Assignment 8

Alessandro Artale, Werner Nutt,
Ognjen Savković, Francesco Sportelli

Graphs

1. Implementating Directed Graphs

In this exercise you are asked to write Java classes that implement directed graphs (digraphs), where the vertices of the graphs are numbers. The implementation should follow the "adjacency list approach" from the lecture. There should be methods for

1. adding individual edges (including the nodes of the edges);
2. constructing a graph from a sequence of edges, specified in a text file;
3. printing the nodes with their outgoing edges onto a file.

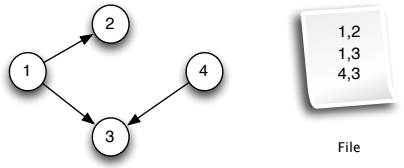
We suggest the following classes:

- A class `Vertex` having the following fields:
  - `int id`: the id of the node;
  - `char color`: that can be 'w', 'g' or 'b';
  - `Vertex pred`: the predecessor vertex (in a spanning tree);
  - `int dist`: the distance from some starting vertex;
  - `VertexList adj`: the vertices reachable from the current vertex traversing an edge;
- A class `VertexList` that suitably reimplements the class `List` from previous exercises to collect `Vertex` objects. A `VertexList` consists of instances of the class `VertexNode`, which are nodes containing a pointer to a `Vertex` and a pointer `next` to a `VertexNode`. The class `VertexList` should support at least the method
  - `Vertex findOrMake(int i)`: search in the `VertexList` for a `Vertex` whose `id` is `i` and return it; if such a vertex is not in the list, create a new `Vertex` having `id` = `i` and an empty adjacency list, add it to the list, and return it.

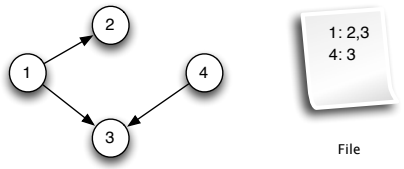
- A class `Graph` having the field:
  - `VertexList vertices`: representing the set of vertices of the graph;

  and at least the following methods:

  - `static Graph readFromFile(String file)`: construct a new graph from a file storing each edge on a different line; for example, the graph on the left is encoded by the file on the right:



  - `void addEdge (int i, int j)`: add an edge from the node having id `i` to the node having id `j`; if the nodes do not exist they are created;
  - `void printToFile(String file)`: print the current graph on the specified file; the output format is shown in the following figure:



  Hint: Suppose you want to construct a graph from a file that contains in each line a pair of numbers, separated by a comma, specifying an edge from the node with the first number to the node with the second number (as described above). Then you have to split each line in two strings to find the two numbers. The Web page with the assignments contains a class `Split.java` with sample code for splitting lines.

(10 Points)

2. Graph Traversal

Add to your `Graph` class the following methods for graph traversal:

- `Graph BFS(Vertex s)`: implementing breadth-first search (BFS) over the graph with start node `s`, and returning a new `Graph`, representing the spanning tree of the BFS traversal;

  (6 Points)

- `Graph recDFS()`: implementing depth-first search (DFS) over the graph as a recursive algorithm; the method returns a new `Graph`, representing the depth-first spanning forest of the traversal;

  (5 Points)

- `Graph itDFS()`: implementing depth-first search (DFS) over the graph by an algorithm that maintains an explicit stack; as the previous algorithm, it returns the spanning forest of the traversal;

  (4 Points)

- `IntList topSort()`: returns a list with a topological sort of the vertex `id`s of the graph, if the graph is acyclic, and returns null otherwise.

  (5 Points)

Please, follow the "Instructions for Submitting Course Work" on the Web page with the assignments, when preparing your coursework.

Also, include name, student ID, code of your lab group (A, B, or C), and email address in your submission.

Submission: Until Tue, 2 June 2015, 11:55 pm, to

`dsa-submissions AT inf DOT unibz DOT it.`