DATA VISUALISATION AND VISUAL ANALYTICS

ASSIGNMENT 3

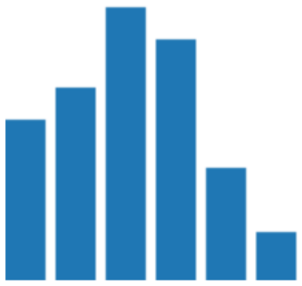
1. Protovis

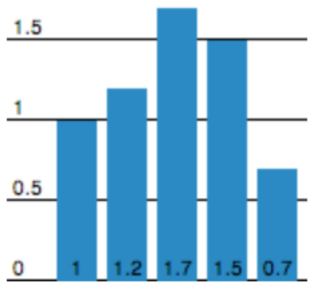
Protovis composes custom views of data with simple marks such as bars and dots. Unlike low-level graphics libraries that quickly become tedious for visualization, Protovis defines marks through dynamic properties that encode data, allowing inheritance, scales and layouts to simplify construction.

Protovis is free and open-source, provided under the BSD License. It uses JavaScript and SVG for web-native visualizations; no plugin required (though you will need a modern web browser)! Although programming experience is helpful, Protovis is mostly declarative and designed to be learned.

Protovis is no longer under active development.

Consider this bar chart, which visually encodes an array of numbers with height:

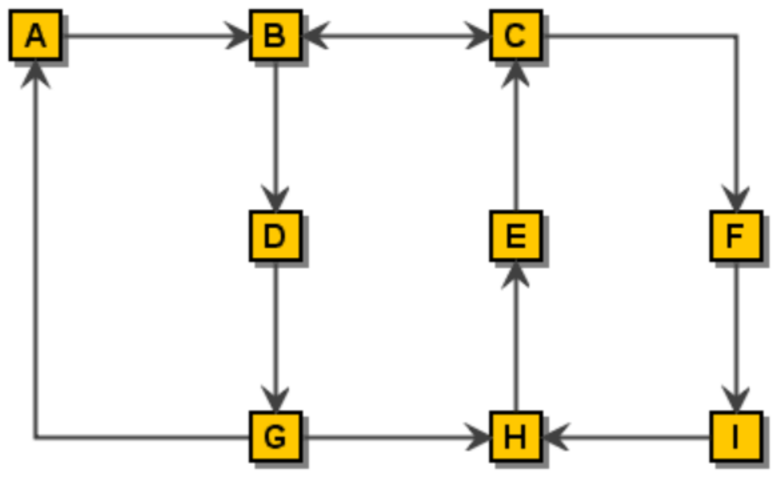
```
var vis = new pv.Panel()
    .width(150)
    .height(150);

vis.add(pv.Bar)
    .data([1, 1.2, 1.7, 1.5, .7, .3])
    .width(20)
    .height(function(d) d * 80)
    .bottom(0)
    .left(function() this.index * 25);

vis.render();
```
	

```
var vis = new pv.Panel()
     .width(150)
     .height(150);
vis.add(pv.Rule)
     .data(pv.range(0, 2, .5))
     .bottom(function(d) d * 80 + .5)
   .add(pv.Label);
vis.add(pv.Bar)
     .data([1, 1.2, 1.7, 1.5, .7])
     .width(20)
     .height(function(d) d * 80)
     .bottom(0)
     .left(function() this.index * 25 + 25)
   .anchor("bottom").add(pv.Label);
vis.render();
```
	

To simplify construction, Protovis supports panels and inheritance. A panel is a container for replicating marks. Inheritance lets you derive new marks from existing ones, sharing some or all of the properties. For example, here we derive labels for a rule and bar as shown above.

2. JGraph

With the JGraph class, you can display objects and their relations. A JGraph object doesn't actually contain your data; it simply provides a view of the data. Like any non-trivial Swing component, the graph gets data by querying its data model. Here's a picture of a graph:



JGraph displays its data by drawing individual elements. Each element displayed by the graph contains exactly one item of data, which is called a cell. A cell may either be a vertex, an edge or a port. Vertices have zero or more neighbours, and edges have one or no source and target vertex. Each cell has zero or more children, and one or no parent.

Creating a Graph

The following code creates a JGraph object:

```
JGraph graph = new JGraph();

...

JScrollPane scrollPane = new JScrollPane(graph)
```

Customizing a Graph

JGraph offers the following forms of interactions:

- In-place editing
- Moving
- Cloning
- Sizing
- Bending (Adding/Removing/Moving edge points)
- Establishing Connections
- Removing Connections

Methods In JGraph

- setEnabled(false)
- setEditable(false)
- setMoveable
- setCloneable
- setSizeable
- setBendable
- setConnectable
- setDisconnectable

Responding to Interaction

- Responding to Mouse Events

```
        graph.addMouseListener(new MouseAdapter() {
          public void mousePressed(MouseEvent e) {
            if (e.getClickCount() == 2) {
              // Get Cell under Mousepointer
              int x = e.getX(), y = e.getY();
              Object cell = graph.getFirstCellForLocation(x, y);
              // Print Cell Label
              if (cell != null) {
                String lab = graph.convertValueToString(cell);
                System.out.println(lab);
              }
            }
          }
});
```

- Responding to Model Events

```
      public class ModelListener implements GraphModelListener {
        public void graphCellsChanged(GraphModelEvent e) {
          System.out.println("Change: "+e.getChange());
        }
       }
       // Add an Instance to the Model
 graph.getModel().addGraphModelListener(new ModelListener());
```

- Responding to View Events

```
public class ViewObserver implements Observer {
  public void update(Observable o, Object arg) {
    System.out.println("View changed: "+o);
  }
 }
 // Add an Instance to the View
 graph.getView().addObserver(new ViewObserver());
```

- Responding to Selection Changes

```
public class MyListener implements GraphSelectionListener {
  public void valueChanged(GraphSelectionEvent e) {
    System.out.println("Selection changed: "+e);
  }
 }
 // Add an Instance to the Graph
 graph.addGraphSelectionListener(new MyListener());
```

Customizing a Graphs Display

- *Adding new Cell Types to a Graph*

```
protected VertexView createVertexView(Object v,
                                GraphModel model,
                                CellMapper cm) {
   // Return an EllipseView for EllipseCells
   if (v instanceof EllipseCell)
     return new EllipseView(v, model, cm);
   // Else Call Superclass
   return super.createVertexView(v, model, cm);
 }
```

- *Adding Tooltips to a Graph*

```
public String getToolTipText(MouseEvent e) {
   if(e != null) {
     // Fetch Cell under Mousepointer
     Object c = getFirstCellForLocation(e.getX(), e.getY());
     if (c != null)
       // Convert Cell to String and Return
       return convertValueToString(c);
   }
   return null;
 }
```

Customizing In-Place Editing

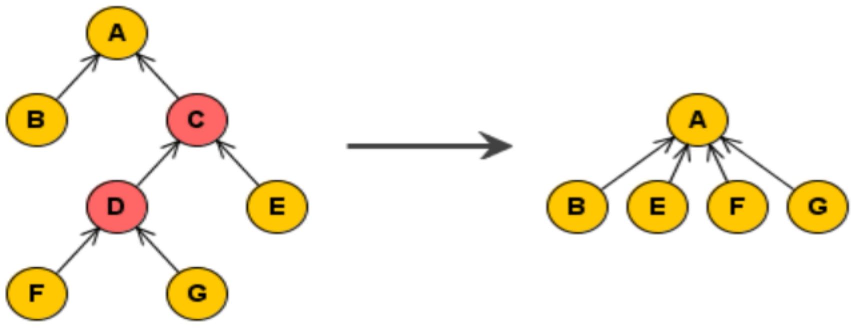
```
        public class DialogGraph extends JGraph {
          // Sets the Custom UI for this graph object
          public void updateUI(){
            // Install a new UI
            setUI(new DialogUI());
            invalidate();
          }
}
```

Dynamically changing a Graph

```
DefaultGraphModel model = new DefaultGraphModel();
JGraph graph = new JGraph(model);
```

Removing Cells from the Model

If a cell is removed from the model, the model checks if the cell has children, and if so, updates the group structure accordingly, that is, for all parents and children that are not to be removed. As a consequence, if a cell is removed with children, it can be reinserted using insert, that is, without providing the children or a ParentMap. If a cell is removed without children, the resulting operation is an "ungroup".



```
Object[] cells = graph.getSelectionCells();
   if (cells != null) {
     // Remove Cells (incl.  Descendants) from the Model
     graph.getModel().remove(graph.getDescendants(cells));
   }
```