CS207 Design and Analysis of Algorithms

Sajith Gopalan

Indian Institute of Technology Guwahati

sajith@iitg.ac.in

January 30, 2022

Dynamic Programming

Dynamic Programming

- Particularly useful for optimization problems. Solution space. Constraints. Many feasible solutions, each of which satisfies the constraints, and has a value. We wish to find one feasible solution that minimises (maximizes) value.
- Steps:
  - Formulate the structure of an optimal solution
  - Recursively define the value of an optimal solution
  - Compute the value of an optimal solution, typically in a bottom-up fashion
  - Construct an optimal solution from computed information
- Works particularly when the following properties hold:
  - Optimal Substructure: an optimal solution to the problem contains within it optimal solutions to subproblems
  - Overlapping subproblems: recursive algorithm for the problem solves the same subproblems repeatedly

Matrix multiplication chain

- Consider matrices A_1, A_2, A_3, A_4 and A_5 of dimensions 7×1, 1×3, 3×5, 5×6, and 6×4 respectively
- Then the product $A_1 A_2 A_3 A_4 A_5$ is defined
- Matrix multiplication is associative; $(A_1 A_2) A_3 = A_1 (A_2 A_3)$; so parenthesization does not affect the result
- However parenthesization can affect the time complexity
- A $p \times q$ matrix and a $q \times r$ matrix can be multiplied using pqr scalar multiplications in a triple-for-loop
- If $A_1(((A_2 A_3) A_4) A_5)$ is the prenthesization used, then it takes $1*3*5+1*5*6+1*6*4+7*1*4 = 15+30+24+28 = 97$ scalar multiplications
- If $((((A_1 A_2) A_3) A_4) A_5)$ is the prenthesization used, then it takes $7*1*3+7*3*5+ 7*5*6+7*6*4 = 21+105+210+168 = 504$ scalar multiplications

Matrix multiplication chain

- Given is a sequence $p_0, p_1, \ldots, p_n$ of positive integers, where $p_{i-1} \times p_i$ is the dimension of matrix A_i
- Find the parenthesization that will minimize the number of scalar multiplications needed to compute $A_1 \ldots A_n$
- This is an optimization problem
- It is not feasible to look at every possible parenthesization
- The number of possible parenthesizations can be obtained using the following recurrence relation:

$$P(n) = \left\{ \begin{array}{ll} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n > 1 \end{array} \right\}$$

Matrix multiplication chain

- It is easy to see that $P(n+1) = C(n)$, the n-th Catalan number
- Recall,

$$C(n) = \left\{ \begin{array}{ll} 1 & \text{if } n = 0 \\ \sum_{k=0}^{n-1} C(k)C(n-k-1) & \text{if } n > 1 \end{array} \right\}$$

- But $C(n) = \frac{1}{n+1}\binom{2n}{n} = \omega(2^n)$
- It is not feasible to look at every possible parenthesization

Matrix multiplication chain: Recursive formulation

- Suppose $m[i,j]$ is the minimum number of scalar multiplications needed to compute $A_i \ldots A_j$
- As $A_i \ldots A_i = A_i$, $m[i,i] = 0$, for $1 \leq i \leq n$
- For $1 \leq i < j \leq n$,

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

- If we choose to split $A_i \ldots A_j$ into $(A_i \ldots A_k)$ and $(A_{k+1} \ldots A_j)$, then we must minimize the number of scalar multiplications needed to compute either and add it to the cost of computing the final product.
- The last involves multiplying a $p_{i-1} \times p_k$ matrix with a $p_k \times p_j$ matrix, and so takes $p_{i-1}p_kp_j$ scalar multiplications
- To compute $m[i,j]$, m-values must be known for pairs that are closer than $j - i$ to each other

Iterative Matrix Multiplication Chain

$m[1 \dots n, 1 \dots n]$ and $s[1 \dots n, 1 \dots n]$ are global arrays

Algorithm 1 IMMC(p, n)

Initialize m to zero at the diagonals and ∞ everywhere else
for $d = 1$ to $n - 1$ **do**
  for $i = 1$ to $n - d$ **do**
    $j = i + d$;
    for $k = i$ to $j - 1$ **do**
      $t = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$
      if $(t < m[i, j])$ **then**
        $m[i, j] = t$; $s[i, j] = k$
      end if
    end for
  end for
end for

Example

$$\langle p_0, p_1, p_2, p_3, p_4, p_5 \rangle = \langle 7, 1, 3, 5, 6, 4 \rangle$$

	1	2	3	4	5
1	0				
2		0			
3			0		
4				0	
5					0

Example

- $m[1,2] = 7 * 1 * 3 = 21$
- $m[2,3] = 1 * 3 * 5 = 15$
- $m[3,4] = 3 * 5 * 6 = 90$
- $m[4,5] = 5 * 6 * 4 = 120$

Example

$\langle p_0, p_1, p_2, p_3, p_4, p_5 \rangle = \langle 7, 1, 3, 5, 6, 4 \rangle$

	1	2	3	4	5
1	0	21			
2		0	15		
3			0	90	
4				0	120
5					0

Example

- $m[1,3] = \min\{m[1,2] + 7 * 3 * 5, m[2,3] + 7 * 1 * 5\} = 50$, $s[1,3] = 1$
- $m[2,4] = \min\{m[2,3] + 1 * 5 * 6, m[3,4] + 1 * 3 * 6\} = 45$, $s[2,4] = 3$
- $m[3,5] = \min\{m[3,4] + 3 * 6 * 4, m[4,5] + 3 * 5 * 4\} = 162$, $s[3,5] = 4$

Example

$$\langle p_0, p_1, p_2, p_3, p_4, p_5 \rangle = \langle 7, 1, 3, 5, 6, 4 \rangle$$

	1	2	3	4	5
1	0	21	50		
2		0	15	45	
3			0	90	162
4				0	120
5					0

Example

$$
\begin{aligned}
m[1,4] = \min\{&m[1,1] + m[2,4] + 7 * 1 * 6,\\
&m[1,2] + m[3,4] + 7 * 3 * 6,\\
&m[1,3] + m[4,4] + 7 * 5 * 6\} = 87\\
s[1,4] = 1&\\
m[2,5] = \min\{&m[2,2] + m[3,5] + 1 * 3 * 4,\\
&m[2,3] + m[4,5] + 1 * 5 * 4,\\
&m[2,4] + m[5,5] + 1 * 6 * 4\} = 69\\
s[2,5] = 4&
\end{aligned}
$$

Example

$\langle p_0, p_1, p_2, p_3, p_4, p_5 \rangle = \langle 7, 1, 3, 5, 6, 4 \rangle$

	1	2	3	4	5
1	0	21	50	87	
2		0	15	45	69
3			0	90	162
4				0	120
5					0

Example

$$
\begin{aligned}
m[1,5] = \min\{&m[1,1] + m[2,5] + 7*1*4,\\
&m[1,2] + m[3,5] + 7*3*4,\\
&m[1,3] + m[4,5] + 7*5*4,\\
&m[1,4] + m[5,5] + 7*6*4\} = 97
\end{aligned}
$$

$$s[1,5] = 1$$

Example

$\langle p_0, p_1, p_2, p_3, p_4, p_5 \rangle = \langle 7, 1, 3, 5, 6, 4 \rangle$

	1	2	3	4	5
1	0	21	50	87	97
2		0	15	45	69
3			0	90	162
4				0	120
5					0

Analysis

- IMMC runs in $O(n^3)$ time
- It can be shown that it runs in $\Theta(n^3)$ time: Exercise!