6.006	Lecture 22	Dec. 6, 2011

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 — count # subprobs.
2. guess (part of solution) — count # choices
3. relate subprob. solutions — compute time/subprob.
4. recurse + memoize — time = time/subprob. · # subprobs.

   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:

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 $1, 2, \dots, F$ $= 5$ 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) $\Rightarrow \infty$ if $f = g$
  weak-finger rule: prefer to avoid $g \in \{4, 5\}$
  $3 \to 4$ & $4 \to 3$ annoying ~ etc.

First attempt:

1. ~~subproblem = min. difficulty for suffix notes[i:]~~
2. ~~guessing = finger f for first note[i]~~
3. recurrence:

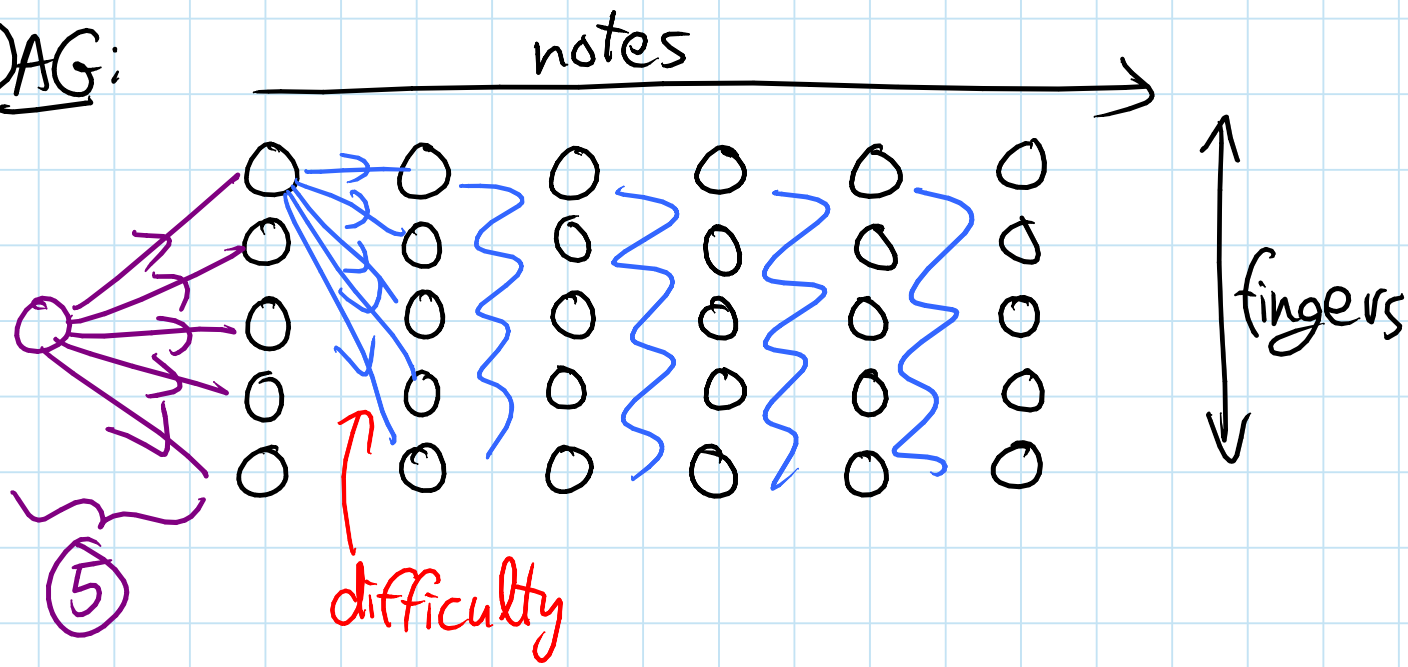
   ~~DP[i] = min(DP[i+1] + d(note[i], f, note[i+1], ?) for f...)~~

   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 = finger g for next note[i+1]
   $\Rightarrow F$ choices
3. recurrence:
   $DP[i,f] = \min(DP[i+1,g] + d(note[i], f, note[i+1], g)$ 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: $\Theta(nF^2)$
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! (S → # strings)
- redefine "finger" = finger playing note + string playing note

$\Rightarrow F \to F \cdot S$

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 $\underbrace{\text{fingers}}_{F}$ to $\underbrace{\text{notes/null}}_{\leq F+1}$

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

1. $n \cdot \underbrace{(F+1)^F}_{\text{how notes[i] is played}}$ subproblems
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: 

- 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: ?~~ 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 $= O(n \cdot h^w)$

③ recurrence: $DP[i, \vec{h}] = \max(DP[i, \vec{m}]$ for valid moves $\vec{m}$ of piece i in $\vec{h})$

$\Rightarrow$ time per subproblem $= O(w)$

④ topo. order: for i in reversed(range(n)): for $\vec{h}$ ...

total time $= O(n\, w\, h^w)$ (DAG as above)

⑤ solution $= DP[\emptyset, \vec{\emptyset}]$

(& use parent pointers to recover moves)

Super Mario Bros / platform video game

n — given entire level (objects, enemies, ...)
— small $w \times h$ screen
— configuration:
  - n { — screen shift
  - $w \cdot$ { — player position & velocity $\nearrow O(1)$
  - $c^{w \cdot h}$ { — object states, monster positions, etc.
    — anything outside screen gets reset
  - S { — score
  - T { — time
— transition function δ: (config, action) $\mapsto$ config'
  nothing, $\boxed{\uparrow}$, $\boxed{\downarrow}$, $\boxed{\leftarrow}$, $\boxed{\rightarrow}$, Ⓑ, Ⓐ 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:
$$DP(C) = \begin{cases} C.\text{score} & \text{if on flag} \\ \infty & \text{if } C.\text{dead or } C.\text{time} = \emptyset \\ \max(DP(\delta(C, A)) \text{ for } A \text{ in actions}) \end{cases}$$
  $\Rightarrow O(1)$ time/subproblem
(4) topo. order: increasing time
(5) orig. prob. = DP(start config.)

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