# Lecture 18

Dynamic Programming : Rod Cutting to maximize profit, Problem Analysis

# *Dynamic Programming*

*Dynamic Programming is a general algorithm design technique for solving problems defined by or formulated as recurrences with overlapping sub instances*

- *Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems and later assimilated by CS*

- *"Programming" here means "planning"*
- ***Main idea**:*
  - *set up a recurrence relating a solution to a larger instance to solutions of some smaller instances*
  - *solve smaller instances once*
  - *record solutions in a table*
  - *extract solution to the initial instance from that table*

## *Dynamic programming*

- ***Dynamic programming*** *is typically applied to optimization problems. In such problem there can be* ***many solutions****. Each solution has a value, and we wish to find a* ***solution*** *with the optimal value.*

## *The development of a dynamic programming*

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.*

# ***Rod cutting:*** *To maximize the profit*

- ***Input:*** *A length $n$ and table of prices $p_i$, for $i = 1, 2, \ldots, n$.*
- ***Output:*** *The maximum revenue obtainable for rods whose lengths sum to $n$, computed as the sum of the prices for the individual rods.*

| length $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| price $p_i$ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

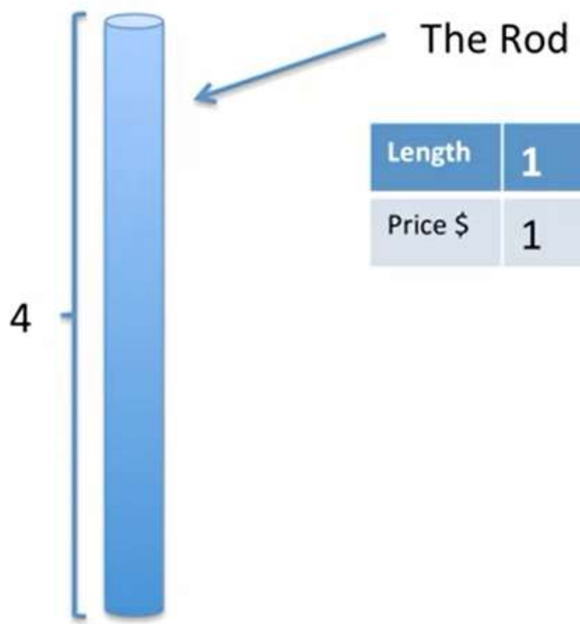
**Recursive Solution**

```
RodCutting(length, Price[ ])
    if(length == 0)
        return 0
    max = −∞
    for(i = 1; i ≤ length; i++)
        tmp = Price[i] + RodCutting(length−i, Price)
        if(tmp > max)
            max = tmp
    return max
```

# Rod Cutting Example

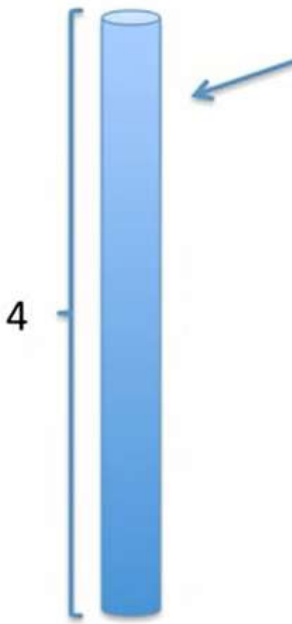

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

# Rod Cutting Example



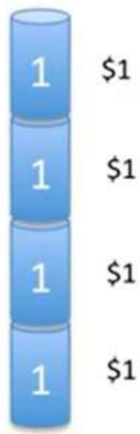

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

# Rod Cutting Example



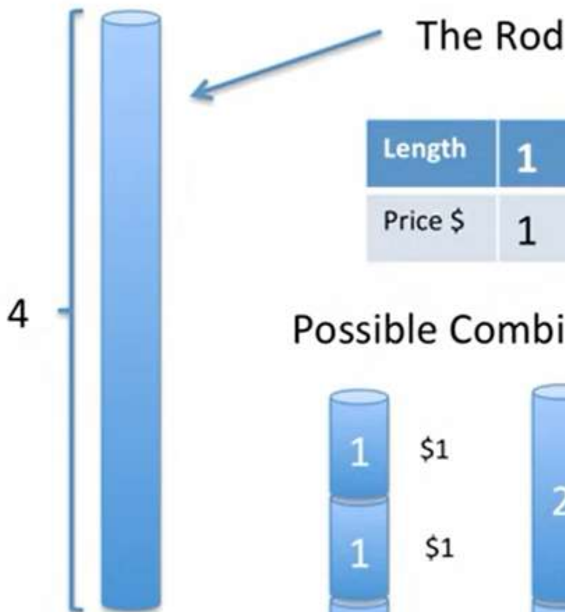

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

Possible Combinations:



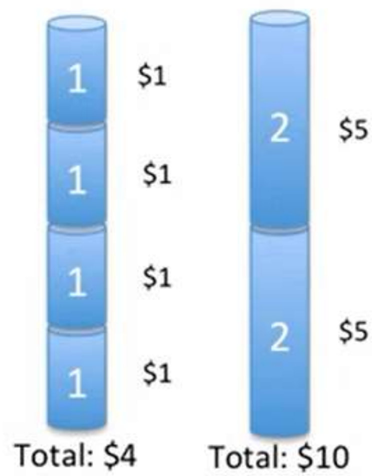

Total: $4

# Rod Cutting Example




| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

Possible Combinations:

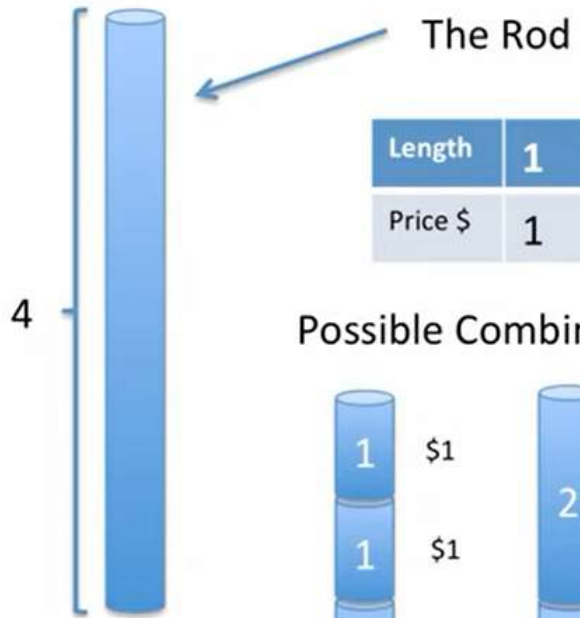


# Rod Cutting Example


| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

Possible Combinations:

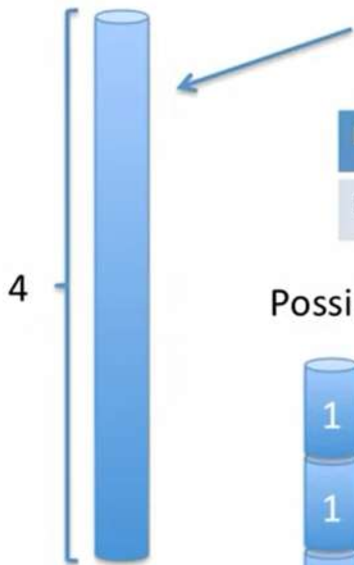
# Rod Cutting Example




| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

Possible Combinations:

# Rod Cutting Example

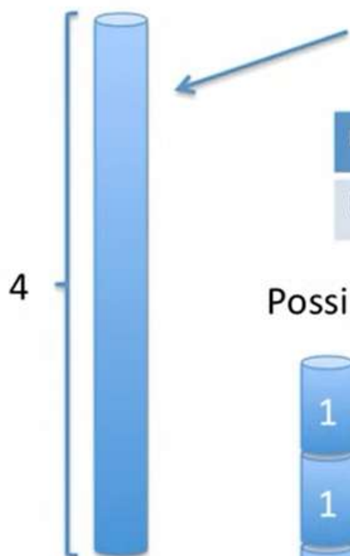



| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

Possible Combinations:


The remaining 3 ways to cut are just permutations of the left arrangements.

# Rod Cutting Example



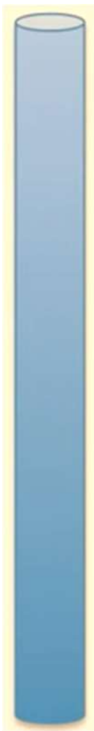

The Rod

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

Possible Combinations:

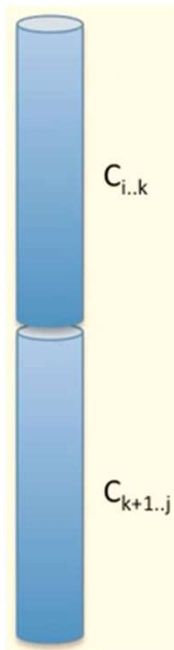

The remaining 3 ways to cut are just permutations of the left arrangements.

# Rod Cutting Dynamic Programming



Let's say we had the optimal solution for cutting the rod $C_{i...j}$ where $C_i$ is the first piece, and $C_j$ is the last piece.

If we take one of the cuts from this solution, somewhere in the middle, say k, and split it so we have two sub problems, $C_{i..k}$, and $C_{k+1..j}$ (Assuming our optimal is not just a single piece)




Let's assume we had a more optimal way of cutting $C_{i..k}$

We would swap the old $C_{i..k}$, and replace it with the more optimal $C_{i..k}$

Overall, the entire problem would now have an even more optimal solution!

But we already had stated that we had the optimal solution! This is a contradiction!

Therefore our original optimal solution is the optimal solution, and this problem exhibits optimal substructure.

# Rod Cutting Solution

Let’s define C(i) as the price of the optimal cut of a rod up until length i

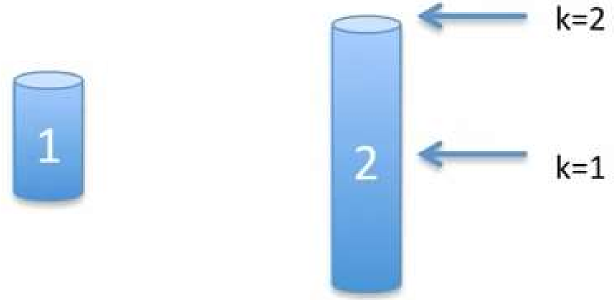
Let $V_k$ be the price of a cut at length k

How to develop a solution:

We define the smallest problems first, and store their solutions.

We increase the rod length, and try all the cuts for that size of rod, taking the most profitable one.

We store the optimal solution for this sized piece, and build solutions to larger pieces from them in some sort of data structure.




$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

Memoization

# Example

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | | | | | | | | |

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

# Example

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | | | | | | | | |

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

$$C(1) = 1$$

# Example

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | 1 | | | | | | | |

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

$$C(2) = \max \begin{cases} V_1 + C(1) = 1 + 1 = 2 \\ V_2 = 5 \end{cases}$$

# Example

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | 1 | 5 | | | | | | |

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

$$C(3) = \max \begin{cases} V_1 + C(2) = 1 + 5 = 6 \\ V_2 + C(1) = 5 + 1 = 6 \\ V_3 = 8 \end{cases}$$

# Example

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | 1 | 5 | 8 | | | | | |

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

$$C(4) = \max \begin{cases} V_1 + C(3) = 1 + 8 = 9 \\ V_2 + C(2) = 5 + 5 = 10 \\ V_3 + C(1) = 8 + 1 = 9 \\ V_4 = 9 \end{cases}$$

# Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | 1 | 5 | 8 | 10 | | | | |

$$C(5) = \max \begin{cases} V_1 + C(4) = 1 + 10 = 11 \\ V_2 + C(3) = 5 + 8 = 13 \\ V_3 + C(2) = 8 + 5 = 13 \\ V_4 + C(1) = 9 + 1 = 10 \\ V_5 = 10 \end{cases}$$

# Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | 1 | 5 | 8 | 10 | 13 | | | |

$$C(6) = \max \begin{cases} V_1 + C(5) = 1 + 13 = 14 \\ V_2 + C(4) = 5 + 10 = 15 \\ V_3 + C(3) = 8 + 8 = 16 \\ V_4 + C(2) = 9 + 5 = 14 \\ V_5 + C(1) = 10 + 1 = 11 \\ V_6 = 17 \end{cases}$$

# Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | 1 | 5 | 8 | 10 | 13 | 17 | | |

# Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | 1 | 5 | 8 | 10 | 13 | 17 | | |

$$C(7) = \max \begin{cases} V_1 + C(6) = 1 + 17 = 18 \\ V_2 + C(5) = 5 + 13 = 18 \\ V_3 + C(4) = 8 + 10 = 18 \\ V_4 + C(3) = 9 + 8 = 17 \\ V_5 + C(2) = 10 + 5 = 15 \\ V_6 + C(1) = 17 + 1 = 18 \\ V_7 = 17 \end{cases}$$

# Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | 1 | 5 | 8 | 10 | 13 | 17 | 18 | |

# Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | 1 | 5 | 8 | 10 | 13 | 17 | 18 | |

$$C(8) = \max \begin{cases} V_1 + C(7) = 1 + 18 = 19 \\ V_2 + C(6) = 5 + 17 = 22 \\ V_3 + C(5) = 8 + 13 = 21 \\ V_4 + C(4) = 9 + 10 = 19 \\ V_5 + C(3) = 10 + 8 = 18 \\ V_6 + C(2) = 17 + 5 = 22 \\ V_7 + C(1)= 17 + 1 = 18 \\ V_8 = 20 \end{cases}$$

# Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

$$C(8) = V_2 + C(6) = 5 + 17 = 22$$

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | 1 | 5 | 8 | 10 | 13 | 17 | 18 | 22 |

# Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

$$C(8) = V_2 + C(6) = 5 + 17 = 22$$


| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

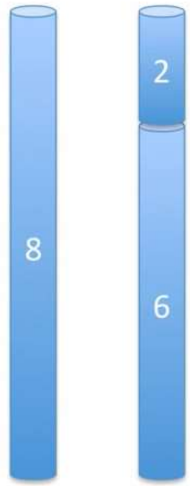
| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | 1 | 5 | 8 | 10 | 13 | 17 | 18 | 22 |

# Example

Rod Cutting DP

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

$$C(8) = V_2 + C(6) = 5 + 17 = 22$$




| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | 1 | 5 | 8 | 10 | 13 | 17 | 18 | 22 |

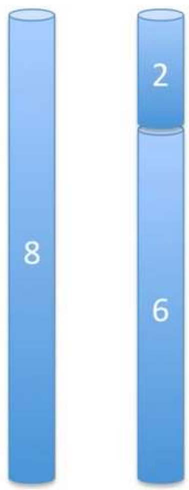
# Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

$$C(8) = V_2 + C(6) = 5 + 17 = 22$$

$$C(6) = V_6 = 17$$




| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | 1 | 5 | 8 | 10 | 13 | 17 | 18 | 22 |

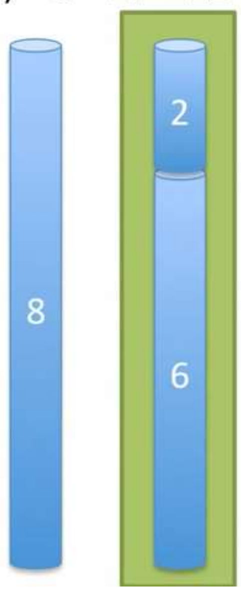
# Example

Rod Cutting DP

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

$$C(8) = V_2 + C(6) = 5 + 17 = 22$$

$$C(6) = V_6 = 17$$




| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price $ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

C(i)

| Len (i) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Opt | 1 | 5 | 8 | 10 | 13 | 17 | 18 | 22 |

The optimal way to cut a rod of length 8!

# RoD Cutting DP Iterative Algorithms

## ITERATIVE SOLUTION 1

```
RodCutting(n, Price[ ])
    allocate Table[0...n]
    Table[0...n] = 0
    for(length = 1; length ≤ n; length++)
        for(i = 1; i ≤ length; i++)
            tmp = Price[i] + Table[length−i]
            if(tmp > Table[length])
                Table[length] = tmp
    return Table[n]
```

Try it by yourself on the below data

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price | 2 | 5 | 9 | 10 | 12 | 13 | 15 | 16 |

## ITERATIVE SOLUTION 2

```
RodCutting(n, Price[ ])
    allocate Table[0...n], Cuts[0...n]
    Table[0...n] = 0
    for(length = 1; length ≤ n; length++)
        for(i = 1; i ≤ length; i++)
            tmp = Price[i] + Table[length−i]
            if(tmp > Table[length])
                Table[length] = tmp
                Cuts[length] = i
    AnswerSet = {}
    while(n > 0)
        AnswerSet.add(Cuts[n])
        n −= Cuts[n]
    return AnswerSet
```

Number of cuts of rod to know with the update of optimal price update

Runtime: $\sum_{i=1}^{n} i = \Theta(n^2)$ (iterative or memoized)

Space: $\Theta(n)$ (iterative or memoized)

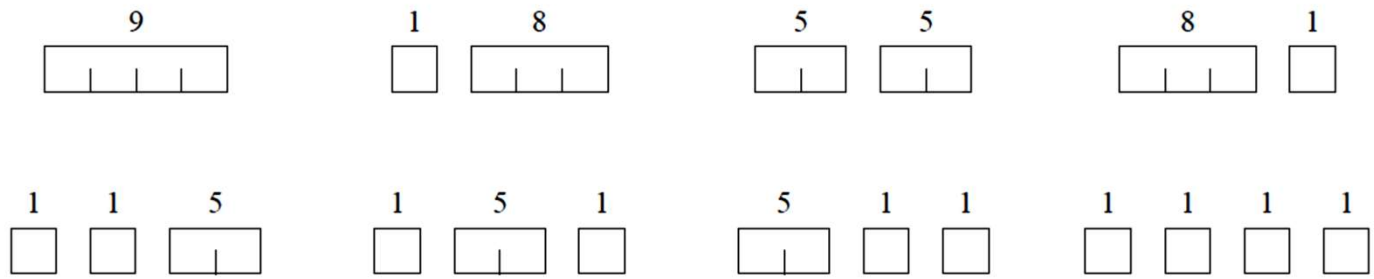
# Time Complexity with/without DP

Without dynamic programming, the problem has a complexity of $O(2^n)$!

For a rod of length 8, there are 128 (or $2^{n-1}$) ways to cut it!

With dynamic programming, and this top down approach, the problem is reduced to $O(n^2)$

# *Ex: a rod of length 4 (Summary)*




| length $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| price $p_i$ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

| $i$ | $r_i$ | optimal solution |
|---|---|---|
| 1 | 1 | 1 (no cuts) |
| 2 | 5 | 2 (no cuts) |
| 3 | 8 | 3 (no cuts) |
| 4 | 10 | 2 + 2 |
| 5 | 13 | 2 + 3 |
| 6 | 17 | 6 (no cuts) |
| 7 | 18 | 1 + 6 or 2 + 2 + 3 |
| 8 | 22 | 2 + 6 |

# *Computing a binomial coefficient by DP*

•Binomial coefficients are coefficients of the binomial formula:

$(a + b)^n = C(n,0)a^n b^0 + \ldots + C(n, k)a^{n-k}b^k + \ldots + C(n, n)a^0 b^n$

•Recurrence: $C(n, k) = C(n-1,k) + C(n-1,k-1)$ for $n > k > 0$

$$C(n,0) = 1, \quad C(n, n) = 1 \quad \text{for } n \geq 0$$

Value of $C(n, k)$ can be computed by filling a table:

| | 0 | 1 | 2 | . . . | $k$-1 | $k$ |
|---|---|---|---|---|---|---|
| 0 | 1 | | | | | |
| 1 | 1 | 1 | | | | |
| . | | | | | | |
| . | | | | | | |
| . | | | | | | |
| $n$-1 | | | | | $C(n-1,k-1)$ | $C(n-1,k)$ |
| $n$ | | | | | | $C(n,k)$ |

# Computing *C*(*n, k*): pseudocode and analysis

**ALGORITHM** *Binomial*$(n, k)$

//Computes $C(n, k)$ by the dynamic programming algorithm
//Input: A pair of nonnegative integers $n \geq k \geq 0$
//Output: The value of $C(n, k)$
**for** $i \leftarrow 0$ **to** $n$ **do**
  **for** $j \leftarrow 0$ **to** $\min(i, k)$ **do**
    **if** $j = 0$ **or** $j = i$
      $C[i, j] \leftarrow 1$
    **else** $C[i, j] \leftarrow C[i-1, j-1] + C[i-1, j]$
**return** $C[n, k]$

Time efficiency: Θ(*nk*)

Space efficiency: Θ(*nk*)

# Lecture 19

Dynamic Programming (Matrix Chain Multiplication): Strassen's Matrix Multiplication, Divide and Conquer Matrix Multiply and its time complexity

# Matrix Operations

$$\begin{bmatrix}3&1&4\\5&3&6\\2&7&5\end{bmatrix}*\begin{bmatrix}4&2&7\\1&5&2\\3&2&6\end{bmatrix}=\begin{bmatrix}12&2&28\\5&15&12\\6&14&30\end{bmatrix}$$ ✗

A bit harder

$$\begin{bmatrix}3&1&4\\5&3&6\\2&7&5\end{bmatrix}+\begin{bmatrix}4&2&7\\1&5&2\\3&2&6\end{bmatrix}=\begin{bmatrix}7&3&11\\6&8&8\\5&9&11\end{bmatrix}$$ ✓

$$\begin{bmatrix}3&1&4\\5&3&6\\2&7&5\end{bmatrix}-\begin{bmatrix}4&2&7\\1&5&2\\3&2&6\end{bmatrix}=\begin{bmatrix}-1&-1&-3\\4&-2&4\\-1&5&-1\end{bmatrix}$$ ✓

$$\underset{n\times m}{A}\begin{bmatrix}a_{11}&a_{12}&\cdots&a_{1m}\\a_{21}&a_{22}&\cdots&a_{2m}\\\vdots&\vdots&\ddots&\vdots\\a_{n1}&a_{n2}&\cdots&a_{nm}\end{bmatrix}*\underset{m\times p}{B}\begin{bmatrix}b_{11}&b_{12}&\cdots&b_{1p}\\b_{21}&b_{22}&\cdots&b_{2p}\\\vdots&\vdots&\ddots&\vdots\\b_{m1}&b_{m2}&\cdots&b_{mp}\end{bmatrix}=\underset{n\times p}{C}\begin{bmatrix}c_{11}&c_{12}&\cdots&c_{1p}\\c_{21}&c_{22}&\cdots&c_{2p}\\\vdots&\vdots&\ddots&\vdots\\c_{n1}&c_{n2}&\cdots&c_{np}\end{bmatrix}$$

$$c_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{im}b_{mj}=\sum_{k=1}^{m}a_{ik}b_{kj}$$

$$\underset{4\times 4}{A}\begin{bmatrix}\boxed{3\;\;5\;\;1\;\;3}\\1\;\;2\;\;3\;\;4\\4\;\;5\;\;6\;\;8\\7\;\;8\;\;9\;\;3\end{bmatrix}*\underset{4\times 4}{B}\begin{bmatrix}\boxed{4}&1&2&3\\\boxed{1}&2&1&6\\\boxed{2}&4&6&2\\\boxed{6}&2&5&4\end{bmatrix}=\underset{4\times 4}{C}\begin{bmatrix}\boxed{37}&&&\\&&&\\&&&\\&&&\end{bmatrix}$$

$$c_{11}=\sum_{k=1}^{4}a_{1k}b_{k1}=3*4+5*1+1*2+3*7=37$$

$$c_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{im}b_{mj}=\sum_{k=1}^{m}a_{ik}b_{kj}$$

# *Strassen's Matrix Multiplication*

*Suppose we want to multiply two matrices of size $N$ x $N$: for example, $A$ X $B = C$.*

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$$\mathbf{C_{11} = a_{11}b_{11} + a_{12}b_{21}}$$
$$\mathbf{C_{12} = a_{11}b_{12} + a_{12}b_{22}}$$
$$\mathbf{C_{21} = a_{21}b_{11} + a_{22}b_{21}}$$
$$\mathbf{C_{22} = a_{21}b_{12} + a_{22}b_{22}}$$

*2x2 matrix multiplication can be accomplished in 8 multiplication.*$\boldsymbol{(2^{\log_2 8} = 2^3)}$

# *Basic Matrix Multiplication*

Algorithm

```
void matrix_mult (){
 for (i = 1; i <= N; i++) {
      for (j = 1; j <= N; j++) {
            compute C_{i,j};            }
 }}
```

Time analysis

$$C_{i,j} = \sum_{k=1}^{N} a_{i,k} b_{k,j}$$

$$\text{Thus } T(N) = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} c = cN^3 = O(N^3)$$

# *Algorithm to Multiply 2 Matrices*

**Input**: Matrices $A_{p\times q}$ and $B_{q\times r}$ (with dimensions $p\times q$ and $q\times r$)

***Result***: *Matrix* $C_{p\times r}$ *resulting from the product* $A\cdot B$

***MATRIX-MULTIPLY***$(A_{p\times q}, B_{q\times r})$

1. ***for*** $i \leftarrow 1$ **to** $p$
2. ***for*** $j \leftarrow 1$ **to** $r$
3. $C[i, j] \leftarrow 0$
4. ***for*** $k \leftarrow 1$ **to** $q$
5. $C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$
6. ***return*** $C$

# *Strassens's Matrix Multiplication*

- *Strassen showed that 2x2 matrix multiplication can be accomplished in 7 multiplication and 18 additions or subtractions.* $(2^{\log_2 7} = 2^{2.807})$

- *This reduce can be done by Divide and Conquer Approach.*

# *Divide-and-Conquer*

- *Divide-and conquer is a general algorithm design paradigm:*
  - *Divide: divide the input data* $S$ *in two or more disjoint subsets* $S_1$, $S_2$, …
  - *Recur: solve the subproblems recursively*
  - *Conquer: combine the solutions for* $S_1$, $S_2$, …, *into a solution for* $S$
- *The base case for the recursion are subproblems of constant size*
- *Analysis can be done using* ***recurrence equations***

# *Divide and Conquer Matrix Multiply*

$$\underset{A}{\begin{bmatrix} A_0 & A_1 \\ A_2 & A_3 \end{bmatrix}} \times \underset{B}{\begin{bmatrix} B_0 & B_1 \\ B_2 & B_3 \end{bmatrix}} = \underset{R}{\begin{bmatrix} A_0 \times B_0 + A_1 \times B_2 & A_0 \times B_1 + A_1 \times B_3 \\ A_2 \times B_0 + A_3 \times B_2 & A_2 \times B_1 + A_3 \times B_3 \end{bmatrix}}$$

- *Divide matrices into sub-matrices: $A_0$, $A_1$, $A_2$ etc*
- *Use blocked matrix multiply equations*
- *Recursively multiply sub-matrices*

# *Divide and Conquer Matrix Multiply*

$$A \times B = R$$

$$\boxed{a_0} \times \boxed{b_0} = \boxed{a_0 \times b_0}$$

- *Terminate recursion with a simple base case*

# *Strassens's Matrix Multiplication*

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$
$P_2 = (A_{21} + A_{22}) * B_{11}$
$P_3 = A_{11} * (B_{12} - B_{22})$
$P_4 = A_{22} * (B_{21} - B_{11})$
$P_5 = (A_{11} + A_{12}) * B_{22}$
$P_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$
$P_7 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$C_{11} = P_1 + P_4 - P_5 + P_7$
$C_{12} = P_3 + P_5$
$C_{21} = P_2 + P_4$
$C_{22} = P_1 + P_3 - P_2 + P_6$

# *Comparison*

$$
\begin{aligned}
C_{11} &= P_1 + P_4 - P_5 + P_7 \\
&= (A_{11}+ A_{22})(B_{11}+B_{22}) + A_{22} * (B_{21} - B_{11}) - (A_{11} + A_{12}) * B_{22}+ \\
&\quad (A_{12} - A_{22}) * (B_{21} + B_{22}) \\
&= A_{11} B_{11} + A_{11} B_{22} + A_{22} B_{11} + A_{22} B_{22} + A_{22} B_{21} – A_{22} B_{11} - \\
&\quad A_{11} B_{22} -A_{12} B_{22} + A_{12} B_{21} + A_{12} B_{22} – A_{22} B_{21} – A_{22} B_{22} \\
&= A_{11} B_{11} + A_{12} B_{21}
\end{aligned}
$$

# *Strassen Algorithm*

```
void matmul(int *A, int *B, int *R, int n) {
 if (n == 1) {
   (*R) += (*A) * (*B);
 } else {
   matmul(A, B, R, n/4);
   matmul(A, B+(n/4), R+(n/4), n/4);
   matmul(A+2*(n/4), B, R+2*(n/4), n/4);
   matmul(A+2*(n/4), B+(n/4), R+3*(n/4), n/4);
   matmul(A+(n/4), B+2*(n/4), R, n/4);
   matmul(A+(n/4), B+3*(n/4), R+(n/4), n/4);
   matmul(A+3*(n/4), B+2*(n/4), R+2*(n/4), n/4);
   matmul(A+3*(n/4), B+3*(n/4), R+3*(n/4), n/4);
 }]
```

Divide matrices in sub-matrices and recursively multiply sub-matrices

## *Time Analysis*

$$T(1) = 1 \qquad (\text{assume } N = 2^k)$$

$$T(N) = 7T(N/2)$$

$$T(N) = 7^k T(N/2^k) = 7^k$$

$$T(N) = 7^{\log N} = N^{\log 7} = N^{2.81}$$

# Strassen Steps and Matrix Multiplication

```python
def strassen2(A, B):
    if len(A) <= 2:
        return brute_force(A, B)
```

# Lecture 20

Dynamic Programming (Matrix Chain Multiplication): Problem Analysis, Notations, Designing DP Algorithm for MCM & its Time Complexity, and Applications of MCM.

# *Matrix-chain Multiplication*

- *Suppose we have a sequence or chain $A_1, A_2, \ldots, A_n$ of $n$ matrices to be multiplied*
  - *That is, we want to compute the product $A_1A_2 \ldots A_n$*

- *There are many possible ways (parenthesizations) to compute the product*

# *Matrix-chain Multiplication*

- *Example: consider the chain $A_1$, $A_2$, $A_3$, $A_4$ of 4 matrices*
  - *Let us compute the product $A_1A_2A_3A_4$*
- *There are 5 possible ways:*

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-chain Multiplication*

- *To compute the number of scalar multiplications necessary, we must know:*
  - *Algorithm to multiply two matrices*
  - *Matrix dimensions*

# Matrix-chain Multiplication

- *Example: Consider three matrices $A_{10\times100}$, $B_{100\times5}$, and $C_{5\times50}$*
- *There are 2 ways to parenthesize*
  - $((AB)C) = D_{10\times5} \cdot C_{5\times50}$
    - $AB \Rightarrow 10\cdot100\cdot5=5{,}000$ *scalar multiplications*
    - $DC \Rightarrow 10\cdot5\cdot50 =2{,}500$ *scalar multiplications*

    Total: 7,500
  - $(A(BC)) = A_{10\times100} \cdot E_{100\times50}$
    - $BC \Rightarrow 100\cdot5\cdot50=25{,}000$ *scalar multiplications*
    - $AE \Rightarrow 10\cdot100\cdot50 =50{,}000$ *scalar multiplications*

Total: 75,000

# *Matrix-chain Multiplication*

- *Matrix-chain multiplication problem*
  - *Given a chain $A_1$, $A_2$, ..., $A_n$ of $n$ matrices, where for $i$=1, 2, ..., $n$, matrix $A_i$ has dimension $p_{i-1} \times p_i$*
  - *Parenthesize the product $A_1A_2...A_n$ such that the total number of scalar multiplications is minimized*
- *Brute force method of exhaustive search takes time exponential in $n$*

# *Dynamic Programming Approach*

- *The structure of an optimal solution*
  - *Let us use the notation, $A_{i..j}$ for the matrix that results from the product $A_i A_{i+1} \ldots A_j$*
  - *An optimal parenthesization of the product $A_1 A_2 \ldots A_n$ splits the product between $A_k$ and $A_{k+1}$ for some integer $k$ where $1 \leq k < n$*
  - *First compute matrices $A_{1..k}$ and $A_{k+1..n}$; then multiply them to get the final matrix $A_{1..n}$*

# *Dynamic Programming Approach*

- *Key observation: parenthesizations of the sub chains $A_1A_2 \ldots A_k$ and $A_{k+1}A_{k+2} \ldots A_n$ must also be optimal if the parenthesizing of the chain $A_1A_2 \ldots A_n$ is optimal (why?)*

- *That is, the optimal solution to the problem contains within it the optimal solution to subproblems*

# Matrix Chain Multiplication

Dot Product

$$\begin{bmatrix} 1 & 5 & 9 & 7 & 3 & 4 \\ 2 & 1 & 9 & 7 & 2 & 6 \\ 9 & 5 & 2 & 2 & 3 & 5 \\ 6 & 6 & 1 & 3 & 1 & 7 \end{bmatrix} \times \begin{bmatrix} 5 & 1 & 3 \\ 9 & 5 & 1 \\ 8 & 7 & 6 \\ 9 & 6 & 8 \\ 8 & 1 & 3 \\ 2 & 2 & 9 \end{bmatrix} = \begin{bmatrix} 217 & & \\ & & \\ & & \\ & & \end{bmatrix}$$

4x6 6x3 4x3

1x5 + 5x9 + 9x8 + 7x9 + 3x8 + 4x2

5 + 45 + 72 + 63 + 24 + 8 = 217

Dot Product

$$\begin{bmatrix} 1 & 5 & 9 & 7 & 3 & 4 \\ 2 & 1 & 9 & 7 & 2 & 6 \\ 9 & 5 & 2 & 2 & 3 & 5 \\ 6 & 6 & 1 & 3 & 1 & 7 \end{bmatrix} \times \begin{bmatrix} 5 & 1 & 3 \\ 9 & 5 & 1 \\ 8 & 7 & 6 \\ 9 & 6 & 8 \\ 8 & 1 & 3 \\ 2 & 2 & 9 \end{bmatrix} = \begin{bmatrix} 217 & 142 & 163 \\ 182 & 126 & 177 \\ 158 & 73 & 114 \\ 141 & 76 & 120 \end{bmatrix}$$

4x6 6x3 4x3

72 Multiplications in Total!

# Matrix Chain Multiplication

| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ |
|---|---|---|---|---|
| 4x10 | 10x3 | 3x12 | 12x20 | 20x7 |

$$\underset{4\times10}{A_1} \times \underset{10\times3}{A_2} \times \underset{3\times12}{A_3} \times \underset{12\times20}{A_4} \times \underset{20\times7}{A_5}$$

4x10x3 + 4x3x12 + 4x12x20 + 4x20x7 = 1784 Multiplication Operations

$$\underset{4\times10}{A_1} \times \underset{10\times3}{A_2} \times \underset{3\times12}{A_3} \times \underset{12\times20}{A_4} \times \underset{20\times7}{A_5}$$

Goal: Find the optimal way to multiply these matrices to perform the fewest multiplications.

Naïve Approach: Try them all, and pick the most optimal one.

Running time: $\Omega(4^n/n^{3/2})$ - $4^n$ dominates! Exponential

# Matrix Chain Multiplication

There is a better way! Dynamic Programming!

Step 1: Check if the problem has Optimal Substructure

If we have an optimal solution for $A_{i...j}$

Assume the solution has the following parentheses:

$$\underline{(A_{i...k})}\underline{(A_{k+1...j})}$$

If there is a better way to multiply $(A_{i...k})$, then we would have a more optimal solution.

This would be a contradiction, as we already stated that we have the optimal solution for $A_{i...j}$

Therefore this problem has optimal substructure.

# Matrix Chain Multiplication

A matrix series $A_{i...j}$ can be broken up into a more efficient solution:

$$(A_{i...k})(A_{k+1...j})$$

We want to find out at which 'k' returns the fewest number of multiplications

We need to define our recursive formula:
M[i,j] is the cost of multiplying matrices from $A_i$ to $A_j$

# Matrix Chain Multiplication

Now we want to try out a bunch of values for 'k' in order to see what the best one is:

$$M[i,j] = \underset{100}{M[i,k]} + \underset{200}{M[k+1,j]} + \underset{2x3x4}{p_{i-1}p_kp_j}$$

Since we don't know what k is, we try this range of k:

$$\overset{100}{\underset{2x3}{(A_{i...k})}}\overset{200}{\underset{3x4}{(A_{k+1...j})}}$$

The minimum returned value is our solution!

$i \leq k < j$

Our Final Recursive Formula:

$$M[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min_{i \leq k < j} \{M[i,j] = M[i,k] + M[k+1,j] + p_{i-1}p_kp_j\} \end{cases}$$

# Matrix Chain Multiplication

We want to start with i = j, then i<j starting with a spread of 1, working our way up

$$M[i,j] = \begin{cases} \boxed{0 \quad \text{if} \quad i=j} \\ \min_{i \leq k < j} \{M[i,j] = M[i,k] + M[k+1,j] + p_{i-1}p_kp_j\} \end{cases}$$

| i \ j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | | | | |
| 2 | x | 0 | | | |
| 3 | x | x | 0 | | |
| 4 | x | x | x | 0 | |
| 5 | x | x | x | x | 0 |

Step 1: Fill the table for i = j

# Matrix Chain Multiplication

$$M[i,j] = \begin{cases} \boxed{0 \quad \text{if} \quad i=j} \\ \min_{i \leq k < j} \{M[i,j] = M[i,k] + M[k+1,j] + p_{i-1}p_kp_j\} \end{cases}$$

We want to start with i = j, then i<j starting with a spread of 1, working our way up

| i \ j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | | | | |
| 2 | x | 0 | | | |
| 3 | x | x | 0 | | |
| 4 | x | x | x | 0 | |
| 5 | x | x | x | x | 0 |

Step 2: Fill the table for:

i=1, j=2

i=2, j=3

i=3, j=4

i=4, j=5

# Matrix Chain Multiplication

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

We want to start with i = j, then i<j starting with a spread of 1, working our way up

$A_1 \times A_2 \times A_3 \times A_4 \times A_5$

| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ |
|---|---|---|---|---|
| 4x10 | 10x3 | 3x12 | 12x20 | 20x7 |
| $p_0$ $p_1$ | $p_1$ $p_2$ | $p_2$ $p_3$ | $p_3$ $p_4$ | $p_4$ $p_5$ |

| i \ j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 120 | | | |
| 2 | x | 0 | 360 | | |
| 3 | x | x | 0 | | |
| 4 | x | x | x | 0 | |
| 5 | x | x | x | x | 0 |

$M[1,2] = \min_{1 \le k < 2} \{M[1,1] + M[1+1,2] + p_0 p_1 p_2\}$

$M[1,2] = \min_{1 \le k < 2} \{0 + 0 + 4x10x3\}$

$M[1,2] = 120$

$M[2,3] = \min_{2 \le k < 3} \{M[2,2] + M[2+1,3] + p_1 p_2 p_3\}$

$M[2,3] = \min_{2 \le k < 3} \{0 + 0 + 10x3x12\}$

$M[2,3] = 360$

$M[3,4] = \min_{3 \le k < 4} \{M[3,3] + M[3+1,4] + p_2 p_3 p_4\}$

$M[3,4] = \min_{3 \le k < 4} \{0 + 0 + 3x12x20\}$

$M[2,3] = 720$

# Matrix Chain Multiplication

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

We want to start with i = j, then i<j starting with a spread of 1, working our way up

$$\underset{\substack{4\times10 \\ p_0\ p_1}}{A_1} \times \underset{\substack{10\times3 \\ p_1\ p_2}}{A_2} \times \underset{\substack{3\times12 \\ p_2\ p_3}}{A_3} \times \underset{\substack{12\times20 \\ p_3\ p_4}}{A_4} \times \underset{\substack{20\times7 \\ p_4\ p_5}}{A_5}$$

| i \ j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 120 | | | |
| 2 | x | 0 | 360 | | |
| 3 | x | x | 0 | 720 | |
| 4 | x | x | x | 0 | 1680 |
| 5 | x | x | x | x | 0 |

$M[4,5] = \min_{4 \le k < 5} \{M[4,4] + M[4+1,5] + p_3 p_4 p_5\}$

$M[4,5] = \min_{4 \le k < 5} \{0 + 0 + 12x20x7\}$

$M[1,2] = 1680$

# Matrix Chain Multiplication

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

We want to start with i = j, then i<j starting with a spread of 1, working our way up

$A_1 \times A_2 \times A_3 \times A_4 \times A_5$

4x10 10x3 3x12 12x20 20x7

$p_0\ p_1$ $p_1\ p_2$ $p_2\ p_3$ $p_3\ p_4$ $p_4\ p_5$

| i \ j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 120 | | | |
| 2 | x | 0 | 360 | | |
| 3 | x | x | 0 | 720 | |
| 4 | x | x | x | 0 | 1680 |
| 5 | x | x | x | x | 0 |

$M[1,3] = \min_{1 \le k < 3}$

k=1

$= M[1,1] + M[1+1,3] + p_0p_1p_3$

$= 0 + 360 + 4\text{x}10\text{x}12$

$= 840$

k=2

$= M[1,2] + M[2+1,3] + p_0p_2p_3$

$= 120 + 0 + 4\text{x}3\text{x}12$

$= 264$ Smaller than 840

# Matrix Chain Multiplication

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

We want to start with i = j, then i<j starting with a spread of 1, working our way up

$$A_1 \times A_2 \times A_3 \times A_4 \times A_5$$

| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ |
|---|---|---|---|---|
| 4x10 | 10x3 | 3x12 | 12x20 | 20x7 |
| $p_0$ $p_1$ | $p_1$ $p_2$ | $p_2$ $p_3$ | $p_3$ $p_4$ | $p_4$ $p_5$ |

| i \ j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 120 | 264 | | |
| 2 | x | 0 | 360 | 1320 | |
| 3 | x | x | 0 | 720 | |
| 4 | x | x | x | 0 | 1680 |
| 5 | x | x | x | x | 0 |

$M[2,4] = \min_{2 \le k < 4}$

k=2

$= M[2,2] + M[2+1,4] + p_1p_2p_4$

$= 0 + 720 + 10 \times 3 \times 20$

$= 1320$

k=3

$= M[2,3] + M[3+1,4] + p_1p_3p_4$

$= 360 + 0 + 10 \times 12 \times 20$

$= 2760$

# Matrix Chain Multiplication

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

We want to start with i = j, then i<j starting with a spread of 1, working our way up

$$\underset{\substack{4\text{x}10 \\ p_0\ p_1}}{A_1} \times \underset{\substack{10\text{x}3 \\ p_1\ p_2}}{A_2} \times \underset{\substack{3\text{x}12 \\ p_2\ p_3}}{A_3} \times \underset{\substack{12\text{x}20 \\ p_3\ p_4}}{A_4} \times \underset{\substack{20\text{x}7 \\ p_4\ p_5}}{A_5}$$

| i \ j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 120 | 264 | | |
| 2 | x | 0 | 360 | 1320 | |
| 3 | x | x | 0 | 720 | |
| 4 | x | x | x | 0 | 1680 |
| 5 | x | x | x | x | 0 |

$M[3,5] = \min_{3 \le k < 5}$

k=3

$= M[3,3] + M[3+1,5] + p_2p_3p_5$

$= 0 + 1680 + 3\text{x}12\text{x}7$

$= 1932$

k=4

$= M[3,4] + M[4+1,5] + p_2p_4p_5$

$= 720 + 0 + 3\text{x}20\text{x}7$

$= 1140$

# Matrix Chain Multiplication (Skipped few steps)

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

We want to start with i = j, then i<j starting with a spread of 1, working our way up

$$\underset{\substack{4x10 \\ p_0\ p_1}}{A_1} \times \underset{\substack{10x3 \\ p_1\ p_2}}{A_2} \times \underset{\substack{3x12 \\ p_2\ p_3}}{A_3} \times \underset{\substack{12x20 \\ p_3\ p_4}}{A_4} \times \underset{\substack{20x7 \\ p_4\ p_5}}{A_5}$$

| i \ j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 120 | 264 | 1080 | |
| 2 | x | 0 | 360 | 1320 | |
| 3 | x | x | 0 | 720 | 1140 |
| 4 | x | x | x | 0 | 1680 |
| 5 | x | x | x | x | 0 |

$M[2,5] = \min_{2 \le k < 5}$

k=2

$= M[2,2] + M[2+1,5] + p_1p_2p_5$

= 0 + 1140+ 10x3x7

= 1350 smaller than 2880 and 2720

k=3

$= M[2,3] + M[3+1,5] + p_1p_3p_5$

= 360 + 1680 + 10x12x7

= 2880

k=4

$= M[2,4] + M[4+1,5] + p_1p_4p_5$

= 1320 + 0 + 10x20x7

= 2720

# Matrix Chain Multiplication

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

We want to start with i = j, then i<j starting with a spread of 1, working our way up

$A_1 \times A_2 \times A_3 \times A_4 \times A_5$

| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ |
|---|---|---|---|---|
| 4x10 | 10x3 | 3x12 | 12x20 | 20x7 |
| $p_0$ $p_1$ | $p_1$ $p_2$ | $p_2$ $p_3$ | $p_3$ $p_4$ | $p_4$ $p_5$ |

| i \ j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 120 | 264 | 1080 | 1344 |
| 2 | x | 0 | 360 | 1320 | 1350 |
| 3 | x | x | 0 | 720 | 1140 |
| 4 | x | x | x | 0 | 1680 |
| 5 | x | x | x | x | 0 |

$M[1,5] = \min_{1 \le k < 5}$

k=1

$= M[1,1] + M[1+1,5] + p_0p_1p_5$

$= 0 + 1350 + 4 \times 10 \times 7$

$= 1630$

k=2

$= M[1,2] + M[2+1,5] + p_0p_2p_5$

$= 120 + 1140 + 4 \times 3 \times 7$

$= 1344$ ← smaller than all

k=3

$= M[1,3] + M[3+1,5] + p_0p_3p_5$

$= 264 + 1680 + 4 \times 12 \times 7$

$= 2280$

k=4

$= M[1,4] + M[4+1,5] + p_0p_4p_5$

$= 1080 + 0 + 4 \times 20 \times 7$

$= 1640$

# Matrix Chain Multiplication

$$M[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min_{i \leq k < j} \{M[i,j] = M[i,k] + M[k+1,j] + p_{i-1}p_kp_j\} \end{cases}$$

$$\underset{\substack{4\text{x}10 \\ p_0\ p_1}}{A_1} \times \underset{\substack{10\text{x}3 \\ p_1\ p_2}}{A_2} \times \underset{\substack{3\text{x}12 \\ p_2\ p_3}}{A_3} \times \underset{\substack{12\text{x}20 \\ p_3\ p_4}}{A_4} \times \underset{\substack{20\text{x}7 \\ p_4\ p_5}}{A_5}$$

We now know that we can multiply $A_1$ to $A_5$ in as few as 1344 multiplication operations!

But where do we put our brackets?

We must focus on the selected k values

$$M[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min_{i \leq k < j} \{M[i,j] = M[i,k] + M[k+1,j] + p_{i-1}p_kp_j\} \end{cases}$$

$$\underset{\substack{4\text{x}10 \\ p_0\ p_1}}{A_1} \times \underset{\substack{10\text{x}3 \\ p_1\ p_2}}{A_2} \times \underset{\substack{3\text{x}12 \\ p_2\ p_3}}{A_3} \times \underset{\substack{12\text{x}20 \\ p_3\ p_4}}{A_4} \times \underset{\substack{20\text{x}7 \\ p_4\ p_5}}{A_5}$$

We now know that we can multiply $A_1$ to $A_5$ in as few as 1344 multiplication operations!

But where do we put our brackets?

We must focus on the selected k values

# Matrix Chain Multiplication

$$M[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min_{i \leq k < j} \{M[i,j] = M[i,k] + M[k+1,j] + p_{i-1}p_k p_j\} \end{cases}$$

$(A_1 \times A_2)(A_3 \times A_4 \times A_5)$

| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ |
|---|---|---|---|---|
| 4x10 | 10x3 | 3x12 | 12x20 | 20x7 |
| $p_0\ p_1$ | $p_1\ p_2$ | $p_2\ p_3$ | $p_3\ p_4$ | $p_4\ p_5$ |

Put () brackets

k=2

Optimal Solution at K=2

$M[1,5] = \underline{M[1,2]} + M[3,5] + p_0p_2p_5$

Check the optimal solution?

$(A_1 \times A_2)((A_3 \times A_4)\ A_5)$

| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ |
|---|---|---|---|---|
| 4x10 | 10x3 | 3x12 | 12x20 | 20x7 |
| $p_0\ p_1$ | $p_1\ p_2$ | $p_2\ p_3$ | $p_3\ p_4$ | $p_4\ p_5$ |

RHS and solution for 3 to 5 is M[3, 5]

k=2

$M[1,5] = \underline{M[1,2]} + \boxed{\underline{M[3,5]}} + p_0p_2p_5$

k=4

Put a bracket between A4 and A5

$M[3,5] = M[3,4] + M[5,5] + p_2p_4p_5$

# Matrix Chain Multiplication

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

$$\underset{4\times10}{(A_1} \times \underset{10\times3}{A_2)}\;\underset{3\times12}{((A_3} \times \underset{12\times20}{A_4)}\;\underset{20\times7}{A_5)}$$

$$\left(\begin{matrix}120\\4\times3\end{matrix}\right)\left(\begin{matrix}720\\3\times20\end{matrix}\qquad\qquad\right)$$

# Matrix Chain Multiplication: Check

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

$(A_1 \times A_2)((A_3 \times A_4)\ A_5)$

4x10 10x3 3x12 12x20 20x7

( 120 4x3 )( 720 3x20 20x7 )

420

3x7

# Matrix Chain Multiplication

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

$$(A_1 \times A_2)((A_3 \times A_4)\, A_5)$$

4x10 10x3 3x12 12x20 20x7

( ) parenthesis around the matrices shows the OPTIMAL SOLUTION

CHECK:

( 120 / 4x3 )( 720 / 3x20 20x7 )

420
3x7

4x3

84
4x7

120 + 720 + 420 + 84 = 1344

# Thank You!!!

Have a good day