

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 (\Rightarrow extra time)

count # subprobs.

count # choices

compute time/subprob.

time = time/subprob.

• # subprobs.

* 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]

- given musical piece to play, say sequence of n (single) notes with right hand
- fingers 1, 2, ..., 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: $p << q \Rightarrow$ uncomfortable legato (smooth) $\Rightarrow \infty$ if $f=g$
 - weak-finger rule: prefer to avoid $g \in \{4, 5\}$
 - $3 \rightarrow 4 \& 4 \rightarrow 3$ annoying ~ etc.

First attempt:

① subproblem = min. difficulty for suffix notes[i:]

② guessing = finger f for first note[i]

③ recurrence:

$DP[i] = \min(DP[i+1] + d(\text{note}[i], f, \text{note}[i+1], ?)) \text{ for } f \dots$

not enough information! \dagger

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:

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

$$DP[n, f] = \emptyset$$

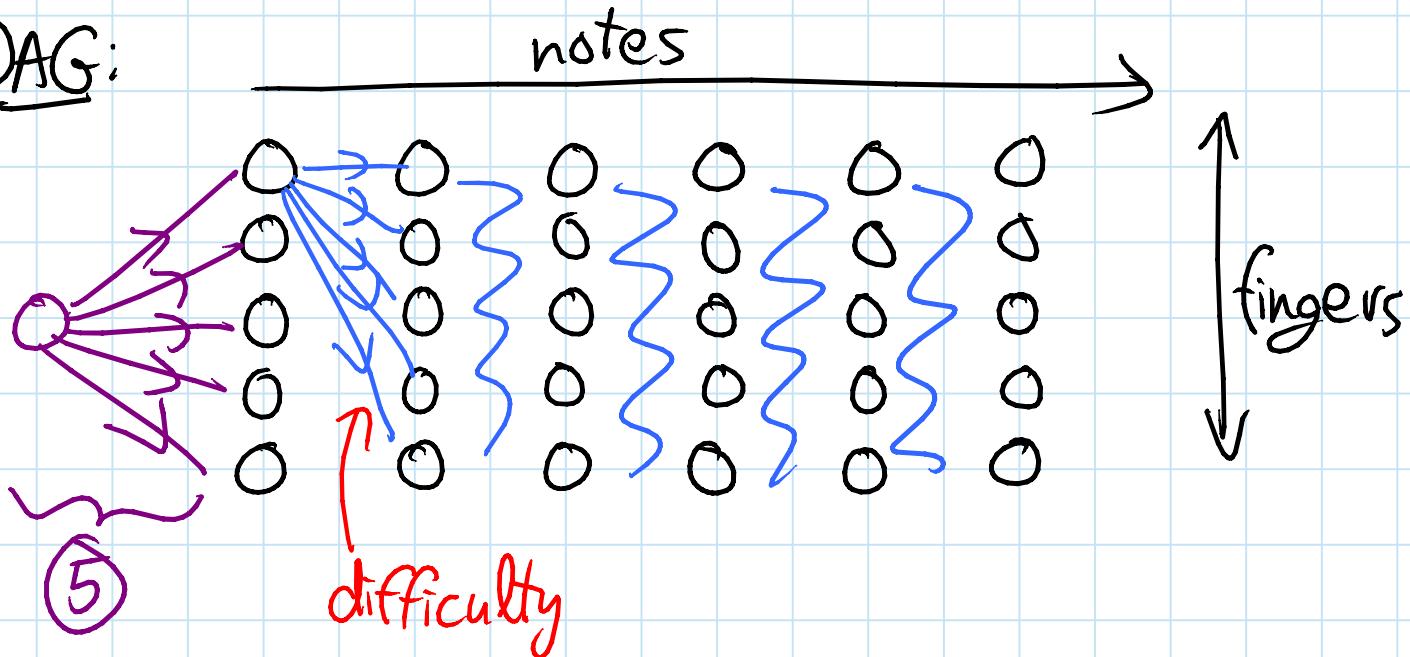
$\Rightarrow \Theta(F)$ time/subproblem

④ topo. order: for i in reversed(range(n)): for f in $1, 2, \dots, F$:

- total time: $\Theta(nF^2)$

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

DAG:



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
 (e.g. chords)

- input: $\text{notes}[i]$ = 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 $\frac{\text{notes}}{F} / \text{null}$

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

② $(F+1)^F$ choices (how $\text{notes}[i+1]$ 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 $\xrightarrow{\text{do not clear}}$
 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
 $\Rightarrow \# \text{ choices} = O(w)$
- 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, \dots, h_{w-1}
 $\Rightarrow \# \text{ subproblems} = O(n \cdot h^w)$
- 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 $\text{reversed}(\text{range}(n))$: for $\vec{h} \dots$
 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

- given entire level (objects, enemies, ...)
- Small $w \times h$ screen
- configuration:

- n { - screen shift $\rightarrow O(1)$
- w { - player position & velocity
- $w \cdot h$ { - object states, monster positions, etc.
- C { - anything outside screen gets reset
- S { - score
- T { - time

- transition function $S: (\text{config}, \text{action}) \mapsto \text{config}'$

nothing, $\uparrow, \downarrow, \leftarrow, \rightarrow, B, A$ 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: $DP(C) = \begin{cases} C.\text{score} & \text{if on flag} \\ \infty & \text{if } C.\text{dead or } C.\text{time} = 0 \\ \max(DP(S(C, A))) & \text{for } A \text{ in actions} \end{cases}$
 $\Rightarrow O(1)$ time/subproblem

④ topo. order: increasing time
 ⑤ orig. prob. = $DP(\text{start config.})$

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