CS3000: Algorithms & Data
Jonathan Ullman

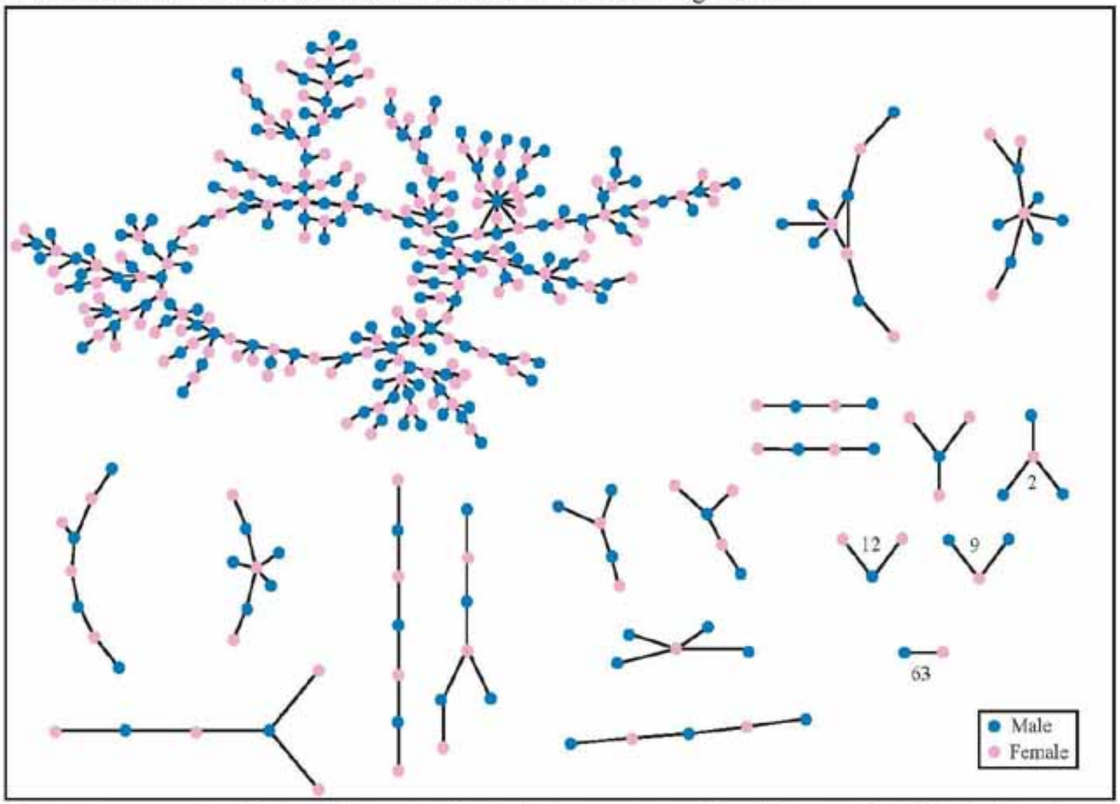
Lecture 11:

- Graphs
- Graph Traversals: BFS

Feb 16, 2018

What’s Next

The Structure of Romantic and Sexual Relations at "Jefferson High School"



Each circle represents a student and lines connecting students represent romantic relations occuring within the 6 months preceding the interview. Numbers under the figure count the number of times that pattern was observed (i.e. we found 63 pairs unconnected to anyone else).

What’s Next

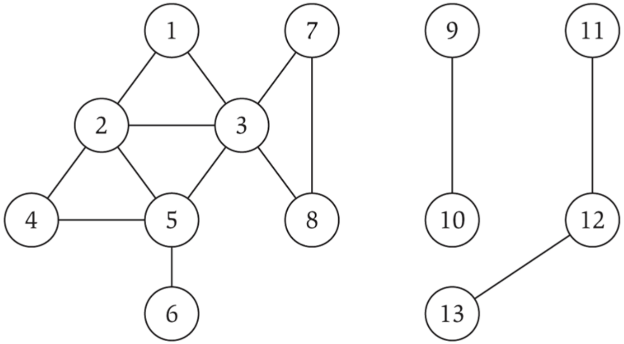
- **Graph Algorithms:**
  - **Graphs:** Key Definitions, Properties, Representations
  - **Exploring Graphs:** Breadth/Depth First Search
    - Applications: Connectivity, Bipartiteness, Topological Sorting
  - **Shortest Paths:**
    - Dijkstra
    - Bellman-Ford (Dynamic Programming)
  - **Minimum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

Graphs

Graphs: Key Definitions

$$\boxed{\begin{array}{l} |V| = n \\ |E| = m \end{array}}$$ Notation

- **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 $e = (u, u)$



Ask the Audience

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

directed: (n nodes) × (n−1 possible edges per node)

$= n(n-1)$

undirected $= \frac{n(n-1)}{2}$ (b/c $(u,v) = (v,u)$)

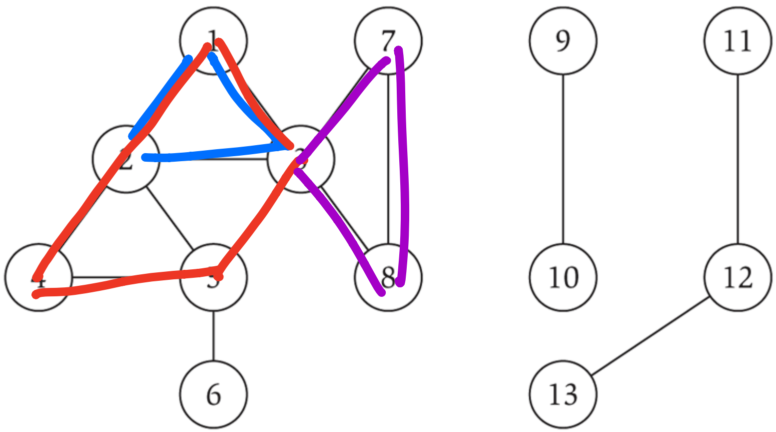
$m = O(n^2)$ for simple graphs!

Paths/Connectivity

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



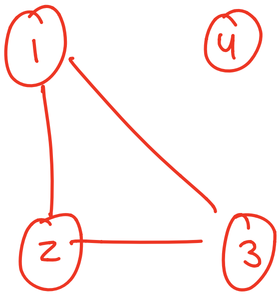
graphs can have many cycles

Ask the Audience

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

Ask the Audience

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



$n = 4$

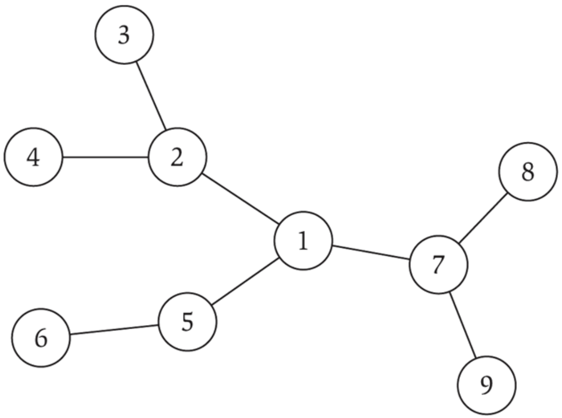
$m = 3 = n - 1$

G is not connected

Trees

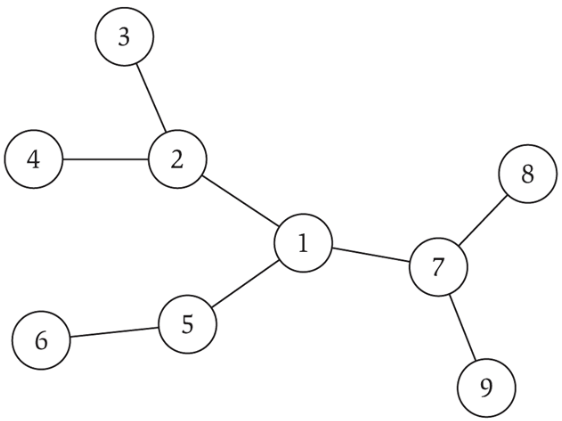
- A simple undirected graph G is a tree if:
  - G is connected
  - G contains no cycles

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

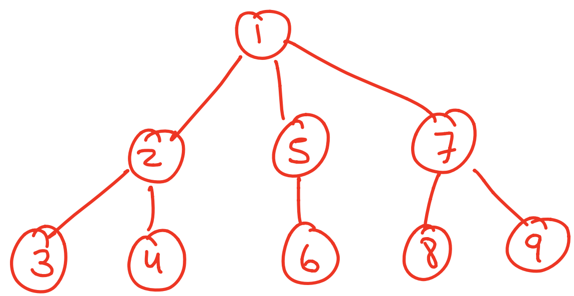


Trees

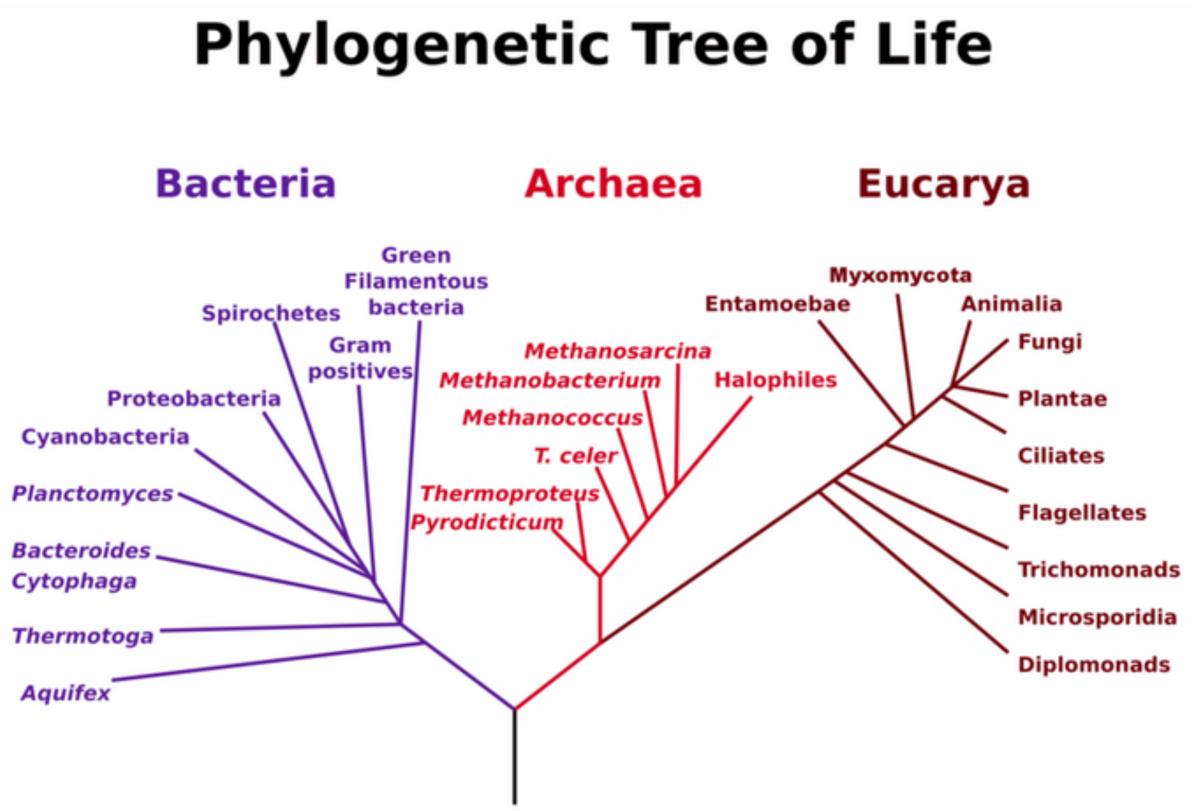
- Rooted tree: choose a root node r and orient edges away from r
  - Models **hierarchical structure**



$r = 1$



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

Exploring a Graph

- **BFS/DFS** are general templates for graph algorithms
  - Extensions of **Breadth-First Search**:
    - 2-Coloring (Bipartiteness)
    - Shortest Paths
    - Minimum Spanning Tree (Prim's Algorithm)
  - Extensions of **Depth-First Search**:
    - Fast Topological Sorting
    - Fast Strongly Connected Components

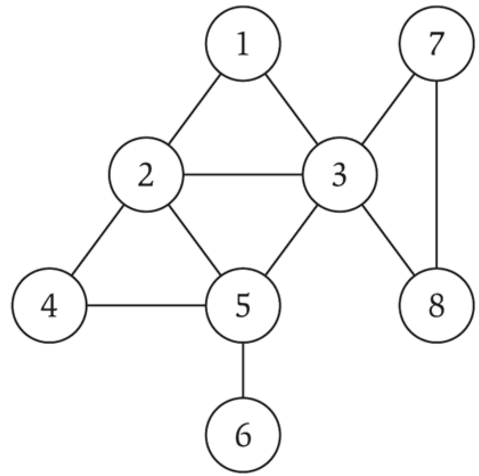
Breadth-First Search (BFS)

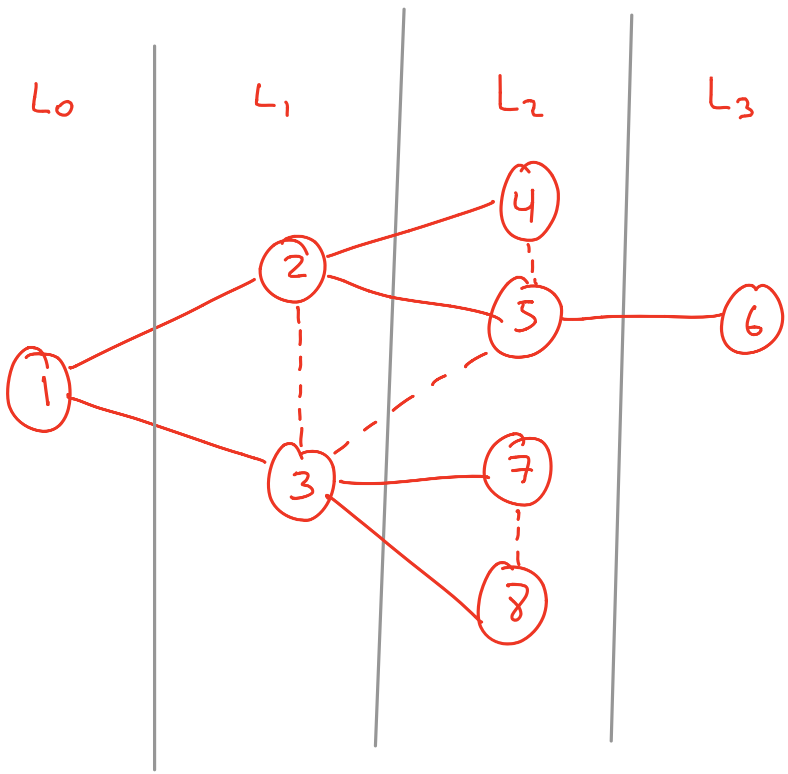
- **Informal Description:** start at s, find neighbors of s, find neighbors of neighbors of s, and so on…

- 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 $L_0, \ldots, L_{d-1}$
  - Stop when L_{d+1} is empty

Ask the Audience

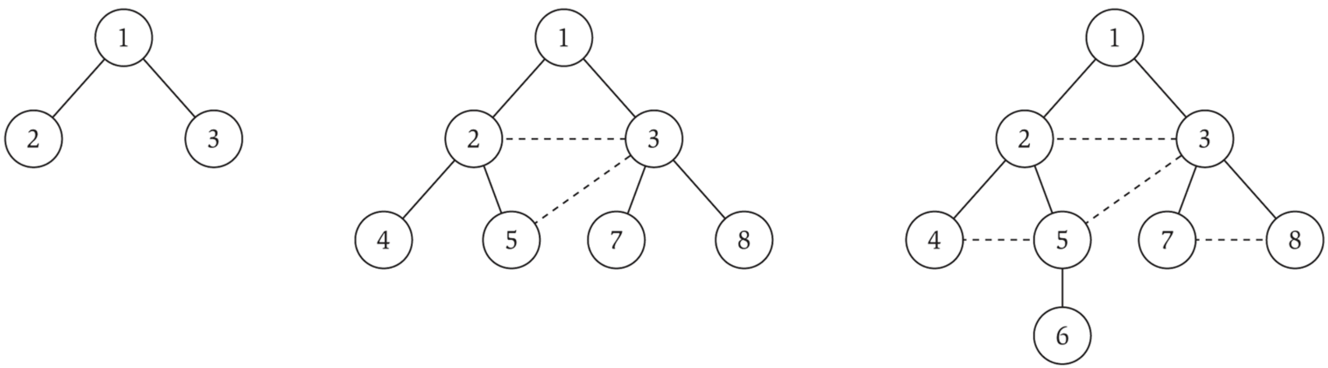
- BFS this graph from $s = 1$





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

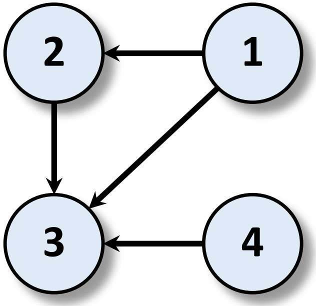


Adjacency Matrices

- 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: $\Theta(V^2)$

Lookup: $\Theta(1)$ time

List Neighbors: $\Theta(V)$ time

Adjacency Lists (Undirected)

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

$$\text{Space} = \sum_{u \in V} O(\deg(u) + 1)$$

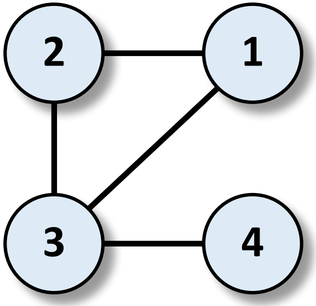
$$= O(n + m)$$

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

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

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

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



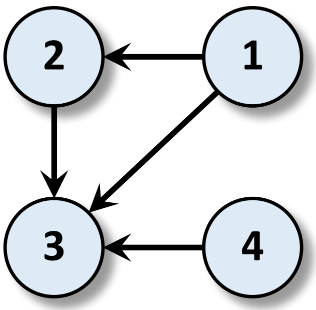
Cost to lookup $(u, v) = O(\deg(u) + 1)$

Cost to List Neighbors of u = $O(\deg(u) + 1)$

Adjacency Lists (Directed)

- The adjacency list of a vertex $v \in V$ are the lists
  - $A_{out}[v]$ of all u s.t. $(v, u) \in E$
  - $A_{in}[v]$ of all u s.t. $(u, v) \in E$

$A_{out}[1] = \{2,3\}$ $A_{in}[1] = \{\,\}$
$A_{out}[2] = \{3\}$ $A_{in}[2] = \{1\}$
$A_{out}[3] = \{\,\}$ $A_{in}[3] = \{1,2,4\}$
$A_{out}[4] = \{3\}$ $A_{in}[4] = \{\,\}$



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, L_0 = {s}, T ← ∅

  While (L_i is not empty):
    Initialize new layer L_{i+1}
    For (u in L_i):
      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 L_{i+1}
    i ← i+1
```

BFS Running Time (Adjacency List)

```
BFS(G = (V,E), s):
  Let found[v] ← false ∀v
  Let found[s] ← true
  Let layer[v] ← ∞ ∀v, layer[s] ← 0
  Let i ← 0, L_0 = {s}, T ← Ø

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

$O(1)$

Total Work:

$\sum_{u \in V} O(\deg(u)+1) = O(n+m)$

Only explore u once!

For every u, we do $O(\deg(u)+1)$ work