Assignment 9

Mouna Kacimi, Werner Nutt, Camilo Thorne

Graphs

1. Implementating Directed Graphs

In this exercise you are asked to write Java classes that implement directed graphs (digraphs), where 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.

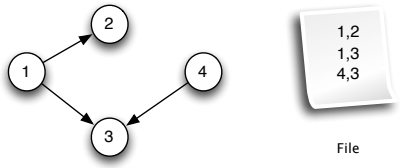
We suggest the following classes:

- A class `GNode` (like "graph node") having the following fields:
  - `int id`: the id of the node;
  - `char color`: that can be 'w', 'g' or 'b';
  - `GNode pred`: the predecessor node (in a spanning tree);
  - `int dist`: the distance from some starting node;
  - `GNodeList adj`: the nodes reachable from the current node traversing an edge;
- A class `GNodeList` that suitably reimplements the class `List` from previous exercises to handle this new kind of `GNode` objects. The class `GNodeList` should support at least the method
  - `GNode findOrMake(int i)`: search in the `GNodeList` for a `GNode` whose `id` is `i` and return it; if such a node is not in the list, create a new `GNode` having `id` = `i` add it the list, and return it.

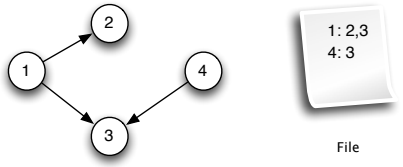
Hint: A `GNodeList`, like the lists we defined earlier, must consist of nodes, however, these nodes are different from the `GNodes`, which are nodes of the graph. We therefore need another class for list nodes, say `LNode`, which has the two fields "`GNode graphNode`" and "`LNode next`". The `next` field supports the list operations, while `graphNode` points to a graph node, which in this way is an element of the list.

- A class `Graph` having the following field:
  - `GNodeList nodes`: representing the set of nodes of the graph;

  and at least the following methods:
  - `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 readFromFile(String file)`: construct the graph from a file storing each edge on a different line; for example, the graph on the left is encoded by the file in the right:



  - `void printToFile(String file)`: print the current graph on the specified file; the output format is shown in the following figure:



(10 Points)

2. Graph Traversal

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

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

  (7 Points)

- `Graph recDFS`: implementing the DFS with a recursive algorithm over the graph; the method returns a new `Graph` representing the spanning tree and prints the topological sort of the nodes as a side effect;

  (7 Points)

- `Graph itDFS`: implementing depth-first search (DFS) algorithm over the graph by an algorithm that maintains an explicit stack; the method returns a new `Graph` representing the spanning tree and prints the topological sort of the nodes as a side effect.

  (6 Points)

Hint: For a DFS that returns a topological sort, you may have to add some fields to the class `Node`.

Submission: Until Thu, 5 June 2014, 8:30 pm, to

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