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:

\begin{itemize}
\item Given musical piece to play, say sequence of \( n \) (single) notes with right hand
\item Fingers 1, 2, ..., \( F = 5 \) for humans
\item Metric \( d(f, p, g, q) \) of difficulty going from note \( p \) with finger \( f \) to note \( q \) with finger \( g \)
\item E.g., \( 1 < f < g \) & \( p > q \) \( \Rightarrow \) uncomfortable stretch rule: \( p << q \Rightarrow \) uncomfortable legato (smooth) \( \Rightarrow \) \( \infty \) if \( f = g \)
\item Weak-finger rule: prefer to avoid \( g \in \{4, 5\} \)
\item 3→4 & 4→3 annoying \( \sim \) etc.
\end{itemize}

First attempt:

\begin{enumerate}
\item Subproblem = min difficulty for suffix notes \([i:]\)
\item Guessing = finger \( f \) for first note \([i]\)
\item Recurrence:
\[ DP[i] = \min(DP[i+1] + d(note[i], f, note[i+1], ?)) \text{ for } f \] not enough information!
\end{enumerate}
Correct DP:

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

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

3. **recurrence**:
   \[ DP[i,f] = \min(DP[i+1,g] + d(note[i], f, note[i+1], g)) \text{ for } g \text{ in } \text{range}(F) \]
   \[ DP[n,f] = \emptyset \]
   \[ \Rightarrow \Theta(F) \text{ time/subproblem} \]

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

5. **orig. prob.** = \( \min(DP[\emptyset,f] \text{ for } f \text{ in } 1,...,F) \) (guessing very first finger)

DAG:

- notes
- fingers

Difficulty
Guitar: up to $S$ ways to play same note!
- redefine "finger" = finger playing note + string playing note

$\Rightarrow F \rightarrow F \cdot S$

**Generalization:** multiple notes at once
- 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 fingers to notes/null

$\Rightarrow (F+1)^F$ such mappings

1. $n \cdot (F+1)^F$ subproblems
2. $(F+1)^F$ choices (how 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

(without these 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 \( i \): \( \subseteq \) WRONG
2. guessing = how to drop piece \( i \)
   \( \Rightarrow \) # choices = \( O(w) \)
3. recurrence: \( DP[i] = DP[i+1] \)?! not enough information!
   \( \Rightarrow \) What do we need to know about prefix \( : i \)?

Correct:
1. subproblem = survive? in suffix \( i \):
   - given initial column occupancies \( h_0, h_1, \ldots, h_{w-1} \)
   \( \Rightarrow \) # subproblems = \( O(n \cdot h^w) \)
2. recurrence: \( DP[i, h^i] = \max (DP[i, m]) \)
   for valid moves \( m \) of piece \( i \) in \( h^i \)
\( \Rightarrow \) time per subproblem = \( O(w) \)
3. topo. order: for \( i \) in reversed(range(n)): for \( h^i \)
   \( \Rightarrow \) total time = \( O(n w h^w) \) (DAG as above)
4. solution = \( DP[0, \hat{h}] \)
   (& use parent pointers to recover moves)
Super Mario Bros/platform video game
- given entire level
- small \( n \times w \times h \) screen
- configuration:
  - screen shift
  - player position & velocity
  - object states, monster positions, etc.
  - anything outside screen gets reset
- score
- time

- transition function \( S: (\text{config}, \text{action}) \rightarrow \text{config} \)
  - nothing, \( \uparrow\downarrow\leftarrow\rightarrow\text{press/release} \)

1. \( \text{subproblem} = \text{best score (or time) from config. } C \Rightarrow n \cdot c^{w \times h} \cdot S \cdot T \text{ subproblems} \)
2. \( \text{guess: next action to take from } C \Rightarrow O(1) \text{ choices} \)
3. \( \text{recurrence: } \text{DP}(C) = \begin{cases} C.\text{score} & \text{if on flag} \\ \infty & \text{if } C.\text{dead or } C.\text{time} = \emptyset \\ \max(\text{DP}(S(C,A))) & \text{for } A \text{ in actions} \end{cases} \Rightarrow O(1) \text{ time/subproblem} \)
4. \( \text{topo. order: increasing time} \)
5. \( \text{orig. prob. = DP(start config.)} \)

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