Graph Depth First Search

261217 Data Structures for Computer Engineers

Patiwet Wuttisarnwattana, Ph.D.

patiwet@eng.cmu.ac.th

Computer Engineering, Chiang Mai University

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 a 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 $v_0, v_1, \ldots, v_n$ so that for all i, (v_i, v_{i+1}) is an edge of G.

Reachability

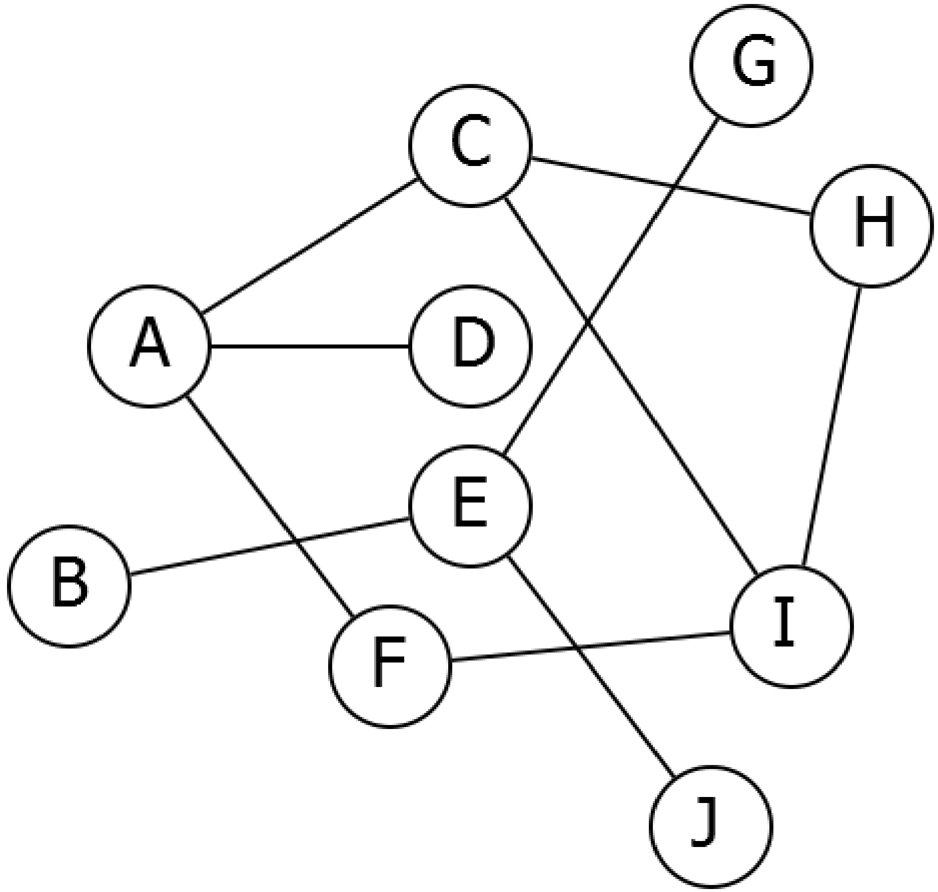
Reachability

Input: Graph G and vertex s

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

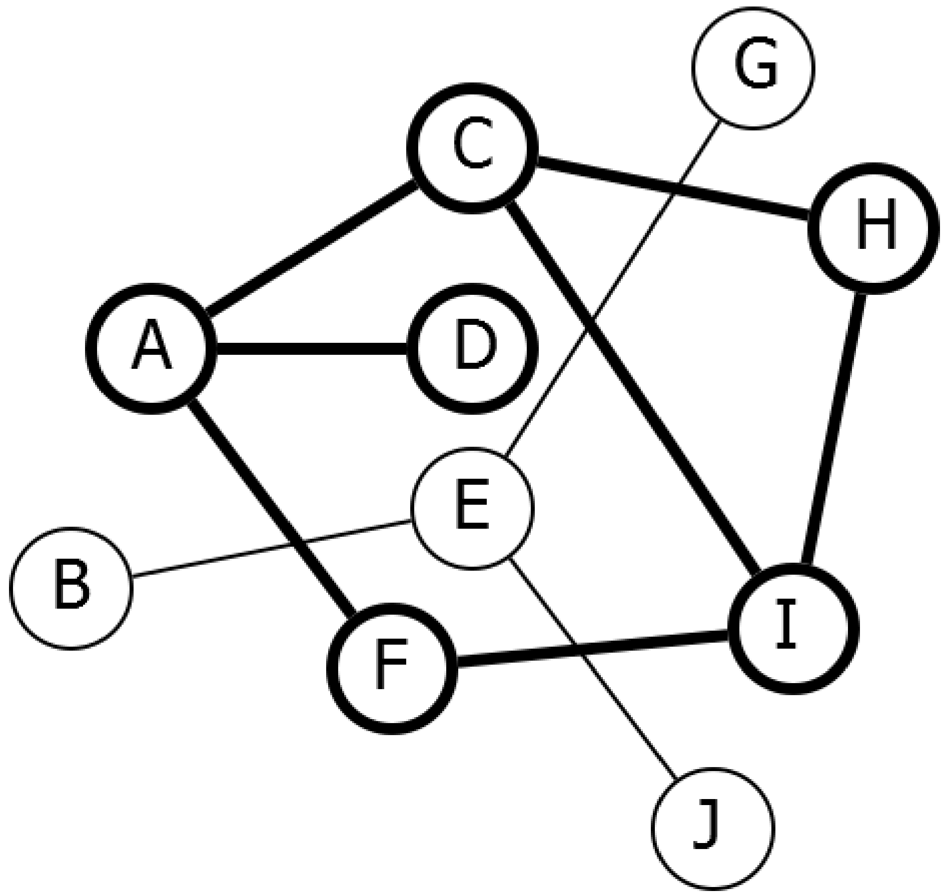
Problem

Which vertices are reachable from A?



Solution

A, C, D, F, H, I



Visit Marker

To keep track of vertices found:

Give each vertex boolean visited(v)

```
If (v.visited != true){
	// Do something
	v.visited = true;
}
```

Depth First Traversal

- To explore new edges in Depth First order
- To follow a long path forward, only backtracking when hitting a dead end

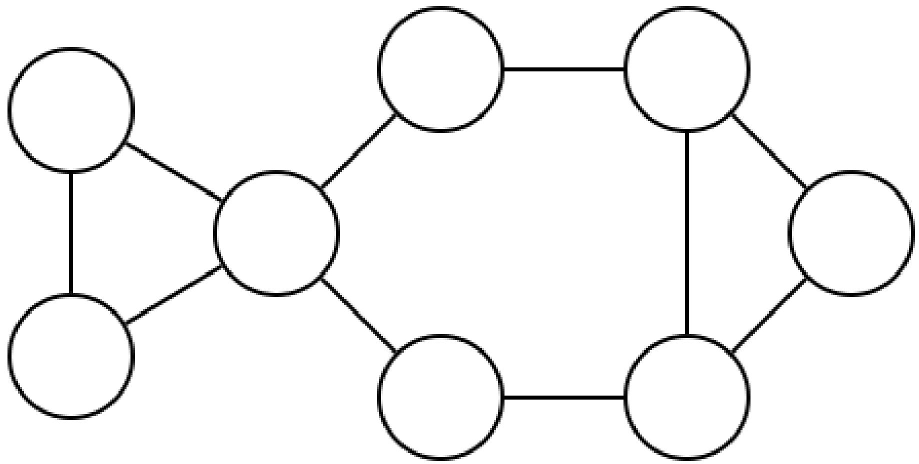
Depth First Exploration

Explore will mark as visited all vertices reachable from v
v, w are vertices; E is a set of all edges in the graph

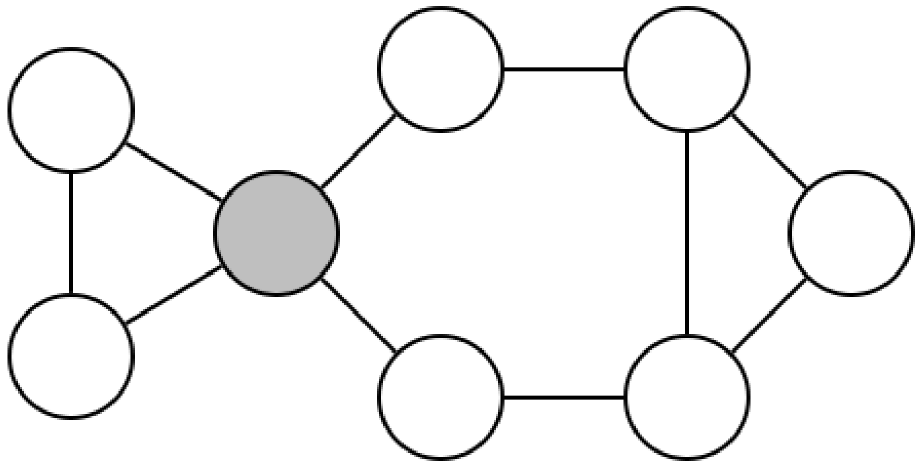
```
Explore(v)

visited(v) ← true
for (v, w) ∈ E:
  if not visited(w):
    Explore(w)
```

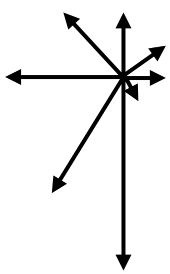
Need adjacency list representation!

Explore Example

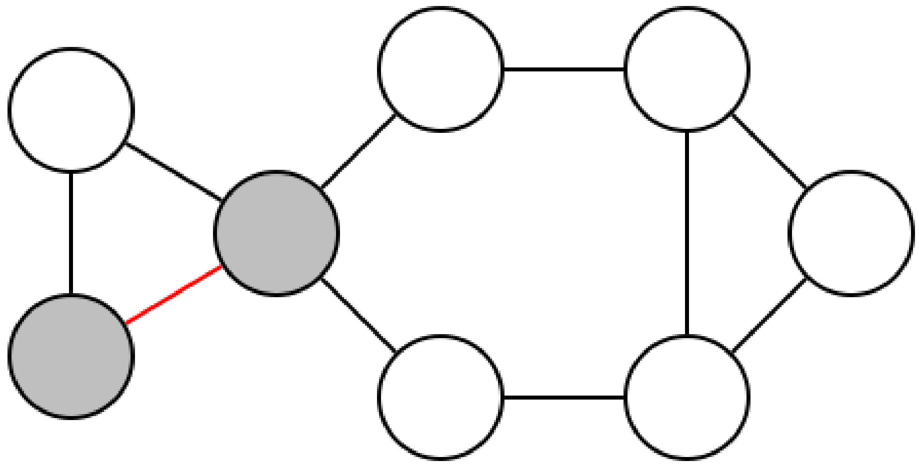


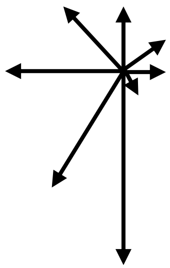
Explore Example



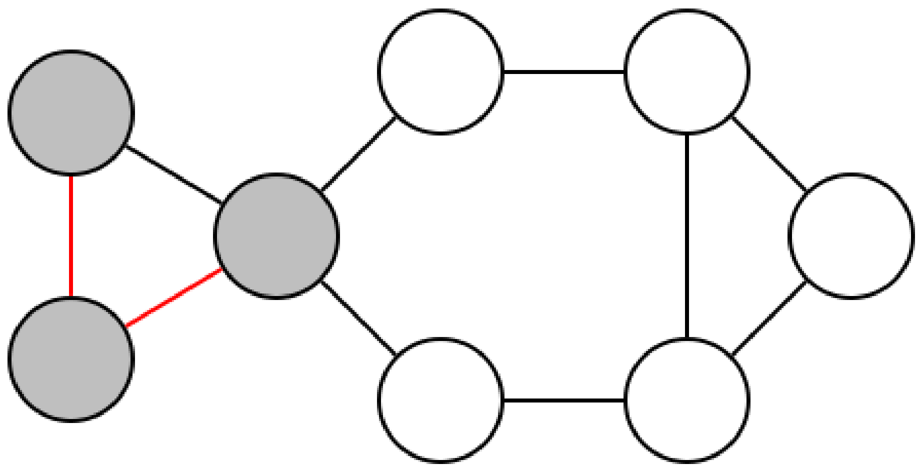


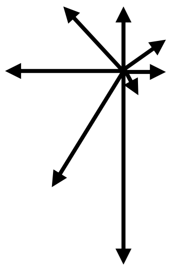
Explore Example



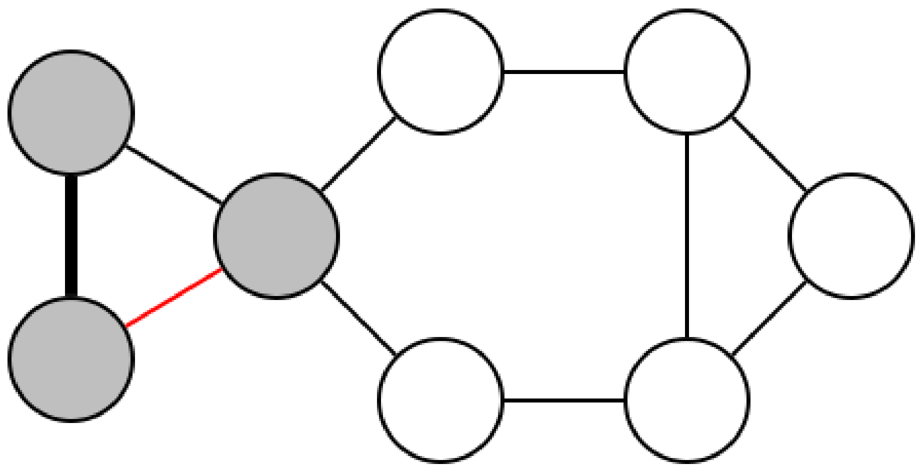


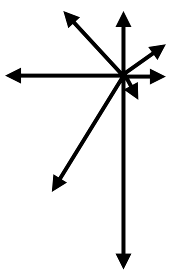
Explore Example



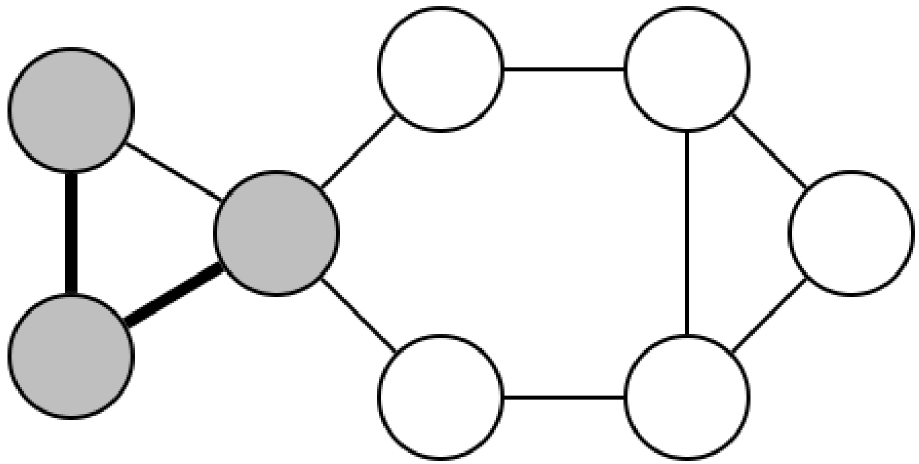


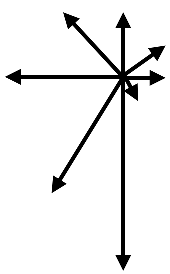
Explore Example



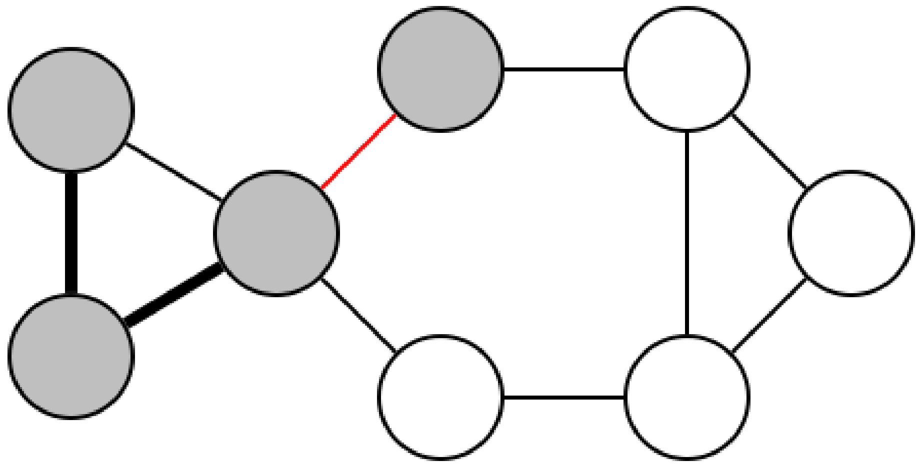


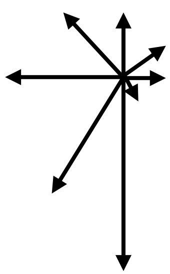
Explore Example



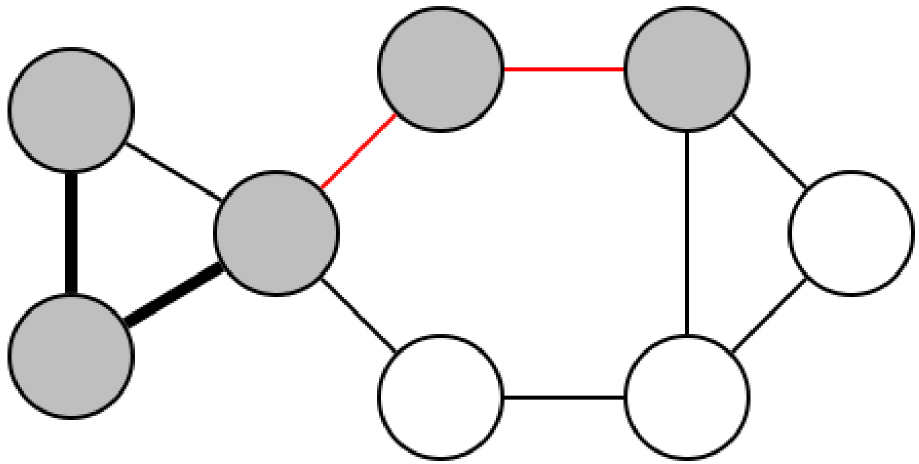


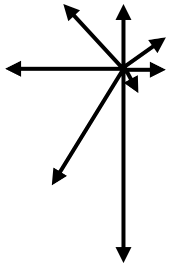
Explore Example



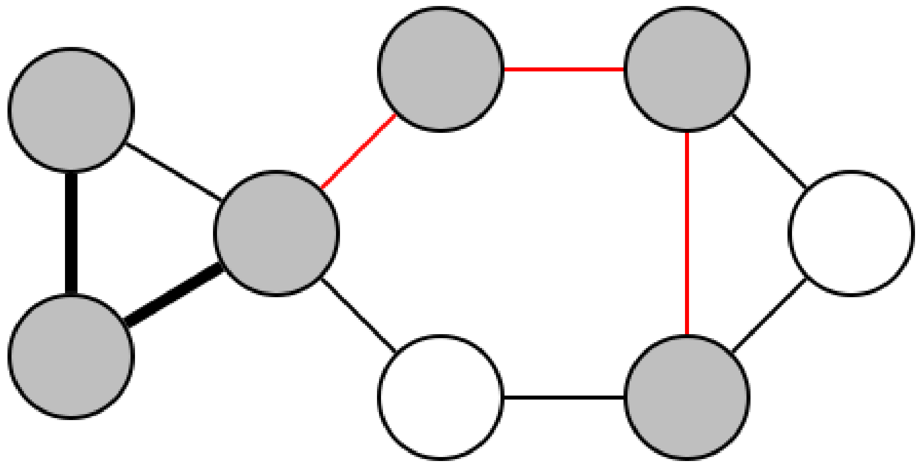


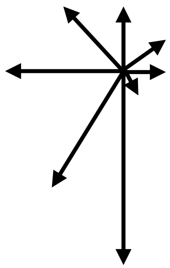
Explore Example



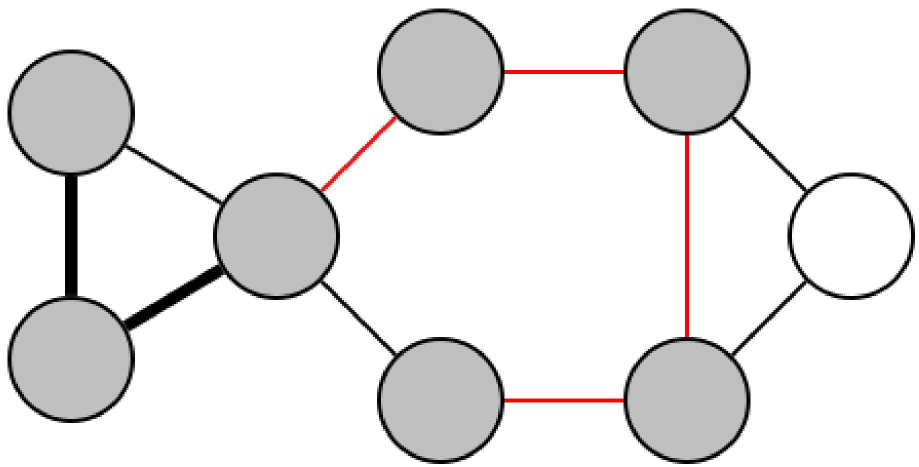


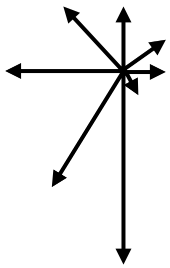
Explore Example



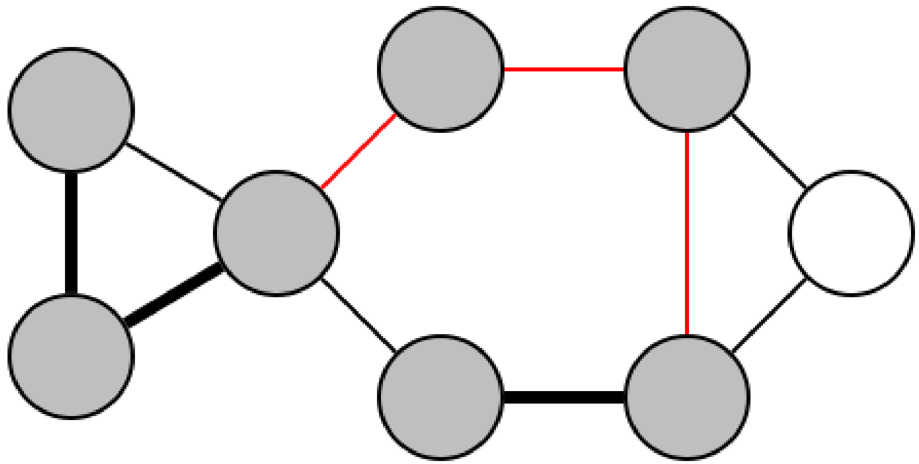


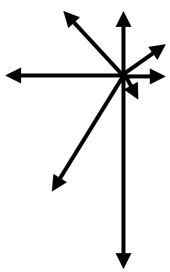
Explore Example



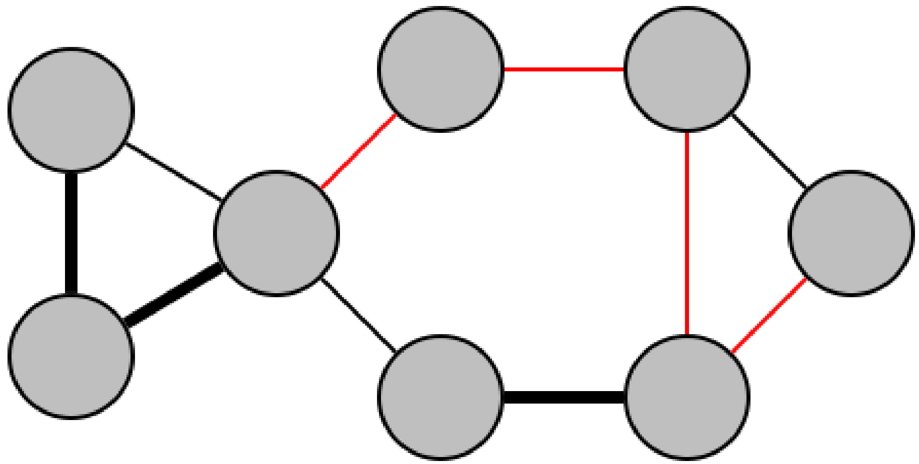


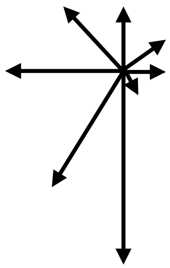
Explore Example



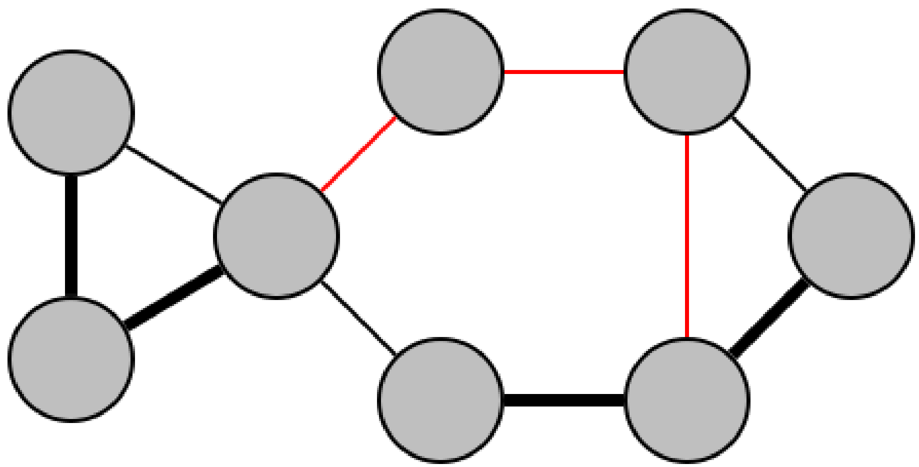


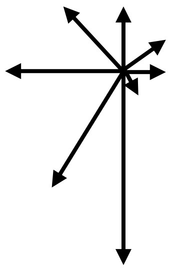
Explore Example



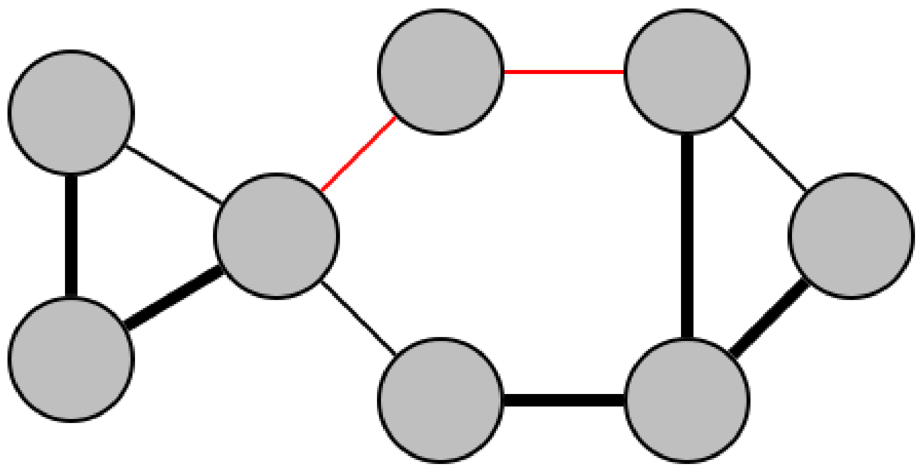


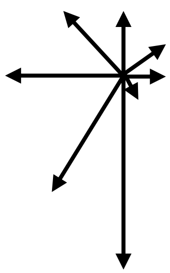
Explore Example



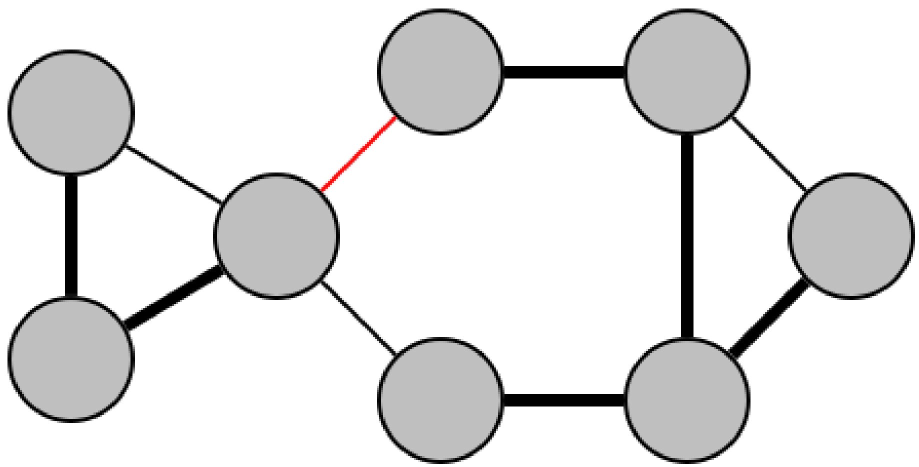


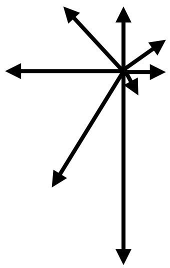
Explore Example



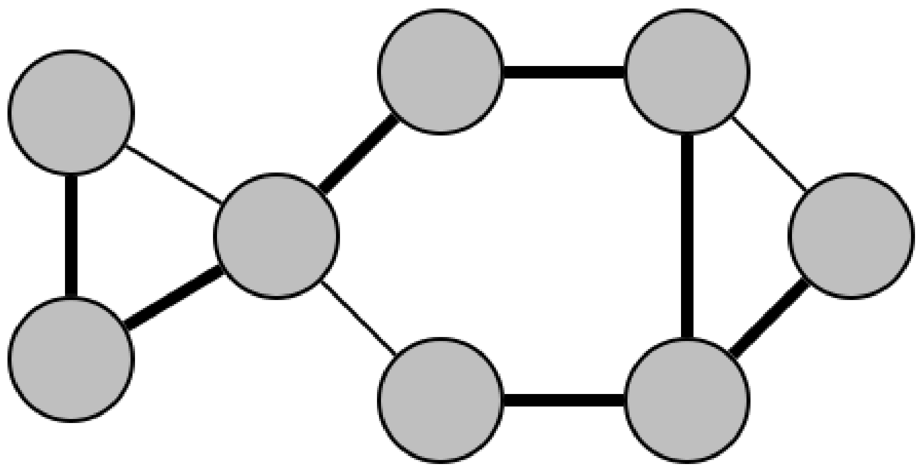


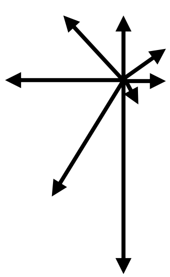
Explore Example





Explore Example





Result

Theorem

If all vertices start unvisited, `Explore`(v) marks as visited exactly the vertices reachable from v.

Depth First Search: DFS

This algorithm will explore every node even though they are not connected

$\mathtt{DFS}(G)$

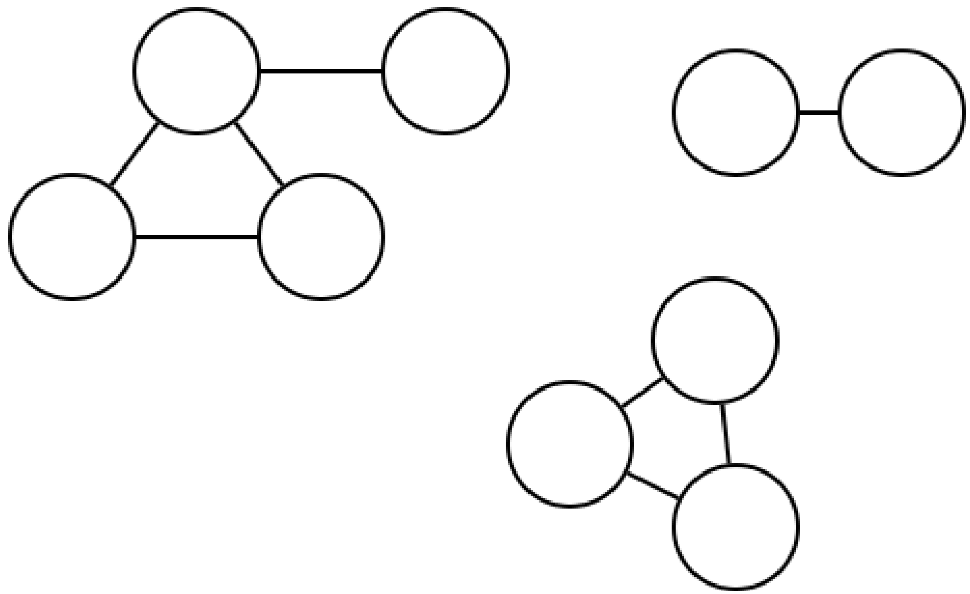
$\texttt{for all } v \in V: \quad \texttt{mark } v \texttt{ unvisited}$

$\texttt{for } v \in V:$

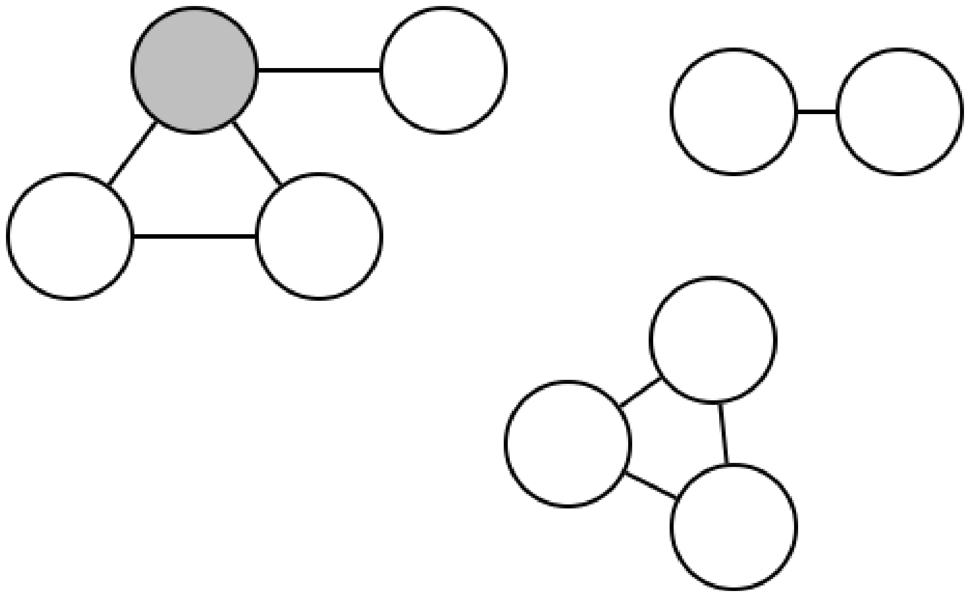
$\quad \texttt{if not visited}(v):$

$\quad\quad \texttt{Explore}(v)$

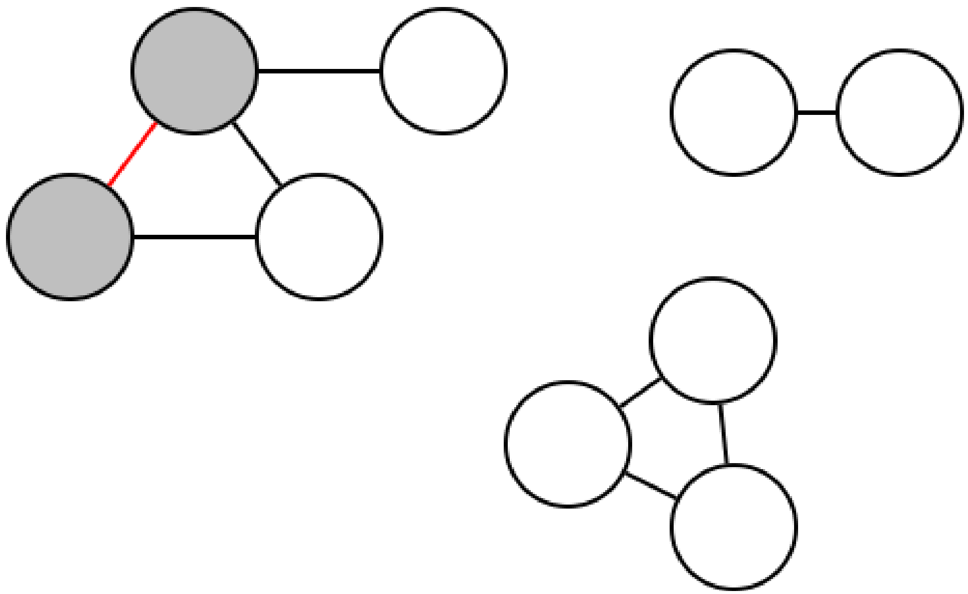
DFS Example



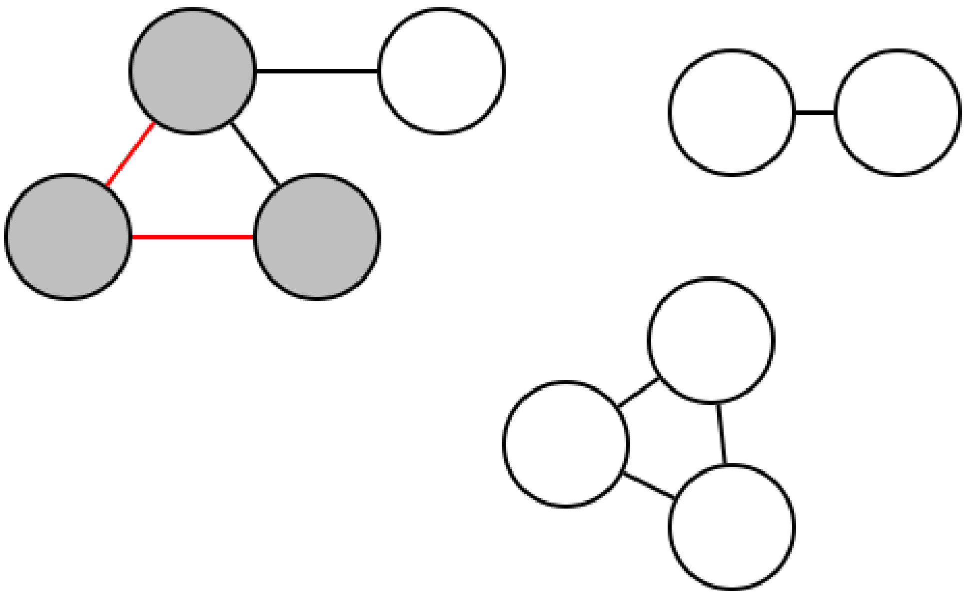
DFS Example



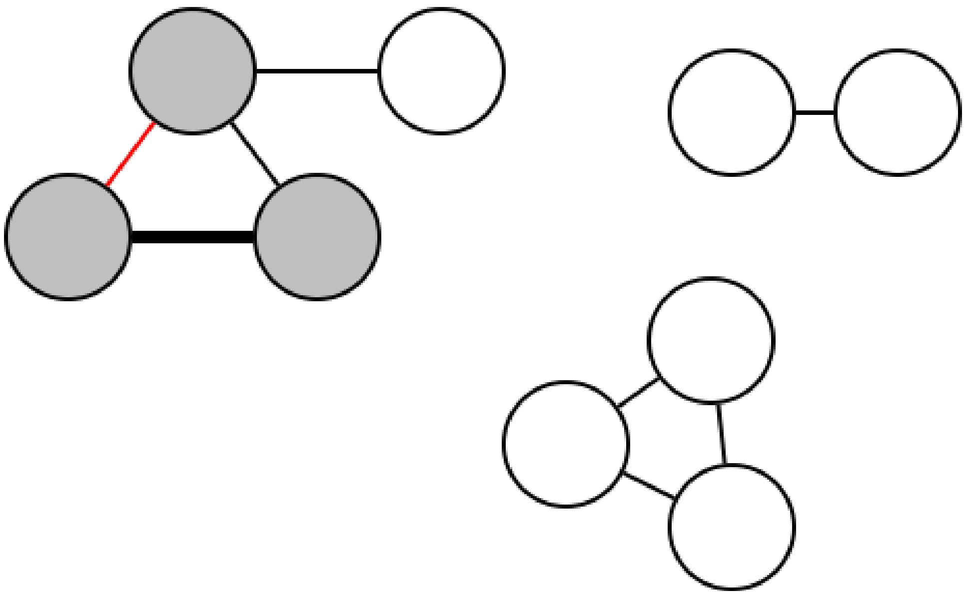
DFS Example



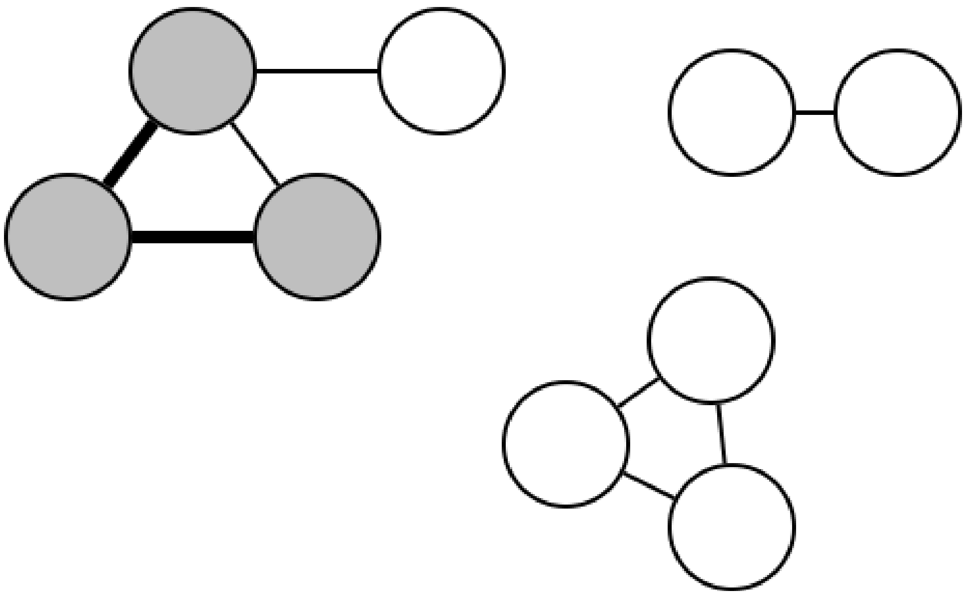
DFS Example



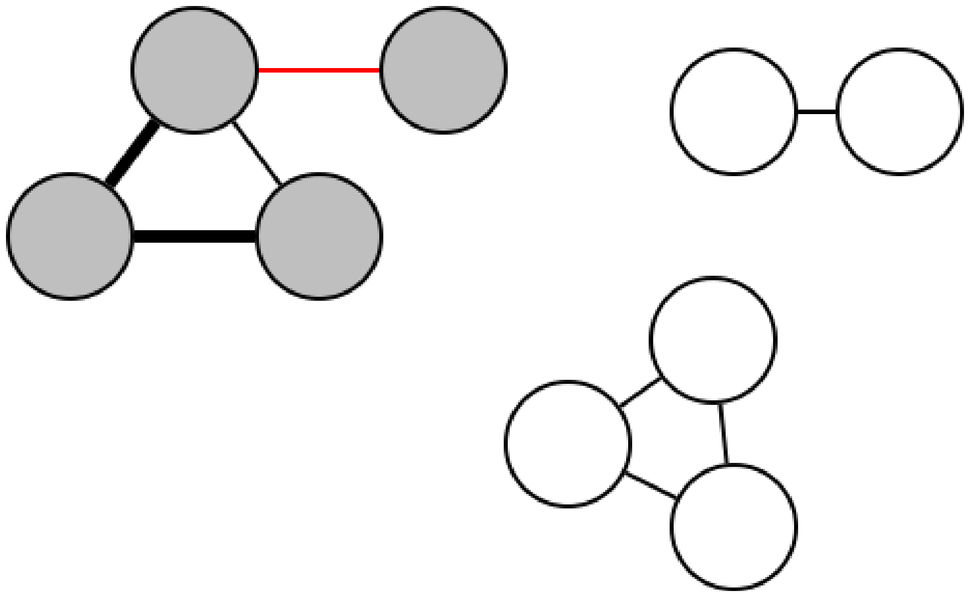
DFS Example



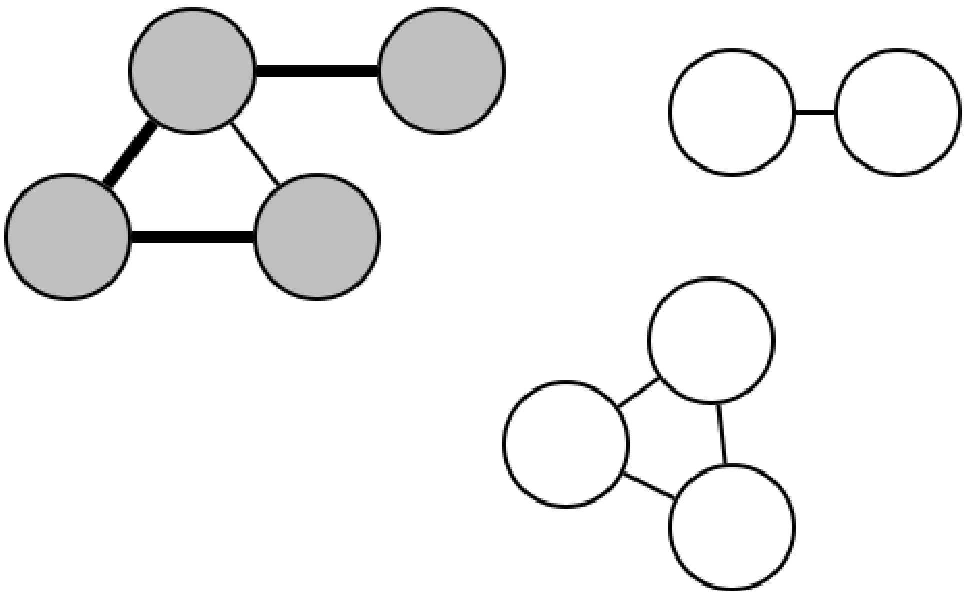
DFS Example



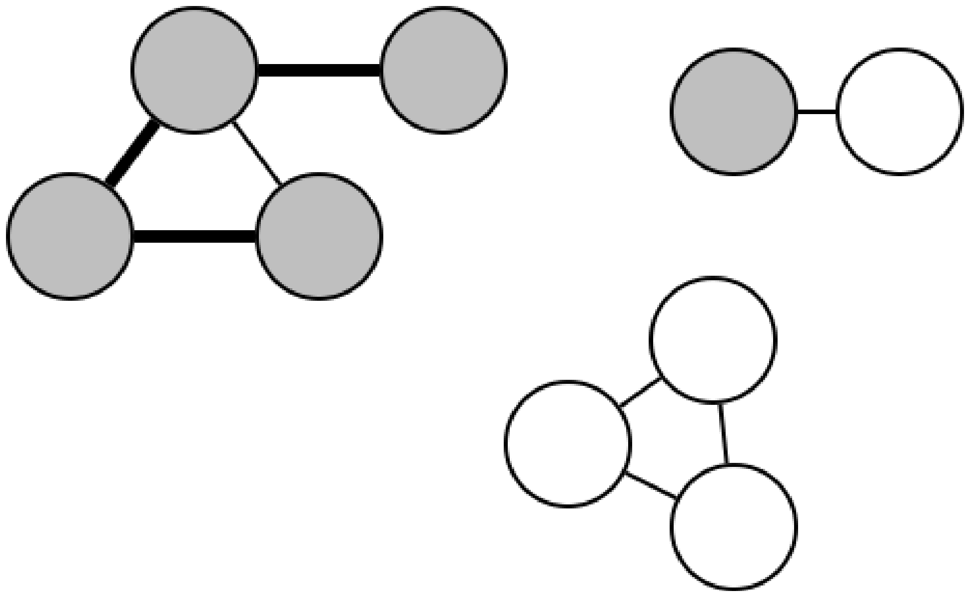
DFS Example



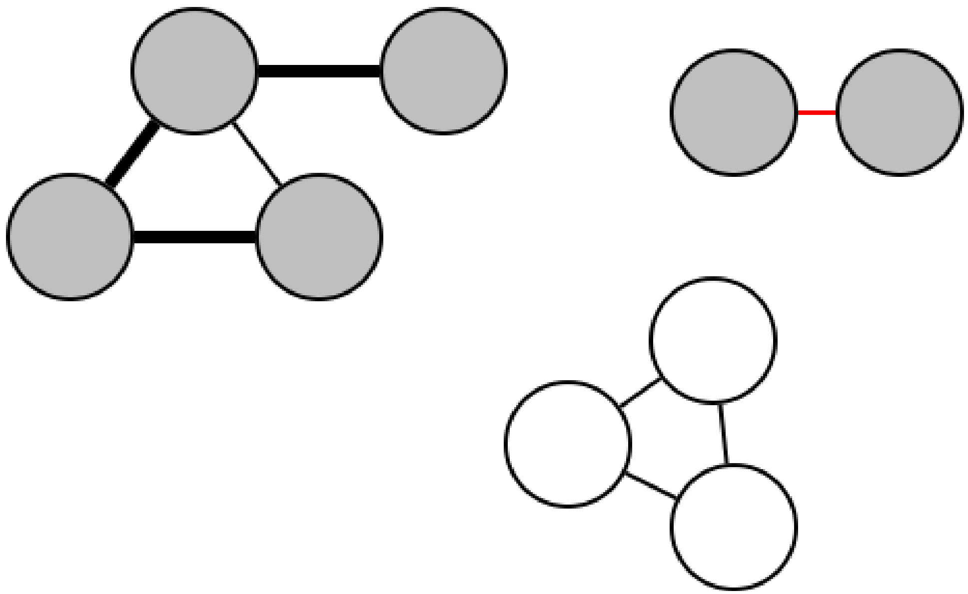
DFS Example



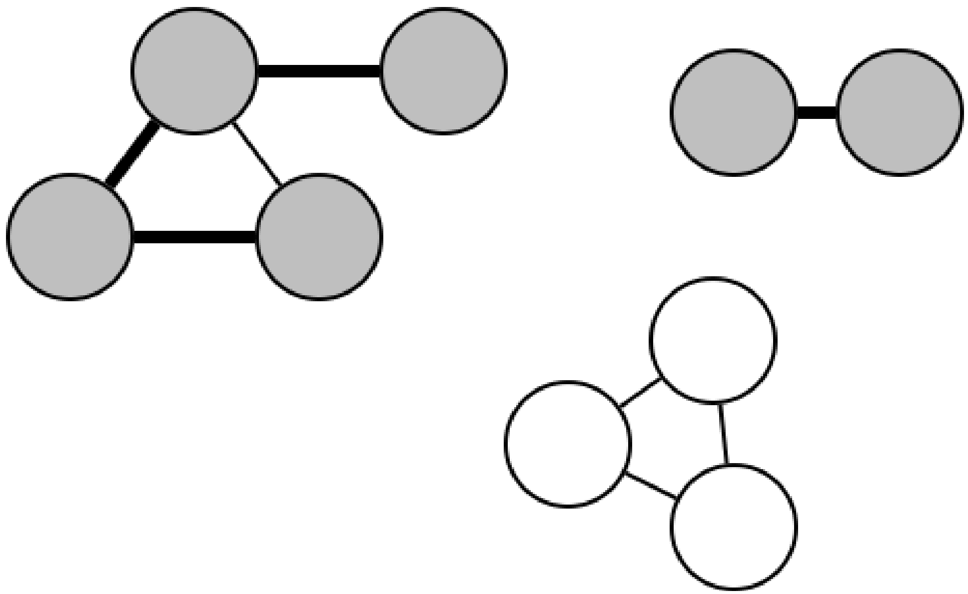
DFS Example



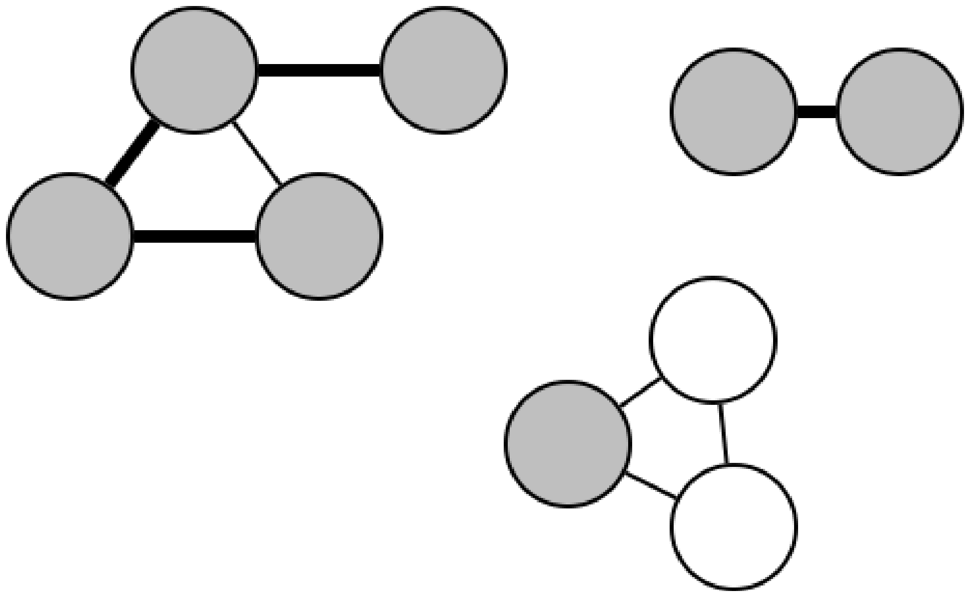
DFS Example



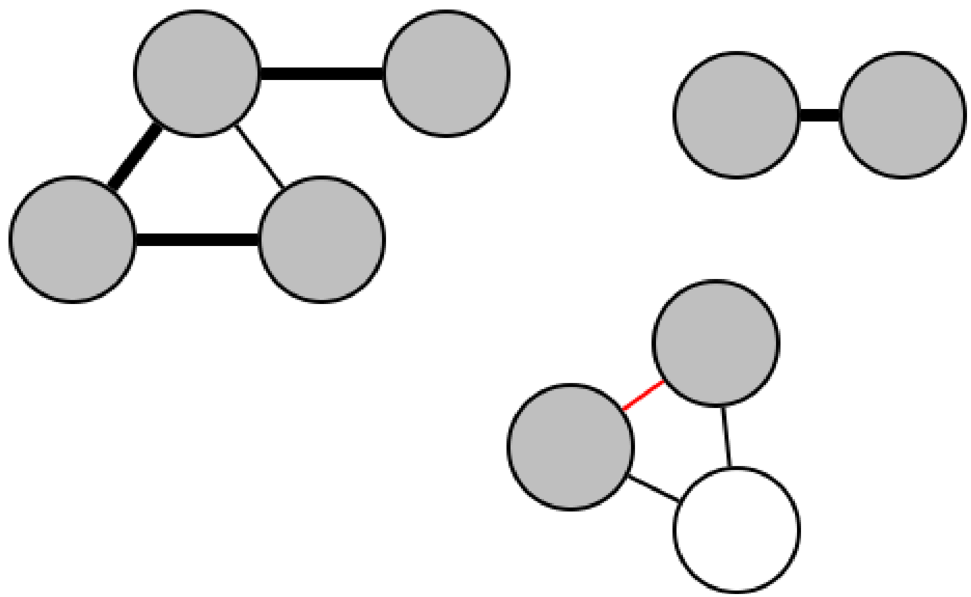
DFS Example



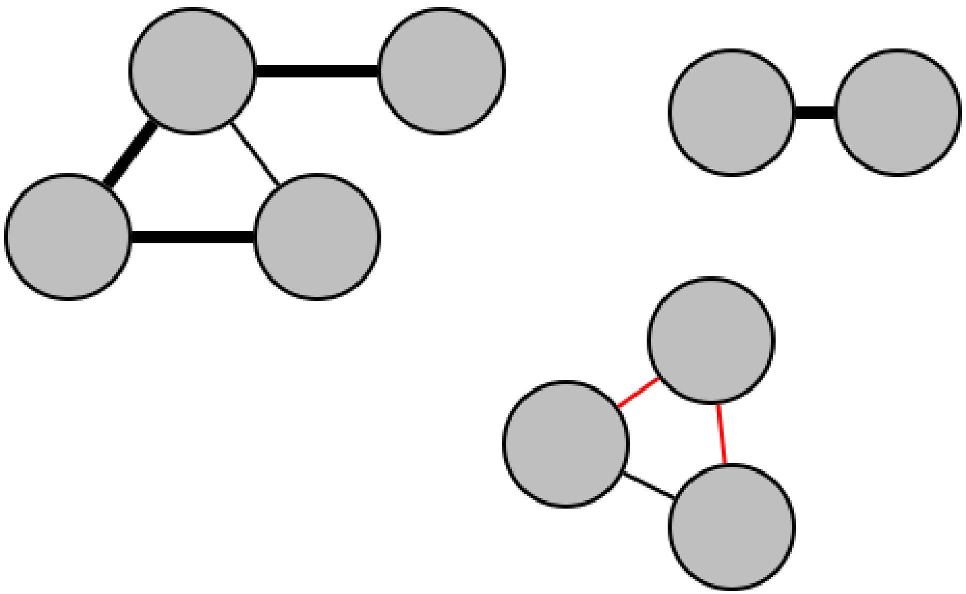
DFS Example



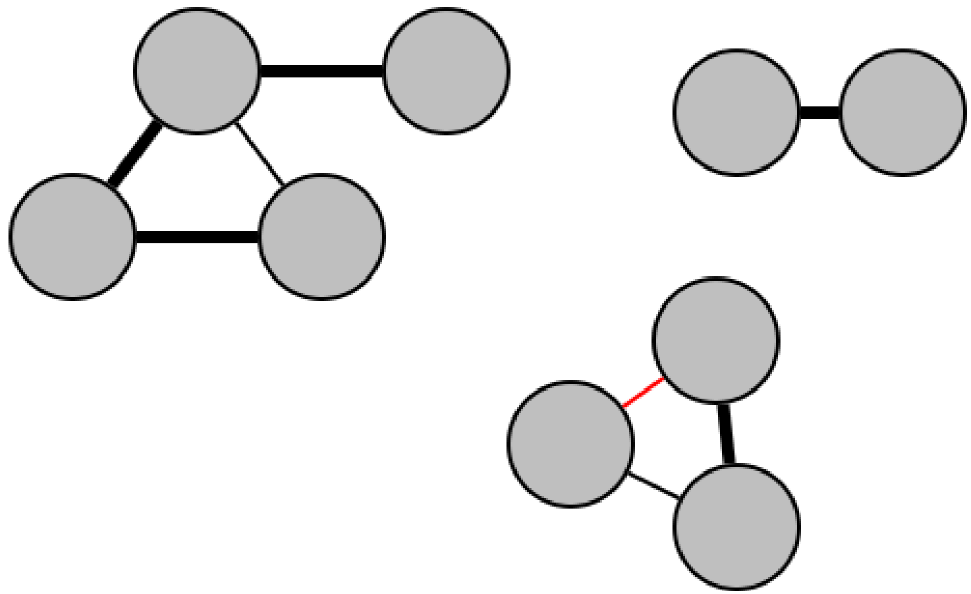
DFS Example



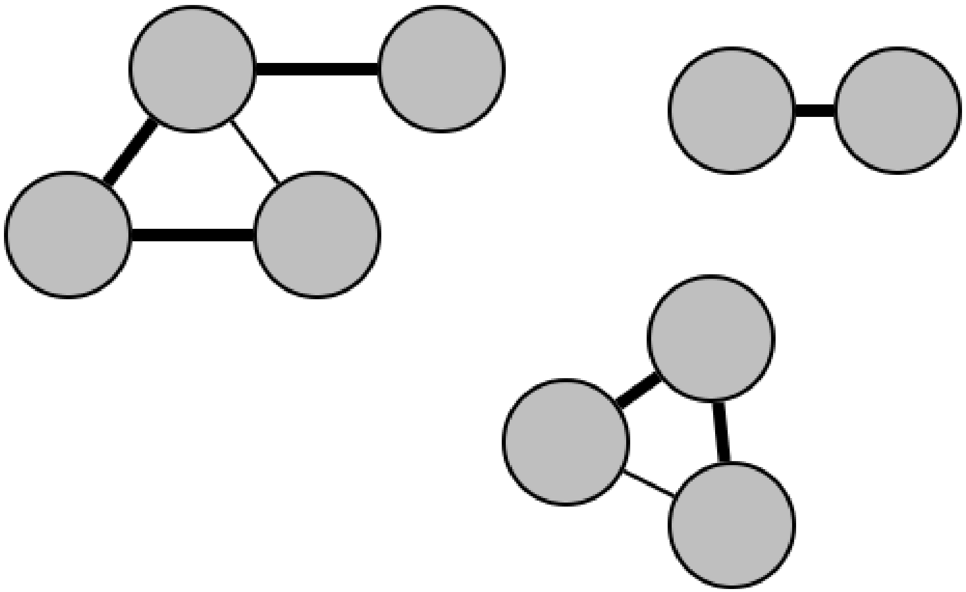
DFS Example



DFS Example



DFS Example



Runtime Analysis

- Number of calls to explore:
  - Each explored vertex is marked visited.
  - No vertex is explored after visited once.
  - Each vertex is explored exactly one.

Runtime Analysis

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

Runtime Analysis

- Total runtime:
  - O(1) work per vertex
  - O(1) work per edge
  - Total O(|V| + |E|)

Connected Components

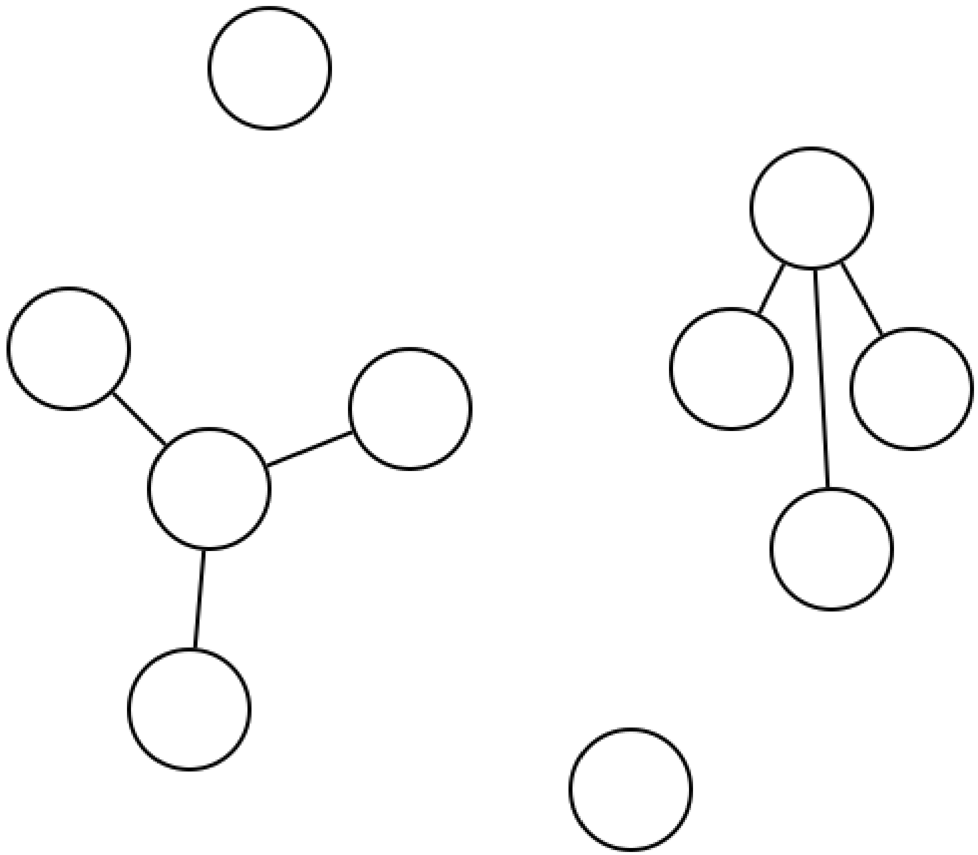
The vertices of a graph G can be partitioned into **Connected Components** so that v is reachable from w

if and only if

they are in the same connected component.

Problem

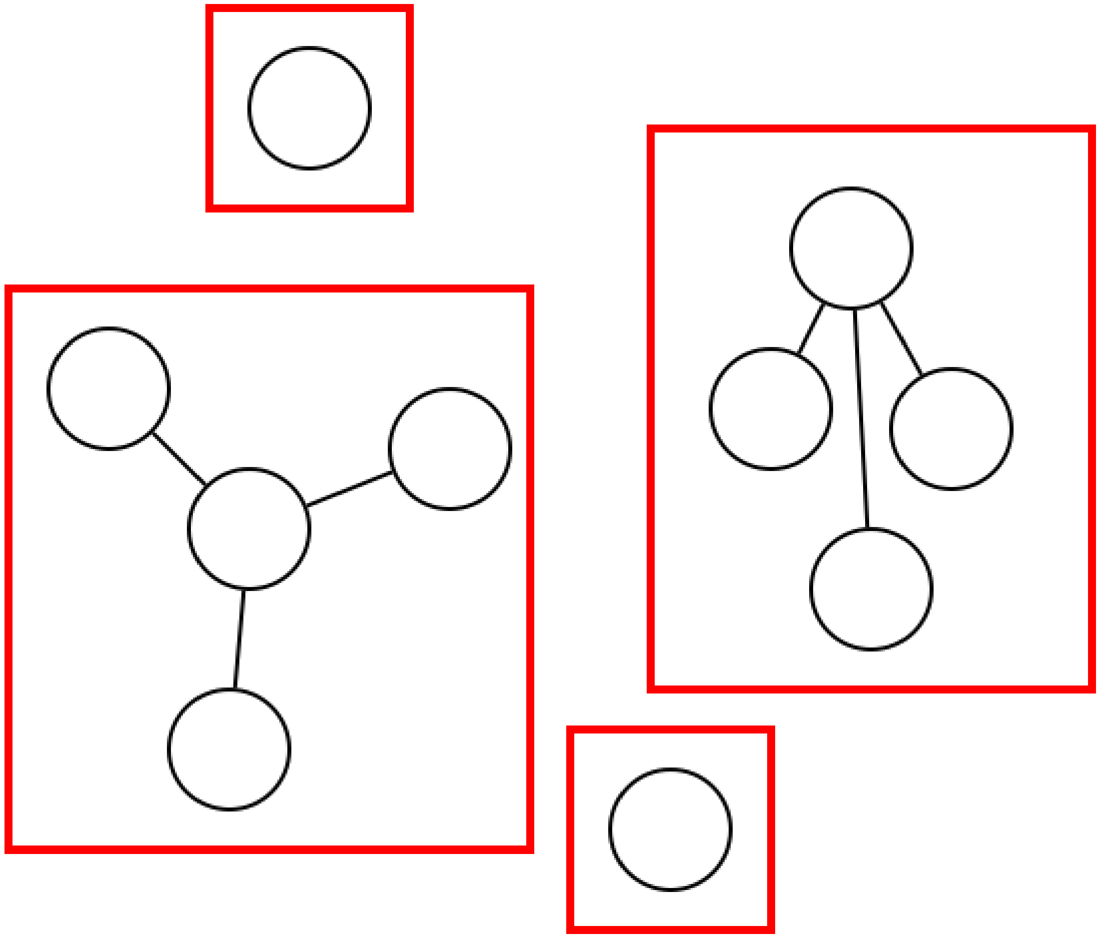
How many connected components does the graph below have?



Solution

How many connected components does the graph below have?

4



Connected Component Algorithm

- Explore(v) finds the connected component of v. Just need to repeat to find other components.
- Modify DFS to do this.
- Modify goal to label connected components.

Modification of Explore

```
Explore(v)

visited(v) ← true
CCnum(v) ← cc
for (v, w) ∈ E:
    if not visited(w):
        Explore(w)
```

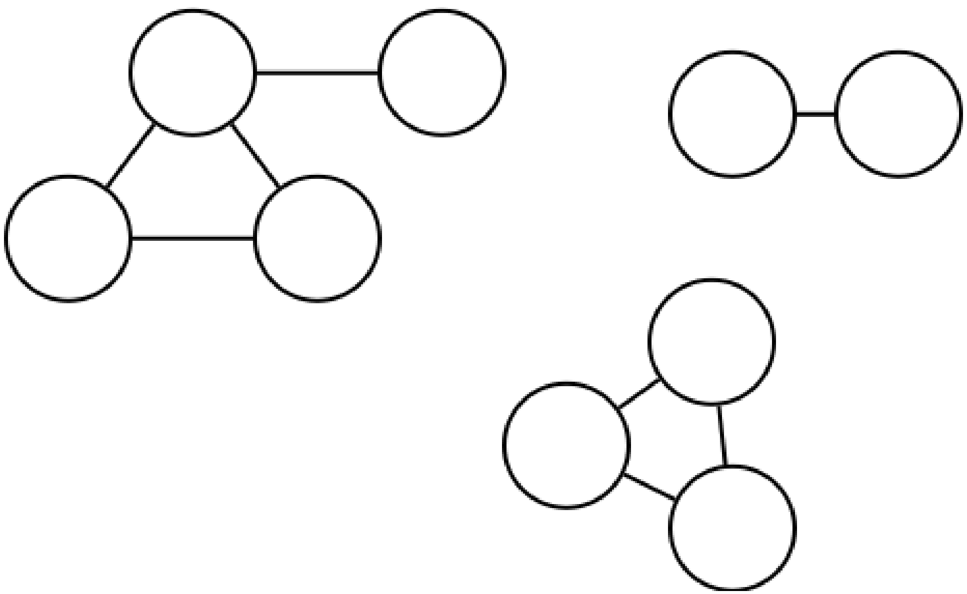
Modification of DFS

DFS(G)

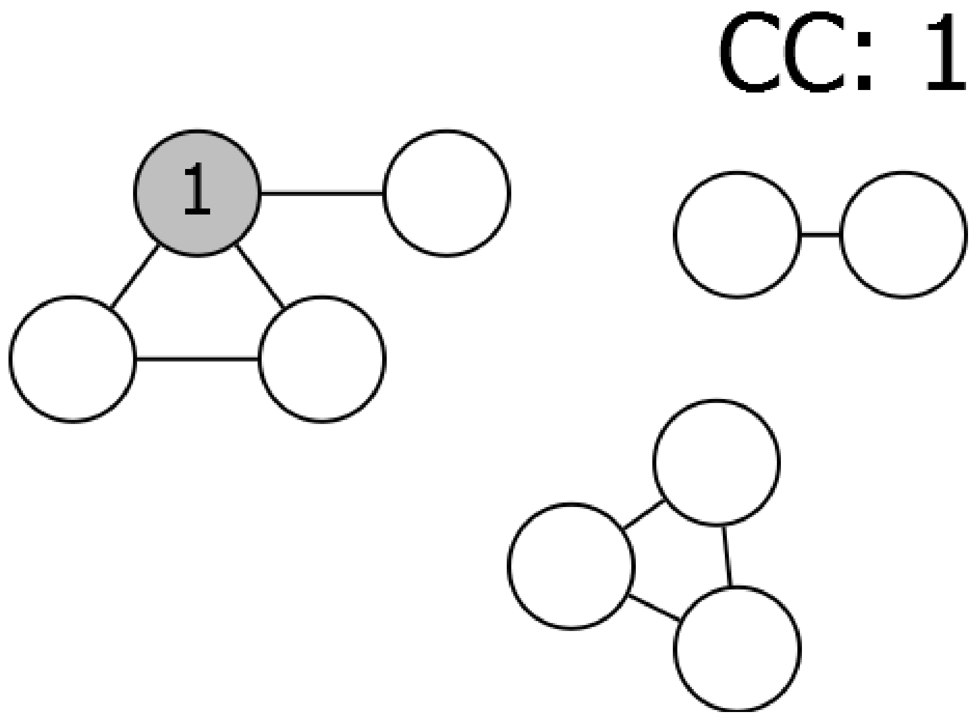
```
for all v ∈ V mark v unvisited
cc ← 1
for v ∈ V:
   if not visited(v):
      Explore(v)
      cc ← cc + 1
```

Example

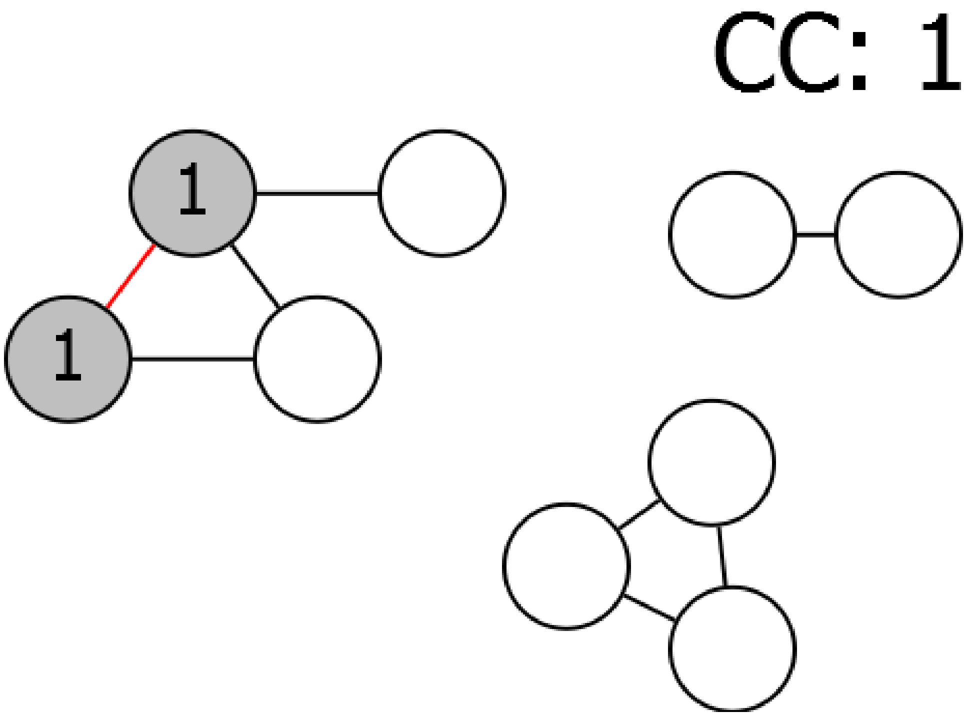
CC: 1



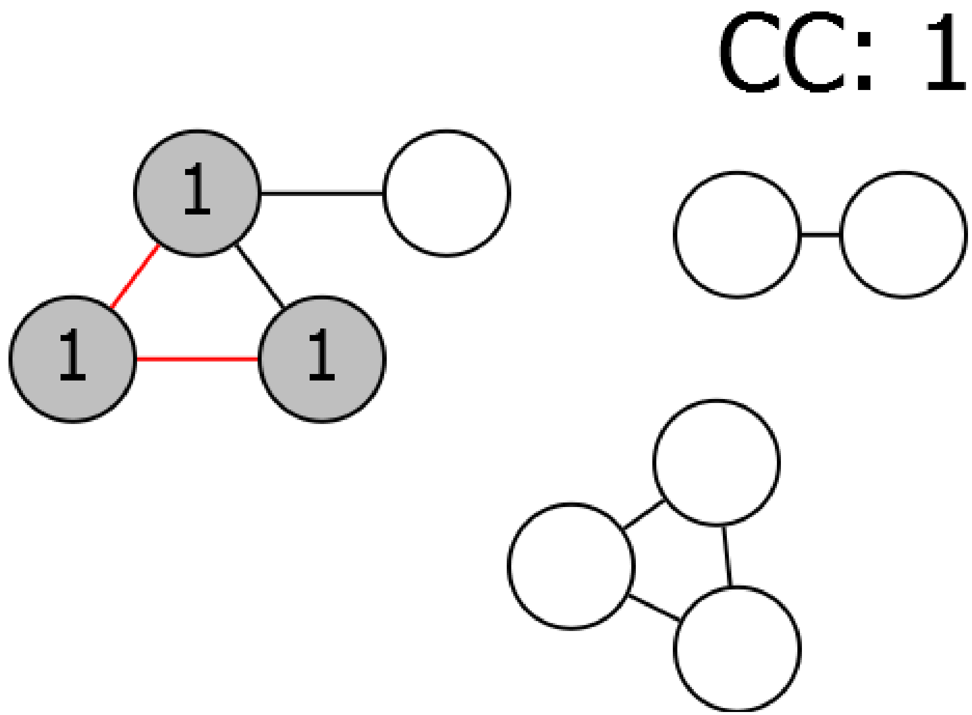
Example



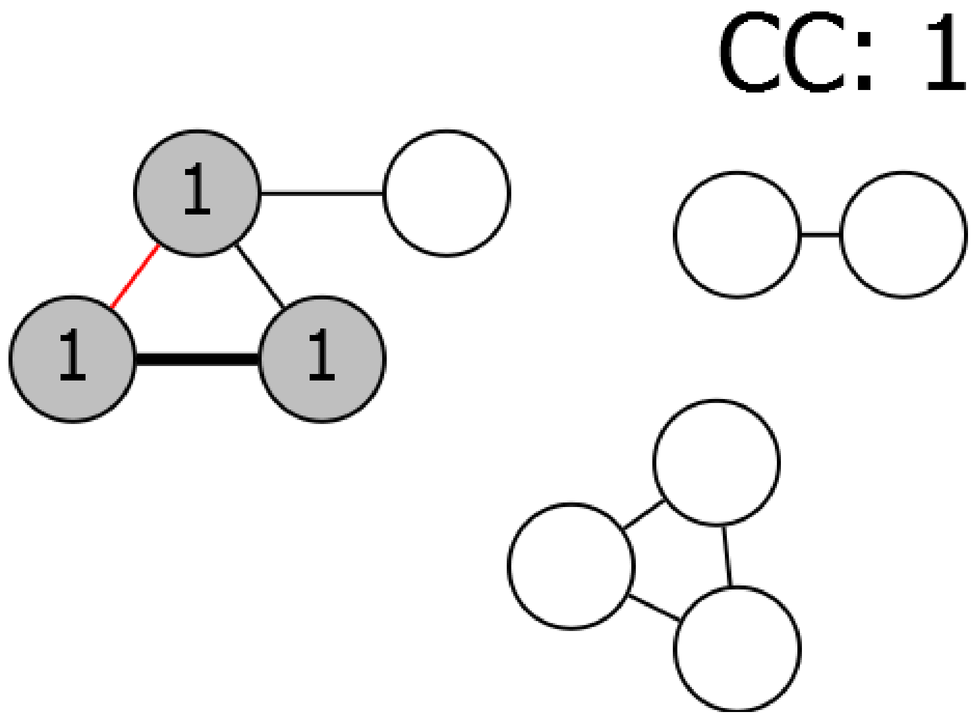
Example



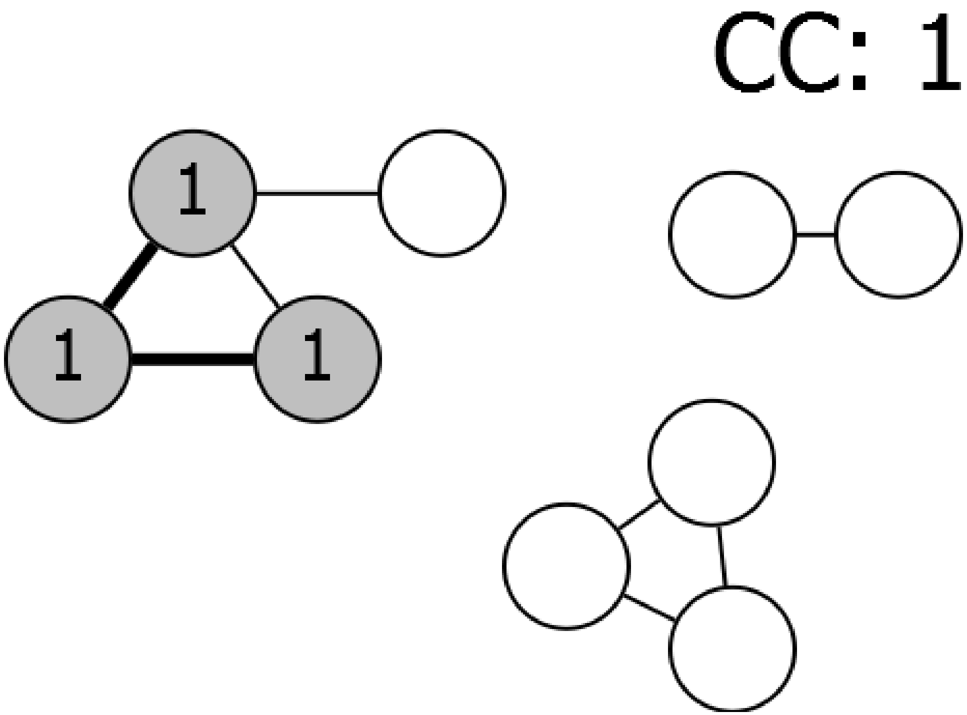
Example



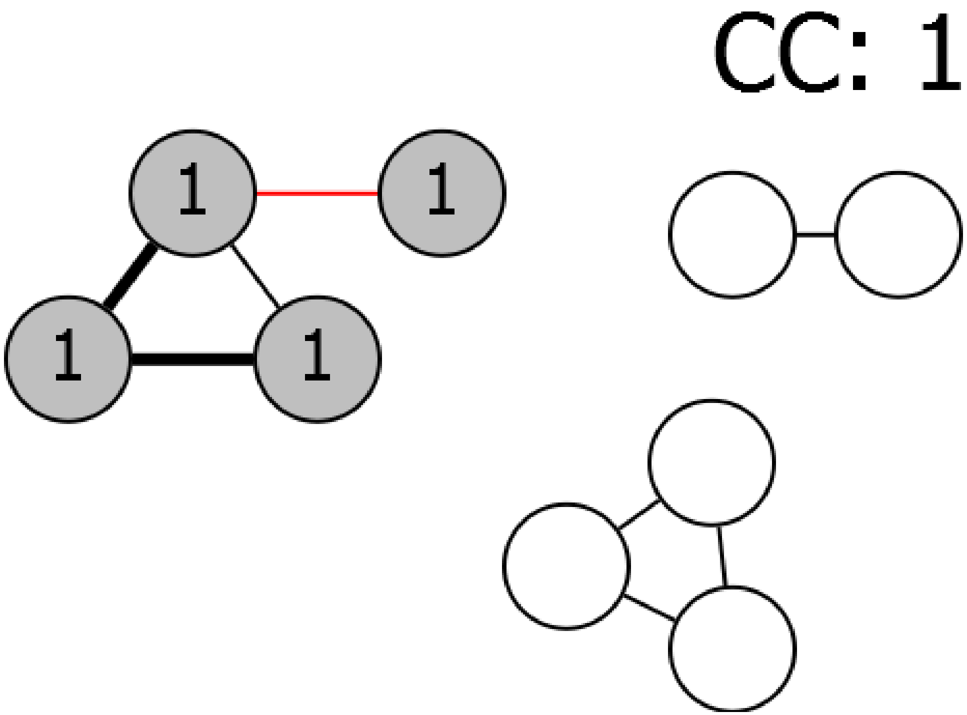
Example



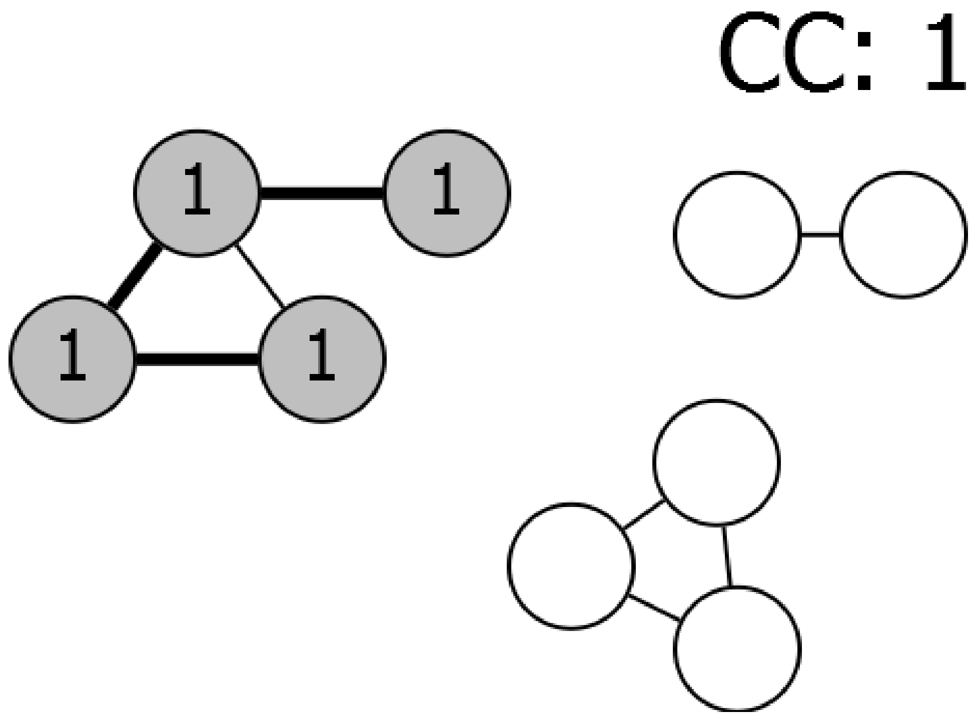
Example



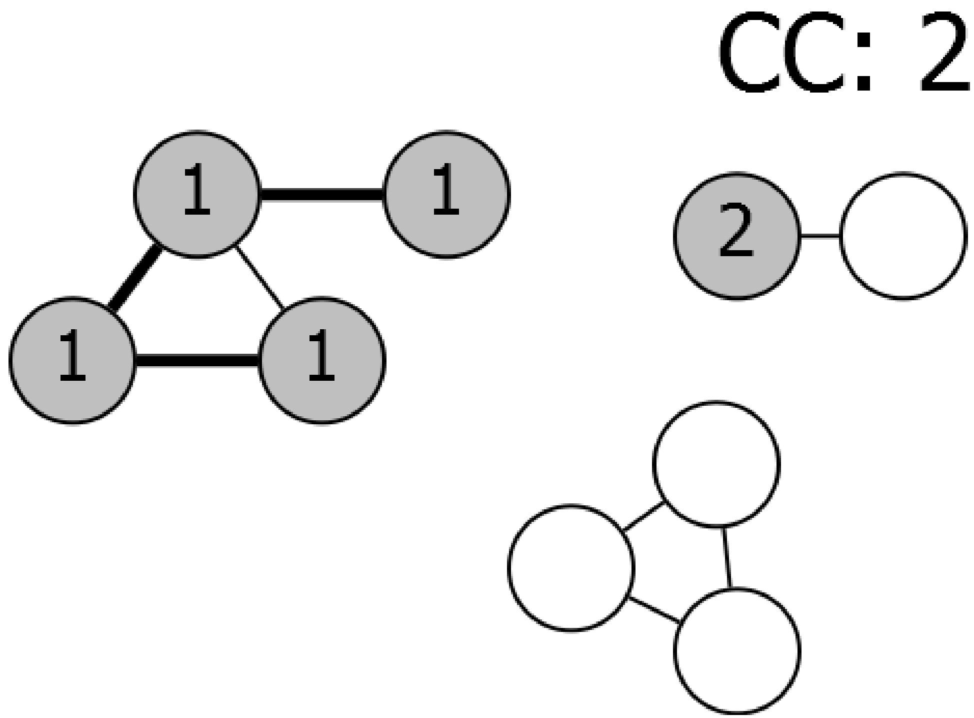
Example



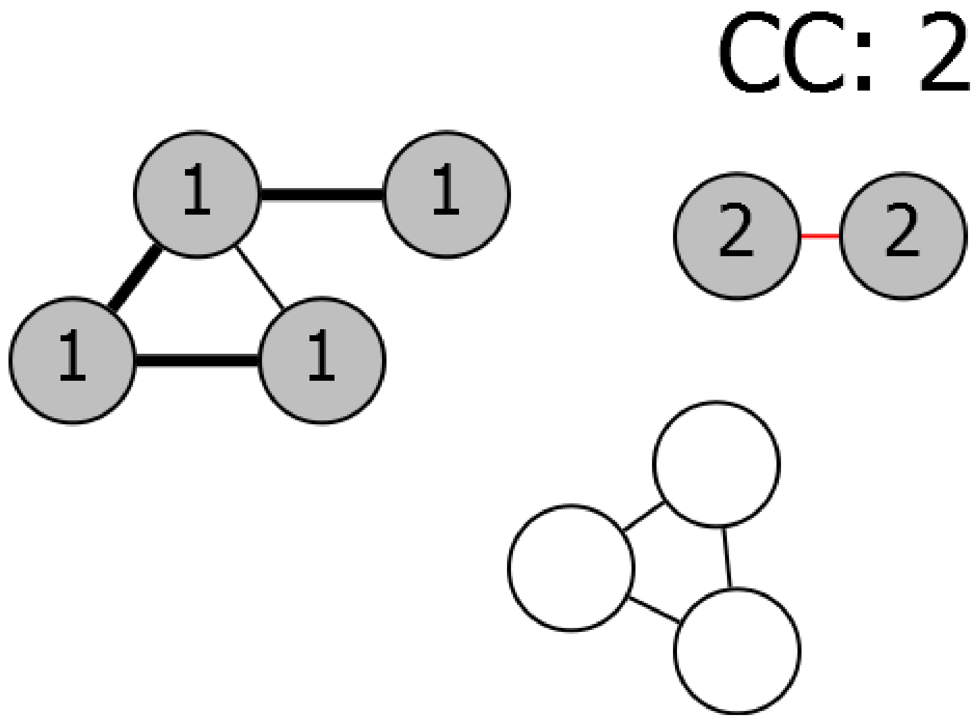
Example



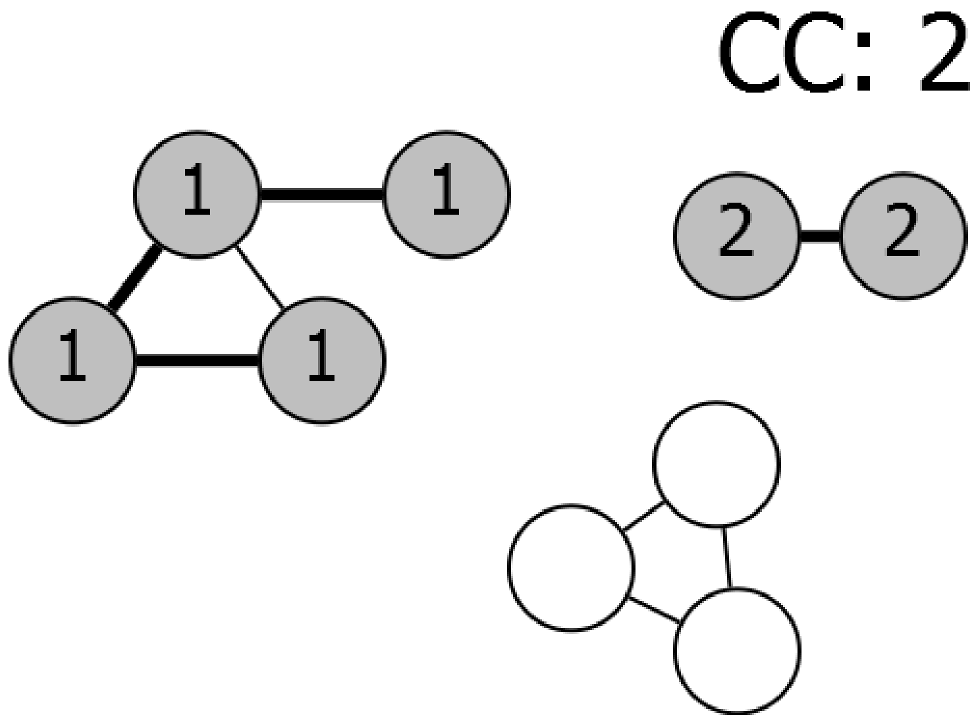
Example



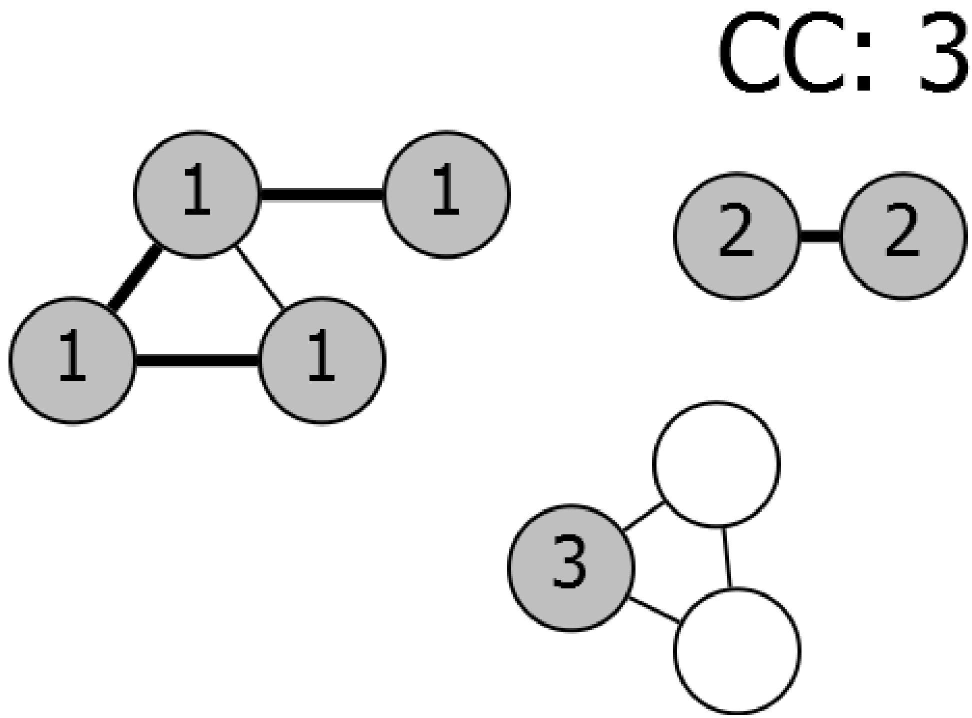
Example



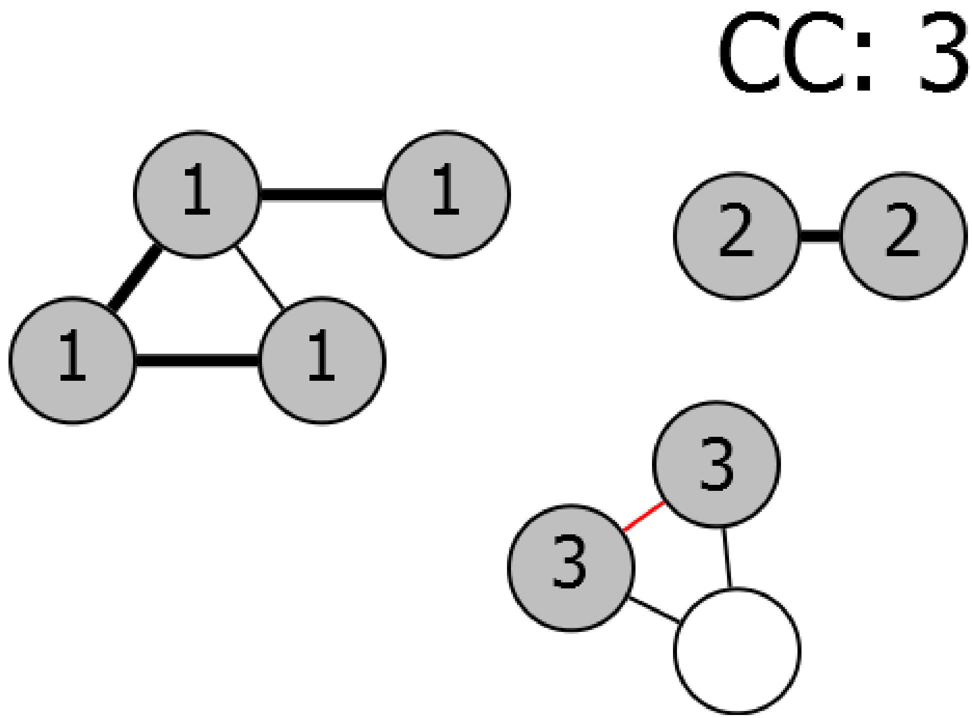
Example



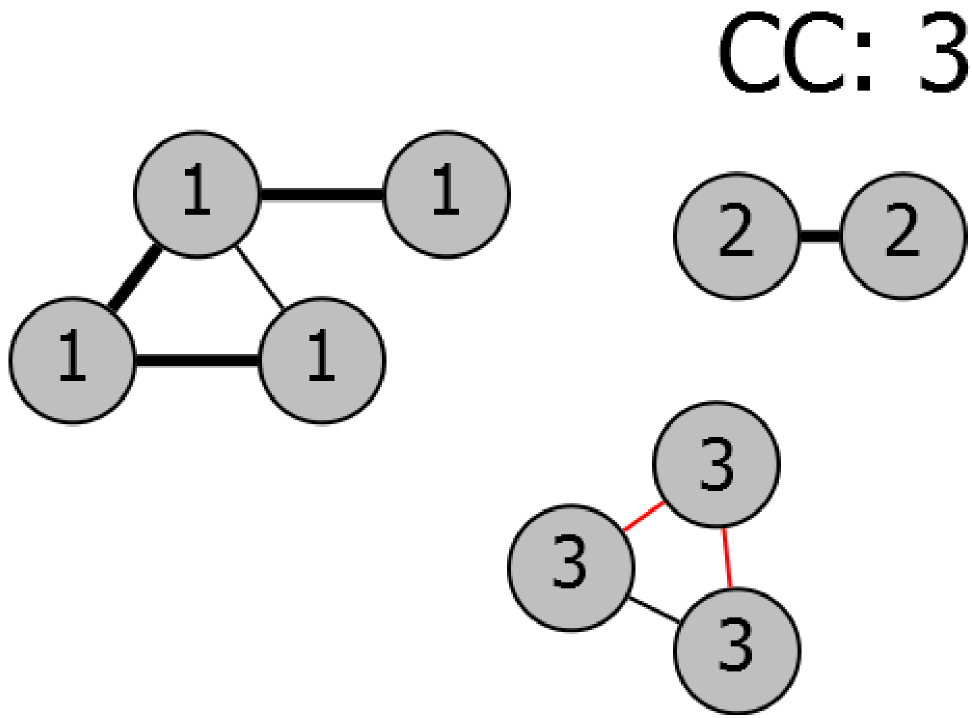
Example



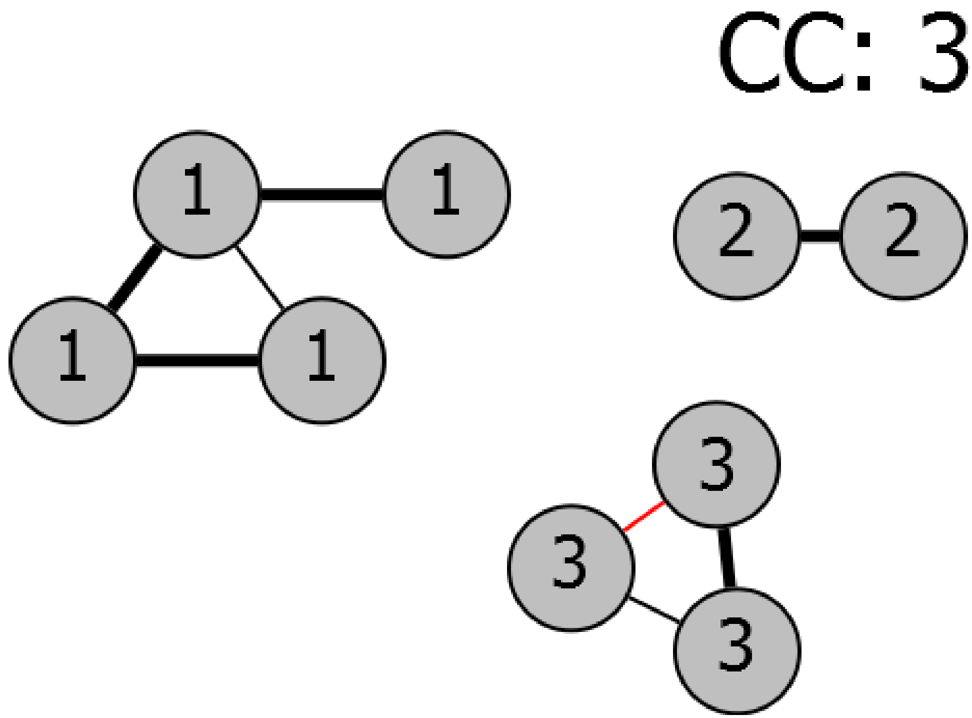
Example



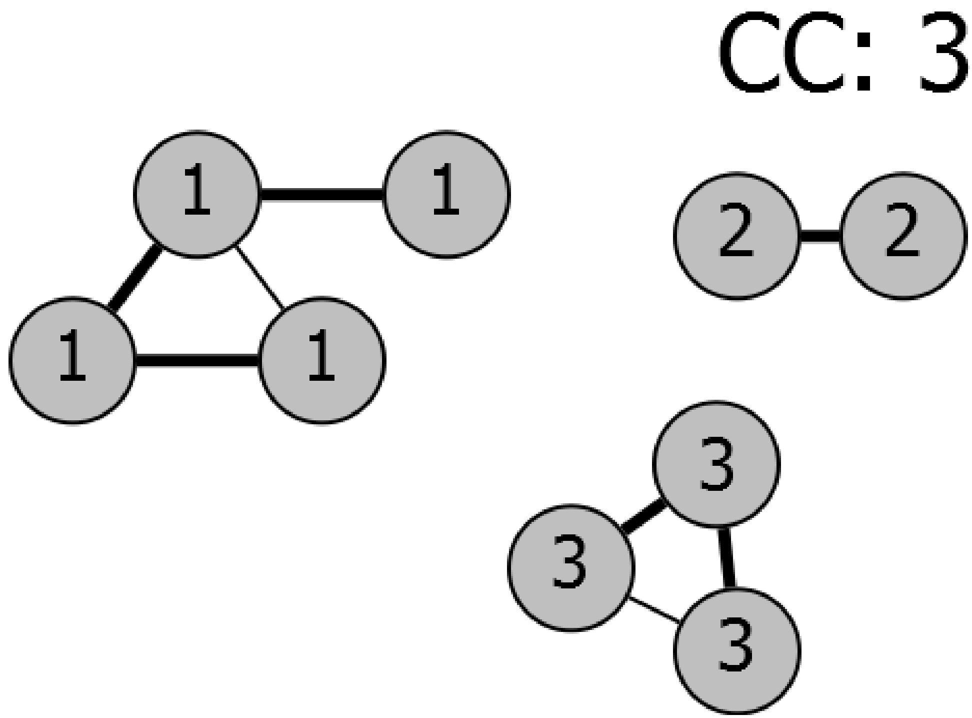
Example



Example



Example



Correctness

- Each new explore finds new connected component.
- Eventually find every vertex
- Runtime still $O(|V|+|E|)$