HW3 due 6/6

No Quiz

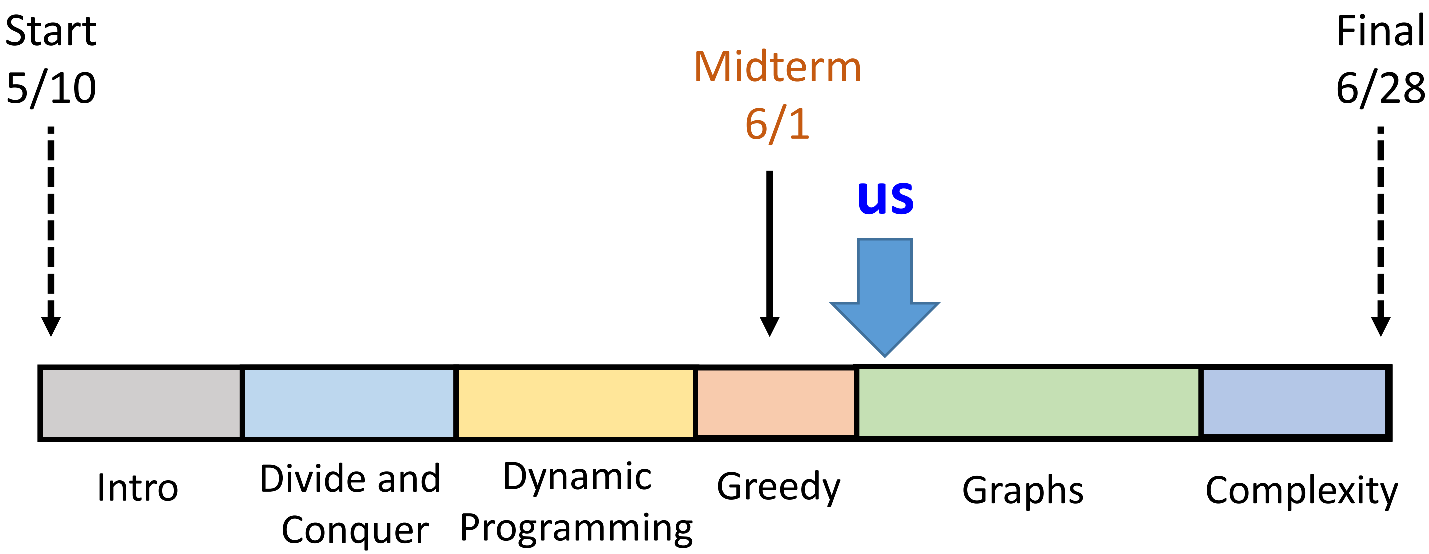
Tag Pages on HW

CS3000: Algorithms & Data
Drew van der Poel

Lecture 14
- Graphs
- Graph Traversals: BFS

June 3, 2021

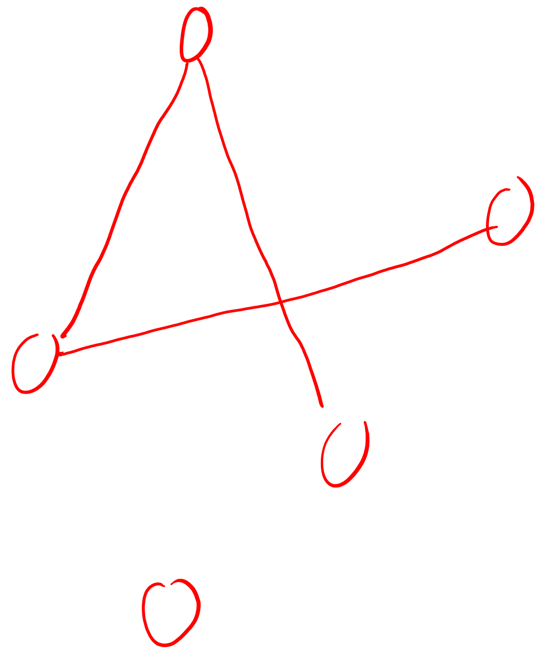
Outline



Last class: greedy: Huffman codes

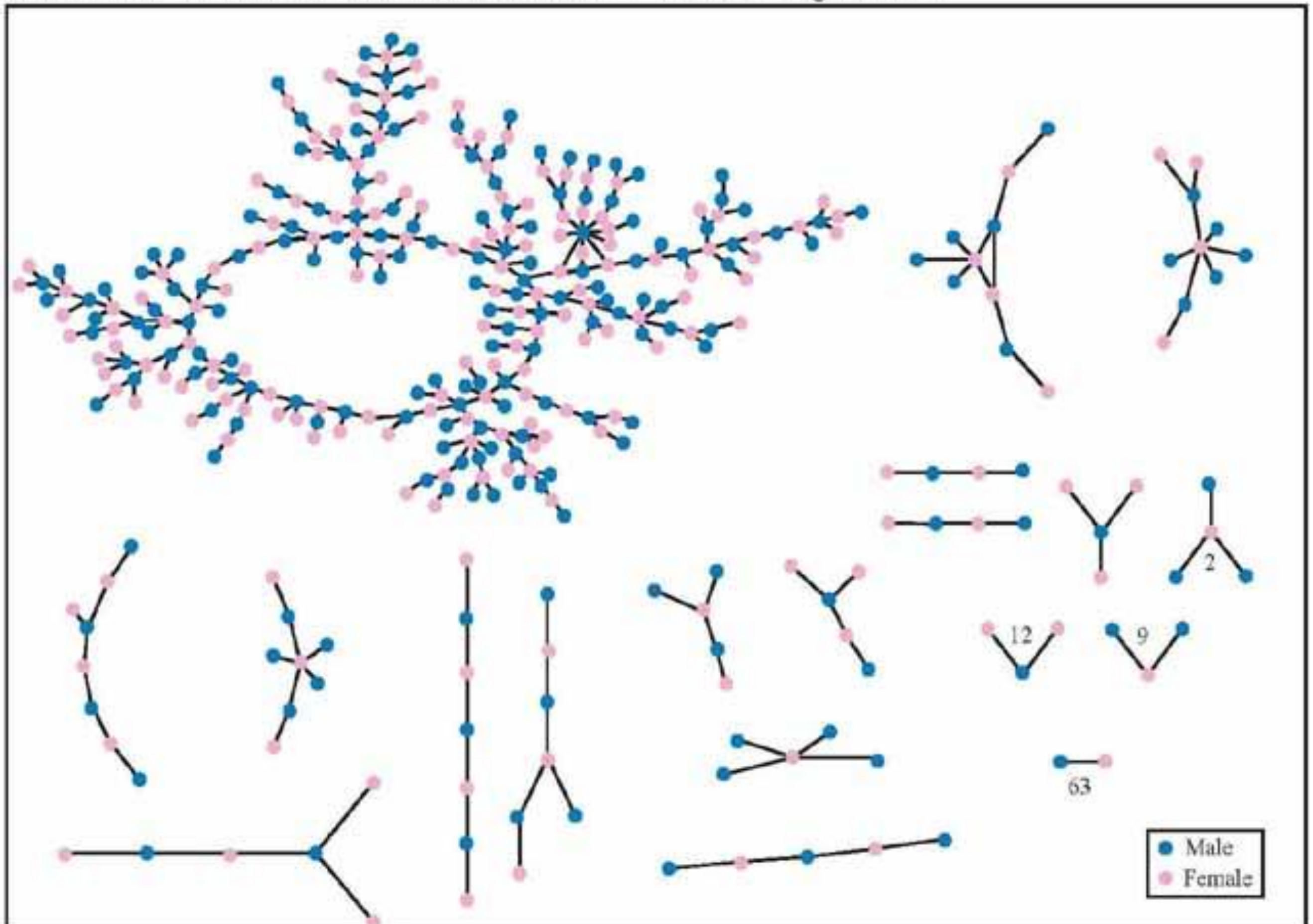
Next class: Graphs: DFS, 2-Colorability & Topological Ordering

Graphs



Graphs Are Everywhere

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).

Graphs Are Everywhere

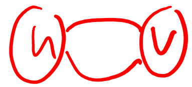
- Transportation networks
- On the internet
- Biological networks
- Citation networks
- Social networks
- …

On the horizon…

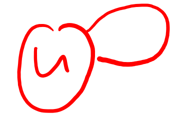
- **Graph Algorithms:**

Today
  - **Graphs:** Key Definitions, Properties, Representations
  - **Exploring Graphs:** Breadth/Depth First Search
    - Applications: Connectivity, Bipartiteness, Topological Sorting, SCCs
  - **Shortest Paths:**
    - Dijkstra
    - Bellman-Ford (Dynamic Programming)
  - **Minimum Spanning Trees:**
    - Borůvka, Prim, Kruskal
  - **Network Flow:**
    - Max Flow/Min Cut
    - Ford-Fulkerson

Graphs: Key Definitions



X not allowed



X not allowed

- **Definition:** An undirected graph $G = (V, E)$
  - V is the set of nodes/vertices
  - $E \subseteq V \times V$ is the set of edges
  - Edges are unordered $e = (u, v)$ "between u and v" (u and v are neighbors)
  - Degree(v) = # of neighbors v has

$(u,v) = (v,u)$

$e = uv$

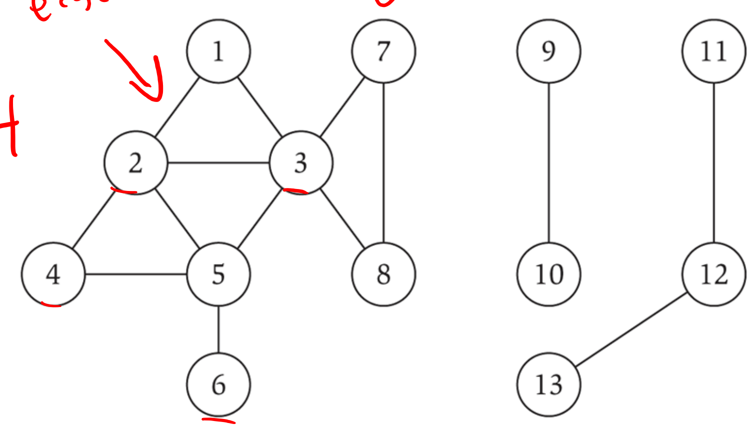
nodes

$n = 13$
$m = 14$

- $|V| = n, |E| = m$

edge

$deg(5) = 4$



- **Simple Graph:**
  - No duplicate edges
  - No self-loops $e = (u, u)$

Graphs: Key Definitions

✓ permitted

X not allowed in simp. graph

- **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" (u is an in-neighbor of v, v is an out-neighbor of u)
  - In/out-degree(v) = # of in/out neighbors of v

$(u,v) \neq (v,u)$

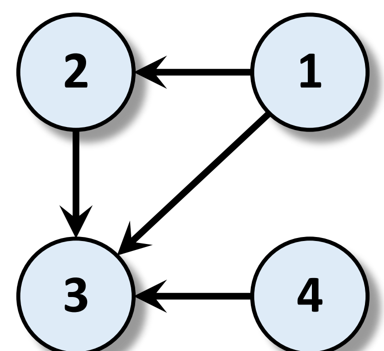
in-deg(1) = 0

out-deg(1) = 2

- $|V| = n, |E| = m$

n = 4

m = 4



- **Simple Graph:**
  - No duplicate edges
  - No self-loops $e = (u, u)$

X not allowed

- Problems: counting students, stable matching, sorting, n-digit mulitiplication, array searching, selection, weighted interval scheduling, segmented least squares, knapsack, prefix-free encoding
- Alg. techniques: divide & conquer, dynamic programming, greedy
- Analysis: asymptotic analysis, recursion trees, Master Thm., **Graph Terminology**
- Proof techniques: (strong) induction, contradiction, greedy stays ahead, exchange argument

Ask the Audience

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

Directed: $O(n^2)$ $\quad (n)(n-1)$

$= n^2 - n$

Undirected: $O(n^2) \leftarrow \dfrac{(n)(n-1)}{2}$

$= \dfrac{n^2 - n}{2}$

$\binom{n}{2}$ pairs of nodes

Ask the Audience

- What is the **total degree** ($\Sigma_{v \in V} \deg(v)$) in an undirected graph with m edges?

$2m$

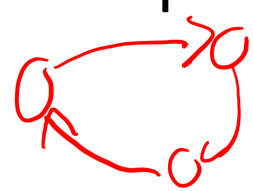
- What is the **total in-degree** ($\Sigma_{v \in V}$ in-deg(v)) in a directed graph with m edges?

m

Paths/Connectivity

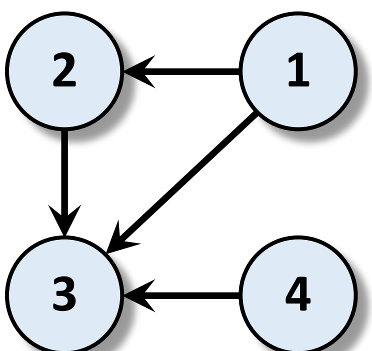
- A path is a sequence of consecutive edges in E
  - $P = \{(u, w_1), (w_1, w_2), (w_2, w_3), \dots, (w_{k-1}, v)\}$
  - $P = u - w_1 - w_2 - w_3 - \cdots - w_{k-1} - v$ len(P) = K
  - The length of the path is the # of edges
  - A path is *simple* if all vertices on the path are unique

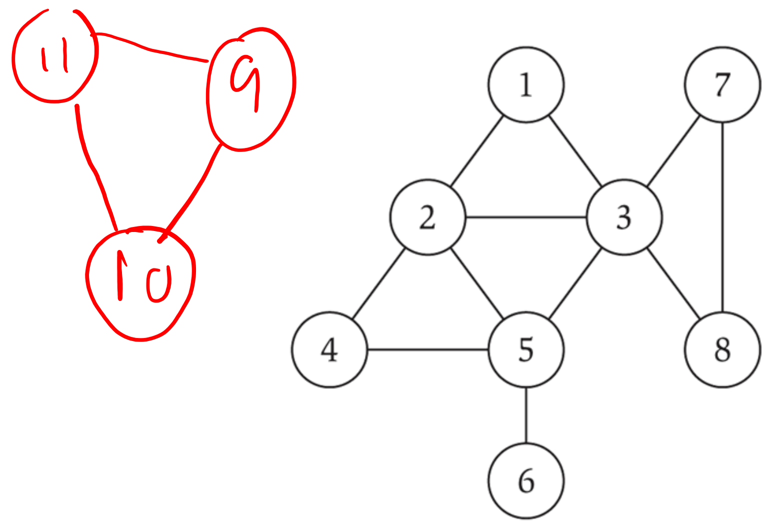
- An undirected graph is connected if for every two vertices $u, v \in V$, there is a path from u to v
- A directed graph is strongly connected if for every two vertices $u, v \in V$, there are paths from u to v and from v to u



Paths/Connectivity

- 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





Strongly connected?

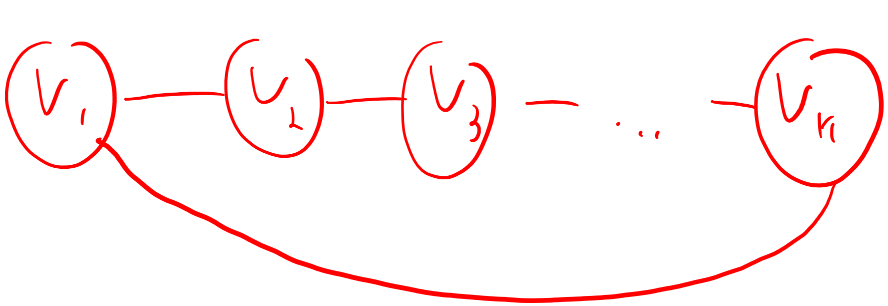
No - no path from 3 to 2 (or 1 or 4)

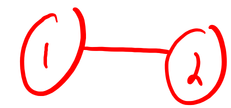
Connected?

yes

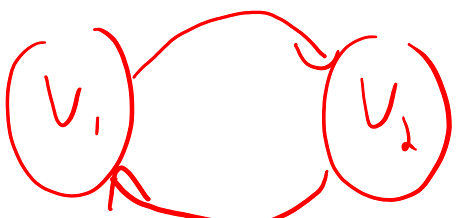
Cycles

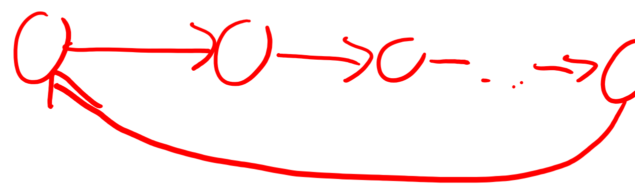
- An undirected cycle is a path $v_1 - v_2 - \cdots - v_k - v_1$ where $k \geq 3$ and $v_1, \ldots, v_k$ are distinct
- Directed cycles can have $k \geq 2$





Not a cycle

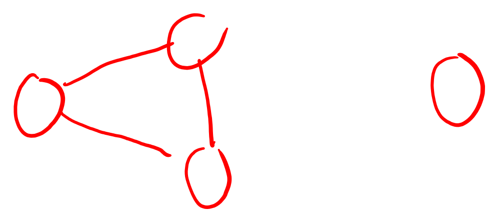




Ask the Audience

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

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



$n = 4$

$m = 3 = n - 1$

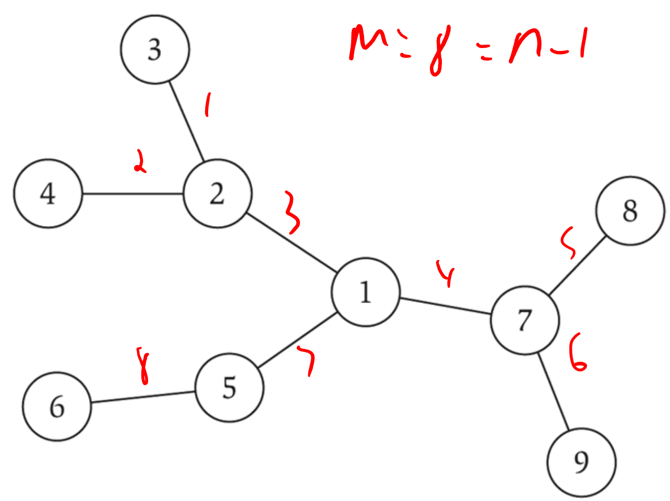
Trees

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

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

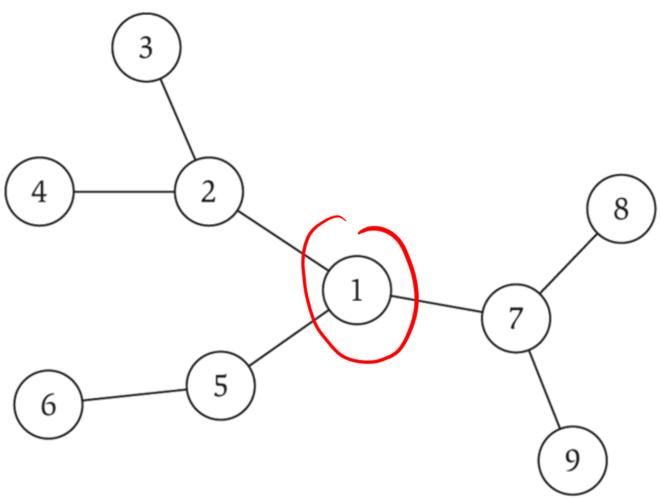
$n = 9$

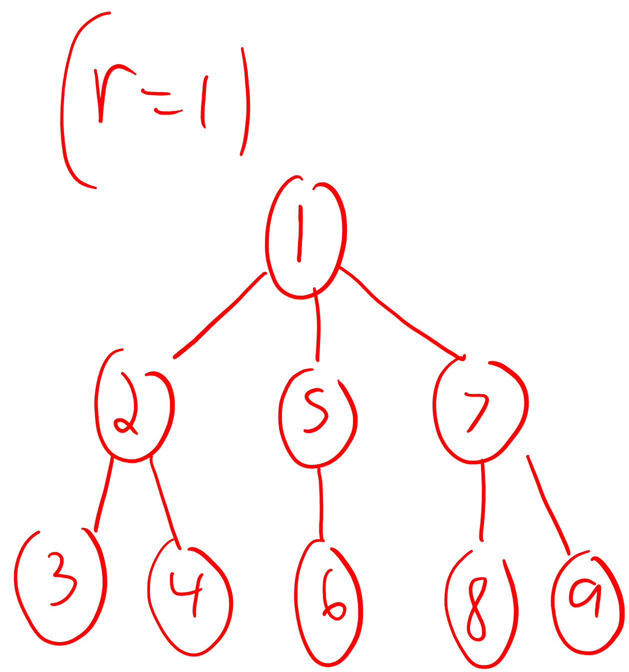
$m = 8 = n - 1$



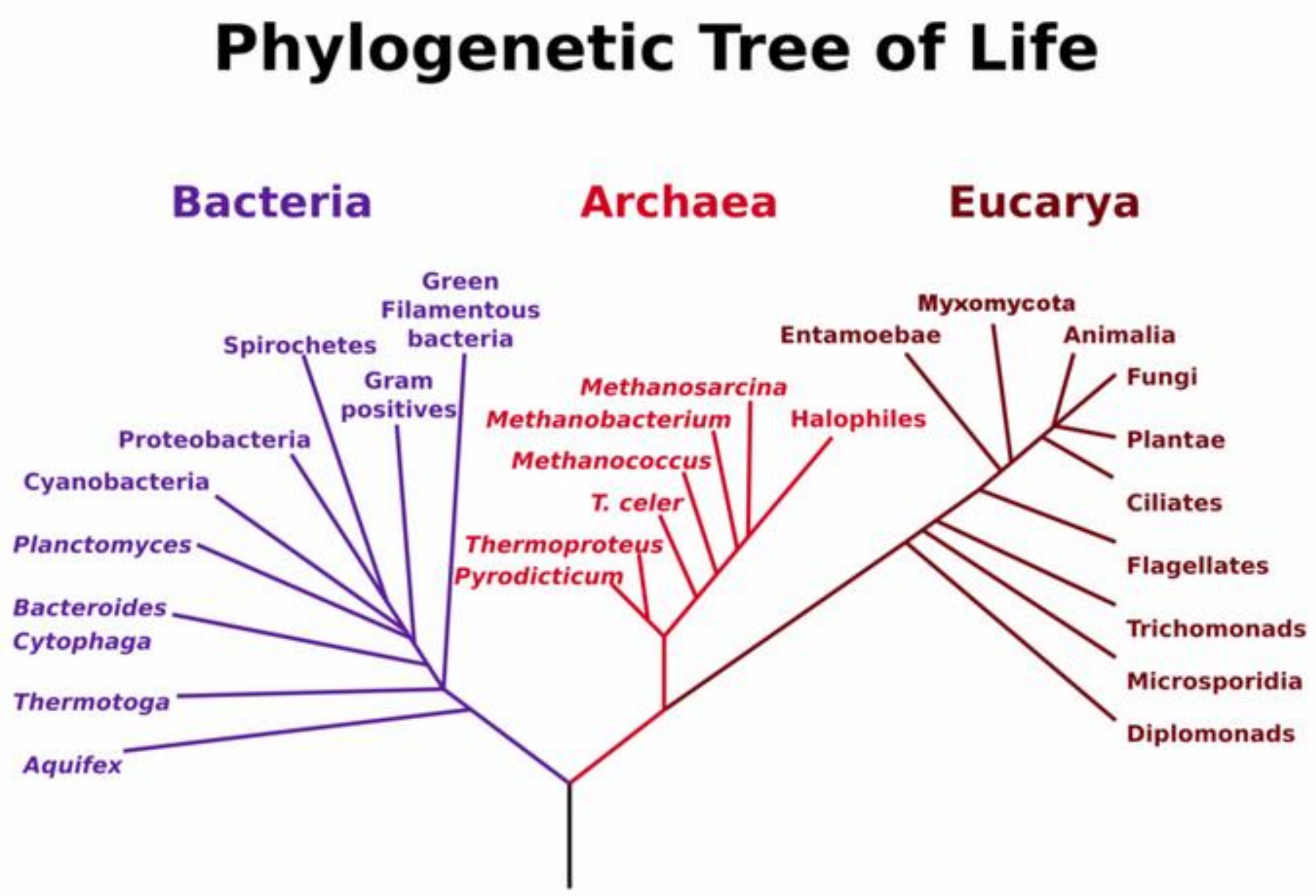
Trees

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





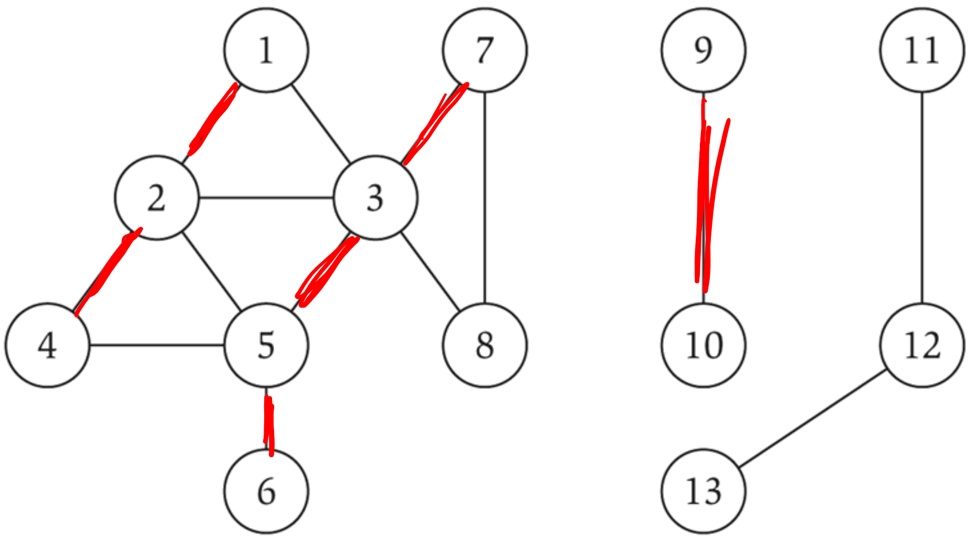
Phylogeny


Phylogenetic Tree of Life
Bacteria
Archaea
Eucarya
Green Filamentous bacteria
Spirochetes
Gram positives
Proteobacteria
Cyanobacteria
Planctomyces
Bacteroides Cytophaga
Thermotoga
Aquifex
Methanosarcina
Methanobacterium
Halophiles
Methanococcus
T. celer
Thermoproteus
Pyrodicticum
Entamoebae
Myxomycota
Animalia
Fungi
Plantae
Ciliates
Flagellates
Trichomonads
Microsporidia
Diplomonads

Exploring a Graph

Exploring a Graph

- **Problem:** Is there a path from s to t?



From 6 to 7? 1 to 4? 9 to 10? 8 to 11?

Yes Yes Yes NO → G is not connected

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

- Problems: counting students, stable matching, sorting, n-digit mulitiplication, array searching, selection, weighted interval scheduling, segmented least squares, knapsack, prefix-free encoding, **graph exploration**

- Alg. techniques: divide & conquer, dynamic programming, greedy

- Analysis: asymptotic analysis, recursion trees, Master Thm., Graph Terminology

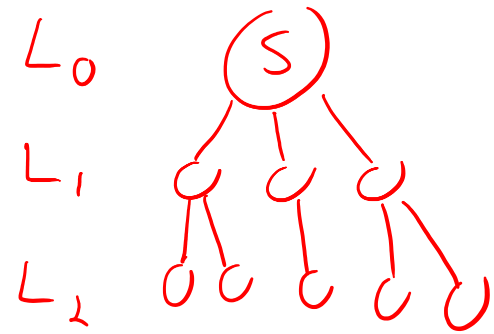
- Proof techniques: (strong) induction, contradiction, greedy stays ahead, exchange argument

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

Breadth-First Search (BFS)

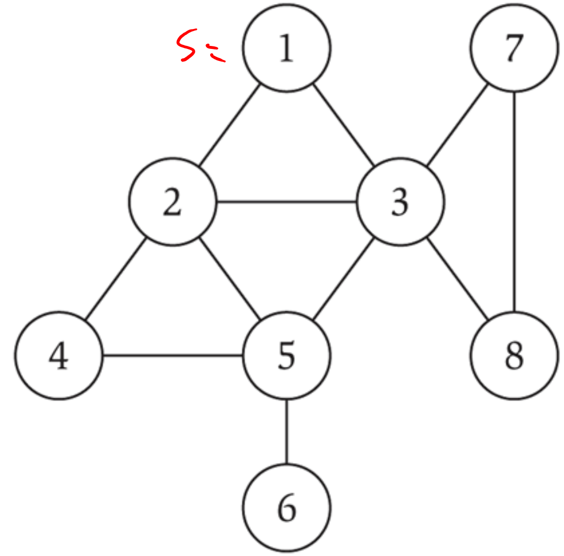
- **Informal Description:** start at s, find neighbors of s, find neighbors of neighbors of s, and so on…

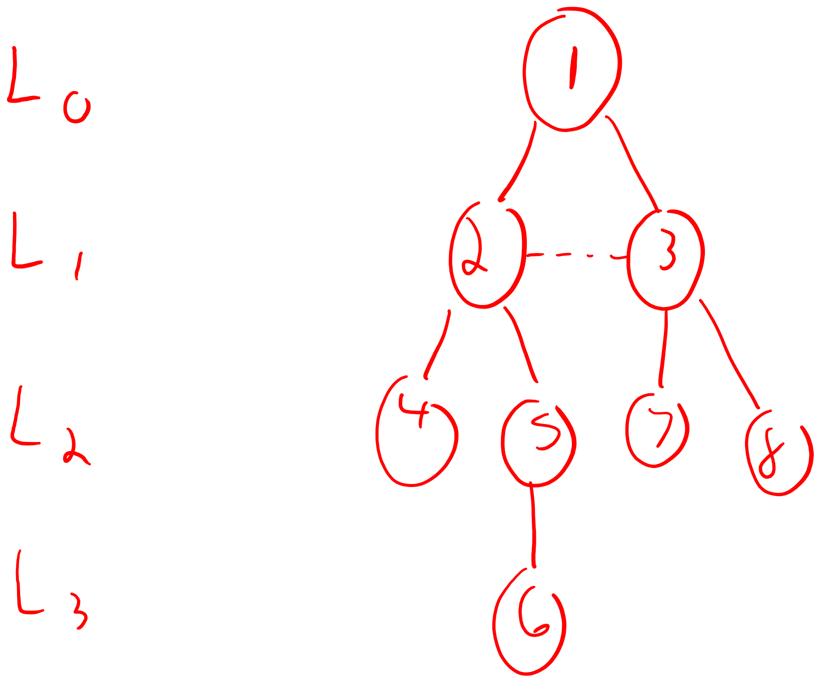


- BFS Tree:
  - $L_0 = \{s\}$
  - $L_1 =$ all neighbors of L_0
  - $L_2 =$ all neighbors of L_1 that are not in L_0, L_1
  - $L_3 =$ all neighbors of L_2 that are not in L_0, L_1, L_2
  - …
  - $L_d =$ all neighbors of L_{d-1} that are not in $L_0, \ldots, L_{d-1}$
  - Stop when L_{d+1} is empty

Ask the Audience

- Run a BFS from $\boldsymbol{s = 1}$

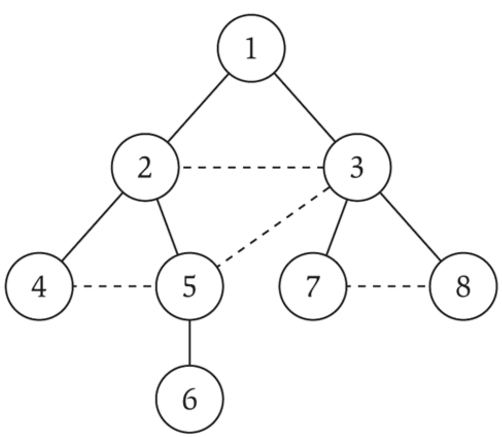




tree edges

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 all other nodes
  - L_i contains all nodes at distance i from s
  - Nodes not in any layer are not reachable from s



$d(1,6) = 3$

(Sidebar) Graphs: Representations/Storage

Adjacency Matrices

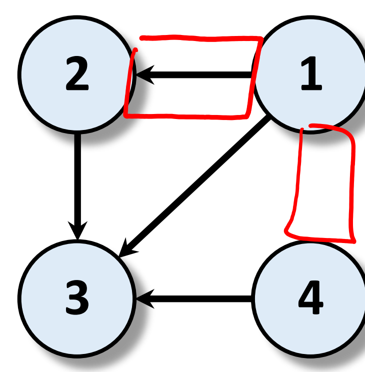
- The adjacency matrix of a graph $G = (V, E)$ with n nodes is the matrix $A[1{:}n\,,1{:}n]$ where

j

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

i

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



Cost

Space: $O(n^2) \leftarrow n \times n = n^2$

Check if edge (u,v) exists: $O(1)$

List(out/in)-Neighbors of *v*: $O(n)$

Adjacency Lists (Undirected)

m = # of edges   n = # of nodes

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

each edge adds 2 elems to the space

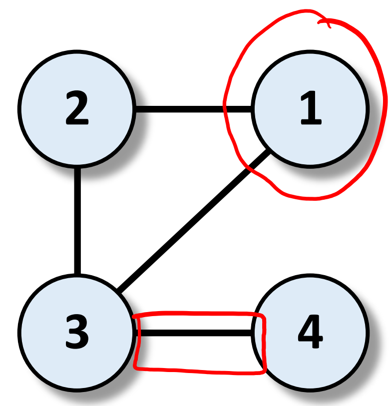
$$A[1] = \{2,3\}$$
$$A[2] = \{1,3\}$$
$$A[3] = \{1,2,4\}$$
$$A[4] = \{3\}$$

<u>Cost</u>

Space: $O(m + n)$

Check if edge (u,v) exists: $O(\deg(u))$

List Neighbors of *v*: $O(\deg(v))$



Adjacency Lists (Directed)

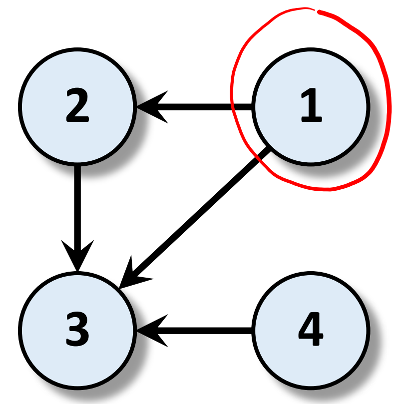
- The adjacency list of a vertex $v \in V$ are the lists
  - $A_{out}[v]$ of all u s.t. $(v, u) \in E$
  - $A_{in}[v]$ of all u s.t. $(u, v) \in E$

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

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

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

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



2 elems, per edge

2 lists per node

<u>Cost</u>

Space: $O(m + n)$

Check if edge (u,v) exists: $O(\text{out-deg}(u))$ or $O(\text{in-deg}(v))$

List (in-/out-)Neighbors of *v*: $O(\text{in-deg}(v))$

- Problems: counting students, stable matching, sorting, n-digit mulitiplication, array searching, selection, weighted interval scheduling, segmented least squares, knapsack, prefix-free encoding, graph exploration

- Alg. techniques: divide & conquer, dynamic programming, greedy

- Analysis: asymptotic analysis, recursion trees, Master Thm., Graph Terminology/**representations**

- Proof techniques: (strong) induction, contradiction, greedy stays ahead, exchange argument

BFS Running Time (Adjacency List)

```
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 (v in A[u]):
        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
  Return T, layer
```

$O(n)$

$O(1)$

$O(n)$ times in total

Adj. list of node u

$\deg(u)$

$O(1)$

$O(1)$

$O(1)$

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

Total: $O(n+m)$

Ask the Audience

- A directed graph is **strongly connected** if for every pair $u, v \in V$, u is reachable from v and vice versa
- How can you (naively) use BFS to determine if a graph is strongly connected? What is the runtime of your approach?