

Lecture 19: Dynamic Programming I: Memoization, Fibonacci, Shortest Paths, Guessing

Lecture Overview

- Memoization and subproblems
- Examples
 - Fibonacci
 - Shortest Paths
- Guessing & DAG View

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 \approx “controlled brute force”

* DP \approx recursion + re-use

History

Richard E. Bellman (1920-1984)

Richard Bellman received the IEEE Medal of Honor, 1979. “Bellman ... explained that he invented the name ‘dynamic programming’ to hide the fact that he was doing mathematical research at RAND under a Secretary of Defense who ‘had a pathological fear and hatred of the term, research’. He settled on the term ‘dynamic programming’ because it would be difficult to give a ‘pejorative meaning’ and because ‘it was something not even a Congressman could object to’ ” [John Rust 2006]

Fibonacci Numbers

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

Goal: compute F_n

Naive Algorithm

follow recursive definition

$\text{fib}(n)$:

if $n \leq 2$: return $f = 1$

else: return $f = \text{fib}(n - 1) + \text{fib}(n - 2)$

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

EXPONENTIAL — BAD!

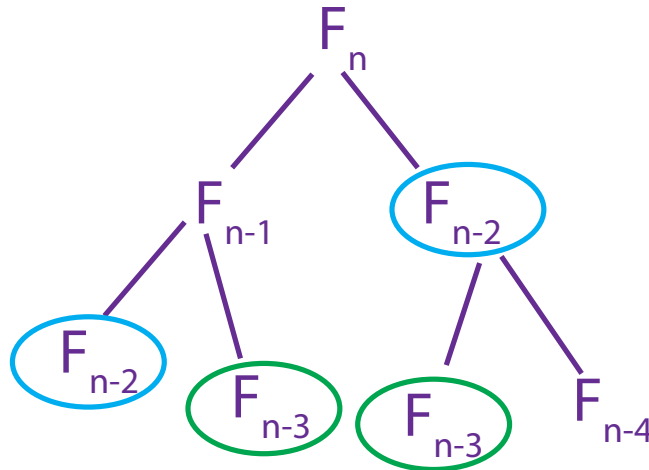


Figure 1: Naive Fibonacci Algorithm.

Memoized DP Algorithm

Remember, remember

memo = { }

$\text{fib}(n)$:

if n in memo: return memo[n]

else: if $n \leq 2$: $f = 1$

else: $f = \text{fib}(n - 1) + \text{fib}(n - 2)$

memo[n] = f

return f

- \implies $\text{fib}(k)$ only recurses first time called, $\forall k$
- \implies only n nonmemoized calls: $k = n, n-1, \dots, 1$
- memoized calls free ($\Theta(1)$ time)
- $\implies \Theta(1)$ time per call (ignoring recursion)

POLYNOMIAL — GOOD!

* $\text{DP} \approx \text{recursion} + \text{memoization}$

- memoize (remember) & re-use solutions to subproblems that help solve problem
 - in Fibonacci, subproblems are F_1, F_2, \dots, F_n

* \implies **time = # of subproblems \cdot time/subproblem**

- Fibonacci: # of subproblems is n , and time/subproblem is $\Theta(1) = \Theta(n)$ (ignore recursion!).

Bottom-up DP Algorithm

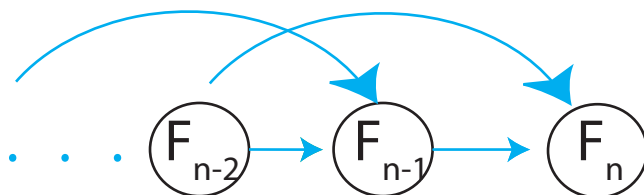
```

fib = {}
for k in [1, 2, ..., n]:
    if k ≤ 2: f = 1
    else: f = fib[k-1] + fib[k-2]
    fib[k] = f
return fib[n]

```

$\left. \begin{array}{l} \Theta(1) \\ \Theta(n) \end{array} \right\}$

- exactly the same computation as memoized DP (recursion “unrolled”)
- **in general: topological sort of subproblem dependency DAG**



- **practically faster:** no recursion
- analysis more obvious
- **can save space:** just remember last 2 fibs $\implies \Theta(1)$

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

Shortest Paths

- **Recursive formulation:**

$$\delta(s, v) = \min\{w(u, v) + \delta(s, u) \mid (u, v) \in E\}$$

- **Memoized DP algorithm: takes infinite time if cycles!**
in some sense necessary to handle negative cycles

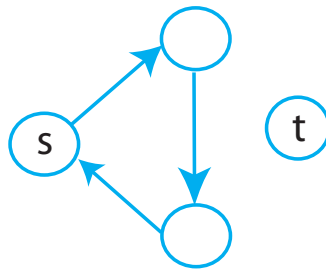


Figure 2: Shortest Paths

- **works for directed acyclic graphs in $O(V + E)$**
effectively DFS/topological sort + Bellman-Ford round rolled into a single recursion

* **Subproblem dependency should be acyclic**

- more subproblems remove cyclic dependence:
 $\delta_k(s, v)$ = shortest $s \rightarrow v$ path using $\leq k$ edges

- recurrence:

$$\begin{aligned}\delta_k(s, v) &= \min\{\delta_{k-1}(s, u) + w(u, v) \mid (u, v) \in E\} \\ \delta_0(s, v) &= \infty \text{ for } s \neq v \text{ (base case)} \\ \delta_k(s, s) &= 0 \text{ for any } k \text{ (base case, if no negative cycles)}\end{aligned}$$

- **Goal:** $\delta(s, v) = \delta_{|V|-1}(s, v)$ (if no negative cycles)

- memoize

- time: $\underbrace{\# \text{ subproblems}}_{|V| \cdot |V|} \cdot \underbrace{\text{time/subproblem}}_{O(v)} = O(V^3)$

- actually $\Theta(\text{indegree}(v))$ for $\delta_k(s, v)$

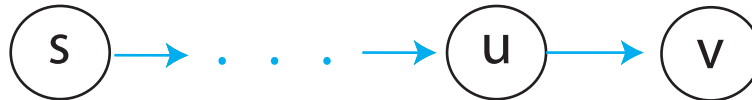
- $\Rightarrow \text{time} = \Theta(V \sum_{v \in V} \text{indegree}(V)) = \Theta(VE)$

BELLMAN-FORD!

Guessing

How to design recurrence

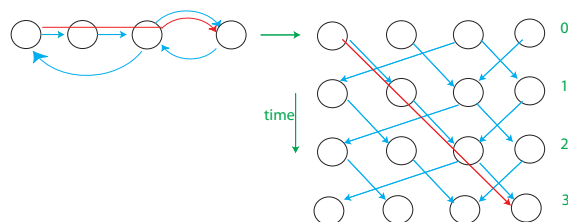
- want shortest $s \rightarrow v$ path



- what is the last edge in path? dunno
- guess it is (u, v)
- path is $\underbrace{\text{shortest } s \rightarrow u \text{ path}}_{\text{by optimal substructure}} + \text{edge } (u, v)$
- cost is $\underbrace{\delta_{k-1}(s, u)}_{\text{another subproblem}} + w(u, v)$
- to find best guess, try all ($|V|$ choices) and use best
- * key: small (polynomial) # possible guesses per subproblem — typically this dominates time/subproblem

* DP \approx recursion + memoization + guessing

DAG view



- like replicating graph to represent time
- converting shortest paths in graph \rightarrow shortest paths in DAG

* DP \approx shortest paths in some DAG

MIT OpenCourseWare
<http://ocw.mit.edu>

6.006 Introduction to Algorithms
Fall 2011

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.