Depth First Search

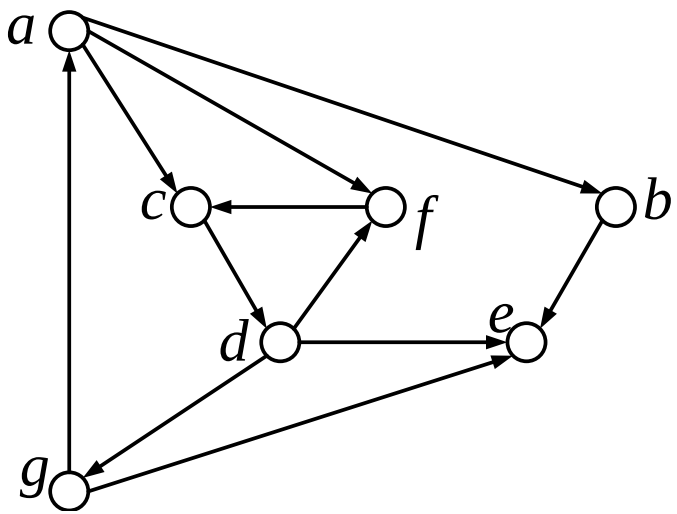
CSCI3160 Tutorial 8

Introduction

You should have learned the depth first search (DFS) algorithm in the "data structure" course. This is a surprisingly powerful algorithm that can solve several non-trivial graph problems. We will review two of those problems today: cycle detection and topological sort.

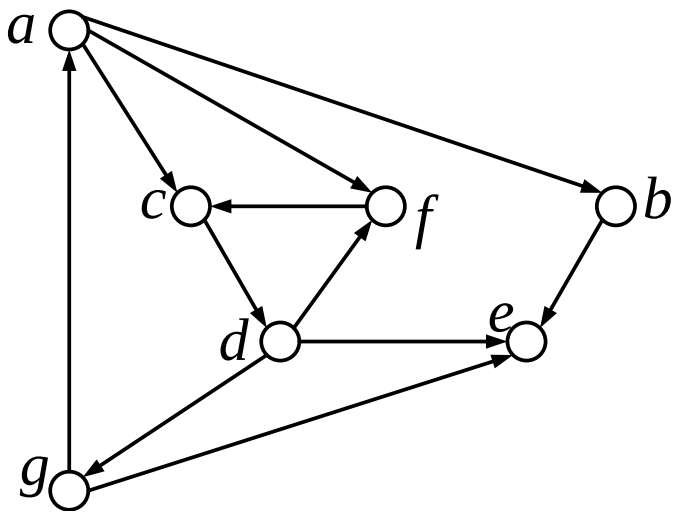
We will not include proofs for the theorems in this talk. Prof. Yufei Tao's offering of CSCI2100 (data structures) covered all these proofs. For thorough understandings, you may refer to the course homepage of his last offering:
http://www.cse.cuhk.edu.hk/∼taoyf/course/2100/18-fall.
Those proofs will not be tested in this course.

Problem (cycle detection): Given a directed graph $G = (V, E)$, determine whether there is a cycle.



The answer is yes.

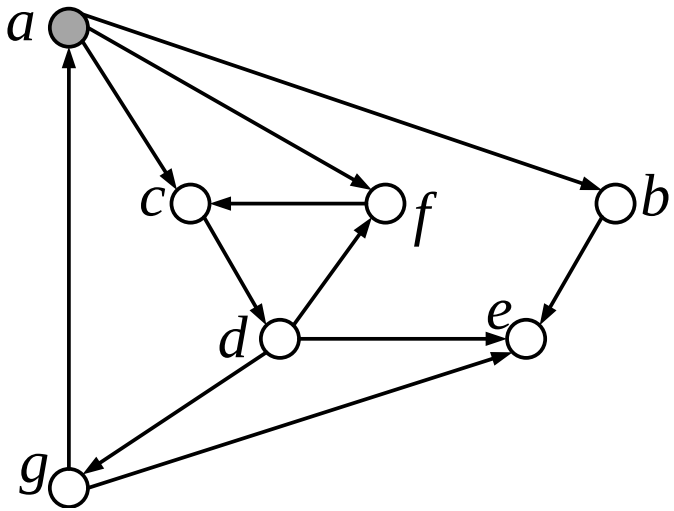
Let us see how DFS can be used to solve the problem.



Choose an arbitrary start vertex, say, *a*. We will perform DFS from *a*, and build a DFS-tree.

DFS

Firstly, create a stack S, push the starting vertex a into S and color it gray. Create a DFS Tree with a as the root. We also maintain the time interval $I(u)$ of each vertex u.

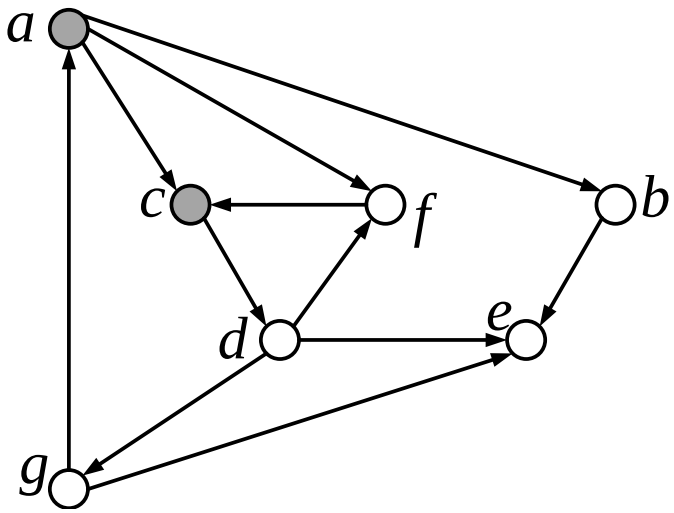


DFS Tree	Time Interval
a	$I(a) = [1, \ ]$

$S = (a)$.

DFS

Top of stack: a, which has white out-neighbors b, c, f. Suppose we access c first (ordering does not matter). Push c into S.

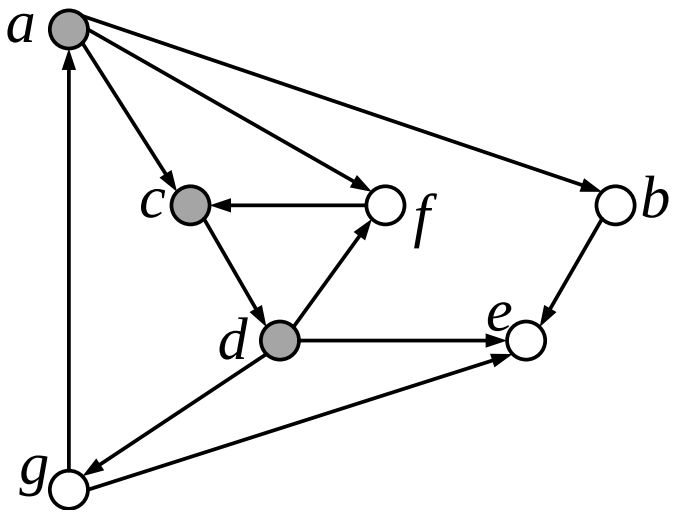


DFS Tree	Time Interval
a	$I(a) = [1, \ ]$
|	
c	$I(c) = [2, \ ]$

$S = (a, c)$.

DFS

After pushing d into S:

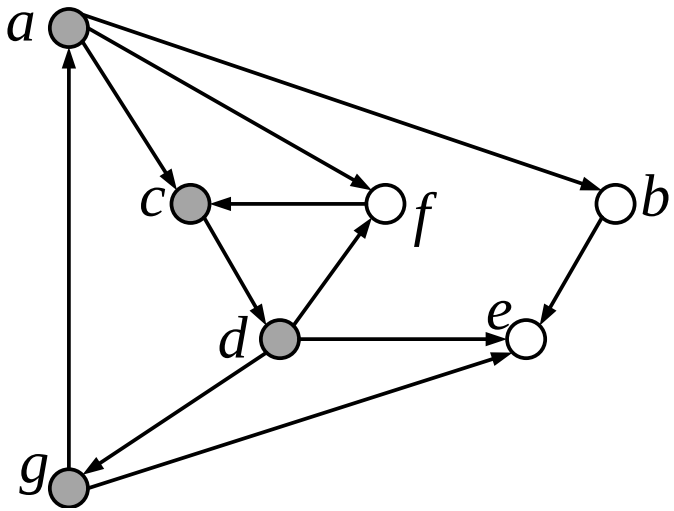


DFS Tree	Time Interval
a	$I(a) = [1, \ ]$
|	
c	$I(c) = [2, \ ]$
|	
d	$I(d) = [3, \ ]$

$S = (a, c, d)$.

DFS

Now d tops the stack. It has white out-neighbors e, f and g. Suppose we visit g first. Push g into S.

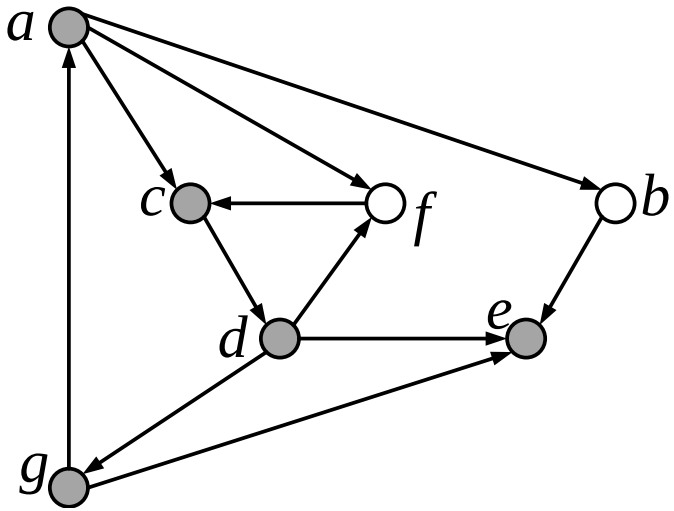


DFS Tree	Time Interval
a	$I(a) = [1, \ ]$
$\vert$	
c	$I(c) = [2, \ ]$
$\vert$	
d	$I(d) = [3, \ ]$
$\vert$	
g	$I(g) = [4, \ ]$

$S = (a, c, d, g)$.

DFS

After pushing e into S:

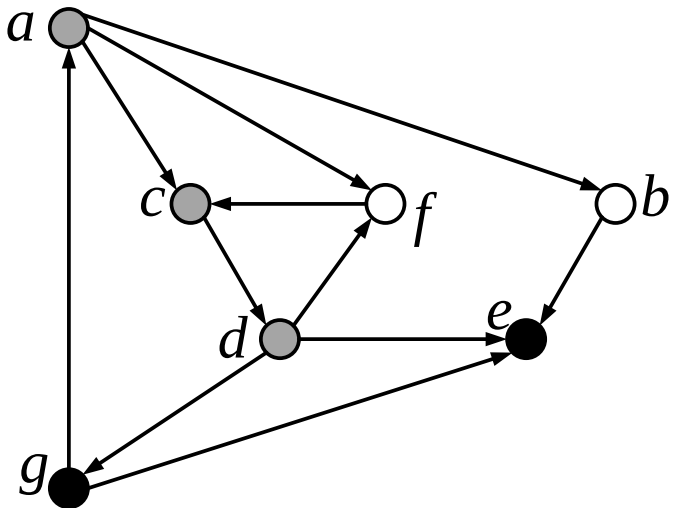


DFS Tree	Time Interval
a	$I(a) = [1, \ ]$
$\vert$	
c	$I(c) = [2, \ ]$
$\vert$	
d	$I(d) = [3, \ ]$
$\vert$	
g	$I(g) = [4, \ ]$
$\vert$	
e	$I(e) = [5, \ ]$

$S = (a, c, d, g, e)$.

DFS

e has no white out-neighbors. So pop it from S, and color it black.
Similarly, g has no white out-neighbors. Pop it from S, and color it black.

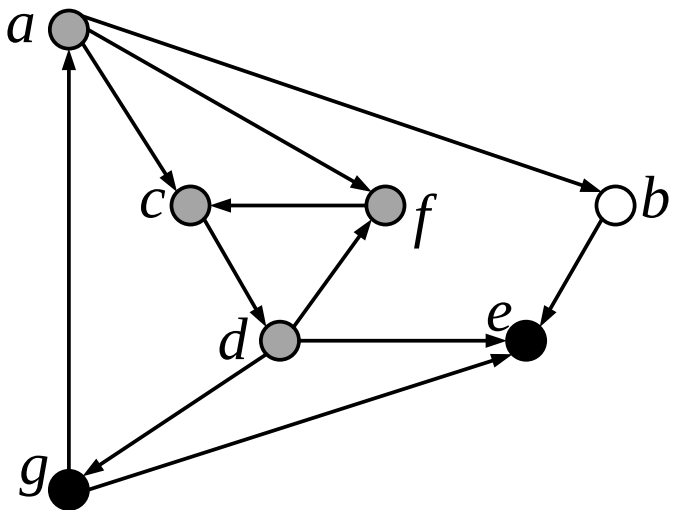


DFS Tree	Time Interval
a	$I(a) = [1, \ ]$
c	$I(c) = [2, \ ]$
d	$I(d) = [3, \ ]$
g	$I(g) = [4, 7]$
e	$I(e) = [5, 6]$

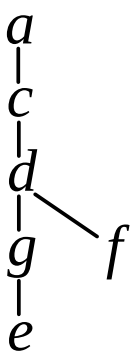
$S = (a, c, d)$.

DFS

Now d tops the stack again. It still has a white out-neighbor f. So, push f into S.



DFS Tree



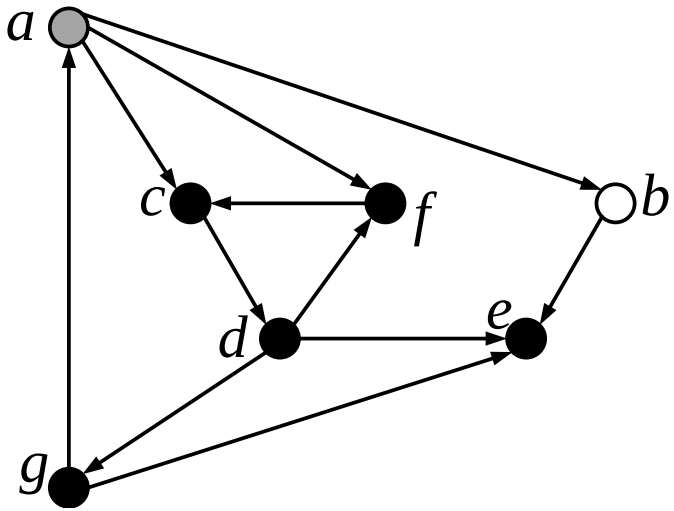
Time Interval

$I(a) = [1, \ ]$
$I(c) = [2, \ ]$
$I(d) = [3, \ ]$
$I(g) = [4, 7]$
$I(e) = [5, 6]$
$I(f) = [8, \ ]$

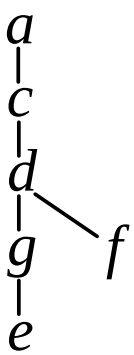
$S = (a, c, d, f)$.

DFS

After popping f, d, c:



DFS Tree



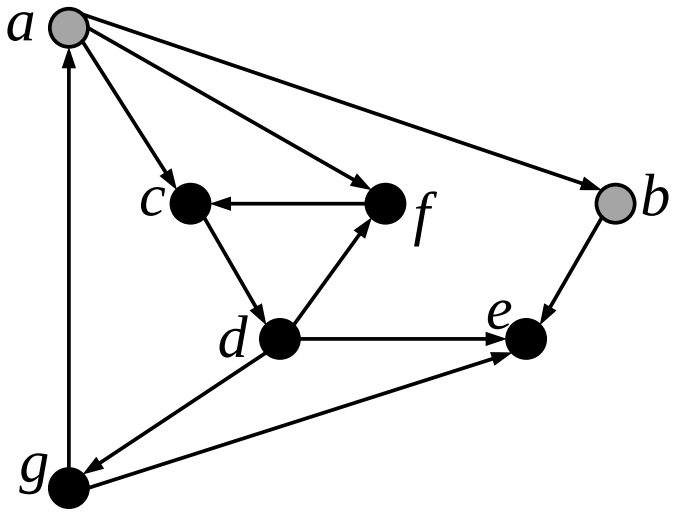
Time Interval

$I(a) = [1, \ ]$
$I(c) = [2, 11]$
$I(d) = [3, 10]$
$I(g) = [4, 7]$
$I(e) = [5, 6]$
$I(f) = [8, 9]$

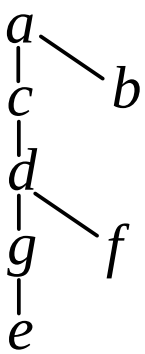
$S = (a)$.

DFS

Now a tops the stack again. It still has a white out-neighbor b. So, push b into S.



DFS Tree



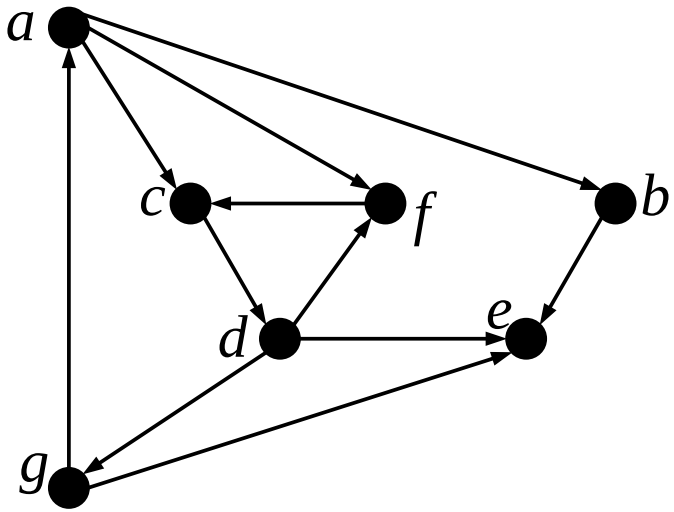
Time Interval

$I(a) = [1, \ ]$
$I(c) = [2, 11]$
$I(d) = [3, 10]$
$I(g) = [4, 7]$
$I(e) = [5, 6]$
$I(f) = [8, 9]$
$I(b) = [12, \ ]$

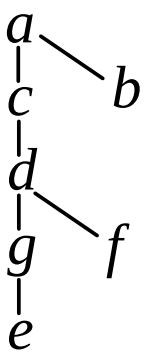
$S = (a, b)$.

DFS

After popping b and a:



DFS Tree



Time Interval

$I(a) = [1, 14]$
$I(c) = [2, 11]$
$I(d) = [3, 10]$
$I(g) = [4, 7]$
$I(e) = [5, 6]$
$I(f) = [8, 9]$
$I(b) = [12, 13]$

$S = ()$.

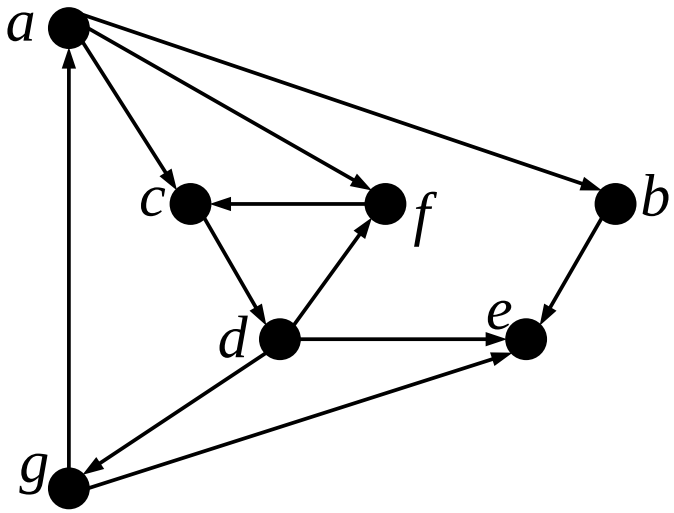
Now, there is no white vertex remaining. The algorithm terminates.
Total running time $O(|V| + |E|)$.

Edge Classification

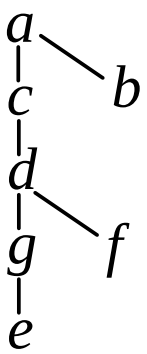
Suppose that we have already built a DFS-forest T.

Let (u, v) be an edge in G (remember that the edge is directed from u to v). It can be classified into

1. Forward edge: u is a proper ancestor of v in a DFS-tree of T.
2. Backward edge: u is a descendant of v in a DFS-tree of T.
3. Cross edge: If neither of the above applies.



DFS Tree



Time Interval

$I(a) = [1, 14]$
$I(c) = [2, 11]$
$I(d) = [3, 10]$
$I(g) = [4, 7]$
$I(e) = [5, 6]$
$I(f) = [8, 9]$
$I(b) = [12, 13]$

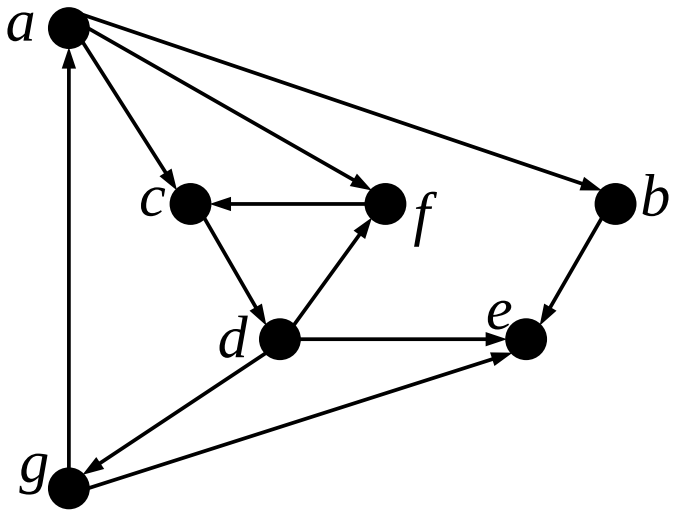
Try to classify each edge into one of the three types mentioned earlier.

Parenthesis Theorem

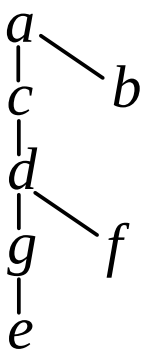
Theorem: All the following are true:

- If u is a proper ancestor of v in a DFS-tree of T, then $I(u)$ contains $I(v)$.
- If u is a proper descendant of v in a DFS-tree of T, then $I(u)$ is contained in $I(v)$.
- Otherwise, $I(u)$ and $I(v)$ are disjoint.

The theorem offers an easy way for us to decide the type of each edge.



DFS Tree



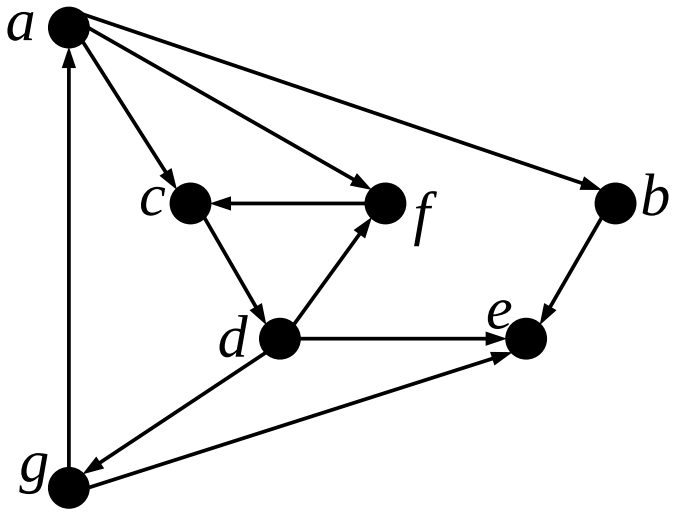
Time Interval

$I(a) = [1, 14]$
$I(c) = [2, 11]$
$I(d) = [3, 10]$
$I(g) = [4, 7]$
$I(e) = [5, 6]$
$I(f) = [8, 9]$
$I(b) = [12, 13]$

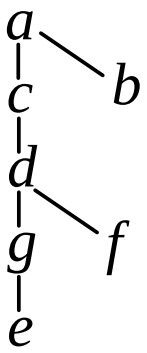
Classify each edge again, but this time using the parenthesis theorem.

Theorem: G contains a cycle if and only if there is a back edge.

Observation



DFS Tree



Time Interval

$I(a) = [1, 14]$
$I(c) = [2, 11]$
$I(d) = [3, 10]$
$I(g) = [4, 7]$
$I(e) = [5, 6]$
$I(f) = [8, 9]$
$I(b) = [12, 13]$

By the parenthesis theorem, we know that the edge (f, c) is a backward edge, because $I(f)$ is contained in $I(c)$. This indicates the existence of a cycle.

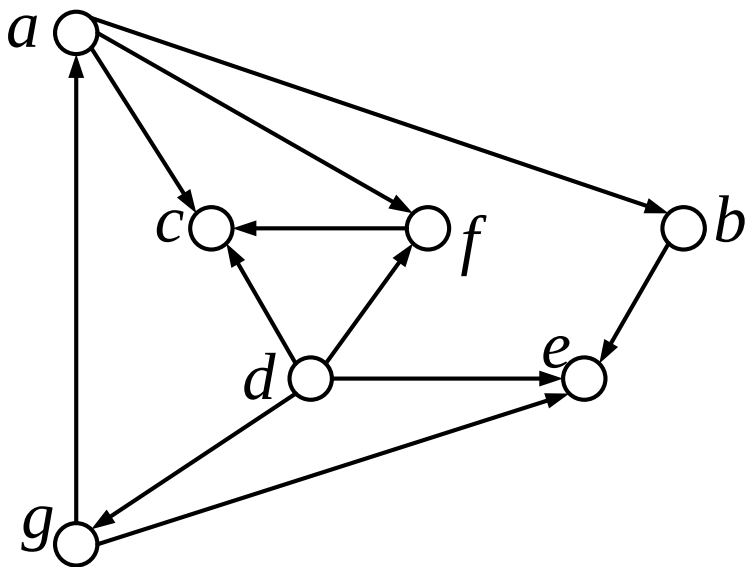
It is worth mentioning that (g, a) is another backward edge.

Topological Sort

- $G = (V, E)$ is a directed acyclic graph (DAG), a topological order of G is an ordering of the vertices in V such that, for any edge (u, v), it must hold that u precedes v in the ordering.
- A directed cyclic graph has no topological orders.
- Every DAG has a topological order.

Problem (Topological Sorting): Given a DAG $G = (V, E)$, find a topological order.

The following is a DAG:



One topological order is d, g, a, f, b, e, c.

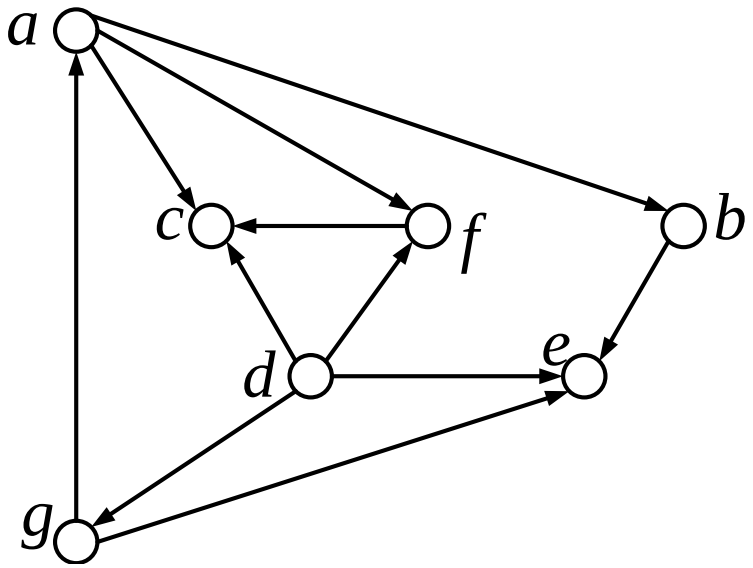
So is d, g, a, b, e, f, c.

Topological sorting can also be solved by DFS.

Run DFS on G, and record the order L by which the vertices turn black (i.e., popped out of the stack).

Theorem: The **reverse** of L is guaranteed to be a topological order of G.

Example



Suppose that we run DFS on the above DAG starting from a, then restarting from g (do this yourself). The following is one possible order by which the vertices turn black:

- c, e, b, f, a, g, d.

Therefore, we output d, g, a, f, b, e, c as a topological order.