Lecture 18: Dynamic Programming III

Previously

- Recurrence: Fibonacci, DAG Relaxation, Rod cutting
- Prefix/Suffix: Text Just, LIS, LCS
- **Subsequence:** Optimal Play
- **Extra state:** Bellman-Ford
- Applications: counting, optimization, string processing

Dynamic Programming Steps (SR. BST)

1. Define **Subproblems** subproblem $x \in X$
   - Describe the meaning of a subproblem **in words**, in terms of parameters
   - **Often subsets of input:** prefixes, suffixes, contiguous subsequences
   - **Often record partial state:** add subproblems by incrementing some auxiliary variables
2. **Relate** Subproblems $x(i) = f(x(j), \ldots)$ for one or more $j < i$
   - State topological order to argue relations are acyclic and form a DAG
3. Identify **Base** Cases
   - State solutions for all reachable independent subproblems
4. Compute **Solution** from Subproblems
   - Compute subproblems via top-down memoized recursion or bottom-up
   - State how to compute solution from subproblems (possibly via parent pointers)
5. Analyze Running **Time**
   - $\sum_{x \in X} \text{work}(x)$, or if $\text{work}(x) = W$ for all $x \in X$, then $|X| \times W$

Edit Distance

- Can modify a string using single character **edit operations**:
  - **Delete** a character
  - **Replace** a character with another character
  - **Insert** a character between two characters
- Can transform A into B in $O(\max(|A|, |B|))$ edit operations. What is fewest?
- Input: two strings A and B, Output: minimum number of edit operations to change A to B
- Example: $A =$ `"dog"`, $B =$ `"dingo"` minimum is 3 edits

1. **Subproblems**
   - Modify A until its last character matches last in B ($A[i]$ is ith letter in A, from 1 to $|A|$)
   - $\boxed{x(i, j)\text{: minimum number of edits to transform prefix ending at } A[i] \text{ into prefix ending at } B[j]}$
2. **Relate**
   - If $A[i] = B[j]$, then match!
   - Otherwise, need to edit to make last element from A equal to $B[j]$
   - Edit is either an insertion, replace, or deletion **(Guess!)**
   - Deletion removes $A[i]$
   - Insertion adds $B[j]$ to start of prefix ending at $A[i]$, then removes it and $B[j]$
   - Replace changes $A[i]$ to $B[j]$ and removes both $A[i]$ and $B[j]$
   - $x(i, j) = \begin{cases} x(i-1, j-1) & \text{if } A[i] = B[i] \\ 1 + \min(x(i-1, j), x(i, j-1), x(i-1, j-1)) & \text{otherwise} \end{cases}$
   - Subproblems $x(i, j)$ only depend on strictly smaller i and j, so acyclic
3. **Base**
   - $x(i, 0) = i$, $x(0, j) = j$ (need many insertions or deletions)
4. **Solution**
   - Solve subproblems via recursive top down or iterative bottom up
   - Solution to original problem is $x(|A|, |B|)$
   - (Can store parent pointers to reconstruct edits transforming A to B)
5. **Time**
   - # subproblems: $(|A| + 1)(|B| + 1)$
   - work per subproblem: $O(1)$
   - $O(|A||B|)$ running time

Arithmetic Parenthesization

- Input: arithmetic expression containing n integers, with integers a_i and a_{i+1} separated by binary operator $o_i(a, b)$ from $\{+, \times\}$
- Start with **positive** integers!
- Output: Where to place parentheses to maximize the evaluated expression
- Example: $7 + 4 \times 3 + 5 \rightarrow ((7) + (4)) \times ((3) + (5)) = 88$

1. **Subproblems**
   - Observation: Parentheses form nested structure (order of operations)
   - Idea: Use contiguous subsets of expression
   - $\boxed{x(i, j)\text{: maximum parenthesized evaluation of subsequence from integer } i \text{ to } j}$
2. **Relate**
   - Guess location of outer-most parenthesis, last operation evaluated
   - $x(i, j) = \max\{o_k(x(i, k), x(k + 1, j)) \mid k \in \{i, \ldots, j - 1\}\}$
   - Subproblems $x(i, j)$ only depend on strictly smaller $j - i$, so acyclic
3. **Base**
   - $x(i, i) = a_i$
4. **Solution**
   - Solve subproblems via recursive top down or iterative bottom up
   - Maximum evaluated expression is given by $x(1, n)$
   - Store parent pointers (two!) to find parenthesization, (forms binary tree!)
5. **Time**
   - # subproblems: less than $n \times n \times 2 = O(n^2)$
   - work per subproblem $O(n) \times 2 \times 2 = O(n)$
   - $O(n^3)$ running time

Arithmetic Parenthesization (Negative)

- Input: arithmetic expression containing n integers, with integers a_i and a_{i+1} separated by binary operator $o_i(a, b)$ from $\{+, \times\}$
- Now allow **negative** integers!
- Output: Where to place parentheses to maximize the evaluated expression
- Example: $7 + (-4) \times 3 + (-5) \rightarrow ((7) + ((-4) \times ((3) + (-5)))) = 15$

1. **Subproblems**
   - Old problems sufficient? $(-3) \times (-3) = 9 > (-2) \times (-2) = 4$
   - $\boxed{x(i, j, +1)\text{: maximum parenthesized evaluation of subsequence from integer } i \text{ to } j}$
   - $\boxed{x(i, j, -1)\text{: minimum parenthesized evaluation of subsequence from integer } i \text{ to } j}$
2. **Relate**
   - Guess location of outer-most parenthesis, last operation evaluated
   - $x(i, j, +1) = \max\{o_k(x(i, k, s_1), x(k+1, j, s_2)) \mid k \in \{i, \ldots, j-1\},\ s_1, s_2 \in \{-1, +1\}\}$
   - $x(i, j, -1) = \min\{o_k(x(i, k, s_1), x(k+1, j, s_2)) \mid k \in \{i, \ldots, j-1\},\ s_1, s_2 \in \{-1, +1\}\}$
   - Subproblems $x(i, j, s)$ only depend on strictly smaller $j - i$, so acyclic
3. **Base**
   - $x(i, i, s) = a_i$
4. **Solution**
   - Solve subproblems via recursive top down or iterative bottom up
   - Maximum evaluated expression is given by $x(1, n, +1)$
   - Store parent pointers (two!) to find parenthesization, (forms binary tree!)
5. **Time**
   - # subproblems: less than $n \times n \times 2 = O(n^2)$
   - work per subproblem $O(n) \times 2 \times 2 = O(n)$
   - $O(n^3)$ running time