

# Lecture 15: Basic Graph Algorithms II

Michael Dinitz

October 14, 2025  
601.433/633 Introduction to Algorithms

# Introduction

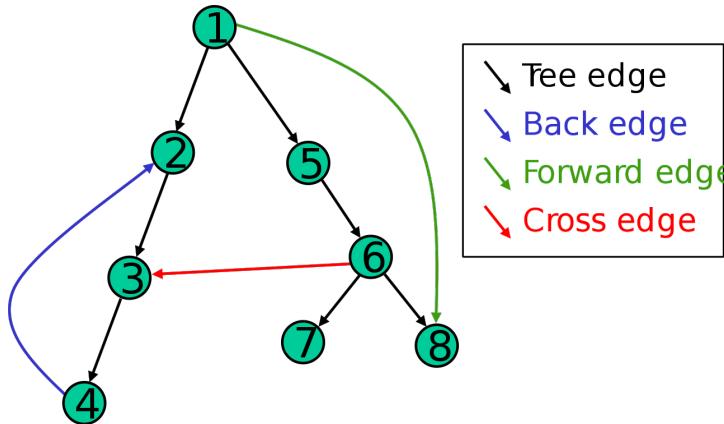
Last time: BFS and DFS

Today: Topological Sort, Strongly Connected Components

- ▶ Both very classical and important uses of DFS!

# Edge Types

DFS naturally gives a spanning forest: edge  $(v, u)$  if  $\text{DFS}(v)$  calls  $\text{DFS}(u)$



**Forward Edges:**  $(v, u)$  such that  $u$  descendant of  $v$  (includes tree edges)

$$\text{start}(v) < \text{start}(u) < \text{finish}(u) < \text{finish}(v)$$

**Back Edges:**  $(v, u)$  such that  $u$  an ancestor of  $v$

$$\text{start}(u) < \text{start}(v) < \text{finish}(v) < \text{finish}(u)$$

**Cross Edges:**  $(v, u)$  such that  $u$  neither a descendant nor an ancestor of  $v$

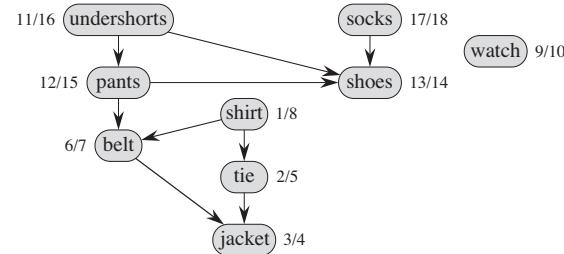
$$\text{start}(u) < \text{finish}(u) < \text{start}(v) < \text{finish}(v)$$

# Topological Sort

# Definitions

## Definition

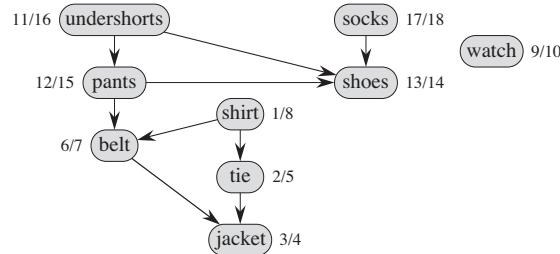
A directed graph  $\mathbf{G}$  is a *Directed Acyclic Graph (DAG)* if it has no directed cycles.



# Definitions

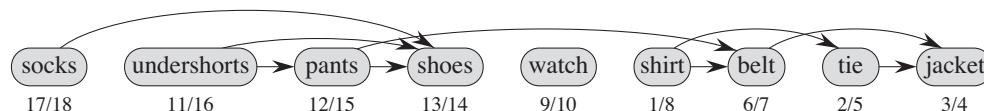
## Definition

A directed graph  $\mathbf{G}$  is a *Directed Acyclic Graph (DAG)* if it has no directed cycles.



## Definition

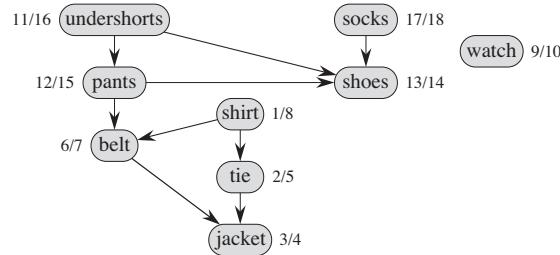
A *topological sort*  $v_1, v_2, \dots, v_n$  of a DAG is an ordering of the vertices such that all edges are of the form  $(v_i, v_j)$  with  $i < j$ .



# Definitions

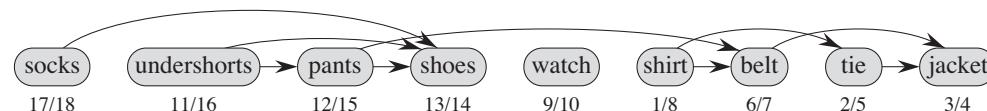
## Definition

A directed graph  $\mathbf{G}$  is a *Directed Acyclic Graph (DAG)* if it has no directed cycles.



## Definition

A *topological sort*  $v_1, v_2, \dots, v_n$  of a DAG is an ordering of the vertices such that all edges are of the form  $(v_i, v_j)$  with  $i < j$ .



Q: Can we always topologically sort a DAG? How fast?

# Topological Sort

Algorithm (informal): Run  $\text{DFS}(G)$ . When  $\text{DFS}(v)$  returns, put  $v$  at beginning of list

# Topological Sort

Algorithm (informal): Run  $\text{DFS}(G)$ . When  $\text{DFS}(v)$  returns, put  $v$  at beginning of list

```
DFS( $G$ ) {  
    list  $\rightarrow$  head = NULL;  
    t = 0;  
    for all  $v \in V$  {  
        start( $v$ ) = 0;  
        finish( $v$ ) = 0;  
    }  
    while  $\exists v \in V$  with  $\text{start}(v) = 0$  {  
        DFS( $v$ );  
    }  
}
```



```
DFS( $v$ ) {  
    t = t + 1;  
    start( $v$ ) = t;  
    for each edge  $(v, u) \in A[v]$  {  
        if  $\text{start}(u) == 0$  then DFS( $u$ );  
    }  
    t = t + 1;  
    finish( $v$ ) = t;  
    temp = list  $\rightarrow$  head;  
    list  $\rightarrow$  head =  $v$ ;  
    list  $\rightarrow$  head  $\rightarrow$  next = temp;  
}
```

# Characterizing DAGs

## Theorem

A directed graph  $\mathbf{G}$  is a DAG if and only if  $DFS(\mathbf{G})$  has no back edges.

# Characterizing DAGs

## Theorem

A directed graph  $\mathbf{G}$  is a DAG if and only if  $DFS(\mathbf{G})$  has no back edges.

## Proof.

Only if ( $\Rightarrow$ ): contrapositive. If  $\mathbf{G}$  has a back edge:

$DFS(\mathbf{G})$

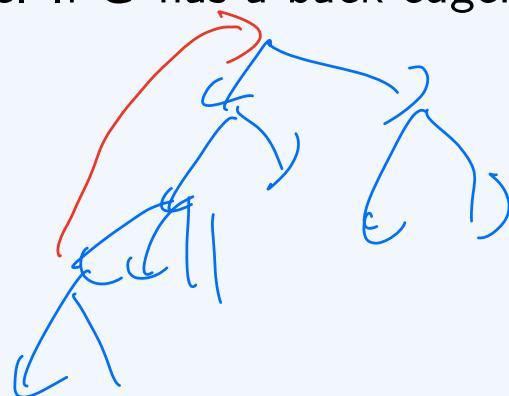
# Characterizing DAGs

## Theorem

A directed graph  $\mathbf{G}$  is a DAG if and only if  $DFS(\mathbf{G})$  has no back edges.

## Proof.

Only if ( $\Rightarrow$ ): contrapositive. If  $\mathbf{G}$  has a back edge: Directed cycle! Not a DAG.



# Characterizing DAGs

## Theorem

A directed graph  $\mathbf{G}$  is a DAG if and only if  $\text{DFS}(\mathbf{G})$  has no back edges.

## Proof.

Only if ( $\Rightarrow$ ): contrapositive. If  $\mathbf{G}$  has a back edge: Directed cycle! Not a DAG.

If ( $\Leftarrow$ ): contrapositive. If  $\mathbf{G}$  has a directed cycle  $\mathbf{C}$ :

# Characterizing DAGs

## Theorem

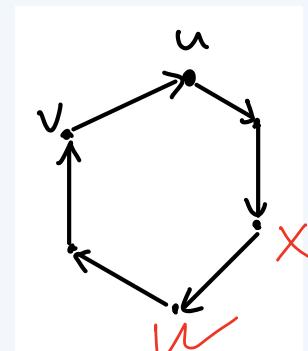
A directed graph  $\mathbf{G}$  is a DAG if and only if  $\text{DFS}(\mathbf{G})$  has no back edges.

## Proof.

Only if ( $\Rightarrow$ ): contrapositive. If  $\mathbf{G}$  has a back edge: Directed cycle! Not a DAG.

If ( $\Leftarrow$ ): contrapositive. If  $\mathbf{G}$  has a directed cycle  $\mathbf{C}$ :

- ▶ Let  $u \in \mathbf{C}$  with minimum start value,  $v$  predecessor in cycle
- ▶ All nodes in  $\mathbf{C}$  reachable from  $u \implies$  all nodes in  $\mathbf{C}$  descendants of  $u$
- ▶  $(v, u)$  a back edge



□

# Topological Sort Analysis

**Correctness:**

# Topological Sort Analysis

**Correctness:** Since  $G$  a DAG, never see back edge

# Topological Sort Analysis

**Correctness:** Since  $G$  a DAG, never see back edge

- ⇒ Every edge  $(v, u)$  out of  $v$  a forward or cross edge
- ⇒  $\text{finish}(u) < \text{finish}(v)$
- ⇒  $u$  already in list when  $v$  added to beginning



# Topological Sort Analysis

**Correctness:** Since  $G$  a DAG, never see back edge

- ⇒ Every edge  $(v, u)$  out of  $v$  a forward or cross edge
- ⇒  $\text{finish}(u) < \text{finish}(v)$
- ⇒  $u$  already in list when  $v$  added to beginning

**Running Time:**

# Topological Sort Analysis

**Correctness:** Since  $G$  a DAG, never see back edge

⇒ Every edge  $(v, u)$  out of  $v$  a forward or cross edge

⇒  $\text{finish}(u) < \text{finish}(v)$

⇒  $u$  already in list when  $v$  added to beginning

**Running Time:** Same as DFS!  $O(m + n)$

# Strongly Connected Components (SCC)

# Definitions

Another application of DFS. “Kosaraju’s Algorithm”: Developed by Rao Kosaraju, professor emeritus at JHU CS!

$G = (V, E)$  a directed graph.

## Definition

$C \subseteq V$  is a *strongly connected component (SCC)* if it is a *maximal* subset such that for all  $u, v \in C$ ,  $u$  can reach  $v$  and vice versa (bireachable).

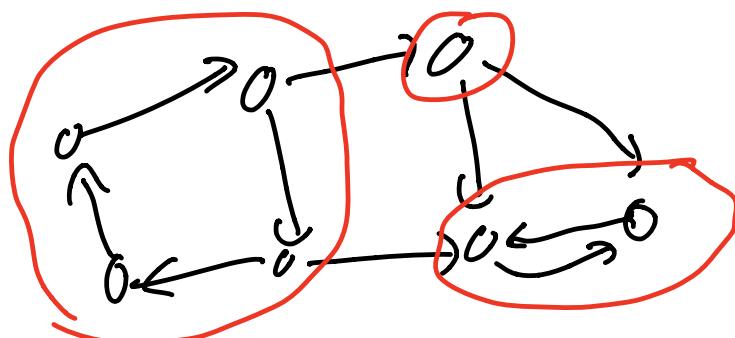
# Definitions

Another application of DFS. “Kosaraju’s Algorithm”: Developed by Rao Kosaraju, professor emeritus at JHU CS!

$G = (V, E)$  a directed graph.

## Definition

$C \subseteq V$  is a *strongly connected component (SCC)* if it is a *maximal* subset such that for all  $u, v \in C$ ,  $u$  can reach  $v$  and vice versa (bireachable).



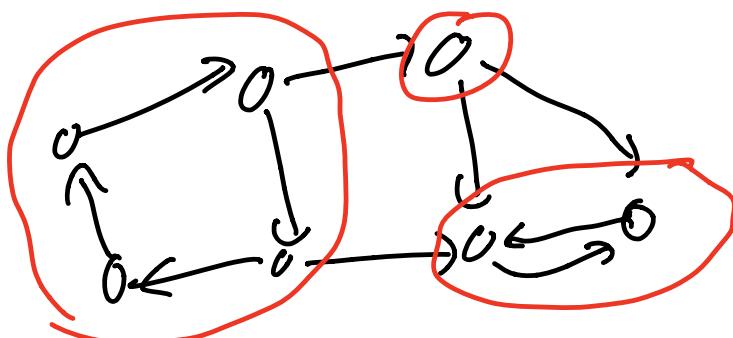
# Definitions

Another application of DFS. “Kosaraju’s Algorithm”: Developed by Rao Kosaraju, professor emeritus at JHU CS!

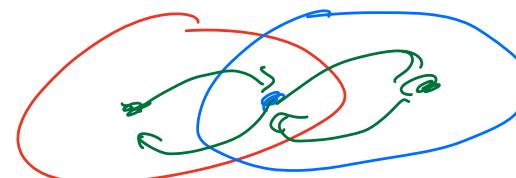
$G = (V, E)$  a directed graph.

## Definition

$C \subseteq V$  is a *strongly connected component (SCC)* if it is a *maximal* subset such that for all  $u, v \in C$ ,  $u$  can reach  $v$  and vice versa (bireachable).



**Fact:** There is a *unique* partition of  $V$  into SCCs



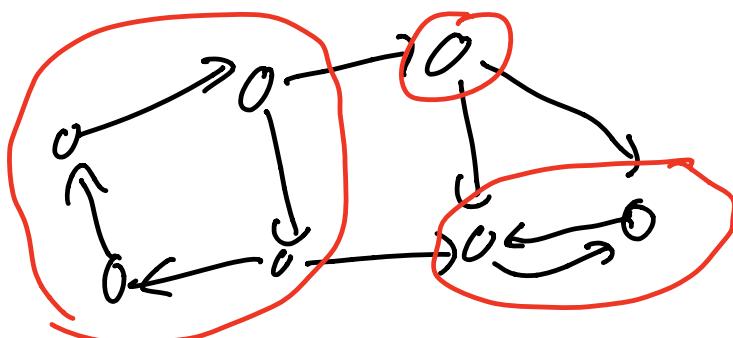
# Definitions

Another application of DFS. “Kosaraju’s Algorithm”: Developed by Rao Kosaraju, professor emeritus at JHU CS!

$G = (V, E)$  a directed graph.

## Definition

$C \subseteq V$  is a *strongly connected component (SCC)* if it is a *maximal* subset such that for all  $u, v \in C$ ,  $u$  can reach  $v$  and vice versa (bireachable).



**Fact:** There is a *unique* partition of  $V$  into SCCs

**Proof:** Bireachability is an equivalence relation: if  $u$  and  $v$  are bireachable, and  $v$  and  $w$  are bireachable, then  $u$  and  $w$  are bireachable.

# SCC Problem

**Problem:** Given  $G$ , compute SCCs (partition  $V$  into the SCCs)

# SCC Problem

**Problem:** Given  $G$ , compute SCCs (partition  $V$  into the SCCs)

**Trivial Algorithm:**

# SCC Problem

**Problem:** Given  $G$ , compute SCCs (partition  $V$  into the SCCs)

**Trivial Algorithm:** DFS/BFS from every node, keep track of what's reachable from where

# SCC Problem

**Problem:** Given  $G$ , compute SCCs (partition  $V$  into the SCCs)

**Trivial Algorithm:** DFS/BFS from every node, keep track of what's reachable from where

- ▶ Running time:  $O(n(m + n))$

# SCC Problem

**Problem:** Given  $G$ , compute SCCs (partition  $V$  into the SCCs)

**Trivial Algorithm:** DFS/BFS from every node, keep track of what's reachable from where

- ▶ Running time:  $O(n(m + n))$

Can we do better?  $O(m + n)$ ?

# Graph of SCCs

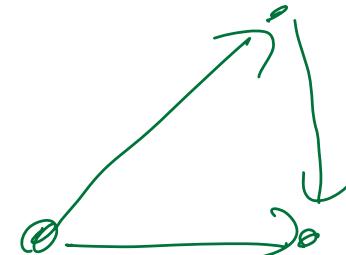
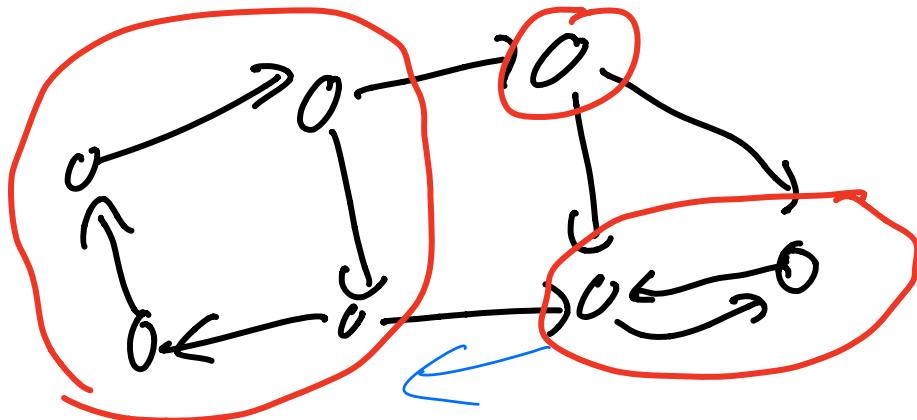
**Definition:** Let  $\hat{G}$  be graph of SCCs:

- ▶ Vertex  $v(C)$  for each SCC  $C$
- ▶ Edge  $(v(C), v(C'))$  if  $\exists u \in C, v \in C'$  such that  $(u, v) \in E$

# Graph of SCCs

**Definition:** Let  $\hat{G}$  be graph of SCCs:

- ▶ Vertex  $v(C)$  for each SCC  $C$
- ▶ Edge  $(v(C), v(C'))$  if  $\exists u \in C, v \in C'$  such that  $(u, v) \in E$



# Graph of SCCs: Structure

Theorem

$\hat{G}$  is a DAG.

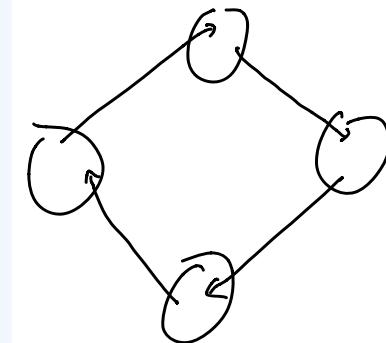
# Graph of SCCs: Structure

Theorem

$\hat{G}$  is a DAG.

Proof.

Suppose  $\hat{G}$  not a DAG. Then there is a directed cycle  $H$ .



# Graph of SCCs: Structure

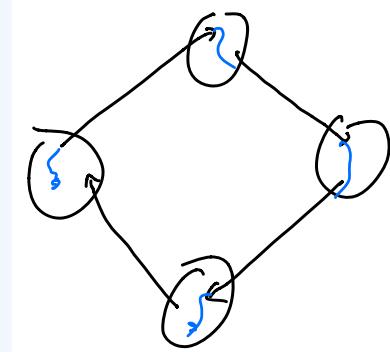
Theorem

$\hat{G}$  is a DAG.

Proof.

Suppose  $\hat{G}$  not a DAG. Then there is a directed cycle  $H$ .

$\implies \bigcup_{C:v(C) \in H} C$  is an SCC



# Graph of SCCs: Structure

Theorem

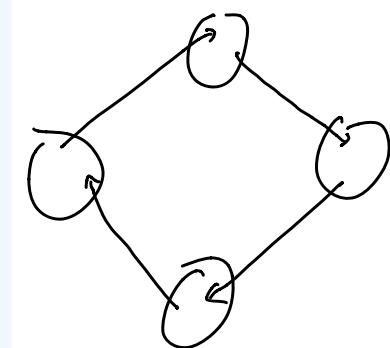
$\hat{G}$  is a DAG.

Proof.

Suppose  $\hat{G}$  not a DAG. Then there is a directed cycle  $H$ .

$\implies \bigcup_{C:v(C) \in H} C$  is an SCC

$\implies v(C)$  not an SCC for  $v(C) \in H$



# Graph of SCCs: Structure

Theorem

$\hat{G}$  is a DAG.

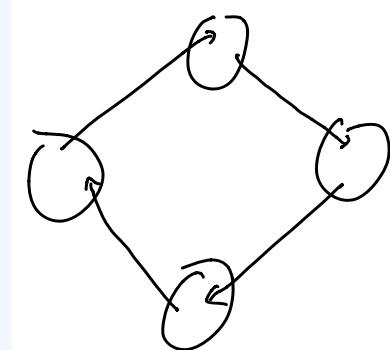
Proof.

Suppose  $\hat{G}$  not a DAG. Then there is a directed cycle  $H$ .

$\implies \bigcup_{C:v(C) \in H} C$  is an SCC

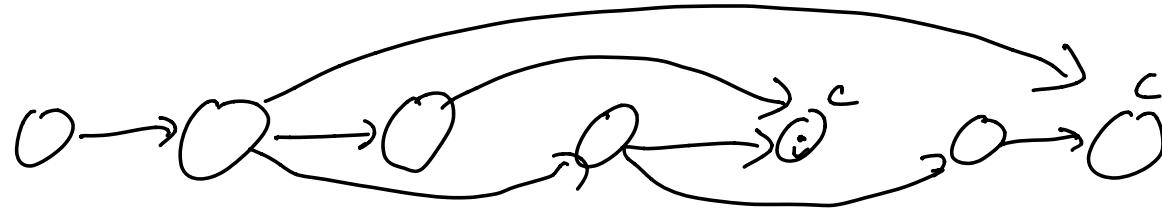
$\implies v(C)$  not an SCC for  $v(C) \in H$

Contradiction!



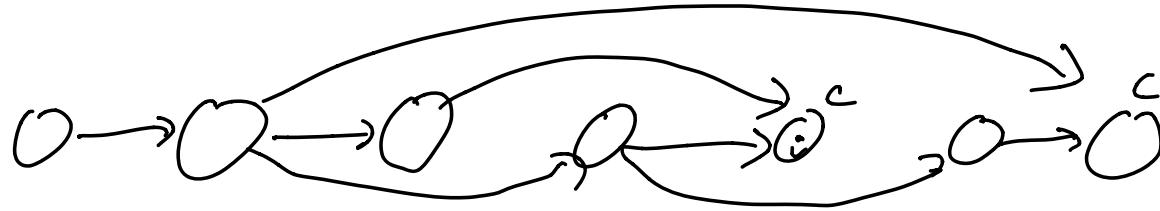
# Sink SCC

Since  $\hat{G}$  a DAG, has a topological sort



# Sink SCC

Since  $\hat{G}$  a DAG, has a topological sort

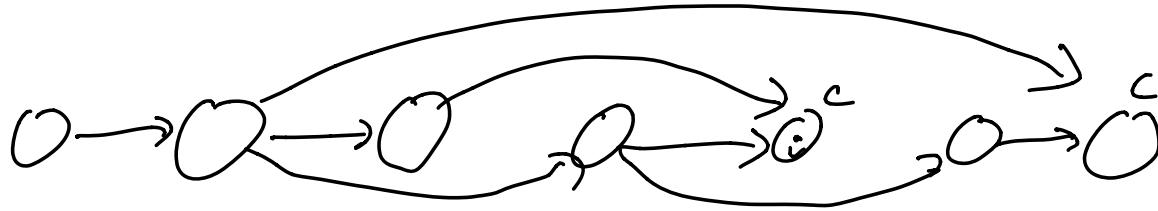


**Definition:** SCC  $C$  is a *sink* SCC if no outgoing edges, *source* SCC if no incoming edges

- ▶ Claim: At least one sink SCC exists

# Sink SCC

Since  $\hat{G}$  a DAG, has a topological sort

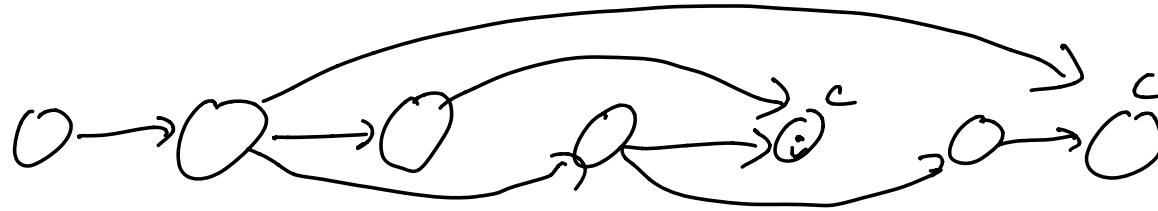


**Definition:** SCC  $C$  is a *sink* SCC if no outgoing edges, *source* SCC if no incoming edges

- ▶ Claim: At least one sink SCC exists
- ▶ Proof: Final SCC in topological sort of  $\hat{G}$  must be a sink.

# Sink SCC

Since  $\hat{G}$  a DAG, has a topological sort



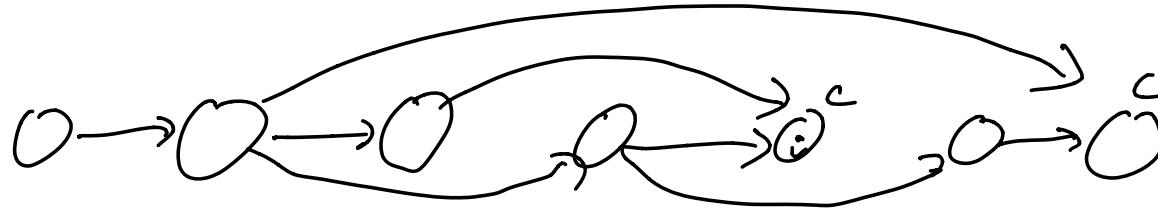
**Definition:** SCC  $C$  is a *sink* SCC if no outgoing edges, *source* SCC if no incoming edges

- ▶ Claim: At least one sink SCC exists
- ▶ Proof: Final SCC in topological sort of  $\hat{G}$  must be a sink.

What happens if we run  $\text{DFS}(v)$  where  $v$  in a sink SCC?

# Sink SCC

Since  $\hat{G}$  a DAG, has a topological sort



**Definition:** SCC  $C$  is a *sink* SCC if no outgoing edges, *source* SCC if no incoming edges

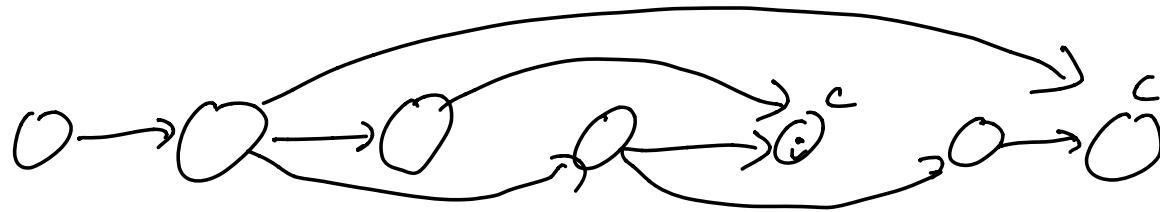
- ▶ Claim: At least one sink SCC exists
- ▶ Proof: Final SCC in topological sort of  $\hat{G}$  must be a sink.

What happens if we run  $\text{DFS}(v)$  where  $v$  in a sink SCC?

- ▶ See exactly nodes in  $C$ !

# Sink SCC

Since  $\hat{G}$  a DAG, has a topological sort



**Definition:** SCC  $C$  is a *sink* SCC if no outgoing edges, *source* SCC if no incoming edges

- ▶ Claim: At least one sink SCC exists
  - ▶ Proof: Final SCC in topological sort of  $\hat{G}$  must be a sink.

What happens if we run  $\text{DFS}(v)$  where  $v$  in a sink SCC?

- ▶ See exactly nodes in  $C$ !

Strategy: find node in sink SCC, run DFS, remove nodes found, repeat

# SCCs and DFS

Run  $\text{DFS}(\mathbf{G})$ , and let  $\text{finish}(C) = \max_{v \in C} \text{finish}(v)$

## Lemma

Let  $C_1, C_2$  distinct SCCs s.t.  $(v(C_1), v(C_2)) \in E(\hat{\mathbf{G}})$ . Then  $\text{finish}(C_1) > \text{finish}(C_2)$ .



# SCCs and DFS

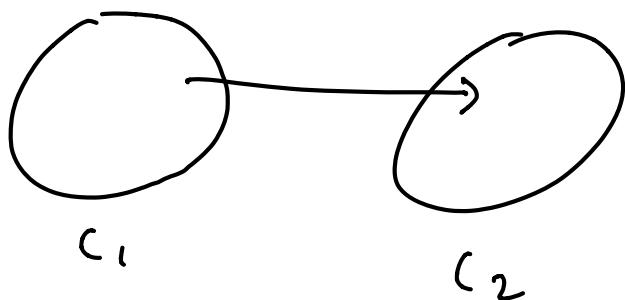
Run  $\text{DFS}(\mathbf{G})$ , and let  $\text{finish}(C) = \max_{v \in C} \text{finish}(v)$

## Lemma

Let  $C_1, C_2$  distinct SCCs s.t.  $(v(C_1), v(C_2)) \in E(\hat{\mathbf{G}})$ . Then  $\text{finish}(C_1) > \text{finish}(C_2)$ .

## Proof.

Let  $x \in C_1 \cup C_2$  be first node encountered by DFS



# SCCs and DFS

Run  $\text{DFS}(\mathbf{G})$ , and let  $\text{finish}(C) = \max_{v \in C} \text{finish}(v)$

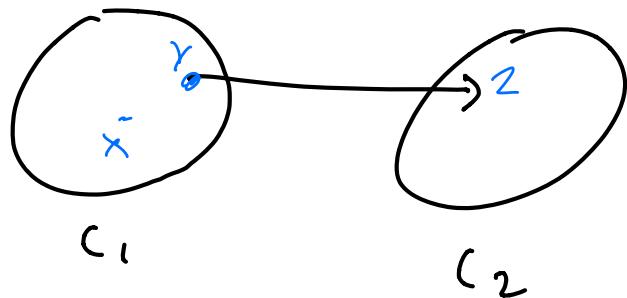
## Lemma

Let  $C_1, C_2$  distinct SCCs s.t.  $(v(C_1), v(C_2)) \in E(\hat{\mathbf{G}})$ . Then  $\text{finish}(C_1) > \text{finish}(C_2)$ .

## Proof.

Let  $x \in C_1 \cup C_2$  be first node encountered by DFS

► If  $x \in C_1$ :



# SCCs and DFS

Run  $\text{DFS}(\mathbf{G})$ , and let  $\text{finish}(C) = \max_{v \in C} \text{finish}(v)$

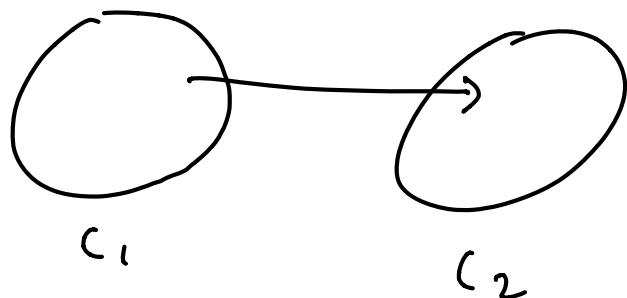
## Lemma

Let  $C_1, C_2$  distinct SCCs s.t.  $(v(C_1), v(C_2)) \in E(\hat{\mathbf{G}})$ . Then  $\text{finish}(C_1) > \text{finish}(C_2)$ .

## Proof.

Let  $x \in C_1 \cup C_2$  be first node encountered by DFS

- ▶ If  $x \in C_1$ : all of  $C_2$  reachable from  $x$ , so  $\text{DFS}(x)$  does not complete until all of  $C_2$  finished



# SCCs and DFS

Run  $\text{DFS}(\mathbf{G})$ , and let  $\text{finish}(C) = \max_{v \in C} \text{finish}(v)$

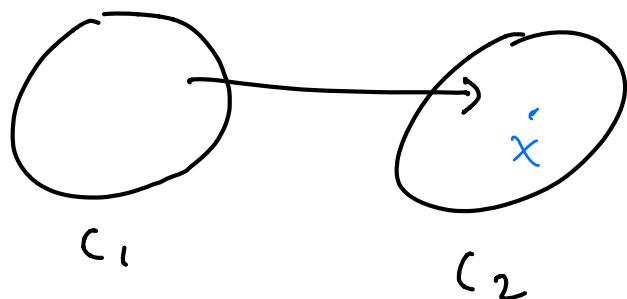
## Lemma

Let  $C_1, C_2$  distinct SCCs s.t.  $(v(C_1), v(C_2)) \in E(\hat{\mathbf{G}})$ . Then  $\text{finish}(C_1) > \text{finish}(C_2)$ .

## Proof.

Let  $x \in C_1 \cup C_2$  be first node encountered by DFS

- ▶ If  $x \in C_1$ : all of  $C_2$  reachable from  $x$ , so  $\text{DFS}(x)$  does not complete until all of  $C_2$  finished
- ▶ If  $x \in C_2$ :



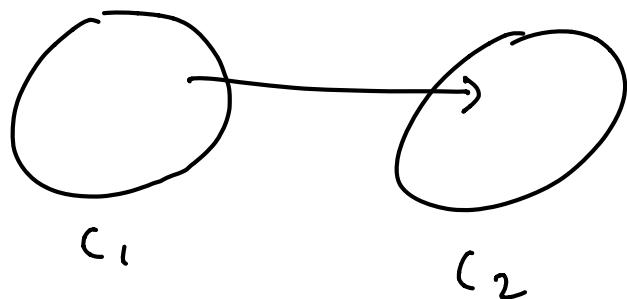
# SCCs and DFS

Run  $\text{DFS}(\mathbf{G})$ , and let  $\text{finish}(C) = \max_{v \in C} \text{finish}(v)$

## Lemma

Let  $C_1, C_2$  distinct SCCs s.t.  $(v(C_1), v(C_2)) \in E(\hat{\mathbf{G}})$ . Then  $\text{finish}(C_1) > \text{finish}(C_2)$ .

## Proof.



Let  $x \in C_1 \cup C_2$  be first node encountered by DFS

- ▶ If  $x \in C_1$ : all of  $C_2$  reachable from  $x$ , so  $\text{DFS}(x)$  does not complete until all of  $C_2$  finished
- ▶ If  $x \in C_2$ : all of  $C_2$  reachable from  $x$  but nothing from  $C_1$ , so all of  $C_2$  finishes before any node in  $C_1$  starts

□

# SCCs and DFS

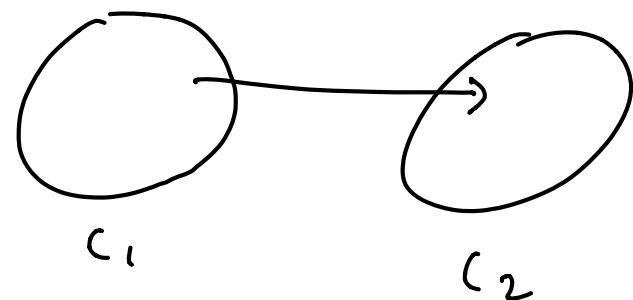
Run  $\text{DFS}(\mathbf{G})$ , and let  $\text{finish}(C) = \max_{v \in C} \text{finish}(v)$



## Lemma

Let  $C_1, C_2$  distinct SCCs s.t.  $(v(C_1), v(C_2)) \in E(\hat{\mathbf{G}})$ . Then  $\text{finish}(C_1) > \text{finish}(C_2)$ .

## Proof.



Let  $x \in C_1 \cup C_2$  be first node encountered by DFS

- ▶ If  $x \in C_1$ : all of  $C_2$  reachable from  $x$ , so  $\text{DFS}(x)$  does not complete until all of  $C_2$  finished
- ▶ If  $x \in C_2$ : all of  $C_2$  reachable from  $x$  but nothing from  $C_1$ , so all of  $C_2$  finishes before any node in  $C_1$  starts

□

So node of *max* finishing time in a *source SCC* (no incoming edges in  $\hat{\mathbf{G}}$ ).

## Useful Corollary

Run DFS( $\mathbf{G}$ ) to get finish times.

### Corollary

Let  $\mathcal{C}$  be collection of all SCCs of  $\mathbf{G}$ , and let  $\mathcal{C}' \subseteq \mathcal{C}$ . Let  $\mathbf{G}' = \mathbf{G} \setminus (\bigcup_{C \in \mathcal{C} \setminus \mathcal{C}'} C)$ . Then the node  $v = \operatorname{argmax}_{u \in \bigcup_{C \in \mathcal{C} \setminus \mathcal{C}'} C} \mathbf{finish}(u)$  is in an SCC of  $\mathbf{G}$  that is a source SCC of  $\mathbf{G}'$ .

$\overset{\wedge}{\mathcal{C}}$   $\overset{\wedge}{\mathcal{C}} \cup \mathcal{C}'$   $\overset{\wedge}{v}$   $\overset{\wedge}{\mathbf{finish}}$

# Useful Corollary

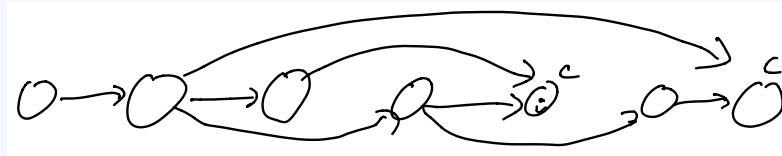
Run DFS( $\mathbf{G}$ ) to get finish times.

## Corollary

Let  $\mathcal{C}$  be collection of all SCCs of  $\mathbf{G}$ , and let  $\mathcal{C}' \subseteq \mathcal{C}$ . Let  $\mathbf{G}' = \mathbf{G} \setminus (\bigcup_{C \in \mathcal{C} \setminus \mathcal{C}'} C)$ . Then the node  $v = \operatorname{argmax}_{u \in \bigcup_{C \in \mathcal{C} \setminus \mathcal{C}'} C} \mathbf{finish}(u)$  is in an SCC of  $\mathbf{G}$  that is a source SCC of  $\mathbf{G}'$ .

## Proof Sketch.

Clearly SCCs of  $\mathbf{G}'$  are precisely  $\mathcal{C} \setminus \mathcal{C}'$ :



# Useful Corollary

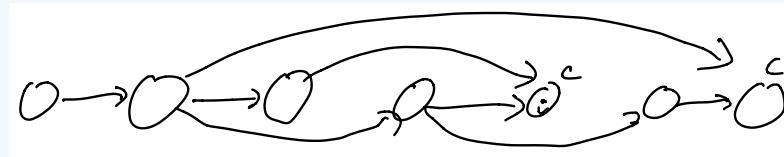
Run DFS( $\mathbf{G}$ ) to get finish times.

## Corollary

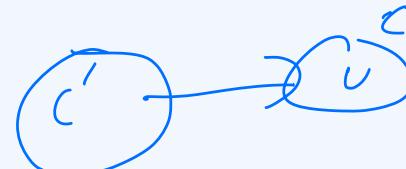
Let  $\mathcal{C}$  be collection of all SCCs of  $\mathbf{G}$ , and let  $\mathcal{C}' \subseteq \mathcal{C}$ . Let  $\mathbf{G}' = \mathbf{G} \setminus (\bigcup_{C \in \mathcal{C} \setminus \mathcal{C}'} C)$ . Then the node  $v = \operatorname{argmax}_{u \in \bigcup_{C \in \mathcal{C} \setminus \mathcal{C}'} C} \mathbf{finish}(u)$  is in an SCC of  $\mathbf{G}$  that is a source SCC of  $\mathbf{G}'$ .

## Proof Sketch.

Clearly SCCs of  $\mathbf{G}'$  are precisely  $\mathcal{C} \setminus \mathcal{C}'$ :



Suppose  $v$  in SCC  $C$  not source SCC of  $\mathbf{G}'$



# Useful Corollary

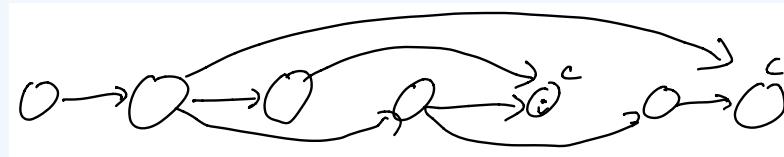
Run DFS( $\mathbf{G}$ ) to get finish times.

## Corollary

Let  $\mathcal{C}$  be collection of all SCCs of  $\mathbf{G}$ , and let  $\mathcal{C}' \subseteq \mathcal{C}$ . Let  $\mathbf{G}' = \mathbf{G} \setminus (\bigcup_{C \in \mathcal{C} \setminus \mathcal{C}'} C)$ . Then the node  $v = \operatorname{argmax}_{u \in \bigcup_{C \in \mathcal{C} \setminus \mathcal{C}'} C} \operatorname{finish}(u)$  is in an SCC of  $\mathbf{G}$  that is a source SCC of  $\mathbf{G}'$ .

## Proof Sketch.

Clearly SCCs of  $\mathbf{G}'$  are precisely  $\mathcal{C} \setminus \mathcal{C}'$ :



Suppose  $v$  in SCC  $C$  not source SCC of  $\mathbf{G}'$

$\implies$  exists edge  $(u', v')$  in  $\mathbf{G}'$  where  $v' \in C$  but  $u'$  in different SCC  $C'$

# Useful Corollary

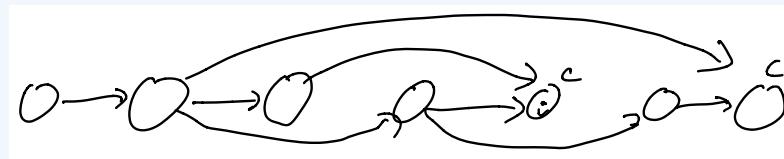
Run DFS( $\mathbf{G}$ ) to get finish times.

## Corollary

Let  $\mathcal{C}$  be collection of all SCCs of  $\mathbf{G}$ , and let  $\mathcal{C}' \subseteq \mathcal{C}$ . Let  $\mathbf{G}' = \mathbf{G} \setminus (\bigcup_{C \in \mathcal{C} \setminus \mathcal{C}'} C)$ . Then the node  $v = \operatorname{argmax}_{u \in \bigcup_{C \in \mathcal{C} \setminus \mathcal{C}'} C} \mathbf{finish}(u)$  is in an SCC of  $\mathbf{G}$  that is a source SCC of  $\mathbf{G}'$ .

## Proof Sketch.

Clearly SCCs of  $\mathbf{G}'$  are precisely  $\mathcal{C} \setminus \mathcal{C}'$ :



Suppose  $v$  in SCC  $C$  not source SCC of  $\mathbf{G}'$

$\implies$  exists edge  $(u', v')$  in  $\mathbf{G}'$  where  $v' \in C$  but  $u'$  in different SCC  $C'$

Lemma  $\implies \mathbf{finish}(C') > \mathbf{finish}(C)$ , contradiction to def of  $v$ .

□

# Kosaraju's Algorithm

## Kosaraju's Algorithm

So node with max finish time in a *source* SCC (no incoming edges in  $\hat{G}$ ). Want sink (no outgoing edges).

## Kosaraju's Algorithm

So node with max finish time in a *source* SCC (no incoming edges in  $\hat{G}$ ). Want sink (no outgoing edges). Reverse all edges!

## Kosaraju's Algorithm

So node with max finish time in a *source* SCC (no incoming edges in  $\hat{G}$ ). Want sink (no outgoing edges). Reverse all edges!

**Definition:**  $G^T$  is  $G$  with all edges reversed.

- ▶ Source SCC in  $G^T$  is sink SCC in  $G$

# Kosaraju's Algorithm

So node with max finish time in a *source* SCC (no incoming edges in  $\hat{G}$ ). Want sink (no outgoing edges). Reverse all edges!

**Definition:**  $G^T$  is  $G$  with all edges reversed.

- ▶ Source SCC in  $G^T$  is sink SCC in  $G$

Kosaraju's Algorithm:

- ▶  $\text{DFS}(G^T)$  to get finishing times and order  $\pi$  on  $V$  from largest finishing time to smallest
- ▶ Set  $\text{mark}(v) = \text{False}$  for all  $v \in V$
- ▶ Forall  $v \in V$  in order of  $\pi$  {
  - if  $\text{mark}(v) = \text{False}$  {
    - Run  $\text{DFS}(v)$ , let  $C$  be all nodes found
    - Return  $C$  as an SCC
- }

# Kosaraju's Algorithm

So node with max finish time in a *source* SCC (no incoming edges in  $\hat{G}$ ). Want sink (no outgoing edges). Reverse all edges!

**Definition:**  $G^T$  is  $G$  with all edges reversed.

- ▶ Source SCC in  $G^T$  is sink SCC in  $G$

Kosaraju's Algorithm:

- ▶ DFS( $G^T$ ) to get finishing times and order  $\pi$  on  $V$  from largest finishing time to smallest
- ▶ Set  $\text{mark}(v) = \text{False}$  for all  $v \in V$
- ▶ Forall  $v \in V$  in order of  $\pi$  {
  - if  $\text{mark}(v) = \text{False}$  {
    - Run DFS( $v$ ), let  $C$  be all nodes found
    - Return  $C$  as an SCC
- }
- }

**Running Time:**

Run DFS( $v$ ), let  $C$  be all nodes found

Return  $C$  as an SCC

}

}

# Kosaraju's Algorithm

So node with max finish time in a *source* SCC (no incoming edges in  $\hat{G}$ ). Want sink (no outgoing edges). Reverse all edges!

**Definition:**  $G^T$  is  $G$  with all edges reversed.

- ▶ Source SCC in  $G^T$  is sink SCC in  $G$

Kosaraju's Algorithm:

- ▶ DFS( $G^T$ ) to get finishing times and order  $\pi$  on  $V$  from largest finishing time to smallest
- ▶ Set  $\text{mark}(v) = \text{False}$  for all  $v \in V$
- ▶ Forall  $v \in V$  in order of  $\pi$  {
  - if  $\text{mark}(v) = \text{False}$  {
    - Run DFS( $v$ ), let  $C$  be all nodes found
    - Return  $C$  as an SCC
- }
- }

**Running Time:**  $O(m + n)$

# Correctness Sketch

Let  $C_1, C_2, \dots, C_k$  be sets identified by algorithm (in order)

## Theorem

$C_i$  is a sink SCC of  $G \setminus \left( \bigcup_{j=1}^{i-1} C_j \right)$ , and an SCC of  $G$ .

# Correctness Sketch

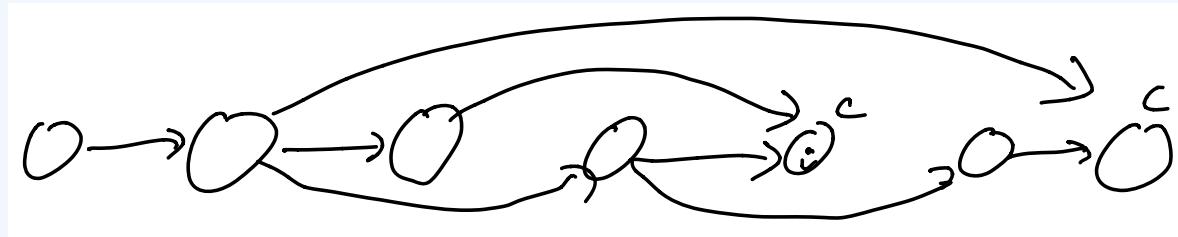
Let  $C_1, C_2, \dots, C_k$  be sets identified by algorithm (in order)

## Theorem

$C_i$  is a sink SCC of  $G \setminus \left(\bigcup_{j=1}^{i-1} C_j\right)$ , and an SCC of  $G$ .

## Proof Sketch.

Induction on  $i$ .



# Correctness Sketch

Let  $C_1, C_2, \dots, C_k$  be sets identified by algorithm (in order)

## Theorem

$C_i$  is a sink SCC of  $G \setminus \left( \bigcup_{j=1}^{i-1} C_j \right)$ , and an SCC of  $G$ .

## Proof Sketch.

Induction on  $i$ .

**Base case:**  $i = 1$ . By previous lemma, largest finishing time in  $G^T \implies$  in sink SCC of  $G$   $\implies C_1$  is sink SCC of  $G$

# Correctness Sketch

Let  $C_1, C_2, \dots, C_k$  be sets identified by algorithm (in order)

## Theorem

$C_i$  is a sink SCC of  $G \setminus \left( \bigcup_{j=1}^{i-1} C_j \right)$ , and an SCC of  $G$ .

## Proof Sketch.

Induction on  $i$ .

**Base case:**  $i = 1$ . By previous lemma, largest finishing time in  $G^T \implies$  in sink SCC of  $G \implies C_1$  is sink SCC of  $G$

**Inductive case:** Let  $i > 1$ . Let  $v$  unmarked node with largest finishing time.

- ▶ By induction, subgraph of unmarked nodes is  $G$  minus  $i - 1$  SCCs of  $G$
- ▶ Corollary  $\implies v$  must be in sink SCC of unmarked nodes so get an SCC of unmarked nodes when run DFS
- ▶ Corollary  $\implies$  SCC of original graph