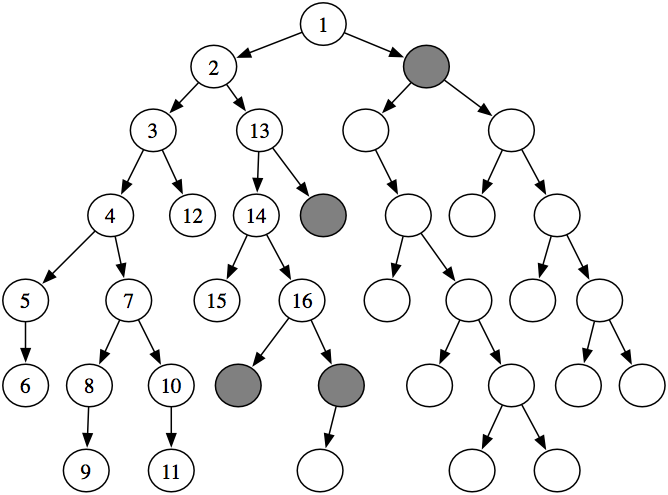
Depth-First Search

The first strategy is depth-first search. In depth-first search, the frontier acts like a last-in first-out stack.The elements are added to the stack one at a time. The one selected and taken off the frontier at any time is the last element that was added.



Consider the tree-shaped graph. Suppose the start node is the root of the tree (the node at the top) and the nodes are ordered from left to right so that the leftmost neighbor is added to the stack last. In depth-first search, the order in which the nodes are expanded does not depend on the location of the goals. The first sixteen nodes expanded are numbered in order of expansion. The shaded nodes are the nodes at the ends of the paths on the frontier after the first sixteen steps.

Notice how the first six nodes expanded are all in a single path. The sixth node has no neighbors. Thus, the next node that is expanded is a child of the lowest ancestor of this node that has unexpanded children.

Implementing the frontier as a stack results in paths being pursued in a depth-first manner - searching one path to its completion before trying an alternative path. This method is said to involve backtracking: The algorithm selects a first alternative at each node, and it *backtracks* to the next alternative when it has pursued all of the paths from the first selection. Some paths may be infinite when the graph has cycles or infinitely many nodes, in which case a depth-first search may never stop.

This algorithm does not specify the order in which the neighbors are added to the stack that represents the frontier. The efficiency of the algorithm is sensitive to this ordering.

**COMPLEXITY**

Algorithm *A* is better than *B*, using a measure of either time, space, or accuracy, could mean:

* the worst case of *A* is better than the worst case of *B*; or
* *A* works better in practice, or the average case of *A* is better than the average case of *B*, where you average over typical problems; or
* you characterize the class of problems for which *A* is better than *B*, so that which algorithm is better depends on the problem; or
* for every problem, *A* is better than *B*.

The worst-case asymptotic complexity is often the easiest to show, but it is usually the least useful. Characterizing the class of problems for which one algorithm is better than another is usually the most useful, if it is easy to determine which class a given problem is in. Unfortunately, this characterization is usually very difficult.

Characterizing when one algorithm is better than the other can be done either theoretically using mathematics or empirically by building implementations. Theorems are only as valid as the assumptions on which they are based. Similarly, empirical investigations are only as good as the suite of test cases and the actual implementations of the algorithms. It is easy to disprove a conjecture that one algorithm is better than another for some class of problems by showing a counterexample, but it is much more difficult to prove such a conjecture.

If there is a solution on the first branch searched, then the time complexity is linear in the length of the path; it considers only those elements on the path, along with their siblings. The worst-case complexity is infinite. Depth-first search can get trapped on infinite branches and never find a solution, even if one exists, for infinite graphs or for graphs with loops. If the graph is a finite tree, with the forward branching factor bounded by *b* and depth *n*, the worst-case complexity is *O(bn)*.