Lecture 24

Dynamic Programming: Rod Cutting (contd.), LCS

Source: Introduction to Algorithms, CLRS

When to Use Dynamic Programming?

When to Use Dynamic Programming?

DP is typically used in optimization problems with the following two properties:

When to Use Dynamic Programming?

DP is typically used in optimization problems with the following two properties:

Optimal Substructure: Optimal solution to the problem contains optimal solutions to

When to Use Dynamic Programming?

DP is typically used in optimization problems with the following two properties:

Optimal Substructure: Optimal solution to the problem contains optimal solutions to subproblems.

When to Use Dynamic Programming?

DP is typically used in optimization problems with the following two properties:

Optimal Substructure: Optimal solution to the problem contains optimal solutions to subproblems.

If length of the first cut in optimal cutting is i, then $profit_n = p[i] + profit_{n-i}$

When to Use Dynamic Programming?

DP is typically used in optimization problems with the following two properties:

Optimal Substructure: Optimal solution to the problem contains optimal solutions to subproblems.

If length of the first cut in optimal cutting is i, then $profit_n = p[i] + profit_{n-i}$

Overlapping Subproblems: Subproblems have common subsubproblems.

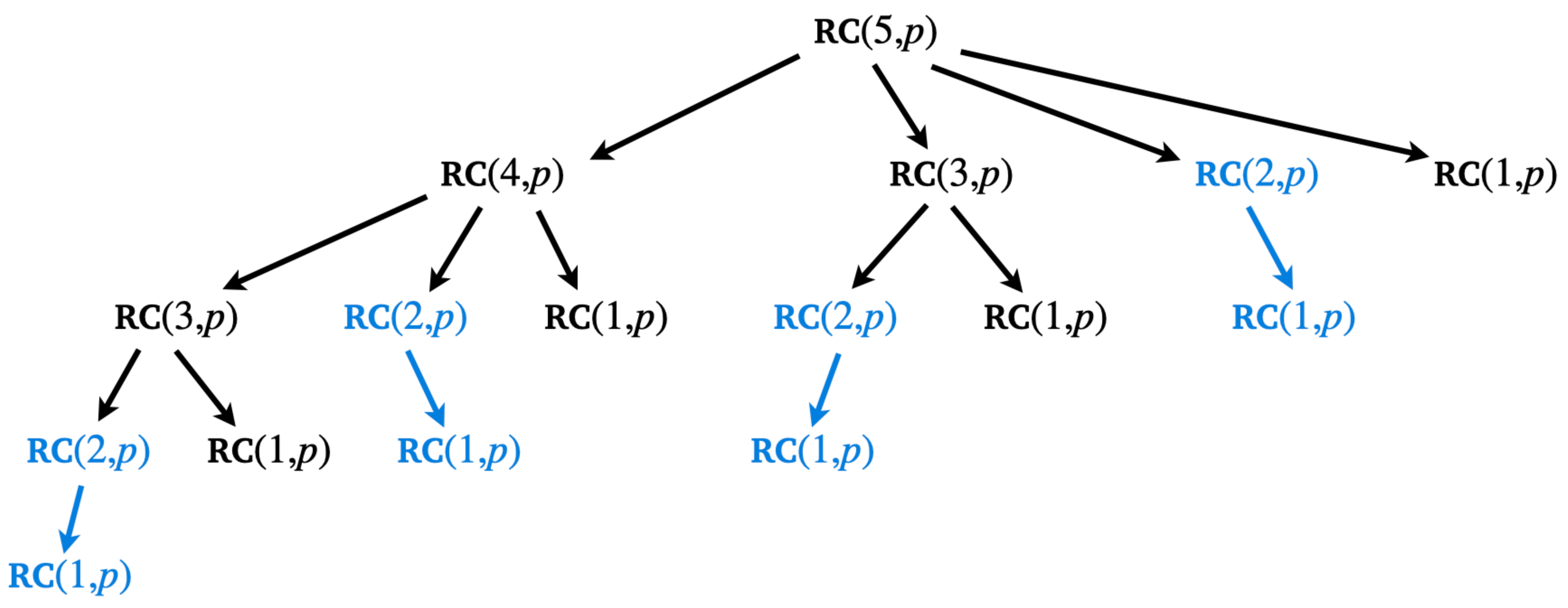
When to Use Dynamic Programming?

DP is typically used in optimization problems with the following two properties:

Optimal Substructure: Optimal solution to the problem contains optimal solutions to subproblems.

If length of the first cut in optimal cutting is i, then $profit_n = p[i] + profit_{n-i}$

Overlapping Subproblems: Subproblems have common subsubproblems.



How to Use Dynamic Programming?

How to Use Dynamic Programming?

Solving a problem using dynamic programming usually takes three steps:

How to Use Dynamic Programming?

Solving a problem using dynamic programming usually takes three steps:

- Find the optimal substructure.

How to Use Dynamic Programming?

Solving a problem using dynamic programming usually takes three steps:

- Find the optimal substructure.
- Recursively define the value of optimal solution.

How to Use Dynamic Programming?

Solving a problem using dynamic programming usually takes three steps:

- Find the optimal substructure.
- Recursively define the value of optimal solution.
- Compute the value of the optimal solution using an array.

Bottom-Up Dynamic Programming

Bottom-Up Dynamic Programming

Note: 1) Recursive dynamic programming is called top-down DP.

Bottom-Up Dynamic Programming

Note: 1) Recursive dynamic programming is called top-down DP.

2) Iterative DP where we start by solving smaller subproblems "first" is called bottom-up DP.

Bottom-Up Dynamic Programming

Note: 1) Recursive dynamic programming is called top-down DP.

2) Iterative DP where we start by solving smaller subproblems "first" is called bottom-up DP.

Bottom-Up-RC(n, p)**:**

Bottom-Up Dynamic Programming

Note: 1) Recursive dynamic programming is called top-down DP.

2) Iterative DP where we start by solving smaller subproblems "first" is called bottom-up DP.

Bottom-Up-RC(n, p)**:**

1. $\mathit{profit}[1:n] = \{p[1], 0, \ldots, 0\}$

Bottom-Up Dynamic Programming

Note: 1) Recursive dynamic programming is called top-down DP.

2) Iterative DP where we start by solving smaller subproblems "first" is called bottom-up DP.

Bottom-Up-RC(n, p)**:**

1. $\mathit{profit}[1:n] = \{p[1], 0, \ldots, 0\}$
2. **for** $j = 2$ **to** n

Bottom-Up Dynamic Programming

Note: 1) Recursive dynamic programming is called top-down DP.

2) Iterative DP where we start by solving smaller subproblems "first" is called bottom-up DP.

Bottom-Up-RC(n, p)**:**

1. $\mathit{profit}[1:n] = \{p[1], 0, \ldots, 0\}$
2. **for** $j = 2$ **to** n

Computing the maximum profit obtainable from j length rod

Bottom-Up Dynamic Programming

Note: 1) Recursive dynamic programming is called top-down DP.

2) Iterative DP where we start by solving smaller subproblems “first” is called bottom-up DP.

Bottom-Up-RC(n, p)**:**

1. $\mathit{profit}[1:n] = \{p[1], 0, \ldots, 0\}$
2. **for** $j = 2$ **to** n
3. $\quad \mathit{profit}[j] = p[j]$

Computing the maximum profit obtainable from j length rod

Bottom-Up Dynamic Programming

Note: 1) Recursive dynamic programming is called top-down DP.

2) Iterative DP where we start by solving smaller subproblems "first" is called bottom-up DP.

Bottom-Up-RC(n, p)**:**

1. $\mathit{profit}[1:n] = \{p[1], 0, \ldots, 0\}$
2. **for** $j = 2$ **to** n
3. $\quad \mathit{profit}[j] = p[j]$
4. $\quad$ **for** $i = 1$ **to** $j - 1$

Computing the maximum profit obtainable from j length rod

Bottom-Up Dynamic Programming

Note: 1) Recursive dynamic programming is called top-down DP.

2) Iterative DP where we start by solving smaller subproblems "first" is called bottom-up DP.

Bottom-Up-RC(n, p)**:**

1. $profit[1:n] = \{p[1], 0, \ldots, 0\}$
2. **for** $j = 2$ **to** n
3. $\quad profit[j] = p[j]$
4. $\quad$ **for** $i = 1$ **to** $j - 1$

Computing the maximum profit obtainable from j length rod

i is the length of the first cut of j length rod

Bottom-Up Dynamic Programming

Note: 1) Recursive dynamic programming is called top-down DP.

2) Iterative DP where we start by solving smaller subproblems "first" is called bottom-up DP.

Bottom-Up-RC(n, p):

1. $profit[1:n] = \{p[1], 0, \ldots, 0\}$
2. **for** $j = 2$ **to** n
3. $\quad profit[j] = p[j]$
4. $\quad$ **for** $i = 1$ **to** $j - 1$
5. $\quad\quad profit[j] = \textbf{Max}(profit[j], p[i] + profit[j - i])$

Computing the maximum profit obtainable from j length rod

i is the length of the first cut of j length rod

Bottom-Up Dynamic Programming

Note: 1) Recursive dynamic programming is called top-down DP.

2) Iterative DP where we start by solving smaller subproblems "first" is called bottom-up DP.

Bottom-Up-RC(n, p)**:**

1. $profit[1:n] = \{p[1], 0, \ldots, 0\}$
2. **for** $j = 2$ **to** n
3. $\quad profit[j] = p[j]$
4. $\quad$ **for** $i = 1$ **to** $j - 1$
5. $\quad\quad profit[j] = \mathbf{Max}(profit[j], p[i] + profit[j-i])$
6. **return** $profit[n]$

Computing the maximum profit obtainable from j length rod

i is the length of the first cut of j length rod

Bottom-Up Dynamic Programming

Note: 1) Recursive dynamic programming is called top-down DP.

2) Iterative DP where we start by solving smaller subproblems "first" is called bottom-up DP.

Bottom-Up-RC(n, p)**:**

1. $\mathit{profit}[1:n] = \{p[1], 0, \ldots, 0\}$
2. **for** $j = 2$ **to** n
3. $\mathit{profit}[j] = p[j]$
4.   **for** $i = 1$ **to** $j - 1$
5. $\mathit{profit}[j] = \mathbf{Max}(\mathit{profit}[j], p[i] + \mathit{profit}[j - i])$
6. **return** $\mathit{profit}[n]$

Computing the maximum profit obtainable from j length rod

i is the length of the first cut of j length rod

Time-Complexity: $O(n^2)$ due to loops of line 2 and 4.

Constructing the Optimal Solution

Constructing the Optimal Solution

How to get optimal cutting not just maximum profit?

Constructing the Optimal Solution

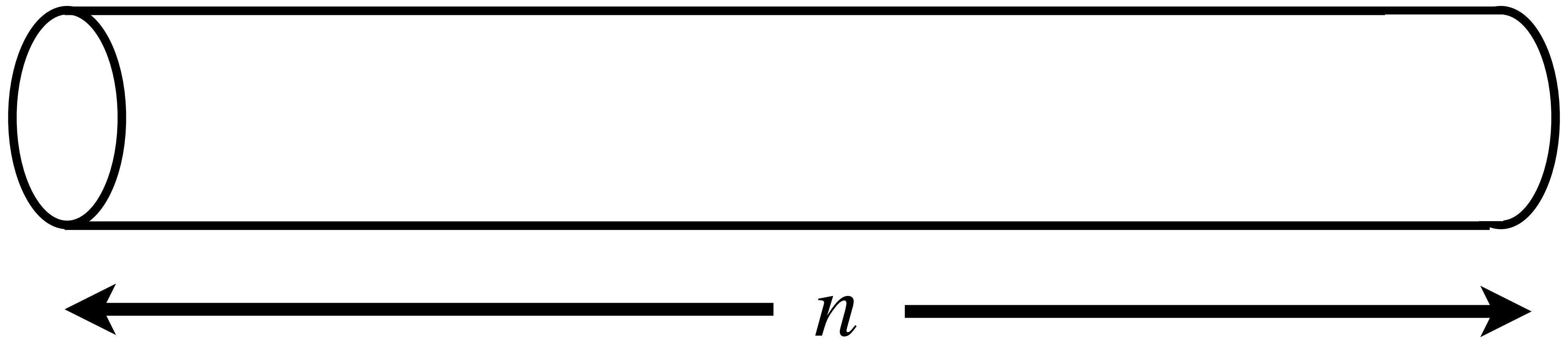
How to get optimal cutting not just maximum profit?

Repeatedly ask what is the length of the first cut in an optimal cutting of the remaining piece.

Constructing the Optimal Solution

How to get optimal cutting not just maximum profit?

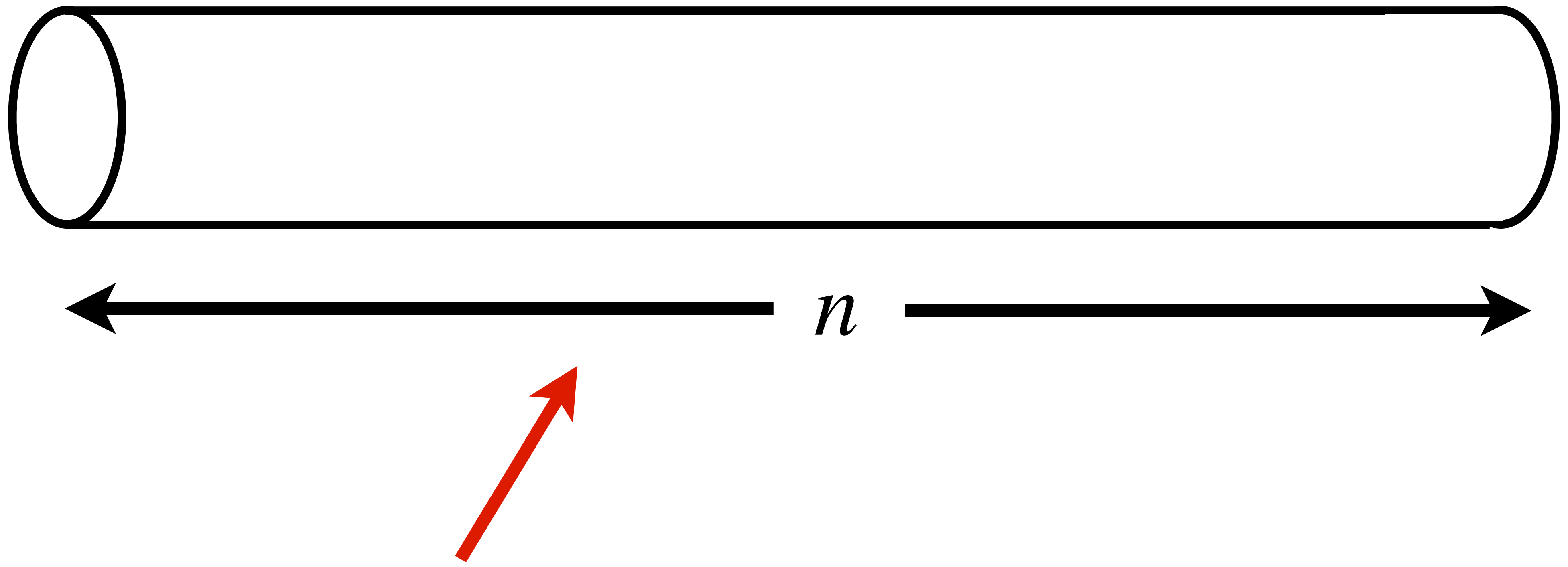
Repeatedly ask what is the length of the first cut in an optimal cutting of the remaining piece.



Constructing the Optimal Solution

How to get optimal cutting not just maximum profit?

Repeatedly ask what is the length of the first cut in an optimal cutting of the remaining piece.

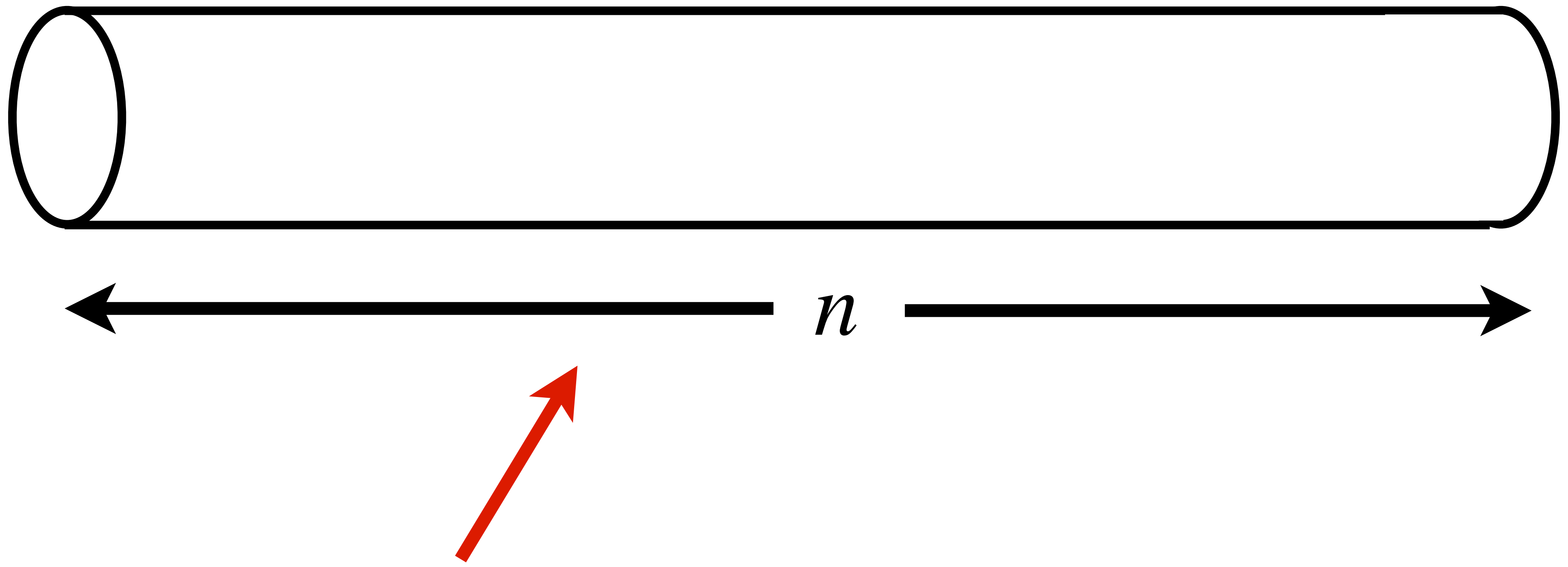


What is the length of the first cut in an n length rod?

Constructing the Optimal Solution

How to get optimal cutting not just maximum profit?

Repeatedly ask what is the length of the first cut in an optimal cutting of the remaining piece.

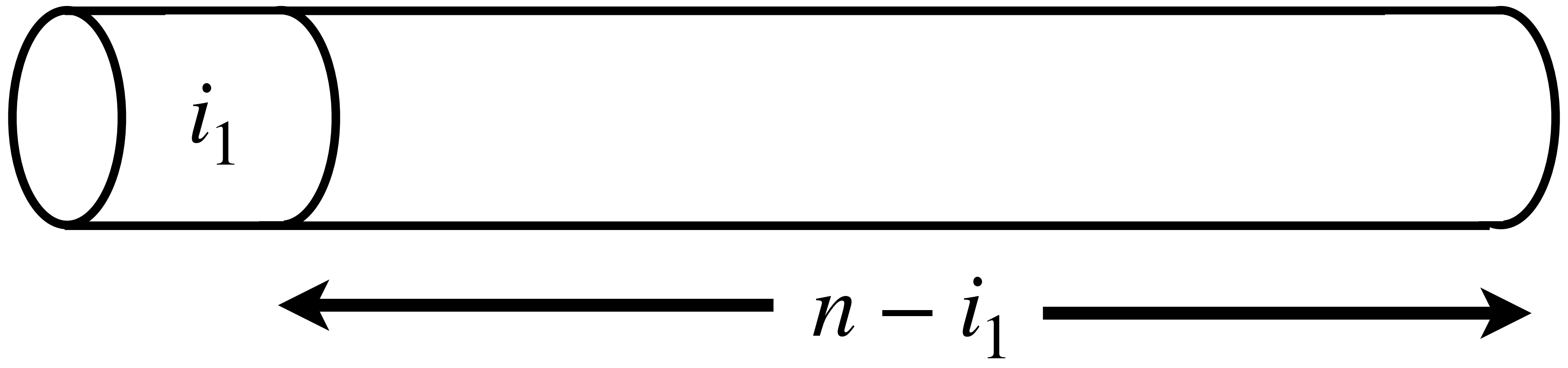


What is the length of the first cut in an n length rod? i_1.

Constructing the Optimal Solution

How to get optimal cutting not just maximum profit?

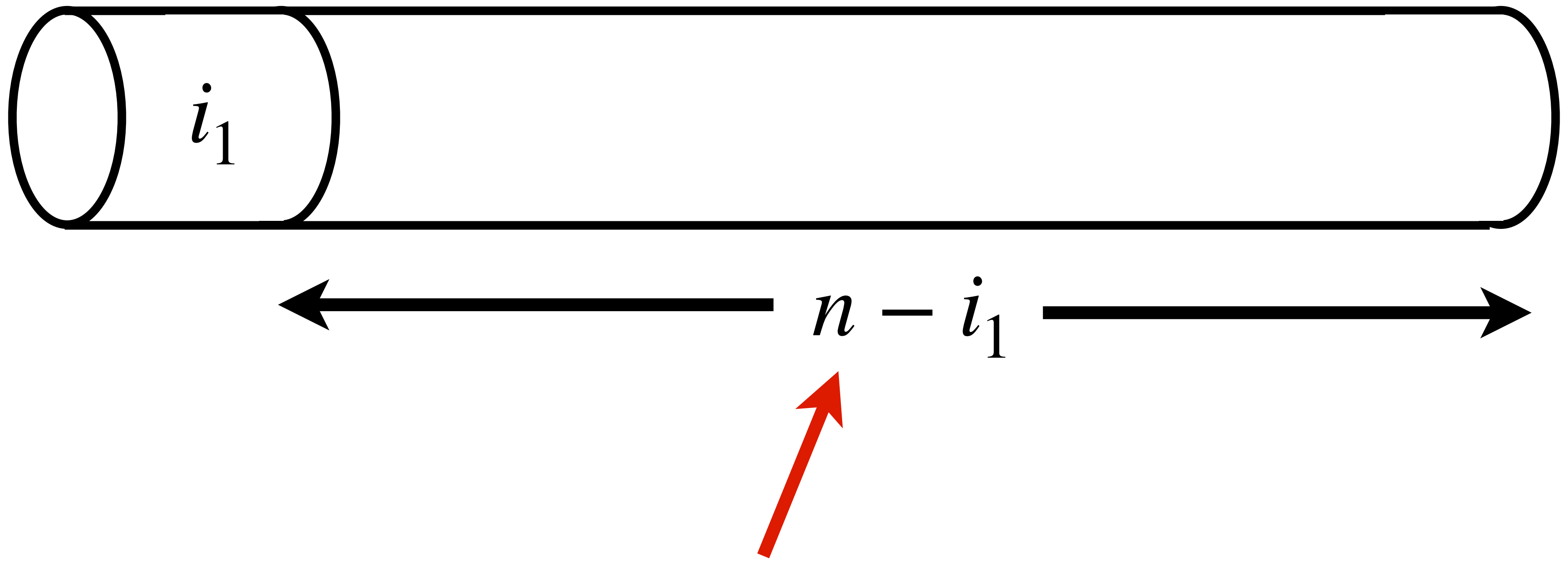
Repeatedly ask what is the length of the first cut in an optimal cutting of the remaining piece.



Constructing the Optimal Solution

How to get optimal cutting not just maximum profit?

Repeatedly ask what is the length of the first cut in an optimal cutting of the remaining piece.

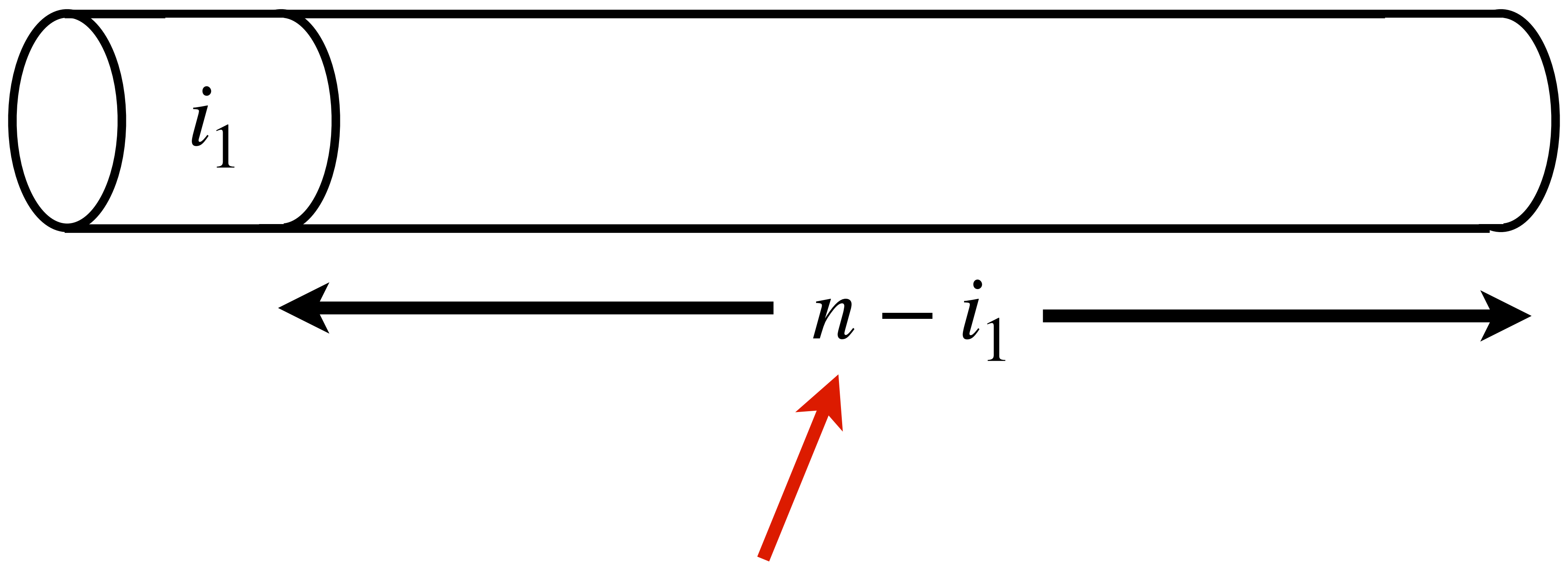


What is the length of the first cut in an $n - i_1$ length rod?

Constructing the Optimal Solution

How to get optimal cutting not just maximum profit?

Repeatedly ask what is the length of the first cut in an optimal cutting of the remaining piece.

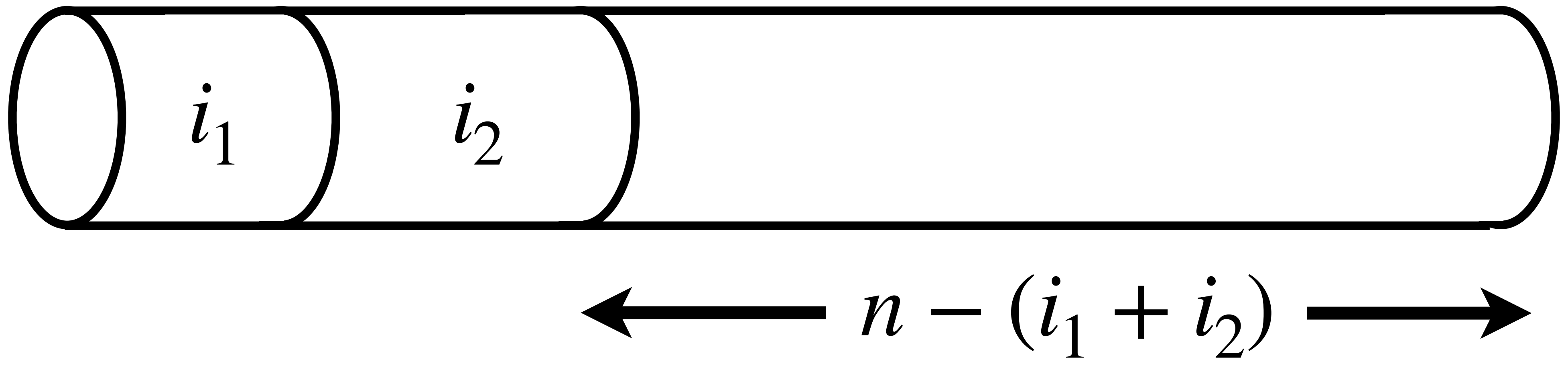


What is the length of the first cut in an $n - i_1$ length rod? i_2.

Constructing the Optimal Solution

How to get optimal cutting not just maximum profit?

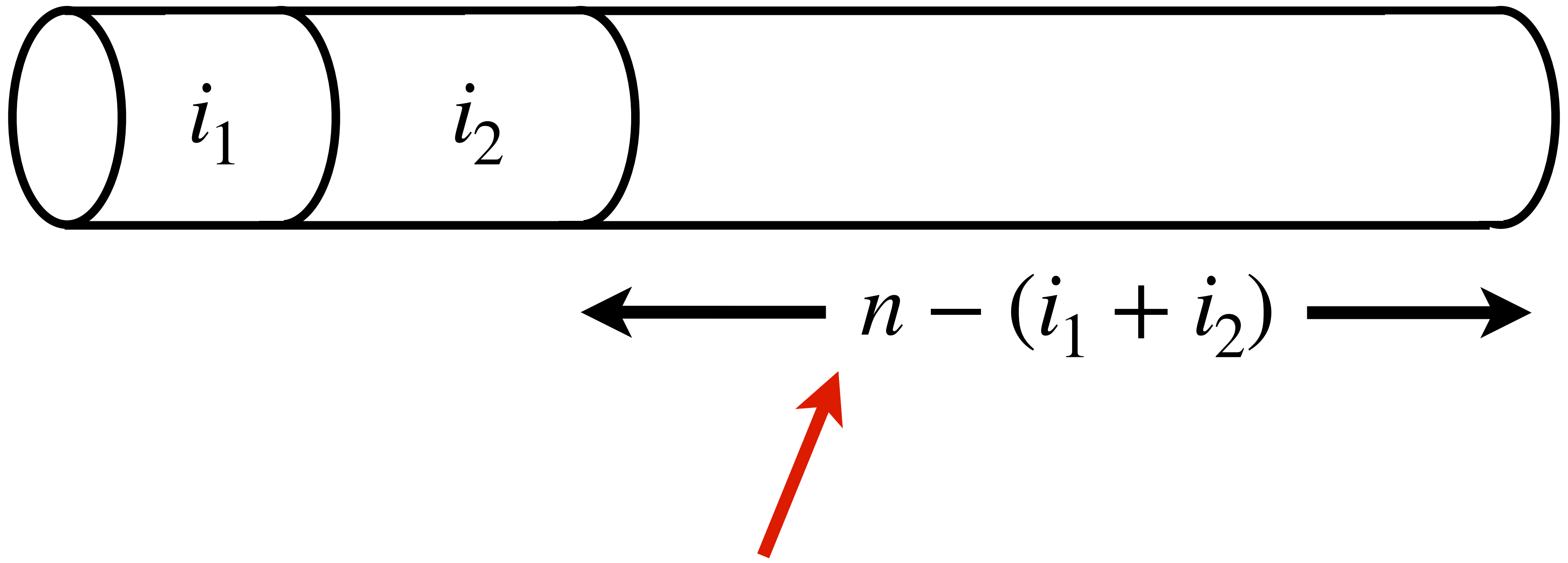
Repeatedly ask what is the length of the first cut in an optimal cutting of the remaining piece.



Constructing the Optimal Solution

How to get optimal cutting not just maximum profit?

Repeatedly ask what is the length of the first cut in an optimal cutting of the remaining piece.

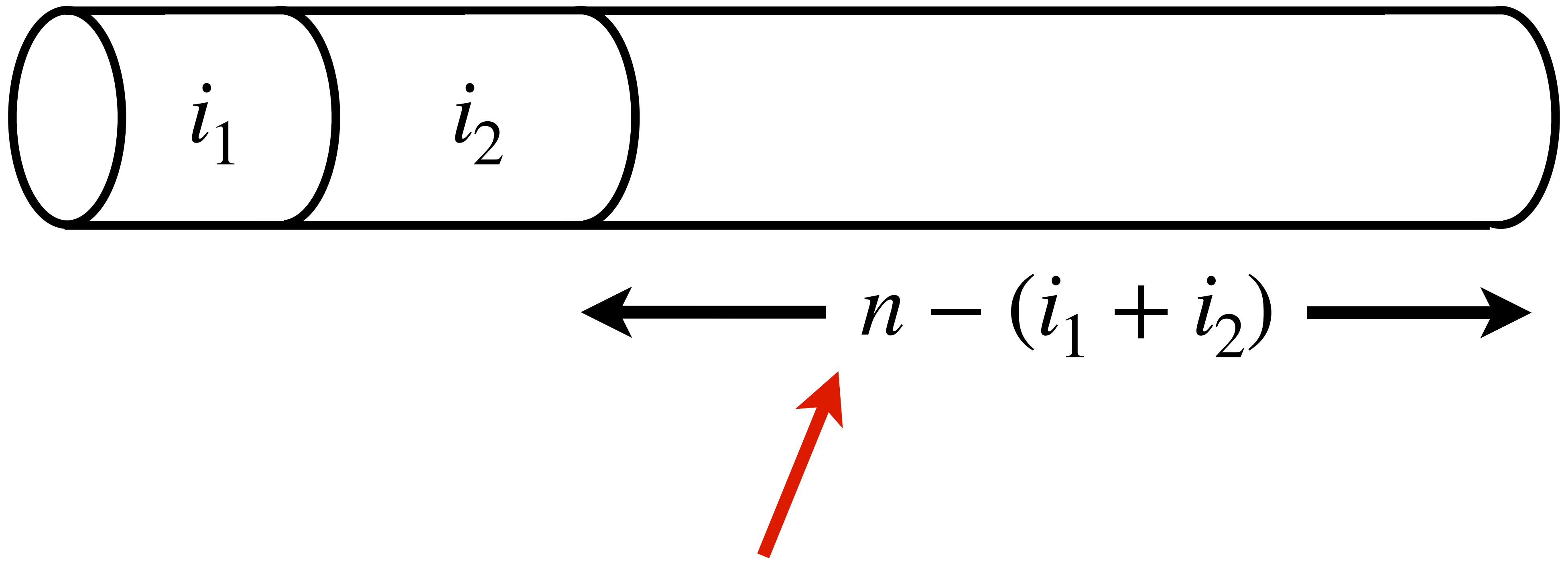


What is the length of the first cut in an $n - (i_1 + i_2)$ length rod?

Constructing the Optimal Solution

How to get optimal cutting not just maximum profit?

Repeatedly ask what is the length of the first cut in an optimal cutting of the remaining piece.

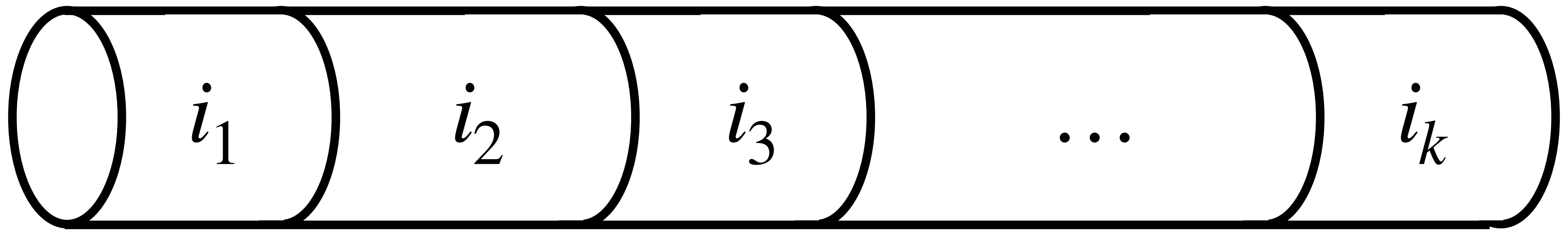


What is the length of the first cut in an $n - (i_1 + i_2)$ length rod? i_3.

Constructing the Optimal Solution

How to get optimal cutting not just maximum profit?

Repeatedly ask what is the length of the first cut in an optimal cutting of the remaining piece.



Optimal cutting of an n length rod.

Constructing the Optimal Solution

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

1. $profit[1:n] = \{p[1], 0, \ldots, 0\}$

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

1. $\mathit{profit}[1:n] \ = \{p[1], 0, \ldots, 0\}$
2. $\mathit{sol}[1:n] = \{1, 0, \ldots, 0\}$

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

1. $profit[1:n] = \{p[1], 0, \ldots, 0\}$
2. $sol[1:n] = \{1, 0, \ldots, 0\}$

$sol[j]$ is the length of the first cut in an optimal cutting of j length rod.

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

1. $profit[1:n] = \{p[1], 0, \ldots, 0\}$
2. $sol[1:n] = \{1, 0, \ldots, 0\}$
3. **for** $j = 2$ **to** n

$sol[j]$ is the length of the first cut in an optimal cutting of j length rod.

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

1. $profit[1:n] = \{p[1],0,\ldots,0\}$
2. $sol[1:n] = \{1,0,\ldots,0\}$
3. **for** $j = 2$ **to** n
4. $\quad profit[j] = p[j]$

$sol[j]$ is the length of the first cut in an optimal cutting of j length rod.

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

1. $profit[1:n] = \{p[1], 0, \ldots, 0\}$
2. $sol[1:n] = \{1, 0, \ldots, 0\}$
3. **for** $j = 2$ **to** n
4. $\quad profit[j] = p[j]$
5. $\quad sol[j] =$ ......

$sol[j]$ is the length of the first cut in an optimal cutting of j length rod.

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

1. $profit[1:n] = \{p[1], 0, \ldots, 0\}$
2. $sol[1:n] = \{1, 0, \ldots, 0\}$
3. **for** $j = 2$ **to** n
4. $\quad profit[j] = p[j]$
5. $\quad sol[j] = j$

$sol[j]$ is the length of the first cut in an optimal cutting of j length rod.

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

1. $profit[1:n] = \{p[1], 0, \ldots, 0\}$
2. $sol[1:n] = \{1, 0, \ldots, 0\}$
3. **for** $j = 2$ **to** n
4. $\quad profit[j] = p[j]$
5. $\quad sol[j] = j$
6. $\quad$**for** $i = 1$ **to** $j - 1$

$sol[j]$ is the length of the first cut in an optimal cutting of j length rod.

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

1. $profit[1:n] = \{p[1], 0, \ldots, 0\}$
2. $sol[1:n] = \{1, 0, \ldots, 0\}$
3. **for** $j = 2$ **to** n
4. $\quad profit[j] = p[j]$
5. $\quad sol[j] = j$
6. $\quad$ **for** $i = 1$ **to** $j - 1$
7. $\quad\quad$ **if** $profit[j] < p[i] + profit[j - i]$

$sol[j]$ is the length of the first cut in an optimal cutting of j length rod.

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

$\quad profit[1:n] = \{p[1], 0, \ldots, 0\}$

$\quad sol[1:n] = \{1, 0, \ldots, 0\}$

$\quad$ **for** $j = 2$ **to** n

$\qquad profit[j] = p[j]$

$\qquad sol[j] = j$

$\qquad$ **for** $i = 1$ **to** $j - 1$

$\qquad\quad$ **if** $profit[j] < p[i] + profit[j - i]$

$\qquad\qquad profit[j] = p[i] + profit[j - i]$

$sol[j]$ is the length of the first cut in an optimal cutting of j length rod.

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

1. $profit[1:n] = \{p[1], 0, \ldots, 0\}$
2. $sol[1:n] = \{1, 0, \ldots, 0\}$
3. **for** $j = 2$ **to** n
4. $\quad profit[j] = p[j]$
5. $\quad sol[j] = j$
6. $\quad$ **for** $i = 1$ **to** $j - 1$
7. $\quad\quad$ **if** $profit[j] < p[i] + profit[j - i]$
8. $\quad\quad\quad profit[j] = p[i] + profit[j - i]$
9. $\quad\quad\quad sol[j] =$ ......

$sol[j]$ is the length of the first cut in an optimal cutting of j length rod.

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

```
SolutionRC(n, p):
    profit[1 : n] = {p[1],0,…,0}
    sol[1 : n] = {1,0,…,0}
    for j = 2 to n
        profit[j] = p[j]
        sol[j] = j
        for i = 1 to j − 1
            if profit[j] < p[i] + profit[j − i]
                profit[j] = p[i] + profit[j − i]
                sol[j] = i
```

$sol[j]$ is the length of the first cut in an optimal cutting of j length rod.

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

$profit[1:n] = \{p[1], 0, \ldots, 0\}$

$sol[1:n] = \{1, 0, \ldots, 0\}$

for $j = 2$ **to** n

    $profit[j] = p[j]$

    $sol[j] = j$

    for $i = 1$ **to** $j - 1$

        if $profit[j] < p[i] + profit[j-i]$

            $profit[j] = p[i] + profit[j-i]$

            $sol[j] = i$

return $profit[n]$

$sol[j]$ is the length of the first cut in an optimal cutting of j length rod.

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

```
SolutionRC(n, p):
1.   profit[1 : n] = {p[1],0,…,0}
2.   sol[1 : n] = {1,0,…,0}
3.   for j = 2 to n
4.       profit[j] = p[j]
5.       sol[j] = j
6.       for i = 1 to j − 1
7.           if profit[j] < p[i] + profit[j − i]
8.               profit[j] = p[i] + profit[j − i]
9.               sol[j] = i
10.  return profit[n]
```

$sol[j]$ is the length of the first cut in an optimal cutting of j length rod.

PrintRC(n, sol):

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

```
profit[1 : n] = {p[1],0,…,0}
sol[1 : n] = {1,0,…,0}
for j = 2 to n
    profit[j] = p[j]
    sol[j] = j
    for i = 1 to j − 1
        if profit[j] < p[i] + profit[j − i]
            profit[j] = p[i] + profit[j − i]
            sol[j] = i
return profit[n]
```

$sol[j]$ is the length of the first cut in an optimal cutting of j length rod.

PrintRC(n, sol):

```
while n > 0
```

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

```
SolutionRC(n, p):
    profit[1 : n] = {p[1],0,…,0}
    sol[1 : n] = {1,0,…,0}
    for j = 2 to n
        profit[j] = p[j]
        sol[j] = j
        for i = 1 to j − 1
            if profit[j] < p[i] + profit[j − i]
                profit[j] = p[i] + profit[j − i]
                sol[j] = i
    return profit[n]
```

sol[*j*] is the length of the first cut in an optimal cutting of *j* length rod.

```
PrintRC(n, sol):
    while n > 0
        print sol[n]
```

Constructing the Optimal Solution

Idea: Store the actual choices, i.e., length of the first cut, in a separate array.

SolutionRC(n, p):

- $\mathit{profit}[1:n] = \{p[1], 0, \ldots, 0\}$
- $\mathit{sol}[1:n] = \{1, 0, \ldots, 0\}$
- **for** $j = 2$ **to** n
  - $\mathit{profit}[j] = p[j]$
  - $\mathit{sol}[j] = j$
  - **for** $i = 1$ **to** $j - 1$
    - **if** $\mathit{profit}[j] < p[i] + \mathit{profit}[j-i]$
      - $\mathit{profit}[j] = p[i] + \mathit{profit}[j-i]$
      - $\mathit{sol}[j] = i$
- **return** $\mathit{profit}[n]$

$\mathit{sol}[j]$ is the length of the first cut in an optimal cutting of j length rod.

PrintRC(n, sol):

- **while** $n > 0$
  - **print** $\mathit{sol}[n]$
  - $n = n - \mathit{sol}[n]$

Subsequence

Subsequence

Defn: A subsequence of a given sequence is just the given sequence with 0 or more elements

Subsequence

Defn: A subsequence of a given sequence is just the given sequence with 0 or more elements dropped.

Subsequence

Defn: A subsequence of a given sequence is just the given sequence with 0 or more elements dropped.

Example:

Subsequence

Defn: A subsequence of a given sequence is just the given sequence with 0 or more elements dropped.

Example: Let $S =$ "michaelscott" be a sequence.

Subsequence

Defn: A subsequence of a given sequence is just the given sequence with 0 or more elements dropped.

Example: Let $S =$ "michaelscott" be a sequence.

Some subsequence of S are : "mcheco"

Subsequence

Defn: A subsequence of a given sequence is just the given sequence with 0 or more elements dropped.

Example: Let $S =$ "michaelscott" be a sequence.

Some subsequence of S are : "mcheco", "m"

Subsequence

Defn: A subsequence of a given sequence is just the given sequence with 0 or more elements dropped.

Example: Let $S =$ "michaelscott" be a sequence.

Some subsequence of S are : "mcheco", "m", "iaeo"

Subsequence

Defn: A subsequence of a given sequence is just the given sequence with 0 or more elements dropped.

Example: Let $S =$ "michaelscott" be a sequence.

Some subsequence of S are : "mcheco", "m", "iaeo", "michaelscott"

Subsequence

Defn: A subsequence of a given sequence is just the given sequence with 0 or more elements dropped.

Example: Let $S =$ "michaelscott" be a sequence.

Some subsequences of S are : "mcheco", "m", "iaeo", "michaelscott", "".

Longest Common Subsequence

Longest Common Subsequence

LCS:

Longest Common Subsequence

LCS:

Input: Two sequences X and Y.

Longest Common Subsequence

LCS:

Input: Two sequences X and Y.

Output: Length of the longest common subsequence.

Longest Common Subsequence

LCS:

Input: Two sequences X and Y.

Output: Length of the longest common subsequence.

Example: $X =$ "iitjodhpur", $Y =$ "iitindore"

Longest Common Subsequence

LCS:

Input: Two sequences X and Y.

Output: Length of the longest common subsequence.

Example: $X =$ "iitjodhpur", $Y =$ "iitindore"

Some common subsequences of X and Y are: "iit", "dr", "tor".

Longest Common Subsequence

LCS:

Input: Two sequences X and Y.

Output: Length of the longest common subsequence.

Example: $X =$ "iitjodhpur", $Y =$ "iitindore"

Some common subsequences of X and Y are: "iit", "dr", "tor".

Some longest common subsequences of X and Y are: "iitor", "iitdr".

Longest Common Subsequence

LCS:

Input: Two sequences X and Y.

Output: Length of the longest common subsequence.

Longest Common Subsequence

LCS:

Input: Two sequences X and Y.

Output: Length of the longest common subsequence.

Application of LCS:

Longest Common Subsequence

LCS:

Input: Two sequences X and Y.

Output: Length of the longest common subsequence.

Application of LCS:

- LCS of two sequences or strings is a measure of how similar they are.

Longest Common Subsequence

LCS:

Input: Two sequences X and Y.

Output: Length of the longest common subsequence.

Application of LCS:

- LCS of two sequences or strings is a measure of how similar they are.
- Used to find similarity of DNAs which can be seen as strings of "A", "C", "G", and "T"

Longest Common Subsequence

LCS:

Input: Two sequences X and Y.

Output: Length of the longest common subsequence.

Application of LCS:

- LCS of two sequences or strings is a measure of how similar they are.
- Used to find similarity of DNAs which can be seen as strings of "A", "C", "G", and "T" characters which represent nucleotides.

Brute Force for LCS

Brute Force for LCS

BruteForceLCS(X, Y):

Brute Force for LCS

BruteForceLCS(X, Y):

1. $lcs = 0$

Brute Force for LCS

BruteForceLCS(X, Y):

1. $lcs = 0$
2. **for** every subsequence x of X

Brute Force for LCS

BruteForceLCS(X, Y):

1. $lcs = 0$
2. **for** every subsequence x of X
3.   **if** x is a subsequence of Y

Brute Force for LCS

BruteForceLCS(X, Y):

1. $lcs = 0$
2. **for** every subsequence x of X
3. **if** x is a subsequence of Y
4. $lcs = \textbf{Max}(lcs, |x|)$

Brute Force for LCS

BruteForceLCS(X, Y):

1. $lcs = 0$
2. **for** every subsequence x of X
3. **if** x is a subsequence of Y
4. $lcs =$ **Max**$(lcs, |x|)$
5. return lcs

Brute Force for LCS

BruteForceLCS(X, Y):

1. $lcs = 0$
2. **for** every subsequence x of X
3. **if** x is a subsequence of Y
4. $lcs =$ **Max**$(lcs, |x|)$
5. return lcs

Generating all the subsequences takes $O(2^{|x|})$ time.

Finding Optimal Substructure in LCS

Finding Optimal Substructure in LCS

Let's try to find LCS of $X =$ "Thiruvananthapuram" and $Y =$ "Visakhapatnam".

Finding Optimal Substructure in LCS

Let's try to find LCS of $X =$ "Thiruvananthapuram" and $Y =$ "Visakhapatnam".

$$X \;=\; \text{T h i r u v a n a n t h a p u r a m}$$

Finding Optimal Substructure in LCS

Let's try to find LCS of $X =$ "Thiruvananthapuram" and $Y =$ "Visakhapatnam".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$

$$Y = \text{V i s a k h a p a t n a m}$$

Finding Optimal Substructure in LCS

Let's try to find LCS of X = "Thiruvananthapuram" and Y = "Visakhapatnam".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$

$$Y = \text{V i s a k h a p a t n a m}$$

Ending characters are same.

Finding Optimal Substructure in LCS

Let's try to find LCS of $X =$ "Thiruvananthapuram" and $Y =$ "Visakhapatnam".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$
$$Y = \text{V i s a k h a p a t n a m}$$
$$\text{LCS}(X, Y) =$$

Ending characters are same.

Finding Optimal Substructure in LCS

Let's try to find LCS of $X =$ "Thiruvananthapuram" and $Y =$ "Visakhapatnam".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$

$$Y = \text{V i s a k h a p a t n a m}$$

$$\text{LCS}(X, Y) = \_\_ \_\_ \_\_ \cdots\cdots \_\_ \_\_$$

Ending characters are same.

Finding Optimal Substructure in LCS

Let's try to find LCS of X = "Thiruvananthapuram" and Y = "Visakhapatnam".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$
$$Y = \text{V i s a k h a p a t n a m}$$
$$\text{LCS}(X, Y) = \_\_ \; \_\_ \; \_\_ \cdots\cdots \_\_ \; \_\_$$

Finding Optimal Substructure in LCS

Let's try to find LCS of X = "Thiruvananthapuram" and Y = "Visakhapatnam".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$

$$Y = \text{V i s a k h a p a t n a m}$$

$$\text{LCS}(X, Y) = \_\_\ \_\_\ \_\_\ \cdots\cdots\ \_\_\ \_\_$$

What should be the last character in LCS(X, Y)?

Finding Optimal Substructure in LCS

Let's try to find LCS of X = "Thiruvananthapuram" and Y = "Visakhapatnam".

X = T h i r u v a n a n t h a p u r a m

Y = V i s a k h a p a t n a m

LCS(X, Y) = __ __ __ $\cdots\cdots$ __ __

What should be the last character in LCS(X, Y)?

"m", because if not, we can append "m" at the

Finding Optimal Substructure in LCS

Let's try to find LCS of $X =$ "Thiruvananthapuram" and $Y =$ "Visakhapatnam".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$

$$Y = \text{V i s a k h a p a t n a m}$$

$$\text{LCS}(X, Y) = \_\_ \; \_\_ \; \_\_ \cdots\cdots \_\_ \; \_\_$$

What should be the last character in LCS(X, Y)?

"m", because if not, we can append "m" at the end of the LCS and get a bigger LCS.

Finding Optimal Substructure in LCS

Let's try to find LCS of X = "Thiruvananthapuram" and Y = "Visakhapatnam".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$

$$Y = \text{V i s a k h a p a t n a m}$$

$$\text{LCS}(X, Y) = \_\,\_\,\_ \cdots\cdots \_\,\underline{\text{m}}$$

What should be the last character in LCS(X, Y)?

"m", because if not, we can append "m" at the end of the LCS and get a bigger LCS.

Finding Optimal Substructure in LCS

Let's try to find LCS of $X =$ "Thiruvananthapuram" and $Y =$ "Visakhapatnam".

$$X \;=\; \mathrm{T\,h\,i\,r\,u\,v\,a\,n\,a\,n\,t\,h\,a\,p\,u\,r\,a\,\cancel{m}}$$

$$Y \;=\; \mathrm{V\,i\,s\,a\,k\,h\,a\,p\,a\,t\,n\,a\,\cancel{m}}$$

$$\mathrm{LCS}(X, Y) \;=\; \_\_\;\_\_\;\_\_\;\cdots\cdots\;\_\_\;\underline{\mathrm{m}}$$

Finding Optimal Substructure in LCS

Let's try to find LCS of X = "Thiruvananthapuram" and Y = "Visakhapatnam".

$$X = \text{T h i r u v a n a n t h a p u r a} \cancel{\text{m}}$$

$$Y = \text{V i s a k h a p a t n a} \cancel{\text{m}}$$

$$\text{LCS}(X, Y) = \underbrace{\_\_ \; \_\_ \; \_\_ \cdots\cdots \_\_}\ \underline{\text{m}}$$

What should be the remaining LCS(X, Y)?

Finding Optimal Substructure in LCS

Let's try to find LCS of $X =$ "Thiruvananthapuram" and $Y =$ "Visakhapatnam".

$$X = \text{T h i r u v a n a n t h a p u r a } \cancel{\text{m}}$$

$$Y = \text{V i s a k h a p a t n a } \cancel{\text{m}}$$

$$\text{LCS}(X, Y) = \underbrace{\_\_\ \_\_\ \_\_\ \cdots\cdots\ \_\_}\ \underline{\text{m}}$$

What should be the remaining LCS(X, Y)?

Intuition says it should be LCS of "Thiruvananthapura~~m~~" and "Visakhapatna~~m~~".

Finding Optimal Substructure in LCS

Finding Optimal Substructure in LCS

Claim: Let $X = x_1x_2...x_m$ and $Y = y_1y_2...y_n$ be two sequences such that $x_m = y_n$. Then,

Finding Optimal Substructure in LCS

Claim: Let $X = x_1 x_2 \ldots x_m$ and $Y = y_1 y_2 \ldots y_n$ be two sequences such that $x_m = y_n$. Then, $Z = \textbf{LCS}(x_1 x_2 \ldots x_{m-1}, y_1 y_2 \ldots y_{n-1}) + x_m$ will be an LCS of X and Y.

Finding Optimal Substructure in LCS

Finding Optimal Substructure in LCS

Let's try to find LCS of $X =$ "Thiruvananthapuram" and $Y =$ "Muzaffarnagar".

Finding Optimal Substructure in LCS

Let's try to find LCS of $X =$ "Thiruvananthapuram" and $Y =$ "Muzaffarnagar".

$$X \;=\; \text{T h i r u v a n a n t h a p u r a m}$$

Finding Optimal Substructure in LCS

Let's try to find LCS of $X =$ "Thiruvananthapuram" and $Y =$ "Muzaffarnagar".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$

$$Y = \text{M u z a f f a r n a g a r}$$

Finding Optimal Substructure in LCS

Let's try to find LCS of X = "Thiruvananthapuram" and Y = "Muzaffarnagar".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$

$$Y = \text{M u z a f f a r n a g a r}$$

Ending characters are different.

Finding Optimal Substructure in LCS

Let's try to find LCS of $X =$ "Thiruvananthapuram" and $Y =$ "Muzaffarnagar".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$

$$Y = \text{M u z a f f a r n a g a r}$$

Finding Optimal Substructure in LCS

Let's try to find LCS of $X =$ "Thiruvananthapuram" and $Y =$ "Muzaffarnagar".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$
$$Y = \text{M u z a f f a r n a g a r}$$

Observation: LCS of X and Y cannot end both "m" and "r":

Finding Optimal Substructure in LCS

Let's try to find **LCS** of $X =$ "**Thiruvananthapuram**" and $Y =$ "**Muzaffarnagar**".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$
$$Y = \text{M u z a f f a r n a g a r}$$

Observation: LCS of X and Y cannot end both "**m**" and "**r**":

- If it doesn't end with "**m**", then **LCS** of X, Y will be:

Finding Optimal Substructure in LCS

Let's try to find **LCS** of X = "**Thiruvananthapuram**" and Y = "**Muzaffarnagar**".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$
$$Y = \text{M u z a f f a r n a g a r}$$

Observation: LCS of X and Y cannot end both "**m**" and "**r**":

- If it doesn't end with "**m**", then **LCS** of X, Y will be:

  LCS of "**Thiruvananthapura~~m~~**" and "**Muzaffarnagar**".

Finding Optimal Substructure in LCS

Let's try to find **LCS** of X = "**Thiruvananthapuram**" and Y = "**Muzaffarnagar**".

$$X = \text{T h i r u v a n a n t h a p u r a m}$$
$$Y = \text{M u z a f f a r n a g a r}$$

Observation: LCS of X and Y cannot end both "**m**" and "**r**":

- If it doesn't end with "**m**", then **LCS** of X, Y will be:

  LCS of "**Thiruvananthapura~~m~~**" and "**Muzaffarnagar**".

- If it doesn't end with "**r**", then **LCS** of X, Y will be:

Finding Optimal Substructure in LCS

Let's try to find **LCS** of $X =$ "**Thiruvananthapuram**" and $Y =$ "**Muzaffarnagar**".

$$X \;=\; \texttt{T h i r u v a n a n t h a p u r a m}$$
$$Y \;=\; \texttt{M u z a f f a r n a g a r}$$

Observation: LCS of X and Y cannot end both "**m**" and "**r**":

- If it doesn't end with "**m**", then **LCS** of X, Y will be:

  LCS of "**Thiruvananthapura~~m~~**" and "**Muzaffarnagar**".

- If it doesn't end with "**r**", then **LCS** of X, Y will be:

  LCS of "**Thiruvananthapuram**" and "**Muzaffarnaga~~r~~**".

Finding Optimal Substructure in LCS

Finding Optimal Substructure in LCS

Claim: Let $X = x_1 x_2 \ldots x_m$ and $Y = y_1 y_2 \ldots y_n$ be sequences such that $x_m \neq y_n$. Then, at least

Finding Optimal Substructure in LCS

Claim: Let $X = x_1x_2\ldots x_m$ and $Y = y_1y_2\ldots y_n$ be sequences such that $x_m \neq y_n$. Then, at least one out of $\mathbf{LCS}(x_1x_2\ldots x_{m-1}, y_1y_2\ldots y_n)$ and $\mathbf{LCS}(x_1x_2\ldots x_m, y_1y_2\ldots y_{n-1})$ will be an $\mathbf{LCS}(X, Y)$.