Experimental Math Notes: Reordering a DAG

Phil Benjamin

This note has Maple code which reorders the vertices of a Directed Acyclic Graph (DAG) so that all edges are from higher numbered nodes to lower numbered nodes. That is this always possible to do relies on the Order Extension principle, i.e. that any partial order can be extended to be a total order. This is a deep theorem for infinite sets but for finite sets the proof is easy and there are algorithms, called Topological Sorting, that run in linear time.

For finite DAG's, one approach is to assign levels to each vertex in the graph as follows:

- For a sink (vertex with no outgoing edges), assign $Level(v) = 0$.
- Remove all the sinks from the graph, then for all new sinks created, assign $Level(v) = 1$.
- Continue until all vertices have an assigned a level.

From this level assignment, it is clear that all edges will go from higher level vertices to lower level vertices, *assuming that the graph is acyclic*. [This should be checked by any program that implements this algorithm!].

Next order the vertices according to assigned level and renumber all vertices and edges according to this numbering scheme. The resulting graph will have the desired property, that is, that all edges will go from a higher numbered vertex to a lower numbered vertex.

<u>Maple Programs</u>

The appendix to this note lists three Maple programs that implement the desired graph rearrangement. For this purpose, our graph data structure is:

- Vertices of the graph are labeled 1, 2, … n
- Vertices are the indices of a list. Each member of the list is a vertex. This element of the list represents an outgoing edge from the vertex.

For example, suppose the graph contains these edges:

$$1 \to 2, 1 \to 3, 2 \to 4, 3 \to 4$$

Then the graph would be represented as $[\{2,3\}, \{4\}, \{4\}, \{\ \}]$

Program	Description
DAGlevel	From a DAG, create a list of levels assigned to each vertex (Fail if the graph is not a DAG!)
LevelOrder	From a list of levels, create an ordering of the vertices with levels from low to high.
GRperm	From a specified graph and permutation, construct an isomorphic graph.

```
# Procs for DAG

Help := proc() print(`DAGlevel(DAG) LevelOrder(Lorder)
GRperm(GR,Perm`); end:

# DAGlevel
# Input: DAG as list of vertices with sets of outedges
# Output: Levels of vertices satisfying:
#    Level(sink) = 0
#    Level(v) = 1 + max(Level(outedges))

DAGlevel := proc(DAG)
     local n, v, e, doMore, Level;

     n := nops(DAG);
     Level := [0$n];
     doMore := true;
     while doMore do
          doMore := false;
          for v from 1 to n do
               if DAG[v] = {} then next; end;
               for e in DAG[v] do
                    if Level[e] >= Level[v] then
                         Level[v] := Level[e] + 1;
                         if Level[v] >= n then
                              print(`Not a DAG!`);
                              return FAIL;
                         end;
                         doMore := true;
                    end;
               end;
          end;
     end;
     return Level;
end:

# From a list of levels for vertices, produce another list of
the vertices
# in non-decreasing level order
# Example: Level = [3,3,1,0,2], then output will be [4,3,5,1,2]
# Note: vertices 1 and 2 could be in either order since they
have the same level

LevelOrder := proc(Level)
     local L, n, i;

     n := nops(Level);
     L := [seq([Level[i],i],i=1..n)];
     L := sort(L);
     return [seq(L[i][2],i=1..n)];
end:

# Given a graph and a Permutation of the vertices (as produced
by LevelOrder),
# create a renumbered (but isomorphic) graph

GRperm := proc(GR,Perm)
     local n, v, e, InvPerm, GRout;

     n := nops(Perm);
     InvPerm := [0$n];
     for v from 1 to n do InvPerm[Perm[v]] := v; end;
     GRout := [{}$n];
     for v from 1 to n do
          for e in GR[v] do
               GRout[InvPerm[v]] := GRout[InvPerm[v]] union
{InvPerm[e]};
          end;
     end;
     return GRout;
end:
```