

# CSE 417

# Algorithms and Complexity

Graphs and Graph Algorithms

Autumn 2023

Lecture 6

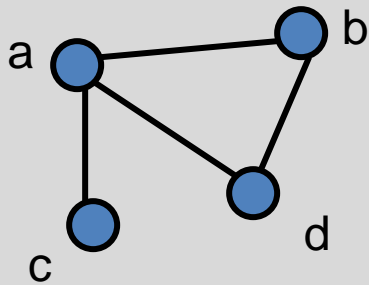
# Announcements

- Reading
  - Chapter 3
  - Start on Chapter 4
- Homework 2

# Graph Theory

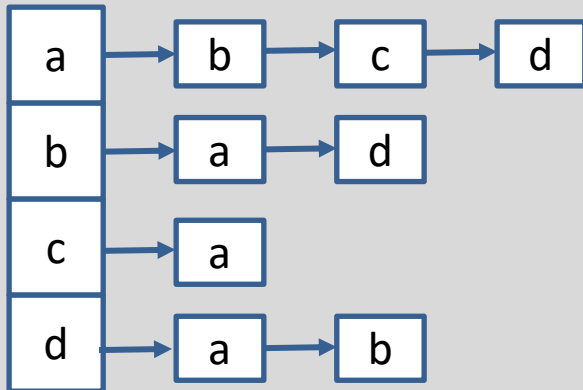
- $G = (V, E)$ 
  - $V$ : vertices,  $|V| = n$
  - $E$ : edges,  $|E| = m$
- Undirected graphs
  - Edges sets of two vertices  $\{u, v\}$
- Directed graphs
  - Edges ordered pairs  $(u, v)$
- Many other flavors
  - Edge / vertices weights
  - Parallel edges
  - Self loops
- Path:  $v_1, v_2, \dots, v_k$ , with  $(v_i, v_{i+1})$  in  $E$ 
  - Simple Path
  - Cycle
  - Simple Cycle
- Neighborhood
  - $N(v)$
- Distance
- Connectivity
  - Undirected
  - Directed (strong connectivity)
- Trees
  - Rooted
  - Unrooted

# Graph Representation



$$V = \{ a, b, c, d \}$$

$$E = \{ \{a, b\}, \{a, c\}, \{a, d\}, \{b, d\} \}$$



Adjacency List

$O(n + m)$  space

	1	1	1
1		0	1
1	0		0
1	1	0	

Incidence Matrix

$O(n^2)$  space

# Implementation Issues

- Graph with  $n$  vertices,  $m$  edges
- Operations
  - Lookup edge
  - Add edge
  - Enumeration edges
  - Initialize graph
- Space requirements

# Graph search

- Find a path from  $s$  to  $t$

$S = \{s\}$

while  $S$  is not empty

$u = \text{Select}(S)$

    visit  $u$

    foreach  $v$  in  $N(u)$

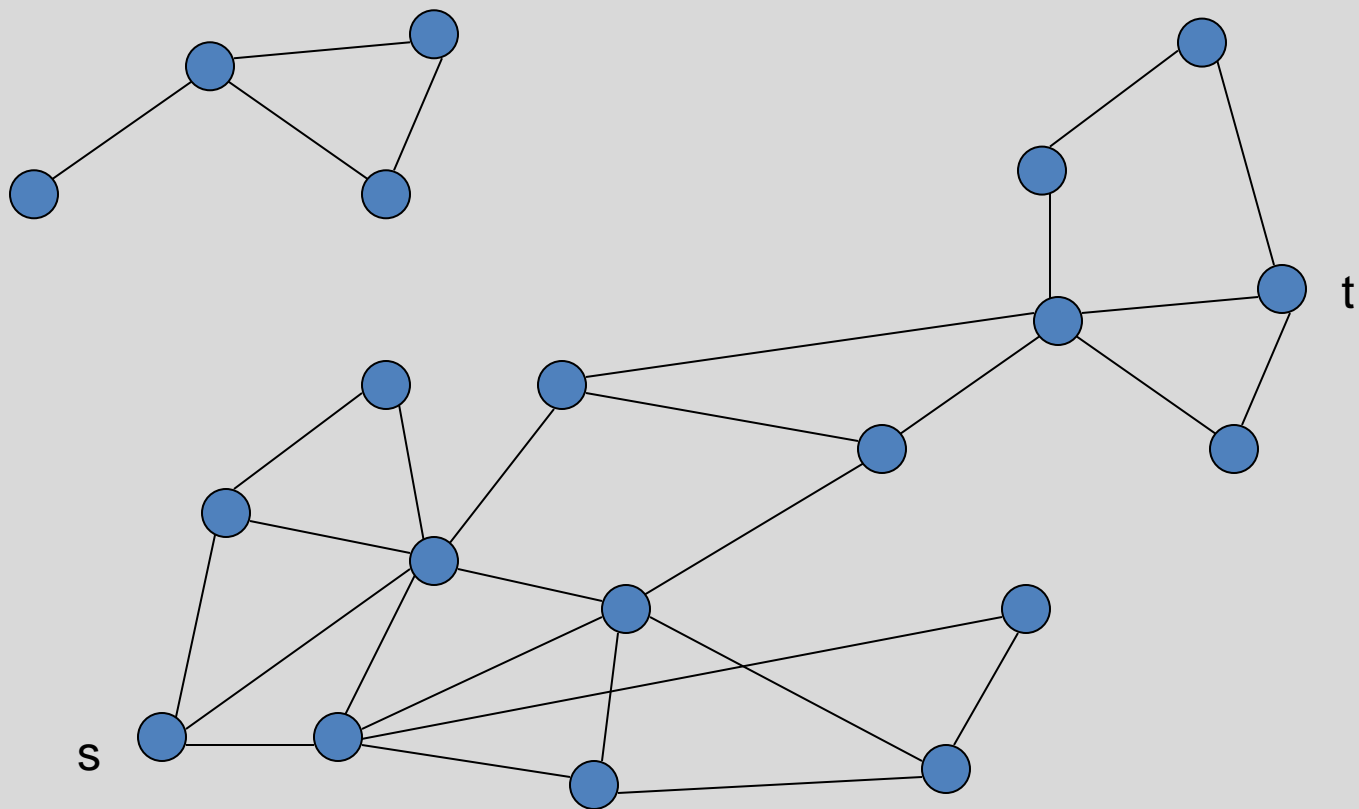
        if  $v$  is unvisited

$\text{Add}(S, v)$

$\text{Pred}[v] = u$

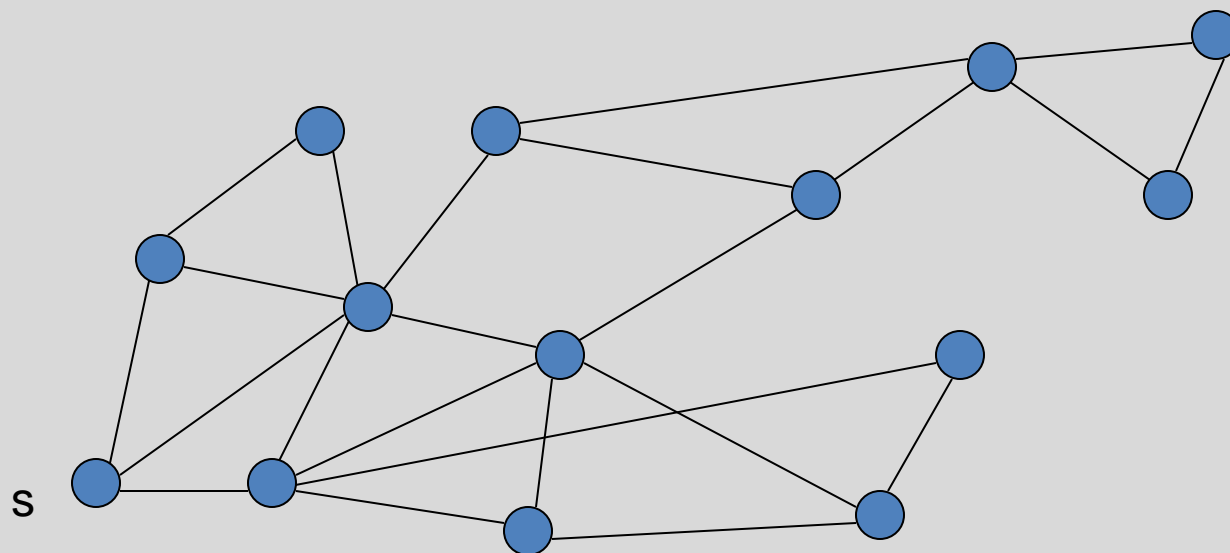
        if ( $v == t$ ) then path found

# Graph Search



# Breadth first search

- Explore vertices in layers
  - $s$  in layer 1
  - Neighbors of  $s$  in layer 2
  - Neighbors of layer 2 in layer 3 . . .





# Breadth First Search

- Build a BFS tree from  $s$

Initialize  $\text{Level}[v] = -1$  for all  $v$ ;

$Q = \{s\}$

$\text{Level}[s] = 1$ ;

while  $Q$  is not empty

$u = Q.\text{Dequeue}()$

    foreach  $v$  in  $N(u)$

        if ( $\text{Level}[v] == -1$ )

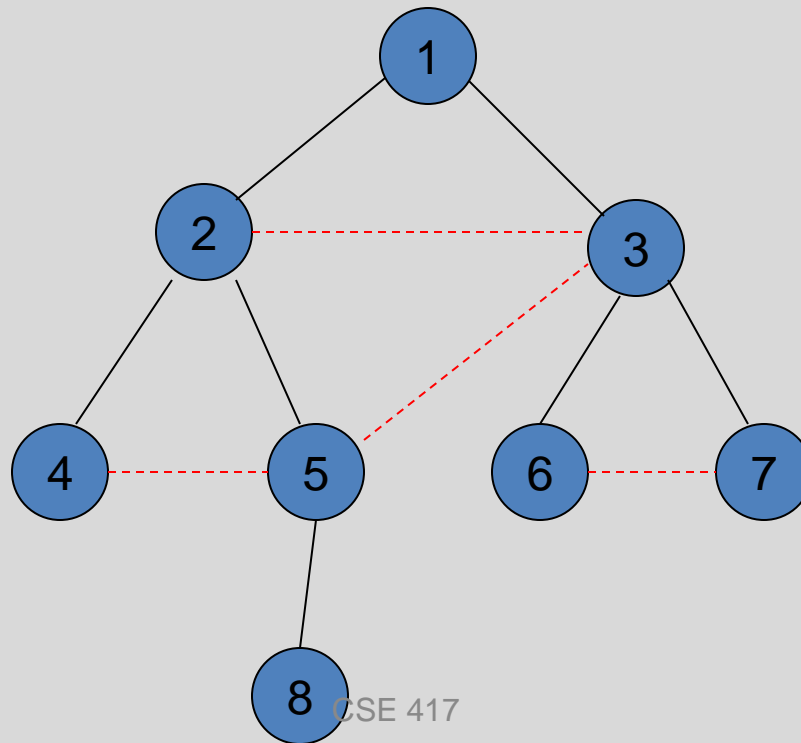
$Q.\text{Enqueue}(v)$

$\text{Pred}[v] = u$

$\text{Level}[v] = \text{Level}[u] + 1$

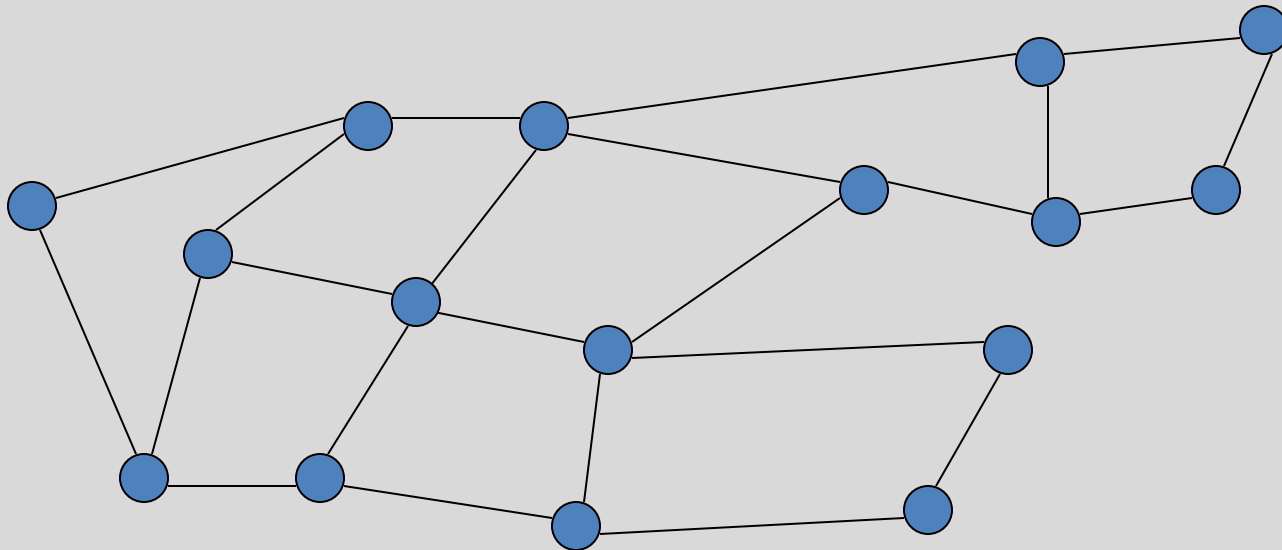
# Key observation

- All edges go between vertices on the same layer or adjacent layers



# Bipartite Graphs

- A graph  $V$  is bipartite if  $V$  can be partitioned into  $V_1, V_2$  such that all edges go between  $V_1$  and  $V_2$  and  $V_2$
- A graph is bipartite if it can be two colored





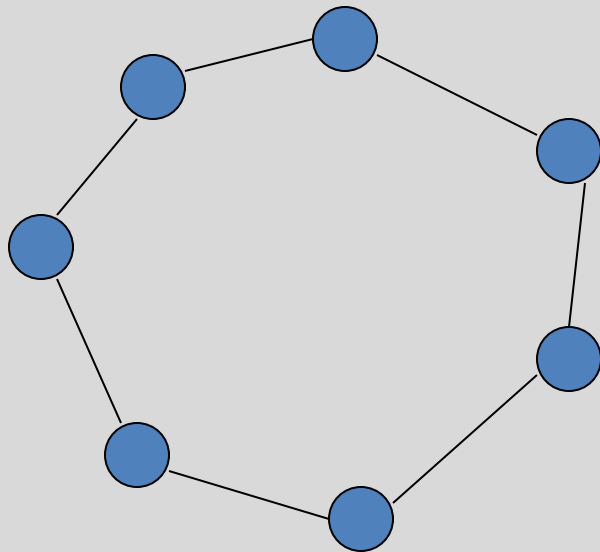
# Algorithm

- Run BFS
- Color odd layers red, even layers blue
- If no edges between the same layer, the graph is bipartite
- If edge between two vertices of the same layer, then there is an odd cycle, and the graph is not bipartite

Theorem: A graph is bipartite if and only if  
it has no odd cycles

# Lemma 1

- If a graph contains an odd cycle, it is not bipartite



# Lemma 2

- If a BFS tree has an *intra-level edge*, then the graph has an odd length cycle

Intra-level edge: both end points are in the same level

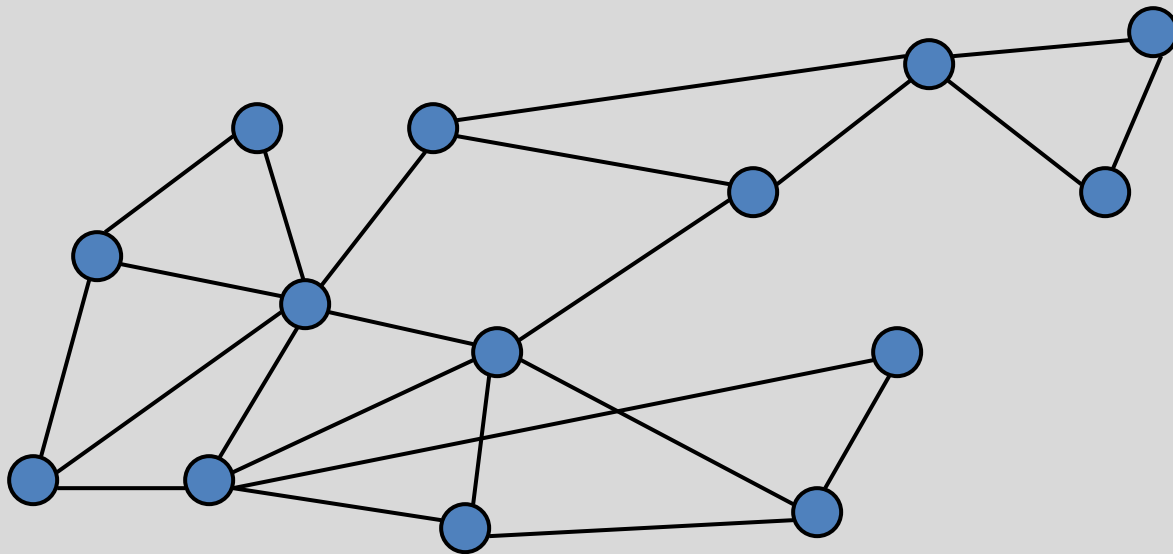


# Lemma 3

- If a graph has no odd length cycles, then it is bipartite

# Graph Search

- Data structure for next vertex to visit determines search order



# Graph search

## Breadth First Search

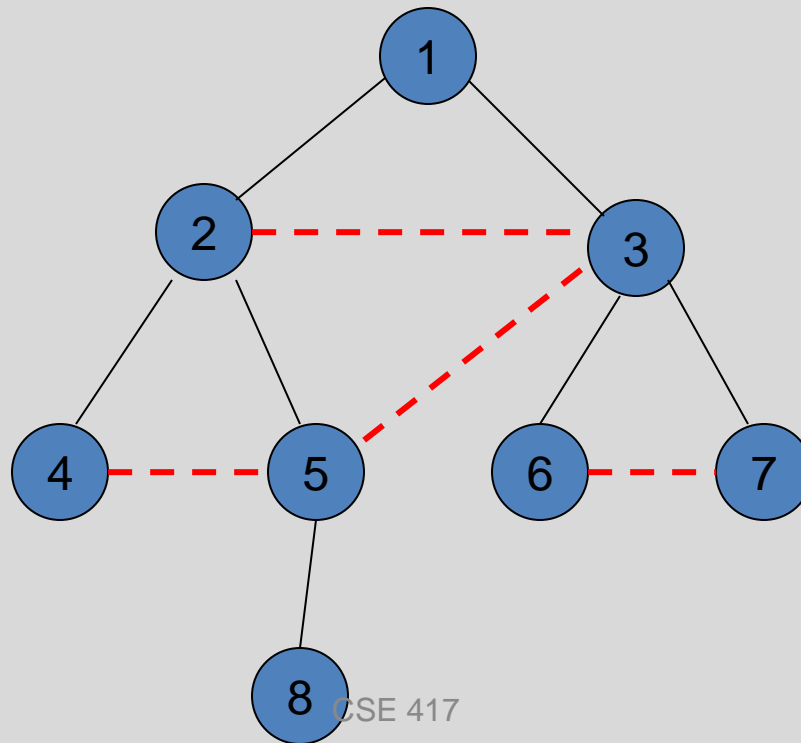
```
S = {s}
while S is not empty
    u = Dequeue(S)
    if u is unvisited
        visit u
        foreach v in N(u)
            Enqueue(S, v)
```

## Depth First Search

```
S = {s}
while S is not empty
    u = Pop(S)
    if u is unvisited
        visit u
        foreach v in N(u)
            Push(S, v)
```

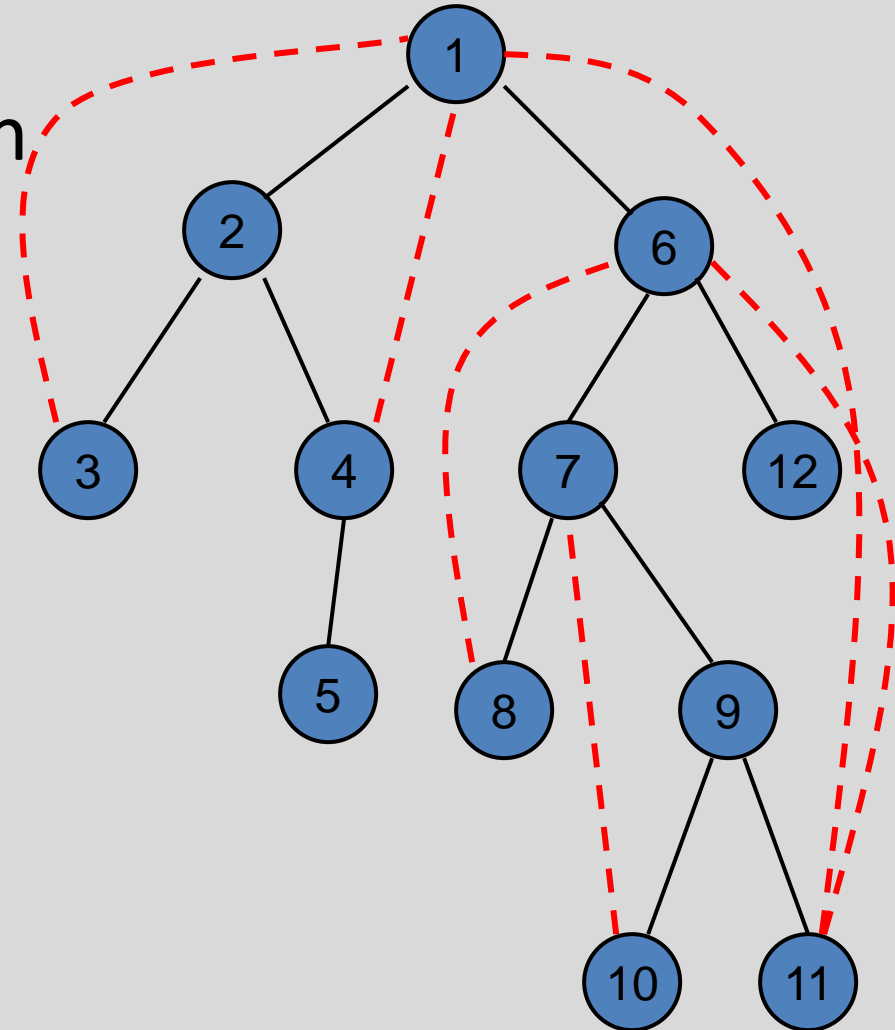
# Breadth First Search

- All edges go between vertices on the same layer or adjacent layers



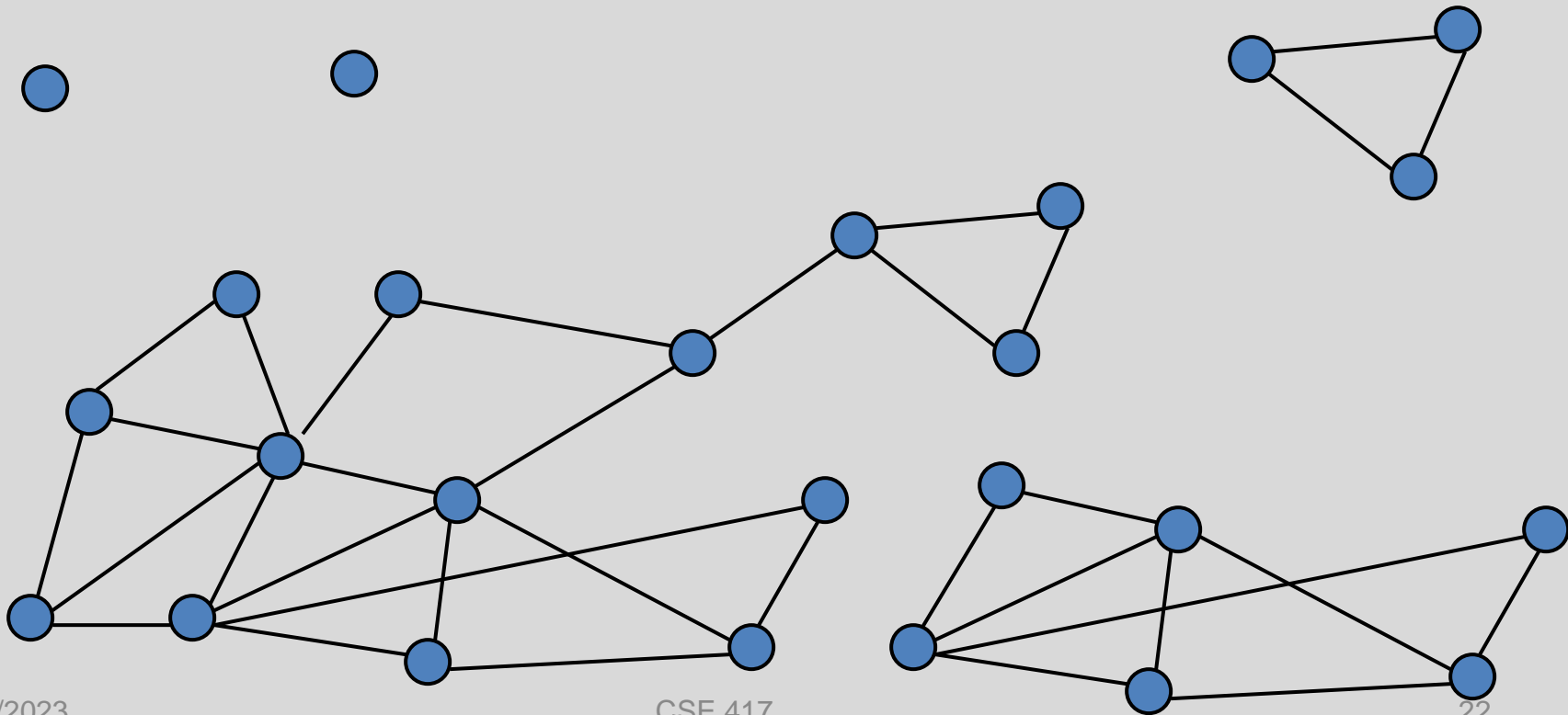
# Depth First Search

- Each edge goes between vertices on the same branch
- No cross edges



# Connected Components

- Undirected Graphs

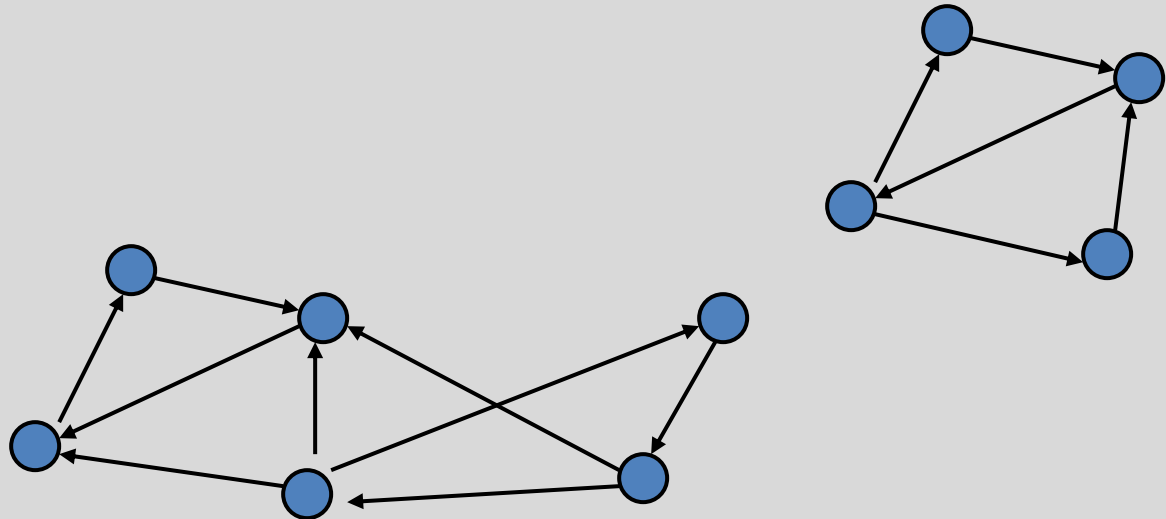


# Computing Connected Components in $O(n+m)$ time

- A search algorithm from a vertex  $v$  can find all vertices in  $v$ 's component
- While there is an unvisited vertex  $v$ , search from  $v$  to find a new component

# Directed Graphs

- A Strongly Connected Component is a subset of the vertices with paths between every pair of vertices.





# Identify the Strongly Connected Components

