Exercise 5. *Deadline: 23 November 2009.*

THE JAVA PROGRAMMING LANGUAGE

GRAPH REPRESENTATIONS

University of Wrocław
Institute of Computer Science *Paweł Rzechonek*

Exercise

A *graph* is an abstract representation of a set of objects where some pairs of the objects are connected by links. The interconnected objects are represented by mathematical abstractions called *vertices*, and the links that connect some pairs of vertices are called *edges*. In computer science, a graph is an abstract data structure that is meant to implement the graph concept. More formally, a graph $G = (V, E)$ is a finite nonempty set V of objects called vertices (we can assume that $V = \{0, 1, 2, \ldots\}$) together with a (possibly empty) set E of unordered pairs of distinct vertices of G called edges. A graph data structure may also associate to each edge some *edge value*, such as a numeric attribute (cost, capacity, length, etc).

The basic operations provided by a graph data structure `G` usually include:

- `G.size()`: tells about the number of vertices in `G`;
- `G.adjacent(x,y)`: tests whether there is an edge from node `x` to node `y`;
- `G.neighbors(x)`: lists all nodes `y` such that there is an edge from `x` to `y`;
- `G.add(x,y)`: adds to `G` the edge from `x` to `y`, if it is not there;
- `G.delete(x,y)`: removes the edge from `x` to `y`, if it is there;
- `G.get_node_value(x)`: returns the value associated with the node `x`;
- `G.set_node_value(x,a)`: sets the value associated with the node `x` to `a`.

Structures that associate values to edges usually provide also:

- `G.get_edge_value(x,y)`: returns the value associated to the edge `(x,y)`;
- `G.set_edge_value(x,y,v)`: sets the value associated to the edge `(x,y)` to `v`≤ 0.

Your task is to define interface `Graph` for mentioned graph operations. Next, create two implementations for the interface: as an *adjacency matrix* for dense graphs (a class `AdjMatrixGraph`), and as an *adjacency lists* for sparse graphs (a class `AdjListsGraph`). A *dense graph* is a graph in which the number of edges is close to the maximal number of edges. The opposite, a graph with only a few edges, is a *sparse graph*.

Finally write a short program, which will test your both graph implementations. Generate a random graph and store it into two representations (adjacency matrix and adjacency lists) and check the graph is connected. A graph $G = (V, E)$ is *connected* if there is a path between all pairs of vertices u and v of V.

Implement the method `toString` in the classes `AdjMatrixGraph` and `AdjListsGraph`.

Hint

Some information about graphs can be found on the webpage:

`http://en.wikibooks.org/wiki/Data_Structures/Graphs`