$
                    for $k \leftarrow i$ to $j - 1$
                        do $q \leftarrow m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$
                            if $q < m[i,j]$
                                then $m[i,j] \leftarrow q$
                                     $s[i,j] \leftarrow k$
return $m$ and $s$
```