Lecture 20: Dynamic Programming II

Lecture Overview

- 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 # subproblems
2. guess (part of solution) — count # choices
3. relate subproblem solutions — compute time/subproblem
4. recurse + memoize — time = time/subproblem · # subproblems
   OR build DP table bottom-up
   check subproblems acyclic/topological order
5. solve original problem: = a subproblem
   OR by combining subproblem solutions $\Longrightarrow$ extra time

Examples:	Fibonacci	Shortest Paths
subprobs:	F_k for $1 \leq k \leq n$	$\delta_k(s, v)$ for $v \in V$, $0 \leq k < \lvert V \rvert$ $= \min s \to v$ path using $\leq k$ edges
# subprobs:	n	V^2
guess:	nothing	edge into v (if any)
# choices:	1	$\text{indegree}(v) + 1$
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/subpr:	$\Theta(1)$	$\Theta(1 + \text{indegree}(v))$
topo. order:	for $k = 1, \ldots, n$	for $k = 0, 1, \ldots \lvert V \rvert - 1$ for $v \in V$
total time:	$\Theta(n)$	$\Theta(VE)$ $+\ \Theta(V^2)$ unless efficient about indeg. 0
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/Open Office) algorithm: put as many words that fit on first line, repeat
- but this can make very bad lines

Figure 1: Good vs. Bad Text Justification.

- Define badness(i, j) for line of words$[i : j]$.
  For example, ∞ if total length $>$ page width, else (page width $-$ total length$)^3$.
- goal: split words into lines to min $\sum$ badness

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

3. recurrence:
   - DP[i] = min(badness $(i, j) + DP[j]$ for j in range $(i+1, n+1)$)
   - $DP[n] = 0$
     $\implies$ time per subproblem $= \Theta(n)$
4. order: for $i = n, n-1, \ldots, 1, 0$
   total time $= \Theta(n^2)$

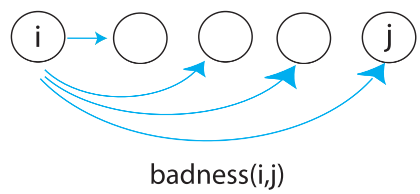

Figure 2: DAG.

5. solution $= DP[0]$

Perfect-Information Blackjack

- Given entire deck order: $c_0, c_1, \cdots c_{n-1}$
- 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 $\underbrace{c_i, \ldots c_{n-1}}_{\text{remaining cards}}$ where i is # cards "already played"
   $\implies$ # subproblems $= n$
2. guess: how many times player "hits" (hit means draw another card)
   $\implies$ # choices $\leq n$
3. recurrence: BJ(i) = max(
   outcome $\in \{+1, 0, -1\}$ + BJ(i + # cards used) $O(n)$
   for # hits in $0, 1, \ldots$ if valid play $\sim$ don't hit after bust $O(n)$

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

4. <u>order</u>: for i in reversed(range(n))
   total time $= \Theta(n^3)$
   time is really $\displaystyle\sum_{i=0}^{n-1} \sum_{\#h=0}^{n-i-O(1)} \Theta(n-i-\#h) = \Theta(n^3)$ still

5. <u>solution</u>: BJ(0)
   detailed recurrence: before memoization (ignoring splits/betting)

$\Theta(n^2)$

BJ(i):
if $n-i<4$: return 0 (not enough cards)
for p in range$(2, n-i-1)$: (# cards taken)

$\Theta(n)$

    player $= \text{sum}(c_i, c_{i+2}, c_{i+4:i+p+2})$
    if player > 21: (bust)
        options.append(-1(*bust*) $+ BJ(i+p+2)$)
        break

$\Theta(n)$ with care

    for d in range$(2, n-i-p)$
        dealer $= \text{sum}(c_{i+1}, c_{i+3}, c_{i+p+2:i+p+d})$
        if dealer ≥ 17: break
    if dealer > 21: dealer $= 0$ (bust)
    options.append(cmp(player, dealer) + BJ($i+p+d$))
return max(options)

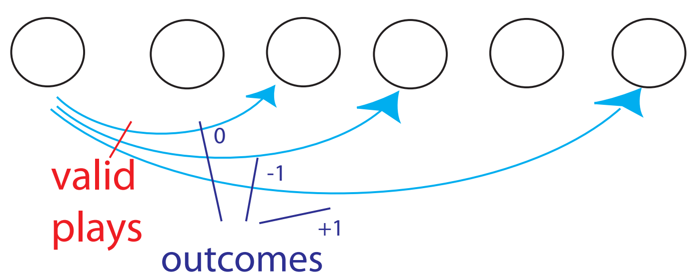

Figure 3: DAG View

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
- example: text justification

```
(3)' DP[i] = min(badness(i,j) + DP[i][0],j)
                 for j in range(i+1,n+1)
     DP[n] = (0, 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