General Algorithms on a DAG

Here DAG = Directed Acyclic Graph

At the highest level, a DAG algorithm typically looks like the following

Given graph G=(V,E) and start node s, to calculate a property “prop” for each node

```
1) for each v in V, initialize v.prop
2) initialize s.prop
3) determine topological order of G (may already be known)

4)
for each u in V, taken in topologic order
    for each v such that (u,v) is an edge
        adjust v.prop based on u.prop

5) (optional) for a specified target node t, return t.prop
```

More specifically for homework 2

In this case the topological order is 1,2,3,…,N. Start node is node 1 and target node is node N. Here we will outline how to compute the number of paths starting at node 1, which will be stored in an array NumPath[1..N]. (Also note that the inside loop on j implicitly assumes that we’re using an adjacency matrix representation.)

```
initialize
NumPath[1]=1
for i = 2 to N
    NumPath[i] = 0

loop
for i = 1 to N-1
    for j=i+1 to N
        if (i,j) is an edge
            then NumPath[j] = NumPath[i] + NumPath[j]
               (update longest path and shortest path values here also)

return
print “number of paths is”  NumPath[N]
```