Dynamic Programming Concluded

April 9–10, 2012
CS 60: Principles of Computer Science

Assignment 10 (Dynamic Programming) due Monday, April 16.
Recall: Fibonacci Numbers

The Fibonacci numbers are easy to compute!

\[
\text{fib}(n) = \begin{cases} 
  n & \text{if } n = 0 \text{ or } n = 1 \\
  \text{fib}(n - 2) + \text{fib}(n - 1) & \text{if } n \geq 2
\end{cases}
\]

The corresponding recursive code is exponentially slow for large \( n \).
**The Problem**

\[
\text{fib}(n) = \begin{cases} 
n & \text{if } n = 0 \text{ or } n = 1 \\
\text{fib}(n - 2) + \text{fib}(n - 1) & \text{if } n \geq 2
\end{cases}
\]
**Dynamic Programming**

**Idea 1: “Memoizing”**

Remember all the inputs and output so far
Return precomputed answers for repeated questions.

\[
\text{fib}(n) = \begin{cases} 
  n & \text{if } n = 0 \text{ or } n = 1 \\
  \text{fib}(n-2) + \text{fib}(n-1) & \text{if } n \geq 2 
\end{cases}
\]
Idea 2: “Dynamic Programming”

Compute each value exactly once, in a “clever” order. Ensure that problems are solved after their subproblems.

\[ \text{fib}(n) = \begin{cases} n & \text{if } n = 0 \text{ or } n = 1 \\ \text{fib}(n - 2) + \text{fib}(n - 1) & \text{if } n \geq 2 \end{cases} \]
Recall: The Knapsack Problem

Suppose we have \( n \) kinds of items.

✓ They have value (or utility) \( v_1, \ldots, v_n \)

✓ Each has weight (or size or cost) \( w_1, \ldots, w_n \).

What should we choose, if the maximum total weight is \( W \)?

\[
\text{opt}_v(0) = 0
\]

\[
\text{opt}_v(W) = \max \begin{cases} 
\text{opt}_v(W - 1) \\
 v_1 + \text{opt}_v(W - w_1) \\
 v_2 + \text{opt}_v(W - w_2) \\
\vdots \\
v_n + \text{opt}_v(W - w_n)
\end{cases}
\]
**Floyd-Warshall**

An algorithm for finding all-pairs shortest paths!

For \( k = 0, 1, 2, \ldots \), compute:

*What is the shortest path from \( s \) to \( d \) via nodes \( 1..k \) only?*
FLOYD-WARSHALL ALGORITHM

Minimum distance from src to dst using intermediate nodes 1..k

\[ T[k][src][dst] = \min \begin{cases} T[k-1][src][dst] \\ T[k-1][src][k] + T[k-1][k][dst] \end{cases} \] use \( k \)

\[ T[k-1][src][dst] \] lose \( k \)
**Example: K = 0**

- **Diagram:**
  - Node 1 connected to node 2 with weight 14.
  - Node 1 connected to node 3 with weight 14.
  - Node 2 connected to node 1 with weight 14.
  - Node 2 connected to node 4 with weight 100.
  - Node 3 connected to node 4 with weight 100.
  - Node 2 connected to node 4 with weight 50.

- **Table (from to):**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>14</td>
<td>inf</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>inf</td>
<td>0</td>
<td>14</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>inf</td>
<td>inf</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>inf</td>
<td>inf</td>
<td>0</td>
</tr>
</tbody>
</table>
Example: k = 1
Example: \( k = 2 \)
**Example:** $k = 2$
**Example:** $k = 4$ (DONE)