Bipartite.java

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

```
/******************************************************************************
 *  Compilation:  javac Bipartite.java
 *  Execution:    java  Bipartite V E F
 *  Dependencies: Graph.java
 *  Data files:   https://algs4.cs.princeton.edu/41graph/tinyG.txt
 *                https://algs4.cs.princeton.edu/41graph/mediumG.txt
 *                https://algs4.cs.princeton.edu/41graph/largeG.txt
 *
 *  Given a graph, find either (i) a bipartition or (ii) an odd-length cycle.
 *  Runs in O(E + V) time.
 *
 ******************************************************************************/


/**
 *  The {@code Bipartite} class represents a data type for
 *  determining whether an undirected graph is <em>bipartite</em> or whether
 *  it has an <em>odd-length cycle</em>.
 *  A graph is bipartite if and only if it has no odd-length cycle.
 *  The <em>isBipartite</em> operation determines whether the graph is
 *  bipartite. If so, the <em>color</em> operation determines a
 *  bipartition; if not, the <em>oddCycle</em> operation determines a
 *  cycle with an odd number of edges.
 *  <p>
 *  This implementation uses <em>depth-first search</em>.
 *  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).
 *  See {@link BipartiteX} for a nonrecursive version that uses breadth-first
 *  search.
 *  <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 Bipartite {
    private boolean isBipartite;   // is the graph bipartite?
    private boolean[] color;       // color[v] gives vertices on one side of bipartition
    private boolean[] marked;      // marked[v] = true iff v has been visited in DFS
    private int[] edgeTo;          // edgeTo[v] = last edge on path to v
    private Stack<Integer> cycle;  // odd-length cycle

    /**
     * Determines whether an undirected graph is bipartite and finds either a
     * bipartition or an odd-length cycle.
     *
     * @param  G the graph
     */
    public Bipartite(Graph G) {
        isBipartite = true;
        color  = new boolean[G.V()];
        marked = new boolean[G.V()];
        edgeTo = new int[G.V()];

        for (int v = 0; v < G.V(); v++) {
            if (!marked[v]) {
                dfs(G, v);
```

```
        }
    }
    assert check(G);
}

private void dfs(Graph G, int v) {
    marked[v] = true;
    for (int w : G.adj(v)) {

        // short circuit if odd-length cycle found
        if (cycle != null) return;

        // found uncolored vertex, so recur
        if (!marked[w]) {
            edgeTo[w] = v;
            color[w] = !color[v];
            dfs(G, w);
        }

        // if v-w create an odd-length cycle, find it
        else if (color[w] == color[v]) {
            isBipartite = false;
            cycle = new Stack<Integer>();
            cycle.push(w);  // don't need this unless you want to include start vertex twice
            for (int x = v; x != w; x = edgeTo[x]) {
                cycle.push(x);
            }
            cycle.push(w);
        }
    }
}

/**
 * Returns true if the graph is bipartite.
 *
 * @return {@code true} if the graph is bipartite; {@code false} otherwise
 */
public boolean isBipartite() {
    return isBipartite;
}

/**
 * Returns the side of the bipartite that vertex {@code v} is on.
 *
 * @param  v the vertex
 * @return the side of the bipartition that vertex {@code v} is on; two vertices
 *         are in the same side of the bipartition if and only if they have the
 *         same color
 * @throws IllegalArgumentException unless {@code 0 <= v < V}
 * @throws UnsupportedOperationException if this method is called when the graph
 *         is not bipartite
 */
public boolean color(int v) {
    validateVertex(v);
    if (!isBipartite)
        throw new UnsupportedOperationException("graph is not bipartite");
    return color[v];
}

/**
 * Returns an odd-length cycle if the graph is not bipartite, and
 * {@code null} otherwise.
 *
 * @return an odd-length cycle if the graph is not bipartite
 *         (and hence has an odd-length cycle), and {@code null}
 *         otherwise
 */
public Iterable<Integer> oddCycle() {
    return cycle;
}

private boolean check(Graph G) {
```

```
    // graph is bipartite
    if (isBipartite) {
        for (int v = 0; v < G.V(); v++) {
            for (int w : G.adj(v)) {
                if (color[v] == color[w]) {
                    System.err.printf("edge %d-%d with %d and %d in same side of bipartition\n", v, w, v, w);
                    return false;
                }
            }
        }
    }

    // graph has an odd-length cycle
    else {
        // verify cycle
        int first = -1, last = -1;
        for (int v : oddCycle()) {
            if (first == -1) first = v;
            last = v;
        }
        if (first != last) {
            System.err.printf("cycle begins with %d and ends with %d\n", first, last);
            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));
}

/**
 * Unit tests the {@code Bipartite} data type.
 *
 * @param args the command-line arguments
 */
public static void main(String[] args) {
    int V1 = Integer.parseInt(args[0]);
    int V2 = Integer.parseInt(args[1]);
    int E  = Integer.parseInt(args[2]);
    int F  = Integer.parseInt(args[3]);

    // create random bipartite graph with V1 vertices on left side,
    // V2 vertices on right side, and E edges; then add F random edges
    Graph G = GraphGenerator.bipartite(V1, V2, E);
    for (int i = 0; i < F; i++) {
        int v = StdRandom.uniformInt(V1 + V2);
        int w = StdRandom.uniformInt(V1 + V2);
        G.addEdge(v, w);
    }

    StdOut.println(G);


    Bipartite b = new Bipartite(G);
    if (b.isBipartite()) {
        StdOut.println("Graph is bipartite");
        for (int v = 0; v < G.V(); v++) {
            StdOut.println(v + ": " + b.color(v));
        }
    }
    else {
        StdOut.print("Graph has an odd-length cycle: ");
        for (int x : b.oddCycle()) {
            StdOut.print(x + " ");
        }
        StdOut.println();
```

```
        }
    }

}
```


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