Assignment 4

Available Since: 31 May 2013   Due Date: 13 June 2013, 08:00 a.m.
You are permitted and encouraged to work in groups of two.

Exercise 1: st-Ordering

Consider the algorithm for computing an st-ordering that was discussed in the lecture (Algorithm 4 on page 39 of the lecture notes). In function process_ears the inner vertices of the path $P: u \rightarrow v \leftarrow w$ were inserted directly before or after $u$ into the list $L$ representing the st-ordering. Give an example to prove that the algorithm would not necessarily compute an st-ordering if the inner vertices of $P$ were directly inserted before or after $w$.

Exercise 2: Topological Sorting

A topological sorting of a directed graph $G = (V, E)$ is an ordering of the vertices $v_1, \ldots, v_n$ such that $i < j$ for all edges $(v_i, v_j) \in E$, that is, the source of any edge has a lower number than its target.

(a) Show that a directed acyclic graph has at least one source, i.e., a vertex of indegree zero.
(b) Show that a directed graph is acyclic if and only if it admits a topological sorting.
(c) Develop an efficient algorithm that computes a topological sorting of a directed graph if one exists and analyze its run time.

Exercise 3: Strongly Connected Graphs

A directed graph is strongly connected if for any pair $v, w$ of vertices there exists a directed path from $v$ to $w$. Prove or disprove the following statements.

(a) The edges of a biconnected undirected graph can be oriented such that the resulting directed graph is strongly connected.
(b) The underlying undirected graph of a strongly connected graph is biconnected.
Exercise 4: Biconnected Components 5 Points

Implement the algorithm for determining biconnected components (Algorithm 3 on page 33 of the lecture notes). Ensure linear time complexity.

- Highlight affiliations to individual components by different colors. Also, mark the cut vertices.

- To visualize your results, you can use the following command to layout the graph:
  ```java
  new OrganicLayouter().doLayout(graph);
  ```

- For changing the node/edge color, you can use, e.g., the following command:
  ```java
  ((Graph2D).graph).getRealizer(node).setFillColor(Color.red)
  ((Graph2D).graph).getRealizer(edge).setLineColor(Color.red)
  ```

- Make sure that the line width is thick enough:
  ```java
  EdgeRealizer().setLineType(LineType.createLineType(...))
  ```

Test the efficiency of your implementation by generating random networks with `material.groupZero.u04.RandomGraphGenerator` which is a GNP random graph generator. Note that by using probability \((\log n)/n\), the resulting graph is connected with high probability. Measure your running times and create a plot (i.e. using gnuplot, R, Excel) that documents your results. Submit the plot with theoretical exercises and via svn.

Note the general hints given on assignment 2.