Dynamic Programming (DP)

Dr. Balaprakasa Rao Killi

Rod-cutting problem

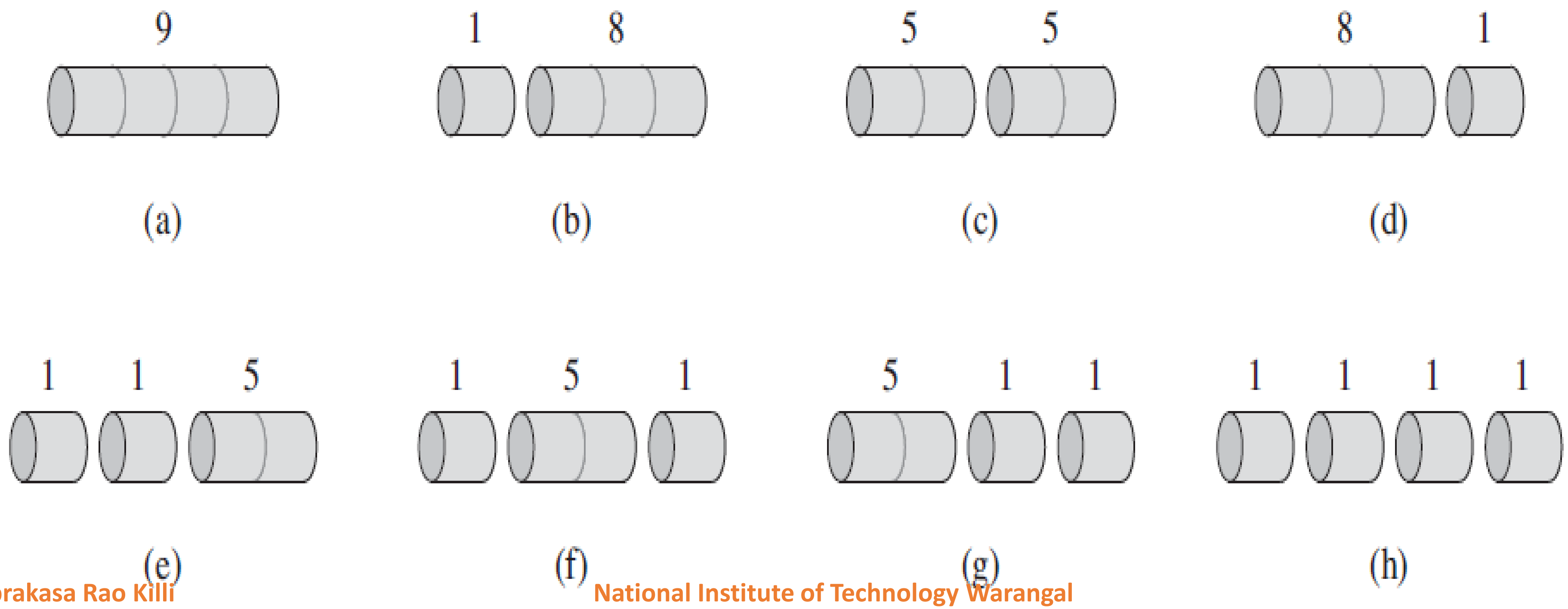
- Input: A large rod of length n inches
- Input: A table of prices p_i , i = 1, 2, 3, ...

- pi is the cost of a rod of length i inches.
- Rod lengths are always an integral number of inches.

- Goal: Determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces.
- If p_n for a rod of length n is large enough, no cutting.

Length	1	2	3	4	5	6	7	8	9	10
Price	1	5	8	9	10	17	17	20	24	30

- When n=4, possible ways of cutting rod



- If an optimal solution cuts the rod into k pieces, for some $p \leq k \leq n$

- optimal solution cuts rods into pieces of length

  $i_1 \; i,_2 , i_3 , \ldots\ldots\ldots , \; i_k$

- Then an optimal decomposition is written as

$$n = i_1 + i_2 + i_3 + \ldots\ldots\ldots I_k$$

- Optimal revenue is

$$r_n = p_{i1} + p_{i2} + p_{i3} + \ldots\ldots\ldots + p_{ik}$$

- Optimal revenue r_i for i = 1, 2, ... , 10 is given as

- r1 = 1 from solution 1 = 1 (no cuts) ;
- r2 = 5 from solution 2 = 2 (no cuts) ;
- r3 = 8 from solution 3 = 3 (no cuts) ;
- r4 = 10 from solution 4 = 2 + 2 ;
- r5 = 13 from solution 5 = 2 + 3 ;
- r6 = 17 from solution 6 = 6 (no cuts) ;
- r7 = 18 from solution 7 = 1 + 6 or 7 = 2 + 2 + 3 ;
- r8 = 22 from solution 8 = 2 + 6 ;
- r9 = 25 from solution 9 = 3 + 6 ;
- r10 = 30 from solution 10 = 10 (no cuts) :

- We can write the optimal revenue r_n for $n \geq 1$ in terms of optimal revenues from shorter rods:

$$r_n = \max \{p_n , r_1 + r_{n-1}, r_2 + r_{n-2} , ... , r_{n-1} + r_1 \}$$

- The first argument corresponds to making no cuts at all.

- The other n - 1 arguments correspond to the maximum revenue obtained by making an initial cut of the rod into two pieces of size i and n - i , for each i = 1, 2 , 3, ... , n - 1, and then optimally cutting up those pieces further.

- We don’t know ahead of time which value of i optimizes revenue.

- Hence, we have to consider all possible values for i and pick the one that maximizes revenue.

- To solve the original problem of size n, we solve smaller problems of the same type, but of smaller sizes.

- Observation: optimal solution incorporates optimal solutions to the two related subproblems: ***optimal substructure***:

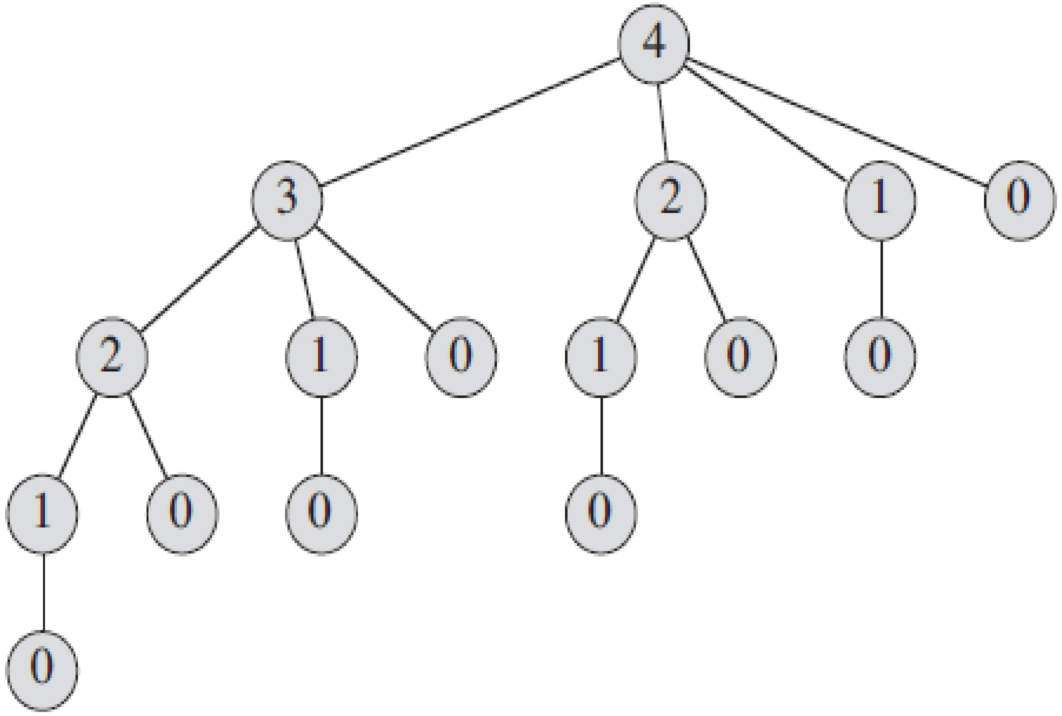
Recursive implementation

- We can rewrite the relation in other equivalent way as

$$r_n = \max_{1 \le i \le n} \{p_i + r_{n-i}\}$$

```
CUT-ROD (p , n)
1      If n==0
2          return 0
3      q=-∞
4      for i=1 to n
5          q=max(q, pi + CUT-ROD(p, n-1))
6     Return q
```

Recursive calls of CUT-ROD (p,4)



Top-Down approach or Memoization

```
MEMOZIED-CUT-ROD (p , n)
        for i=o to n
                r[i]= ∞
        return MEMOZIED-CUT-ROD-AUX(p,n,r)
```

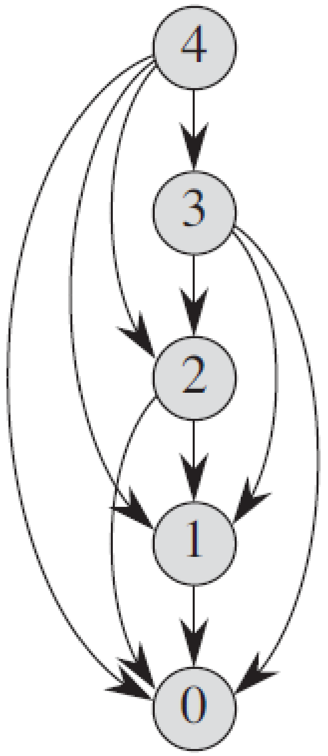
Top-Down DP approach

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
if r[n] < ∞
   return r[n]
If n==0
   q=0
else q=-∞
   for i=1 to n
          q = max (q, pi, MEMOIZED-CUT-ROD-AUX(p, n-i, r)
 r[n] = q
 return q
```

Bottom-Up DP approach

```
BOTTOM-UP-CUT-ROD (p, n)
1 let r[0 . . . n] be a new array
2 r[0] = 0
3 for j=1 to n
4         q=-∞
5         for i=1 to j
6                 q= max(q, pi + r[j-i])
7         r[j] = q
8 return r[n]
```

Sub problem graph of rod cutting with n=4



Constructing solution

```
EXTENDED-BOTTOM-UP-CUT-ROD (p , n)
    Let r[0 . . . n] and s[0 . . . n] be new arrays
    r[0] =0
    for j=1 to n
        q=-∞
        for i = 1 to j
            If q < p[i] + r[j - i]
                q = p[i] + r[j - i]
                s[j] = i
        r[j] = q
    return r and s
```

Printing solution

```
PRINT-CUT-ROD-SOLUTION (p, n)
    (r , s) = EXTENDED-BOTTOM-UP-CUT-ROD (p , n)
    while n > 0
        print s[n]
        n =  n – s[n]
```

i	0	1	2	3	4	5	6	7	8	9	10
r[i]	0	1	5	8	10	13	17	18	22	25	30
s[i]	0	1	2	3	2	2	6	1	2	3	10

Maxrix Chain Multiplication

- Given Chain of n matrices (A_1, A_2, A_3, ... , A_n)

- Compute the product A_1 A_2 A_3 . . . . A_n
- We can use standard algorithm for multiplying pairs of matrices once we have parenthesized it.

- A product of matrices is ***fully parenthesized***
  - ✓if it is either a single matrix or
  - ✓the product of two fully parenthesized matrix products, surrounded by parentheses.

Parenthesization of $A_1A_2A_3A_4$

1. $(A_1(A_2(A_3A_4)))$
2. $(A_1((A_2A_3)A_4))$
3. $((A_1A_2)(A_3A_4))$
4. $((A_1(A_2A_3))A_4)$
5. $(((A_1A_2)A_3)A_4)$

- Matrix multiplication is associative.
- *Hence, all Parenthesizations* yield the same product.

Standard Matrix Multiplication

MATRIX-MULTIPLY (A ,B)

```
if A.columns == B.rows
        error "incompatible dimensions"
else let C be a new A.rows × B.columns matrix
        for i = 1 to A.rows
                for j = 1 to B.columns
                        cij = 0
                        for k = 1 to A.columns
                                cij = cij + a_ik * b_kj
return C
```

Need of *Parenthesization*

- Consider a chain of matrices (A_1 , $A_{2\,,}$ A_3)
- Dimension of A_1 10×100.
- Dimension of A_2 100×5.
- Dimension of A_3 5×50.

- Scalar Multiplications in $((A_1A_2)A_3)$ is 10 *100 *5 + 10 *5 *50 = 7500
- Scalar Multiplications in $(A_1(A_2A_3))$ is 100 * 5 *50 + 10 *100 *50 = 75000

Matrix-chain Multiplication

- Input: Chain of n matrices (A_1, A_2, A_3, ... , A_n)
- Input: A table of dimensions p_i , i = 0, 1, 2, 3, ..., n.
- Dimension of A_i is $p_{i-1} \times p_i$.
- Goal: Fully parenthesize the product $A_1A_2\,A_3 \ldots\ A_n$ in a way that minimizes the number of scalar multiplications.
- **we are not actually multiplying matrices.**
- **Our goal is only to determine an order for multiplying matrices that has the lowest cost.**

Simple Recusion – Exhaustive Search

- Let P(n)= Number of alternative parenthesizations of a chain of n matrices.
- When n = 1, only one way to fully parenthesize the matrix product.
- When n ≥ 2, a fully parenthesized matrix product is the product of two fully parenthesized matrix subproducts.
- The split between the two subproducts may occur between the kth and (k + 1)st matrices.

Simple Recusion – Exhaustive Search

- Possible values of k is 1, 2, 3, ... , n-1.

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

- Solution to the recurrence is $\Omega(2^n)$

Matrix Chain Parenthesization

- Let $A_{i\,..\,j} = A_i\, A_{i+1}\, A_{i+2} \,.\,.\,.\,.\,\, A_j$, where $i \le j$.

- The problem is non trivial if $i < j$.
- Any solution to a nontrivial instance of the matrix-chain multiplication problem requires us to split the product.

- If $i < j$, then to parenthesize the product $A_i\, A_{i+1}\, A_{i+2} \,.\,.\,.\,.\,\, A_j$ we must split the product between A_k and A_{k+1} , where $i \le k < j$.

$$A_i\, A_{i+1}\, A_{i+2} \,.\,.\,.\,.\,\, A_j = A_i\, A_{i+1} \,.\,.\,\, A_k\, A_{k+1}\, A_{k+2} \,.\,.\,\, A_j$$

$$A_{i\,..\,j} = A_{i\,..\,k}\, A_{k+1\,..\,j}$$

Optimal Parenthesization

- Cost of parenthesizing ($A_{i\,..\,j}$) = cost of parenthesizing ($A_{i\,..\,K}$) + cost of parenthesizing ($A_{k+1\,..\,j}$) + Cost of multiplying ($A_{i\,..\,k}\,A_{k+1\,..\,j}$)

- Optimal parenthesization of $A_{i\,..\,j}$ must be an optimal parenthesization of $A_{i\,..\,k}$ and an optimal parenthesization of $A_{k+1\,..\,J.}$
  - ✓Optimal Substrcture.
  - ✓Proof by contradiction

- How to find correct place to split the product?
- We have to consider all possible places.

Optimal Parenthesization

- Let m[i,j] be the minimum number of scalar multiplications needed to compute the matrix $A_{i\,..\,j}$.

- Let S[i, j] be a value of k at which we split the product

  $A_i\ A_{i+1}\ A_{i+2}\ .\ .\ .\ .\ A_j$ in an optimal parenthesization.

- m[1,n] = the lowest cost way to compute $A_{1\,..\,n}$.

Optimal Parenthesization

- When the problem is trivial.
  - ✓ $i=j$.
  - ✓ $A_{i\,..\,i} = A_i$.
  - ✓ No scalar multiplications are necessary.
  - ✓ Thus, $m[i,i] = 0$ for $i = 1, 2, 3, \ldots, n$.

Optimal Parenthesization

- When the problem is non trivial.

  ✓ $i < j$.

  ✓ Split the product between A_k and A_{k+1}.

  ✓ No of scalar multiplications for $A_{i\,..\,K}$

  ✓ No of scalar multiplications for $A_{k+1\,..\,j}$

  ✓ Cost of multiplying $A_{i\,..\,K}\,A_{k+1\,..\,j}$

Optimal Parenthesization

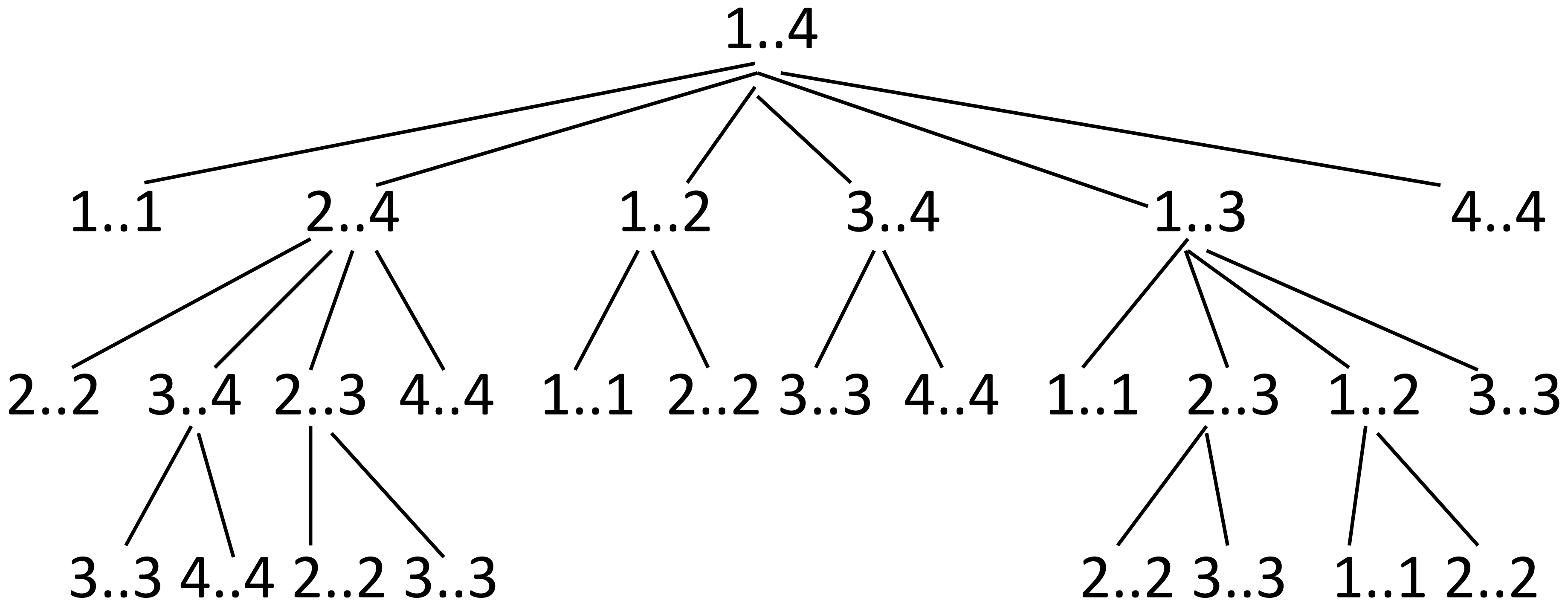
- When the problem is non trivial.
  - ✓ $m[i\,,\,j] = m[i,\,k] + m[k+1,\,j] + p_{i-1}p_kp_j$
  - ✓ possible values of $k = j\text{-}i$.
    - ✓ $k = i,\ i+1,\ i+2,\ .\ .\ .,\ j\ \text{-}1.$
  - ✓ we need to check them all to find the best.

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

Simple Recursive algorithm

```
RECURSIVE-MATRIX-CHAIN(m , p , i , j)
if i == j
        return 0
m[i , j]  = ∞
for k = i to j -1
        q = RECURSIVE-MATRIX-CHAIN(m , p , i , k)
                + RECURSIVE-MATRIX-CHAIN(m, 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 RECURSIVE-MATRIX-CHAIN(p,1,4)



Recursive algorithm Running time

$$T(n) = \begin{cases} 1 & \text{if } \ n = 1 \\ 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) & \text{if } \ n > 1 \end{cases}$$

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

$$T(n) = O(2^n)$$

Time complexity of simple recursion

- Recursive algorithm computes the minimum cost m[1, n].
- However, it takes exponential time.
- The number of distinct subproblems are relatively few.
  - ✓one subproblem for each choice of i and j satisfying $1 \le i \le j \le n$.
  - ✓$\Theta(n^2)$ subproblems.
- Overlapping subproblems: A recursive algorithm encounters each subproblem many times in different branches of its recursion tree.

Matrix chain Multiplication by Memoization

MEMOIZED-MATRIX-CHAIN(p)

n = p:*length*-1

let m[1 . . n, 1 . . n] be a new table

for i = 1 **to** n

    for j = i **to** n

        m[i , j] = ∞

return LOOKUP-CHAIN(m,p,1,n)

Matrix chain Multiplication by Memoization

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

Matrix chain Multiplication Bottom up approach

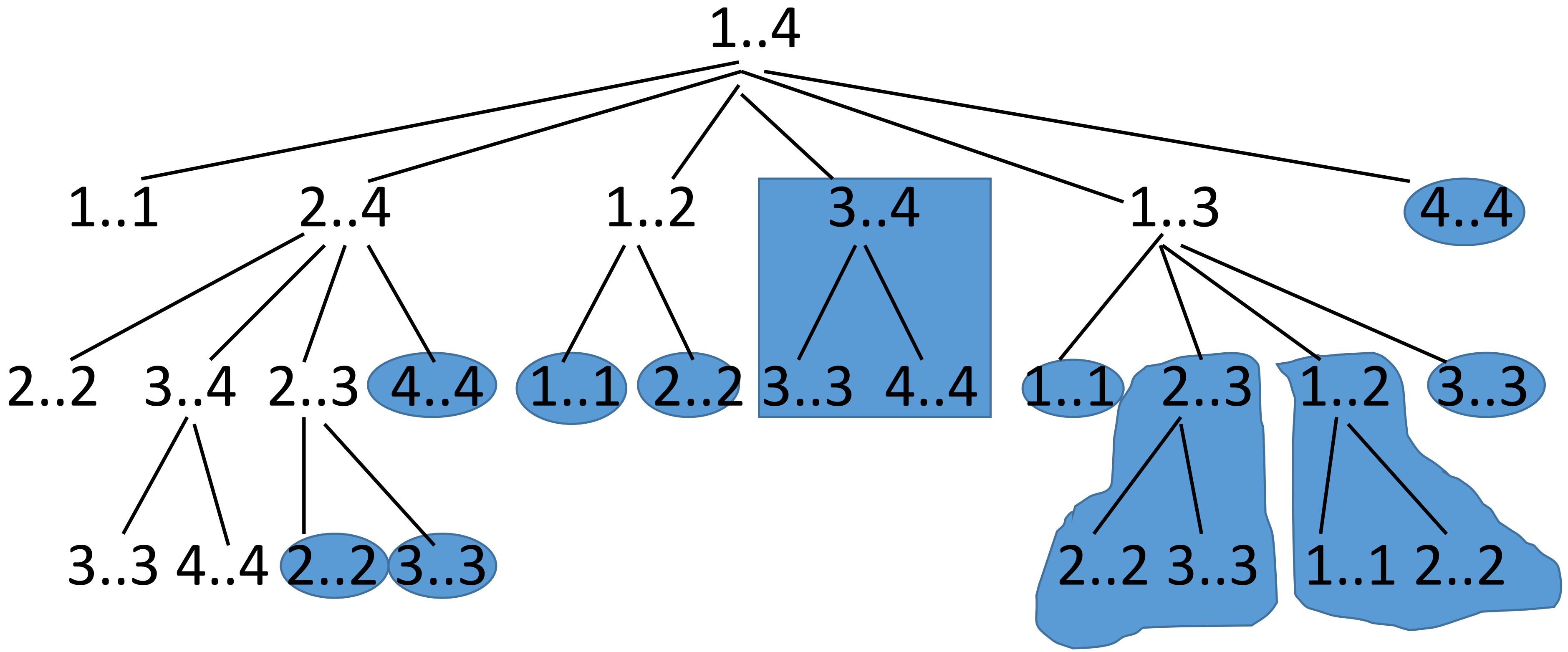
MATRIX-CHAIN-ORDER-BOTTOM-UP-DP(p)

```
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
```

Matrix chain Multiplication Bottom up approach

```
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
```

Subproblems in DP-MATRIX-CHAIN(p,1,4)



Computing the optimal costs

Matrix	A1	A2	A3	A4	A5	A6
Dimension ($p_{i-1} \times p_i$)	30 35	35 15	15 5	5 10	10 20	20 25

$$m[i, i] = 0, \quad \text{for } i = 1, 2, 3, 4, 5, 6.$$

$$m[i, i+1] = p_{i-1}\, p_i\, p_{i+1} \quad \text{for } i = 1, 2, 3, 4, 5$$

$$m[1,3] = \min\begin{cases} m[1,1] + m[2,3] + p_0 p_1 p_3 \\ m[1,2] + m[3,3] + p_0 p_2 p_3 \end{cases} = 7875$$

$$m[2,4] = \min\begin{cases} m[2,2] + m[3,4] + p_1 p_2 p_4 \\ m[2,3] + m[4,4] + p_1 p_3 p_4 \end{cases} = 4375$$

Computing the optimal costs

$$m[3,5] = \min \begin{cases} m[3,4] + m[5,5] + p_2 p_4 p_5 \\ m[3,3] + m[4,5] + p_2 p_3 p_5 \end{cases} = 2500$$

$$m[4,6] = \min \begin{cases} m[4,4] + m[5,6] + p_3 p_4 p_6 \\ m[4,5] + m[6,6] + p_3 p_5 p_6 \end{cases} = 3500$$

$$m[1,4] = \min \begin{cases} m[1,2] + m[3,4] + p_0 p_2 p_4 \\ m[1,1] + m[2,4] + p_0 p_1 p_4 \\ m[1,3] + m[4,4] + p_0 p_3 p_4 \end{cases} = 9375$$

Computing the optimal costs

$$m[2,5] = \min\begin{cases} m[2,3] + m[4,5] + p_1 p_3 p_5 \\ m[2,2] + m[3,5] + p_1 p_2 p_5 \\ m[2,4] + m[5,5] + p_1 p_4 p_5 \end{cases} = 7125$$

$$m[3,6] = \min\begin{cases} m[3,4] + m[5,6] + p_2 p_4 p_6 \\ m[3,3] + m[4,6] + p_2 p_3 p_6 \\ m[3,5] + m[6,6] + p_2 p_5 p_6 \end{cases} = 5375$$

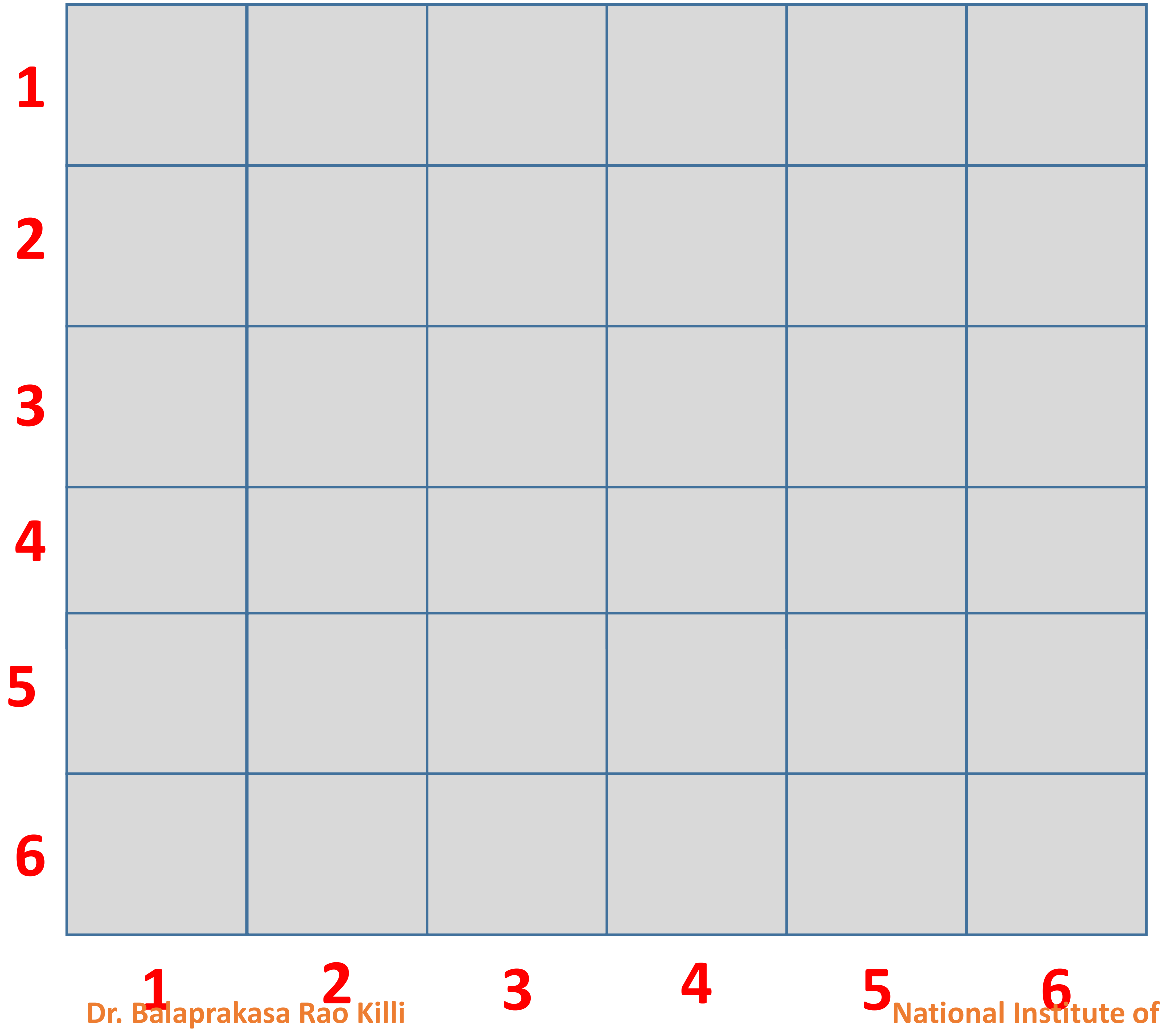
$$m[1,5] = \min\begin{cases} m[1,1] + m[2,5] + p_0 p_1 p_5 \\ m[1,4] + m[5,5] + p_0 p_4 p_5 \\ m[1,2] + m[3,5] + p_0 p_2 p_5 \\ m[1,3] + m[4,5] + p_0 p_3 p_5 \end{cases} = 11875$$

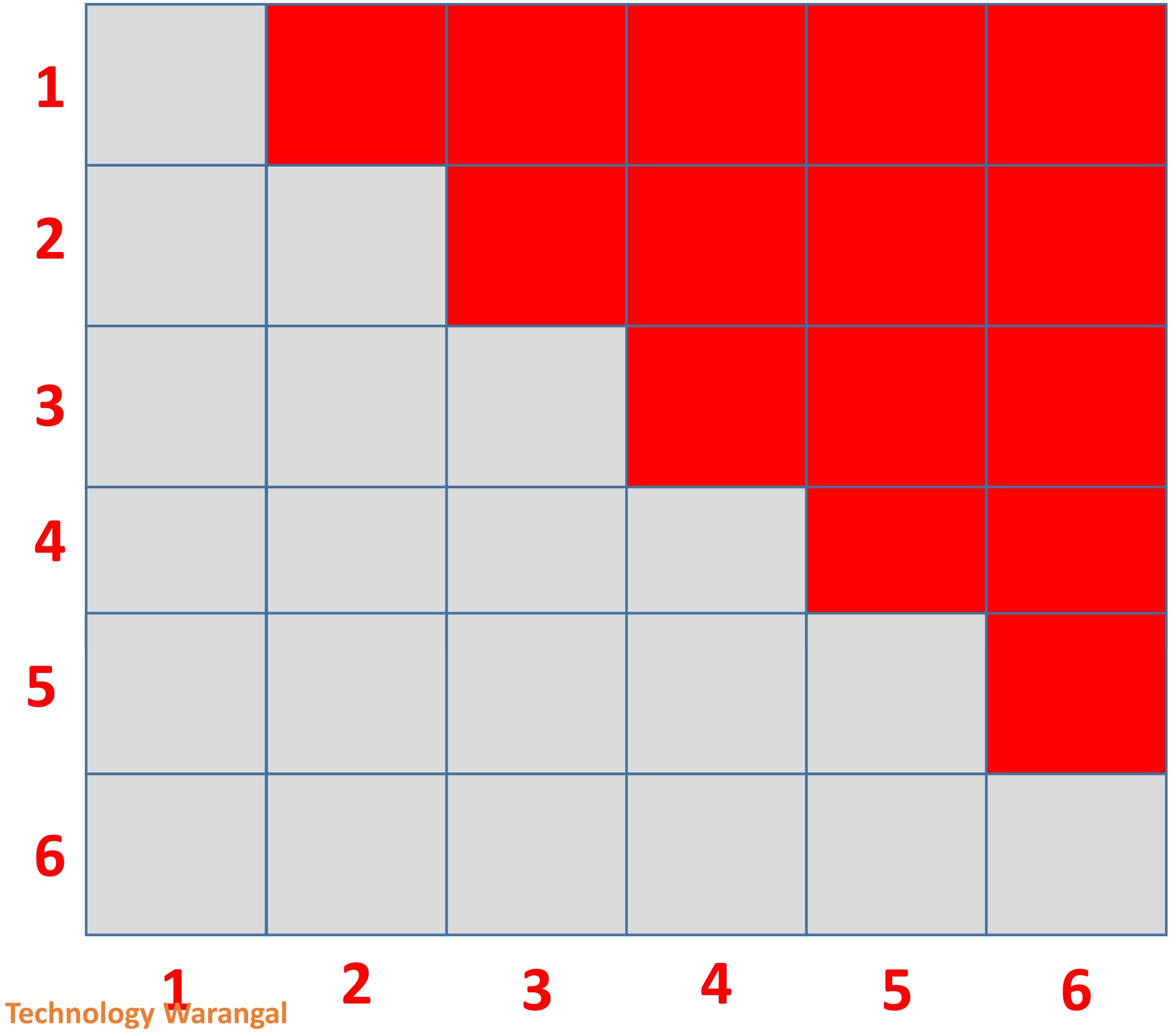
Computing the optimal costs

$$m[2,6]=\min\begin{cases} m[2,2]+m[3,6]+p_1p_2p_6 \\ m[2,5]+m[6,6]+p_1p_5p_6 \\ m[2,3]+m[4,6]+p_2p_3p_6 \\ m[2,4]+m[5,6]+p_1p_4p_6 \end{cases}=10500$$

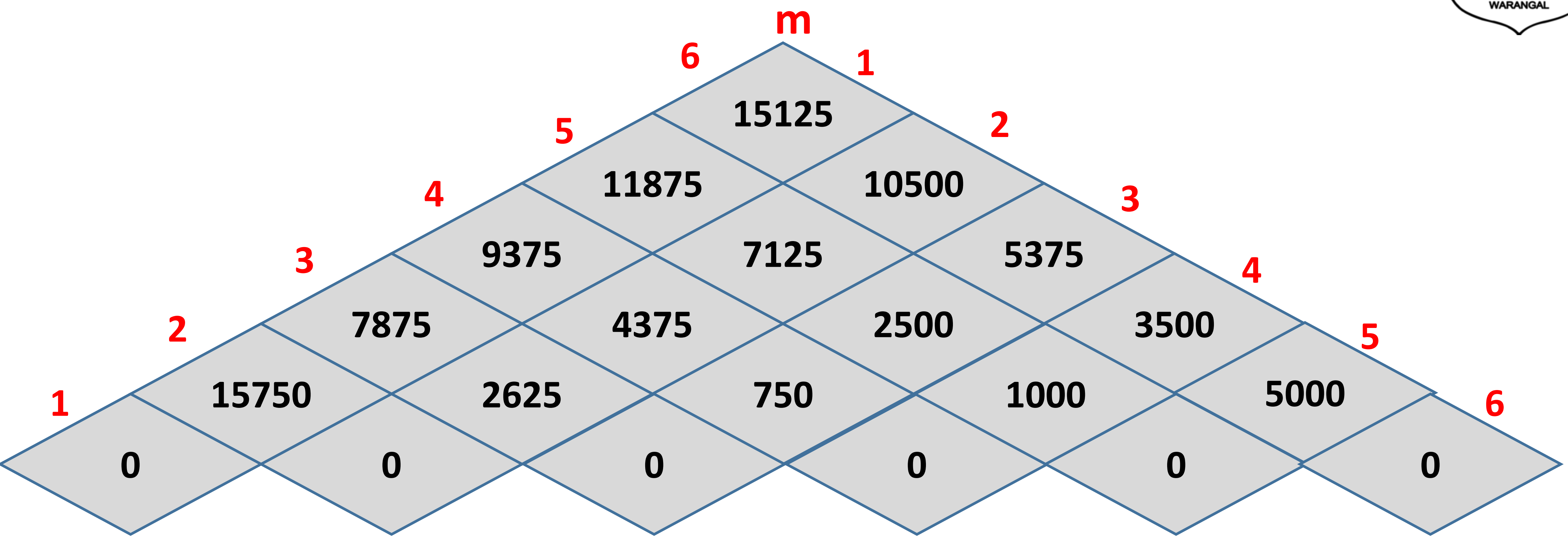
$$m[1,6]=\min\begin{cases} m[1,1]+m[2,6]+p_0p_1p_6 \\ m[1,5]+m[6,6]+p_0p_5p_6 \\ m[1,2]+m[3,6]+p_0p_2p_6 \\ m[1,3]+m[4,6]+p_0p_3p_6 \\ m[1,4]+m[5,6]+p_0p_4p_6 \end{cases}=15125$$

Subproblems computation



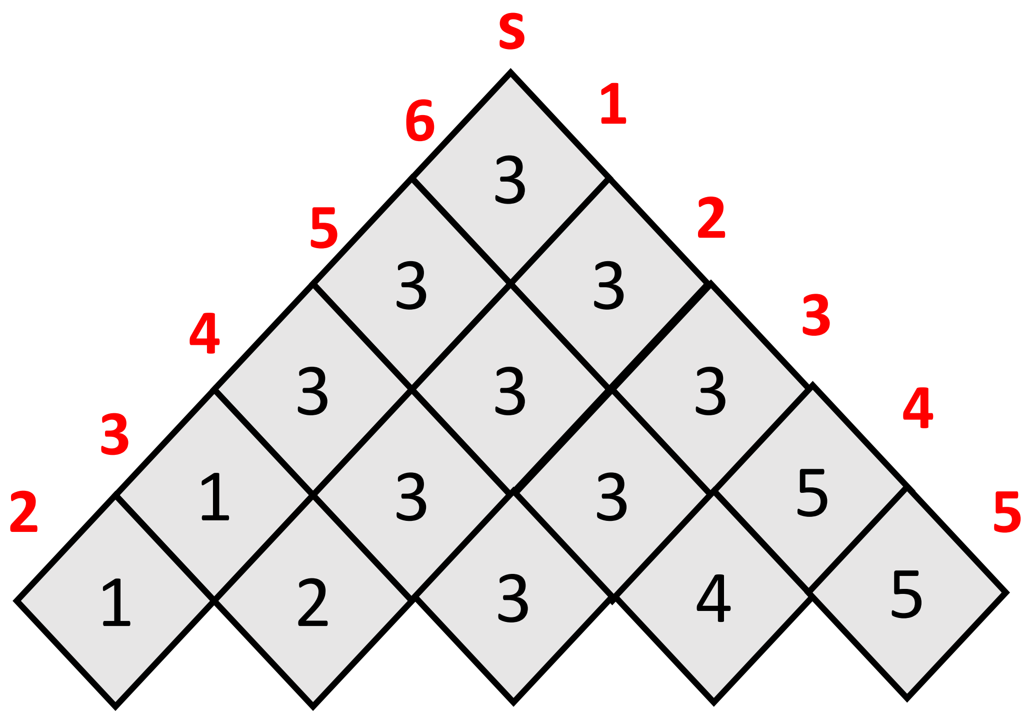


Computing the optimal costs



The tables are rotated so that the diagonals run horizontally

Constructing an optimal solution



Constructing an optimal solution

PRINT-OPTIMAL-PARENS(s, i, j)

1 **if** i == j

2 print “A_i”

3 **else** print “(”

4 PRINT-OPTIMAL-PARENS(s, i, s[i, j])

5 PRINT-OPTIMAL-PARENS(s, s[i, j]+1, j)

6 print “)”

Time Complexity = No of subproblems * Time for each subproblem.

$$= n^2 * n = O(n^3)$$

Ananlysis of Matrix chain multiplication by DP

- non-memoized calls -- Called for the first time-calls in which $m[i,j] = \infty$
- Memoized calls-time-calls in which $m[i,j] > \infty$

- Number of non memoized calls are $\Theta(n^2)$.
- All calls of the second type are made as recursive calls by calls of the first type.

- Each non memoized call makes makes $O(n)$ memoized calls
- Hence, Number of memoized calls are $\Theta(n^3)$.

Ananlysis of Matrix chain multiplication by DP

- work done per each non memoized call is n.
- work done per each memoized call is 1.
- Time = Number of subproblems * Work per each subproblem.

$$= n^2 * n + n^3 * 1 = O(n^3)$$

- **We don’t have to solve recurrences in DP**


Thank you!