Memoization vs Dynamic Programming

- **Definition** A recurrence is a way to define a function on the (tuples of) nonnegative integers by giving two things: (1) values of the function for a finite number of small base cases, (2) values of the function for the rest of the cases in terms of values of the function on smaller arguments.

Example: The Fibonacci numbers are defined as

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

- As demonstrated in class, naively implementing the recurrence gives an inefficient algorithm because of repeated computation.

- Memoization is one technique for writing efficient programs to solve a recurrence whose definition gives an inefficient algorithm if implemented top-down.

- Ideas of memoization:
  - Use a table to store previously computed values.
  - Initialize the whole table to “UNKNOWN VALUE.”
  - Each time a value is computed and known, immediately store it into the table.
  - Any time a value is needed, check first whether it is already stored in the table. If it is, use it; otherwise, compute its value.
  - To compute the value of the function on any argument, use the recurrence and the above two ideas.

- Like memoization, dynamic programming is appropriate for solving a problem that can be described by a recurrence.

- Ideas of dynamic programming:
– Use a table to store computed values.

– To solve a problem by dynamic programming, perform the following two steps.

1. Fill in all entries of the table **bottom-up** using the recurrence.

2. Return the desired value from the table.

• Comparison of Memoization and Dynamic Programming:

<table>
<thead>
<tr>
<th></th>
<th>memoization</th>
<th>dp</th>
</tr>
</thead>
<tbody>
<tr>
<td>implementation of the recurrence</td>
<td>top-down</td>
<td>bottom-up</td>
</tr>
<tr>
<td>initialization of table</td>
<td>needed</td>
<td>not needed</td>
</tr>
<tr>
<td># table entries computed</td>
<td>partial</td>
<td>complete</td>
</tr>
<tr>
<td>space-saving improvement</td>
<td>difficult</td>
<td>easier</td>
</tr>
</tbody>
</table>