Practical Assignment 2

Available Since: April 29, 2011   Due Date: May 11, 2011, 2:30 p.m.
You are permitted and encouraged to work in groups of two.

Topic of this assignment is the layout of rooted trees.

General Hints

• implement all assignments in the framework from the course website

• create one class for each assignment in your own package:
  gd.assignments.lastname1lastname2
  on change of groups create new packages according to group memberships
  your class should implement the interface gd.material.GDLayouter
  unless mentioned otherwise
  ensure getName() returns the names of the algorithm and group members

• to integrate your class into the GUI, add your class to the list in
  gd.material.MainWindow.layouts

• rescale the coordinates that your algorithm produces

• send your results and questions to uwe.nagel@uni-konstanz.de

In this assignment we will require a number of example trees and in that we will restrict ourselfs to binary trees. To generate those, you can use the code from the class BinaryTreeGenerator which can be downloaded at http://www.inf.uni-konstanz.de/algo/lehre/ss09/gd/projekte/BinaryTreeGenerator.java.
Exercise 1: Circular Layout 4 Points

Create a layout algorithm that distributes the nodes of the given graph equidistant and counter-clockwise on a unit circle. Use the order that is induced by the iterator and begin at coordinate (1, 0).

- set the position in the ordering of each node as its label
- you can use the code `(Graph2D)graph).getRealizer(n).setLabelText(s);` to set the string s as the label of node n in the graph graph

Exercise 2: Pre and Post Order Layouts 6 Points

Implement simple tree layouts using pre and post ordering.

- implement the pre-order algorithm and determine for each node its height in the tree (height meaning distance from root here) in class PreOrder
- derive coordinates from height and pre-order number
- set the number from the ordering as node label
- hint: use a recursion with a mapping from nodes to height as parameter
- assume that graph.firstNode() is the root of the tree

Implement the same layout with a post order - ordering of the nodes in class PostOrder.

Exercise 3: Reingold-Tilford Algorithm 10 Points

Implement the Reingold-Tilford algorithm presented in the lecture. Ensure linear time consumption of your implementation!

- use class name ReingoldTilford
- remember the contours in NodeMaps contourLeft and contourRight
- you will also need maps to save x-offsets with respect to the predecessor in the tree and in the current contour

Test the efficiency of your implementation in a new class RTTest. This class does not need to implement the interface but is required to have a main() method which starts the test.

In the test, create random trees node numbers between 1000 and 25,000 (step width 200). Measure your running times (not those of the generator) and create a chart of them (i.e. using gnuplot) that documents your results. Submit the chart as pdf or postscript via mail together with your other implementations.