6.006 Introduction to Algorithms
Spring 2008

Lecture 19: Dynamic Programming I: Memoization, Fibonacci, Crazy Eights, Guessing

Lecture Overview

- Fibonacci Warmup
- Memoization and subproblems
- Shortest Paths
- Crazy Eights
- Guessing Viewpoint

Readings

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

$$F_1 = F_2 = 1; \quad F_n = F_{n-1} + F_{n-2}$$

Naive Algorithm

follow recursive definition

fib(n):
 if $n \leq 2$: return 1
 else return fib$(n-1)$ + fib$(n-2)$

$\implies T(n) = T(n-1) + T(n-2) + O(1) \approx \phi^n$
 $\geq 2T(n-2) + O(1) \geq 2^{n/2}$
 EXPONENTIAL - BAD!

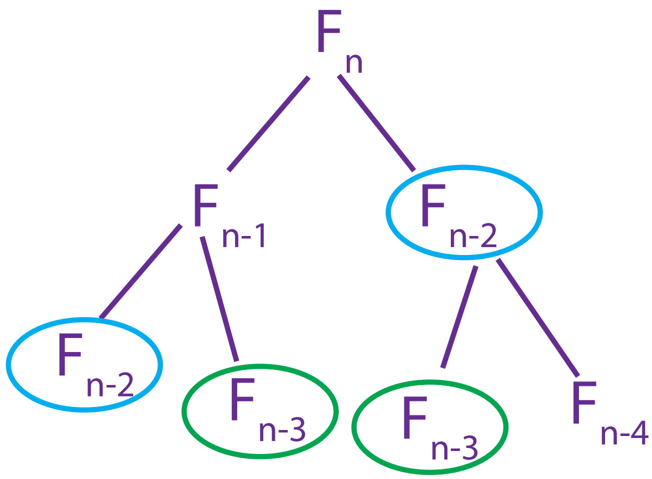

Figure 1: Naive Fibonacci Algorithm

Simple Idea

memoize

memo = { }
fib(n):
  if n in memo: return memo[n]
  else: if $n \leq 2 : f = 1$
    else: $f =$ fib$(n-1) + \underbrace{\text{fib}(n-2)}_{\text{free}}$
    memo[n] $= f$
    return f

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

[*Side Note*: There is also an $O(\lg n)$- time algorithm for Fibonacci, via different techniques]

* DP $\approx$ recursion + memoization

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

* $\implies$ time $= \sharp$ of subproblems $\cdot$ time/subproblem

- – in fib: $\sharp$ of subproblems is $O(n)$ and time/subproblem is $O(1)$ - giving us a total time of $O(n)$.

Shortest Paths

- Recursive formulation:
  $\delta(s,t) = \min\{w(s,v) + \delta(v,t) \,\big|\, (s,v) \,\epsilon\, E\}$
- does this work with memoization?
  no, cycles $\implies$ infinite loops (see Figure 2).

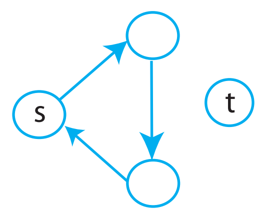

Figure 2: Shortest Paths

- 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 dependency.
  - $\delta_k(s,t)$ = shortest path using $\leq k$ edges
    $= \min\{\delta_{k-1}(s,t)\} \cup \{w(s,v) + \delta_{k-1}(v,t) \,\big|\, (s,v) \,\epsilon\, E\}$
    $\ldots$except $\delta_k(t,t) = \phi$, $\delta_\phi(s,t) = \infty$ if $s \neq t$
  - $\delta(s,t) = \delta_{n-1}(s,t)$ assuming no negative cycles

$$\implies \text{time} = \underbrace{\sharp\,\text{subproblems}}_{O(n^3)\text{ for } s,t,k\cdots \text{ really } O(n^2)} \cdot \underbrace{\text{time/subproblem}}_{O(n)\cdots\text{ really deg}V}$$

$$= O(V \cdot \sum_V \deg(V)) = O(VE)$$

* Subproblem dependency should be acyclic.

Crazy Eights Puzzle

- given a sequence of cards $c[\phi], c[1], \cdots, c[n-1]$
  e.g., $7\heartsuit, 6\heartsuit, 7\diamondsuit, 3\diamondsuit, 8\clubsuit, J\spadesuit$
- find longest left-to-right "trick" (subsequence)

  $c[i_1], c[i_2], \cdots c[i_k] \quad (i_1 < i_2 < \cdots i_k)$
  where $c[i_j]$ & $c[i_{j+1}]$ "match" for all j
  have some suit or rank or one has rank 8

- recursive formulation:

$$\begin{aligned}
\text{trick}(i) &= \text{length of best trick starting at } c[i] \\
&= 1 + \max(\text{trick}(j) \text{ for } j \text{ in range}(i+1, n) \text{ if match } (c[i], c[j])) \\
\text{best} &= \max(\text{trick}(i) \text{ for } i \text{ in range}(n))
\end{aligned}$$

- memoize: trick(i) depends only on trick($> i$)

$$\begin{aligned}
\implies \text{time} &= \underbrace{\sharp\,\text{subproblems}}_{O(n)} \cdot \underbrace{\text{time/subproblem}}_{O(n)} \\
&= O(n^2) \qquad \text{(to find actual trick, trace through max's)}
\end{aligned}$$

"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 $\underbrace{\sharp\,\text{choices/guesses}}_{\text{poly}(n) \sim O(1)}$ per subproblem