COP3530
DATA STRUC/ALGORITHMS

Spring 2013 Discussion 13
Dynamic Programming

- Used to solve optimization problems
- Akin to divide and conquer; breaks complex problems into simpler sub-problems
- Can be used if the problem exhibits
  - Overlapping sub-problems
  - Optimal substructure
Dynamic Programming

- Overlapping sub-problems
  - Any recursive algorithm solving the problem should encounter the same sub-problem over and over

```
Fibonacci(5)
  └── Fibonacci(3)
        └── Fibonacci(1)
        └── Fibonacci(2)
              └── Fibonacci(0)
```
Dynamic Programming

- Optimal substructure
  - The solution to a given optimization problem can be obtained by the combination of optimal solutions to its sub-problems
  - The shortest path from A to G passing through D must contain the shortest path from A to D and the shortest path from D to G
Dynamic Programming

Unlike dynamic programming, divide and conquer involves combining optimal solutions to non-overlapping sub-problems.

- Merge sort
- Quick sort
A dynamic programming algorithm will examine all possible ways to solve the problem and will pick the best solution:

- Think of it as an intelligent brute force algorithm.

- We save computation time and not solve problems that we’ve solved before by employing memoization.
Example: Longest Common Subsequence (LCS)

Given the two sequences $x[1…m]$ and $y[1…n]$, find a longest subsequence common to both.

$x$: A B C B D A B

$y$: B D C A B A

Can we use dynamic programming?

- Optimization problem?
- Overlapping sub-problems?
- Optimal substructure?
Dynamic Programming

- Approaching the problem
  - Look at the length of the longest common subsequence
  - Extend the algorithm to find the LCS itself

### Recursive Algorithm

Let $c[i, j] = |LCS(x[1 \ldots i], y[1 \ldots j])|$  

$LCS(i, j)$

if $x[i] = y[j]$
  then $c[i, j] \leftarrow LCS(i - 1, j - 1) + 1$
else $c[i, j] \leftarrow \max\{LCS(i - 1, j), LCS(i, j - 1)\}$

return $c[i, j]$
Dynamic Programming

**Recursive Algorithm**
Let \( c[i, j] = |LCS(x[1 \ldots i], y[1 \ldots j])| \)

\( LCS(i, j) \)
- if \( x[i] = y[j] \)
  - then \( c[i, j] \leftarrow LCS(i - 1, j - 1) + 1 \)
  - else \( c[i, j] \leftarrow \max\{LCS(i - 1, j), LCS(i, j - 1)\} \)
- return \( c[i, j] \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>B</th>
<th>D</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>D</th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
Dynamic Programming

- Recursion Tree

The height is $m+n$, so we will be doing exponential work, but we can use memoization!
Dynamic Programming

- Memoization: keeping track of solutions to problems we’ve solved before

```
     A   B   C   B   D   A   B
   --- --- --- --- --- --- ---
    0   0   0   0   0   0   0   0
   B   0   0   1   1   1   1   1   1
   D   0   0   1   1   1   2   2   2
   C   0   0   1   2   2   2   2   2
   A   0   1   1   2   2   2   3   3
   B   0   1   2   2   3   3   3   4
   A   0   1   2   2   3   3   4   4
```
Dynamic Programming

- Backtrack to find the answer

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>B</th>
<th>D</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</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>B</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</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>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>A</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>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Space: $O(mn)$
Time: $O(mn)$
Determine if you can use dynamic programming
- Optimization problem?
- Overlapping sub-problems?
- Optimal substructure?

Formulate recursive algorithm
- Examine all possible ways to solve the problem, brute-force

Use memoization to reduce computation time