DAGs and Topological Ordering

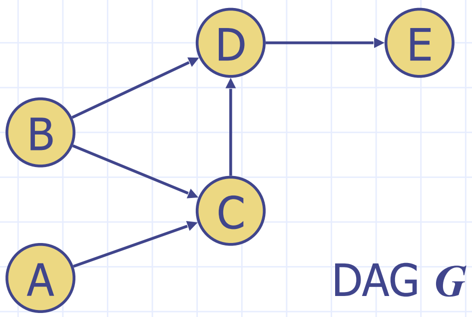
- A directed acyclic graph (DAG) is a digraph that has no directed cycles
- A topological ordering of a digraph is a numbering

  $$v_1, \ldots, v_n$$

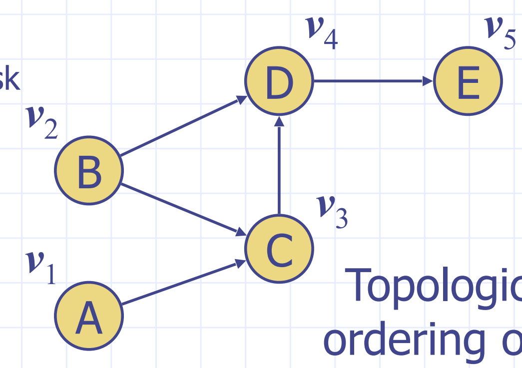
  of the vertices such that for every edge (v_i, v_j), we have $i < j$
- Example: in a task scheduling digraph, a topological ordering a task sequence that satisfies the precedence constraints

Theorem

A digraph admits a topological ordering if and only if it is a DAG



DAG G

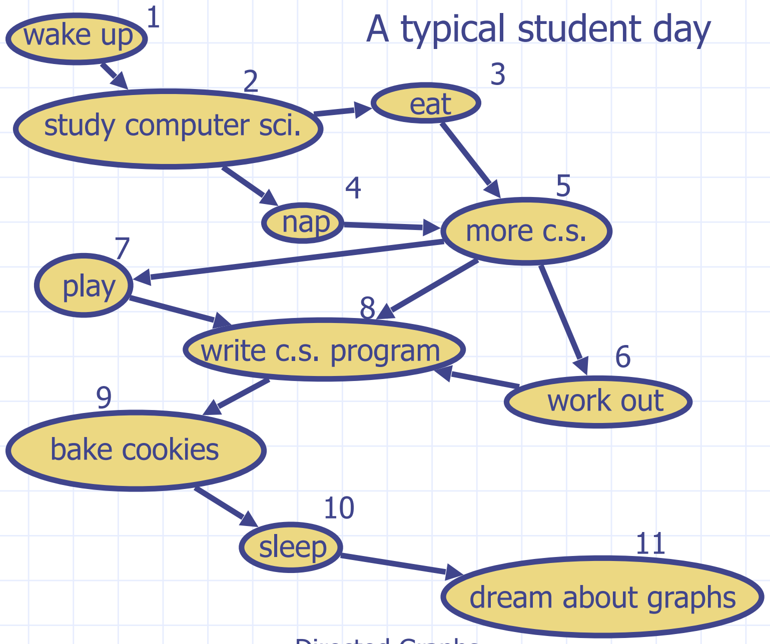


Topological ordering of G

© 2010 Goodrich, Tamassia

Topological Sorting

- Number vertices, so that (u,v) in E implies u < v



© 2010 Goodrich, Tamassia

Algorithm for Topological Sorting

- Note: This algorithm is different than the one in the book

```
Algorithm TopologicalSort(G)
    H ← G          // Temporary copy of G
    n ← G.numVertices()
    while H is not empty do
        Let v be a vertex with no outgoing edges
        Label v ← n
        n ← n – 1
        Remove v from H
```

- Running time: $O(n + m)$

Implementation with DFS

- Simulate the algorithm by using depth-first search
- O(n+m) time.

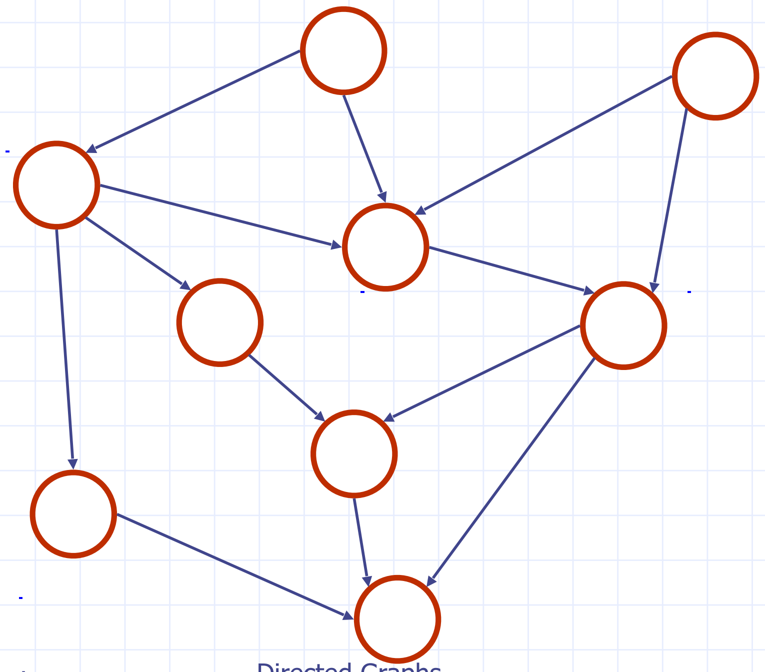
Algorithm ***topologicalDFS(G)***
  Input dag ***G***
  Output topological ordering of ***G***
  n ← ***G.numVertices***()
  for all ***u*** ∈ ***G.vertices***()
    u.setLabel(***UNEXPLORED***)
  for all ***v*** ∈ ***G.vertices***()
    if ***v.getLabel***() = ***UNEXPLORED***
      topologicalDFS(***G, v***)

Algorithm ***topologicalDFS(G, v)***
  Input graph ***G*** and a start vertex ***v*** of ***G***
  Output labeling of the vertices of ***G*** in the connected component of ***v***
  v.setLabel(***VISITED***)
  for all ***e*** ∈ ***v.outEdges***()
    { outgoing edges }
    w ← ***e.opposite***(***v***)
    if ***w.getLabel***() = ***UNEXPLORED***
      { ***e*** is a discovery edge }
      topologicalDFS(***G, w***)
    else
      { ***e*** is a forward or cross edge }
  Label ***v*** with topological number ***n***
  n ← ***n*** - 1

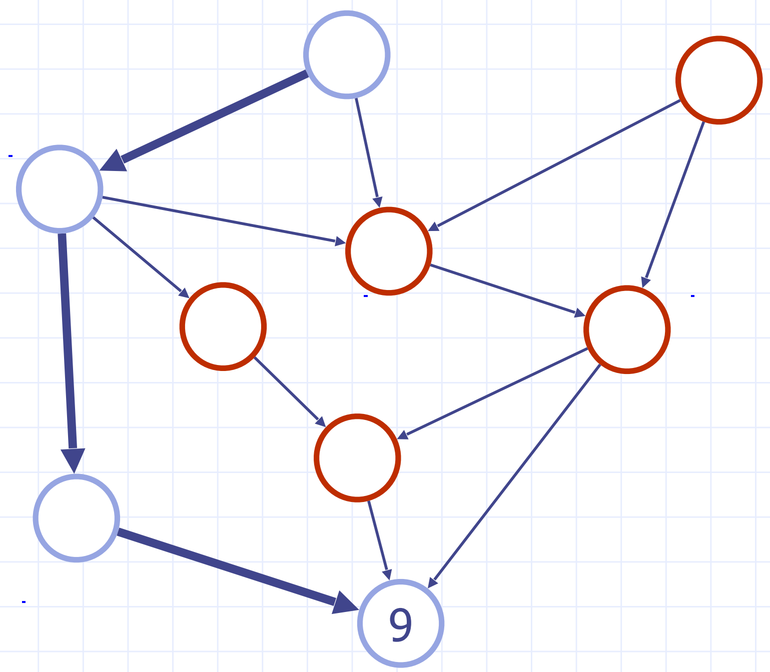
Topological Sort Algorithm

- We can use a DFS in our algorithm to determine a topological sort!
  - The idea is:
    - When you do a DFS on a directed acyclic graph, eventually you will reach a node with no outgoing edges. Why?
      - Because if this never happened, you hit a cycle, because the number of nodes if finite.
    - This **node** that you reach in a DFS is "**safe**" to place at the end of the topological sort.
      - Think of leaving for work!
  - Now what we find is
    - If we have added each of the vertices "**below**" a vertex into our topological sort, it is safe then to add this one in.
      - If we added in Leaving for work at the end, then we can surely add taking a shower.

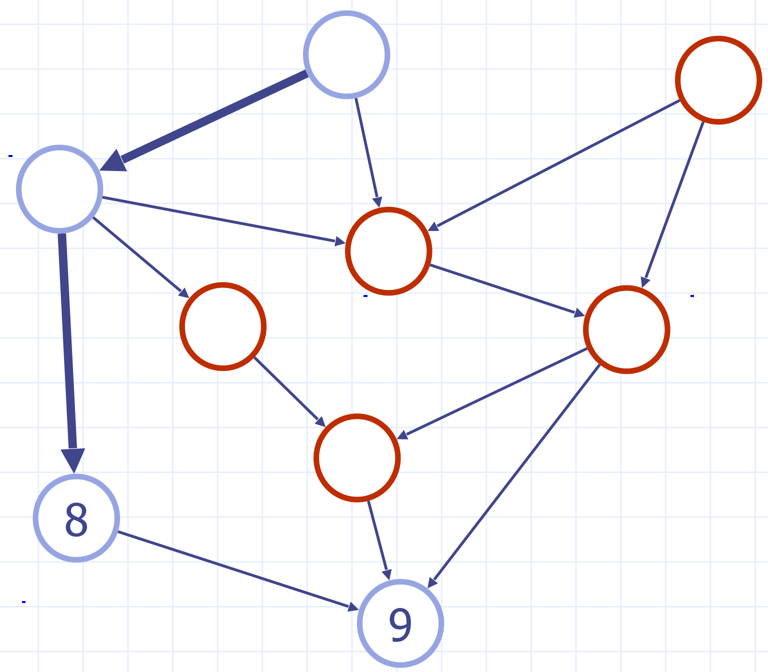
Topological Sorting Example



Topological Sorting Example

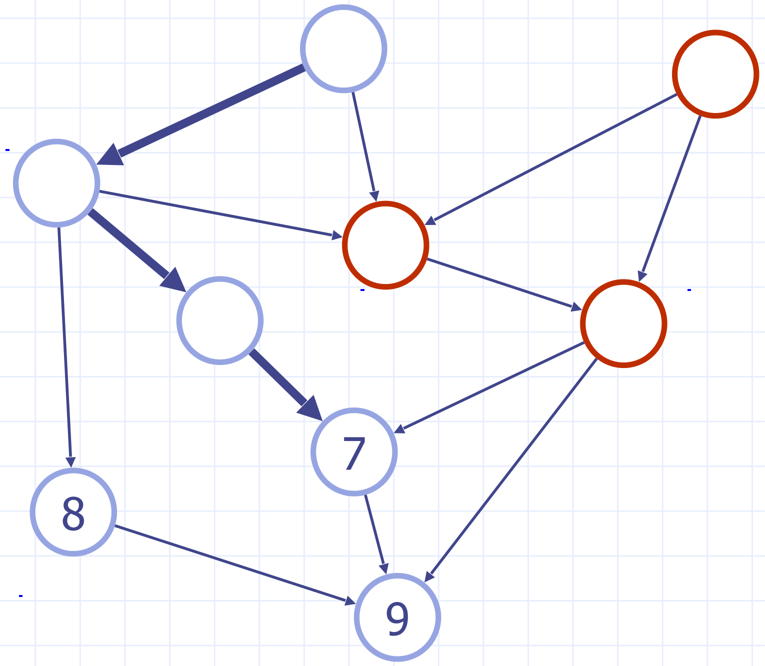


Topological Sorting Example

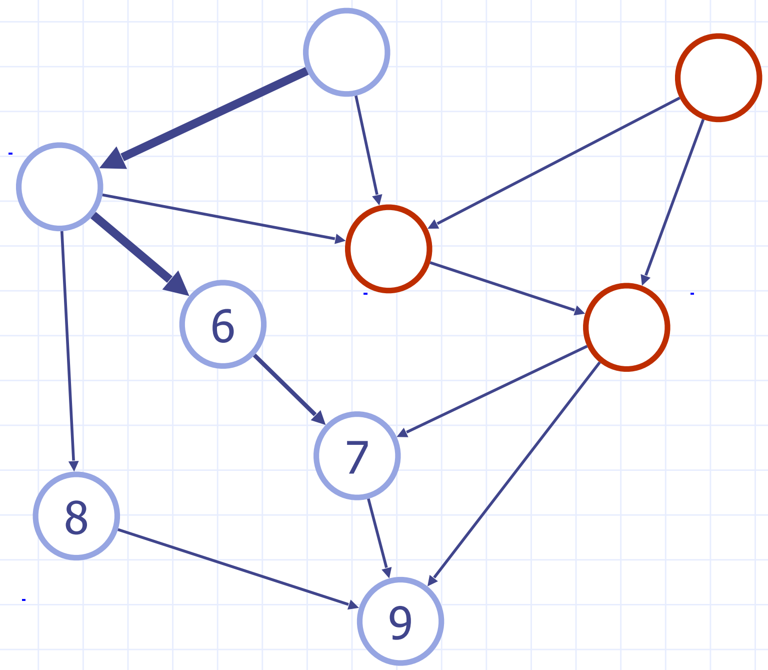


© 2010 Goodrich, Tamassia

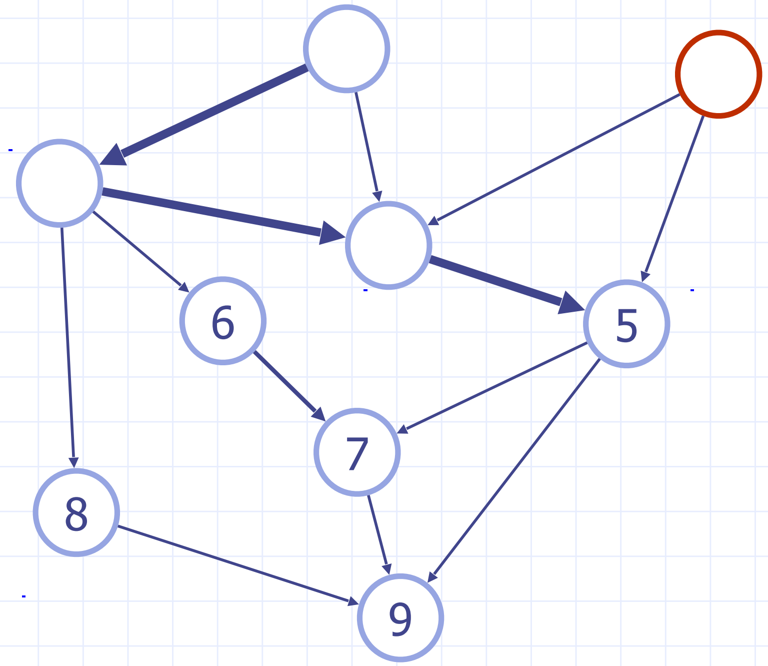
Topological Sorting Example



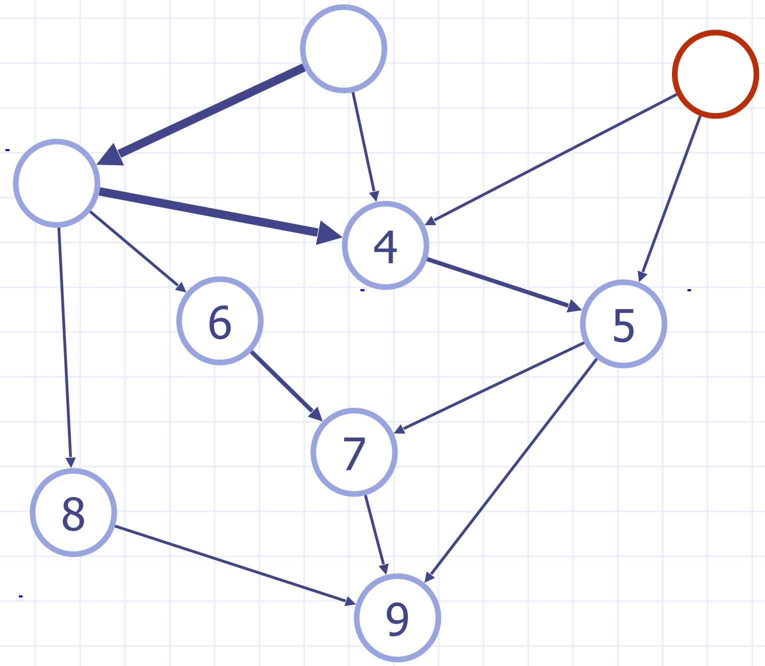
Topological Sorting Example



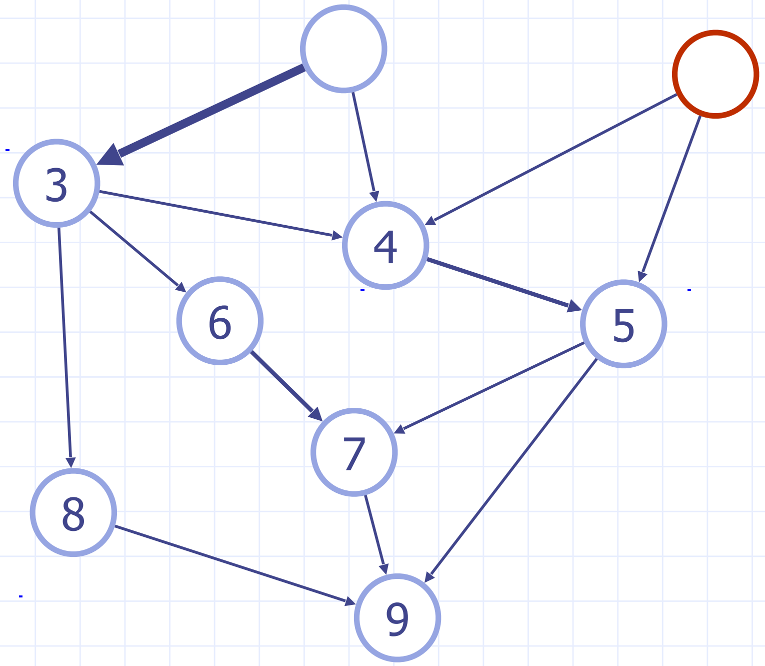
Topological Sorting Example



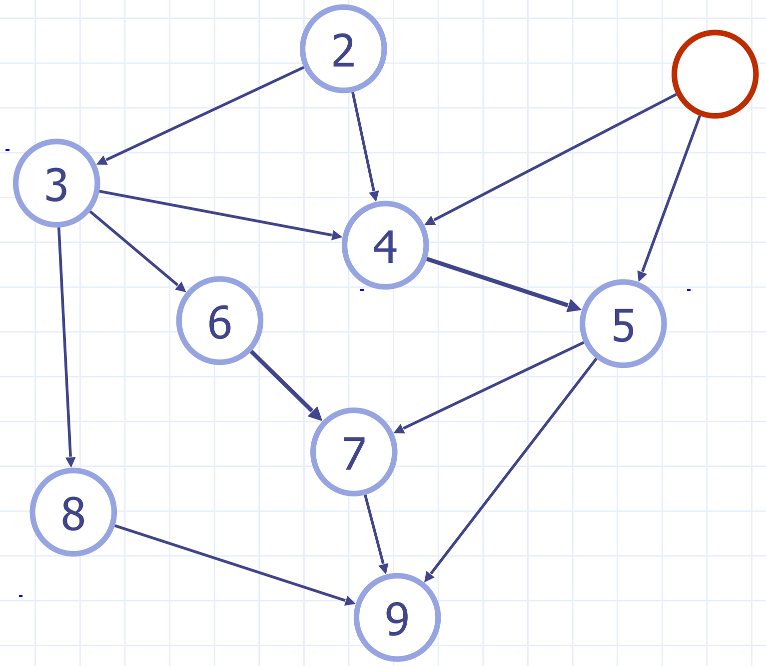
Topological Sorting Example



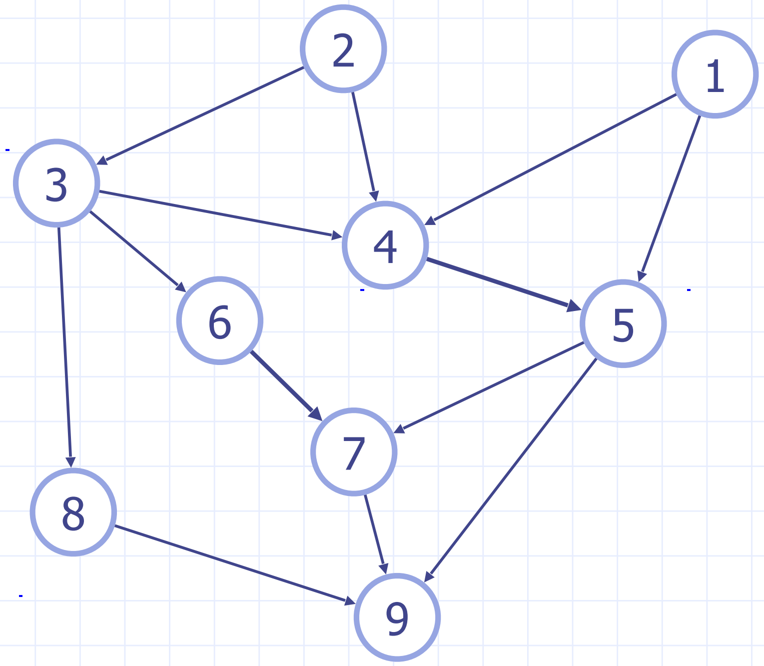
Topological Sorting Example



Topological Sorting Example

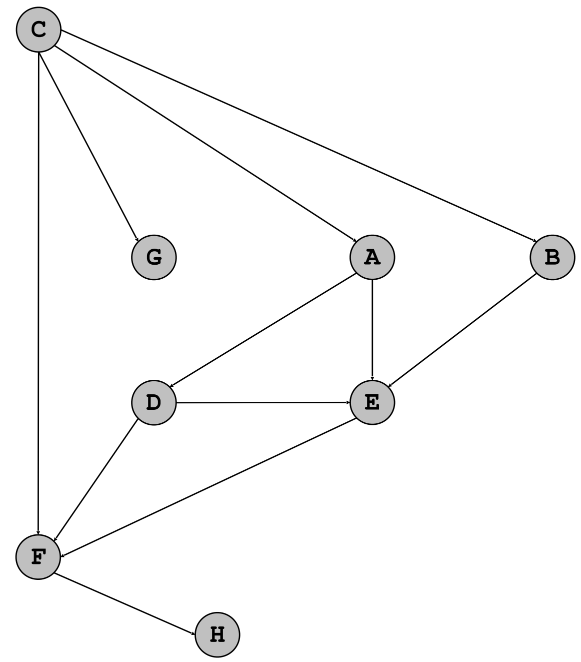


Topological Sorting Example



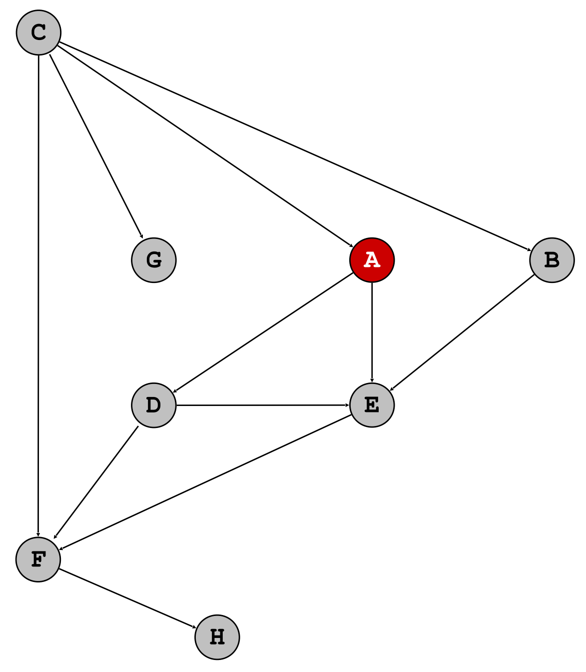
© 2010 Goodrich, Tamassia

Topological Sort: DFS



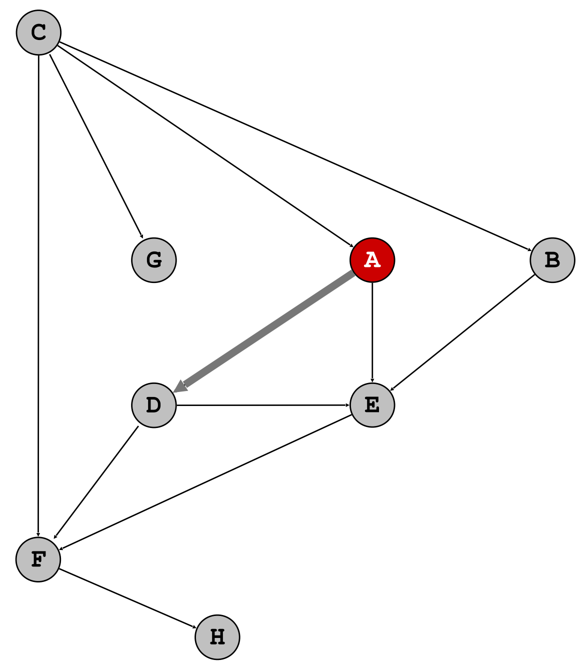
A ~~1~~ 0 2
B ~~1~~ 0 3
C 0 1
D ~~1~~ 0 5
E ~~3~~ ~~2~~ ~~1~~ ~~0~~ 6
F ~~3~~ ~~2~~ ~~1~~ 0 7
G ~~1~~ 0 4
H ~~2~~ ~~1~~ 0 8

Topological Sort: DFS



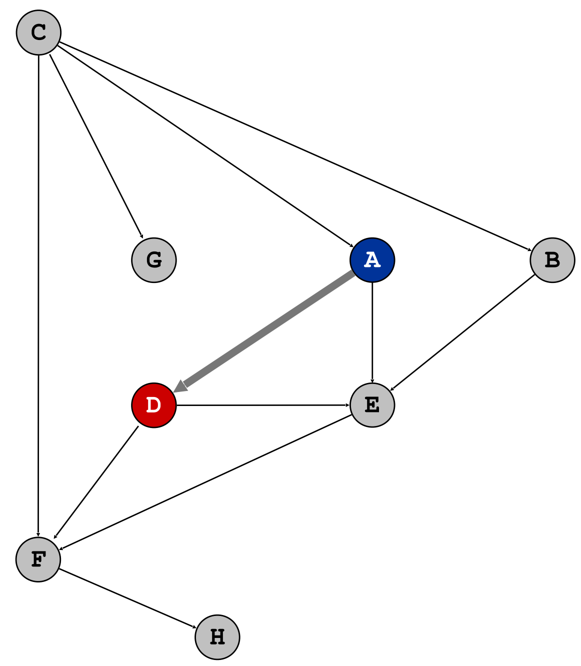
`dfs(A)`

Topological Sort: DFS



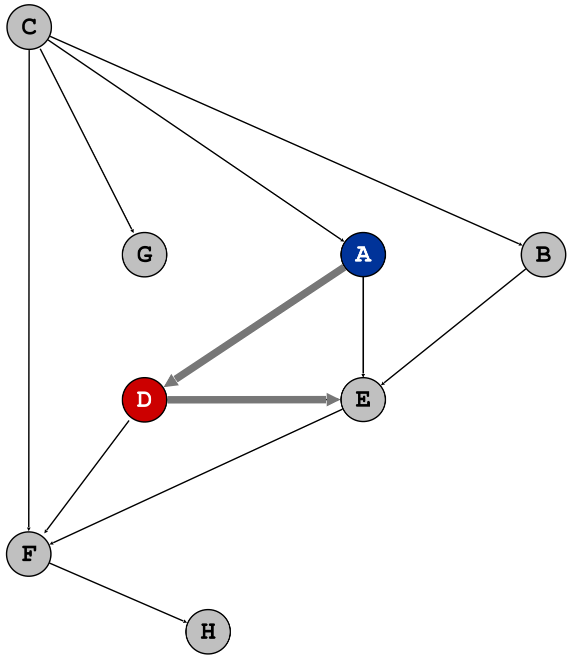
`dfs(A)`

Topological Sort: DFS



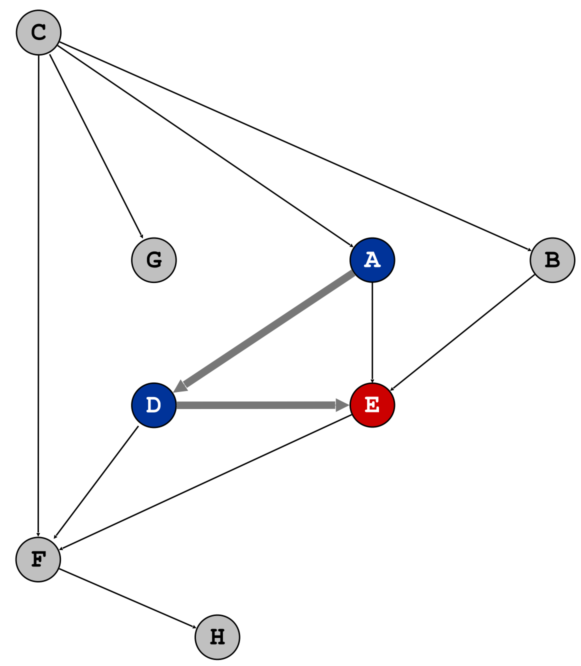

```
dfs(A)
  dfs(D)
```

Topological Sort: DFS



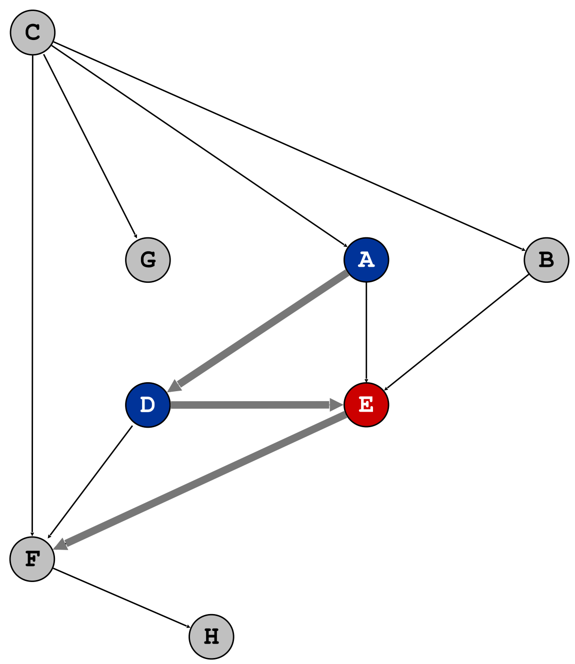

```
dfs(A)
  dfs(D)
```

Topological Sort: DFS



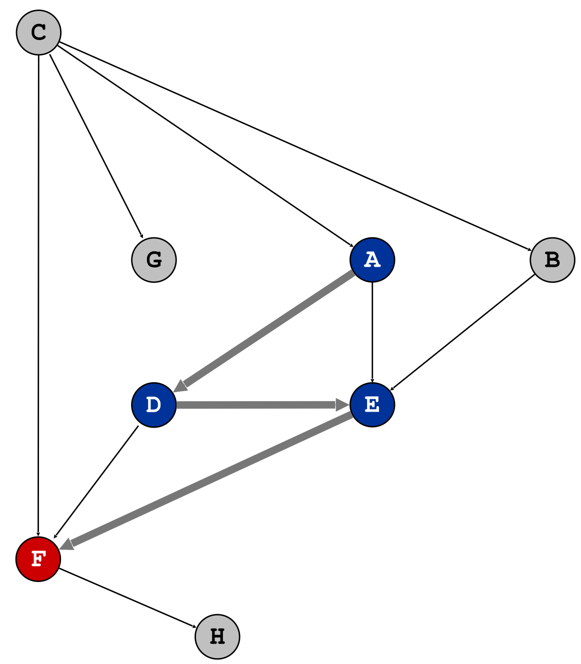

```
dfs(A)
  dfs(D)
    dfs(E)
```

Topological Sort: DFS



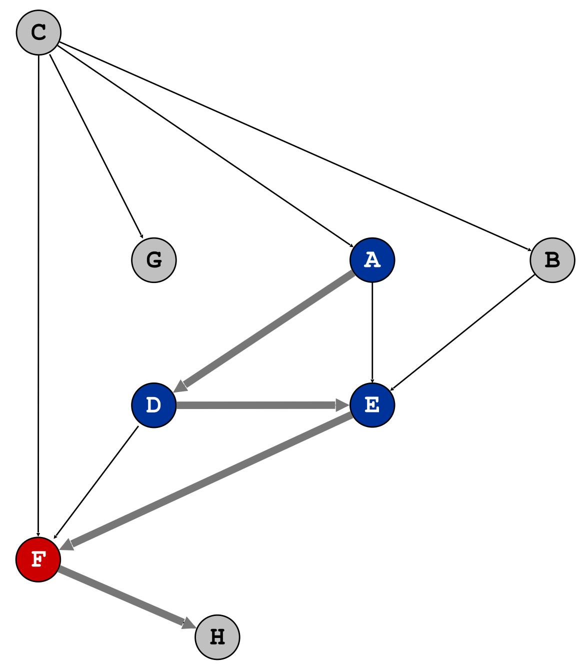

```
dfs(A)
  dfs(D)
    dfs(E)
```

Topological Sort: DFS



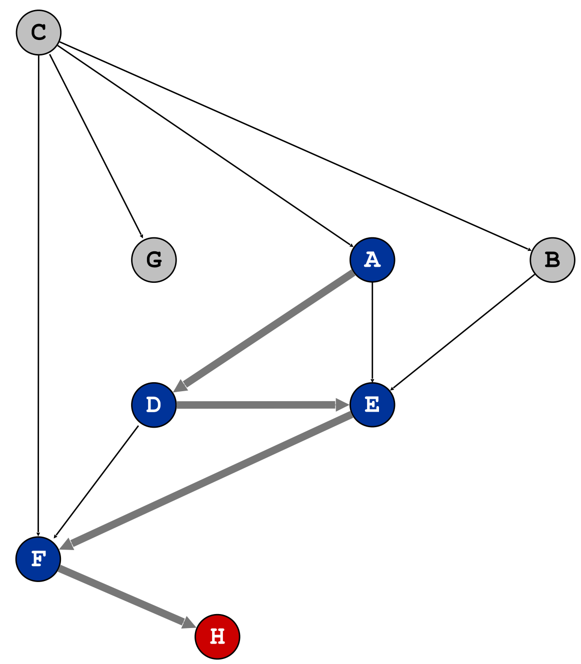

```
dfs(A)
  dfs(D)
    dfs(E)
      dfs(F)
```

Topological Sort: DFS



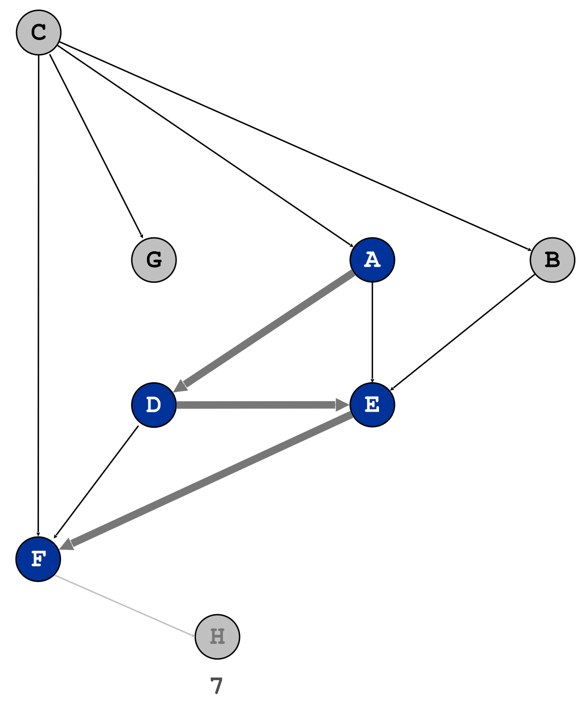

```
dfs(A)
  dfs(D)
    dfs(E)
      dfs(F)
```

Topological Sort: DFS



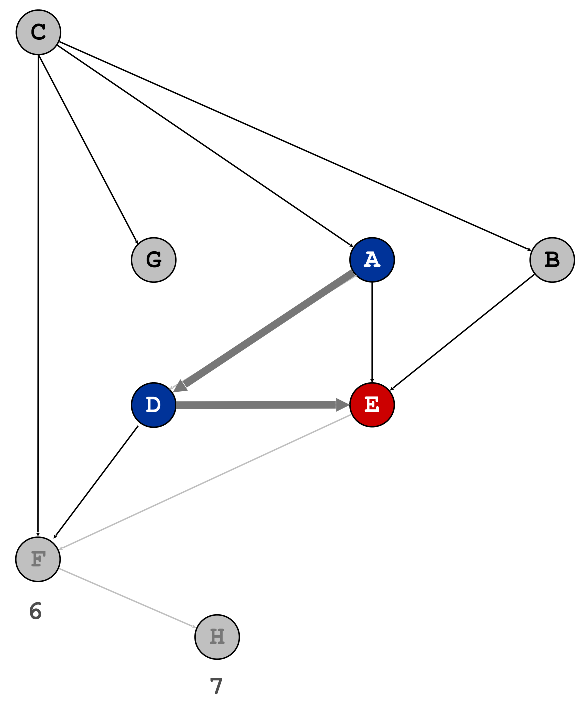

```
dfs(A)
  dfs(D)
    dfs(E)
      dfs(F)
         dfs(H)
```

Topological Sort: DFS



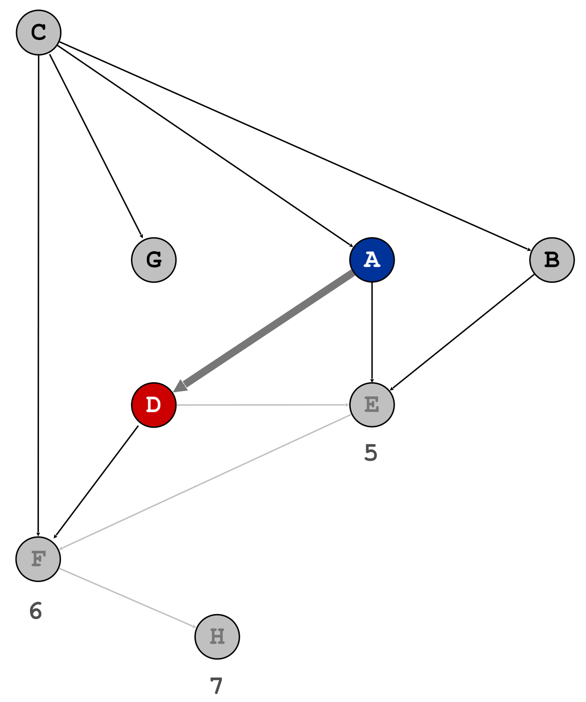

```
dfs(A)
  dfs(D)
    dfs(E)
      dfs(F)
```

Topological Sort: DFS



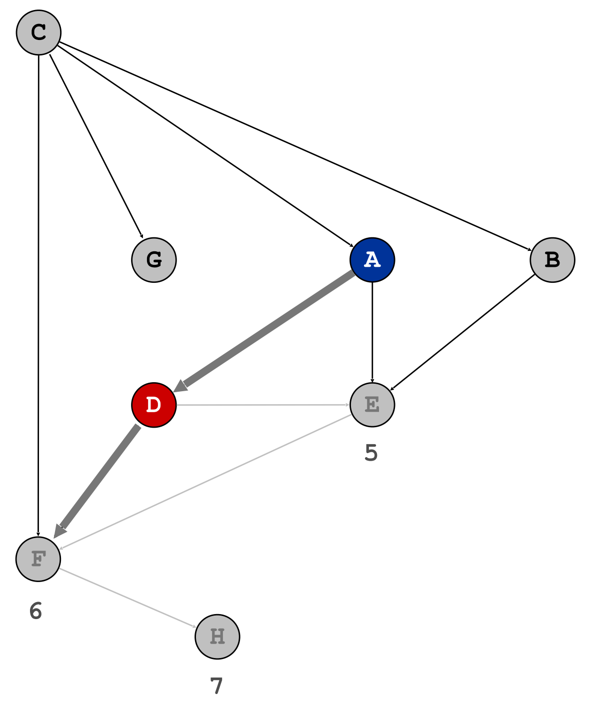
dfs(A)
  dfs(D)
    dfs(E)

Topological Sort: DFS



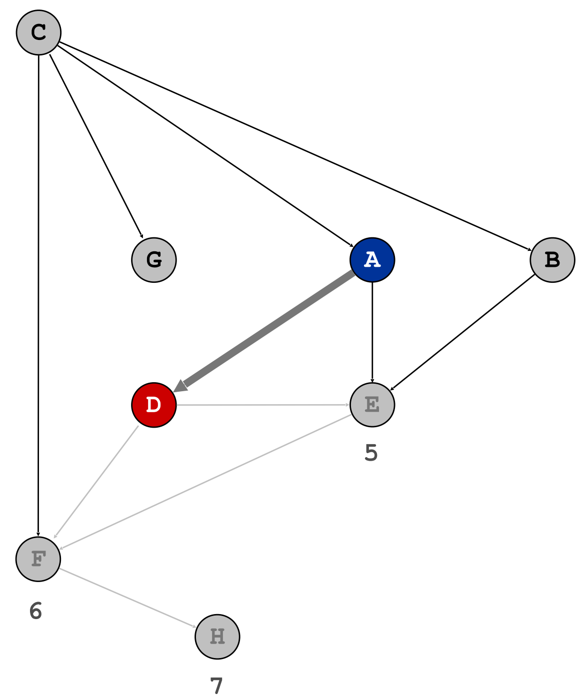

```
dfs(A)
  dfs(D)
```

Topological Sort: DFS



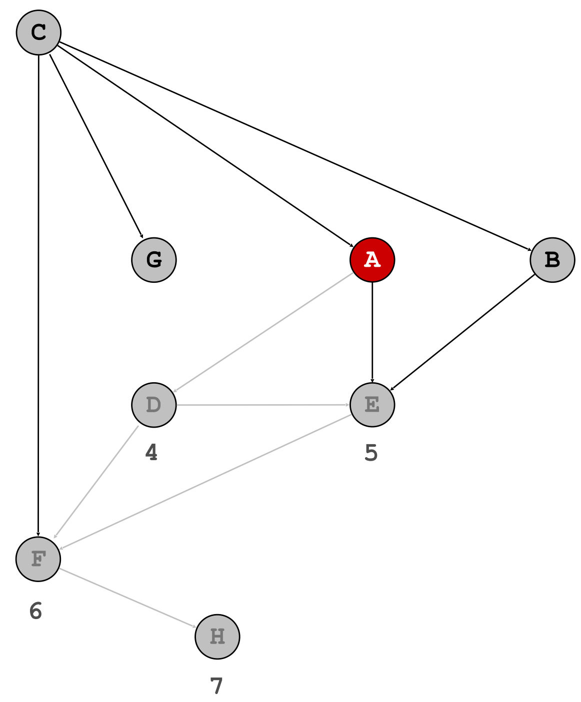

```
dfs(A)
  dfs(D)
```

Topological Sort: DFS



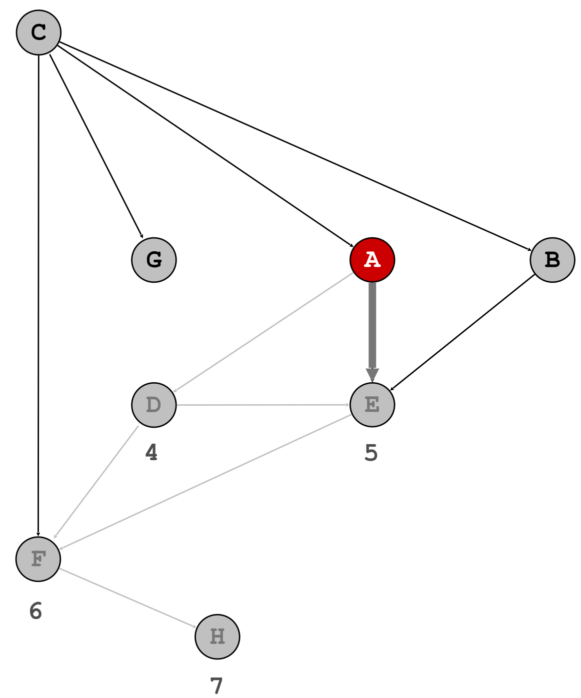

```
dfs(A)
  dfs(D)
```

Topological Sort: DFS



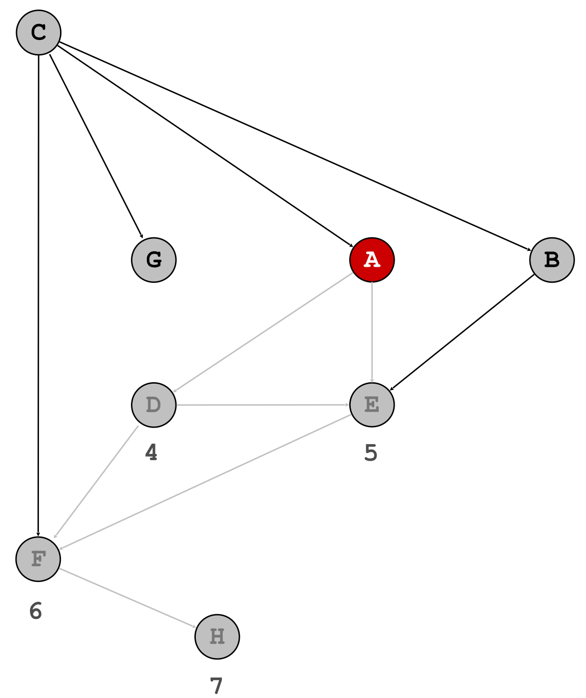
dfs(A)

Topological Sort: DFS



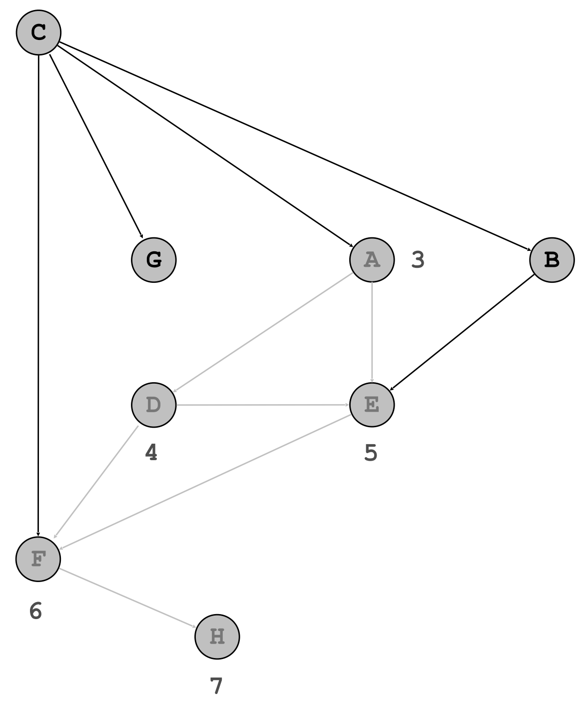
dfs(A)

Topological Sort: DFS

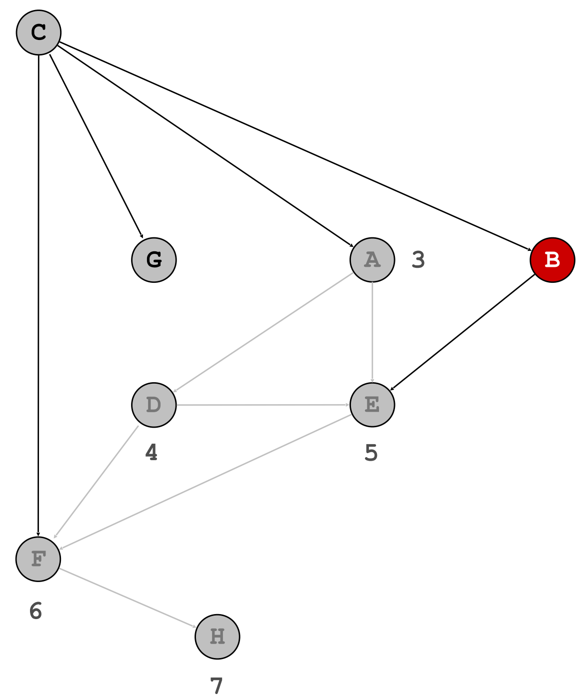


dfs(A)

Topological Sort: DFS

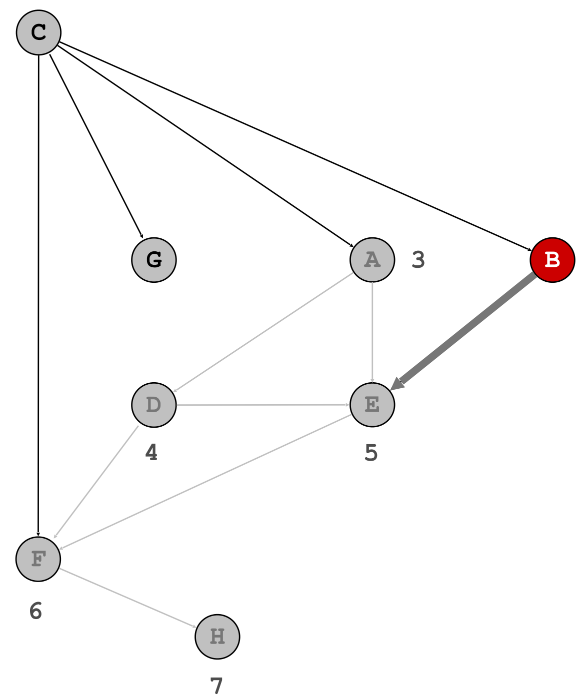


Topological Sort: DFS



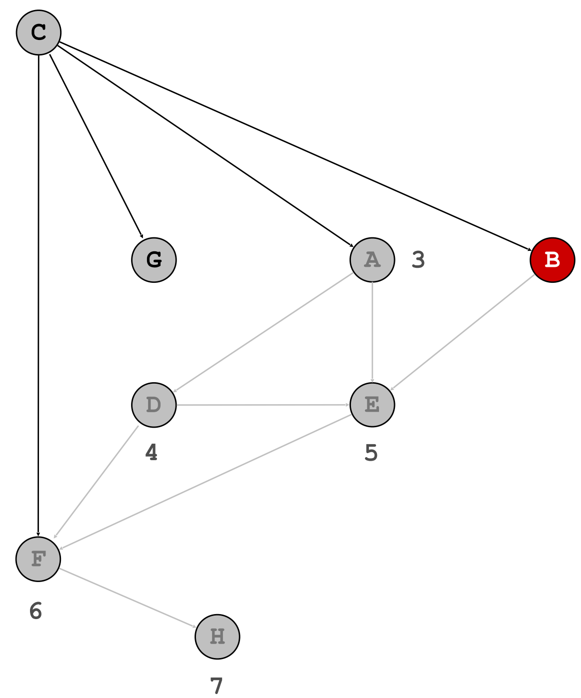
dfs(B)

Topological Sort: DFS



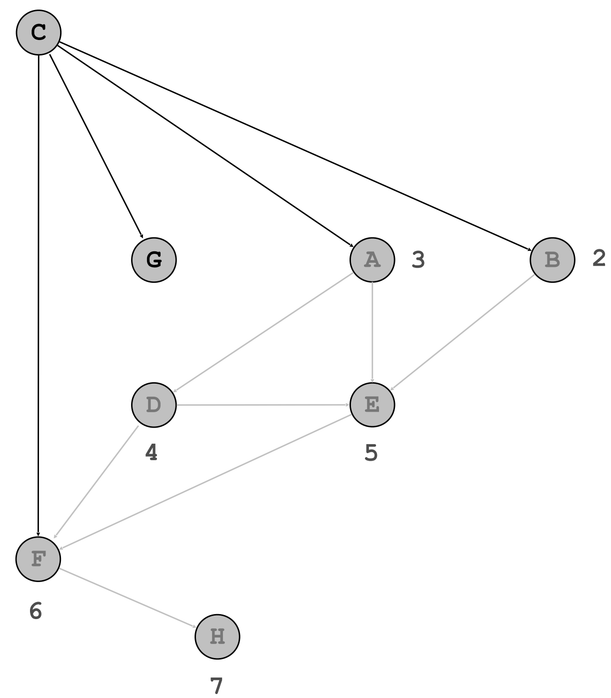
dfs(B)

Topological Sort: DFS

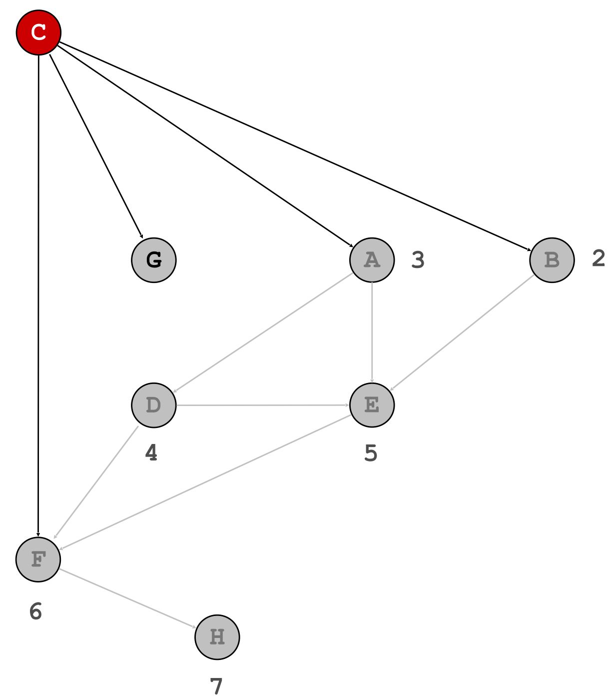


`dfs(B)`

Topological Sort: DFS

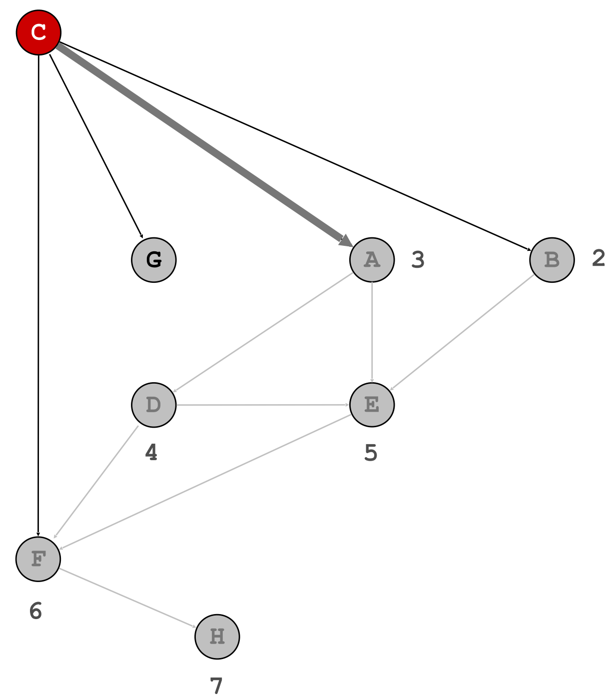


Topological Sort: DFS



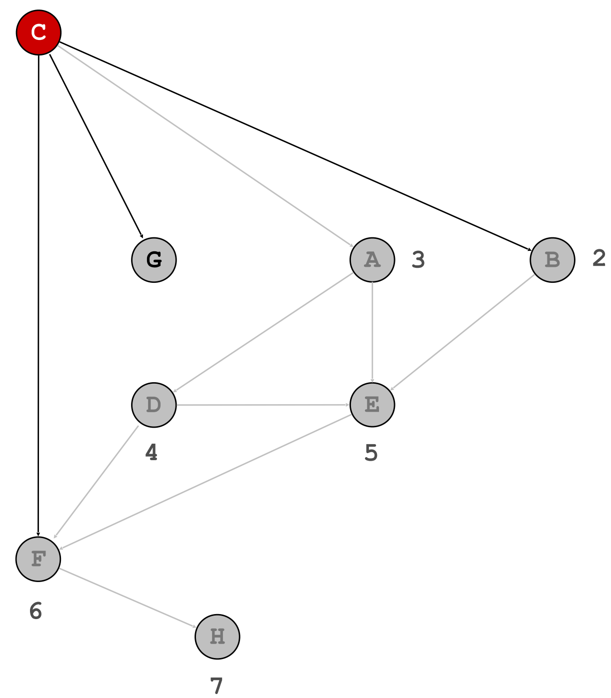
dfs(C)

Topological Sort: DFS



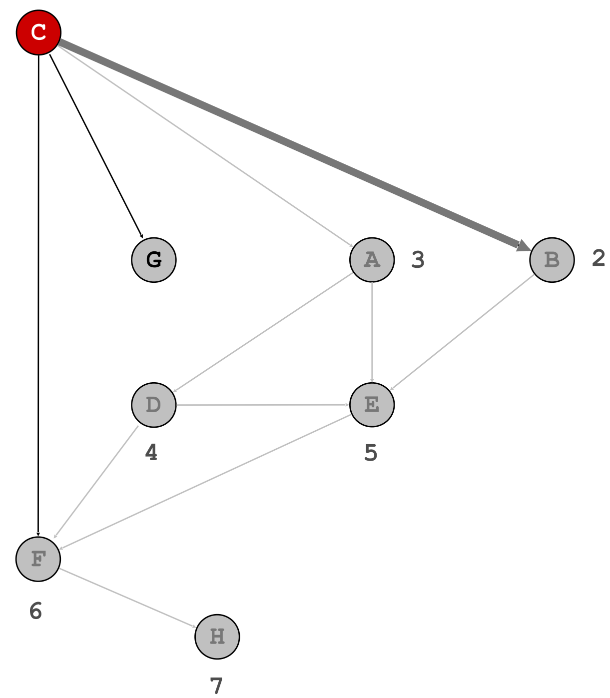
dfs(C)

Topological Sort: DFS



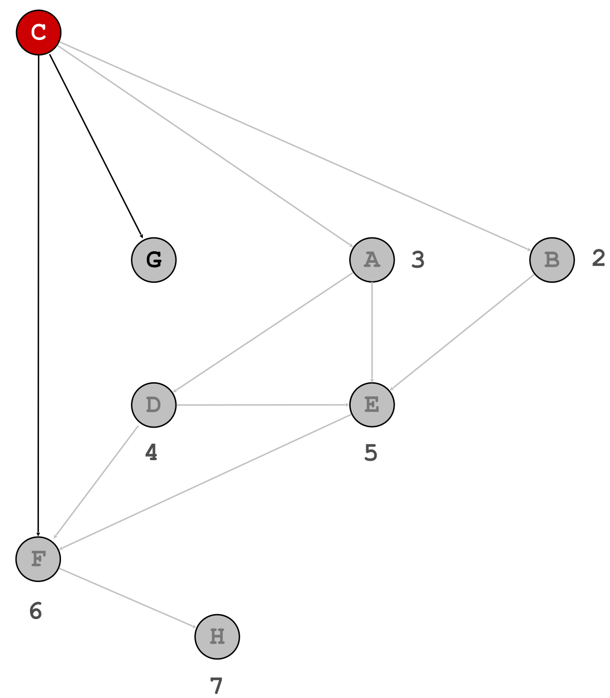

```
dfs(C)
```

Topological Sort: DFS



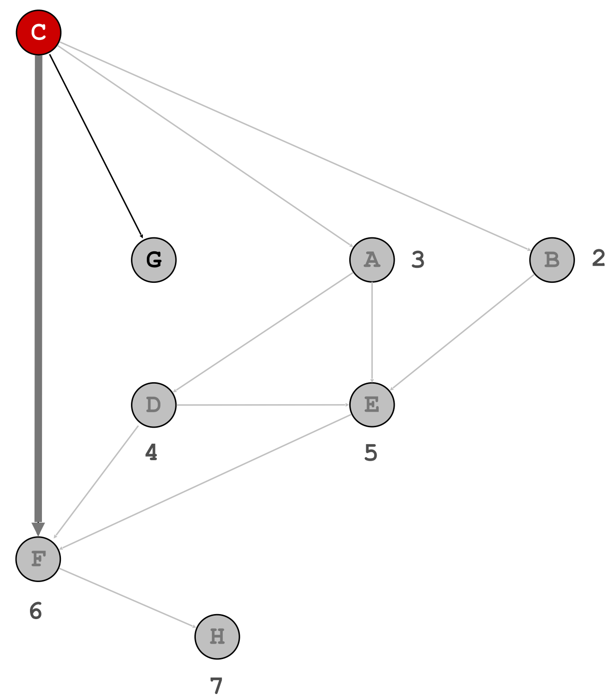
dfs(C)

Topological Sort: DFS



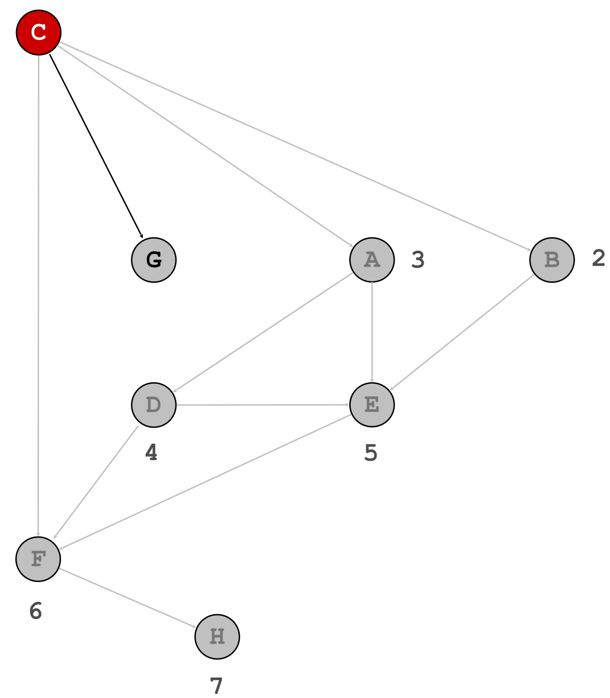
dfs(C)

Topological Sort: DFS



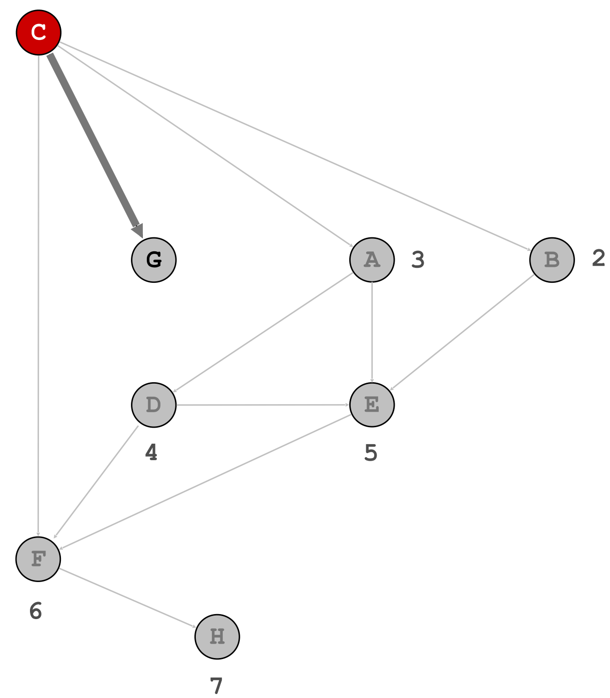
`dfs(C)`

Topological Sort: DFS



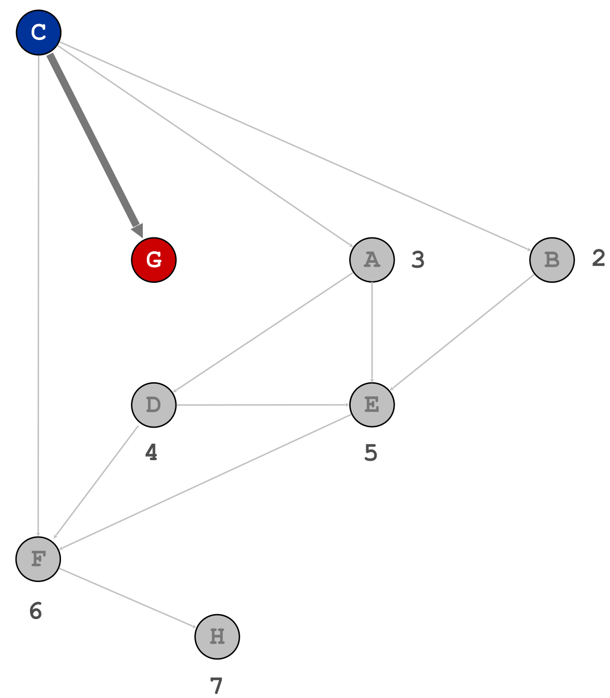
dfs(C)

Topological Sort: DFS



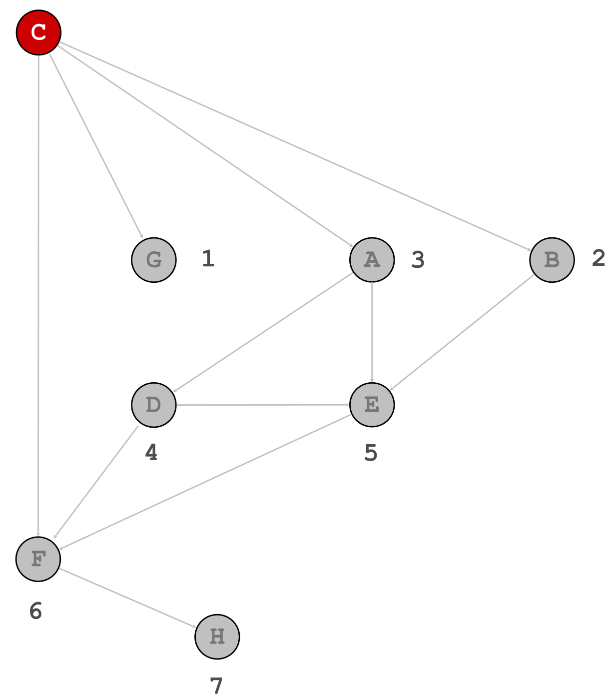
dfs(C)

Topological Sort: DFS



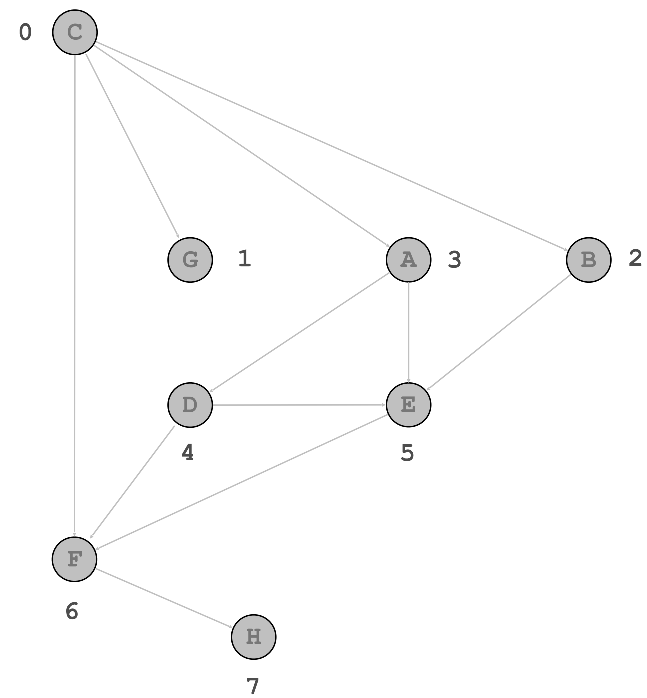

```
dfs(C)
  dfs(G)
```

Topological Sort: DFS

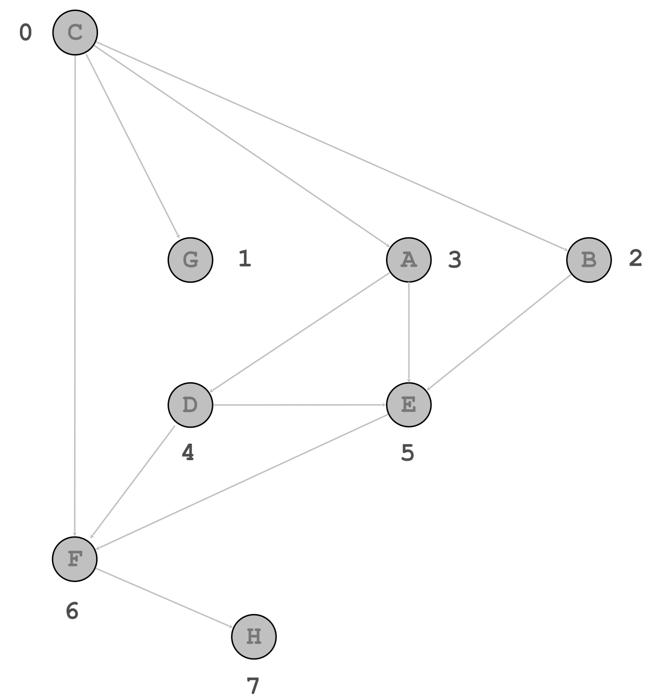


dfs(C)

Topological Sort: DFS



Topological Sort: DFS



Topological order: **C G B A D E F H**

Breadth First Topological Ordering

- In the breadth first topological ordering we first find a vertex that has no predecessor vertex and place it first in the topological ordering.
- We next find the vertex, say `v`, all of whose predecessors have been placed in the topological ordering and place `v` next in the topological ordering.
- To keep track of the number of vertices of a vertex we use the array `predCount`.
- Initially, `predCount[j]` is the number of predecessors of the vertex v_j.
- The queue used to guide the breadth first traversal is initialized to those vertices v_k such that `predCount[k]` is zero.

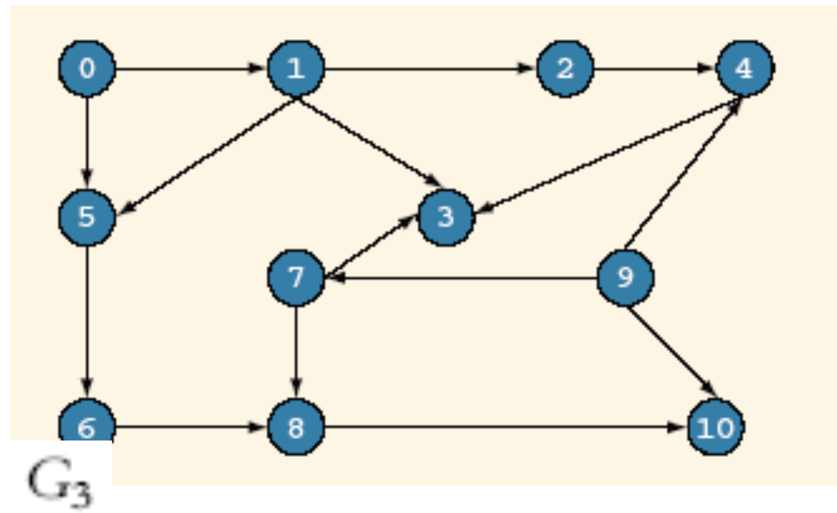
- The general algorithm is:

1. Create the array `predCount` and initialize it so that `predCount[i]` is the number of predecessors of the vertex v_i.
2. Initialize the queue, say, `queue`, to all those vertices v_k so that `predCount[k]` is zero. (Clearly, `queue` is not empty because the graph has no cycles.)
3. **`while`** the queue is not empty
   - 3.1. Remove the front element, `u`, of the queue.
   - 3.2. Put `u` in the next available position, say, `topologicalOrder[topIndex]`, and increment `topIndex`.
   - 3.3. For all the immediate successors `w` of `u`
     - 3.3.1. Decrement the predecessor count of `w` by `1`.
     - 3.3.2. **`if`** the predecessor count of `w` is zero, add `w` to `queue`.

More on Topological Sorting

Runs in O(n+m) time.

Useful starting point for many algorithms that involve acyclic graphs.



- The vertices of G_3 in breadth first topological ordering are

  `0 9 1 7 2 5 4 6 3 8 10`

- We illustrate the breadth first topological ordering of the graph G_3

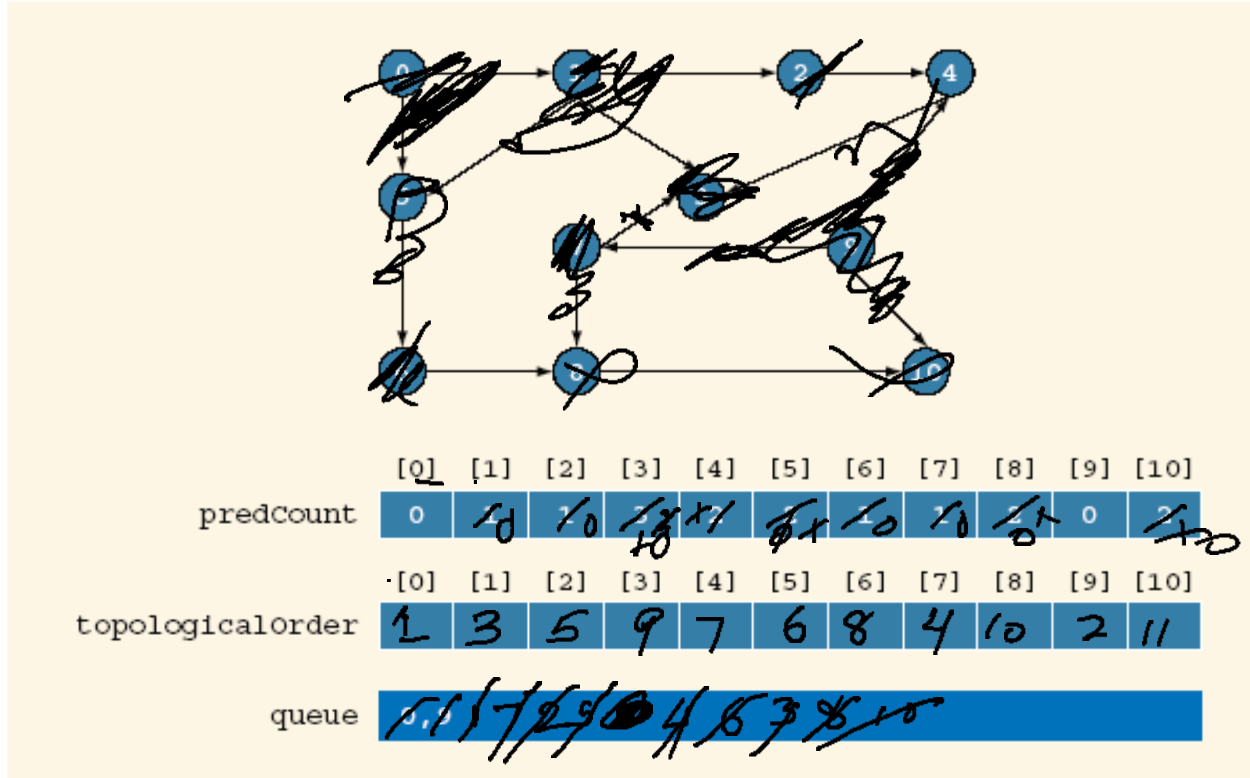


FIGURE T-1 Arrays `predCount`, `topologicalOrder`, and `queue` after Steps 1 and 2 execute

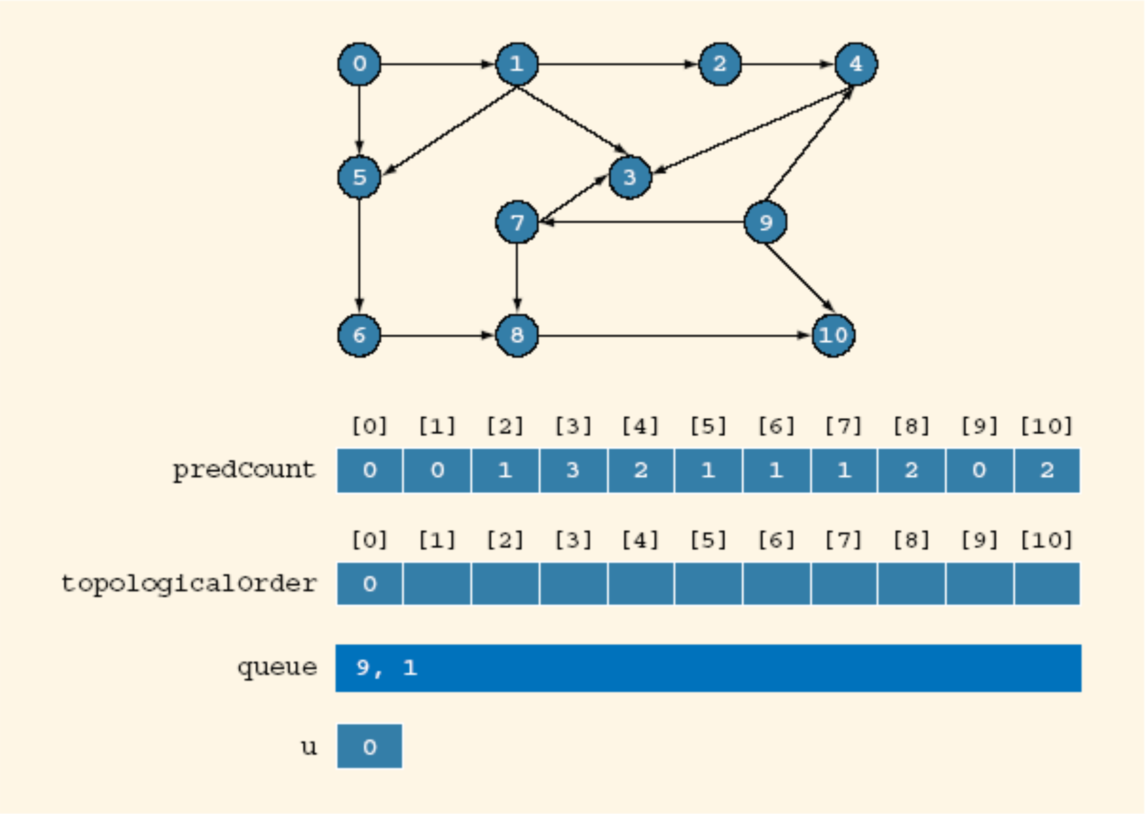
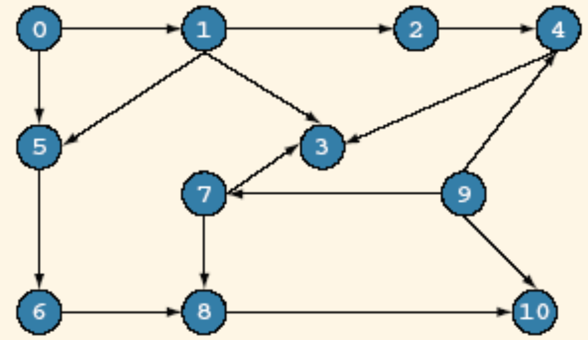


FIGURE T-2 Arrays `predCount`, `topologicalOrder`, and `queue` after the first iteration of Step 3



predCount

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
0	0	1	3	1	1	1	0	2	0	1

topologicalOrder

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
0	9									

queue: 1, 7

u: 9

FIGURE T-3 Arrays `predCount`, `topologicalOrder`, and `queue` after the second iteration of Step 3

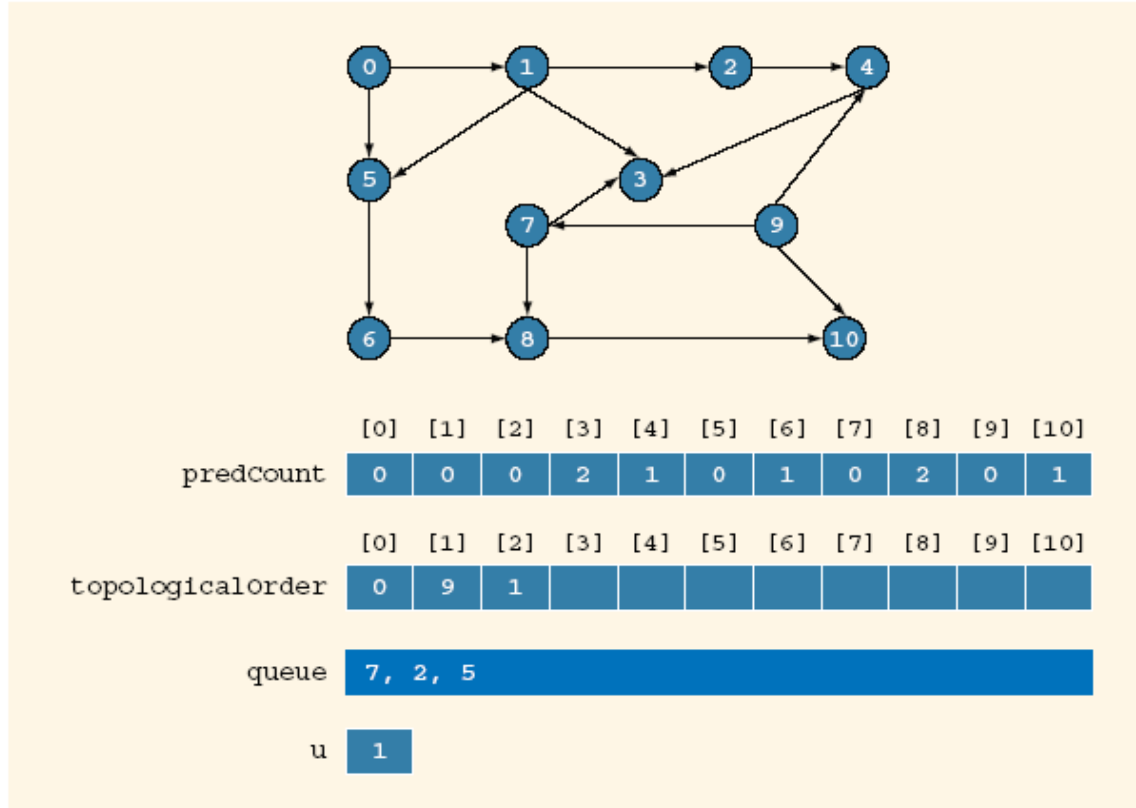


FIGURE T-4 Arrays `predCount`, `topologicalOrder`, and `queue` after the third iteration of Step 3

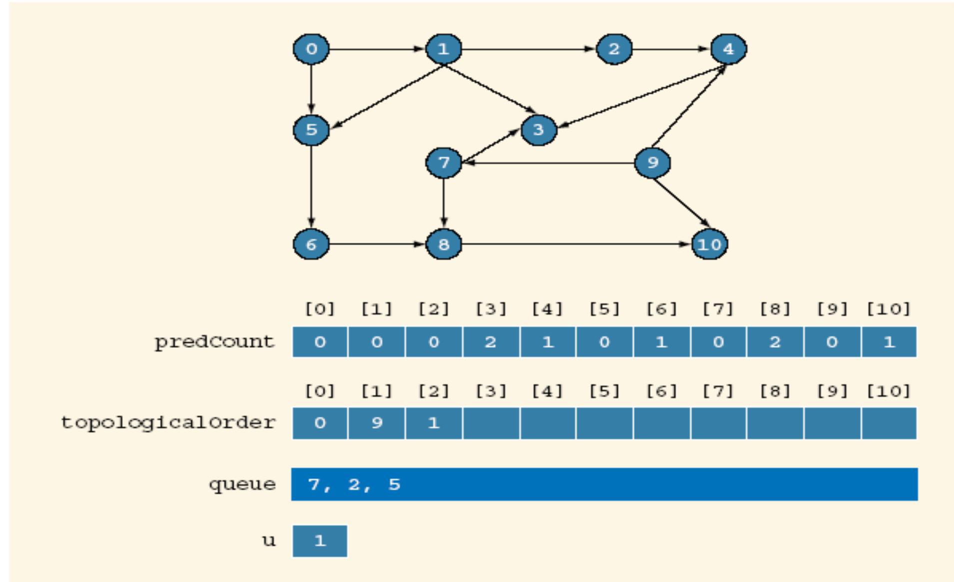


FIGURE T-4 Arrays predCount, topologicalOrder, and queue after the third iteration of Step 3

Pred Count	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	2	1	0	1	0	2	0	1

Topological order 0 9 1 7

queue 2 5 7

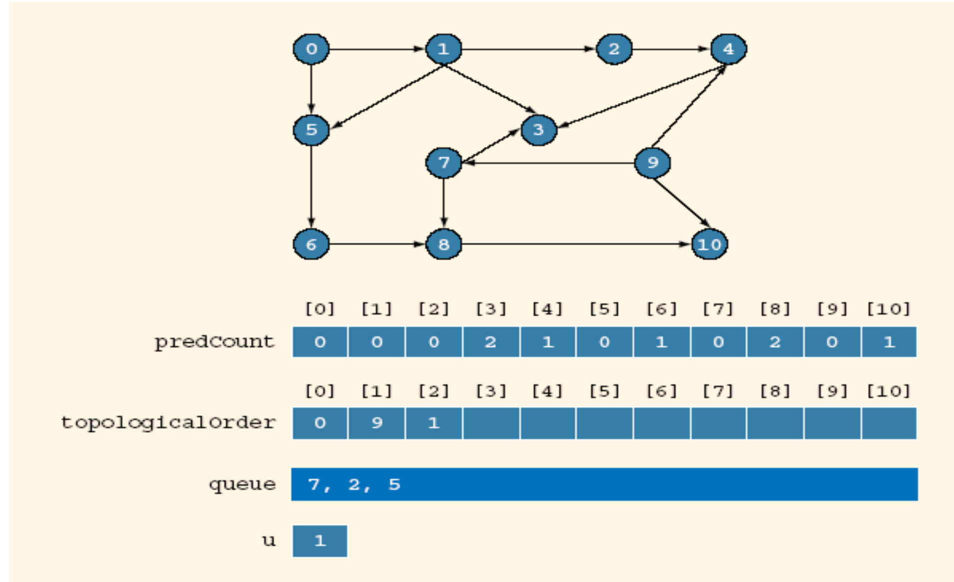
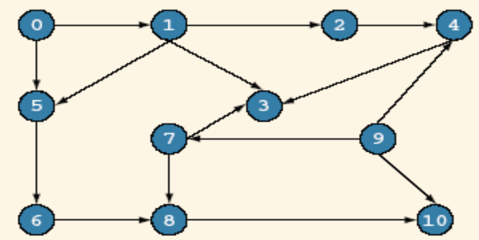


FIGURE T-4 Arrays predCount, topologicalOrder, and queue after the third iteration of Step 3

Pred Count	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	1	1	0	1	0	1	0	1

Topological order 0 9 1 7

queue 2 5 7



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
predCount	0	0	0	2	1	0	1	0	2	0	1

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
topologicalOrder	0	9	1								

queue 7, 2, 5

u 1

FIGURE T-4 Arrays `predCount`, `topologicalOrder`, and `queue` after the third iteration of Step 3

Pred Count	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	1	1	0	1	0	1	0	1

Topological order 0 9 1 7 2

queue 5 2

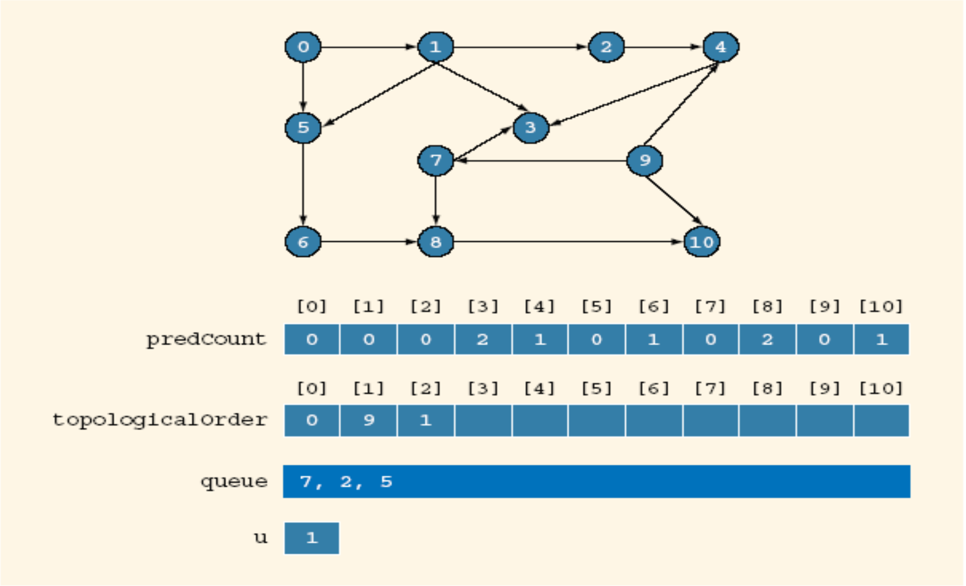


FIGURE T-4 Arrays `predCount`, `topologicalOrder`, and `queue` after the third iteration of Step 3

Pred Count 0 1 2 3 4 5 6 7 8 9 10
0 0 0 1 0 0 1 0 1 0 1

Topological order 0 9 1 7 2

queue 5 4 2

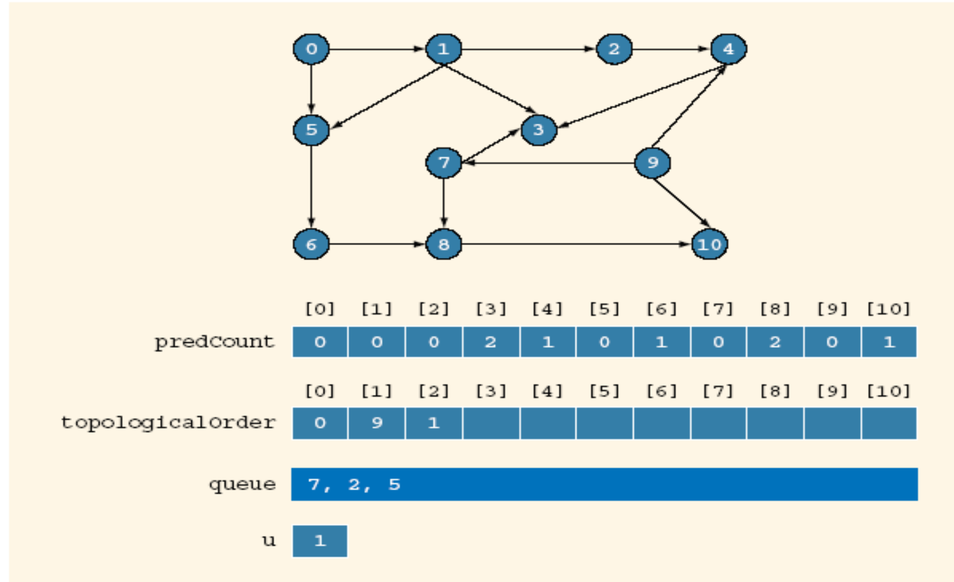


FIGURE T-4 Arrays predCount, topologicalOrder, and queue after the third iteration of Step 3

Pred Count	0 1 2 3 4 5 6 7 8 9 10
	0 0 0 1 0 0 1 0 1 0 1
Topological order	0 9 1 7 2
queue	4 5

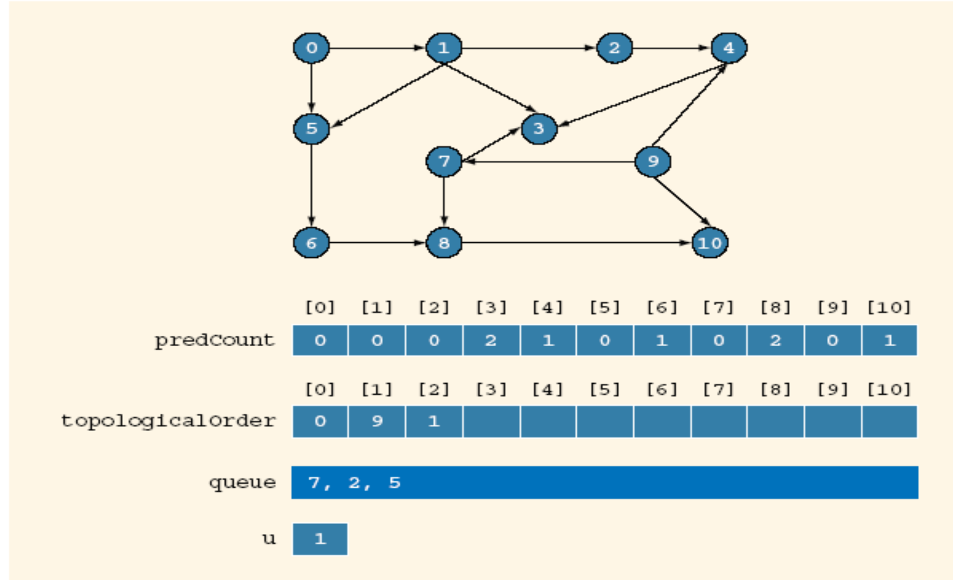


FIGURE T-4 Arrays predCount, topologicalOrder, and queue after the third iteration of Step 3

Pred Count	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	1	0	0	0	0	1	0	1

Topological order 0 9 1 7 2 5

queue 4 6 5

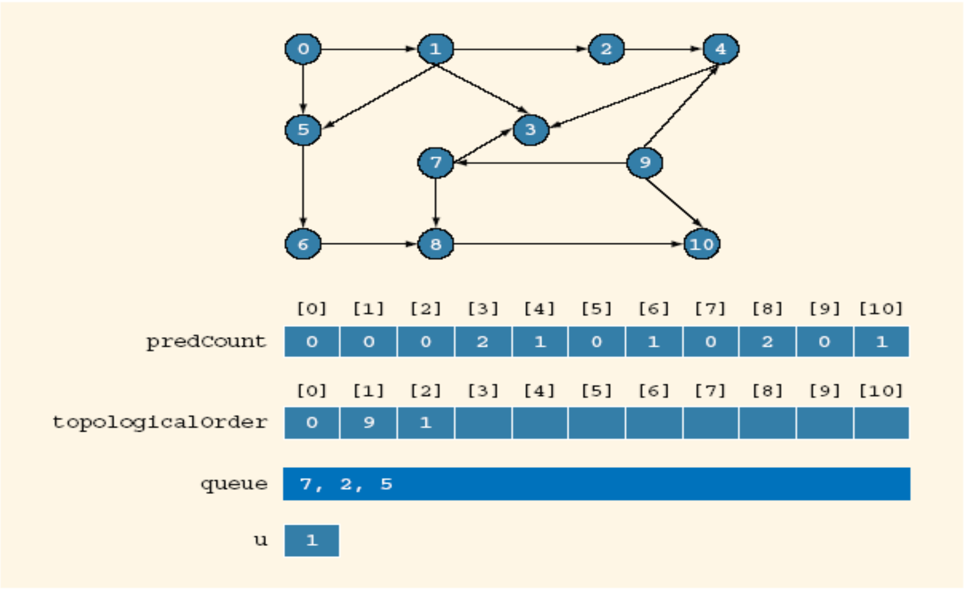


FIGURE T-4 Arrays `predCount`, `topologicalOrder`, and `queue` after the third iteration of Step 3

Pred Count	0 1 2 3 4 5 6 7 8 9 10
	0 0 0 1 0 0 0 0 1 0 1
Topological order	0 9 1 7 2 5
queue	6 4

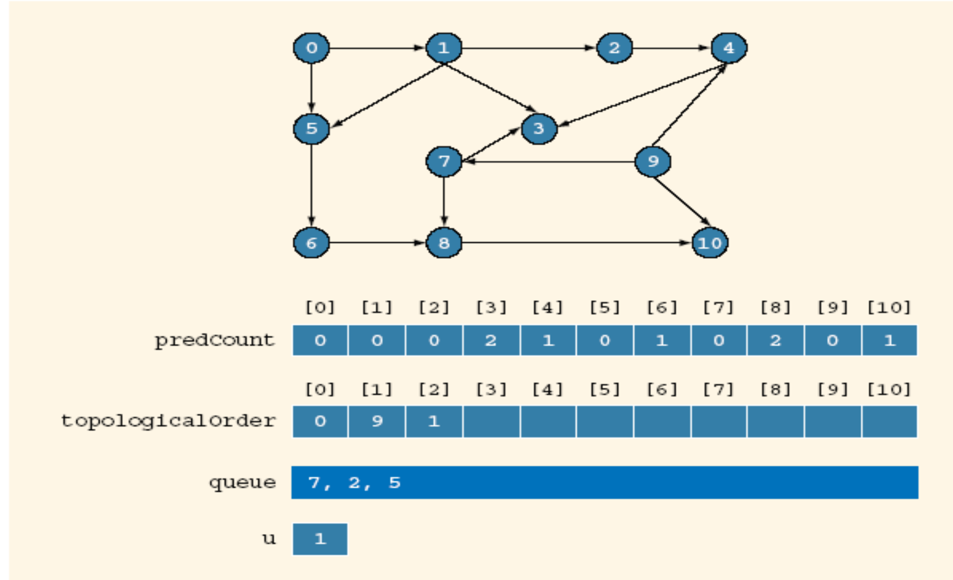


FIGURE T-4 Arrays predCount, topologicalOrder, and queue after the third iteration of Step 3

Pred Count 0 1 2 3 4 5 6 7 8 9 10
0 0 0 0 0 0 0 0 1 0 1

Topological order 0 9 1 7 2 5 4

queue 6 3 4

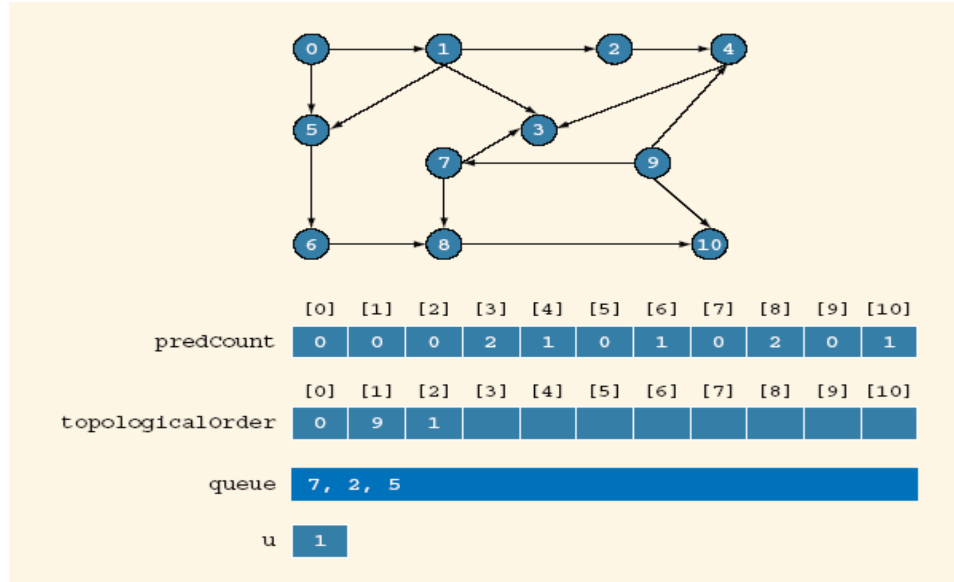


FIGURE T-4 Arrays predCount, topologicalOrder, and queue after the third iteration of Step 3

Pred Count 0 1 2 3 4 5 6 7 8 9 10
0 0 0 0 0 0 0 0 1 0 1

Topological order 0 9 1 7 2 5 4 6

queue 3 6

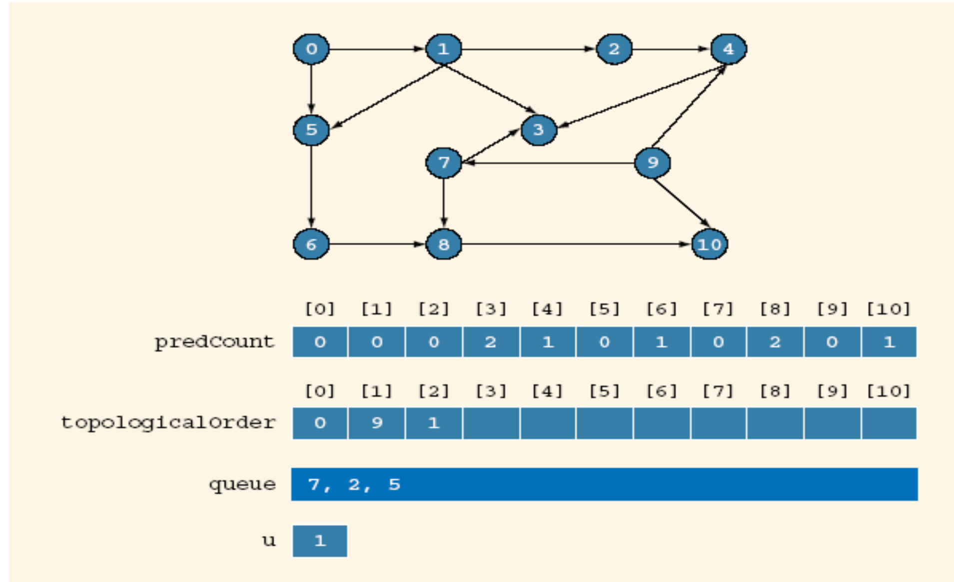


FIGURE T-4 Arrays `predCount`, `topologicalOrder`, and `queue` after the third iteration of Step 3

Pred Count	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	1

Topological order 0 9 1 7 2 5 4 6

queue 3 8 6

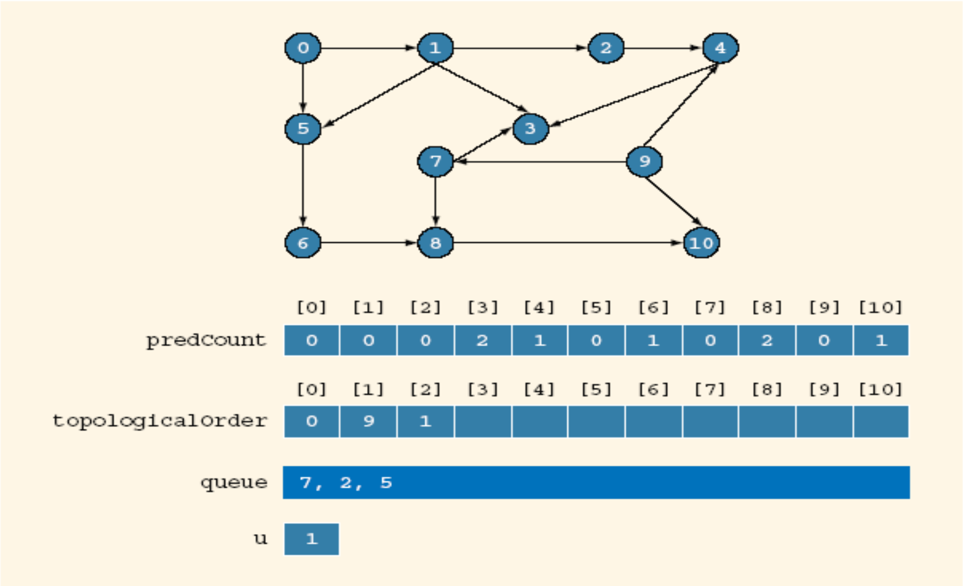


FIGURE T-4 Arrays predCount, topologicalOrder, and queue after the third iteration of Step 3

Pred Count	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	1

Topological order 0 9 1 7 2 5 4 6 3

queue 8 3

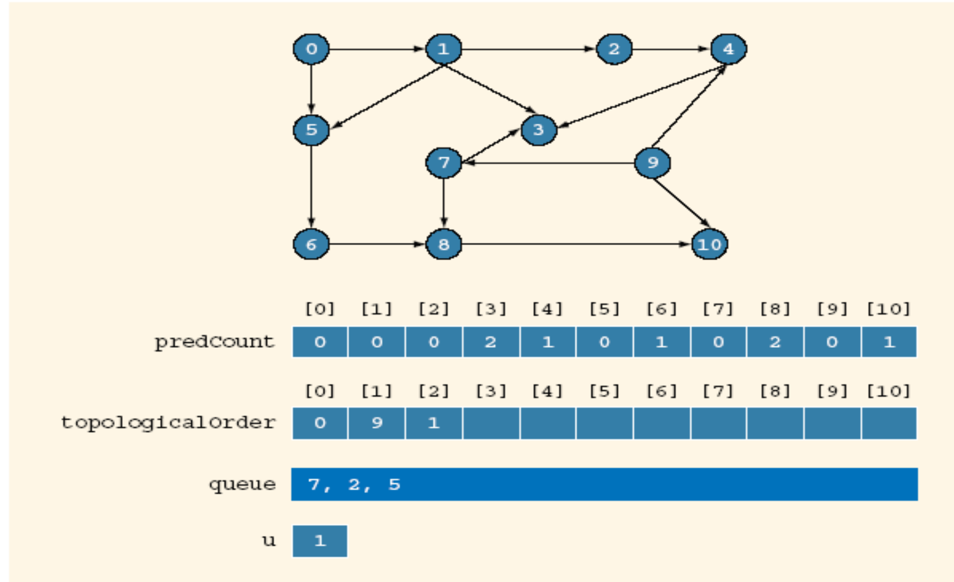


FIGURE T-4 Arrays `predCount`, `topologicalOrder`, and `queue` after the third iteration of Step 3

Pred Count	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	1

Topological order 0 9 1 7 2 5 4 6 3 8

queue 8

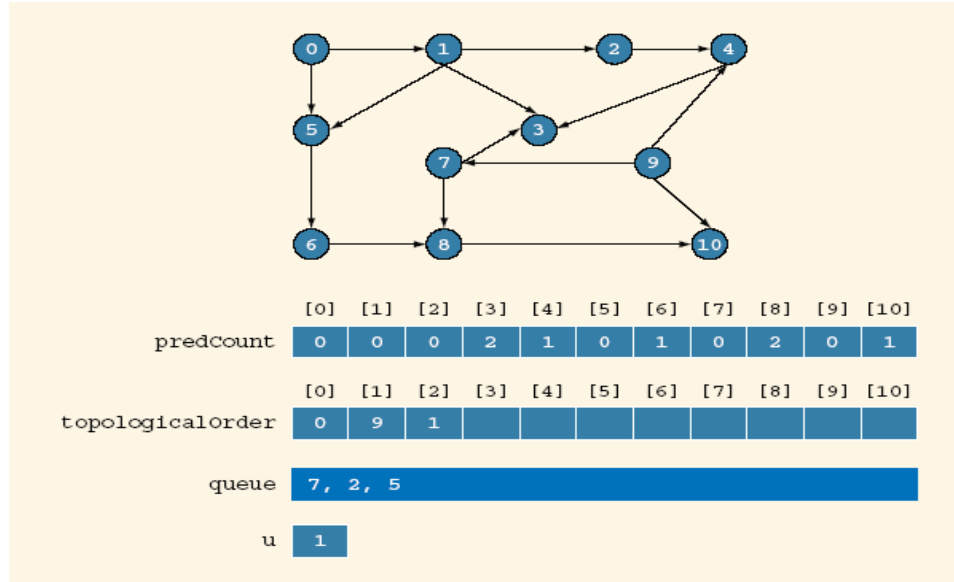


FIGURE T-4 Arrays predCount, topologicalOrder, and queue after the third iteration of Step 3

Pred Count	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	0

Topological order 0 9 1 7 2 5 4 6 3 8

queue 10 8

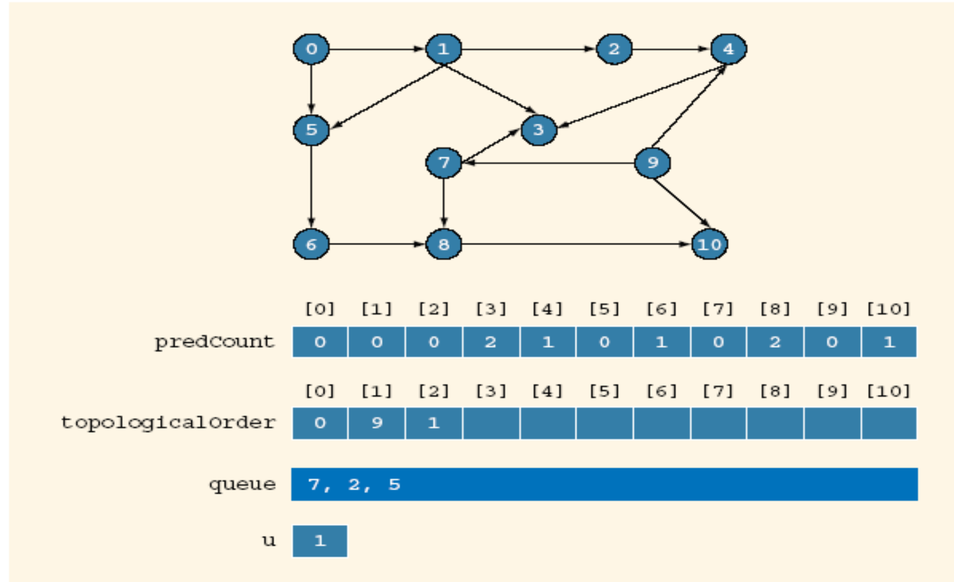
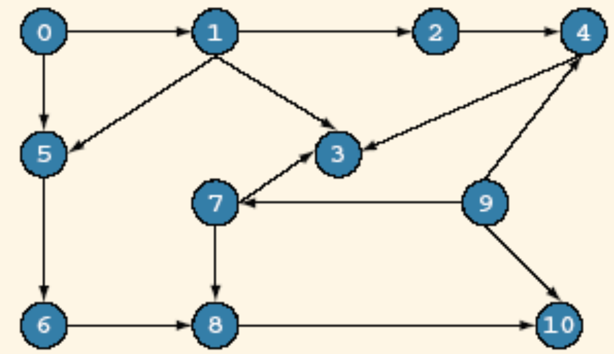


FIGURE T-4 Arrays `predCount`, `topologicalOrder`, and `queue` after the third iteration of Step 3

Pred Count	0 1 2 3 4 5 6 7 8 9 10
	0 0 0 0 0 0 0 0 0 0 0
Topological order	0 9 1 7 2 5 4 6 3 8 10
queue	10



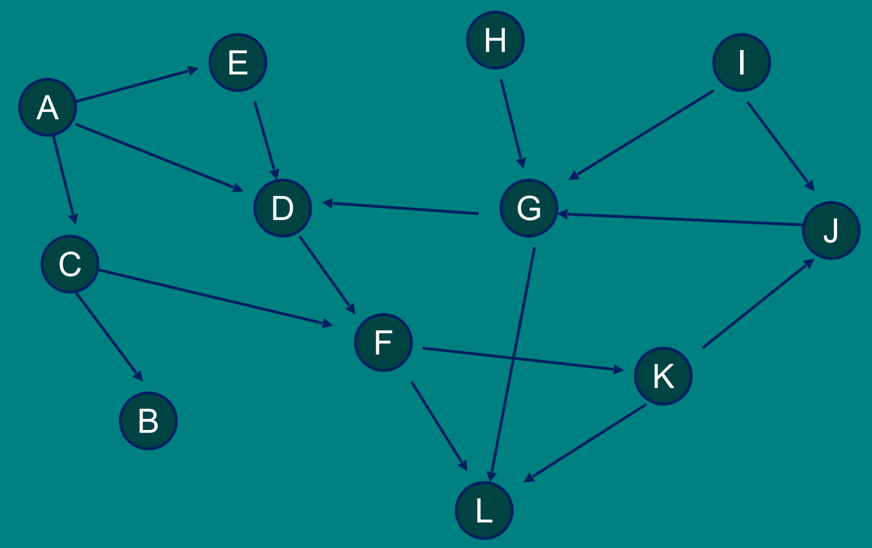
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
predCount	0	0	0	0	0	0	0	0	0	0	0

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
topologicalOrder	0	9	1	7	2	5	4	6	3	8	10

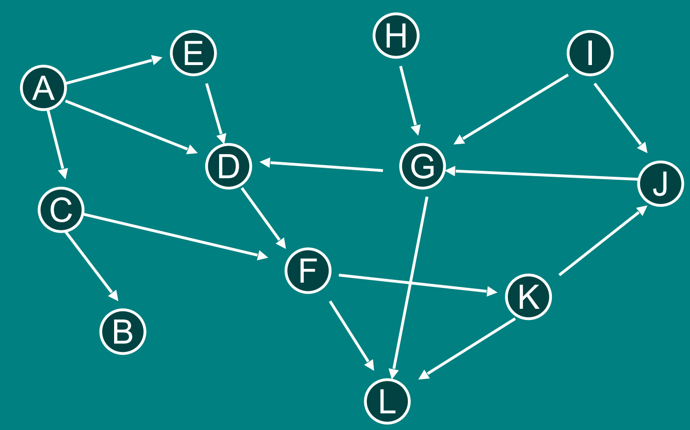
queue	

FIGURE T-5 Arrays `predCount`, `topologicalOrder`, and `queue` after Step 3 executes eight more times

Find a topological order for the following graph



Find a topological order for the following graph



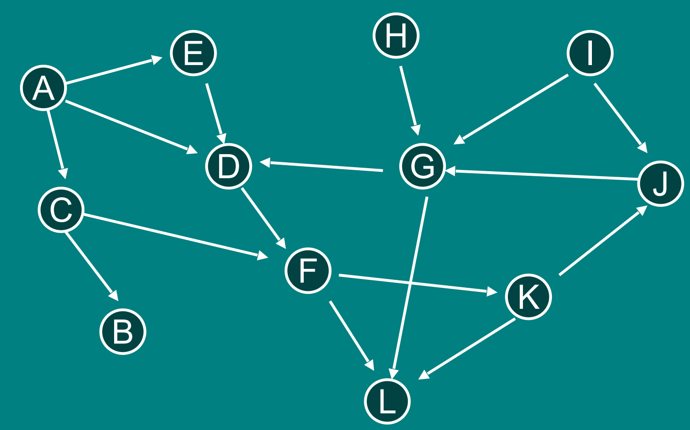
Pred Count

A	B	C	D	E	F	G	H	I	J	K	L
0	1	1	3	1	2	3	0	0	2	1	3

Topological order

queue A H I

Find a topological order for the following graph



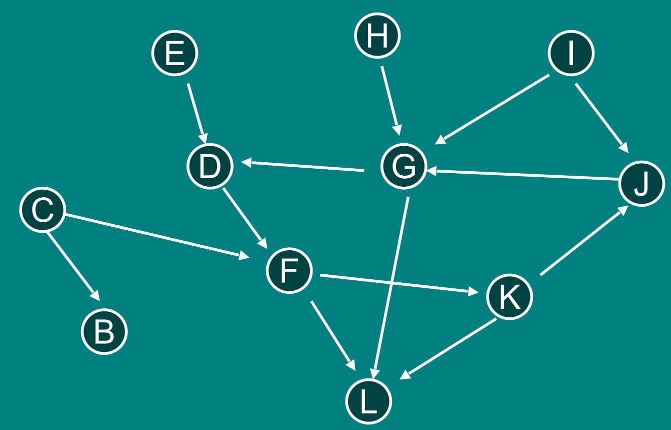
Pred Count

A	B	C	D	E	F	G	H	I	J	K	L
0	1	1	3	1	2	3	0	0	2	1	3

Topological order A

queue H I A

Find a topological order for the following graph



Pred Count

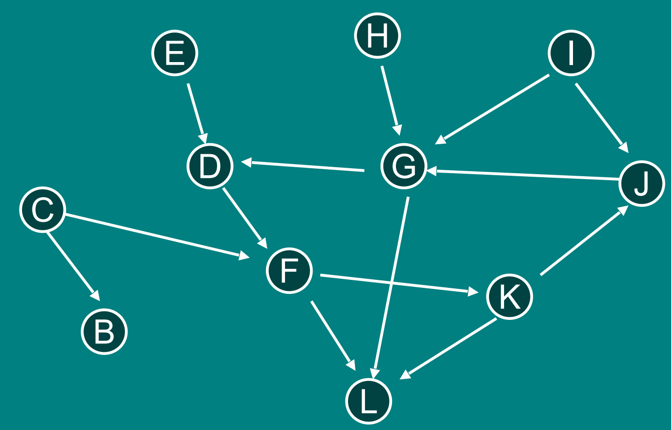
A	B	C	D	E	F	G	H	I	J	K	L
0	1	0	2	0	2	3	0	0	2	1	3

Topological order A

queue H I C E

A

Find a topological order for the following graph



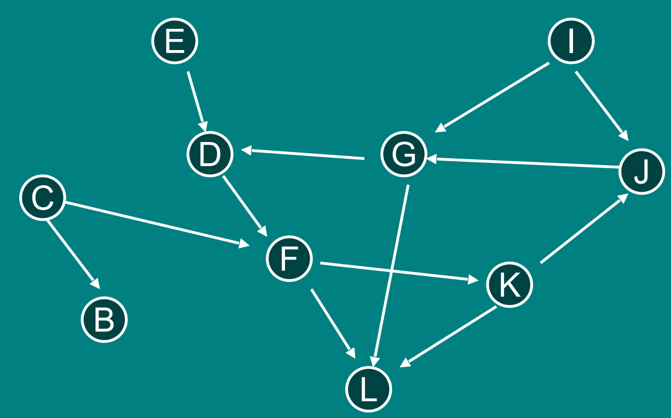
	A	B	C	D	E	F	G	H	I	J	K	L
Pred Count	0	1	0	2	0	2	3	0	0	2	1	3

Topological order A H

queue I C E

H

Find a topological order for the following graph



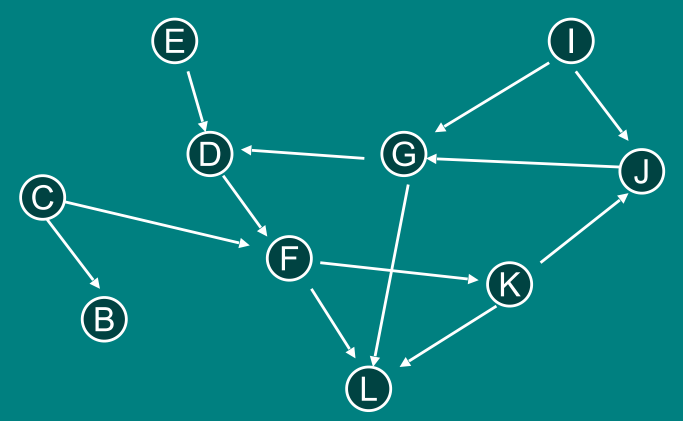
	A	B	C	D	E	F	G	H	I	J	K	L
Pred Count	0	1	0	2	0	2	2	0	0	2	1	3

Topological order A H

queue I C E

H

Find a topological order for the following graph



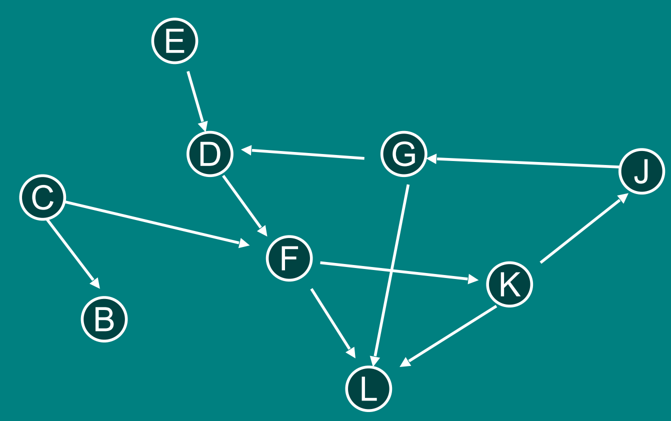
	A	B	C	D	E	F	G	H	I	J	K	L
Pred Count	0	1	0	2	0	2	2	0	0	2	1	3

Topological order A H I

queue C E

I

Find a topological order for the following graph



Pred Count

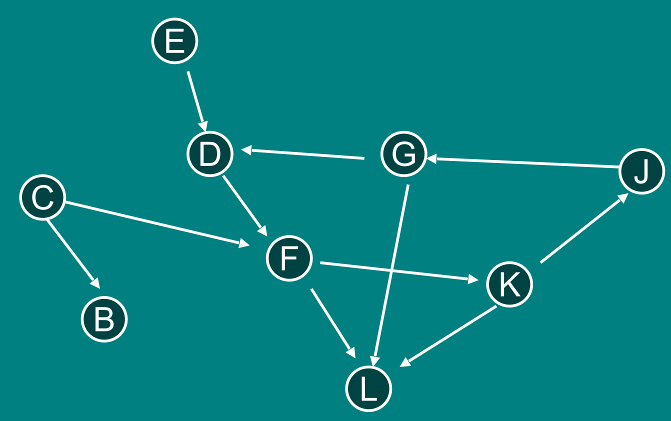
A	B	C	D	E	F	G	H	I	J	K	L
0	1	0	2	0	2	1	0	0	1	1	3

Topological order A H I

queue C E

I

Find a topological order for the following graph



Pred Count

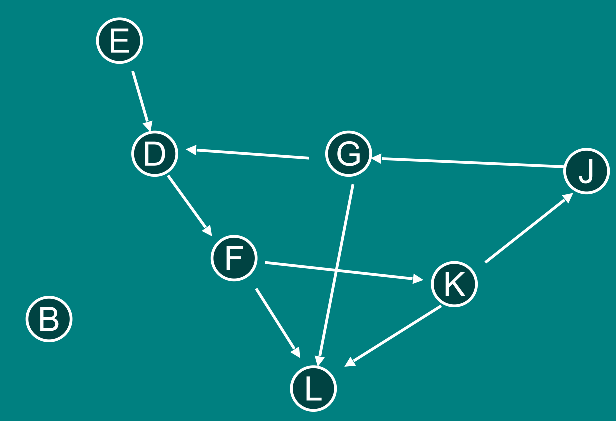
A	B	C	D	E	F	G	H	I	J	K	L
0	1	0	2	0	2	1	0	0	1	1	3

Topological order A H I C

queue E

C

Find a topological order for the following graph



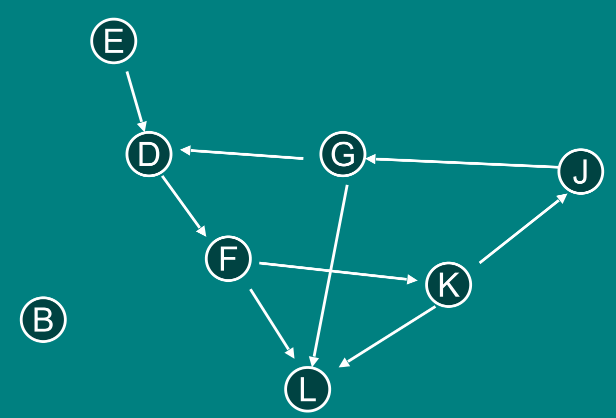
Pred Count	A	B	C	D	E	F	G	H	I	J	K	L
	0	0	0	2	0	1	1	0	0	1	1	3

Topological order A H I C

queue E B

C

Find a topological order for the following graph



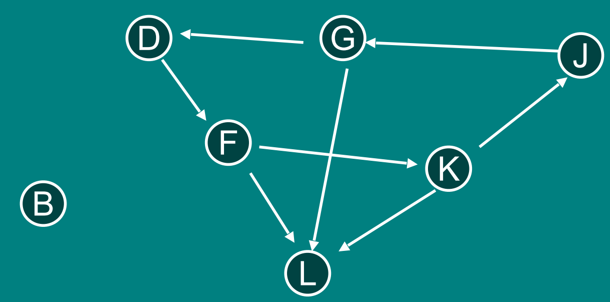
Pred Count

A	B	C	D	E	F	G	H	I	J	K	L
0	0	0	2	0	1	1	0	0	1	1	3

Topological order A H I C E

queue B E

Find a topological order for the following graph

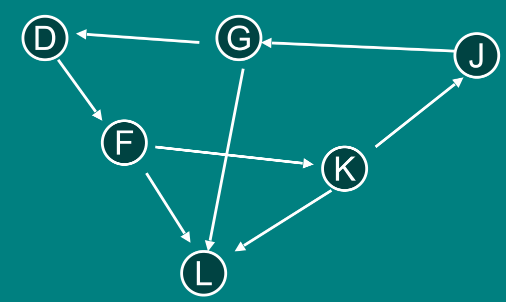


	A	B	C	D	E	F	G	H	I	J	K	L
Pred Count	0	0	0	1	0	1	1	0	0	1	1	3

Topological order A H I C E

queue B E

Find a topological order for the following graph



Pred Count	A	B	C	D	E	F	G	H	I	J	K	L
	0	0	0	1	0	1	1	0	0	1	1	3

Topological order A H I C E B

queue B