Lecture 22: Dynamic Programming IV

Lecture Overview

• 2 kinds of guessing
• Piano/Guitar Fingering
• Tetris Training
• Super Mario Bros.

Review:

* 5 easy steps to dynamic programming

   (a) define subproblems                     count # subproblems
   (b) guess (part of solution)              count # choices
   (c) relate subproblem solutions           compute time/subproblem
   (d) recurse + memoize                     time = time/subproblem · # subproblems
      OR build DP table bottom-up
      check subproblems acyclic/topological order
   (e) solve original problem: = a subproblem
      OR by combining subproblem solutions   ⇒ extra time

* 2 kinds of guessing:

   (A) In (3), guess which other subproblems to use (used by every DP except Fibonacci)
   (B) In (1), create more subproblems to guess/remember more structure of solution
        used by knapsack DP
        • effectively report many solutions to subproblem.
        • lets parent subproblem know features of solution.
Piano/Guitar Fingering:

Piano

[Parncutt, Sloboda, Clarke, Raekallio, Desain, 1997]
[Hart, Bosch, Tsai 2000]
[Al Kasimi, Nichols, Raphael 2007] etc.

- given musical piece to play, say sequence of \( n \) (single) notes with right hand
- fingers 1, 2, \ldots, \( F = 5 \) for humans
- metric \( d(f, p, g, q) \) of difficulty going from note \( p \) with finger \( f \) to note \( q \) with finger \( g \)

  e.g., \( 1 < f < g \) \& \( p > q \) \implies \text{uncomfortable}

  stretch rule: \( p \ll q \) \implies \text{uncomfortable}

  legato (smooth) \implies \infty \text{ if } f = g

  weak-finger rule: prefer to avoid \( g \in \{4, 5\} \)

  \( 3 \rightarrow 4 \) \& \( 4 \rightarrow 3 \) annoying \sim \text{ etc.}

First Attempt:

1. **subproblem** = \(\min\) difficulty for suffix notes \([i :]\)
2. **guessing** = finger \( f \) for first note \([i]\)
3. **recurrence**:
   
   \[ DP[i] = \min(DP[i+1]+d(note[i], f, note[i+1], g) \text{ for } g \in \text{range}(F)) \]

   \( \rightarrow \) not enough information!

Correct DP:

1. **subproblem** = \(\min\) difficulty for suffix notes \([i :]\) given finger \( f \) on first note \([i]\)
\(\implies n \cdot F \) subproblems
2. **guessing** = finger \( g \) for next note \([i+1]\)
\(\implies F \) choices
3. **recurrence**:
   
   \[ DP[i, f] = \min(DP[i+1, g]+d(note[i], f, note[i+1], g) \text{ for } g \in \text{range}(F)) \]

   \[ DP[n, f] = 0 \]

   \(\implies \Theta(F) \) time/subproblem
4. topo. order: for $i$ in reversed(range($n$)):
   for $f$ in 1, 2, ..., $F$:
   total time $O(nF^2)$

5. orig. prob. = min(DP[0, $f$] for $f$ in 1, ..., $F$)
   (guessing very first finger)

Figure 1: DAG.
Guitar

Up to $S$ ways to play same note! (where $S$ is # strings)

- redefine “finger” = finger playing note + string playing note
- $F \to F \cdot S$

Generalization:

Multiple notes at once e.g. chords

- input: notes[i] = list of $\leq F$ notes
  (can’t play $> 1$ note with a finger)

- state we need to know about “past” now assignment of $F$ fingers to $\leq F + 1$
  notes/null
  $\implies (F + 1)^F$ such mappings

(1) $n \cdot (F + 1)^F$ subproblems where $(F + 1)^F$ is how notes[i] is played
(2) $(F + 1)^F$ choices (how notes[i + 1] played)
(3) $n \cdot (F + 1)^{2F}$ total time

- works for 2 hands $F = 10$
- just need to define appropriate $d$

Tetris Training:

![Tetris pieces](image)

Figure 2: Tetris.

- given sequence of $n$ Tetris pieces & an empty board of small width $w$
- must choose orientation & $x$ coordinate for each
- then must drop piece till it hits something
• full rows do not clear
  without the above two artificialities WE DON’T KNOW!
  (but: if nonempty board & \( w \) large then NP-complete)

• goal: survive i.e., stay within height \( h \)

**First Attempt:**

1. subproblem = survive in suffix \( i \)? WRONG

2. guessing = how to drop piece \( i \) \( \implies \) # choices = \( O(w) \)

3. recurrence: \( DP[i] = DP[i+1] \) ?! not enough information!
   What do we need to know about prefix : \( i \)?

**Correct:**

• 1. subproblem = survive in suffix \( i \):
  given initial column occupancies \( h_0, h_1, \ldots, h_{w-1} \), call it \( h \)
  \( \implies \) # subproblems = \( O(n \cdot h^w) \)

• 3. recurrence: \( DP[i, h] = \max(DP[i, m] \text{ for valid moves } m \text{ of piece } i \text{ in } h) \)
  \( \implies \) time per subproblem = \( O(w) \)

• 4. topo. order: for \( i \) in reversed(range(n)): for \( h \)
  total time = \( \tilde{O}(nwh^w) \) (DAG as above)

• 5. solution = \( DP[0, 0] \)
  (& use parent pointers to recover moves)

**Super Mario Bros**

Platform Video Game

• given entire level (objects, enemies, . . . ) (\( \leftarrow n \))

• small \( w \times h \) screen

• configuration
  – screen shift (\( \leftarrow n \))
  – player position & velocity (\( O(1) \)) (\( \leftarrow w \))
- object states, monster positions, etc. ($\leftarrow c^{w \cdot h}$)
- anything outside screen gets reset ($\leftarrow c^{w \cdot h}$)
- score ($\leftarrow S$)
- time ($\leftarrow T$)

- transition function $\delta$: (config, action) $\rightarrow$ config'
  nothing, ↑, ↓, ←, →, B, A press/release

(1) subproblem: best score (or time) from config. $C$
  $\Rightarrow n \cdot c^{w \cdot h} \cdot S \cdot T$ subproblems

(2) guess: next action to take from $C$
  $\Rightarrow O(1)$ choices

(3) recurrence:

$$DP(C) = \begin{cases} 
C.\text{score} & \text{if on flag} \\
\infty & \text{if } C.\text{dead} \text{ or } C.\text{time} = 0 \\
\max(DP(\delta(C, A))) & \text{for } A \text{ in actions}
\end{cases}$$

$\Rightarrow O(1)$ time/subproblem

(4) topo. order: increasing time

(5) orig. prob.: DP(start config.)

- pseudopolynomial in $S$ & $T$
- polynomial in $n$
- exponential in $w \cdot h$