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

Summary:
* DP ≈ "careful brute force"
* DP ≈ guessing + recursion + memoization
* DP ≈ dividing into reasonable # subproblems whose solutions relate — acyclically — usually via guessing parts of solution

* time = # subproblems • time/subproblem treating recursive calls as O(1) (usually mainly guessing)
  - essentially an amortization  
  - count each subproblem only once; after first time, costs O(1) via memoization

* DP ≈ shortest paths in some DAG
5 easy steps to dynamic programming:

1. **Define subproblems**
   - Count # subproblems

2. **Guess (part of solution)**
   - Count # choices

3. **Relate subprob. solutions**
   - Compute time/subprob.

4. **Recurse + memoize or build DP table bottom-up**
   - Time = time/subprob.

5. **Check subprob. acyclic/topological order**
   - # subproblems.

- Solve original problem: = a subproblem 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 \( k = 1, \ldots, n \)
  5. **Total time:** \( \Theta(n) \)
  6. **Orig. prob.:** \( F_n \)
  7. **Extra time:** \( \Theta(1) \)

- **Shortest Paths**
  1. **Subprobs:** \( S_k(s,v) \) for \( v \in V \), \( 0 \leq k < |V| \)
  2. **Guess:** nothing
  3. **Recurrence:** \( S_k(s,v) = \min \{ S_{k-1}(s,w) + w(u,v) \mid (u,v) \in E \} \)
  4. **Total time:** \( \Theta(1 + \text{indegree}(V)) \) for \( k = 0, 1, \ldots, |V|-1 \)
  5. **Orig. prob.:** \( S_{|V|-1}(s,v) \) for \( v \in V \)
  6. **Extra time:** \( \Theta(1) \)
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:
  
  | really long word  | vs. | really long word  |
  | b l a h b l a h | ° | b l a h b l a h | ° |

- define $\text{badness}(i,j)$ for line of words $[i:j]$
  e.g. $\{\infty$ if total length $> \text{page width}$
  $\{ (\text{page width} - \text{total length})^3 \}$ 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 $= \text{where to end first line, say } i:j$
   $\Rightarrow$ # choices $= n-i = O(n)$
3. recurrence:
   - $DP[i] = \min (\text{badness}(i,j) + DP[j] \text{ for } j \text{ in range } \{i+1, n+1\})$
   - $DP[n] = \emptyset$
   $\Rightarrow$ time per subproblem $= \Theta(n)$
4. order: for $i = n, n-1, \ldots, 1, \emptyset$
   total time $= \Theta(n^2)$
5. solution $= DP[\emptyset]$

\[ \text{DAG:} \quad \frac{i \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow j}{\text{badness}(i,j)} \]
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: \( BJ(i) = \text{best play of } c_i, \ldots, c_{n-1} \)
   \( \Rightarrow \) # subproblems = \( n \)

2. guess: how many times player “hits”
   \( \Rightarrow \) # choices \( \leq n \)

3. recurrence: \( BJ(i) = \max ( \)
   \( 0(n) \Rightarrow \text{outcome } \{ +1, -1 \} + BJ(i+\#\text{cards used}) \)
   \( 0(n) \Rightarrow \text{for } \#\text{hits in } c_i, 1, \ldots \)
   \( \text{if valid play } \sim \text{don’t hit after bust} \)
   \( \Rightarrow \text{time/subproblem } = O(n^2) \)

4. order: for \( i \) in reversed(range(\( n \))
   \( \Rightarrow \text{total time } = O(n^3) \)

5. solution = \( BJ(\emptyset) \)
- Detailed recurrence: (before memoization)
  (ignoring splits/betting)

  \[ \text{BJ}(i) : \]
  
  if \( n - i < 4 \): return \( \emptyset \) (not enough cards)
  
  for \( p \) in range(2, \( n - i - 1 \)): (# cards taken)
  
  \[ \text{\( \Theta(n) \) \{ } \]
  
  player = \text{sum}(c_i, c_{i+2}, c_{i+4} : i+p+2)
  
  if player > 21: (bust)
  
  options.append(-1 + BJ(i+p+2))
  
  break
  
  options.append(BJ(i+p+2))

  \[ \text{\( \emptyset \) with care } \]
  
  for \( d \) in range(2, \( n - i - p \)):
  
  dealer = \text{sum}(c_{i+1}, c_{i+3}, c_{i+p+2} : 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)

DAG view:

- Valid plays
- Outcomes

-1
+1
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. \[ \text{DP}[i] = \min \left( \left( \text{badness}(i, j) + \text{DP}[i][j] \right), j \right) \]
  for \( j \) in \( \text{range}(i+1, n+1) \)

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

  \[ i = \emptyset \]

  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)