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 — acyclicly —
  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
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 (⇒ extra time)

Examples: Fibonacci Shortest Paths

<table>
<thead>
<tr>
<th>Step</th>
<th>Fibonacci</th>
<th>Shortest Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$F_k$ for $1 \leq k \leq n$</td>
<td>$S_k(s,v)$ for $v \in V$, $0 \leq k \leq</td>
</tr>
<tr>
<td></td>
<td>$n$</td>
<td>$= \min_{s \rightarrow v}$ path using $\leq k$ edges</td>
</tr>
<tr>
<td>2</td>
<td>nothing</td>
<td>$V^2$</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>edge into v (if any)</td>
</tr>
<tr>
<td>3</td>
<td>$F_k = F_{k-1} + F_{k-2}$</td>
<td>$\delta_k(s,v) = \min { \delta_{k-1}(s,w) + w(u,v)</td>
</tr>
<tr>
<td></td>
<td>$\Theta(1)$</td>
<td>$\Theta(1 + \text{indegree}(v))$ for $k=0, 1, \ldots,</td>
</tr>
<tr>
<td></td>
<td>$\Theta(n)$</td>
<td>for $v \in V$</td>
</tr>
<tr>
<td>4</td>
<td>for $k=0, 1, \ldots,</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>total time:</td>
<td>efficient about indeg.</td>
</tr>
<tr>
<td>5</td>
<td>$F_n$</td>
<td>$S_{</td>
</tr>
<tr>
<td></td>
<td>$\Theta(1)$</td>
<td>$\Theta(V)$</td>
</tr>
<tr>
<td></td>
<td>extra time:</td>
<td></td>
</tr>
</tbody>
</table>
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:
  
  \[ \begin{align*}
  \text{b l a h b l a h b l a h} & \quad \text{vs.} \quad \text{b l a h b l a h b l a h} \\
  \text{really long word} & \quad \text{vs.} \quad \text{really long word}
  \end{align*} \]

- define \( \text{badness}(i,j) \) for line of words \([i:j]\)
  
  \[
  \begin{cases}
  \infty & \text{if total length} \geq \text{page width} \\
  (\text{page width} - \text{total length})^3 & \text{else}
  \end{cases}
  \]

- goal: split words into lines to min. \( \Sigma \text{badness} \)

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

\[ \text{DAG:} \quad \begin{align*}
  i & \rightarrow \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc \\
  \text{badness}(i,j) & \end{align*} \]
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**: \( BJ(i) = \max \left( \begin{array}{l}
\Omega(n) \Rightarrow \text{outcome } \in \{+1, \varnothing_{n-1}\} + BJ(i+\text{#cards used}) \\
\Omega(n) \Rightarrow \text{for # hits in } \varnothing_{n-1}, \ldots, \text{if valid play } \sim \text{don’t hit after bust} \end{array} \right) \)
   \( \Rightarrow \) time/subproblem = \( \Theta(n^3) \)

4. **order**: for \( i \) in reversed(range(n))
   - total time = \( \Theta(n^3) \)

\[
\left[ \text{time is really } \sum_{i=0}^{n-1} \sum_{#h=0}^{n-i-0(1)} \Theta(n-i-#h) = \Theta(n^3) \right] \text{ still}
\]

5. **solution** = \( BJ(n) \)
- 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)
  \[ \Theta(n) \{ \]
  \( \text{player} = \text{sum}(c_{i \cdot c_{i+2} \cdot c_{i+4} : i+p+2}) \)
  if \( \text{player} > 21 \): (bust)
  \( \text{options.append}(-1 + \text{BJ}(i+p+2)) \)
  break
  \( \text{b} \text{ust} \)
  for \( d \) in range(2, n-i-p):
    \[ \Theta(n) \{ \]
    \( \text{dealer} = \text{sum}(c_{i+1 \cdot c_{i+3} \cdot c_{i+p+2} : i+p+d}) \)
    if \( \text{dealer} \geq 17 \): break
    if \( \text{dealer} > 21 \): \( \text{dealer} = \emptyset \) (bust)
    \( \text{options.append(cmp(player, dealer) + BJ(i+p+d))} \)
  \[ \} \]
return \( \max(\text{options}) \)

DAG view:

[Diagram of a Directed Acyclic Graph (DAG) showing the structure of the algorithm with valid plays and outcomes indicated.]
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:

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

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

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

- just like memoization & bottom-up, this transformation is automatic (no thinking required)