6.006	Lecture 19	Apr. 17, 2008

Outline: Dynamic Programming I (of 4)
- Fibonacci warmup
- memoization & subproblems
- shortest paths
- Crazy Eights
- guessing viewpoint

Reading: CLRS 15

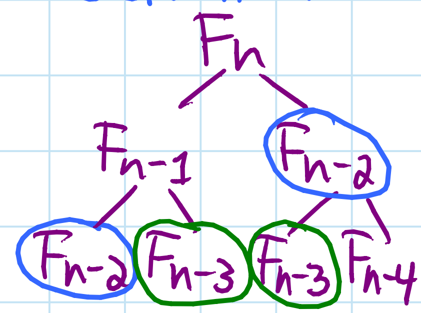
Dynamic programming: (DP) - big idea, hard, yet simple
powerful algorithmic design technique
- large class of seemingly exponential problems have a polynomial solution ("only") via DP
- particularly for optimization problems (min/max) (e.g. shortest paths)

* DP ≈ "controlled brute force"
* DP ≈ recursion + re-use

Fibonacci numbers: $F_1 = F_2 = 1$; $F_n = F_{n-1} + F_{n-2}$

- naive algorithm: follow recursive definition

```
fib(n):
    if n ≤ 2: return 1
    else: return fib(n-1)
                 + fib(n-2)
```



$$\Rightarrow T(n) = T(n-1) + T(n-2) + O(1) \approx \varphi^n$$
$$\geq 2T(n-2) + O(1) \geq 2^{n/2}$$

EXPONENTIAL — BAD!

- simple idea: memoize

```
memo = {}
fib(n):
    if n in memo: return memo[n]
    else: if n ≤ 2: f = 1
          else: f = fib(n-1) + fib(n-2)
          memo[n] = f
          return f
```

$\underbrace{\text{fib(n-2)}}_{\text{free}}$

$$\Rightarrow T(n) = T(n-1) + O(1) = O(n)$$

[side note: there is also an $O(\lg n)$-time algorithm for Fibonacci via different techniques]

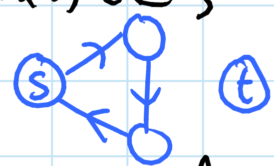
* DP ≈ recursion + memoization

- remember (memoize) previously solved "subproblems" that make up problem
  - in Fibonacci, subproblems are $F_0, F_1, \ldots, F_n$
- if subproblem already solved, re-use solution

* $\Rightarrow$ time = # subproblems · time/subproblem
  - in fib: $O(n) \cdot O(1) = O(n)$

Shortest paths:

- recursive formulation:
  $$\delta(s,t) = \min\{w(s,v) + \delta(v,t) \mid (s,v) \in E\}$$
- does this work with memoization?
  no, cycles $\Rightarrow$ infinite loops:



- in some sense necessary for neg.-weight cycles
- works for directed acyclic graphs in $O(V+E)$
  (recursion effectively DFS/topological sort)
- trick for shortest paths: removing cyclic dep.
  - $\delta_k(s,t)$ = shortest path using $\leq k$ edges
    $= \min\{\delta_{k-1}(s,t)\} \cup \{w(s,v) + \delta_{k-1}(v,t) \mid (s,v) \in E\}$
    ... except $\delta_k(t,t) = \emptyset$, $\delta_\emptyset(s,t) = \infty$ if $s \neq t$
  - $\delta(s,t) = \delta_{n-1}(s,t)$ assuming no neg. cycles

  $\Rightarrow$ time $= \underbrace{\#\text{ subproblems}}_{O(n^3) \text{ for } s,\underline{t},k \ldots \text{ really } O(n^2)} \cdot \underbrace{\text{time/subproblem}}_{O(n) \ldots \text{ really } \deg(v)}$

  $= O(V \cdot \sum_v \deg(v)) = O(VE)$

* Subproblem dependency should be acyclic

Crazy Eights puzzle:

- given a sequence of cards $c[\emptyset], c[1], \ldots, c[n-1]$
  e.g. 7♥, 6♥, 7♦, 3♦, 8♣, J♠
- find longest left-to-right "trick" (subsequence)
  $c[i_1], c[i_2], \ldots, c[i_k]$ $\quad (i_1 < i_2 < \cdots < i_k)$
  where $c[i_j]$ & $c[i_{j+1}]$ "match" for all j:
  have same suit or rank or one has rank 8
- recursive formulation:
  trick(i) = length of best trick starting at c[i]
  = 1 + max(trick(j) for j in range(i+1, n) if match(c[i], c[j]))
  best = max(trick(i) for i in range(n))
- memoize: trick(i) depends only on trick(>i)

$$\Rightarrow \text{time} = \underbrace{\#\text{ subproblems}}_{O(n)} \cdot \underbrace{\text{time/subproblem}}_{O(n)}$$

$$= O(n^2)$$

(to find actual trick, trace through max's)

"Guessing" viewpoint:

- what is the first card in best trick? guess!
  i.e. try all possibilities & take best result
  - only $O(n)$ choices
- what is next card in best trick from i? guess!
  - if you pretend you knew, solution becomes easy (using other subproblems)
  - actually pay factor of $O(n)$ to try all

* - use only small # choices/guesses per subproblem
  poly(n) ~O(1)