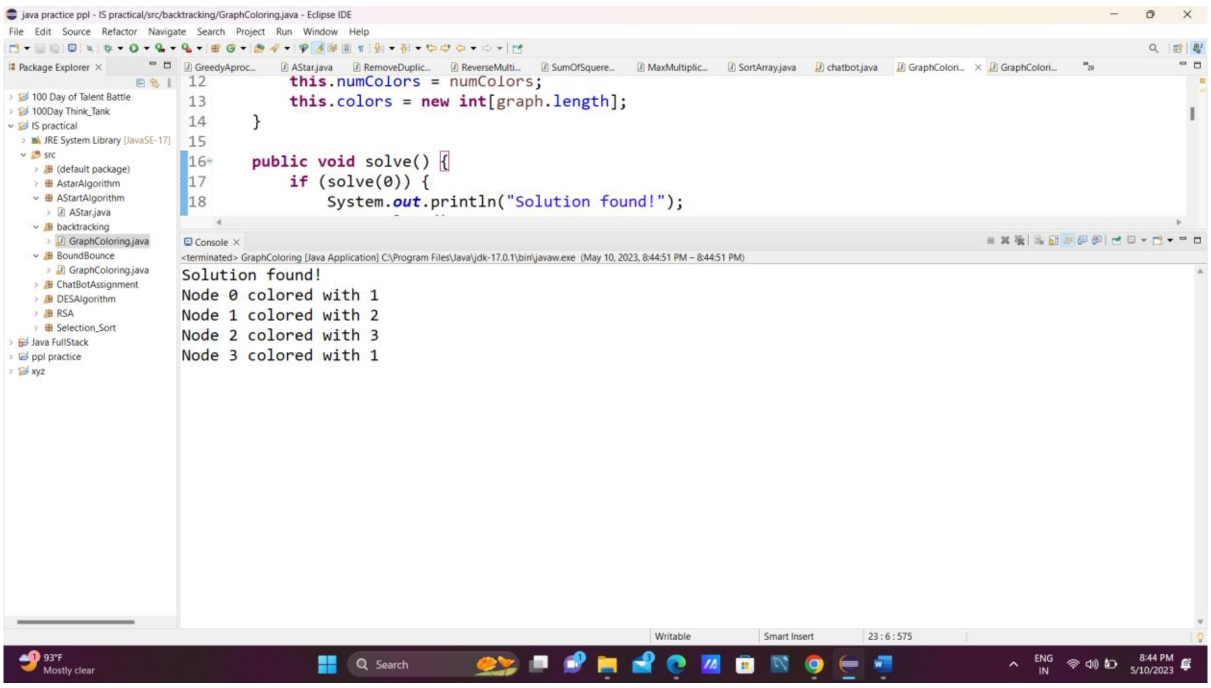
Name: Rameshwar Gosavi
Roll No.: 331
Batch: B

Title: Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.

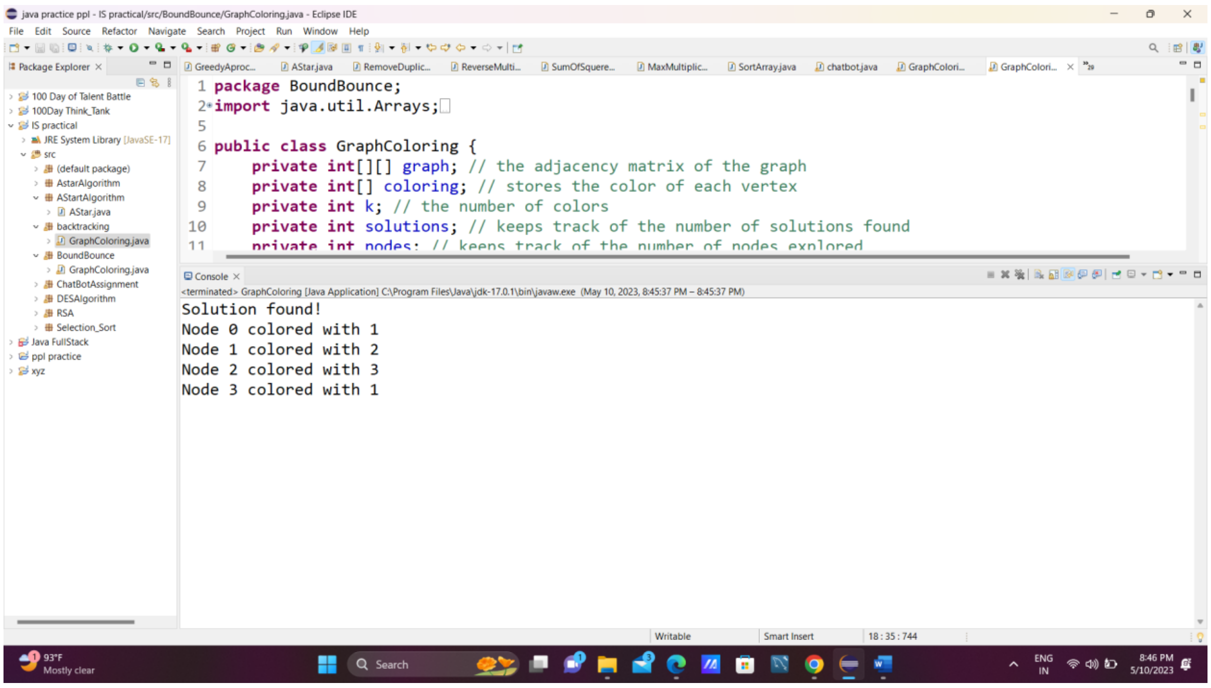
```java
package backtracking;
import java.util.*;

public class GraphColoring {

    private int[][] graph;
    private int[] colors;
    private int numColors;

    public GraphColoring(int[][] graph, int numColors) {
        this.graph = graph;
        this.numColors = numColors;
        this.colors = new int[graph.length];
    }

    public void solve() {
        if (solve(0)) {
            System.out.println("Solution found!");
            printColors();
        } else {
            System.out.println("No solution found.");
        }
    }

    private boolean solve(int node) {
        if (node == graph.length) {
            return true;
        }

        for (int color = 1; color <= numColors; color++) {
            if (isColorValid(node, color)) {
                colors[node] = color;
                if (solve(node + 1)) {
                    return true;
                }
                colors[node] = 0;
            }
        }

        return false;
    }
    private boolean isColorValid(int node, int color) {
        for (int i = 0; i < graph.length; i++) {
            if (graph[node][i] == 1 && colors[i] == color) {
                return false;
            }
        }
        return true;
    }

    private void printColors() {
        for (int i = 0; i < colors.length; i++) {
            System.out.println("Node " + i + " colored with " +
colors[i]);
        }
    }

    public static void main(String[] args) {
        int[][] graph = {{0, 1, 1, 0},
                         {1, 0, 1, 1},
                         {1, 1, 0, 1},
                         {0, 1, 1, 0}};
        int numColors = 3;

        GraphColoring gc = new GraphColoring(graph, numColors);
        gc.solve();
    }
}
```


java practice ppl - IS practical/src/backtracking/GraphColoring.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Console

```java
        this.numColors = numColors;
        this.colors = new int[graph.length];
    }

    public void solve() {
        if (solve(0)) {
            System.out.println("Solution found!");
```

```
<terminated> GraphColoring [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (May 10, 2023, 8:44:51 PM – 8:44:51 PM)
Solution found!
Node 0 colored with 1
Node 1 colored with 2
Node 2 colored with 3
Node 3 colored with 1
```

```
package BoundBounce;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

public class GraphColoring {
    private int[][] graph; // the adjacency matrix of the graph
    private int[] coloring; // stores the color of each vertex
    private int k; // the number of colors
    private int solutions; // keeps track of the number of solutions
found
    private int nodes; // keeps track of the number of nodes
explored
    private Set<Integer>[] usedColors; // used to mark which colors
are already used by neighboring vertices

    public GraphColoring(int[][] graph, int k) {
        this.graph = graph;
        this.k = k;
        this.coloring = new int[graph.length];
        Arrays.fill(coloring, -1);
        this.usedColors = new HashSet[graph.length];
        for (int i = 0; i < graph.length; i++) {
            usedColors[i] = new HashSet<Integer>();
        }
    }

    // Returns an upper bound on the number of solutions for the
remaining vertices
    private int upperBound(int v) {
        int remaining = graph.length - v;
        int used = 0;
        for (int i = 0; i < v; i++) {
            if (graph[v][i] == 1) {
                used |= (1 << coloring[i]); // mark the used colors
            }
        }
        int free = ~(used | ((1 << k) - 1)); // colors that are
still free
        int count = 0;
        while (free != 0) {
            int color = Integer.numberOfTrailingZeros(free); //
choose the first free color
            free ^= (1 << color); // mark the color as used
            if (v + 1 < graph.length) {
                count += upperBound(v + 1);
            } else {
                count++;
            }
        }
        return count;
    }

    // Colors vertices starting from the specified vertex
    private void colorVertices(int v) {
        if (v == graph.length) {
            solutions++;
            return;
        }
        nodes++;
        int freeColors = ~(1 << k) & ~(1 << coloring[v]);
        for (int color = 0; color < k; color++) {
            if ((freeColors & (1 << color)) != 0 &&
!usedColors[v].contains(color)) {
                coloring[v] = color;
                for (int i = 0; i < graph.length; i++) {
                    if (graph[v][i] == 1) {
                        usedColors[i].add(color);
                    }
                }
                if (solutions < upperBound(v)) {
                    colorVertices(v + 1); // explore this branch
                }
                for (int i = 0; i < graph.length; i++) {
                    if (graph[v][i] == 1) {
                        usedColors[i].remove(color);
                    }
                }
                coloring[v] = -1;
            }
        }
    }

    // Prints the number of solutions found and the number of nodes
explored
    public void printStats() {
        System.out.println("Solutions: " + solutions);
        System.out.println("Nodes explored: " + nodes);
    }
    public class Main {
        public static void main(String[] args) {
            int[][] graph = {
                {0, 1, 0, 1},
                {1, 0, 1, 0},
                {0, 1, 0, 1},
                {1, 0, 1, 0}
            };
            int k = 2;
            GraphColoring gc = new GraphColoring(graph, k);
            gc.colorVertices(0);
            gc.printStats();
        }
    }

}
```



```
package BoundBounce;
import java.util.Arrays;

public class GraphColoring {
    private int[][] graph; // the adjacency matrix of the graph
    private int[] coloring; // stores the color of each vertex
    private int k; // the number of colors
    private int solutions; // keeps track of the number of solutions found
    private int nodes; // keeps track of the number of nodes explored
```

```
<terminated> GraphColoring [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (May 10, 2023, 8:45:37 PM – 8:45:37 PM)
Solution found!
Node 0 colored with 1
Node 1 colored with 2
Node 2 colored with 3
Node 3 colored with 1
```