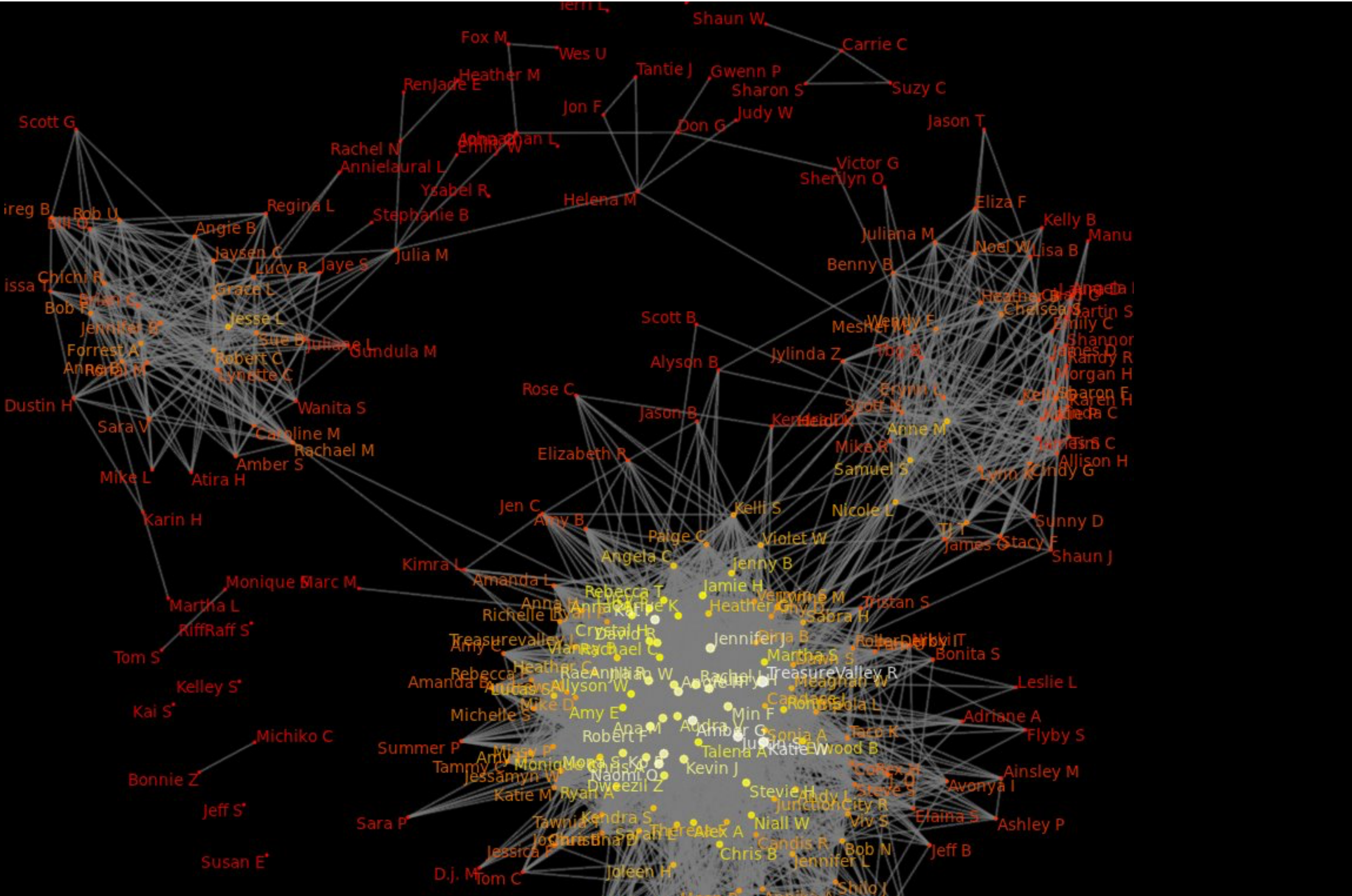
CS3000: Algorithms & Data
Paul Hand

Lecture 13:

- Introduction to Graphs
- Breadth First Search

Feb 25, 2019

Graphs


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

Graphs Are Everywhere

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

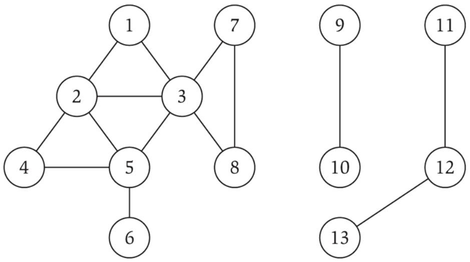
What's Next

- **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

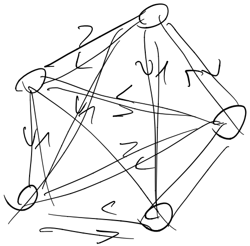
- How many edges can there be in a **simple** directed/undirected graph with n nodes?

Directed

$n \cdot (n-1)$ — # directed edges

"source nodes"

"destination nodes"



Undirected

$$\binom{n}{2} = \frac{n \cdot (n-1)}{2}$$

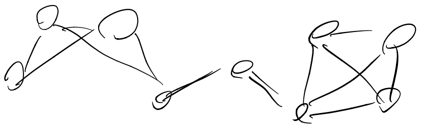
b/c all directed edges above are repeated twice

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

Eg. Path Given by (U, V) is of length $\underline{1}$

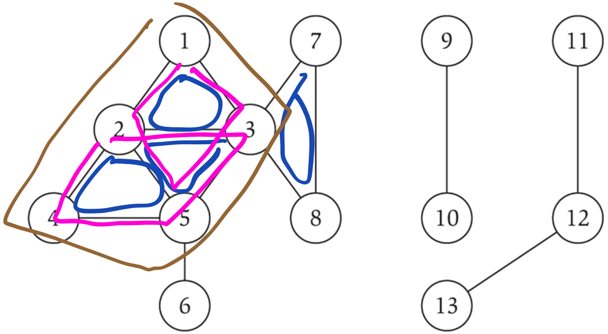
Connected



Not Connected

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



Count
1-2-3-1 & 2-3-1-2
as same
and
1-2-3-1 1-3-2-1
as same

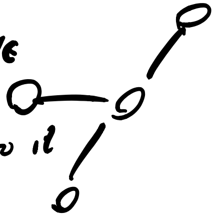
k	# of cycles of length k
3	4
4	2
5	1
	7

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

Start w/ 1 node
Add another node
that is connected to it
Repeat (n-1) times



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

n= 5
4 edges

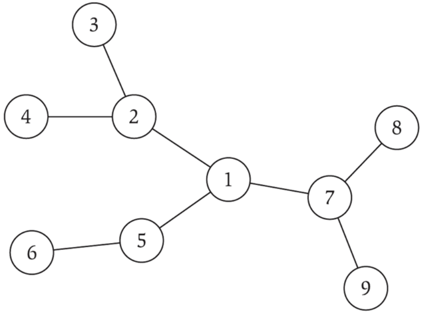
not connected

Trees

no repeat edges no self loops

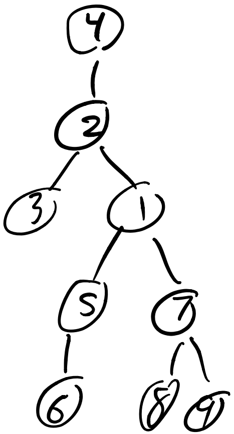
- A simple undirected graph G is a tree if:
  - G is connected
  - G contains no cycles

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

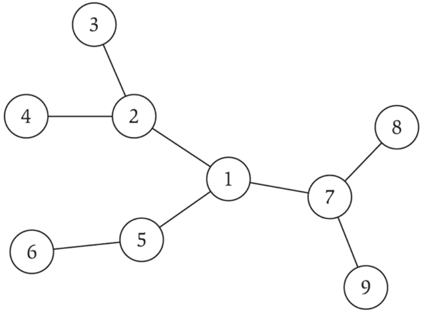


Trees

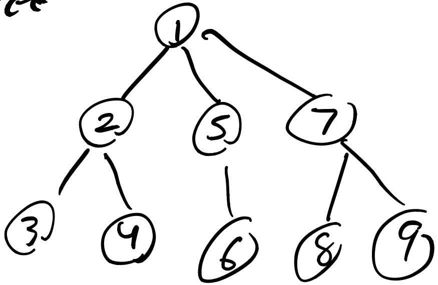
Could call $r = 4$ the root



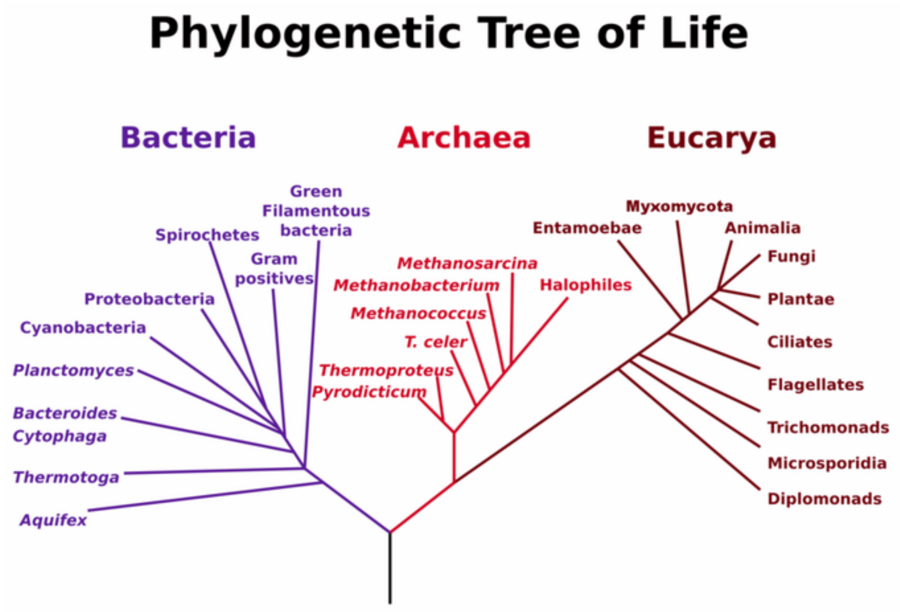
- **Rooted tree:** choose a root node r and orient edges away from r
  - Models **hierarchical structure**



Call $r = 1$ the root of this tree



Phylogeny Trees



Exploring a Graph

Exploring a Graph

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

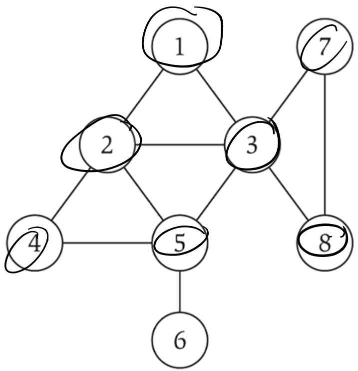
  If t is not in this set, then no path from s to t.

- 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:
  - $L_0 = \{s\}$ — where you start
  - 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 $\{L_0, \ldots, L_{d-1}\}$
  - Stop when L_{d+1} is empty



$L_0 = \{1\}$

$L_1 = \{2, 3\}$

$L_2 = \{4, 5, 7, 8\}$

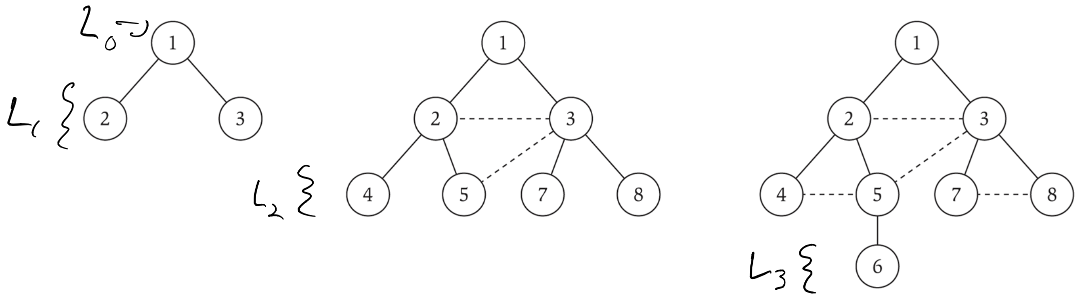
$L_3 = \{6\}$

$L_4 = \{\}$

This Builds a tree through this connected component of the graph

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

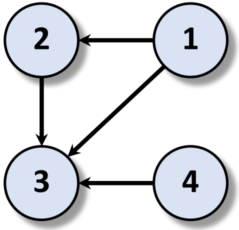
applies for directed or undirected graphs

- 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

1,2 Entry is a 1 means Edge from 1 to 2



If we have an undirected graph $A[i,j] = A[j,i]$

<u>Cost</u>

Space: $\Theta(n^2)$ doesn't depend on # edges

Edge Lookup: $\Theta(1)$ time

List Neighbors: $\Theta(n)$ time

of a particular node

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								

$L_0 = \{1\}$

$L_1 = \{3, 8\}$

$L_2 = \{6, 7\}$

$L_3 = \{\}$

2 is not connected to 1

Adjacency Lists (Undirected)

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

n # nodes

m # edges

Each edge listed twice

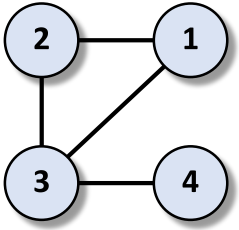
2 m cost

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

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

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

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



Cost

Space: $\Theta(n + m)$

edgs of v

Edge Lookup: $\Theta(\deg(\cancel{u}\, v) + 1)$ time

List Neighbors: $\Theta(\deg(\cancel{u}\, v) + 1)$ time

Breadth-First Search Implementation

graph ~~target~~ source

```
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
```

counts which layer of BFS tree s belongs to

$\mathcal{O}$

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