This Class: Dynamic Programming

- Solves optimization problems;
- Sub-problems repeats; optimal solutions to sub-problems — part of overall optimal solution;
- Recursively formulation; typically tabular solution;
- Solution space typically exponential, yet problem solvable in polynomial time.

Matrix chain multiplication

Given: n matrices $M_1, M_2, M_3 \ldots M_n$.

Task: Compute $\prod_{i=1}^{n} Mi = M_1 * M_2 * M_3 \cdots * M_n$, as fast as possible.

Example:

$M_1 = 10 \times 100$
$M_2 = 100 \times 5$
$M_3 = 5 \times 50$

There are two ways to calculate this matrix multiplication:

1. $((M_1 * M_2) * M_3) \to 10 \times 100 \times 5 + 10 \times 5 \times 50 = 75000$
2. $(M_1 * (M_2 * M_3)) \to 100 \times 5 \times 50 + 10 \times 100 \times 50 = 75000$

Obviously, the first way has fewer numbers of operations than the second one. In general, when we do the matrix chain multiplication, the order of multiplication matters in terms of numbers of operations.

Q: how we choose the order so that the matrix multiplication has fewest numbers of operations?

Example: let’s say we have $M_1, M_2, M_3, M_4$

1. The intuitive way: exploits space of all possible orders, and pick the best solution.

   $((M_1 * M_2) * M_3) * M_4) \quad M_{13} \quad M_4$
   $(M_1 * (M_2 * (M_3 * M_4))) \quad M_{1} \quad M_{24}$
   $((M_1 * M_2) * (M_3 * M_4)) \quad M_{12} \quad M_{34}$
   $(M_1 * (M_2 * M_3) * M_4) \quad M_1, M_{24}$
   $((M_1 * (M_2 * M_3) * M_4) \quad M_{13}, M_4$
Analysis:

\[
\begin{array}{cccc}
M1 & M2 & M3 & M4 \\
\uparrow & \uparrow & \uparrow & \uparrow \\
n \text{Matrices}
\end{array}
\]

\(n - 1\) position of last step

\[
P (n) = \begin{cases} 
\sum_{k=1}^{n-1} P(k) + P(n - k) & n > 1 \\
0 & n = 1
\end{cases}
\]

Claim: the running time in this case is \(\Omega (2^n)\).

\(\textcircled{2}\) Try regular recursive solution:

\(M_{ij}\) — Number of operations to multiply \(M_i \ldots M_j\), such that \(j \geq i\);

e.g. \(M_{13} = \text{Cost} (M1 \ast M2 \ast M3)\) (two options to break, (1,2) and (2,3))

\[
= \min \left\{ 
M_{11} + M_{23} + r_0 \ast r_1 \ast r_3 \\
M_{12} + M_{33} + r_0 \ast r_2 \ast r_3
\right\}
\]

In general,

\(M_{ij} = \min \left\{ 
M_{ik} + M_{k+1j} + r_{i-1} \ast r_k \ast r_j \right\}
\]

where \(i \leq k < j\)

\[
T (n) = \begin{cases} 
\sum_{k=1}^{n-1} T(k) + T(n - k) + 3 & n > 1 \\
0 & n = 1
\end{cases}
\]

Claim : \(T(n) = \Omega (2^n)\).

\(\textcircled{3}\) Try bottom-up and store \(M_{ij}\) in a N*N table

Example:

\(M1 : 10 \ast 20 ; M2 : 20 \ast 50 ; M3 : 50 \ast 1 ; M4 : 1 \ast 100\)
\(r_0 = 10; r_1 = 20; r_2 = 50; r_3 = 1; r_4 = 100;\)

\(M_{12} = M_{11} + M_{22} + r_0 \ast r_1 \ast r_2 = 0 + 0 + 10 \ast 20 \ast 50 = 10000\)
\(M_{23} = M_{22} + M_{33} + r_1 \ast r_2 \ast r_3 = 0 + 0 + 20 \ast 50 \ast 1 = 1000\)
\(M_{34} = M_{33} + M_{44} + r_2 \ast r_3 \ast r_4 = 0 + 0 + 50 \ast 1 \ast 100 = 5000\)
The optimal order is: \(((M1* (M2*M3))*M4)\)

Claim: the running time in this case is \(O(n^3)\).

Read from the book: Memoization