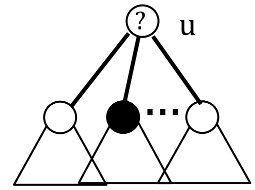
Analyzing depth-first search

David Gries

Here is procedure dfs1: a recursive version of depth-first search with no precondition:

```
/** Visit every node reachable along a path of unvisited nodes from node u. */
public static void dfs1(Node u) {
    if (u is visited) return;
    Visit u;
    for each neighbor w of u:
        dfs1(w);
}
```



When procedure dfs1 is called, u may —or may not— have already been visited. If u is already visited, no nodes are to be visited so the method returns immediately. On the other hand, if u is not visited, it is visited and then call dfs1(w) is executed for each neighbor w of u.

Analyzing execution time

Assume we are working with a directed graph with m nodes. Suppose that the initial call dfs1(u) requires n nodes to be visited and that, in total, these n nodes have e edges leaving them. As an aside if u is already visited, n and e are both 0. We determine the number of times each statement in dfs1 is executed.

First, the if-statement is executed at the beginning of the first call dfs1(u).

Second, the if-condition should be false exactly n times, because n unvisited nodes are to be visited, and if it is false, node u is immediately visited,. This means that Visit u; is executed n times.

Therefore, the for-each loop is executed n times, once for each node that is visited.

That means that a recursive call dfs1(w) is made for every neighbor of every one of the n nodes, which means a total of e times. But then the if-statement is executed another e times, once for each recursive call.

Since the if-condition is false n times, it is true 1+e-n times, so that the return statement is executed 1+e-n times.

Note that we are *not* describing the complexity of executing the for-each loop. We cannot do that unless we know how the graph is implemented. If the graph is implemented using an adjacency list, so that each list of out-going edges is in an array or a linked list, the time is proportional to the outdegree of the node —it could be the number of nodes in the graph.

Reducing the number of recursive calls.

This method makes e recursive calls, one for each edge leaving one of the nodes to be visited. If the graph is dense, this is *very* expensive –it could be quadratic in the number of nodes in the graph. That could be a lot bigger than n, the number of nodes to be visited.

To reduce the number of calls, call dfs1(w) only if w is not yet visited. The if-statement is then executed e times, but the recursive call only n-1 times in total. This means that the first if-statement is executed n times. It is false at most once —only if the first node is already visited.

```
/** Visit every node reachable along a path of unvisited nodes from node u. */
public static void dfs1(Node u) {
    if (u is visited)            1 + n - 1 times
        return;                  at most once
    Visit u;                     n times
    for each neighbor w of u:    n times (number of execution, not total number of iterations performed)
        if (w is not visited)    e times
            dfs1(w);             n-1 times
}
```

Conclusion

The if-statement in the loop body is not needed for correctness, but it is essential for execution speed if dfs1 is going to be used on large graphs.