6.006	Lecture 20	Nov. 29, 2011

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 · $\underbrace{\text{time/subproblem}}$
  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 # subprobs.
2. guess (part of solution) — count # choices
3. relate subprob. solutions — compute time/subprob.
4. recurse + memoize — time = time/subprob. · #subprobs.

   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
1. subprobs:	F_k for $1 \le k \le n$	$\delta_k(s,v)$ for $v \in V$, $0 \le k < \lvert V \rvert$ = min. $s \to v$ path using $\le k$ edges
# subprobs:	n	V^2
2. guess:	nothing	edge into v (if any)
# choices:	1	indegree(v) + 1
3. recurrence:	$F_k = F_{k-1} + F_{k-2}$	$\delta_k(s,v) = \min\{\delta_{k-1}(s,u) + w(u,v) \mid (u,v) \in E\}$
time/subprob.:	$\Theta(1)$	$\Theta(1 + \text{indegree}(v))$
4. topo. order:	for $k = 1, \ldots, n$	for $k = \emptyset, 1, \ldots, \lvert V \rvert - 1$ for $v \in V$
total time:	$\Theta(n)$	$\Theta(VE)$ + $\Theta(V^2)$ unless efficient about indeg. $\emptyset$
5. orig. prob.:	F_n	$\delta_{\lvert V \rvert - 1}(s,v)$ for $v \in V$
extra time:	$\Theta(1)$	$\Theta(V)$

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:

	blah blah blah		blah blah
☹	b l a h	vs.	blah blah ☺
	reallylongword		reallylongword

- define badness(i, j) for line of words[i:j]

  e.g. $\begin{cases} \infty & \text{if total length} > \text{page width} \\ (\text{page width} - \text{total length})^3 & \text{else} \end{cases}$

- goal: split words into lines to min. $\sum_i$ badness

① subproblem = min. badness for suffix words[i:]

$\Rightarrow$ # subproblems = $\Theta(n)$ where n = # words

② guessing = where to end first line, say i:j

$\Rightarrow$ # choices = $n - i = O(n)$

③ recurrence:

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

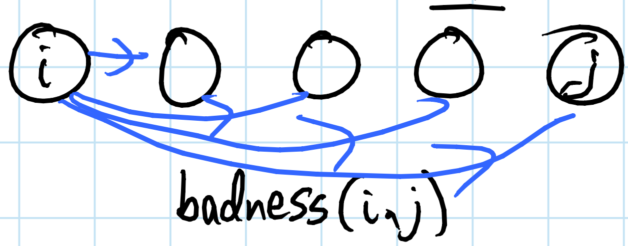
$\Rightarrow$ time per subproblem = $\Theta(n)$

④ order: for $i = n, n-1, \ldots, 1, \emptyset$

total time = $\Theta(n^2)$

⑤ solution = $DP[\emptyset]$

DAG:



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: $BJ(i)$ = best play of $\overbrace{c_i, \ldots, c_{n-1}}^{\text{remaining cards}}$

   $\uparrow$ # cards "already played"

   $\Rightarrow$ # subproblems $= n$

2. guess: how many times player "hits"

   draws another card $\uparrow$

   $\Rightarrow$ # choices $\leq n$

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

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

   $O(n) \to$ for #hits in $0, 1, \ldots$

   if valid play ~ don't hit after bust$)$

   $\Rightarrow$ time/subproblem $= \Theta(n^2)$

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

   - total time $= \Theta(n^3)$

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

5. solution $= BJ(0)$

- detailed recurrence: (before memoization)
(ignoring splits/betting)

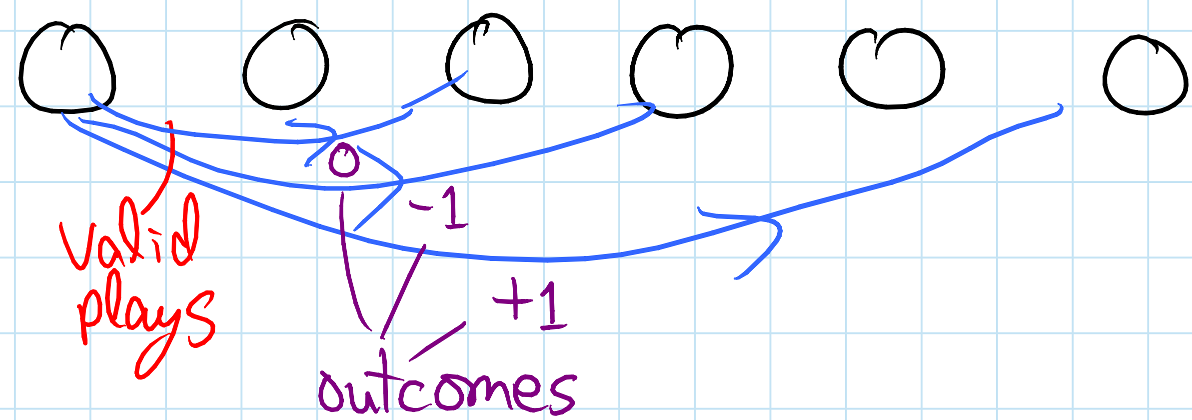
```
BJ(i):
    if n-i < 4: return 0            (not enough cards)
    for p in range(2, n-i-1):        (# cards taken)
        player = sum(c_i, c_i+2, c_i+4:i+p+2)
        if player > 21:              (bust)
            options.append(-1 + BJ(i+p+2))
            break                           ↳ bust
        for d in range(2, n-i-p):
            dealer = sum(c_i+1, c_i+3, c_i+p+2:i+p+d)
            if dealer ≥ 17: break
        if dealer > 21: dealer = 0   (bust)
        options.append(cmp(player, dealer) +
                                      BJ(i+p+d))
    return max(options)
```

$\Theta(n^2)$ (for the whole `for p` loop)

$\Theta(n)$ (player sum)

$\Theta(n)$ with care (for the `for d` loop)

DAG view:



Parent pointers:

to recover actual solution in addition to cost, store <u>parent pointers</u> (which guess used at each subproblem) & walk back

- typically: remember argmin/argmax in addition to min/max
- e.g. text justification:

(3′) $DP[i] = \min\big((\text{badness}(i,j) + DP[j][0],\ j)$ for j in range$(i+1, n+1)\big)$

$DP[n] = (0, \text{None})$

(5′)
```
i = 0
while i is not None:
    start line before word i
    i = DP[i][1]
```

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