BreadthFirstPaths.java

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

```
/******************************************************************************
 *  Compilation:  javac BreadthFirstPaths.java
 *  Execution:    java BreadthFirstPaths G 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
 *                https://algs4.cs.princeton.edu/41graph/largeG.txt
 *
 *  Run breadth first search on an undirected graph.
 *  Runs in O(E + V) time.
 *
 *  %  java Graph tinyCG.txt
 *  6 8
 *  0: 2 1 5
 *  1: 0 2
 *  2: 0 1 3 4
 *  3: 5 4 2
 *  4: 3 2
 *  5: 3 0
 *
 *  %  java BreadthFirstPaths tinyCG.txt 0
 *  0 to 0 (0):  0
 *  0 to 1 (1):  0-1
 *  0 to 2 (1):  0-2
 *  0 to 3 (2):  0-2-3
 *  0 to 4 (2):  0-2-4
 *  0 to 5 (1):  0-5
 *
 *  %  java BreadthFirstPaths largeG.txt 0
 *  0 to 0 (0):  0
 *  0 to 1 (418):  0-932942-474885-82707-879889-971961-...
 *  0 to 2 (323):  0-460790-53370-594358-780059-287921-...
 *  0 to 3 (168):  0-713461-75230-953125-568284-350405-...
 *  0 to 4 (144):  0-460790-53370-310931-440226-380102-...
 *  0 to 5 (566):  0-932942-474885-82707-879889-971961-...
 *  0 to 6 (349):  0-932942-474885-82707-879889-971961-...
 *
 ******************************************************************************/


/**
 *  The {@code BreadthFirstPaths} class represents a data type for finding
 *  shortest paths (number of edges) from a source vertex <em>s</em>
 *  (or a set of source vertices)
 *  to every other vertex in an undirected graph.
 *  <p>
 *  This implementation uses breadth-first search.
 *  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.
 *  Each instance method takes &Theta;(1) time.
 *  It uses &Theta;(<em>V</em>) extra space (not including the graph).
 *  <p>
 *  For additional documentation,
```

```java
 *  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 BreadthFirstPaths {
    private static final int INFINITY = Integer.MAX_VALUE;
    private boolean[] marked;  // marked[v] = is there an s-v path
    private int[] edgeTo;      // edgeTo[v] = previous edge on shortest s-v path
    private int[] distTo;      // distTo[v] = number of edges shortest s-v path

    /**
     * Computes the shortest path between the source vertex {@code s}
     * and every other vertex in the graph {@code G}.
     * @param G the graph
     * @param s the source vertex
     * @throws IllegalArgumentException unless {@code 0 <= s < V}
     */
    public BreadthFirstPaths(Graph G, int s) {
        marked = new boolean[G.V()];
        distTo = new int[G.V()];
        edgeTo = new int[G.V()];
        validateVertex(s);
        bfs(G, s);

        assert check(G, s);
    }

    /**
     * Computes the shortest path between any one of the source vertices in {@code sources}
     * and every other vertex in graph {@code G}.
     * @param G the graph
     * @param sources the source vertices
     * @throws IllegalArgumentException if {@code sources} is {@code null}
     * @throws IllegalArgumentException if {@code sources} contains no vertices
     * @throws IllegalArgumentException unless {@code 0 <= s < V} for each vertex
     *         {@code s} in {@code sources}
     */
    public BreadthFirstPaths(Graph G, Iterable<Integer> sources) {
        marked = new boolean[G.V()];
        distTo = new int[G.V()];
        edgeTo = new int[G.V()];
        for (int v = 0; v < G.V(); v++)
            distTo[v] = INFINITY;
        validateVertices(sources);
        bfs(G, sources);
    }


    // breadth-first search from a single source
    private void bfs(Graph G, int s) {
        Queue<Integer> q = new Queue<Integer>();
        for (int v = 0; v < G.V(); v++)
            distTo[v] = INFINITY;
        distTo[s] = 0;
        marked[s] = true;
        q.enqueue(s);

        while (!q.isEmpty()) {
            int v = q.dequeue();
            for (int w : G.adj(v)) {
                if (!marked[w]) {
                    edgeTo[w] = v;
                    distTo[w] = distTo[v] + 1;
                    marked[w] = true;
                    q.enqueue(w);
                }
            }
        }
    }

    // breadth-first search from multiple sources
    private void bfs(Graph G, Iterable<Integer> sources) {
        Queue<Integer> q = new Queue<Integer>();
        for (int s : sources) {
            marked[s] = true;
            distTo[s] = 0;
            q.enqueue(s);
        }
        while (!q.isEmpty()) {
            int v = q.dequeue();
            for (int w : G.adj(v)) {
                if (!marked[w]) {
                    edgeTo[w] = v;
                    distTo[w] = distTo[v] + 1;
                    marked[w] = true;
                    q.enqueue(w);
                }
            }
        }
    }

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

    /**
     * Returns the number of edges in a shortest path between the source vertex {@code s}
     * (or sources) and vertex {@code v}?
     * @param v the vertex
     * @return the number of edges in such a shortest path
     *         (or {@code Integer.MAX_VALUE} if there is no such path)
     * @throws IllegalArgumentException unless {@code 0 <= v < V}
     */
    public int distTo(int v) {
        validateVertex(v);
        return distTo[v];
    }

    /**
     * Returns a shortest path between the source vertex {@code s} (or sources)
     * and {@code v}, or {@code null} if no such path.
     * @param  v the vertex
     * @return the sequence of vertices on a shortest path, as an Iterable
     * @throws IllegalArgumentException unless {@code 0 <= v < V}
     */
    public Iterable<Integer> pathTo(int v) {
        validateVertex(v);
        if (!hasPathTo(v)) return null;
        Stack<Integer> path = new Stack<Integer>();
        int x;
        for (x = v; distTo[x] != 0; x = edgeTo[x])
```

```
            path.push(x);
        path.push(x);
        return path;
    }


    // check optimality conditions for single source
    private boolean check(Graph G, int s) {

        // check that the distance of s = 0
        if (distTo[s] != 0) {
            StdOut.println("distance of source " + s + " to itself = " + distTo[s]);
            return false;
        }

        // check that for each edge v-w dist[w] <= dist[v] + 1
        // provided v is reachable from s
        for (int v = 0; v < G.V(); v++) {
            for (int w : G.adj(v)) {
                if (hasPathTo(v) != hasPathTo(w)) {
                    StdOut.println("edge " + v + "-" + w);
                    StdOut.println("hasPathTo(" + v + ") = " + hasPathTo(v));
                    StdOut.println("hasPathTo(" + w + ") = " + hasPathTo(w));
                    return false;
                }
                if (hasPathTo(v) && (distTo[w] > distTo[v] + 1)) {
                    StdOut.println("edge " + v + "-" + w);
                    StdOut.println("distTo[" + v + "] = " + distTo[v]);
                    StdOut.println("distTo[" + w + "] = " + distTo[w]);
                    return false;
                }
            }
        }

        // check that v = edgeTo[w] satisfies distTo[w] = distTo[v] + 1
        // provided v is reachable from s
        for (int w = 0; w < G.V(); w++) {
            if (!hasPathTo(w) || w == s) continue;
            int v = edgeTo[w];
            if (distTo[w] != distTo[v] + 1) {
                StdOut.println("shortest path edge " + v + "-" + w);
                StdOut.println("distTo[" + v + "] = " + distTo[v]);
                StdOut.println("distTo[" + w + "] = " + distTo[w]);
                return false;
            }
        }

        return true;
    }

    // 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));
    }

    // throw an IllegalArgumentException if vertices is null, has zero vertices,
    // or has a vertex not between 0 and V-1
    private void validateVertices(Iterable<Integer> vertices) {
        if (vertices == null) {
            throw new IllegalArgumentException("argument is null");
        }
        int vertexCount = 0;
        for (Integer v : vertices) {
            vertexCount++;
            if (v == null) {
                throw new IllegalArgumentException("vertex is null");
            }
            validateVertex(v);
        }
        if (vertexCount == 0) {
            throw new IllegalArgumentException("zero vertices");
        }
    }

    /**
     * Unit tests the {@code BreadthFirstPaths} 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);
        // StdOut.println(G);

        int s = Integer.parseInt(args[1]);
        BreadthFirstPaths bfs = new BreadthFirstPaths(G, s);

        for (int v = 0; v < G.V(); v++) {
            if (bfs.hasPathTo(v)) {
                StdOut.printf("%d to %d (%d):  ", s, v, bfs.distTo(v));
                for (int x : bfs.pathTo(v)) {
                    if (x == s) StdOut.print(x);
                    else        StdOut.print("-" + x);
                }
                StdOut.println();
            }

            else {
                StdOut.printf("%d to %d (-):  not connected\n", s, v);
            }

        }
    }


}
```

Last updated: Sun Nov 27 05:35:48 EST 2022.