CSC 212—Algorithms and Complexity
Digraph manipulation
Practical 1 Term 4 5 September 2016

Submit by 23h59 11 September 2016.

1. In your home directory, inside your `surname,firstname` directory, create a new directory called `14practical`. Copy any previous practical into this new directory and edit the files to contain this week's practical. Ensure that the directory contains an appropriate `Makefile`. Work inside your `14practical` directory.

2. **Develop** a program to accept graphs and digraphs with the specifications given here. Use an adjacency list to represent your graphs. Your program must be compatible with `dot`, which differentiates between a `graph` and a `digraph`. Our syntax describes `graphs` and their semantics. A `<graph>` is:
   `<graph>`→`<graphs> <name> "{" <arcList> "}"`
   `<graphs>`→`"graph"` | `"digraph"`
   where a `graph` is an undirected graph using only `--` for its arcs, and a `digraph` is a *directed* graph using only `->` arrows for its arcs. Your code should allow arcs only appropriate to its graph type.
   An undirected arc has the form: `<undArc>`→`<name> "--" <name> ";"`
   A directed arc has the form: `<dirArc>`→`<name> "->" <name> ";"`
   Arcs in general are given by: `<arc>` → `<undArc>` | `<dirArc>`
   An `<arcList>` has the form: `<arcList>`→`<arc>` | `<arcList> <arc>`
   A `<name>` is any sensible string that names a node.
   An example of an undirected graph is:

   ```
   graph G {
     A -- B; A -- C; A -- D;
     B -- C;
     C -- D; }
   ```

   An example of a directed graph is:

   ```
   digraph G {
     A -> B; A -> C; A -> D;
     B -> D; B -> C;
     C -> D; }
   ```

3. **Test** that your code can read these two simple graphs Fix the given example if it does not conform to the specifications.

4. **Your code** reads in some *digraph*s, checks that each is acyclic, and if it is acyclic it must display the nodes in topologically sorted order.

5. Since there is often not a unique starting point or ending point, your code must **add a unique starting node** called `start` that points to each of the nodes with an indegree of zero **and another node** called `end` to which all the nodes with outdegree zero must point. This augmented graph must be sorted topologically.

6. *Hand in by running* `cd; make submit` with today's work before leaving the practical. If no submission is made *today*—it need not be fully functional—you will be scored zip for this practical. Final submissions may be made before 23h59 11 September 2016.