Section 6: What Tool?

1. DP on Trees

You are given a tree $T = (V, E)$ with nonnegative edge weights. You want to determine the diameter of the tree, which is the longest distance between any two nodes. The goal is to design a DP which runs in $\mathcal{O}(n)$ where $n = |V|$.

1.1. Write the Dynamic Program

(a) Formulate the problem recursively – what are you looking for (in English!!), and what parameters will you need as you're doing the calculation? It will be useful to fix a root node in the tree first, then every node in the tree has a list of children nodes you can access.

(b) Write a recurrence for solving the problem you defined in the last part (the recurrence is for the answer, **not** the running time).

(c) What is your final answer (e.g. what parameters for the recurrence do you need? Is it a single value or the max/min of a set of values?)?

(d) Give a brief justification for why your recurrence is correct. You do not need a formal inductive proof, but your intuition will likely resemble one.

1.2. Analyze the Dynamic Program

(a) Describe a memoization structure for your algorithm.

(b) Describe a filling order for your memoization structure.

(c) State and justify the running time of an iterative solution.

2. Longest Increasing Subsequence AGAIN

We've already seen a recurrence for Longest Increasing Subsequence. Let's write another!

As before, $[10, -2, 5, 0, 3, 11, 8]$ has a longest increasing subsequence of 4 elements: $[-2, 0, 3, 8]$

2.1. Write the Dynamic Program

(a) Formulate the problem recursively – what are you looking for (in English!!), and what parameters will you need as you're doing the calculation? To make sure you get a different solution than the one from class, you should ask yourself to answer the question "what's the longest increasing subsequence where the first included element is the one at index i, and how would I find that?"

(b) Write a recurrence for solving the problem you defined in the last part (the recurrence is for the answer, **not** the running time).

(c) What is your final answer (e.g. what parameters for the recurrence do you need? Is it a single value or the max/min of a set of values?)?

(d) Give a brief justification for why your recurrence is correct. You do not need a formal inductive proof, but your intuition will likely resemble one.

2.2. Analyze the Dynamic Program

(a) Describe a memoization structure for your algorithm.

(b) Describe a filling order for your memoization structure.

(c) State and justify the running time of an iterative solution.

3. Longest Common Subsequence

Given two strings s with length m and t with length n, find the length of their longest common subsequence. A subsequence is a (possibly non-contiguous) substring.

Examples:

Input s="backs", t="arches": the longest common subsequence is "acs", so the output should be 3.

Input s="skaters", t="hated": the longest common subsequence is "ate", so the output should be 3.

3.1. Write the Dynamic Program

(a) Formulate the problem recursively – what are you looking for (in English!!), and what parameters will you need as you're doing the calculation?

(b) Write a recurrence for solving the problem you defined in the last part (the recurrence is for the answer, **not** the running time).

(c) What is your final answer (e.g. what parameters for the recurrence do you need? Is it a single value or the max/min of a set of values?)?

(d) Give a brief justification for why your recurrence is correct. You do not need a formal inductive proof, but your intuition will likely resemble one.

3.2. Analyze the Dynamic Program

(a) Describe a memoization structure for your algorithm.

(b) Describe a filling order for your memoization structure.

(c) State and justify the running time of an iterative solution.

4. k Minimum Disjoint Subarrays, Each with Target Sum t

You are given an array of positive integers $A[n]$ and an positive integer target t. You need to find k disjoint (non-overlapping) subarrays of A that each have a sum of their elements equal to the target and such that the sum of the lengths of the subarrays is minimized. If there are not k such subarrays, return ∞.

Examples:

Input $k = 4$, $t = 2$, and the array $[1, 1, 8, 2, 3, 2, 1, 1, 2]$: the four smallest disjoint subarrays whose elements each sum to 2 are $[[1, 1], 8, [2], 3, [2], 1, 1, [2]]$, so the output should be 5.

Input $k = 2$, $t = 5$, and the array $[2, 2, 2, 2]$: there aren't two subarrays whose elements each sum to 5, so the output should be ∞.

Input $k = 3$, $t = 4$, and the array $[1, 3, 1, 2, 1, 4, 2, 2]$: the three smallest disjoint subarrays whose elements each sum to 4 are $[[1, 3], 1, 2, 1, [4], [2, 2]]$ so the output should be 5.

4.1. Write the Dynamic Program

(a) Formulate the problem recursively – what are you looking for (in English!!), and what parameters will you need as you're doing the calculation?

(b) Write a recurrence for solving the problem you defined in the last part (the recurrence is for the answer, **not** the running time).

(c) What is your final answer (e.g. what parameters for the recurrence do you need? Is it a single value or the max/min of a set of values?)?

(d) Give a brief justification for why your recurrence is correct. You do not need a formal inductive proof, but your intuition will likely resemble one.

4.2. Analyze the Dynamic Program

(a) Describe a memoization structure for your algorithm.

(b) Describe a filling order for your memoization structure.

(c) State and justify the running time of an iterative solution.