# Decomposition of Graphs: Computing Strongly Connected Components

Daniel Kane

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

## Graph Algorithms
## Data Structures and Algorithms

## Learning Objectives

- Efficiently compute the strongly connected components of a directed graph.

# Last Time

- Connectivity in directed graphs.
- Strongly connected components.
- Metagraph.

# Problem

## Strongly Connected Components

Input: A directed graph $G$

Output: The strongly connected components of $G$.

# Easy Algorithm

**EasySCC($G$)**

```
for each vertex v:
  run explore(v) to determine
    vertices reachable from v
for each vertex v:
  find the u reachable from v that
    can also reach v
these are the SCCs
```

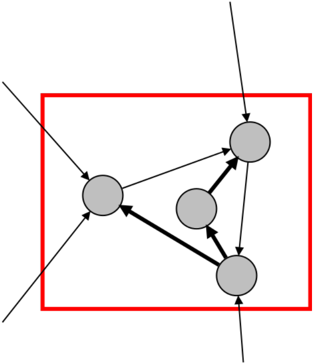
Runtime $O(|V|^2 + |V||E|)$. Want faster.

# Outline

1. Sink Components
2. Algorithm

# Sink Components

Idea: If $v$ is in a sink SCC, `explore`($v$) finds vertices reachable from $v$. This is exactly the SCC of $v$.



# Finding Sink Components

Need a way to find a sink SCC.

# Theorem

**Theorem**

If $\mathcal{C}$ and $\mathcal{C}'$ are two strongly connected components with an edge from some vertex of $\mathcal{C}$ to some vertex of $\mathcal{C}'$, then largest post in $\mathcal{C}$ bigger than largest post in $\mathcal{C}'$.

# Proof

Cases:

- Visit $\mathcal{C}$ before visit $\mathcal{C}'$
- Visit $\mathcal{C}'$ before visit $\mathcal{C}$

# Case I

Visit $\mathcal{C}$ first.

- Can reach everything in $\mathcal{C}'$ from $\mathcal{C}$.
- Explore all of $\mathcal{C}'$ while exploring $\mathcal{C}$.
- $\mathcal{C}$ has largest post.

# Case II

Visit $\mathcal{C}'$ first.

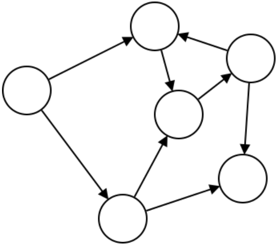
- Cannot reach $\mathcal{C}$ from $\mathcal{C}'$
- Must finish exploring $\mathcal{C}'$ before exploring $\mathcal{C}$
- $\mathcal{C}$ has largest post.

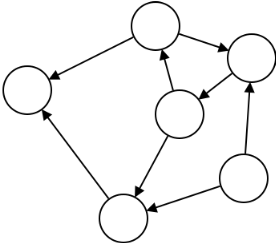
# Conclusion

The vertex with the largest postorder number is in a source component!
Problem: We wanted a sink component.

# Reverse Graph

Let $G^R$ be the graph obtained from $G$ by reversing all of the edges.





# Reverse Graph Components

- $G^R$ and $G$ have same SCCs.
- Source components of $G^R$ are sink components of $G$.

Find sink components of $G$ by running `DFS` on $G^R$.

# Problem

Which of the following is true?

- The vertex with largest postorder in $G^R$ is in a sink SCC of $G$.
- The vertex with the largest preorder in $G$ is in a sink SCC of $G$.
- The vertex with the smallest postorder in $G$ is in a sink SCC of $G$.

# Solution

Which of the following is true?

- The vertex with largest postorder in $G^R$ is in a sink SCC of $G$.
- The vertex with the largest preorder in $G$ is in a sink SCC of $G$.
- The vertex with the smallest postorder in $G$ is in a sink SCC of $G$.

# Outline

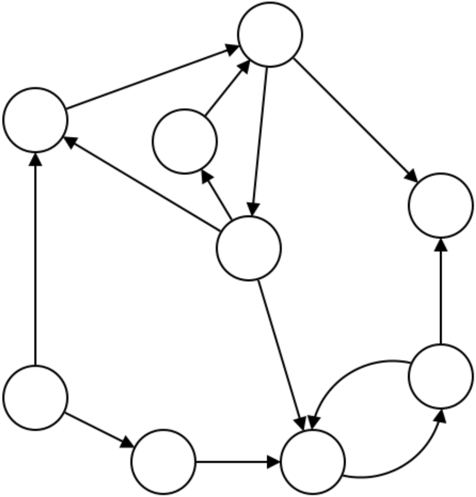
1. Sink Components
2. Algorithm

# Basic Algorithm

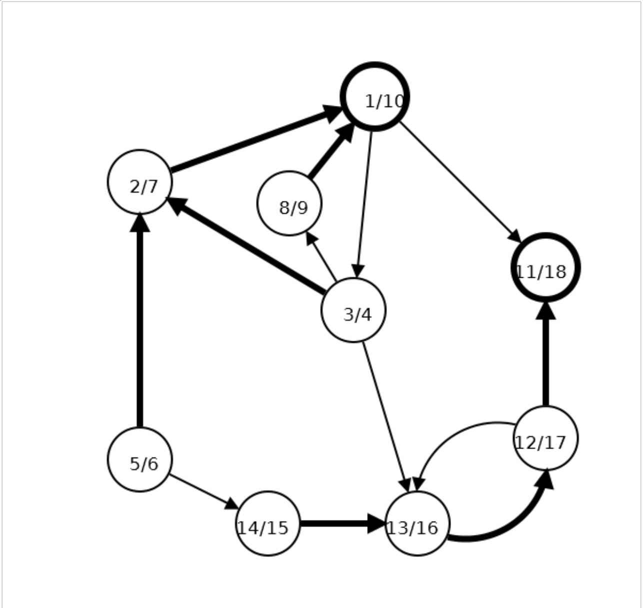
**SCCs($G$)**

run DFS($G^R$)
let $v$ have largest post number
run Explore($v$)
vertices found are first SCC
Remove from $G$ and repeat

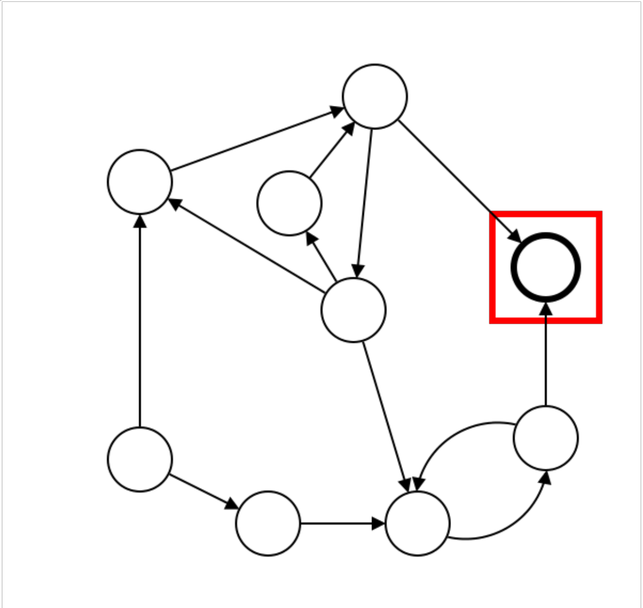
# Example



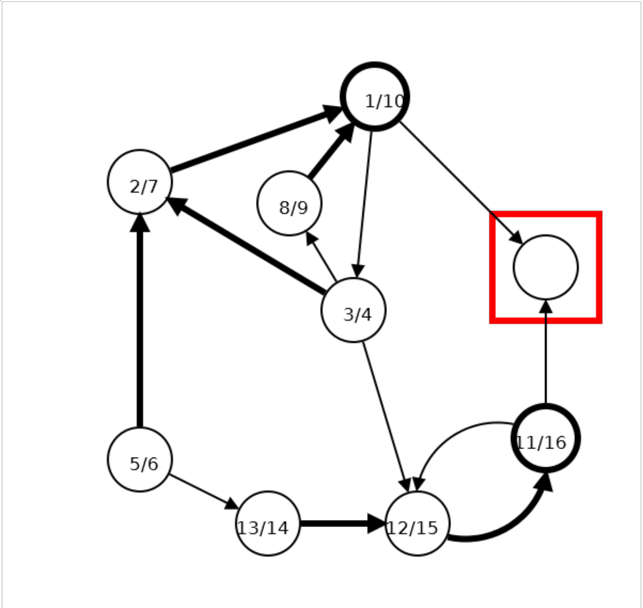
# Example



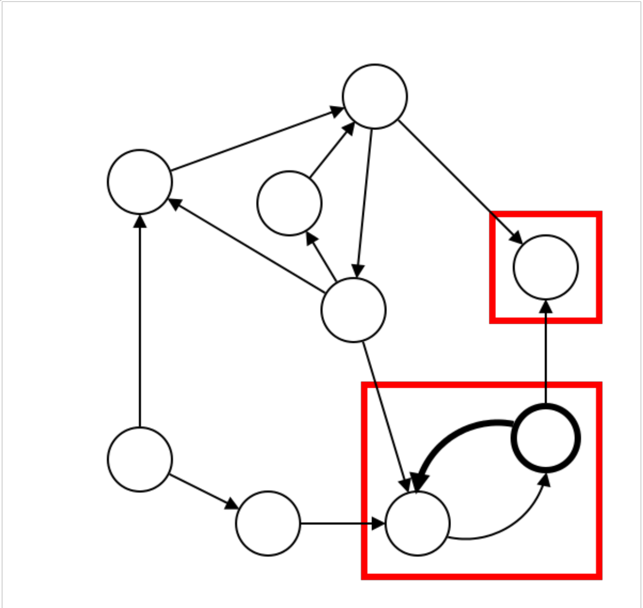
# Example



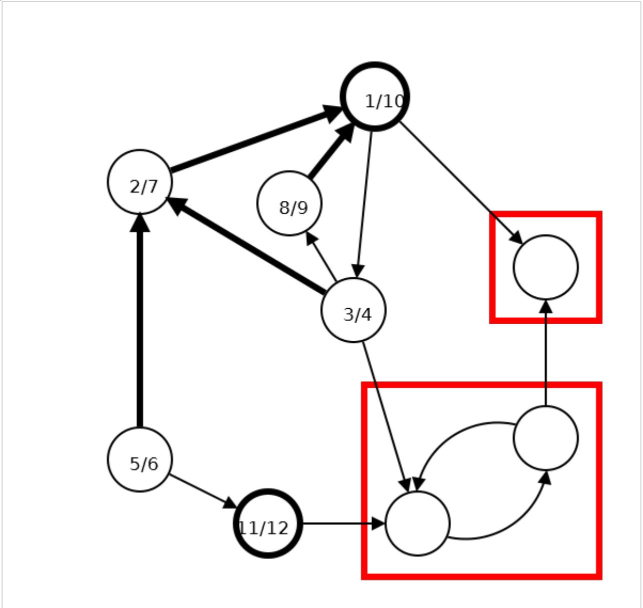
# Example



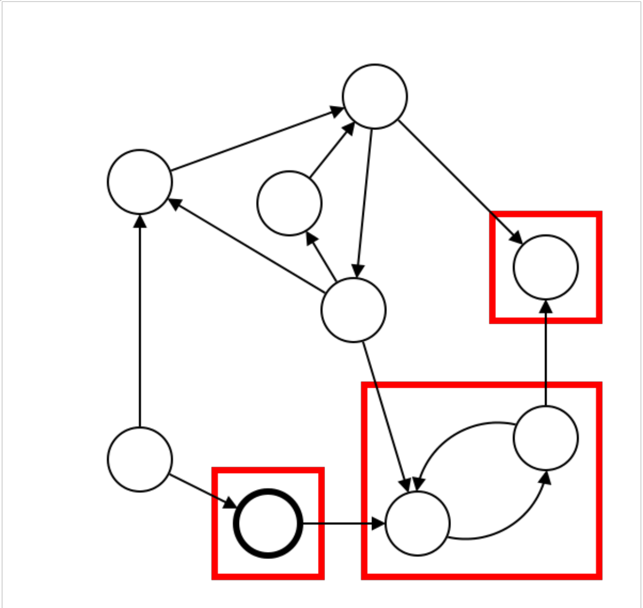
# Example



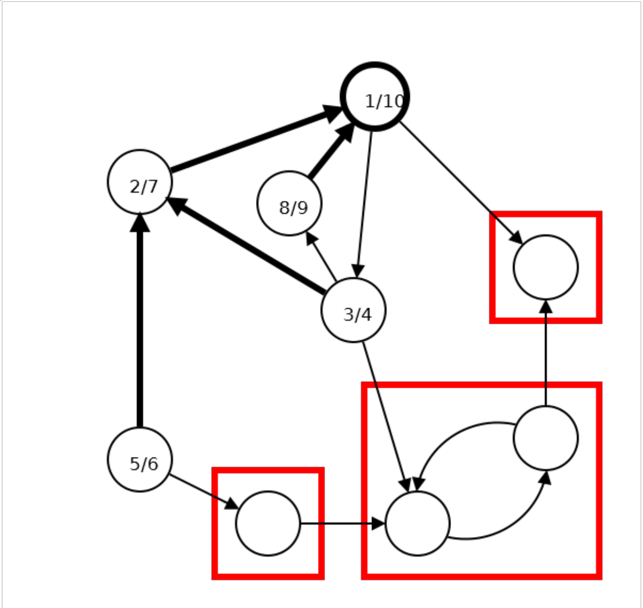
# Example



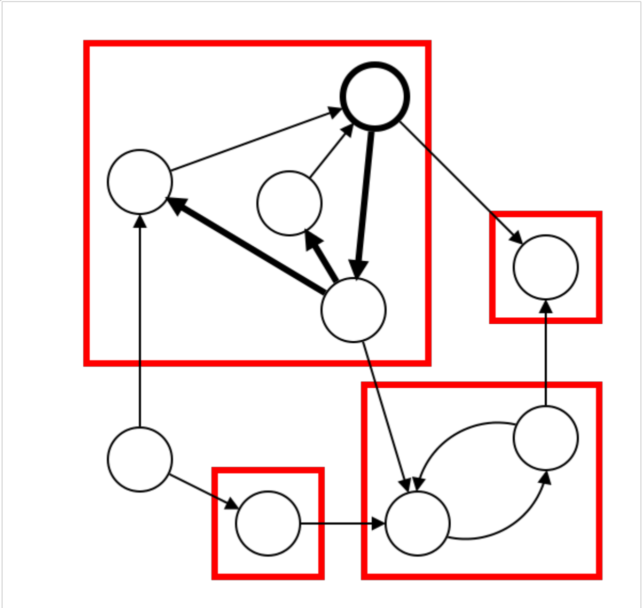
# Example



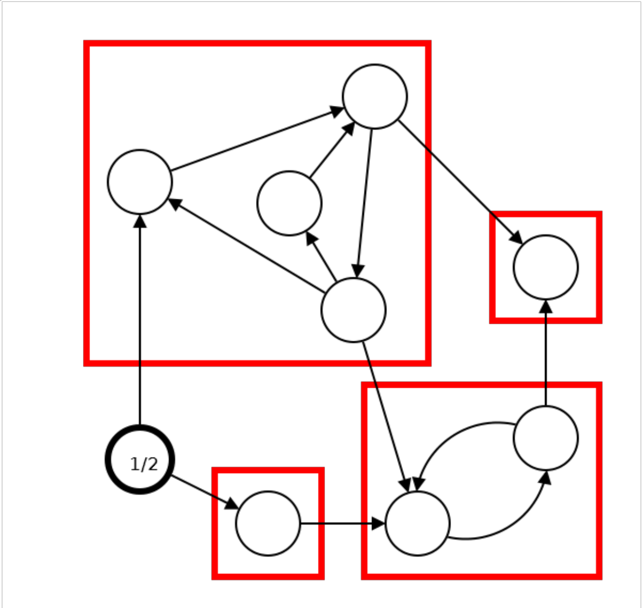
# Example



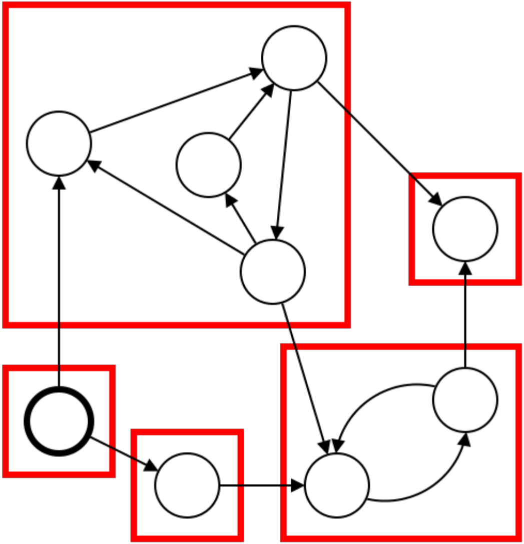
# Example



# Example



# Example



# Improvement

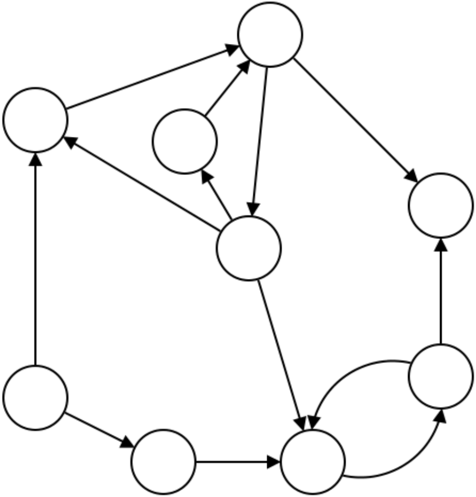
- Don't need to rerun `DFS` on $G^R$.
- Largest remaining post number comes from sink component.

# New Algorithm

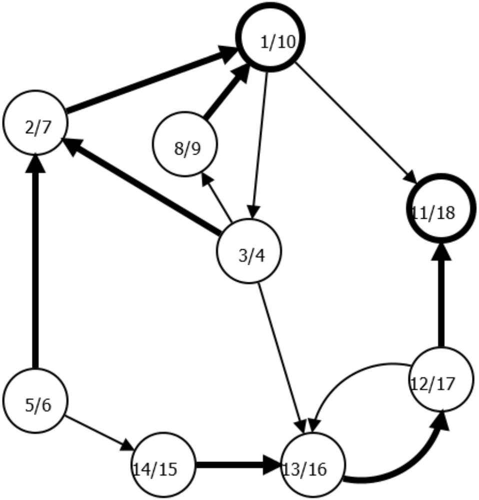
**SCCs($G$)**

```
Run DFS(G^R)
for v ∈ V in reverse postorder:
  if not visited(v):
    Explore(v)
    mark visited vertices
      as new SCC
```

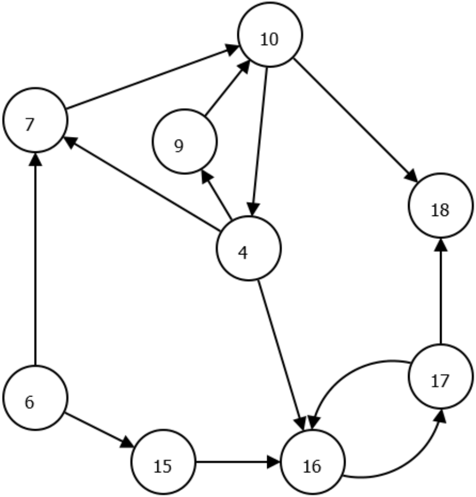
# Example



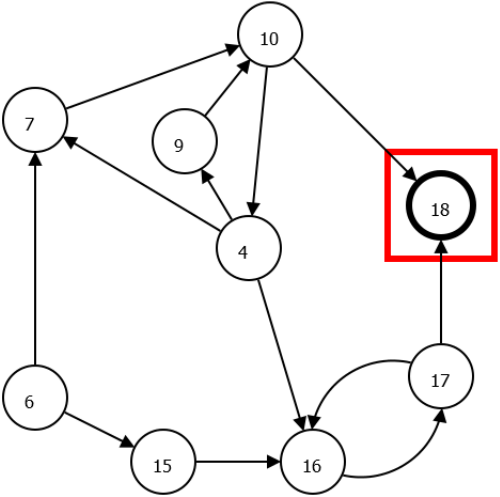
# Example



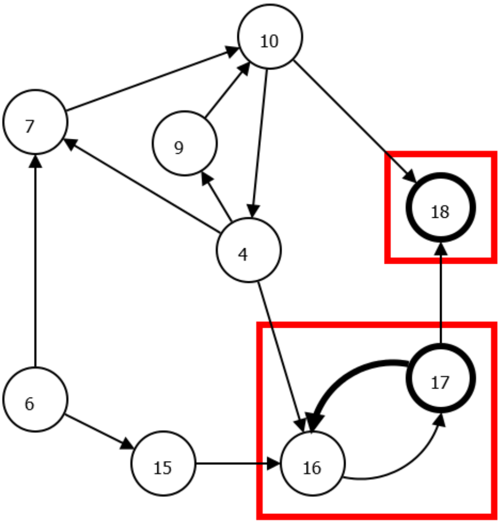
# Example



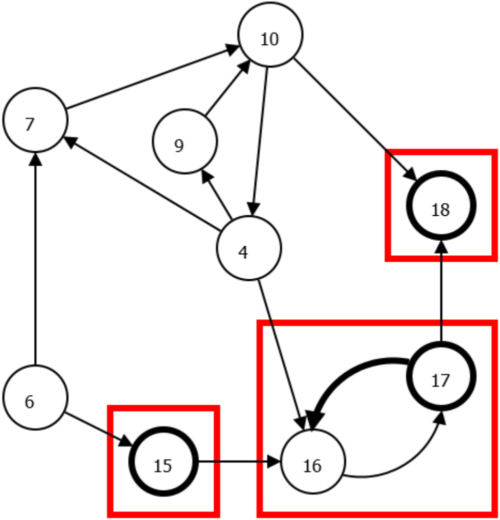
# Example



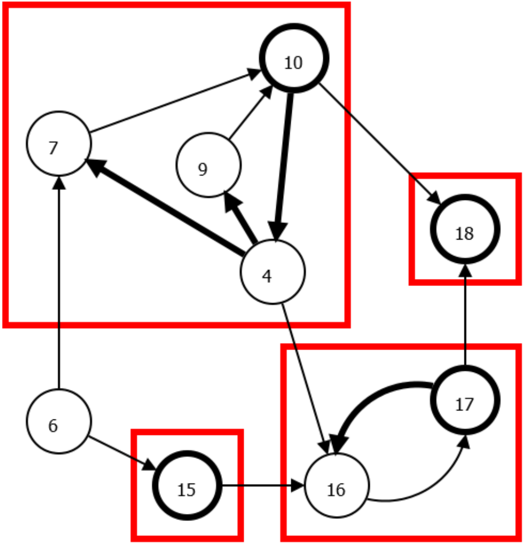
# Example



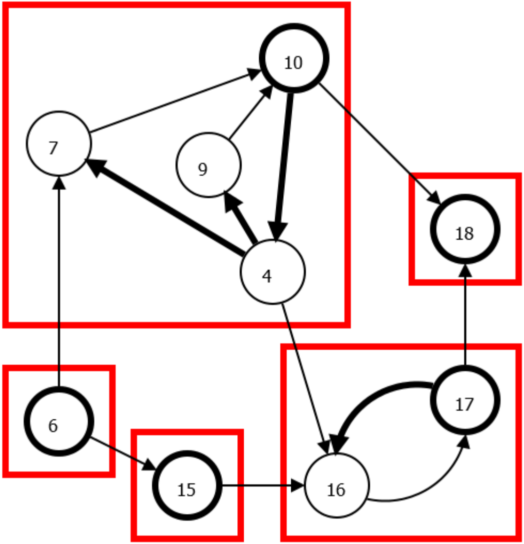
# Example



# Example



# Example



# Runtime

- Essentially `DFS` on $G^R$ and then on $G$.
- Runtime $O(|V| + |E|)$.