CSE 417 Autumn 2025

Lecture 9: Graph search

Glenn Sun

Why graphs?

Graphs can be used to **model many real-world situations**.

Applications for just today:

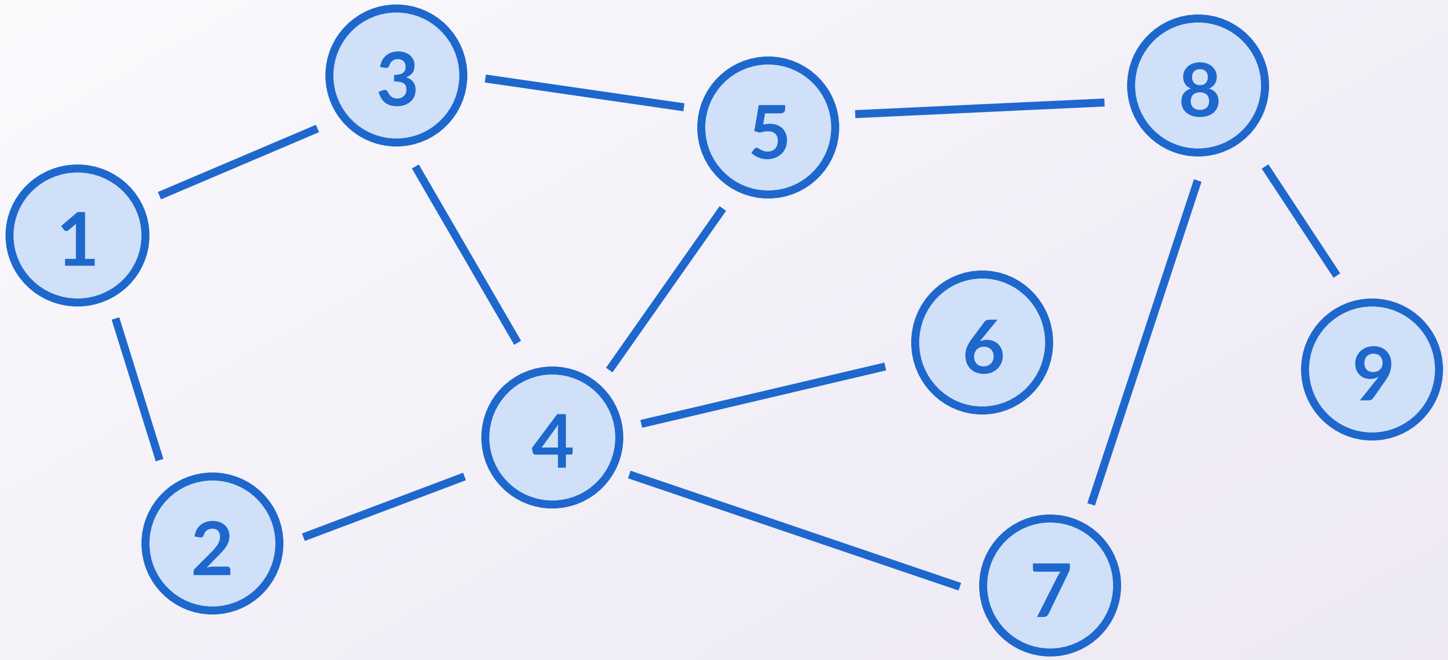
- MS Paint bucket fill tool
- LinkedIn 1st/2nd/3rd degree connections
- Prerequisite planning in universities

Basics about graphs

What is a graph?

A **graph** is a set of **vertices** V and a set of **edges** E connecting them.

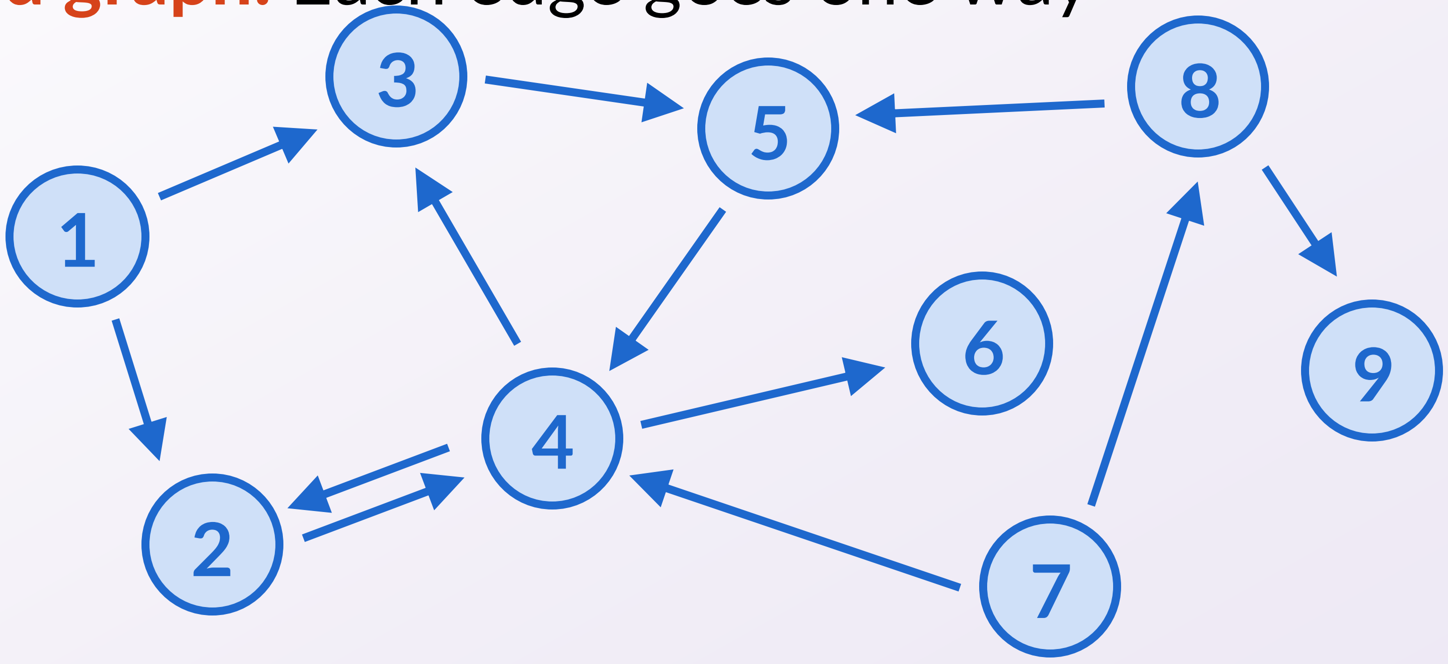
- **Undirected graph:** All edges go both ways



What is a graph?

A **graph** is a set of vertices V and a set of edges E connecting them.

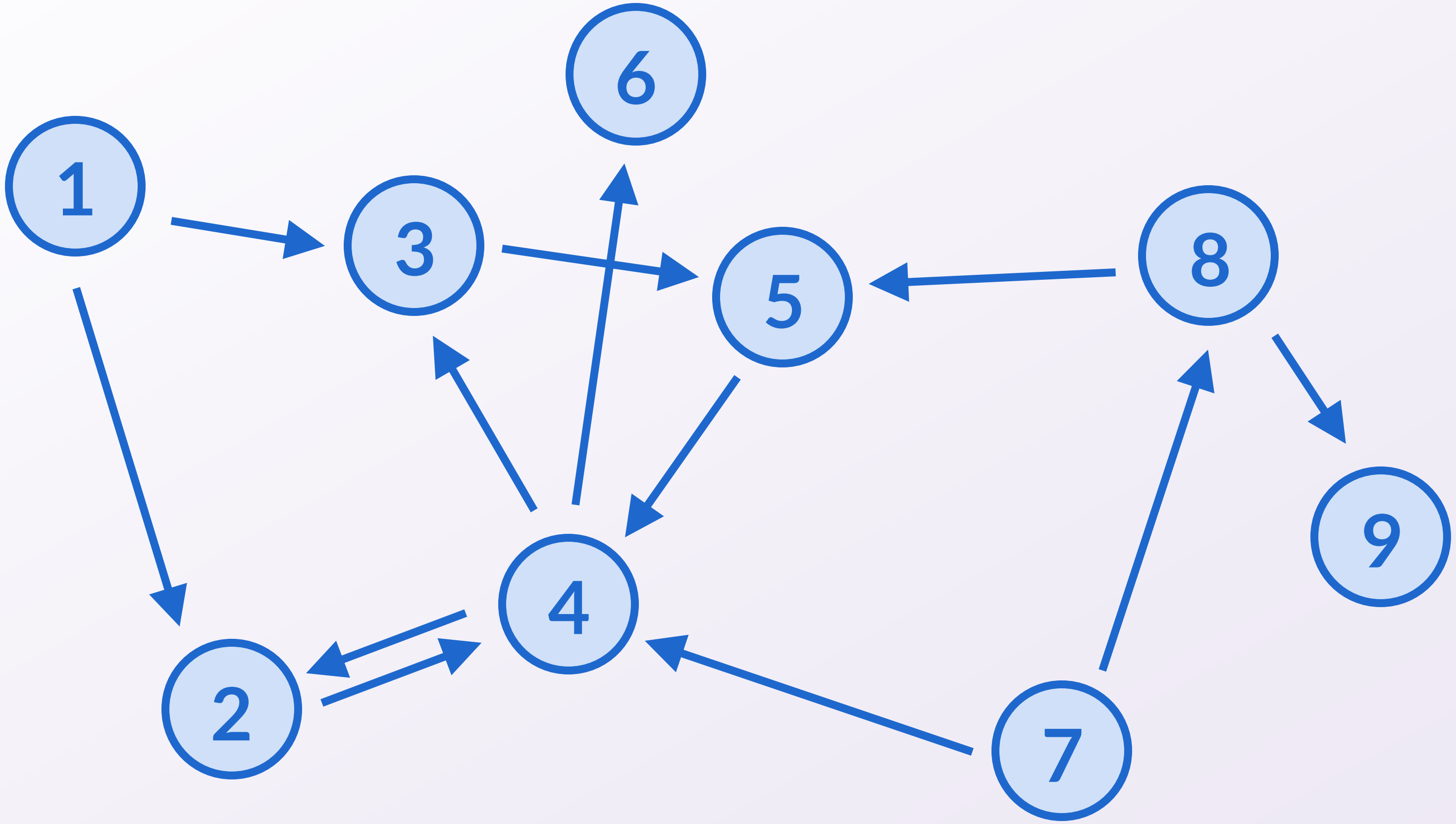
- **Undirected graph:** All edges go both ways
- **Directed graph:** Each edge goes one way



What is a graph?

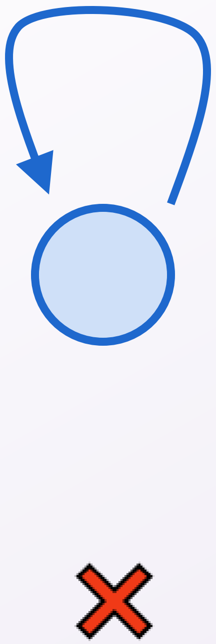
Positioning doesn't matter.

Graph only encodes connectivity.

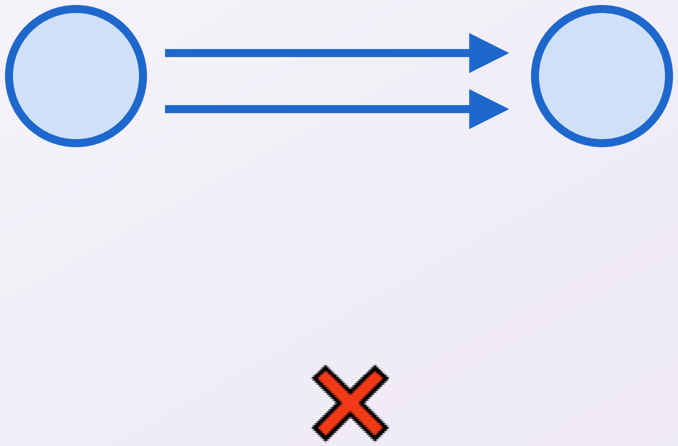


Two default assumptions

No self-loops

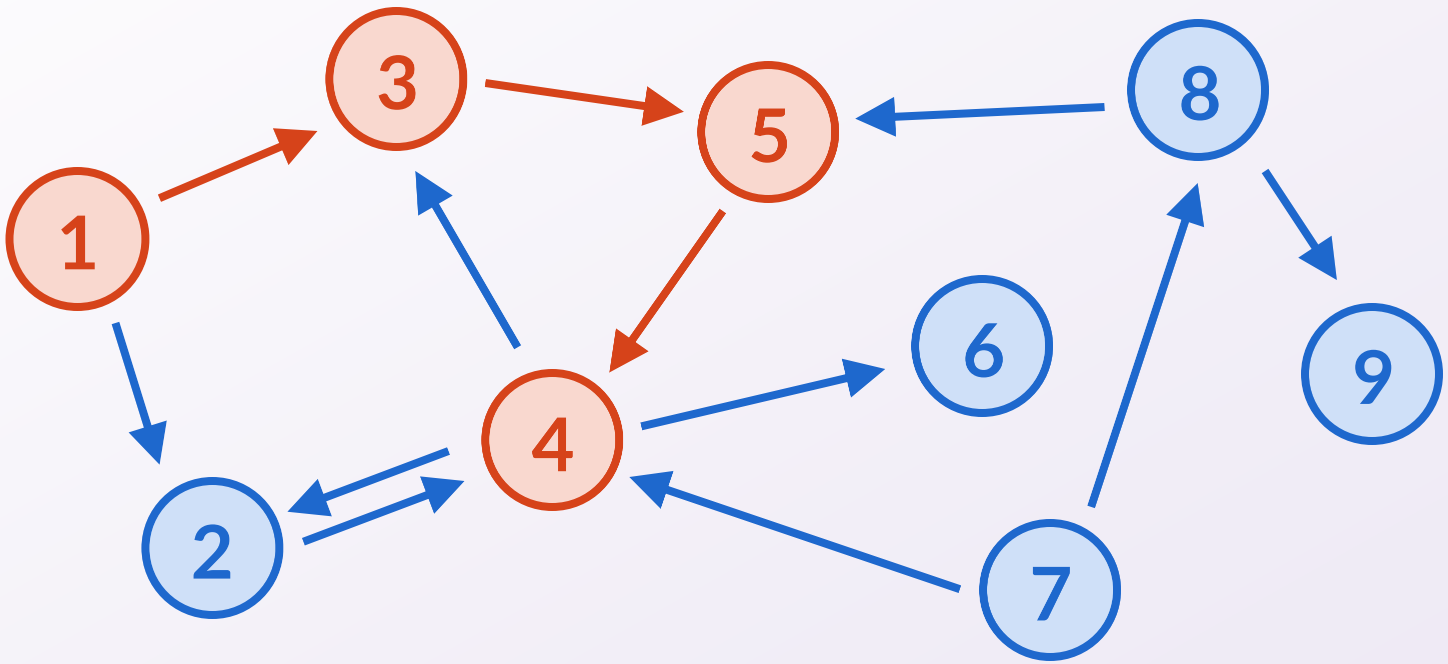


No “parallel” edges



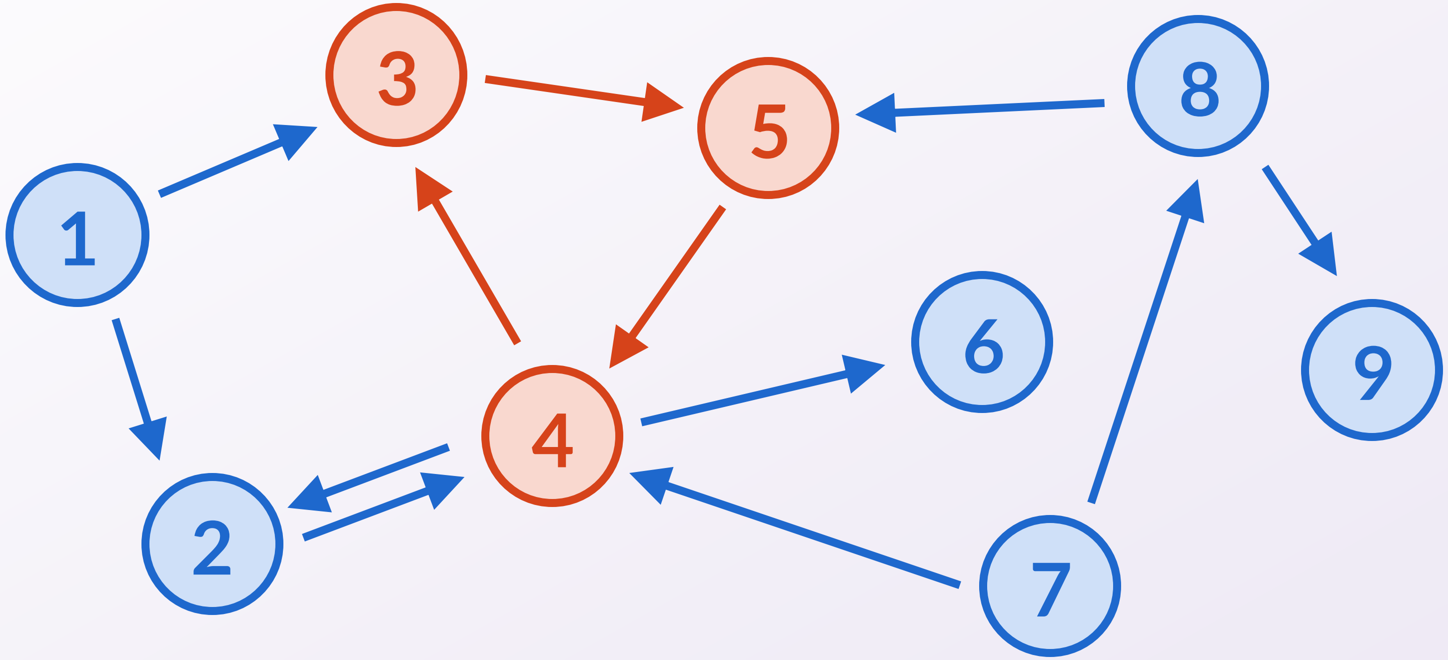
Some basic vocabulary

Path: sequence of ***distinct*** vertices connected by edges



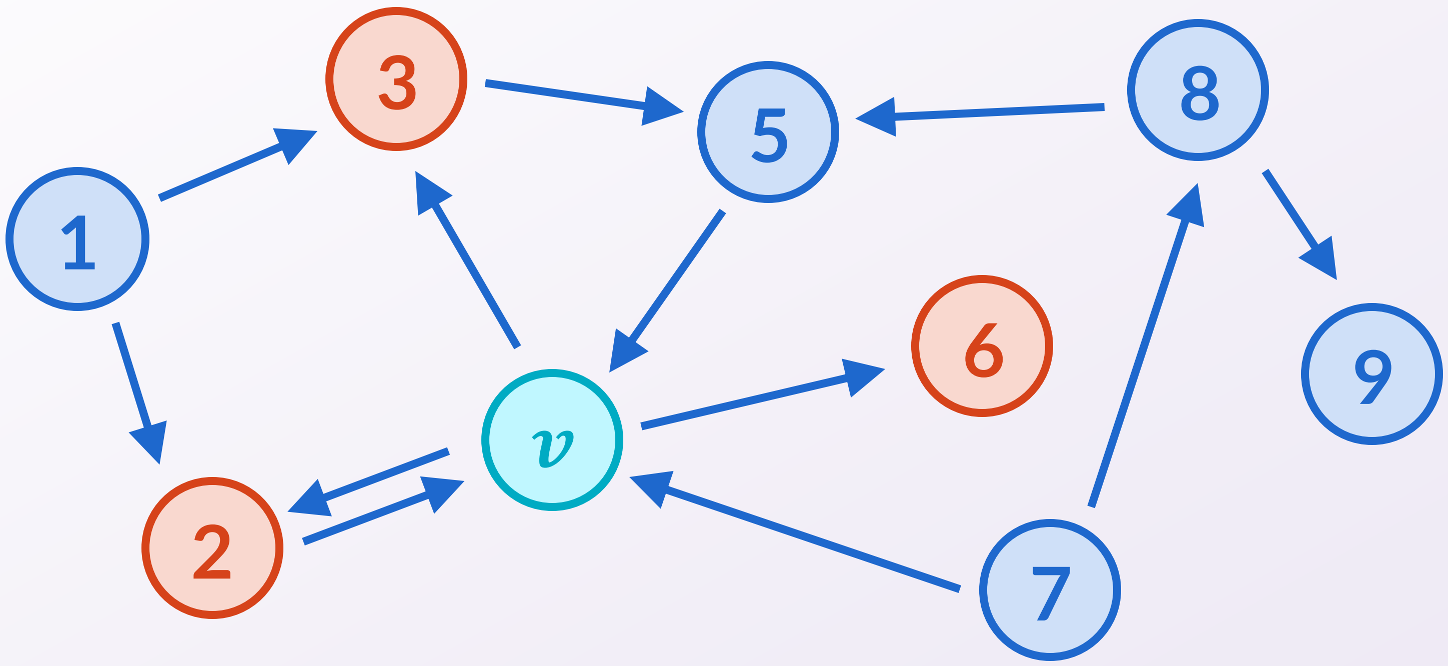
Some basic vocabulary

Cycle: sequence of ***distinct*** vertices connected by edges, except the first and last vertex are the same



Some basic vocabulary

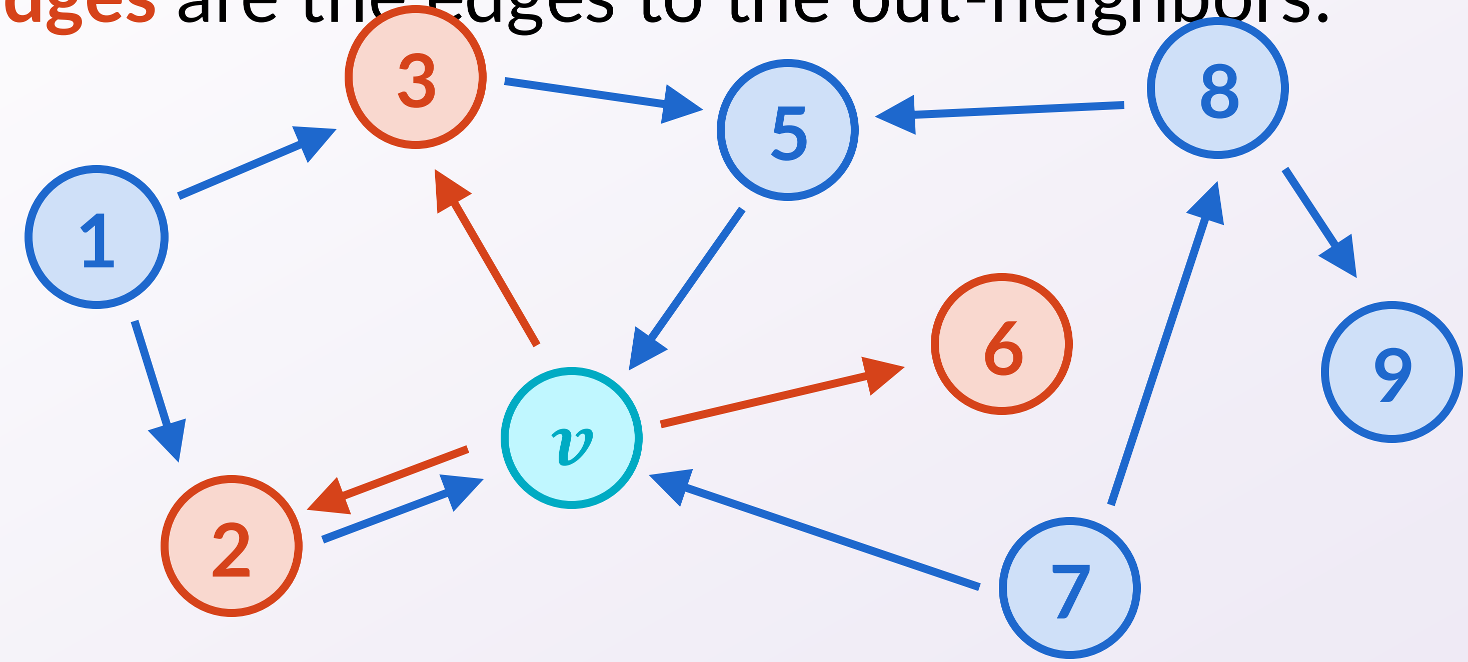
The **out-neighbors** of v are the vertices that v can go to by an edge.



Some basic vocabulary

The **out-neighbors** of v are the vertices that v **can go to** by an edge.

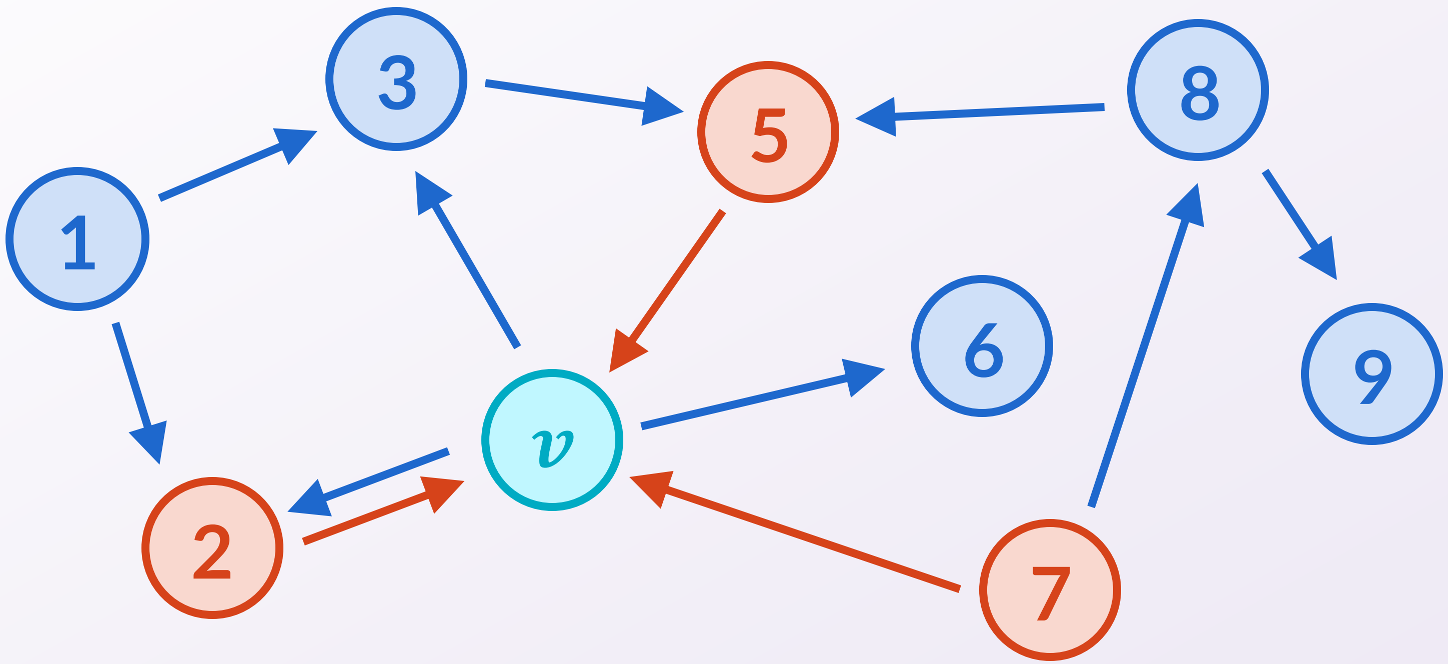
The **out-edges** are the edges to the out-neighbors.



Some basic vocabulary

The **in-neighbors** of v are the vertices that **can go to** v by an edge.

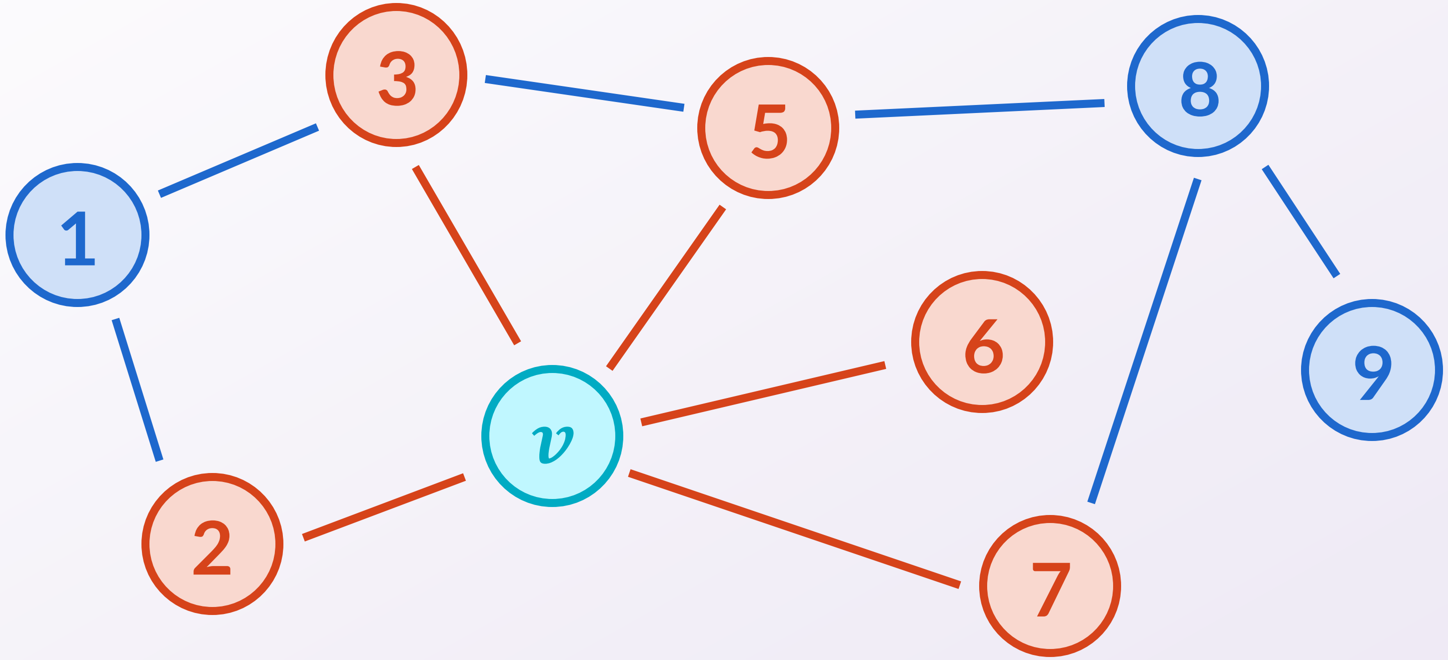
The **in-edges** are the edges to the in-neighbors.



Some basic vocabulary

For undirected graphs, all edges go both directions.

We often just say "**neighbors** of v" and "**edges adjacent** to v".



Some basic vocabulary

Careful! The word “connected” is ambiguous.

u and v are connected

Some basic vocabulary

Instead, say:

For undirected edges:

- $\{u, v\}$ is an edge
- u and v are neighbors
- u and v are adjacent

For directed edges:

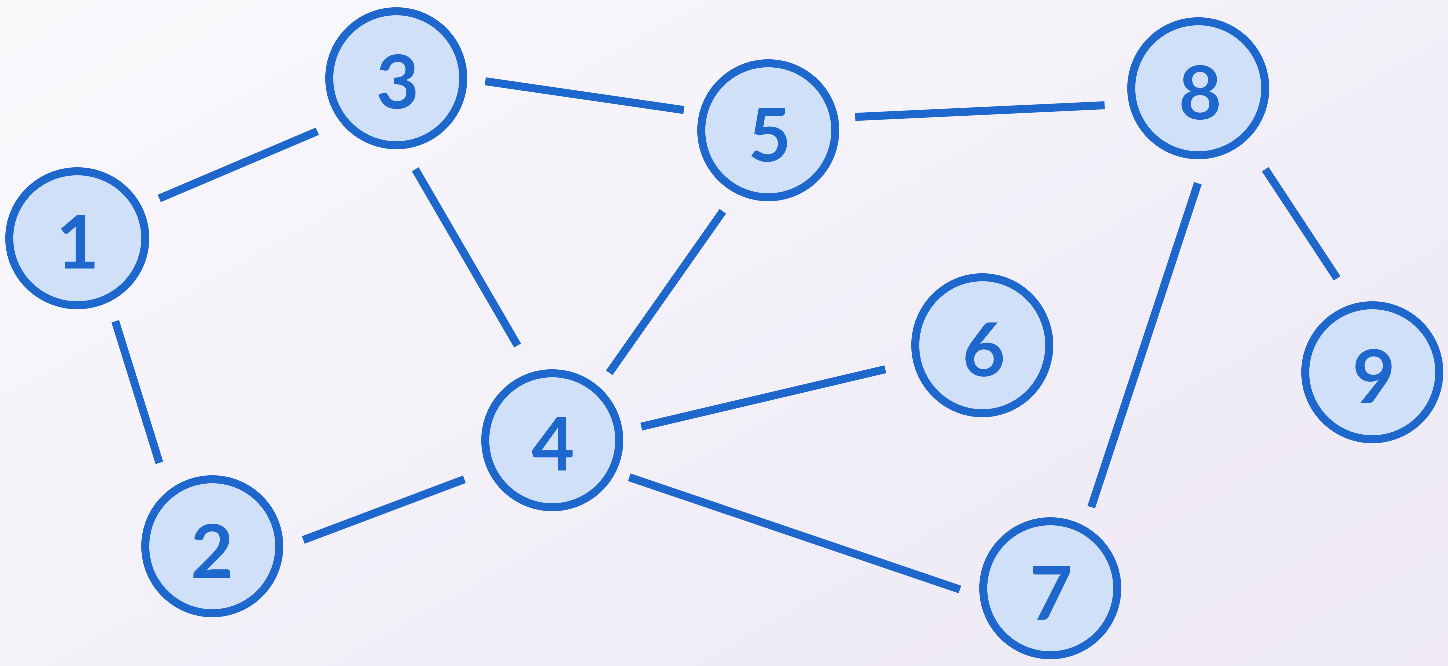
- (u, v) is an edge
- v is an out-neighbor of u

For paths:

- there is a path from u to v

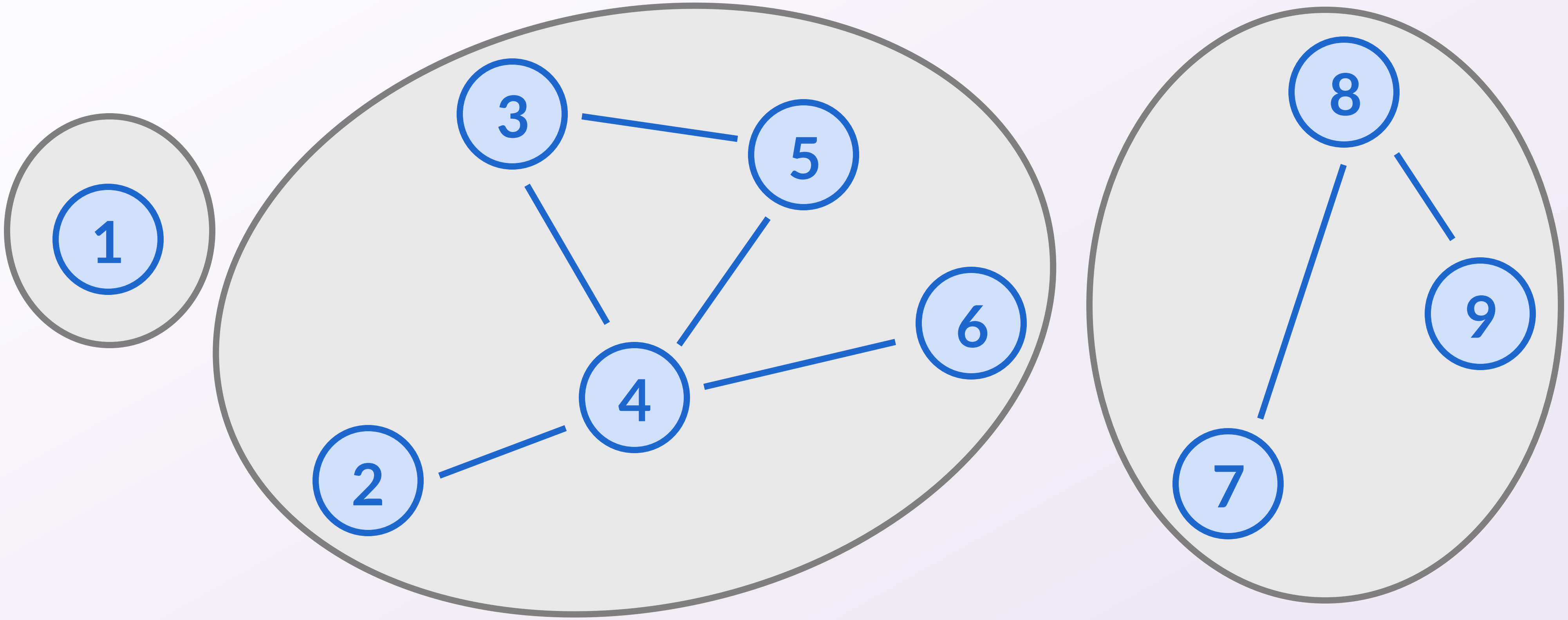
Some basic vocabulary

An undirected graph is **connected** if you can go from any vertex to any other vertex.



Some basic vocabulary

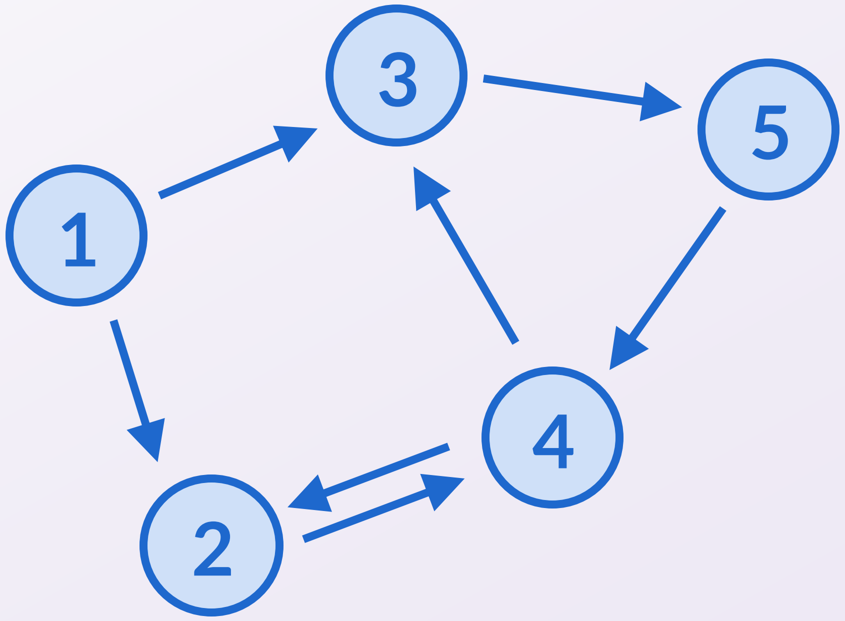
If the graph is not connected, it can be broken down into **connected components**, which are maximal connected subgraphs (means you can't add vertices without becoming disconnected).



How graphs work in programming

We use **adjacency list** representation:
list of out-neighbors for every vertex.

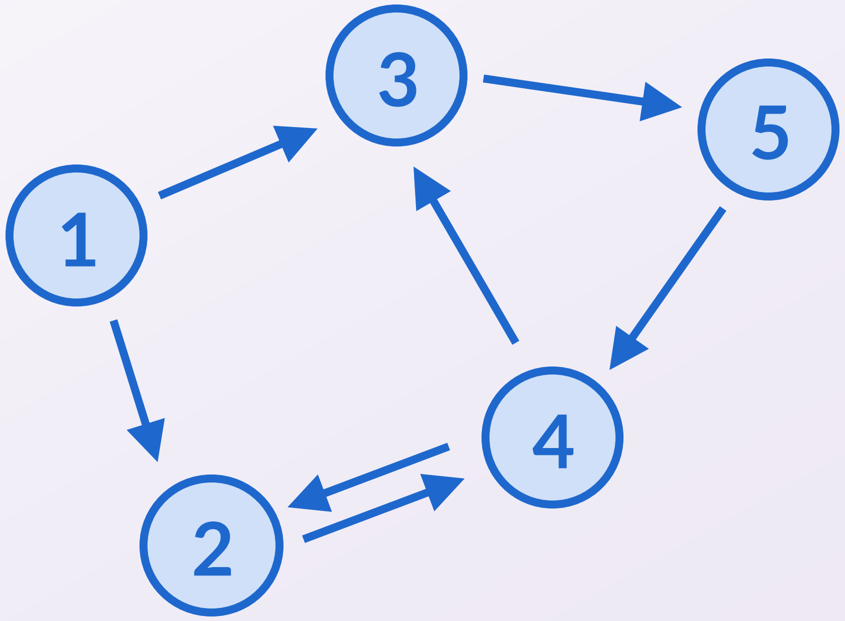
out[1] = [2, 3]
out[2] = [4]
out[3] = [5]
out[4] = [2, 3]
out[5] = [4]



How graphs work in programming

A **bidirectional adjacency list** also lists the in-neighbors separately.

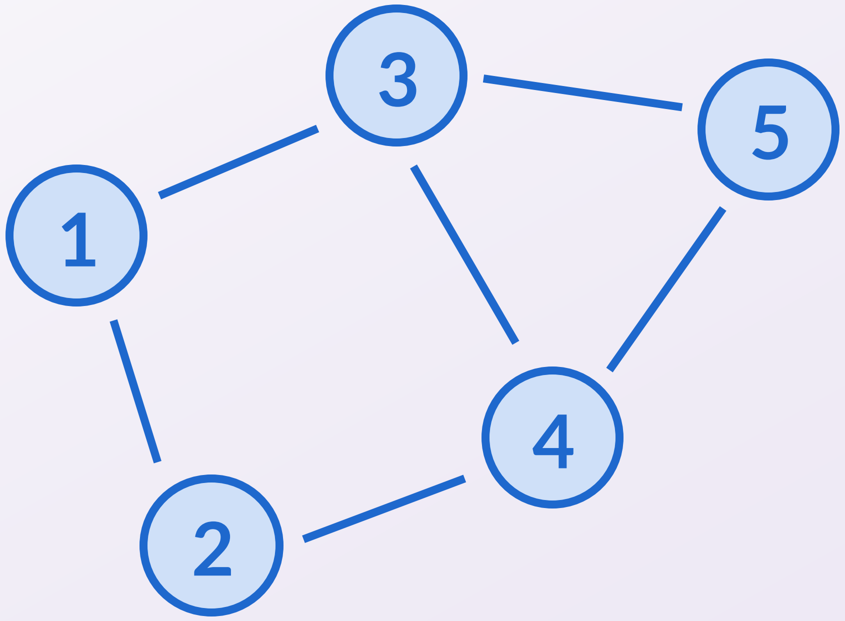
out[1] = [2, 3]	in[1] = []
out[2] = [4]	in[2] = [1, 4]
out[3] = [5]	in[3] = [1, 4]
out[4] = [2, 3]	in[4] = [2, 5]
out[5] = [4]	in[5] = [3]



How graphs work in programming

For undirected graphs, one adjacency list can be used for both in/out.

neighbors[1] = [2, 3]
neighbors[2] = [1, 4]
neighbors[3] = [1, 4, 5]
neighbors[4] = [2, 3, 5]
neighbors[5] = [3, 4]



BFS and DFS

What is graph search?

Graph search is traversing through a graph by **following edges**.

Input: Graph with vertices V and edge E, starting vertex s

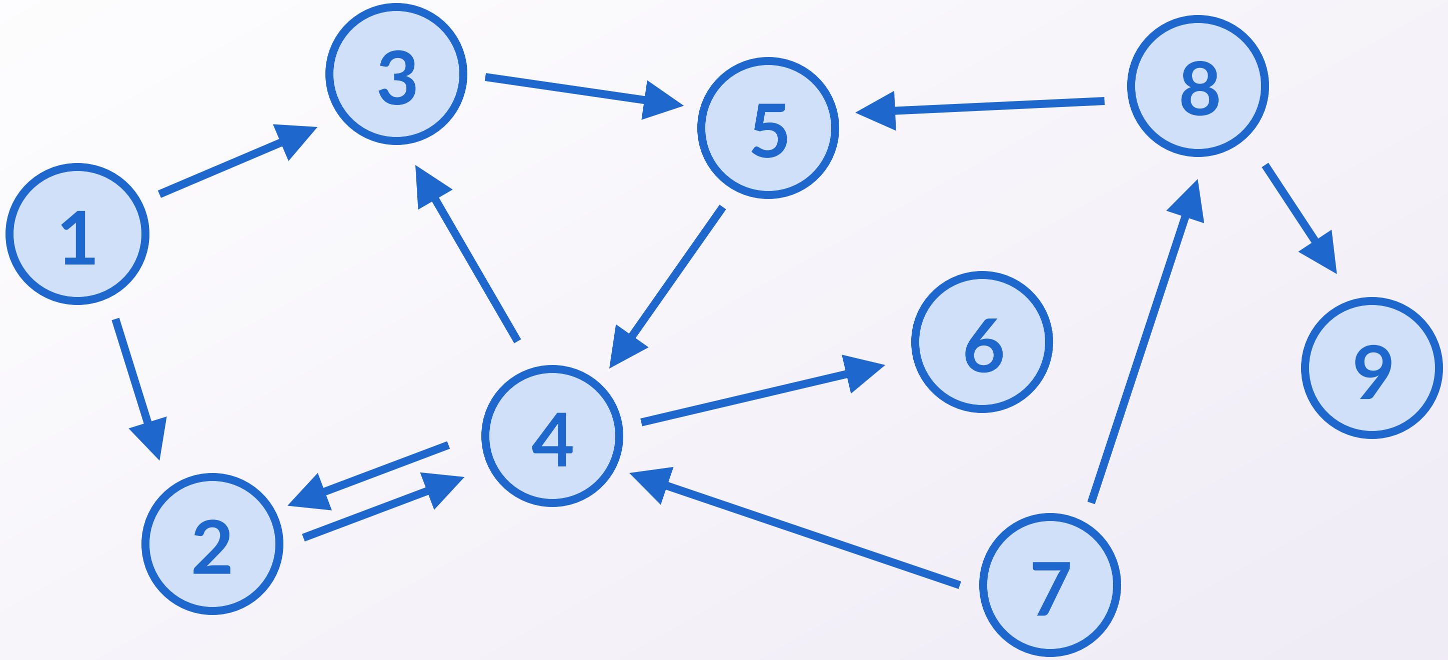
Goal template: Do something at every reachable vertex from s

(Reachable = exists a path from s)

Two basic techniques (intuition):

- **Breadth-first (BFS):** Search vertices close to s first.
- **Depth-first (DFS):** Keep going until you can't, then backtrack.

BFS demonstration



Process order: 123456

BFS pseudocode

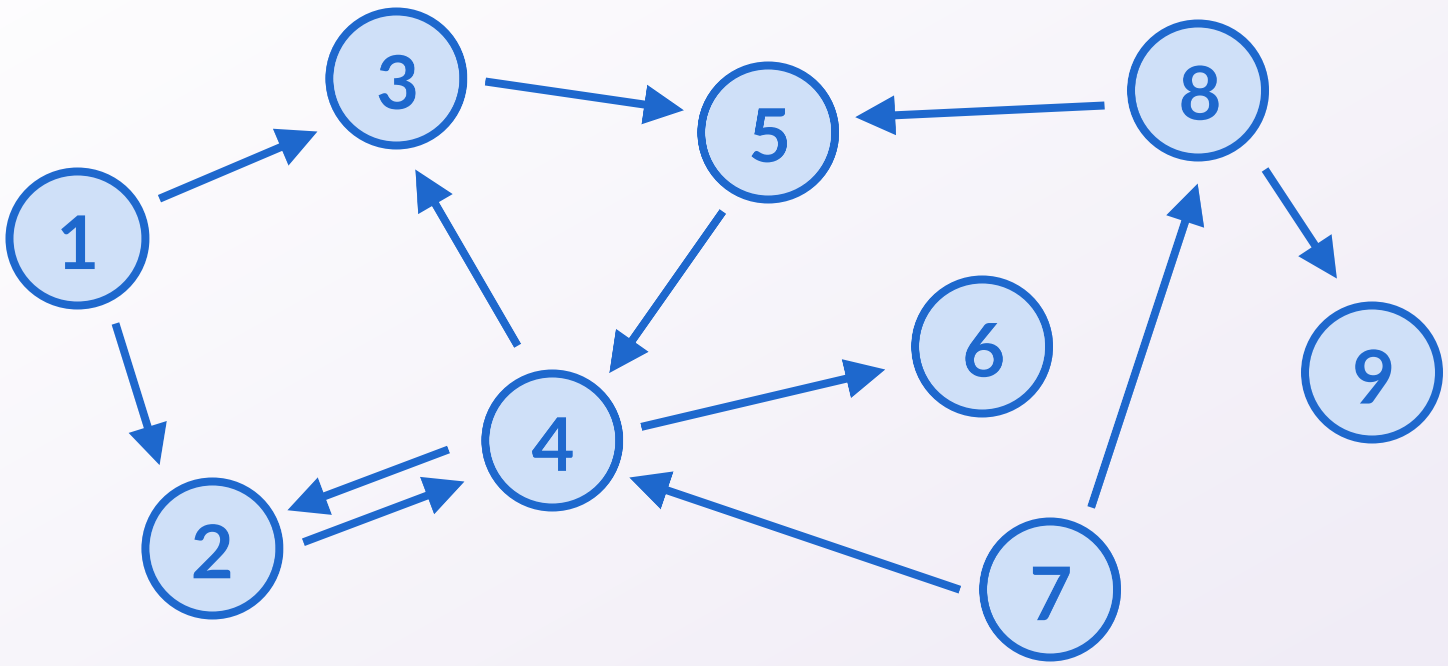
```
Initialize a queue Q with the starting point s.
Mark all vertices "unseen", except s is "seen".
while Q is not empty do
    Get/remove the next vertex x from Q.
    Process x somehow.
    for all unseen out-neighbors y of x do
        Mark y "seen" and add y to Q.
```

BFS pseudocode

1. Initialize a queue Q with the starting point s.
2. Mark all vertices "unseen", except s is "seen".
3. **while** Q is not empty **do**
4. Get/remove the next vertex x from Q.
5. Process x somehow.
6. **for all** unseen out-neighbors y of x **do**
7. Mark y "seen" and add y to Q.

BFS/DFS is not single algorithms, but **families** of algorithms!

DFS demonstration



Process order: 124356

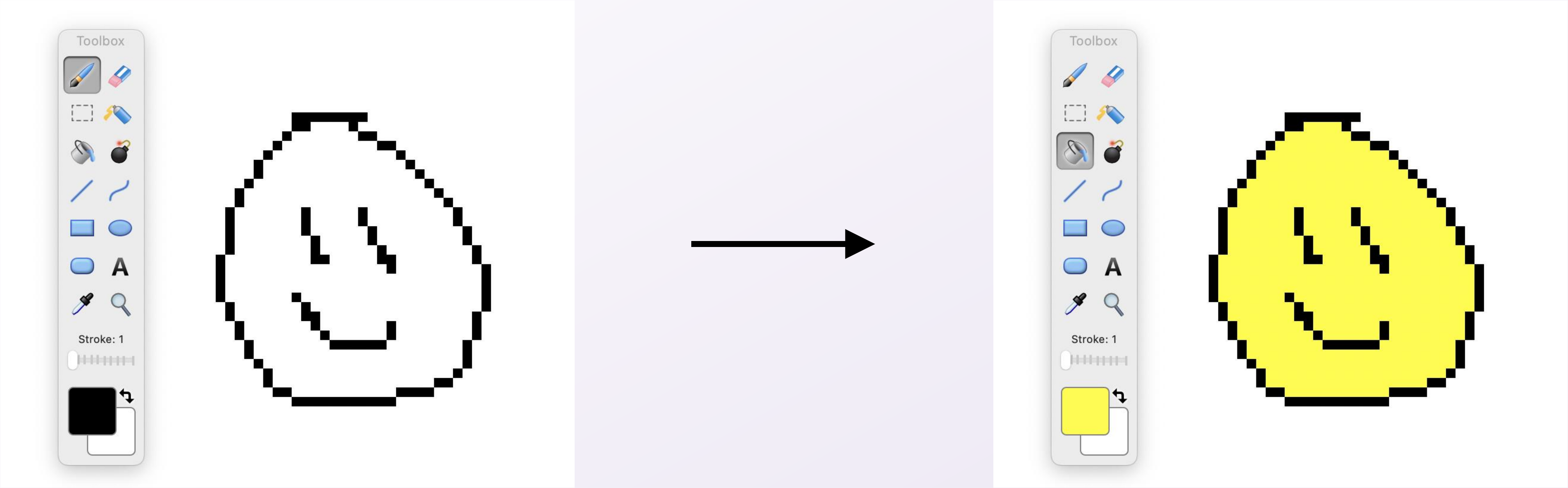
DFS pseudocode

1. Initialize a stack S with the starting point s.
2. Mark all vertices “unprocessed”.
3. **while** S is not empty **do**
4.   Get/remove the next vertex x from S.
5.   **if** x is unprocessed **then**
6.   Process x somehow.
7.   Add all out-neighbors of x to S.
8.   Mark x “processed”.

MS Paint bucket tool

Input: Grid of pixels, a color, and a starting pixel s

Goal: Recolor all pixels that are connected to s by pixels of the same color (and have the same color as s)

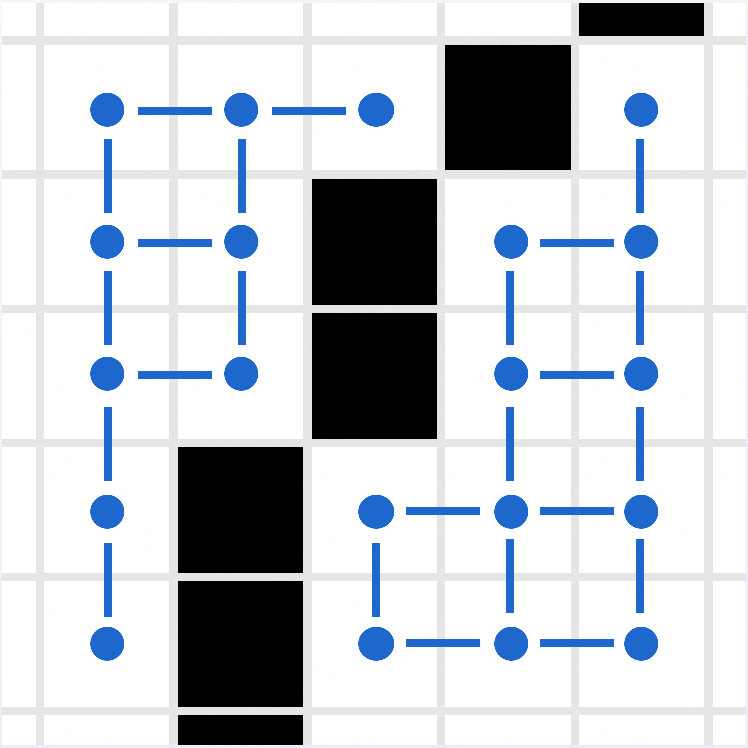


Graph modeling template

Q: What are my objects (vertices)?
A: Pixels that have the same color as the starting pixel s.

Q: How are they related (edges)?
A: Connect all vertices that are adjacent pixels.

We'll use undirected edges.



Bucket tool algorithm

1. Make a vertex for every pixel with the same color as s.
2. **for** every vertex **do**
3. Check the 4 adjacent pixels, add edges if they are vertices.
4. Run BFS or DFS starting at s, where the “process” step is recoloring the pixel.

Alternative solution

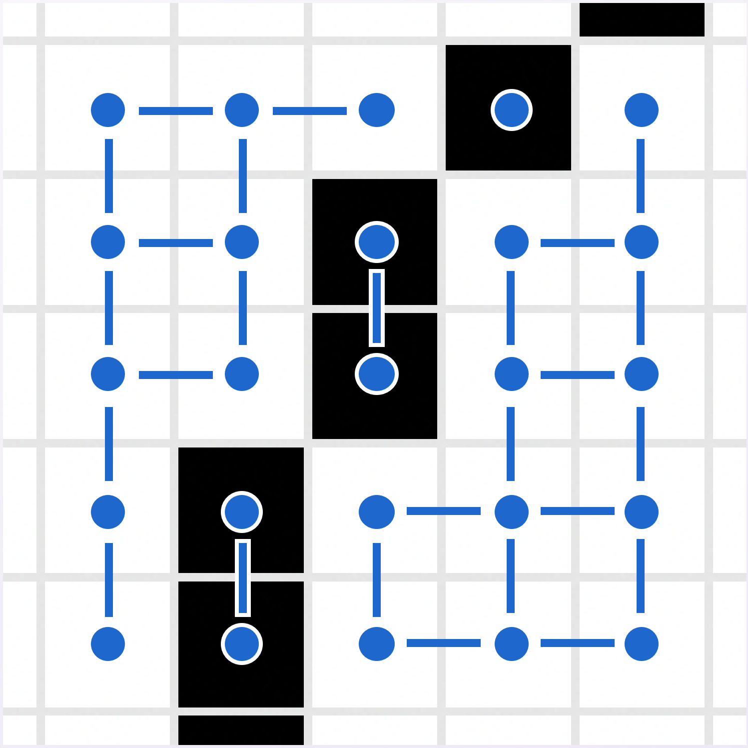
Q: What are my objects (vertices)?

A: All pixels.

Q: How are they related (edges)?

A: Connect all pixels that are adjacent and have the same color.

Pixels of different colors are never connected by edges, so we still find the same connected component!



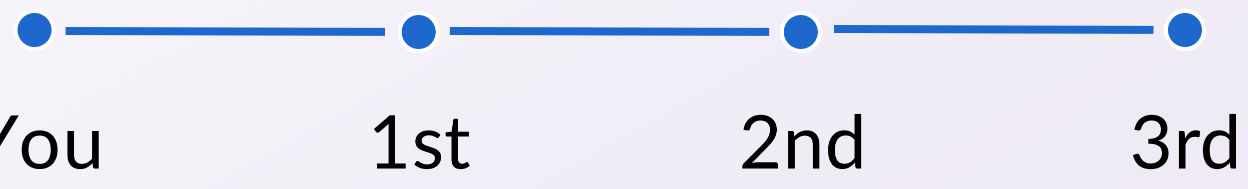
LinkedIn connection degrees

What does this mean?

LinkedIn connection degrees

On LinkedIn, connections are mutual.

- **1st:** People you are directly connected to.
- **2nd:** People with whom you have a mutual direct connection, but are not your 1st degree connections.
- **3rd:** People who are directly connected to your 2nd degree connections, but are not your 1st/2nd degree connections.



LinkedIn connection degrees

Input: A list of people, each person's connections, and a person p

Goal: Find all 3rd degree connections of p

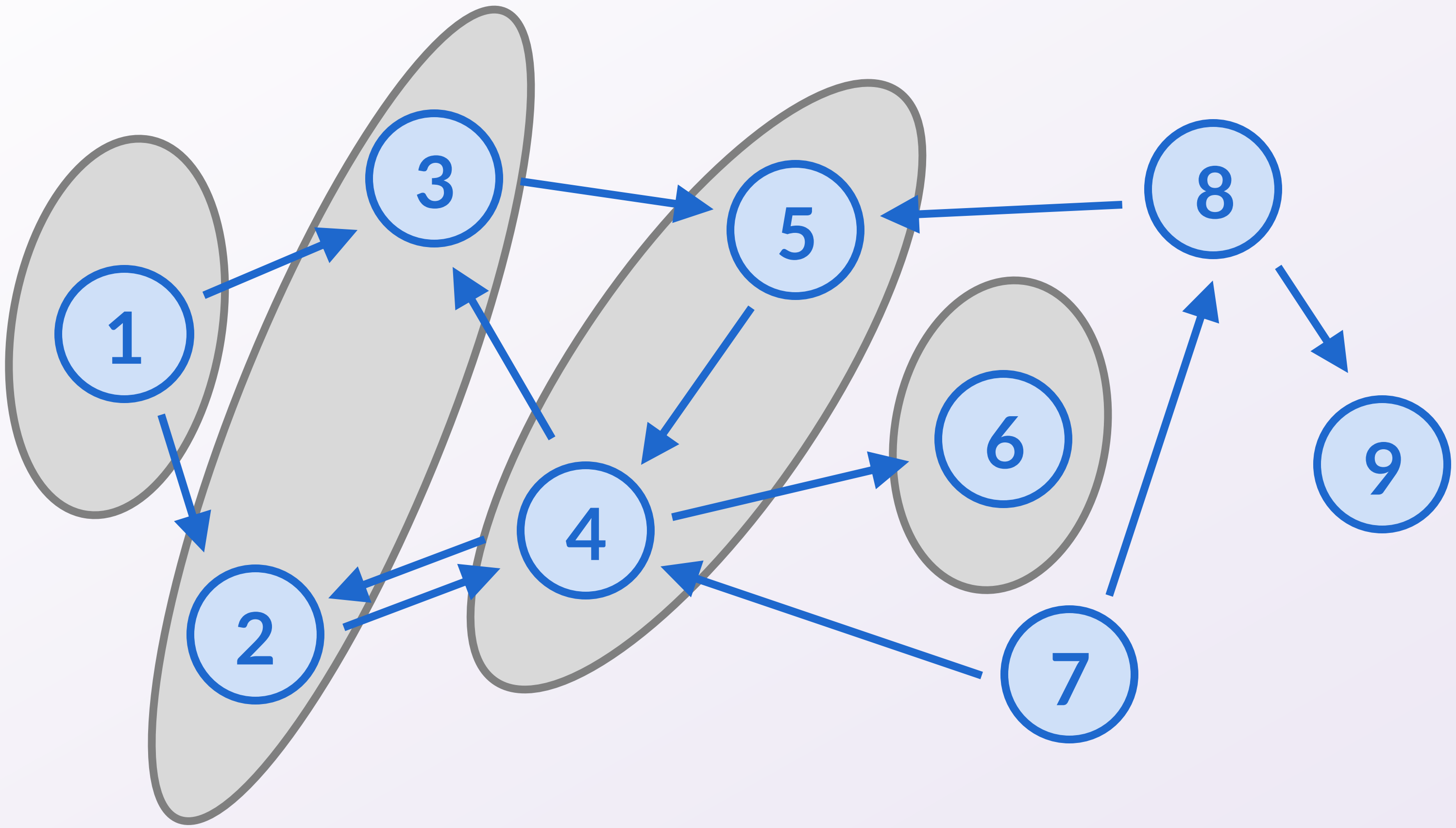
This problem is really asking us to solve:

Input: A graph with vertices V and edges E, and a vertex $s \in V$

Goal: Find all vertices that are distance 3 from s

BFS with layers

Vertices in layer i are distance i from the start.



BFS with layers pseudocode

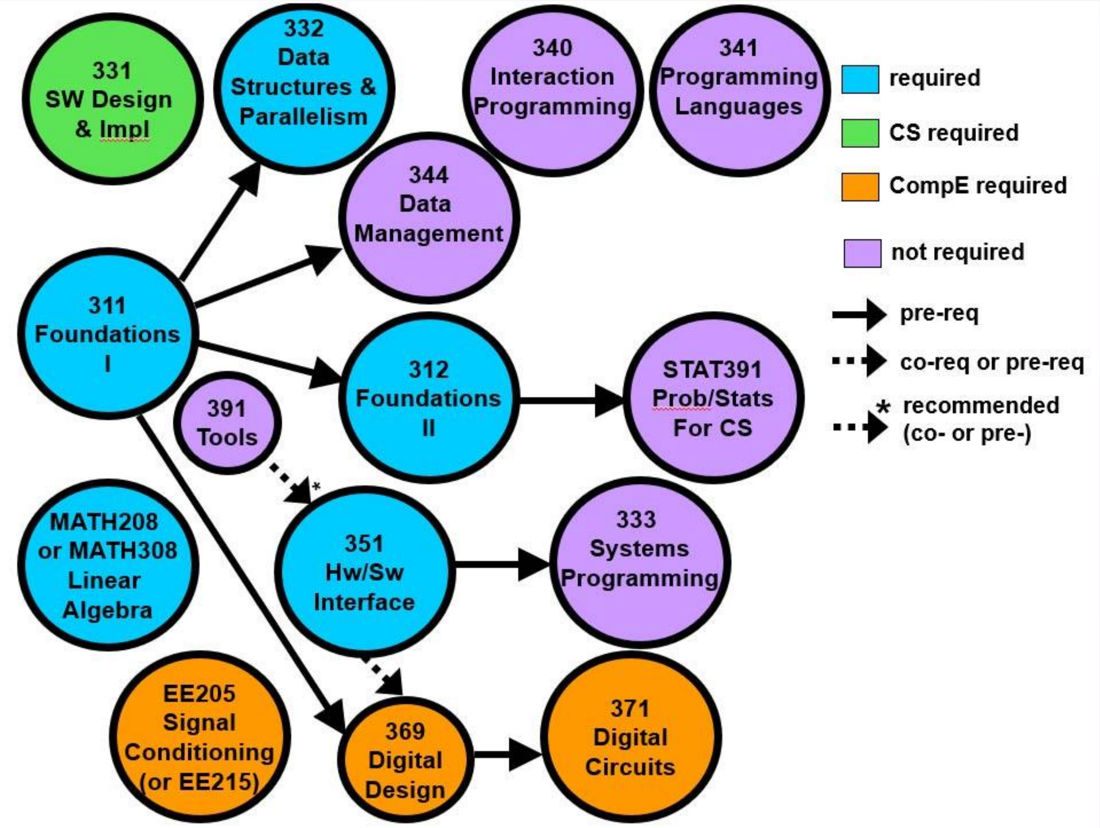
1. Initialize a queue $\boldsymbol{Q}$ with the starting point and layer $(\boldsymbol{s}, \boldsymbol{0})$.
2. Mark all vertices "unseen", except $\boldsymbol{s}$ is "seen".
3. **while** $\boldsymbol{Q}$ is not empty **do**
4.   Get/remove the next vertex-layer pair $(\boldsymbol{x}, \boldsymbol{i})$ from $\boldsymbol{Q}$.
5.   Process $\boldsymbol{x}$ somehow.
6.   **for all** unseen out-neighbors $\boldsymbol{y}$ of $\boldsymbol{x}$ **do**
7.   Mark $\boldsymbol{y}$ "seen" and add $(\boldsymbol{y}, \boldsymbol{i}+\boldsymbol{1})$ to $\boldsymbol{Q}$.

Course planning

Input: List of courses and their prerequisites

Goal: Determine if there is an order in which all courses can be taken

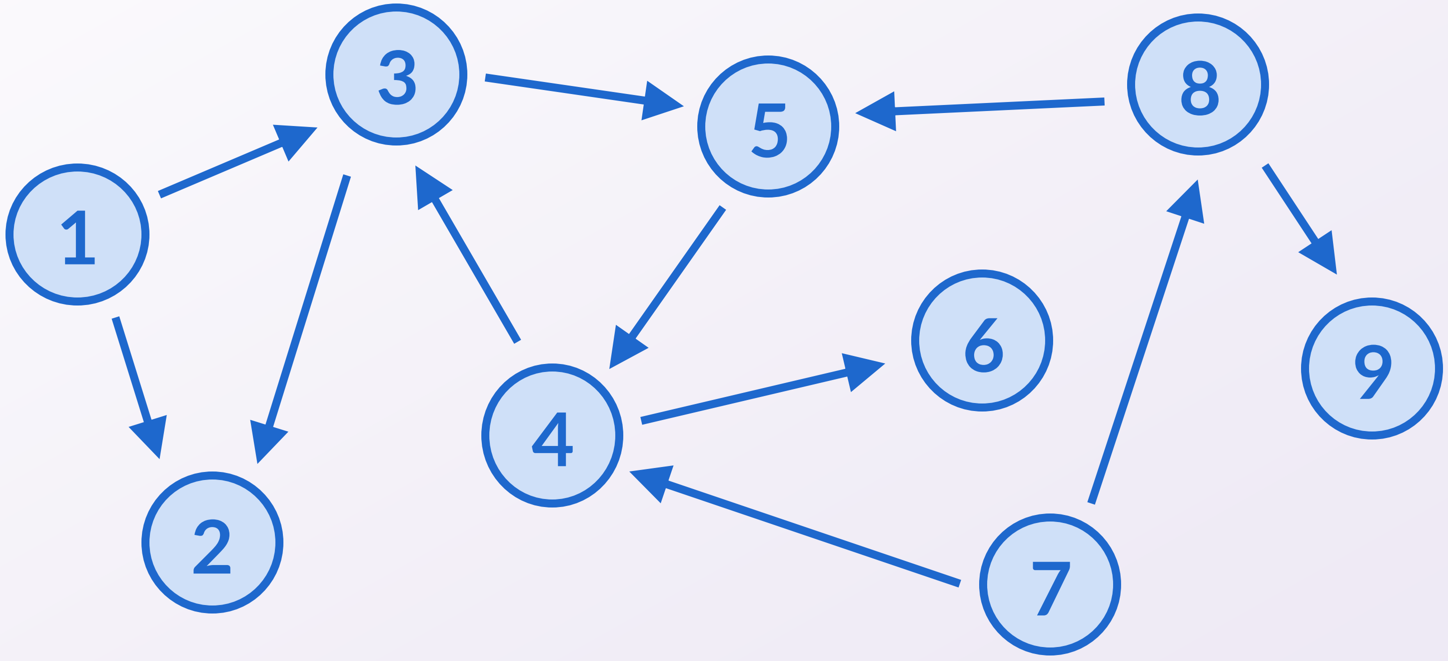
The bad case is when the prerequisites form a cycle!



Cycle detection with DFS

Divide “processed” state into “in-progress” and “finished”.

Seeing vertices “in-progress” means there is a cycle!



Cycle detection with DFS

1. Initialize a stack S with the starting point $(s, \text{"start"})$.
2. Mark all vertices "unstarted".
3. **while** S is not empty **do**
4. Get/remove the next vertex (x, action) from S.
5. **if** x is "in-progress" and $\text{action} = \text{"start"}$, **return** "cycle"
6. **else if** x is "unstarted" and $\text{action} = \text{"start"}$ **then**
7. Add $(s, \text{"end"})$ to S.
8. Add all out-neighbors of x to S.
9. Mark x "in progress".
10. **else if** $\text{action} = \text{"end"}$, mark x "finished"

Course planning

Make a vertex for every course.
for every course v **do**
    Make a directed edge (u, v) if u is a prerequisite for v.
while there is a unfound vertex s **do**
    if DFS starting at s detects a cycle **then**
        return “bad”
    else (DFS finishes and finds all reachable from s)
        Update the list of found vertices.
return “good”

Final reminders

HW3 due Friday @ 11:59pm.

I have OH now-12:30pm:

- Meet at front of classroom, we'll walk over together
- CSE (Allen) 214 if you're coming later

Nathan has online OH 12–1pm:

- Link on Canvas/course website
- https://washington.zoom.us/my/nathanbrunelle