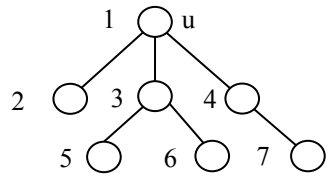
Breadth-first search
David Gries

Breadth-first search of a graph visits all nodes of a graph that are reachable along unvisited paths from node u in the following order:



First u.
Then all nodes that are 1 edge from u.
Then all nodes that are 2 edges from u,
And so forth.

Here is the iterative depth-first search algorithm that we developed earlier:

```
/** Visit every node reachable along a path of unvisited nodes from node u.
    Precondition: u has not been visited. */
public static void dfsIterative(Node u) {
    Stack s= (u); // Not Java!
    // Invariant: all nodes (and only those nodes) that have to be visited are
    // reachable along a path of unvisited nodes from some node in s.
    while (s is not empty) {
        u= s.pop();
        if (u is not visited) {
            Visit u;
            For each neighbor w of u:
                s.push(w);
        }
    }
}
```

We change it into a breadth-first search simply by changing s from a stack to a queue!

```
/** Visit every node reachable along a path of unvisited nodes from node u.
    Precondition: u has not been visited. */
public static void bfs(Node u) {
    Queue s= (u); // Not Java!
    // Invariant: all nodes (and only those nodes) that have to be visited are
    // reachable along a path of unvisited nodes from some node in s.
    while (s is not empty) {
        u= s.remove(); // remove first element of queue and store it in u
        if (u is not visited) {
            Visit u;
            For each neighbor w of u:
                s.add(w); // append w to queue
        }
    }
}
```

We explain why this results in a breadth-first search. First, for any integer $i \geq 0$, nodes that are i edges from u are put in the queue before nodes that are i+1 edges from u. Second, nodes are removed from the *front* of the queue and visited (if not yet visited), so those closer to u are visited first.

Reducing the potential maximum size of the queue

Note: The following is not part of the transcript of the video.

The queue can grow huge! Essentially, for each edge, a vertex is added to it at some point. We can reduce the maximum size of the queue by changing the statement s.add(w); to this:

```
if (w has never been in the queue) s.add(w);
```

If w was added to the queue before, there is no need to add w to the queue. We can show this by investigating two cases:

1. w is no longer in the queue. Therefore, it was already popped from the queue and visited, and it won't be visited again.
2. w is still in the queue. Then, adding w to the queue again has no effect, because the one in the queue will be processed first —removed from the queue and visited.

How do we implement "w has never been in the queue"? If nodes are numbered, then have a boolean array to maintain that information. If the nodes are not numbered but implemented in some other way, one can maintain the information "w has never been in the queue" the same way one implements "u is not visited". Often, it is implemented with a HashSet, so it's expected constant time to test whether w has been in the queue or not.