Dynamic Programming IV [MIT OpenCourseware 6.006]

- 5 easy steps to DP:
  1. define subproblems
     # subproblems ?
  2. Guess / Try part of solution
     # guesses ?
  3. Recurrence
  4. Recurse + Memoize
  5. Solve original problem

- **Question:** What types of guessing have we seen?
  - Prefix
  - Suffix
  - Substring

- Today we will see an additional type;
- In step 2 we are guessing which sub-problem to use in order to solve the original problem

- But there is a higher-level:

  In step 1 we can add more subproblems to guess / remember more

  This happened in the knapsack problem:

  - The obvious solution was to use suffixes;
  - But using suffixes alone was inefficient to solve the problem
  - We also need the ability to remember how much capacity remained.

- Lets look at some examples

Piano / Guitar fingering -

- Given sequence of n notes, find fingering for each note
- By fingering we mean: which fingers should be used to play each note since some transitions are easier than others.

- For humans we have 1, 2, 3, ..., 10
- Generalizing, fingers : 1, ..., F
- To keep it simple assume:
  - Only a single note on the piano
  - Only the right-hand is used
- **Objective:**
  Assign one of each finger to each node
- But we also need to express the difficulty of being on a certain note that is being played with a certain finger and we want to transition another note using another finger.
- **Question:**
  How can we express this difficulty?
- We can define a function:

$d(p, f, q, g)$:

- p - first note
- f - finger that is playing note p
- q - note that we want to transition
- g - finger that will play note q

- There is a huge literature for this on piano!, e.g.:
  - If $p \leq < q$ then you need to stretch your fingers a lot, which is hard!
  - If you are playing legato the player transitions from one note to another note with no intervening silence:
    - You cannot use the same finger for the same note
    - This implies that if $f == g$ then $p = q$
- Weak finger rule: avoid the two rightmost fingers (fingers 4 & 5) of right hand
- Many more other examples, including for guitar
- These transitions can be encoded into a table that represents function d
- Lets try to solve this using DP:
  - We have a sequence of notes and we want to find the sequence of fingers

Question:
- If we have a sequence of something what can we try?
  - Always the same answer: { prefixes, suffixes, substrings

Question:
which one do you think it will be?

- Intuitively: We want to process the sequence of notes from left to right ⇒ suffices

Notes: This initial formulation is wrong! It is used to exemplify a problem

① Subproblem = how to play notes_[i:]

② guess = which finger to use for note i

③ recurrence: (we want to minimize difficulty)

$$DP(i) = \min\Big(\underbrace{DP[i+1]}_{\text{play remaining notes}} + d(i, f, i+1, ?) \Big)$$

$$\underbrace{\text{for } f \text{ in } 1, \ldots, F}_{\text{for all the fingers}} \Big)$$

We are playing note i with finger f then we have to go to note i+1 but then the problem is we have no idea what finger we are going to guess for not i+1

∴ We cannot solve the problem this way!!!

↳ But it was the first thing we should try

- We need to add more subproblems...

Question: Any guesses to what we can do for subproblems?

① Subproblems = how to play notes[i:] when using finger f for notes[i]

#notes = n
#fingers = F
⇒ #subproblems = n · F

② Try/Guess = what is the finger g for notes[i+1]

#guesses = "for each note we need to try all fingers" = F

③ Recurrence

$$DP(i,f) = \min\left(DP(i+1, g) + d(i, f, i+1, g) \text{ for all } g \text{ in } 1, \dots, F \right)$$

i := which am note am I in
f := what is the finger for note i

notes[i] notes[i+1]

④ (Always the same thing)

⑤ Original problem:

- We cannot just start in any finger;
- We need to select the finger f that minimizes difficulty
- I.e.:

$$\min\left(DP(0, f) \text{ for } f \text{ in } 1, \dots, F \right)$$

Question: What is the running time?

- #subproblems = n F
- # guesses = F (time / subproblem)
- Determining initial finger = F
- Total time: n . F . F . F = nF^3
  - Given that F is usually pretty small this is approximately linear time, i.e. $\Theta(n)$
- Lets look at another subproblem

Tetris training — (unconstrained / constrained version)

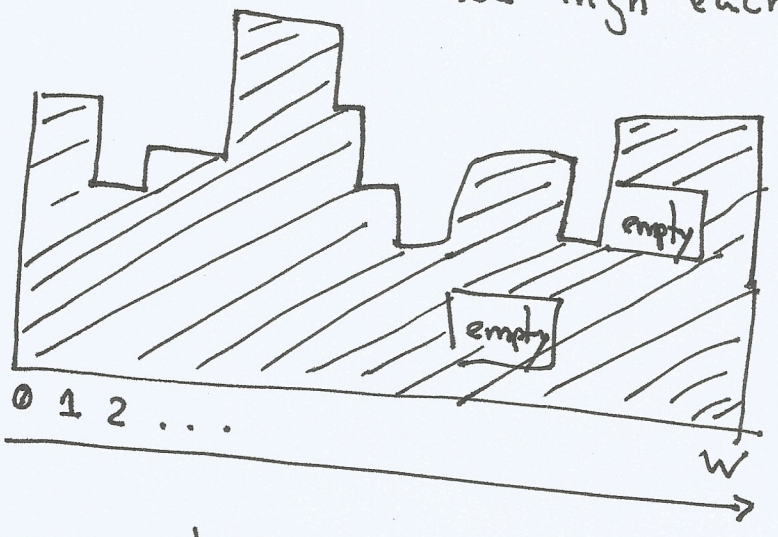
Assume:

- Given sequence of n pieces
- For each of them we must drop the piece from the top
- No rotations are allowed whilst the piece is falling
- Full rows normally clear, but in our version they will not clear → hardcore tetris ;)
- The width of the board w is small
- The board is initially empty

- We need all these assumptions to be able to use dynamic programming

① Subproblems:

- Obvious thing to try is suffixes;
- How to play suffix pieces[i:] (initial subproblem formulation)
- Just like the fingering problem this is not enough information
- We need to know what the board looks like
- All we care about is how high each column is:



This is called the skyline

- The new subproblem formulation:

How to play suffix pieces[i:] given board skyline

- Question: How many possible configurations do we have for the skyline?
  - height of the board := h (from 0 to h)
  - Therefore we have: (h+1) combinations/spaces for 1st column
    (h+1) " " " 2nd "
    ⋮
    (h+1) " " " " wth column

∴ We have $\underbrace{(h+1)\times(h+1)\times\cdots\times(h+1)}_{w \text{ times}} = (h+1)^w$ combinations