Developing depth-first search
David Gries

We show you how to *develop* depth-first search from this specification —the method will use recursion:

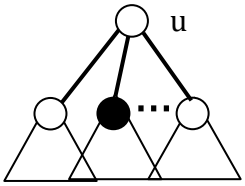
```
/** Visit every node reachable along a path of unvisited nodes from node u.
    Precondition: u is not visited. */
public static void dfs(Node u)
```

The same algorithm works for both directed and undirected graphs.

We write a very abstract version of this algorithm. For example, we don't look at class Node, and we don't describe how to maintain information about which nodes have been visited. We just allow ourselves to ask "if u is not yet visited" and to write "Visit u" to say that node u has now been visited. We also process the neighbors of node u using a loop like this:

```
for each neighbor w of u …
```

How to start

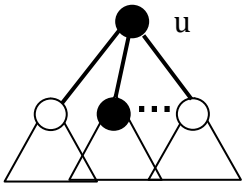


First, draw an outline of the graph (to the right), showing node u and its neighbors. Visited nodes are black and unvisited nodes are white. Thus, u is white, and since we don't know what n's neighbors are, we make two white and one black.

We draw each neighbor as a triangle, indicating that the neighbor may have neighbors, which may also have neighbors, etc. We overlap some triangles to remind ourselves that there could be edges between nodes of different triangles —this *is*, after all, a general graph. You don't *have* to draw the triangles overlapping, like this, but it does give us a good reminder that the triangles are not trees.

According to the specification, the nodes to be visited are on unvisited paths starting at u and going through its left (unvisited) neighbor or its right (unvisited) neighbor.

Writing the body of the method



We now write the body of the method. The specification says to visit every node reachable along a path of unvisited nodes from node u. It also says that u is unvisited. Looking at the graph outline, therefore, the first step is to:

```
Visit u;
```

But now, the unvisited neighbors of u are no longer reachable on an unvisited path starting at u. So instead, we have to,

(1) for each neighbor w of u, visit all nodes reachable along paths of unvisited nodes from w.

The similarity of this statement to the method spec indicates that a recursive call dfs(w) will do the job. Respecting the precondition that the parameter of a call on dfs is unvisited, we write:

```
for each neighbor w of u:
   if (w is not visited)  dfs(w);
```

And that's it! The method has been written.

Complete method

```
/** Visit every node reachable along a path of unvisited nodes from node u.
    Precondition: u is not visited. */
public static void dfs(Node u) {
   Visit u;
   for each neighbor w of u:
       if (w is not visited)  dfs(w);

}
```