Week 3 - Wednesday

COMP 4500

Last time

- What did we talk about last time?
- Finished proofs by induction
- Graph definitions
- Graph applications

Questions?

Assignment 2

Logical warmup

- By their nature, manticores lie every Monday, Tuesday, and Wednesday and speak the truth on other days
- However, unicorns lie on Thursdays, Fridays, and Saturdays and speak the truth the rest of the week
- A manticore and a unicorn meet and have the following conversation:

  Manticore: Yesterday I was lying.

  Unicorn: So was I.
- On which day did they meet?

Path definitions

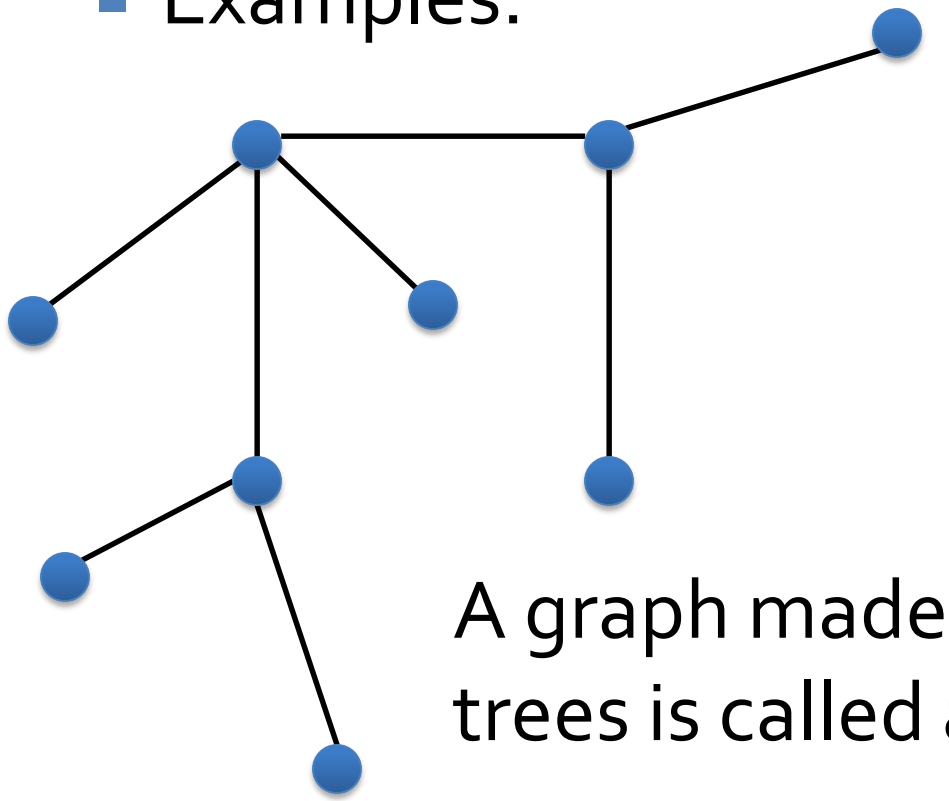
- A **path from *u* to *v*** is sequence of vertices where each vertex in the sequence is connected by an edge to the next
- A **simple path from *u* to *v*** is a path that does not contain a repeated vertex
- A **cycle** is a path that starts and ends at the same vertex but repeats no other vertices

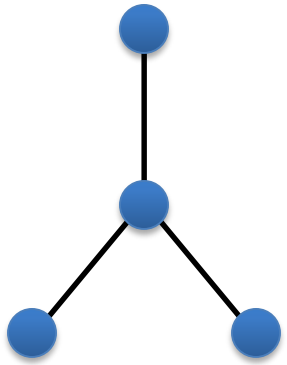
Connectedness

- Vertices ***u*** and ***v*** of ***G*** are **connected** iff there is a path from ***u*** to ***v***
- An undirected graph ***G*** is connected iff all pairs of vertices ***u*** and ***v*** are connected
- A directed graph G is is strongly connected iff all pairs of vertices ***u*** and ***v*** are connected, in both directions
- The (unweighted) **distance** between vertices ***u*** and ***v*** is the minimum number of edges in a ***u*-*v*** path

Trees

- A **tree** is a graph that is **circuit-free** and **connected**
- Examples:





A graph made up of disconnected trees is called a **forest**

Rooted trees

- In a **rooted tree**, one vertex is distinguished and called the **root**
- The **level** of a vertex is the number of edges along the unique path between it and the root
- The **children** of any vertex ***v*** in a rooted tree are all those nodes that are adjacent to ***v*** and one level further away from the root than ***v***
- If ***w*** is a child of ***v***, then ***v*** is the **parent** of ***w***
- If ***v*** is on the unique path from ***w*** to the root, then ***v*** is an ancestor of ***w*** and ***w*** is a descendant of ***v***
- Rooted trees allow us to represent a **hierarchy**

Trees have $n - 1$ edges

- **Proof:** Pick a node ***r*** to use as a root of the tree. Every node ***u*** (except for ***r***) only has one parent. Thus, every edge goes upward from exactly one non-root node. Since there are $n - 1$ non-root nodes, there are $n - 1$ edges in a tree.

Three-Sentence Summary of Graph Connectivity and Traversal and Implementations with Queues and Stacks

Graph Connectivity and Traversal

s-*t* connectivity

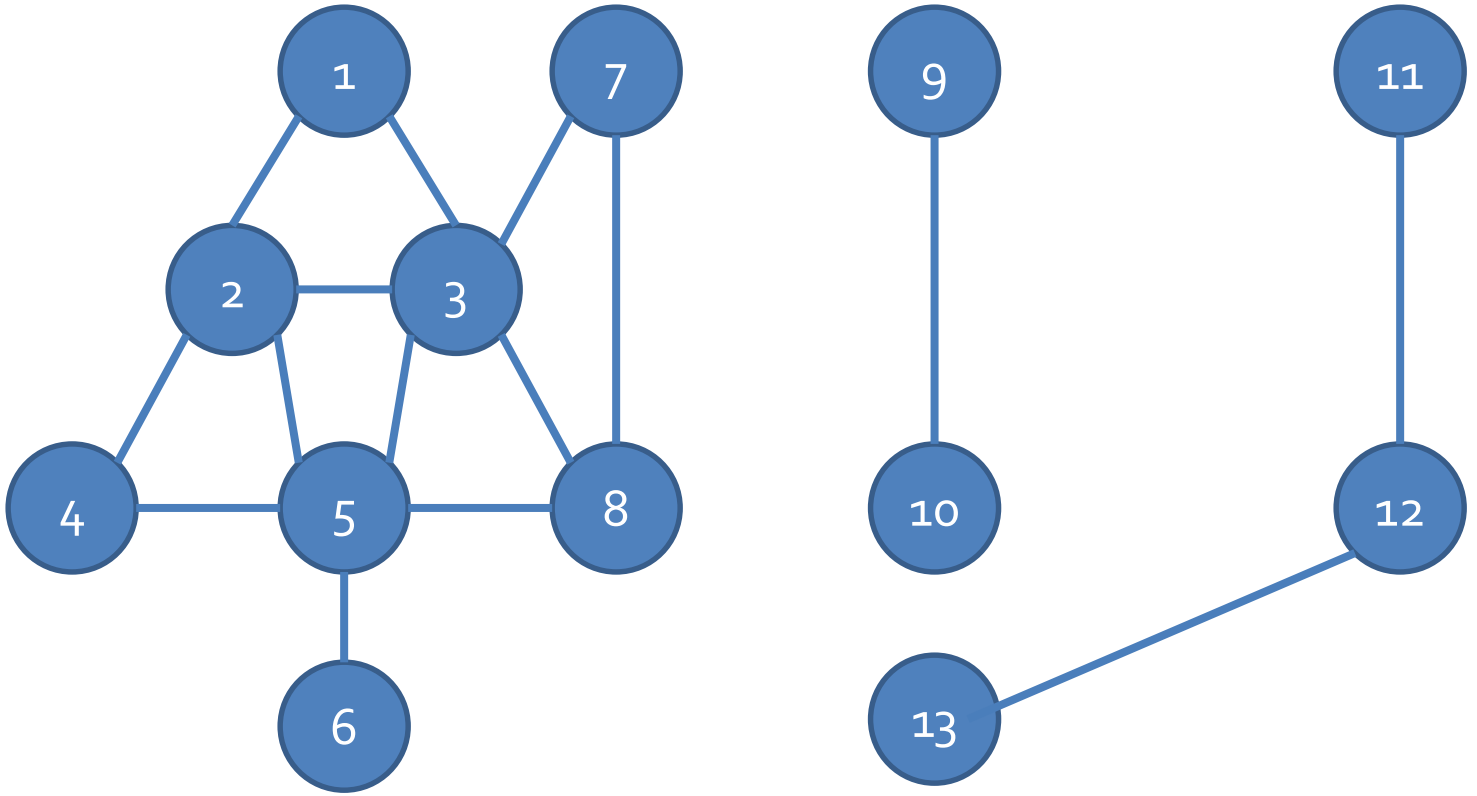
- Imagine that we have two particular nodes, ***s*** and ***t***, in our graph and we want to see if there is **any** path from ***s*** to ***t***
- This problem is like a maze solving problem
- What's an efficient way to do this, knowing nothing about the relative connections between ***s*** and ***t***?

Breadth first search

- Although we started with **depth first search** (DFS) in Data Structures, a **breadth first search** (BFS) is perhaps a more natural way to search
- Explore out from node ***s*** in layers, where each layer is one more step away from ***s*** (provided that it hasn't already been visited)
  - Think of it as a flood from ***s***
- When we can't reach any more nodes, either ***t*** will have been in one of the layers (reachable) or not (unreachable)

Example graph

- Given the following graph, describe each layer, when ***s*** = node 1



BFS trees

- Both a BFS and a DFS will produce a tree based on the order in which nodes are visited
- When examining a node ***u***, a new node ***v*** might be found
  - When that happens, draw an edge from ***u*** to ***v*** in the breadth first search tree
- Draw the breadth first search tree for the example graph

Connected components

- The set of nodes discovered by a BFS are exactly those reachable from node ***s***
- We call this set of nodes a **connected component**
- A graph could have only a single connected component or many

Depth first search

- In contrast, a depth first search goes as far from the starting node ***s*** as possible until it hits a dead end
- Only then will it backtrack
- Both BFS and DFS are special cases of the generic algorithm that will keep adding nodes to a connected component (***R***) until it can't find new ones

Depth first search algorithm

- DFS(***u***):
  - Mark ***u*** as "Explored" and add ***u*** to set ***R***
  - For each edge (***u***, ***v***)
    - If ***v*** is not marked "Explored" then
      - Recursively invoke DFS(***v***)

DFS trees

- Although a BFS and a DFS will both visit all of the nodes in a connected component, the orders are usually different
  - BFS trees tend to be bushy and not very deep
  - DFS trees tend to be narrow and deep
- A **depth first search tree** can be built by putting an edge between ***u*** and ***v*** if DFS(***v***) is invoked while visiting ***u***
- Draw the depth first search tree for the example graph

Connected components

- **Claim:** For all nodes ***s*** and ***t*** in a graph, their connected components are either the same or disjoint
- **Proof:** Consider nodes ***s*** and ***t*** where there is a path between them. For any node ***v*** to be in the component of ***s***, there must be a path from ***s*** to ***v***. For any node ***v*** to be in the component of ***t***, there must be a path from ***t*** to ***v***. Since these paths exist and one could always take a path from ***s*** to ***t*** before visiting ***v*** or vice versa, all such nodes must be in a single connected component. Now consider ***s*** and ***t*** with no path between them. There cannot be a node ***v*** that is in the connected component of both, since it would create a path from ***s*** to ***t***.

Representing Graphs

Representing graphs

- We think of a graph $G = (V, E)$
- We will often let variables $n = |V|$ and $m = |E|$
- Is $O(m^2)$ or $O(n^3)$ a better running time?
  - Depends!
- For graphs, we will consider $O(m + n)$ to be linear, since that's how much input we need to describe the graph

Adjacency matrix

- A simple way of keeping track of the edges in a graph is an **adjacency matrix**
- An adjacency matrix is an n x n matrix where n is the number of nodes
- The number in row i column j is the number of edges between node i and node j
- Undirected graphs have symmetrical adjacency matrices
- Two weaknesses:
  - $\Theta(n^2)$ space, even for sparse graphs
  - $\Theta(n)$ time to check all of the edges for a node

Adjacency lists

- An adjacency matrix wastes a lot of space if the graph is not very dense
- An alternative is an adjacency list
- The form of an adjacency list is an array of length ***n*** where the ***i***th element is a pointer to a linked list (or dynamically allocated array) of the nodes adjacent to node ***i***
- The book assumes this implementation

Queues and stacks

- A **queue** is a set where we extract elements in **first-in, first-out** (FIFO) order
- A **stack** is a set where we extract elements in **last-in, first-out** (LIFO) order
- Both data structures can be efficiently implemented by a doubly-linked list

Upcoming

Next time…

- Finish graph representations
- Testing for bipartiteness
- Directed connectivity
- Topological sort

Reminders

- **Work on Assignment 2**
  - **Due next Friday before midnight**
- Read sections 3.4, 3.5, and 3.6