```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  cutoff_occurred? ← false
  if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
  else if DEPTH[node] = limit then return cutoff
  else for each successor in EXPAND(node, problem) do
      result ← RECURSIVE-DLS(successor, problem, limit)
      if result = cutoff then cutoff_occurred? ← true
      else if result ≠ failure then return result
  if cutoff_occurred? then return cutoff else return failure
```

Figure 3.13 A recursive implementation of depth-limited search.

the left subtree were of unbounded depth but contained no solutions, depth-first search would never terminate; hence, it is not complete. In the worst case, depth-first search will generate all of the $O(b^m)$ nodes in the search tree, where m is the maximum depth of any node. Note that m can be much larger than d (the depth of the shallowest solution), and is infinite if the tree is unbounded.

Depth-limited search

The problem of unbounded trees can be alleviated by supplying depth-first search with a predetermined depth limit ℓ. That is, nodes at depth ℓ are treated as if they have no successors. This approach is called **depth-limited search**. The depth limit solves the infinite-path problem. Unfortunately, it also introduces an additional source of incompleteness if we choose $\ell < d$, that is, the shallowest goal is beyond the depth limit. (This is not unlikely when d is unknown.) Depth-limited search will also be nonoptimal if we choose $\ell > d$. Its time complexity is $O(b^\ell)$ and its space complexity is $O(b\ell)$. Depth-first search can be viewed as a special case of depth-limited search with $\ell = \infty$.

Sometimes, depth limits can be based on knowledge of the problem. For example, on the map of Romania there are 20 cities. Therefore, we know that if there is a solution, it must be of length 19 at the longest, so $\ell = 19$ is a possible choice. But in fact if we studied the map carefully, we would discover that any city can be reached from any other city in at most 9 steps. This number, known as the **diameter** of the state space, gives us a better depth limit, which leads to a more efficient depth-limited search. For most problems, however, we will not know a good depth limit until we have solved the problem.

Depth-limited search can be implemented as a simple modification to the general tree-search algorithm or to the recursive depth-first search algorithm. We show the pseudocode for recursive depth-limited search in Figure 3.13. Notice that depth-limited search can terminate with two kinds of failure: the standard *failure* value indicates no solution; the *cutoff* value indicates no solution within the depth limit.