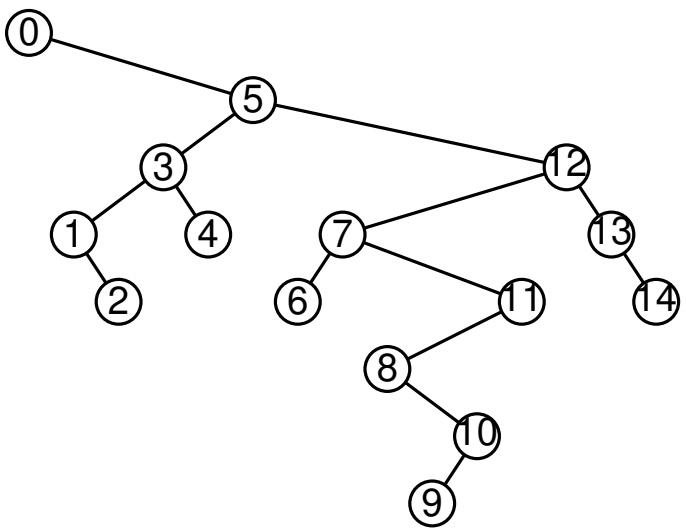
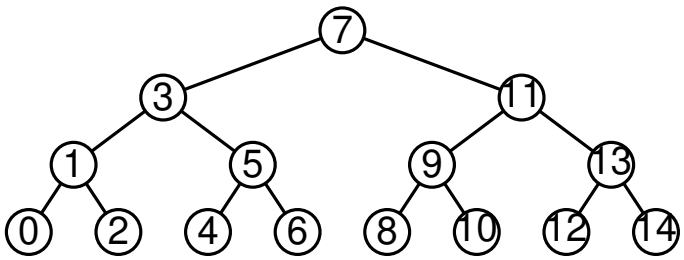
What is a Binary Search Tree?

A **binary tree** is a structure for storing ordered data. Binary trees are composed of **nodes** arranged hierarchically. Each node also contains a data value. Branching at each node occurs in at most two directions: left or right.

Here are two different ways to represent the integers from 0 to 14 in a binary tree:





More properly, these binary trees are called **binary search trees**, because in them each node's right child is always greater than its parent's value, and each left child always has a lesser value than its parent. Binary search trees are the most common use of binary trees, so the word "search" is often omitted.

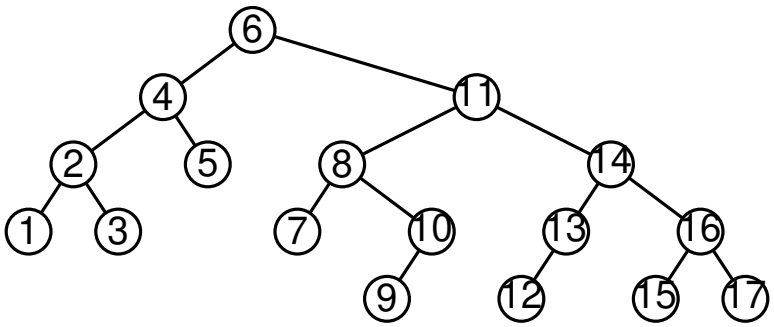
Why are Binary Trees Useful?

Binary trees are handy for rapid searching in computer programs, when they are properly arranged. To search a binary tree for a speci£ed "target" value, start at the topmost node, called the **root**, then repeatedly move down the tree, either to the left if the target is less than the current node's value, or to the right if the target is greater. Eventually, the target will be found or the search will run off the bottom of the tree. In the latter case, the target is not in the tree.

What is a Balanced Tree?

Any of the data values in the binary tree on the right above can be found using 4 or fewer comparisons. Of course, it can take longer to search the binary tree on the left: as many as 8 comparisons. This displays the advantage of binary trees with small heights. A **balanced tree** is a binary search tree that conforms to a set of rules that act to minimize its height.

As an example, consider the rules for constructing **AVL trees**. Take the **height** of a binary tree as the number of levels of nodes that it contains, the **left subtree** of a node as the binary tree with the node's left child as its root, and similarly for the **right subtree**. Then, a binary tree is an AVL tree if, for each node in the tree, the height of its left subtree differs from the height of its right subtree by no more than 1. The binary tree above on the right is an AVL tree, but the one on the left is not. Here's another AVL tree:



LIBAVL

LIBAVL is a collection of computer subroutines for constructing and manipulating binary trees and balanced trees. LIBAVL began in 1998 as an implementation of AVL trees only, hence the name. The newest released version also supports **red-black trees** and some AVL tree variants called **threaded AVL trees** and **right-threaded AVL trees**. This version of LIBAVL is used in a number of programs available on the Internet.

What is a Literate Program?

A **literate program** is intended to be useful, as is all software. But a literate program is also intended to be a work of technical literature that explains the program's inner workings. A literate program is at the same a functional design and an essay that describes that design.

The ideas behind literate programming date back to at least the 1960s, but credit for its popularization is due to eminent computer scientist Donald Knuth, who coined the name itself in his 1984 article "Literate Programming."

The bene£ts of literate programming are numerous. Literate programs tend to have good organization and design because of the need to justify decisions to readers. They can be easier to maintain because reasons for design decisions are clear. They can be easier to write because of the enhanced ability, available with most literate programming systems, to write code in the order most convenient for the programmer, instead of that most suitable for the compiler.

Literate programming does have some drawbacks. The most apparent is that it requires additional skills on the part of the programmer. Not every programmer is also a talented essayist, but some writing ability is necessary to be an effective literate programmer.

The most well-known examples of literate programs are TEX and METAFONT by Knuth himself. Taken as literature, both of these programs are published in book form as well as available electronically. Taken as software, they are commonly used for publishing articles and textbooks, especially those containing mathematics.

LIBAVL 2.0:

A Literate Balanced Tree Library

The original version of LIBAVL worked well, as evidenced by its use in several programs. But over time it turned out that the knowledge that it embodied was as important as the actual code. That is, the code itself is a realization of one way to manipulate a few kinds of balanced trees, but it gives few hints of the reasons for trade-offs made during its design or how to construct similar libraries given different priorities. Reasons to build new, similar libraries kept cropping up, but each time much of the rationale behind LIBAVL had to be rediscovered, even by its original author. The needed information was scattered among books, papers, and online sources.

Eventually, a solution presented itself: rewrite LIBAVL as a literate program! Upon investigation, this idea came to look better and better. Early in 2000, the rewrite began. This rewrite is still underway, but considerable progress has been made. A development version of the rewritten work from December 2000 is included in the binder by this poster. A considerably newer edition exists, but this is the latest fully self-consistent version.

The £rst step in writing LIBAVL 2.0 was to decide on a literate programming system. Several of these exist. Eventually, it was determined that none of them exactly met LIBAVL's needs. As a result, LIBAVL uses its own custom literate programming system, called TexiWEB. This name is taken by combining 'Texinfo', the publishing system that TexiWEB is based on, with 'WEB', Knuth's own literate programming system.