

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 string;
3. printing the nodes with their outgoing edges onto a string.

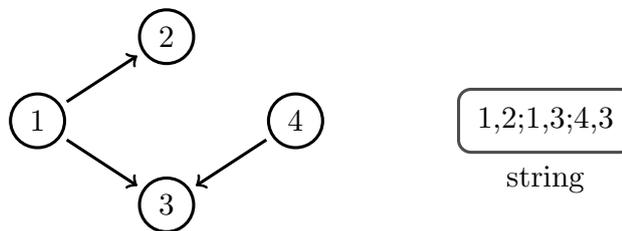
We suggest the following classes:

- 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.

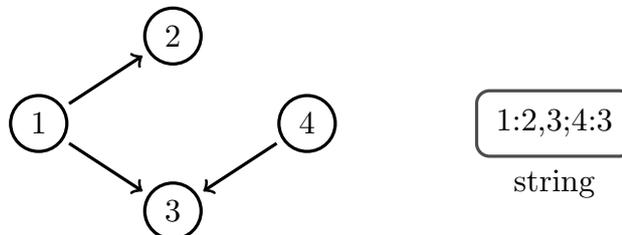
- A class `Graph` having the field:
 - `VertexList vertices`: representing the set of vertices of the graph;

and at least the following methods:

- `static Graph readFromString(String input)`: construct a new graph from a string storing each all edges divided by semi-colons; for example, the graph on the left is encoded by the string 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;
- `String writeToString()`: output the current graph as a string, formatted as shown in the following figure:



Hint: You want to construct a graph from a string that contains pairs of numbers separated by a semicolon. Each pair specifies an edge from the node with the first number to the node with the second number. You have to split the string to find each edge, and the two nodes in the edge. In the `DSA_A9` zip file, you find a class `Split.java` with sample code for splitting strings. You will need to extend this code to use it in the methods.

(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 ids of the graph, if the graph is acyclic, and returns null otherwise.
(5 Points)

Deliverables.

Submit the code for both exercises via the OLE website of the course. Please, follow the “Instructions for Submitting Course Work” on the Web page with the assignments, when preparing your coursework.

Submission: Until Mon, June 13, 2016, 23:55 hrs, to the OLE submission page of

Lab A / Lab B / Lab C