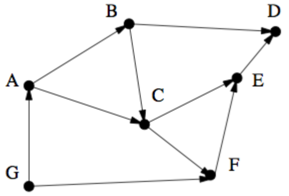
## Recall the Topological Sorting Problem in DAG

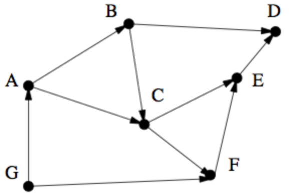
Recall:

- A DAG: a directed acyclic graph, i.e. a directed graph with no (directed) cycles
- Topological Sorting: Determine a linear ordering of the vertices of a DAG so that for every edge, its starting vertex is listed before its ending vertex.



- In this graph, the first vertex in a topological sort must be G.  Why?

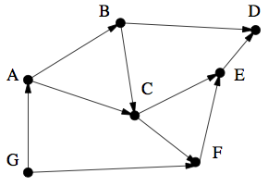
## Source Removal Algorithm for Topological Sorting



- G must come first since it is the only vertex with no incoming edges

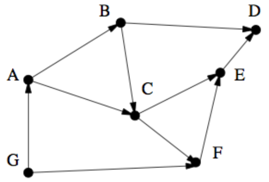
- What must be true of any vertex that is the first vertex in a topological sort?
- How can we determine what vertex comes next?

## Source Removal Algorithm - 1



- What must be true of any vertex that is the first vertex in a topological sort?
  - It must have no incoming edges --- **its in-degree = 0**
- How can we determine what vertex comes next?
  - It must have no incoming edges after the first vertex and its outgoing edges are removed from the graph!

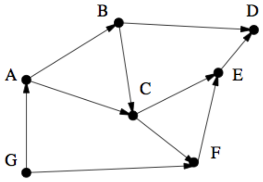
## Source Removal Algorithm - 2



- This gives us a pretty good idea of an iterative algorithm that can be used to produce a topological sort. Assuming we can compute and keep track of the in-degree of each vertex efficiently as we remove vertices and their associated edges from the graph.

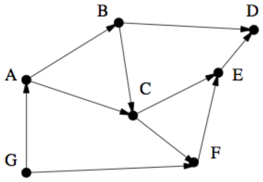
- How can we efficiently compute the in-degree of a directed graph?

## Source Removal Algorithm - 3



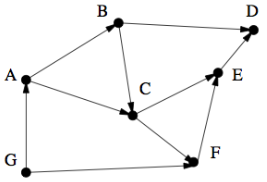
- How can we efficiently compute the in-degree of each vertex of a directed graph?
  - For each vertex have a field that will keep the count of incoming edges.
  - Traverse the adjacency list (or matrix) and increment the target vertex's in-degree field for each edge.

## Source Removal Algorithm - 4



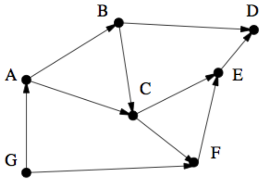
- How can we efficiently update in-degree of the remaining directed graph after a vertex has been removed?

## Source Removal Algorithm - 5



- How can we efficiently update in-degree of the remaining directed graph after a vertex has been removed?
  - Traverse the adjacency list of the vertex being removed and decrement the target vertex in-degree field for each edge.

## Source Removal Algorithm - 6



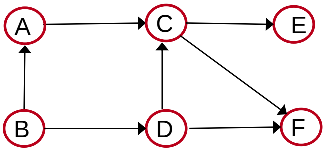
| | | remove G | remove A | remove B | remove C | remove F | remove E | remove D |
|---|---|---|---|---|---|---|---|---|
| A | 1 | →0 | *order* | | | | | |
| B | 1 | 1 | →0 | *order* | | | | |
| C | 2 | 2 | →1 | →0 | *order* | | | |
| D | 2 | 2 | 2 | →1 | 1 | 1 | →0 | *order* |
| E | 2 | 2 | 2 | 2 | →1 | →0 | *order* | |
| F | 2 | →1 | 1 | 1 | →0 | *order* | | |
| G | 0 | *order* | | | | | | |

## Source Removal Algorithm

```
Compute in-degree of all vertices
Repeat
    - identify a source vertex (in-degree = 0)
       (a vertex with no incoming edges)
    - remove the source and all the edges from
      it, update in-degrees of target vertices
Until either
    - no vertex is left-all vertices marked done
     (problem is solved) or
    - no source among remaining vertices (not a dag)
```

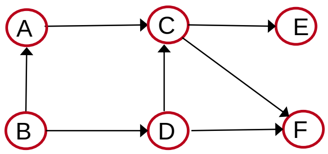
- Efficiency: same as efficiency of the DFS-based algorithm

## Source Removal Algorithm Example



- Every DAG has at least one source and a sink.
- Above graph has:
  - One Source, two Sinks
  - 4 different possible topological sorts. What are they?

## Source Removal Algorithm Example



4 linearizations (topological sorts) are

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