Announcements

- Project #4: "Intelligent scissors"
  - Due: today
- Project #5: Gene Sequence Alignment
  - Begin discussing main ideas on Wednesday
- Reading: worth your time
- Mid-term Exam: coming up next week

Objectives

- Use the Dynamic Programming strategy to solve another example problem: "the coins problem"
- Extract the composition of a solution from a DP table

The Coins Problem: Making Change

- Problem: Given \( T \), what's the smallest number of coins to make change for \( T \) from denominations \( d = \{ d_1, d_2, ..., d_m \} \)?
  - Have unlimited supply of each type of coin
- Example:
  - \( m = 4 \) different denominations
  - Denominations \( d = \{ 1, 5, 10, 25 \} \)
  - Need a volunteer
  - Take one coin at a time.
- That's a greedy algorithm

The Coins Problem: Making Change

- In general, does a greedy algorithm always give the optimal answer?
  - Consider: \( d = \{ 1, 3, 4 \} \), \( j = 6 \)
  - No. Greedy can fail.
- Goal: Give optimal change for any balance using any currency system.
- Let's apply the Dynamic Programming strategy.

DP Strategy: From Problem to Table to Algorithm

1. Start with a problem definition
2. Devise a minimal description (address) for any problem instance and sub-problem
3. Define recurrence to specify the relationship of problems to sub-problems
   - i.e., Define the conceptual DAG on sub-problems
4. Embed the DAG in a table
   - Use the address as indexes:
     - E.g., 2-D case: index rows and columns
5. Two possible strategies
   1. Fill in the sub-problem cells, proceeding from the smallest to the largest.
   2. Draw the DAG in the table from the top problem down to the smallest sub-problems, solve the relevant sub-problems in their table cells, from smallest to largest.
   - Equivalently: solve the top problem instance recursively, using the table as a memory function
Making Optimal Change using DP

- **Problem:** What's the smallest number of coins to make change for balance \( t \) from denominations \( d_1, d_2, \ldots, d_n \)?

- **Define problem representation:**
  - Let \( C(t, j) \) = minimum number of coins needed to make change for balance \( t \) using coins of type 1 through \( j \).

- **Define sub-problem representation:**
  - Let \( C(i, j) \) = minimum number of coins needed to make change for balance \( i \) using coins of type 1 through \( j \).

- **Key ideas for relationship among these sub-problems as a recurrence relation:**
  - Take one coin at a time.
  - If \( i < j \), then you cannot use a coin of value \( j \).
  - If \( j \geq d_i \), binary choice: either take a coin of value \( j \) or do not take one.
  - If you do take one, then make change for the remaining balance using coins of values \( j \) through \( j-1 \).
  - If you don't take one, then make change for the unchanged balance using coins of values \( j \) through \( 0 \).

\[
C(i, j) = \begin{cases} 
0 & \text{if } i < 1 \\
\frac{a}{h} & \text{if } i = 1 \\
\min(C(i-j, d_j) + 1, C(i-1, j)) & \text{if } i > 1 \land j \geq d_i \\
C(i-1, j) & \text{if } i > 1 \land 0 < j < d_i \\
0 & \text{if } i > 1 \land j \leq 0 
\end{cases}
\]

**Next Steps Using DP**

- Design the table to contain the needed sub-problem results
- Design the DP algorithm to walk the table and apply the recurrence relation
- Solve an example by running the DP algorithm
- Modify the resulting algorithm for space and time, if necessary

**Example**

- \( m = 4 \) different denominations
- Denominations \( d = [1, 2, 4, 7] \)
- Look familiar?

\[
C(i, j) = \min \{C(i-j, d_j) + 1, C(i-1, j)\}
\]

<table>
<thead>
<tr>
<th>Amount</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>seni=1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>sean=2</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>shum=3</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>simnah=4</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

\( C(i, j) = \text{min. number of "coins" to make change with coins } 1 \ldots i. \)
Making Change

How does one compute \( C(2,2) \)?

\[
C(i, j) = \min \{ C(i, j - d_i) + 1, C(i - 1, j) \}
\]

\[
C(2, 2) = \min \{ C(2, 1) + 1, C(1, 2) \} = \min \{ 1, 1 \} = 1
\]

Making Change

How does one compute \( C(2,3) \)?

\[
C(i, j) = \min \{ C(i, j - d_i) + 1, C(i - 1, j) \}
\]

\[
C(2, 3) = \min \{ C(2, 2) + 1, C(1, 3) \} = \min \{ 1, 1 \} = 1
\]

Making Change

\[
C(i, j) = \min \{ C(i, j - d_i) + 1, C(i - 1, j) \}
\]

\[
C(3, 3) = \min \{ C(3, 2) + 1, C(2, 3) \} = \min \{ 2, 1 \} = 1
\]

Making Change

\[
C(i, j) = \min \{ C(i, j - d_i) + 1, C(i - 1, j) \}
\]

\[
C(3, 4) = \min \{ C(3, 3) + 1, C(2, 4) \} = \min \{ 2, 1 \} = 1
\]
### Making Change

<table>
<thead>
<tr>
<th>Amount</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>senine=1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>seon=2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>shum=4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>limnah=7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

**Recurrence Relation**

\[ C(i, j) = \min \{ C(i, j - d_i) + 1, C(i - 1, j) \} \]

**Extracting a Solution:**

\[ C(4, 7) = C(4, 7-7)+1, \text{ so include a limnah} \]

**Move to C(4,7-7).**

### Question

- Which coins?
- Extract the composition of a solution from the table.
Extracting a Solution: C(3,7)

Algorithm in Pseudo-code

Efficiency

Pre-computing vs. on-the-fly

Making Change: Top Down

Comparison
Assignment

- HW #15
- Read 6.3
- Read Project #5 Instructions