CSE 421 Section 6

Dynamic Programming

Problem 1:
DP on Trees

Problem 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 $O(n)$ where $n = |V|$.

Problem 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.

Problem 1.1a

- 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.

Problem 1.1a

- 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.

Let START(v) be the distance of the longest path starting and v and ending in node in the subtree rooted at v. Let THRU(v) be the longest path which passes through v but stays inside the subtree rooted at v.

Problem 1.1b

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

Problem 1.1b

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

$$\text{START}(v) = \begin{cases} 0 & \text{if } v \text{ has no children} \\ \max\limits_{c \text{ is a child of } v} \{w_{(c,v)} + \text{START}(c)\} & \text{otherwise} \end{cases}$$

$$\text{THRU}(v) = \begin{cases} 0 & \text{if } v \text{ has less than two children} \\ \max\limits_{\substack{c_1, c_2 \text{ distinct} \\ \text{children of } v}} \{w_{(c_1,v)} + \text{START}(c_1) + w_{(c_2,v)} + \text{START}(c_2)\} & \text{otherwise} \end{cases}$$

Problem 1.1c

- 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?)?

Problem 1.1c

- 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?)?

$$\max_v \{\text{START}(v), \text{THRU}(v)\}.$$

Problem 1.1d

- 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.

Problem 1.1d

- 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.

The longest path that starts at a node v and ends in a node in a subtree rooted at v has two cases. Either this longest path is empty since v has no children, or the longest path goes through some child of v. That path must consist of the edge from v to its child plus the longest path starting at the child and staying within the subtree rooted at the child.

Similarly, for the longest path that passes through a node v, if it has less than two children then there is no such path. Otherwise we know that the path must go through two of v's children. Hence the longest path is the sum of the two edges that go to v's children plus the longest paths that start at each children respectively and stays within the subtrees rooted at each child.

Both recurrences encode only distance of paths. Also the longest path in the tree will have a unique node which is closest to the root and that path can either start at that node or pass through it, thus maxing over all vertices in both START and THRU is sufficient for finding this distance.

Problem 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.

Problem 1.2 - Analyze the Dynamic Program

a) Describe a memoization structure for your algorithm.

We need two lists both of size n

b) Describe a filling order for your memoization structure.

We fill from leaves up to the root, making sure that we have evaluated on all children node before evaluating a parent node.

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

For each node, we need to only access the weight of the edges between the node and its children. For START this means we in total only need E many calls total.
For THRU we can implement the max over two children by just finding the largest and second largest child, so again E many calls in total.
Thus our runtime is $O(|E|) = O(|E|)$

Problem 2:
Longest Increasing Subsequence

Problem 2 – Longest Increasing Subsequence

Given an array A, find the longest sub-array of A whose elements are in increasing order.

As an example, [10, −2, 5, 0, 3, 11, 8] has a longest increasing subsequence of 4 elements: [−2, 0, 3, 8]

Problem 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.

Problem 2.1a

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?"

Problem 2.1a (2)

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?"

Let LISAlt(i) be the length of the longest increasing subsequence of $A[]$ where element i is the first element of the subsequence.

Problem 2.1b

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

Problem 2.1b (2)

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

$$\text{LISAlt}(i) = \begin{cases} 1 & \text{if } i = n \\ 1 + \max_{j>i} \mathbf{1}\left[A[i] < A[j]\right] \cdot \text{LISAlt}(i) & \end{cases}$$

Problem 2.1c

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?)?

Problem 2.1c (2)

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?)?

$$\max_i \text{LISAlt}(i)$$

Problem 2.1d

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.

Problem 2.1d (2)

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.

For the base case, since n is the farthest right element, it is the only element in a subsequence starting from that location.

If we begin at element i, then either it is the only element or there is an element after. The recurrence checks all elements after – if they are the second element in that sequence, they must be after i, have the element be greater than $A[i]$. That new location j will then start the rest of the increasing subsequence, so making all those recursive calls suffices to find the best one

Problem 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.

Start brainstorming some answers to these questions.

Problem 2.2a

a) Describe a memoization structure for your algorithm.

Problem 2.2a (2)

a) Describe a memoization structure for your algorithm.

We need a (1D) array of size n.

Problem 2.2b

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

Problem 2.2b (2)

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

We fill from n down to 1.

Problem 2.2c

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?)?

That's All, Folks!

Thanks for coming to section this week!
Any questions?