```
import java.io.File;

/**
 * ColorableGraph class
 * @author Caroline
 *
 */
public class ColorableGraph {

    private File file;//file to get vertex numbers and edges from
    private int numVert;//total number of vertices existing in the graph
    private Scanner scan;//scanner to read file from
    private VertexObj[] graph;//array of VertexObj (vertices) that creates a graph
    private int[] errorSpot;//intended to use for backtracking and inserting vertices of the cycle with odd numbers

    /**
     * takes a file as an input
     * @param f
     */
    public ColorableGraph(File f){
        file = f;
        try {
            scan = new Scanner(file);//scanner to iterate through the file
            if(scan.hasNextInt()){
                numVert = scan.nextInt();//grabs first number in the file which is the total number of vertices in the graph
            }
            graph = new VertexObj[numVert];
            errorSpot = new int[numVert];
            for(int i=0; i<graph.length; i++){//initializing graph with unique vertex id
                graph[i] = new VertexObj(i+1, numVert);
            }
            while(scan.hasNextInt()){//reading edges from the file and setting the edge between the "from" and "to" vertices
                int from = scan.nextInt();
                int to = scan.nextInt();
                graph[from-1].setEdge(to);
            }

        } catch (FileNotFoundException e) {//throws exception if the file is not found
            e.printStackTrace();
        }
    }

    /**
     * uses breadth first search in order to see if this graph is 2-colorable
     * @return
     */
    public boolean colorable(){
        LinkedListQueue queue = new LinkedListQueue();//queue for breadth first search
        queue.enqueue(graph[0]);//enqueues the starting point (1)
        String currColor = "red";
        LinkedListQueue backtrack = new LinkedListQueue();//intended to use to backtrack and print out cycle but couldn't
        while(!queue.isEmpty()){
            VertexObj u = (VertexObj) queue.peek();
            if(graph[u.getId()-1].getColor().equals("white")){
                graph[u.getId()-1].setColor(currColor);//sets the starting point to the color "red"
            }
```

```java
            currColor = u.getColor();//changes current color to whatever the color of current vertex is
            if(graph[u.getId()-1].getDone() == true) {//checks if this vertex has been visited already
                queue.dequeue();
                continue;
            }
            int[] adj = u.getEdge();//gets the adjacency list of the current vertex
            for(int i=0; i<adj.length; i++){//loop to check every vertex in the adjacency list
                if(adj[i]!=0){//ignore 0 because that means the adjacency list is null at that point
                    if(graph[adj[i]-1].getDone() == true) continue;//if this adjacent vertex has been checked already, move to next one
                    String diffColor;
                    if(currColor.equals("red"))//changes "diffColor" according to the current "currColor"
                        diffColor = "blue";
                    else
                        diffColor = "red";
                    if (graph[adj[i]-1].getColor().equals("white")){//if the adjacent vertex hasn't been colored yet, color it with "diffColor"
                        graph[adj[i]-1].setColor(diffColor);
                        queue.enqueue(graph[adj[i]-1]);//enqueue to the queue
                    }
                    else{
                        if(graph[adj[i]-1].getColor().equals(currColor)){//if the adjacent vertex already has a color which is the same as the color of current vertex
                            System.out.println("No");//print out no to indicate this is not a 2-colorable graph
                            System.out.println(adj[i]);//print the adjacent vertex
                            System.out.println(u.getId());//print the current vertex
                            return false;
                        }
                    }

                }
            }
            graph[u.getId()-1].setDone();//set done so that this vertex has been visited
            queue.dequeue();//dequeue this vertex from the queue


        }
        System.out.println("yes");//if false is not returned during this method, the graph is 2-colorable
        for(int i=0; i<graph.length; i++){
            System.out.println(graph[i].getId() + graph[i].getColor());//print out all the vertex numbers and colors
        }
        return true;
    }



    public static void main(String[] args) {
        System.out.println("Type in file path:");//ask for file path
        Scanner s = new Scanner(System.in);//scanner receives file path
        File file = new File(s.nextLine());//turns the file path into actual file
        ColorableGraph cgraph = new ColorableGraph(file);//make ColorableGraph object with the file received
        cgraph.colorable();//run colorable method to check if it is colorable
    }

}
```

```
/**
 * VertexObj class that has information about one vertex of the graph
 * @author Caroline
 *
 */
public class VertexObj {
    private int vertId; //number that the vertex is associated with
    private int numVert;//number of vertices in total in the graph
    private int[] edgeArr;//int array that contains the number of vertices this vertex has edge to
    private boolean done;//boolean to check if this vertex has been visited already
    private String color;//color of this vertex (white, red, blue)

    public VertexObj(int id, int numV){
        vertId = id;
        numVert = numV;
        edgeArr = new int[numVert];
        done = false;
        color = "white";
    }

    /**
     * returns vertex ID
     * @return
     */
    public int getId(){
        return vertId;
    }

    /**
     * receives a number of vertex that this vertex has an edge to
     * and adds it to the array of vertices(edgeArr) that this vertex has an edge to
     * @param e
     */
    public void setEdge(int e){
        for(int i=0; i<edgeArr.length; i++){
            if(edgeArr[i] == 0){
                edgeArr[i] = e;
                break;
            }
            else{
                continue;
            }
        }
    }

    /**
     * returns the edgeArr
     * @return
     */
    public int[] getEdge(){
        return edgeArr;
    }

    /**
     * changes done variable to true to indicate that this vertex has been visited
     * and all of its adjacent vertices has been checked
     */
    public void setDone(){
        done = true;
    }

    /**
     * returns done boolean
     * @return
     */
    public boolean getDone(){
        return done;
    }

    /**
     * sets the color of this vertex to another color
     * @param c
     */
    public void setColor(String c){
        color = c;
    }

    /**
     * returns the color of this vertex
     * @return
     */
    public String getColor(){
        return color;
    }
}
```

```
/**
 * Queue data structure implemented with linked list
 * @author Caroline
 *
 */
public class LinkedListQueue {

    /**
     * Node that will make linked list when connected together
     * @author Caroline
     *
     * @param <T>This node will take any type of objects
     */
    private static class Node<T>{
        private T data;
        private Node next;

        public Node(T n){
            data = n;
        }
    }

    private Node head;
    private Node tail;

    /**
     * checks if the queue is empty (head is empty)
     * @return
     */
    public boolean isEmpty(){
        if(head == null) return true;
        else return false;
    }

    /**
     * adds a node with data to the end of the queue
     * @param data
     */
    public <T>void enqueue(T data){
        Node<T> n = new Node(data);
        if(isEmpty()){
            n.next = head;
            head = n;
            tail = n;
        }else{
            tail.next = n;
            tail = n;
            tail.next = null;
        }
    }

    /**
     * removes the first node in the queue
     */
    public void dequeue(){
        if(head.next == null){
            tail = null;
            head = null;
        }else{
            head = head.next;
        }
    }

    /**
     * returns the data of the first node in the queue
     * @return
     */
    public <T>Object peek(){
        return head.data;
    }

}
```

Description:

When this algorithm runs, the user is asked to put in the file path. With that, the graph gets initialized or throws an exception if the file is not found. My algorithm uses an array of VertexObj. VertexObj is a vertex and the class contains the number that the vertex is called, an array of integers with vertex id that this vertex has an edge to, Boolean called “done” that is an indicator of if every adjacent vertex of this vertex has been visited or not, and the color of this vertex initially set as “white”. It also uses an LinkedListQueue which is a queue data structure that is implemented with Linked List. In the class ColorableGraph, a file is taken in from scanner and puts the data into the array VertexObj called “graph”. In the method called “colorable”, a breadth first search is done on the “graph” array. In the breadth first search, the color is changed to each vertex according to the rule. When a vertex that is adjacent to current vertex is the same color as the current vertex, it prints out “No” and the number of the current vertex and the adjacent vertex. (This should backtrack and print out all the vertices from the cycle, but I couldn’t implement that part).
If none of this happens, it will print out “Yes” and the color of all the vertices in the graph.

I am aware of how this code is supposed to create an output file instead of just printing it out in the console. But because of the way I implemented the code, it can only run with small files. Or there will not be enough memory to run the file. So, I ended up just writing the code so that it prints the result out in the console.

Correctness:

This algorithm works because the breadth first search works. This is because through breadth first search, we can eliminate the back edges. Doing this and making the graph into a tree will make is so that if the vertices in the same level ae connected, it creates a cycle of odd numbers. This will automatically make the graph not colorable since you cannot have same colors in the adjacent vertices. Therefore, using breadth first search makes this algorithm work.

Time Analysis:

Because I used the breadth first search, the time analysis is the same as the breadth first search. For every vertex in the graph, it makes the array of adjacency list and checks those vertices. So, it is $\theta(1)$ per vertex and edge, and in total $\theta(V+E)$. If I could make this algorithm work the way I wanted it to and have it print out the cycle when it finds out that the graph is not colorable, it would have taken more time because it would have had to backtrack and use another data structure.

Even though the search part was $\theta(V+E)$, the initializing part of the code took more time. It made an array for graph which made an array for all the edges. So, there was extra $\theta(VE)$ in the process.

Timing:

Smallgraph:

Real: 0m3.188s

User: 0m0.164s

Sys: 0m0m032s

LargeGraph1:

Exception was thrown because of out of memory error

Real: 0m 22.116s

User: 2m 13.212s

Sys: 0m52.080s

LargeGraph2:

Exception was thrown because of out of memory error

Real: 0m 31.808s

User: 2m 5.772s

Sys: 0m55.352s

These times were tested on EDLAB, and I used my laptop HP Envy for it.

Outputs:

For smallgraph, the graph was colorable (YES).

The output was:

```
yes
1red
2blue
3red
4red
5blue
6red
7blue
8blue
9red
10blue
```

Because this graph was small enough, I could try this by hand and could confirm that this was the correct result. But unfortunately, my code didn't run for largegraph 1 and 2. My code used up too much memory when testing with large files and couldn't get the outputs on it.

EDLAB:

In /courses/cs300/cs311/kyuminkim/CS311_Graph there is the jar file that is executable. The java file of the code is in the src folder in the same place.