CS141: Intermediate Data Structures and Algorithms

Dynamic Programming

Amr Magdy
Programming?

- In this context, programming is a tabular method
  - Storing previously calculated results in a table, and look it up later

- Other examples:
  - Linear programming
  - Integer programming
Main idea

Fibonacci recursion tree

$F_n$

$F_{n-1}$

$F_{n-2}$ + $F_{n-3}$

$F_{n-3}$

$F_{n-4}$

0 + 1

0 + 3 1
Main idea

Fibonacci recursion tree
Main idea

- Do not repeat same work, store the result and look it up later

Fibonacci recursion tree
Main idea: DP vs Divide & Conquer

- Do not repeat same work, store the result and look it up later
- Is MergeSort(A, 1, n/2) and MergeSort(A, n/2, n) the same?
- Is Fib(n-2) and Fib(n-2) the same?
Main idea: DP vs Divide & Conquer

- Do not repeat same work, store the result and look it up later
- Is MergeSort(A, 1, n/2) and MergeSort(A, n/2, n) the same?
  - No
- Is Fib(n-2) and Fib(n-2) the same?
  - Yes
Main idea: DP vs Divide & Conquer

- Do not repeat same work, store the result and look it up later
- Is MergeSort(A, 1, n/2) and MergeSort(A, n/2, n) the same?
  - No
- Is Fib(n-2) and Fib(n-2) the same?
  - Yes
- Same function + same input $\rightarrow$ same output (DP)
- DC same function has different inputs
Rod Cutting Problem
Rod Cutting Problem

<table>
<thead>
<tr>
<th>length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>

The table gives the price $p_i$ for a rod of metal of length $i$, for example, a piece of length 8 is sold for $20. Given a rod of length 6 as shown in the picture, cut this rod to sell it for maximum price. For example, if you cut this rod into two pieces, each of length 3, you can sell it total for $8 + $8 = $16.
Rod Cutting Problem
Rod Cutting Problem
Rod Cutting Problem
Rod Cutting Problem
## Rod Cutting Problem

<table>
<thead>
<tr>
<th>length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>
Rod Cutting Problem

<table>
<thead>
<tr>
<th>length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>
Rod Cutting Problem

Given a rod of length $n$ and prices $p_i$, find the cutting strategy that makes the maximum revenue.

In the example: (2+2) cutting makes $r=5+5=10$
Rod Cutting Problem

- Naïve: try all combinations
Rod Cutting Problem

- Naïve: try all combinations
  - How many?
Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1
    - 1 cut: \((n-1)\)
    - 2 cuts: \(\binom{n-1}{2} = \Theta(n^2)\)
    - 3 cuts: \(\binom{n-1}{3} = \Theta(n^3)\)
      ...............
    - n cuts: \(\binom{n-1}{n-1} = \Theta(1)\)
Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1
    - 1 cut: \((n-1)\)
    - 2 cuts: \(\binom{n-1}{2} = \Theta(n^2)\)
    - 3 cuts: \(\binom{n-1}{3} = \Theta(n^3)\)
    - ……
    - n cuts: \(\binom{n-1}{n-1} = \Theta(1)\)
  - Total: \(\Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots+n^3+n^2+n+1)\)
  - Total: \(O(n^n)\)
Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1
    - 1 cut: (n-1)
    - 2 cuts: \( n^{-1}C_2 = \Theta(n^2) \)
    - 3 cuts: \( n^{-1}C_3 = \Theta(n^3) \)
    - …
    - n cuts: \( n^{-1}C_{n-1} = \Theta(1) \)
  - Total: \( \Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots+n^3+n^2+n+1) \)
  - Total: \( O(n^n) \)

- Better solution? Can I divide and conquer?
Rod Cutting Problem

› Naïve: try all combinations
  › How many?
    › 0 cut: 1
    › 1 cut: (n-1)
    › 2 cuts: \( n^{-1}C_2 = \Theta(n^2) \)
    › 3 cuts: \( n^{-1}C_3 = \Theta(n^3) \)
    › n cuts: \( n^{-1}C_{n-1} = \Theta(1) \)
    › Total: \( \Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots+n^3+n^2+n+1) \)
    › Total: \( O(n^n) \)

› Better solution? Can I divide and conquer?
Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1
    - 1 cut: \( (n-1) \)
    - 2 cuts: \( n^{-1}C_2 = \Theta(n^2) \)
    - 3 cuts: \( n^{-1}C_3 = \Theta(n^3) \)
      - ………… n cuts: \( n^{-1}C_{n-1} = \Theta(1) \)
    - Total: \( \Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots+n^3+n^2+n+1) \)
    - Total: \( O(n^n) \)

- Better solution? Can I divide and conquer?
Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1
    - 1 cut: \((n-1)\)
    - 2 cuts: \(n-1\)C\(_2\) = \(\Theta(n^2)\)
    - 3 cuts: \(n-1\)C\(_3\) = \(\Theta(n^3)\)
    - \(n\) cuts: \(n-1\)C\(_{n-1}\) = \(\Theta(1)\)
  - Total: \(\Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots+n^3+n^2+n+1)\)
  - Total: \(O(n^n)\)

- Better solution? Can I divide and conquer?
  - But I don’t really know the best way to divide

```
<table>
<thead>
<tr>
<th>divide</th>
<th>conquer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1  
    - 1 cut: (n-1)  
    - 2 cuts: \( n^{-1}C_2 = \Theta(n^2) \)
    - 3 cuts: \( n^{-1}C_3 = \Theta(n^3) \)
    - \( \ldots \)
    - \( n \) cuts: \( n^{-1}C_{n-1} = \Theta(1) \)
  - Total: \( \Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots+n^3+n^2+n+1) \)
  - Total: \( O(n^n) \)

- Better solution? Can I divide and conquer?
  - But I don’t really know the best way to divide
Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1
    - 1 cut: \( (n-1) \)
    - 2 cuts: \( \binom{n-1}{2} = \Theta(n^2) \)
    - 3 cuts: \( \binom{n-1}{3} = \Theta(n^3) \)
    - \( n \) cuts: \( \binom{n-1}{n-1} = \Theta(1) \)
  - Total: \( \Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots+n^3+n^2+n+1) \)
  - Total: \( O(n^n) \)

- Better solution? Can I divide and conquer?
  - But I don’t really know the best way to divide
Rod Cutting Problem

- Naïve: try all combinations
  - How many?
    - 0 cut: 1
    - 1 cut: \((n-1)\)
    - 2 cuts: \(n^{-1}C_2 = \Theta(n^2)\)
    - 3 cuts: \(n^{-1}C_3 = \Theta(n^3)\)
    - \(\ldots\)
    - \(n\) cuts: \(n^{-1}C_{n-1} = \Theta(1)\)
  - Total: \(\Theta(1+n+n^2+n^3+\ldots+n^{n/2}+\ldots+n^3+n^2+n+1)\)
  - Total: \(O(n^n)\)

- Better solution? Can I divide and conquer?
  - But I don’t really know the best way to divide

\[
r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})
\]
Rod Cutting Problem

Recursive top-down algorithm

\[ r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i}) \]

\textbf{Cut-Rod}(p, n)
1  \textbf{if} n == 0
2    return 0
3  q = -\infty
4  \textbf{for} i = 1 \textbf{to} n
5    q = \max(q, p[i] + \textbf{Cut-Rod}(p, n - i))
6  return q
Rod Cutting Problem

- Better solution? Can I divide and conquer?
  - But I don’t really know the best way to divide

\[ r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i}) \]

- How many subproblems (recursive calls)?
Rod Cutting Problem

Better solution? Can I divide and conquer?
  But I don’t really know the best way to divide

\[ r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i}) \]

How many subproblems (recursive calls)?

```
CUT-ROD(p, n)
1   if n == 0
2     return 0
3   q = -\infty
4   for i = 1 to n
5     q = max(q, p[i] + CUT-ROD(p, n - i))
6   return q
```
Rod Cutting Problem

Better solution? Can I divide and conquer?
  But I don’t really know the best way to divide

\[ r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i}) \]

How many subproblems (recursive calls)?

\[ T(n) = 1 + \sum_{j=0}^{n-1} T(j) . \]
Rod Cutting Problem

- Better solution? Can I divide and conquer?
  - But I don’t really know the best way to divide

\[ r_n = \max_{1 \leq i \leq n} \left( p_i + r_{n-i} \right) \]

- How many subproblems (recursive calls)?

\[ T(n) = 1 + \sum_{j=0}^{n-1} T(j) \]

\[ T(n) = 2^n \]

(Prove by induction)
Rod Cutting Recursive Complexity

Find the complexity of  \( T(n) = 1 + \sum_{j=0}^{n-1} T(j) \)

Proof by induction:

- Assume the solution is some function \( X(n) \)
- Show that \( X(n) \) is true for the smallest \( n \) (the base case), e.g., \( n=0 \)
- Prove that \( X(n+1) \) is a solution for \( T(n+1) \) given \( X(n) \)
- You are done

Given \( T(n) = 1 + \sum_{j=0}^{n-1} T(j) \)

Assume \( T(n) = 2^n \)

\[
T(0) = 1 + \sum_{j=0}^{-1} T(j) = 1 = 2^0 \quad \text{(base case)}
\]

\[
T(n + 1) = 1 + \sum_{j=0}^{n} T(j) = 1 + \sum_{j=0}^{n-1} T(j) + T(n) = T(n) + T(n) = 2T(n) = 2 \times 2^n = 2^{n+1}
\]

Then, \( T(n) = 2^n \)
Rod Cutting Problem

› Better solution? Can I divide and conquer?
  › But I don’t really know the best way to divide

\[ r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i}) \]

› How many subproblems (recursive calls)?

\[ T(n) = 1 + \sum_{j=0}^{n-1} T(j) . \]

\[ T(n) = 2^n \]  

(Prove by induction)

› Can we do better?
Rod Cutting Problem

Better solution? Can I divide and conquer?
  But I don’t really know the best way to divide

$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$

How many subproblems (recursive calls)?

\[ T(n) = 1 + \sum_{j=0}^{n-1} T(j) \]

Can we do better?
Rod Cutting Problem

\[ r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i}) \]

› Subproblem overlapping
  › No need to re-solve the same problem
Rod Cutting Problem

\[ r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i}) \]

- Subproblem overlapping
  - No need to re-solve the same problem
- Idea:
  - Solve each subproblem once
  - Write down the solution in a lookup table (array, hashtable, … etc)
  - When needed again, look it up in \( \Theta(1) \)
Rod Cutting Problem

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

- Subproblem overlapping
  - No need to re-solve the same problem
- Idea:
  - Solve each subproblem once
  - Write down the solution in a lookup table (array, hashtable,…etc)
  - When needed again, look it up in $\Theta(1)$

Dynamic Programming
Rod Cutting Problem

Recursive top-down dynamic programming algorithm

```
MEMOIZED-CUT-ROD (p, n)
1  let r[0..n] be a new array
2  for i = 0 to n
3      r[i] = -\infty
4  return MEMOIZED-CUT-ROD-AUX (p, n, r)

MEMOIZED-CUT-ROD-AUX (p, n, r)
1  if r[n] >= 0
2      return r[n]
3  if n == 0
4      q = 0
5  else q = -\infty
6  for i = 1 to n
7      q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX (p, n - i, r))
8  r[n] = q
9  return q
```
Rod Cutting Problem

Recursive top-down dynamic programming algorithm

\textbf{MEMOIZED-CUT-ROD}\,(p, n)

1. let \( r[0..n] \) be a new array
2. \textbf{for} \( i = 0 \) \textbf{to} \( n \)
3. \hspace{1em} \( r[i] = -\infty \)
4. \textbf{return} \textbf{MEMOIZED-CUT-ROD-AUX}\,(p, n, r)

\textbf{MEMOIZED-CUT-ROD-AUX}\,(p, n, r)

\begin{align*}
1 & \textbf{if} \, r[n] \geq 0 \\
2 & \hspace{1em} \textbf{return} \, r[n] \\
3 & \textbf{if} \, n == 0 \\
4 & \hspace{1em} q = 0 \\
5 & \textbf{else} \, q = -\infty \\
6 & \textbf{for} \, i = 1 \textbf{ to } n \\
7 & \hspace{1em} q = \max(q, p[i] + \text{MEMOIZED-CUT-ROD-AUX}(p, n - i, r)) \\
8 & r[n] = q \\
9 & \textbf{return} \, q
\end{align*}

\( \Theta(n^2) \)
Rod Cutting Problem

- Bottom-up dynamic programming algorithm
  - I know I will need the smaller problems → solve them first
  - Solve problem of size 0, then 1, then 2, then 3, … then n
Rod Cutting Problem

- Bottom-up dynamic programming algorithm
  - I know I will need the smaller problems → solve them first
  - Solve problem of size 0, then 1, then 2, then 3, … then n

```plaintext
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, p[i] + r[j - i])
7      r[j] = q
8   return r[n]
```
Rod Cutting Problem

- Bottom-up dynamic programming algorithm
  - I know I will need the smaller problems → solve them first
  - Solve problem of size 0, then 1, then 2, then 3, … then n

\[
\Theta(n^2) \]

```
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 = -\infty
5        for i = 1 to j
6            q = \max(q, p[i] + r[j - i])
7        r[j] = q
8    return r[n]
```
Elements of a Dynamic Programming Problem

- Optimal substructure
  - Optimal solution of a larger problem comes from optimal solutions of smaller problems

- Subproblem overlapping
  - Same exact sub-problems are solved again and again
Optimal Substructure

- Longest path from A to F LP(A,F) includes node B
  - But it does not include LP(A,B) and LP(B,F)
  - i.e., optimal solutions for the subproblems A→B, and B→F cannot be combined to find an optimal solution for A→F
Dynamic Programming vs. D&C

› How different?
Dynamic Programming vs. D&C

How different?

- No subproblem overlapping
  - Each subproblem with distinct input is a new problem
- Not necessarily optimization problems, i.e., no objective function
Reconstructing Solution

- Rod cutting problem: What are the actual cuts?
  - Not only the best revenue (the optimal objective function value)
Reconstructing Solution

- Rod cutting problem: What are the actual cuts?
  - Not only the best revenue (the optimal objective function value)

```plaintext
EXTENDED-BOTTOM-UP-CUT-Rod(p, n)

1  let r[0..n] and s[1..n] be new arrays
2  r[0] = 0
3  for j = 1 to n
4      q = -∞
5      for i = 1 to j
6          if q < p[i] + r[j - i]
7              q = p[i] + r[j - i]
8              s[j] = i
9          r[j] = q
10  return r and s
```
Reconstructing Solution

- Rod cutting problem: What are the actual cuts?
  - Not only the best revenue (the optimal objective function value)

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

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>r[i]</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>10</td>
<td>13</td>
<td>17</td>
<td>18</td>
<td>22</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>s[i]</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

- Let’s trace examples
Matrix Chain Multiplication

- How to multiply a chain of four matrices $A_1 A_2 A_3 A_4$?
Matrix Chain Multiplication

How to multiply a chain of four matrices $A_1 A_2 A_3 A_4$?

$$(A_1 (A_2 (A_3 A_4)))$$
$$((A_1 A_2) (A_3 A_4))$$
$$((A_1 A_2 A_3) A_4)$$
$$(((A_1 A_2) A_3) A_4)$$
Matrix Chain Multiplication

How to multiply a chain of four matrices $A_1 A_2 A_3 A_4$?

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

Does it really make a difference?
Matrix Chain Multiplication

› How to multiply a chain of four matrices $A_1 A_2 A_3 A_4$?

\[
\begin{align*}
(A_1 (A_2 (A_3 A_4))) \\
(A_1 ((A_2 A_3) A_4)) \\
((A_1 A_2) (A_3 A_4)) \\
(((A_1 A_2) A_3) A_4)
\end{align*}
\]

› Does it really make a difference?

› # of multiplications: $A\.\text{rows} * B\.\text{cols} * A\.\text{cols}$

**Matrix-Multiply** $(A, B)$

1. if $A\.\text{columns} \neq B\.\text{rows}$
2. error “incompatible dimensions”
3. else let $C$ be a new $A\.\text{rows} \times B\.\text{columns}$ matrix
4. for $i = 1$ to $A\.\text{rows}$
5. for $j = 1$ to $B\.\text{columns}$
6. $c_{ij} = 0$
7. for $k = 1$ to $A\.\text{columns}$
8. $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
9. return $C$
Matrix Chain Multiplication

Does it really make a difference?

# of multiplications:
A.rows*B.cols*A.cols

Example:
A1*A2*A3
Dimensions:
10x100x5x50

# of multiplications in ((A1*A2)*A3)=10*100*5+10*5*50=7.5K

# of multiplications in (A1*(A2*A3))=100*5*50+10*100*50=75K
Matrix Chain Multiplication

Given n matrices $A_1 \ A_2 \ \ldots \ A_n$ of dimensions $p_0 \ p_1 \ \ldots \ p_n$, find the optimal parentheses to multiply the matrix chain
Matrix Chain Multiplication

- Given $n$ matrices $A_1, A_2, \ldots, A_n$ of dimensions $p_0, p_1, \ldots, p_n$, find the optimal parentheses to multiply the matrix chain $A_1 A_2 A_3 A_4 A_5 \ldots A_n$
Matrix Chain Multiplication

- Given n matrices $A_1 \ A_2 \ ... \ A_n$ of dimensions $p_0 \ p_1 \ ... \ p_n$, find the optimal parentheses to multiply the matrix chain
- $A_1 \ A_2 \ A_3 \ A_4 \ A_5 \ ... \ A_n$
- $(A_1 \ A_2 \ A_3)(A_4 \ A_5 \ ... \ A_n)$
Matrix Chain Multiplication

- Given n matrices $A_1 A_2 \ldots A_n$ of dimensions $p_0 p_1 \ldots p_n$, find the optimal parentheses to multiply the matrix chain
- $A_1 A_2 A_3 A_4 A_5 \ldots A_n$
- $(A_1 A_2 A_3)(A_4 A_5 \ldots A_n)$
- Sub-chains $C1 = (A_1 A_2 A_3), C2 = (A_4 A_5 \ldots A_n)$
Matrix Chain Multiplication

- Given n matrices $A_1 A_2 \ldots A_n$ of dimensions $p_0 p_1 \ldots p_n$, find the optimal parentheses to multiply the matrix chain
- $A_1 A_2 A_3 A_4 A_5 \ldots A_n$
- $(A_1 A_2 A_3)(A_4 A_5 \ldots A_n)$
- Sub-chains $C1 = (A_1 A_2 A_3), C2 = (A_4 A_5 \ldots A_n)$
- Total Cost $C = \text{cost}(C1) + \text{cost}(C2) + p_0 p_3 p_n$
Matrix Chain Multiplication

- Given $n$ matrices $A_1 A_2 \ldots A_n$ of dimensions $p_0 p_1 \ldots p_n$, find the optimal parentheses to multiply the matrix chain $A_1 A_2 A_3 A_4 A_5 \ldots A_n$
- $(A_1 A_2 A_3)(A_4 A_5 \ldots A_n)$
- Sub-chains $C_1 = (A_1 A_2 A_3)$, $C_2 = (A_4 A_5 \ldots A_n)$
- Total Cost $C = \text{cost}(C_1) + \text{cost}(C_2) + p_0 p_3 p_n$
- Then, if $\text{cost}(C_1)$ and $\text{cost}(C_2)$ are minimal (i.e., optimal), then $C$ is optimal (optimal substructure holds)
Matrix Chain Multiplication

Given n matrices $A_1, A_2, \ldots, A_n$ of dimensions $p_0, p_1, \ldots, p_n$, find the optimal parentheses to multiply the matrix chain

$$A_1 \ A_2 \ A_3 \ A_4 \ A_5 \ \ldots \ \ A_n$$

$$(A_1 \ A_2 \ A_3)(A_4 \ A_5 \ \ldots \ A_n)$$

Sub-chains $C_1 = (A_1 \ A_2 \ A_3), \ C_2 = (A_4 \ A_5 \ \ldots \ A_n)$

Total Cost $C = \text{cost}(C_1) + \text{cost}(C_2) + p_0p_3p_n$

Then, if $\text{cost}(C_1)$ and $\text{cost}(C_2)$ are minimal (i.e., optimal), then $C$ is optimal (optimal substructure holds)

Proof by contradiction:
- Given $C$ is optimal, are $\text{cost}(C_1) = c_1$ and $\text{cost}(C_2) = c_2$ optimal?
- Assume $c_1$ is NOT optimal, then $\exists$ an optimal solution of cost $c_1' < c_1$
- Then $c_1' + c_2 + p < c_1 + c_2 + p \rightarrow C' < C$
- Then $C$ is not optimal $\rightarrow$ contradiction!
- Then $C_1$ has to be optimal $\rightarrow$ optimal substructure holds
Matrix Chain Multiplication

- Given $n$ matrices $A_1, A_2, \ldots, A_n$ of dimensions $p_0, p_1, \ldots, p_n$, find the optimal parentheses to multiply the matrix chain $A_1 A_2 A_3 A_4 A_5 \ldots A_n$
- $(A_1 A_2 A_3)(A_4 A_5 \ldots A_n)$
- Sub-chains $C_1 = (A_1 A_2 A_3)$, $C_2 = (A_4 A_5 \ldots A_n)$
- Total Cost $C = \text{cost}(C_1) + \text{cost}(C_2) + p_0 p_3 p_n$
- Then, if $\text{cost}(C_1)$ and $\text{cost}(C_2)$ are minimal (i.e., optimal), then $C$ is optimal (optimal substructure holds)
- Optimal $C_1, C_2$ might be one of different options
  - $C_1 = (A_1 A_2)$, $C_2 = (A_3 A_4 A_5 \ldots A_n)$
  - $C_1 = (A_1)$, $C_2 = (A_2 A_3 A_4 A_5 \ldots A_n)$
  - $C_1 = (A_1 A_2 A_3 A_4)$, $C_2 = (A_5 \ldots A_n)$
  - .......
Matrix Chain Multiplication

- Assume $k$ is length of first sub-chain $C_1$
Matrix Chain Multiplication

- Assume $k$ is length of first sub-chain $C_1$
Matrix Chain Multiplication

› Assume $k$ is length of first sub-chain $C1$

\[
\begin{align*}
(A_1) & \quad (A_2 A_3 A_4 A_5) \\
(A_1 A_2) & \quad (A_3 A_4 A_5) \\
(A_1 A_2 A_3) & \quad (A_4 A_5)
\end{align*}
\]

› Obviously, a lot of overlapping subproblems appear
Matrix Chain Multiplication

Assume k is length of first sub-chain C1

Obviously, a lot of overlapping subproblems appear
Optimal substructure + subproblem overlapping = dynamic programming
Matrix Chain Multiplication

- Generally: $A_i \ldots A_k \ldots A_j$ of dimensions $p_i \ldots p_k \ldots p_j$
- $(A_i \ldots A_k)(A_{k+1} \ldots A_j)$, where $k=i,i+1,\ldots j-1$
- Then, solve each sub-chains recursively
Matrix Chain Multiplication

- Generally: $A_i \ldots A_k \ldots A_j$ of dimensions $p_i \ldots p_k \ldots p_j$
- $(A_i \ldots A_k) (A_{k+1} \ldots A_j)$, where $k = i, i+1, \ldots j-1$
- Then, solve each sub-chains recursively

\[
m[i, j] = \begin{cases} 
0 & \text{if } i = j \\
\min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j\} & \text{if } i < j
\end{cases}
\]
Matrix Chain Multiplication: Designing Algorithm

What is the smallest subproblem?
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2

- Solve all chains of length 2, then 3, then 4, …n
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n
- Example: n=6,
  A1: 30x35
  A2: 35x15
  A3: 15x5
  A4: 5x10
  A5: 10x20
  A6: 20x25
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2

- Solve all chains of length 2, then 3, then 4, … n

- Example: n=6,

- Chains of length 2:
  - (A1 A2)
  - (A2 A3)
  - (A3 A4)
  - (A4 A5)
  - (A5 A6)
Matrix Chain Multiplication:
Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n
- Example: n=6,
- Chains of length 2:
  
  \[(A_1 \ A_2) = 30 \times 35 \times 15 = 15750\]
  
  \[(A_2 \ A_3)\]
  
  \[(A_3 \ A_4)\]
  
  \[(A_4 \ A_5)\]
  
  \[(A_5 \ A_6)\]

\[A_1: 30 \times 35\]
\[A_2: 35 \times 15\]
\[A_3: 15 \times 5\]
\[A_4: 5 \times 10\]
\[A_5: 10 \times 20\]
\[A_6: 20 \times 25\]
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n
- Example: n=6,
  - Chains of length 2:
    - \((A1 \ A2) = 30*35*15 = 15750\)
    - \((A2 \ A3) = 35*15*5 = 2625\)
    - \((A3 \ A4)\)
    - \((A4 \ A5)\)
    - \((A5 \ A6)\)

\[\begin{align*}
A1 & : 30 \times 35 \\
A2 & : 35 \times 15 \\
A3 & : 15 \times 5 \\
A4 & : 5 \times 10 \\
A5 & : 10 \times 20 \\
A6 & : 20 \times 25
\end{align*}\]
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2

- Solve all chains of length 2, then 3, then 4, …n

- Example: n=6,

- Chains of length 2:
  
  \[(A1 \ A2) = 30 \times 35 \times 15 = 15750\]
  
  \[(A2 \ A3) = 35 \times 15 \times 5 = 2625\]
  
  \[(A3 \ A4) = 15 \times 5 \times 10 = 750\]
  
  \[(A4 \ A5) = 5 \times 10 \times 20 = 1000\]
  
  \[(A5 \ A6) = 10 \times 20 \times 25 = 5000\]

A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n
- Example: n=6,
- Chains of length 3:
  (A1 A2 A3)
  (A2 A3 A4)
  (A3 A4 A5)
  (A4 A5 A6)

A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n
- Example: n=6,
- Chains of length 3:
  \[(A_1 A_2 A_3) = (A_1 A_2) A_3\]
  \[\text{or } A_1 (A_2 A_3)\]

\[(A_2 A_3 A_4)\]
\[(A_3 A_4 A_5)\]
\[(A_4 A_5 A_6)\]
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n
- Example: n=6,
- Chains of length 3:

\[(A_1 A_2 A_3) = (A_1 A_2) A_3 [15750 + 30 \times 15 \times 5]\]

\[\text{or } A_1 (A_2 A_3) [30 \times 35 \times 5 + 2625]\]

\[= 7875\]

\[(A_2 A_3 A_4)\]
\[(A_3 A_4 A_5)\]
\[(A_4 A_5 A_6)\]
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n
- Example: n=6,
- Chains of length 3:
  (A1 A2 A3) = 7875
  (A2 A3 A4) = (A2 A3) A4
    or A2 (A3 A4)

(A3 A4 A5)
(A4 A5 A6)
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …\(n\)
- Example: \(n=6\),
- Chains of length 3:
  
  \[(A1 \ A2 \ A3) = 7875\]

  \[(A2 \ A3 \ A4) = (A2 \ A3) \ A4 \ [2625+35\times5\times10] \]
  
  or \(A2 \ (A3 \ A4) \ [35\times15\times10+750]\)
  
  = 4375

\[(A3 \ A4 \ A5)\]
\[(A4 \ A5 \ A6)\]

A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n
- Example: n=6,
- Chains of length 3:
  - (A1 A2 A3) = 7875
  - (A2 A3 A4) = 4375
  - (A3 A4 A5) = 2500
  - (A4 A5 A6) = 3500
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2

- Solve all chains of length 2, then 3, then 4, … n

- Example: n=6,

- Chains of length 4:
  - (A1 A2 A3 A4) = A1 (A2 A3 A4)
    - Or (A1 A2)(A3 A4)
    - Or (A1 A2 A3) A4

(A2 A3 A4 A5)
(A3 A4 A5 A6)
Matrix Chain Multiplication: Designing Algorithm

› What is the smallest subproblem?
  › A chain of length 2
› Solve all chains of length 2, then 3, then 4, …n
› Example: n=6,
› Chains of length 4:
› \((A1 \ A2 \ A3 \ A4) = A1 \ (A2 \ A3 \ A4)\)
  Or \((A1 \ A2)(A3 \ A4)\)
  Or \((A1 \ A2 \ A3) \ A4\)
  
  = 9375

\((A2 \ A3 \ A4 \ A5)\)
\((A3 \ A4 \ A5 \ A6)\)
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n
- Example: n=6,
- Chains of length 4:
  - (A1 A2 A3 A4) = 9375
  - (A2 A3 A4 A5) = 7125
  - (A3 A4 A5 A6) = 5375
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n
- Example: n=6,
- Chains of length 5:
- \((A_1 \ A_2 \ A_3 \ A_4 \ A_5) = 11875\)
- \((A_2 \ A_3 \ A_4 \ A_5 \ A_6) = 10500\)

A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem? A chain of length 2
- Solve all chains of length 2, then 3, then 4, ...n
- Example: n=6,
- Chains of length 6:
- (A1 A2 A3 A4 A5 A6) = 15125
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n

A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n

A1: 30x35  
A2: 35x15  
A3: 15x5  
A4: 5x10  
A5: 10x20  
A6: 20x25
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n

A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n

\[ A_1A_2A_3 = (A_1A_2)A_3 \]

Or \[ A_1(A_2A_3) \]
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, ... n

\[ A_2A_3A_4 = (A_2A_3)A_4 \]

Or \[ A_2(A_3A_4) \]
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n

\[ A_3A_4A_5 = (A_3A_4)A_5 \]
\[ \text{Or } A_3(A_4A_5) \]
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n

\[ A_4 A_5 A_6 = (A_4 A_5)A_6 \]

Or \[ A_4 (A_5 A_6) \]
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2

- Solve all chains of length 2, then 3, then 4, ...n

$A_1A_2A_3A_4 = A_1(A_2A_3A_4)$
Or $(A_1A_2)(A_3A_4)$
Or $(A_1A_2A_3)A_4$. 

<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>15,750</td>
<td>2,625</td>
<td>750</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>7,875</td>
<td>4,375</td>
<td>2,500</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>9,375</td>
<td>2,500</td>
<td>3,500</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>3,500</td>
<td>5,000</td>
<td>0</td>
</tr>
</tbody>
</table>

A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25
Matrix Chain Multiplication: Designing Algorithm

› What is the smallest subproblem?
  › A chain of length 2

› Solve all chains of length 2, then 3, then 4, …n

A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, ... n

A1: 30x35
A2: 35x15
A3: 15x5
A4: 5x10
A5: 10x20
A6: 20x25
Matrix Chain Multiplication: Designing Algorithm

- What is the smallest subproblem?
  - A chain of length 2
- Solve all chains of length 2, then 3, then 4, …n
Matrix Chain Multiplication

**MATRIX-CHAIN-ORDER**(*p*)

1. \( n = \text{p.length} - 1 \)
2. let \( m[1..n, 1..n] \) and \( s[1..n-1, 2..n] \) be new tables
3. for \( i = 1 \) to \( n \)
4. \( m[i, i] = 0 \)
5. for \( l = 2 \) to \( n \) \hspace{1cm} // l is the chain length
6. for \( i = 1 \) to \( n - l + 1 \)
7. \( j = i + l - 1 \)
8. \( m[i, j] = \infty \)
9. for \( k = i \) to \( j - 1 \)
10. \( q = m[i, k] + m[k + 1, j] + p_i \cdot p_k \cdot p_j \)
11. if \( q < m[i, j] \)
12. \( m[i, j] = q \)
13. \( s[i, j] = k \)
14. return \( m \) and \( s \)
Longest Common Subsequence

\[ S_1 = \text{ACCGGTTCAGTGCAGCGGAAAGCCGGCCGAA} \]
\[ S_2 = \text{GTCGTTTCGGAATGCGTGGCTCTGTAATA} \]

- A string subsequence is an ordered set of characters (not necessarily consecutive)

- A common subsequence of two strings is a subsequence that exist in both strings.

- The longest common subsequence is the common subsequence of the maximum length.
Longest Common Subsequence

Given two strings:

\[ X = \langle x_1, x_2, \ldots, x_m \rangle \]
\[ Y = \langle y_1, y_2, \ldots, y_n \rangle \]

Find the longest common subsequence of X and Y

\[ \text{LCS}(X,Y) \]
Longest Common Subsequence

Given two strings:

\[ X = \langle x_1, x_2, \ldots, x_m \rangle \]

\[ Y = \langle y_1, y_2, \ldots, y_n \rangle \]

Find the longest common subsequence of \( X \) and \( Y \)

\[ \text{LCS}(X,Y) \]

Brute force?
Longest Common Subsequence

Given two strings:

\[ X = \langle x_1, x_2, \ldots, x_m \rangle \]
\[ Y = \langle y_1, y_2, \ldots, y_n \rangle \]

Find the longest common subsequence of X and Y

LCS(X,Y)

- Brute force? \( O(n^2 m) \) or \( O(m^2 n) \) [enumerate all subsequences of X and check in Y, or vice versa]
Longest Common Subsequence

Given two strings: \[ X = \langle x_1, x_2, \ldots, x_m \rangle \]
\[ Y = \langle y_1, y_2, \ldots, y_n \rangle \]

Find the longest common subsequence of X and Y \( \text{LCS}(X,Y) \)

- Brute force? \( O(n^2 m) \) or \( O(m^2 n) \) [enumerate all subsequences of X and check in Y, or vice versa]

- Are smaller problems simpler?
Longest Common Subsequence

Given two strings: 

\[ X = \langle x_1, x_2, \ldots, x_m \rangle \]
\[ Y = \langle y_1, y_2, \ldots, y_n \rangle \]

Find the longest common subsequence of X and Y 
\[ \text{LCS}(X,Y) \]

- Brute force? \( O(n^2m) \) or \( O(m^2n) \) [enumerate all subsequences of X and check in Y, or vice versa]
- Are smaller problems simpler?
- Let’s define string prefixes

\[ X_i = \langle x_1, x_2, \ldots, x_i \rangle, \text{ for } i = 0, 1, \ldots, m \]

- and same for \( Y_j \) for \( j = 0,1,\ldots, n \)
Longest Common Subsequence

Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of $X$ and $Y$.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and $Z_{k-1}$ is an LCS of $X_{m-1}$ and $Y_{n-1}$.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that $Z$ is an LCS of $X_{m-1}$ and $Y$.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that $Z$ is an LCS of $X$ and $Y_{n-1}$.

› Prove by contradiction
Longest Common Subsequence

Let \( X = \langle x_1, x_2, \ldots, x_m \rangle \) and \( Y = \langle y_1, y_2, \ldots, y_n \rangle \) be sequences, and let \( Z = \langle z_1, z_2, \ldots, z_k \rangle \) be any LCS of \( X \) and \( Y \).

1. If \( x_m = y_n \), then \( z_k = x_m = y_n \) and \( Z_{k-1} \) is an LCS of \( X_{m-1} \) and \( Y_{n-1} \).
2. If \( x_m \neq y_n \), then \( z_k \neq x_m \) implies that \( Z \) is an LCS of \( X_{m-1} \) and \( Y \).
3. If \( x_m \neq y_n \), then \( z_k \neq y_n \) implies that \( Z \) is an LCS of \( X \) and \( Y_{n-1} \).

Prove by contradiction
Longest Common Subsequence

Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of $X$ and $Y$.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and $Z_{k-1}$ is an LCS of $X_{m-1}$ and $Y_{n-1}$.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that $Z$ is an LCS of $X_{m-1}$ and $Y$.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that $Z$ is an LCS of $X$ and $Y_{n-1}$.

Prove by contradiction
Longest Common Subsequence

Let $c[i,j]$ is LCS length of $X_i$ and $Y_j$

$$c[i, j] = \begin{cases} 
0 & \text{ if } i = 0 \text{ or } j = 0, \\
c[i - 1, j - 1] + 1 & \text{ if } i, j > 0 \text{ and } x_i = y_j, \\
\max(c[i, j - 1], c[i - 1, j]) & \text{ if } i, j > 0 \text{ and } x_i \neq y_j.
\end{cases}$$
Longest Common Subsequence

Example: X="CS141"  Y="CS111"

\[ c[i, j] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 , \\
 c[i - 1, j - 1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j , \\
 \max(c[i, j - 1], c[i - 1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j . 
\end{cases} \]
## Longest Common Subsequence

Example: \(X=\text{“CS141”} \quad \text{Y=“CS111”}\)

\[
c[i, j] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0, \\
0 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\
\max(c[i, j - 1], c[i - 1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j.
\end{cases}
\]

<table>
<thead>
<tr>
<th></th>
<th>“”</th>
<th>C</th>
<th>S</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>“”</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Longest Common Subsequence

Example: X="CS141"  Y="CS111"

\[
c[i, j] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0, \\
 c[i - 1, j - 1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\
\max(c[i, j - 1], c[i - 1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j . 
\end{cases}
\]

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>S</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot;</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Longest Common Subsequence

LCS-LENGTH(X, Y)

1. \( m \)  \( \text{Length} \)
2. \( n \)  \( \text{Length} \)
3. let \( b[1..m, 1..n] \) and \( c[0..m, 0..n] \) be new tables
4. \( \text{for} \ i = 1 \ \text{to} \ m \)
5. \( c[i, 0] = 0 \)
6. \( \text{for} \ j = 0 \ \text{to} \ n \)
7. \( c[0, j] = 0 \)
8. \( \text{for} \ i = 1 \ \text{to} \ m \)
9. \( \text{for} \ j = 1 \ \text{to} \ n \)
10. \( \text{if} \ x_i == y_j \)
11. \( c[i, j] = c[i - 1, j - 1] + 1 \)
12. \( b[i, j] = "\leftarrow" \)
13. \( \text{elseif} \ c[i - 1, j] >= c[i, j - 1] \)
14. \( c[i, j] = c[i - 1, j] \)
15. \( b[i, j] = "\uparrow" \)
16. \( \text{else} \ c[i, j] = c[i, j - 1] \)
17. \( b[i, j] = "\leftarrow" \)
18. \( \text{return} \ c \ \text{and} \ b \)
Book Readings

› Ch. 15: 15.1-15.4