Design and analysis of algorithm

Rizwan Ul Haq

Dynamic Programming for Solving Optimization Problems

(Chain Matrix Multiplication Problem)

Topic to read

- Optimization problem?
- Steps in Development of Dynamic Algorithms
- Why dynamic in optimization problem?
- Introduction to Catalan numbers
- Chain-Matrix Multiplication
- Problem Analysis
  - Brute Force approach
  - Time Complexity
- Conclusion

Optimization Problems

- If a problem has only ***one correct solution***, then optimization is not required
  - For example, there is only one sorted sequence containing a given set of numbers
- **Optimization problems** have ***many solutions***
  - We want to compute an ***optimal solution e. g.*** with minimal cost and maximal gain
  - There could be many solutions having optimal value
  - Dynamic programming is very effective technique
  - Development of dynamic programming algorithms can be broken into a sequence steps as in the next.

Development of Dynamic

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

   Note: Steps 1-3 form the basis of a dynamic programming solution to a problem. Step 4 can be omitted only if the value of an optimal solution is required.

Why Dynamic Programming

- Dynamic programming, like divide and conquer method, solves problems by combining the solutions to sub-problems
- Divide and conquer algorithms:
  - partition the problem into independent sub-problem
  - Solve the sub-problem recursively and
  - Combine their solutions to solve the original problem
- In contrast, dynamic programming is applicable when the sub-problems are dependent
- Dynamic programming is typically applied to optimization problems

Complexity in Dynamic Algorithms

- Time complexity:
  - If there are polynomial number of sub-problems
  - If each sub-problem can be computed in polynomial time
  - Then the solution of whole problem can be found in polynomial time

Remark:

Greedy also applies a top-down strategy but usually on one sub-problem so that the order of computation is clear

Catalan Numbers

Multiplying n Numbers

Objective:

- Find C(n), the number of ways to compute product $x_1 . x_2 .... x_n$.

n	multiplication order
2	$(x_1 \cdot x_2)$
3	$(x_1 \cdot (x_2 \cdot x_3))$
	$((x_1 \cdot x_2) \cdot x_3)$
4	$(x_1 \cdot (x_2 \cdot (x_3 \cdot x_4)))$
	$(x_1 \cdot ((x_2 \cdot x_3) \cdot x_4))$
	$((x_1 \cdot x_2) \cdot (x_3 \cdot x_4))$
	$((x_1 \cdot (x_2 \cdot x_3)) \cdot x_4)$
	$(((x_1 \cdot x_2) \cdot x_3) \cdot x_4)$

Multiplying n Numbers – small n

n	C_{n-1}	C_{n-1}
1	C_0	1
2	C_1	1
3	C_2	2
4	C_3	5
5	C_4	14
6	C_5	42
7	C_6	132

Multiplying n Numbers - small

Recursive equation:

where is the last multiplication?

$$C(n) = \sum_{k=1}^{n-1} C(k) \cdot C(n-k)$$

Catalan numbers: $C(n) = \frac{1}{n}\binom{2n-2}{n-1}.$

Catalan numbers: $C(n) = \frac{1}{n+1}\binom{2n}{n}.$

Asymptotic value: $C(n) \approx \frac{4^n}{n^{3/2}}$

Chain-Matrix Multiplication

Problem Statement: Chain Matrix Multiplication

Statement: The chain-matrix multiplication problem can be stated as below:

- Given a chain of $[A_1, A_2, \ldots, A_n]$ of n matrices where for i = 1, 2, . . . , 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 $A_1 \times A_2 \times \ldots \times A_n$ is $p_0 \times p_n$

Objective is to find order not multiplication

- Given a sequence of matrices, we want to find a most efficient way to multiply these matrices
- It means that problem is not actually to perform the multiplications but decide the order in which these must be multiplied to reduce the cost
- This problem is an optimization type which can be solved using dynamic programming
- The problem is not limited to find an efficient way of multiplication of matrices but can be used to be applied in various purposes
- But how to transform the original problem into chain matrix multiplication, this is another issue, which is common in systems modeling

Problem is of Optimization

- If these matrices are all square and of same size, the multiplication order will not affect the total cost.
- If matrices are of different sizes but compatible for multiplication, then order can make big difference.
- **Brute Force approach**
  - The number of possible multiplication orders are exponential in n, and so trying all possible orders may take a very long time.
- **Dynamic Programming**
  - To find an optimal solution, we will discuss it using dynamic programming to solve it efficiently.

Assumptions (Only Multiplications Considered)

- We really want is the **minimum cost** to multiply
- But we know that cost of an algorithm **depends** on how many number of **operations** are performed i.e.
  - We must be interested to minimize number of operations, needed to multiply out the matrices.
  - As in matrices multiplication, there will be addition as well multiplication operations in addition to other
  - Since cost of multiplication is **dominated** over addition therefore, we will minimize the number of multiplication operations in this problem.
- In case of **two** matrices, there is only one way to multiply them, so the cost **fixed**.

Chain Matrix Multiplication

(Brute Force Approach)

Force Chain Matrix Multiplication

- If we wish to multiply two matrices:

  A = a[i, j]$_{\mathbf{p,\, q}}$ and **B = b[i, j]$_{\mathbf{q,\, r}}$**
- Now if **C = AB** then order of **C** is **p x r**.
- Since in each entry **c[i, j],** there are **q** number of scalar of multiplications
- Total number of scalar multiplications in computing C = Total entries in C x Cost of computing a single entry = **p . r . q**
- Hence the computational cost of **AB = p . q . r**

$$C[i,j] = \sum_{k=1}^{q} A[i,k]B[k,j]$$

Brute Force Chain Matrix Multiplication

Example

- Given a sequence [A_1, A_2, A_3, A_4]
- Order of A_1 = 10 x 100
- Order of A_2 = 100 x 5
- Order of A_3 = 5x 50
- Order of A_4 = 50x 20

Compute the order of the product $A_1 . A_2 . A_3 . A_4$ in such a way that minimizes the total number of scalar multiplications.

Brute Force Chain Matrix Multiplication

- There are five ways to parenthesize this product
- Cost of computing the matrix product may vary, depending on order of parenthesis.
- All possible ways of parenthesizing

$$(A_1 \cdot (A_2 \,.\, (A_3 \,.\, A_4)))$$

$$(A_1 \cdot ((A_2 \,.\, A_3) .\, A_4))$$

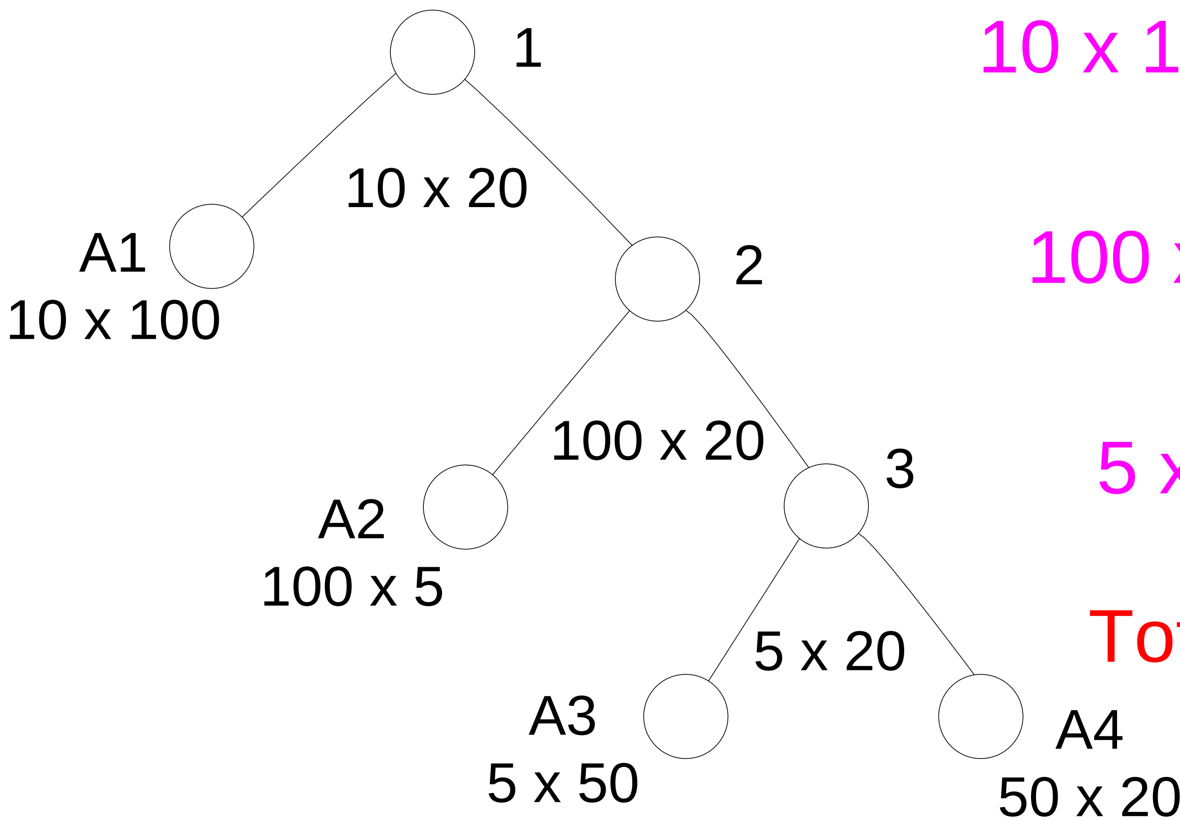
$$((A_1 \cdot A_2) .\, (A_3 \,.\, A_4))$$

$$((A_1 \cdot (A_2 \,.\, A_3)) .\, A_4)$$

$$(((A_1 \cdot A_2) .\, A_3) .\, A_4)$$

Kinds of problems solved by algorithms

First Chain: (A1 . (A2 . (A3. A4))



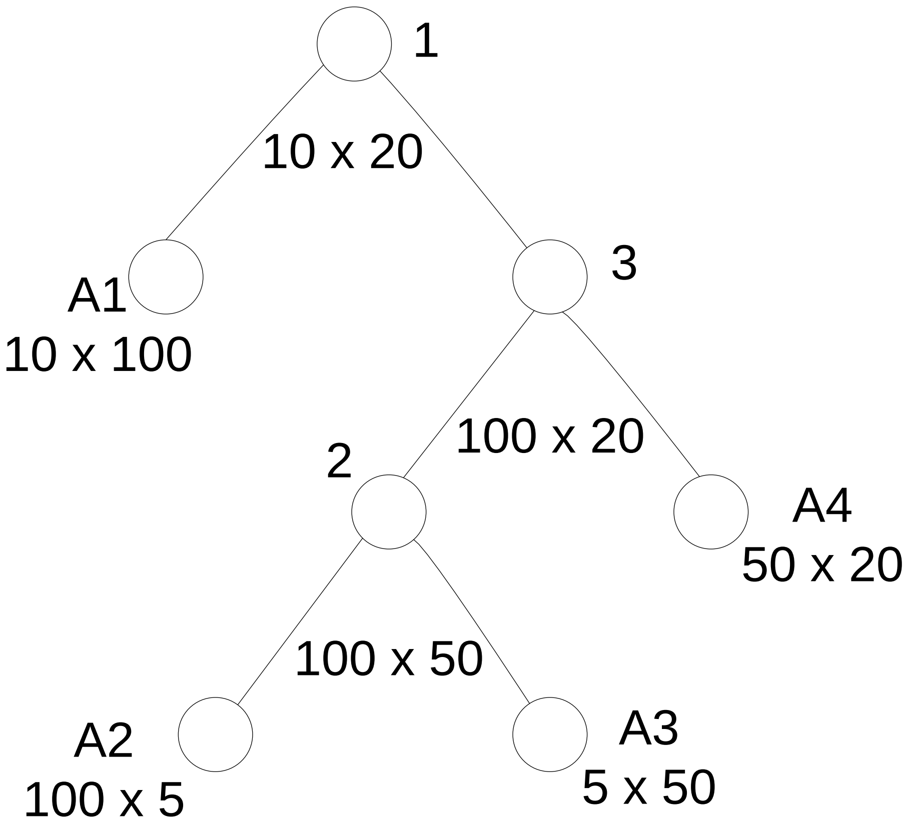
10 x 100 x 20 = 20000

100 x 5 x 20 = 10000

5 x 50 x 20 = 5000

Total Cost = 35000

Second Chain : (A1 · ((A2 . A3). A4))



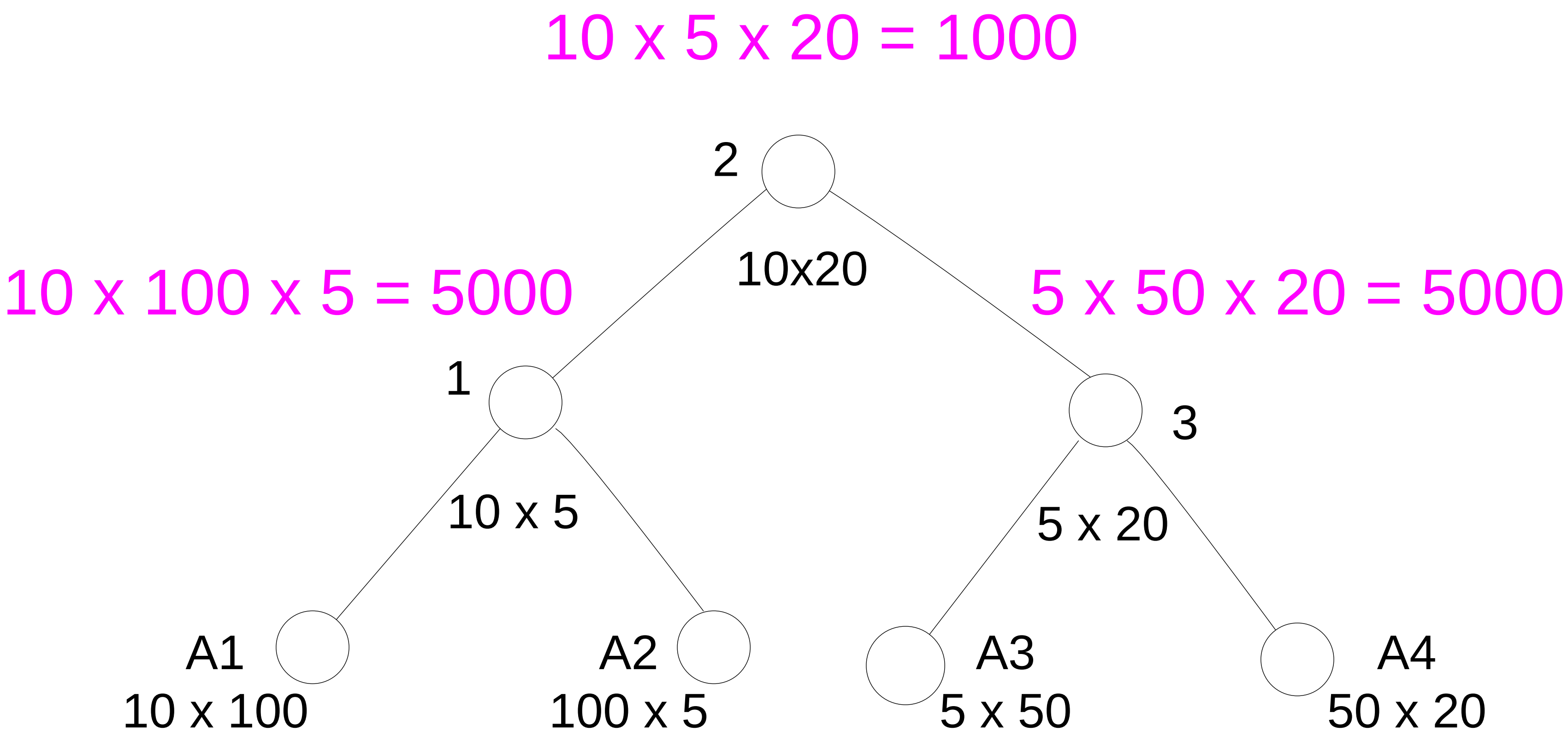
10 x100 x 20 = 20000

100 x 50 x 20 = 100000

100 x 5 x 50 = 25000

Total Cost = 145000

Chain : ((A1 · A2). (A3 . A4))



Total Cost = 11000

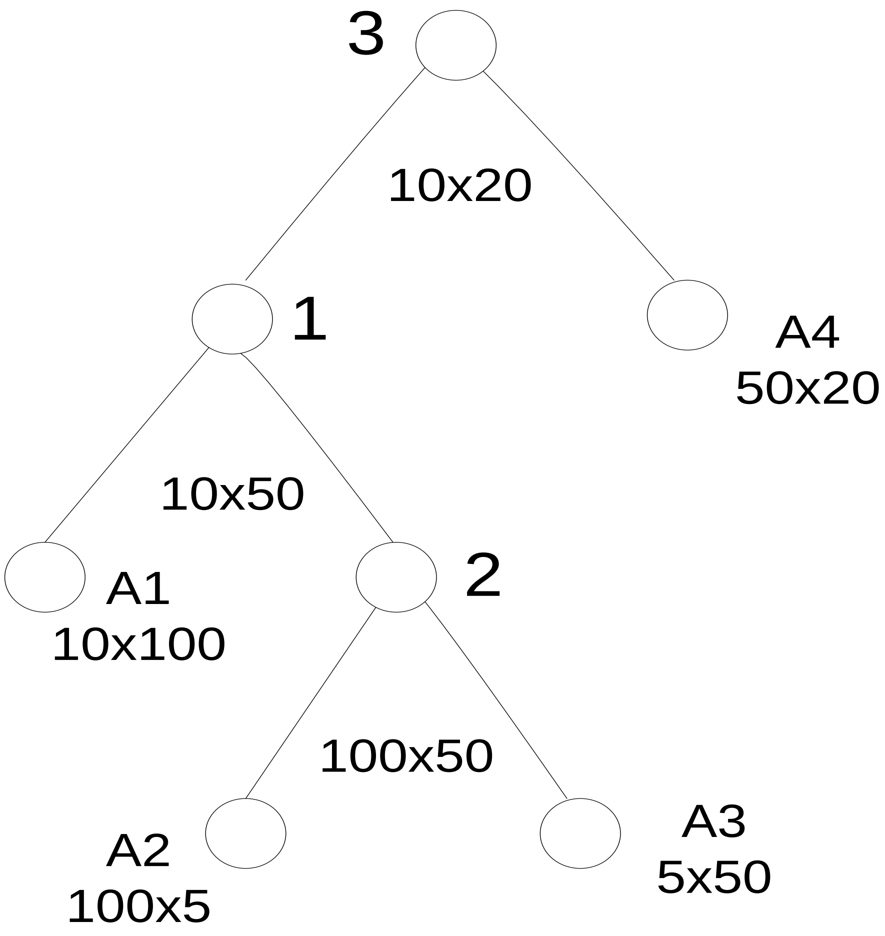
Fourth Chain : ((A1 · (A2 . A3)). A4)

10 x 50 x 20 = 10000

10 x 100 x 50 = 50000

100 x 5 x 50 = 25000

Total Cost = 85000



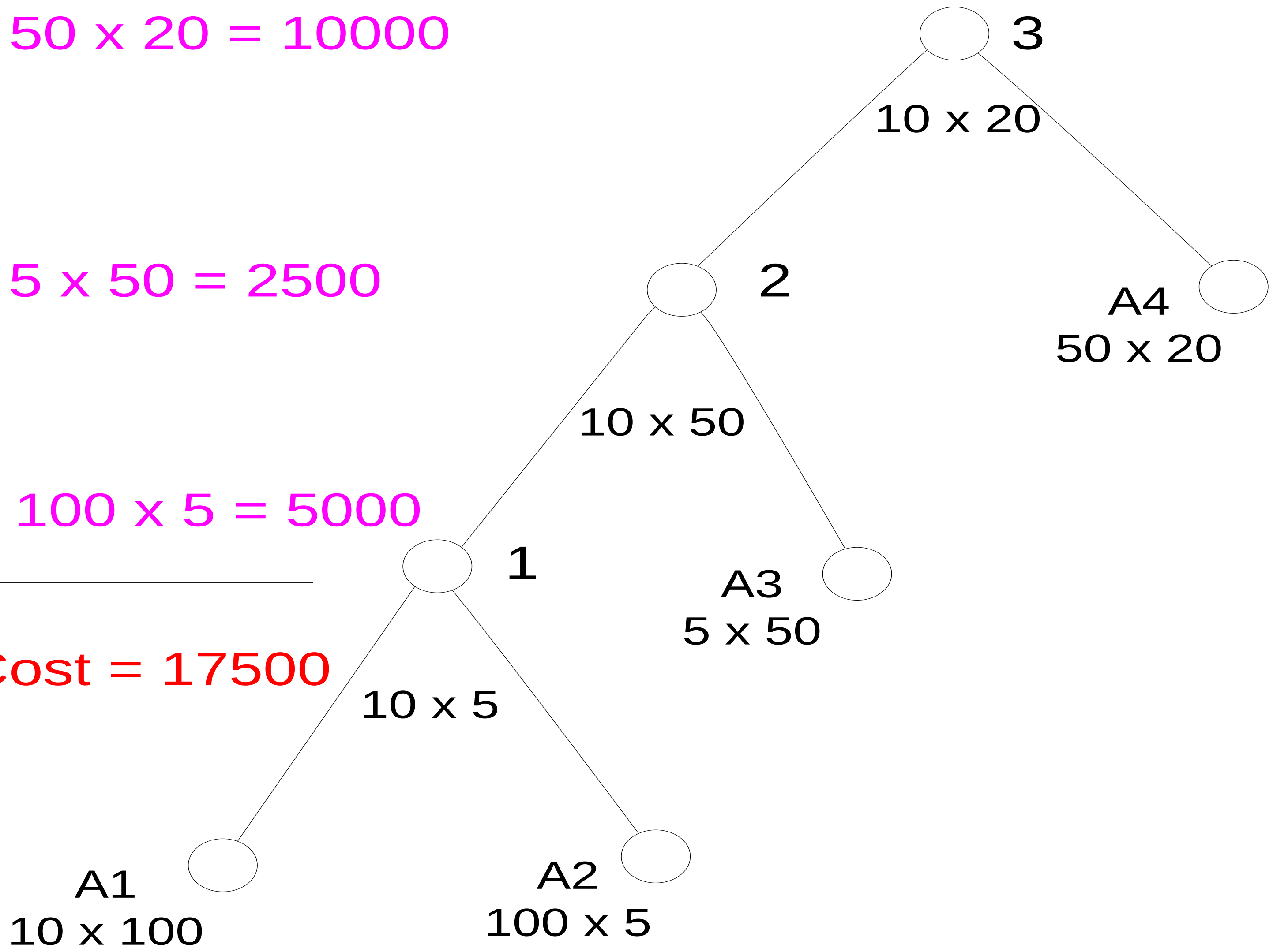
Fifth Chain : (((A1 · A2). A3). A4)

10 x 50 x 20 = 10000

10 x 5 x 50 = 2500

10 x 100 x 5 = 5000

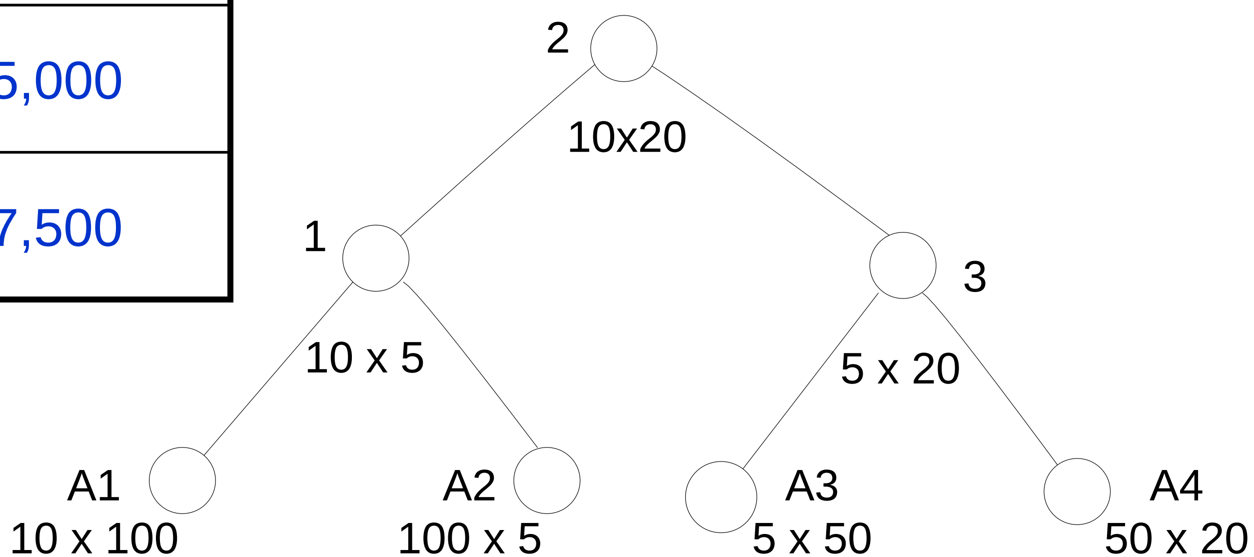
Total Cost = 17500



Chain Matrix Cost

First Chain (A1 . (A2 . (A3. A4))	35,000
Second Chain (A1 · ((A2 . A3). A4))	145,000
Third Chain ((A1 · A2). (A3 . A4))	11,000
Fourth Chain ((A1 · (A2 . A3)). A4)	85,000
Fifth Chain (((A1 · A2). A3). A4)	17,500

((A1 · A2). (A3 . A4))



Generalization of Brute Force Approach

- If there is sequence of **n** matrices, $[A_1, A_2, . . . , A_n]$
- A_i has dimension $p_{i-1} \times p_i$, where for $i = 1, 2, . . . , n$
- Find order of multiplication that minimizes number of scalar multiplications using brute force approach

Recurrence Relation: After **k**th matrix, create two sub-lists, one with k and other with n - k matrices i.e.

$$(A_1\ A_2A_3A_4A_5\ .\ .\ .\ A_k)\ (A_{k+1}A_{k+2}\ldots A_n)$$

- Let **P(n)** be the number of different ways of parenthesizing n items

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

Generalization of Brute Force Approach

If n = 2

$P(2) = P(1).P(1) = 1.1 = 1$

If n = 3

$P(3) = P(1).P(2) + P(2).P(1) = 1.1 + 1.1 = 2$

$(A_1\ A_2A_3) = ((A_1\ .\ A_2).\ A_3)$ OR $(A_1\ .\ (A_2.\ A_3))$

If n = 4

$P(4) = P(1).P(3) + P(2).P(2) + P(3).P(1) = 1.2 + 1.1 + 2.1 = 5$

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

Why Brute Force Approach not Economical

- This is related to a famous function in combinatory called the Catalan numbers
- Catalan numbers are related with the number of different binary trees on n nodes

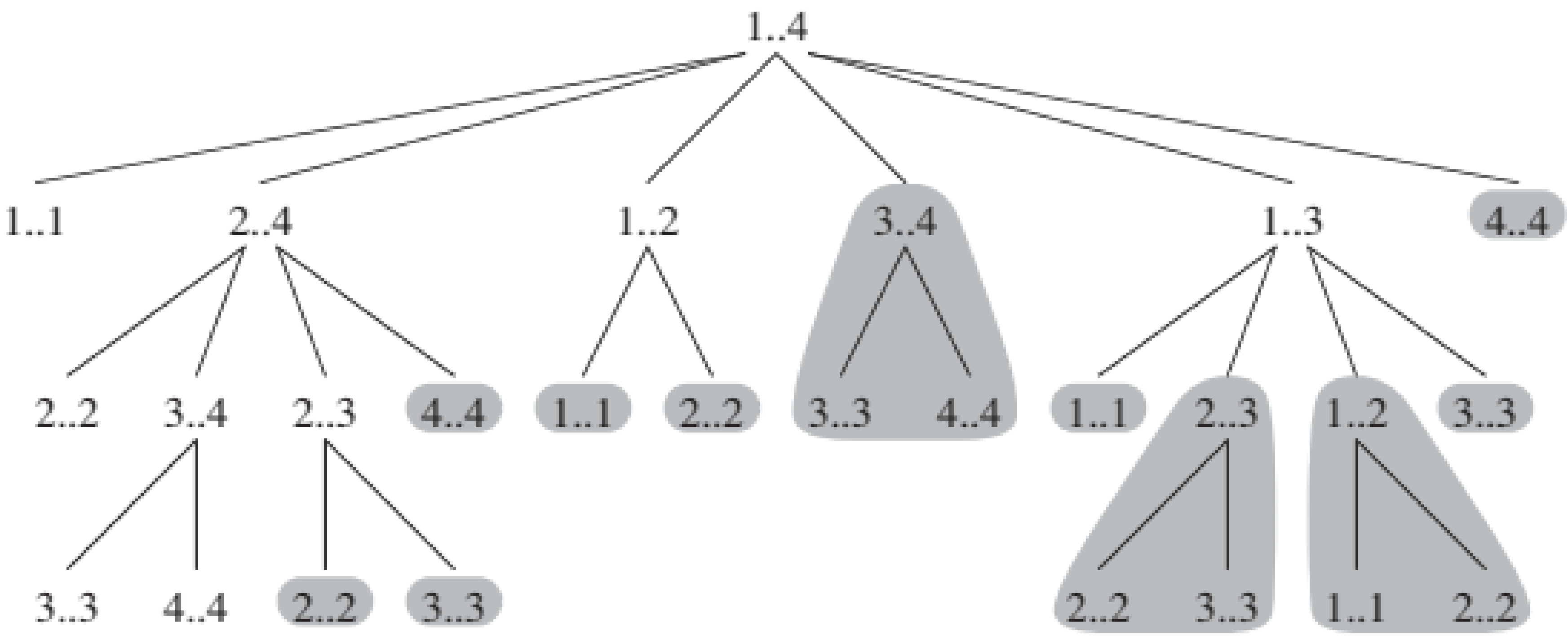
$$\mathbf{P(n) \in (4^n/n^{3/2})}$$

- The dominating term is the exponential 4^n thus P(n) will grow large very quickly
- And hence this approach is not economical

Chain-Matrix-Recursive

```
RECURSIVE-MATRIX-CHAIN(p, i, j)
    if i == j
        return 0
    m[i, j] = ∞
    for k = i to j - 1
        q = RECURSIVE-MATRIX-CHAIN(p, i, k)
            + RECURSIVE-MATRIX-CHAIN(p, k + 1, j)
            + p_{i-1} p_k p_j
        if q < m[i, j]
            m[i, j] = q
    return m[i, j]
```

Recursion Tree for computation of RECURSIVE-MATRIX-CHAIN(p,1,4)



Dynamic Programming Formulation

- Let $A_{i..j} = A_i \,.\, A_{i+1} \,.\,.\,.\, A_j$
- Order of $A_i = p_{i-1} \text{ x } p_i$, and
- Order of $A_j = p_{j-1} \text{ x } p_j$,
- 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,

  $A_{i..j} = A_{i..k} \times A_{k+1..j}$ $\qquad\qquad i \le k < j$
- Let m[i, j] = minimum number of multiplications needed to compute $A_{i..j}$, for $1 \le i \le j \le n$
- **Objective function** = Finding minimum number of multiplications needed to compute $A_{1..n}$ i.e. to compute m[1, n]

Mathematical Model

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

$$i \leq k < j$$

- Order of $\mathbf{A_{i..k} = p_{i-1} \times p_k}$, and order of $\mathbf{A_{k+1..j} = p_k \times p_j}$,
- m[i, k] = minimum number of multiplications needed to compute $A_{i..k}$
- m[k+1, j] = minimum number of multiplications needed to compute $A_{k+1..j}$

Chain-Matrix-Order(p)

```
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] + p_{i-1} . p_k . p_j
                    if q < m[i, j]
                        then m[i, j] = q
                            s[i, j] ← k
return m and s,
```

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]

“l is chain length”

Example: Dynamic Programming

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

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

$P_0 = 10$

$P_1 = 100$

$P_2 = 5$

$P_3 = 50$

$P_4 = 20$

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]

Main Diagonal

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

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

Main Diagonal

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

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

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

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

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

m[1, 2] = 0 + 0 + 10 . 100 . 5

= 5000

s[1, 2] = k = 1

Computing m[2, 3]

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

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

$$m[2,3] = \min (m[2,2] + m[3,3] + p_1.p_2.p3)$$

m[2, 3] = 0 + 0 + 100 . 5 . 50

= 25000

s[2, 3] = k = 2

Computing m[3, 4]

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

$$m[3,4] = \min_{3 \le k < 4} \left(m[3,k] + m[k+1,4] + p_2.p_k.p_4 \right)$$

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

m[3, 4] = 0 + 0 + 5 . 50 . 20

= 5000

s[3, 4] = k = 3

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

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

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

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

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

m[1, 3] = min(0+25000+10.100.50, 5000+0+10.5.50)

= min(75000, 2500) = 2500

s[1, 3] = k = 2

Computing m[2, 4]

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

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

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

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

m[2, 4] = min(0+5000+100.5.20, 25000+0+100.50.20)

= min(15000, 125000) = 15000

s[2, 4] = k = 2

Computing m[1, 4]

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

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

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

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

m[1, 4] = min(0+15000+10.100.20, 5000+5000+
10.5.20, 2500+0+10.50.20)

= min(35000, 11000, 35000) = 11000

s[1, 4] = k = 2

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 to Minimum m[i, j]

0	1	2	2
	0	2	2
		0	3
			0

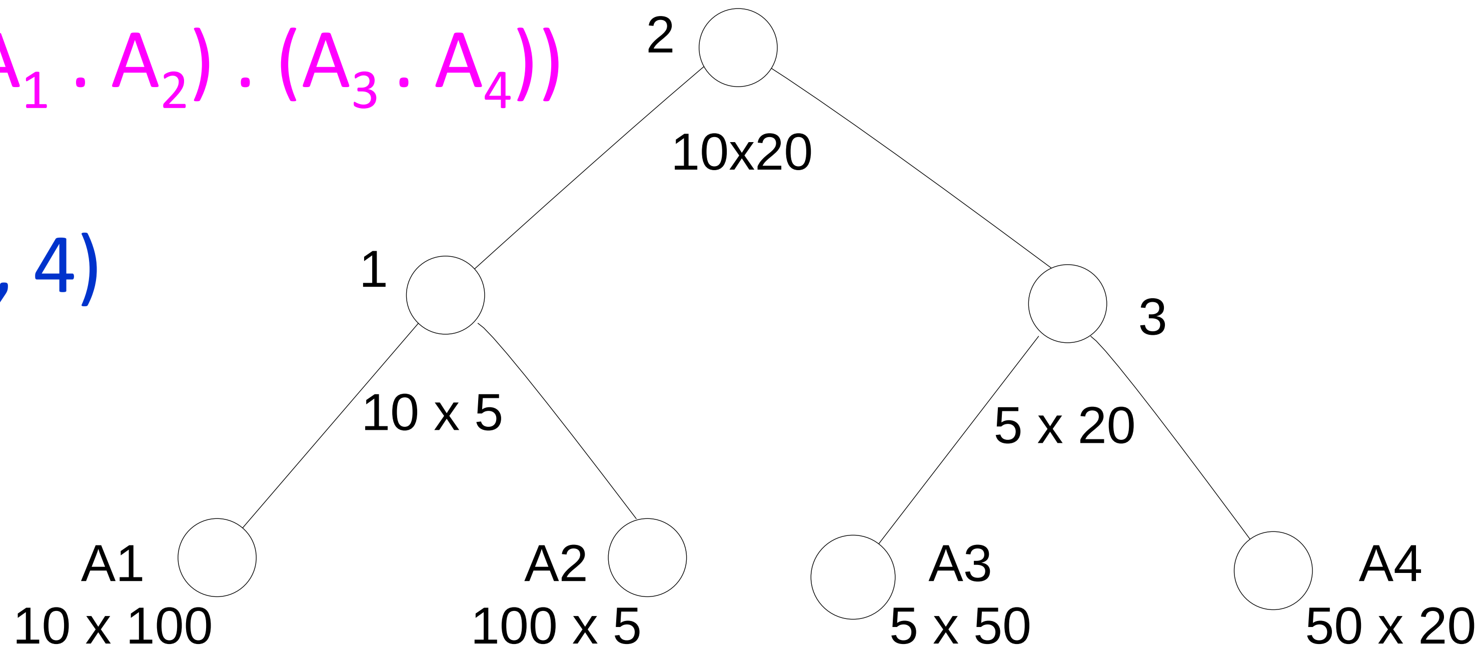
Representing Order using Binary Tree

- The above computation shows that the minimum cost for multiplying those four matrices is 11000.
- The optimal order for multiplication is

  $((A_1 . A_2) . (A_3 . A_4))$

For, m(1, 4)

k = 2



Chain-Matrix-Order(p)

```
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 ← l + l – 1
            m[i, j] ← ∞
            for k ← i to j – 1
                do q ← m[i, k] + m[k+1, j] + p_{i-1} . p_k . p_j
                    if q < m[i, j]
                        then m[i, j] = q
                            s[i, j] ← k
return m and s,
```

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]

“l is chain length”

Comparison Brute Force Dynamic Programming

Dynamic Programming

There are three loop

- The most two loop for i, j, satisfy the condition:

  $1 <= i <= j <= n$

- Cost = ${}^{n}C_{2} + n = n(n-1)/2 + n = \Theta\ (n^2)$
- The third one most inner loop for k satisfies the condition, $i <= k < j$, in worst case, it cost n and
- Hence total cost = $\Theta\ (n^2 . n) = \Theta\ (n^3)$

Brute Force Approach

- $P(n) = C(n - 1)\ C(n) \in\ (4^n/n^{3/2})$