Graphs

November 26, 2018

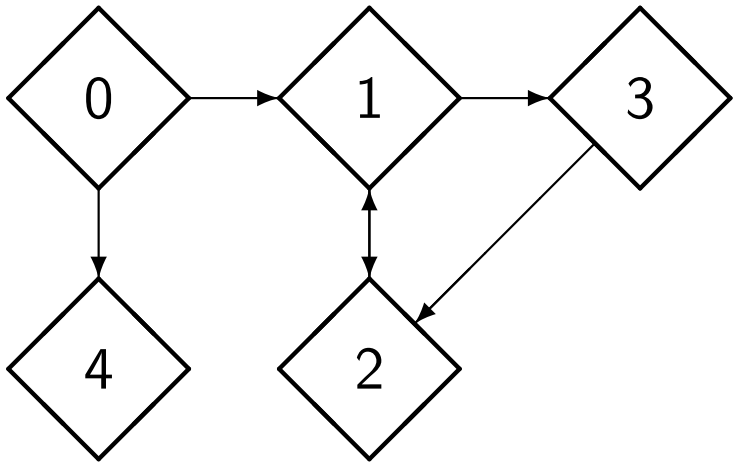
- Non-Cyclic Graphs
- Topological Sorting
- Implementation

Paths through Graphs

1: Non-Cyclic Graphs
1.1: Cycles in Graphs
1.1.1: Paths through Graphs
Definition: A *path* is a list of edges, each of which follows the other, in a given graph.

Example

Example: In the graph:



the following...

...are paths in this graph:

$\mathbf{0 \longrightarrow 4}$

$\mathbf{0 \longrightarrow 1 \longrightarrow 2 \longrightarrow 1 \longrightarrow 2}$

$\mathbf{2 \longrightarrow 1 \longrightarrow 3 \longrightarrow 2}$

...are not paths in this graph:

$\mathbf{0 \longrightarrow 3 \longrightarrow 2}$

$\mathbf{0 \longrightarrow 1 \longrightarrow 2 \longrightarrow 4}$

$\mathbf{0 \longrightarrow 5}$

Cycles in Graphs

1.1.2: Cycles in Graphs

Definition: A *cycle* is a (non-empty) path that starts and ends at the same node or vertex.

Example: Using the graph in the previous example, the following are all cycles:

- $\mathbf{2 \longrightarrow 1 \longrightarrow 2}$
- $\mathbf{1 \longrightarrow 3 \longrightarrow 2 \longrightarrow 1}$
- $\mathbf{2 \longrightarrow 1 \longrightarrow 3 \longrightarrow 2 \longrightarrow 1 \longrightarrow 3 \longrightarrow 2}$

Non-cyclic Graphs

1.2: Ayclic Graphs

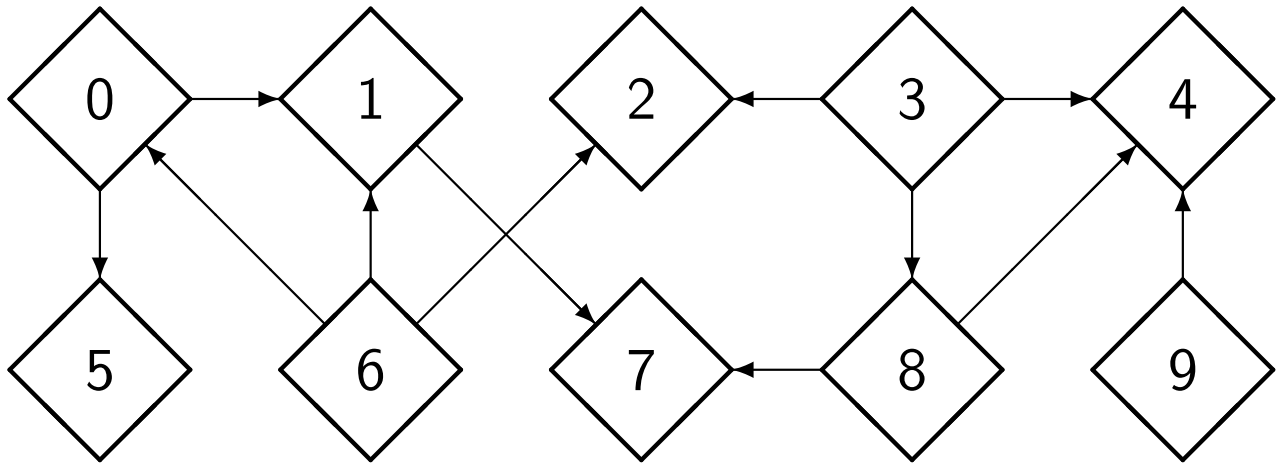
Definition: A *non-cyclic* graph is simply a graph that does not contain any cycles.

Trees are an obvious source of examples of non-cyclic graphs.

Topological Sorting

2: Topological Sorting

Consider the following non-cyclic graph:



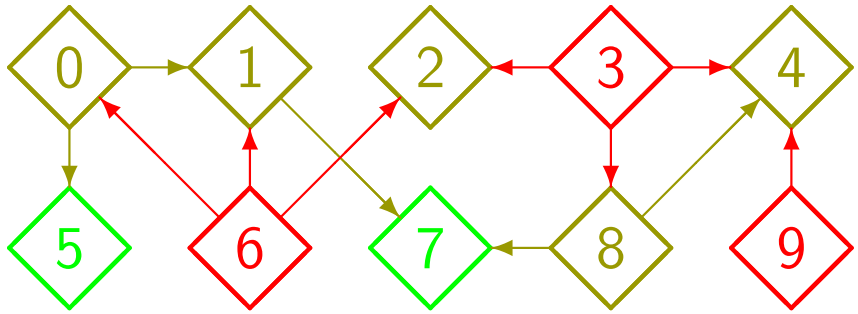
Is it possible to list the nodes of this graph in such a way that:

if n *and* m *are nodes in the graph*
and *there is a path from* n *to* m
then n *occurs* before m *in our listing?*

Definition: Any such listing of the nodes of a non-cyclic graph is called a *topological sorting*.

Example

Example:



- node 0 occurs before nodes 1 and 5 (and before node 7)
- node 1 occurs before node 7
- node 3 occurs before nodes 2, 4 and 8 (and before node 7)
- node 6 occurs before nodes 0, 1 and 2 (and before nodes 5 and 7)
- node 8 occurs before nodes 4 and 7
- node 9 occurs before node 4

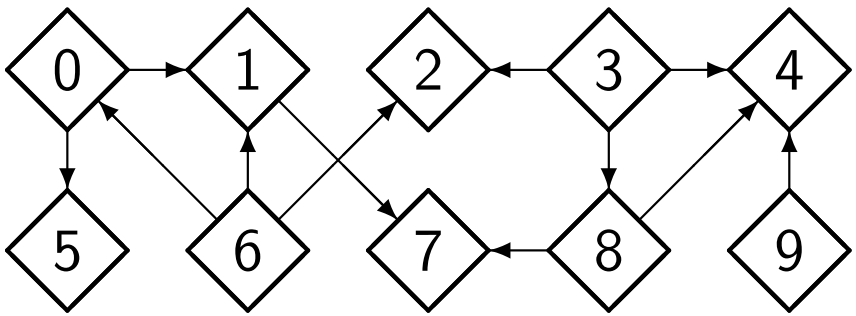
Depth-first traversal

2.1: Attempt 1

Will depth-first traversal yield a topological sort for non-cyclic graphs?

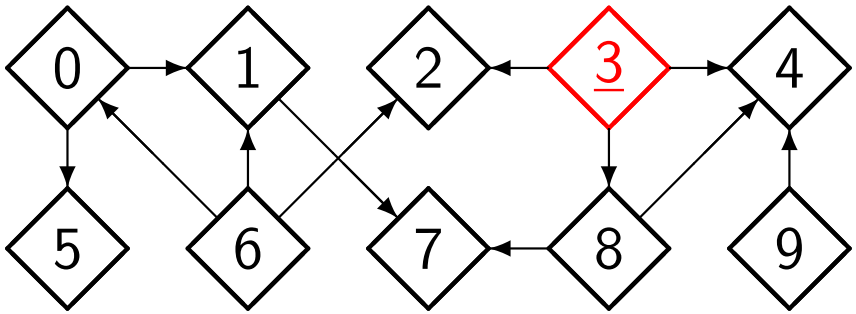
Depth-first traversal

Example:



0	$\{1, 5\}$
1	$\{7\}$
2	$\emptyset$
3	$\{2, 4, 8\}$
4	$\emptyset$
5	$\emptyset$
6	$\{0, 1, 2\}$
7	$\emptyset$
8	$\{4, 7\}$
9	$\{4\}$

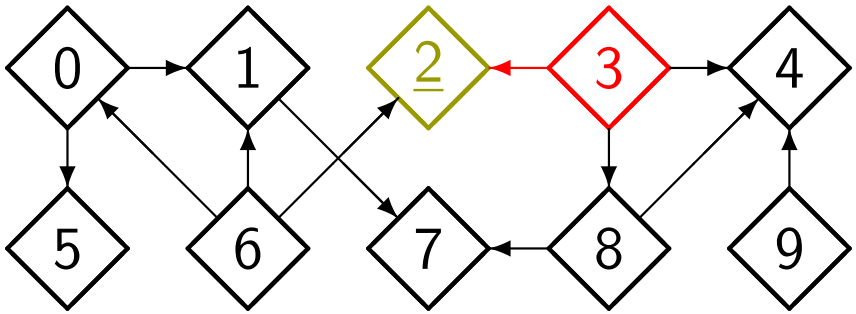
Depth-first traversal



0	X	$\{1, 5\}$
1	X	$\{7\}$
2	X	$\emptyset$
3	✓	$\{2, 4, 8\}$
4	X	$\emptyset$
5	X	$\emptyset$
6	X	$\{0, 1, 2\}$
7	X	$\emptyset$
8	X	$\{4, 7\}$
9	X	$\{4\}$

[3]

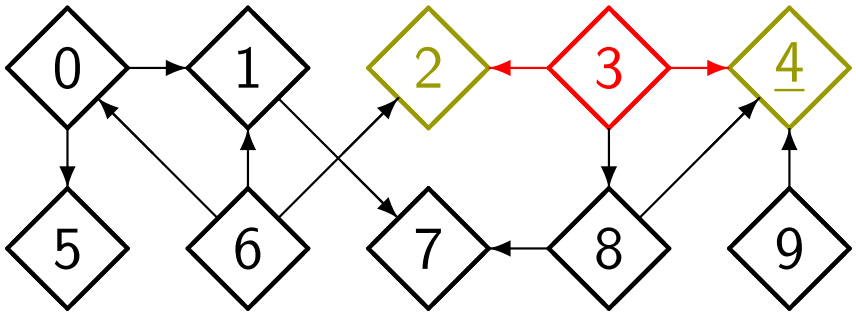
Depth-first traversal



0	X	$\{1, 5\}$
1	X	$\{7\}$
2	✓	$\emptyset$
3	✓	$\{2, 4, 8\}$
4	X	$\emptyset$
5	X	$\emptyset$
6	X	$\{0, 1, 2\}$
7	X	$\emptyset$
8	X	$\{4, 7\}$
9	X	$\{4\}$

[3,2]

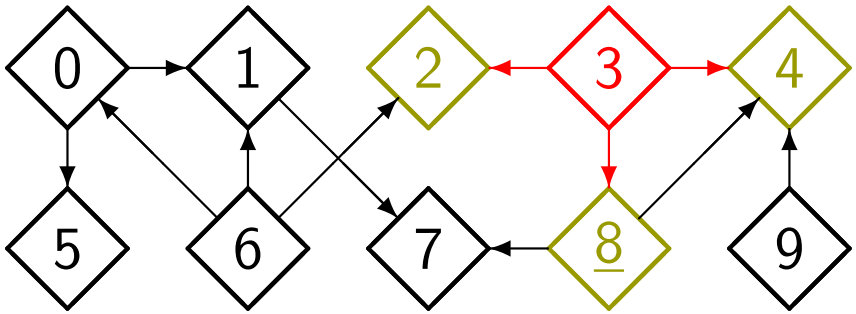
Depth-first traversal



0	X	$\{1, 5\}$
1	X	$\{7\}$
2	✓	$\emptyset$
3	✓	$\{2, 4, 8\}$
4	✓	$\emptyset$
5	X	$\emptyset$
6	X	$\{0, 1, 2\}$
7	X	$\emptyset$
8	X	$\{4, 7\}$
9	X	$\{4\}$

[3,2,4]

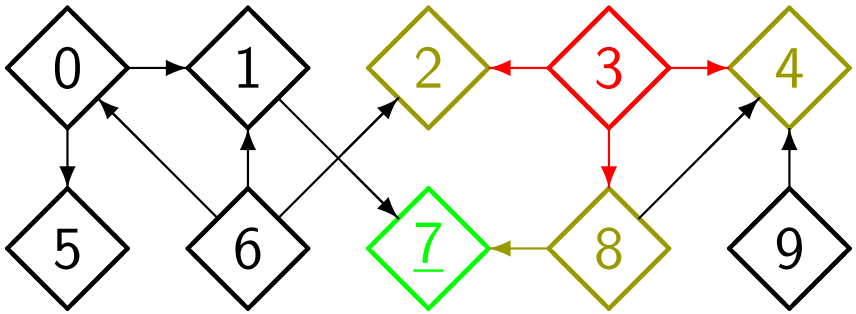
Depth-first traversal



0	X	$\{1, 5\}$
1	X	$\{7\}$
2	✓	$\emptyset$
3	✓	$\{2, 4, 8\}$
4	✓	$\emptyset$
5	X	$\emptyset$
6	X	$\{0, 1, 2\}$
7	X	$\emptyset$
8	✓	$\{4, 7\}$
9	X	$\{4\}$

[3,2,4,8]

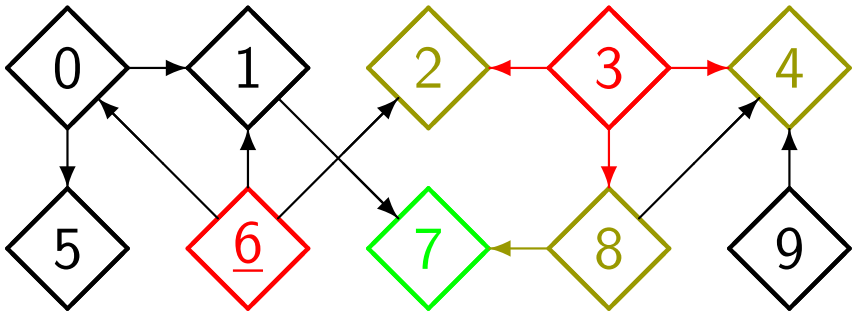
Depth-first traversal



0	X	$\{1, 5\}$
1	X	$\{7\}$
2	✓	$\emptyset$
3	✓	$\{2, 4, 8\}$
4	✓	$\emptyset$
5	X	$\emptyset$
6	X	$\{0, 1, 2\}$
7	✓	$\emptyset$
8	✓	$\{8, 7\}$
9	X	$\{4\}$

[3,2,4,8,7]

Depth-first traversal



0	X	$\{1, 5\}$
1	X	$\{7\}$
2	✓	$\emptyset$
3	✓	$\{2, 4, 8\}$
4	✓	$\emptyset$
5	X	$\emptyset$
6	✓	$\{0, 1, 2\}$
7	✓	$\emptyset$
8	✓	$\{4, 7\}$
9	X	$\{4\}$

[3,2,4,8,7,6]

Depth-first traversal

Depth-first traversal gives a traversal that starts with

$$[3, 2, 4, 8, 7, 6, \ldots]$$

but

- node 2 is before node 6 in this list
- $6 \to 2$ is an edge!

Reversing the list to give

$$[\ldots, 6, 7, 8, 4, 2, 3]$$

also fails to give a topological sort since **4** is before **3** in the list, but there is an edge $3 \to 4$.

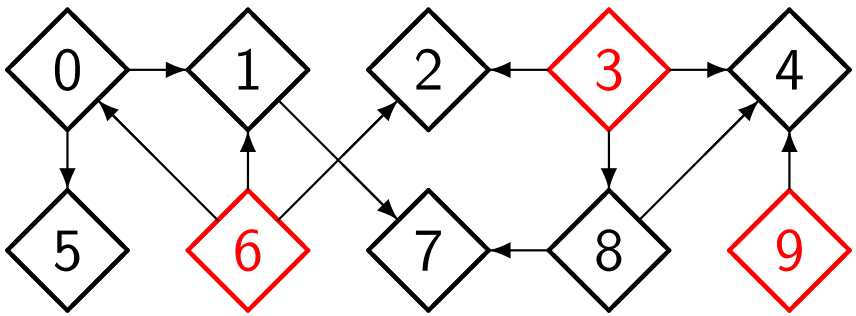
Breadth-first traversal

2.2: Attempt 2

Will breadth-first traversal yield a topological sort for non-cyclic graphs?

No, for similar reasons to the depth-first traversal.

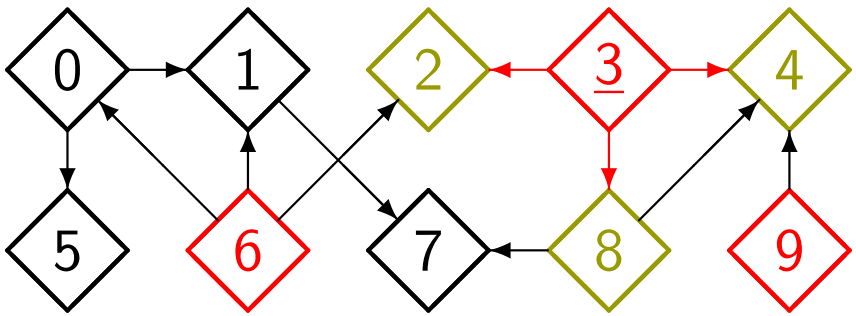
Breadth-first traversal



visited: []
to do: [3,6,9]

0	✓	**{1, 5}**
1	X	**{7}**
2	X	**∅**
3	X	**{2, 4, 8}**
4	X	**∅**
5	X	**∅**
6	X	**{0, 1, 2}**
7	X	**∅**
8	X	**{4, 7}**
9	X	**{4}**

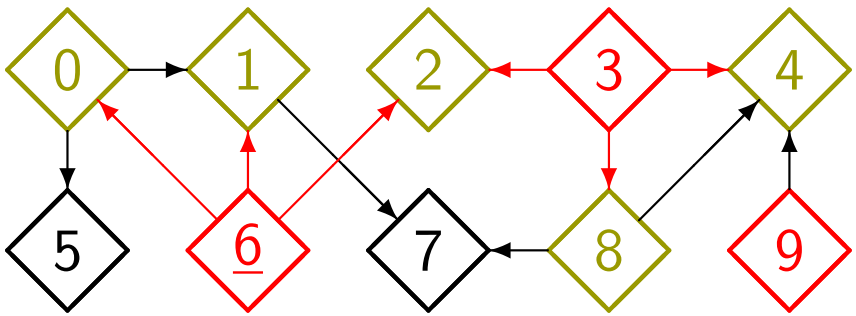
Breadth-first traversal



visited: [3]
to do: [6,9,2,4,8]

0	X	$\{1, 5\}$
1	X	$\{7\}$
2	X	$\emptyset$
3	✓	$\{2, 4, 8\}$
4	X	$\emptyset$
5	X	$\emptyset$
6	X	$\{0, 1, 2\}$
7	X	$\emptyset$
8	X	$\{4, 7\}$
9	X	$\{4\}$

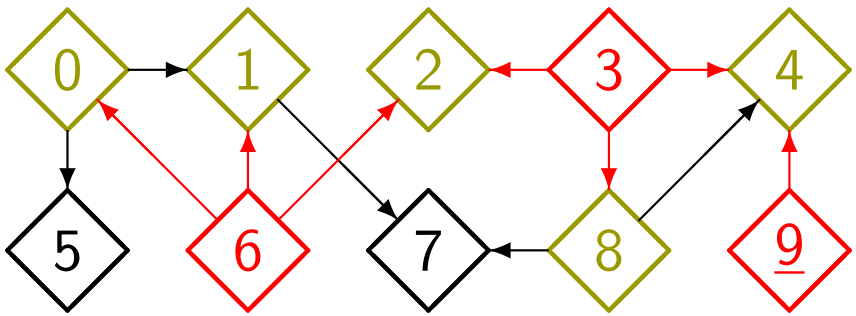
Breadth-first traversal



visited: [3,6]
to do: [9,2,4,8,0,1,2]

0	X	{1, 5}
1	X	{7}
2	X	∅
3	✓	{2, 4, 8}
4	X	∅
5	X	∅
6	✓	{0, 1, 2}
7	X	∅
8	X	{4, 7}
9	X	{4}

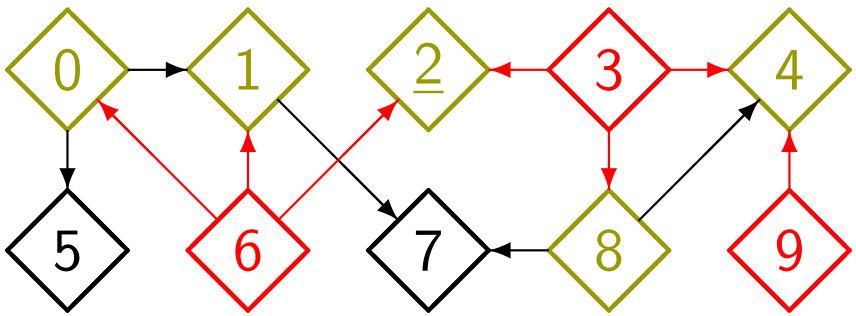
Breadth-first traversal



visited: [3,6,9]
to do: [2,4,8,0,1,2,4]

0	X	{1, 5}
1	X	{7}
2	X	∅
3	✓	{2, 4, 8}
4	X	∅
5	X	∅
6	✓	{0, 1, 2}
7	X	∅
8	X	{4, 7}
9	✓	{4}

Breadth-first traversal

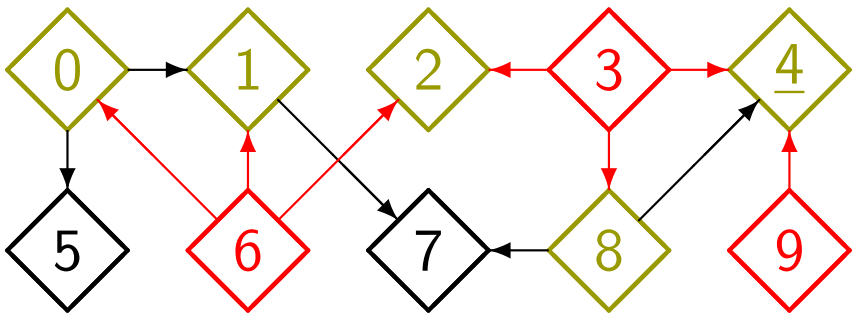


visited: [3,6,9,2]

to do: [4,8,0,1,2,4]

0	X	{1, 5}
1	X	{7}
2	X	∅
3	✓	{2, 4, 8}
4	X	∅
5	X	∅
6	✓	{0, 1, 2}
7	X	∅
8	X	{4, 7}
9	✓	{4}

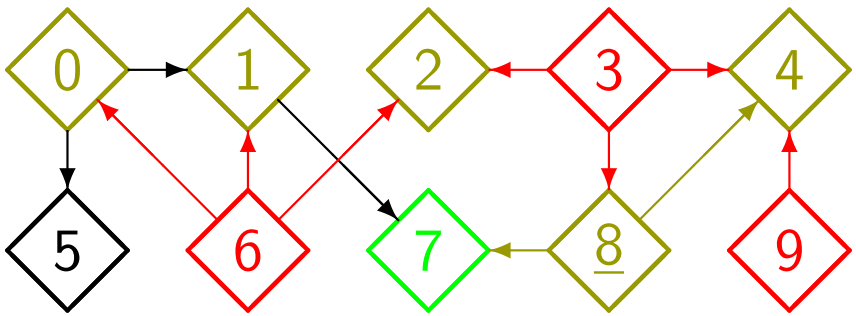
Breadth-first traversal



visited: [3,6,9,2,4]
to do: [8,0,1,2,4]

0	X	{1, 5}
1	X	{7}
2	X	∅
3	✓	{2, 4, 8}
4	X	∅
5	X	∅
6	✓	{0, 1, 2}
7	X	∅
8	X	{4, 7}
9	✓	{4}

Breadth-first traversal



visited: [3,6,9,2,4,8]

to do: [0,1,2,4,7]

0	X	{1, 5}
1	X	{7}
2	X	∅
3	✓	{2, 4, 8}
4	X	∅
5	X	∅
6	✓	{0, 1, 2}
7	X	∅
8	X	{4, 7}
9	✓	{4}

Breadth-first traversal

Breadth-first traversal gives a traversal that starts:

$$[3, 6, 9, 2, 4, 8, \ldots]$$

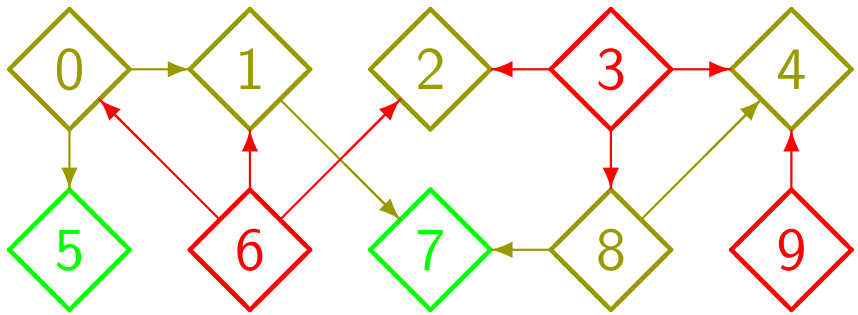
but

- node 4 is before node 8
- $8 \rightarrow 4$ is an edge

The reversed list $[\ldots, 8, 4, 2, 9, 6, 3]$ is even worse, since all the root level nodes come at the end of the list.

Topological sorts

Do topological sorts exist?



- node 0 occurs before nodes 1 and 5
- node 1 occurs before node 7
- node 3 occurs before nodes 2, 4 and 8
- node 6 occurs before nodes 0, 1 and 2
- node 8 occurs before nodes 4 and 7
- node 9 occurs before node 4

Topological sorts

- node 0 occurs before nodes 1 and 5
- node 1 occurs before node 7
- node 3 occurs before nodes 2, 4 and 8
- node 6 occurs before nodes 0, 1 and 2
- node 8 occurs before nodes 4 and 7
- node 9 occurs before node 4

A topological sort:

$$\mathbf{[9, 6, 3, 8, 4, 2, 0, 5, 1, 7]}$$

Existence of Topological Sorts

2.3: Existence of Topological Sorts

- Find a node with no (unsorted) successors.
- This node can safely be placed in the sort after any remaining unsorted nodes in the graph (and before any previously sorted ones).
- "Remove" this node from the graph
- Repeat the process

This will give a topological sort.

Depth-first topological sort

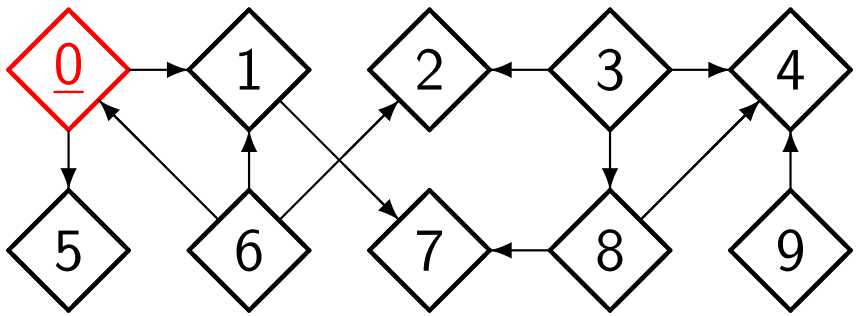
2.4: Depth-first Topological Sort

```
the sort is empty
while (there are nodes left in the graph) {
    find a node with no successors;
    add it to the sort in front of any previously sorted noc
    remove it from the graph;
}
```

Depth-first topological sort

- `find a node with no successors`
  Use a depth-first traversal — when it backtracks the node has no (unvisited) successors.
- `add it in front of any previously sorted nodes`
  Keep an index, starting with the last element of the sort, and decrease it each time a node is added, or use, e.g., a stack, or other list structure that adds elements at the head of the list.
- `remove it from the graph`
  Mark the node as visited.

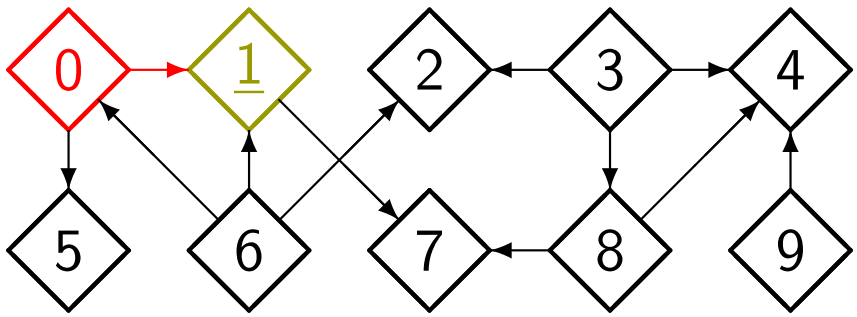
Depth-first topological sort



$$[\ ,\ ,\ ,\ ,\ ,\ ,\ ,\ ,\ ,\ ,\ ]$$

0	X	$\{1, 5\}$
1	X	$\{7\}$
2	X	$\emptyset$
3	X	$\{2, 4, 7, 8\}$
4	X	$\emptyset$
5	X	$\emptyset$
6	X	$\{0, 1, 2\}$
7	X	$\emptyset$
8	X	$\{4, 7\}$
9	X	$\{4\}$

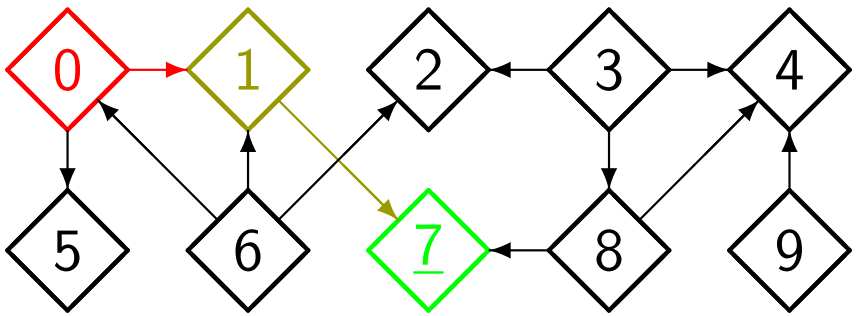
Depth-first topological sort



$$[\ ,\ ,\ ,\ ,\ ,\ ,\ ,\ ,\ ,\ ,\ ]$$
$$\uparrow$$

0	X	$\{1, 5\}$
1	X	$\{7\}$
2	X	$\emptyset$
3	X	$\{2, 4, 7, 8\}$
4	X	$\emptyset$
5	X	$\emptyset$
6	X	$\{0, 1, 2\}$
7	X	$\emptyset$
8	X	$\{4, 7\}$
9	X	$\{4\}$

Depth-first topological sort

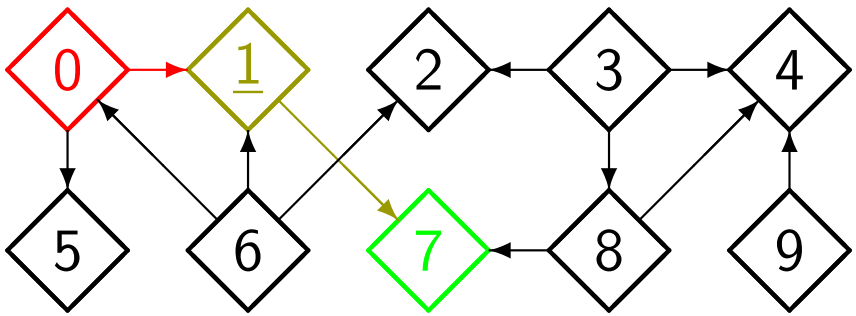


$$\downarrow$$

$$[\ ,\ ,\ ,\ ,\ ,\ ,\ ,\ ,\ ,\ 7]$$

0	X	$\{1, 5\}$
1	X	$\{7\}$
2	X	$\emptyset$
3	X	$\{2, 4, 7, 8\}$
4	X	$\emptyset$
5	X	$\emptyset$
6	X	$\{0, 1, 2\}$
7	✓	$\emptyset$
8	X	$\{4, 7\}$
9	X	$\{4\}$

Depth-first topological sort

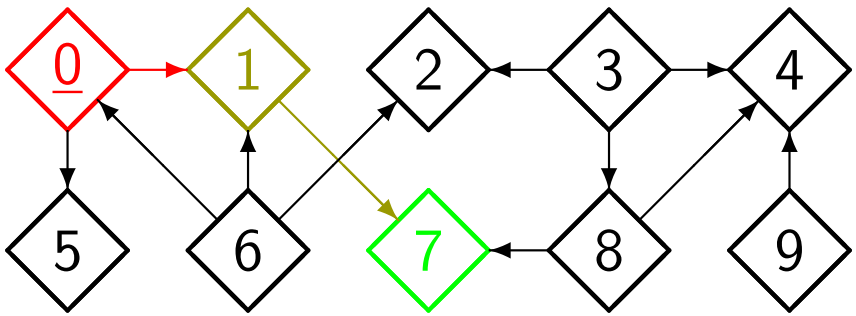


$\downarrow$

$[\ ,\ ,\ ,\ ,\ ,\ ,\ ,\ \ ,1,7]$

0	X	$\{1, 5\}$
1	✓	$\{7\}$
2	X	$\emptyset$
3	X	$\{2, 4, 7, 8\}$
4	X	$\emptyset$
5	X	$\emptyset$
6	X	$\{0, 1, 2\}$
7	✓	$\emptyset$
8	X	$\{4, 7\}$
9	X	$\{4\}$

Depth-first topological sort

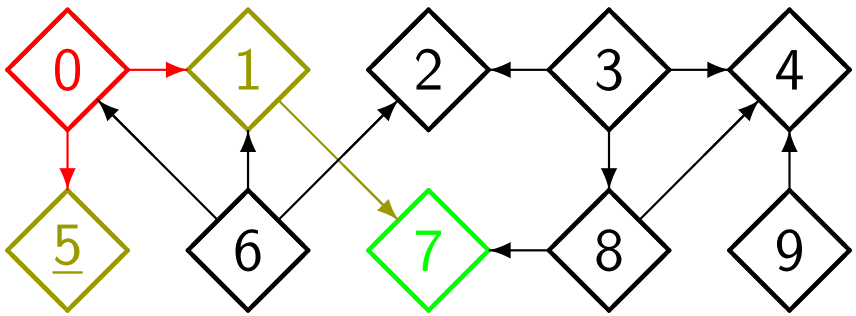


$$\downarrow$$

$$[\ ,\ ,\ ,\ ,\ ,\ ,\ ,\ \ ,1,7]$$

0	X	$\{1, 5\}$
1	✓	$\{7\}$
2	X	$\emptyset$
3	X	$\{2, 4, 7, 8\}$
4	X	$\emptyset$
5	X	$\emptyset$
6	X	$\{0, 1, 2\}$
7	✓	$\emptyset$
8	X	$\{4, 7\}$
9	X	$\{4\}$

Depth-first topological sort

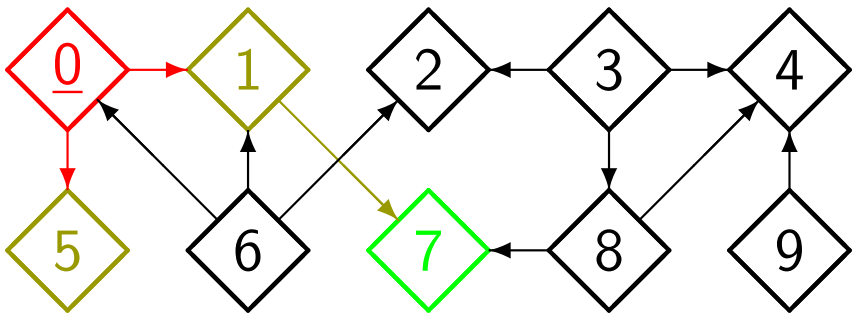


↓

[, , , , , , , ,5,1,7]

0	X	{1, 5}
1	✓	{7}
2	X	∅
3	X	{2, 4, 7, 8}
4	X	∅
5	✓	∅
6	X	{0, 1, 2}
7	✓	∅
8	X	{4, 7}
9	X	{4}

Depth-first topological sort

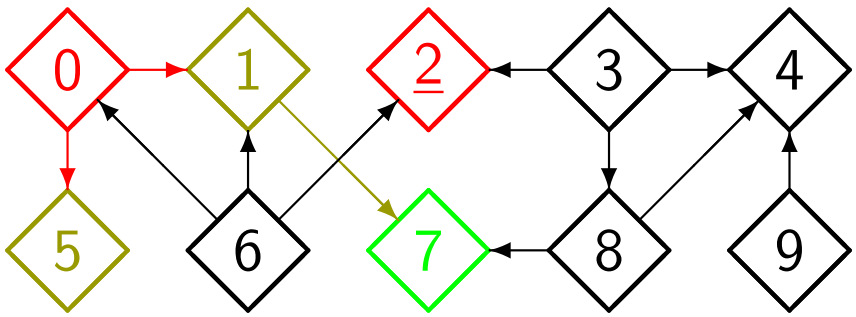


$\downarrow$

[, , , , , , ,**0**,**5**,**1**,**7**]

0	✓	$\{1, 5\}$
1	✓	$\{7\}$
2	X	$\emptyset$
3	X	$\{2, 4, 7, 8\}$
4	X	$\emptyset$
5	✓	$\emptyset$
6	X	$\{0, 1, 2\}$
7	✓	$\emptyset$
8	X	$\{4, 7\}$
9	X	$\{4\}$

Depth-first topological sort

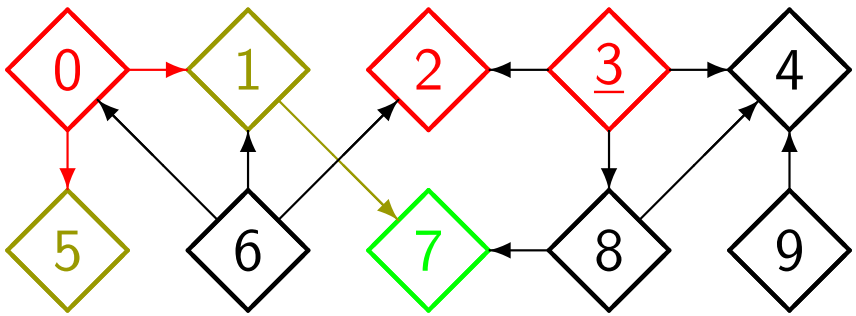


↓

[, , , , , ,2,0,5,1,7]

0	✓	{1, 5}
1	✓	{7}
2	✓	∅
3	X	{2, 4, 7, 8}
4	X	∅
5	✓	∅
6	X	{0, 1, 2}
7	✓	∅
8	X	{4, 7}
9	X	{4}

Depth-first topological sort

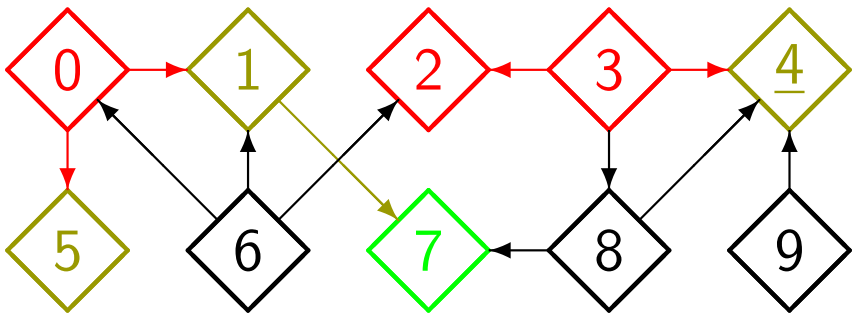


↓

[, , , , , ,2,0,5,1,7]

0	✓	{1, 5}
1	✓	{7}
2	✓	∅
3	X	{2, 4, 7, 8}
4	X	∅
5	✓	∅
6	X	{0, 1, 2}
7	✓	∅
8	X	{4, 7}
9	X	{4}

Depth-first topological sort

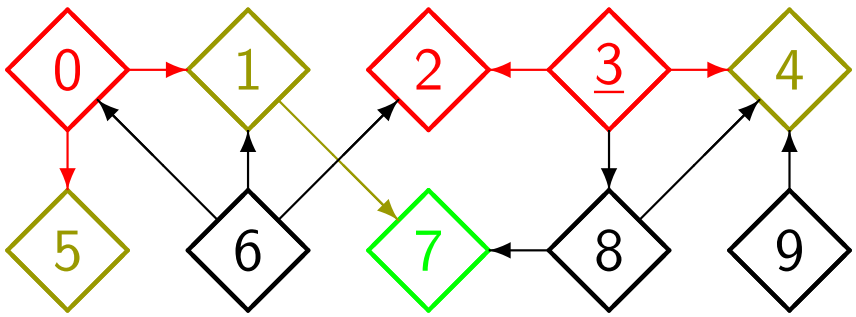


↓

[, , , ,4,2,0,5,1,7]

0	✓	{1, 5}
1	✓	{7}
2	✓	∅
3	X	{2, 4, 7, 8}
4	✓	∅
5	✓	∅
6	X	{0, 1, 2}
7	✓	∅
8	X	{4, 7}
9	X	{4}

Depth-first topological sort

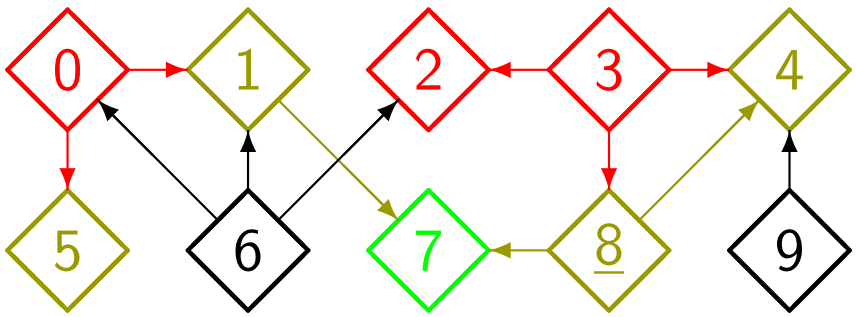


↓

[, , , ,4,2,0,5,1,7]

0	✓	{1, 5}
1	✓	{7}
2	✓	∅
3	X	{2, 4, 7, 8}
4	✓	∅
5	✓	∅
6	X	{0, 1, 2}
7	✓	∅
8	X	{4, 7}
9	X	{4}

Depth-first topological sort

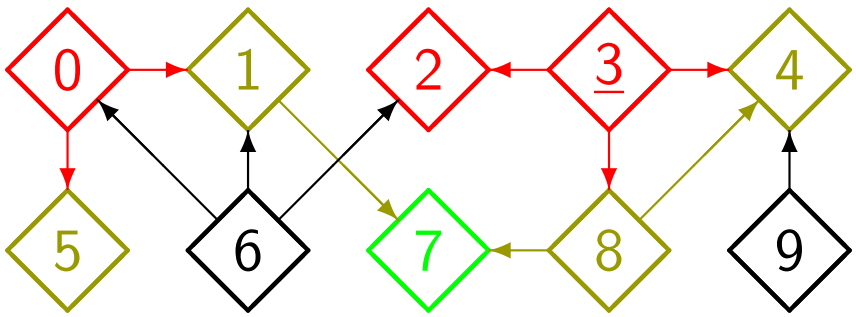


↓

[, , ,8,4,2,0,5,1,7]

0	✓	{1, 5}
1	✓	{7}
2	✓	∅
3	X	{2, 4, 7, 8}
4	✓	∅
5	✓	∅
6	X	{0, 1, 2}
7	✓	∅
8	✓	{4, 7}
9	X	{4}

Depth-first topological sort

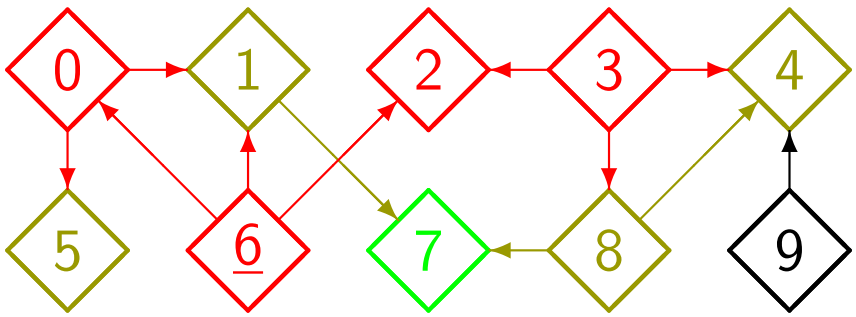


↓

[, , ,3,8,4,2,0,5,1,7]

0	✓	{1, 5}
1	✓	{7}
2	✓	∅
3	✓	{2, 4, 7, 8}
4	✓	∅
5	✓	∅
6	X	{0, 1, 2}
7	✓	∅
8	✓	{4, 7}
9	X	{4}

Depth-first topological sort

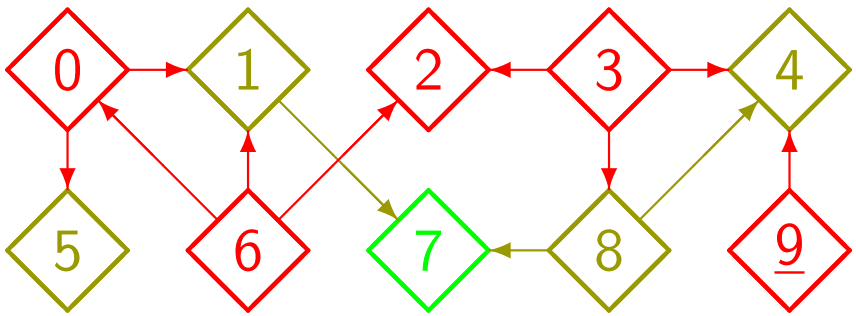


↓

[,6,3,8,4,2,0,5,1,7]

0	✓	{1, 5}
1	✓	{7}
2	✓	∅
3	✓	{2, 4, 7, 8}
4	✓	∅
5	✓	∅
6	✓	{0, 1, 2}
7	✓	∅
8	✓	{4, 7}
9	X	{4}

Depth-first topological sort



[9,6,3,8,4,2,0,5,1,7]

0	✓	$\{1, 5\}$
1	✓	$\{7\}$
2	✓	$\emptyset$
3	✓	$\{2, 4, 7, 8\}$
4	✓	$\emptyset$
5	✓	$\emptyset$
6	✓	$\{0, 1, 2\}$
7	✓	$\emptyset$
8	✓	$\{4, 7\}$
9	X	$\{4\}$

Depth-first topological sort

The depth-first topological sort is

$$[9, 6, 3, 8, 4, 2, 0, 5, 1, 7]$$

It is easy to verify that this satisfies:

- node 0 occurs before nodes 1 and 5
- node 1 occurs before node 7
- node 3 occurs before nodes 2, 4 and 8
- node 6 occurs before nodes 0, 1 and 2
- node 8 occurs before nodes 4 and 7
- node 9 occurs before node 4

Reference-counting topological sort

2.5: Reference-counting Topological Sort

- Find a node with no predecessors.
- Add it to the end of the current sort — i.e. after any previously sorted nodes, and before any of the currently unsorted nodes.
- "Remove" it from the graph.
- Repeat the process until no nodes are left.

This process will terminate with a topological sort.

Reference-counting topological sort

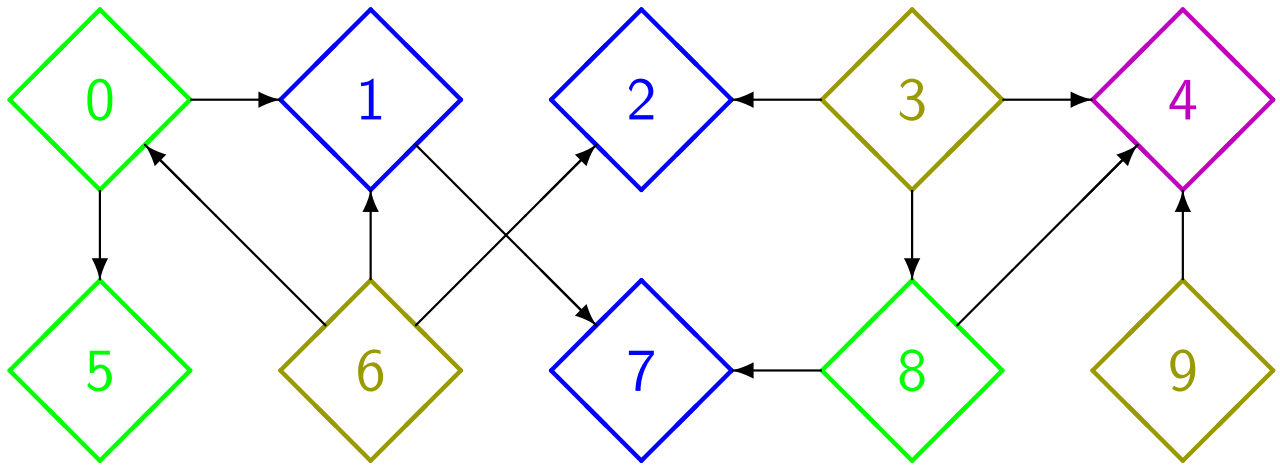
```
the sort is empty
while (there are nodes left in the graph) {
    find a node with no predecessors;
    add it after any previously sorted nodes;
    remove it from the graph;
}
```

Reference-counting topological sort

- `find a node with no predecessors`
  Maintain a "reference count" count for each node. Look for nodes with a zero reference count.
- `add it after any previously sorted nodes`
  Use an incrementing pointer, or use a list structure in which elements are added at the tail of the list (e.g. `add(element)` in `Lists`).
- `remove it from the graph`
  Mark the node as visited, and decrease the reference count of any successors

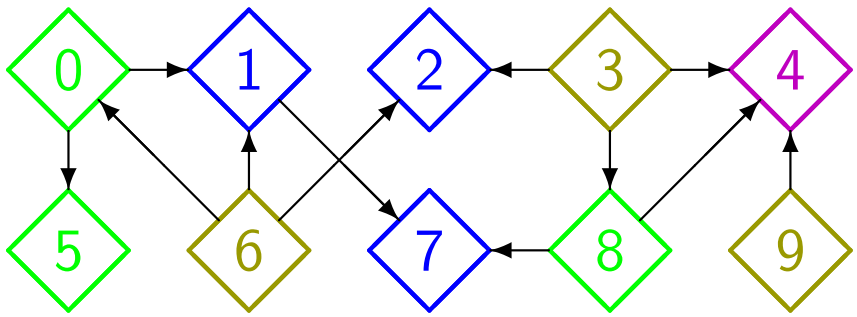
Reference Count

The reference count is simply the number of edges leading in to a given node.



Node	0	1	2	3	4	5	6	7	8	9
Reference count	1	2	2	0	3	1	0	2	1	0

Reference-counting topological sort

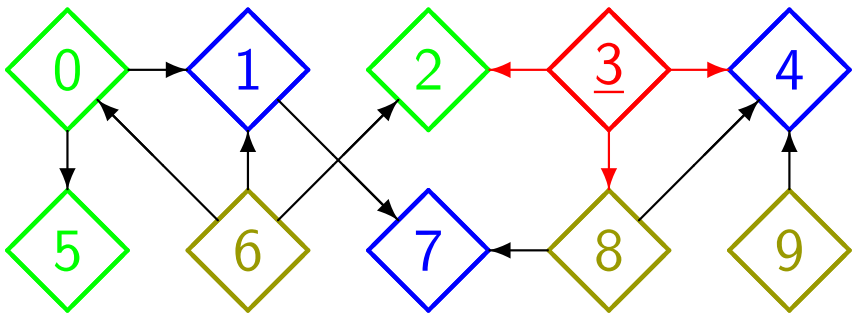


$\downarrow$

[, , , , , , , , ,]

0	X	1	{1, 5}
1	X	2	{7}
2	X	2	∅
3	X	0	{2, 4, 8}
4	X	3	∅
5	X	1	∅
6	X	0	{0, 1, 2}
7	X	2	∅
8	X	1	{4, 7}
9	X	0	{4}

Reference-counting topogical sort

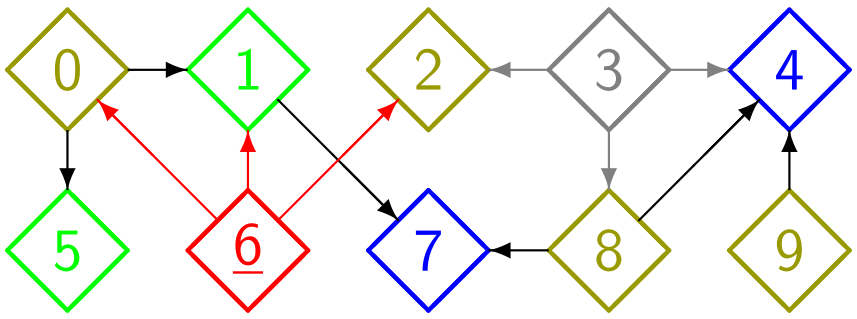


↓

[3, , , , , , , , ,]

0	X	1	{1, 5}
1	X	2	{7}
2	X	1	∅
3	✓	0	{2, 4, 8}
4	X	2	∅
5	X	1	∅
6	X	0	{0, 1, 2}
7	X	2	∅
8	X	0	{4, 7}
9	X	0	{4}

Reference-counting topological sort

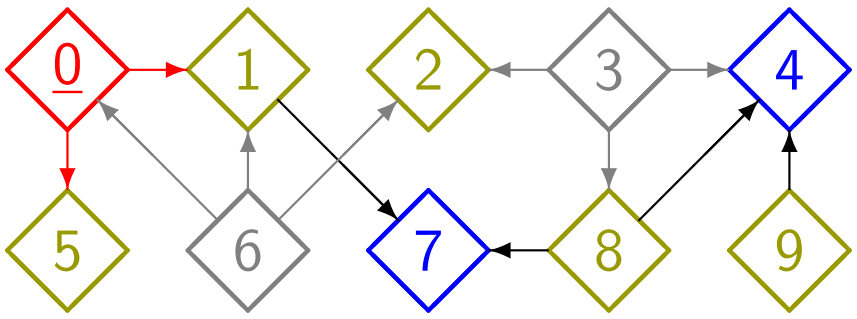


↓

[3,6, , , , , , , ,]

0	X	0	{1, 5}
1	X	1	{7}
2	X	0	∅
3	✓	0	{2, 4, 8}
4	X	2	∅
5	X	1	∅
6	✓	0	{0, 1, 2}
7	X	2	∅
8	X	0	{4, 7}
9	X	0	{4}

Reference-counting topological sort

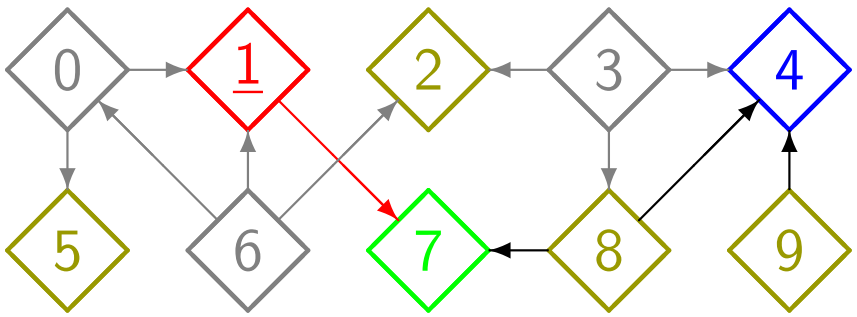


↓

[3,6,0, , , , , , ,]

0	✓	0	{1, 5}
1	X	0	{7}
2	X	0	∅
3	✓	0	{2, 4, 8}
4	X	2	∅
5	X	0	∅
6	✓	0	{0, 1, 2}
7	X	2	∅
8	X	0	{4, 7}
9	X	0	{4}

Reference-counting topological sort

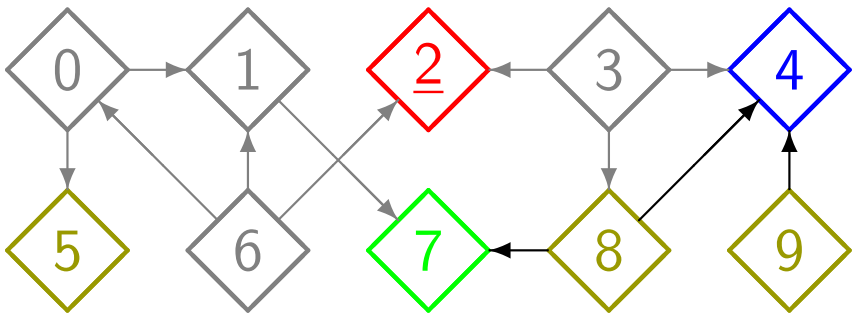


$\downarrow$

[3,6,0,1, , , , , ,]

0	✓	0	{1, 5}
1	✓	0	{7}
2	X	0	∅
3	✓	0	{2, 4, 8}
4	X	2	∅
5	X	0	∅
6	✓	0	{0, 1, 2}
7	X	1	∅
8	X	0	{4, 7}
9	X	0	{4}

Reference-counting topological sort

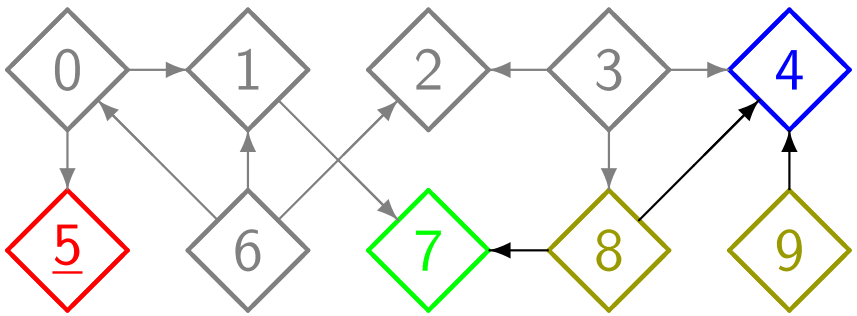


↓

[3,6,0,1,2, , , , ,]

0	✓	0	{1, 5}
1	✓	0	{7}
2	✓	0	∅
3	✓	0	{2, 4, 8}
4	X	2	∅
5	X	0	∅
6	✓	0	{0, 1, 2}
7	X	1	∅
8	X	0	{4, 7}
9	X	0	{4}

Reference-counting topological sort

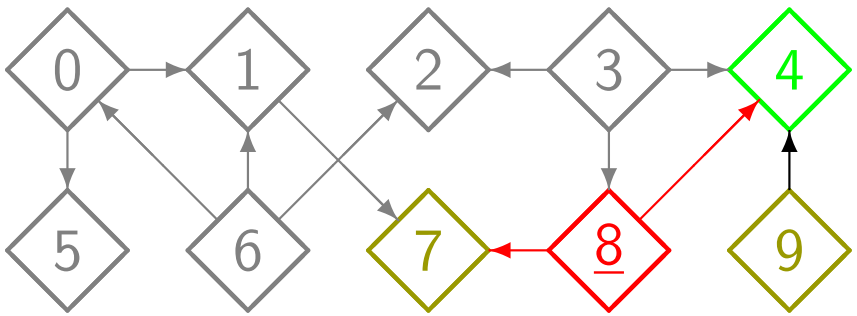


$\downarrow$

[3,6,0,1,2,5, , , ,]

0	✓	0	{1, 5}
1	✓	0	{7}
2	✓	0	∅
3	✓	0	{2, 4, 8}
4	X	2	∅
5	✓	0	∅
6	✓	0	{0, 1, 2}
7	X	1	∅
8	X	0	{4, 7}
9	X	0	{4}

Reference-counting topological sort

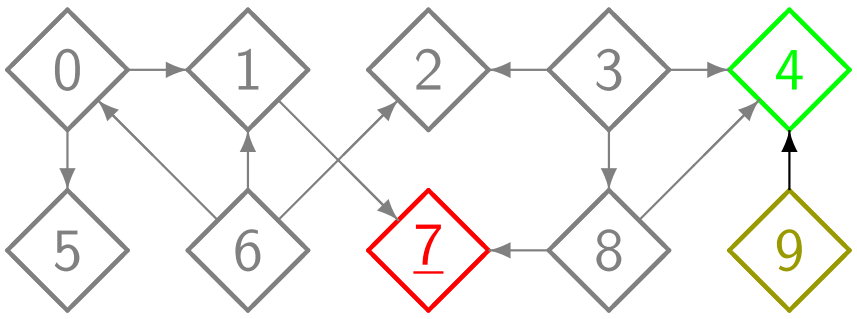


$\downarrow$

[3,6,0,1,2,5,8, , ,]

0	✓	0	{1, 5}
1	✓	0	{7}
2	✓	0	∅
3	✓	0	{2, 4, 8}
4	X	1	∅
5	✓	0	∅
6	✓	0	{0, 1, 2}
7	X	0	∅
8	✓	0	{4, 7}
9	X	0	{4}

Reference-counting topological sort

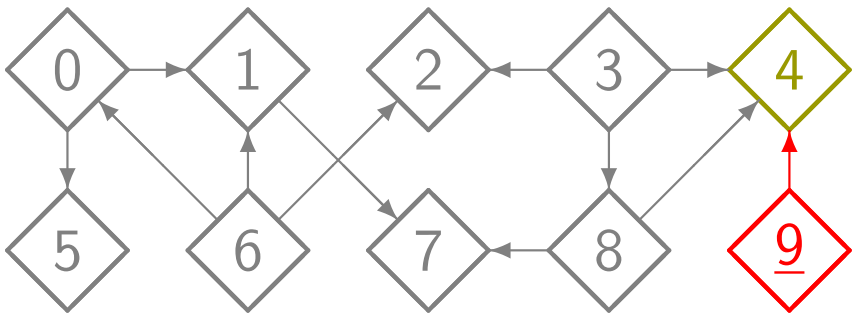


$\downarrow$

[3,6,0,1,2,5,8,7, ,]

0	✓	0	{1, 5}
1	✓	0	{7}
2	✓	0	∅
3	✓	0	{2, 4, 8}
4	X	1	∅
5	✓	0	∅
6	✓	0	{0, 1, 2}
7	✓	0	∅
8	✓	0	{4, 7}
9	X	0	{4}

Reference-counting topological sort

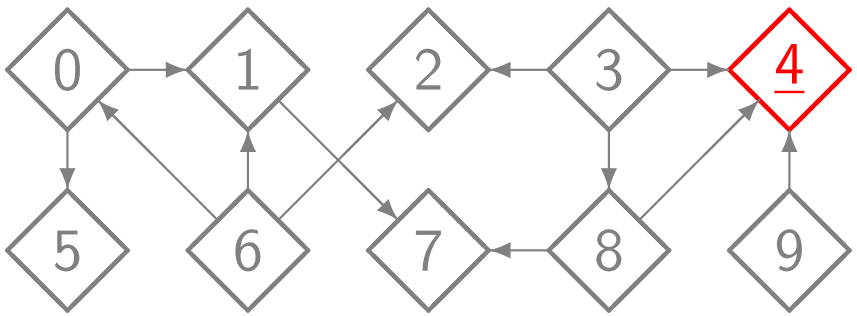


$\downarrow$

[3,6,0,1,2,5,8,7,9,]

0	✓	0	{1, 5}
1	✓	0	{7}
2	✓	0	∅
3	✓	0	{2, 4, 8}
4	X	0	∅
5	✓	0	∅
6	✓	0	{0, 1, 2}
7	✓	0	∅
8	✓	0	{4, 7}
9	✓	0	{4}

Reference-counting topological sort



[3,6,0,1,2,5,8,7,9,4]

0	✓	0	$\{1, 5\}$
1	✓	0	$\{7\}$
2	✓	0	$\emptyset$
3	✓	0	$\{2, 4, 8\}$
4	✓	**0**	$\emptyset$
5	✓	0	$\emptyset$
6	✓	0	$\{0, 1, 2\}$
7	✓	0	$\emptyset$
8	✓	0	$\{4, 7\}$
9	✓	0	$\{4\}$

Reference-counting topological sort

The reference-counting topological sort is

$$\mathbf{[3, 6, 0, 1, 2, 5, 8, 7, 9, 4]}$$

It is easy to verify that this satisfies:

- node 0 occurs before nodes 1 and 5
- node 1 occurs before node 7
- node 3 occurs before nodes 2, 4 and 8
- node 6 occurs before nodes 0, 1 and 2
- node 8 occurs before nodes 4 and 7
- node 9 occurs before node 4

Remarks

2.6: Notes

2.6.1: Topological Sorts are not Unique

Notice that topological sorts are not unique. In general, a non-cyclic graph will have many topological sorts.

Uses

2.6.2: Some Uses for Topological Sorts

- Compiler garbage collection — any data items on the heap that occur before the known valid data items on the heap, must be garbage (since they are not referenced by any valid heap data items).
- Determining a course of study — modules can only be studied when their prerequisites have been passed.

Implementation

3: Implementation
3.1: Depth First
3.1.1: Visiting a Node
On visiting a node:

- check if it has already been visited;
- if not:
  - note that it is being visited
  - visit its children;
  - add it to the sort;

Has the node been sorted?

3.1.1 A: Has the node been sorted?

```
if (visited.contains(node)) return;
```

Note the visit

3.1.1 B: Note that the node is being visited

```
visited.add(node);
```

Sort the node's children

3.1.1 C: Sort the node's children

```
for (T neighbour:  getNeighbours(node)) {
    visitNode(neighbour);
}
```

Add the node to the sort

3.1.1 D: Add the node to the sort

```
sort.push(node); // use e.g. Stack
```

3.1.1 E: The visitNode method

```
private void visitNode(T node) {
    if (visited.contains(node)) return;
    visited.add(node);
    for (T neighbour:  getNeighbours(node)) {
        visitNode(neighbour);
    }
    sort.push(node);
}
```

Full sort

3.1.2: The Full Sort

```
private Stack<T> sort = new Stack<T>();
private Set<T> visited = new HashSet<T>();

private List<T> getSort() {
    for (T node:  nodes()) {
        if (!visited.contains(node)) {
            visitNode(node);
        }
    }
    Collections.reverse(sort);
    return sort;
}
```

Reference Count

3.2: Reference Count

We need to:

- set up the reference counts;
- initialise the sort results;
- perform the sort.

Setting up the reference counts

3.2.1: Setting Up the Reference Counts

```
private void setUpReferenceCounts() {
    for (T node:  nodes()) {
        for (T successor:  successors(node)) {
            successor.increaseReferenceCount();
        }
    }
}
```

Initialise sort results

3.2.2: Initialise the Sort Results

```
private List<T> sort = new ArrayList<T>();
```

Perform the sort

3.2.3: Visit a Node

On visiting a node we need to:

- add the node to the sort;
- reduce the reference count of its children.

Add the node to the sort

3.2.3 A: Add the node to the sort

```
sort.add(node);
```

Reduce the reference count of the children

3.2.3 B: Reduce the reference count of the children

```
for (T successor:  successors(node)) {
    successor.decreaseReferenceCount();
}
```

The full visit method

3.2.3 C: The visitNode method

```
private void visitNode(T node) {
    sort.add(node);
    for (T successor:  successors(node)) {
        successor.decreaseReferenceCount();
    }
}
```

The full sorting method

3.2.4: The Full Sorting

```
private void doSort() {
    T node;
    while ((node = nextReferenceZeroNode()) != null) {
        visitNode(node);
    }
}
```

End of topological sort lecture