Design and Analysis of Algorithms

Lecture-32

Dharmendra Kumar (Associate Professor)
Department of Computer Science and Engineering
United College of Engineering and Research,
Prayagraj

Unit-4

Dynamic Programming

Dynamic programming

- Dynamic programming, like the divide-and-conquer method, solves problems by combining the solutions to subproblems.

- In contrast to the divide-and-conquer method, dynamic programming applies when the subproblems overlap—that is, when subproblems share subsubproblems.

- A dynamic-programming algorithm solves each subsubproblem just once and then saves its answer in a table.

Optimization problems

- We typically apply dynamic programming to ***optimization problems***.

- Such problems can have many possible solutions.
- Each solution has a value, and we wish to find a solution with the optimal (minimum or maximum) value.
- We call such a solution *an* optimal solution to the problem.

Dynamic programming

When developing a dynamic-programming algorithm, we follow a sequence of four steps:

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, typically in a bottom-up fashion.
4) Construct an optimal solution from computed information.

Dynamic programming

Using dynamic programming, we shall solve the following problems:-

- Matrix-chain multiplication
- Longest common subsequence problem
- 0-1 Knapsack problem
- All pairs shortest path problem

Matrix-chain multiplication problem

This problem is stated as following:-

Given a sequence (chain) $< A_1, A_2, \ldots\ldots\ldots, A_n >$ of n matrices where for $i = 1, 2, 3, \ldots\ldots\ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product $A_1A_2 \ldots\ldots\ldots A_n$ in a way that minimizes the number of scalar multiplications.

Example: Find all parenthesization of matrix for n=4.

Solution:

1) ((A1A2)(A3A4))
2) (((A1A2)A3)A4)
3) (A1(A2(A3A4)))
4) ((A1(A2A3))A4)
5) (A1((A2A3)A4))

Matrix-chain multiplication problem

Example: Consider the following chain of matrices < A1, A2, A3 > . The dimension of matrices are the following:-

A1 = 10x100, A2 = 100x5, A3= 5x50

Find the optimal parenthesization of these matrices.

Solution: All the parenthesization of these matrices are:-

1) ((A1A2)A3)
2) (A1(A2A3))

Number of scalar multiplications in (1)

= 10*100*5 + 10*5*50

= 5000 + 2500 = 7500

Number of scalar multiplications in (2)

= 100*5*50 + 10*100*50

= 25000 + 50000 = 75000

Therefore, the solution ((A1A2)A3) is optimal solution.

Number of parenthesizations

Let P(n) denote the number of parenthesizations for n matrices. It is computed as following:-

$P(n) = 1$ if $n=1$

$= \sum_{k=1}^{n-1} P(k)P(n-k)$ if $n >= 2$

Dynamic programming approach for Matrix-chain multiplication problem

Step 1: In this step, we find the optimal substructure and then use it to construct an optimal solution to the problem.

Let $A_{i..j} \rightarrow$ Matrix that results from evaluating the product $A_i A_{i+1} \ldots\ldots\ldots A_j$. Where $i \leq j$.

- If $i < j$, then to parenthesize the product $A_i A_{i+1} \ldots\ldots\ldots A_j$, we must split the product between A_k and A_{k+1} for some integer k in the range $i \leq k < j$. That is, for some value of k, we first compute the matrices $A_{i..k}$ and $A_{k+1..j}$ and then multiply them together to produce the final product $A_{i..j}$.
- The cost of parenthesizing this way is the cost of computing the matrix $A_{i..k}$, plus the cost of computing $A_{k+1..j}$, plus the cost of multiplying them together.

Matrix-chain multiplication problem

Step 2:

- Let m[i, j] be the minimum number of scalar multiplications needed to compute the matrix $A_{i..j}$.
- m[i,j] is recursively defined as following:-

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

- We define s[i, j] to be the value of k at which m[i, k] + m[k+1, j] + $p_{i-1}p_kp_j$ is minimum.
- We compute the optimal solution using matrix s.
- And matrix m is used to compute the value of optimal solution.

Matrix-chain multiplication problem

Step 3: In this step, we compute the matrices m and s.

Following algorithm is used to compute the matrices m and s. This algorithm assumes that matrix A_i has dimensions $p_{i-1} x p_i$ for i = 1, 2, ………, n. Its input is a sequence p = < p_0, p_1, ….., p_n >, where p.*length* = n+1.

```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Matrix-chain multiplication problem

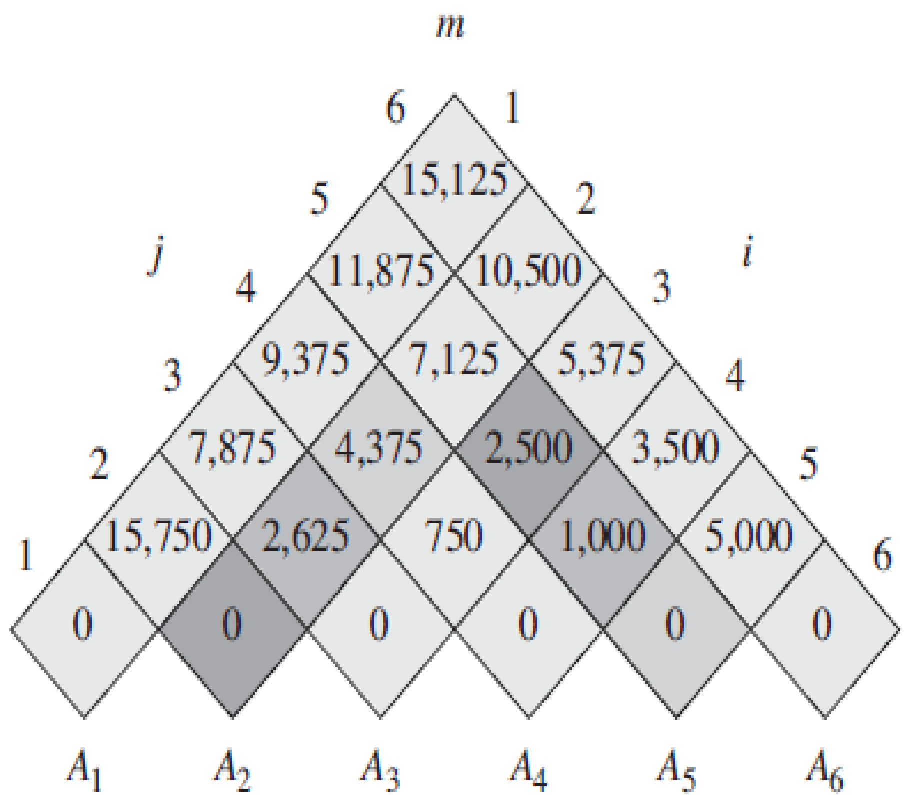
Example: Consider the following matrix chain multiplication problem.

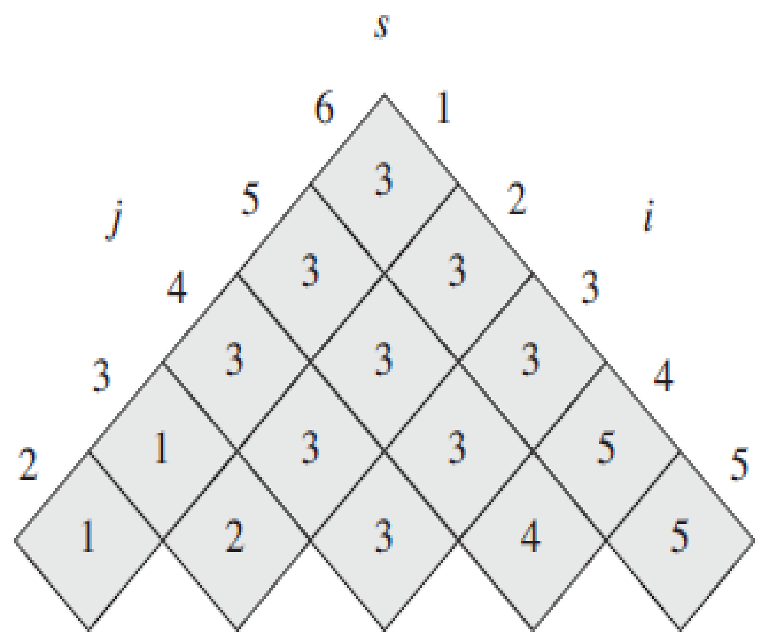
$A_1 \rightarrow 30x35$, $A_2 \rightarrow 35x15$, $A_3 \rightarrow 15x5$,

$A_4 \rightarrow 5x10$, $A_5 \rightarrow 10x20$, $A_6 \rightarrow 20x25$.

Find the optimal solution and its value.

Solution:





Matrix-chain multiplication problem

Step 4: Constructing an optimal solution

Following algorithm is used to create optimal solution i.e. it finds optimal parenthesization.

```
PRINT-OPTIMAL-PARENS(s, i, j)
if i == j
    print "A"_i
else print "("
    PRINT-OPTIMAL-PARENS(s, i, s[i, j])
    PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
    print ")"
```

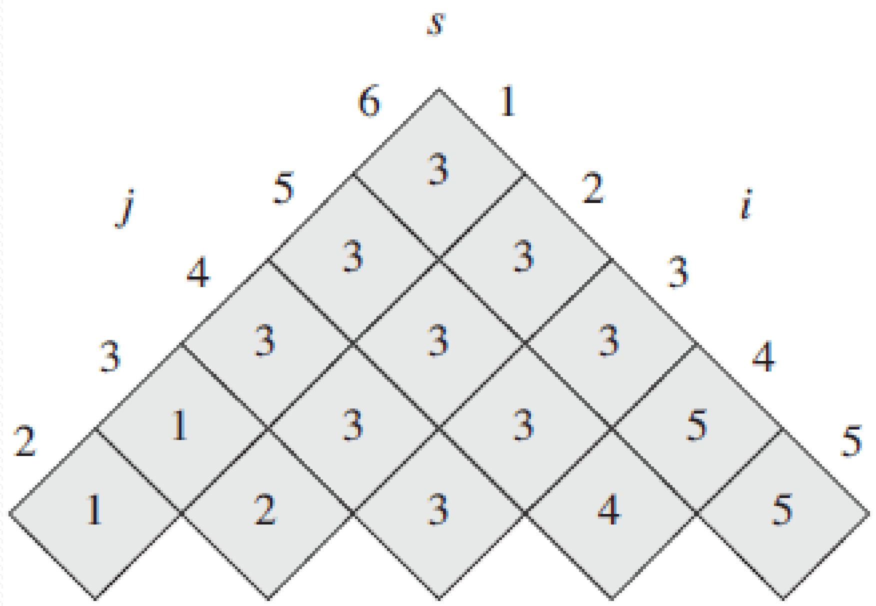
- The initial call PRINT-OPTIMAL-PARENS(s, 1, n) prints an optimal parenthesization of < A_1,A_2 ,……. , A_n >.

Time complexity of this algorithm is O(n).

Matrix-chain multiplication problem

Step 4: Constructing an optimal solution for previous example

The initial call PRINT-OPTIMAL-PARENS(s, 1, 6) prints an optimal parenthesization .



Optimal solution will be

((A_1(A_2A_3))((A_4A_5)A_6))