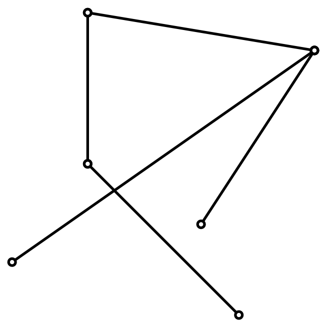
Drawing Trees

ECE 1762 Algorithms and Data Structures
Fall Semester, 2004

1 How to draw a tree

1.1 Defining the problem

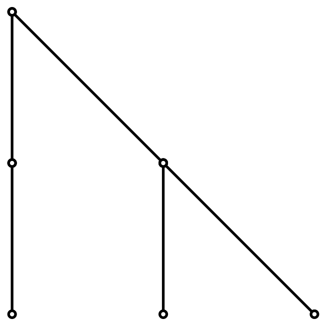
The first algorithm we examine is one to determine a clean way to draw a tree. For instance, we could draw a tree in this fashion:



This is undesirable. We can give an *aesthetic* to define properties that we would like to see in the output:

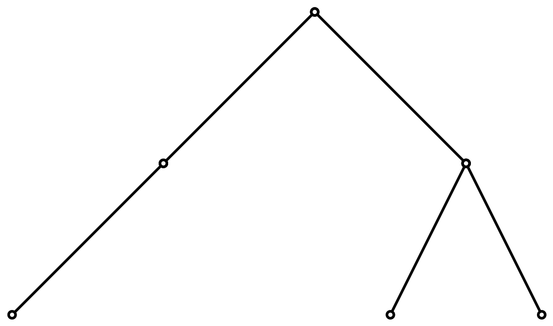
Aesthetic 1: Nodes on the same level should be on a straight line, and lines defining levels should be parallel and evenly spaced. As well, nodes on a level should be in the same left-to-right order as in the level-order traversal.

These observations lead to this drawing:



This is better but still not great. Let us insist on a second aesthetic:

Aesthetic 2: A left child must be to the left of its parent; a right child must be to the right of its parent.



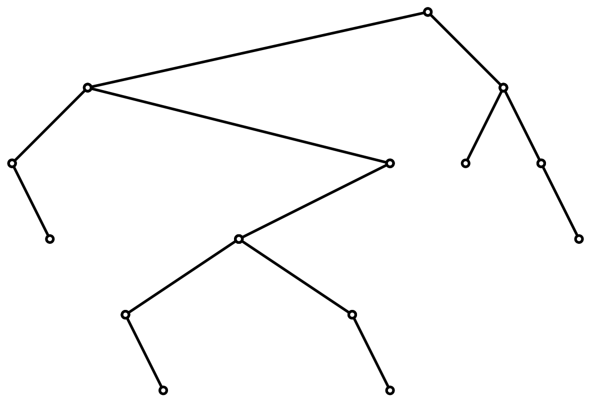
How can we accomplish this algorithmically?

1.2 Knuth's algorithm (1971)

Knuth's algorithm is a simple recursive algorithm for drawing a tree. To understand the order of the operations, think of the tree being printed on a line printer *on its side*, so the right part of the tree comes out first.

1. Go down a level and print the *right subtree* recursively.
2. Go up a level and to the left and print the *root*.
3. Go down a level and to the right and print the *left subtree* recursively.

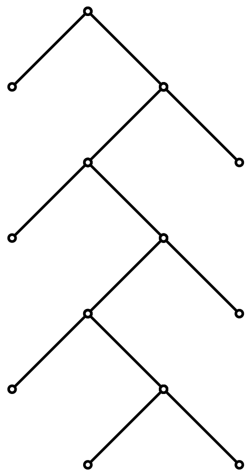
This algorithm yields results like this:



1.3 Redefining the problem: additional aesthetics

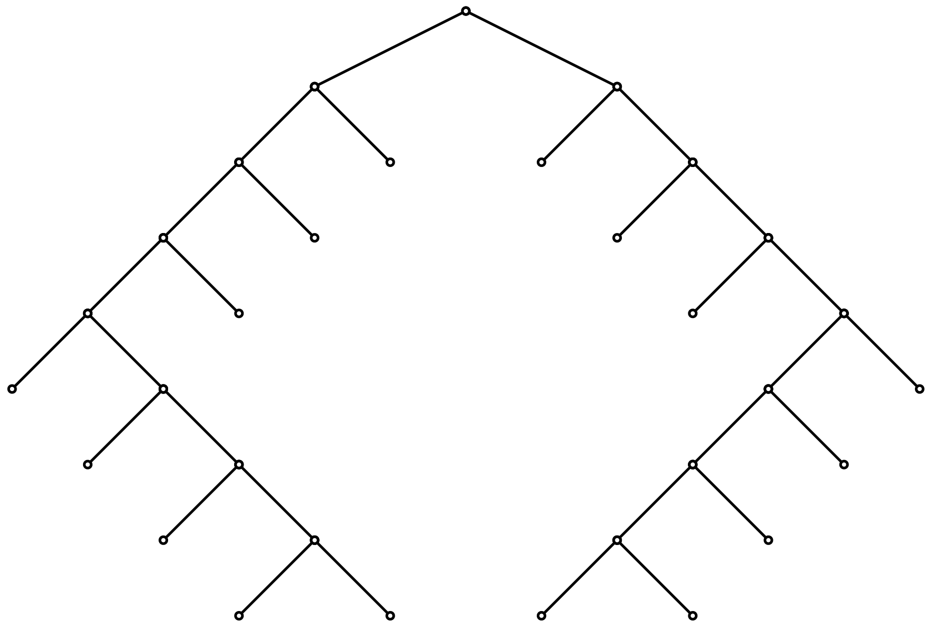
This is reasonable but still not quite what we'd like. We can propose a third aesthetic:

Aesthetic 3: A parent should be centered over its children.

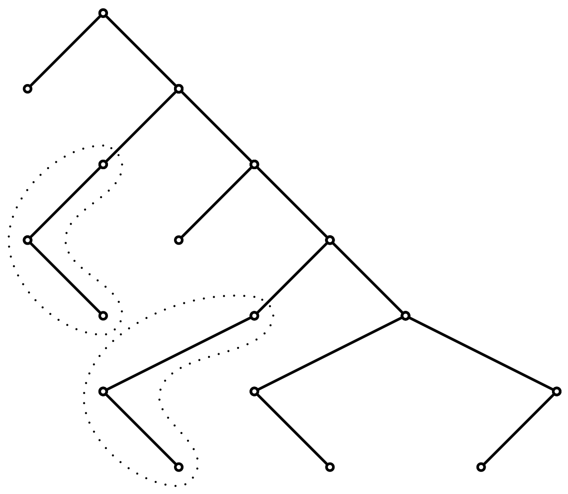


A fourth aesthetic that we propose demands some form of symmetry:

Aesthetic 4: A tree and its mirror image should yield drawings that are mirror images. More generally, a subtree should be drawn in the same way, wherever it occurs in the tree.



Note that this aesthetic requires that certain trees be drawn in more space than absolutely necessary. Notice that this tree, drawn as narrowly as possible, requires that two identical subtrees be drawn differently, violating aesthetic 4:



1.4 A new algorithm

1. Recursively place the left subtree.
2. Recursively place the right subtree.
3. Put the two rigidly formed subtrees as close together as possible. That is, place the two subtrees so their roots overlap, and then move them apart just enough so no part of the left subtree overlaps a part of the right subtree.

How can we follow the inner contours of the subtrees? We can use the notion of a *threaded tree* to help us. Nodes whose "contour successors" are not their children must be leaves and thus have must each have a null pointer. If, in the place of these null pointers, we insert auxiliary pointers, or threads, we can follow the contour of a tree beyond a leaf.

1.5 What about narrow trees?

What happens if we modify our aesthetics and demand the narrowest output possible? The problem then becomes significantly more difficult. The difficulty lies in the idea that to find the narrowest possible drawing of a tree, its subtrees must sometimes be drawn more widely than necessary. More generally, optimality of the substructure does not necessarily yield optimality of the whole structure. This is similar to the difficulty found in the traveling salesman problem. We will return to this problem later in the semester.