

# Decomposition of Graphs: Exploring Graphs

Daniel Kane

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

Graph Algorithms  
Data Structures and Algorithms

# Learning Objectives

- Implement the explore algorithm.
- Figure out whether or not one vertex of a graph is reachable from another.

# Outline

- 1 Problem Discussion
- 2 Ideas
- 3 Explore
- 4 Correctness
- 5 DFS

# Motivation

You're playing a video game and want to make sure that you've found everything in a level before moving on.  
How do you ensure that you accomplish this?

# Examples

This notion of exploring a graph has many applications:

- Finding routes
- Ensuring connectivity
- Solving puzzles and mazes

# Paths

We want to know what is reachable from a given vertex.

## Definition

A **path** in a graph  $G$  is a sequence of vertices  $v_0, v_1, \dots, v_n$  so that for all  $i$ ,  $(v_i, v_{i+1})$  is an edge of  $G$ .

# Formal Description

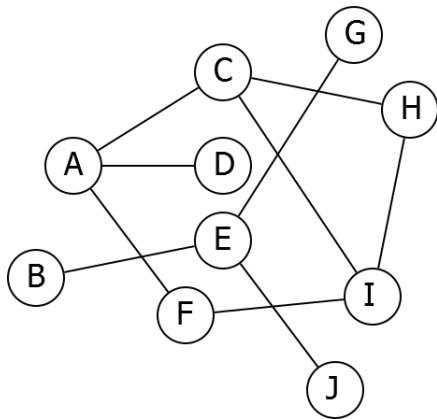
## Reachability

**Input:** Graph  $G$  and vertex  $s$

**Output:** The collection of vertices  $v$  of  $G$  so that there is a path from  $s$  to  $v$ .

# Problem

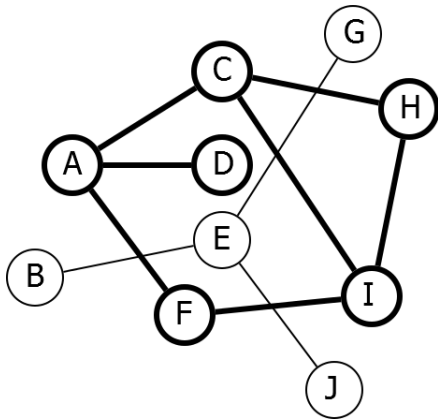
Which vertices are reachable from *A*?





# Solution

*A, C, D, F, H, I.*

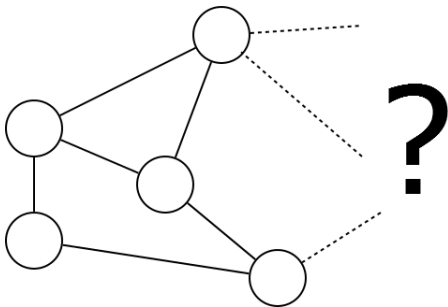


# Outline

- 1 Problem Discussion
- 2 Ideas
- 3 Explore
- 4 Correctness
- 5 DFS

# Basic Idea

We want to make sure that we have explored every edge leaving every vertex we have found.



# Pseudocode

## Component(*s*)

*DiscoveredNodes*  $\leftarrow \{s\}$

while there is an edge *e* leaving  
*DiscoveredNodes* that has not been  
explored:

    add vertex at other end of *e* to  
    *DiscoveredNodes*

return *DiscoveredNodes*

# Formal Specification

We need to do some work to handle the bookkeeping for this algorithm.

- How do we keep track of which edges/vertices we have dealt with?
- What order do we explore new edges in?

# Outline

- 1 Problem Discussion
- 2 Ideas
- 3 Explore
- 4 Correctness
- 5 DFS

# Visit Markers

To keep track of vertices found:

Give each vertex boolean `visited(v)`.

# Unprocessed Vertices

Keep a list of vertices with edges left to check.

This will end up getting hidden in the program stack.



# Depth First Ordering

We will explore new edges in **Depth First** order. We will follow a long path forward, only backtracking when we hit a dead end.

# Explore

Explore( $v$ )

visited( $v$ )  $\leftarrow$  true

for ( $v, w$ )  $\in E$ :

    if not visited( $w$ ):

        Explore( $w$ )

# Explore

Explore( $v$ )

visited( $v$ )  $\leftarrow$  true

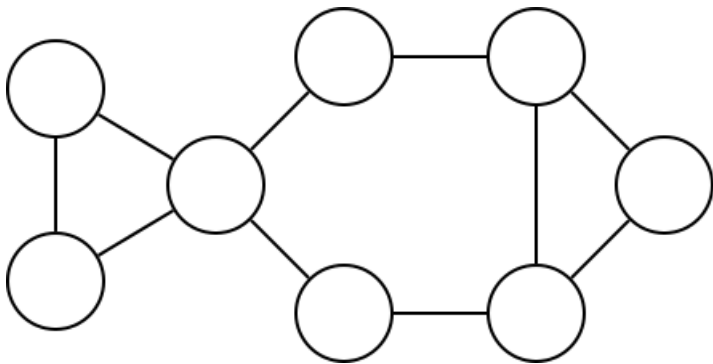
for ( $v, w$ )  $\in E$ :

    if not visited( $w$ ):

        Explore( $w$ )

Need adjacency list representation!

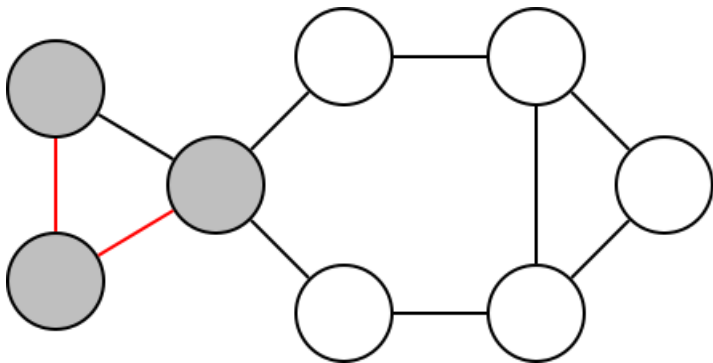
# Example



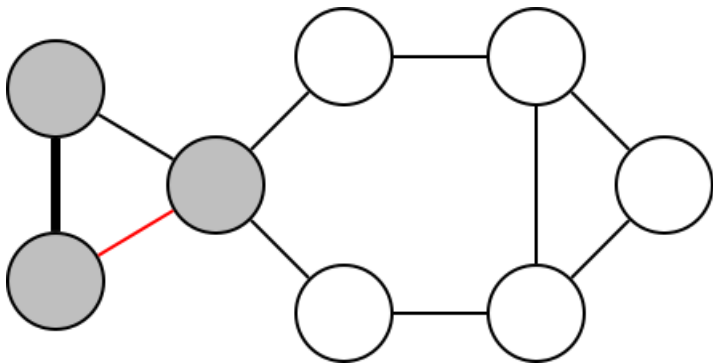




# Example

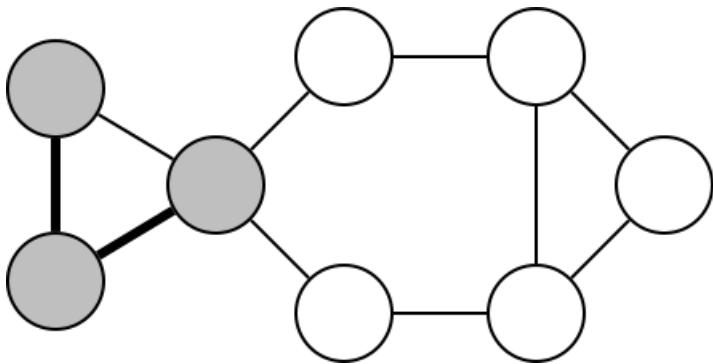


# Example

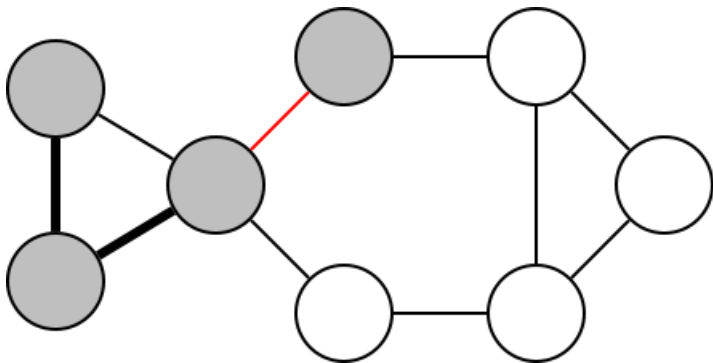




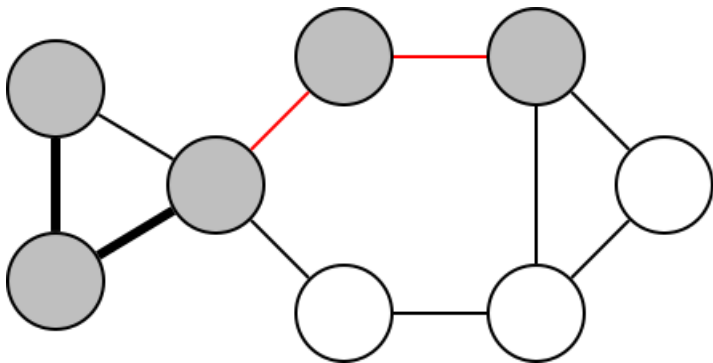
# Example



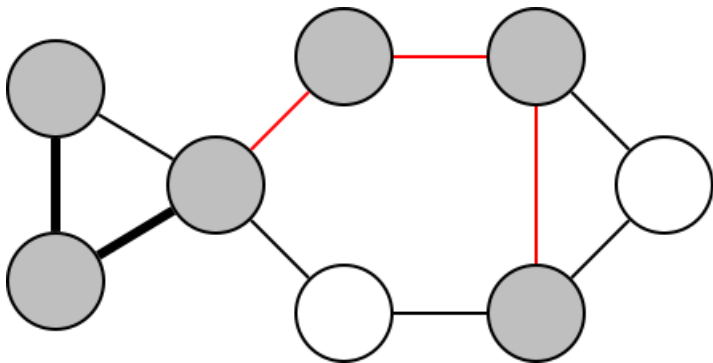
## Example



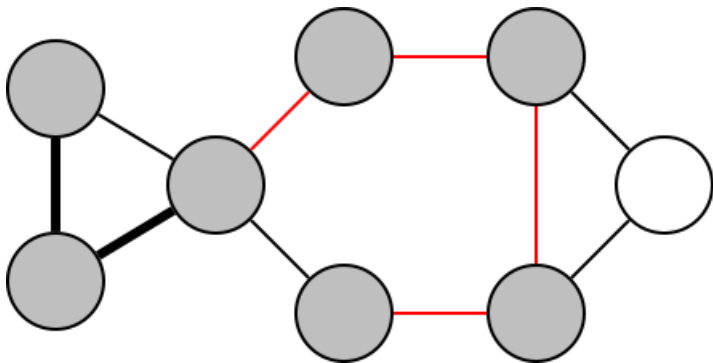
# Example



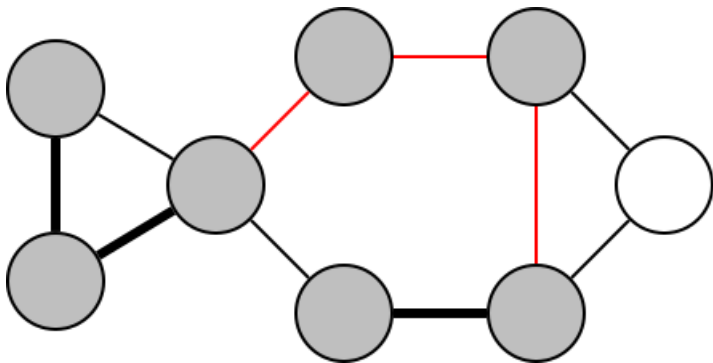
# Example



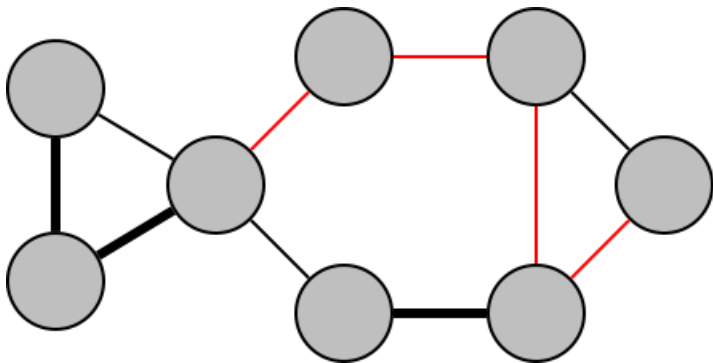
# Example



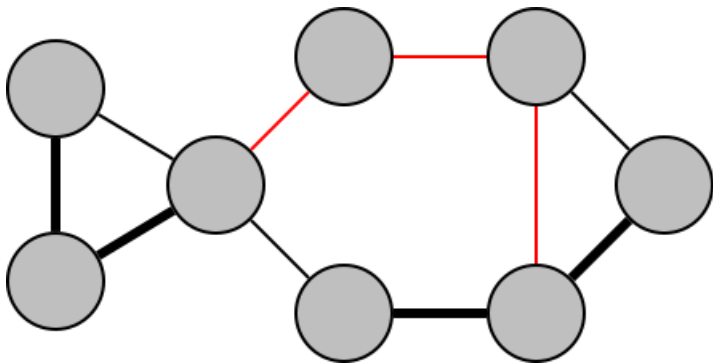
## Example



# Example

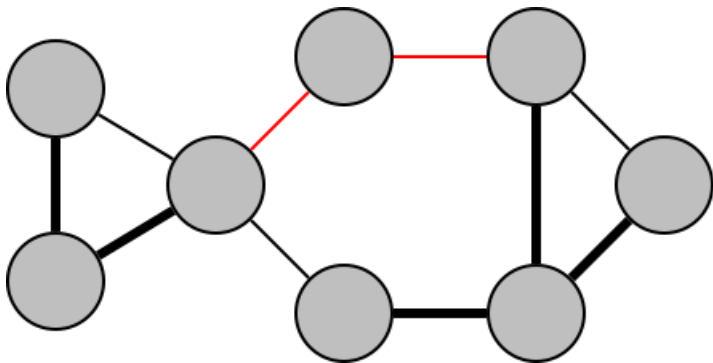


# Example

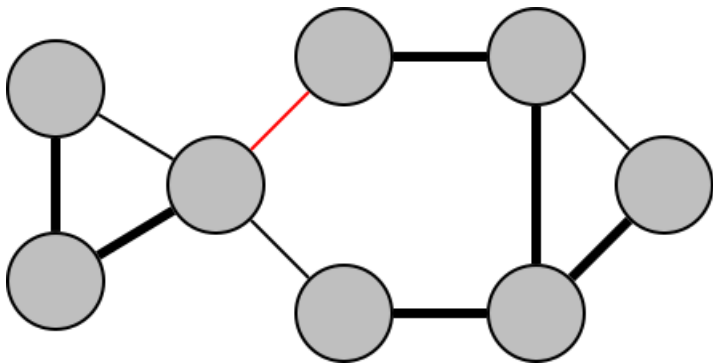




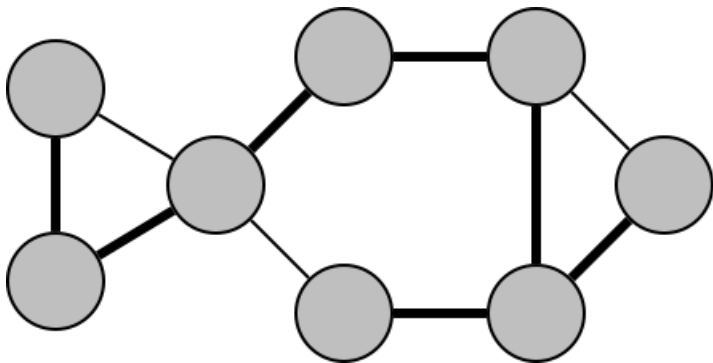
# Example



# Example



# Example



# Outline

- 1 Problem Discussion
- 2 Ideas
- 3 Explore
- 4 Correctness
- 5 DFS

# Result

## Theorem

If all vertices start unvisited,  $\text{Explore}(v)$  marks as visited exactly the vertices reachable from  $v$ .

# Proof

## Proof.

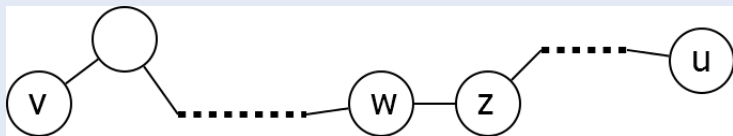
- Only explores things reachable from  $v$ .
- $w$  not marked as visited unless explored.
- If  $w$  explored, all neighbors explored.



# Proof (continued)

## Proof.

- $u$  reachable from  $v$  by path.
- Assume  $w$  furthest along path explored.



- Must explore next item.



# Outline

- 1 Problem Discussion
- 2 Ideas
- 3 Explore
- 4 Correctness
- 5 DFS



# Reach all Vertices

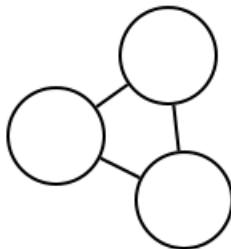
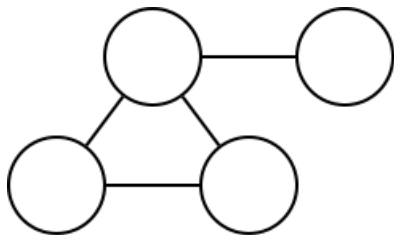
Sometimes you want to find all vertices of  $G$ , not just those reachable from  $v$ .

# DFS

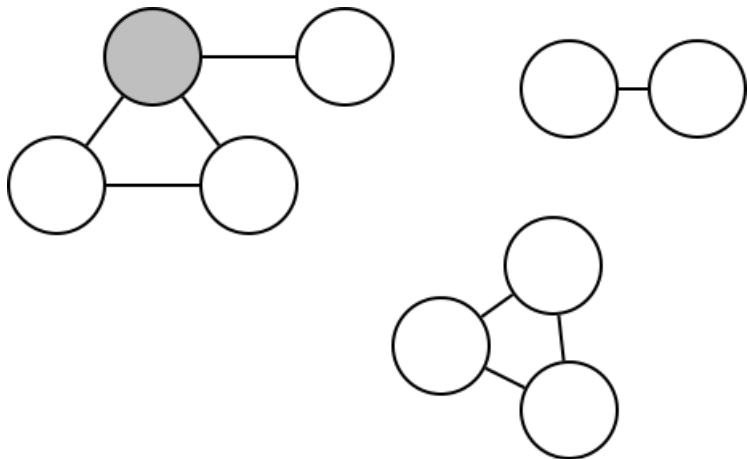
DFS( $G$ )

```
for all  $v \in V$ :      mark  $v$  unvisited
for  $v \in V$ :
    if not visited( $v$ ):
        Explore( $v$ )
```

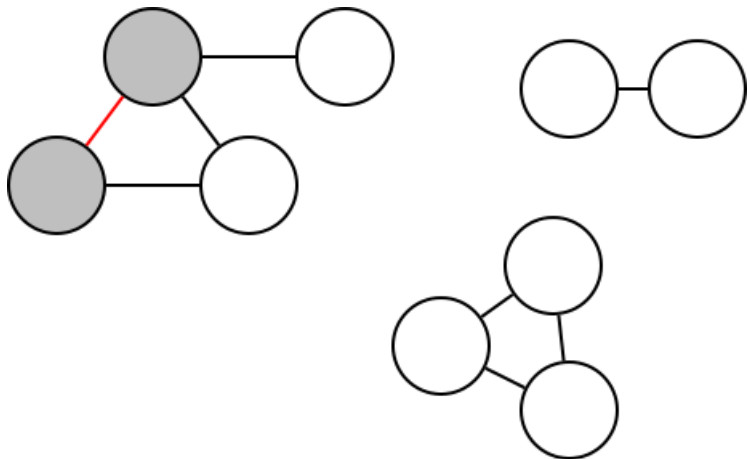
# Example



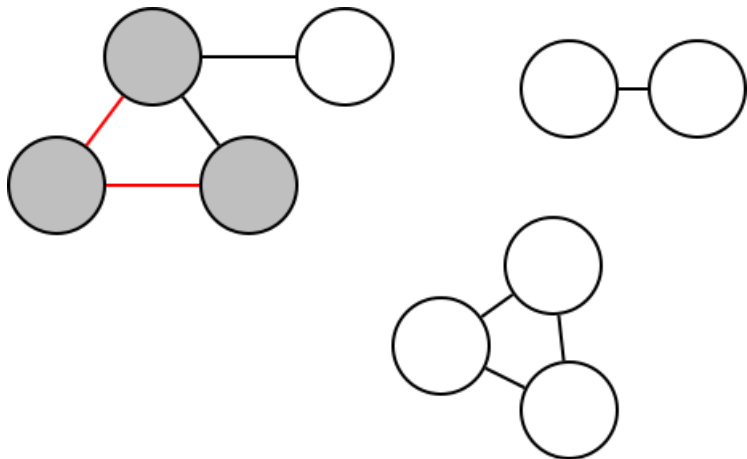
# Example



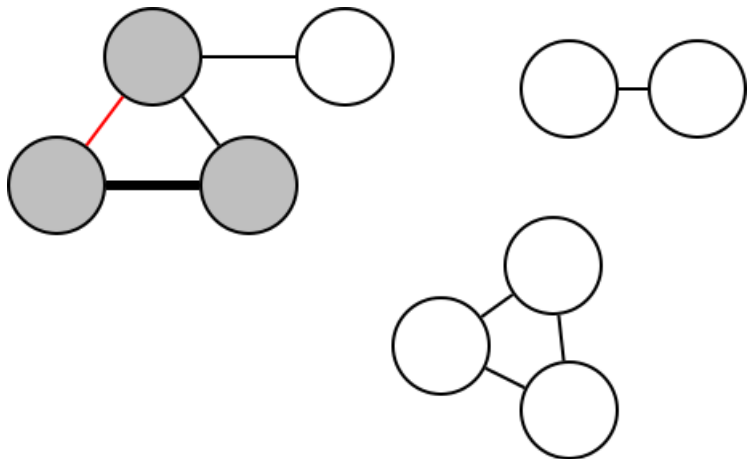
# Example



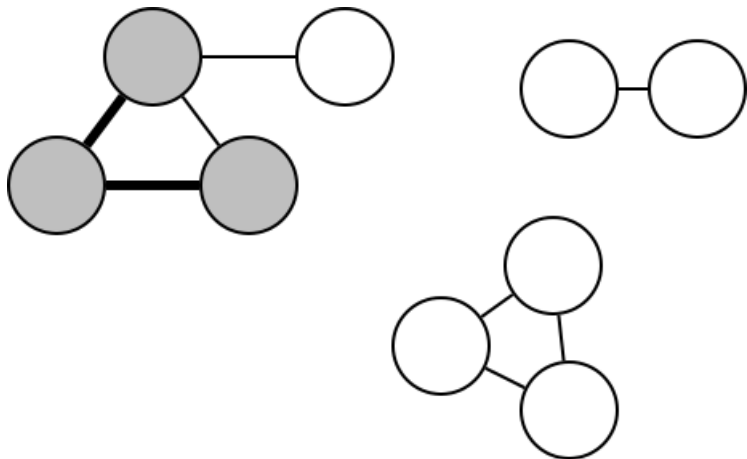
# Example



# Example

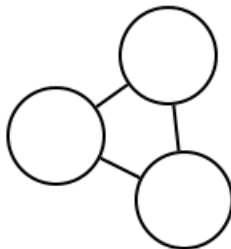
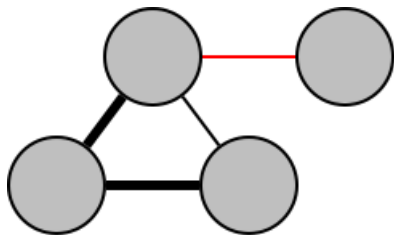


# Example

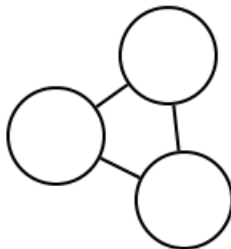
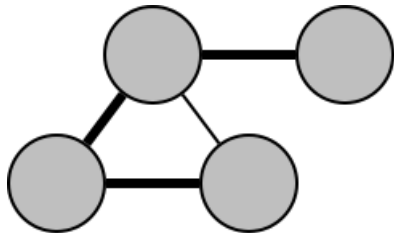




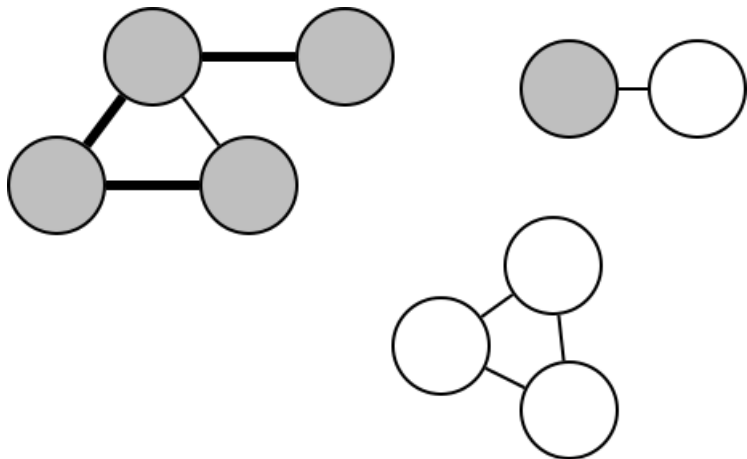
# Example



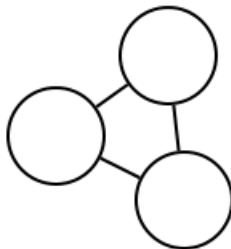
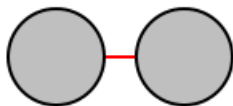
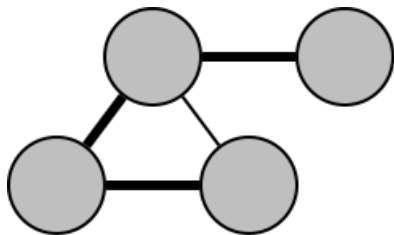
# Example



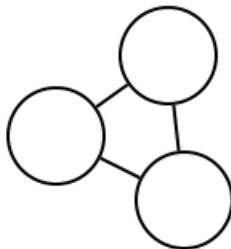
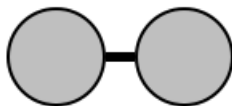
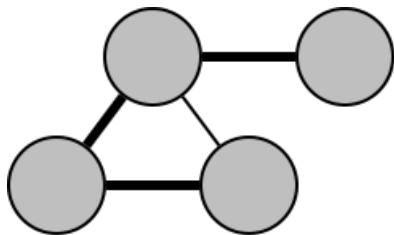
# Example



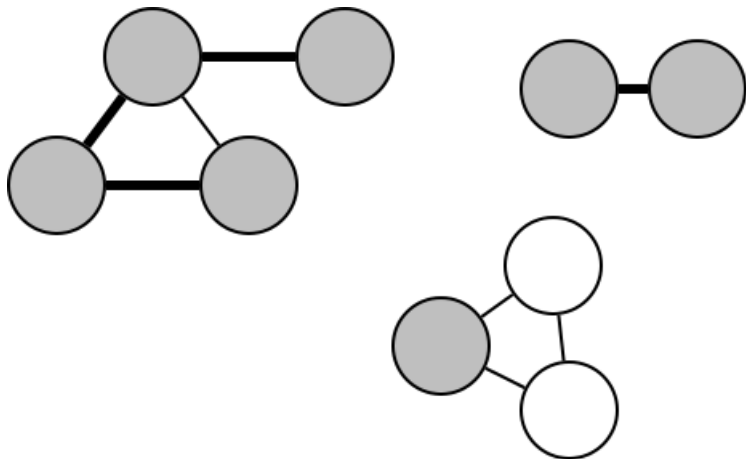
# Example



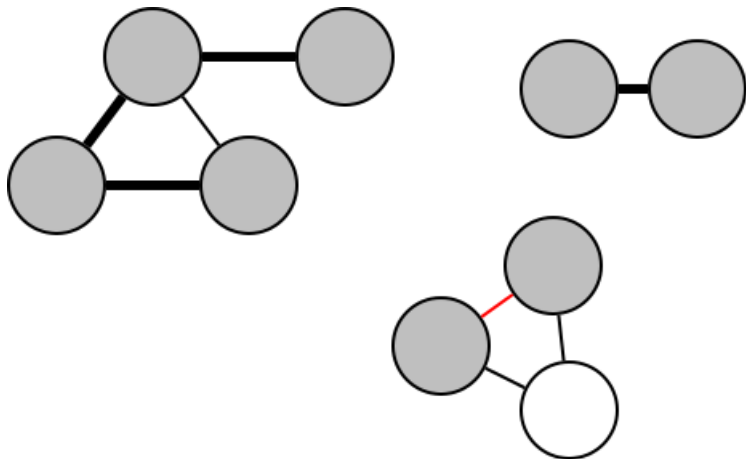
# Example



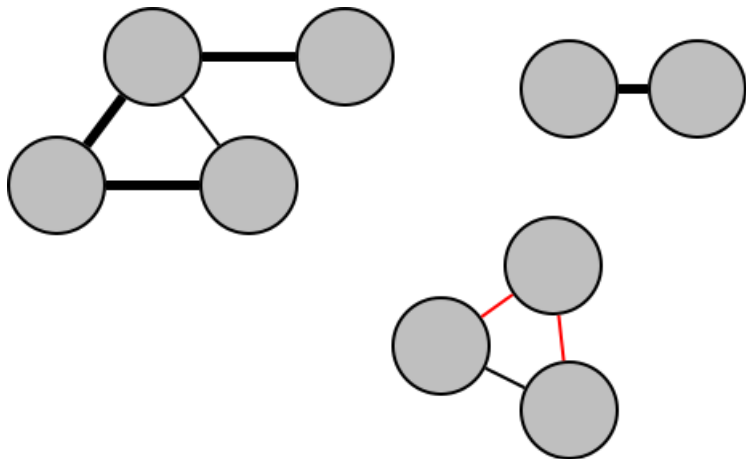
# Example



# Example

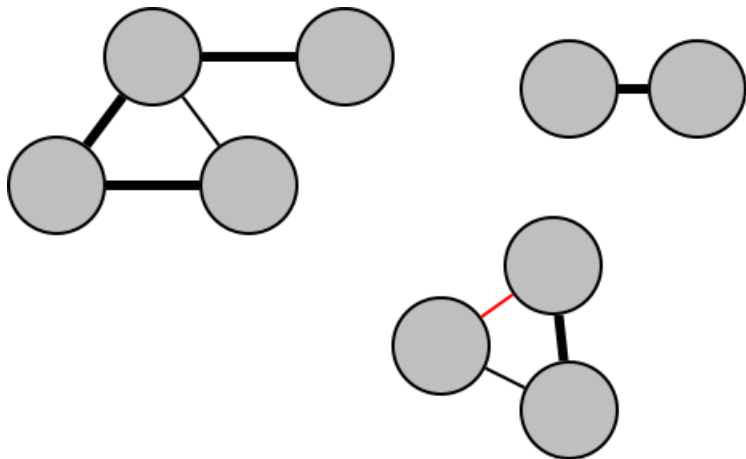


# Example

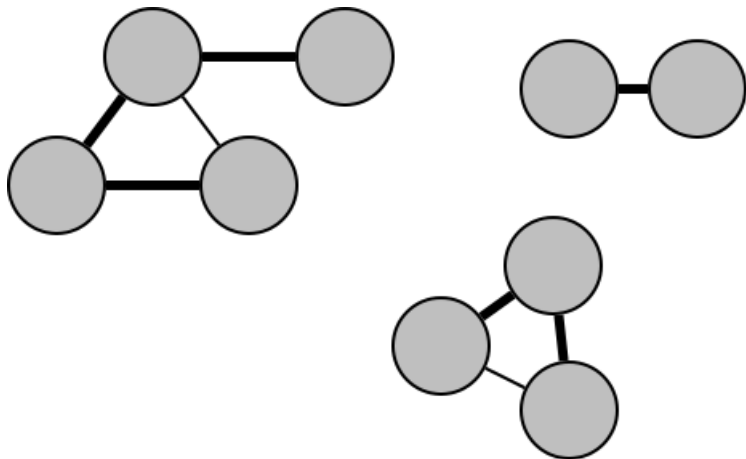




# Example



# Example



# Runtime

Number of calls to explore:

- Each explored vertex is marked visited.
- No vertex is explored after visited once.
- Each vertex is explored exactly once.

# Runtime

Checking for neighbors:

- Each vertex checks each neighbor.
- Total number of neighbors over all vertices is  $O(|E|)$ .

# Runtime

Total runtime:

- $O(1)$  work per vertex.
- $O(1)$  work per edge.
- Total  $O(|V| + |E|)$ .

# Next Time

- More on reachability in graphs.
- Application of DFS.