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 programing
  - Integer programming
Main idea

Fibonacci recursion tree

\[ F_n = F_{n-1} + F_{n-2} \]

\[ F_{n-1} = F_{n-2} + F_{n-3} \]

\[ F_{n-2} = F_{n-3} + F_{n-4} \]

\[ 0 + 1 \]

\[ 0 + 3 \]
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
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>
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: \(\binom{n-1}{3} = \Theta(n^3)\)
    - \(\cdots\)
    - \(n\) 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: $n^{-1}C_2 = \Theta(n^2)$
    - 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: $O(n^n)$
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: \(\binom{n-1}{3} = \Theta(n^3)\)
    - \(\cdots\)
    - \(n\) 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?
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: \( \binom{n-1}{3} = \Theta(n^3) \)
    - \( \cdots \)
    - \( n \) cuts: \( \binom{n-1}{n-1} = \Theta(1) \)

  - Total: \( \Theta(1+n+n^2+n^3+\cdots+n^{n/2}+\cdots+n^3+n^2+n+1) \)
  - Total: \( O(n^n) \)

- Better solution? Can I divide and conquer?

```
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\(_3\) = \(\Theta(n^3)\)
    - ... 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
    - divide
    - 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: \(n^{-1}C_3 = \Theta(n^3)\)
    - \(\ldots\)
    - \(n\) 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: $n^{-1}C_2 = \Theta(n^2)$
    › 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
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

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

Recursive top-down algorithm

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

\[
\text{CUT-ROD}(p, n)
1 \quad \text{if } n == 0
2 \quad \text{return } 0
3 \quad q = -\infty
4 \quad \text{for } i = 1 \text{ to } n
5 \quad q = \max(q, p[i] + \text{CUT-ROD}(p, n - i))
6 \quad \text{return } q \]
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)?

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

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

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

- How many subproblems (recursive calls)?

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

(Prove by induction)
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 $T(n) = 2^n$

- $T(0) = 1 + \sum_{j=0}^{0} T(j) = 1 = 2^0$ (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)?

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

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

- 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 \( \Theta(1) \)
Rod Cutting Problem

Recursive top-down dynamic programming algorithm

\[
\text{MEMOIZED-CUT-ROD}(p, n)
\]

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\((p, n, r)\)

\[
\text{MEMOIZED-CUT-ROD-AUX}(p, n, r)
\]

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

Recursive top-down dynamic programming algorithm

\textsc{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 \textsc{Memoized-Cut-Rod-Aux}(p, n, r)

\textsc{Memoized-Cut-Rod-Aux}(p, n, r)

1. if \( r[n] \geq 0 \)
2. return \( r[n] \)
3. if \( n == 0 \)
4. \( q = 0 \)
5. else \( q = -\infty \)
6. for \( i = 1 \) to \( n \)
7. \( q = \max(q, p[i] + \textsc{Memoized-Cut-Rod-Aux}(p, n - i, r)) \)
8. \( r[n] = q \)
9. return \( q \)
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 \( \rightarrow \) solve them first
  - Solve problem of size 0, then 1, then 2, then 3, … then \( n \)

\[
\text{BOTTOM-UP-CUT-ROD}(p, n)
\]

1. let \( r[0..n] \) be a new array
2. \( r[0] = 0 \)
3. for \( j = 1 \) to \( n \)
   - \( q = -\infty \)
4. for \( i = 1 \) to \( j \)
   - \( q = \max(q, p[i] + r[j - i]) \)
5. \( r[j] = q \)
6. return \( r[n] \)

\( \Theta(n^2) \)
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)

```
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 \((r, s) = \text{EXTENDED-BOTTOM-UP-CUT-ROD}(p, n)\)
2 \textbf{while } n > 0 \\
3 \hspace{0.5cm} \text{print } s[n] \\
4 \hspace{0.5cm} n = n - s[n] \\

\[\begin{array}{c|ccccccccccc}
  i & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
  \hline
  r[i] & 0 & 1 & 5 & 8 & 10 & 13 & 17 & 18 & 22 & 25 & 30 \\
  s[i] & 1 & 2 & 3 & 2 & 2 & 6 & 1 & 2 & 3 & 10 & \\
\end{array}\]

Let’s trace examples
Matrix Chain Multiplication

How to multiply a chain of four matrices $A_1A_2A_3A_4$?
Matrix Chain Multiplication

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

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

How to multiply a chain of four matrices $A_1A_2A_3A_4$?

$(A_1(A_2(A_3A_4)))$

$(A_1(((A_2A_3)A_4)))$

$((A_1A_2)(A_3A_4))$

$((A_1(A_2A_3))A_4)$

$(((A_1A_2)A_3)A_4)$

Does it really make a difference?
Matrix Chain Multiplication

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

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

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

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

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

- Does it really make a difference?

- # of multiplications: $A.\text{rows} \times B.\text{cols} \times A.\text{cols}$

### 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: \(A.\text{rows} \times B.\text{cols} \times A.\text{cols}\)

Example:

\(A1 \times A2 \times A3\)

Dimensions:

10x100x5x50

- # of multiplications in \(((A1 \times A2) \times A3) = 10 \times 100 \times 5 + 10 \times 5 \times 50 = 7.5K\)
- # of multiplications in \((A1 \times (A2 \times A3)) = 100 \times 5 \times 50 + 10 \times 100 \times 50 = 75K\)

```
MATRIX-MULTIPLY(A, B)
1    if A.columns ≠ B.rows
2        error "incompatible dimensions"
3    else let C be a new A.rows × B.columns matrix
4        for i = 1 to A.rows
5            for j = 1 to B.columns
6                c_{ij} = 0
7                for k = 1 to A.columns
8                    c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}
9        return C
```
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 \( A_1 A_2 \ldots A_n \) of dimensions \( p_0 p_1 \ldots p_n \), find the optimal parentheses to multiply the matrix chain
  
  \[
  A_1 A_2 A_3 A_4 A_5 \ldots A_n 
  \]
  
  \[
  (A_1 A_2 A_3)(A_4 A_5 \ldots A_n) 
  \]
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 $C_1 = (A_1 \ A_2 \ A_3)$, $C_2 = (A_4 \ A_5 \ \ldots \ A_n)$
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 $A_1 \ A_2 \ A_3 \ A_4 \ A_5 \ … \ A_n$
- $(A_1 \ A_2 \ A_3)(A_4 \ A_5 \ … \ A_n)$
- Sub-chains $C1 = (A_1 \ A_2 \ A_3)$, $C2 = (A_4 \ A_5 \ … \ A_n)$
- Total Cost $C = \text{cost}(C1)+\text{cost}(C2)+p_0p_3p_n$
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 $C = \text{cost}(C_1)+\text{cost}(C_2)+p_0p_3p_n$
- Then, if $\text{cost}(C_1)$ and $\text{cost}(C_2)$ are minimal (i.e., optimal), then $C$ is optimal (optimal substructure holds)
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

- $C_1 = (A_1 A_2)$, $C_2 = (A_3 A_4 A_5 \ldots A_n)$
- $C_1 = (A_1)$, $C_2 = (A_2 A_3 A_4 A_5 \ldots A_n)$
- $C_1 = (A_1 A_2 A_3 A_4)$, $C_2 = (A_5 \ldots A_n)$
- \ldots
Matrix Chain Multiplication

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

- Assume \( k \) is length of first sub-chain \( C_1 \)

\[
(A_1)(A_2 A_3 A_4 A_5) \quad (A_1 A_2)(A_3 A_4 A_5) \quad (A_1 A_2 A_3)(A_4 A_5) \quad (A_1 A_2 A_3 A_4)(A_5)
\]

\[
(A_2 A_3)(A_4 A_5) \quad (A_2 A_3 A_4)(A_5) \quad (A_1 A_2)(A_3)
\]
Matrix Chain Multiplication

- Assume $k$ is length of first sub-chain $C_1$

  $\begin{align*}
  &k=1 \\
  &k=2 \\
  &k=3 \\
  &k=4 \\
  (A_1)(A_2A_3)(A_4A_5) & (A_1A_2)(A_3A_4A_5) & (A_1A_2A_3)(A_4A_5) & (A_1A_2A_3)(A_4A_5) \\
  (A_2)(A_3A_4A_5) & (A_3)(A_4A_5) & (A_1)(A_2A_3A_4) & (A_1)(A_2A_3A_4) \\
  \end{align*}$

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

› Assume \( k \) is length of first sub-chain \( C_1 \)

- \( A_1 \ A_2 \ A_3 \ A_4 \ A_5 \)
- \( k=1 \)
- \( (A_1) \ (A_2 \ A_3 \ A_4 \ A_5) \)
- \( k=1 \)
- \( (A_2 \ A_3) \ A_4 \ A_5 \)
- \( k=2 \)
- \( (A_2 \ A_3) \ A_4 \)

- \( A_1 \ A_2 \ A_3 \ A_4 \)
- \( k=2 \)
- \( (A_1 \ A_2) \ A_3 \ A_4 \ A_5 \)
- \( k=3 \)
- \( (A_1 \ A_2) \ A_3 \)
- \( k=2 \)
- \( (A_1 \ A_2) \ A_3 \)

- \( A_1 \ A_2 \ A_3 \ A_4 \ A_5 \)
- \( k=4 \)
- \( (A_1 \ A_2 \ A_3) \ A_4 \ A_5 \)

- \( (A_2 \ A_3) \ A_4 \)

- \( (A_1 \ A_2) \ A_3 \)

† 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

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

\begin{array}{cccccc}
A_1 & A_2 & A_3 & A_4 & A_5 & A_6 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 15,750 \\
2 & 0 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 1 & 2 & 0 \\
4 & 0 & 1 & 3 & 2 & 0 \\
5 & 1 & 3 & 2 & 2 & 0 \\
6 & 2 & 3 & 2 & 2 & 0 \\
\end{array}

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

```
  6 1
  5 2 3
  4 3
  3
  2 1
  1
0 0 0 0 0 0 0
A1 A2 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

\[ A_1A_2A_3 = (A_1A_2)A_3 \]

Or \[ A_1(A_2A_3) \]

\[
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 3 & 5 & 5 & 7 & 7 & 7 \\
0 & 4 & 8 & 12 & 15 & 15 & 15 \\
0 & 7 & 13 & 20 & 30 & 45 & 60 \\
0 & 11 & 21 & 35 & 65 & 110 & 165 \\
0 & 16 & 32 & 75 & 145 & 315 & 480 \\
0 & 22 & 48 & 130 & 300 & 740 & 1360 \\
\end{array}
\]

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

$$A_2A_3A_4 = (A_2A_3)A_4$$
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 \((A_1A_2A_3)A_4\).

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: 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

**Matrix-Chain-Order** \((p)\)

1. \(n = p.length - 1\)
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 \(n\) // \(l\) 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. \(q = m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\)
      11. if \(q < m[i, j]\)
         12. \(m[i, j] = q\)
         13. \(s[i, j] = k\)
14. return \(m\) and \(s\)
Longest Common Subsequence

$S_1 = \text{ACCGGTTCGAGTGCAGCGAAGCCGGGCCGAA}$
$S_2 = \text{GTCGTTTCGGAATGCCGTTTGCTCTCTGTAAA}$

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

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 

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

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

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

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

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

<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></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Longest Common Subsequence

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

The longest common subsequence of X and Y can be found using dynamic programming. The problem can be solved by filling a matrix `c[i, j]` where `c[i, j]` represents the length of the longest common subsequence of the first `i` characters of X and the first `j` characters of Y.

The recurrence relation is:

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

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

Example: $X=\text{"CS141"}$  $Y=\text{"CS111"}$

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

<table>
<thead>
<tr>
<th></th>
<th>””</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>1</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>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</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>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Longest Common Subsequence

\textbf{LCS-LENGTH}(X, Y)

1 \hspace{1em} m \hspace{1em} X-length
2 \hspace{1em} n \hspace{1em} Y-length
3 \hspace{1em} let \( b[1 \ldots m, 1 \ldots n] \) and \( c[0 \ldots m, 0 \ldots n] \) be new tables
4 \hspace{1em} \textbf{for} \hspace{0.5em} i \hspace{0.5em} = \hspace{0.5em} 1 \hspace{0.5em} \textbf{to} \hspace{0.5em} m
5 \hspace{1em} \hspace{1em} c[i, 0] = 0
6 \hspace{1em} \textbf{for} \hspace{0.5em} j \hspace{0.5em} = \hspace{0.5em} 0 \hspace{0.5em} \textbf{to} \hspace{0.5em} n
7 \hspace{1em} \hspace{1em} c[0, j] = 0
8 \hspace{1em} \textbf{for} \hspace{0.5em} i \hspace{0.5em} = \hspace{0.5em} 1 \hspace{0.5em} \textbf{to} \hspace{0.5em} m
9 \hspace{1em} \hspace{1em} \textbf{for} \hspace{0.5em} j \hspace{0.5em} = \hspace{0.5em} 1 \hspace{0.5em} \textbf{to} \hspace{0.5em} n
10 \hspace{1em} \hspace{1em} \hspace{1em} \textbf{if} \hspace{0.5em} x_i \hspace{0.5em} == \hspace{0.5em} y_j
11 \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} c[i, j] = c[i - 1, j - 1] + 1
12 \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} b[i, j] = \text{"\hspace{1em} \downarrow\hspace{1em} \"}
13 \hspace{1em} \hspace{1em} \textbf{elseif} \hspace{0.5em} c[i - 1, j] \geq c[i, j - 1]
14 \hspace{1em} \hspace{1em} \hspace{1em} c[i, j] = c[i - 1, j]
15 \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} b[i, j] = \text{"\hspace{1em} \uparrow\hspace{1em} \"}
16 \hspace{1em} \hspace{1em} \textbf{else} \hspace{0.5em} c[i, j] = c[i, j - 1]
17 \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} b[i, j] = \text{"\hspace{1em} \leftarrow \hspace{1em} \"}
18 \hspace{1em} \textbf{return} \hspace{0.5em} c \hspace{0.5em} \text{and} \hspace{0.5em} b
Book Readings

- Ch. 15: 15.1-15.4