6.006 Lecture 18 Nov. 15, 2016

Today: Dynamic Programming II

- 5 easy steps
- rod cutting
- text justification
- parent pointers

} examples

Summary

* DP $\approx$ "careful brute force"
  - $\approx$ guessing + recursion + memoization
  - $\approx$ dividing into reasonable # subproblems whose solutions relate - acyclicly - usually via guessing parts of solution

* time = #subproblems $\cdot$ $\underbrace{\text{time / subproblem}}_{\text{treat recursive calls as } O(1)}$

  = # subproblems $\cdot$ # guess choices $\cdot$ time / guess

- essentially an $\boxed{\text{amortization}}$
- count each subproblem only once after 1st time, costs $O(1)$ via memoization

} different runtime analysis from what we've done in past

Out-of-the-box-analysis:

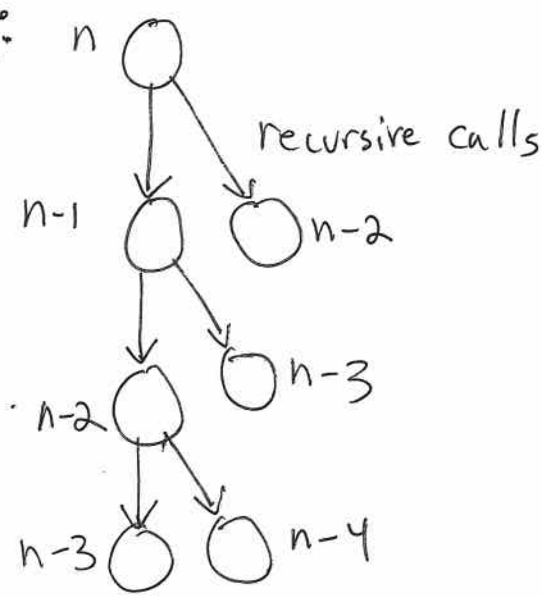
view algorithm from "outside"

how many times is subproblem j called from above? DP $\Rightarrow$ $\leq$ once!

how much time do we spend just in that subroutine, excluding recursion?

$$\text{Total time} = \sum_j \text{time spent over whole computation on subproblem } j \text{ excluding recursion}$$

Fibonacci:



* DP often ≈ shortest path in some DAG

* 5 easy steps to dynamic programming:

(1) define subproblems count # subproblems

usually min or max over these

(2) guess (part of solution) count # choices

(3) relate subproblem solutions compute time/subproblem

(4) recurse + memoize time = time/subprob × #subprobs
<u>OR</u> build DP table bottom up

+ check subprobs acyclic / topological order

(5) solve original problem:
= a subproblem
<u>or</u> by combining subprob solutions ($\Rightarrow$ extra time)

Examples

	Fibonacci	SSSP (Bellman Ford)	APSP (Floyd Warshall)
1 subprobs	F_k for $1 \le k \le n$	$\delta_k(s,v)$ $\forall v \in V$, $0 \le k < \lvert V \rvert$ = min $s \to v$ path using k edges	$C_{uv}^{(k)}$ $\forall u,v \in V$, $\forall 0 \le k \le \lvert V \rvert$ = min u-v path using only intermediate nodes in $\{1..k\}$
#subprobs	n	V^2	$V^2 \cdot V = V^3$
2 guess	nothing	edge into v	whether to use node k
# choices	1	indegree(v)+1	2
3 recurrence	$F_k = F_{k-1} + F_{k-2}$	$\delta_k(s,v) = \min_u \{\delta_{k-1}(s,u) + w(u,v) \mid (u,v) \in E\}$	$C_{uv}^{(k)} = \min\{C_{uv}^{(k-1)}, C_{uk}^{(k-1)} + C_{kv}^{(k-1)}\}$
time/subprob	$\Theta(1)$	$\Theta(1 + \text{indegree}(v))$	$\Theta(1)$
4 topo order	For $k = 1 \ldots n$	For $k = 0, 1, \ldots, \lvert V \rvert - 1$ for $v \in V$	For $k = 1, \ldots, n$ For $u \in V$ For $v \in V$
total time	$\Theta(n)$	$\Theta(VE)$	$\Theta(V^3)$
5 Original Problem	F_n	$\delta_{\lvert V \rvert - 1}(s,v)$ $\forall v \in V$	$C_{uv}^{(n)}$ $\forall u,v \in V$
extra time?	no	no	no

Rod Cutting:

Scenario:
- you manufacture steel rods
- you sell in integer length segments
- different lengths get different prices

(price / length not constant! size 1 can be more expensive than size 2?)

- you are given prices for each length

length i	1	2	3	4	5	6	7	...
price p_i	3	4	10	11	7	15	15	

- what is max revenue?

	1	2	3	4	5	6	7
max revenue r_n	3	6	10	13	16	20	23
	p_1	p_1+p_1 $(>p_2)$	p_3 $(>3p_1)$	p_1+p_3	r_1+r_4 $=p_1+p_1+p_3$	p_3+p_3	$p_1+p_3+p_3$

Formulation

subproblems:

r_i = max revenue for rod of length i

subproblems = n

guess: best initial cut (leftmost)

choices i.

recurrence:

$$r_i = \max_j \left(p_j + r_{n-j} \right)$$

time / subprob $\Theta(i)$

topo order for $i = 1..n$

total time $\Theta(n^2)$

original problem

r_n

Algorithm with memoization

```
memo = {}    ← init empty dictionary

def r(p,n)
    if n in memo: return memo[n]
    if n == 0 : return 0
    ans = -1
    for i in range (1, n.)
        ans = max (ans, p(i) + r(p, n-i))
    memo [n] = ans
    return ans
```

computes $\max_i (p(i) + r(p, n-i))$

runtime:

solve $r(p, i)$ once $\forall\ 1 \le i \le n$

"for loop" is $O(n)$

$\Rightarrow \Theta(n^2)$

Text Justification: split text into "good" lines

- obvious (MS Word/Open Office) algorithm:
  put as many words ~~fit~~ on first line, repeat
- can make very bad lines

blah blah blah
b l a h
reallylonglong longword

vs.

blah blah
blah blah
reallylonglonglongword

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

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

- goal: split words into lines to minimize $\sum$ badness

1. subproblem = min badness for suffix words $[i:]$
   $\Longrightarrow$ # subproblems $= \Theta(n)$ where $n =$ # words
2. guessing = where to end 1st line, say $[i:j]$
   $\Longrightarrow$ # choices $= n - i = O(n)$

(3) recurrence:

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

$-DP(n) = 0$

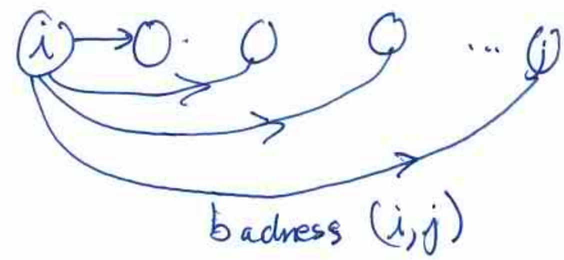
$\Rightarrow$ time/subprob $= \Theta(n)$

(4) topo order:

for $i = n, n-1, \ldots, 1, 0$

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

DAG



(5) Solution $= DP[0]$

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') $DP[i] = \min\big(\underbrace{\text{badness}(i,j) + DP[j][0]}_{\text{minimize this}},\ j\big)$ for j in range$(i+1, n+1)$)

keep this pointer

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

(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)