Graphs Introduction – Labs

frederic.guinand@univ-lehavre.fr

1 Introduction

For the labs we will use a java library for manipulating graphs. This library allows the visualization of graphs as well as the modification you made on them during the processing. This first session has for main goals to:

1. setting up the environmente of development
2. creation and visualization of two simple graphs,
3. some basic manipulations for modifying the aspect of the graph

2 Installation

2.1 Environment Verification

1. Before starting, verify whether if java is installed on your computer or not:
   → https://www.java.com/en/download/help/version_manual.html
2. if it is not installed yet, then proceed by following the process described here:
   → https://www.java.com/en/download/help/index_installing.html
3. when java is installed check if eclipse or IntelliJ IDEA (Community edition) is installed,
4. if not install one of them:
   → https://www.eclipse.org/downloads/packages/ (choose eclipse IDE for java developers)
   → https://www.jetbrains.com/idea/download/#section=windows

2.2 Testing java without any IDE (optionnal)

1. open a text editor (`notepad` for instance)
2. enter the following code:

```
// Test.java

public class Test {
   public static void main(String args[]) {
       System.out.println("ok javac and java are working well");
   }
}
```

3. save the file with the name Test.java
4. in the command prompt use the command `javac Test.java`

   if the command is not found this might be because the environment variable *PATH* should be modified
5. and then execute it with `java Test`
6. the message *ok javac and java are working well* should be displayed in the console

2.3 Getting and installing the GraphStream library

1. go to https://graphstream-project.org
2. then click on Download GraphStream and Release 1.3
3. get the 3 zip files `gs-core, gs-algo and gs-ui`
4. open the archives and get the 3 corresponding jar files: gs-core-1.3.jar, ...
5. if everything is ok up to now I will show you how to add libraries to a new eclipse project

3 First Steps

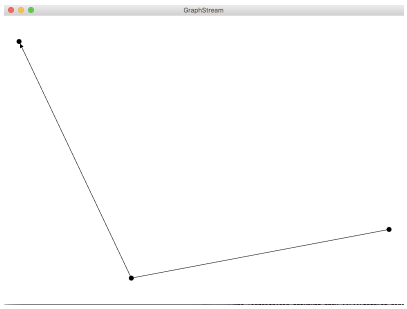
3.1 First Graph

Try this first code that should display a very simple graph composed of only 3 vertices with an edge and oen arc.

```
// FirstGraph.java
import org.graphstream.graph.implementations.SingleGraph;

public class FirstGraph {
    public static void main(String args[]) {
        SingleGraph graph = new SingleGraph("First Graph");
        graph.display();
        graph.addNode("a");
        graph.addNode("b");
        graph.addNode("c");
        graph.addEdge("ab","a","b",true);
        graph.addEdge("ac", "a","c");
    }
}
```

The execution of the code should make appear a window with the graph like on the picture below.



3.2 Basic Methods

- **new SingleGraph("id")** creates an instance of a simple graph.

  on this graph, it is possible to invoke multiple methods for changing the display, its set of vertices and edges (or arcs if the graph is directed).

- **graph.display()** enables graph dispaly. The way vertices are organized is automatically computed in order to make the visibility of the graph the best possible.

- **addNode(idf)** adds a new vertex to the graph. Each vertex is identified by a string

  ⚠ two vertices cannot have the same identifer

- **addEdge(idf, id vertex1, id vertex2, directed)** adds a new edge (if *directed* is false or absent) or a new arc (if *directed* is positioned at true)

3.3 Addition of attributes

When processing graphs it is often interesting to display more information than just its vertices and edges. In the following code, we display also the identifiers of vertices and edges and we change the color of nodes and edges. But for that, we need to walk through all nodes and edges of the graph, what can be done by invoking the methods **getNodeSet()** and **getEdgeSet()** on the instance of the graph. All vertices and nodes wil become resp. red and blue.

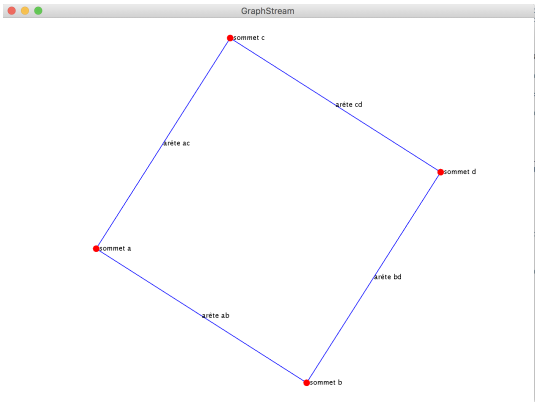
```
// SecondGraph.java
import org.graphstream.graph.implementations.SingleGraph;
import org.graphstream.graph.Node;
import org.graphstream.graph.Edge;

public class SecondGraph {
  public static void main(String args[]) {
    SingleGraph graph = new SingleGraph("Second Graph");
    graph.display();
    graph.addNode("a");
    graph.addNode("b");
    graph.addNode("c");
    graph.addNode("d");
    graph.addEdge("ab","a","b");
```

```
        graph.addEdge("ac","a","c");
        graph.addEdge("cd","c","d");
        graph.addEdge("bd","b","d");

        // we iterate on the set of nodes of the graph
        for(Node n:graph.getNodeSet()) {
          n.setAttribute("ui.style","fill-color:red;");
          n.setAttribute("ui.label","vertex "+n.getId());
        }

        // we iterate on the set of edges of the graph
        for(Edge e:graph.getEdgeSet()) {
          e.setAttribute("ui.style","fill-color:blue;");
          e.setAttribute("ui.label","edge "+e.getId());
        }
    }
}
```



3.4 Basic Methods (cont.)

- **n.setAttribute(”name”,value)** associates an attribute to an element of the graph (vertex or edge).
- some attributes have a predefined name:
- the name of the element (vertex or edge):

  → **e.setAttribute(”ui.label”,val)** where `val` is an instance of String that will displayed next to element e.
- color of the element:

  → **e.setAttribute(”ui.style”,”fill-color:red;”)**
- it is also possible to code the color using the RGB coding using the **#rrggbb** form where `rr` varies between 00 and FF (hexadecimal).
- Conversely it is possible to get the value of an attribute of element e by invoking the method **e.getAttribute(”key”)**

4 Some Simple Operations

4.1 Average Degree of a Graph

Using what has been done in Section 3.3, and using methods presented in Section 3.4 and 4.3, propose a code for computing the average degree of your graph.

4.2 Neighbors

Many graph algorithm need to list all the neighbors of a particular vertex. By using the methods described in Section 4.3 and according to what has been done in the previous sections, choose a random vertex in your graph, change its color to red and color all its neighbors in blue.

4.3 Basic Methods (cont.)

- **graph.getNodeCount()** returns the number of nodes of the graph
- **graph.getEdgeCount()** returns the number of edges of the graph
- **n.getDegree()** returns the degree of vertex n
- method for randomly picking a vertex in the graph:
  - add **import static org.graphstream.algorithm.Toolkit.randomNode;**
  - **Node nodeRandomlyChosen = randomNode(graph)**
- method for listing all neighbors of a vertex:

  the set of neighbors is returned under the form of an Iterator (example of Iterator below):

  Iterator<Node> theNeighbors = vertex.getNeighborNodeIterator()
  - you should also add **import java.util.Iterator;**

Example of use of `Iterator` :

```
import java.util.Iterator;
import java.util.Arrays;

public class ExampleIterator {

   public static void main(String args[]) {
     Iterator<String> theStrings = Arrays.asList(args).iterator();
     while(theStrings.hasNext()) {
       String oneString = theStrings.next();
       System.out.println("--->"+oneString);
     }
   }
}
```