6.006
Introduction to Algorithms

Lecture 21: Dynamic Programming IV
Prof. Erik Demaine
Today

• Piano fingering
• Platform video games
• Structural dynamic programming
• Vertex cover
• Widget layout
Recall: What is Dynamic Programming?

- "Controlled" brute force / exhaustive search

Key ideas:

- **Subproblems**: like original problem, but smaller
  - Write solution to one subproblem in terms of solutions to smaller subproblems - acyclic

- **Memoization**: remember the solution to subproblems we’ve already solved, and re-use
  - Avoid exponentials

- **Guessing**: if you don’t know something, guess it! (try all possibilities)
Recall:
How to Dynamic Program

Five easy steps!

1. Define subproblems
2. Guess something (part of solution)
3. Relate subproblem solutions (recurrence)
4. Recurse and memoize (top down) or Build DP table bottom up
5. Solve original problem via subproblems (usually easy)
Recall: How to Analyze Dynamic Programs

Five easy steps!

1. Define subproblems \(\text{count} \# \text{subproblems}\)
2. Guess something \(\text{count} \# \text{choices}\)
3. Relate subproblem solutions  
   \(\text{analyze time per subproblem}\)
4. \(\text{DP running time} = \# \text{subproblems} \cdot \text{time per subproblem}\)
5. Sometimes \(\text{additional running time}\) to solve original problem
Two Kinds of Guessing

1. Within a subproblem
   - Crazy Eights: previous card in trick
   - Sequence alignment: align/drop one character
   - Bellman-Ford: previous edge in path
   - Floyd-Warshall: use vertex $k$?
   - Parenthesization: last multiplication
   - Knapsack: include item $i$?
   - Tetris training: how to place piece $i$

2. Using additional subproblems
   - Knapsack: how much space left in knapsack
   - Tetris training: current board configuration
Piano Fingering
Piano Fingering

Piano Fingering
[Parncutt, Sloboda, Clarke, Raekallio, Desain 1997; Hart, Bosch, Tsai 2000; Al Kasimi, Nichols, Raphael 2007]

• Given musical piece to play
  – Say, sequence of single notes with right hand
  – (Can extend to both hands, multiple notes, etc.)

• Given metric \( d(f, p, g, q) \) of difficulty going from finger \( f \) on note \( p \) to finger \( g \) on note \( q \)
  – Crossing: High if \( 1 < f < g \) and \( p > q \)
  – Stretch: High if \( p \ll q \)
  – Legato: \( \infty \) if \( f = g \)
  – Weak finger: High if \( g \in \{4, 5\} \)
  – \( 3 \leftrightarrow 4 \): High if \( \{f, g\} = \{3, 4\} \)
  – ...
Piano Fingering DP

1. **Subproblems:** for $1 \leq i \leq n$: minimum difficulty possible for note[i: ]
2. **Guess:** finger $f$ for note[i]
3. **Recurrence:** $P(i) = \min(P(i + 1) + d(\text{note}[i], f, \text{note}[i + 1], \ldots ?? \ldots) \text{ for } f \text{ in fingers})$

- *How to know fingering for next note $i + 1$?*
- *Guess!*
Piano Fingering DP

1. **Subproblems:** for $1 \leq i \leq n$ & finger $f$:
   minimum difficulty possible for note[$i$:
   starting on finger $f$

2. **Guess:** finger $g$ for note[$i + 1$]

3. **Recurrence:** $P(i, f) = \min(P(i + 1, g) +
   d(\text{note}[i], f, \text{note}[i + 1], g)
   \text{for } g \text{ in fingers})$

4. **DP time** = # subproblems $\cdot$ time/subproblem
   \[\text{O}(F) = O(nF^2)\]

5. **Original problem** = $\min(P(1, f) \text{ for } f \text{ in fingers})$
Platform Video Games

• Given entire level: objects, enemies, etc.
• Anything outside $w \times h$ screen is reset
• *Configuration* = screen state, score, velocity, ...
• Given transition function for each time step
  $\delta: (\text{config}, \text{action}) \mapsto \text{config}'$
  – Movement, enemies, ...
• **Goal:** Maximize score subject to surviving and reaching goal
Platform Video Game DP

1. **Subproblems:** for configuration \( C \): best possible score starting from \( C \)

2. **Guess:** which action to take (if any)

3. **Recurrence:**
   \[
   P(C) = \max(P(\delta(C, A)) \text{ for } A \text{ in actions}) \]
   \[
   P(\text{goal } C) = C.\text{score}; \quad P(\text{dead } C) = -\infty
   \]

4. **DP time** = \( \# \) subproblems \( \cdot \) time/subproblem
   \[
   O(1)^{w.h} \cdot n \cdot m \cdot S \cdot V \cdot O(1) = O(1)^{(\text{pseudo polynomial})}
   \]

5. **Original problem** = \( P(\text{init}) \) for \( w.h = O(\lg(nmSV)) \)
Cycles in Subproblems

• $c_1 \rightarrow \delta(c_1, a_1) = c_2 \rightarrow \delta(c_2, a_2) = c_3 \rightarrow \ldots$ might lead to cycles

• In this problem, never helps to cycle
  – $c$ captures entire state, including score

• So mark subproblem at start, and if cycle, ignore that subproblem

• OR: SMB timer in $c$, so actually no cycles
Structural Dynamic Programming

• Follow a combinatorial structure other than a sequence / a few sequences
  – Like structural vs. regular induction

• **Main example:** Tree structure

• **Useful subproblems:** for every vertex \( v \), subtree rooted at \( v \)
Vertex Cover

• Given an undirected graph $G = (V, E)$
• Find a minimum-cardinality set $S$ of vertices containing at least one endpoint of every edge
  – Equivalently, find a minimum set of guards for a building of corridors, or (unaligned) streets in city

Example:
Vertex Cover Algorithms

• Extremely unlikely to have a polynomial-time algorithm, even for planar graphs (see Lecture 25)

• But polynomially solvable on trees, using dynamic programming
Vertex Cover in Tree DP

0. **Root the tree arbitrarily.**

1. **Subproblems:** for $v \in V$: size of smallest vertex cover in subtree rooted at $v$

2. **Guess:** is $v$ in the cover?
   - **YES:**
     - Cover children edges
     - Left with children subtrees
   - **NO:**
     - All children must be in cover
     - Left with grandchildren subtrees
Vertex Cover in Tree DP

1. **Subproblems:** for \( v \in V \): size of smallest vertex cover in subtree rooted at \( v \) \( \left\{ 1 | V | \right\} \)

2. **Guess:** is \( v \) in the cover? \( \left\{ 3 \text{ guesses} \right\} \)

3. **Recurrence:** \( V(v) = \min \{ \begin{array}{l} \text{YES:} \quad 1 + \sum V(c) \text{ for } c \text{ in } v \text{. children}, \\
\text{NO:} \quad \text{len}(v \text{. children}) + \\
\quad \quad \sum V(g) \text{ for } c \text{ in } v \text{. children for } g \text{ in } c \text{. children} \} \end{array} \} - O(v) \)

4. **DP time** = \# subproblems \( \cdot \) time/subproblem

5. **Original problem** = \( V(\text{root}) \) \( \text{actually } O(v) \text{ because each vertex visited twice: parent & grandpar} \)
Improved Vertex Cover in Tree DP

1. **Subproblems:** for \( v \in V \) & \( y \in \{\text{YES, NO, MAYBE}\} \): size of smallest vertex cover \( S \) in subtree rooted at \( v \) such that \([v \in S?] = y\) (unconstrained if \( y = \text{MAYBE} \))

2. **Guess:** Does \( \text{MAYBE} = \text{YES} \) or \( \text{NO} \)? \( \exists \leq 2 \) choices

3. **Recurrence:**
   \[
   V(v, \text{MAYBE}) = \min\{V(v, \text{YES}), V(v, \text{NO})\} - O(1)
   \]
   \[
   V(v, \text{YES}) = 1 + \sum (V(c, \text{MAYBE}) \text{ for } c \text{ in } v. \text{ children})
   \]
   \[
   V(v, \text{NO}) = \sum (V(c, \text{YES}) \text{ for } c \text{ in } v. \text{ children})
   \]

4. **DP time** = \( \# \text{ subproblems} \cdot \text{time/subproblem} \)
   \[
   \sum_{v \in V} 3 \cdot \# \text{children}(v) = O(V)
   \]

5. **Original problem** = \( V(\text{root, MAYBE}) \)
Widget Layout

• Given a hierarchy of **widgets**
• **Leaf** widget = button, image, ...
  – List of possible rectangular sizes
• **Internal** widget = rectangular container
  – Can join children horizontally or vertically
• **Goal:** Fit into a given rectangular screen

Widget Layout DP

1. **Subproblems:** for \( v \in V \& 0 \leq w \leq W \):
   minimum \( h \) such that widget \( v \) fits into \( w \times h \)

2. **Guess:** Leaf \( v \): Which size to use?
   Internal \( v \): Horizontal or vertical?

3. **Recurrence:**
   \[
   L(\text{leaf } v, w) = \min(h' \text{ for } (w', h') \text{ in } v. \text{ sizes if } w' \leq w)
   \]
   \[
   L(\text{internal } v, w) = \min \{ \sum(L(c, w) \text{ for } c \text{ in } v. \text{ child}),
   H(v, w, 1) \}\]
Horizontal Layout DP

1. **Subproblems:** for $v \in V, 0 \leq w \leq W$, $1 \leq i \leq \text{len}(v.\text{children})$: minimum $h$ such that horizontal layout of $v.\text{child}[i:]$ fits into $w \times h$ rectangle

2. **Guess:** Width $0 \leq w' \leq W$ of child $i$ \{W choices\}

3. **Recurrence:** $H(v, w, i) = \min(\max\{L(v.\text{child}[i], w'), H(v, w - w', i + 1)\}$ for $1 \leq w' \leq w)$ \{O(w)\}

4. **DP time** = $\# \text{subproblems} \cdot \text{time/subproblem}$
   \[= \frac{W \cdot |E| \cdot O(w) = O(w^2E)}{O(w)} = O(w^2E)\]
Widget Layout DP

1. **Subproblems**: for \( v \in V \) & \( 0 \leq w \leq W \): find minimum \( h \) such that widget \( v \) fits into \( w \times h \)

2. **Guess**: Leaf \( v \): Which size to use? \( \deg(v) \)  
   Internal \( v \): Horizontal or vertical? \( \Delta \)

3. **Recurrence**:
   \[
   L(\text{leaf } v, w) = \cdots \text{for } \cdots \text{in } v\text{'s sizes } \cdots \left\{ \begin{array}{l}
   O(\deg(v)) \end{array} \right.
   
   L(\text{internal } v, w) = \cdots \text{for } \cdots \text{in } v\text{'s child } \cdots
   
   \]

4. **DP time** = \( \# \) subproblems \( \cdot \) time/subproblem \( \sum_{v \in V} W \cdot O(\deg(v)) = O(WE) \)

5. **Original problem** = \( S(\text{root}, W) \leq H \)
Widget Layout Summary

• Two “levels” of dynamic programming
  1. Optimal height for given width of subtree rooted at \( v \)
  2. Optimal layout (partitioning) of children into horizontal arrangement

• Really just one bigger dynamic program

• Pseudopolynomial running time:
  \[ O(W^2E + WE) = O(W^2E) \]