Portland State University

W'21 CS 584/684
Algorithm Design & Analysis

Fang Song

Lecture 4

- Graphs
- Graph traversal
  - BFS
  - DFS

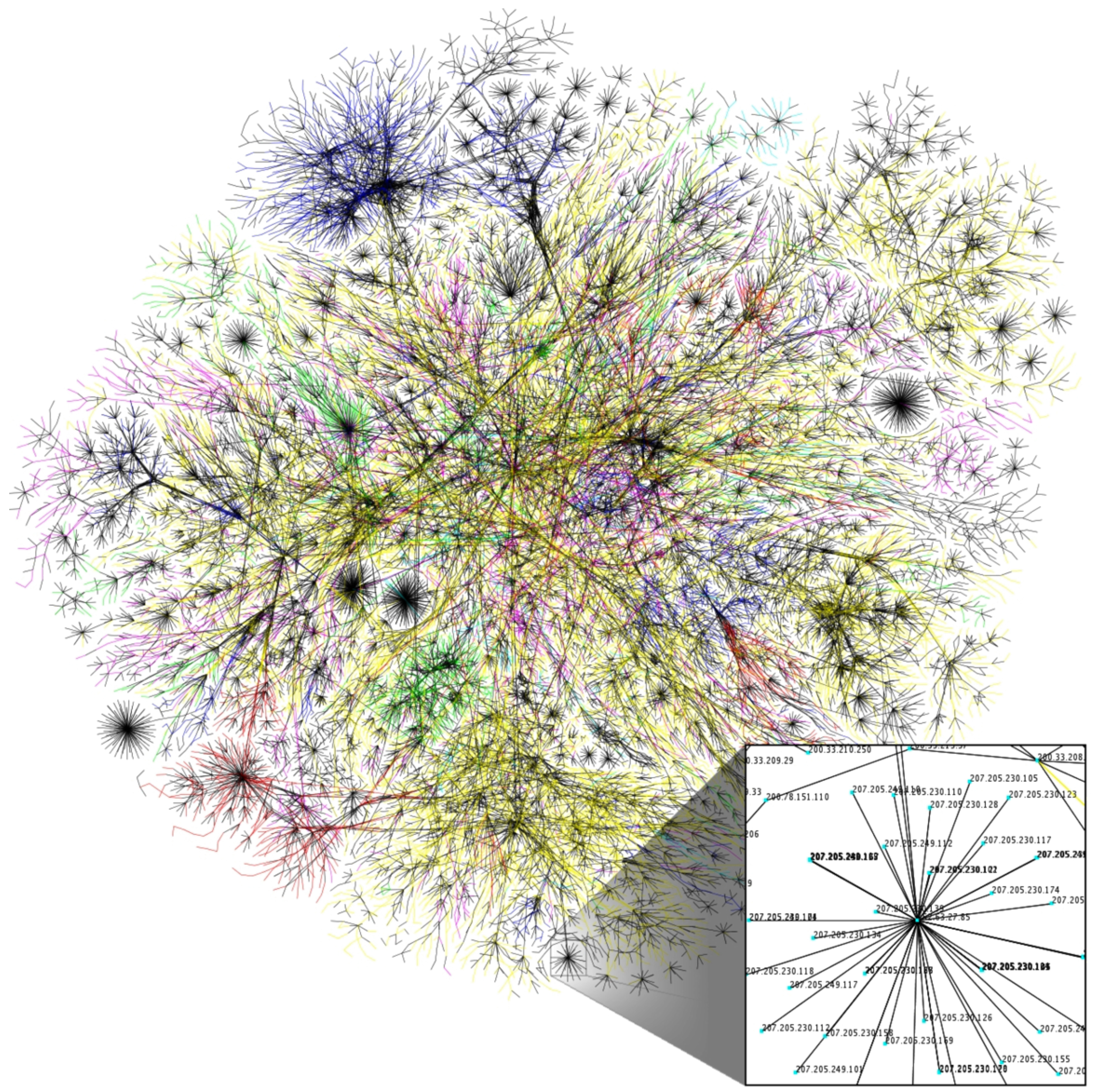
Credit: based on slides by A. Smith & K. Wayne

Recall: master theorem

	$T(n)$	$f(n)$ vs. $n^{\log_b a}$
1	$\Theta(n^{\log_b a})$	$f(n) = O(n^{(\log_b a)-\epsilon})$ for some $\epsilon > 0$.
2	$\Theta(n^{\log_b a} \log n)$	$f(n) = O(n^{\log_b a})$
3	$\Theta(f(n))$	$f(n) = \Omega(n^{(\log_b a)+\epsilon})$ for some $\epsilon > 0$, and $af(n/b) \leq cf(n)$ for some $c < 1$.

Graphs

- **A graph is a set of vertices that are pairwise connected by edges.**
  - Two categories: directed vs. undirected.



- **Why care about graphs?**
  - Graphs are a very useful abstraction.
  - Graphs have numerous applications.
  - A lot of graph algorithms exist (and more under way).

Versatile abstraction

Application	Vertices	Edges
Traffic	Intersections	Roads
Social network	People	Friendship
Game	Board position	Legal move
Financial	Stock/currency	Transactions
Programs	Procedures	Procedure call

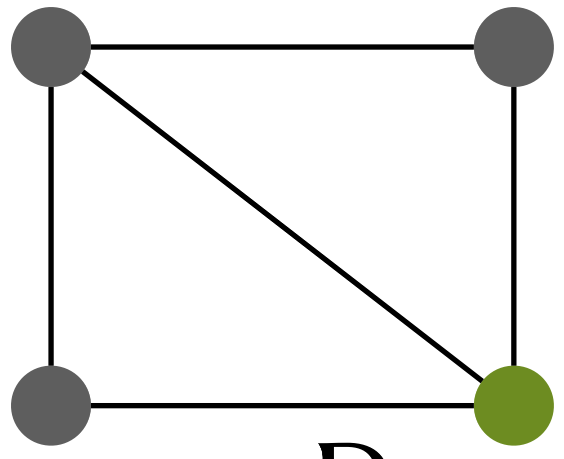
Defining graphs

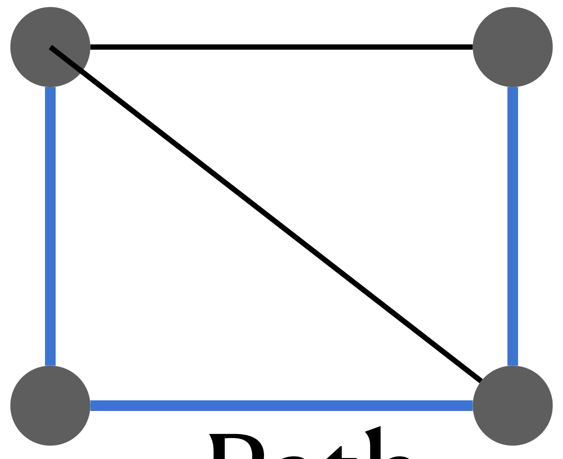
- **An undirected graph $G = (V, E)$ consists of**
  - V: a finite set. (Vertex/node set)
  - $E \subseteq \{(u, v) : u, v \in V\}$. (Edge set)
  - NB. Self loop (u, u) not allowed.

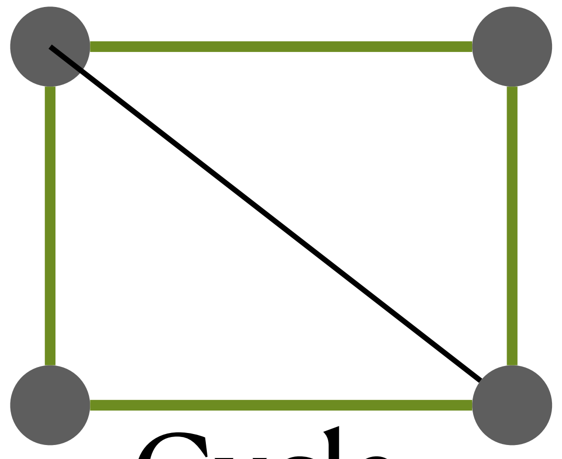
- **A directed graph $G = (V, E)$ consists of**
  - V: a finite set. (Vertex/node set)
  - $E \subseteq \{u \to v : u, v \in V\}$. (Edge set)
  - The set of edges need NOT be symmetric.

Graph terminology

- If $e = (u, v)$ is an edge in a graph, then v is called adjacent to u. (a.k.a neighbors)
- Edge e is said to be incident to u and v .
- Degree of a vertex $d(u)$: the number of edges incident to the vertex u.


Degree $d(u) = 3$


Path

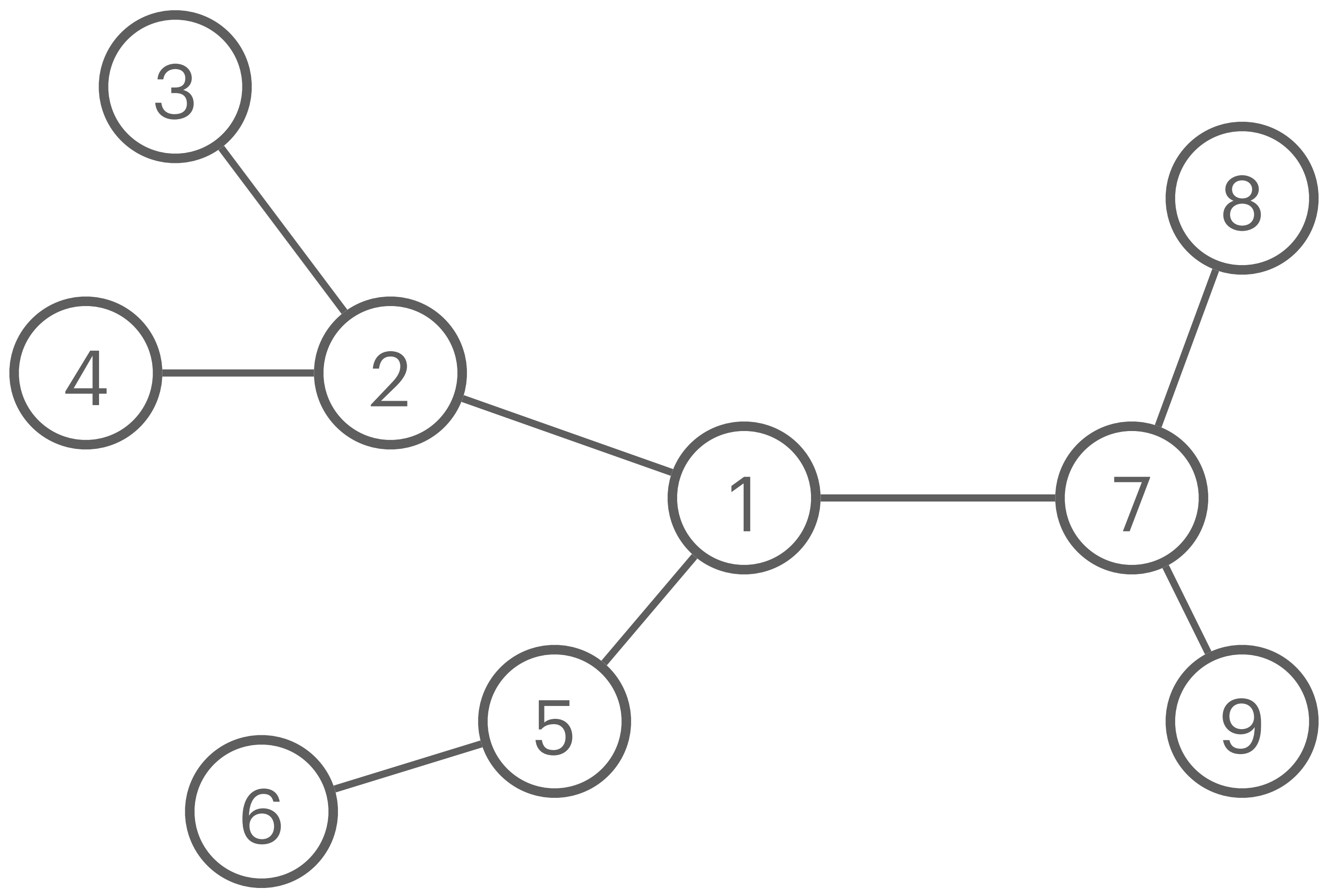

Cycle

- A path is a sequence of vertices that are connected by edges.
  - $\{v_1, \ldots, v_k\}$, s.t. $(v_i, v_{i+1}) \in E$ for all $i = 1,\ldots,k-1$.
- A cycle is a path whose first and last vertices are the same.
- Two vertices are connected if and only if (iff.) there is a path between them.

Trees

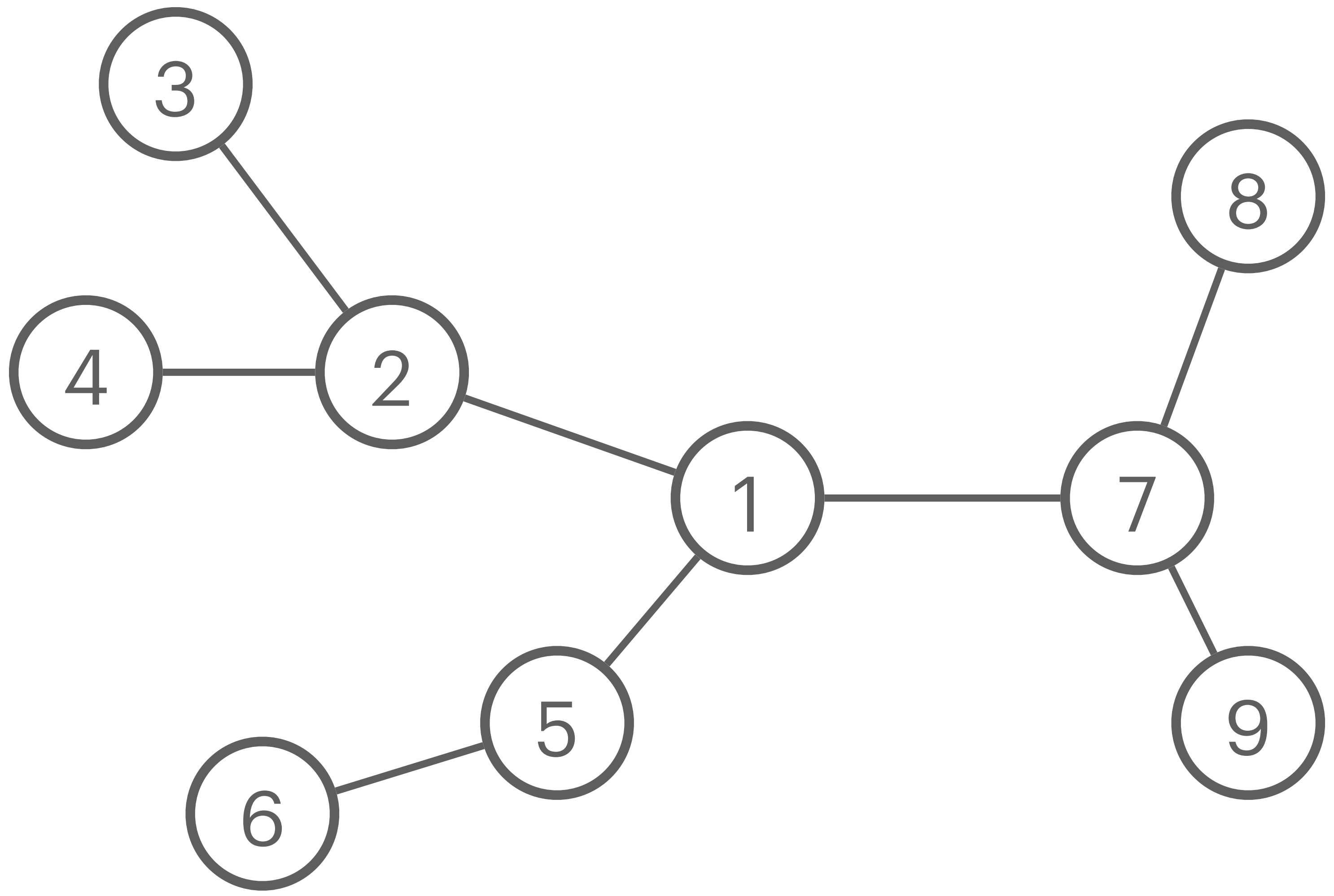
- A graph is connected if every pair of vertices u & v are connected.
- A tree is an undirected graph that is connected and does not contain a cycle.

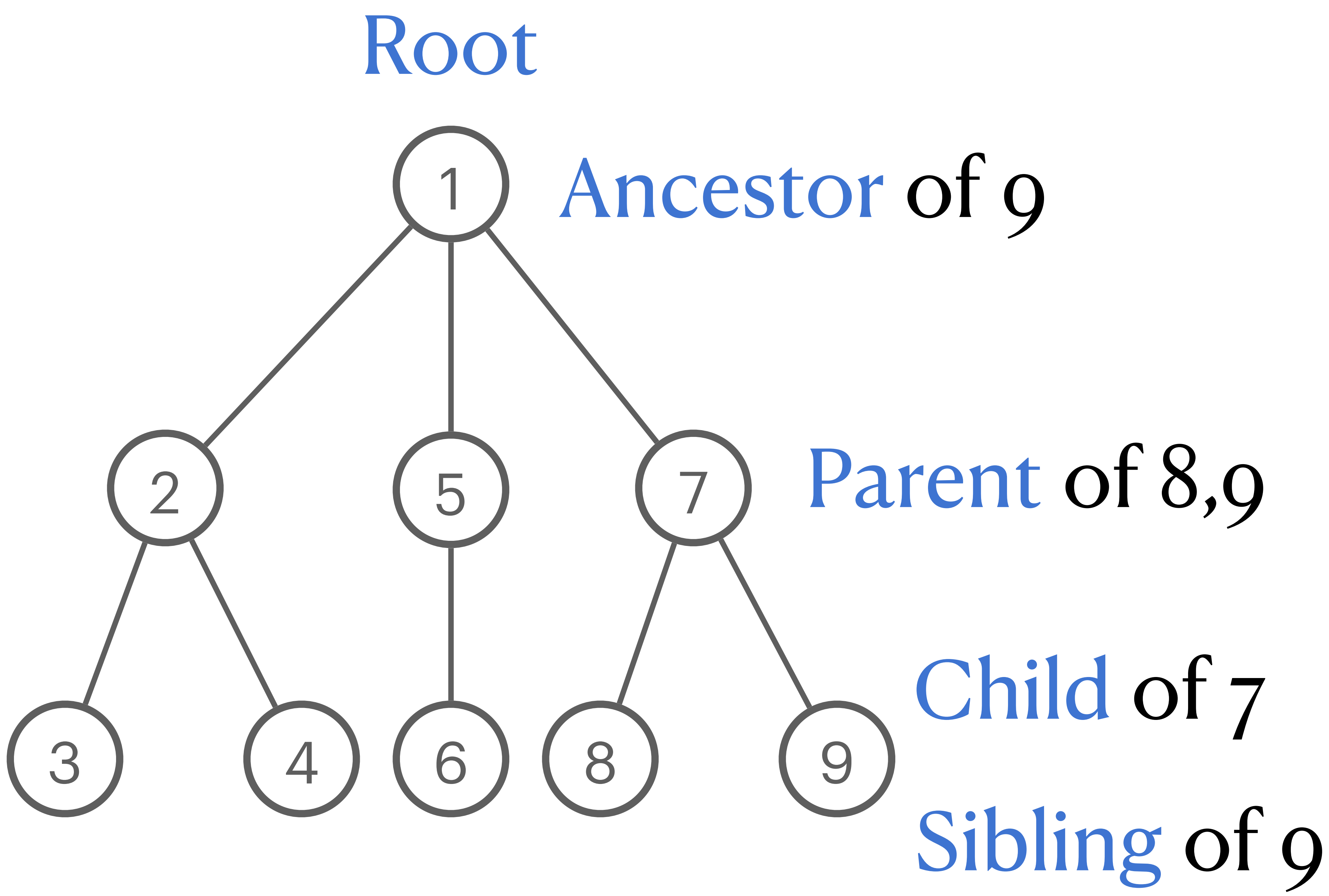
- Theorem. Let G be an undirected graph on n nodes.
  Any two of the following statements imply the third.
  a. G is connected.
  b. G does not contain a cycle.
  c. G has $n-1$ edges.



Rooted trees

- **Given a tree, choose a root node r, and orient each edge away from r.**
  - Models hierarchical structure.





Exploring a graph

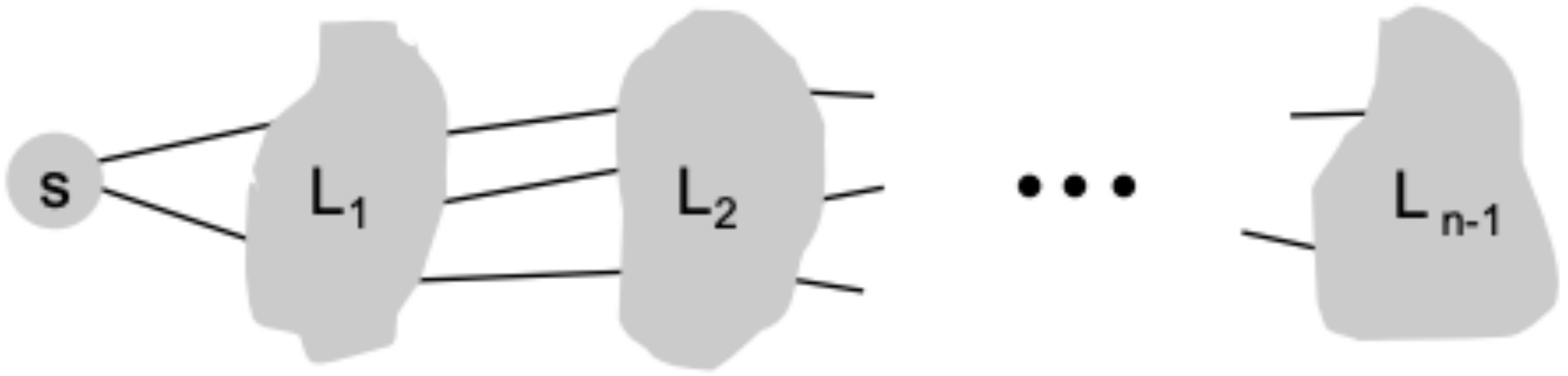
Given: vertices $s, t \in V$.

Goal: decide if there is a path from s to t.

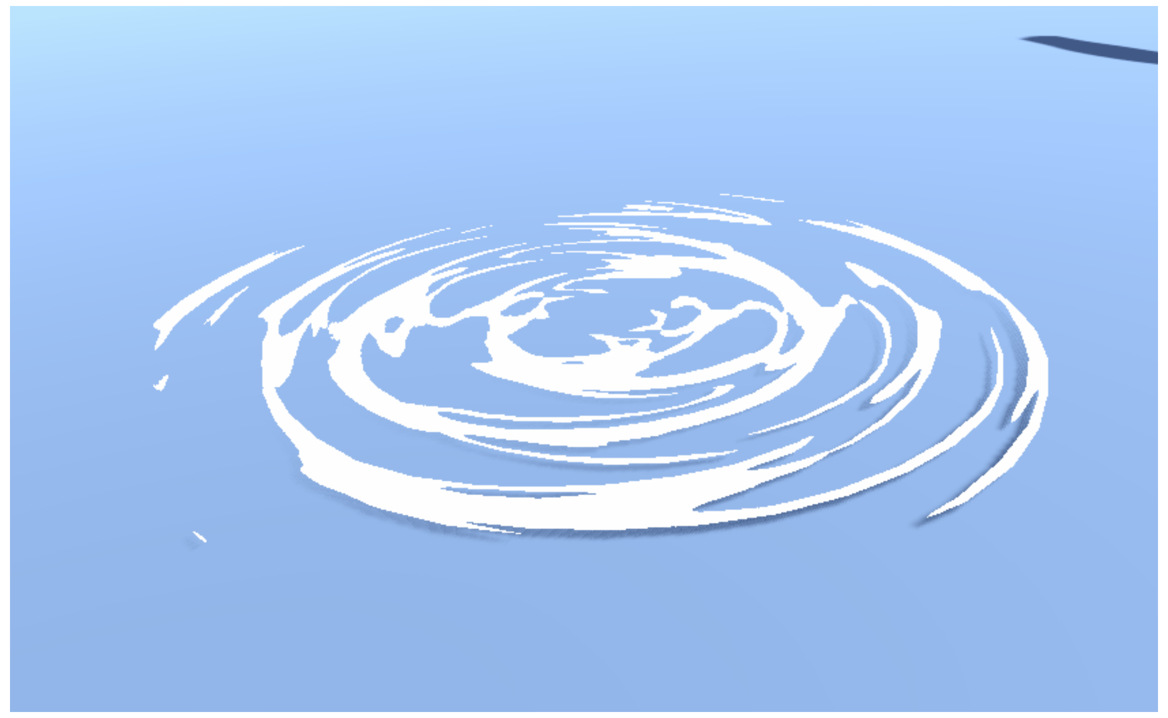
- **Breadth-first search (BFS)**
  - Explore children in order of distance to start node.
- **Depth-first search (DFS)**
  - Recursively explore node's children before exploring siblings.

Breadth-first search

- Intuition. Explore outward from s in all possible directions.
  - Adding nodes one layer at a time.



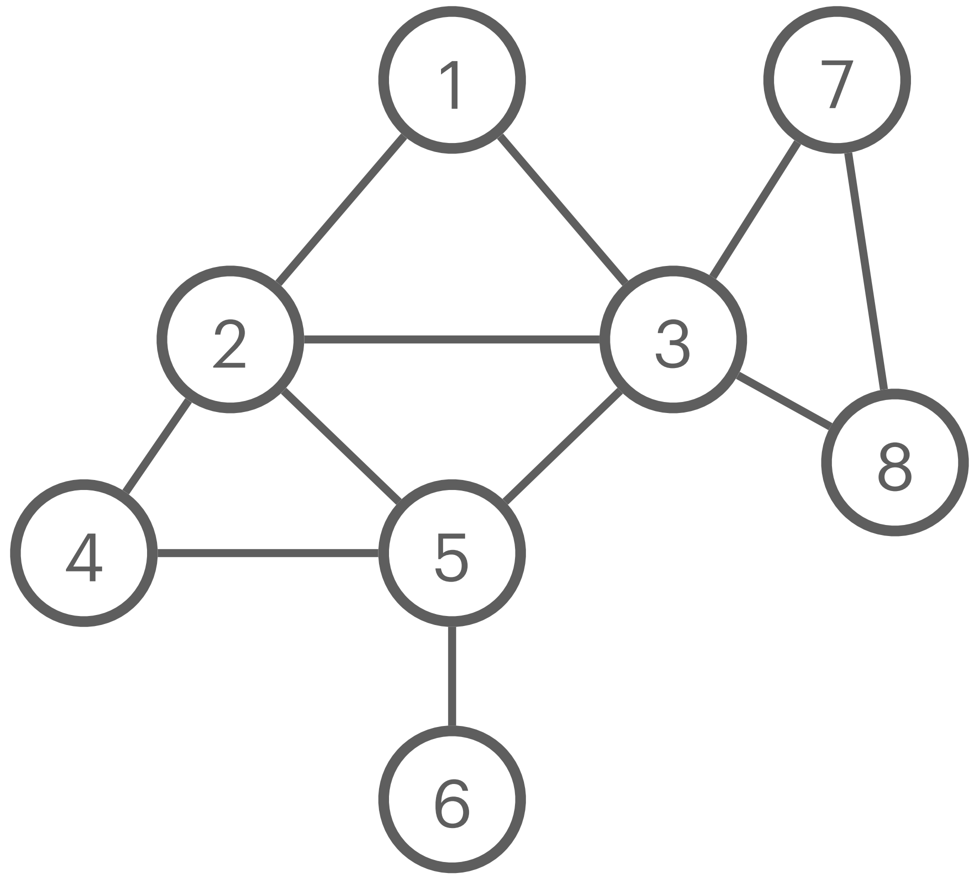
- $L_0 = \{s\}$
- $L_1 = \{\text{neighbors of } L_0\}$
- $L_1 = \{\text{neighbors of } L_1 \text{ not in } L_0 \ \& \ L_1\}$

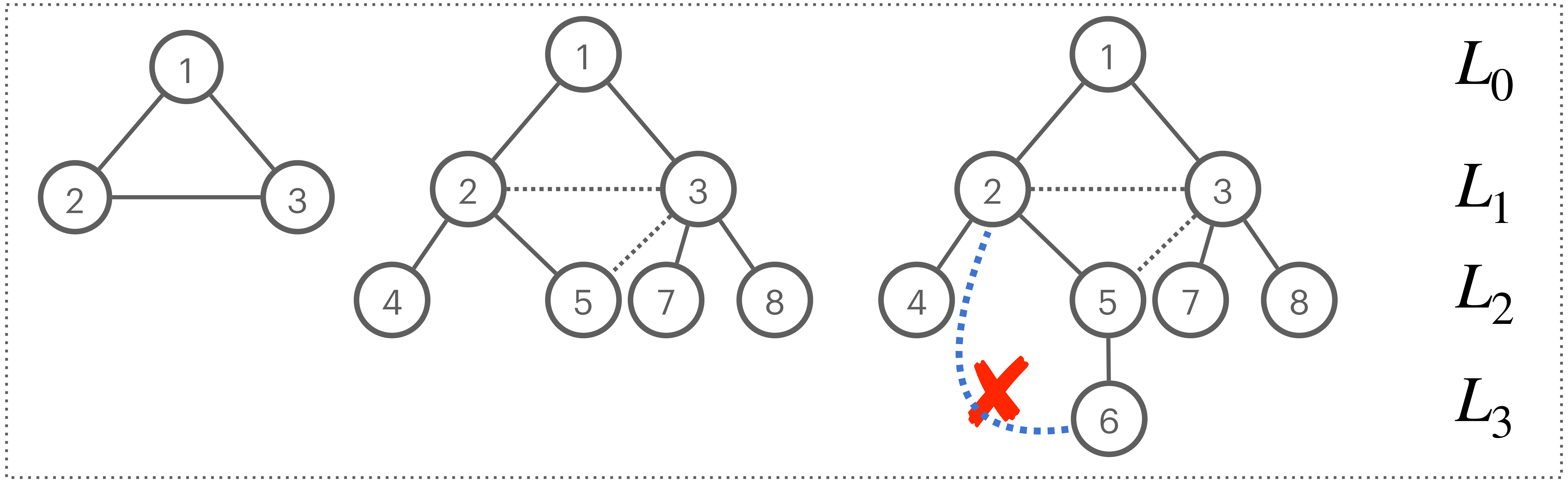


Analogy: wave front of a ripple

Understanding BFS

- BFS demo.

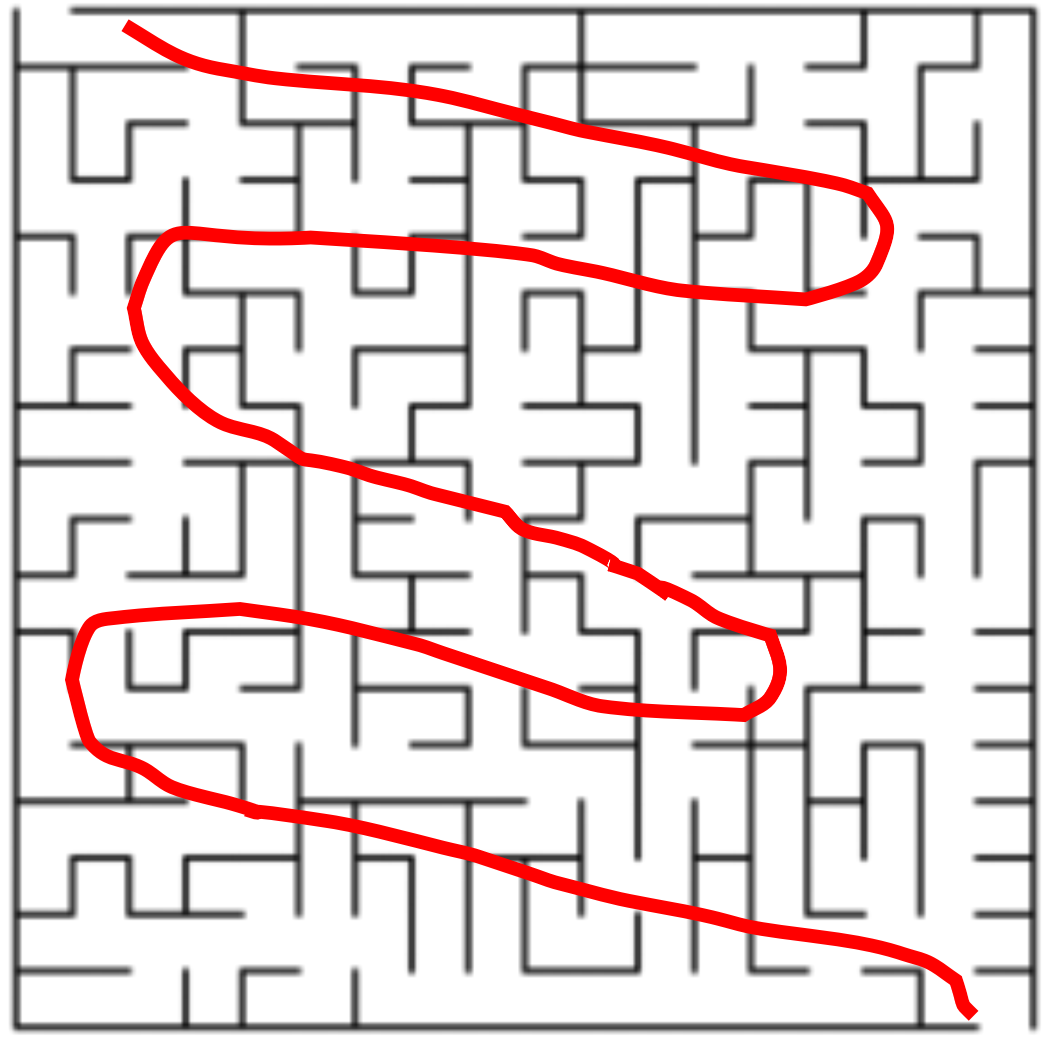




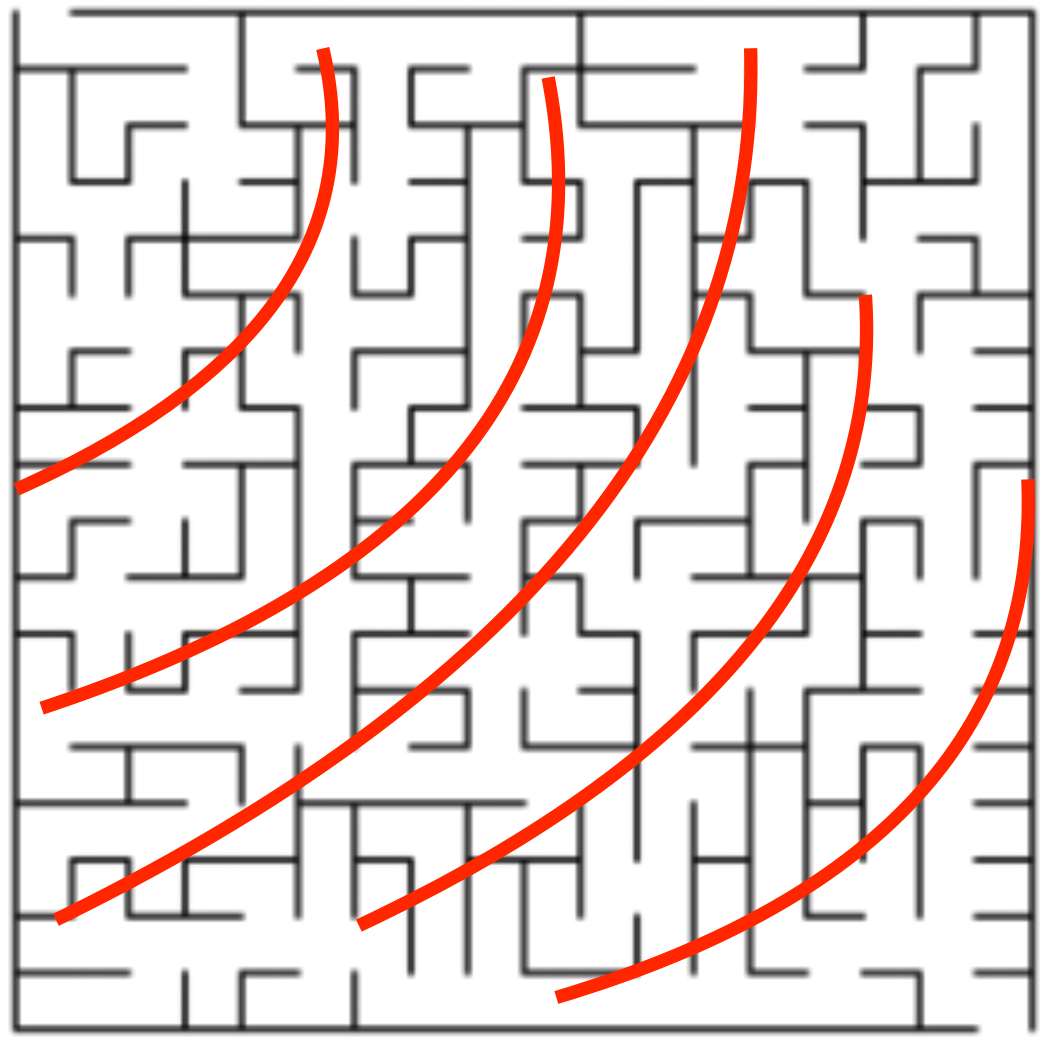
- Running time: linear $O(|V| + |E|)$ [more to come]
- For each i, L_i consists of all nodes at distance exactly i from s.
- There is a path from s to t iff. t appears in some layer.
- Let T be a BFS tree of $G = (V, E)$, and (u, v) an edge of G. Then the levels of u and v differ by at most 1.

Depth-first search

- Intuition. Childen prior to siblings.

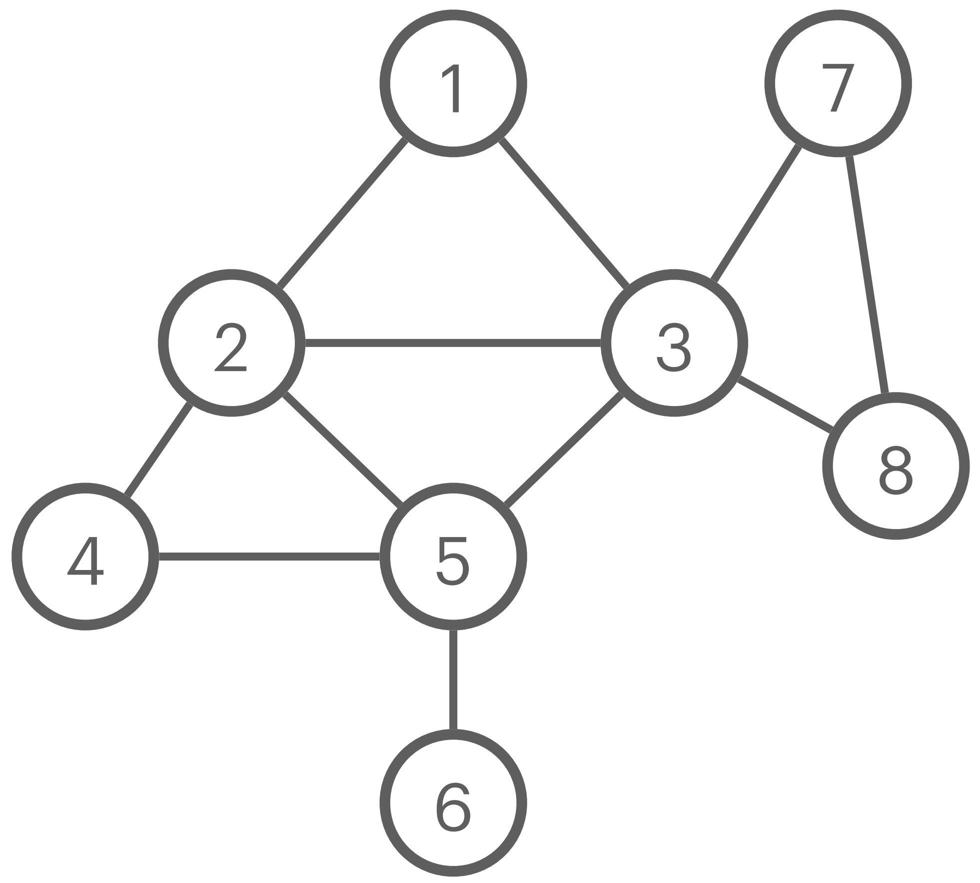


DFS: an impatient maze runner

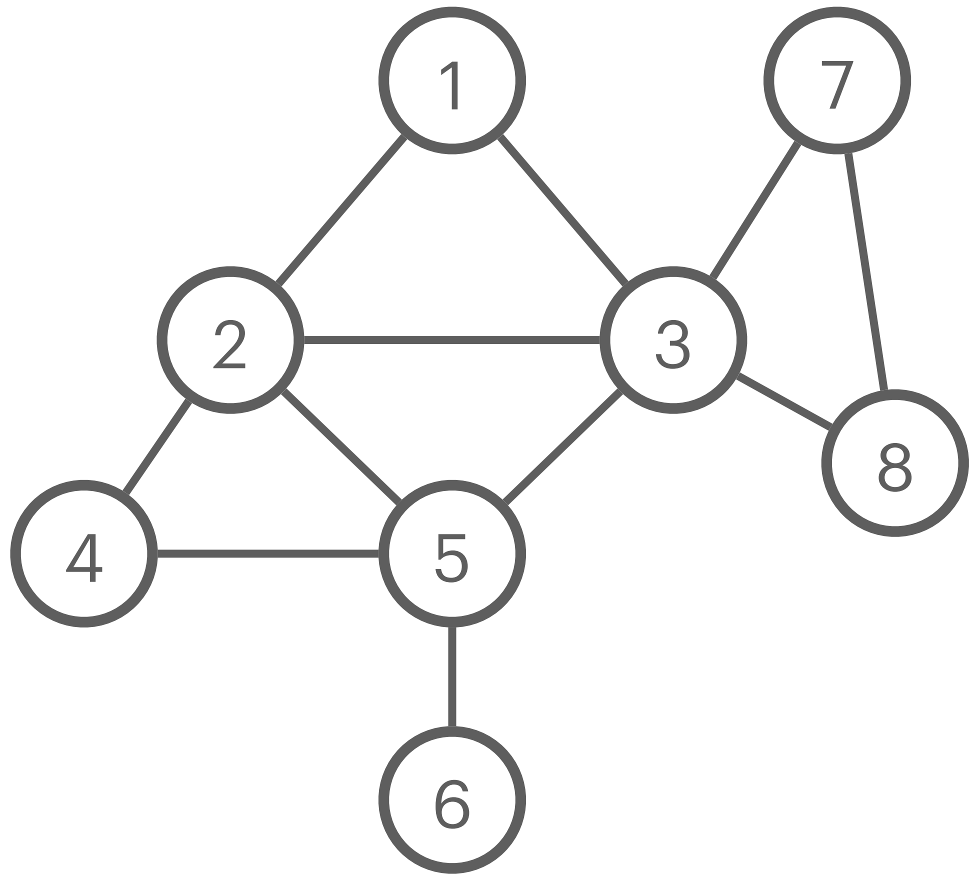


BFS: a patient maze runner

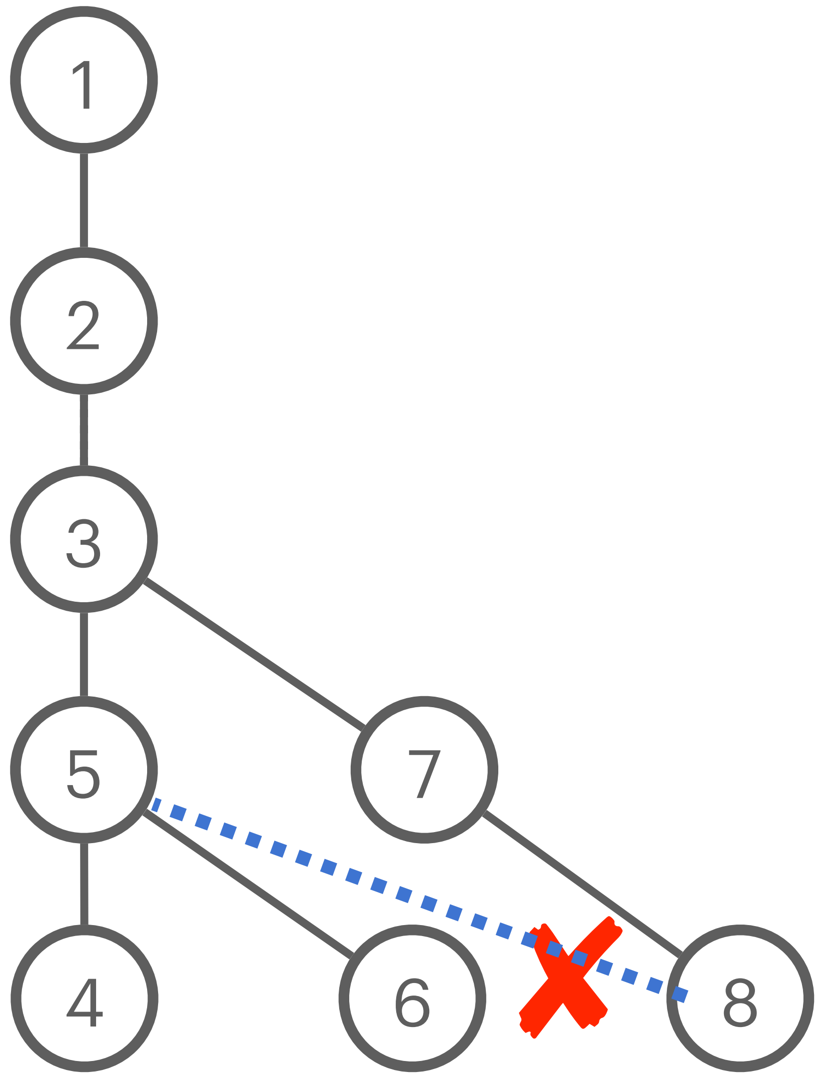
DFS in action



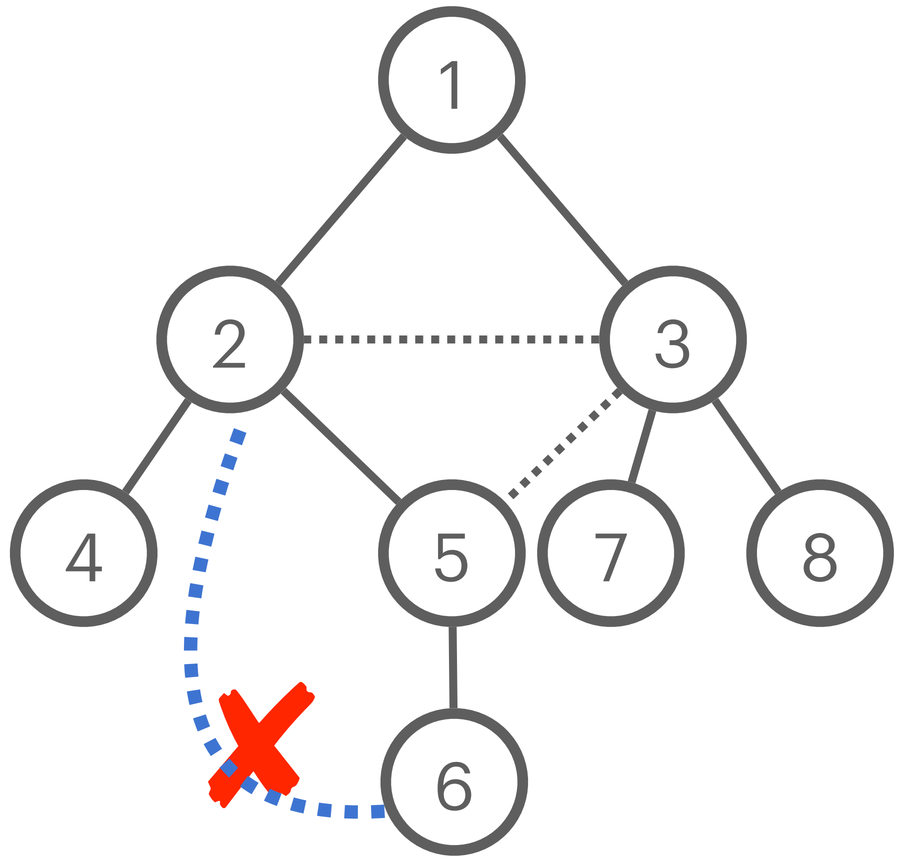
Understanding DFS



DFS Tree



Contrast with BFS Tree



- **Running time: linear $O(|V| + |E|)$ [more to come]**
- **Let T be a DFS tree of $G = (V, E)$, and let u & v be nodes in T.**
  - If (u, v) is an edge of G that is not an edge of T.
  - Then one of u or v is an ancestor of the other.

Implementing BFS/DFS

- **Generic traversal algorithm**

1. $R = \{s\}$
2. While there is an edge (u, v) where $u \in R$ and $v \notin R$, add v to R.

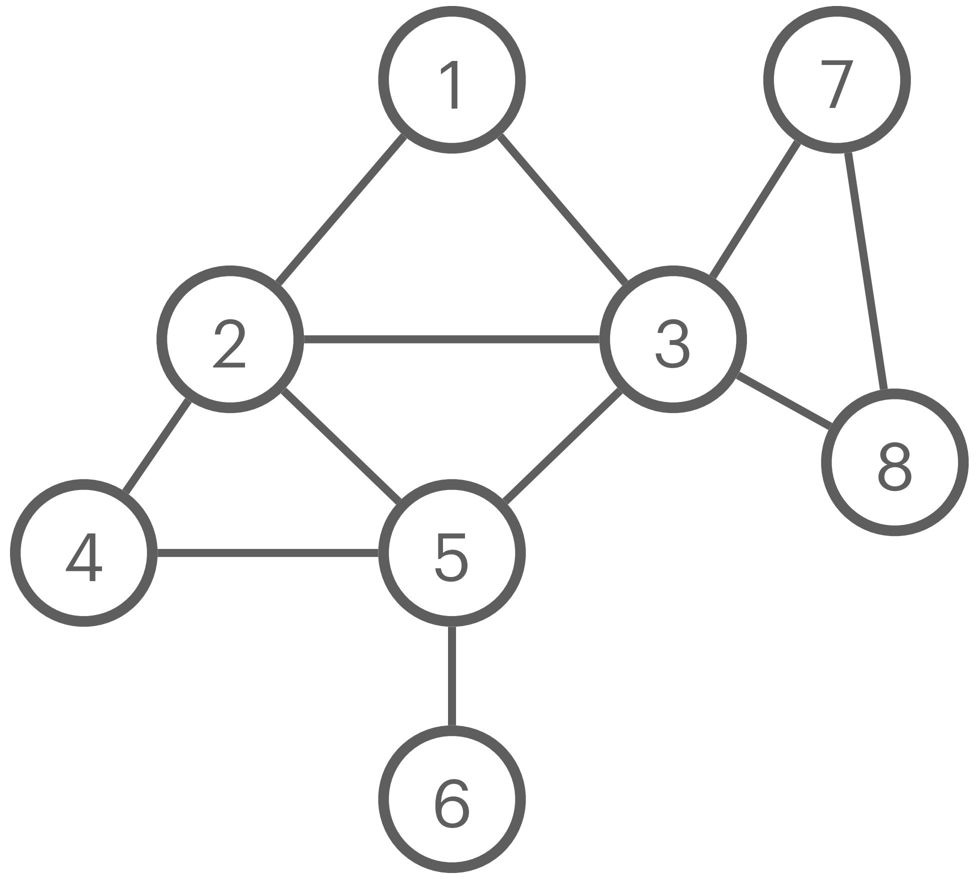
To implement it, need to choose

- **Graph representation**
- **Data structure to track …**
  - Vertices already explored.
  - Edges to be followed next.

These choices affect the order of traversal

Graph representation 1: adjacency matrix

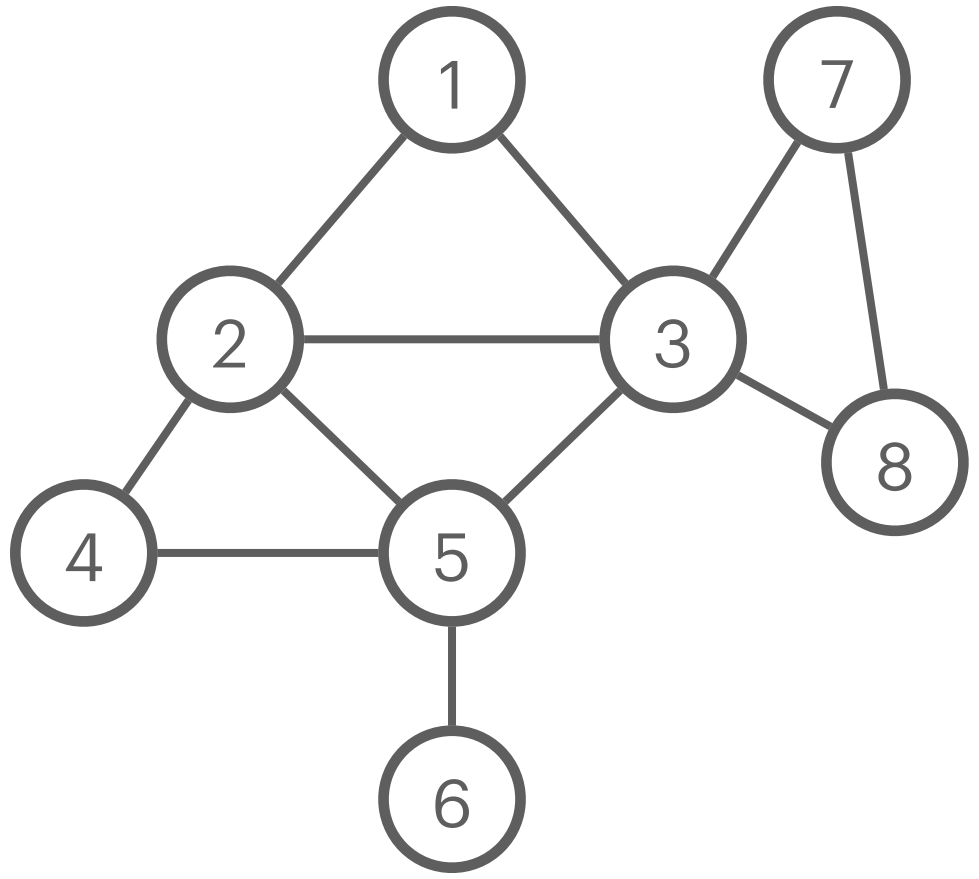
- **Given:** $G = (V, E), |V| = n, |E| = m$.
- **Adjacency matrix** A**:** $n \times n$, $A_{uv} = 1$ **iff.** $(u, v) \in E$ **is an edge.**



- **Basic properties**
  - Lookup an edge: $\Theta(1)$.
  - List all neighbors: $\Theta(n)$.
  - Symmetric for undirected graphs.
  - Space: $\Theta(n^2)$, good for dense graphs.

Graph representation 1: adjacency list

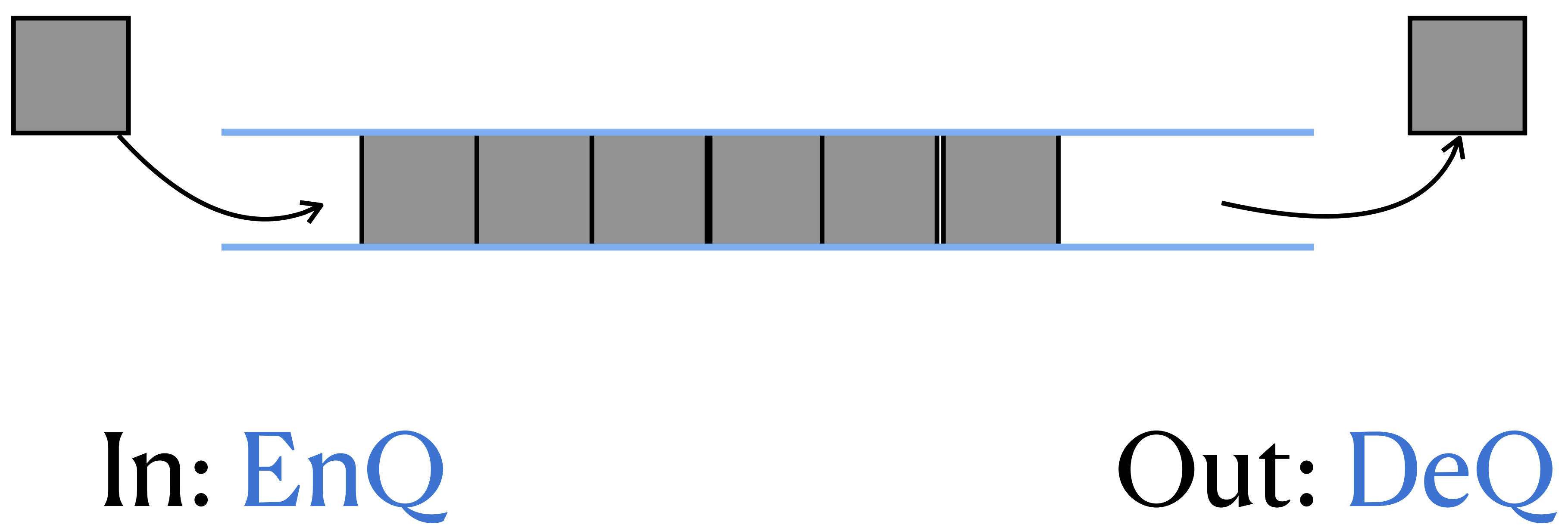
- **Given:** $G = (V, E), |V| = n, |E| = m$.
- **Adjacency list** Adj: $\forall u \in V, Adj[u] = \{v : v$ **adjacent to** $u\}$.



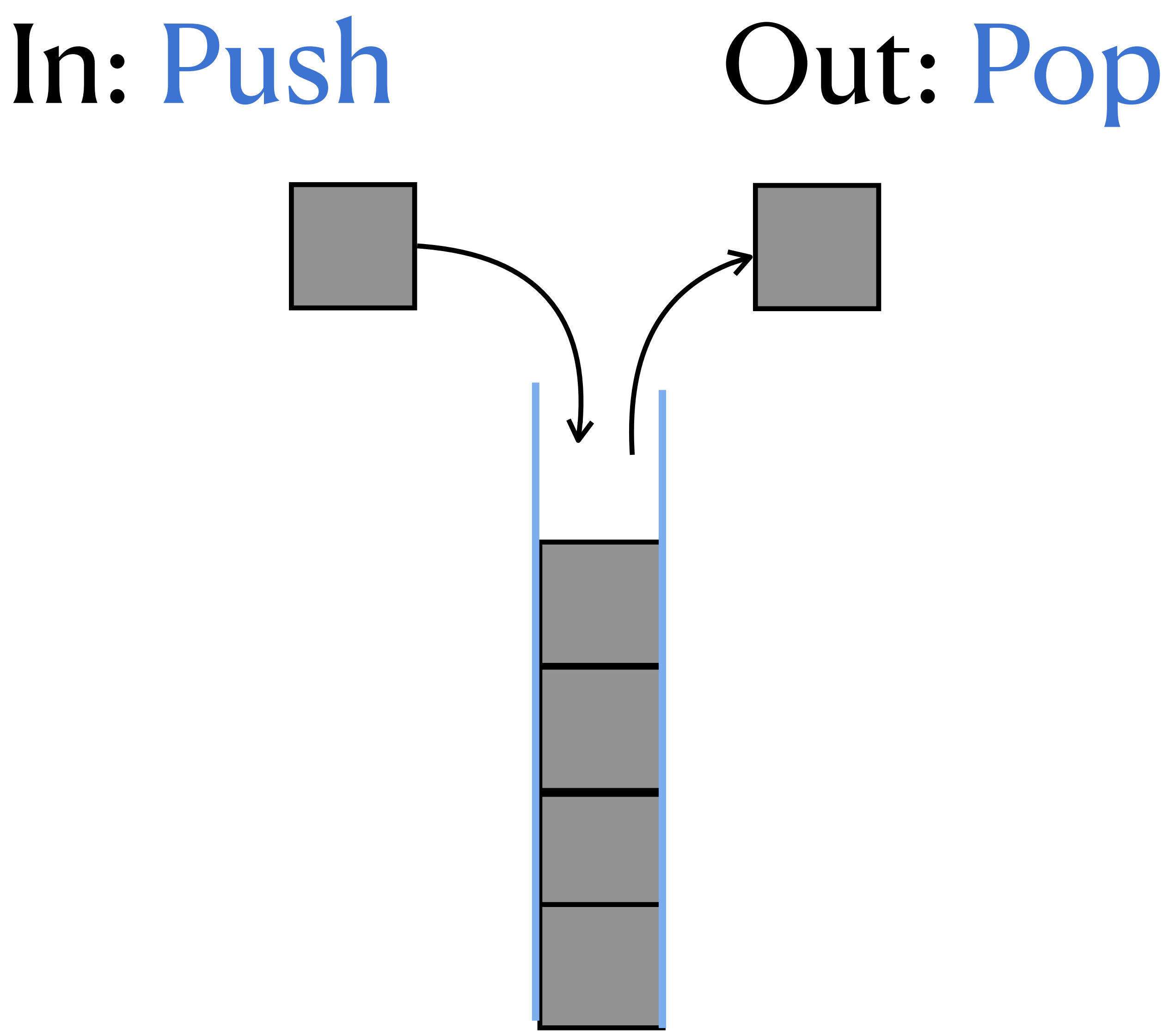
- **Basic properties**
  - Lookup an edge: $\Theta(\text{degree}(u))$.
  - Space: $\Theta(m + n)$, good for sparse graphs.

Review: queue & stack

1. Queue: first-in first-out (FIFO)



2. stack: last-in first-out (FIFO)

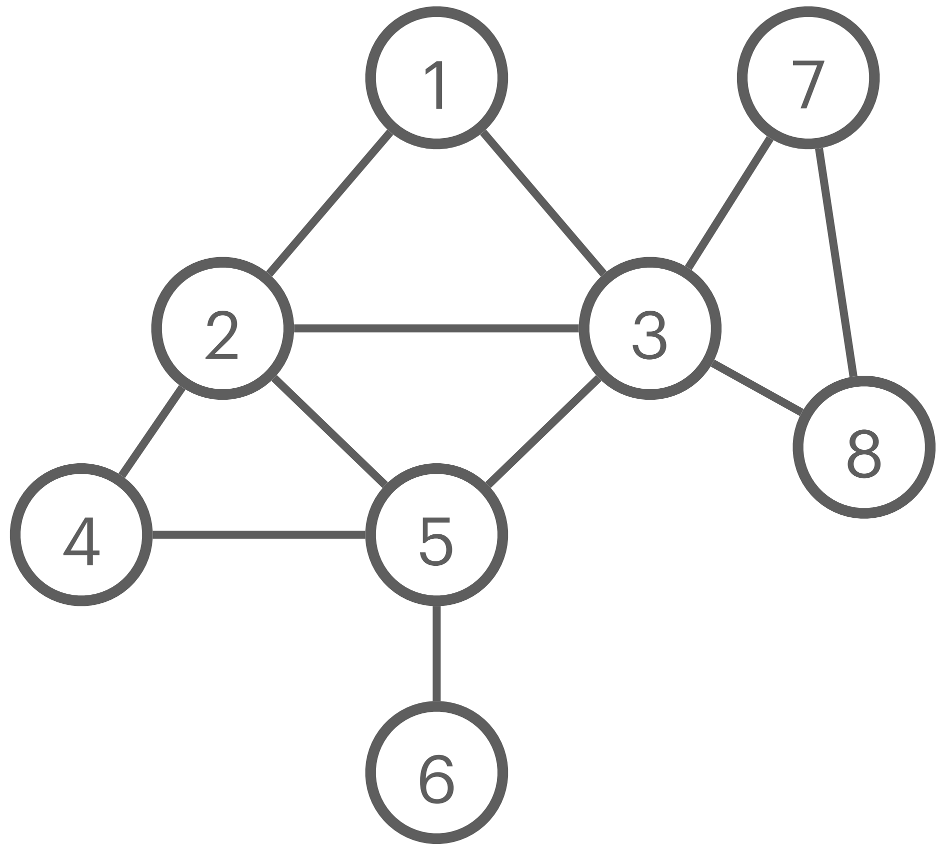


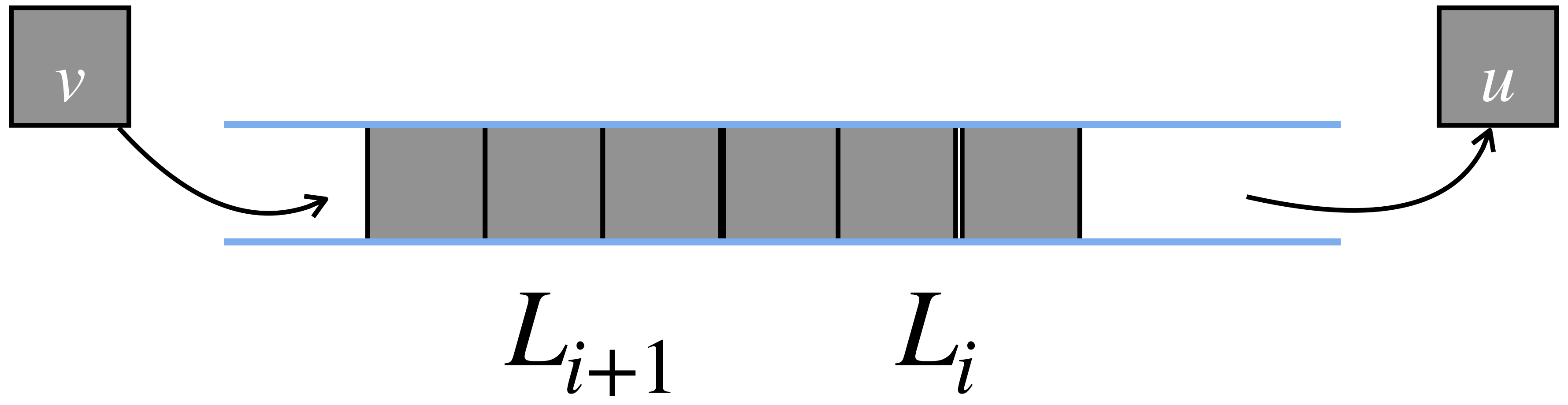
BFS implementation

Input: $G = (V, E)$ by adjacency list Adj. Start node s.

Goal: BFS tree T (rooted at s).

BFS(s):
// Discoverd[1,...,n] array of bits (explored or not), initialized to all zeros.
// Queue $Q \leftarrow \varnothing$
1. Set Discovered[s] = 1
2. EnQ(s) // add s to Q
3. While Q not empty DeQ(u)
    For each (u,v) incident to u
        If Discovered[v]=0 then
            Set Discovered[v]=1
            Add edge (u,v) to T
            EnQ(v)





BFS running time

```
BFS(s):
// Discoverd[1,...,n] array of bits (explored or not),
initialized to all zeros.
// Queue Q ← ∅
1.  Set Discovered[s] = 1
2.  EnQ(s) // add s to Q
3.  While Q not empty  DeQ(u)
       For  each (u,v) incident to u
          If Discovered[v]=0 then
             Set Discovered[v]=1
             Add edge (u,v) to T
             EnQ(v)
```

$O(1)$, run once for all

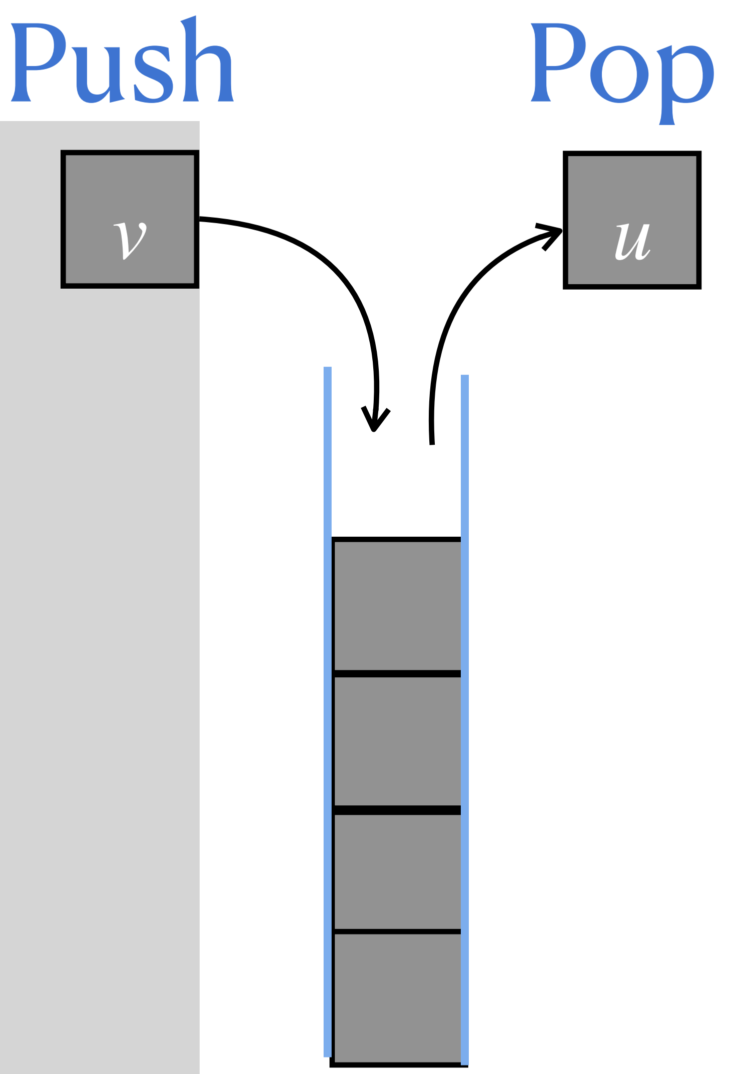
$O(1)$, run once per vertex

$O(1)$, run $\leq$ twice per edge

Theorem. BFS takes $O(m+n)$ time (linear in input size).

DFS implementation

BFS(s):
// Discoverd[1,...,n] array of bits (explored or not), initialized to all zeros.
// Stack $S \leftarrow \varnothing$
1. Set Discovered[s] = 1
2. Push(s) // add s to S
3. While S not empty Pop(u)
   If Discovered[v]=0 then
   Set Discovered[u]=1
   For each (u, v) incident to u
   Push(v)



BFS(s):
...
3. While Q not empty DeQ(u)
   For each (u,v) incident to u
   If Discovered[v]=0 then
   Set Discovered[v]=1
   Add edge (u,v) to T
   EnQ(v)

Theorem. DFS takes $O(m + n)$ time (linear in input size).

- **Exercise. How to build DFS tree T along the way?**

Scratch