Traversals and Graphs

CSE 373 Winter 2020

Instructor: Hannah C. Tang

Teaching Assistants:

Aaron Johnston	Ethan Knutson	Nathan Lipiarski
Amanda Park	Farrell Fileas	Sam Long
Anish Velagapudi	Howard Xiao	Yifan Bai
Brian Chan	Jade Watkins	Yuma Tou
Elena Spasova	Lea Quan	

Announcements

- Homework 5: k-d trees is released and due *next Friday*
  - This is the first of our "hard" homeworks
  - Suggestion: pretend it's due Tuesday so you don't panic while prepping for midterm. Start early!
  - Hint: start with a version that doesn't prune; then implement a version that chooses good/bad sides; then finally a pruning version

- Midterm is *also* next Friday
  - If your student number ends in an odd number, go to KNE 2**1**0
  - If your student ends in an even number, go to KNE 2**2**0
  - We've released a practice midterm
  - Review session Thursday night: 4:30-6:30 @ ARC 147

Feedback from the Reading Quiz

- The reading didn't mention weighted/unweighted graphs
- I'm still confused about pre-, in- and post-order graphs
- It's interesting how we can finally use a queue to implement BFS
  - Why does BFS matter? It's also really confusing

Lecture Outline

- **Tree Traversals**

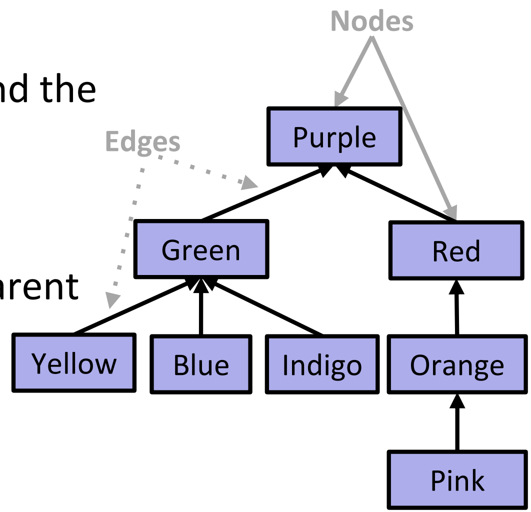
maybe on midterm

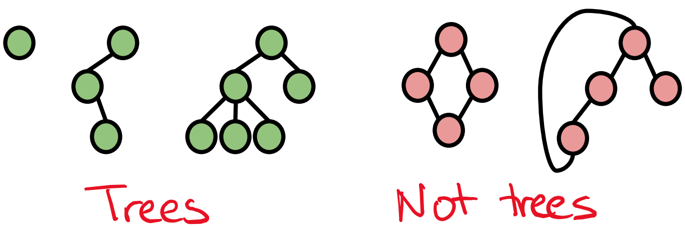
definitely NOT on midterm (but maybe the final?)

- Introduction to Graphs
  - Definitions
  - Graph Problems
- Graph Traversals: DFS

Review: The Tree Data Structure

- A **Tree** is a collection of nodes; each node has <= 1 parent and >= 0 children
  - **Root node**: the "top" of the tree and the only node with no parent
  - **Leaf node**: a node with no children
  - **Edge**: the connection between a parent and child
  - There is exactly one path between any pair of nodes

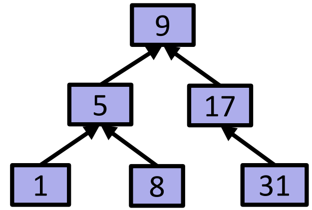


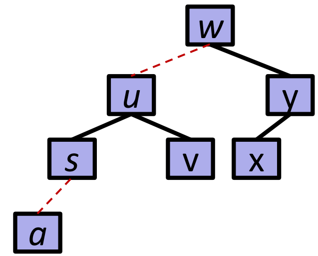



```
class Node<Value> {
  Value v;
  List<Node> children;
}
```

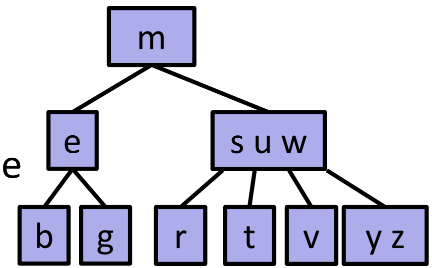
Review: Trees We've Seen

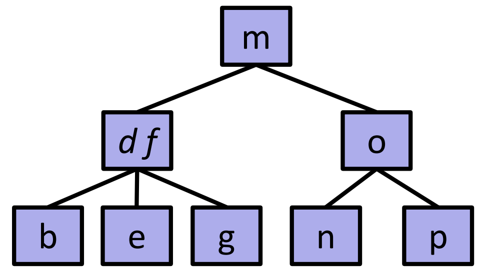
- Binary Search Trees
  - And one of its balanced variants: the Left-Leaning Red-Black Tree



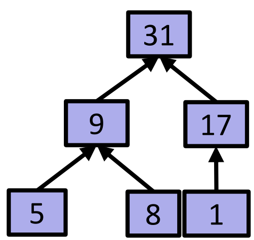


- B-Trees
  - Specifically, a 2-3 Tree



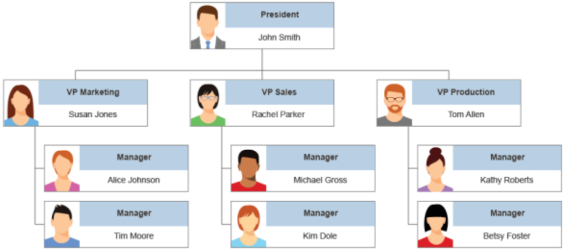


- Binary Heaps

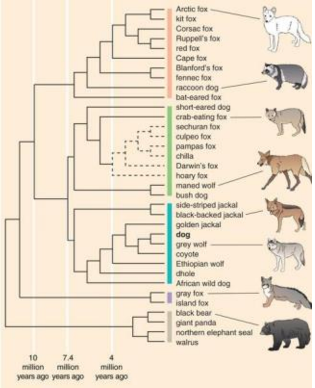


Tree Applications

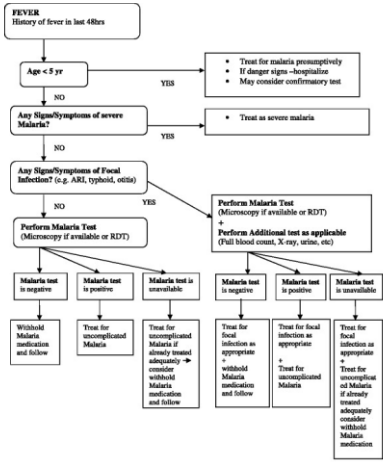
- Trees are a more general concept
  - Organization charts



  - Family lineages* including phylogenetic trees



  - MOH Training Manual for Management of Malaria.

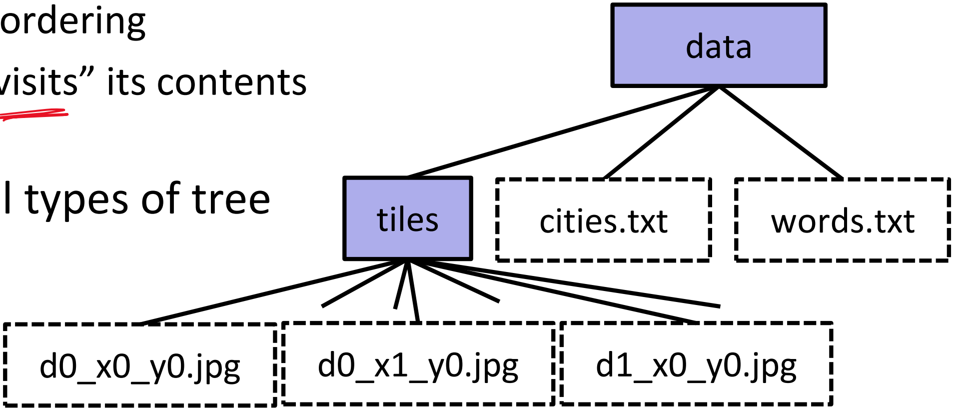


Source: MOH (2009) Training Manual for the Management of Malaria at Health Facilities in Ghana

*: *Not all family lineages are trees!*

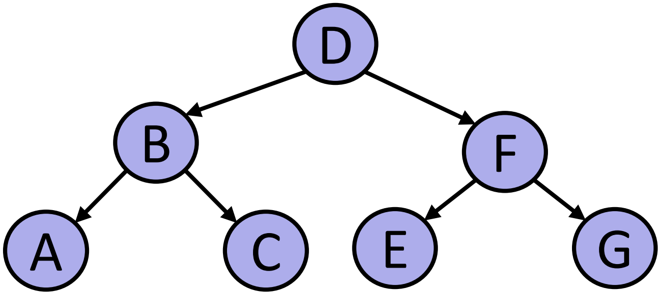
Tree Traversals

- Thus far, we've talked about *searching* a tree. Let's back up and talk about *traversing* a tree
- A traversal:
  - Iterates over every node in a tree in some defined ordering
  - "Processes" or "visits" its contents
- There are several types of tree traversals



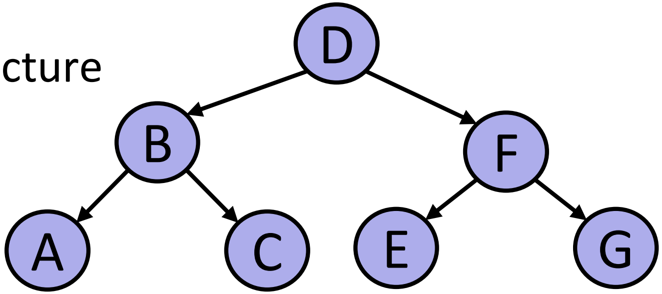
Tree Traversal Types

- Level Order Traversal aka Breadth-First Traversal
- Depth-First Traversal
  - Pre-order Traversal
  - In-order Traversal
  - Post-order Traversal



Level-Order / Breadth-First Traversal

- Traverse and visit top-to-bottom, left-to-right
  - Like reading in English
- Looks like how we converted our binary heap (ie, a complete tree) to its array representation
- Needs a supporting data structure to implement
  - See next lecture!



Depth-First Traversal

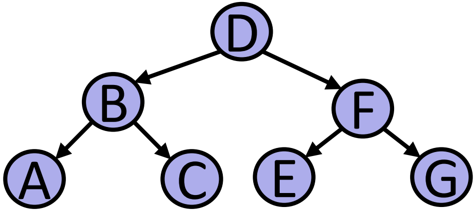
- Basic idea: traverse "deep nodes" (eg, A) before shallow ones (eg, F)

- Remember that *traversing* a node is different than *visiting/processing* a node

Depth-First: Pre-Order

```
preOrder(BSTNode x) {
  if (x == null)
    return;
  process(x.key)
  preOrder(x.left)
  preOrder(x.right)
}
```

- Pre-order "visits" the node before traversing its children
  - DBACFEG



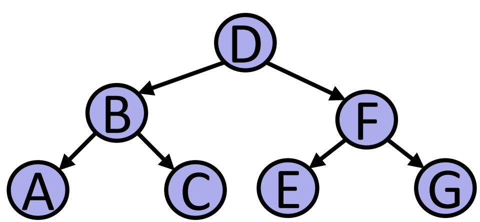
Depth-First: In-Order

- Pre-order "visits" the node before traversing its children
  - DBACFEG

- In-order traverses the left child, visits the node, then traverses the right child
  - ABCDEF

```
preOrder(BSTNode x) {
  if (x == null)
    return;
  process(x.key)
  preOrder(x.left)
  preOrder(x.right)
}
```

```
inOrder(BSTNode x) {
  if (x == null)
    return;
  inOrder(x.left)
  process(x.key)
  inOrder(x.right)
}
```



Depth-First: Post-Order

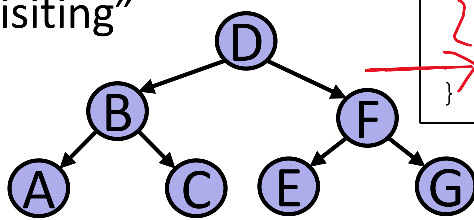
where is the root in all 3 traversals?

- Pre-order "visits" the node before traversing its children
  - DBACFEG
- In-order traverses the left child, "visits" the node, then traverses the right child
  - ABCDEF
- Post-order traverses its children before "visiting" the node
  - ACBEGFD

```
preOrder(BSTNode x) {
  if (x == null)
    return;
  process(x.key)
  preOrder(x.left)
  preOrder(x.right)
}
```

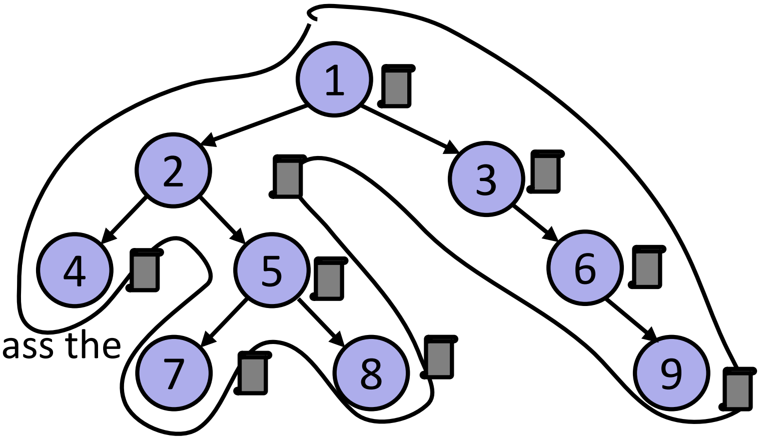
```
inOrder(BSTNode x) {
  if (x == null)
    return;
  inOrder(x.left)
  process(x.key)
  inOrder(x.right)
}
```

```
postOrder(BSTNode x) {
  if (x == null)
    return;
  postOrder(x.left)
  postOrder(x.right)
  process(x.key)
}
```



A Useful Visual Trick for Depth-First Traversals

- (Useful for humans, not algorithms)
- Trace a path around the graph, from the top going counter-clockwise
  - Pre-order: “Visit” when you pass the LEFT side of a node
  - In-order: “Visit” when you pass the BOTTOM of a node
  - Post-order: “Visit” when you pass the RIGHT side of a node.

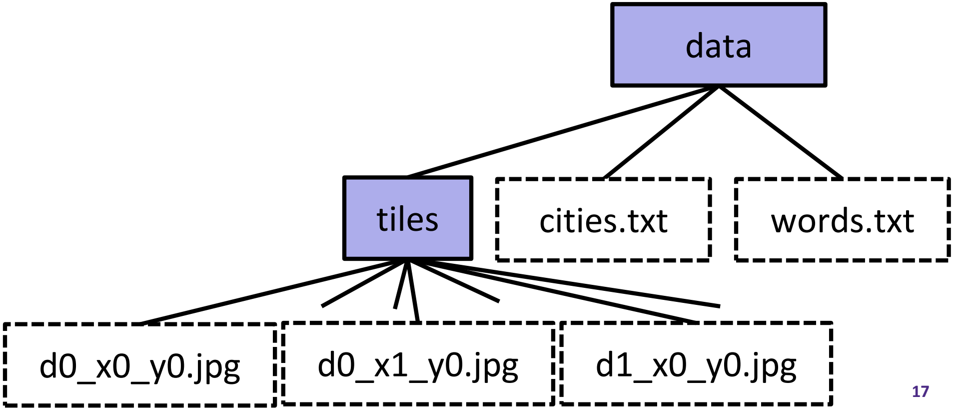


Example: post-order
4 7 8 5 2 9 6 3 1

Traversal Applications (1 of 2)

- Pre-order Traversal for printing directory listing

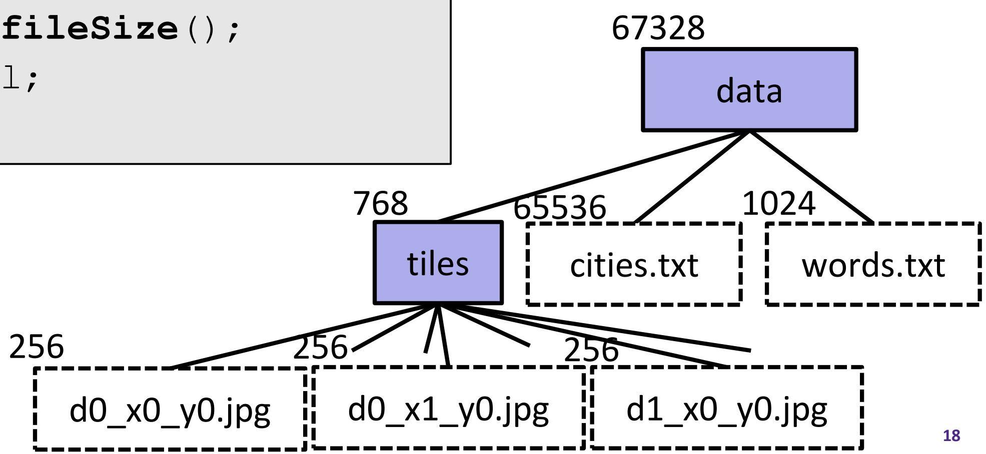
```
data/
  tiles/
    d0_x0_y0.jpg
    d0_x1_y0.jpg
    d1_x0_y0.jpg
  cities.txt
  words.txt
```



Traversal Applications (2 of 2)

- Post-order Traversal for calculating directory size

```
postOrder(BSTNode x) {
  if (x == null)
   return 0;
  int total = 0;
  for (BSTNode c : x.children())
    total += postOrder(c)
  total += x.fileSize();
  return total;
}
```



Lecture Outline

- Tree Traversals
- **Introduction to Graphs**
  - **Definitions**
  - Graph Problems
- Graph Traversals: DFS

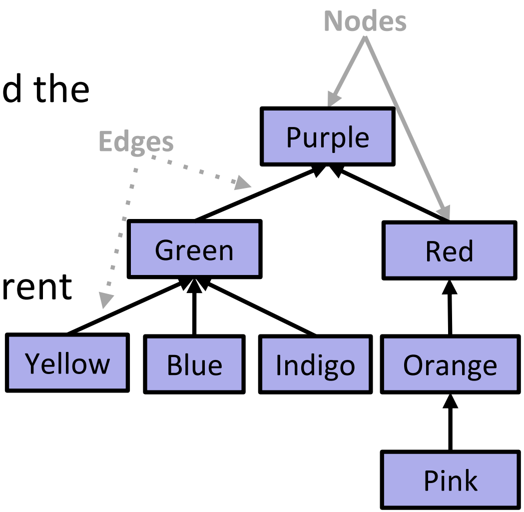
Trees are Hierarchical!

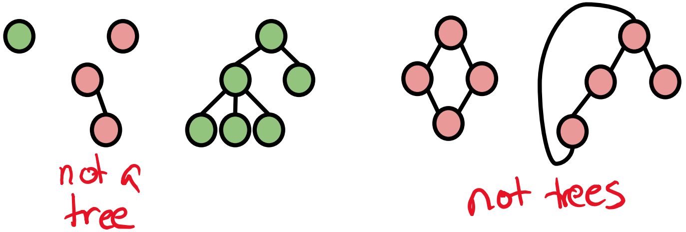
- Trees are fantastic for representing strict hierarchical relationships
  - Not every relationship is hierarchical
  - Eg: (Proposed) Light rail map

- This is not a tree: contains cycles!

Review (AGAIN?!?!): The Tree Data Structure

- A **Tree** is a collection of nodes; each node has <= 1 parent and >= 0 children
  - **Root node**: the "top" of the tree and the only node with no parent
  - **Leaf node**: a node with no children
  - **Edge**: the connection between a parent and child
  - There is exactly one path between any pair of nodes

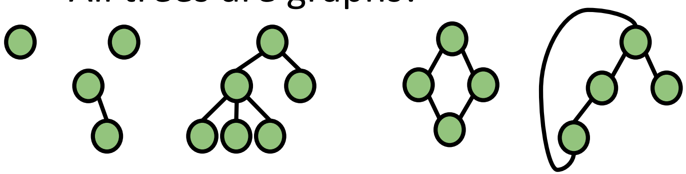




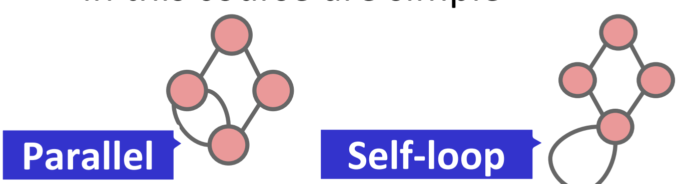

```
class Node<Value> {
  Value v;
  List<Node> children;
}
```

The Graph Data Structure

- A **Graph** is a collection of nodes, and zero or more edges connecting two nodes
  - All trees are graphs!



- A **Simple Graph** has no **self-loops** or **parallel edges**
  - In a simple graph, E is $O(V^2)$
  - Unless otherwise stated, all graphs in this course are simple



Graph Terminology (1 of 2)

- **Graph**:
  - Set of **vertices** aka **nodes**
  - Set of **edges**: pairs of vertices
  - Vertices with an edge between them are **adjacent**
  - Vertices or edges may have optional **labels**
    - Numeric edge labels are sometimes called **weights**

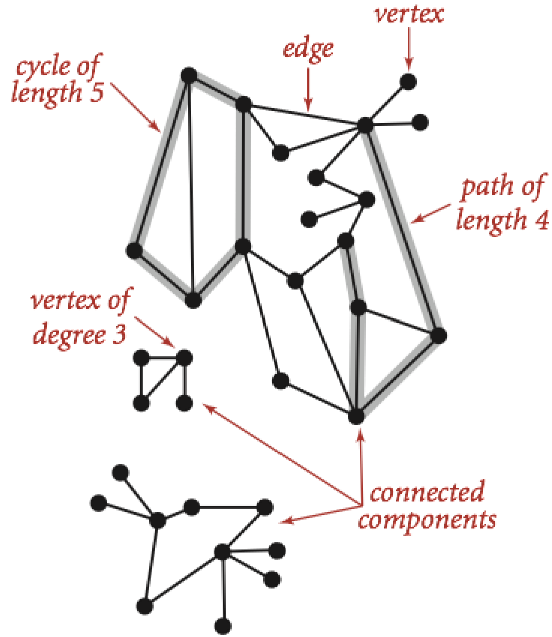


Figure from Algorithms 4th Edition

Graph Terminology (2 of 2)

- Two vertices are **connected** if there is a path between them
  - If all the vertices are connected, we say the graph is **connected**
  - The number of edges leaving a vertex is its **degree**

- A **path** is a sequence of vertices connected by edges
  - A **simple path** is a path without repeated vertices
  - A **cycle** is a path whose first and last edges are the same
    - A graph with a cycle is **cyclic**

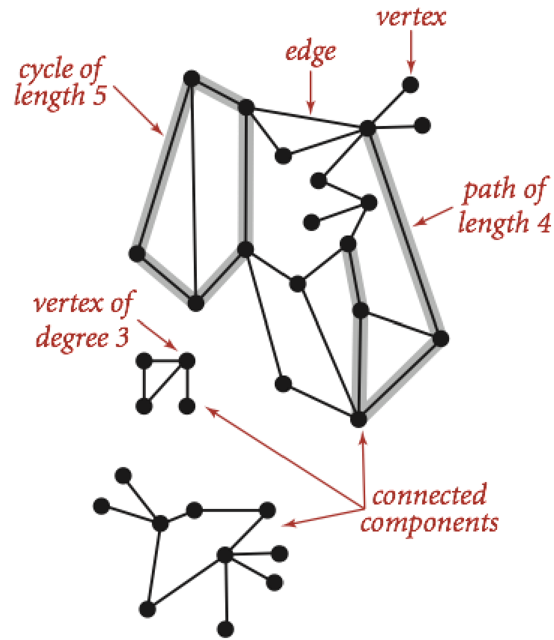
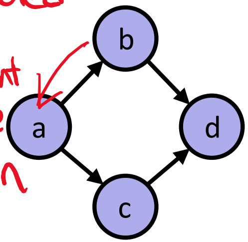
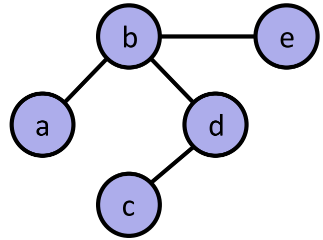
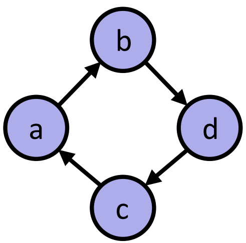
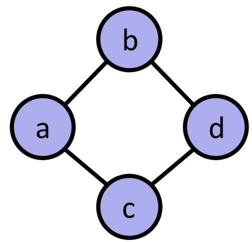


Figure from Algorithms 4th Edition

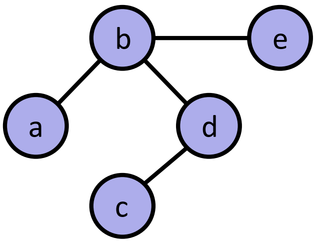
Directed vs Undirected; Acyclic vs Cyclic

Not a parallel edge! In a directed graph, parallel edges must point in the same direction

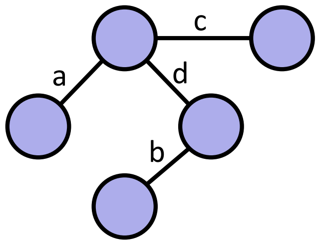
	Directed	Undirected
Acyclic:		
Cyclic:		

Labeled and Weighted Graphs

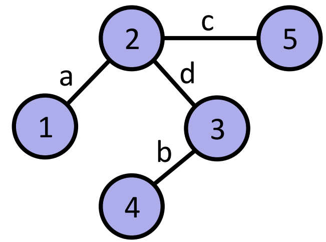
Vertex Labels



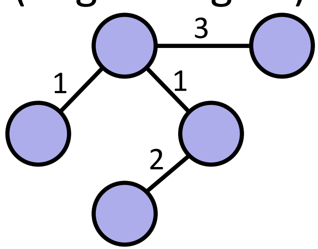
Edge Labels



Vertex and Edge Labels



Edge Labels (Edge Weights)



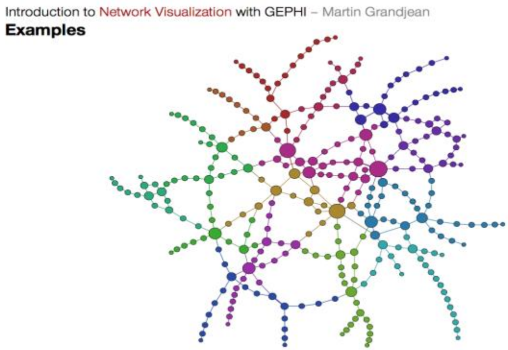
Poll Everywhere

pollev.com/uwcse373

This schematic map of the Paris Métro is a graph. It exhibits the following characteristics:

Same

A. Undirected / Connected / Cyclic / Vertex-labeled
B. Directed / Connected / Cyclic / Vertex-labeled
C. Undirected / Connected / Cyclic / Edge-labeled
D. Directed / Connected / Cyclic / Edge-labeled
E. I'm not sure …

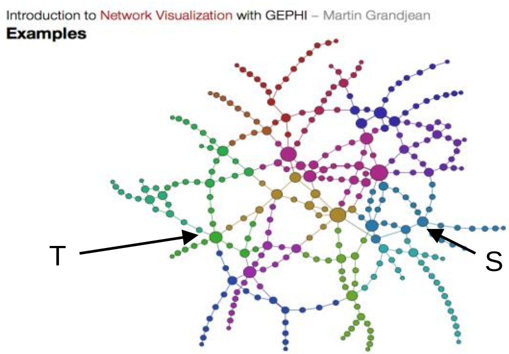


Lecture Outline

- Tree Traversals
- **Introduction to Graphs**
  - Definitions
  - **Graph Problems**
- Graph Traversals: DFS

Graph Queries

- There are lots of interesting questions we can ask about a graph:
  - What is the shortest route from S to T? What is the longest without cycles?
  - Are there cycles?
  - Is there a tour you can take that only uses each node (station) exactly once?
  - Is there a tour that uses each edge exactly once?



Graph Queries More Theoretically

- Some well known graph problems and their common names:
  - **s-t Path**. Is there a path between vertices s and t?
  - **Connectivity.** Is the graph connected?
  - **Biconnectivity.** Is there a vertex whose removal disconnects the graph?
  - **Shortest s-t Path.** What is the shortest path between vertices s and t?
  - **Cycle Detection.** Does the graph contain any cycles?
  - **Euler Tour.** Is there a cycle that uses every edge exactly once?
  - **Hamilton Tour.** Is there a cycle that uses every vertex exactly once?
  - **Planarity**. Can you draw the graph on paper with no crossing edges?
  - **Isomorphism**. Are two graphs the same graph (in disguise)?

- Often can’t tell how difficult a graph problem is without very deep consideration.

Graph Problem Difficulty

- Some well known graph problems:
  - **Euler Tour:** Is there a cycle that uses every *edge* exactly once?
  - **Hamilton Tour:** Is there a cycle that uses every *vertex* exactly once?

- Difficulty can be deceiving
  - An efficient Euler tour algorithm O(# edges) was found as early as 1873 [Link].
  - Despite decades of intense study, no efficient algorithm for a Hamilton tour exists. Best algorithms are exponential time.

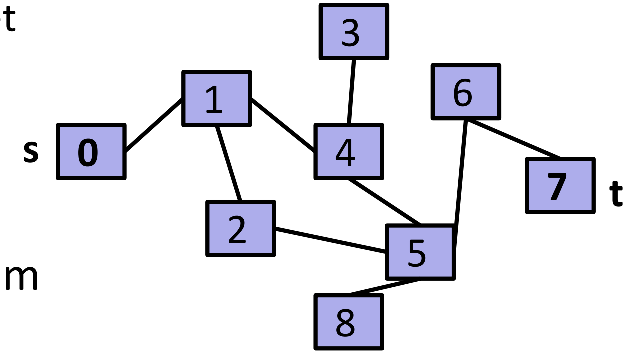
- Graph problems are among the most mathematically rich areas of CS theory

Lecture Outline

- Tree Traversals
- Introduction to Graphs
  - Definitions
  - Graph Problems
- **Graph Traversals: DFS**

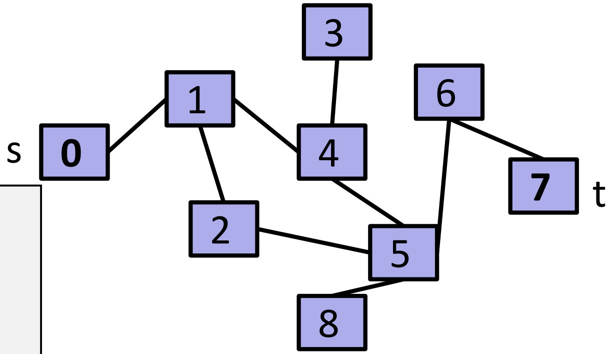
s-t Connectivity Problem

- s-t connectivity problem
  - Given source vertex **s** and a target vertex **t**, does there exist a path between **s** and **t**?



- Try to come up with an algorithm for connected(**s**, **t**)

s-t Connectivity Problem: Proposed Solution

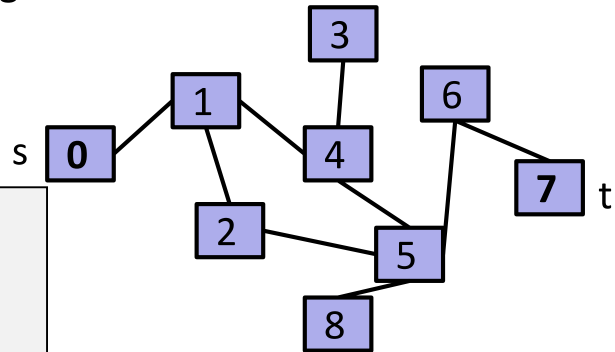



```
connected(Node s, Node t) {
  if (s == t) {
    return true;
  } else {
    for (Node n : s.neighbors) {
      if (connected(n, t)) {
        return true;
      }
    }
    return false;
  }
}
```

Poll Everywhere

pollev.com/uwcse373

- What is wrong with the proposed algorithm?

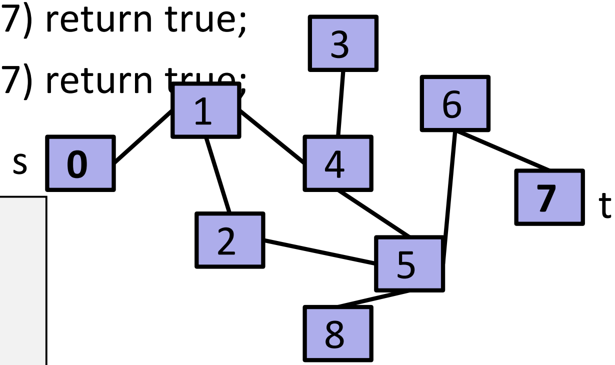



```
connected(Node s, Node t) {
  if (s == t) {
    return true;
  } else {
    for (Node n : s.neighbors) {
      if (connected(n, t)) {
        return true;
      }
    }
    return false;
  }
}
```

s-t Connectivity Problem

- What is wrong with the proposed algorithm?
  - Does 0 == 7? No; if(connected(1, 7) return true;
  - Does 1 == 7? No; if(connected(0, 7) return true;
  - Does 0 == 7?

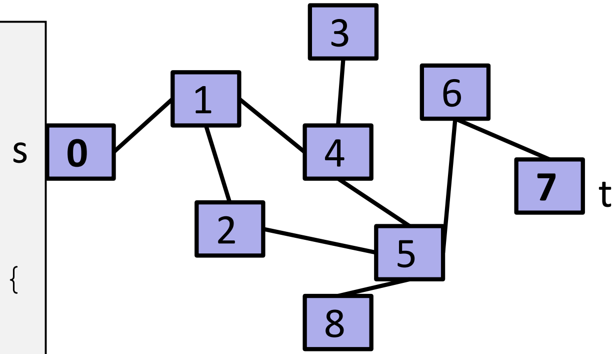
```
connected(Node s, Node t) {
  if (s == t) {
    return true;
  } else {
    for (Node n : s.neighbors) {
      if (connected(n, t)) {
        return true;
      }
    }
    return false;
  }
}
```



s-t Connectivity Problem: Depth-First Search

- Mark each node as visited!

```
connected(Node s, Node t) {
  if (s == t) {
    return true;
  } else {
    s.visited = true;
    for (Node n : s.neighbors) {
      if (n.visited) {
        continue;
      }
      if (connected(n, t)) {
        return true;
      }
    }
    return false;
  }
}
```



Is this a pre-order traversal or a post-order traversal?

Do in-order traversals exist for graphs?

s-t Connectivity Problem: Depth-First Search

- Demo: https://docs.google.com/presentation/d/1OHRI7Q_f8hlwjRJc8NPBUc1cMu5KhINH1xGXWDfs_dA/present?ueb=true&slide=id.g76e0dad85_2_380

- Is this a pre-order traversal or a post-order traversal?
  - Do in-order traversals exist for graphs?

tl;dr

- Traversals are an order in which you visit/process vertices
- Trees have level-order traversals and 3 depth-first traversals
- Graphs are a more general idea than a tree
  - Key terms: Directed/Undirected, Cyclic/Acylic, Path, Cycle
  - Traversals are a common tool for solving almost all graph problems
    - DFS pre-order, DFS post-order, BFS (next lecture!)