Disclaimer: The following problem is taken from the textbook *Algorithms by Jeff Erickson*, and is intended for students in a first / second course in algorithms at UIUC. As per the author's stipulations, the instructional staff did solve *all* parts of the problem beforehand in a reasonable time frame before setting it. There is a certain threshold of efficiency you must meet to receive full credit.[1]

Problem 1

Rooted minors are generalizations of subsequences. A rooted minor of a rooted tree T is any tree obtained by *contracting* one or more edges. When we contract an edge $u \to v$, where u is the parent of v, the children of v become new children of u and then v is deleted. In particular, the root of T is also the root of every rooted minor of T.

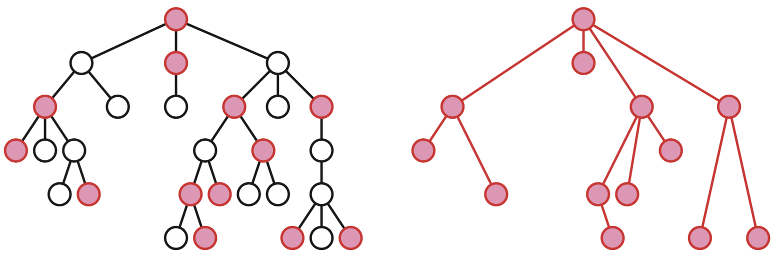

Figure 1: A rooted tree and one of its rooted minors.

a) Let T be a rooted tree with labeled nodes. We say that T is *boring* if, for each node x, all children of x have the same label[2]; children of different nodes may have different labels. Describe a dynamic programming algorithm to find the size of the largest boring rooted minor of a given labeled rooted tree. Here, we understand the size as the number of vertices on the tree.

b) Suppose we are given a rooted tree T whose nodes are labeled with numbers. Describe an algorithm to find the size of the largest *heap-ordered rooted minor* of T. That is, your algorithm should return a rooted minor M such that every node in M has a smaller label than its children in M.

c) Suppose we are given a *binary* tree T whose nodes are labeled with numbers. Describe an algorithm to find the size of the largest *binary-search-ordered rooted minor*[3] of T. That is, your algorithm should return a rooted minor M such that every node in M has at most two children, and an inorder traversal of M is an increasing subsequence of an inorder traversal of T.

[1]We will not give out these thresholds, but a correct suboptimal solution will still receive partial credit.

[2]not necessarily the same label as x

[3]what a mouthful...

d) In an *ordered* rooted tree, each node has a *sequence* of children. Describe an algorithm to find the size of the largest binary-search-ordered [rooted] minor of an *arbitrary* ordered [rooted] tree T whose nodes are labeled with numbers. That is, your algorithm should return a rooted minor M such that M is a BST, and the left-to-right order of nodes in M are consistent with their order in T.

e) Two ordered rooted trees are isomorphic if they are both empty, or if their ith subtrees are isomorphic for every index i. Describe an algorithm to find the size of the largest common ordered rooted minor of two ordered labeled rooted trees. *[this will require some thought]*

f) **(10% extra credit)** In an *unordered* rooted tree, each node has an unordered *set* of children. Two unordered rooted trees are isomorphic if they are both empty, or the subtrees of each root *can be ordered so that* their ith subtrees are isomorphic for every index i. Describe an algorithm to find the size of the largest common unordered rooted minor of two unordered labeled rooted trees. *[Hint: Combine dynamic programming with maximum flows.]*

If any of your answers incorporate dynamic programming, they should include the following:

i. A description of your DP states, in plain English, including the dimension of your table.

ii. A mathematical recurrence relation between subproblems. Don't forget your base case(s). Briefly explain why your recurrence yields the correct answer.

iii. How do you get the final answer from the entries of your table.

iv. State the runtime of your design (in big-$\mathcal{O}$ notation) and briefly justify your answer.

For this homework only: For problems that ask for an algorithm that computes an optimal structure—such as a subset, partition, subsequence, or tree—an algorithm that computes only the *value* or *cost* (such as size) of the optimal structure is sufficient for full credit.