Today: Dynamic Programming II (of 4)
- 5 easy steps
- text justification
- perfect-information Blackjack \{ examples
- parent pointers

Summary:
* \( \text{DP} \approx \) “careful brute force”
* \( \text{DP} \approx \) guessing + recursion + memoization
* \( \text{DP} \approx \) dividing into reasonable # subproblems whose solutions relate — acyclically — usually via guessing parts of solution

- time = # subproblems \cdot \text{time/subproblem} treating recursive calls as \( O(1) \)
  = # subproblems \cdot # guess choices \cdot \text{time/guess}

- essentially an amortization
- count each subproblem only once; after first time, costs \( O(1) \) via memoization

* DP often \approx\ shortest paths in some DAG
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
5. check subprobs: acyclic/topological order

or by combining subprob. solutions (⇒ extra time)

Examples:

- Fibonacci
  
  1. subprobs: \( F_k \) for \( 1 \leq k \leq n \)
  
  2. guess: nothing
  
  3. recurrence: \( F_k = F_{k-1} + F_{k-2} \)
  
  4. topo. order: for \( k = 1, \ldots, n \)

  total time: \( O(n) \)

  5. orig. prob.: \( F_n \)

  extra time: \( O(1) \)

- Shortest Paths

  1. subprobs: \( S_k(s,v) \) for \( v \in V \), \( 0 \leq k \leq |V| \)

  2. guess: nothing

  3. recurrence: \( S_k(s,v) = \min \{ S_{k-1}(s,w) + w(u,v) \} \)

  4. topo. order: for \( k = 0, 1, \ldots, |V|-1 \)

  total time: \( O(V) \)

  5. orig. prob.: \( S_{|V|-1}(s,v) \) for \( v \in V \)

  extra time: \( O(V) \) (Single Source)
Text justification: split text into "good" lines
- obvious (MS Word/OpenOffice) algorithm: put as many words fit on first line, repeat
- but this can make very bad lines:
  \[
  \text{blah blah blah} \quad \text{vs.} \quad \text{blah blah really long word}
  \]
- define \( \text{badness}(i,j) \) for line of words \([i:j]\)
  - e.g. \( \{ \infty \) if total length \( > \) page width
  - \( (\text{page width} - \text{total length})^3 \) else
- goal: split words into lines to min. \( \sum \text{badness} \)

1. subproblem = \( \min \text{badness} \) for suffix words \([i:]\)
   \( \Rightarrow \) # subproblems = \( \Theta(n) \) where \( n = \# \text{words} \)
2. guessing = where to end first line, say \( i:j \)
   \( \Rightarrow \) # choices = \( n-i = O(n) \)
3. recurrence:
   - \( \text{DP}[i] = \min (\text{badness}(i,j) + \text{DP}[j]) \) for \( j \) in range \((i+1, n+1)\)
   - \( \text{DP}[n] = \emptyset \)
   \( \Rightarrow \) time per subproblem = \( \Theta(n) \)
4. order: for \( i = n, n-1, \ldots, 1, \emptyset \)
   total time = \( \Theta(n^2) \)
5. solution = \( \text{DP}[\emptyset] \)

\[\text{DAG:} \quad i \rightarrow O \quad O \quad O \quad O \quad O \quad j \]
\[\text{badness}(i,j)\]
Parent pointers:
to recover actual solution in addition to cost, store parent pointers (which guess used at each subproblem) & walk back
  - typically: remember argmin/argmax in addition to min/max
  
  - e.g. text justification:

  3) \( \text{DP}[i] = \min((\text{badness}(i,j) + \text{DP}[j][\emptyset], j)) \) for \( j \in \text{range}(i+1, n+1) \)

\( \text{DP}[n] = (\emptyset, \text{None}) \)

5) \( i = \emptyset \)
while \( i \) is not None:
  start line before word \( i \)
  \( i = \text{DP}[i][1] \)

- just like memoization & bottom-up, this transformation is automatic (no thinking required)
Perfect-information Blackjack:
- given entire deck order: \(c_0, c_1, \ldots, c_{n-1}\)
- 1-player game against stand-on-17 dealer
- when should you hit or stand? GUESS
- goal: maximize winnings for fixed bet $1
- may benefit from losing one hand to improve future hands!

1. **subproblems**: \(BJ(i) = \text{best play of } c_i, \ldots, c_{n-1}\) up to the remaining cards

\[ \Rightarrow \# \text{ subproblems} = n \]

2. **guess**: how many times player "hits"

\[ \Rightarrow \# \text{ choices} \leq n \]

3. **recurrence**: \(BJ(i) = \max (\)

\[ O(n) \Rightarrow \text{outcome } \in \{+1, 0, -1\} + BJ(i + \# \text{ cards used}) \]

\[ O(n) \Rightarrow \text{for \# hits in } 0, 1, \ldots \]

\[ \text{if valid play \sim \text{don't hit after bust})} \]

\[ \Rightarrow \text{time/subproblem} = \Theta(n^2) \]

4. **order**: for \(i\) in reversed(range(n))

\[ \text{total time} = \Theta(n^3) \]

5. **solution** = \(BJ(0) \)

\[ \Rightarrow \text{if we treat 21 as constant: \# hits} = \Theta(1) \]

\[ \Rightarrow \Theta(n) \text{ time overall} \]
Detailed recurrence: (before memoization)
(ignoring splits/betting & blackjacks)

BJ(i):
- if \( n - i < 4 \): return \( \emptyset \) (not enough cards)
- options = []
- for \( p \) in range(2, \( n - i - 1 \)): (# cards taken)
- \( \Theta(n^2) \) \{
  - \( \Theta(n) \) \{
    - player = \( \sum(c_i \cdot c_{i+2} \cdot c_{i+4} \cdot i + p + 2) \)
    - treat Aces as 11; while sum > 21: 11 \( \rightarrow 1 \) if exists
    - if player > 21: (bust)
      - options.append(-1 + BJ(i+p+2))
      - break (bust)
    - for \( d \) in range(2, \( n - i - p \)):
      - dealer = \( \sum(c_{i+1} \cdot c_{i+3} \cdot c_{i+p+2} \cdot i + p + d) \)
      - if dealer \( \geq 17 \): break
      - if dealer > 21: dealer = \( \emptyset \) (bust)
      - options.append(cmp(player, dealer) + BJ(i+p+d))
  - return \( \max(options) \)
- \( \Theta(n) \) with care

- should generalize to complex betting etc.

DAG view:

Valid plays → -1 → outcomes +1