Decomposition of Graphs: Topological Sort

Daniel Kane

Department of Computer Science and Engineering
University of California, San Diego

Graph Algorithms
Data Structures and Algorithms

Learning Objectives

- Implement the topological sort algorithm.
- Prove that a DAG can be linearly ordered.

Outline

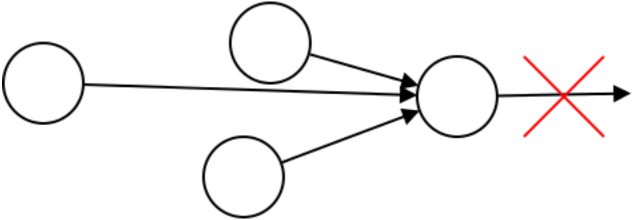
1. Idea
2. Algorithms
3. Correctness

Last Time

- Directed graphs.
- Linearly order vertices.
- Requires DAG.

Last Vertex

Consider the last vertex in the ordering. It cannot have any edges pointing out of it.



Sources and Sinks

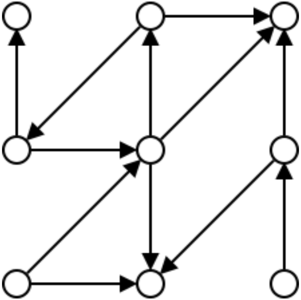
Definition

A source is a vertex with no incoming edges.

A sink is a vertex with no outgoing edges.

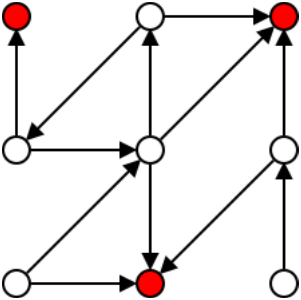
Problem

How many sinks does the graph below have?



Solution

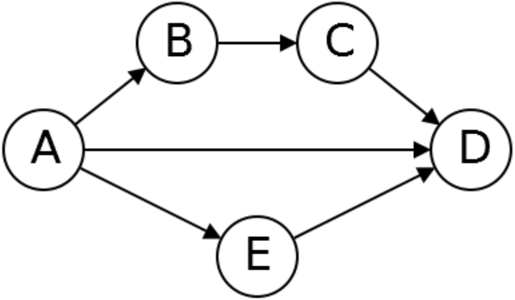
3.



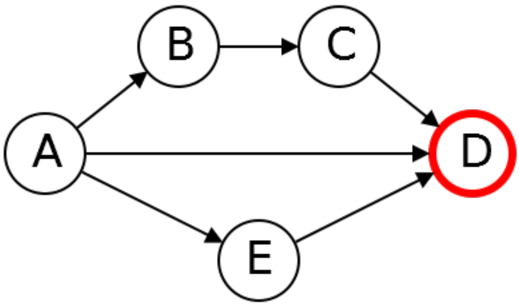
Idea

- Find sink.
- Put at end of order.
- Remove from graph.
- Repeat.

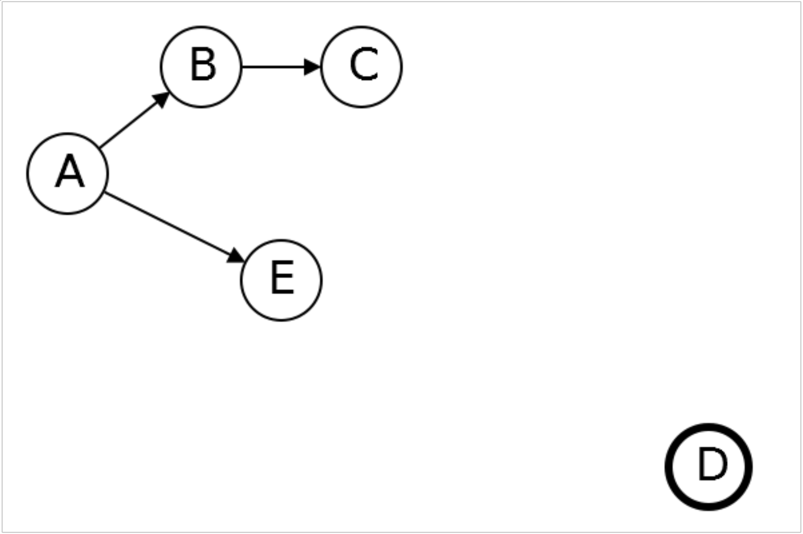
Example



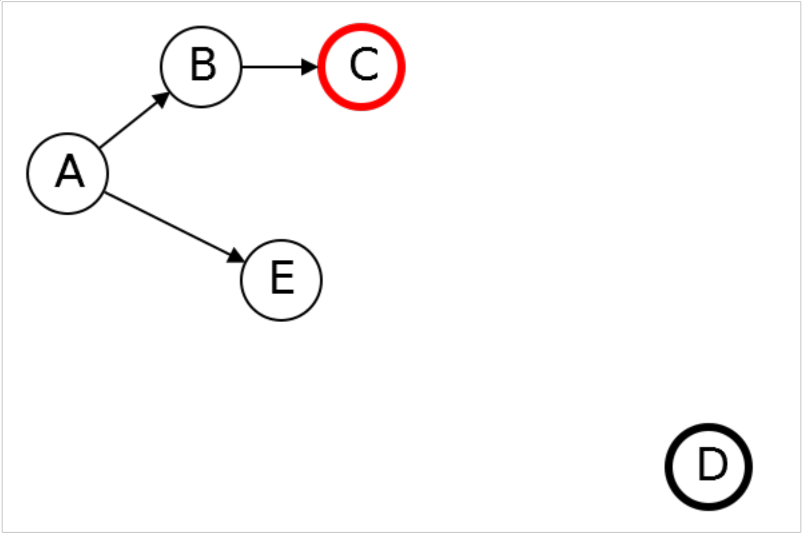
Example



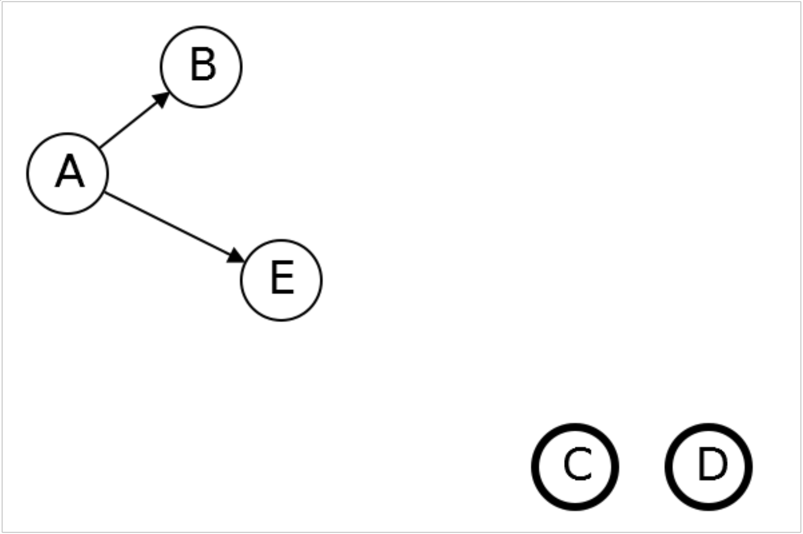
Example



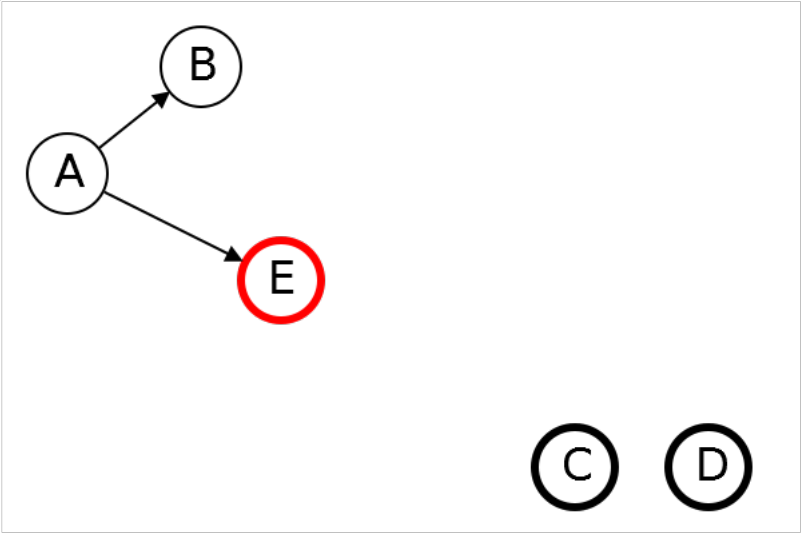
Example



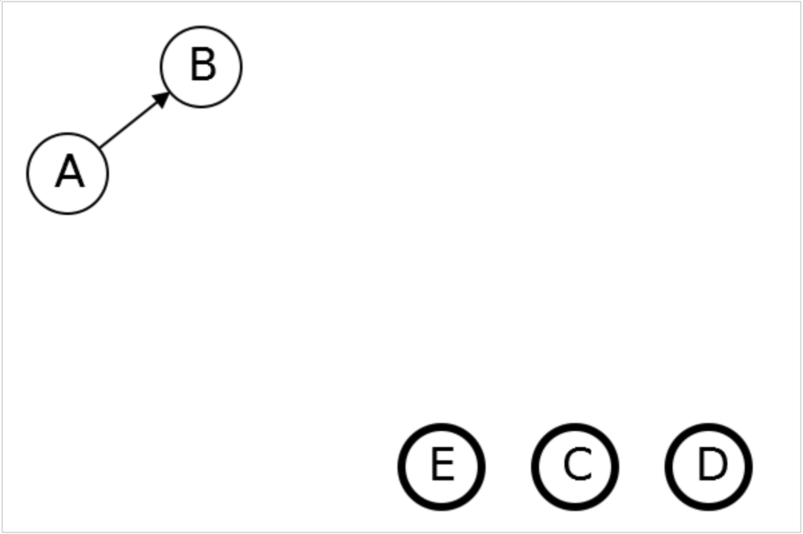
Example



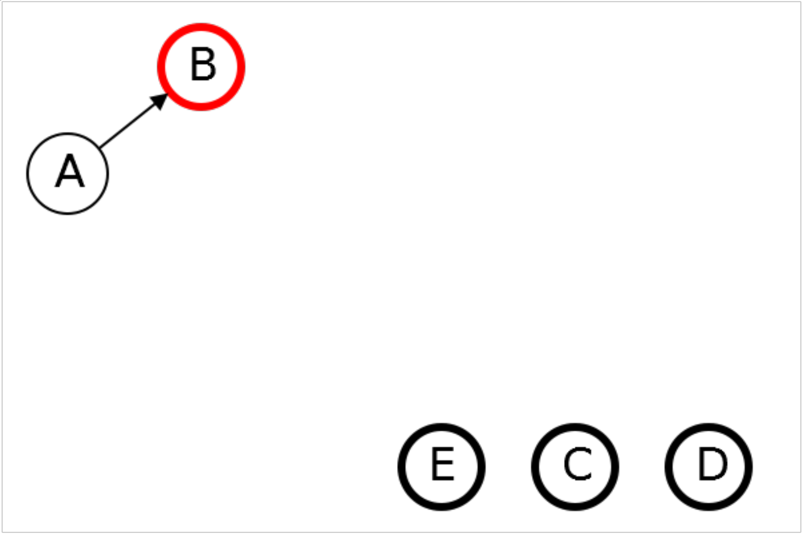
Example



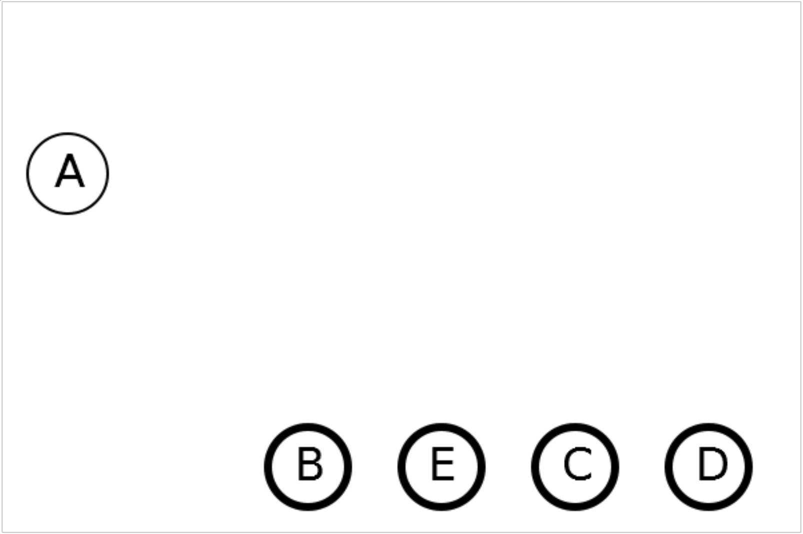
Example



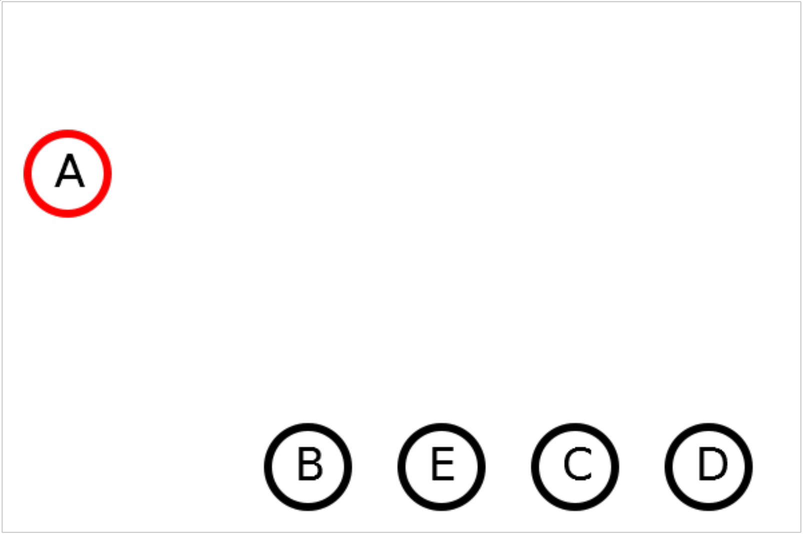
Example



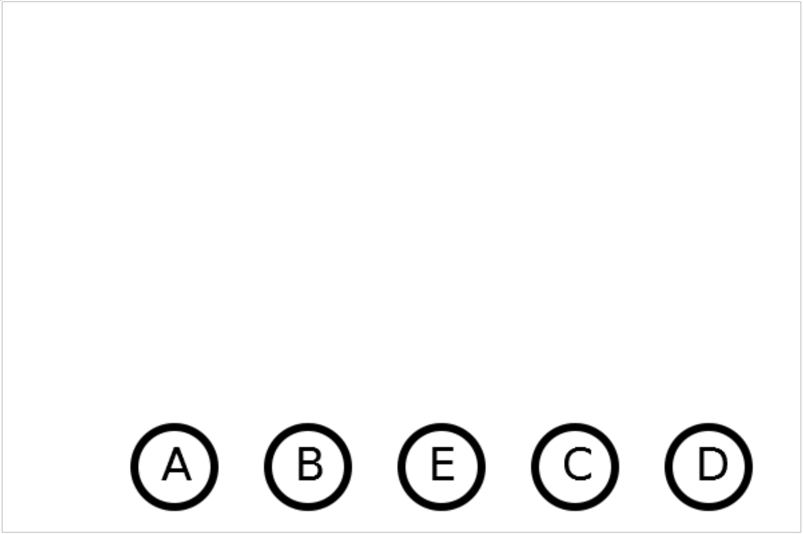
Example



Example



Example



Finding Sink

Question: How do we know that there is a sink?

Follow Path

Follow path as far as possible

$v_1 \to v_2 \to \ldots \to v_n$. Eventually either:

- Cannot extend (found sink).
- Repeat a vertex (have a cycle).

Outline

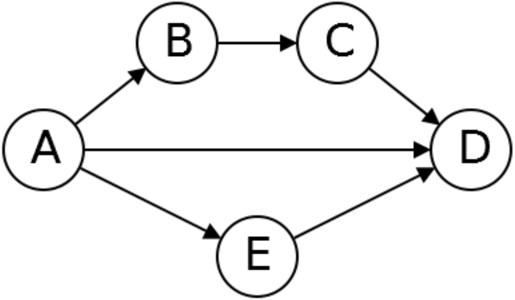
1. Idea
2. Algorithms
3. Correctness

First Try

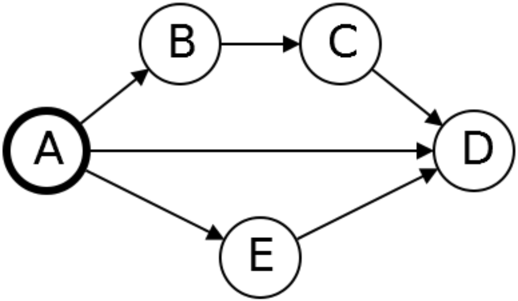
```
LinearOrder(G)

while G non-empty:
   Follow a path until cannot extend
   Find sink v
   Put v at end of order
   Remove v from G
```

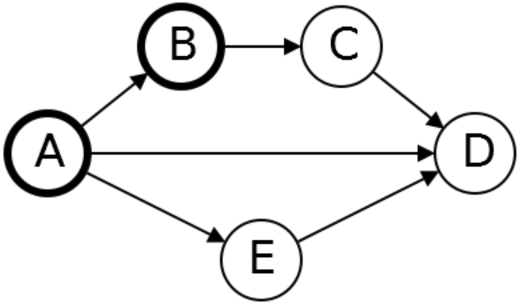
Example



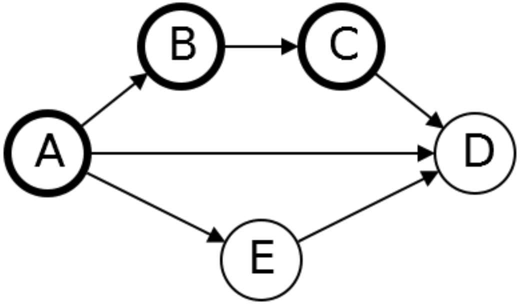
Example



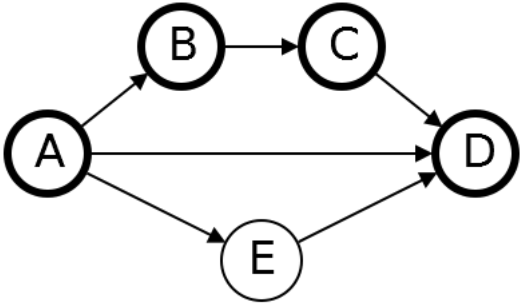
Example



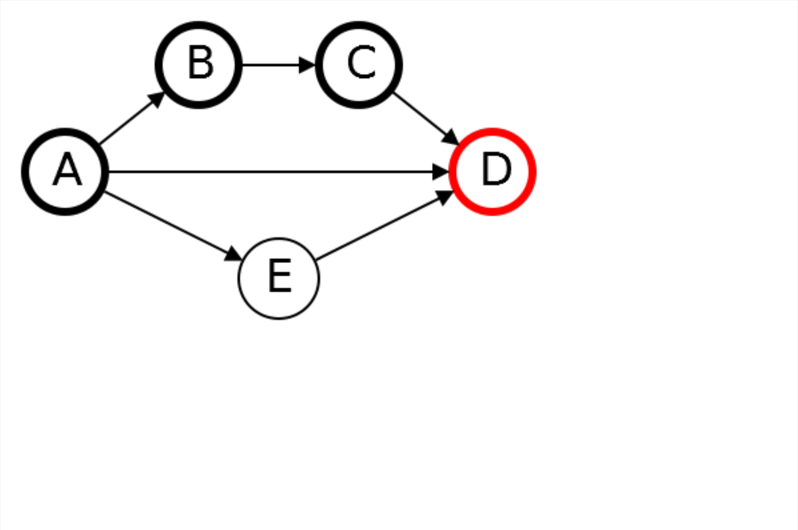
Example



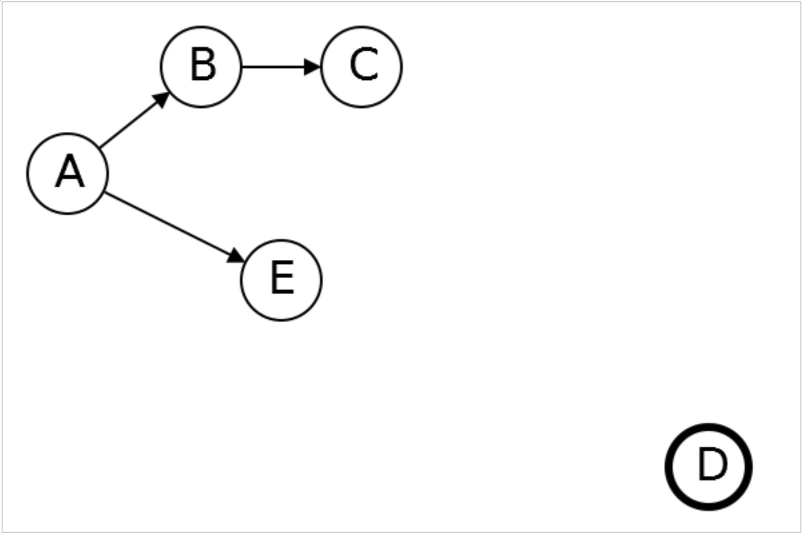
Example



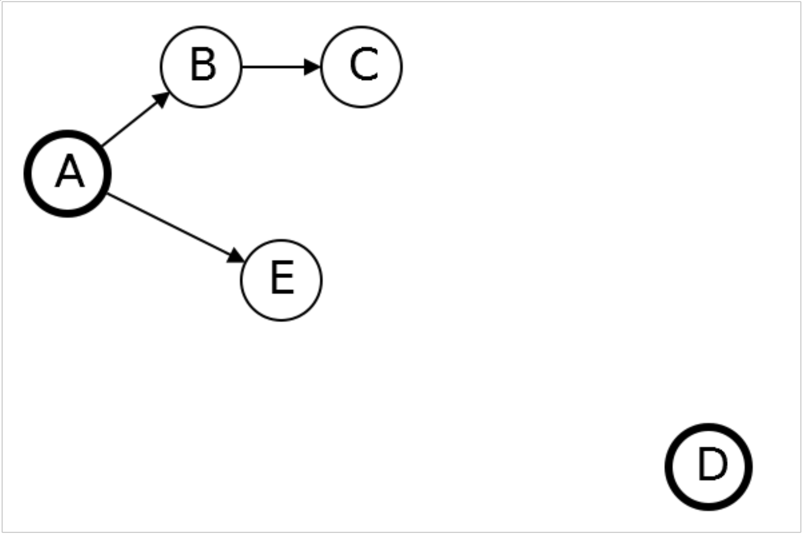
Example



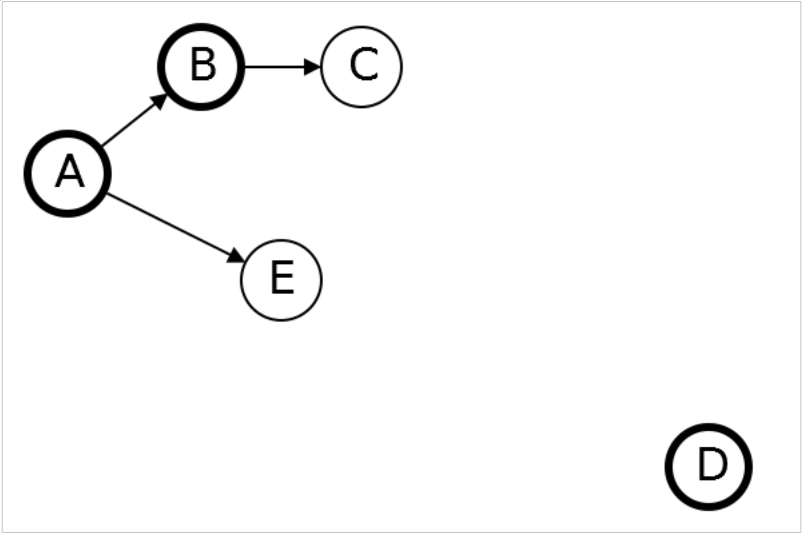
Example



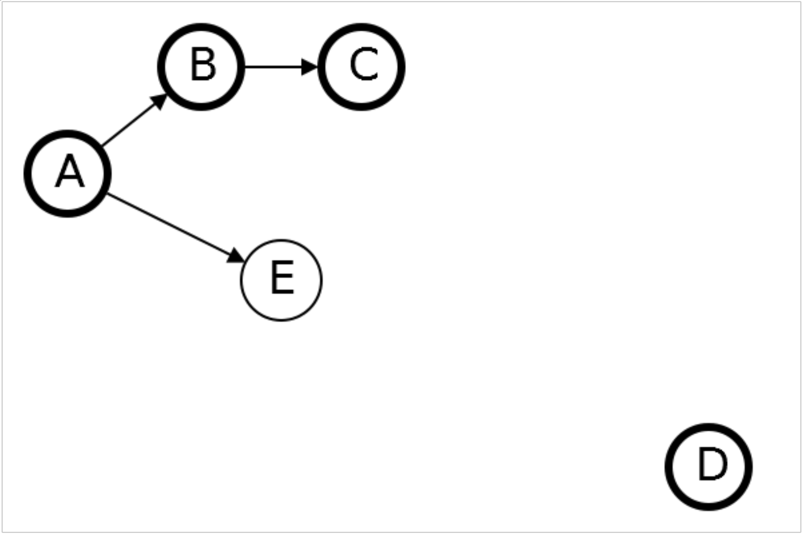
Example



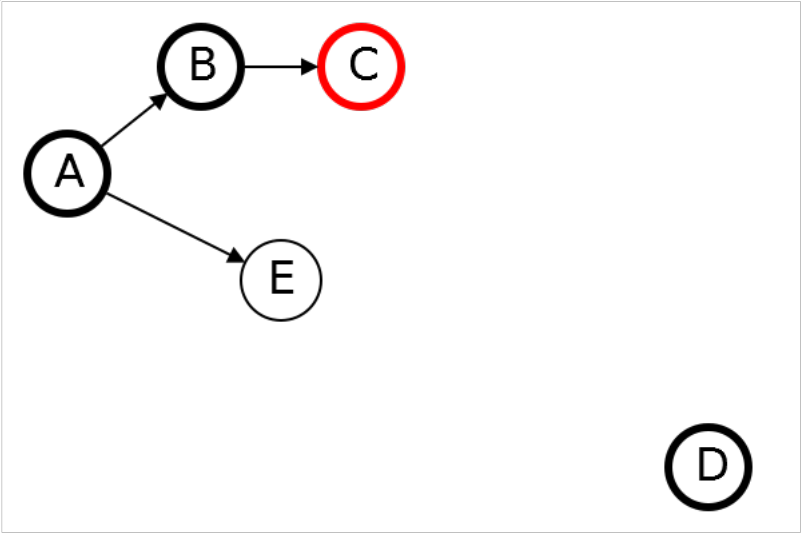
Example



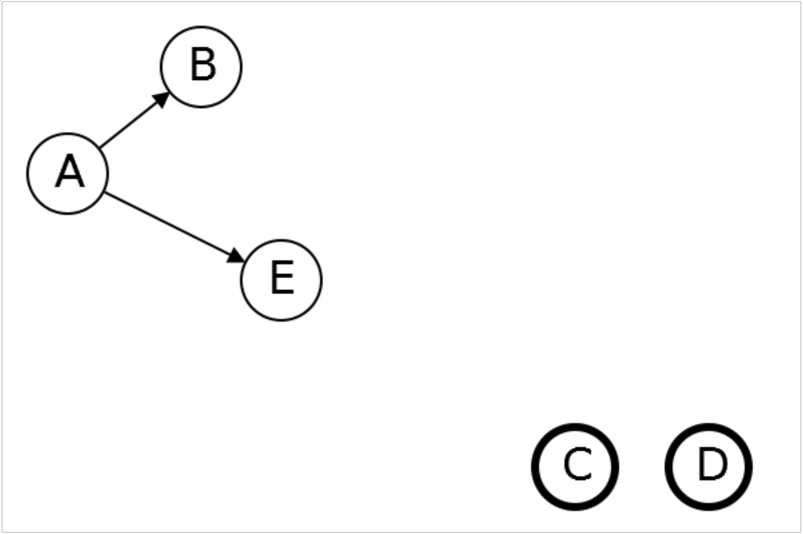
Example



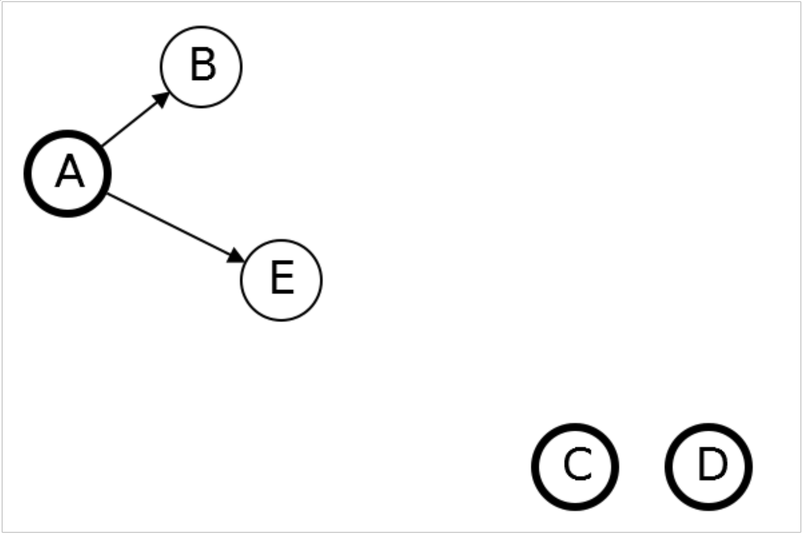
Example



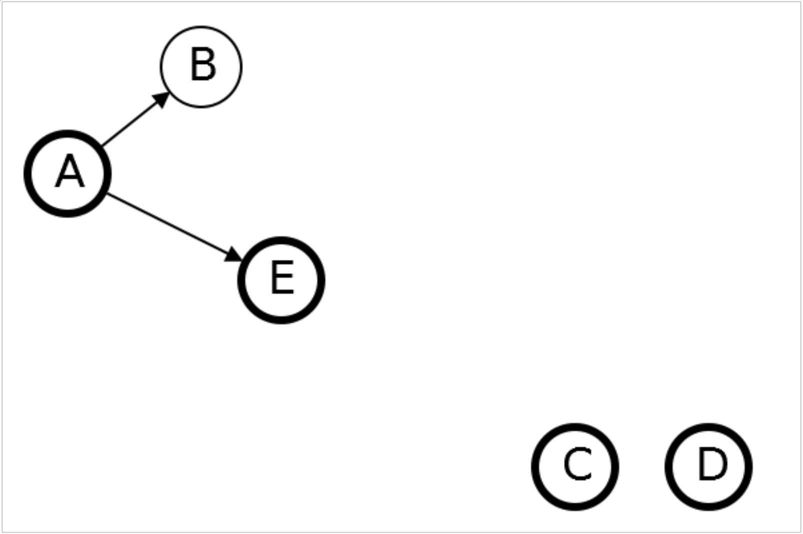
Example



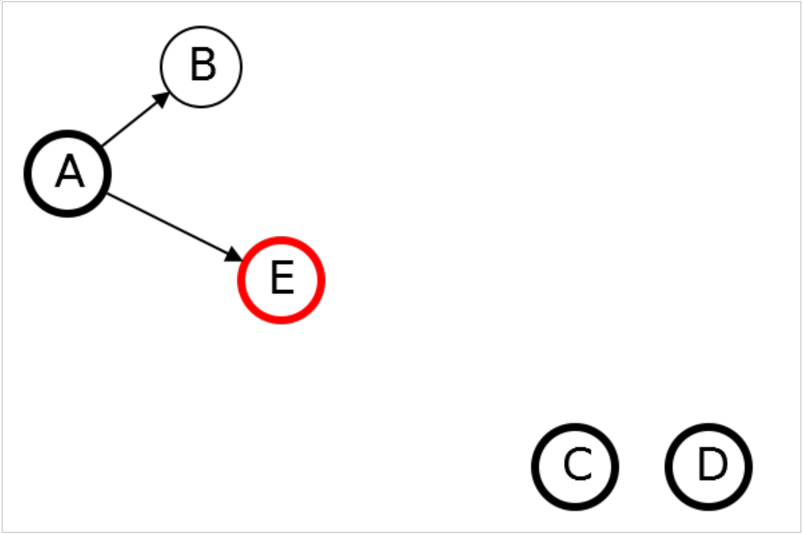
Example



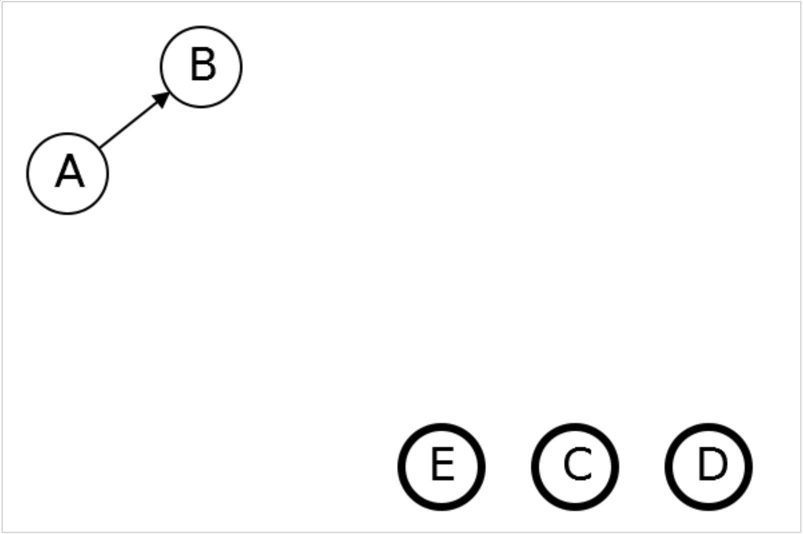
Example



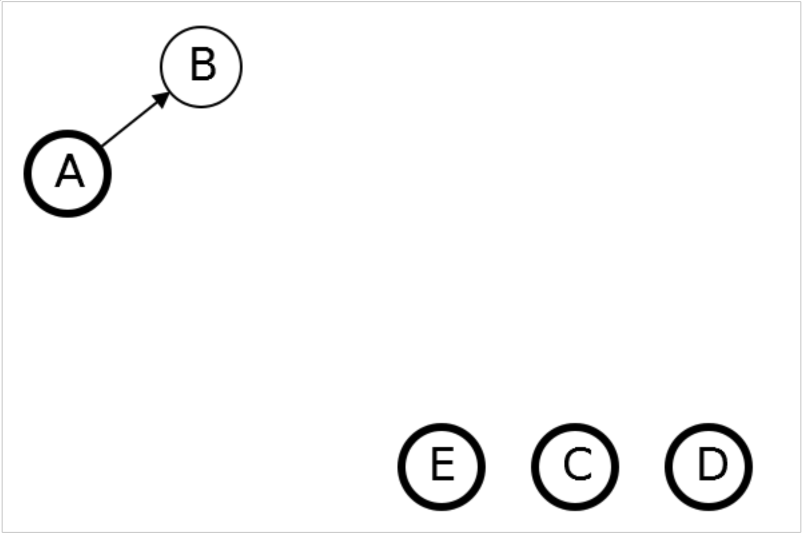
Example



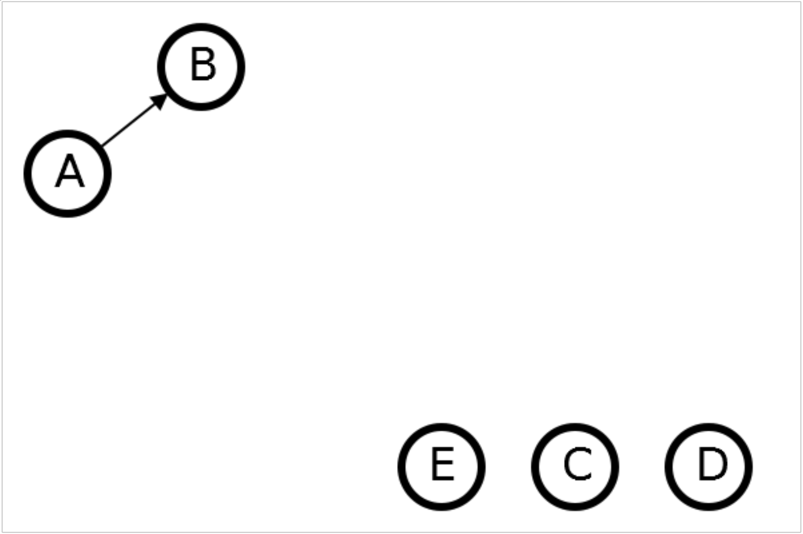
Example



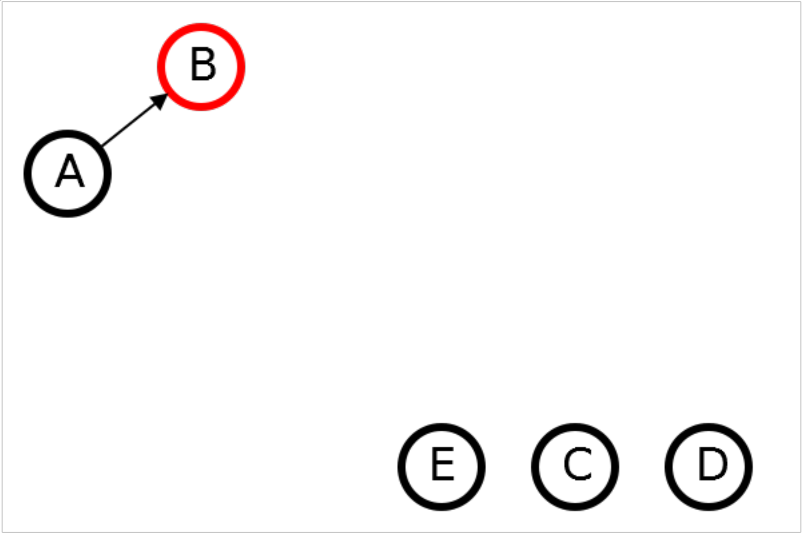
Example



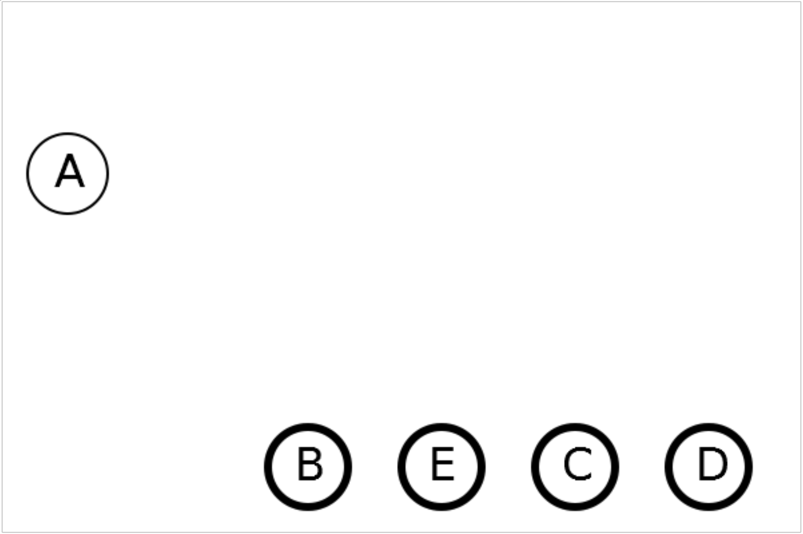
Example



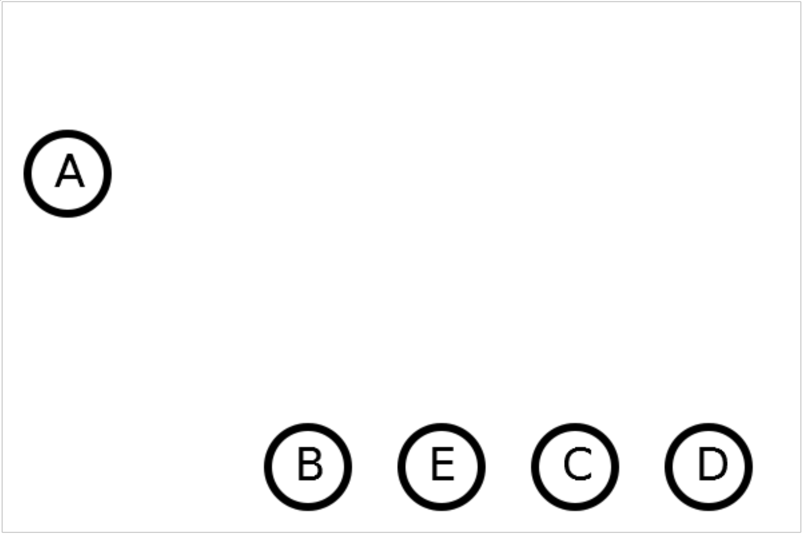
Example



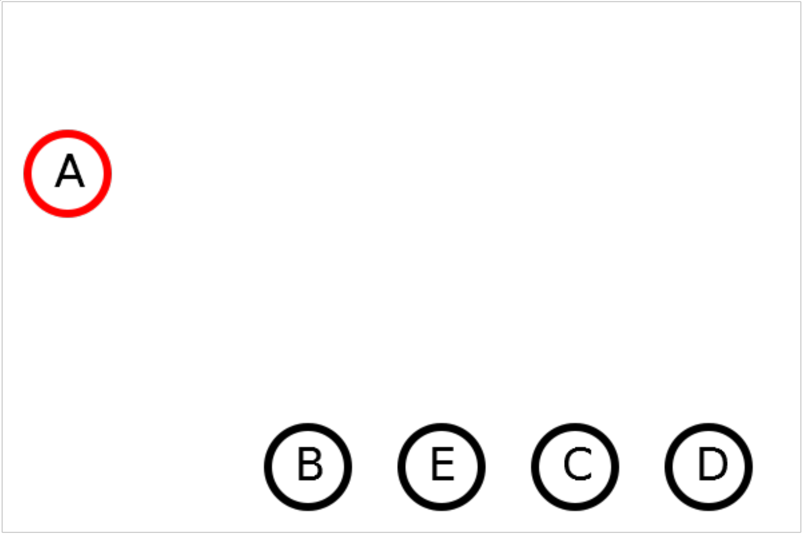
Example



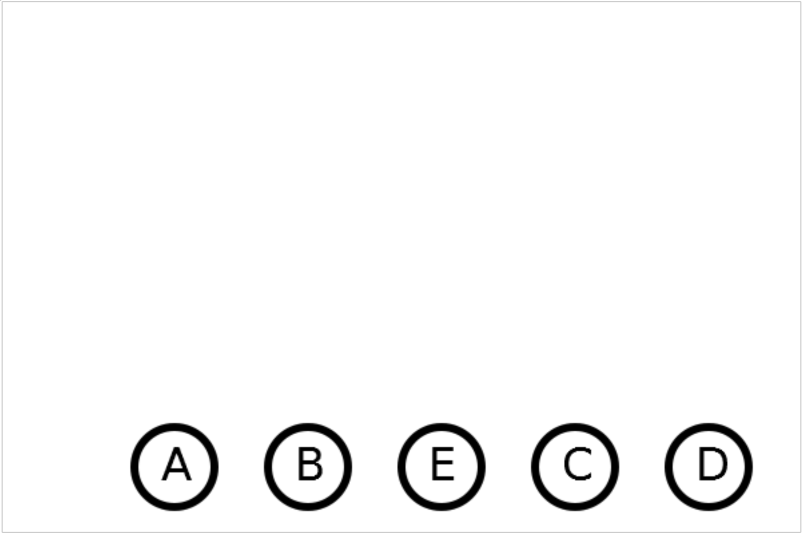
Example



Example



Example



Runtime

- $O(|V|)$ paths.
- Each takes $O(|V|)$ time.
- Runtime $O(|V|^2)$.

We need to compute one path for each vertex in the grid which has a sink as the final vertex in the end hence O(V) paths

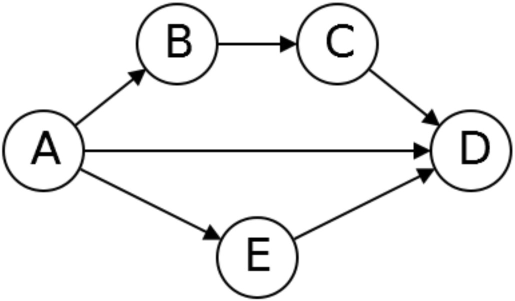
Each takes maximum worst case time of O(V) as the path could be as long as all the vertices in the graph

Speed Up

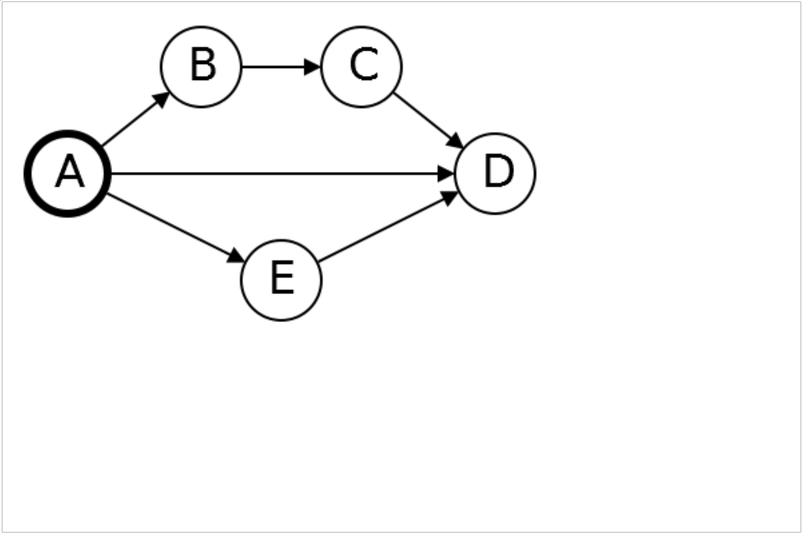
- Retrace same path every time.
- Instead only back up as far as necessary.

we're doing something a little bit inefficient with this algorithm. We started with this vertex,we followed this huge long path until we found the sink and then we remove it from the graph. Then what we probably did was we started that same vertex again, followed basically the same path until we got almost to the very end, until we get to right before that vertex that we removed. And we're sort of repeating all of this work. Why do we need to follow this whole path again? We could just back up one step along the path and then keep going from there, and so that's what we're going to do.

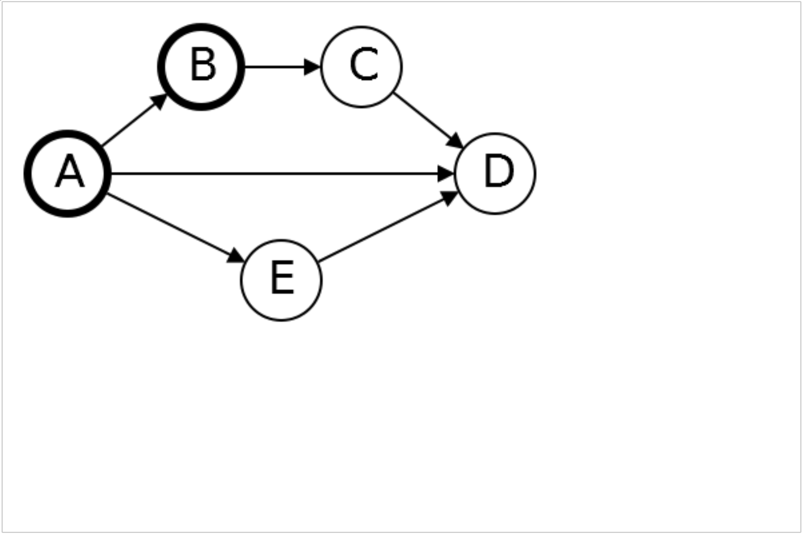
Example



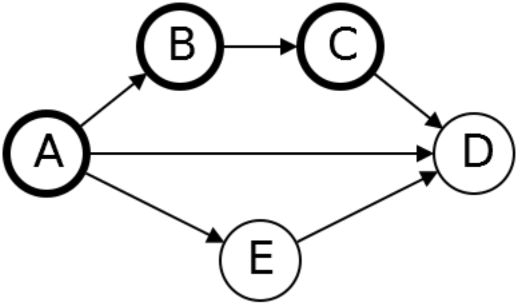
Example



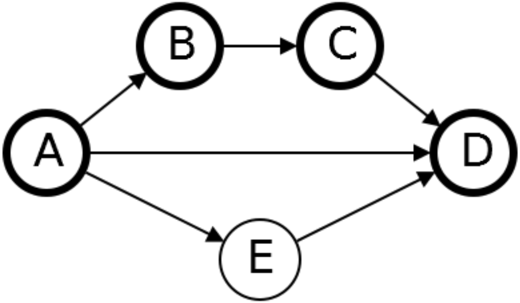
Example



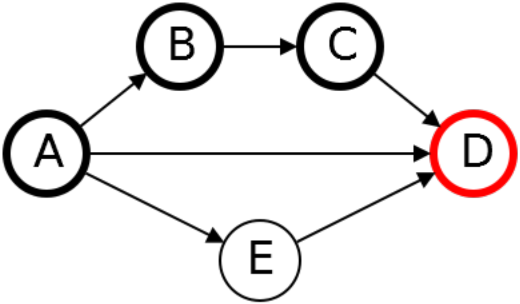
Example



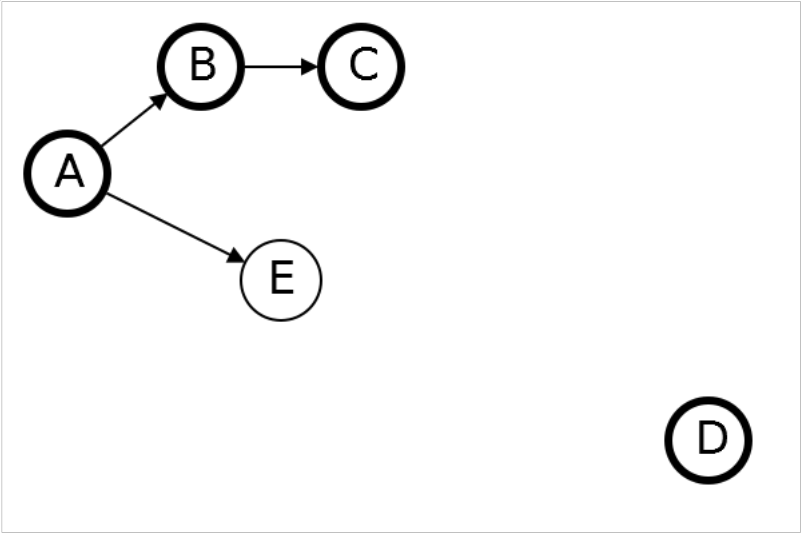
Example



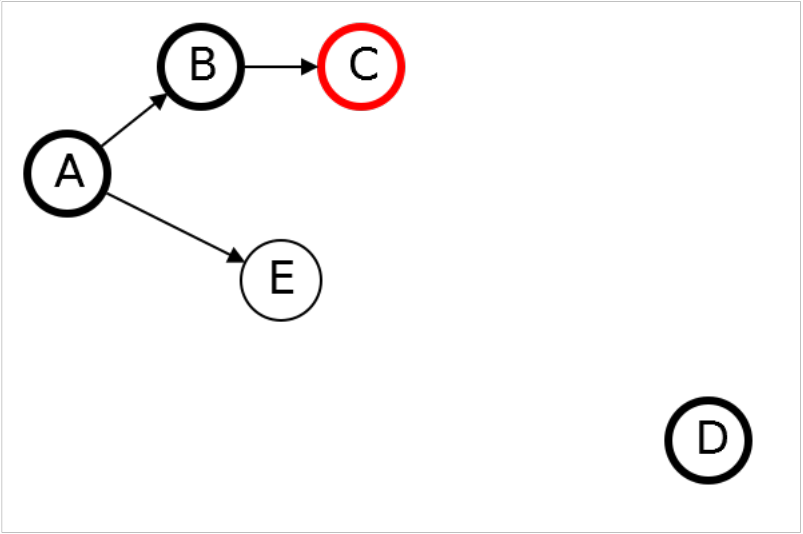
Example



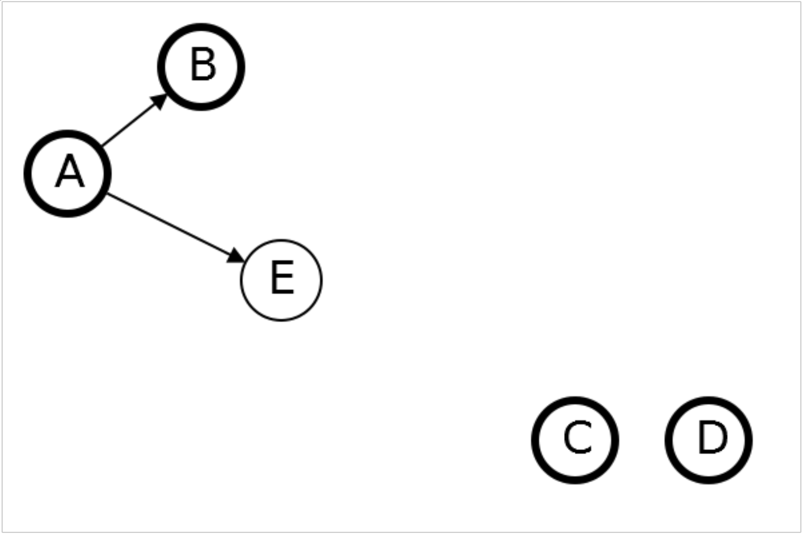
Example



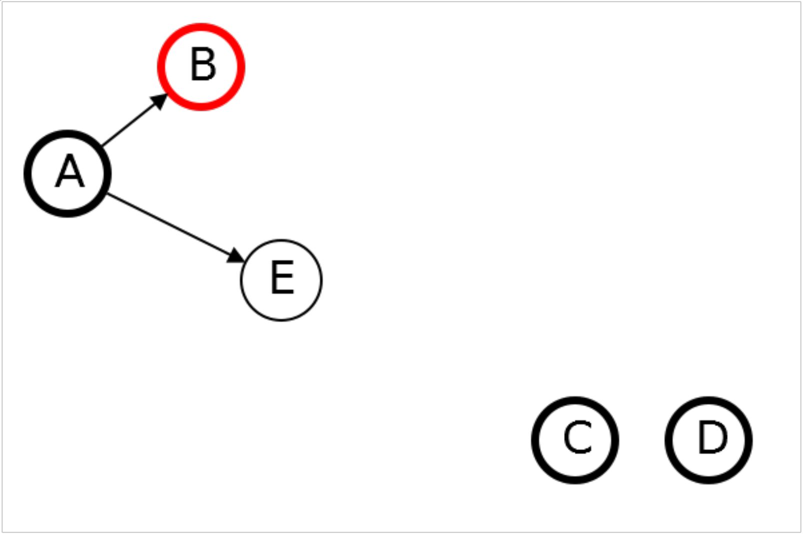
Example



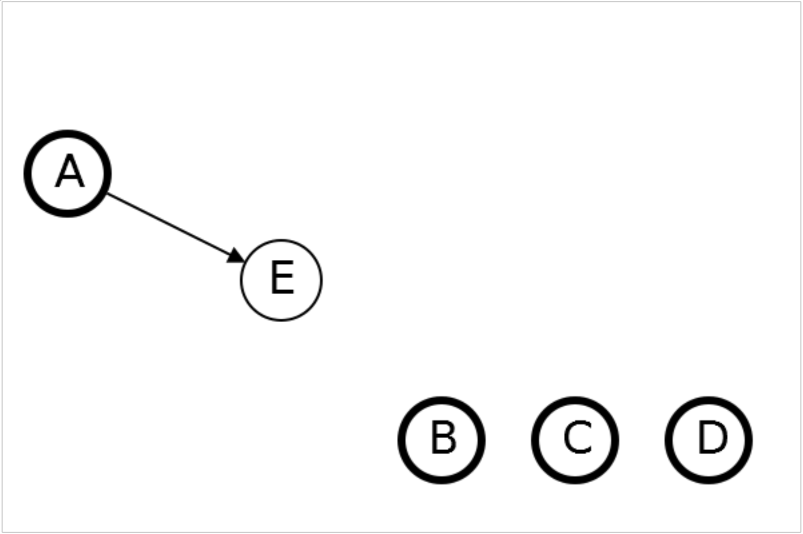
Example



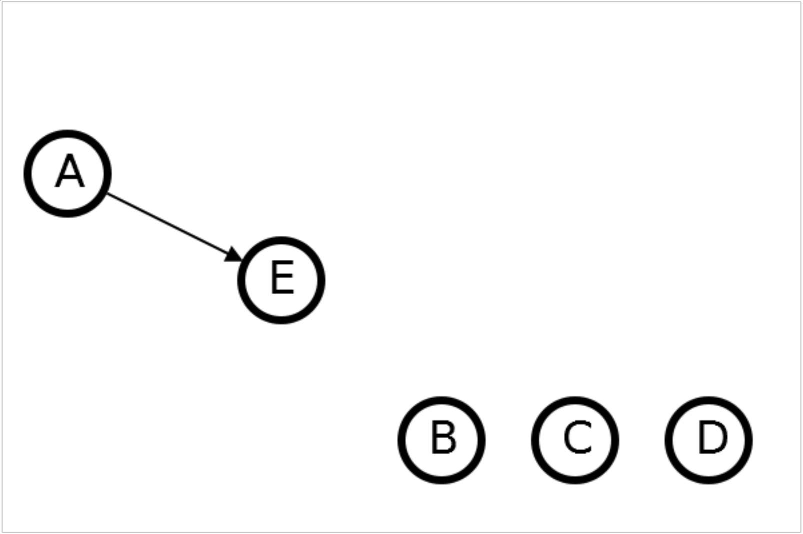
Example



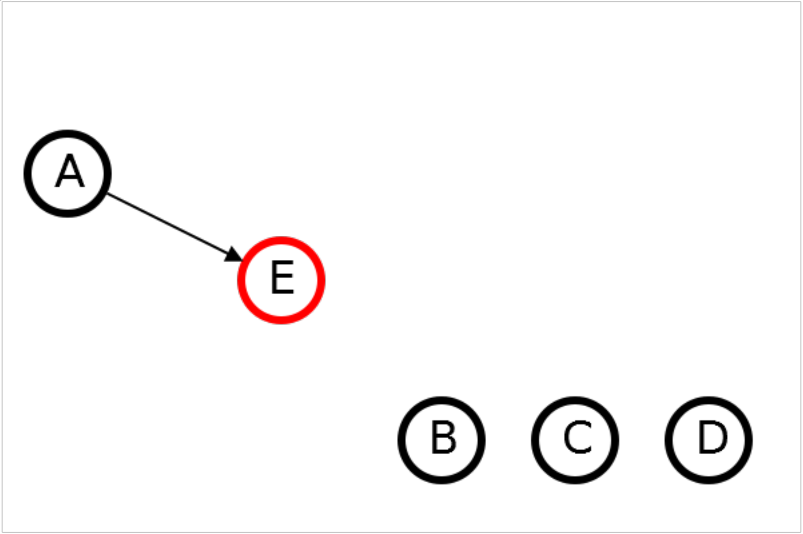
Example



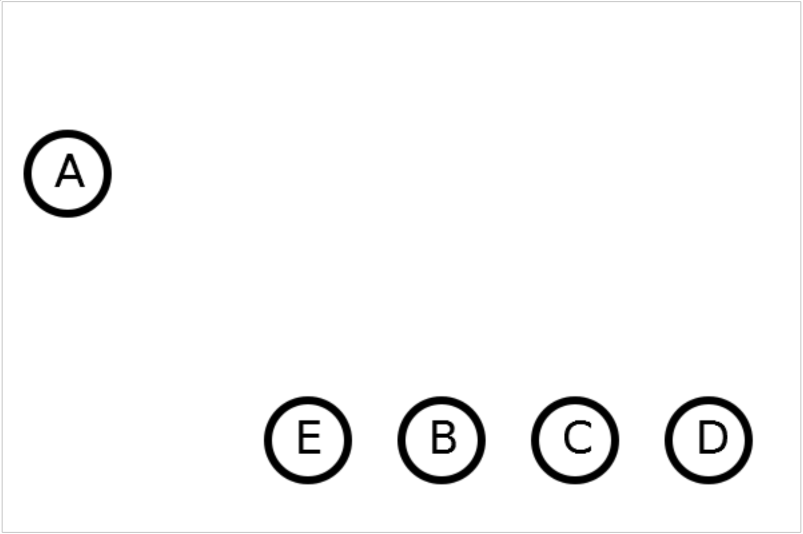
Example



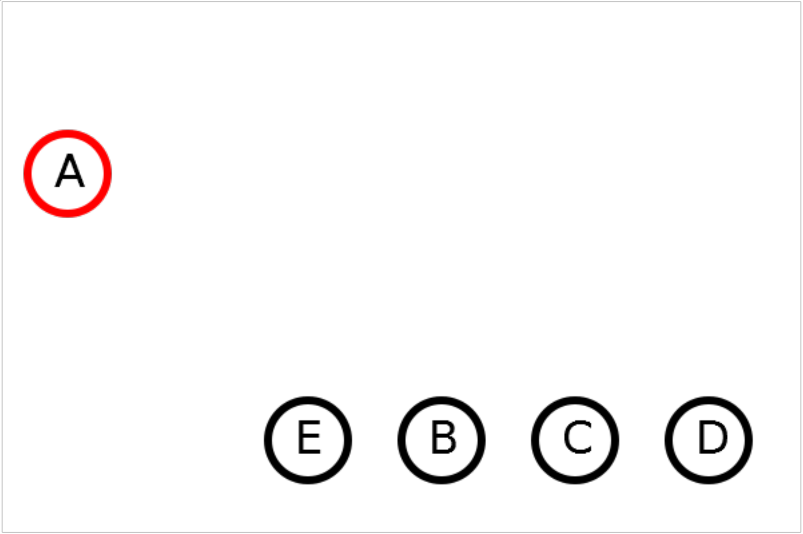
Example



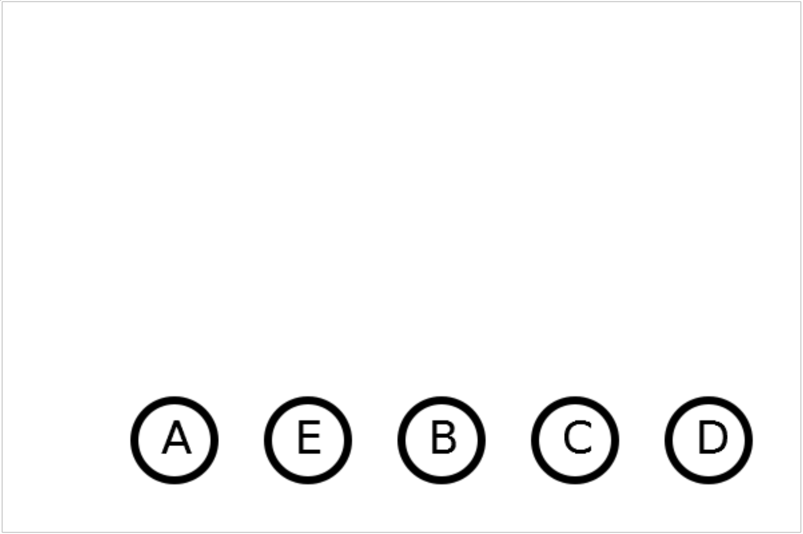
Example



Example



Example



Observation

This is just DFS!

Observation

This is just DFS!

We are sorting vertices based in postorder!

Better Algorithm

```
TopologicalSort(G)
DFS(G)
sort vertices by reverse post-order
```

We just have to like, remember the order in which we left our vertices

Outline

1. Idea
2. Algorithms
3. Correctness

Theorem

Theorem

If G is a DAG, with an edge u to v, $\texttt{post}(u) > \texttt{post}(v)$.

Proof

Consider cases

- Explore v before exploring u.
- Explore v while exploring u.
- Explore v after exploring u (cannot happen since there is an edge).

Case I

Explore v before exploring u.

- Cannot reach u from v (DAG)
- Must finish exploring v before find u
- $\texttt{post}(u) > \texttt{post}(v)$.

Case II

Explore v while exploring u.
Must finish exploring v before can finish exploring u. Therefore $\texttt{post}(u) > \texttt{post}(v)$.

Next Time

Connectivity in directed graphs.