Dynamic Programming II

Solving a problem by identifying a collection of subproblems and solving them smallest to largest.

Last Time:
1. Fibonacci
2. Shortest paths in DAGs
3. Longest increasing subsequences
4. Edit distance

This time: More problems.

DP design steps:

1. Identifying subproblems \((F_1, ..., F_n)\)
2. Compute recurrence \((F_n = F_{n-1} + F_{n-2})\)
3. Design algorithm (bottom-up Fib)
Knapsack

Input: total weight $W$
- $n$ items with weights $w_1, \ldots, w_n$
- and values $V_1, \ldots, V_n$

Output: Most valuable combination of items
- with total weight $\leq W$

Two variants: with repetition v. without repetition

$W = 10$

<table>
<thead>
<tr>
<th>Item</th>
<th>Weight</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>$30</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>$14</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>$16</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>$9</td>
</tr>
</tbody>
</table>

With replacement:
- $(\text{item 1}) + 2 \cdot (\text{item 4}) = $48

Without replacement:
- $(\text{item 1}) + (\text{item 3}) = $46
Knapsack with replacement/repetition

1. Identify subproblems

best knapsack of weight \( W \)

\[ l_1 l_2 \ldots l_{k-1} l_k = l \]

best knapsack with weight \( \leq W - w_i \)

\[ K(l) = \max \{ v_i + K(l - w_i) \} \]

\( i : w_i \leq l \)

\[ K(l) = \max \{ v_i + K(l - w_i) \} \]

\( i : w_i \leq l \)

K(0) = 0

for \( c = 1 \) to \( W \)

\[ K(c) = \max \{ v_i + K(c - w_i) \} \]

\( i : w_i \leq c \)

return \( K(W) \)

Runtime: \( O(n W) \)
Knapsack without repetition

1. Identify subproblems
   \[ K(c) = \max \text{ value achievable w/ weight } \leq C \]
   \[ \text{best knapsack w/ weight } \leq C - w_i \]
   \[ \text{not including } i \]
   \[ K(C, j) = \max \text{ value achievable w/ total weight } \leq C \]
   \[ \text{only using items } 1, \ldots, j \]

2. Recurrence?
   \[ \text{Opt } K(c, j) = \begin{cases} 1 & \text{Yes} \\ 2 & \text{No} \\ \ldots & \text{Yes} \\ j & \text{Yes/No} \end{cases} \]
   \[ \text{Opt } K(C - w_j, j-1), K(C, j-1) \]
   \[ K(C, j) = \max \left\{ v_j + K(C - w_j, j-1), K(C, j-1) \right\} \]

Base case:
   \[ K(0, j) = 0, \quad K(C, 0) = 0 \]
3. Algorithm

Initialize all $K(0,j)$ and $K(C,0)$ to 0

for $j = 1$ to $n$
  
  for $C = 1$ to $W$
    
    if $w_j > C$: $K(C,j) = K(C,j-1)$
    
    else: $K(C,j) = \max \{ v_j + K(C-w_j,j-1), K(C,j-1) \}$

return $K(W,n)$

\[
K: \quad \text{Runtime: } \# \text{subproblems} = O(nW) \\
\phantom{K:} \text{time per subproblem} = O(1) \\
\phantom{K:} \text{total: } O(nW)
\]

\[
\text{Space: } O(nW) \\
\phantom{\text{Space: } } O(W)
\]
Shortest path

Input: directed \( G = (V,E) \), lengths for each edge \( \ell(u,v) \)
\( s,t \in V \), integer \( K \)

Output: Length of shortest \( s \to t \) path only using \( K \) edges

1. Subproblems?

\[ \text{dist}(v,i) = \text{length of shortest } s \to v \text{ path using } \leq i \text{ edges} \]

2. Recurrence?

Shortest \( s \to v \) path w/ \( \leq i \) edges

\[ S \to V_1 \to V_2 \to \ldots \to V_{i-1} \to v_0 = v \]

\[ \text{dist}(v,i) = \min \{ \text{dist}(u,i-1) + \ell(u,v), \text{dist}(v,i-1) \} \]

(positive or negative)
3. Algorithm

Initialize all \( \text{dist}(v, 0) = \infty \)

\[ \text{dist}(s, 0) = 0 \]

for \( k = 1, \ldots, K \)

for each \( v \in V \)

\[ \text{dist}(v, i) = \min_{u : (u, v) \in E} \{ \text{dist}(u, i-1) + l(u, v) \} \]

return \( \text{dist}(t, K) \)

Runtime: \( O(k \cdot n + km) \)

Shortest Path: set \( k = n-1 \)
All pairs shortest paths (APSP)

Input: $G=(V,E)$, lengths $l(u,v)$ for each edge

Output: $\forall u, v \in V$, $\text{dist}(u,v) =$ length of shortest $u \rightarrow v$ path

Idea 1: use Bellman-Ford for each $s \in V$

$O(n) \cdot O(n(n+m)) \approx O(n^2 m)$

$\approx O(n^3)$

Idea 2: use DP

we'll prove $O(n^3)$
1. Subproblems? Suppose $V = \{1, \ldots, n\}$

Any vertex $i \to j_1 \to j_2 \to \cdots \to j < \text{any}
intermediate vertices $j_1j_2\ldots \leq k < \text{any vertex}$

$\text{dist}(i, j, k) = \text{length of shortest } i \to j \text{ path where intermediate vertices are in } 1, \ldots, k$

$\text{dist}(i, j, 0) = l(i, j)$

2. Recurrence for $\text{dist}(i, j, k)$

$\text{dist}(i, j, k) = \min \left\{ \text{dist}(i, j, k-1), \text{dist}(i, k, k-1) + \text{dist}(k, j, k-1) \right\}$
Algorithm

For all \((i,j) \in E\)
\[
dist(i,j) = \ell(i,j) \quad (\infty \text{ if not edge})
\]

for \(k = 1 \to n\)
for \(i = 1 \to n\)
for \(j = 1 \to n\)
\[
dist(i,j, k) = \min \{ dist(i,k,k-1) + dist(k,j,k-1), dist(i,j,k-1) \}
\]

Runtime: \(O(n^3)\) time

Fun fact: runtime of best APSP alg: \(O\left(\frac{n^3}{\log^6(n)}\right)\)