

Wizualizacja grafów

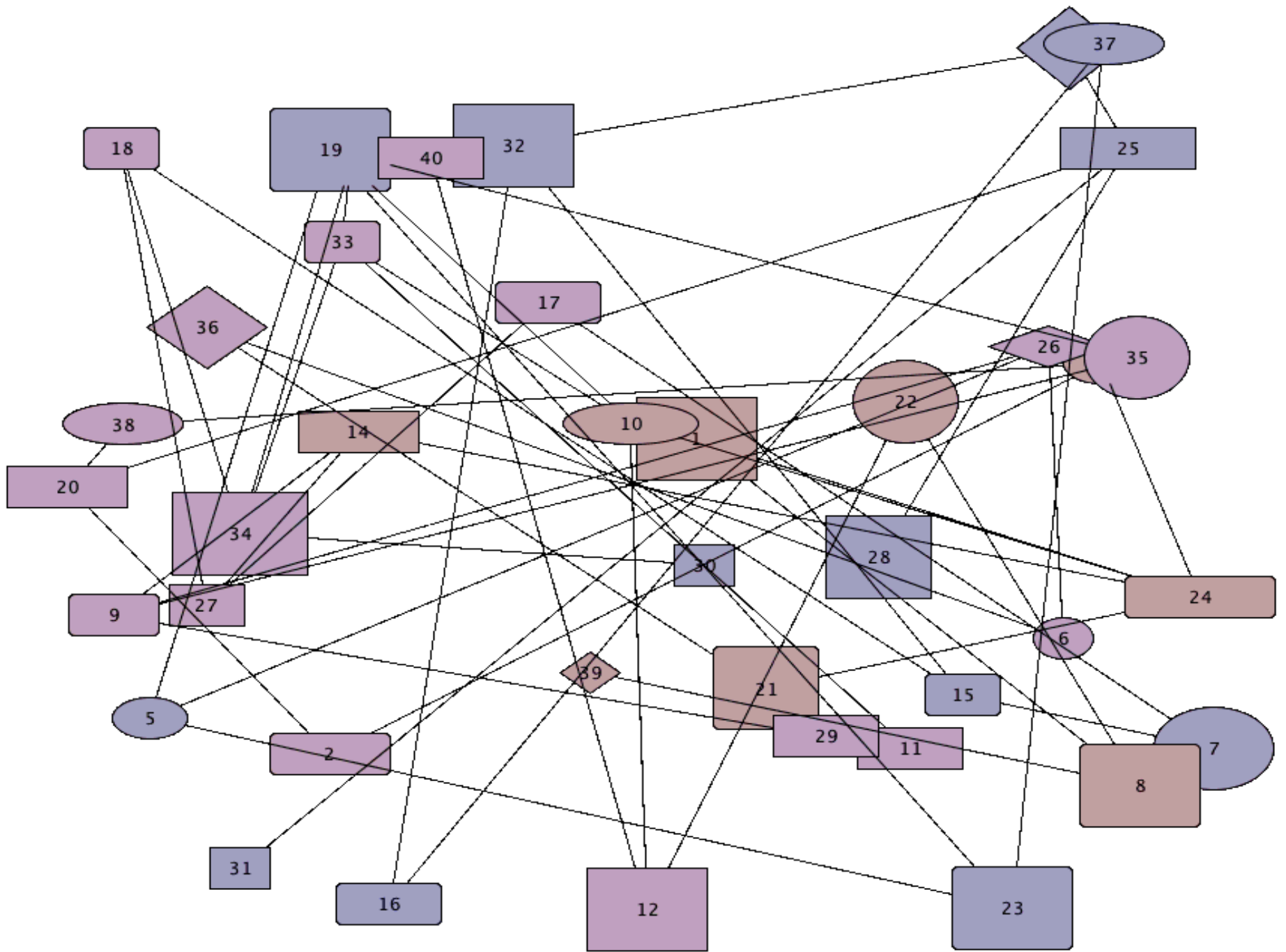
Szymon Matejczyk

Plan prezentacji

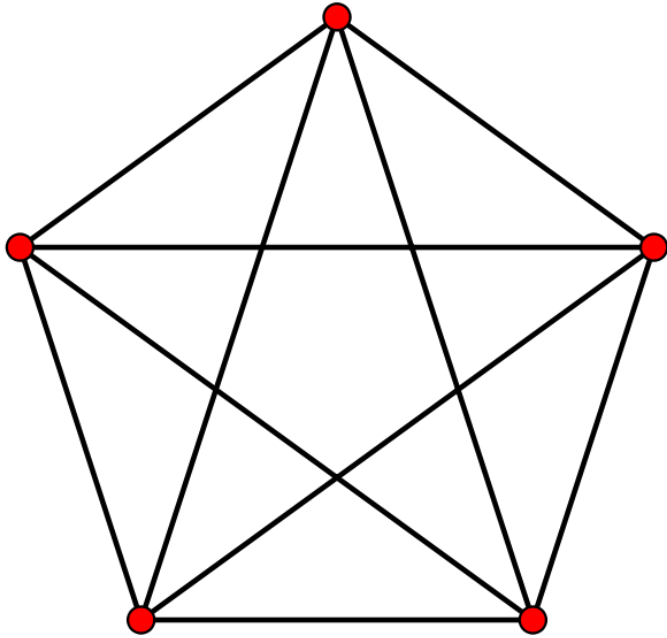
1. Wprowadzenie
2. Zarys teorii
3. Reprezentacja grafów
4. Programy do rysowania (nasze wymagania)
 1. graphViz
 2. diagram.ly
 3. yEd
5. Biblioteki
 1. jgraph (mxGraph)
 2. yFiles
 3. JUNG
6. Podsumowanie

Po co?

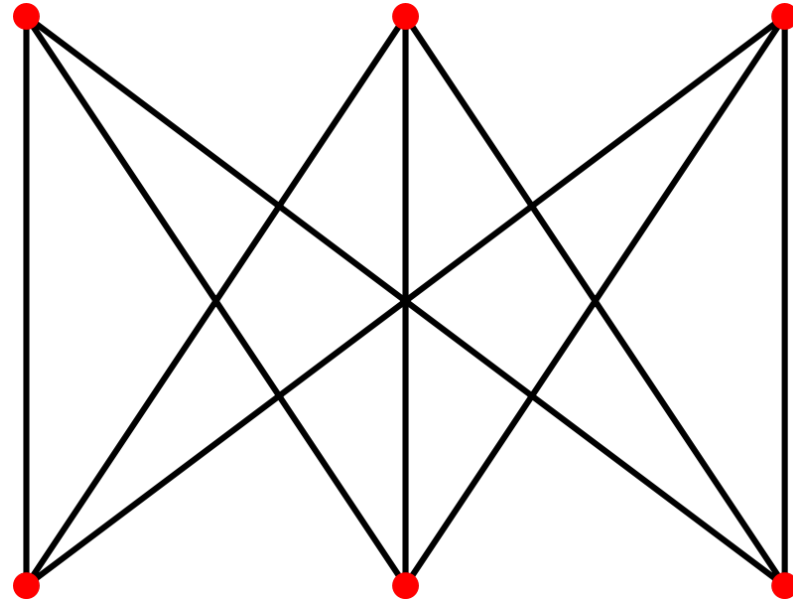
- bioinformatyka
- business process modeling
- data mining
- bazy danych
- zarządzanie sieciami
- sieci społecznościowe
- projekty informatyczne
- wizualizacja WWW
- visual programming



Planarność



K_5



$K_{3,3}$

Graf skończony jest planarny wtedy i tylko wtedy, gdy nie zawiera podgrafu homeomorficznego z grafem K_5 ani z grafem $K_{3,3}$.

Planarność

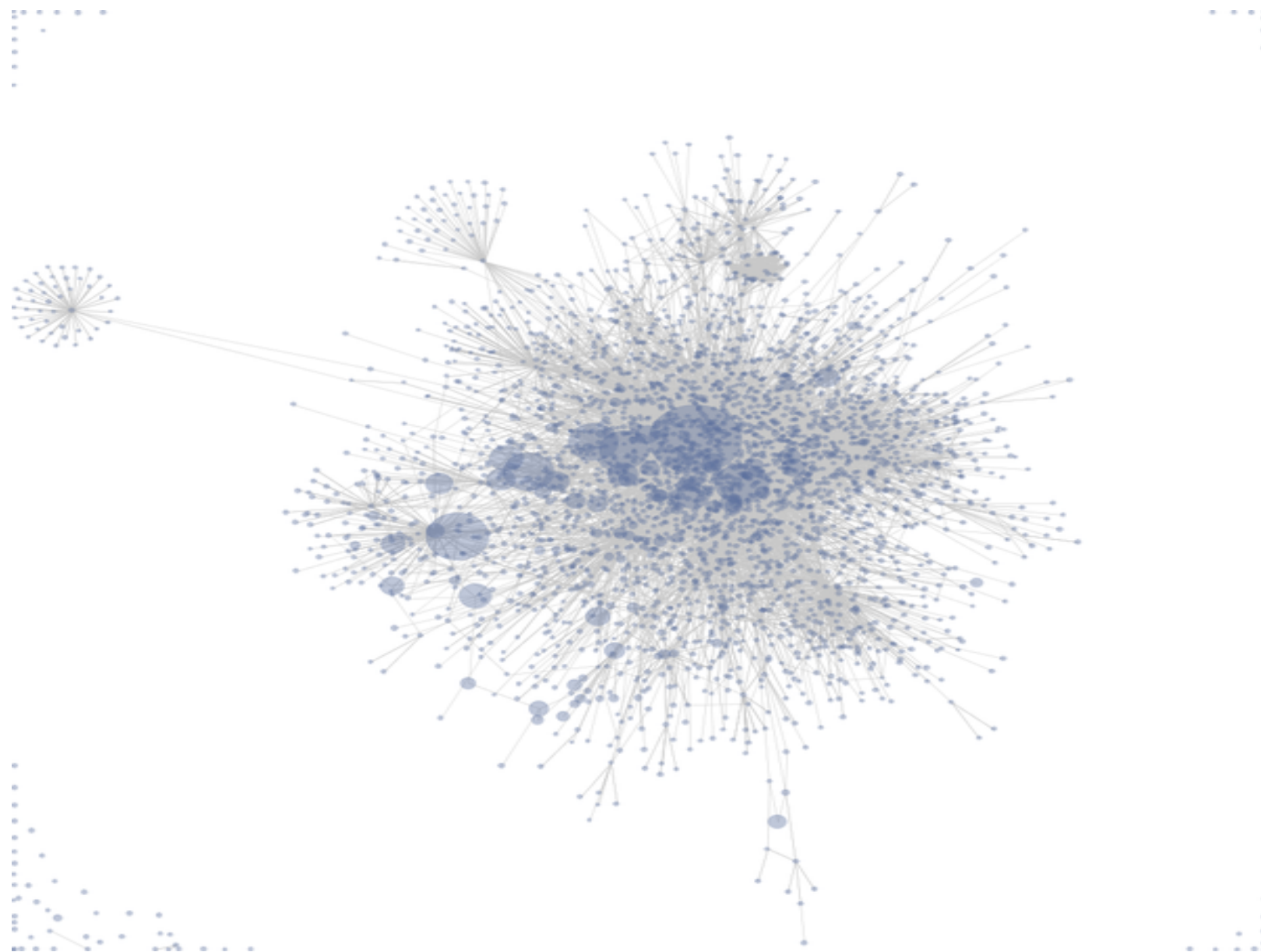
Wzór Eulera

Jeżeli G jest grafem spójnym i planarnym, to $|V| + |S| - |E| = 2$,
gdzie V - zbiór wierzchołków, E - zbiór krawędzi, S - zbiór ścian
dowolnego rysunku płaskiego grafu G .

Wnioski:

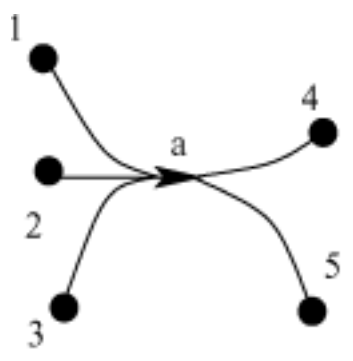
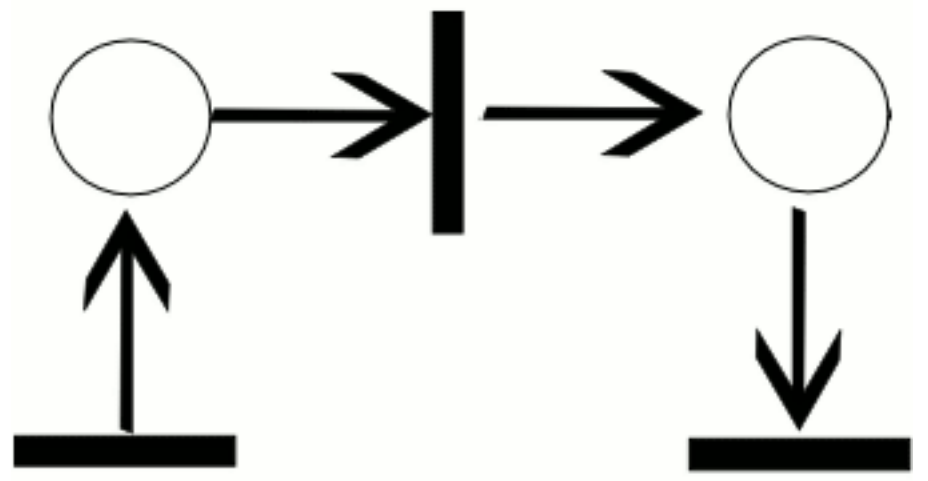
- Jeżeli G jest planarny, to $|E| \leq 3 \cdot |V| - 6$
- Jeżeli G jest planarny, to wierzchołek o najmniejszym stopniu jest stopnia co najwyżej 5.

Force-directed algorithm

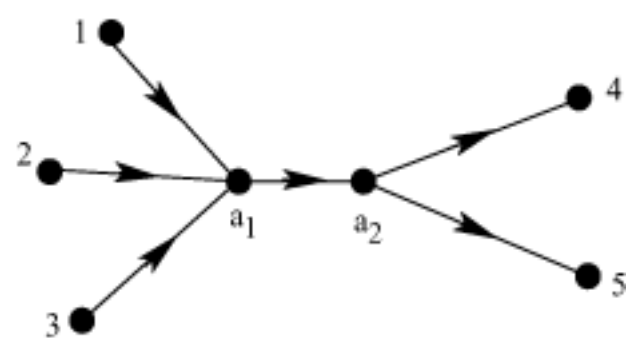


Inne grafy

- sieci Petriego
- hipergrafy
- grafy 3d?
- etykietowanie?



(a)



(b)

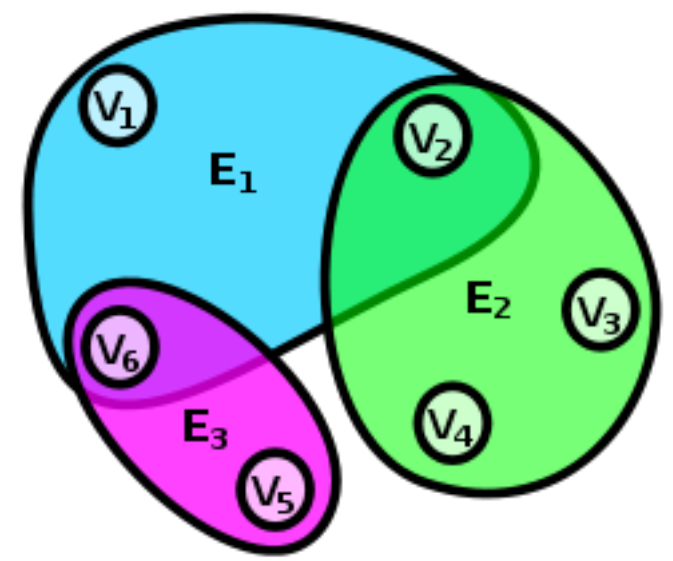


Figure 3 – Grouping origin and destination of a hyperarc.

Reprezentacje grafów: graphml

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml ...>
<key id="d0" for="node" attr.name="color" attr.type="string">
<default>yellow</default>
</key>
<key id="d1" for="edge" attr.name="weight" attr.type="double" />
<graph id="G" edgedefault="undirected">
<node id="n0">
<data key="d0">green</data>
</node>
<node id="n1" />
<node id="n2">
<data key="d0">blue</data>
</node>
<edge id="e0" source="n0" target="n2">
<datakey="d1">1.0</data>
</edge>
<edge id="e1" source="n0" target="n1">
<data key="d1">1.0</data>
</edge>
<edge id="e2" source="n1" target="n3">
<data key="d1">2.0</data>
</edge>
</graph>
</graphml>
```

Reprezentacje grafów: gml

```
graph
[
  hierarchic 1
  label ""
  directed 1
  node
  [
    id 0
    label "1"
    graphics
    [
      ...
    ]
  ]
  node
  [
    id 1
    label "2"
  ]
  edge
  [
    source 25
    target 4
  ]
]
```

Reprezentacje grafów: DOT language

```
digraph unix {  
size="6,6";  
node [color=lightblue2, style=filled];  
"5th Edition" -> "6th Edition";  
"5th Edition" -> "PWB 1.0";  
"6th Edition" -> "LSX";  
"6th Edition" -> "1 BSD";  
"6th Edition" -> "Mini Unix";  
"6th Edition" -> "Wollongong";  
"6th Edition" -> "Interdata";  
"Interdata" -> "Unix/TS 3.0";  
"Interdata" -> "PWB 2.0";  
"Interdata" -> "7th Edition";  
"7th Edition" -> "8th Edition";  
"7th Edition" -> "32V";  
"7th Edition" -> "V7M";  
"7th Edition" -> "Ultrix-11";
```

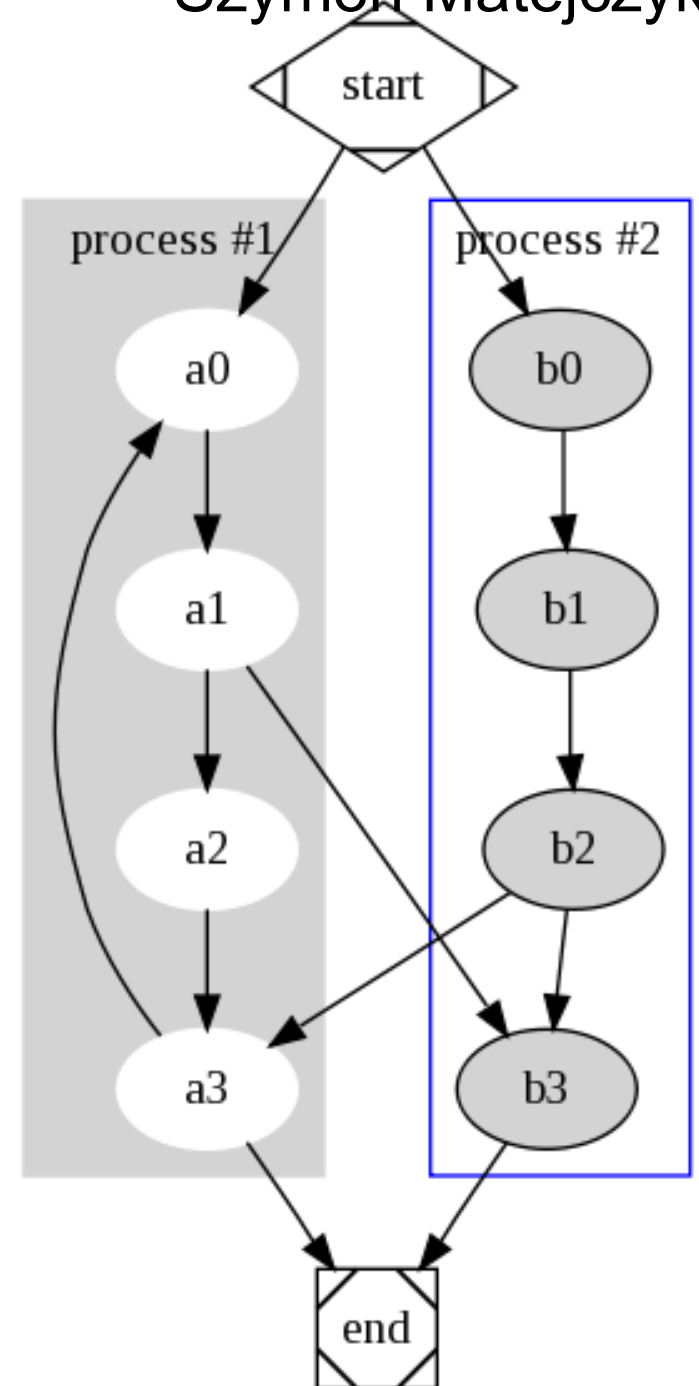
Reprezentacje grafów

Inne:

- gxl
- trivial graph format
- XGGML
- ...

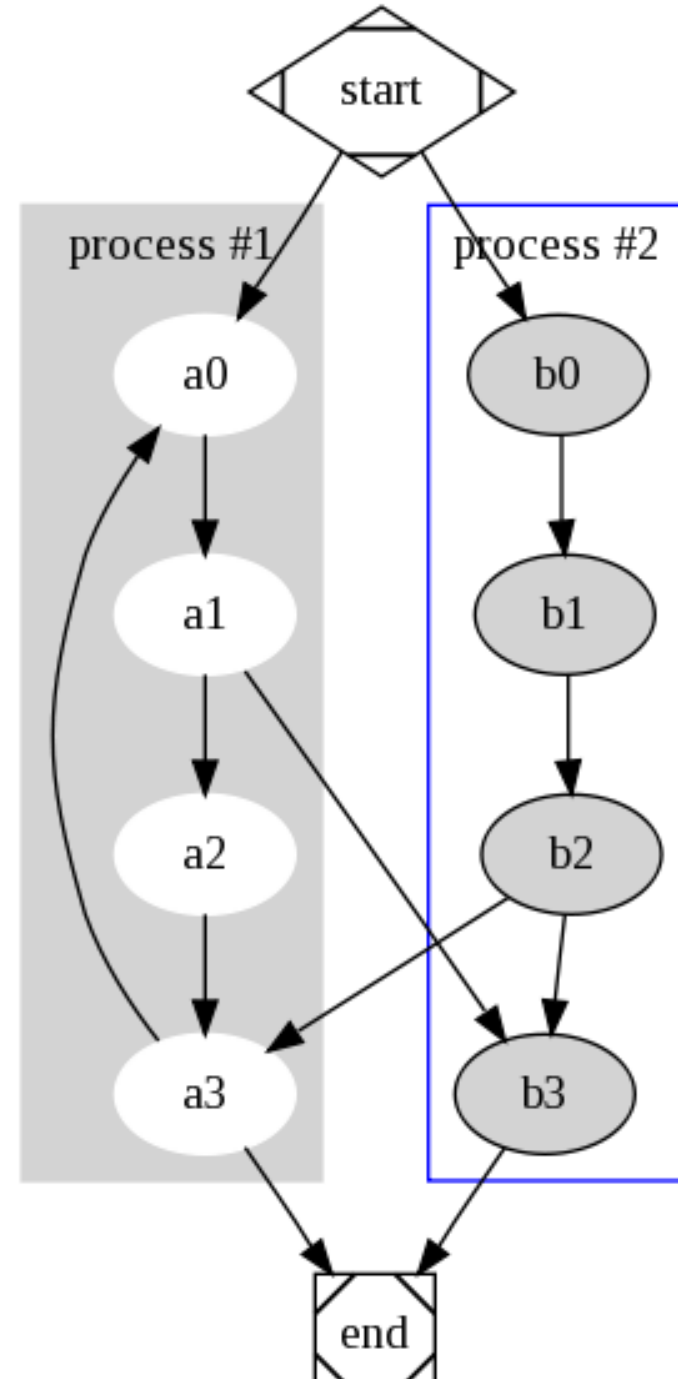
Graphviz

- Open source
- input: DOT language
- obsługa z linii poleceń
- brak edycji
- biblioteki:
 - C
 - Tcl/Tk
- output: svg, PDF, PS
- layouty - pluginy



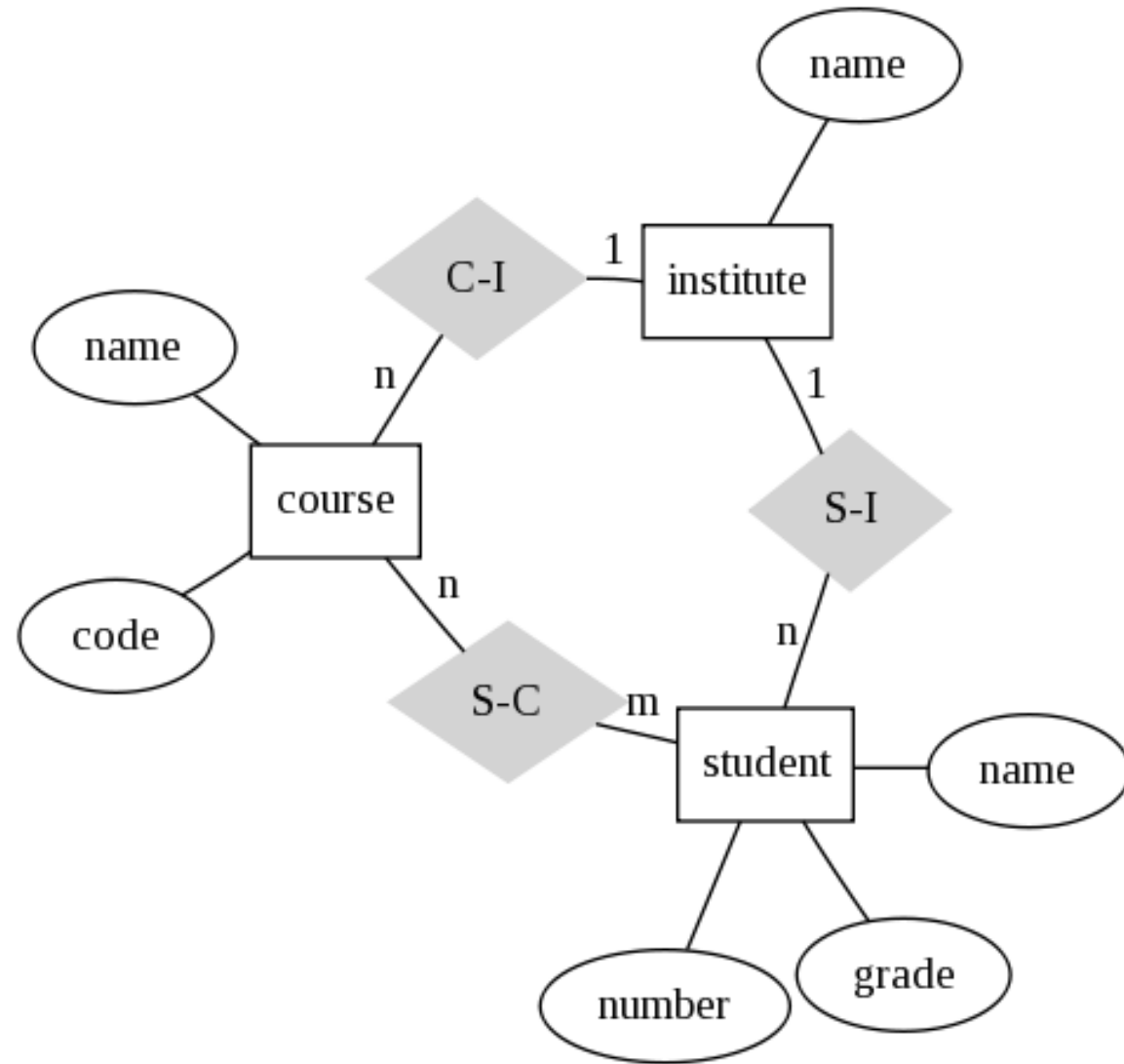
Graphviz - neato

dot - "hierarchical" or layered drawings of directed graphs. This is the default tool to use if edges have directionality. dot aims edges in the same direction (top to bottom, or left to right) and then attempts to avoid edge crossings and reduce edge length.



Graphviz - neato

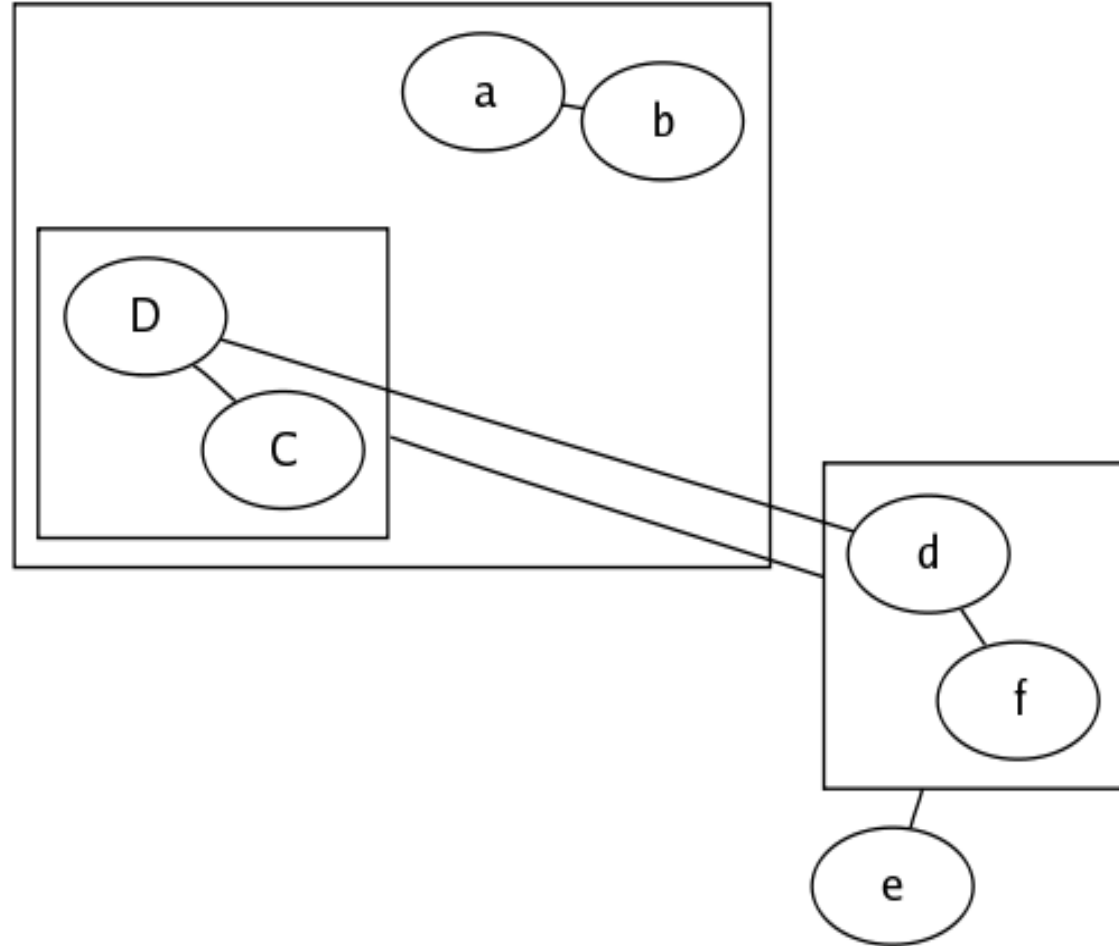
neato - This is the default tool to use if the graph is not too large (about 100 nodes) and you don't know anything else about it. Neato attempts to minimize a global energy function, which is equivalent to statistical multi-dimensional scaling.



Entity Relation Diagram

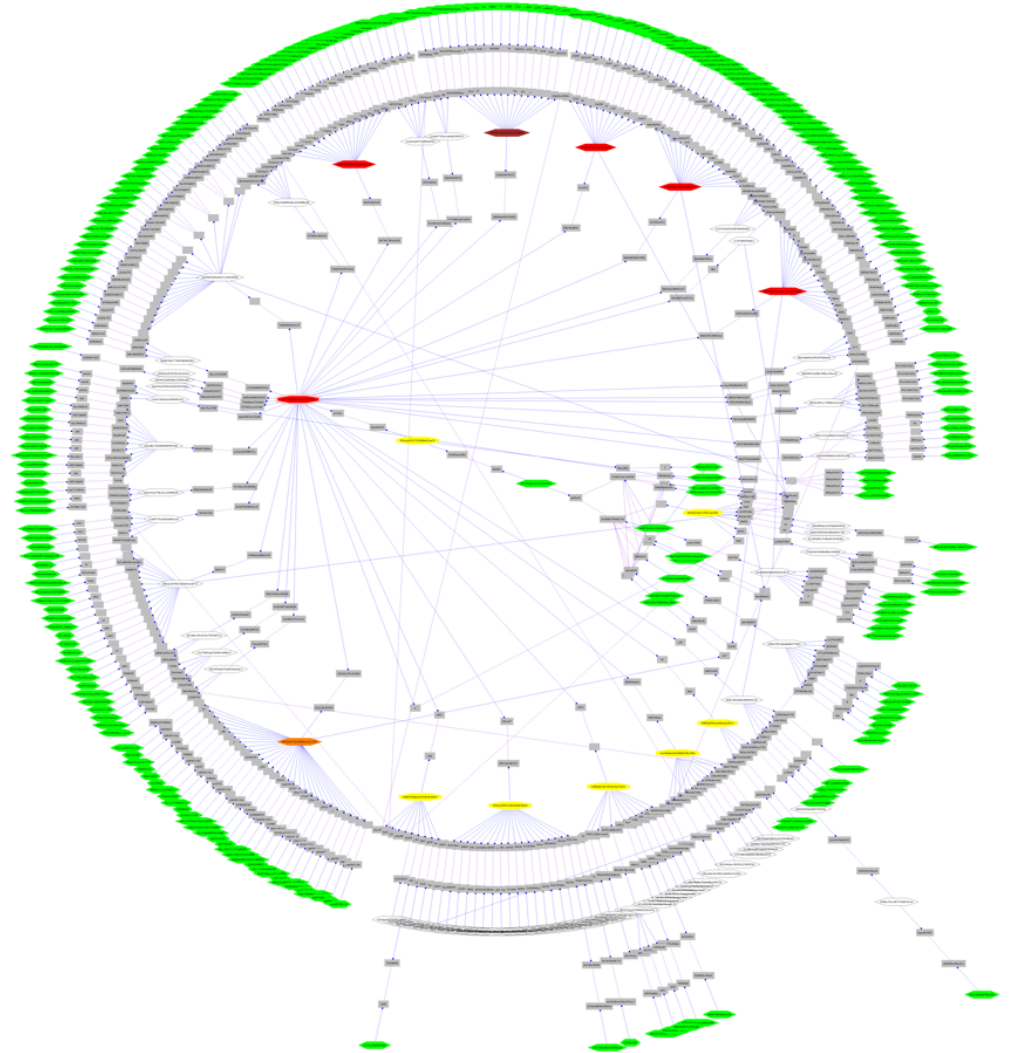
Graphviz - fdp

fdp - "spring model" layouts similar to those of neato, but does this by reducing forces rather than working with energy. Fdp implements the Fruchterman-Reingold heuristic including a multigrid solver that handles larger graphs and clustered undirected graphs.



Graphviz - wopi

wopi - radial layouts, after Graham Wills 97. Nodes are placed on concentric circles depending their distance from a given root node. You can set the root node, or let twopi do it.



Graphviz - circo

circo - circular layout, after Six and Tollis 99, Kauffman and Wiese 02. This is suitable for certain diagrams of multiple cyclic structures, such as certain telecommunications networks.



diagram.ly

- Java script
- input/export: własny xml
- export: jpg, png, svg
- bogate opcje graficzne
- grupy/porty
- różne typy diagramów
- wszystkie przeglądarki
- integracja z php/.Net/AJAX



Wizualizacja grafów

yEd

Szymon Matejczyk



Wizualizacja grafów

yEd

- free
- import: xml, excel, graphml
- export: graphml, graficzne
- automatic layouts
- Windows, Unix, MacOs

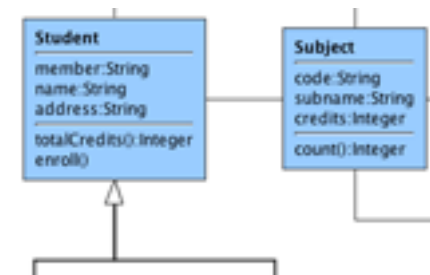
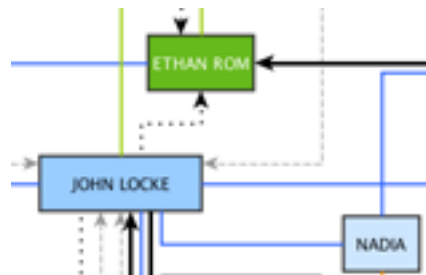
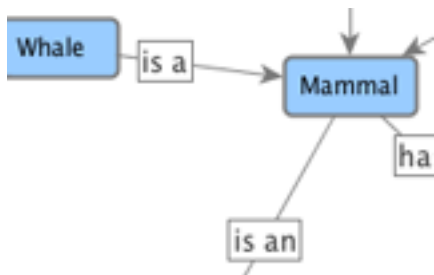
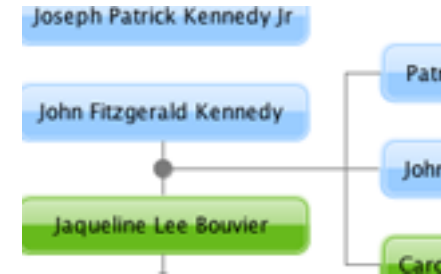
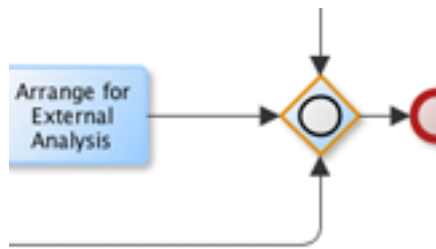
Szymon Matejczyk



Wizualizacja grafów

yEd - supported diagrams

Szymon Matejczyk



Biblioteki - wymagania

- import/export do popularnych formatów
- wspierane typy grafów, grupowanie
- przybliżanie i oddalanie, przesuwanie, itp.
- wbudowany edytor
- reakcja na zdarzenia użytkownika
- dodatkowe dane (labele, opisy, obrazki)
- layouty
- algorytmy grafowe
- wbudowane typy wyświetlania grafu
- wsparcie, dokumentacja
- otwartość kodu
- wspierane systemy
- technologie webowe

jgraph

- Free, open-source (BSD)
- SWING
- wczytywanie z ~CSS
- bogaty zbiór wbudowanych symboli
- import export do własnego xml
- walidacja
- framework JavaScript, diagram.ly
- komercyjne wsparcie
- bardzo skromny manual + przykłady
- tranzakcje

jgraph - grafy

- multigrafy
- porty
- grupy
 - rozwijanie i zwijanie, ukrywanie
 - wspólne wartwy
 - widok tylko danej grupy

jgraph

```
// Adds cells to the model in a single step graph.getModel().beginUpdate(); try { Object  
v1 = graph.addVertex(parent, null, "Hello,", 20, 20, 80, 30); Object v2 = graph.  
addVertex(parent, null, "World!", 200, 150, 80, 30); Object e1 = graph.addEdge(parent,  
null, "", v1, v2); } finally { // Updates the display graph.getModel().endUpdate(); }
```

Wizualizacja grafów

Szymon Matejczyk

jgraph



yFiles

- komercyjny (30 dniowy trial)
- także w .NET
- import/export z graphML, gml
- export do formatów grafiki rastrowej i wektorowej
- notacja UML, BPMN, wbudowane style całych diagramów
- wsparcie dla JS, Silverlight, Flex
- bogaty manual i przykłady



JUNG

The screenshot shows a Chrome browser window displaying the JUNG website. The browser's address bar shows the URL `jung.sourceforge.net/index.html`. The page content includes a navigation menu on the left with links like Overview, Download, Documentation, Examples, Wiki, Projects Using JUNG, FAQ, Support, Team, Presentations, Bug Tracker, Sourceforge, and Acknowledgements. The main content area features an "Overview" section with the following text:

Overview

JUNG – the Java Universal Network/Graph Framework--is a software library that provides a common and extendible language for the modeling, analysis, and visualization of data that can be represented as a graph or network. It is written in Java, which allows JUNG-based applications to make use of the extensive built-in capabilities of the Java API, as well as those of other existing third-party Java libraries.

The JUNG architecture is designed to support a variety of representations of entities and their relations, such as directed and undirected graphs, multi-modal graphs, graphs with parallel edges, and hypergraphs. It provides a mechanism for annotating graphs, entities, and relations with metadata. This facilitates the creation of analytic tools for complex data sets that can examine the relations between entities as well as the metadata attached to each entity and relation.

The current distribution of JUNG includes implementations of a number of algorithms from graph theory, data mining, and social network analysis, such as routines for clustering, decomposition, optimization, random graph generation, statistical analysis, and calculation of network distances, flows, and importance measures (centrality, PageRank, HITS, etc.).

JUNG also provides a visualization framework that makes it easy to construct tools for the interactive exploration of network data. Users can use one of the layout algorithms provided, or use the framework to create their own custom layouts. In addition, filtering mechanisms are provided which allow users to focus their attention, or their algorithms, on specific portions of the graph.

As an open-source library, JUNG provides a common framework for graph/network analysis and visualization. We hope that JUNG will make it easier for those who work with relational data to make use of one another's development efforts, and thus avoid continually re-inventing the wheel.

– The JUNG Framework Development Team

On the right side of the browser window, a sidebar is visible with a "Palette" section containing various graph shapes (circle, triangle, square, pentagon, hexagon, heptagon, diamond, trapezoid) and a "Properties View" section.

JUNG

- open source
- input: graphml, PAJEK
- layouts
- export: png, jpg, eps
- lekki
- brak przykładów i tutoriala - źródła i API dostępne

JUNG

```
Graph<Integer, String> g = new SparseMultigraph<Integer, String>();  
// Add some vertices. From above we defined these to be type Integer.  
g.addVertex((Integer)1);  
g.addVertex((Integer)2);  
g.addVertex((Integer)3);  
// Add some edges. From above we defined these to be of type String  
// Note that the default is for undirected edges.  
g.addEdge("Edge-A", 1, 2); // Note that Java 1.5 auto-boxes primitives  
g.addEdge("Edge-B", 2, 3);  
System.out.println("The graph g = " + g.toString());
```

JUNG

```
public static void main(String[] args) {
    SimpleGraphView sgv = new SimpleGraphView(); //We create our graph in here
    Layout<Integer, String> layout = new CircleLayout(sgv.g);
    layout.setSize(new Dimension(300,300));

    BasicVisualizationServer<Integer,String> vv =
        new BasicVisualizationServer<Integer,String>(layout);
    vv.setPreferredSize(new Dimension(350,350));

    JFrame frame = new JFrame("Simple Graph View");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().add(vv);
    frame.pack();
    frame.setVisible(true);
}
```


Przykłady

- jgraph
 - HelloWorld
 - Port
 - walidacja
- yFiles
 - layout.withoutview.GroupingLayoutWithoutAView
 - layout
 - genealogy
 - orgchart
 - view.realizer
 - io.graphml