LEC 14

CSE 373

Graphs

BEFORE WE START

Instructor Hunter Schafer

TAs		
	Ken Aragon	Paul Pham
	Khushi Chaudhari	Mitchell Szeto
	Joyce Elauria	Batina Shikhalieva
	Santino Iannone	Ryan Siu
	Leona Kazi	Elena Spasova
	Nathan Lipiarski	Alex Teng
	Sam Long	Blarry Wang
	Amanda Park	Aileen Zeng

Announcements

- P2 late cutoff tonight at 11:59pm
- P3 due just under weeks on Friday, 11/13
  - Start early!
  - Remember that changePriority and contains aren't efficient on a heap alone – you should **use an extra data structure**!
  - Recommendation: just get it working first, then analyze where inefficiencies are – what data structure could help?
- EX3 published this Friday, 11/06
  - Focusing on post-Exam I content, especially this week

LEC 12 PQs & Heaps

LEC 13 Heaps II, Interviews

P3 Heap

LEC 14 Graphs

LEC 15 BFS, DFS, SP

LEC 16 Dijkstra's

EX3 BFS/DFS/Dijkstra's

Learning Objectives

After this lecture, you should be able to...

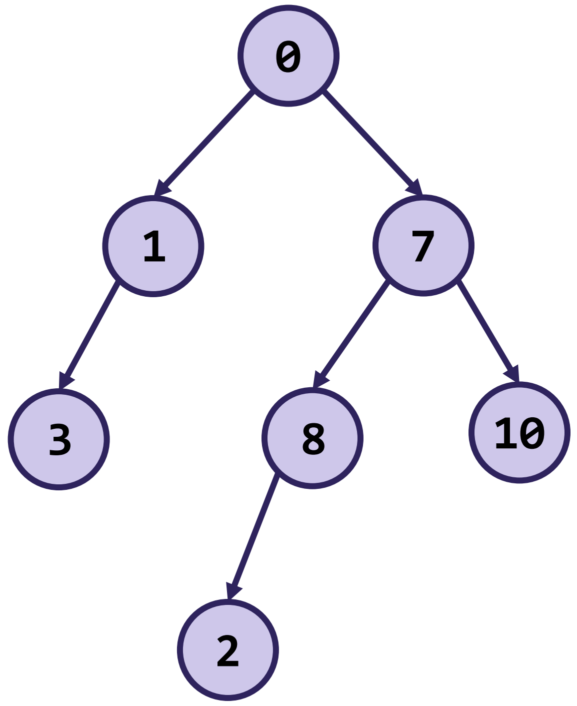
1. Categorize graph data structures based on which properties they exhibit
2. Select which properties of a graph would be most appropriate to model a scenario (e.g. Directed/Undirected, Cyclic/Acyclic, etc.)
3. Compare the runtimes of Adjacency Matrix and Adjacency List graph implementations, and select the most appropriate one for a particular problem
4. Describe the high-level algorithm for solving the s-t Connectivity Problem, and be prepared to expand on it going forward

Lecture Outline

- Graphs
  - Definitions
  - Choosing Graph Types
- Graph Implementations
- s-t Connectivity Problem

Review Trees

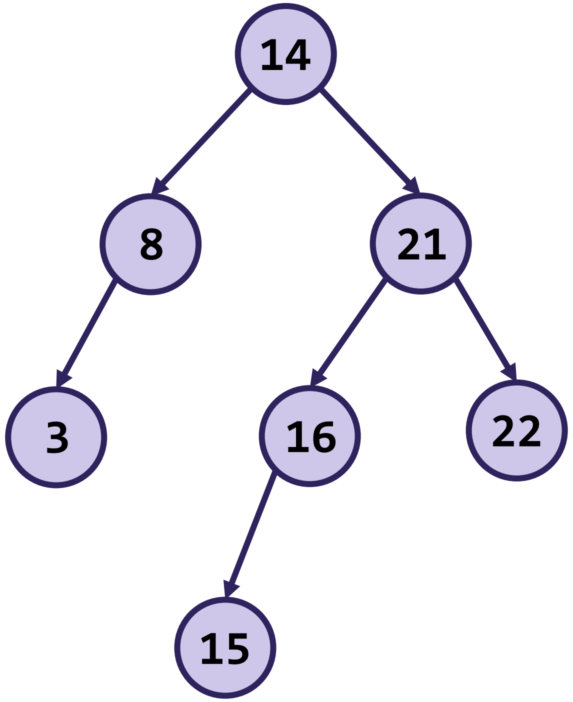
- A **tree** is a collection of nodes where each node has at most 1 parent and at least 0 children
  - A **binary tree** is a tree where each node has at most 2 children



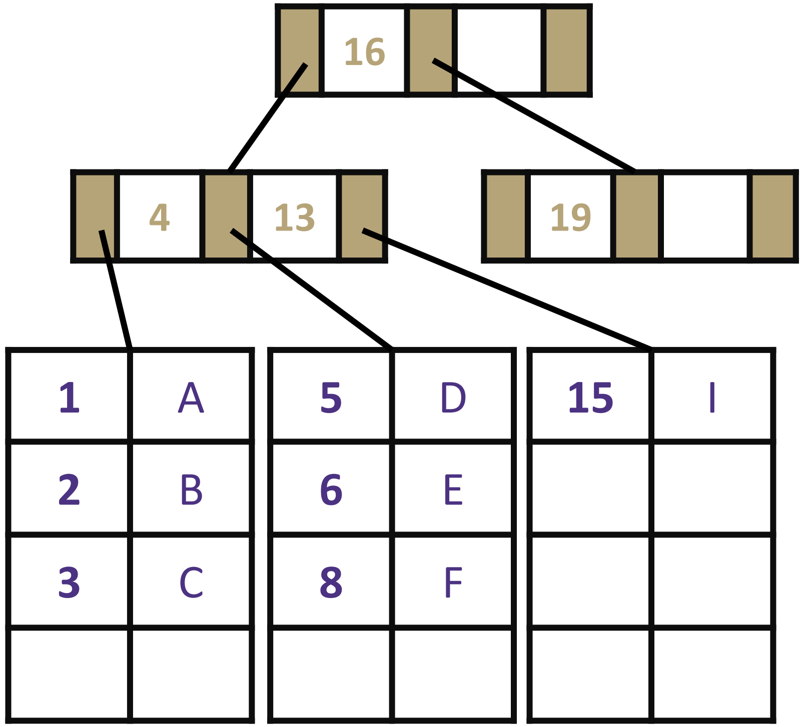
- **Root node:** the single node with no parent, "top" of the tree
- **Leaf node:** a node with no children
- **Subtree:** a node and all its descendants
- **Edge:** connection between parent and a child

Review Trees We've Seen So Far

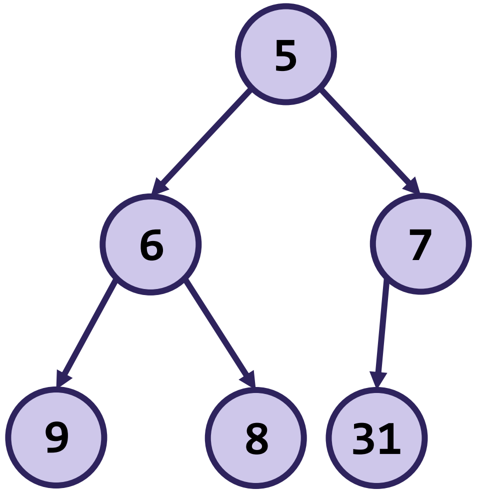
Binary Search Trees
- And variant: **AVL Trees**



B+ Trees



Binary Min-Heaps



Inter-data Relationships

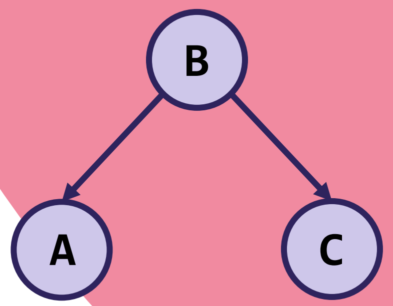
Arrays

- Elements only store pure data, no connection info
- Only relationship between data is order

0	1	2
A	B	C

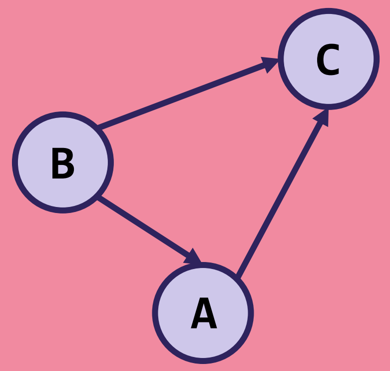
Trees

- Elements store data and connection info
- Directional relationships between nodes; limited connections



Graphs

- Elements AND connections can store data
- Relationships dictate structure; huge freedom with connections



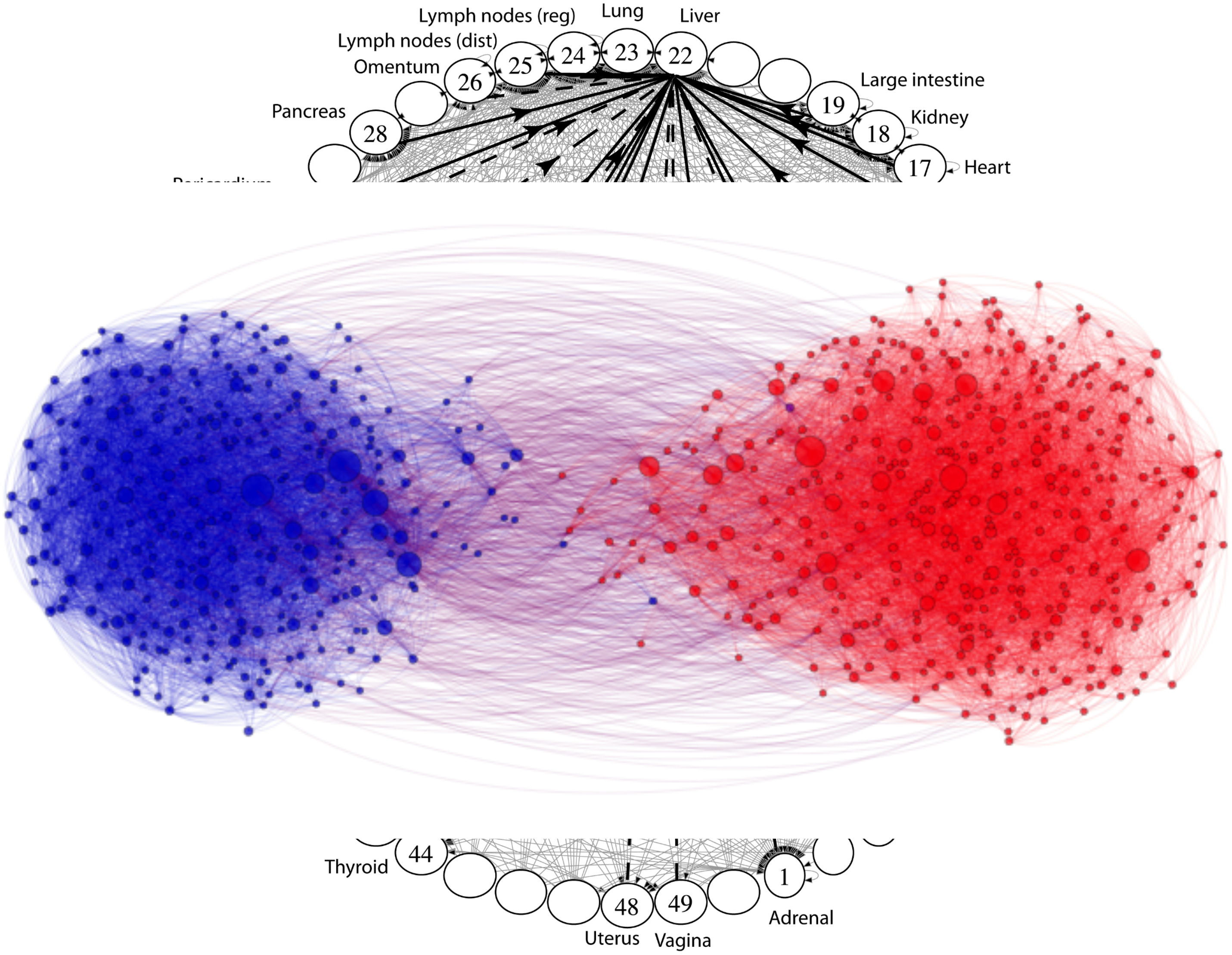
Everything is Graphs

- **Everything** is graphs.
- Most things we've studied this quarter can be represented by graphs.
  - BSTs are graphs
  - Linked lists? Graphs.
  - Heaps? Also can be represented as graphs.
  - Those trees we drew in the tree method? Graphs.
- But it's not just data structures that we've discussed…
  - Google Maps database? Graph.
  - Facebook? They have a "graph search" team. Because it's a graph
  - Gitlab's history of a repository? Graph.
  - Those pictures of prerequisites in your program? Graphs.
  - Family tree? That's a graph

Applications

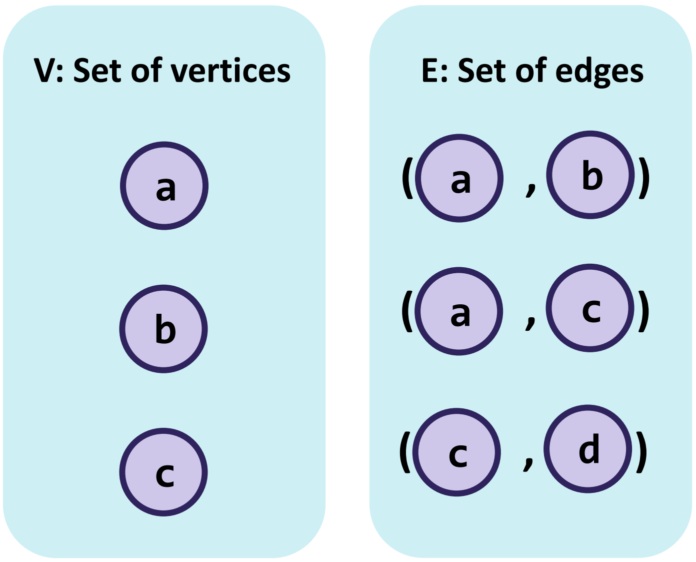
- Physical Maps
  - Airline maps
    - Vertices are airports, edges are flight paths
  - Traffic
    - Vertices are addresses, edges are streets
- Relationships
  - Social media graphs
    - Vertices are accounts, edges are follower relatior
  - Code bases
    - Vertices are classes, edges are usage
- Influence
  - Biology
    - Vertices are cancer cell destinations, edges are m
- Related topics
  - Web Page Ranking
    - Vertices are web pages, edges are hyperlinks
  - Wikipedia
    - Vertices are articles, edges are links

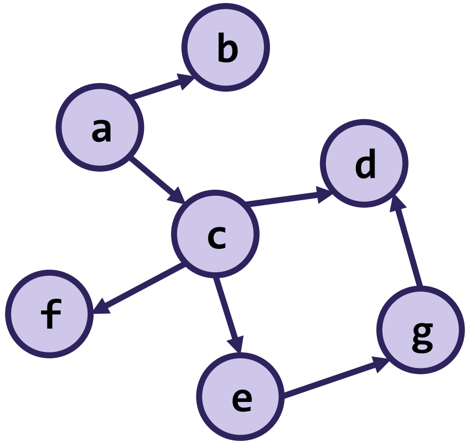
So many more:
www.allthingsgraphed.com

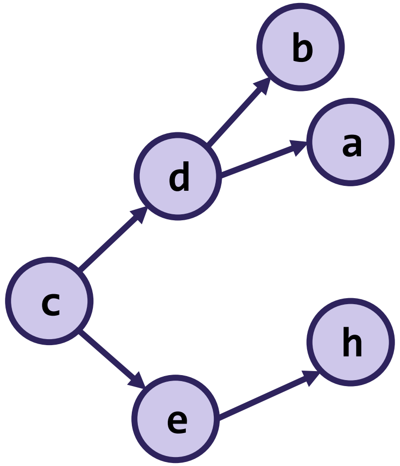


Graphs

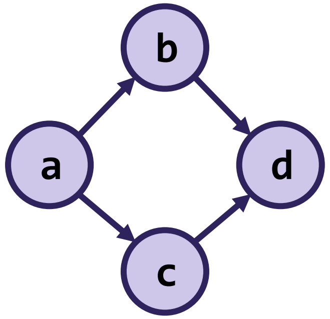
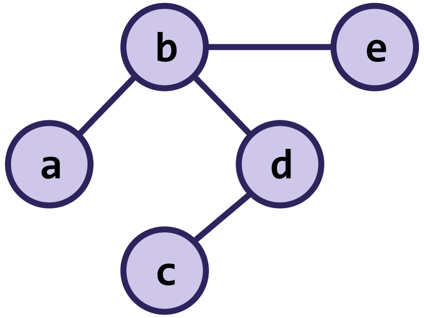
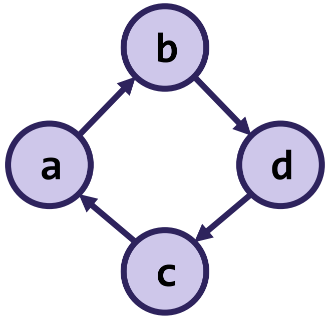
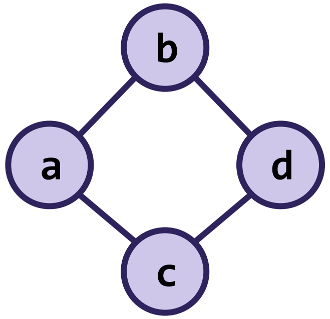
- A **Graph** consists of two sets, V and E:
  - V: Set of **vertices** (aka **nodes**)
  - E: Set of **edges** (pairs of vertices)
  - |V|: Size of V (also called n)
  - |E|: Size of E (also called m)





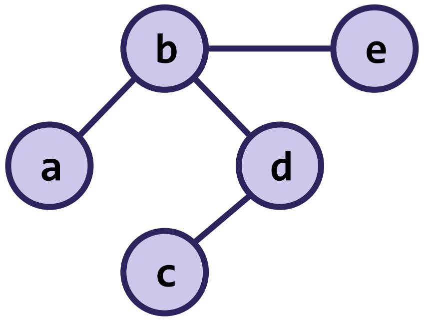


Directed vs Undirected; Acyclic vs Cyclic

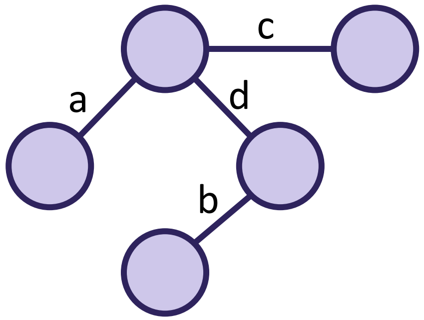
	Directed:	Undirected:
Acyclic:		
Cyclic:		

Labeled and Weighted Graphs

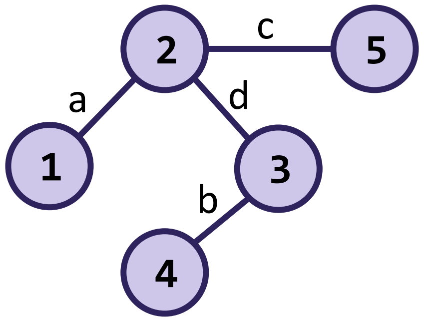
Vertex Labels



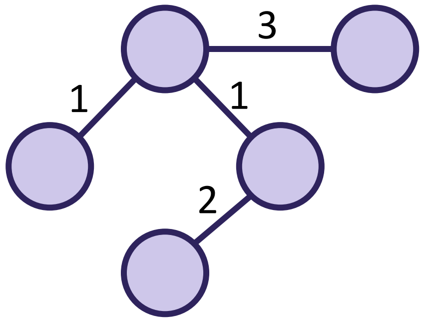
Edge Labels



Vertex & Edge Labels

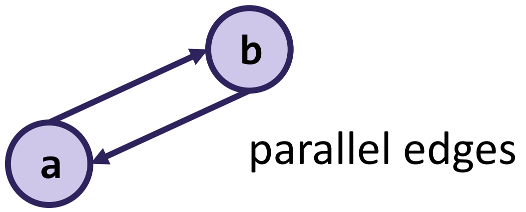


Numeric Edge Labels
(**Edge Weights**)



More Graph Terminology

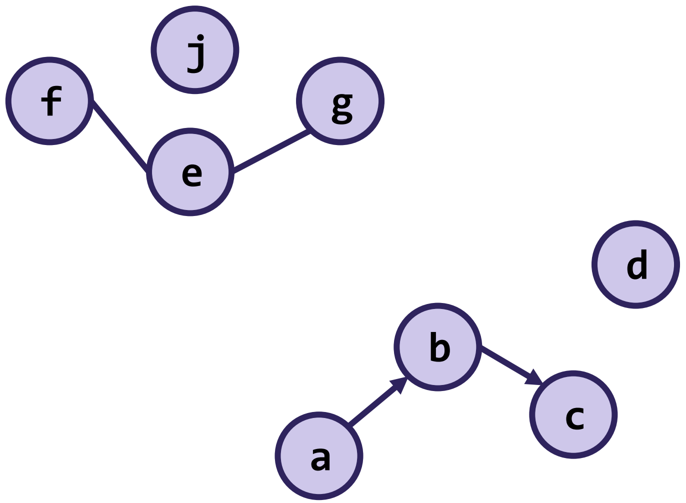
- A **Simple Graph** has no **self-loops** or **parallel edges**
  - In a simple graph, |E| is O(|V|^2)
  - Unless otherwise stated, all graphs in this course are simple



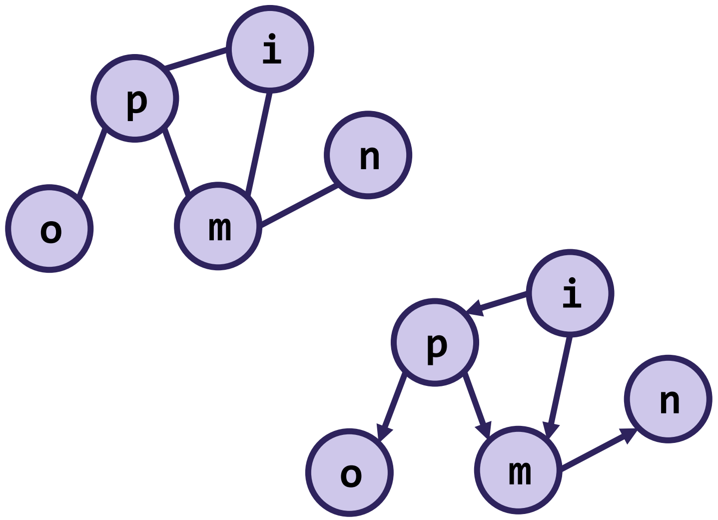
- Vertices with an edge between them are **adjacent**
  - Vertices or edges may have optional **labels**
    - Numeric edge labels are sometimes called **weights**

More More Graph Terminology

- 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 vertices are the same
    - A graph with a cycle is **cyclic**



Lecture Outline

- **Graphs**
  - Definitions
  - **Choosing Graph Types**
- Graph Implementations
- s-t Connectivity Problem

Some examples

- For each of the following: what should you choose for vertices and edges? Directed?
- **Webpages on the Internet**
- **Ways to walk between UW buildings**
- **Course Prerequisites**

Some examples

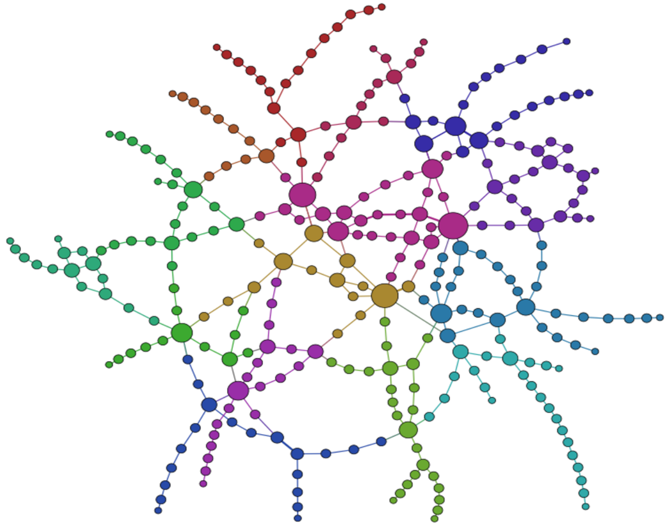
- For each of the following: what should you choose for vertices and edges? Directed?
- **Webpages on the Internet**
  - **Vertices**: webpages. **Edges** from a to b if a has a hyperlink to b.
  - Directed, since hyperlinks go in one direction
- **Ways to walk between UW buildings**
  - **Vertices**: buildings. **Edges**: A street name or walkway that connects 2 buildings
  - Undirected, since each route can be walked both ways
- **Course Prerequisites**
  - **Vertices**: courses. **Edge**: from a to b if a is a prereq for b.
  - Directed, since one course comes before the other

Itempool

Pause Video when Prompted

This schematic map of the Paris Métro is a graph. Which of the following characteristics make sense here?

Introduction to Network Visualization with GEPHI – Martin Grandjean
Examples



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

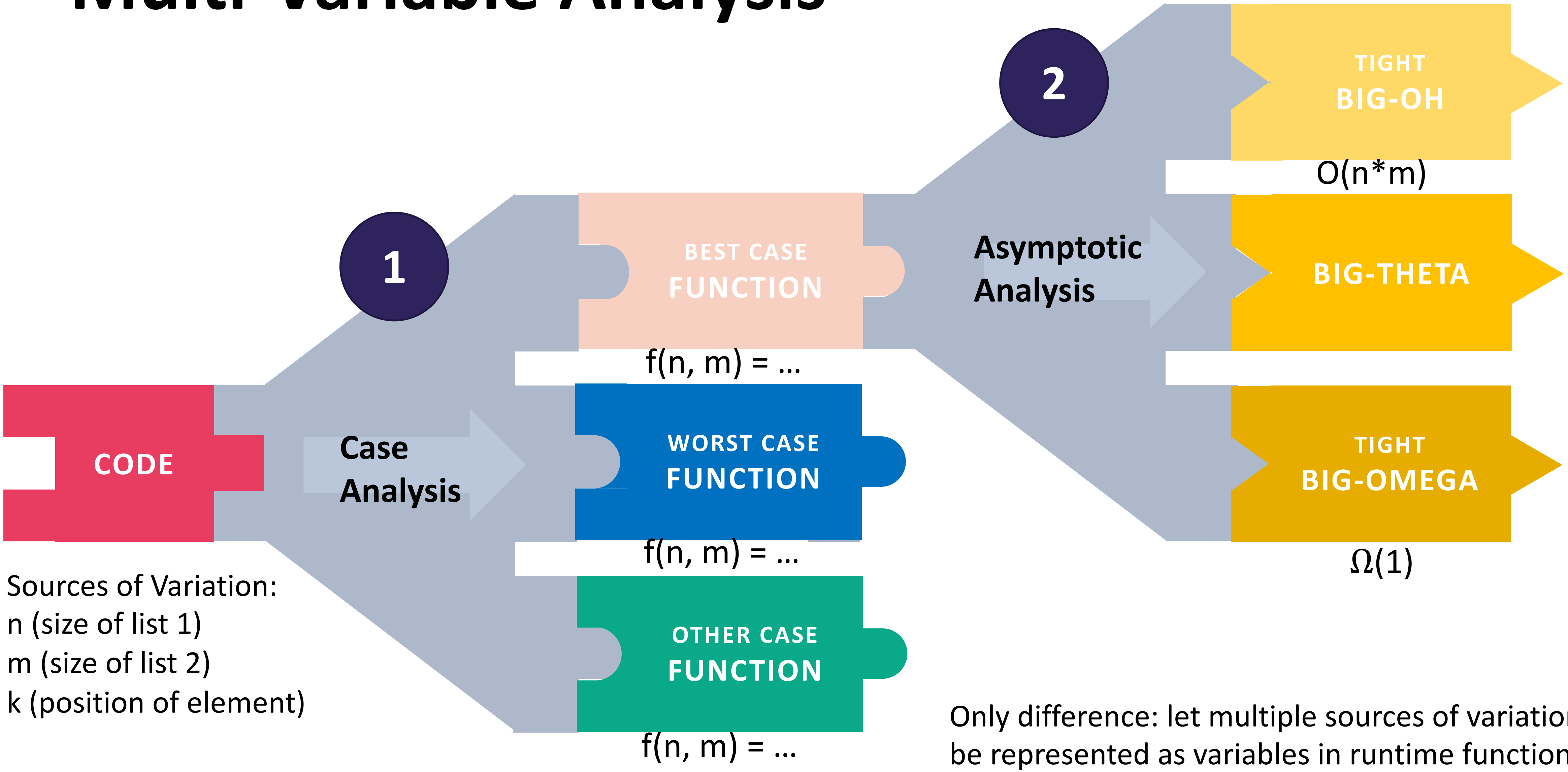
- Graphs
  - Definitions
  - Choosing Graph Types
- **Graph Implementations**
- s-t Connectivity Problem

Multi-Variable Analysis

- So far, we thought of everything as being in terms of some single argument “n” (sometimes its own parameter, other times a size)
  - But there’s no reason we can’t do reasoning in terms of multiple inputs!
- Why multi-variable?
  - Remember, algorithmic analysis is just a tool to help us understand code. Sometimes, it helps our understanding more to build a Oh/Omega/Theta bound for multiple factors, rather than handling those factors in case analysis.

- With graphs, we usually do our reasoning in terms of:
  - n (or |V|): total number of vertices (sometimes just call it V)
  - m (or |E|): total number of edges (sometimes just call it E)
  - deg(u): degree of node u (how many outgoing edges it has)

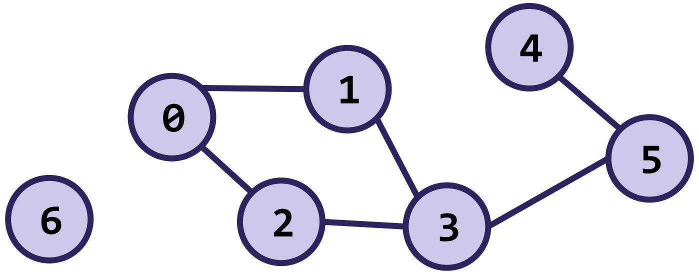
Multi-Variable Analysis



Sources of Variation:
n (size of list 1)
m (size of list 2)
k (position of element)

Only difference: let multiple sources of variation be represented as variables in runtime functions, instead of wrapping them up into cases!

Adjacency Matrix



- Create a 2D matrix that is |V| x |V|
- In an adjacency matrix, a[u][v] is 1 if there is an edge (u,v), and 0 otherwise.
- Symmetric for undirected graphs

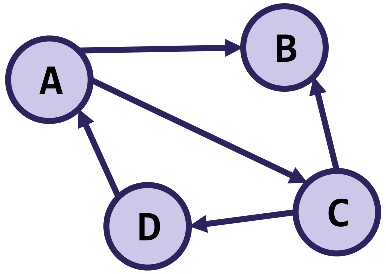
	0	1	2	3	4	5	6
0	0	1	1	0	0	0	0
1	1	0	0	1	0	0	0
2	1	0	0	1	0	0	0
3	0	1	1	0	0	1	0
4	0	0	0	0	0	1	0
5	0	0	0	1	1	0	0
6	0	0	0	0	0	0	0

Add Edge	$\Theta(1)$
Remove Edge	$\Theta(1)$
Check if edge (u, v) exists	$\Theta(1)$
Get out-neighbors of u	$\Theta(n)$
Get in-neighbors of v	$\Theta(n)$
(Space Complexity)	$\Theta(n^2)$

(|V| = n, |E| = m)

Adjacency List

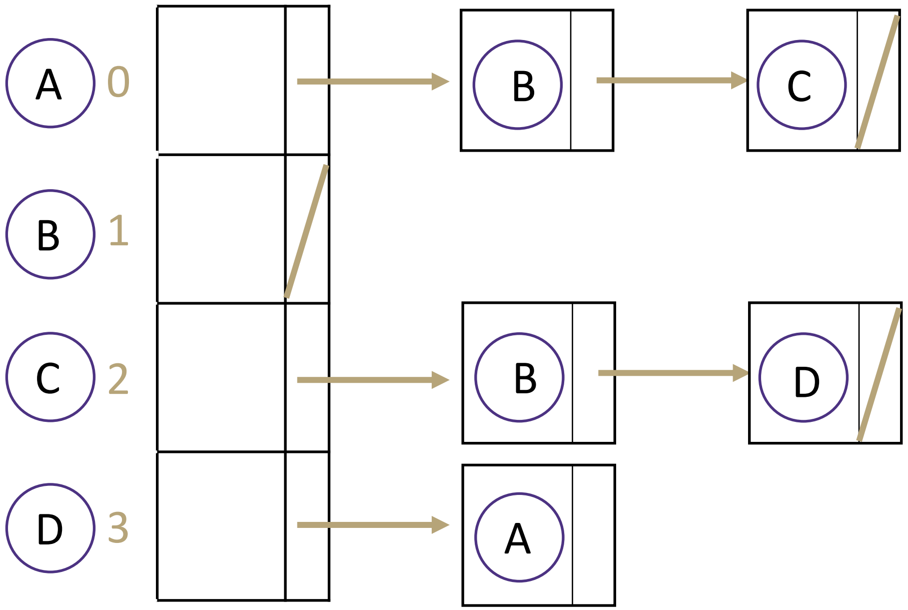
- Create a Map from V to some Collection of E
- In an adjacency list, if (u,v) ∈ E, then v is found in the collection under key u
- Since each node maps to a list of its neighbors, in undirected graph every edge will be included twice
    - In directed graph, every edge from u is in list associated with key u.



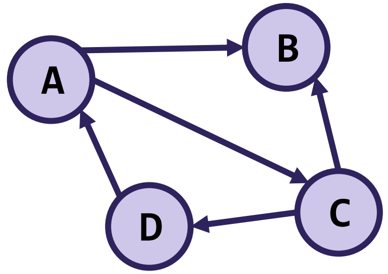
Add Edge	$\Theta(1)$
Remove Edge	$\Theta(\deg(u))$
Check if edge (u, v) exists	$\Theta(\deg(u))$
Get out-neighbors of u	$\Theta(\deg(u))$
Get in-neighbors of v	$\Theta(n+m)$
(Space Complexity)	$\Theta(n+m)$

(|V| = n, |E| = m)

Linked Lists



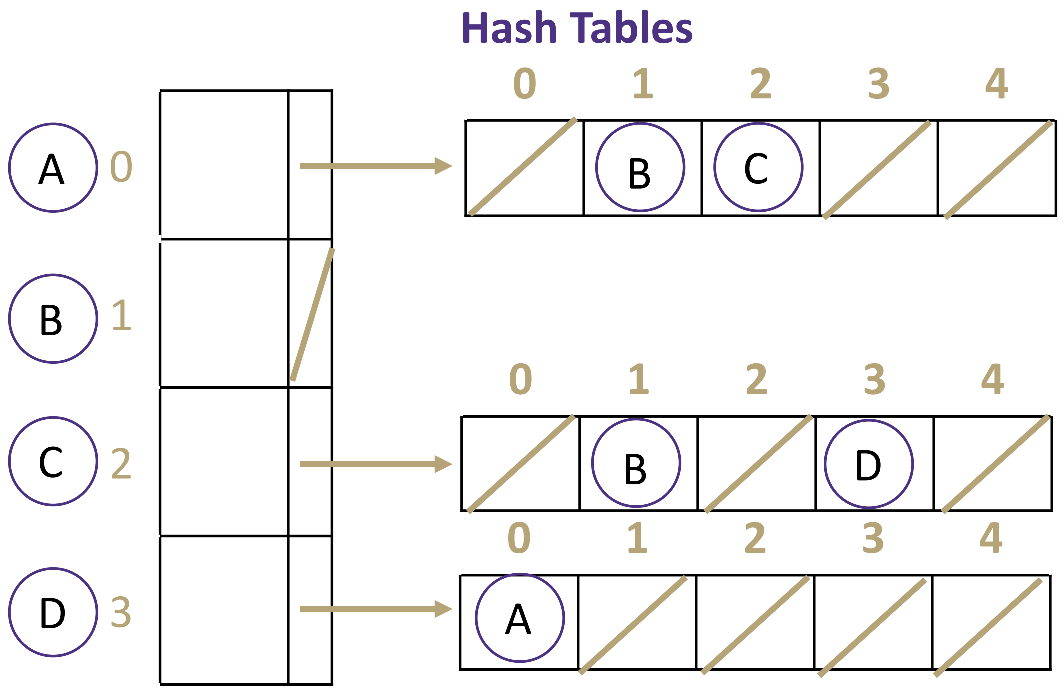
Best of Both Worlds?



- Can use Hashing as an in-between solution
- Represent the Adjacency Matrix as a `Map<Node, Map<Node, EdgeLabel>>`
- Not quite as much space as Adjacency Matrix but get the constant-time "in practice" lookup.

Add Edge	$\Theta(1)$
Remove Edge	$\Theta(1)$
Check if edge (u, v) exists	$\Theta(1)$
Get out-neighbors of u	$\Theta(\deg(u))$
Get in-neighbors of v	$\Theta(n)$
(Space Complexity)	$\Theta(n+m)$

(|V| = n, |E| = m)



Tradeoffs

- Adjacency Matrices take more space, why would you use them?
  - For **dense** graphs (where m is close to n^2), the running times will be close
  - And the constant factors can be much better for matrices than for lists.
  - Sometimes the matrix itself is useful ("spectral graph theory")
- What's the tradeoff between using linked lists and hash tables for the list of neighbors?
  - A hash table still *might* hit a worst-case
  - And the linked list might not
    - Graph algorithms often just need to iterate over all the neighbors, so you might get a better guarantee with the linked list.

373: Graph Implementations

- For this class, unless we say otherwise, we'll assume the hash tables operations **on graphs** are all $O(1)$.
  - Because you can probably control the keys.
- Unless we say otherwise, assume we're using the hash table approach.