

Information Visualization – Exercises

Due date:

Please hand in the solution for this exercise no later than May 19th, 8:00.

Solutions by e-mail to nestler@in.tum.de and huberma@in.tum.de and schlegem@in.tum.de are acceptable.

Exercise 1 (H) Drawing graphs with prefuse

In this exercise we would like to visualize a data set. To do this, we map individual data records into visual items. Prefuse provides a set of visual objects to represent the data. Based on the code of the second lesson you should visualize a small set of a social network. Encode the information about the age of the persons into your visualization.

Choose a data set from the web, which describes the relationship of at least 20 people and visualize it as a relationship graph with encoding of peoples ages.

Exercise 2 (H) Layouting trees with prefuse

This exercise deals with the visualization of trees. Prefuse contains different layout algorithms for the visualization of trees:

- **BalloonTreeLayout**

Layout that computes a circular “balloon-tree“ layout of a tree. This layout places children nodes radially around their parents, and is equivalent to a top-down flattened view of a ConeTree.

The algorithm used is that of G. Melan and I. Herman from their research paper Circular Drawings of Rooted Trees, Reports of the Centre for Mathematics and Computer Sciences, Report Number INS 9817, 1998.

- **NodeLinkTreeLayout**

TreeLayout that computes a tidy layout of a node-link tree diagram. This algorithm lays out a rooted tree such that each depth level of the tree is on a shared line. The orientation of the tree can be set such that the tree goes left-to-right (default), right-to-left, top-to-bottom, or bottom-to-top.

The algorithm used is that of Christoph Buchheim, Michael J and Sebastian Leipert from their research paper Improving Walker’s Algorithm to Run in Linear Time, Graph Drawing 2002. This algorithm corrects performance issues in Walker’s algorithm, which generalizes Reingold and Tilford’s method for tidy drawings of trees to support trees with an arbitrary number of children at any given node.

- **RadialTreeLayout**

TreeLayout instance that computes a radial layout, laying out subsequent depth levels of a tree on circles of progressively increasing radius.

The algorithm used is that of Ka-Ping Yee, Danyel Fisher, Rachna Dhamija, and Marti Hearst in their research paper *Animated Exploration of Dynamic Graphs with Radial Layout*, *InfoVis 2001*. This algorithm computes a radial layout which factors in possible variation in sizes, and maintains both orientation and ordering constraints to facilitate smooth and understandable transitions between layout configurations.

- **ForceDirectedLayout**

Layout that positions graph elements based on a physics simulation of interacting forces; by default, nodes repel each other, edges act as springs, and drag forces (similar to air resistance) are applied. This algorithm can be run for multiple iterations for a run-once layout computation or repeatedly run in an animated fashion for a dynamic and interactive layout.

The running time of this layout algorithm is the greater of $O(N \log N)$ and $O(E)$, where N is the number of nodes and E the number of edges. The addition of custom force calculation modules may, however, increase this value.

The `ForceSimulator` used to drive this layout can be set explicitly, allowing any number of custom force directed layouts to be created through the user's selection of included `Force` components. Each node in the layout is mapped to a `ForceItem` instance and each edge to a `Spring` instance for storing the state of the simulation. See the `prefuse.util.force` package for more.

Remove some edges from the graph of exercise 1 if the graph is not circle-free and / or add some edges to the graph if it is not connected in order to get a tree. Layout the tree with the four different algorithms and discuss them with a friend.