Dynamic Programming

"Dynamic Programming"

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

---

Dynamic Programming

- Outline:
  - Example 1 — Licking Stamps
  - General Principles
  - Example 2 — Knapsack (§ 5.10)
  - Example 3 — Sequence Comparison (§ 6.8)

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.
  
  \[
  \text{for } a = 0, \ldots, N \{ \\
  \quad \text{for } b = 0, \ldots, N \{ \\
  \quad \quad \text{for } c = 0, \ldots, N \{ \\
  \quad \quad \quad \text{if } (5a + 4b + c == N \&\& a+b+c \text{ is new min}) \\
  \quad \quad \quad \quad \{ \text{retain } (a,b,c) \}; \}\}
  \]
  
  output retained triple;

- Time: \(O(N^3)\)
  
  (Not too hard to see some optimizations, but we’re after bigger fish…)

---

**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 \(N-v\).

\[
M(i) = \min \begin{cases} 
0 & i=0 \\
1+M(i-5) & i\geq5 \\
1+M(i-4) & i\geq4 \\
1+M(i-1) & i\geq1 
\end{cases}
\]

where \(M(0) = \text{min number of stamps totaling } i\)

---

**New Idea: Recursion**

\[
M(i) = \min \begin{cases} 
0 & i=0 \\
1+M(i-5) & i\geq5 \\
1+M(i-4) & i\geq4 \\
1+M(i-1) & i\geq1 
\end{cases}
\]

Time: \(>3^{4/3}\)
Another New Idea: Avoid Recomputation

- Tabulate values of solved subproblems
  - Top-down: “memoization”
  - Bottom up:
    
    for \( i = 0, \ldots, N \) 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 How Many Stamps

Finding Which Stamps: Trace-Back

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 – \( \sim 2^{64} \) steps for 64 bit input.)

- 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
  A non-example: min (number of stamps mod 2)

• “Repeated Subproblems”
  The same subproblems arise in various ways