Today: Dynamic Programming IV (of 4)
- 2 kinds of guessing
- piano/guitar fingering
- Tetris training
- Super Mario Bros.

* 5 easy steps to dynamic programming:
  1. define subproblems
  2. guess (part of solution)
  3. relate subprob. solutions
  4. recurse & memoize
  or build DP table bottom-up
  - check subprobs. acyclic/topological order
  5. solve original problem: = a subproblem
  or by combining subprob. 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 subprob.
    - lets parent subproblem know features of sol.
Piano/guitar fingering:

- 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. $1 < f < g \& p > q \Rightarrow$ uncomfortable
  stretch rule: $p \ll q \Rightarrow$ uncomfortable
  legato (smooth) \Rightarrow \infty \text{ if } f = g
  weak-finger rule: prefer to avoid $g \in \{4, 5\}$
  $3 \rightarrow 4 \& 4 \rightarrow 3$ annoying \text{ etc.}

First attempt:

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

not enough information!
Correct DP:

1. subproblem $= \min$ difficulty for suffix notes $i:]$
given finger $f$ on first note $i$
$
\Rightarrow n \cdot F$ subproblems

2. guessing $= \text{finger } g$ for next note $i+1$
$
\Rightarrow F$ choices

3. recurrence:
$DP[i, f] = \min(DP[i+1, g] + d(\text{note}[i], f, \text{note}[i+1], g)$
for $g$ in range($F$)

$DP[n, f] = \emptyset$
$
\Rightarrow \Theta(F)$ time/subproblem

4. topo. order: for $i$ in reversed(range($n$)):
   for $f$ in $1, 2, \ldots, F$:
   - total time: $\Theta(nF^3)$

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

DAG:
Guitar: up to $S$ ways to play same note!
- redefine "finger" = finger playing note + string playing note
- $F \rightarrow F \cdot S$

Generalization: multiple notes at once
- input: $\text{notes}[i] = \text{list of } \leq F \text{ notes}$
  (can't play >1 note with a finger)
- state we need to know about "past"
  now assignment of fingers to notes/null
- $\sum_{F}^{\leq F+1}$

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

- works for 2 hands ($F=10$)
- just need to define appropriate $d$
Tetris training:
- 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
  
  (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 $i$
   \[ \Rightarrow \# \text{choices} = O(w) \]
3. recurrence: $DP[i] = DP[i+1]$?! not enough information!
   \[ \Rightarrow \text{What do we need to know about prefix } : i ? \]

Correct:
1. subproblem = survive? in suffix $i$:
   \[ \text{given initial column occupancies } h_0, h_1, \ldots, h_{w-1} \]
   \[ \Rightarrow \# \text{subproblems} = O(n \cdot h^w) \]
2. recurrence: $DP[i, h^i] = \max (DP[i, m]$
   \[ \text{for valid moves } m \text{ of piece } i \text{ in } h^i) \]
   \[ \Rightarrow \text{time per subproblem} = O(w) \]
3. topo. order: for $i$ in reversed(range($n$)): for $h^i$ ...
   \[ \text{total time} = O(n \cdot w \cdot h^w) \] (DAG as above)
4. solution = $DP[\emptyset, \emptyset]$
   (& use parent pointers to recover moves)
- Super Mario Bros/platform video game (objects, enemies, ...)
  - given entire level
  - small w x h screen
  - configuration:
    - screen shift
    - player position & velocity
    - object states, monster positions, etc.
    - anything outside screen gets reset
  - score
  - time
  - transition function S: (config, action) \rightarrow config'
    - nothing, ↑↓←→ press/release
  - subproblem = best score (or time) from config. C
    \Rightarrow n \cdot c^{w \cdot h} \cdot S \cdot T subproblems
  - guess: next action to take from C
    \Rightarrow O(1) choices
  - recurrence: \[ \text{DP}(C) = \begin{cases} 
    C\text{.score} & \text{if on flag} \\
    \infty & \text{if } C\text{.dead or } C\text{.time} = \emptyset \\
    \max (\text{DP}(S(C,A)) & \text{ for } A \text{ in actions}) 
  \end{cases} \]
    \Rightarrow O(1) time/subproblem
  - topo. order: increasing time
  - orig. prob. = DP(start config.)
- pseudopolynomial in S & T
- polynomial in n
- exponential in w x h