SLIIT ACADEMY

BSc (IT) – Year 2, Semester 2

Design and Analysis of Algorithms
Dynamic Programming

Anuruddha Abeysinghe
anuruddha.a@sliit.lk

Today's Lecture.

- What is dynamic programming?
- Elements of dynamic programming.
- Matrix multiplication.
  - What is Parenthesization?
    - Optimal parenthesization.
  - Recursive solution for optimal parenthesization.
  - Dynamic programming solution.

What is dynamic programming?

- **Definition:** An algorithmic technique in which an optimization problem is solved by caching sub problem solutions rather than re computing them.
- Similar to divide & conquer, but sub-problems not independent.
- Programming refers to the tabular method.(not writing code)
- Solution to each sub-problem is saved, rather than recomputed.

Optimization Problem

- **Definition**: A computational problem in which the object is to find the best of all possible solutions.
- More formally, find a solution in the feasible region which has the minimum (or maximum) value of the objective function.

When can we use dynamic programming?

- Dynamic programming computes recurrences efficiently by storing partial results.

- Thus dynamic programming can only be efficient when there are not too many partial results to compute.

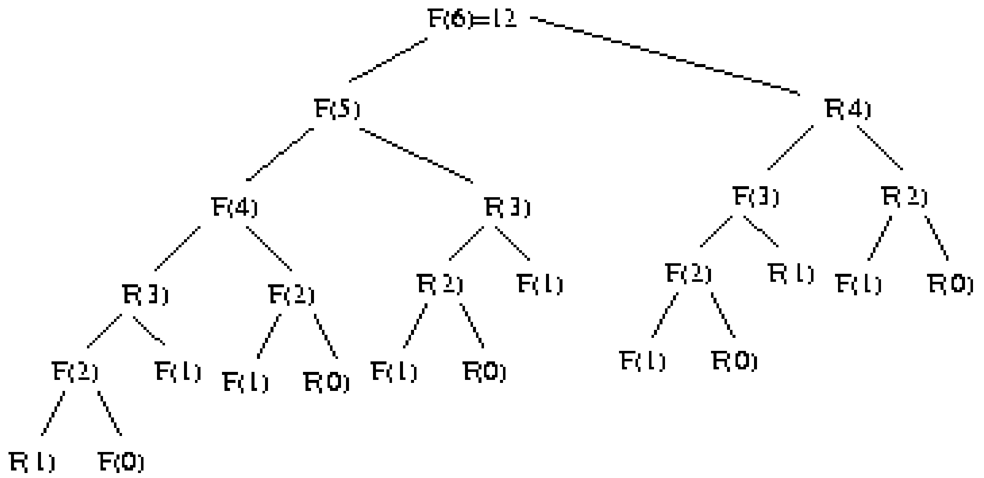
Elements of dynamic programming.

- **Optimal substructure.**

If optimal solution to the problem contains within it’s optimal solutions to sub problems, Then we say that the problem exhibits optimal substructure.

- **Overlapping sub problems.**

When recursive algorithm revisits same sub a problem over and over again, then we say that the optimization problem has overlapping sub problem.



Matrix-chain Multiplication

Consider the matrix multiplication procedure

```
MATRIX_MULTIPLY(A,B)
if columns[A] ≠ rows[B]
    then error "incompatible dimensions”
    else for i ← 1 to rows[A]
             do for j ←1 to columns[B]
                 do C[i,j] ←0;
                     for k ← 1 to columns [A]
                         do C[i,j] ← C[i,j]+A[i,k]*B[k,j];
         return C
```

Scalar Multiplication

- The time to compute a matrix product is dominated by the number of scalar multiplications in line 7.

- If matrix A is of size (**p×q**) and B is of size (**q×r**), then the time to compute the product matrix is given by **pqr**.

Paraenthesization

Example:

Consider three matrices A1, A2, and A3 whose dimensions are respectively (10×100), (100×5) and (5×50).

Now there are two ways to parenthesize these multiplications

I $((A_1 \times A_2) \times A_3)$

II (A1× (A2×A3))

First Parenthesization

Product $A_1 \times A_2$ requires $10 \times 100 \times 5 = 5000$ scalar multiplications.
$A_1 \times A_2$ is a (10×5) matrix

$(A_1 \times A_2) \times A_3$ requires $10 \times 5 \times 50 = 2500$ scalar multiplications.
Total : 7,500 multiplications

Second Parenthesization

Product $A2 \times A3$ requires $100 \times 5 \times 50 = 25{,}000$ scalar multiplications
$A2 \times A3$ is a (100×50) matrix

$A1 \times (A2 \times A3)$ requires $10 \times 100 \times 50 = 50{,}000$ scalar multiplications
Total : 75,000 multiplications

The first parenthesization is 10 times faster than the second one!!
How to pick the best parenthesization.

The matrix-chain

- Given a chain $(A_1, A_2, \ldots, A_n)$ of n matrices, where for $i = 1,2, \ldots, n$ matrix A_i has dimension

$$p_{i-1} \times p_i$$

- The order in which these matrices are multiplied together can have a significant effect on the total number of operations required to evaluate the product.

Let, P(n) : The number of alternative parenthesizations of a sequence of n matrices

We can split a sequence of n matrices between kth and (k+1)th matrices for any k = 1, 2, …, n-1 and we can then parenthesize the two resulting subsequences independently,

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k) \cdot P(n-k) & \text{if } n \geq 2 \end{cases}$$

This is exponential in n

Consider $A1 \times A2 \times A3 \times A4$

if k =1, then

$A1 \times (A2 \times (A3 \times A4))$

$A1 \times ((A2 \times A3) \times A4)$

if k =2 then

$(A1 \times A2) \times (A3 \times A4)$

if k =3 then

$((A1 \times A2) \times A3) \times A4$

and $(A1 \times (A2 \times A3)) \times A4$

Structure of the Optimal Parenthesization

$A_{i..j} = A_i \times A_{i+1} \times \ldots \times A_j$

An optimal parenthesization splits the product

$A_{i..j} = (A_i \times A_{i+1} \times \ldots \times A_k) \times (A_{k+1} \times A_{k+2} \times \ldots \times A_j)$

for $1 \leq k < n$

The total cost of computing Ai..j

= cost of computing (Ai×Ai+1× . . . × Ak)

+ cost of computing (Ak+1× Ak+2× . . . × Aj)

+ cost of multiplying the matrices Ai..k and Ak+1..j.

Recursive Solution

- We'll define the value of an optimal solution recursively in terms of the optimal solutions to subproblems.

m[i,j] = minimum number of scalar multiplications needed to compute the matrix **Ai..j**

m[1,n] = minimum number of scalar multiplications needed to compute the matrix **A1..n.**

If **i =j** ; the chain consists of just one matrix

A i..i = Ai - no scalar multiplications

m[i,i] = 0 for **i = 1, 2, …, n.**

Recursive Solution

- m[i,j] = minimum cost of computing the sub products Ai..k and A k+1 ..j + cost of multiplying these two matrices
- Multiplying Ai..k and Ak+1....j takes pi-1.pk.pj scalar multiplications

m[i,j] = m[i,k] + m[k+1,j] + pi-1.pk.pj
for i≤ k < j

Recursive Solution

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

Let s[i,j] be the value of k at which we can split the product Ai× Ai+1× . . . × Aj to obtain the optimal parenthesization.

s[i,j] equals a value of k such that
m[i,j] = m[i,k] + m[k+1,j] + pi-1.pk.pj
for i≤ k < j

Procedure Matrix_Chain_Order (p)

Input: sequence (p0, p1,…pn)

Output: an auxiliary table m[1..n,1.. n] with m[i,j] costs and another auxiliary table s[1.. n,1.. n] with records of index k which achieves optimal cost in computing m[i, j]

```
n ←length[p]-1;
for i ← 1 to n
    do m[i, i] ← 0;
for l ← 2 to n
    do for i ← 1 to n- l +1
        do j ← i + l -1
            m[i, j] ←∝;
            for k ← i to j -1
                do q ← m[i, k]+ m[k +1, j]+ pi-1 pk pj;
                    if q < m[i, j];
                        then m[i, j] ← q;
                            s[i, j] ← k;
return m and s
```

Example – 2019 Jun Past paper

Consider the following set of metrics ,

A_1 - 1 x 3

A_2 - 3 x 2

A_3 - 2 x 6

A_4 - 6 x 4

chains of length 1

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

A_1	1 x 3	$p_0 \times p_1$
A_2	3 x 2	$p_1 \times p_2$
A_3	2 x 6	$p_2 \times p_3$
A_4	6 x 4	$p_3 \times p_4$

$i \backslash j$	1	2	3	4
1	0			
2		0		
3			0	
4				0

m table

$i \backslash j$	2	3	4
1			
2			
3			

s table

m[1,1]=0
m[2,2]=0
m[3,3]=0
m[4,4]=0

chains of length 2

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

A_1	1 x 3	p_0 x p_1
A_2	3 x 2	p_1 x p_2
A_3	2 x 6	p_2 x p_3
A_4	6 x 4	p_3 x p_4

$i \backslash j$	1	2	3	4
1	0	6		
2		0	36	
3			0	48
4				0

m table

$i \backslash j$	2	3	4
1	1		
2		2	
3			3

s table

*m[1,2]=m[1,1]+m[2,2]+ 1*3*2= 6*
*m[2,3]=m[2,2]+m[3,3]+ 3*2*6 =36*
*m[3,4]= m[3,3]+m[4,4]+ 2*6*4 =48*

chains of length 3

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

A_1	1 x 3	p_0 x p_1
A_2	3 x 2	p_1 x p_2
A_3	2 x 6	p_2 x p_3
A_4	6 x 4	p_3 x p_4

i \ j	1	2	3	4
1	0	6	18	
2		0	36	72
3			0	48
4				0

m table

i \ j	2	3	4
1	1	2	
2		2	2
3			3

s table

$$m[1,3]=\min\begin{cases}m[1,1]+m[2,3]+1*3*6=54\\ m[1,2]+m[3,3]+1*2*6=18\end{cases}$$

$$m[2,4]=\min\begin{cases}m[2,2]+m[3,4]+3*2*4=72\\ m[2,3]+m[4,4]+3*6*4=108\end{cases}$$

chains of length 4

A_1	1 x 3	$p_0 \times p_1$
A_2	3 x 2	$p_1 \times p_2$
A_3	2 x 6	$p_2 \times p_3$
A_4	6 x 4	$p_3 \times p_4$

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

$i \backslash j$	1	2	3	4
1	0	6	18	42
2		0	36	72
3			0	48
4				0

m table

$i \backslash j$	2	3	4
1	1	2	3
2		2	2
3			3

s table

$$m[1,4] = \min \begin{cases} m[1,1] + m[2,4] + 1*3*4 = 84 \\ m[1,2] + m[3,4] + 1*2*4 = 62 \\ m[1,3] + m[4,4] + 1*6*4 = 42 \end{cases}$$

Print optimal parenthesis from *s*

Print-Optimal-Parens (*s*, *i*, *j*)

if *i*=*j*

  then print "A"*i*

  else print "("

    Print-Optimal-Parens (*s*, *i*, s[*i*,*j*])

    Print-Optimal-Parens (*s*, *s*[*i*,*j*]+1, *j*)

    print ")"

Print optimal parenthesis from *s*

$i \backslash j$	2	3	4
1	1	2	3
2		2	2
3			3

s table

Print-Optimal-Parens (*s*, *i*, *j*)

if *i*=*j*

  then print “A”*i*

  else print “(“

    Print-Optimal-Parens (*s*, *i*, s[*i*,*j*])

    Print-Optimal-Parens (*s*, *s*[*i*,*j*]+1, *j*)

    print “)“

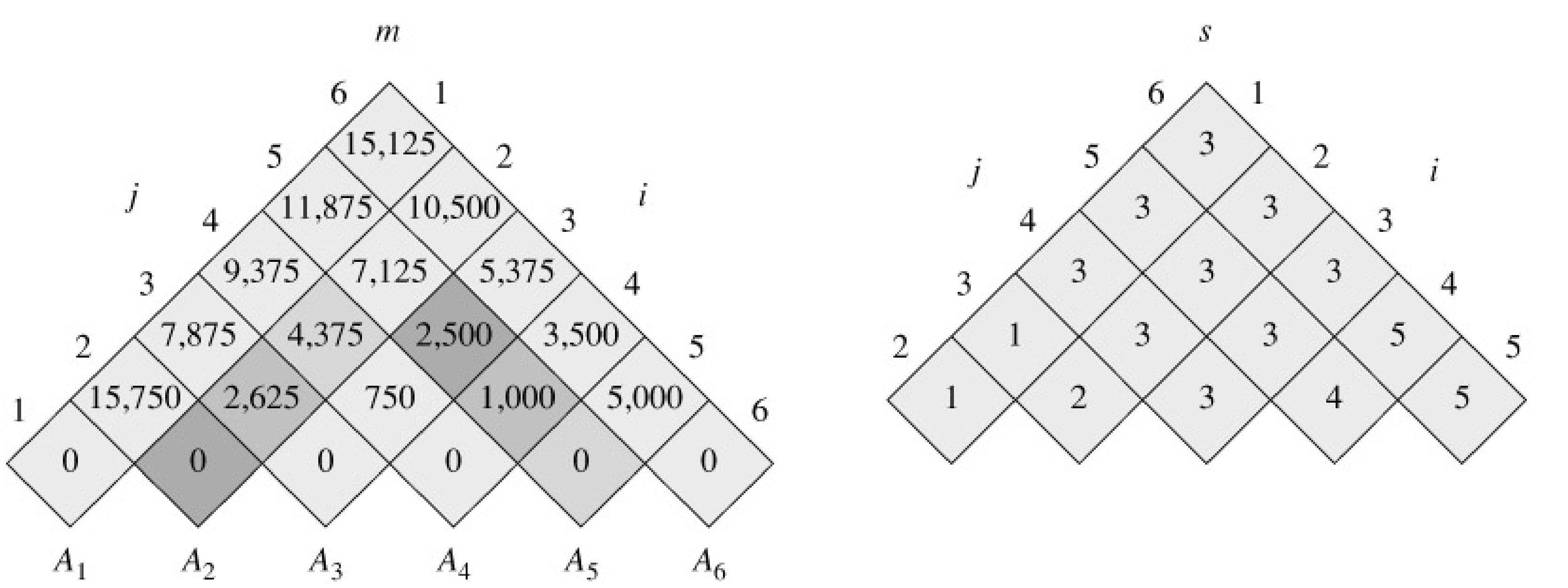
Print(s,1,4)

1) *(*
2) *Print(s,1,3)*
   1. *(*
   2. *Print(s,1,2)*
      I. *(*
      II. *Print(s,1,1) -> A1*
      III. *Print(s,2,2) ->A2*
      IV. *)*
   3. *Print(s,3,3) ->A3*
   4. *)*
3) *Print(s,4,4) ->A4*
4) *)*

(((A1A2)A3)A4)

Home Work

Matrix	Dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25



- Optimal parenthesization is found from the previous Table.

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

Questions ???

Thank You

anuruddha.a@sliit.lk