Exhaustive Graph traversals. Topological Sorting with DFS

Lecture 03.04

By Marina Barsky

Recap: Depth-First Search (Recursive)

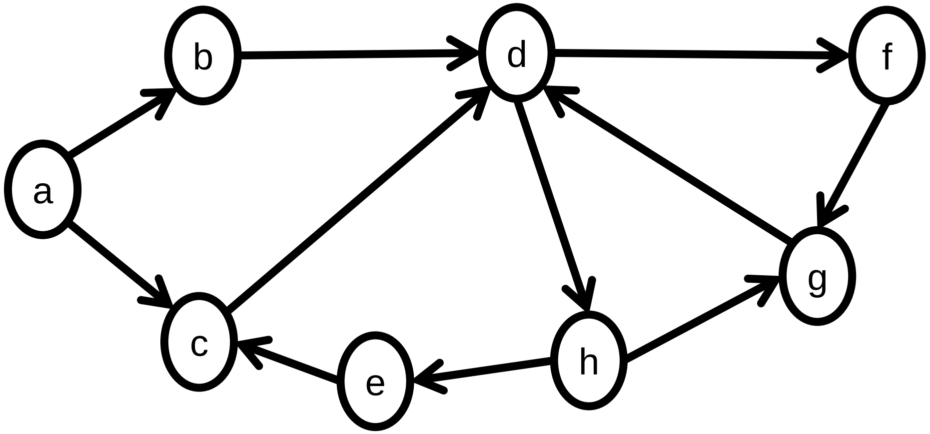
Recursive implementation implicitly replaces the `todo` stack with the call stack.

```
Algorithm DFS(G, current)

    current.state:= "discovered"
    for each u in neighbors(current)
        if u.state = "undiscovered" then
            DFS(G, u)
    current.state:="processed"

for each u in vertices of G
    u.state:= "undiscovered"
DFS(G, start)  // start is a vertex in G
```

This is an exhaustive algorithm, because it visits every node and every edge in graph G

It runs in time O(n + m) if implemented using adjacency list

DFS in Directed Graph



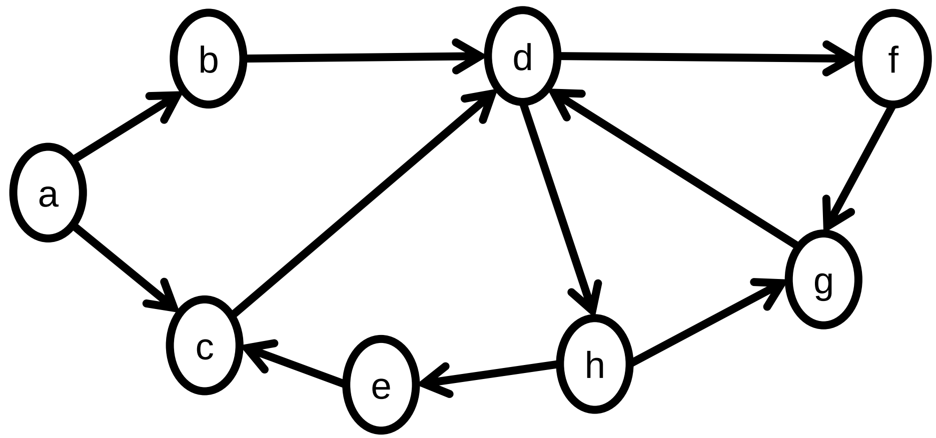
The algorithm for Directed Graphs is exactly the same

By the end we discover all the nodes in digraph G that are reachable from the source node *start*

```
Algorithm DFS(digraph G, current)

    current.state:= "discovered"
    for each u in out_arcs(current)
        if u.state = "undiscovered" then
           DFS(G, u)
    current.state:="processed"

for each u in vertices of G
    u.state:= "undiscovered"
DFS(digraph G, start)  // start is a vertex in G
```

The time of discovery and finishing time

- Unlike in BFS (with its removal from the front of a queue) the order in which we discover a new unprocessed vertex differs from the order in which we mark vertices as processed



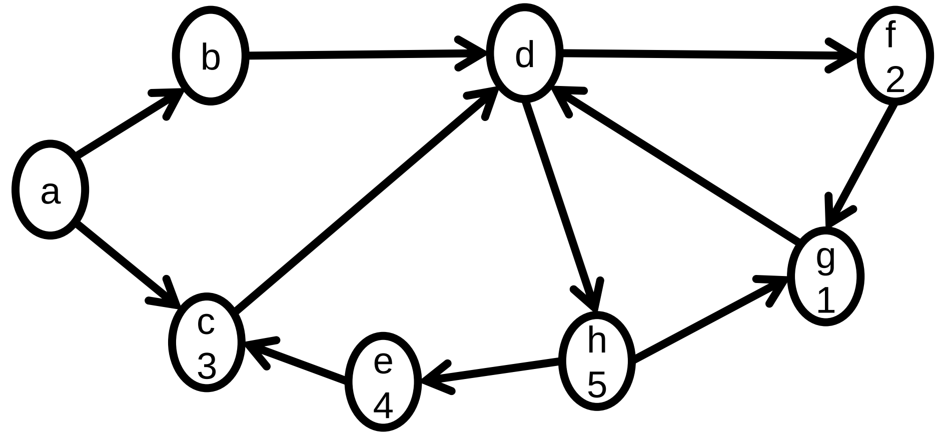
- Imagine that we have a clock, and before we begin the clock is set to 1.
- The moment that we mark some node as processed, we also mark it with the current value of the clock, and we increment the clock value by 1

Definition
Let ***finishing time f(v)*** of node v be the value of *clock* variable at the moment that v was marked as processed by the DFS algorithm

In essence f(v) is the count of all the vertices processed before v

Example of computing finishing time

- Let's start DFS from an arbitrary vertex, say, vertex d
- We traverse the tree and collect all nodes reachable from d



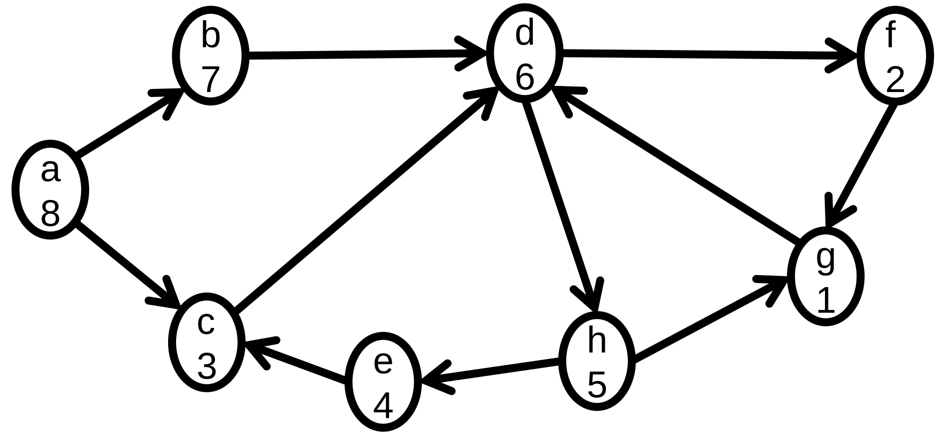
d	h	e	c						

g1	f2	c3	e4	h5	d6				

We finished with all the nodes reachable from d

Example of computing finishing time

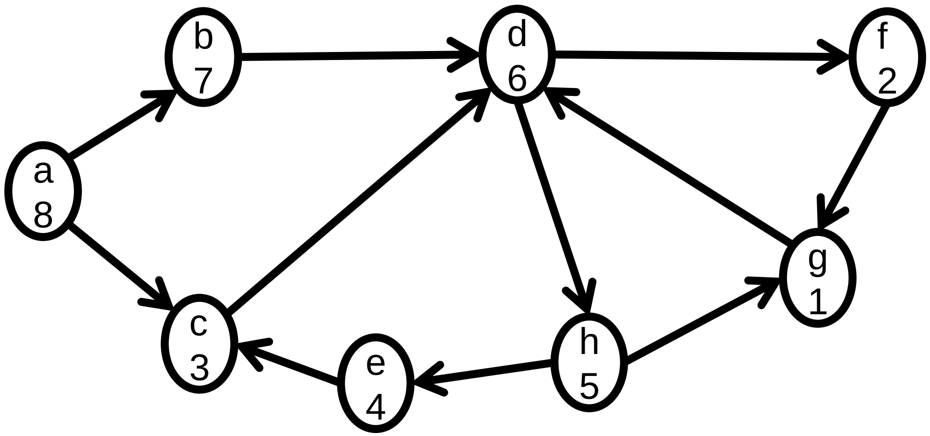
- We start another DFS from vertex a, say
- We traverse the tree and add two new nodes to the stack



a	b								

g1	f2	c3	e4	h5	d6	b7	a8		

Example of computing finishing time



g1	f2	c3	e4	h5	d6	b7	a8		

Finishing time for each node in G

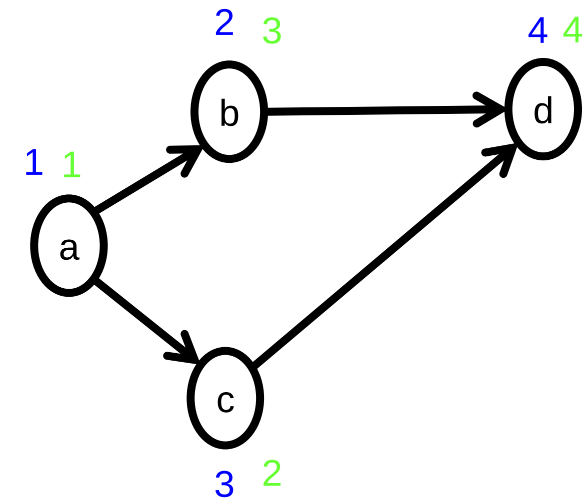
We obtained some sort of an order on graph vertices, in essence saying that if f(v) > f(u) then u is processed first in the DFS

That means that there is a directed path from v to u

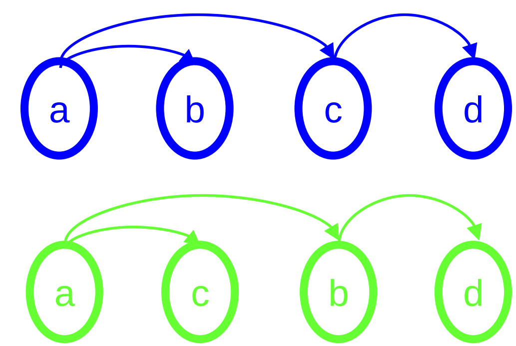
Topological Order

- Topological sort is an ordering of vertices in a Directed Acyclic Graph [DAG] in which *each node comes before all nodes to which it has outgoing edges.*

- Each node is assigned a label t(v):
  - t(v) is a unique order of node v from 1 to n
  - If there is a directed edge u->v, then t(u)<t(v)



Consider the course prerequisite structure at universities. A directed edge (v,w) indicates that course *v* must be completed before course *w*. Topological ordering in this case is the sequence which does not violate the prerequisite requirement.



- Topological sort is not possible if the graph has a cycle, since for two vertices u and v on the cycle, it is not possible that t(u)<t(v) and at the same time t(v)<t(u).

Topological Order is not unique

Computing Topological Order

- The topological order is exactly opposite to the finishing time
- The finishing time of the vertex indicates that all nodes reachable from it have been processed, that means it is not a prerequisite for any one of them
- Thus the node without prerequisites (with the smallest t(v)) finishes last (has the largest f(v))

- This gives an algorithm for computing topological order using DFS

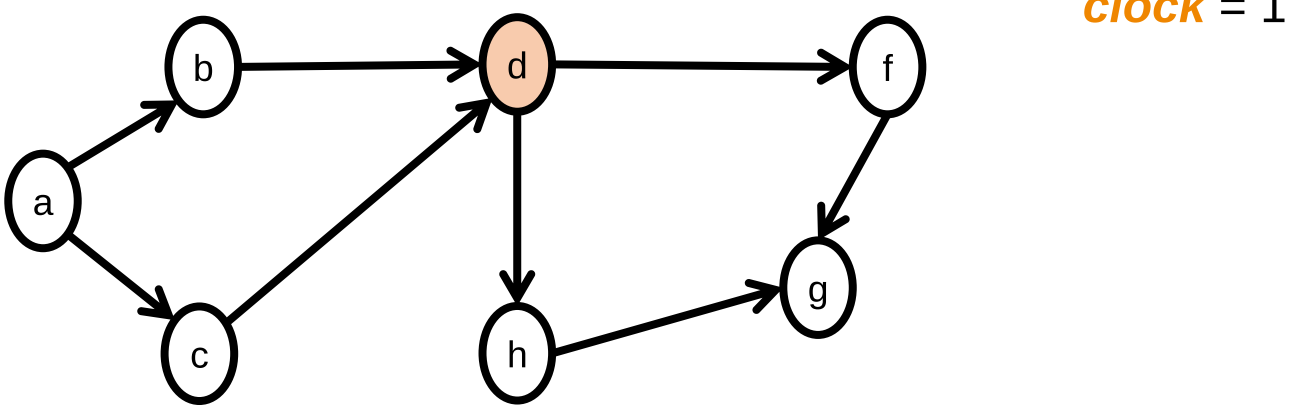
Topological Sort via DFS

```
global sorted_nodes:= empty linked list
global clock: = 1
```

```
Algorithm DFS(DAG G, current)

    current.state:= “discovered”
    for each u in out_arcs(current)
        if u.state = “undiscovered” then
            DFS(G, u)
    current.state:=“processed”
    current.finishing_time: = clock
    clock: = clock + 1
    sorted_nodes.add_in_front(current)
```

```
Algorithm DFS_loop(DAG G)

    mark all nodes of G as “undiscovered”
    for each u in vertices of G
        if u.state = “undiscovered”
            DFS(DAG G, u)
```

Example

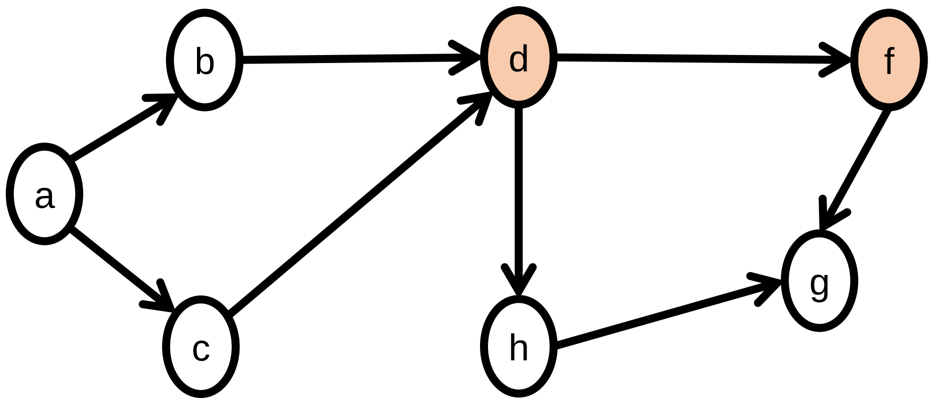


current_vertex		Recursion stack:	d					

Finishing time							
Sorted list							

Example

clock = 1



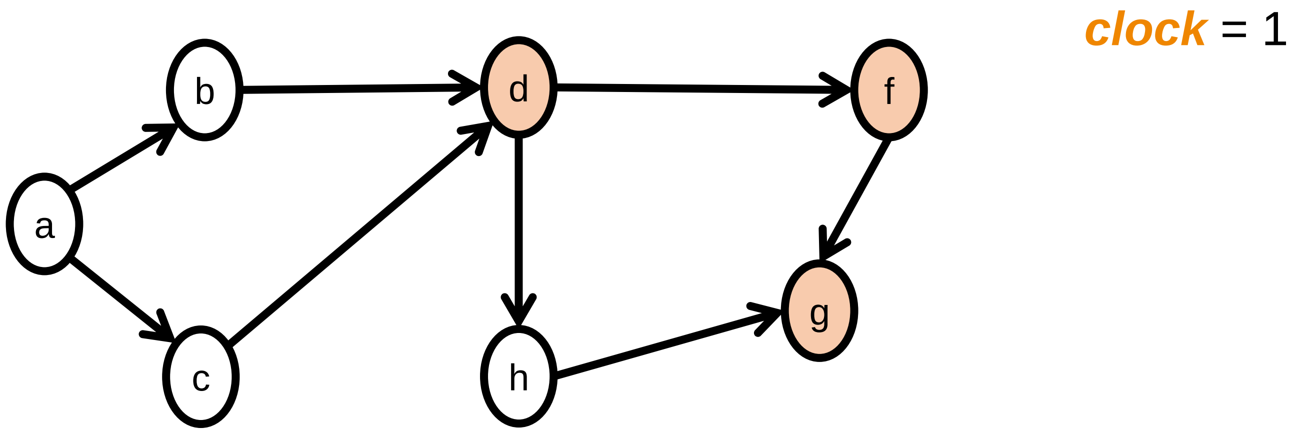
current_vertex

Recursion stack:

d	f				

Finishing time							
Sorted list							

Example



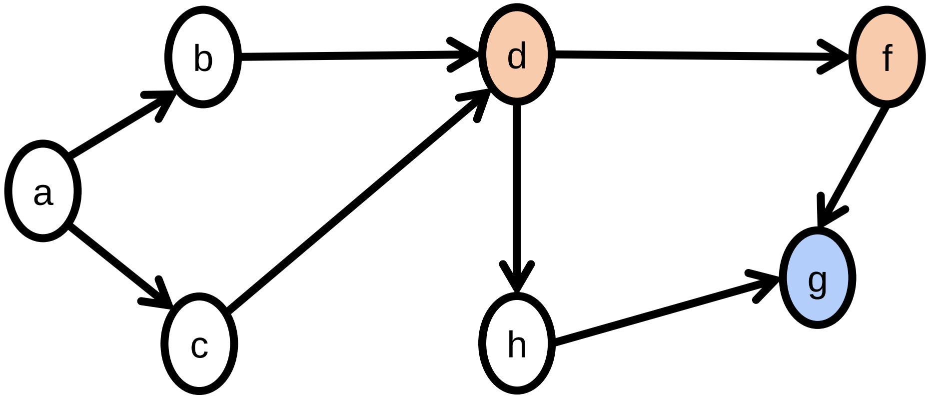
current_vertex	

Recursion stack:

d	f	g			

Finishing time							
Sorted list							

Example



clock = 1

current_vertex: g

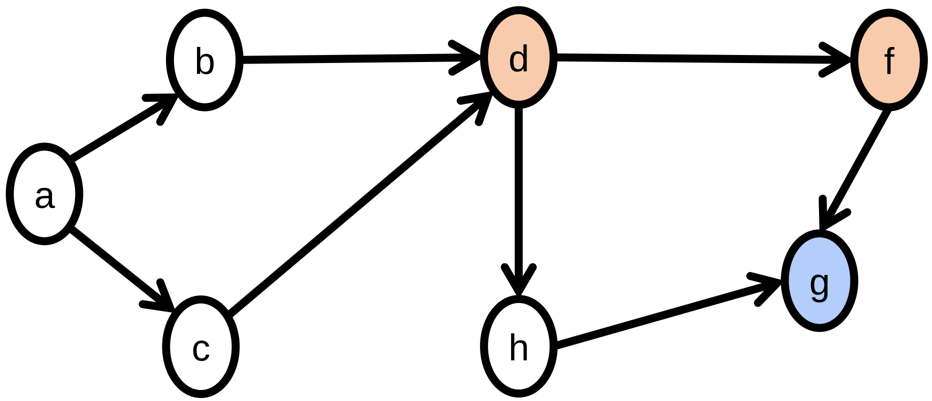
Recursion stack: d, f

Finishing time

Sorted list

Example

clock = 2



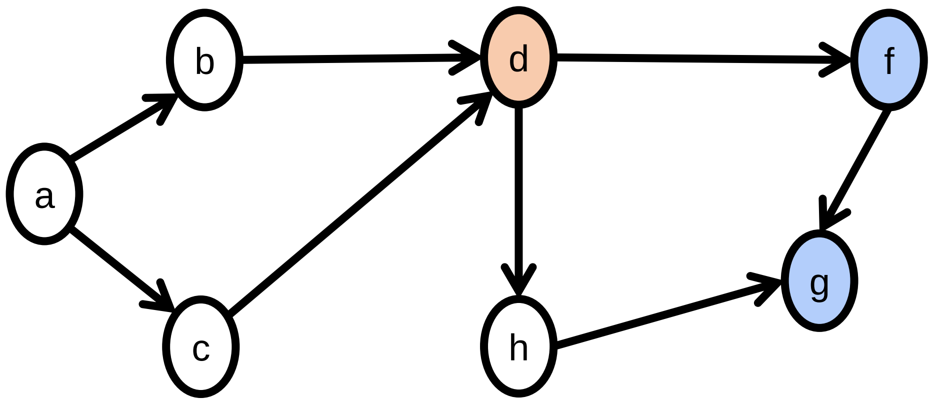
current_vertex: g

Recursion stack:

d	f				

Finishing time	1						
Sorted list	g						

Example



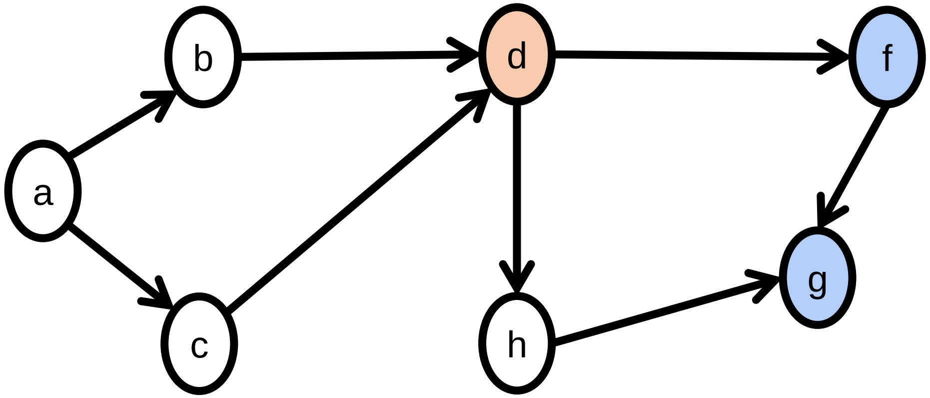
clock = 2

current_vertex: f

Recursion stack: d

Finishing time	1						
Sorted list	g						

Example



clock = 3

current_vertex: f

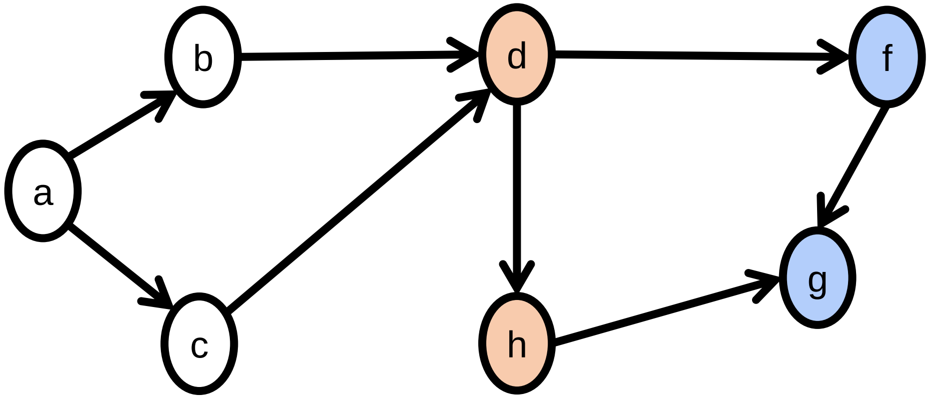
Recursion stack:

d					

Finishing time	2	1					
Sorted list	f	g					

Example

clock = 3



current_vertex

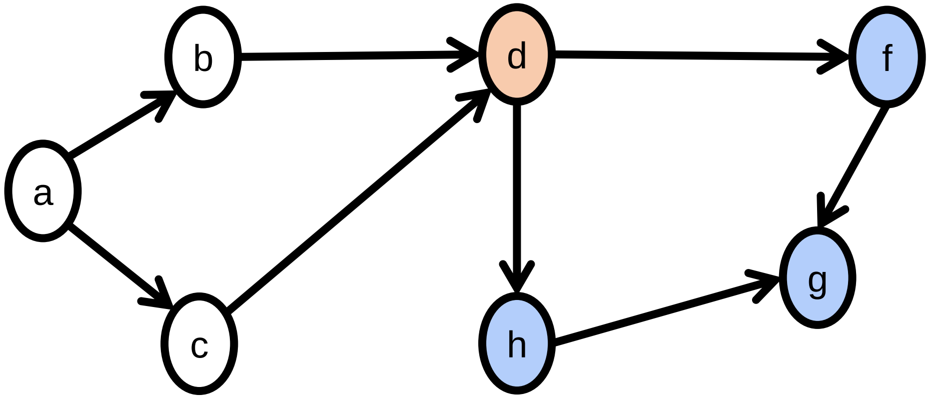
Recursion stack:

d	h				

Finishing time	2	1					
Sorted list	f	g					

Example

clock = 3



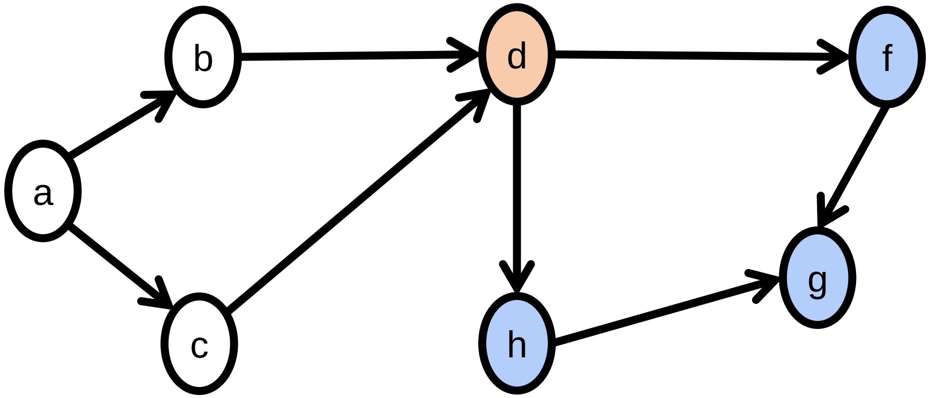
current_vertex	h

Recursion stack:

d					

Finishing time	2	1					
Sorted list	f	g					

Example



clock = 4

current_vertex: h

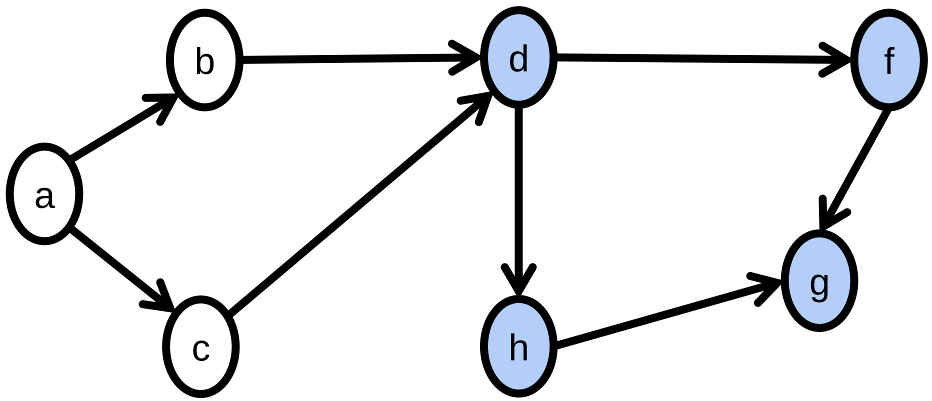
Recursion stack:

d					

Finishing time	3	2	1				
Sorted list	h	f	g				

Example

clock = 4



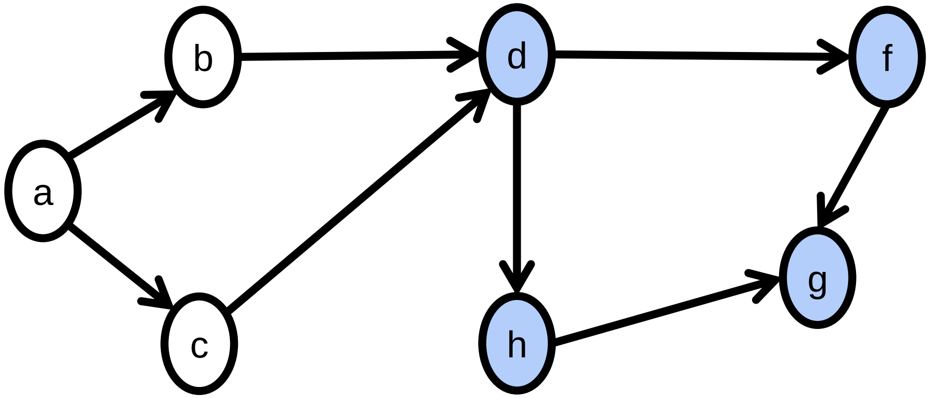
current_vertex: d

Recursion stack:

Finishing time	3	2	1				
Sorted list	h	f	g				

Example

clock = 5



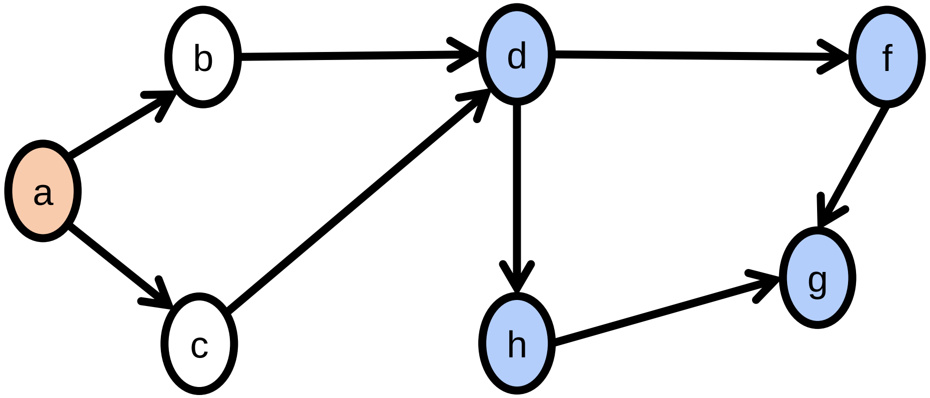
current_vertex: d

Recursion stack:

Finishing time	4	3	2	1			
Sorted list	d	h	f	g			

Example

clock = 5

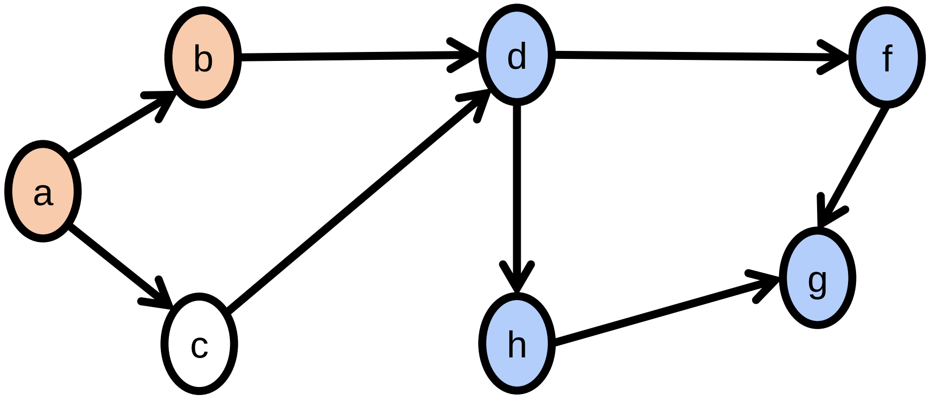


current_vertex | Recursion stack: a

Finishing time	4	3	2	1			
Sorted list	d	h	f	g			

Example

clock = 5



current_vertex

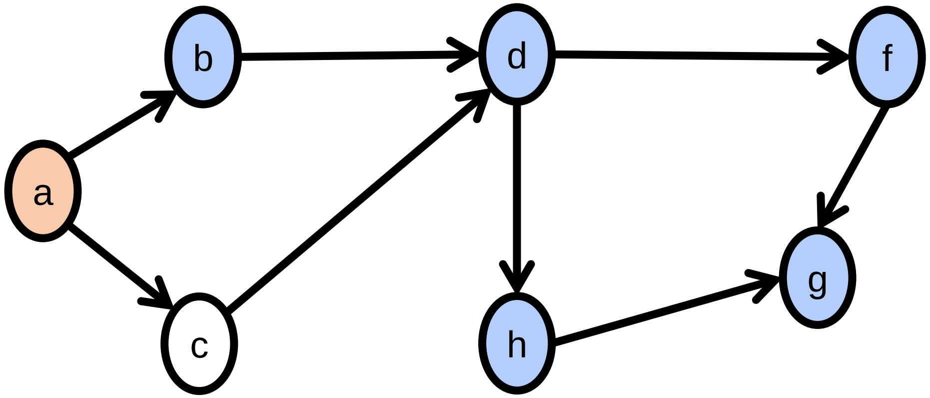
Recursion stack:

a	b				

Finishing time	4	3	2	1			
Sorted list	d	h	f	g			

Example

clock = 5

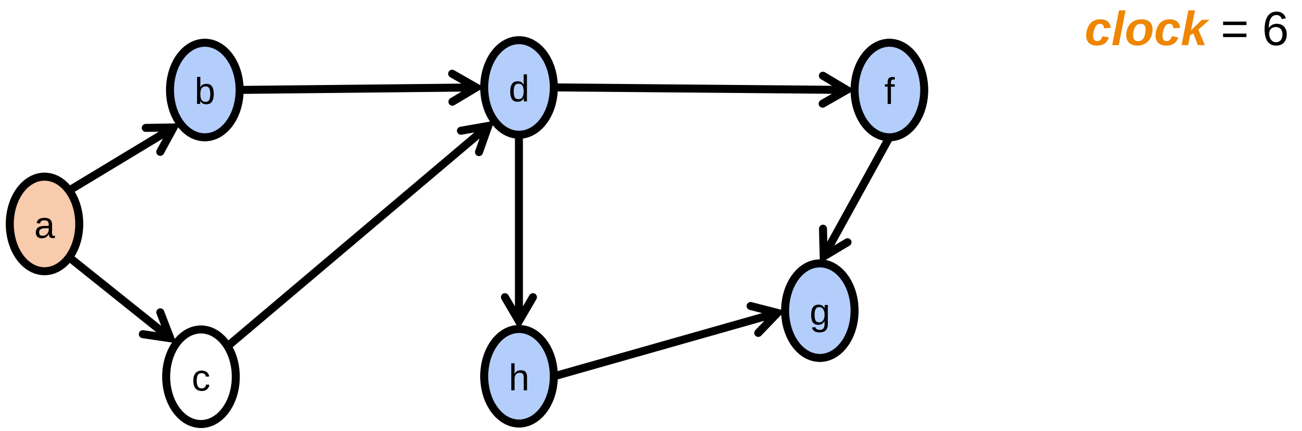


current_vertex: b

Recursion stack: a

Finishing time	4	3	2	1			
Sorted list	d	h	f	g			

Example



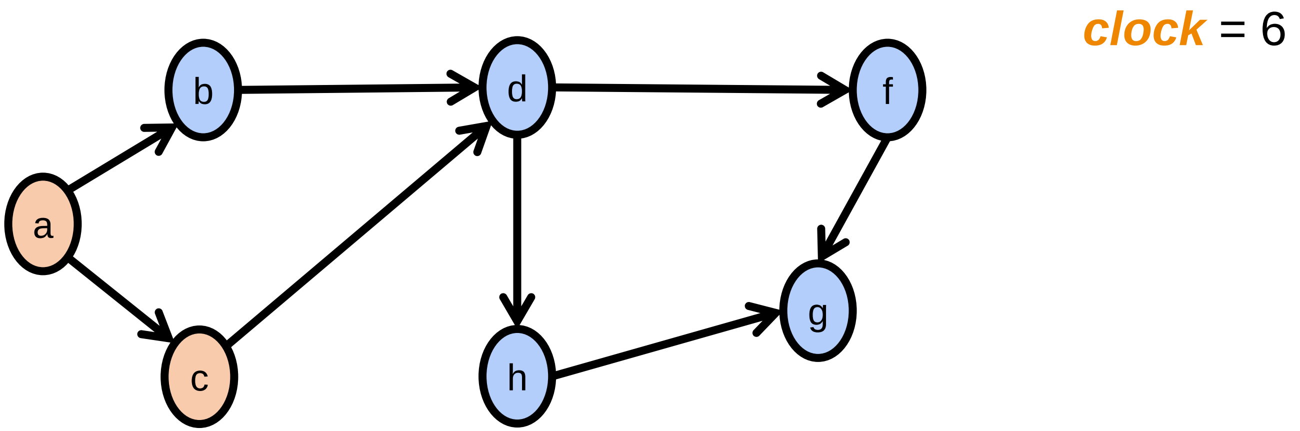
current_vertex | |

Recursion stack:

a					

Finishing time	5	4	3	2	1		
Sorted list	b	d	h	f	g		

Example



current_vertex

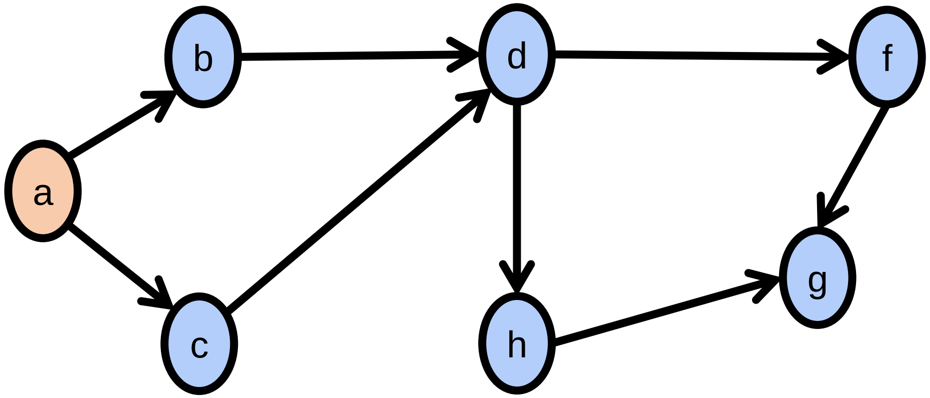
Recursion stack:

a	c				

Finishing time	5	4	3	2	1		
Sorted list	b	d	h	f	g		

Example

clock = 6



current_vertex: c

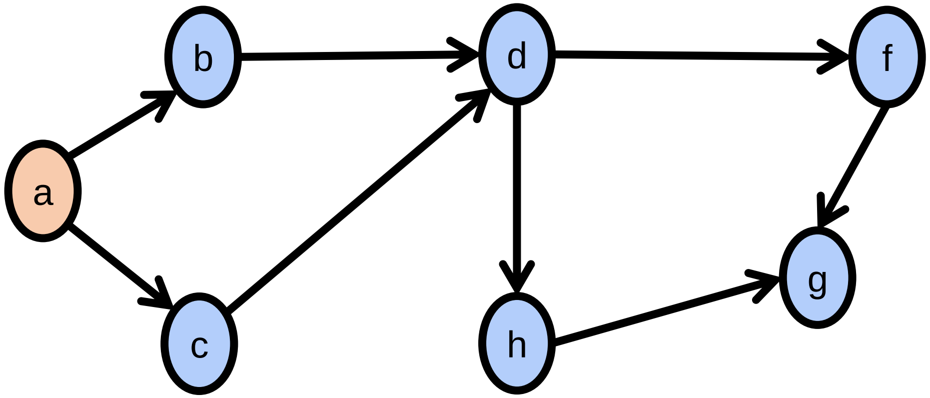
Recursion stack:

a					

Finishing time	5	4	3	2	1		
Sorted list	b	d	h	f	g		

Example

clock = 7



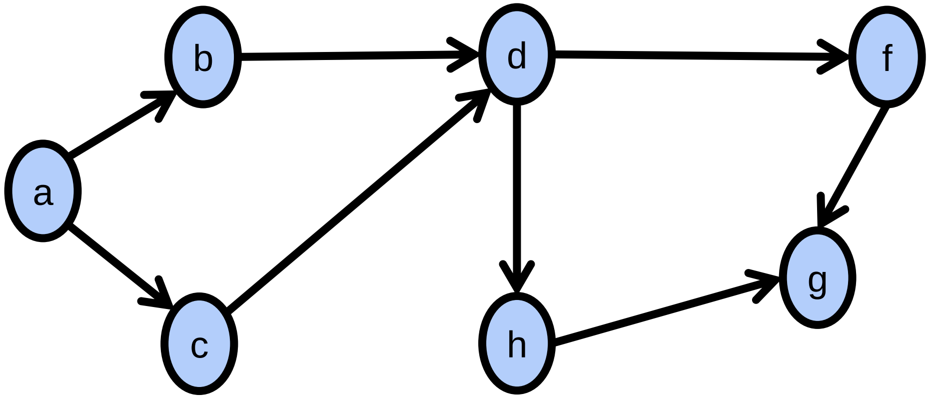
current_vertex

Recursion stack:					
a					

Finishing time	6	5	4	3	2	1	
Sorted list	c	b	d	h	f	g	

Example

clock = 7

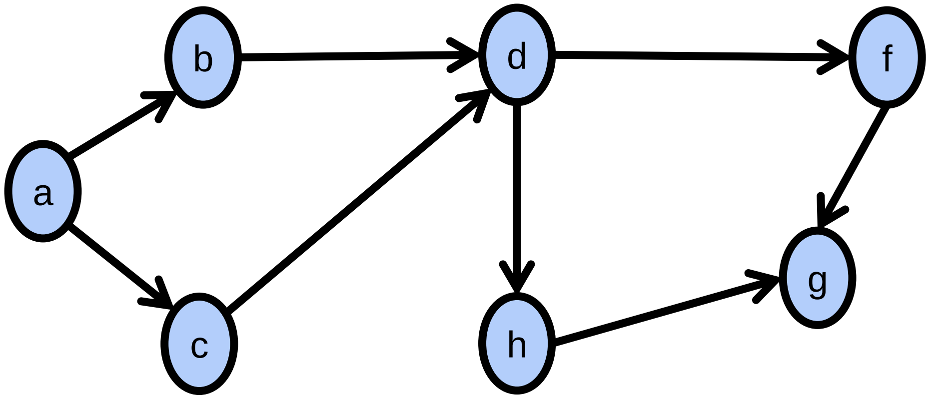


current_vertex: a

Recursion stack:

Finishing time	6	5	4	3	2	1	
Sorted list	c	b	d	h	f	g	

Example



current_vertex a

Recursion stack:

Sorted list	a	c	b	d	h	f	g
Finishing time	7	6	5	4	3	2	1

Question to think about

- How the same DFS loop can be used to determine if the graph is cycle-free?