Lecture 20: Dynamic Programming II

Lecture Overview

• 5 easy steps
• Text justification
• Perfect-information Blackjack
• Parent pointers

Summary

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

* time = # subproblems \times \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 $\approx$ shortest paths in some DAG

5 Easy Steps to Dynamic Programming

1. define subproblems \hspace{1cm} \text{count # subproblems}
2. guess (part of solution) \hspace{1cm} \text{count # choices}
3. relate subproblem solutions \hspace{1cm} \text{compute time/subproblem}
4. recurse + memoize \hspace{1cm} \text{time} = \text{time/subproblem} \times \# \text{subproblems}
   OR build DP table bottom-up
   check subproblems acyclic/topological order
5. solve original problem: = a subproblem \hspace{1cm} \implies \text{extra time}
   OR by combining subproblem solutions
Examples: Fibonacci Shortest Paths

| subprobs: | \( F_k \) | \( \delta_k(s, v) \) for \( v \in V, 0 \leq k < |V| \) |
| --- | --- | --- |
| for 1 \( \leq k \leq n \) | | = min \( s \to v \) path using \( \leq k \) edges |
| # subprobs: | \( n \) | \( V^2 \) |
| guess: | nothing | edge into \( v \) (if any) |
| # choices: | 1 | 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 \( k = 0, 1, \ldots |V| - 1 \) for \( v \in V \) |
| total time: | \( \Theta(n) \) | \( \Theta(VE) \) + \( \Theta(V^2) \) unless efficient about indeg. 0 |
| orig. prob.: | \( F_n \) | \( \delta_{|V|-1}(s, v) \) 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

![Bad vs. Good Text Justification](image)

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

1. subproblem = min. badness for suffix words\([i : ]\)
   \(\Rightarrow\) # subproblems = \(\Theta(n)\) where \(n = \#\) words
2. guessing = where to end first line, say \(i : j\)
   \(\Rightarrow\) # choices = \(n - i = O(n)\)
3. recurrence:
   - \( \text{DP}[i] = \min(\text{badness}(i, j) + \text{DP}[j] \text{ for } j \text{ in range } (i + 1, n + 1)) \)
   - \( \text{DP}[n] = 0 \)
   \( \implies \) time per subproblem = \( \Theta(n) \)

4. order: for \( i = n, n - 1, \ldots, 1, 0 \)
   total time = \( \Theta(n^2) \)

5. solution = \( \text{DP}[0] \)

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: \( \text{BJ}(i) = \text{best play of } \underbrace{c_1, \ldots, c_{n-1}}_{\text{remaining cards}} \) 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: \( \text{BJ}(i) = \max( \)
   \( \text{outcome } \in \{+1, 0, -1\} + \text{BJ}(i + \# \text{ cards used}) \) \( O(n) \)
   for \# hits in \( 0, 1, \ldots \) if valid play \sim \text{don’t hit after bust} \( O(n) \)
time/subproblem = Θ(\(n^2\))

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

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

\[
\text{BJ}(i): \quad \begin{cases} 
  \text{if } n - i < 4: & \text{return 0 (not enough cards)} \\
  \text{for } p \text{ in range}(2, n - i - 1): & (\# \text{ cards taken}) \\
  \quad \text{player} = \text{sum}(c_i, c_{i+2}, c_{i+4}; i+p+2) \\
  \quad \text{if player} > 21: & \text{(bust)} \\
  \quad \quad \text{options.append}(-1(bust) + BJ(i+p+2)) \\
  \quad \quad \text{break} \\
  \text{for } d \text{ in range}(2, n - i - p) \\
  \quad \text{dealer} = \text{sum}(c_{i+1}, c_{i+3}, c_{i+p+2}; i+p+d) \\
  \quad \text{if dealer} \geq 17: & \text{break} \\
  \quad \text{if dealer} > 21: & \text{dealer} = 0 (\text{bust}) \\
  \quad \text{options.append(cmp(player, dealer) + BJ(i+p+d))} \\
  \text{return } \max(\text{options}) 
\end{cases}
\]

Figure 3: DAG View

Parent Pointers

To recover actual solution in addition to cost, store parent pointers (which guess used at each subproblem) & walk back
• typically: remember argmin/argmax in addition to min/max

• example: text justification

(3)’ \[ DP[i] = \min(badness(i,j) + DP[i][0],j) \]
\[ \text{for } j \text{ in range}(i+1,n+1) \]
\[ DP[n] = (0, None) \]

(5)’ i = 0
\[ \text{while } i \text{ is not None:} \]
\[ \text{start line before word } i \]
\[ i = DP[i][1] \]

• just like memoization & bottom-up, this transformation is \textbf{automatic}
  no thinking required
6.006 Introduction to Algorithms
Fall 2011

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