# Matrix Multiplication

# (Dynamic Programming)

# Matrix Multiplication: Review

- Suppose that $A_1$ is of size $S_1$ x $S_2$, and $A_2$ is of size $S_2$ x $S_3$.
- What is the time complexity of computing $A_1 * A_2$?
- What is the size of the result?

# Matrix Multiplication: Review

- Suppose that $A_1$ is of size $S_1$ x $S_2$, and $A_2$ is of size $S_2$ x $S_3$.
- What is the time complexity of computing $A_1$ * $A_2$?
- What is the size of the result? $S_1$ x $S_3$.
- Each number in the result is computed in $O(S_2)$ time by:
  - multiplying $S_2$ pairs of numbers.
  - adding $S_2$ numbers.
- Overall time complexity: $O(S_1 * S_2 * S_3)$.

# Optimal Ordering for Matrix Multiplication

- Suppose that we need to do a sequence of matrix multiplications:
  - result = $A_1 * A_2 * A_3 * \ldots * A_K$
- The number of columns for $A_i$ must equal the number of rows for $A_{i+1}$.
- What is the time complexity for performing this sequence of multiplications?

# Optimal Ordering for Matrix Multiplication

- Suppose that we need to do a sequence of matrix multiplications:
  - result = $A_1$ * $A_2$ * $A_3$ * ... * $A_K$
- The number of columns for $A_i$ must equal the number of rows for $A_{i+1}$.
- What is the time complexity for performing this sequence of multiplications?
- The answer is: it depends on the order in which we perform the multiplications.

# An Example

- Suppose:
  - $A_1$ is17x2.
  - $A_2$ is 2x35.
  - $A_3$ is 35x4.
- $(A_1 * A_2) * A_3$:

- $A_1 * (A_2 * A_3)$:

# An Example

- Suppose:
  - $A_1$ is17x2.
  - $A_2$ is 2x35.
  - $A_3$ is 35x4.
- $(A_1 * A_2) * A_3$:
  - 17*2*35 = 1190 multiplications and additions to compute $A_1 * A_2$.
  - 17*35*4 = 2380 multiplications and additions to compute multiplying the result of $(A_1 * A_2)$ with $A_3$.
  - Total: 3570 multiplications and additions.
- $A_1 * (A_2 * A_3)$:
  - 2*35*4 = 280 multiplications and additions to compute $A_2 * A_3$.
  - 17*2*4 = 136 multiplications and additions to compute multiplying $A_1$ with the result of $(A_2 * A_3)$.
  - Total: 416 multiplications and additions.

# Adaptation to Dynamic Programming

- Suppose that we need to do a sequence of matrix multiplications:
  - result = $A_1 * A_2 * A_3 * ... * A_K$
- To figure out if and how we can use dynamic programming, we must address the standard two questions we always need to address for dynamic programming:

1. Can we define a set of smaller problems, such that the solutions to those problems make it easy to solve the original problem?
2. Can we arrange those smaller problems in a sequence **of reasonable size**, so that each problem in that sequence **only depends on problems that come earlier** in the sequence?

# Defining Smaller Problems

1. Can we define a set of smaller problems, whose solutions make it easy to solve the original problem?
   - Original problem: optimal ordering for $A_1 * A_2 * A_3 * ... * A_K$

- Yes! Suppose that, for every i between 1 and K-1 we know:
  - The best order (and best cost) for multiplying matrices $A_1, ..., A_i$.
  - The best order (and best cost) for multiplying matrices $A_{i+1}, ..., A_K$.
- Then, for every such i, we obtain a possible solution for our original problem:
  - Multiply matrices $A_1, ..., A_i$ in the best order. Let $C_1$ be the cost of that.
  - Multiply matrices $A_{i+1}, ..., A_K$ in the best order. Let $C_2$ be the cost of that.
  - Compute $(A_1 * ... * A_i) * (A_{i+1} * ... * A_K)$. Let $C_3$ be the cost of that.
    - $C_3$ = rows of $(A_1 * ... * A_i)$ * cols of $(A_1 * ... * A_i)$ * cols of $(A_{i+1} * ... * A_K)$.
      = rows of $A_1$ * cols of $A_i$ * cols of $A_K$
  - Total cost of this solution = $C_1 + C_2 + C_3$.

# Defining Smaller Problems

1. Can we define a set of smaller problems, whose solutions make it easy to solve the original problem?
   – Original problem: optimal ordering for $A_1 * A_2 * A_3 * ... * A_K$

- Yes! Suppose that, for every i between 1 and K-1 we know:
  – The best order (and best cost) for multiplying matrices $A_1, ..., A_i$.
  – The best order (and best cost) for multiplying matrices $A_{i+1}, ..., A_K$.
- Then, for every such i, we obtain a possible solution.
- We just need to compute the cost of each of those solutions, and choose the smallest cost.
- Next question:

2. Can we arrange those smaller problems in a sequence **of reasonable size**, so that each problem in that sequence **only depends on problems that come earlier** in the sequence?

# Defining Smaller Problems

2. Can we arrange those smaller problems in a sequence **of reasonable size**, so that each problem in that sequence **only depends on problems that come earlier** in the sequence?

- To compute answer for $A_1 * A_2 * A_3 * \ldots * A_K$:

  For i = 1, …, K-1, we had to consider solutions for:
  - $A_1, \ldots, A_i$.
  - $A_{i+1}, \ldots, A_K$.
- So, what is the set of all problems we must solve?

# Defining Smaller Problems

2. Can we arrange those smaller problems in a sequence **of reasonable size**, so that each problem in that sequence **only depends on problems that come earlier** in the sequence?

- To compute answer for $A_1 * A_2 * A_3 * ... * A_K$ :
  For i = 1, ..., K-1, we had to consider solutions for:
  - $A_1, ..., A_i$.
  - $A_{i+1}, ..., A_K$.
- So, what is the set of all problems we must solve?
- For M = 1, ..., K.
  - For N = 1, ..., M.
    - Compute the best ordering for $A_N * ... * A_M$.
- What this the number of problems we need to solve? Is the size reasonable?
  - We must solve $\Theta(K^2)$ problems. We consider this a reasonable number.

# Defining Smaller Problems

- The set of all problems we must solve:
- For M = 1, ..., K.
  - For N = 1, ..., M.
    - Compute the best ordering for $A_N * ... * A_M$.
- What is the order in which we must solve these problems?

# Defining Smaller Problems

- The set of all problems we must solve, in the correct order:
- For M = 1, ..., K.
  - For N = M, ..., 1.
    - Compute the best ordering for $A_N$ * ... * $A_M$.
- N must go from M to 1, NOT the other way around.
- Why? Because, given M, the larger the N is, the smaller the problem is of computing the best ordering for $A_N$ * ... * $A_M$.

# Solving These Problems

- For M = 1, ..., K.
  - For N = M, ..., 1.
    - Compute the best ordering for $A_N$ * ... * $A_M$.
- What are the base cases?
- N = M.
  - costs[N][M] = 0.
- N = M - 1.
  - costs[N][M] = rows($A_N$) * cols($A_N$) * cols($A_M$).
- Solution for the recursive case:

# Solving These Problems

- For M = 1, ..., K.
  - For N = M, ..., 1.
    - Compute the best ordering for $A_N * ... * A_M$.
- Solution for the recursive case:

- minimum_cost = 0
- For R = N, ..., M-1:
  - cost1 = costs[N][R]
  - cost2 = costs[R+1][M]
  - cost3 = rows($A_N$) * cols($A_R$) * cols($A_M$)
  - cost = cost1 + cost2 + cost3
  - if (cost < minimum_cost) minimum_cost = cost
- costs[N][M] = minimum_cost