A breadth-first walk of a graph

A breadth-first search of this graph starting at \( u \) visits the nodes in breadth-first order:

First \( u \). Then all nodes 1 edge from \( u \). Then all nodes 2 edges from \( u \). And so forth.

However, a person walking the graph as if it were a town with intersections and roads between them would not be able to jump around like that. Instead, the walker could only walk from one intersection (node) to another along a road (an edge).

We show Walker Red making such a walk to visit all nodes of the graph, visiting nodes in bfs order. It’s clear that a bfs walk is inefficient. Actually, a dfs walk is much better, and we now develop it.

The specification of the depth-first walk of a graph

We develop an implementation of a dfs walk, showing one way to organize it.

```java
public static void dfsWalk(State s) {
    Node u = s.standingOn();
    Visit u;
    for each neighbor w of u {
        if (w is unvisited) {
            s.moveTo(w);
            dfsWalk(s);
            s.moveTo(u);
        }
    }
}
```

The method body

We now develop the method body from the specification. First, let’s save the node on which the walker is standing in a local variable \( u \) and then visit \( u \), since the spec says it is unvisited.

Next, each unvisited neighbor \( w \) of \( u \) has to be processed—a recursive dfs has to be done on each unvisited neighbor. How to do that?

According to the spec of dfsWalk, the walker must be standing on the node, so the first step is to move to node \( w \). Now, dfsWalk can be called, using \( s \) as the argument. When that call terminates, according to the method spec, the walker will be standing on that same node \( w \). But the walker needs to be on node \( u \)—either to process the next neighbor or because the method is done. Therefore, we insert the move to \( u \). That completes the development.

/** The walker is standing on a Node \( u \) (say) given by State \( s \).
   Visit every node reachable along paths of unvisited nodes from node \( u \).
   End with walker standing on Node \( u \).
   Precondition: \( u \) is unvisited. */
public static void dfsWalk(State s) {
    Node u = s.standingOn();
    Visit u;
    for each neighbor w of u {
        if (w is unvisited) {
            s.moveTo(w);
            dfsWalk(s);
            s.moveTo(u);
        }
    }
}

©David Gries, 2018