sehes333@stud.fci-cu.edu.eg

Introduction:

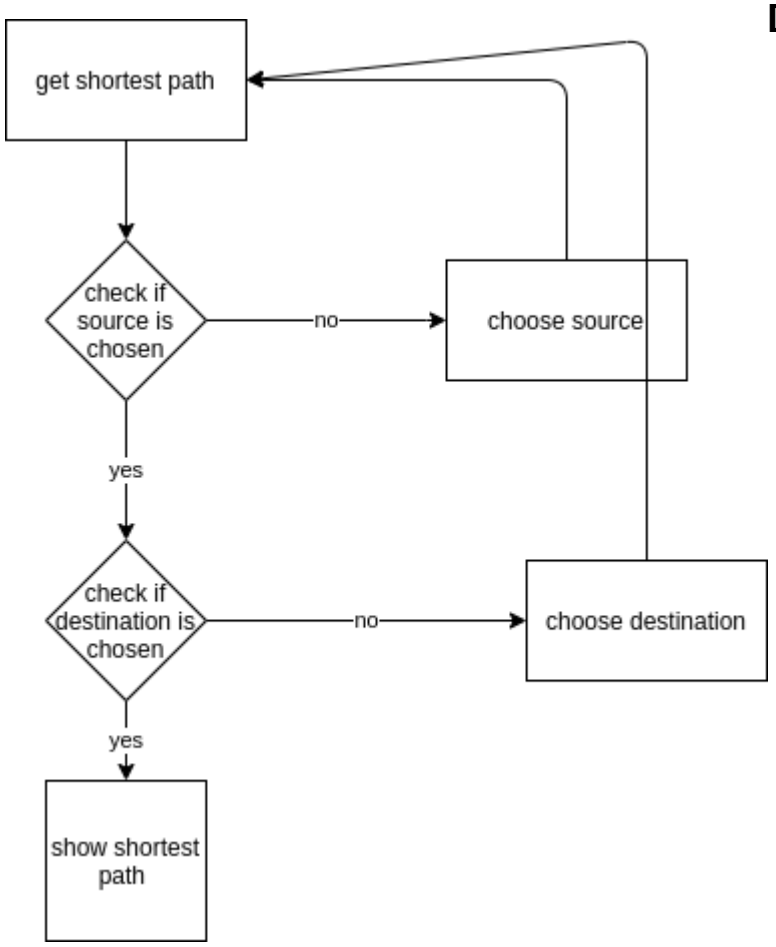
this system is to help you to easily calculate the maximum flow of a graph between two nodes and the shortest path between them

Visibility study:

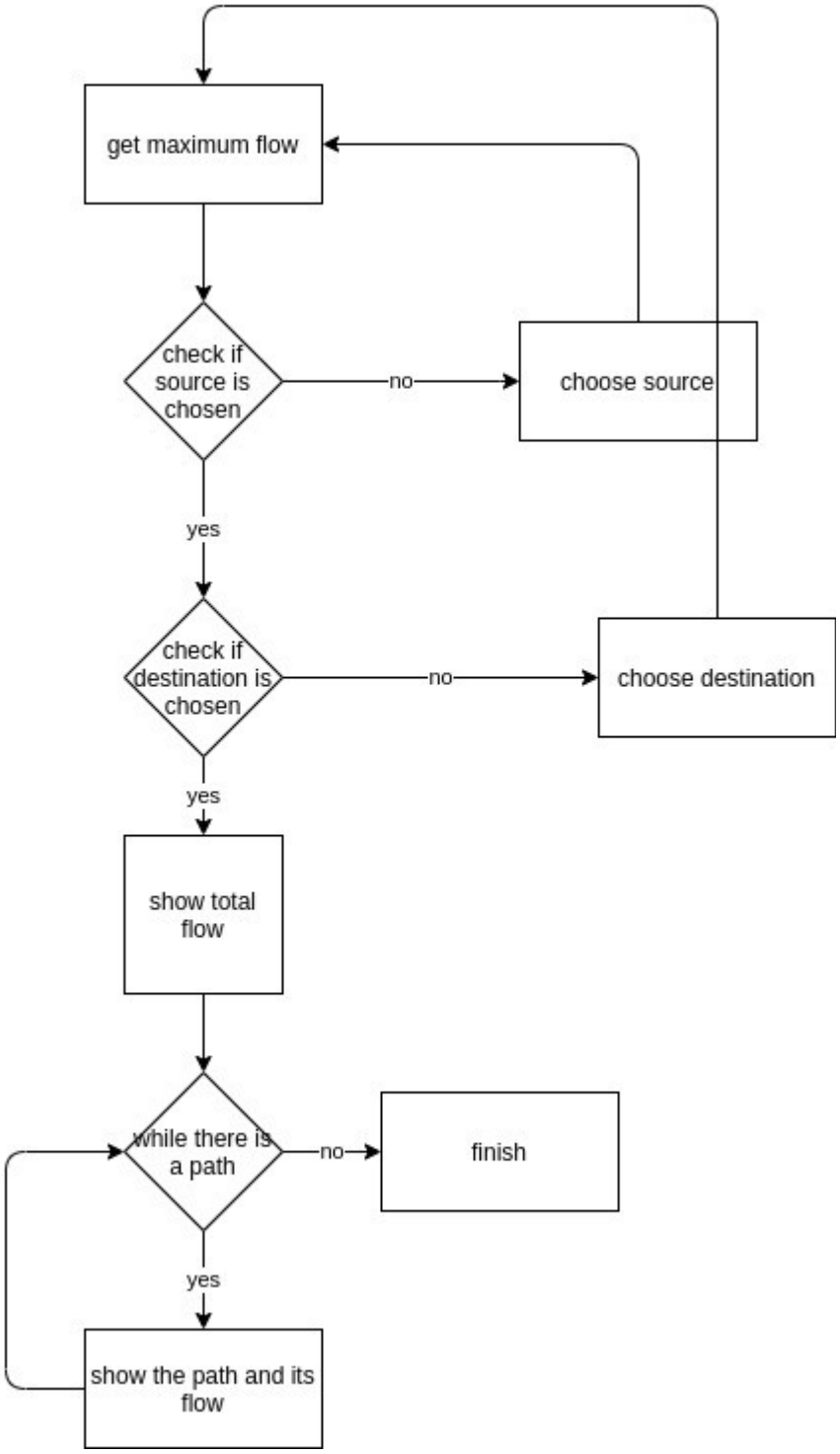
first this system needs to draw graphs so it needs some GUI for adding and deleting node , and also for adding and deleting edges and putting weights on edges , then we need a button that when pressed will show the shortest path between two selected nodes , so we need a way for selecting a source and a destination , also we need a button for getting the maximum flow from some source , and we need a way of showing each path separately .

Flow-charts:

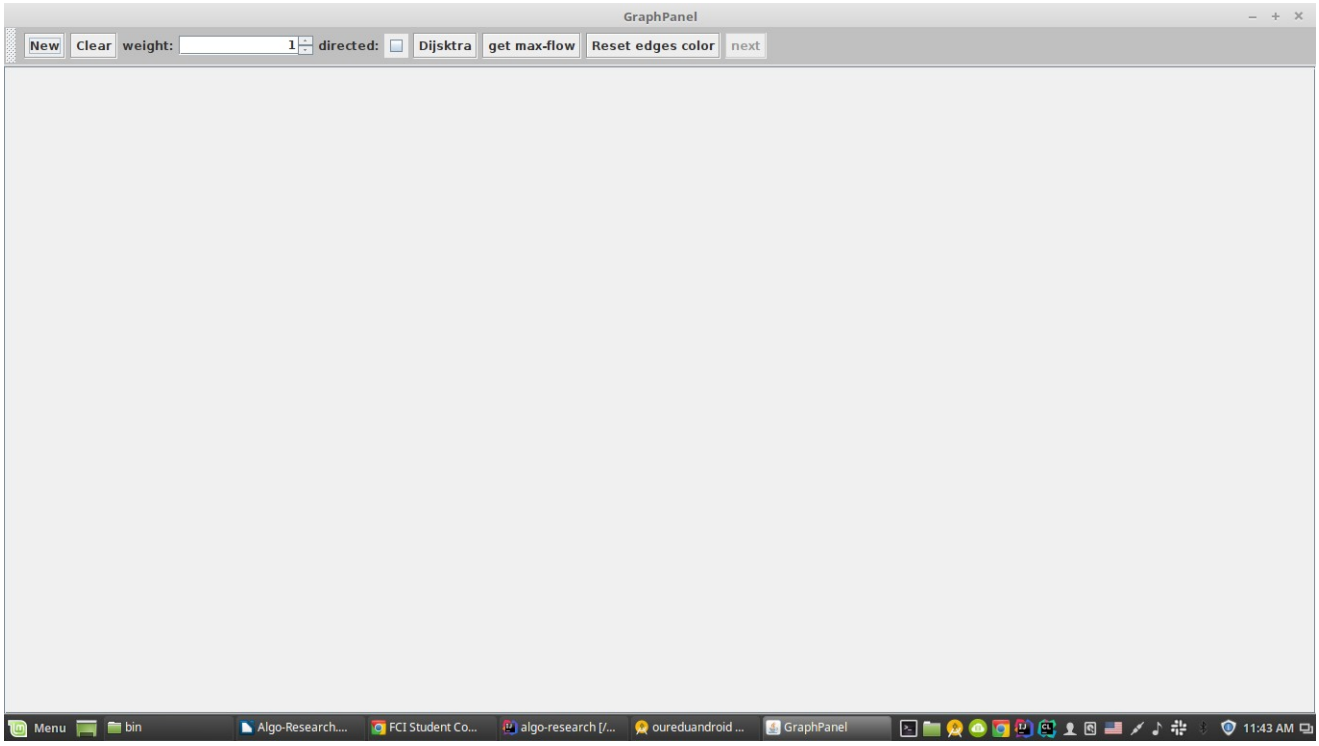
Dijkstra



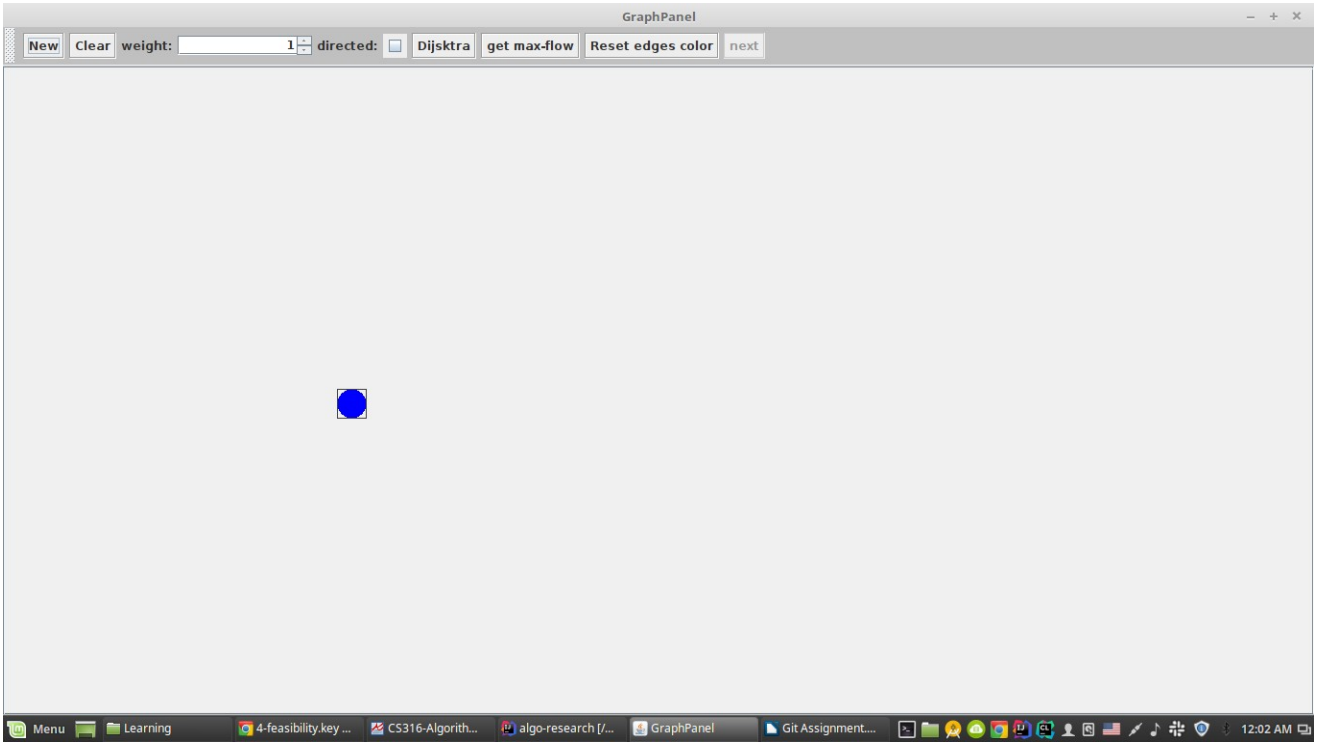
Max-flow

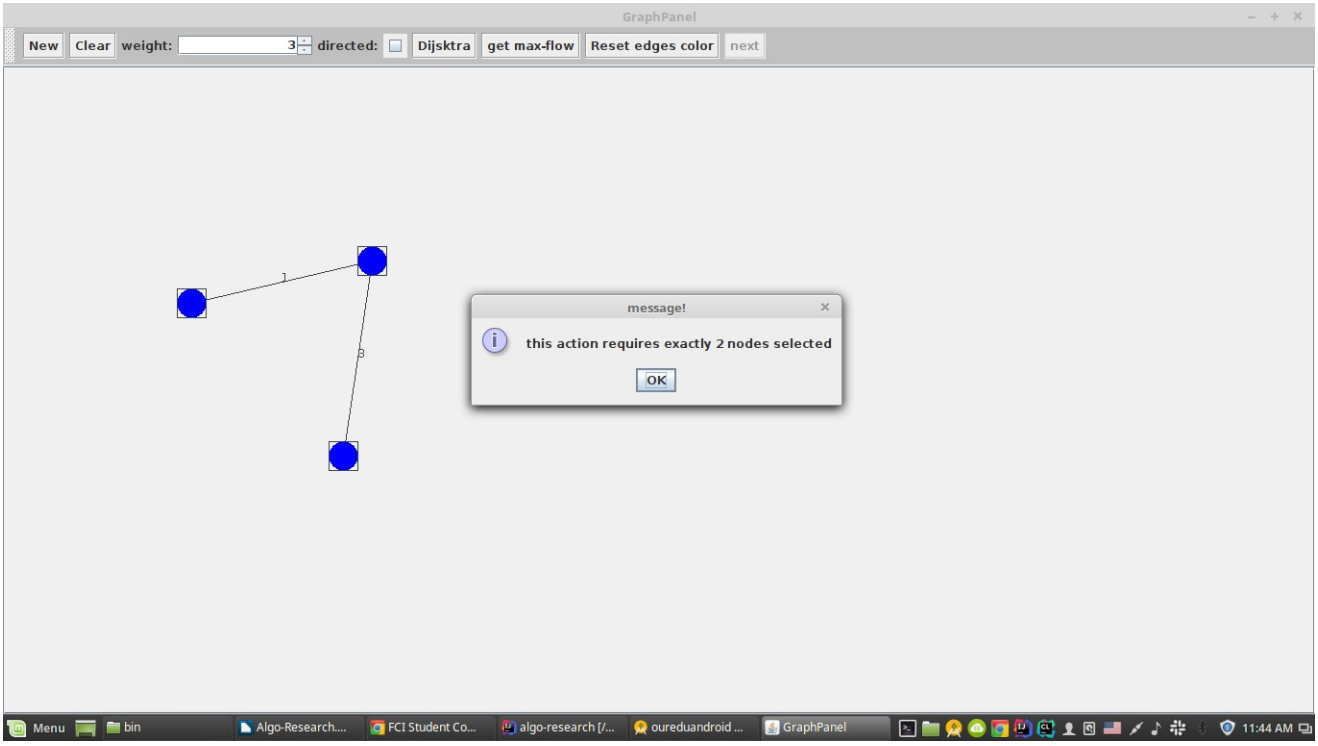

get maximum flow
check if source is chosen
no
choose source
yes
check if destination is chosen
no
choose destination
yes
show total flow
while there is a path
no
finish
yes
show the path and its flow

and here are some screen-shots for the system :

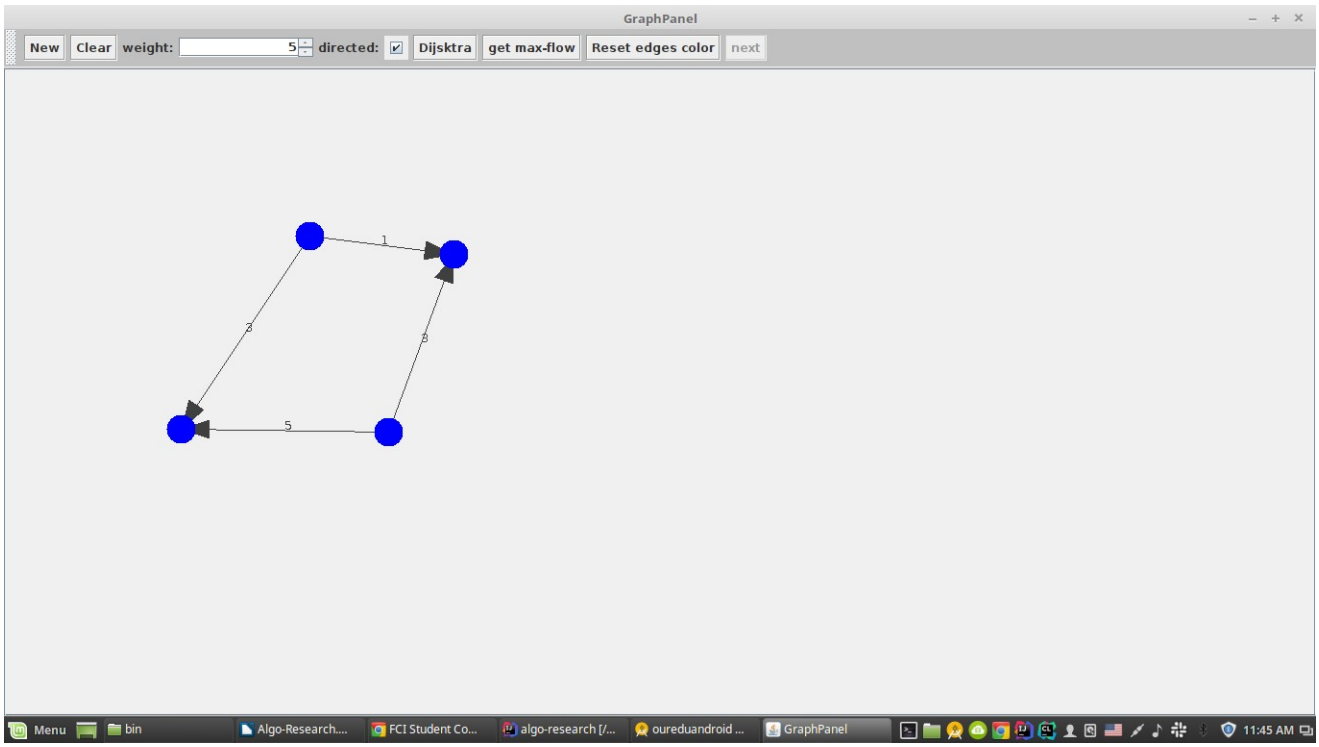


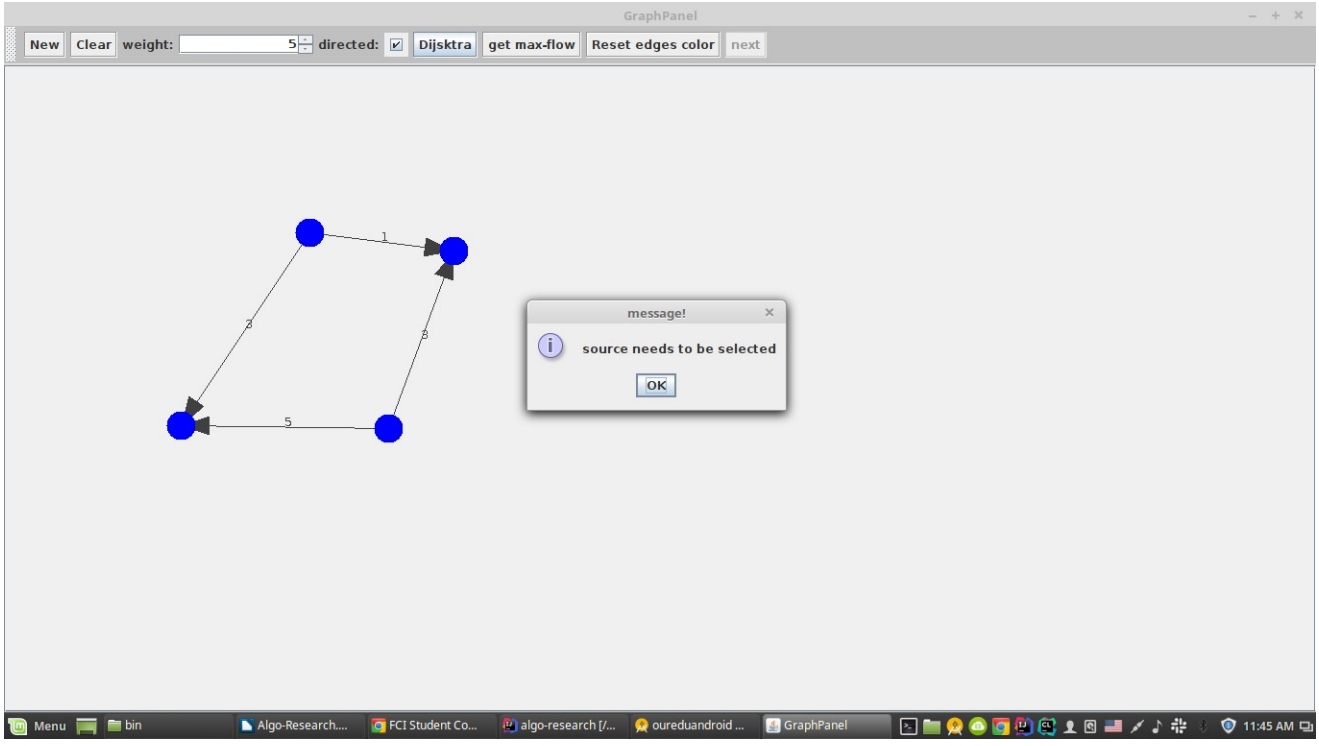
when you first start the program you will see this screen , the “New” button adds a new node , the “clear” button deletes the graph , the weight field is to choose the weight of an edge before adding it “notice that you must change the field’s value with the arrows beside” , isDirected checkbox when checked makes the whole directed in random directions , you can change the direction by selecting the two nodes that are connected and right-click and choose change edge direction , to add an edge choose the two nodes and right-click and choose “connect”

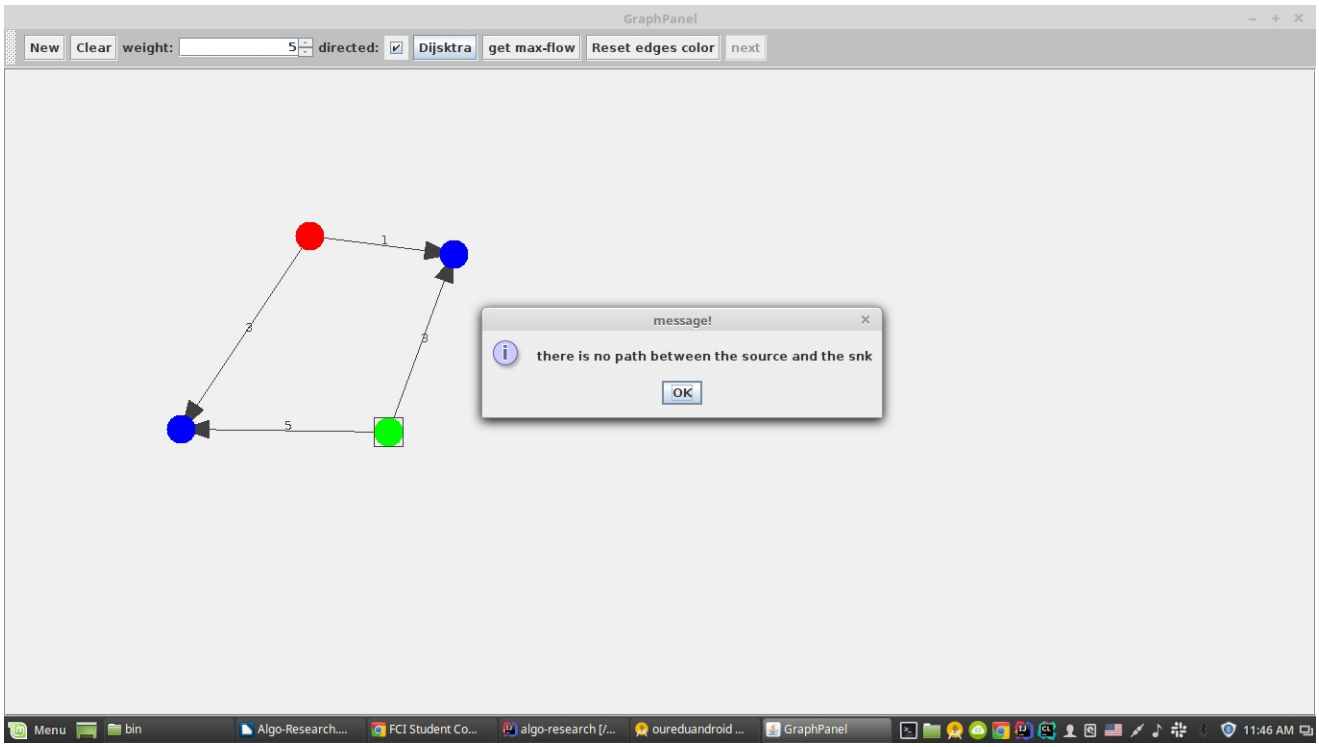


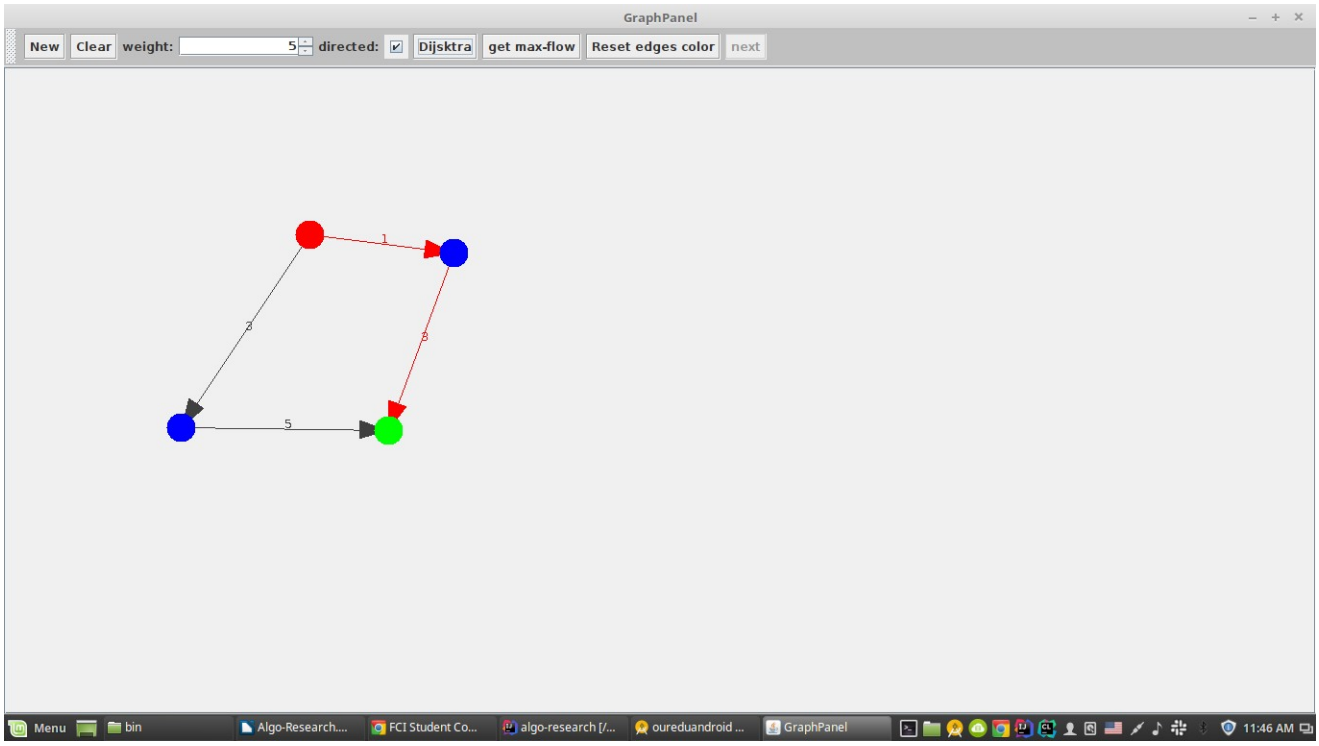


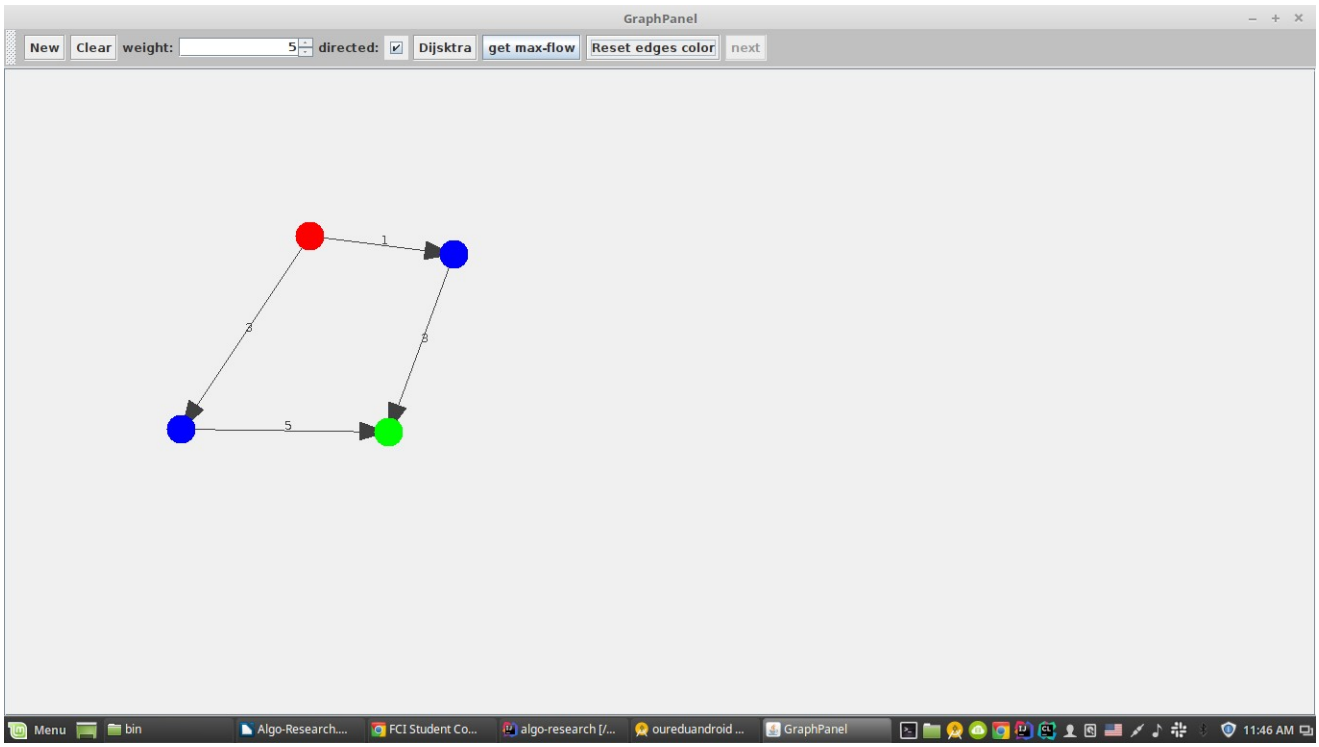
this message is shown when you choose an action that requires two nodes like adding an edge

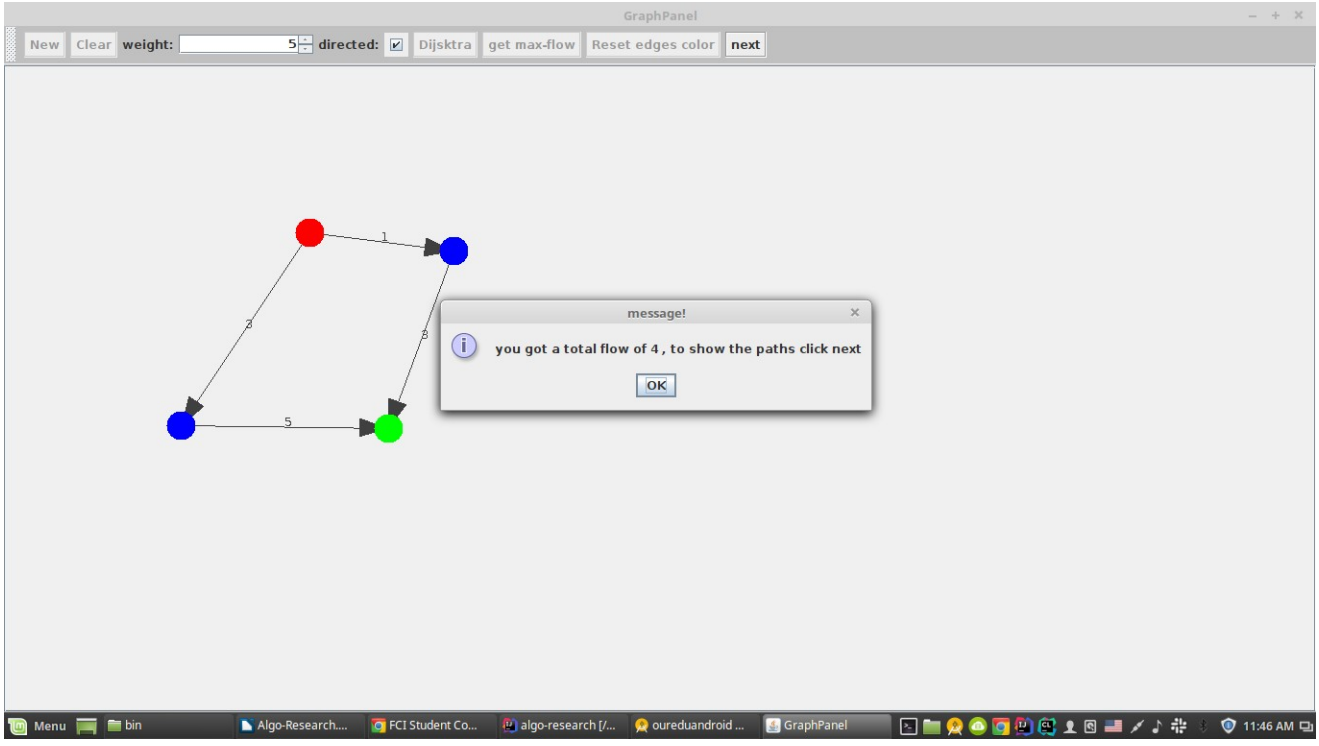

GraphPanel
New
Clear
weight:
5
directed:
Dijsktra
get max-flow
Reset edges color
next
1
3
3
5
Menu
bin
Algo-Research....
FCI Student Co...
algo-research [/...
oureduandroid ...
GraphPanel
11:45 AM

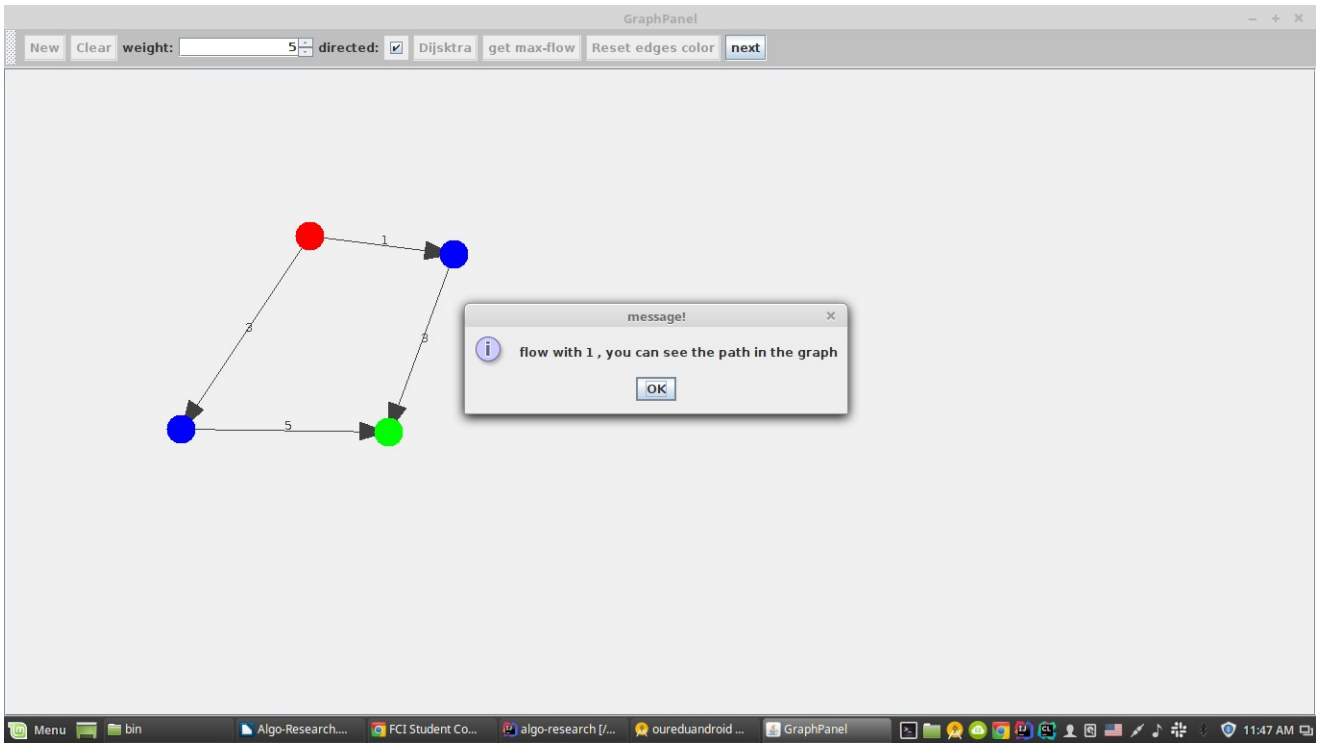

GraphPanel
New
Clear
weight:
5
directed:
Dijsktra
get max-flow
Reset edges color
next
1
3
3
5
message!
source needs to be selected
OK
Menu
bin
Algo-Research....
FCI Student Co...
algo-research [/...
oureduandroid ...
GraphPanel
11:45 AM

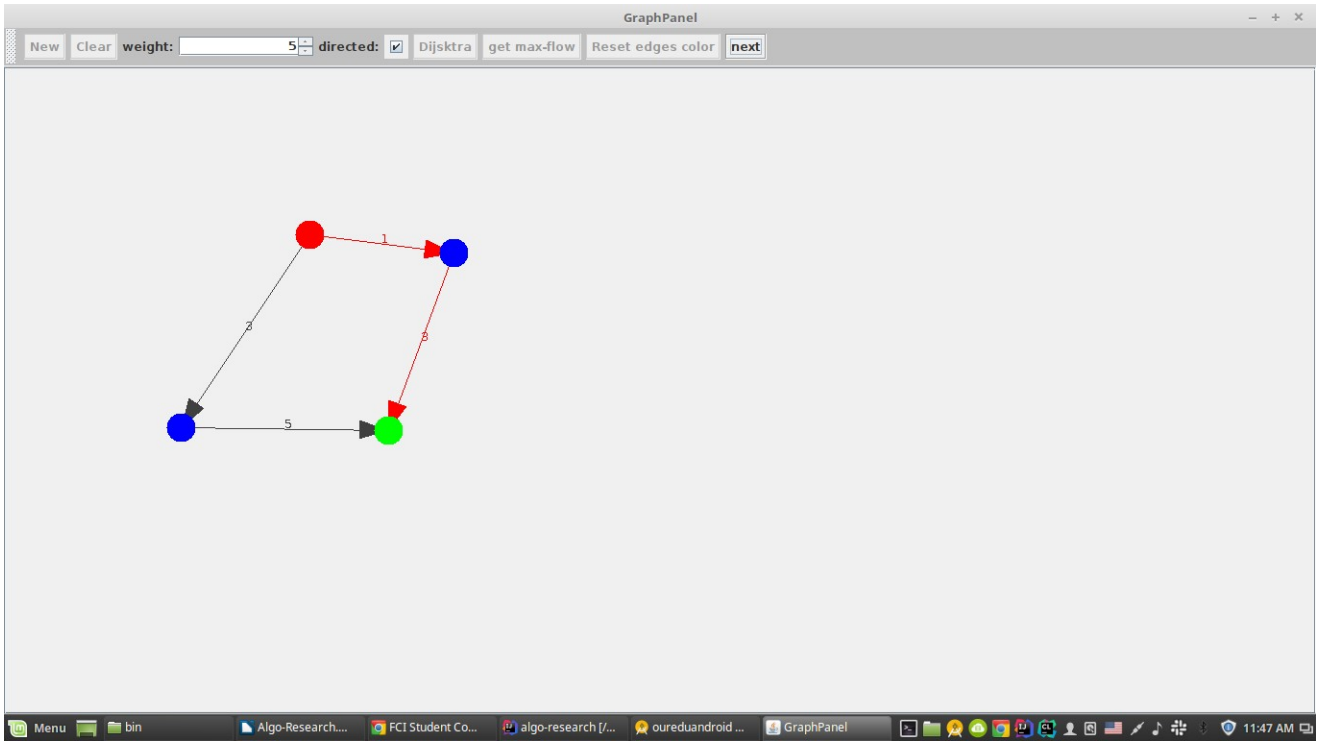

GraphPanel
New
Clear
weight:
5
directed:
Dijsktra
get max-flow
Reset edges color
next
1
3
3
5
message!
there is no path between the source and the snk
OK
Menu
bin
Algo-Research....
FCI Student Co...
algo-research [/...
oureduandroid ...
GraphPanel
11:46 AM

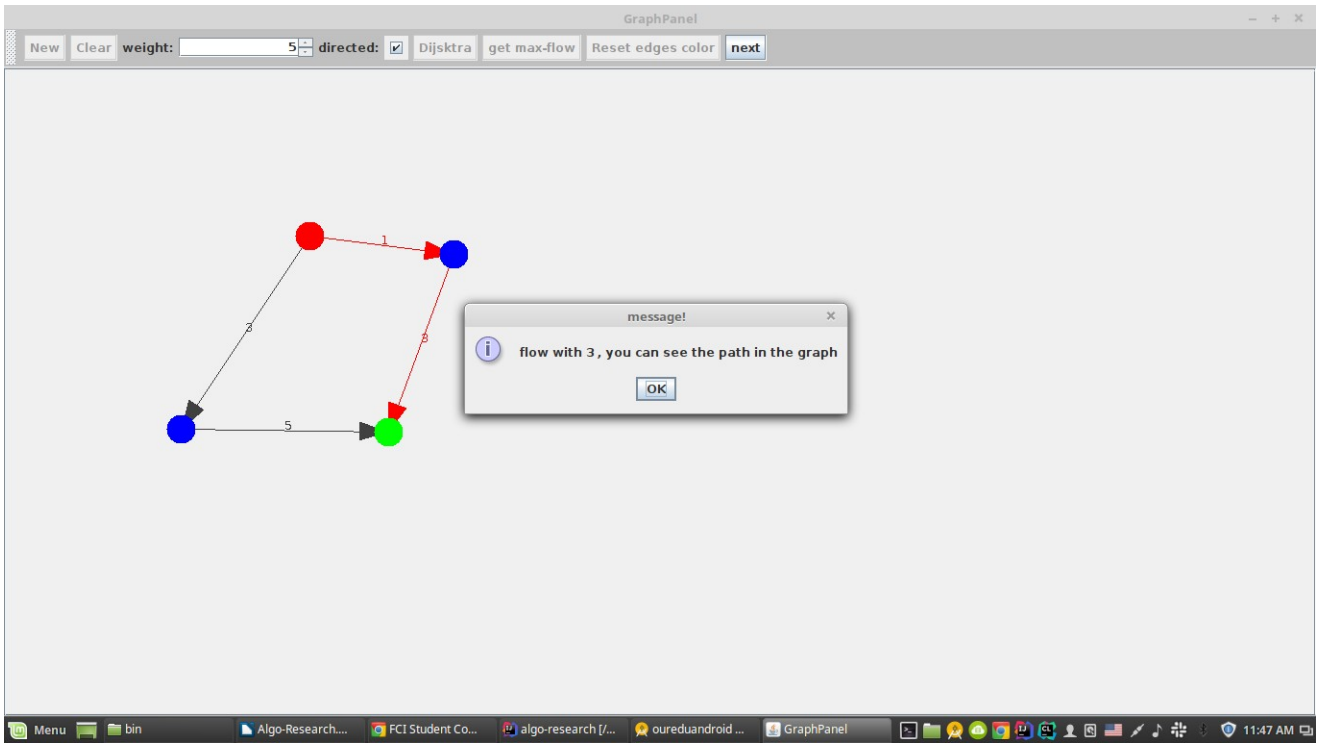

GraphPanel
New
Clear
weight:
5
directed:
Dijsktra
get max-flow
Reset edges color
next
1
3
3
5
Menu
bin
Algo-Research....
FCI Student Co...
algo-research [/...
oureduandroid ...
GraphPanel
11:46 AM

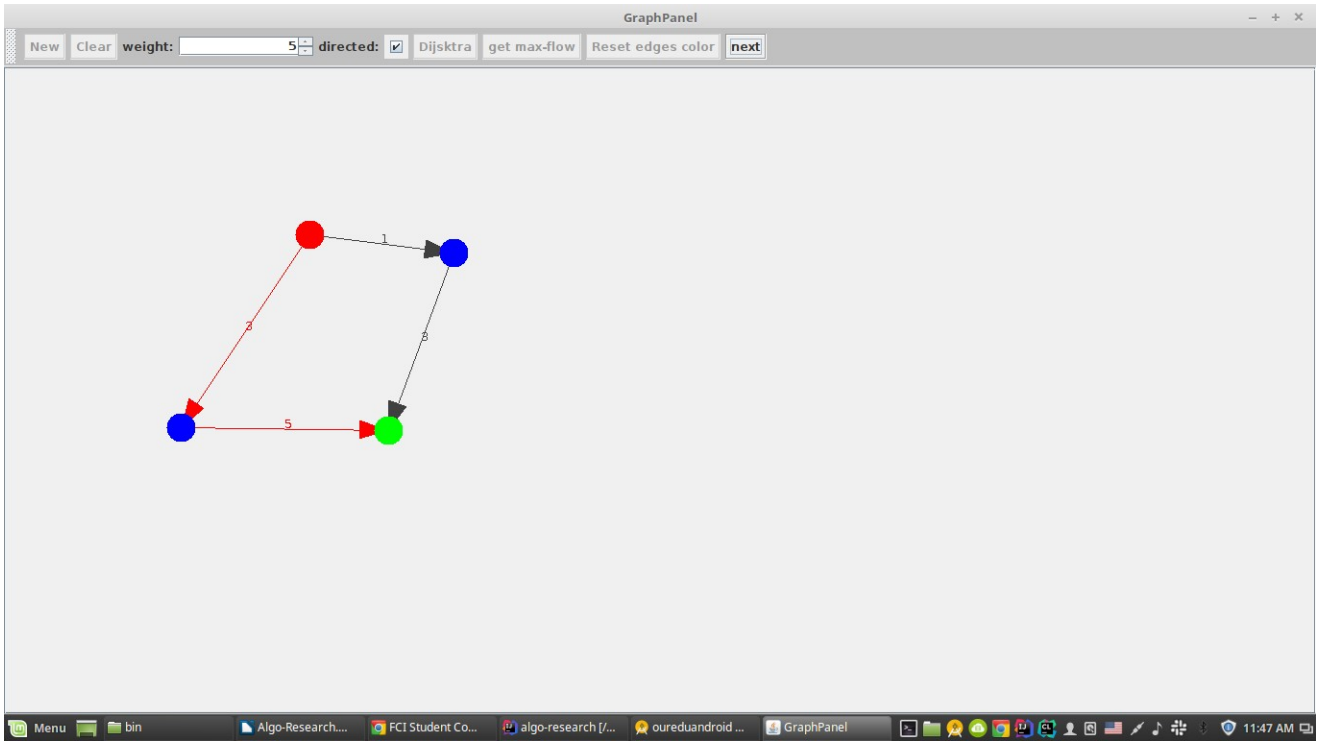

GraphPanel
New
Clear
weight:
5
directed:
Dijsktra
get max-flow
Reset edges color
next
1
3
3
5
Menu
bin
Algo-Research....
FCI Student Co...
algo-research [/...
oureduandroid ...
GraphPanel
11:46 AM

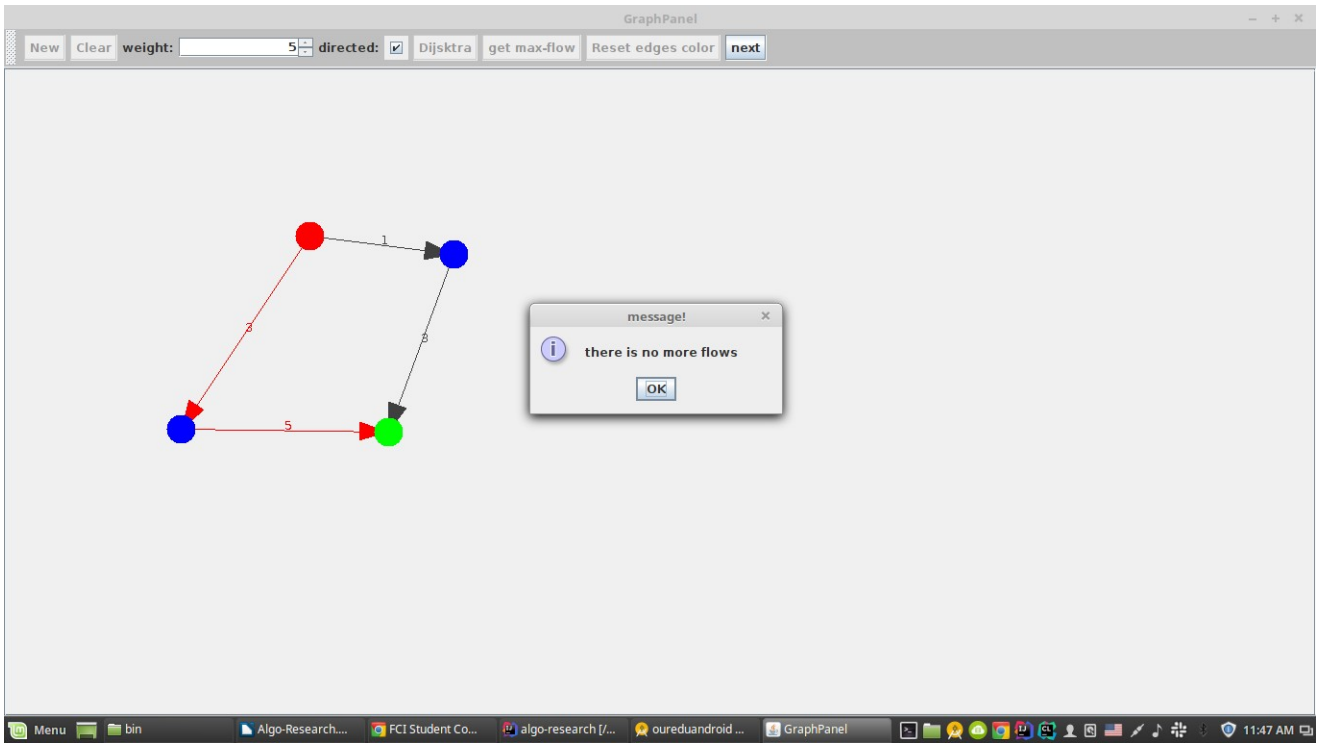

GraphPanel
New
Clear
weight:
5
directed:
Dijsktra
get max-flow
Reset edges color
next
1
3
3
5
message!
you got a total flow of 4 , to show the paths click next
OK
Menu
bin
Algo-Research....
FCI Student Co...
algo-research [/...
oureduandroid ...
GraphPanel
11:46 AM


GraphPanel
New
Clear
weight:
5
directed:
Dijsktra
get max-flow
Reset edges color
next
1
3
3
5
message!
flow with 1 , you can see the path in the graph
OK
Menu
bin
Algo-Research....
FCI Student Co...
algo-research [/...
oureduandroid ...
GraphPanel
11:47 AM


GraphPanel
New
Clear
weight:
5
directed:
Dijsktra
get max-flow
Reset edges color
next
1
3
3
5
Menu
bin
Algo-Research....
FCI Student Co...
algo-research [/...
oureduandroid ...
GraphPanel
11:47 AM


GraphPanel
New
Clear
weight:
5
directed:
Dijsktra
get max-flow
Reset edges color
next
1
3
3
5
message!
flow with 3 , you can see the path in the graph
OK
Menu
bin
Algo-Research....
FCI Student Co...
algo-research [/...
oureduandroid ...
GraphPanel
11:47 AM


GraphPanel
New
Clear
weight:
5
directed:
Dijsktra
get max-flow
Reset edges color
next
1
3
3
5
Menu
bin
Algo-Research....
FCI Student Co...
algo-research [/...
oureduandroid ...
GraphPanel
11:47 AM



and here is the implementation :

```
import java.awt.event.ActionEvent;
public class ChangeDirAction extends MyAction {

 ChangeDirAction(String name, Main main) {

  super(name,main);

 }

 public void actionPerformed(ActionEvent e) {

  Node.getSelected(main.nodes, main.selected);

  if (main.selected.size() != 2) {

   showMsg("this action requires exactly 2 nodes
selected");

   return;
```

```
    }
    Node n1 = main.selected.get(0);
    Node n2 = main.selected.get(1);
    boolean found = false;
    for (Edge edge : main.edges) {
     if (edge.n1 == n1 && edge.n2 == n2) {
      main.edges.add(new Edge(n2, n1, edge.val, true));
      main.edges.remove(edge);
      found = true;
      break;
     } else if (edge.n2 == n1 && edge.n1 == n2) {
      main.edges.add(new Edge(n1, n2, edge.val, true));
      main.edges.remove(edge);
      found = true;
      break;
     }
    }
    if (!found) {
     showMsg("this action requires 2 connected nodes");
    }
    main.repaint();
  }
import java.awt.event.ActionEvent;
```

```
public class ClearAction extends MyAction {
 ClearAction(String name, Main main) {
  super(name,main);
 }
 public void actionPerformed(ActionEvent e) {
  main.nodes.clear();
  main.edges.clear();
  main.repaint();
 }
}

import java.awt.event.ActionEvent;
public class ConnectAction extends MyAction {
 ConnectAction(String name, Main main) {
  super(name,main);
 }
 public void actionPerformed(ActionEvent e) {
  Node.getSelected(main.nodes, main.selected);
  if (main.selected.size() != 2) {
   showMsg("this action requires exactly 2 nodes to
be selected");
   return;
  }
```

```
   int x = (int) main.js2.getValue();
   Node n1 = main.selected.get(0);
   Node n2 = main.selected.get(1);
   if (main.isDir.isSelected()) {
    main.edges.add(new Edge(n1, n2, x, true));
   } else {
    main.edges.add(new Edge(n1, n2, x, false));
   }
   main.repaint();
  }
}

import java.awt.event.ActionEvent;
import java.util.ListIterator;
public class DeleteAction extends MyAction {
 DeleteAction(String name, Main main) {
  super(name,main);
 }
 public void actionPerformed(ActionEvent e) {
  ListIterator<Node> iter =
main.nodes.listIterator();
  while (iter.hasNext()) {
   Node n = iter.next();
```

```
    if (n.isSelected()) {
     deleteEdges(n);
     iter.remove();
    }
   }
   main.repaint();
  }
  void deleteEdges(Node n) {
   main.edges.removeIf(e -> e.n1 == n || e.n2 == n);
  }
}

import java.awt.event.ActionEvent;
public class DeleteEdge extends MyAction {
 DeleteEdge(String name, Main main) {
  super(name,main);
 }
 public void actionPerformed(ActionEvent e) {
  Node.getSelected(main.nodes, main.selected);
  if (main.selected.size() != 2) {
   showMsg("this action requires exactly 2 nodes
selected");
   return;
```

```
   }
   Node n1 = main.selected.get(0);
   Node n2 = main.selected.get(1);
   Edge edge = main.isConnected(n1, n2);
   if (edge == null) {
    showMsg("there is no edge between");
    return;
   }
   main.edges.remove(edge);
   main.repaint();
  }
}

import java.awt.*;
class Edge {
 Node n1;
 Node n2;
 int val;
 boolean isDir;
 private Color color = Color.darkGray;
 Edge(Node n1, Node n2, int val, boolean isDir) {
  this.n1 = n1;
  this.n2 = n2;
```

```
  this.val = val;
  this.isDir = isDir;
 }
 // this is just to be used with max flow
 // it doesn't compare weights
 boolean equals(Edge e2) {
  return this.n1 == e2.n1 && this.n2 == e2.n2 &&
this.isDir == e2.isDir;
 }
 Node getOtherNode(Node n) {
  if (n == n1) return n2;
  return n1;
 }
 void resetColor() {
  color = Color.darkGray;
 }
 void setColor(Color color) {
  this.color = color;
 }
 void draw(Graphics g) {
  Point p1 = n1.getLocation();
  Point p2 = n2.getLocation();
  g.setColor(color);
```

```
 g.drawLine(p1.x, p1.y, p2.x, p2.y);
 int xMid = (p1.x + p2.x) / 2;
 int yMid = (p1.y + p2.y) / 2;
 g.drawString(Integer.toString(val), xMid, yMid);
 if (isDir) {
  int d = 30;
  int h = 10;
  int dx = p2.x - p1.x, dy = p2.y - p1.y;
  double D = Math.sqrt(dx * dx + dy * dy);
  double xm = D - d, xn = xm, ym = h, yn = -h, x;
  double sin = dy / D, cos = dx / D;
  x = xm * cos - ym * sin + p1.x;
  ym = xm * sin + ym * cos + p1.y;
  xm = x;
  x = xn * cos - yn * sin + p1.x;
  yn = xn * sin + yn * cos + p1.y;
  xn = x;
  int[] xpoints = {p2.x, (int) xm, (int) xn};
  int[] ypoints = {p2.y, (int) ym, (int) yn};
  g.fillPolygon(xpoints, ypoints, 3);
 }
}
Edge copy() {
```

```
   Edge e = new Edge(n1, n2, val, isDir);
   e.setColor(color);
   return e;
  }
}


import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.List;
public class Main extends JComponent implements
ShowMsg {
 static final int WIDE = 800;
 static final int HIGH = 500;
 static final int RADIUS = 15;
 static final Color src = Color.RED;
 static final Color snk = Color.GREEN;
 static final Color defalt = Color.BLUE;
 ControlPanel control = new ControlPanel();
 int radius = RADIUS;
 List<Node> nodes = new ArrayList<>();
 List<Node> selected = new ArrayList<>();
 List<Edge> edges = new ArrayList<>();
```

```
 List<Pair<List<Edge>,Integer>> nextEdges = new
ArrayList<>();

 Point mousePt = new Point(WIDE / 2, HIGH / 2);

 Rectangle mouseRect = new Rectangle();

 boolean selecting = false;

 boolean isDirected = false;

 JSpinner js2;

 JCheckBox isDir;

 private Main() {

  this.setOpaque(true);

  this.addMouseListener(new MouseHandler(this));

  this.addMouseMotionListener(new
MouseMotionHandler(this));

 }

 public static void main(String[] args) {

  EventQueue.invokeLater(() -> {

   JFrame f = new JFrame("GraphPanel");

   f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   Main gp = new Main();

   f.add(gp.control, BorderLayout.NORTH);

   f.add(new JScrollPane(gp), BorderLayout.CENTER);


f.getRootPane().setDefaultButton(gp.control.defaultBu
tton);
```

```
   f.pack();
   f.setLocationByPlatform(true);
   f.setVisible(true);
  });
}
@Override
public Dimension getPreferredSize() {
  return new Dimension(WIDE, HIGH);
}
@Override
public void paintComponent(Graphics g) {
  g.setColor(new Color(0x00f0f0f0));
  g.fillRect(0, 0, getWidth(), getHeight());
  for (Edge e : edges) {
   e.draw(g);
  }
  for (Node n : nodes) {
   n.draw(g);
  }
  if (selecting) {
   g.setColor(Color.darkGray);
   g.drawRect(mouseRect.x, mouseRect.y,
     mouseRect.width, mouseRect.height);
```

```
  }
 }
 void deleteDuplicateEdges() {
  List<Edge> toBeRemoved = new ArrayList<>();
  for (Edge edge : edges) {
   int cnt = 0;
   for (Edge edge1 : edges) {
    if ((edge.n1 == edge1.n1 && edge.n2 == edge1.n2)
      || (edge.n1 == edge1.n2 && edge.n2 ==
edge1.n1)) cnt++;
    if (cnt > 1) toBeRemoved.add(edge1);
   }
  }
  edges.removeAll(toBeRemoved);
 }
 ArrayList<Edge> getEdges(Node node, List<Edge>
edges) {
  ArrayList<Edge> res = new ArrayList<>();
  for (Edge edge : edges) {
   if (edge.isDir && edge.n1 == node) {
    res.add(edge);
   } else if (!edge.isDir && (edge.n1 == node ||
edge.n2 == node)) {
    res.add(edge);
```

```
  }
 }
 return res;
}
Node getSrc() {
 for (Node node : nodes) {
  if (node.isSrc) return node;
 }
 return null;
}
Node getSnk() {
 for (Node node : nodes) {
  if (node.isSnk) return node;
 }
 return null;
}
Edge isConnected(Node n1, Node n2) {
 for (Edge edge : edges) {
  if (edge.n1 == n1 && edge.n2 == n2) return edge;
  if (edge.n1 == n2 && edge.n2 == n1) return edge;
 }
 return null;
}
```

```
 class ControlPanel extends JToolBar {

  Action newNode = new
NewNodeAction("New",Main.this);

  JButton defaultButton = new JButton(newNode);

  JPopupMenu popup = new JPopupMenu();

  Action clearAll = new
ClearAction("Clear",Main.this);

  Action dij = new
ShortestPathAction("Dijsktra",Main.this);

  Action connect = new
ConnectAction("Connect",Main.this);

  Action reset_edges_color = new ResetEdges("Reset
edges color",Main.this);

  Action changeDir = new ChangeDirAction("Change Edge
Direction",Main.this);

  Action makeDirInBoth = new MakeDirBothAction("Make
Direction in both ends", Main.this);

  Action delete = new DeleteAction("Delete selected
nodes",Main.this);

  Action deleteEdge = new DeleteEdge("Delete
Edge",Main.this);

  Action setSrc = new SrcAction("Set as
Source",Main.this);

  Action setSnk = new SnkAction("Set as
destination",Main.this);

  Action mxFlow = new MxFlowAction("get max-flow",
Main.this);
```

```
Action nxt = new NextAction("next", Main.this);
ControlPanel() {
 this.setLayout(new FlowLayout(FlowLayout.LEFT));
 this.setBackground(Color.lightGray);
 this.add(defaultButton);
 this.add(new JButton(clearAll));
 isDir = new JCheckBox();
 isDir.addActionListener(e -> {
  isDirected = isDir.isSelected();
  if (!isDirected) {
   deleteDuplicateEdges();
   for (Edge edge : edges) {
    edge.isDir = false;
   }
  } else {
   for (Edge edge : edges) {
    edge.isDir = true;
   }
  }
  Main.this.repaint();
 });
 js2 = new JSpinner();
```

```
    js2.setModel(new SpinnerNumberModel(1, 0,
10000000, 1));
    this.add(new JLabel("weight:"));
    this.add(js2);
    this.add(new JLabel("directed:"));
    this.add(isDir);
    this.add(dij);
    this.add(mxFlow);
    this.add(reset_edges_color);
    this.add(nxt);
    nxt.setEnabled(false);
    //todo popup menu
    popup.add(new JMenuItem(newNode));
    popup.add(new JMenuItem(connect));
    popup.add(new JMenuItem(changeDir));
    popup.add(new JMenuItem(makeDirInBoth));
    popup.add(new JMenuItem(delete));
    popup.add(new JMenuItem(deleteEdge));
    popup.add(new JMenuItem(setSrc));
    popup.add(new JMenuItem(setSnk));
  }
  void setClick(boolean click) {
    newNode.setEnabled(click);
```

```
    defaultButton.setEnabled(click);
    clearAll.setEnabled(click);
    dij.setEnabled(click);
    connect.setEnabled(click);
    reset_edges_color.setEnabled(click);
    changeDir.setEnabled(click);
    makeDirInBoth.setEnabled(click);
    delete.setEnabled(click);
    deleteEdge.setEnabled(click);
    setSrc.setEnabled(click);
    setSnk.setEnabled(click);
    mxFlow.setEnabled(click);
    nxt.setEnabled(!click);
   }
  }
}

import java.awt.event.ActionEvent;
public class MakeDirBothAction extends MyAction{
 MakeDirBothAction(String name, Main main) {
  super(name,main);
 }
 public void actionPerformed(ActionEvent e) {
```

```
  Node.getSelected(main.nodes, main.selected);
  if(main.selected.size()!=2){
   showMsg("this action requires exactly 2 nodes
selected");
   return;
  }
  Node n1=main.selected.get(0);
  Node n2=main.selected.get(1);
  boolean found = false;
  for(Edge edge:main.edges){
   if(edge.n1==n1 && edge.n2==n2){
    main.edges.add(new Edge(n2,n1,edge.val,true));
    if(!edge.isDir){
     main.edges.remove(edge);
     main.edges.add(new Edge(n1,n2,edge.val,true));
    }
    found=true;
    break;
   }else if(edge.n2==n1 && edge.n1==n2){
    main.edges.add(new Edge(n1,n2,edge.val,true));
    if(!edge.isDir){
     main.edges.remove(edge);
     main.edges.add(new Edge(n2,n1,edge.val,true));
```

```
      }
      found=true;
      break;
     }
    }
    if(!found){
     showMsg("this action requires 2 connected nodes");
    }
    main.repaint();
  }
}

import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
public class MouseHandler extends MouseAdapter {
 Main main;
 MouseHandler(Main main){
  this.main=main;
 }
 @Override
 public void mouseReleased(MouseEvent e) {
  main.selecting = false;
  main.mouseRect.setBounds(0, 0, 0, 0);
```

```
    if (e.isPopupTrigger()) {
      showPopup(e);
    }
    e.getComponent().repaint();
  }
  @Override
  public void mousePressed(MouseEvent e) {
    main.mousePt = e.getPoint();
    if (e.isShiftDown()) {
      Node.selectToggle(main.nodes, main.mousePt);
    } else if (e.isPopupTrigger()) {
      Node.selectOne(main.nodes, main.mousePt);
      showPopup(e);
    } else if (Node.selectOne(main.nodes,
main.mousePt)) {
      main.selecting = false;
    } else {
      Node.selectNone(main.nodes);
      main.selecting = true;
    }
    e.getComponent().repaint();
  }
  void showPopup(MouseEvent e) {
```

```
  main.control.popup.show(e.getComponent(), e.getX(),
e.getY());
 }
}


import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;
public class MouseMotionHandler extends
MouseMotionAdapter {
 Main main;
 MouseMotionHandler(Main main){
  this.main=main;
 }
 Point delta = new Point();
 @Override
 public void mouseDragged(MouseEvent e) {
  if (main.selecting) {
   main.mouseRect.setBounds(
     Math.min(main.mousePt.x, e.getX()),
     Math.min(main.mousePt.y, e.getY()),
     Math.abs(main.mousePt.x - e.getX()),
     Math.abs(main.mousePt.y - e.getY()));
```

```
    Node.selectRect(main.nodes, main.mouseRect);
   } else {
    delta.setLocation(
      e.getX() - main.mousePt.x,
      e.getY() - main.mousePt.y);
    Node.updatePosition(main.nodes, delta);
    main.mousePt = e.getPoint();
   }
   e.getComponent().repaint();
  }
}

import java.awt.event.ActionEvent;
import java.util.ArrayList;
import java.util.List;
public class MxFlowAction extends MyAction{
 MxFlowAction(String name, Main main) {
  super(name,main);
 }
 static class MnVal{
  int val=(int)1e9;
 }
 public void actionPerformed(ActionEvent e) {
```

```
  if(!main.isDirected){
   showMsg("can't get max flow for an undirected
graph");
   return;
  }
  for(Edge ed:main.edges){
   ed.resetColor();
  }
  List<Edge> es=new ArrayList<>();
  for(Edge edge:main.edges){
   es.add(edge.copy());
  }
  Node src=main.getSrc();
  Node snk=main.getSnk();
  if (src == null) {
   showMsg("source needs to be selected");
   main.repaint();
   return;
  }
  if (snk == null) {
   showMsg("destination needs to be selected");
   main.repaint();
   return;
```

```
  }
  int total=0;
  while(true){
   MnVal mnVal= new MnVal();
   List<Node> lst=new ArrayList<>();
   lst.add(src);
   List<Edge> path =
getPath(src,snk,null,es,mnVal,lst);
   if(path==null)break;
   for(Edge edge:path){
    edge.val-=mnVal.val;
   }
   total+=mnVal.val;
   main.nextEdges.add(new Pair<>(path,mnVal.val));
  }
  main.control.setClick(false);
  showMsg("you got a total flow of "+total+" , to
show the paths click next");
 }
 private List<Edge> getPath(Node cur, Node dist, Edge
e, List<Edge> edges, MnVal mnVal, List<Node> nodes){
  if(cur==dist){
   List<Edge> res=new ArrayList<>();
   // this should never be false
```

```
    if(e!=null) {
     mnVal.val = Math.min(mnVal.val, e.val);
     res.add(e);
    }
    return res;
   }
   List<Edge> con = main.getEdges(cur,edges);
   for(Edge edge:con){
    if(edge.val<=0)continue;
    Node n2=edge.getOtherNode(cur);
    if(nodes.contains(n2))continue;
    nodes.add(n2);
    List<Edge>
res=getPath(n2,dist,edge,edges,mnVal,nodes);
    if(res!=null){
     if(e!=null) {
      mnVal.val = Math.min(mnVal.val, e.val);
      res.add(e);
     }
     return res;
    }
   }
   return null;
```

```
 }
}


import javax.swing.*;
abstract class MyAction extends AbstractAction
implements ShowMsg {
 Main main;
 MyAction(String name, Main main){
  super(name);
  this.main=main;
 }
}


import java.awt.*;
import java.awt.event.ActionEvent;
public class NewNodeAction extends MyAction {
 private boolean shown=false;
 NewNodeAction(String name, Main main) {
  super(name,main);
 }
 public void actionPerformed(ActionEvent e) {
  if(!shown) {
```

```
    showMsg("some operations can be done using click-
right after selecting required nodes (like adding
edge)");
    shown=true;
   }
   Node.selectNone(main.nodes);
   Point p = new Point((int) (Math.random() *
Math.min(Main.WIDE, Main.HIGH)),
     (int) (Math.random() * Math.min(Main.WIDE,
Main.HIGH)));
   Node n = new Node(p, main.radius, Main.defalt);
   n.setSelected(true);
   main.nodes.add(n);
   main.repaint();
  }
}


import java.awt.*;
import java.awt.event.ActionEvent;
import java.util.List;
public class NextAction extends MyAction {
 NextAction(String name, Main main) {
   super(name,main);
 }
```

```
 public void actionPerformed(ActionEvent e) {
  for(Edge ed:main.edges){
   ed.resetColor();
  }
  if(main.nextEdges.size()==0){
   showMsg("there is no more flows");
   main.control.setClick(true);
   return;
  }
  List<Edge> curEdges = main.nextEdges.get(0).first;
  int mn = main.nextEdges.get(0).second;
  for(Edge edge:curEdges){
   for(Edge edge1:main.edges){
    if(edge.equals(edge1)){
     edge1.setColor(Color.RED);
     break;
    }
   }
  }
  showMsg("flow with "+mn+" , you can see the path in
the graph");
  main.nextEdges.remove(0);
  main.repaint();
```

```
 }
}

import java.awt.*;
import java.util.List;
public class Node implements Comparable<Node> {
 private Point p;
 private int r;
 private Color color;
 private boolean selected = false;
 private Rectangle b = new Rectangle();
 boolean isSrc, isSnk;
 int mnDist = (int) 1e9;
 Node parent = null;
 /**
 * Construct a new node.
 */
 Node(Point p, int r, Color color) {
  this.p = p;
  this.r = r;
  this.color = color;
  setBoundary(b);
 }
```

```
 /**
  * Collected all the selected nodes in list.
  */
 static void getSelected(java.util.List<Node> list,
java.util.List<Node> selected) {
  selected.clear();
  for (Node n : list) {
   if (n.isSelected()) {
    selected.add(n);
   }
  }
 }
 /**
  * Select no nodes.
  */
 static void selectNone(java.util.List<Node> list) {
  for (Node n : list) {
   n.setSelected(false);
  }
 }
 /**
  * Select a single node; return true if not already
selected.
```

```
 */
 static boolean selectOne(java.util.List<Node> list,
Point p) {
  for (Node n : list) {
   if (n.contains(p)) {
    if (!n.isSelected()) {
     Node.selectNone(list);
     n.setSelected(true);
    }
    return true;
   }
  }
  return false;
 }
 /**
 * Select each node in r.
 */
 static void selectRect(java.util.List<Node> list,
Rectangle r) {
  for (Node n : list) {
   n.setSelected(r.contains(n.p));
  }
 }
```

```
 /**
  * Toggle selected state of each node containing p.
  */
 static void selectToggle(java.util.List<Node> list,
Point p) {
  for (Node n : list) {
   if (n.contains(p)) {
    n.setSelected(!n.isSelected());
   }
  }
 }
 /**
  * Update each node's position by d (delta).
  */
 static void updatePosition(List<Node> list, Point d)
{
  for (Node n : list) {
   if (n.isSelected()) {
    n.p.x += d.x;
    n.p.y += d.y;
    n.setBoundary(n.b);
   }
  }
```

```
}
void resetForDij() {
 parent = null;
 mnDist = (int) 1e9;
}
@Override
public int compareTo(Node o) {
 return Integer.compare(this.mnDist, o.mnDist);
}
void setSrc(List<Node> nodes) {
 for (Node node : nodes) {
  node.undoSrc();
 }
 isSrc = true;
 this.color = Main.src;
 isSnk = false;
}
private void undoSrc() {
 isSrc = false;
 if (!isSnk)
  color = Main.defalt;
}
private void undoSnk() {
```

```
 isSnk = false;
 if (!isSrc)
  color = Main.defalt;
}
void setSnk(List<Node> nodes) {
 for (Node node : nodes) {
  node.undoSnk();
 }
 isSrc = false;
 isSnk = true;
 this.color = Main.snk;
}
/**
* Calculate this node's rectangular boundary.
*/
private void setBoundary(Rectangle b) {
 b.setBounds(p.x - r, p.y - r, 2 * r, 2 * r);
}
/**
* Draw this node.
*/
void draw(Graphics g) {
 g.setColor(this.color);
```

```
 g.fillOval(b.x, b.y, b.width, b.height);
 if (selected) {
  g.setColor(Color.darkGray);
  g.drawRect(b.x, b.y, b.width, b.height);
 }
}
/**
* Return this node's location.
*/
Point getLocation() {
 return p;
}
/**
* Return true if this node contains p.
*/
private boolean contains(Point p) {
 return b.contains(p);
}
/**
* Return true if this node is selected.
*/
boolean isSelected() {
 return selected;
```

```
 }
 /**
  * Mark this node as selected.
  */
 void setSelected(boolean selected) {
  this.selected = selected;
 }
}
public class Pair<T,E> {
 T first;
 E second;
 Pair(){}
 public Pair(T first, E second) {
  this.first = first;
  this.second = second;
 }
}
import java.awt.event.ActionEvent;
public class ResetEdges extends MyAction {
 ResetEdges(String name, Main main) {
  super(name,main);
 }
 public void actionPerformed(ActionEvent e) {
```

```
   for (Edge ed : main.edges) {
    ed.resetColor();
   }
   main.repaint();
  }
}
import java.awt.*;
import java.awt.event.ActionEvent;
import java.util.PriorityQueue;
public class ShortestPathAction extends MyAction {
 ShortestPathAction(String name, Main main) {
  super(name,main);
 }
 public void actionPerformed(ActionEvent e) {
  if (!main.isDirected) {
   main.deleteDuplicateEdges();
  }
  for (Edge ed : main.edges) {
   ed.resetColor();
  }
  Node src = main.getSrc();
  Node snk = main.getSnk();
  if (src == null) {
```

```
    showMsg("source needs to be selected");
    main.repaint();
    return;
   }
   if (snk == null) {
    showMsg("destination needs to be selected");
    main.repaint();
    return;
   }
   for (Node node : main.nodes) {
    node.resetForDij();
   }
   src.mnDist = 0;
   PriorityQueue<Node> priorityQueue = new
PriorityQueue<>();
   priorityQueue.add(src);
   while (!priorityQueue.isEmpty()) {
    Node u = priorityQueue.poll();
    for (Edge edge : main.getEdges(u, main.edges)) {
     Node v = edge.n2;
     if (v == u) v = edge.n1;
     int weight = edge.val;
     int minDistance = u.mnDist + weight;
```

```
     if (minDistance < v.mnDist) {
      priorityQueue.remove(u);
      v.parent = u;
      v.mnDist = minDistance;
      priorityQueue.add(v);
     }
    }
   }
   if (snk.parent == null) {
    showMsg("there is no path between the source and
the snk");
    main.repaint();
    return;
   }
   Node cur = snk;
   while (cur != src) {
    Edge edge = main.isConnected(cur, cur.parent);
    // this if should never be true
    if (edge == null) {
     showMsg("there is no path between the source and
the snk");
     return;
    }
```

```
    edge.setColor(Color.RED);
    cur = cur.parent;
   }
   main.repaint();
 }
}

import javax.swing.*;
public interface ShowMsg {
 default void showMsg(String msg){
  JOptionPane.showMessageDialog(null
     , msg
     , "message!",
     JOptionPane.INFORMATION_MESSAGE);
 }
}

import java.awt.event.ActionEvent;
public class SnkAction extends MyAction {
 SnkAction(String name, Main main) {
  super(name,main);
 }
 public void actionPerformed(ActionEvent e) {
```

```
   Node.getSelected(main.nodes, main.selected);
   if (main.selected.size() < 1) return;
   if (main.selected.size() > 1) {
    showMsg("you can't set more than one
destination.");
   } else {
    main.selected.get(0).setSnk(main.nodes);
   }
   main.repaint();
 }
}


import java.awt.event.ActionEvent;
public class SrcAction extends MyAction {
 SrcAction(String name, Main main) {
  super(name,main);
 }
 public void actionPerformed(ActionEvent e) {
  Node.getSelected(main.nodes, main.selected);
  if (main.selected.size() < 1) return;
  if (main.selected.size() > 1) {
   showMsg("you can't set more than one source");
  } else {
```

```
    main.selected.get(0).setSrc(main.nodes);
   }
   main.repaint();
  }
}
```

Conclusion :
pros – friendly GUI with a lot of specified actions and the ability to reshape the graph and change the nodes locations easily and the ability to view the maximum flow paths , path by path to see each how flow is sent

cons – not well tested , so it might have small bugs ,
no way of changing the edge's weight directly (to do that you will have delete the edge and change the weight field and then add the edge again)

References :

authors : Thomas H. Cormen - Charles E. Leiserson
Ronald L. Rivest - Clifford Stein

Book name : Introduction to Algorithms, Second Edition

Year of publishing : 2001