Dynamic Programming

Chapter 15
Dictionary Definition

- **Program** (noun) \ˈprō-ˌgram , -grəm\ 
  a sequence of coded instructions that can be inserted into a mechanism (such as a computer)

- **Programming** (noun) \ˈprō-ˌgra-miŋ , -grə-\ 
  a plan of action to accomplish a specified end

- **Synonyms**
  progressing, plan out, arrange
Rod Cutting Problem
Rod Cutting Problem

- Given a rod of length $n$ inches and a table of prices $p_i$ for $i = 1, 2, \ldots, n$, determine the maximum revenue $r_n$ obtainable by cutting up the rod and selling the pieces.
- Naïve solution?
  - Try all possibilities. Exponential!
Recursive Solution

- Define $r_i$ as the maximum possible revenue you can get for a rod of length $i$
- We can express $r_i$ as follows

\[
 r_i = \begin{cases} 
 0 & i = 0 \\
 \max_{1 \leq j < i} (p_j + r_{i-j}) & i > 0 
\end{cases}
\]

- Final answer is $r_n$
def rodcut(i, p)
    return 0 if i == 0
    best_cut = p[i]
    for j in (1..i-1)
        value = rodcut(j, p) + rodcut(i-j, p)
        best_cut = [best_cut, value].max
    end
    return best_cut
end
Execution Recurrence Tree

$n_5$

$n_4$

$n_3$

$n_2$

$n_1$

$n_3$

$n_2$

$n_1$

$n_2$

$n_1$

$n_1$

$n_1$
Memoized Recursive Solution

```python
@best_cuts = []
def rodcut_memoized(i,p):
    return 0 if i == 0
    return @best_cuts[i] if @best_cuts[i]
    best_cut = p[i]
    for j in (1..i-1):
        value = rodcut_memoized(j,p) + rodcut_memoized(i-j,p)
        best_cut = [best_cut, value].max
    end
    return @best_cuts[i] = best_cut
end
```
def rodcut_bottomup(n, p):
    best_cuts = []
    for i in range(1, n+1):
        best_cuts[i] = p[i]
        for j in range(1, i):
            value = best_cuts[j] + best_cuts[i-j]
            best_cuts[i] = [best_cuts[i], value].max
    return best_cuts[n]
Optimal Substructure

- The problem is to find the optimal cut for a rod of length $n$
- Assuming that we cut $n$ at position $i$
- This leads to two subproblem, cutting two rods of lengths $i$ and $n - i$
- The optimal solution when $n$ is cut at position $i$ must include the optimal cuts of rods of lengths $i$ and $n - i$
- Proof by contradiction
Dynamic Programming

- You have a big problem
- You can break it down into smaller subproblems
- You don’t know the best way to split it
  - So, you try all possibilities
- Identify similar subproblems that are solved many times
- Devise a better (polynomial) algorithm
Matrix Chain Multiplication
Example

\[ Cost((A_1A_2)A_3) = 10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50 = 7,500 \]

\[ Cost(A_1(A_2A_3)) = 10 \cdot 100 \cdot 50 + 100 \cdot 5 \cdot 50 = 75,000 \]
Problem

- Given a sequence of (not necessarily square) matrices \(A_1, A_2, \ldots, A_n\) where the dimensions of matrix \(A_i\) is \(p_{i-1} \times p_i\). We want to find the order of matrix multiplication operations to compute the product \(A_1 \cdot A_2 \cdot \cdots \cdot A_n\) while minimizing the number of scalar multiplication operations.

- **Hint:** Number of scalar multiplication operations to compute \(X \cdot Y\) with dimensions \(p \times q\) and \(q \times r\), respectively, is \(p \cdot q \cdot r\).
Recursive Implementation

Matrix-Chain-Multiplication(p, s, e)

- if e <= s then return 0
- min_cost = ∞
- for i = s to e – 1
  - cost = Matrix-Chain-Multiplication(p, s, i) + Matrix-Chain-Multiplication(p, i+1, e) + p[s-1] * p[i+1] * p[e]
  - min_cost = Min(cost, min_cost)
- return min_cost
With Memoization

Matrix-Chain-Multiplication(p, s, e)

if e < s then return 0

return table[s,e] if exists

min_cost = \infty

for i = s to e – 1

\quad cost = \text{Matrix-Chain-Multiplication}(p, s, i) + \\
\quad \text{Matrix-Chain-Multiplication}(p, i+1, e) + \\
\quad p[s-1] * p[i+1] * p[e]

\quad min\_cost = \text{Min}(cost, min\_cost)

\quad table[s,e] = min\_cost

return min\_cost
Recurrence Relation

- $m[i, j]$: Minimum cost for multiplying matrices $i$ through $j$; $1 \leq i \leq j \leq n$
- $m[i, j] = \begin{cases} 
0 & i = j \\
\min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\} & i < j
\end{cases}$
Bottom-up Approach

Cost[$A_i \ldots A_j$]

Final answer
Longest Common Subsequence
Definitions

Sequence: A C C G G T C G A G

Subsequence: A C G A A

Subsequence: G C T A A T

Sequence: G T C G T C G G A A T G C
Definitions

<table>
<thead>
<tr>
<th>Sequence</th>
<th>A</th>
<th>C</th>
<th>C</th>
<th>G</th>
<th>G</th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Common subsequence</strong></td>
<td>G</td>
<td>G</td>
<td>T</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sequence</strong></td>
<td>G</td>
<td>T</td>
<td>C</td>
<td>G</td>
<td>T</td>
<td>C</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>
Definitions

Sequence

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>C</th>
<th>G</th>
<th>G</th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
</table>

Common subsequence

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>G</th>
<th>T</th>
<th>C</th>
</tr>
</thead>
</table>

Longer Common subsequence

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>G</th>
<th>T</th>
<th>C</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
</table>

Sequence

|   | G | T | C | G | T | C | G | G | A | A | T | G | C |
Definitions

Sequence: A C C G G T C G A G

Common subsequence: G G T C

Longer Common subsequence: G G T C A G

Longest Common subsequence: G G T C G A G

Sequence: G T C G T C G G A A T G C
Problem Definition

Given two sequences $X$ and $Y$, we say that a sequence $Z$ is a common subsequence of $X$ and $Y$ if it is a subsequence of both $X$ and $Y$. For example, if $X = \langle A, B, C, B, D, A, B \rangle$ and $Y = \langle B, D, C, A, B, A \rangle$, the sequence $\langle B, C, A \rangle$ is a common subsequence of $X$ and $Y$; not a longest one though. The problem is to find the longest common subsequence of $X$ and $Y$. 
Optimal Substructure

Let $x_m$ and $y_n$ be the last two characters in the two sequences and $z_k$ be the last character in the longest common subsequence.

- $x_m = y_n$
- $x_m \neq y_n$
  - $z_k = x_m$?
  - $z_k = y_n$?
Recursive Algorithm

LCS(X, Y)

- if X[m] == Y[n]
  - return LCS(X[1..m-1], Y[1..n-1]) + 1
- if X[m] != Y[n]
  - return \text{MAX}(LCS(X[1..m], Y[1..n-1]), LCS(X[1..m-1], Y[1..n]))
# Bottom-up Algorithm

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>D</th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Bottom-up Algorithm

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>B</th>
<th>D</th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>j</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Bottom-up Algorithm

<table>
<thead>
<tr>
<th>j</th>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Bottom-up Algorithm

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>D</th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B 2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>C 3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>B 4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>D 5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>A 6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>B 7</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
## Bottom-up Algorithm

<table>
<thead>
<tr>
<th>j</th>
<th>B</th>
<th>D</th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>↑ 0</td>
<td>↑ 0</td>
<td>↑ 0</td>
<td>↘ 1</td>
<td>← 1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>↘ 1</td>
<td>← 1</td>
<td>← 1</td>
<td>↑ 1</td>
<td>↘ 2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>↑ 1</td>
<td>↑ 1</td>
<td>↘ 2</td>
<td>← 2</td>
<td>↑ 2</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>↘ 1</td>
<td>↑ 1</td>
<td>↑ 2</td>
<td>↑ 2</td>
<td>↘ 3</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>↑ 1</td>
<td>↘ 2</td>
<td>↑ 2</td>
<td>↑ 2</td>
<td>↑ 3</td>
</tr>
<tr>
<td>A</td>
<td>6</td>
<td>↑ 1</td>
<td>↑ 2</td>
<td>↑ 2</td>
<td>↘ 3</td>
<td>↑ 3</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>↘ 1</td>
<td>↑ 2</td>
<td>↑ 2</td>
<td>↑ 3</td>
<td>↘ 4</td>
</tr>
</tbody>
</table>
**Bottom-up Algorithm**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>↑0</td>
<td>↑0</td>
<td>↑0</td>
<td>←1</td>
<td>←1</td>
<td>↑1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>←1</td>
<td>←1</td>
<td>←1</td>
<td>↑1</td>
<td>↑1</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>↑1</td>
<td>↑1</td>
<td>←2</td>
<td>←2</td>
<td>↑2</td>
<td>↑2</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>←1</td>
<td>↑1</td>
<td>↑2</td>
<td>↑2</td>
<td>↑2</td>
<td>←3</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>↑1</td>
<td>←2</td>
<td>↑2</td>
<td>↑2</td>
<td>←3</td>
<td>↑3</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>←1</td>
<td>←2</td>
<td>↑2</td>
<td>↑2</td>
<td>↑3</td>
<td>←4</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>↑1</td>
<td>↑2</td>
<td>↑2</td>
<td>←3</td>
<td>↑3</td>
<td>←4</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>←1</td>
<td>↑2</td>
<td>↑2</td>
<td>←3</td>
<td>↑3</td>
<td>←4</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>↑2</td>
<td>↑2</td>
<td>←3</td>
<td>↑3</td>
<td>←4</td>
<td>↑4</td>
</tr>
</tbody>
</table>
LCS-LENGTH(X, Y)

1  m = X.length
2  n = Y.length
3  let b[1..m, 1..n] and c[0..m, 0..n] be new
4  for i = 1 to m
5      c[i, 0] = 0
6  for j = 0 to n
7      c[0, j] = 0
8  for i = 1 to m
9      for j = 1 to n
10         if x_i == y_j
11             c[i, j] = c[i - 1, j - 1] + 1
12             b[i, j] = “↖”
13         elseif c[i - 1, j] >= c[i, j - 1]
14             c[i, j] = c[i - 1, j]
15             b[i, j] = “↑”
16         else c[i, j] = c[i, j - 1]
17             b[i, j] = “←”
18  return c and b
Print-LCS

\[ \text{PRINT-LCS}(b, X, i, j) \]
1 \hspace{1em} \textbf{if} \ i \ == \ 0 \ \textbf{or} \ j \ == \ 0
2 \hspace{1em} \textbf{return}
3 \hspace{1em} \textbf{if} \ b[i, j] \ == \ "\leftarrow"
4 \hspace{1em} \text{PRINT-LCS}(b, X, i - 1, j - 1)
5 \hspace{1em} \text{print } x_i
6 \hspace{1em} \textbf{elseif} \ b[i, j] \ == \ "\uparrow"
7 \hspace{1em} \text{PRINT-LCS}(b, X, i - 1, j)
8 \hspace{1em} \textbf{else} \ \text{PRINT-LCS}(b, X, i, j - 1)
Edit Distance
Minimum Edit Distance

- How to measure the similarity of words or strings?
- Auto corrections: “rationg” -> {“rating”, “ration”}
- Alignment of DNA sequences

An Example of DNA sequence alignment

Adapted from Klug p. 384

Determine the matching score.
Edit Distance

- Defined as the number of edit operations from string $X$ to a string $Y$.
- The edit operations are:
  - Insertion: “ratio” → “ration”
  - Deletion: “rationg” → “rationg”
  - Substitution: “rationg” → “rations”
- Example:
  - Edit distance of “Mickey” and “Mikey” is one deletion operation
Minimum Edit Distance

Given two strings $X$ and $Y$, find the minimum edit distance between them. That is, find the minimum number of edit operations that can be applied on $X$ to transform it to $Y$. Also, find this sequence of operations.
Subproblem Formulation

G O B L I N

G O L D E N
If the last character in both strings is the same, then we know that the optimal solution can copy this value. Why?

The subproblem is formulated by removing the last character of both
If the last character is different, then we do not really what would be the optimal edit.

If we do not know the optimal edit, the next best thing is to try everything.
Subproblem Formulation

$X$

| G | O | B | L | I |

$Y$

| G | O | L | D | E |

$P$

$\begin{array}{c}
X[1..n] \\
Y[1..m-1]
\end{array}$

| G | O | B | L | I |

$P_1$ Insert last character in $Y$

$P_2$ Delete last character in $X$

$P_3$ Substitute the last characters
Recurrence Relation

\( D_{i,j} \): The cost of transforming \( X[1..i] \) to \( Y[1..j] \)

\[
D_{i,j} = \begin{cases} 
\max\{i, j\} & ; i = 0 \lor j = 0 \\
D_{i-1,j-1} & ; i > 0 \land j > 0 \land x_i = y_j \\
\min \left\{ 
D_{i,j-1} + \text{insertion cost}(y_j), \\
D_{i-1,j} + \text{deletion cost}(x_i), \\
D_{i-1,j-1} + \text{substitution cost}(x_i, y_j)
\right\} & ; i > 0 \land j > 0 \land x_i \neq x_j
\end{cases}
\]

\[
D_{i,j} = \begin{cases} 
\max\{i, j\} & ; i = 0 \lor j = 0 \\
D_{i,j-1} + \text{insertion cost}(y_j) & ; i > 0 \land j > 0 \\
D_{i-1,j} + \text{deletion cost}(x_i) & \text{substitution cost}(a, b) = \begin{cases} 
0 & ; a = b \\
1 & ; a \neq b
\end{cases}
\end{cases}
\]

\( D_{i,j} = \begin{cases} 
\max\{i, j\} & ; i = 0 \lor j = 0 \\
D_{i,j-1} + \text{insertion cost}(y_j) & ; i > 0 \land j > 0 \\
D_{i-1,j} + \text{deletion cost}(x_i) & \text{substitution cost}(a, b) = \begin{cases} 
0 & ; a = b \\
1 & ; a \neq b
\end{cases}
\end{cases}
\]
## Bottom-up Algorithm

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>O</th>
<th>L</th>
<th>D</th>
<th>E</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>j</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>