Lecture 17: Dyn. Prog. III

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 $|X| \times W$

Previously

- Prefixes/suffixes, constant dependencies (MSS, Text Just, LCS, Edit Distance)
- Prefixes/suffixes, linear dependencies (Rod Cutting, LIS)
- Contiguous Subsequences, linear dependencies (Coin Game)
- Adding information (like graph duplication!)

Arithmetic Parenthesization

- Input: arithmetic expression containing n integers $A = \{a_1, \ldots, a_n\}$, with integers a_i and a_{i+1} separated by binary operator function $f_i(\cdot, \cdot)$ from $\{+, \times\}$
- Output: Where to place parentheses to maximize the evaluated expression
- Example: $7 + 4 \times 3 + 5 \rightarrow ((7) + (4)) \times ((3) + (5)) = 88$
- Allow **negative** integers!
- Example: $7 + (-4) \times 3 + (-5) \rightarrow ((7) + ((-4) \times ((3) + (-5)))) = 15$

1. **Subproblems**
   - Sufficient to maximize each subarray? No! $(-3) \times (-3) = 9 > (-2) \times (-2) = 4$
   - $\boxed{x(i, j, +1)\text{: maximum parenthesized evaluation of expression from integer } a_i \text{ to } a_j}$
   - $\boxed{x(i, j, -1)\text{: minimum parenthesized evaluation of expression from integer } a_i \text{ to } a_j}$
2. **Relate**
   - Guess location of outer-most parenthesis, last operation evaluated
   - $x(i, j, +1) = \max\{f_k(x(i, k, s_1), x(k+1, j, s_2)) \mid k \in \{i, \ldots, j-1\},\ s_1, s_2 \in \{-1, +1\}\}$
   - $x(i, j, -1) = \min\{f_k(x(i, k, s_1), x(k+1, j, s_2)) \mid k \in \{i, \ldots, j-1\},\ s_1, s_2 \in \{-1, +1\}\}$
   - for all $i, j \in \{1, \ldots, n\}$ where $i < j$
   - Subproblems $x(i, j, s)$ only depend on strictly smaller $j - i$, so acyclic
3. **Base Cases**
   - $x(i, i, s) = a_i$ for $s \in \{1, -1\}$, only one number, no operations left!
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

Egg Drop

- Drop eggs from floors of an n story building
- Want to find highest floor an egg can be dropped without breaking
- Want to minimize the number of drops for a fixed number of eggs
- If you only have one egg, test each floor going up until it breaks (n)
- If you have infinite eggs, binary search ($\log n$)
- If allowed to break at most k eggs, somewhere in between

1. **Subproblems**
   - Store number of floors remaining to check and number of unbroken eggs
   - $\boxed{x(f, e)\text{: minimum number of drops to check any sequence of } f \text{ floors using } e \text{ eggs}}$

2. **Relate**
   - Case 1: drop an egg from a floor and it breaks
   - Case 2: drop an egg from a floor and it does not break
   - In the worst cast, an adversary picks the case that maximizes drops
   - $x(f, e) = 1 + \min\Big\{\max\{x(f' - 1, e - 1), x(f - f', e)\} \;\Big|\; f' \in \{1, \ldots, f\}\Big\}$
   - for all $f \in \{1, \ldots, n\}$ and $e \in \{1, \ldots, k\}$
   - Subproblems $x(f, e)$ only depend on subproblems with strictly smaller f, so acyclic

3. **Base Cases**
   - $x(0, e) = 0$ (no floors to check, we're done!)
   - $x(f, 0) = \infty$ for $f > 0$ (can't succeed without eggs)

4. **Solution**
   - Solve subproblems via recursive top down or iterative bottom up
   - For bottom up, can solve in order of increasing f, then increasing e
   - Final answer is $x(n, k)$
   - Can store parent pointers to reconstruct worst case optimal floor sequence

5. **Time**
   - # subproblems: $(n + 1)(k + 1) = O(nk)$
   - work per subproblem: $O(f) = O(n)$
   - $O(n^2 k)$ running time