Can you fill a $8 \times 8$ board with the corners missing using dominoes?

Can you tile this?

With these?
Can you fill a $8 \times 8$ board with the corners missing using dominoes?

Can you tile this?

With these?
Today’s Keywords

- Midterm Review
- Dynamic Programming
- Longest Common Subsequence
CLRS Readings

• Chapter 15
Administrativa

- HW5 due March 27 at 11pm
  - Seam Carving!
  - Dynamic Programming (implementation)
  - Java or Python
- Grades Released
  - HW3
  - Midterm
Dynamic Programming

• Requires **Optimal Substructure**
  – Solution to larger problem contains the solutions to smaller ones
• Idea:
  1. Identify recursive structure of the problem
     • What is the “last thing” done?
  2. Select a good order for solving subproblems
     • “Top Down”: Solve each recursively
     • “Bottom Up”: Iteratively solve smallest to largest
  3. Save solution to each subproblem in memory
mem = {}
def myDPalgo(problem):
    if mem[problem] not blank:
        return mem[problem]
    if baseCase(problem):
        solution = solve(problem)
        mem[problem] = solution
        return solution
    for subproblem of problem:
        subsolutions.append(myDPalgo(subproblem))
    solution = OptimalSubstructure(subsolutions)
    mem[problem] = solution
    return solution
Longest Common Subsequence

Given two sequences $X$ and $Y$, find the length of their longest common subsequence

Example:

$X = ATCTGAT$

$Y = TGCATA$

$LCS = TCTA$

Brute force: Compare every subsequence of $X$ with $Y$

$\Omega(2^n)$
Dynamic Programming

• Requires **Optimal Substructure**
  – Solution to larger problem contains the solutions to smaller ones

• Idea:

1. Identify recursive structure of the problem
   • What is the “last thing” done?

2. Select a good order for solving subproblems
   • “Top Down”: Solve each recursively
   • “Bottom Up”: Iteratively solve smallest to largest

3. Save solution to each subproblem in memory
1. Identify Recursive Structure

Let $LCS(i, j) =$ length of the LCS for the first $i$ characters of $X$, first $j$ character of $Y$

Find $LCS(i, j)$:

Case 1: $X[i] = Y[j]$

$X = ATCTGCGT$
$Y = TGCATA$
$LCS(i, j) = LCS(i - 1, j - 1) + 1$

Case 2: $X[i] \neq Y[j]$

$X = ATCTGCGA$
$Y = TGCATAAT$
$LCS(i, j) = LCS(i, j - 1)$

$LCS(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
LCS(i - 1, j - 1) + 1 & \text{if } X[i] = Y[j] \\
\max(LCS(i, j - 1), LCS(i - 1, j)) & \text{otherwise}
\end{cases}$
Dynamic Programming

• Requires **Optimal Substructure**
  – Solution to larger problem contains the solutions to smaller ones

• Idea:
  1. Identify recursive structure of the problem
     • What is the “last thing” done?
  2. Select a good order for solving subproblems
     • “Top Down”: Solve each recursively
     • “Bottom Up”: Iteratively solve smallest to largest
  3. Save solution to each subproblem in memory
1. Identify Recursive Structure

Let $LCS(i, j) = \text{length of the LCS for the first } i \text{ characters of } X,$
first $j$ character of $Y$

Find $LCS(i, j)$:

Case 1: $X[i] = Y[j]$

$X = ATCTGCGT$
$Y = TGCATAT$
$LCS(i, j) = LCS(i - 1, j - 1) + 1$

Case 2: $X[i] \neq Y[j]$

$X = ATCTCGCA$
$Y = TGCATAT$
$LCS(i, j) = LCS(i, j - 1)$

$LCS(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
LCS(i - 1, j - 1) + 1 & \text{if } X[i] = Y[j] \\
\max(LCS(i, j - 1), LCS(i - 1, j)) & \text{otherwise}
\end{cases}$

Save to $M[i,j]$

Read from $M[i,j]$ if present
Dynamic Programming

• Requires **Optimal Substructure**
  – Solution to larger problem contains the solutions to smaller ones

• Idea:
  1. Identify recursive structure of the problem
     • What is the “last thing” done?
  2. Select a good order for solving subproblems
     • “Top Down”: Solve each recursively
     • “Bottom Up”: Iteratively solve smallest to largest
  3. Save solution to each subproblem in memory
2. Solve in a Good Order

\[
LCS(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
LCS(i - 1, j - 1) + 1 & \text{if } X[i] = Y[j] \\
\max(LCS(i, j - 1), LCS(i - 1, j)) & \text{otherwise}
\end{cases}
\]

To fill in cell \((i, j)\) we need cells \((i - 1, j - 1), (i - 1, j), (i, j - 1)\)

Fill from Top->Bottom, Left->Right (with any preference)
Run Time?

\[ LCS(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
LCS(i - 1, j - 1) + 1 & \text{if } X[i] = Y[j] \\
\max(LCS(i, j - 1), LCS(i - 1, j)) & \text{otherwise}
\end{cases} \]

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X =</td>
<td>0</td>
<td>A</td>
<td>T</td>
<td>C</td>
<td>T</td>
<td>G</td>
<td>A</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>Y =</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></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>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Run Time: \( \Theta(n \cdot m) \) (for \( |X| = n, |Y| = m \))
Reconstructing the LCS

\[ LCS(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
LCS(i - 1, j - 1) + 1 & \text{if } X[i] = Y[j] \\
\max(LCS(i, j - 1), LCS(i - 1, j)) & \text{otherwise}
\end{cases} \]

Start from bottom right, if symbols matched, print that symbol then go diagonally else go to largest adjacent

```
X = 0  A  T  C  T  G  A  T
Y = T 1  G 2  C 3  A 5 4
    0  0  0  0 0 0 0  0
    0  0 0 1 1 1 1 1 1
    0 0   1 1 1 2 2 2 2
    0 0   1 2 2 2 3 3 3
    0 1  1 2 2 2 3 3 4
    0 1  2 2 3 3 4 4 4
```
Reconstructing the LCS

\[ \text{LCS}(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
\text{LCS}(i - 1, j - 1) + 1 & \text{if } X[i] = Y[j] \\
\max(\text{LCS}(i, j - 1), \text{LCS}(i - 1, j)) & \text{otherwise}
\end{cases} \]

Start from bottom right,
if symbols matched, print that symbol then go diagonally
else go to largest adjacent
Reconstructing the LCS

\[
LCS(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
LCS(i - 1, j - 1) + 1 & \text{if } X[i] = Y[j] \\
\max(LCS(i, j - 1), LCS(i - 1, j)) & \text{otherwise}
\end{cases}
\]

Start from bottom right,
if symbols matched, print that symbol then go diagonally
else go to largest adjacent