Dynamic Programming

- Examples: 5.10, 6.8
- Today:
  - Example 1 – Licking Stamps
  - General Principles
  - Example 2 – Knapsack
- Tomorrow
  - Example 3 – Sequence Comparison

Licking Stamps

- Given:
  - Large supply of 5¢, 4¢, and 1¢ stamps
  - An amount N
- Problem: choose fewest stamps totaling N

How to Lick 27¢

<table>
<thead>
<tr>
<th># of 5¢ Stamps</th>
<th># of 4¢ Stamps</th>
<th># of 1¢ Stamps</th>
<th>Total Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

Moral: Greed doesn’t pay

A Simple Algorithm

- At most N stamps needed, etc.
  ```
  for a = 0, ..., N {
      for b = 0, ..., N {
          for c = 0, ..., N {
              if (5a+4b+c == N && a+b+c is new min)
                  {retain (a,b,c);}})
      output retained triple;
  ```
- Time: O(N^3)
  (Not too hard to see some optimizations, but we’re after bigger fish…)

“Dynamic Programming”

Program — A plan or procedure for dealing with some matter — Webster’s New World Dictionary
Better Idea

**Theorem:** If last stamp licked in an optimal solution has value v, then previous stamps form an optimal solution for N-v.

**Proof:** if not, we could improve the solution for N by using opt for N-v.

\[
M(i) = \min \begin{cases} 
0 & i = 0 \\
1 + M(i-5) & i \geq 5 \\
1 + M(i-4) & i \geq 4 \\
1 + M(i-1) & i \geq 1
\end{cases}
\]

where \(M(i)\) = min number of stamps totaling \(i\)

New Idea: Recursion

\[
M(i) = \min \begin{cases} 
0 & i = 0 \\
1 + M(i-5) & i \geq 5 \\
1 + M(i-4) & i \geq 4 \\
1 + M(i-1) & i \geq 1
\end{cases}
\]

Finding How Many Stamps

\[1 + \text{Min}(3, 1, 3) = 2\]

Another New Idea: Avoid Recomputation

- Tabulate values of solved subproblems
  - Top-down: “memoization”
  - Bottom up:
    \[
    \text{for } i = 0, \ldots, N \text{ do } M(i) = \min \begin{cases} 
0 & i = 0 \\
1 + M(i-5) & i \geq 5 \\
1 + M(i-4) & i \geq 4 \\
1 + M(i-1) & i \geq 1
\end{cases}
\]
- Time: \(O(N)\)

Finding Which Stamps: Trace-Back

\[1 + \text{Min}(3, 1, 3) = 2\]

Complexity Note

- \(O(N)\) is better than \(O(N^3)\) or \(O(3^{N/5})\)
- But still exponential in input size (log N bits)
  - (E.g., miserably slow if N is 64 bits)
- Note: can do in \(O(1)\) for 5¢, 4¢, and 1¢ but not in general. See “NP-Completeness” later
Elements of Dynamic Programming

- What feature did we use?
- What should we look for to use again?
- “Optimal Substructure”
  Optimal solution contains optimal subproblems
- “Repeated Subproblems”
  The same subproblems arise in various ways

The Knapsack Problem (§ 5.10)

**Given** positive integers $W, w_1, w_2, \ldots, w_n$

**Find** a subset of the $w_i$'s totaling exactly $W$.

(Like stamp problem, but limited supply of each.)

**Motivation:** simple 1-d abstraction of packing boxes, trucks, VLSI chips, …