Graph Applications: Topological Sort

CS 112 Spring 2020 – Apr 7

Precedence Graph

A precedence graph is a Directed Acyclic Graph (DAG), i.e. a directed graph that does not have any cycles

$$\begin{tikzcd}
 & B \arrow[dd] \\
A \arrow[ur] \arrow[dr] & \\
 & C
\end{tikzcd}$$

acyclic

$$\begin{tikzcd}
 & B \arrow[dd] \\
A \arrow[ur] & \\
 & C \arrow[ul]
\end{tikzcd}$$

has cycle

$$\begin{tikzcd}
 & B \arrow[dd, shift right] \\
A \arrow[ur] \arrow[dr] & \\
 & C \arrow[uu, shift right]
\end{tikzcd}$$

has cycle

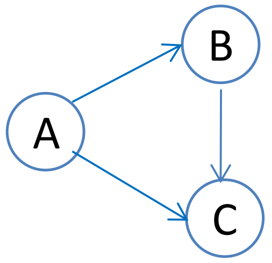
An edge x → y means that x *precedes* y.

Precedence graphs are used in practice for task scheduling in various domains such as project management, computational graphs, etc.

Each vertex is a task, and a precedence x → y means that task y can only start after task x is completed.

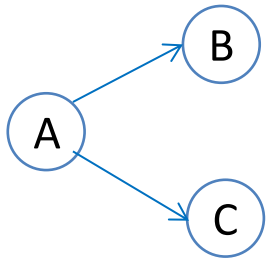
Precedence Graph

When there are a bunch of tasks to be done, and there are precedence constraints among them, we need to know in what sequence to do the tasks without violating any of the precedence .



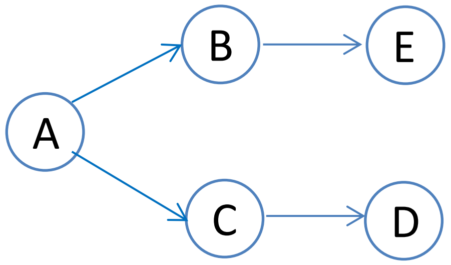
Possible sequences:

- A, B, C



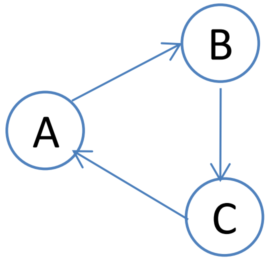
Possible sequences:

- A, B, C
- A, C, B



Possible sequences:

- A, B, C, D, E
- A, B, C, E, D
- A, B, E, C, D
- A, C, B, D, E
- A, C, B, E, D
- A, C, D, B, E

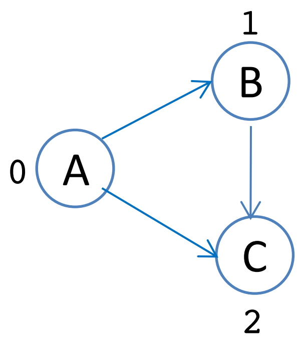


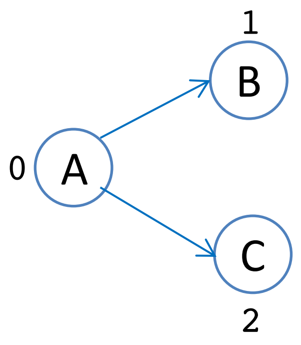
has cycle

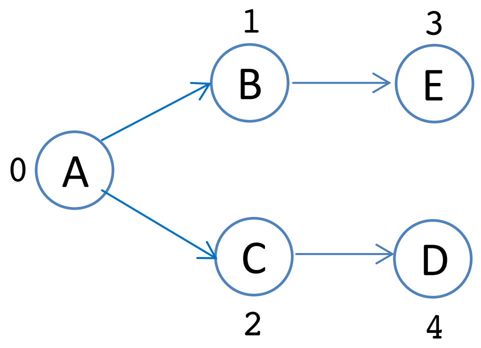
Because this graph has a cycle, there is no way to sequence the vertices without violating a precedence constraint.

Precedence Graph

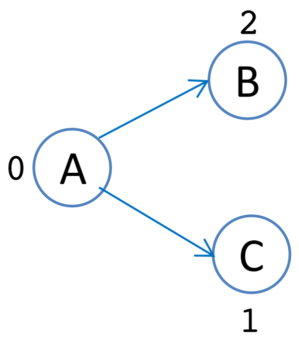
To sequence the vertices, we need to assign numbers to vertices – the number is the position of the vertex in the sequence

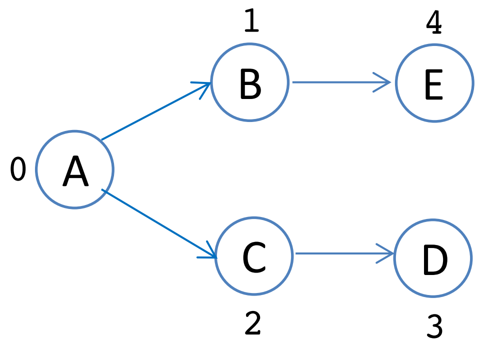






The process of assigning sequence numbers to vertices is called topological sorting, and the sequence numbers are called topological numbers.





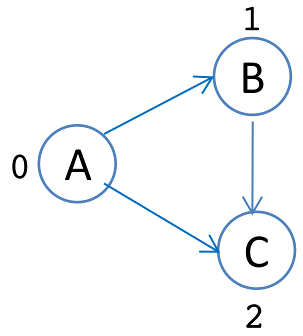
Plus other possibilities

Topological Sorting using DFS

DFS Topsort

In order to assign topological numbers, we need to navigate the topology of the graph to detect the precedence constraints, so that if x → y is an edge (a precedence that constrains x to happen before y), the topological number for x is less than that for y.

Could we use DFS to do the topological numbering, assigning numbers from 0..n-1 as we visit the vertices?



Start at A
Visit A (0)
Visit B (1)
Visit C(2)

✓

Start at A
Visit A (0)
Visit C (1)
Back up to A
Visit B (2)

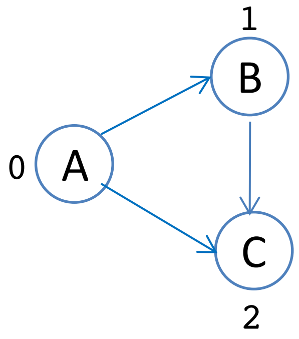
✗

Assigning numbers in ascending order does not work. We wouldn't know to pick B instead of C to visit next, because the next vertex is chosen arbitrarily, depending on storage in adjacency linked lists. And if C is the first neighbor, then we're stuck. (Also, what if we start at C first?)

DFS Topsort

The good news is that we can still use DFS, with a clever trick:

- Number the vertices NOT when you visit, but when it is a dead end (options exhausted, about to back up), and
- Assign numbers in descending order: n-1 for the first number, n-2 for the second number, etc.



Start at A
Visit A
Visit B
Visit C
Dead end
C: 2
Back up to B
Dead end
B: 1
Back up to A
Dead end
A: 0

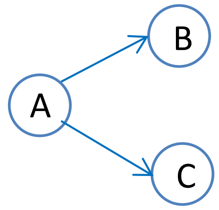
Start at A
Visit A
Visit C
Dead end
C: 2
Back up to A
Visit B
Dead end
B: 1
Back up to A
Dead end
A: 0

Start at C
Visit C
Dead end
C: 2
Restart at B
Visit B
Dead end
B: 1
Restart at A
Visit A
Dead end
A: 0

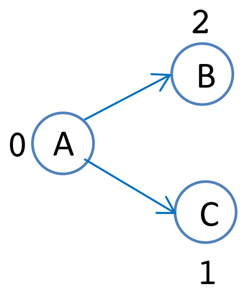
DFS Topsort

The good news is that we can still use DFS, with a clever trick:

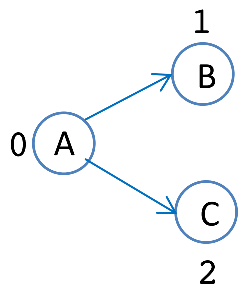
- Number the vertices NOT when you visit, but when it is a dead end (options exhausted, about to back up), and
- Assign numbers in descending order: n-1 for the first number, n-2 for the second number, etc.



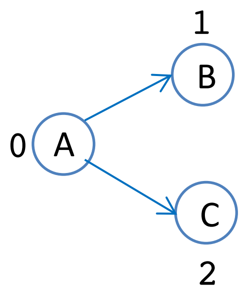
Start at A
Visit A
Visit B
Dead end
B: 2
Back up to A
Visit C
Dead end
C: 1
Back up to A
Dead end
A: 0



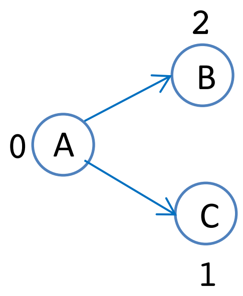
Start at A
Visit A
Visit C
Dead end
C: 2
Back up to A
Visit B
Dead end
B: 1
Back up to A
Dead end
A: 0



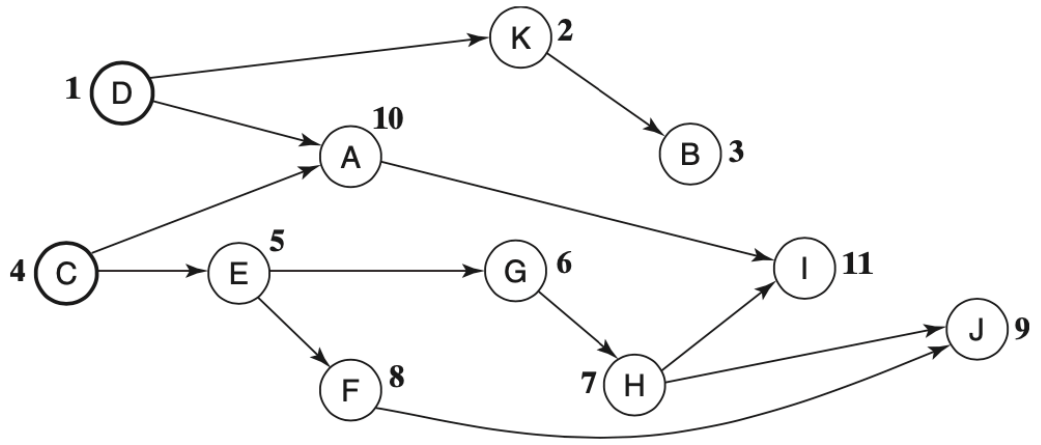
Start at C
Visit C
Dead end
C: 2
Restart at B
Visit B
Dead end
B: 1
Restart at A
Visit A
Dead end
A: 0



Start at B
Visit B
Dead end
B: 2
Restart at A
Visit A
Visit C
Dead end
C: 1
Back up to A
Dead end
A: 0



DFS Topsort



Vertices are numbered 1..11, but actually they should be 0..10

Start at A
Visit A
Visit I
Dead end
I: 11
Back up to A
Dead end
A: 10
Restart at C
Visit C
Visit E
Visit F
Visit J
Dead end
J: 9
Back up to F
Dead end
F: 8
Back up to E
Visit G
Visit H
Dead end
H: 7
Back up to G
Dead end
G: 6
Back up to E
Dead end
E: 5
Back up to C
Dead end
C: 4
Restart at D
…

DFS Topsort Implementation

```
// recursive dfs
private void dfs(int v, boolean[] visited) {
     visited[v] = true;
     for (Neighbor e=adjLists[v].adjList; e != null; e=e.next) {
          if (!visited[e.vertexNum]) {
               dfs(e.vertexNum, visited);
          }
     }
}
```

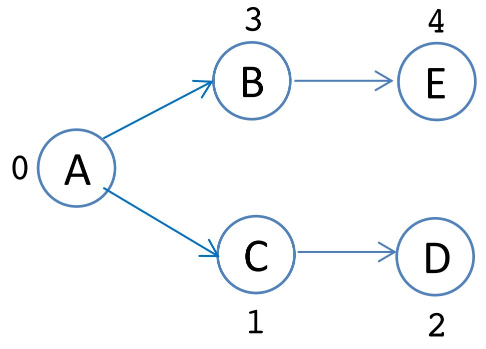
Modification 1: Need to accept as parameters the current highest topological number, and an array in which topological sequence can be written in

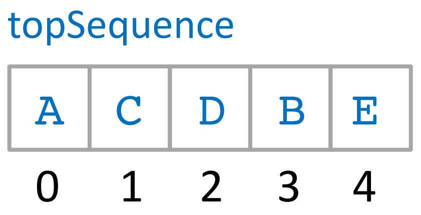
```
// recursive dfs
private void dfs(int v, boolean[] visited, int num, int[] topSequence) {
     visited[v] = true;
     for (Neighbor e=adjLists[v].adjList; e != null; e=e.next) {
          if (!visited[e.vertexNum]) {
               dfs(e.vertexNum, visited, num, topSequence);
          }
     }
}
```

DFS Topsort Implementation

Modification 2: Need to assign topological number when at a dead end (just about to back up), and decrement the topological number in preparation for assigning to the next vertex

```
// recursive dfs
private void dfs(int v, boolean[] visited, int num, int[] topSequence) {
     visited[v] = true;
     for (Neighbor e=adjLists[v].adjList; e != null; e=e.next) {
          if (!visited[e.vertexNum]) {
               dfs(e.vertexNum, visited, num, topSequence);
          }
     }
     topSequence[num] = v; // slot vertex in its sequence position
     num--;
}
```



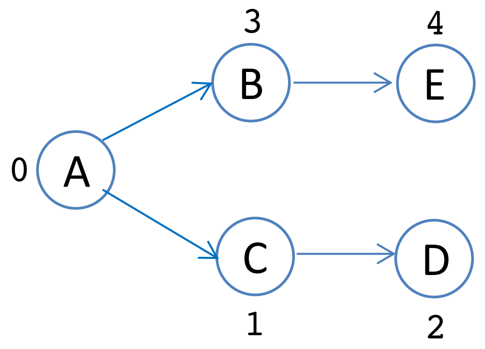

topSequence

A	C	D	B	E
0	1	2	3	4

DFS Topsort Implementation

Modification 3: But num is a local variable, so when it is decremented, the new value is not going to be carried through to subsequent calls – so we need to return it from the method:

```
// recursive dfs
private int dfs(int v, boolean[] visited, int num, int[] topSequence) {
     visited[v] = true;
     for (Neighbor e=adjLists[v].adjList; e != null; e=e.next) {
          if (!visited[e.vertexNum]) {
               num = dfs(e.vertexNum, visited, num, topSequence);
          }
     }
     topSequence[num] = v; // slot vertex in its sequence position
     return num-1;
}
```

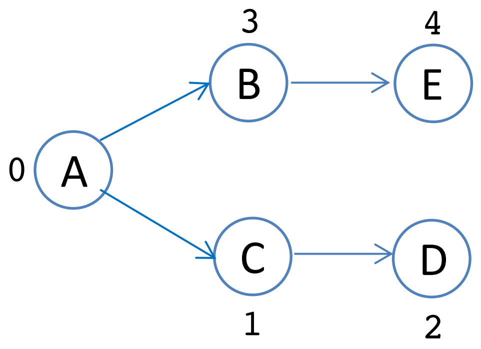


Start at A
num=4 → Visit A
num=4 → Visit B
num=4 → Visit E
Dead end
E: 4
num=3 ← Back up to B
Dead end
B: 3
num=2 ← Back up to A
num=2 → Visit C
num=2 → Visit D
Dead end
D: 2
num=1 ← Back up to C
Dead end
C: 1
num=0 ← Back up to A
Dead end
A: 0

DFS Topsort Implementation

Modification 4: Driver needs to start with the highest number, and also restart with current highest number. It also needs to set up an array for the topological sequence

```
// driver (this is the method called from any application
public void dfs() {
     boolean[] visited = new boolean[adjLists.length];
     int[] topSequence = new int[adjLists.length];
     int num = adjLists.length-1;
     for (int v=0; v < visited.length; v++) {
          if (!visited[v]) { // start/restart at v
               num = dfs(v, visited, num, topSequence);
          }
     }
}
```



	Start at B
num=4 →	Visit B
num=4 →	Visit E
	Dead end
	E: 4
num=3 ←	Back up to B
	Dead end
	B: 3
num=2 →	Return to driver
num=2 →	Restart at A
num=2 →	Visit C
num=2 →	Visit D
	Dead end
	D: 2
num=1 ←	Back up to C
	Dead end
	C: 1
num=0 ←	Back up to A
	Dead end
	A: 0

DFS Topsort Big O Running Time

Basic DFS running time is *O(n+e)*

What additional work is done for the topological numbering?

O(*n*) Recursive dfs

- Assign each vertex to a slot in topSequence array: O(1)

O(*1*) Driver

- Initialize num

Total: O(*n+e*)

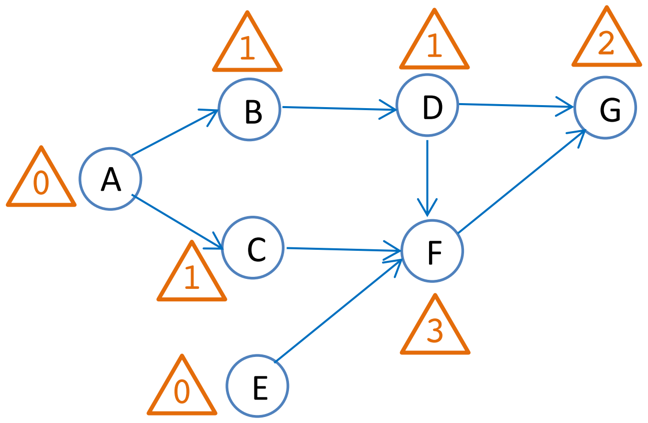
Topological Sorting using BFS

BFS Topsort

Since BFS is not recursive, and once a vertex is visited, there is no backing up to it, topological numbers need to be assigned in increasing order 0..n-1

This means we MUST start with vertices that have no incoming edges, i.e. indegree=0 (indegree is number of edges coming in/pointing at a vertex, outdegree is number of edges going out of a vertex)

So before we even start BFS, we need to go through the graph and compute indegree for each vertex

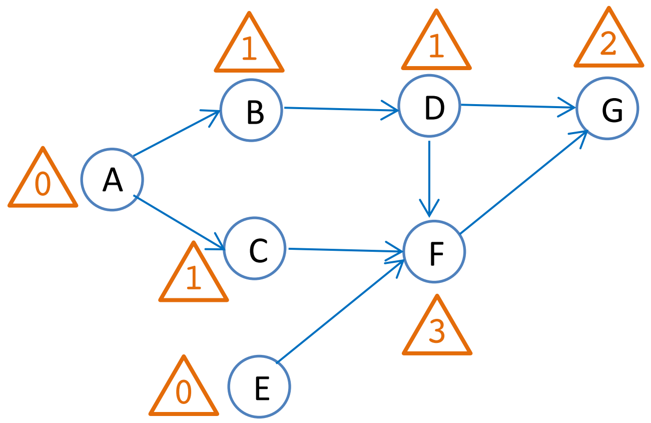


indegrees

A	B	C	D	E	F	G
0	1	1	1	0	3	2

BFS Topsort

After the indegrees are computed, we scan the indegrees array.
For each vertex that has indegree=0, we assign it the next higher topological number, and enqueue it



indegrees

0	1	1	1	0	3	2
A	B	C	D	E	F	G

queue

A E

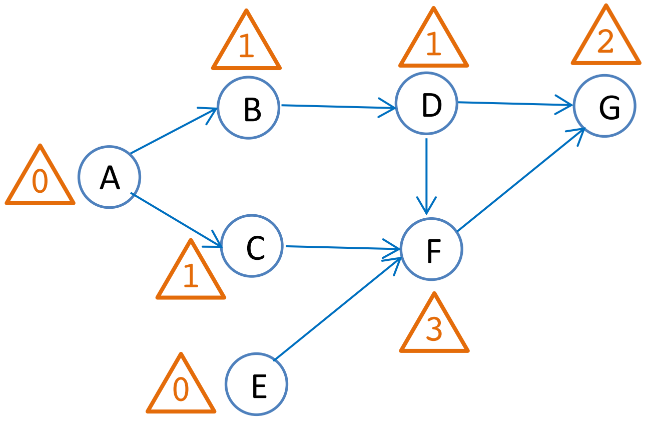
topSequence

A	E	-	-	-	-	-
0	1	2	3	4	5	6

BFS Topsort

Then we run a loop as long as queue is not empty.
In each iteration of the loop, we dequeue a vertex, v.
For each neighbor of v, we decrement the indegree count.
If the count goes to 0, we assign it the next higher topological number, and enqueue it.

Iteration 1



queue

← A E

indegrees

A	B	C	D	E	F	G
0	0	0	1	0	3	2

topSequence

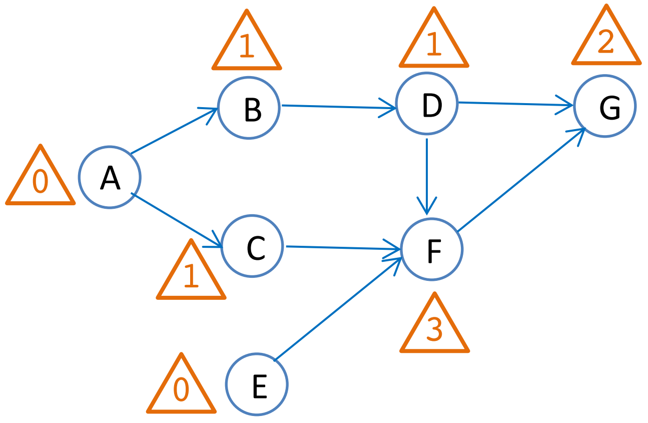
0	1	2	3	4	5	6
A	E	B	C	-	-	-

queue

E B C

BFS Topsort

Iteration 2



queue

← E B C

indegrees

A	B	C	D	E	F	G
0	0	0	1	0	2	2

topSequence

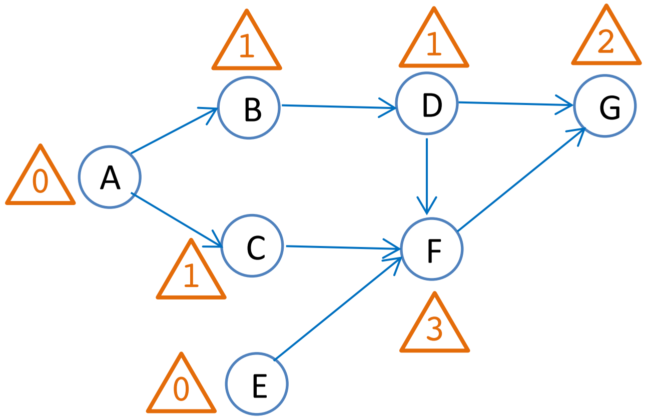
0	1	2	3	4	5	6
A	E	B	C	-	-	-

queue

B C

BFS Topsort

Iteration 3



queue

← B C

indegrees

A	B	C	D	E	F	G
0	0	0	0	0	2	2

topSequence

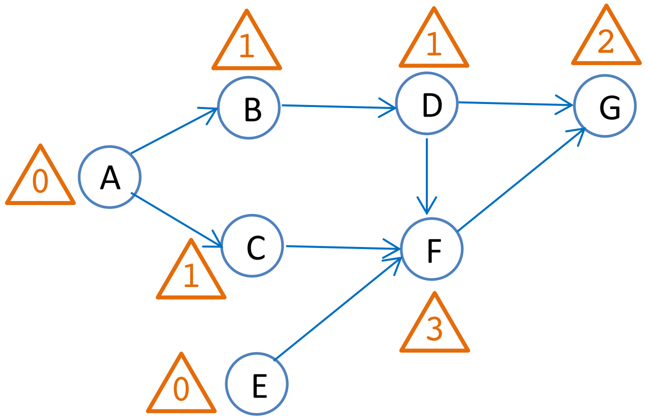
0	1	2	3	4	5	6
A	E	B	C	D	-	-

queue

C D

BFS Topsort

Iteration 4



queue

← C D

indegrees

A	B	C	D	E	F	G
0	0	0	0	0	1	2

topSequence

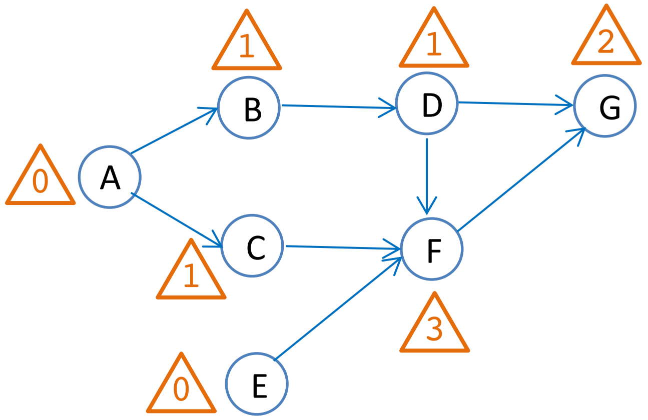
0	1	2	3	4	5	6
A	E	B	C	D	-	-

queue

D

BFS Topsort

Iteration 5



queue

← D

indegrees

A	B	C	D	E	F	G
0	0	0	0	0	0	1

topSequence

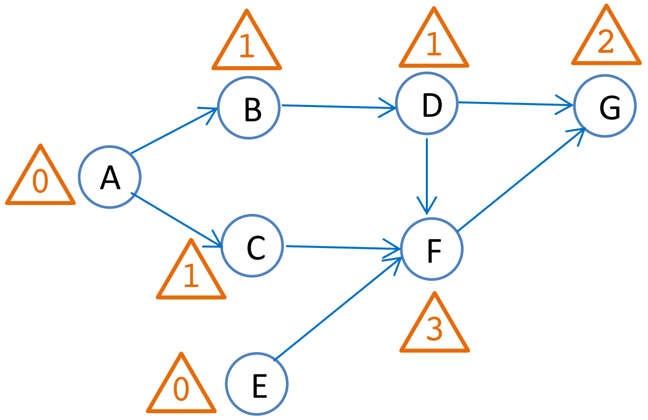
0	1	2	3	4	5	6
A	E	B	C	D	F	-

queue

F

BFS Topsort

Iteration 6



queue

← F

indegrees

0	0	0	0	0	0	0
A	B	C	D	E	F	G

topSequence

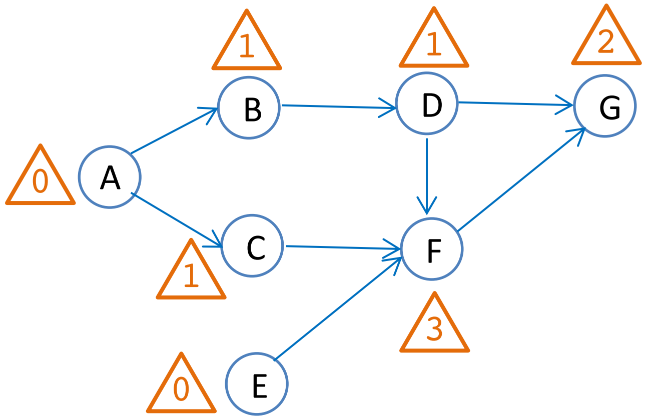
A	E	B	C	D	F	G
0	1	2	3	4	5	6

queue

G

BFS Topsort

Iteration 7



queue

← G

indegrees

A	B	C	D	E	F	G
0	0	0	0	0	0	0

topSequence

0	1	2	3	4	5	6
A	E	B	C	D	F	G

queue

BFS Topsort

```
bfsTopsort()
   compute indegrees of all vertices
   topnum ← 0
   for each vertex, v, do
       if indegrees[v] == 0 then
           topSequence[topnum] = v
           topnum++
           enqueue(v)
        endif
   endfor
   while queue is not empty do
      v ← dequeue()
      for each neighbor, w, of v do
          indegrees[v]--
          if indegrees[v] == 0 then
              topSequence[topnum] = v
              topnum++
              enqueue(v)
          endif
      endfor
   endwhile
```

Since we start with all vertices that don't have any incoming edges, all other vertices should be reachable from these initial vertices. Which means a driver is not needed.

BFS Topsort Big O Running Time

Basic BFS running time is *O(n+e)*

What additional work is done for the topological numbering?

O(*n+e*) Indegrees computation

- Go through entire graph and count incoming edges for each vertex

O(*n*) BFS

- Assign each vertex to a slot in topSequence array: O(1)
- Decrement indegree

Total: O(*n+e*)