Outline: Dynamic Programming II (of 4)
- review of big ideas & examples so far
- bottom-up implementation
- longest common subsequence
- parent pointers for guesses

Reading: CLRS 15

Summary:
* DP ≈ "controlled brute force"
* BP ≈ guessing + recursion + memoization
* BP ≈ dividing into reasonable # subproblems whose solutions relate — acyclically — usually via guessing parts of solution

* time = # subproblems \cdot \frac{\text{time/subproblem}}{} \text{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
Examples: Fibonacci, Shortest Paths, Crazy Eights

- **subprobs.**
  - \(\text{fib}(k)\) for \(0 \leq k \leq n\)
  - \(\delta_k(s,t)\) for \(k < n\)
  - \(\text{trick}(i)\)

- **#subprobs.**
  - \(\Theta(n)\)
  - \(\Theta(V^2)\)
  - \(\Theta(n)\)

- **guessing:**
  - None
  - \(\text{edge from } s\) if any
  - \(\text{next card } j\)

- **#choices:**
  - 1
  - \(\text{deg}(s)\)
  - \(n-i\)

- **relation:**
  - \(= \text{fib}(k-1) + \text{fib}(k-2)\)
  - \(\leq \min\{\delta_{k-1}(s,t)\} \cup \{w(s,v) + \delta_{k-1}(v,t)\ : v \in \text{Adj}[s]\}\)
  - \(1 + \max\{\text{trick}(j)\} \text{ for } i < j < n \text{ if } \text{match}(c[i], c[j])\)

- **time/subpr.**
  - \(\Theta(1)\)
  - \(\Theta(1 + \text{deg}(s))\)
  - \(\Theta(n-i)\)

- **DP time:**
  - \(\Theta(n)\)
  - \(\Theta(VE)\)
  - \(\Theta(n^2)\)

- **orig. prob:**
  - \(\text{fib}(n)\)
  - \(\delta_{n-1}(s,t)\)
  - \(\max\{\text{trick}(i)\}_{10 \leq i < n}\)

- **extra time:**
  - \(\Theta(1)\)
  - \(\Theta(1)\)
  - \(\Theta(n)\)
Bottom-up implementation of DP: alternative to recursion

- subproblem dependencies form DAG
- imagine topological sort
- iterate through subproblems in that order
  ⇒ when solving a subproblem, have already solved all dependencies
  - often just: "solve smaller subproblems first"

E.g. Fibonacci:

for k in range(n+1): fib[k] = ...

Shortest Paths:

for k in range(n): for v in V: d[k,v,t] = ...

Crazy Eights:

for i in reversed(range(n)): trick[i] = ...

- no recursion or memoized tests
  ⇒ faster in practice

- building DP table of solutions to all subprobs.
- can often optimize space:
  - Fibonacci: PS6
  - Shortest Paths: re-use same table ∀k
Longest common subsequence: (LCS)
A.K.A. edit distance, diff, CVS/SVN, spellchecking, DNA comparison, plagiarism detection, etc.

Given two strings/sequences x & y, find longest common subsequence LCS(x,y) sequential but not necessarily contiguous
- e.g.: HIEROGLYPHOLOGY vs. MICHAELANGELO => HELLO
-equivalent to “edit distance” (unit costs):
  # character insertions/deletions to transform x→y (everything except the matches)

- brute force: try all $2^{x_1}$ subsequences of x
  \[ \Theta(2^{x_1} \cdot y_1) \text{ time} \]
- instead: DP on two sequences simultaneously

* useful subproblems for strings/sequences x:
- suffixes $x[i:]$ \[ \Theta(x_1) \text{ cheaper} \Rightarrow \text{ use if possible} \]
- prefixes $x[:i]$ \[ \Theta(x_1^2) \]
- substrings $x[i:j]$ \[ \Theta(x_1^2) \]

- idea: combine such subproblems for x & y (suffixes or prefixes work)
**LCS DP:**

- **subproblem** \( c(i, j) = |LCS(x[i:], y[j:])| \)
  for \( 0 \leq i, j < n \)

  \( \Rightarrow \Theta(n^2) \) subproblems

- original problem \( \approx c[0, 0] \) (length \~\ find \ seq. \ later)

- **idea:** either \( x[i] = y[j] \) part of LCS
  or not \( \Rightarrow \) either \( x[i] \) or \( y[j] \) (or both)
    not in LCS (with anyone)

- **guess:** drop \( x[i] \) or \( y[j] \)? (2 choices)

- **relation among subproblems:**
  \[
  \begin{align*}
  &\text{if } x[i] = y[j]: \quad c(i, j) = 1 + c(i+1, j+1) \\
  &\text{otherwise } x[i] \text{ or } y[j] \text{ unused } \sim \text{can't help} \\
  &\text{else: } c(i, j) = \max \{ c(i+1, j), c(i, j+1) \} \\
  &\text{base cases: } c(|x|, |y|) = c(0, 0) = \emptyset
  \end{align*}
  \]

  \( \Rightarrow \Theta(1) \) time per subproblem

  \( \Rightarrow \Theta(n^2) \) total time for DP

- **DP table:**

  \[
  \begin{array}{c|c|c|c|c|c}
  \hline
  & \emptyset & 1 \cdots |y| \\
  \hline
  1 \cdots |x| & \emptyset & \cdots & \cdots & \cdots & \cdots \\
  \hline
  \end{array}
  \]

  \[
  \begin{align*}
  &\uparrow \emptyset \downarrow \\
  &\downarrow \text{if } x[i] \neq y[j] \\
  &\downarrow \text{if } x[i] = y[j] \\
  \end{align*}
  \]

  \( \leftarrow \text{linear space via antidiagonal order} \rightarrow \]

  \( \leftarrow \text{linear space via antidiagonal order} \rightarrow \]
- recursive DP:
  ```python
def LCS(x, y):
    seen = {}
    def c(i, j):
        if i >= len(x) or j >= len(y): return φ
        if (i, j) not in seen:
            if x[i] == y[j]:
                seen[i, j] = 1 + c(i+1, j+1)
            else:
                seen[i, j] = max(c(i+1, j), c(i, j+1))
        return seen[i, j]
    return c(0, 0)
```

- bottom-up DP:
  ```python
def LCS(x, y):
    c = {}
    for i in range(len(x)):
        c[i, len(y)] = φ
    for j in range(len(y)):
        c[len(x), j] = φ
    for i in reversed(range(len(x))):
        for j in reversed(range(len(y))):
            if x[i] == y[j]:
                c[i, j] = 1 + c[i+1, j+1]
            else:
                c[i, j] = max(c[i+1, j], c[i, j+1])
    return c[0, 0]
```
Recovering LCS: [material covered in recitation]
- to get LCS, not just its length,
  store parent pointers (like shortest paths)
  to remember correct choices for guesses:

\[
\begin{align*}
\text{if } x[i] &= y[j]: \\
& \quad c[i, j] = 1 + c[i+1, j+1] \\
& \quad \text{parent}[i, j] = (i+1, j+1)
\end{align*}
\]
\[
\text{else:
}\begin{align*}
& \quad \text{if } c[i+1, j] > c[i, j+1]: \\
& \quad \quad c[i, j] = c[i+1, j] \\
& \quad \quad \text{parent}[i, j] = (i+1, j) \\
& \quad \text{else:
}\begin{align*}
& \quad \quad c[i, j] = c[i, j+1] \\
& \quad \quad \text{parent}[i, j] = (i, j+1)
\end{align*}
\end{align*}
\]
- ... and follow them at the end:
\[
\begin{align*}
lcs &= [] \\
\text{here} &= (0, 0) \\
\text{while } & c[\text{here}]: \\
& \quad \text{if } x[i] = y[j]: \\
& \quad \quad lcs.\text{append}(x[i]) \\
& \quad \quad \text{here} = \text{parent}[\text{here}]
\end{align*}
\]