**Graph Colouring Solver**

|  |
| --- |
| public class Graph {  /\*\*  \* Creates a new <code>Graph</code> instance.  \* These are random instances with n nodes and  \* an edge probability of 0.5  \*/  public Graph(int n, double p) {  neighbours = new ArrayList<List<Integer> >(n);  for (int i = 0; i < n; i++) {  neighbours.add(new ArrayList<Integer>());  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < i; j++) {  if (Math.random()<p) {  neighbours.get(i).add(j);  }  }  }  }  public static void main(String[] args) {  Graph graph = new Graph(15, 0.5);  Colouring colouring = graph.bestColouring(3);  graph.show(colouring);  }    public int degree(int i) {  return neighbours.get(i).size();  }  public int neighbour(int i, int j) {  return neighbours.get(i).get(j);  }  public int size() { return neighbours.size();}  public Colouring bestColouring(int k) {  int[] colouring = new int[size()];  for (int i = 0; i < size(); i++) {  colouring[i] = -1;  }  Colouring best\_colouring = new Colouring(size(), k);  int[] partial\_cost = new int[size()];  int current = 0;  partial\_cost[0]=0;  while (true) {  if (colouring[current]<k-1) {  colouring[current] = colouring[current]+1;  if (current!=0) {  partial\_cost[current] = partial\_cost[current-1];  for (int neigh: neighbours.get(current)) {  if (colouring[current]==colouring[neigh]) {  partial\_cost[current]++;  }  }  }  if (current==size()-1) {  if (partial\_cost[current]<best\_colouring.cost()) {  best\_colouring = new Colouring(colouring,  partial\_cost[current], k);  }  } else {  current++;  }  } else {  if (current==0) {  return best\_colouring;  }  colouring[current] = -1;  current--;  }  }  }  public void show(Colouring colouring) {  Color[] colour = {Color.red, Color.yellow, Color.blue, Color.green,  Color.black, Color.orange, Color.magenta, Color.cyan,  Color.gray, Color.pink};  System.out.println("Number of colour conflicts = "  + colouring.cost());  if (colouring.no\_colours()>colour.length) {  return;  }  int cnt = 0;  GraphDisplay gd = new GraphDisplay();  gd.showInWindow(400,400,"Best Colouring");  for (int k = 0; k < colouring.no\_colours() ; k++) {  for (int i = 0; i < size(); i++) {  if (colouring.get(i)==k) {  double angle = 2\*cnt\*Math.PI/size();  gd.addNode(i, Math.sin(angle), Math.cos(angle), colour[k]);  cnt++;  }  }  }  for (int i = 1; i < size(); i++) {  for (int j: neighbours.get(i)) {  gd.addEdge(i,j);  }  }  }    private List<List<Integer> > neighbours;  } |

Graph.java

|  |
| --- |
| class GraphDisplay extends JComponent  {  public GraphDisplay()  {  minX = minY = Double.POSITIVE\_INFINITY;  maxX = maxY = Double.NEGATIVE\_INFINITY;  }    public synchronized void addNode(Object identifier, double x, double y,  Color col)  {  maxX = Math.max(maxX,x);  maxY = Math.max(maxY,y);  minX = Math.min(minX,x);  minY = Math.min(minY,y);  nodes.put(identifier,new Node(x,y,col));  repaint();  }  public synchronized void addNode(Object identifier, double x, double y)  {  maxX = Math.max(maxX,x);  maxY = Math.max(maxY,y);  minX = Math.min(minX,x);  minY = Math.min(minY,y);  nodes.put(identifier,new Node(x,y,NODE\_COLOR));  repaint();  }    public synchronized void addEdge(Object start, Object end, Color c)  {  removeEdge(start,end);  edges.add(new Edge(start,end,c));  repaint();  }    public synchronized boolean removeEdge(Object start, Object end)  {  Iterator<Edge> it = edges.iterator();  while(it.hasNext())  {  Edge tmp = it.next();  if(tmp.joins(start,end))  {  it.remove();  repaint();  return true;  }  }  return false;  }    public void addEdge(Object start, Object end)  {  addEdge(start,end,Color.black);  }      public JFrame showInWindow(int width, int height, String title)  {  JFrame f = new JFrame();  f.add(this);  f.setSize(width,height);  f.setTitle(title);  f.addWindowListener(new WindowAdapter() {  public void windowClosing(WindowEvent we) {  System.exit(0);  }  });  f.setVisible(true);  return f;  }  public void paint(Graphics g)  {  if(nodes.isEmpty())  return;    double xscl = (getSize().width -2\*MARGIN) / (maxX-minX);  double yscl = (getSize().height-2\*MARGIN) / (maxY-minY);      g.translate(+MARGIN,+MARGIN);    synchronized(this){  for(Edge e: edges)  e.paint(g,xscl,yscl,minX,minY);  for(Node n: nodes.values())  n.paint(g,xscl,yscl,minX,minY);  }    g.translate(-MARGIN,-MARGIN);  }        protected double minX,maxX,minY,maxY;  protected HashMap<Object,Node> nodes = new HashMap<Object,Node>();  protected Vector<Edge> edges = new Vector<Edge>();    protected int MARGIN = 20;  protected int NODE\_RADIUS = 5;  protected Color NODE\_COLOR = Color.blue.brighter();    private class Node  {  public Node(double x, double y, Color col)  {  this.x = x;  this.y = y;  this.col = col;  }    public void paint(Graphics g, double xscl, double yscl, double tx, double ty)  {  g.setColor(col);  g.fillOval(  (int)((x-tx)\*xscl - NODE\_RADIUS),  (int)((y-ty)\*yscl - NODE\_RADIUS),  2\*NODE\_RADIUS,  2\*NODE\_RADIUS  );  }    protected double x,y;  protected Color col;    }    private class Edge  {  public Edge(Object start, Object end, Color col)  {  this.start = start;  this.end = end;  this.col = col;  }    public boolean joins(Object a, Object b)  {  return (start.equals(a) && end.equals(b))  || (start.equals(b) && end.equals(a));  }    public void paint(Graphics g, double xscl, double yscl, double tx, double ty)  {  Node a = nodes.get(start);  Node b = nodes.get(end);  g.setColor(col);  g.drawLine(  (int)(xscl\*(a.x-tx)),  (int)(yscl\*(a.y-ty)),  (int)(xscl\*(b.x-tx)),  (int)(yscl\*(b.y-ty))  );  }    protected Object start,end;  protected Color col;  }  } |

GraphDisplay.java

|  |
| --- |
| import java.util.\*;  import java.io.\*;  /\*\*  \* Describe class Colouring here.  \*  \*  \* Created: Tue Jan 27 16:06:57 2009  \*  \* @author <a href="mailto:apb@apb-desktop">Adam Prugel-Bennett</a>  \* @version 1.0  \*/  public class Colouring {  public Colouring(int n, int maxcol) {  colours = new int[n];  for (int i = 0; i < n; i++) {  colours[i] = -1;  }  c = 100000;  k = maxcol;  }  public Colouring(Colouring colouring) {  colours = new int[colouring.size()];  System.arraycopy(colouring.colours, 0, colours, 0, colouring.size());  c = colouring.cost();  k = colouring.no\_colours();  }  public Colouring(int[] cols, int the\_cost, int maxcol) {  colours = new int[cols.length];  System.arraycopy(cols, 0, colours, 0, cols.length);  c = the\_cost;  k = maxcol;    }  public int get(int node) {return colours[node];}  public void set(int node, int col) {colours[node]=col;}  public int cost() {return c;}  public void set\_cost(int cc) {c=cc;}  public int no\_colours() {return k;}  public int size() {return colours.length;}  private int[] colours;  private int c;  int k;  } |

Colouring.java