Outline: Dynamic Programming (DP)
- parsing grammars
- facility location
  in a tree
- map folding

Review: [6.006, in particular Spring 2008]
- DP is all about subproblems & guessing analysis:
  # Subproblems
  • #choices for guess
  • time/subprob.& guess

- 5 steps:
  ① define subproblems
  ② guess (part of solution)
  ③ relate subproblem solutions acyclicly, using guess
  ④ recurse & memoize
    or build table bottom up
  ⑤ solve original problem using one or more subprob.
- Two kinds of guessing:
  - within a subproblem (e.g. use which other subprobs.)
  - additional subproblems (report multiple solutions)
- Common subproblems:
  - for strings: suffixes s[i:], prefixes s[:i], substrings
  - for trees: subtree rooted at node s[i:j]
Context-free grammars: (CFGs)

integer → sign digits
sign → + | - | ∈
digits → digit | digit digits
digit → 0 | 1 | 2 | ... | 9
expr → expr * expr ← left associative, high precedence
| expr + expr ← left associative, low precedence
| ( expr ) | tuple | integer
tuple → ( ) | ( expr , ) | ( expr , exprs)
exprs → expr | expr , exprs

rules $R$

nonterminal ∈ $N$
empty string
terminal ∈ $T$

How to parse a string $s$ according to CFG?

[CYK: Cocke, Younger, Kasami 1965-1970]

1. subproblem = substring $s[i: j] +$ nonterminal $X$
2. guessing = which rule to apply: $X → X_0 X_1 ... X_{m-1}$
   & where $w_k$ each symbol $X_k$ starts
   (define $w_0 = i$ & $w_m = j$; require $w_0 ≤ w_1 ≤ ... ≤ w_m$)
3. $parse(i, j , X) = \text{any}(\text{all}(parse(w_k, w_{k+1}, X_k)
   \text{for } k \text{ in range}(k))
   \text{for each guess})$
   [can add in precedence, associativity, etc.]
4. base cases:
   $parse(i, j , \text{terminal}) = i + 1 = = j$
   and $s[i] = = \text{terminal}$
   $parse(i, j , \varepsilon) = i = = j$
5. $parse(0, l s l , \text{start})$
Analysis:
- \(|S|^2 \cdot |N| \) subproblems
- \((\text{#rules for } X) \cdot |S|^{m-1}\) choices/subproblem
- \(O(m)\) time/guess
- total time = \(|S|^{m+1} \sum_{X \in N} \text{(#rules for } X) \cdot O(m)\)
  = \(O(|S|^{m+1} |CFG| \cdot m)\)
  \(\text{longest right-hand side}\)
- good if right-hand sides are bounded
- e.g. can convert to Chomsky Normal Form: terms on right-hand side are of the form nonterm. nonterm. OR terminal

Always polynomial time:
① subproblem = substring \(S[i:j]\)
  + suffix \(X_k:\) of right-hand side
② guessing = which rule to apply for \(X_k:\), say \(X_k \rightarrow y_0:\)
  + where \(w\) \(X_{k+1}\) begins
③ \(\text{parse}(i, j, X_k:) = \text{any}(\text{parse}(i, w, y_0:)\) and
  \(\text{parse}(w, j, X_{k+1}:)\) for ea. guess)
④ same base cases \((X_m:\) treated like \(\varepsilon\))
⑤ add rule \(\text{start}' \rightarrow \text{start}\) ⇒ “start” is a RHS
\(\text{parse}(0, |S|, \text{start})\)

time = \(|S|^2 \cdot |CFG| \cdot |S| \cdot |CFG| \cdot O(1) = O(|S|^3 |CFG|^2)\)
  \(\#\text{subproblems} \quad \#\text{choices/s.p.} \quad \text{time/choice}\)
Facility location: given a graph \( G = (V, E) \) & int. \( k \), choose subset \( S \subseteq V \) of \( k \) vertices to minimize maximum/average/root-mean-squared shortest-path distance from any \( v \in V \) to some \( s \in S \).

- Motivation: fire/police stations in city, water fountains/bathrooms/bike racks/Athena clusters
- NP-complete in general; polynomial on trees

**Binary tree DP:** (assume rooted tree)

1. Subproblem = subtree rooted at node \( x \) & number \( k \) of vertices in \( S \) within subtree & distance \( d_l \) to nearest \( s \in S \) within subtree & distance \( d_r \) to nearest \( s \in S \) outside subtree

2. Guessing = split of \( k \) among children (\& root) & split of \( d_l \) among children

3. \( \text{OPT}(x, k, d_l, d_r) = \)

   - if \( d_l > 0 \):
     \[
     \min \left( d_l, d_r \right)^2 + \min \left( \min(\text{OPT}(x^l, k^l, d_l-1, d_r+1) + \text{OPT}(x^r, k-k^l, d_l, d_r+1)), \text{OPT}(x^l, k^l, d_l, d_r+1) + \text{OPT}(x^r, k-k^l, d_l-1, d_r+1) \right)
     \]
     for \( d_l \) in range \((d_l-1, \text{depth}(x))\) & \( k^l \) in range \((k+1)\)

   - else:
     \[
     \min(\text{OPT}(x^l, k^l, d_l^l, 1) + \text{OPT}(x^r, k-k^l-1, d_l^r, 1))
     \]
     for \( d_l^l \& d_l^r \) in range \((\text{depth}(x))\) & \( k^l \) in range \((k)\)
base cases: \( \text{OPT}(\text{null}, k, d_v, d_r) = 0 \)

\( \min(\text{OPT}(\text{root}, k, d_v, 0) \text{ for } d_v \text{ in range(depth})) \)

**Analysis:** \( n \cdot (k+1) \cdot \text{depth}^2 \cdot \text{depth}^2 \cdot (k+1) \cdot O(1) \)

\( \text{choices/subprob.} \cdot \text{time/choice} \)

\( = O(n k^2 \text{depth}^4) \)

**Splitting at a general tree node:** (replace above)

1. **subproblem** = suffix \( i \) of children & remaining \( k \) & \( d_v \) within these children
2. **guessing** = split of \( k \) & \( d_v \) into child \( i \) vs. rest \( i+1 \):
3. \( \text{node}(i, k, d_v) = \)
   \( \text{if } d_v > 0: \) min
   \( \min(\text{OPT}(\text{child } i, k', d_v-1, d_v' + 1) + \text{node}(i+1, k-k', d_v'), \)
   \( \text{OPT}(\text{child } i, k', d_v', d_v' + 1) + \text{node}(i+1, k-k', d_v')) \)
   \( \text{for } d_v' \text{ in range}(d_v-1, \text{depth}(x)) \)
   \( \text{for } k' \text{ in range}(k+1) \)

   else:
   \( \min(\text{OPT}(\text{child } i, k', d_v', 1) + \text{node}(i+1, k-k'-1, d_v'')) \)
   \( \text{for } d_v' \text{ and } d_v'' \text{ in range}(\text{depth}(x)) \)
   \( \text{for } k' \text{ in range}(k) \)

4. base case: \( \text{node}(\# \text{children}, k, d_v) = \emptyset \)
5. \( \text{node}(0, k, d_v) \)

\( \leq n \cdot \text{depth} \text{ for } x \text{ & } d_r \text{ values} \)

**Analysis:** \( [\text{degree} \cdot (k+1) \cdot \text{depth} \cdot \text{depth}^2 \cdot (k+1) \cdot O(1)] \)

\( \text{# subproblems} \cdot \text{# choices/subprob.} \cdot \text{time/} \)

**Total:** \( O(n k^2 \text{depth}^4) \)
Map folding: \([\text{Arkin, Bender, Demaine, Demaine, Mitchell, Sethia, Skiena 2004}] + \text{idea by Jeff Erickson}\)

- given mountain-valley assignment on grid, fold it flat using fewest possible simple folds

  \[\text{e.g.} \quad \begin{array}{c|c|c|c}
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  \end{array} \quad \Rightarrow \quad \begin{array}{c|c|c|c}
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  \end{array} \quad \Rightarrow \quad \begin{array}{c|c|c|c}
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  \end{array} \quad \Rightarrow \quad \boxed{\text{square}}\]

- if foldable then each vertex looks like:

  \[\begin{array}{c|c|c|c}
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  \end{array} \quad \#M=3 \quad \text{OR} \quad \begin{array}{c|c|c|c}
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  \end{array} \quad \#V=1 \quad \text{OR} \quad \begin{array}{c|c|c|c}
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  & \\[-1em]
  \hline
  \end{array} \quad \#V=3\]

  \[\Rightarrow \text{uniform lines (all M or all V) are either all vertical or all horizontal}\]

  \[\Rightarrow \text{decomposes into sequence of 1D problems:}\]

  \[\begin{array}{c|c|c|c}
  \hline
  m & m & & \\
  \hline
  & & V & V & V \\
  \hline
  \end{array} \quad \text{no fold}\]

DP:

1. subproblem = substring of 1D map
2. guessing = where to make first fold
3. check valid guess: left = opposite of right:

\[\begin{array}{c|c|c|c}
  \hline
  & m & m & \\
  \hline
  & V & V & V \\
  \hline
  \end{array}\]

  after making first fold, longer side hasn't changed at all – recurse on that, & add 1 for first fold (& min over guesses)

4. base case: no creases \(\Rightarrow\) \(\emptyset\) folds
5. apply to whole map

Analysis: \(n^2 \cdot n \cdot O(n) = O(n^4)\) time
Doing better:
- if one substring contains another, smaller substring has smaller cost
- as candidate fold goes left to right, resulting subproblem gets smaller on left
  until middle of the paper, then switches sides & gets larger on the right

⇒ only 2 options worth considering:
  nearest valid fold left & right of middle
- using Karp-Rabin fingerprinting (string matching) can find these middlemost folds in $O(n)$ time
  - as we walk left to right, shift fingerprints

⇒ $n^2 \cdot 2 \cdot O(n) = O(n^3)$ expected time

Can you do better?