

Lecture 16 Chain Matrix Multiplication Problem using Dynamic Programming

Problem Statement: Chain Matrix Multiplication

Given a chain of $[A_1, A_2, \dots, A_n]$ of n matrices where for $i = 1, 2, \dots, n$, matrix A_i has dimension $p_{i-1} \times p_i$, find the order of multiplication which minimizes the number of scalar multiplications.

Note:

Order of A_1 is $p_0 \times p_1$,

Order of A_2 is $p_1 \times p_2$,

Order of A_3 is $p_2 \times p_3$, etc.

Order of $A_1 \times A_2 \times A_3$ is $p_0 \times p_3$,

Order of $A_1 \times A_2 \times \dots \times A_n$ is $p_0 \times p_n$

Why Dynamic Programming in this problem?

- Problem is of type optimization
- Sub-problems are dependent
- Optimal structure can be characterized and
- Can be defined recursively
- Solution for base cases exists
- Optimal solution can be constructed
- Hence here is dynamic programming

Dynamic Programming Formulation

- Let $A_{i..j} = A_i \cdot A_{i+1} \cdot \dots \cdot A_j$
- Order of $A_i = p_{i-1} \times p_i$ and order of $A_j = p_{j-1} \times p_j$
- Order of $A_{i..j}$ = rows in A_i x columns in $A_j = p_{i-1} \times p_j$
- At the highest level of parenthesisation,
 $A_{i..j} = A_{i..k} \times A_{k+1..j} \quad i \leq k < j$
- Let $m[i, j]$ = minimum number of multiplications needed to compute $A_{i..j}$, for $1 \leq i \leq j \leq n$
- Objective function = finding minimum number of multiplications needed to compute $A_{1..n}$ i.e. to compute $m[1, n]$
- $A_{i..j} = (A_i \cdot A_{i+1} \cdot \dots \cdot A_k) \cdot (A_{k+1} \cdot A_{k+2} \cdot \dots \cdot A_j) = A_{i..k} \times A_{k+1..j} \quad i \leq k < j$
- Order of $A_{i..k} = p_{i-1} \times p_k$, and order of $A_{k+1..j} = p_k \times p_j$,
- $m[i, k]$ = minimum number of multiplications needed to compute $A_{i..k}$
- $m[k+1, j]$ = minimum number of multiplications needed to compute $A_{k+1..j}$

Mathematical Model

$$m[i, j] = 0$$

$$m[i, j] = \min_{i \leq k < j} (m[i, k] + m[k+1, j] + p_{i-1} p_k p_j)$$

Example: Compute optimal multiplication order for a series of matrices given below

$$\frac{A_1}{10 \times 100} \cdot \frac{A_2}{100 \times 5} \cdot \frac{A_3}{5 \times 50} \cdot \frac{A_4}{50 \times 20}$$

m[1,1]	m[1,2]	m[1,3]	m[1,4]
	m[2,2]	m[2,3]	m[2,4]
		m[3,3]	m[3,4]
			m[4,4]

$$P_0 = 10$$

$$P_1 = 100$$

$$P_2 = 5$$

$$P_3 = 50$$

$$P_4 = 20$$

$$m[i, i] = 0, \forall i = 1, \dots, 4$$

$$m[i, j] = \min_{i \leq k < j} (m[i, k] + m[k + 1, j] + p_{i-1} \cdot p_k \cdot p_j)$$

Main Diagonal:

$$m[1, 1] = 0, \quad m[2, 2] = 0$$

$$m[3, 3] = 0, \quad m[4, 4] = 0$$

Computing m[1, 2]

$$m[i, j] = \min_{i \leq k < j} (m[i, k] + m[k + 1, j] + p_{i-1} \cdot p_k \cdot p_j)$$

$$m[1, 2] = \min_{1 \leq k < 2} (m[1, k] + m[k + 1, 2] + p_0 \cdot p_k \cdot p_2)$$

$$m[1, 2] = \min (m[1, 1] + m[2, 2] + p_0 \cdot p_1 \cdot p_2)$$

$$m[1, 2] = 0 + 0 + 10 \cdot 100 \cdot 5 = 5000$$

$$s[1, 2] = k = 1$$

Computing m[2, 3]

$$m[i, j] = \min_{i \leq k < j} (m[i, k] + m[k + 1, j] + p_{i-1} \cdot p_k \cdot p_j)$$

$$m[2, 3] = \min_{2 \leq k < 3} (m[2, k] + m[k + 1, 3] + p_1 \cdot p_k \cdot p_3)$$

$$m[2, 3] = \min (m[2, 2] + m[3, 3] + p_1 \cdot p_2 \cdot p_3)$$

$$m[2, 3] = 0 + 0 + 100 \cdot 5 \cdot 50 = 25000$$

$$s[2, 3] = k = 2$$

Computing m[3, 4]

$$m[i, j] = \min_{i \leq k < j} (m[i, k] + m[k+1, j] + p_{i-1} \cdot p_k \cdot p_j)$$

$$m[3, 4] = \min_{3 \leq k < 4} (m[3, k] + m[k+1, 4] + p_2 \cdot p_k \cdot p_4)$$

$$m[3, 4] = \min (m[3, 3] + m[4, 4] + p_2 \cdot p_3 \cdot p_4)$$

$$m[3, 4] = 0 + 0 + 5 \cdot 50 \cdot 20 = 5000$$

$$s[3, 4] = k = 3$$

Computing m[1, 3]

$$m[i, j] = \min_{i \leq k < j} (m[i, k] + m[k+1, j] + p_{i-1} \cdot p_k \cdot p_j)$$

$$m[1, 3] = \min_{1 \leq k < 3} (m[1, k] + m[k+1, 3] + p_0 \cdot p_k \cdot p_3)$$

$$m[1, 3] = \min (m[1, 1] + m[2, 3] + p_0 \cdot p_1 \cdot p_3,$$

$$m[1, 2] + m[3, 3] + p_0 \cdot p_2 \cdot p_3))$$

$$m[1, 3] = \min (0 + 25000 + 10 \cdot 100 \cdot 50, 5000 + 0 + 10 \cdot 5 \cdot 50) \\ = \min (75000, 2500) = 2500$$

$$s[1, 3] = k = 2$$

Computing m[2, 4]

$$m[i, j] = \min_{i \leq k < j} (m[i, k] + m[k+1, j] + p_{i-1} \cdot p_k \cdot p_j)$$

$$m[2, 4] = \min_{2 \leq k < 4} (m[2, k] + m[k+1, 4] + p_1 \cdot p_k \cdot p_4)$$

$$m[2, 4] = \min (m[2, 2] + m[3, 4] + p_1 \cdot p_2 \cdot p_4,$$

$$m[2, 3] + m[4, 4] + p_1 \cdot p_3 \cdot p_4))$$

$$m[2, 4] = \min (0 + 5000 + 100 \cdot 5 \cdot 20, 25000 + 0 + 100 \cdot 50 \cdot 20) \\ = \min (15000, 35000) = 15000$$

$$s[2, 4] = k = 2$$

Computing m[1, 4]

$$m[i, j] = \min_{i \leq k < j} (m[i, k] + m[k+1, j] + p_{i-1} \cdot p_k \cdot p_j)$$

$$m[1, 4] = \min_{1 \leq k < 4} (m[1, k] + m[k+1, 4] + p_0 \cdot p_k \cdot p_4)$$

$$m[1, 4] = \min (m[1, 1] + m[2, 4] + p_0 \cdot p_1 \cdot p_4,$$

$$m[1, 2] + m[3, 4] + p_0 \cdot p_2 \cdot p_4, m[1, 3] + m[4, 4] + p_0 \cdot p_3 \cdot p_4)$$

$$\begin{aligned}
 m[1, 4] &= \min(0 + 15000 + 10 \cdot 100 \cdot 20, 5000 + 5000 + 10 \cdot 5 \cdot 20, 2500 + 0 + 10 \cdot 50 \cdot 20) \\
 &= \min(35000, 11000, 35000) = 11000 \\
 s[1, 4] &= k = 2
 \end{aligned}$$

Final Cost Matrix and Its Order of Computation

Final cost matrix	0	5000	2500	11000
		0	25000	15000
			0	5000
				0

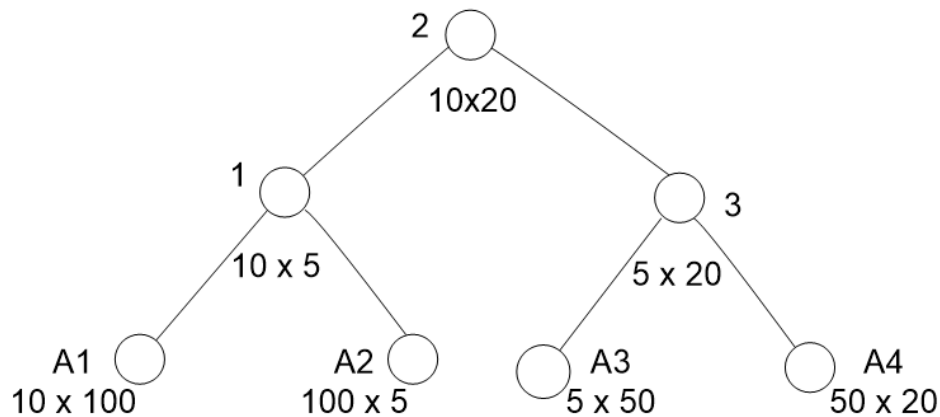
Order of Computation	1	5	8	10
		2	6	9
			3	7
				4

k's Values Leading Minimum $m[i, j]$	0	1	2	2
		0	2	2
			0	3
				0

Representing Order using Binary Tree

- The above computation shows that the minimum cost for multiplying those four matrices is 11000.
- The optimal order for multiplication is $((A_1 \cdot A_2) \cdot (A_3 \cdot A_4))$ for, $m(1, 4)$

$$k = 2$$



Chain-Matrix-Order(p)

1. $n \leftarrow \text{length}[p] - 1$
2. for $i \leftarrow 1$ to n
3. do $m[i, i] \leftarrow 0$
4. for $l \leftarrow 2$ to n ,
5. do for $i \leftarrow 1$ to $n-l+1$
6. do $j \leftarrow i+l-1$
7. $m[i, j] \leftarrow \infty$
8. for $k \leftarrow i$ to $j-1$
9. do $q \leftarrow m[i, k] + m[k+1, j] + p_{i-1} \cdot p_k \cdot p_j$
10. if $q < m[i, j]$
11. then $m[i, j] = q$
12. $s[i, j] \leftarrow k$
13. return m and s , “ l is chain length”

Computational Cost

$$T(n) = n + \sum_{i=1}^n \sum_{j=i+1}^n (j-i) = \sum_{i=1}^n \sum_{k=1}^{n-i} k$$

$$T(n) = n + \sum_{i=1}^n \frac{(n-i)(n-i+1)}{2}$$

$$T(n) = n + \frac{1}{2} \sum_{i=1}^n (n^2 - 2ni + i^2 + n - i)$$

$$T(n) = n + \frac{1}{2} \left(\sum_{i=1}^n n^2 - \sum_{i=1}^n 2ni + \sum_{i=1}^n i^2 + \sum_{i=1}^n n - \sum_{i=1}^n i \right)$$

$$T(n) = n + \frac{1}{2} \left(n^2 \sum_{i=1}^n 1 - 2n \sum_{i=1}^n i + \sum_{i=1}^n i^2 + n \sum_{i=1}^n 1 - \sum_{i=1}^n i \right)$$

$$T(n) = n + \frac{1}{2} \left(n^2 \cdot n - 2n \cdot \frac{n(n+1)}{2} + \frac{n(n+1)(2n+1)}{6} + n \cdot n - \frac{n(n+1)}{2} \right)$$

$$T(n) = n + \frac{1}{2} \left(n^3 - n^2(n+1) + \frac{n(n+1)(2n+1)}{6} + n^2 - \frac{n(n+1)}{2} \right)$$

$$T(n) = n + \frac{1}{12} (6n^3 - 6n^3 - 6n^2 + 2n^3 + 3n^2 + n + 6n^2 - 3n^2 - 3n)$$

$$T(n) = \frac{1}{12} (12n + 2n^3 - 2n) = \frac{1}{12} (10n + 2n^3) = \frac{1}{6} (5n + n^3)$$

Cost Comparison Brute Force Dynamic Programming

A simple inspection of the nested loop structure yields a running time of $O(n^3)$ for the algorithm. The loops are nested three deep, and each loop index (l , i , and k) takes on at most $n-1$ values.

Brute Force Approach: $P(n) = C(n-1) \cdot C(n) \in (4^n/n^{3/2})$

Generalization: Sequence of Objects

Although this algorithm applies well to the problem of matrix chain multiplication. Many researchers have noted that it generalizes well to solving a more abstract problem

- given a linear sequence of objects
- an associative binary operation on those objects hold
- the objective to find a way to compute the cost of performing that operation on any two given objects
- and finally computing the minimum cost for grouping these objects to apply the operation over the entire sequence.

It is obvious that this problem can be solved using chain matrix multiplication, because there is a one to one correspondence between both problems.

Generalization: String Concatenation

One common special case of chain matrix multiplication problem is string concatenation.

- For example, we are given a list of strings.
 - The cost of concatenating two strings of length m and n is for example $O(m + n)$
 - Since we need $O(m)$ time to find the end of the first string and $O(n)$ time to copy the second string onto the end of it.
 - Using this cost function, we can write a dynamic programming algorithm to find the fastest way to concatenate a sequence of strings
 - It is possible to concatenate all in time proportional to sum of their lengths, but here we are interested to link this problem with chain matrix multiplication.

Generalization: Parallel Processors

- Another generalization is to solve the problem when many parallel processors are available.
- In this case, instead of adding the costs of computing each subsequence, we just take the maximum, because we can do them both simultaneously.
- This can drastically affect both the minimum cost and the final optimal grouping
- But of course more balanced groupings that keep all the processors busy is more favorable solution
- There exists some more sophisticated approaches to solve this problem