NonrecursiveDFS.java

Below is the syntax highlighted version of NonrecursiveDFS.java from §4.1 Undirected Graphs.

```
/******************************************************************************
 *  Compilation:  javac NonrecursiveDFS.java
 *  Execution:    java NonrecursiveDFS graph.txt s
 *  Dependencies: Graph.java Queue.java Stack.java StdOut.java
 *  Data files:   https://algs4.cs.princeton.edu/41graph/tinyCG.txt
 *                https://algs4.cs.princeton.edu/41graph/tinyG.txt
 *                https://algs4.cs.princeton.edu/41graph/mediumG.txt
 *
 *  Run nonrecurisve depth-first search on an undirected graph.
 *  Runs in O(E + V) time using O(V) extra space.
 *
 *  Explores the vertices in exactly the same order as DepthFirstSearch.java.
 *
 *  %  java Graph tinyG.txt
 *  13 vertices, 13 edges
 *  0: 6 2 1 5
 *  1: 0
 *  2: 0
 *  3: 5 4
 *  4: 5 6 3
 *  5: 3 4 0
 *  6: 0 4
 *  7: 8
 *  8: 7
 *  9: 11 10 12
 *  10: 9
 *  11: 9 12
 *  12: 11 9
 *
 *  % java NonrecursiveDFS tinyG.txt 0
 *  0 1 2 3 4 5 6
 *
 * % java NonrecursiveDFS tinyG.txt 9
 * 9 10 11 12
 *
 ******************************************************************************/

import java.util.Iterator;

/**
 *  The {@code NonrecursiveDFS} class represents a data type for finding
 *  the vertices connected to a source vertex <em>s</em> in the undirected
 *  graph.
 *  <p>
 *  This implementation uses a nonrecursive version of depth-first search
 *  with an explicit stack.
 *  See {@link DepthFirstSearch} for the classic recursive version.
 *  The constructor takes &Theta;(<em>V</em> + <em>E</em>) time in the worst
 *  case, where <em>V</em> is the number of vertices and <em>E</em> is the
 *  number of edges.
 *  The {@link #marked(int)} instance method takes &Theta;(1) time.
 *  It uses &Theta;(<em>V</em>) extra space (not including the graph).
 *  <p>
 *  For additional documentation,
 *  see <a href="https://algs4.cs.princeton.edu/41graph">Section 4.1</a>
```

```
 *  of <i>Algorithms, 4th Edition</i> by Robert Sedgewick and Kevin Wayne.
 *
 *  @author Robert Sedgewick
 *  @author Kevin Wayne
 */
public class NonrecursiveDFS {
    private boolean[] marked;  // marked[v] = is there an s-v path?
    /**
     * Computes the vertices connected to the source vertex {@code s} in the graph {@code G}.
     * @param G the graph
     * @param s the source vertex
     * @throws IllegalArgumentException unless {@code 0 <= s < V}
     */
    public NonrecursiveDFS(Graph G, int s) {
        marked = new boolean[G.V()];

        validateVertex(s);

        // to be able to iterate over each adjacency list, keeping track of which
        // vertex in each adjacency list needs to be explored next
        Iterator<Integer>[] adj = (Iterator<Integer>[]) new Iterator[G.V()];
        for (int v = 0; v < G.V(); v++)
            adj[v] = G.adj(v).iterator();

        // depth-first search using an explicit stack
        Stack<Integer> stack = new Stack<Integer>();
        marked[s] = true;
        stack.push(s);
        while (!stack.isEmpty()) {
            int v = stack.peek();
            if (adj[v].hasNext()) {
                int w = adj[v].next();
                // StdOut.printf("check %d\n", w);
                if (!marked[w]) {
                    // discovered vertex w for the first time
                    marked[w] = true;
                    // edgeTo[w] = v;
                    stack.push(w);
                    // StdOut.printf("dfs(%d)\n", w);
                }
            }
            else {
                // StdOut.printf("%d done\n", v);
                stack.pop();
            }
        }
    }

    /**
     * Is vertex {@code v} connected to the source vertex {@code s}?
     * @param v the vertex
     * @return {@code true} if vertex {@code v} is connected to the source vertex {@code s},
     *    and {@code false} otherwise
     * @throws IllegalArgumentException unless {@code 0 <= v < V}
     */
    public boolean marked(int v) {
        validateVertex(v);
        return marked[v];
    }

    // throw an IllegalArgumentException unless {@code 0 <= v < V}
    private void validateVertex(int v) {
        int V = marked.length;
        if (v < 0 || v >= V)
            throw new IllegalArgumentException("vertex " + v + " is not between 0 and " + (V-1));
```

```
    }

    /**
     * Unit tests the {@code NonrecursiveDFS} data type.
     *
     * @param args the command-line arguments
     */
    public static void main(String[] args) {
        In in = new In(args[0]);
        Graph G = new Graph(in);
        int s = Integer.parseInt(args[1]);
        NonrecursiveDFS dfs = new NonrecursiveDFS(G, s);
        for (int v = 0; v < G.V(); v++)
            if (dfs.marked(v))
                StdOut.print(v + " ");
        StdOut.println();
    }


}
```


Last updated: Thu Aug 11 09:22:35 EDT 2022.