2 Cycles and DAGS

Scheduling problems

A widely applicable problem-solving model has to do with arranging for the completion of a set of jobs, under a set of constraints, by specifying when and how the jobs are to be performed. The most important type of constraints is precedence constraints, which specify that certain tasks must be performed before certain others.

A diagram of a computer program

Description automatically generated

Precedence-constrained scheduling problem. Given a set of jobs, with precedence constraint, how can schedule jobs such that all are completed? This amount to:

Topological sort. Given a digraph, put the vertices in order such that all its directed edges point from a vertex earlier in the order to a vertex later in the order.

A diagram of a machine learning process

Description automatically generated

Directed cycle detection problem. Does a given digraph have a directed cycle? If so, find it.

A directed acyclic graph (DAG) is a digraph with no directed cycles.

Algorithm, Directed Cycle Detection

:start

global marked[]

global stack // cycle

global onStack[]

dfs(graph, source)

onStack[source] = true

marked[source] = true

for adjV in graph.adj(source)

if stack!=null

there is a cycle

else if !marked[source]

dfs(graph, adjV)

else if onStack[w]

there is a cycle

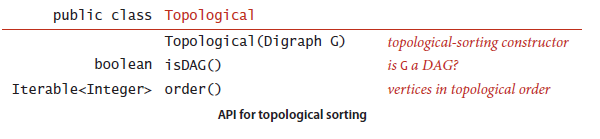
create cycle if necessary

end if

onStack[source] false

:end

Depth-first orders and topological sort



Theorem A digraph has a topological order if and only if it is a DAG.

Algorithm Topological Sort

Add to dfs a single line to remember the given vertex in each call.

There are variations to store the visited vertices in dfs:

Preorder : Put the vertex on a queue before the recursive calls.

Postorder : Put the vertex on a queue after the recursive calls.

Reverse postorder : Put the vertex on a stack after the recursive calls.

A screenshot of a computer

Description automatically generated