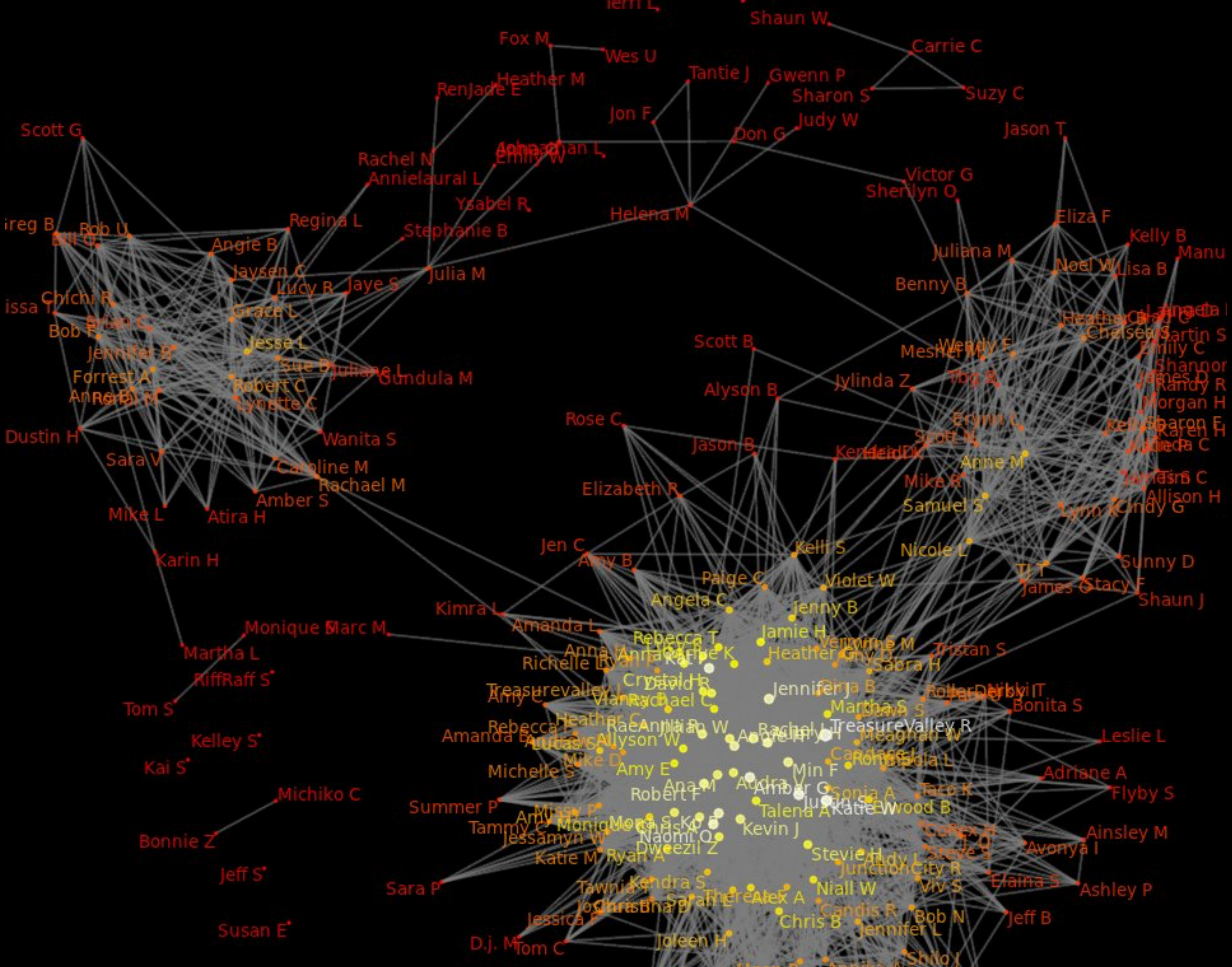
CS3000: Algorithms & Data
Paul Hand

Lecture 13:

- Introduction to Graphs
- Breadth First Search

Feb 25, 2019

Graphs



Shaun W
Carrie C
Fox M
Wes U
Tantie J
Gwenn P
Sharon S
Suzy C
Heather M
RenJade E
Jon F
Judy W
Jason T
Scott G
Don G
Rachel N
Victor G
Annielaural L
Sherilyn O
Ysabel R
Helena M
Eliza F
Regina L
Stephanie B
Kelly B
Bob U
Juliana M
Manu
Angie B
Noel W
Lisa B
Jaysen C
Julia M
Lucy R
Jaye S
Benny B
Chichi R
Grace L
Bob F
Jesse L
Scott B
Martin S
Emily C
Jennifer B
Sue D
Juliane L
Gundula M
Jylinda Z
Forrest A
Robert C
Alyson B
Randy R
Lynette C
Morgan H
Rose C
Dustin H
Wanita S
Jason B
Anne M
Sara V
Caroline M
Rachael M
Elizabeth R
Mike R
Amber S
Samuel S
Allison H
Mike L
Atira H
Cindy G
Karin H
Jen C
Kelli S
Nicole L
Amy B
Sunny D
Paige C
Violet W
James G
Stacy F
Angela C
Shaun J
Kimra L
Jenny B
Monique S
Marc M
Amanda L
Jamie H
Martha L
Rebecca T
Tristan S
RiffRaff S
Richelle L
Sabra H
Crystal H
Jennifer J
Tom S
Bonita S
Martha S
TreasureValley R
Kelley S
Leslie L
Amanda B
Meaghan W
Alyson W
Kai S
Mike D
Michelle S
Amy E
Min F
Adriane A
Michiko C
Flyby S
Robert F
Amber G
Summer P
Sonia A
Talena A
Katie W
Kevin J
Ainsley M
Naomi Q
Bonnie Z
Dweezil Z
Katie M
Ryan A
Stevie H
Avonya I
Jeff S
JunctionCity R
Ashley P
Elaina S
Sara P
Kendra S
Niall W
Viv S
Tawnia T
Alex A
Candis R
Bob N
Jeff B
Susan E
Jessica F
Chris B
Jennifer L
D.j. M
Tom C
Joleen H

Graphs Are Everywhere

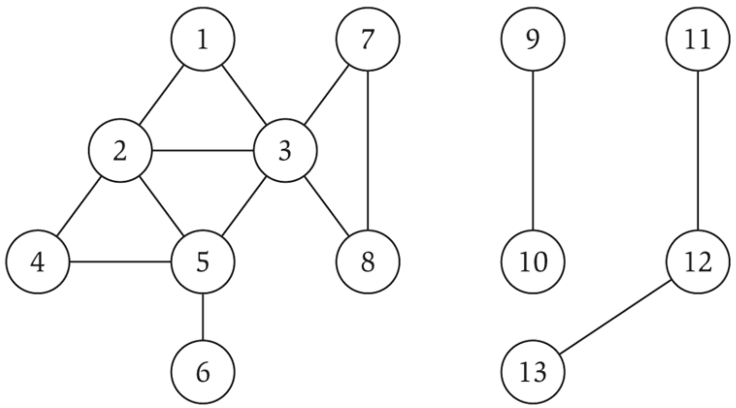
- Transportation networks
- Communication networks
- WWW
- Biological networks
- Citation networks
- Social networks
- …

What's Next

- **Graph Algorithms:**
  - **Graphs:** Key Definitions, Properties, Representations
  - **Exploring Graphs:** Breadth/Depth First Search
    - Applications: Connectivity, Bipartiteness, Topological Sorting
  - **Shortest Paths:**
    - Dijkstra
    - Bellman-Ford (Dynamic Programming)
  - **Minimum Spanning Trees:**
    - Borůvka, Prim, Kruskal
  - **Network Flow:**
    - Algorithms
    - Reductions to Network Flow

Graphs: Key Definitions

- **Definition:** A directed graph $G = (V, E)$
  - V is the set of nodes/vertices
  - $E \subseteq V \times V$ is the set of edges
  - An edge is an ordered $e = (u, v)$ "from u to v"
- **Definition:** An undirected graph $G = (V, E)$
  - Edges are unordered $e = (u, v)$ "between u and v"
- **Simple Graph:**
  - No duplicate edges
  - No self-loops $e = (u, u)$



Activity

- How many edges can there be in a **simple** directed/undirected graph with n nodes?

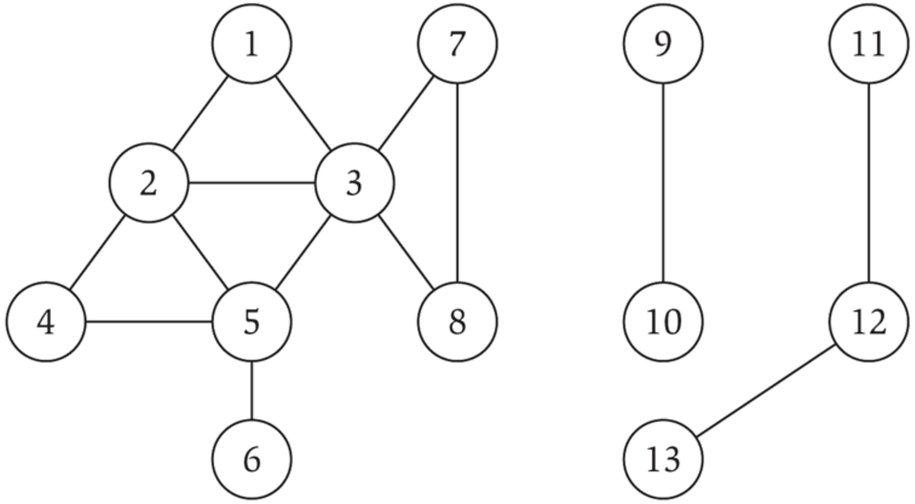
Paths/Connectivity

- A path is a sequence of consecutive edges in E
  - $P = \{(u, w_1), (w_1, w_2), (w_2, w_3), \dots, (w_{k-1}, v)\}$
  - $P = u - w_1 - w_2 - w_3 - \cdots - w_{k-1} - v$
  - The length of the path is the # of edges

- An undirected graph is connected if for every two vertices $u, v \in V$, there is a path from u to v
- A directed graph is strongly connected if for every two vertices $u, v \in V$, there are paths from u to v and from v to u

Cycles

- A cycle is a path $v_1 - v_2 - \cdots - v_k - v_1$ where $k \geq 3$ and $v_1, \ldots, v_k$ are distinct



Activity: how many cycles are there in this graph?

Activity

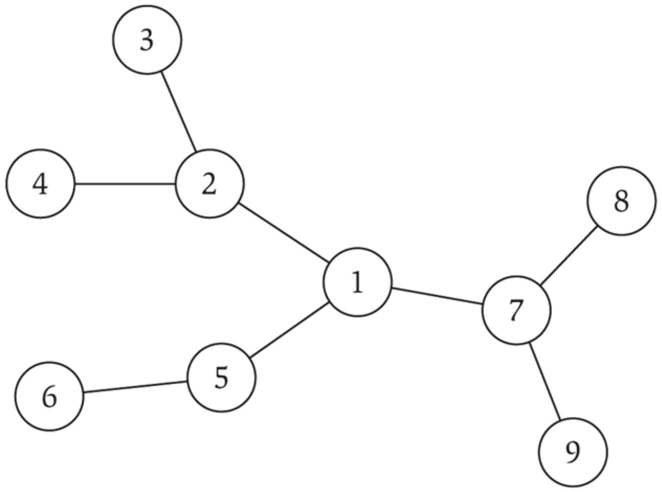
- Suppose an n-node undirected graph G is connected
  - True/False? G has at least $n - 1$ edges

- Suppose an n-node undirected graph G has $n - 1$ edges
  - True/False? G is connected

Trees

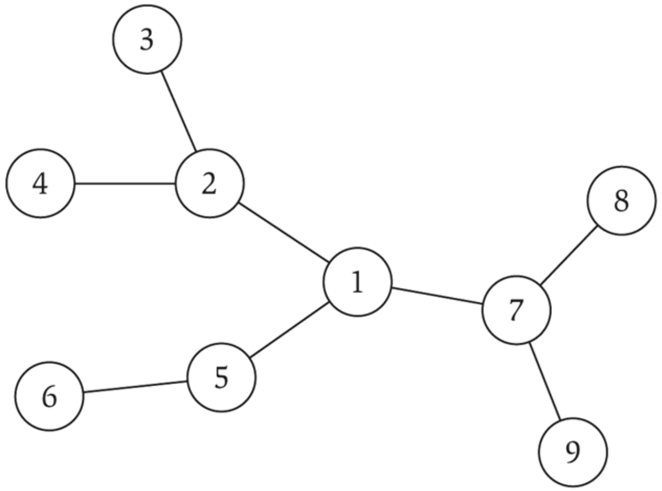
- A simple undirected graph G is a tree if:
  - G is connected
  - G contains no cycles

- **Theorem:** any two of the following implies the third
  - G is connected
  - G contains no cycles
  - G has $= n - 1$ edges

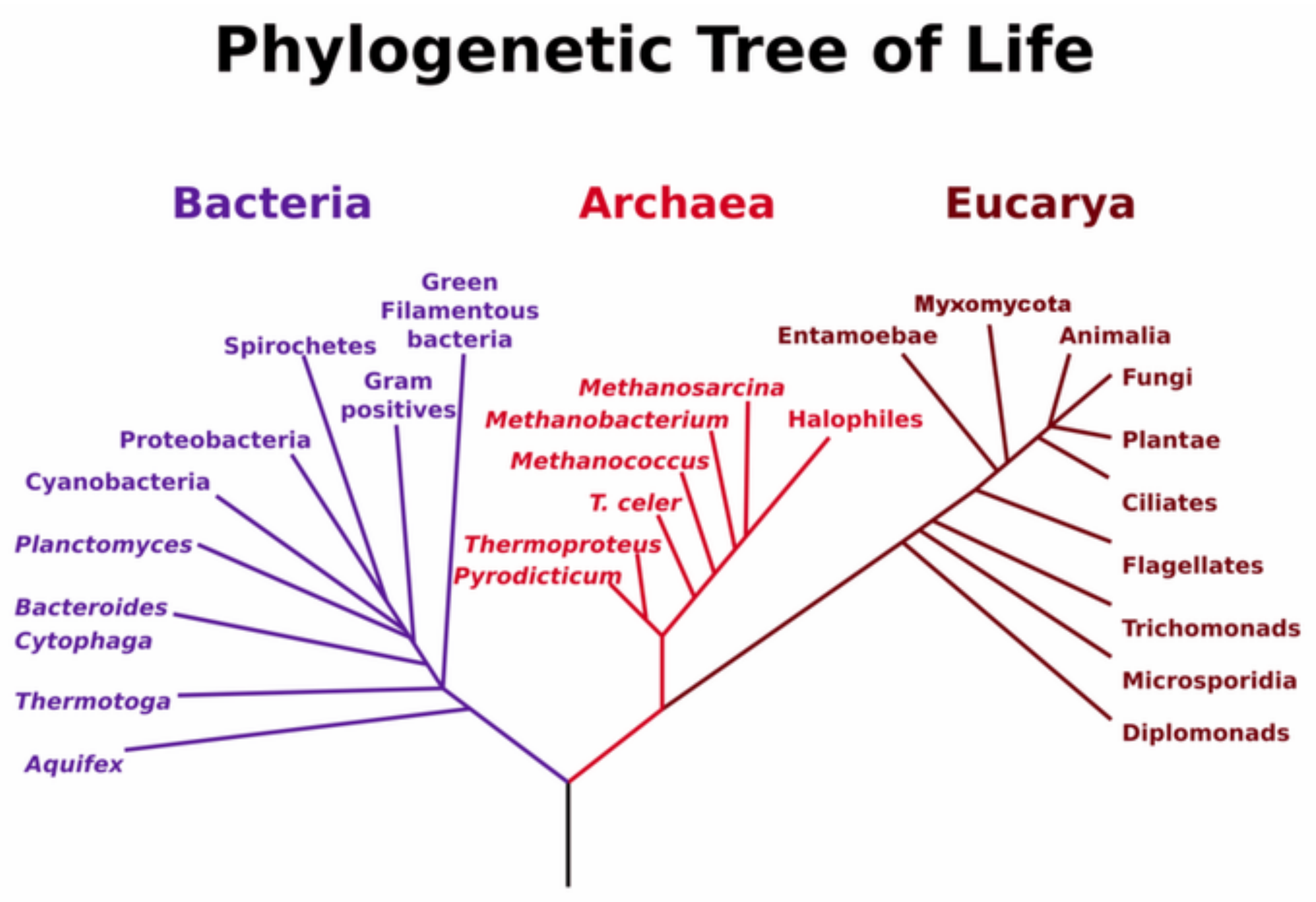


Trees

- Rooted tree: choose a root node r and orient edges away from r
  - Models **hierarchical structure**



Phylogeny Trees



Exploring a Graph

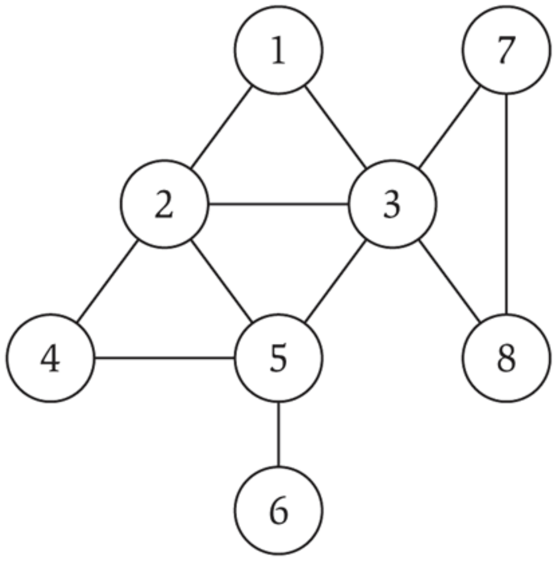
Exploring a Graph

- **Problem:** Is there a path from s to t?
- **Idea:** Explore all nodes reachable from s.

- Two different search techniques:
  - **Breadth-First Search:** explore nearby nodes before moving on to farther away nodes
  - **Depth-First Search:** follow a path until you get stuck, then go back

Breadth-First Search (BFS)

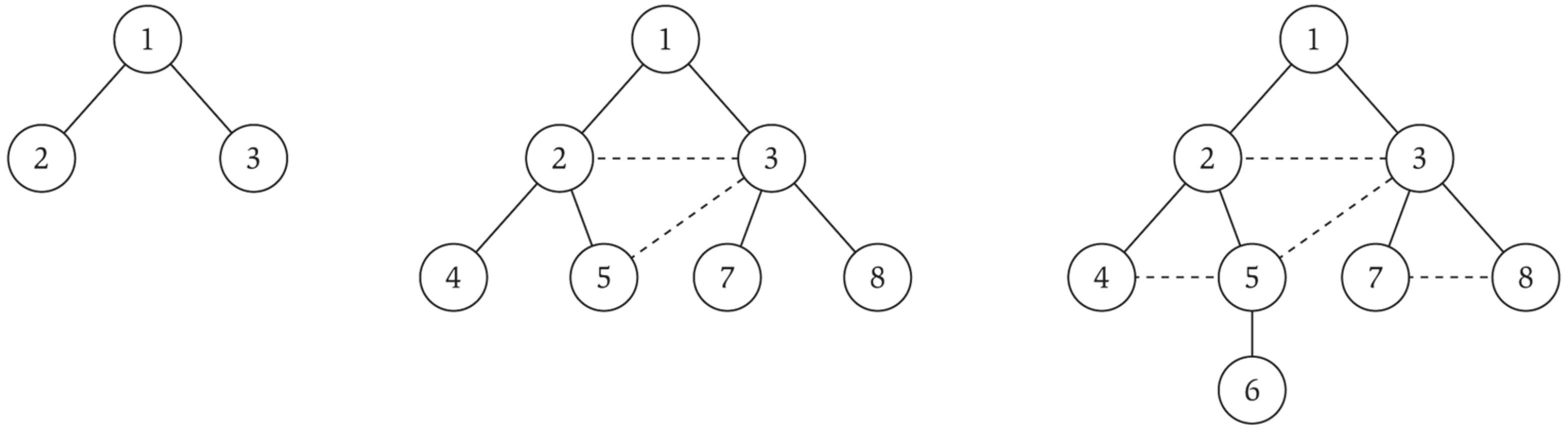
- **Informal Description:** start at s, find neighbors of s, find neighbors of neighbors of s, and so on…



- BFS Tree:
  - $L_0 = \{s\}$
  - $L_1 =$ all neighbors of L_0
  - $L_2 =$ all neighbors of L_1 that are not in $\{L_0, L_1\}$
  - $L_3 =$ all neighbors of L_2 that are not in $\{L_0, L_1, L_2\}$
  - …
  - $L_d =$ all neighbors of L_{d-1} that are not in $\{L_0, \dots, L_{d-1}\}$
  - Stop when L_{d+1} is empty

Breadth-First Search (BFS)

- **Definition:** the distance between s, t is the number of edges on the shortest path from s to t
- **Thm:** BFS finds distances from s to other nodes
  - L_i contains all nodes at distance i from s
  - Nodes not in any layer are not reachable from s

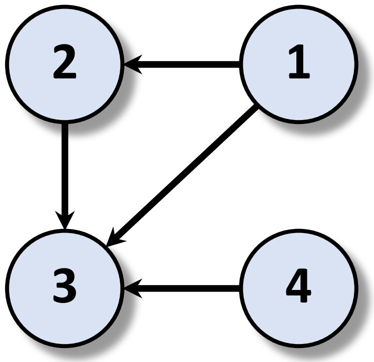


Adjacency Matrices

- The adjacency matrix of a graph $G = (V, E)$ with n nodes is the matrix $A[1{:}n\,, 1{:}n]$ where

$$A[i,j] = \begin{cases} 1 & (i,j) \in E \\ 0 & (i,j) \notin E \end{cases}$$

A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0



<u>Cost</u>

Space: $\Theta(n^2)$

Lookup: $\Theta(1)$ time

List Neighbors: $\Theta(n)$ time

Activity

- Determine if there is path between nodes 1 and 2
  All omitted entries are zero

A	1	2	3	4	5	6	7	8	9	10
1			1					1		
2				1	1					1
3	1					1		1		
4		1			1				1	
5		1		1					1	
6			1				1	1		
7						1		1		
8	1		1			1	1			
9				1	1					
10		1								

Adjacency Lists (Undirected)

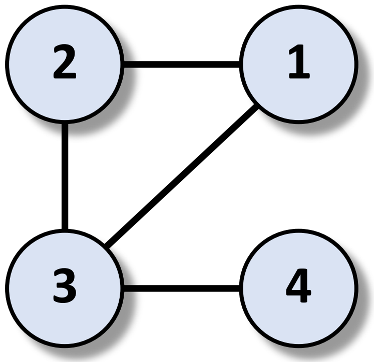
- The adjacency list of a vertex $v \in V$ is the list $A[v]$ of all u s.t. $(v, u) \in E$

$A[1] = \{2,3\}$

$A[2] = \{1,3\}$

$A[3] = \{1,2,4\}$

$A[4] = \{3\}$



<u>Cost</u>

Space: $\Theta(n + m)$

Lookup: $\Theta(\deg(u) + 1)$ time

List Neighbors: $\Theta(\deg(u) + 1)$ time

Breadth-First Search Implementation

```
BFS(G = (V,E), s):
  Let found[v] ← false ∀v, found[s] ← true
  Let layer[v] ← ∞ ∀v, layer[s] ← 0
  Let i ← 0, L₀ = {s}, T ← ∅

  While (Lᵢ is not empty):
    Initialize new layer Lᵢ₊₁
    For (u in Lᵢ):
      For ((u,v) in E):
        If (found[v] = false):
          found[v] ← true, layer[v] ← i+1
          Add (u,v) to T and add v to Lᵢ₊₁
    i ← i+1
```

Implements BFS in $O(n+m)$ time