CSE 373: Data Structures and Algorithms

Lecture 14: Introduction to Graphs

Instructor: Lilian de Greef
Quarter: Summer 2017

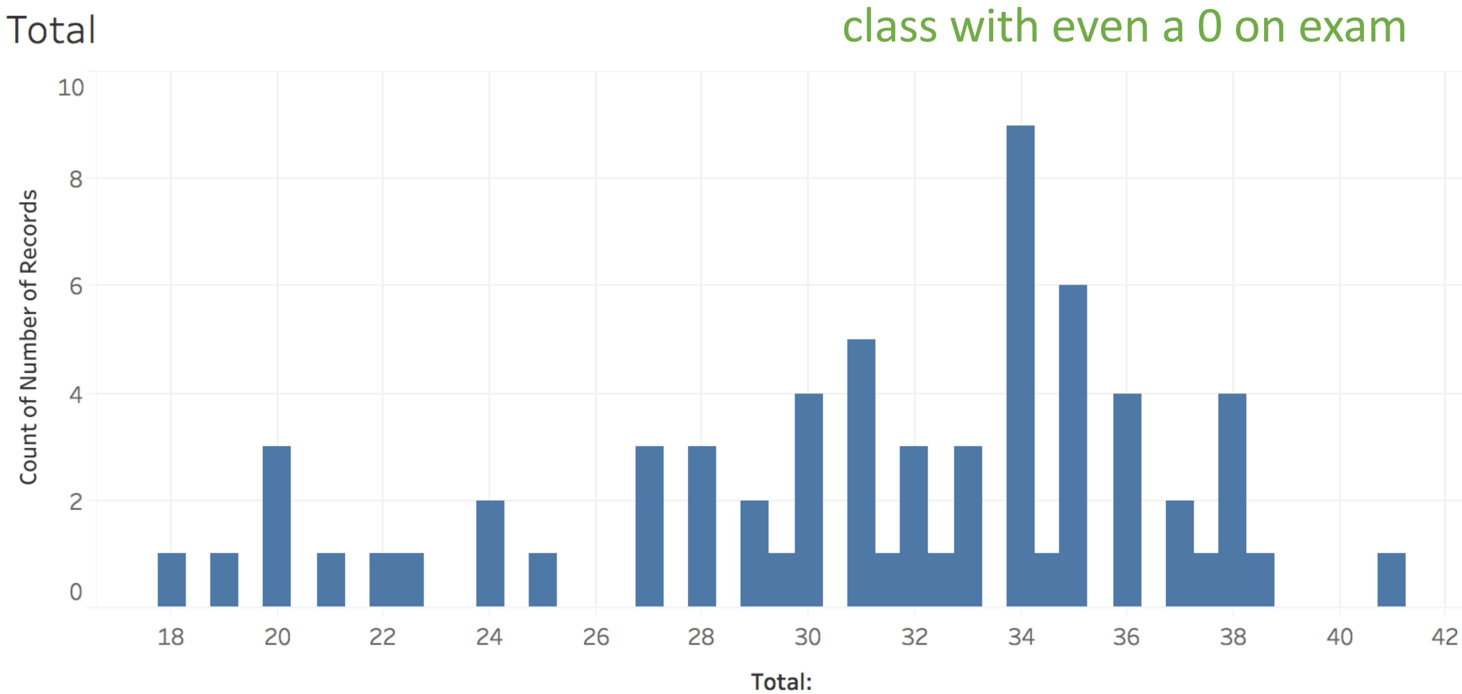
Today

- Overview of Midterm
- Introduce Graphs
  - Mathematical representation
  - Undirected & Directed Graphs
  - Self edges
  - Weights
  - Paths & Cycles
  - Connectedness
  - Trees as graphs
  - DAGs
  - Density & Sparsity

Midterm: Statistics and Distribution

Remember: it's curved

20% of grade → can pass class with even a 0 on exam



Mean 31.2 /43

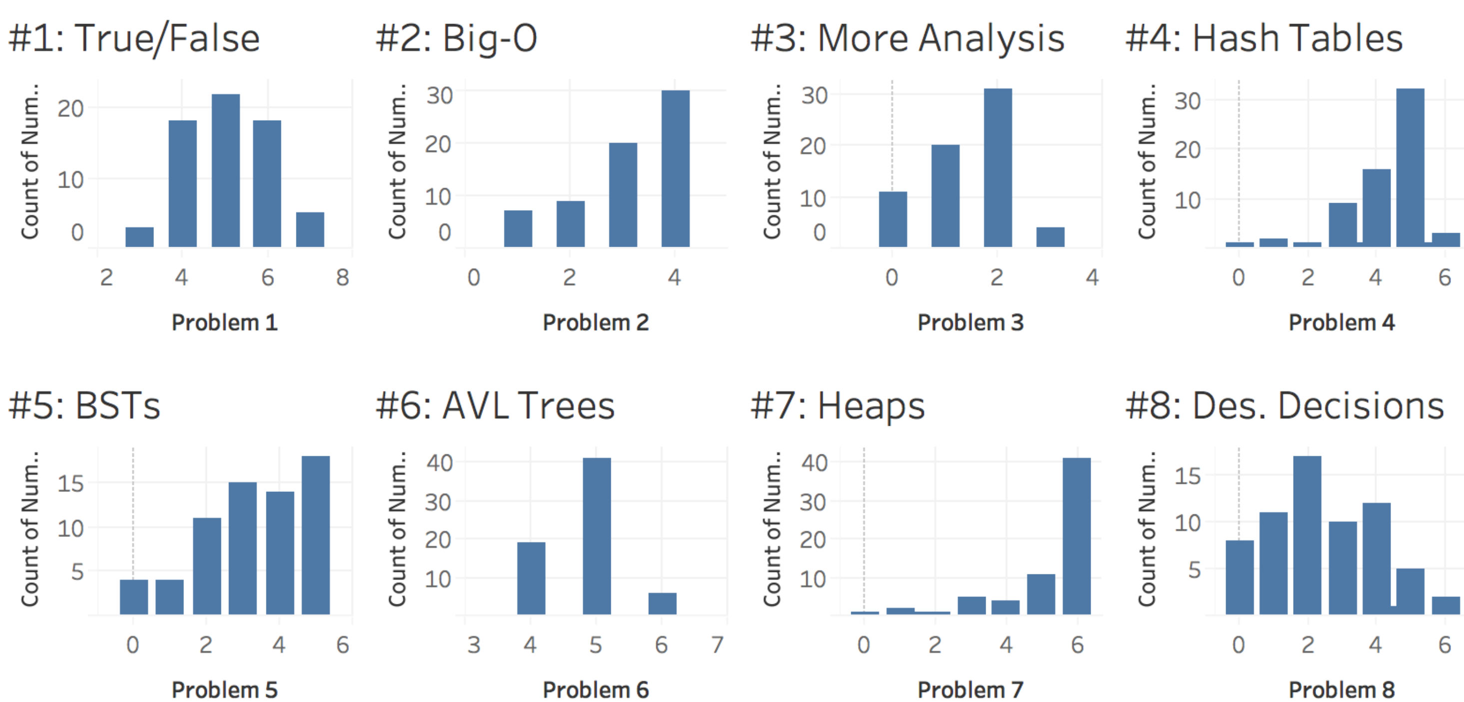
Std. dev. 5.48

Median 32.5 /43

Mode 34 /43

Max 41 /43

Midterm: Distribution by Problem



Hash Tables

There is a hash table implemented with linear probing that doubles in size every time its load factor is strictly greater than 1⁄2.

What is the worst-case condition for insert in this table?

What is the asymptotic worst-case running time to insert an item?
(let n = # items in table)

What is the amortized running time to insert an item to this table?

Hash Tables

Now we have a hash table implemented with separate chaining in which each chain stores its keys in sorted order.

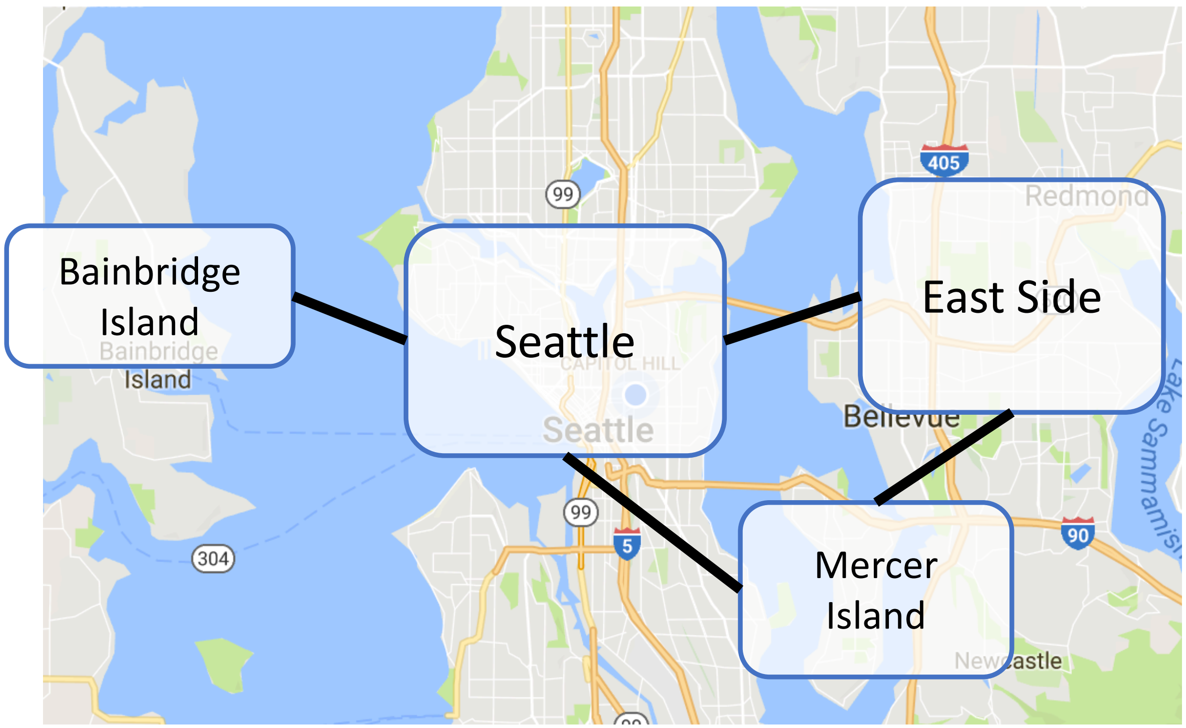
What is the worst-case condition for insert in this table?

What is the asymptotic worst-case running time to insert an item into this table?

Introducing: Graphs

Vertices, edges, and paths (oh my!)

Introductory Example



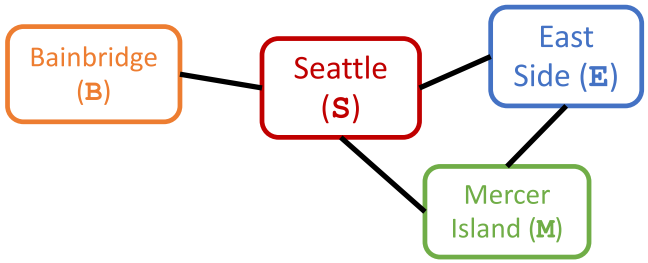
This representation is called a

In this example, locations (Seattle, Bainbridge Island, the East Side, and Mercer Island) are the

And the roads, bridges, and ferry lines are the

Graphs

- A graph is a formalism for representing relationships among items
  - Very general definition because very general concept
- A **graph** is a pair
  - A set of **vertices**, also known as
    $V = \{v_1, v_2, \ldots, v_n\}$
  - A set of **edges**
    $E = \{e_1, e_2, \ldots, e_m\}$
    - An edge "connects" the vertices
    - Each edge e_i is a pair of vertices

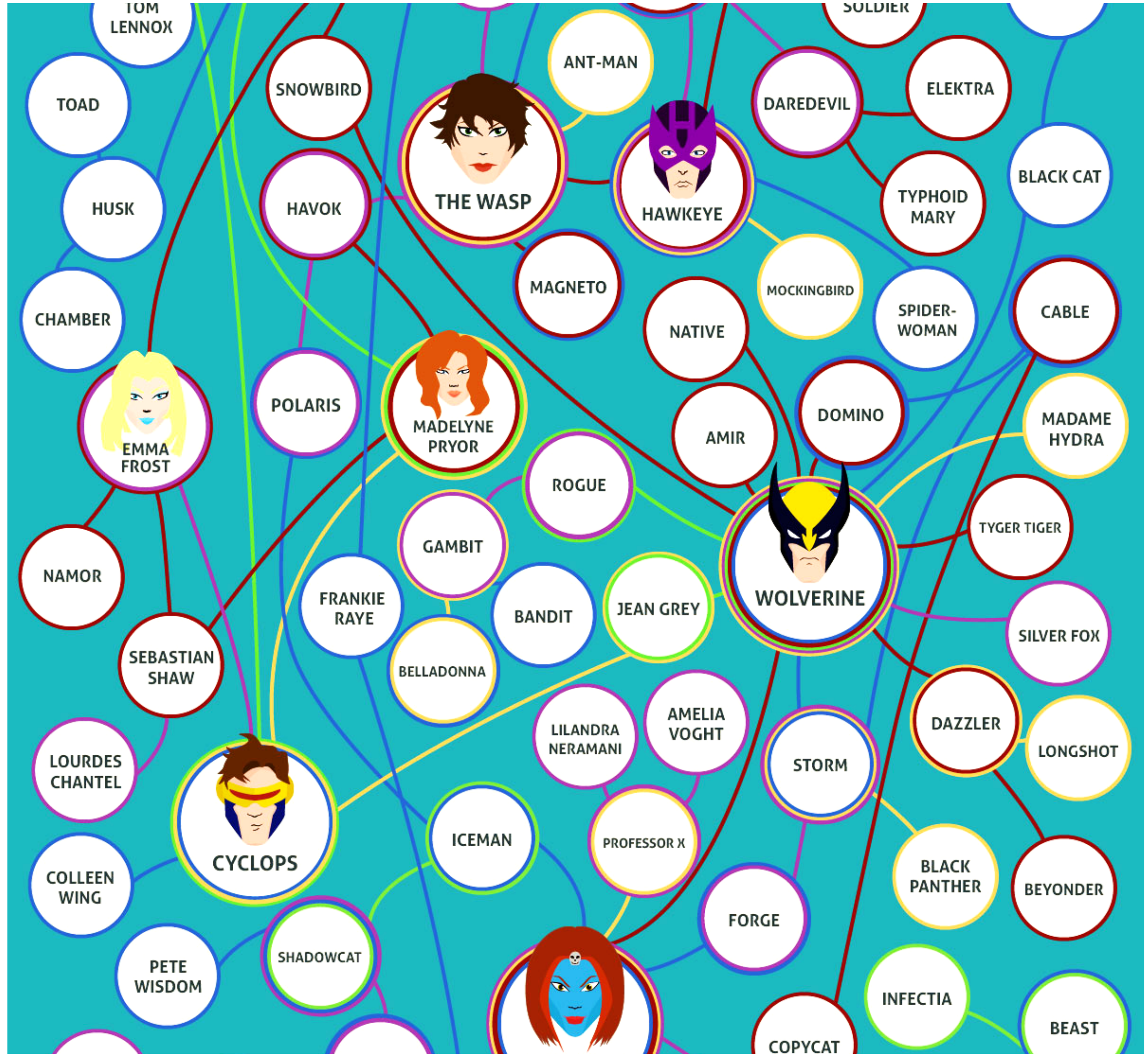


$V = \{S, M, E, B\}$
$E = \{(S, B), (S, E), (S, M), (M, E)\}$

- Graphs can be directed or undirected

Another Example:

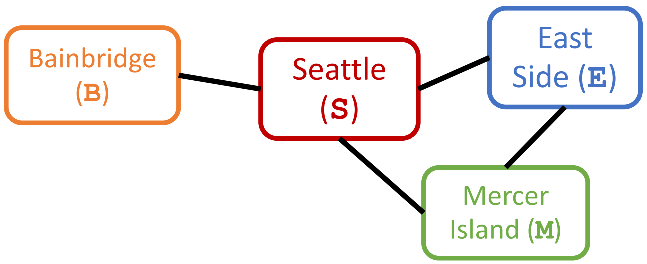
$(\mathbb{V} = \{\text{ characters }\},$
$\mathbb{E} = \{\text{ romances }\})$

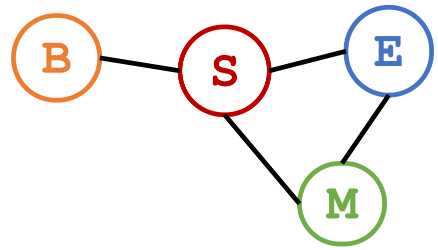


Source: http://www.webhostingbuzz.com/blog/2015/02/10/superlove-marvels-romantic-relationships-mapped/

Undirected Graphs

- In **undirected graphs**, edges have no specific direction
  - Edges are always





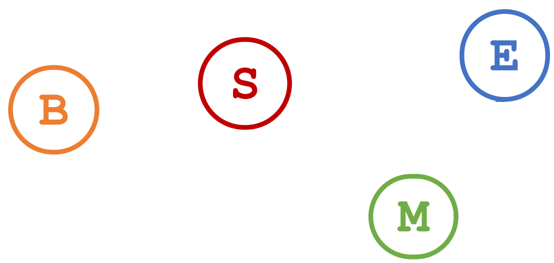
- Thus, `(u,v) ∈ E` implies `(v,u) ∈ E`
  – Only one of these edges needs to be in the set
  – The other is implicit, so normalize how you check for it
- **Degree** of a vertex: number of edges containing that vertex
  – Put another way: the number of adjacent vertices

degree(**S**) =

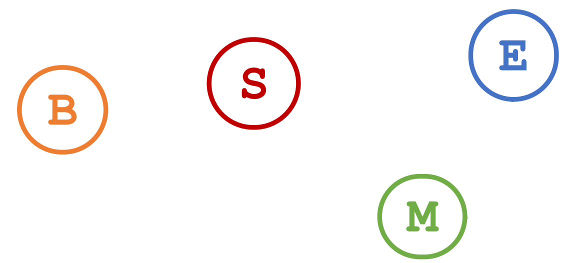
degree(**B**) =

Directed Graphs

- In **directed graphs** (sometimes called **digraphs**), edges have a direction



or



- Thus, `(u,v)` $\in$ `E` does *not* imply `(v,u)` $\in$ `E`.
  - Let `(u,v)` $\in$ `E` mean `u` $\rightarrow$ `v`
  - Call `u` the **source** and `v` the **destination**
- **In-degree** of a vertex: number of in-bound edges,
  i.e., edges where the vertex is the destination
- **Out-degree** of a vertex: number of out-bound edges
  i.e., edges where the vertex is the source

In-degree(**E**) =

Out-degree(**B**) =

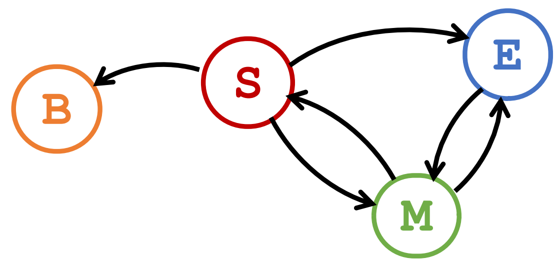
Self-Edges, Connectedness

- A **self-edge** a.k.a. a **loop** is an edge of the form `(u,u)`
  - Depending on the use/algorithm, a graph may have:
    - No self edges
    - Some self edges
    - All self edges (often therefore implicit, but we will be explicit)

- A node can have a degree / in-degree / out-degree of

- A graph does not have to be connected
  - Even if every node has non-zero degree

More notation



V = {**S**,**M**,**E**,**B**}
E = {(**S**,**B**),
(**S**,**E**),
(**S**,**M**),
(**M**,**E**)}

For a graph G = (V,E):

- |V| is the number of vertices
- |E| is the number of edges
    - Minimum?
    - Maximum for undirected?
    - Maximum for directed?

(assuming self-edges allowed, else subtract |V|)

- If (u,v) ∈ E
    - Then v is a **neighbor** of u, i.e., v is **adjacent** to u
    - Order matters for directed edges
        - u is not **adjacent** to v unless (v,u) ∈ E

Is **M** adjacent to **B**?

Is **S** adjacent to **B**?

Is **B** adjacent to **S**?

Examples

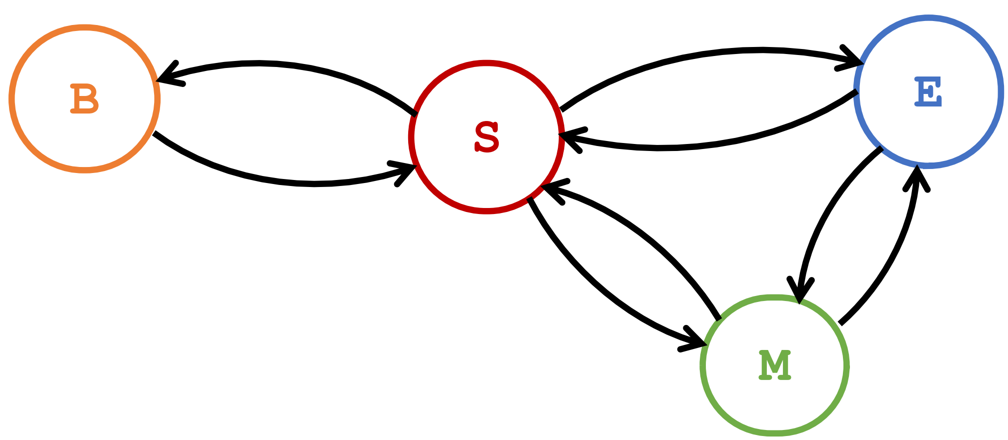
Which would…

Use directed edges? Have self-edges? Be connected? Have 0-degree nodes?

1. Web pages with links
2. Facebook friends
3. Methods in a program that call each other
4. Road maps (e.g., Google maps)
5. Airline routes
6. Family trees
7. Course pre-requisites

Weighted Graphs

- In a weighed graph, each edge has a **weight** a.k.a. **cost**
  - Typically numeric (most examples use ints)
  - Some graphs allow *negative weights*; many do not



Examples

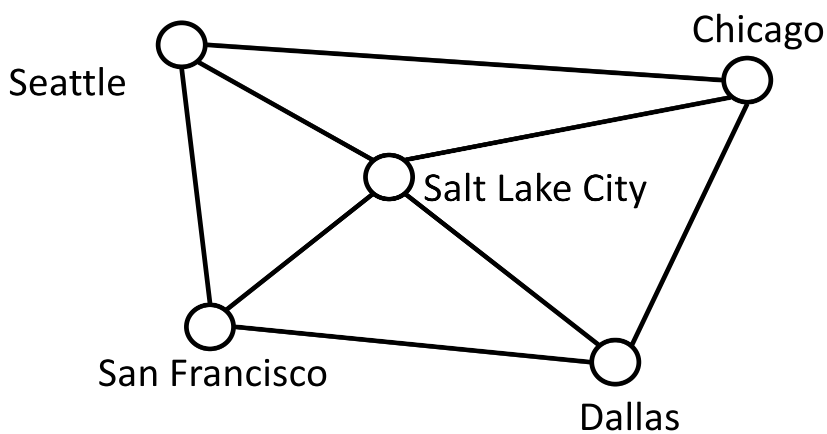
What, if anything, might weights represent for each of these?
Do negative weights make sense?

- Web pages with links
- Facebook friends
- Methods in a program that call each other
- Road maps (e.g., Google maps)
- Airline routes
- Family trees
- Course pre-requisites

Paths and Cycles

- A **path** is a list of vertices $[v_0, v_1, \ldots, v_n]$ such that $(v_i, v_{i+1}) \in E$ for all $0 \le i < n$. Said as *"a path from* v_0 *to* v_n*"*

- A **cycle** is a path that begins and ends at the same node ($v_0 == v_n$)



Example: [Seattle, Salt Lake City, Chicago, Dallas, San Francisco, Seattle]

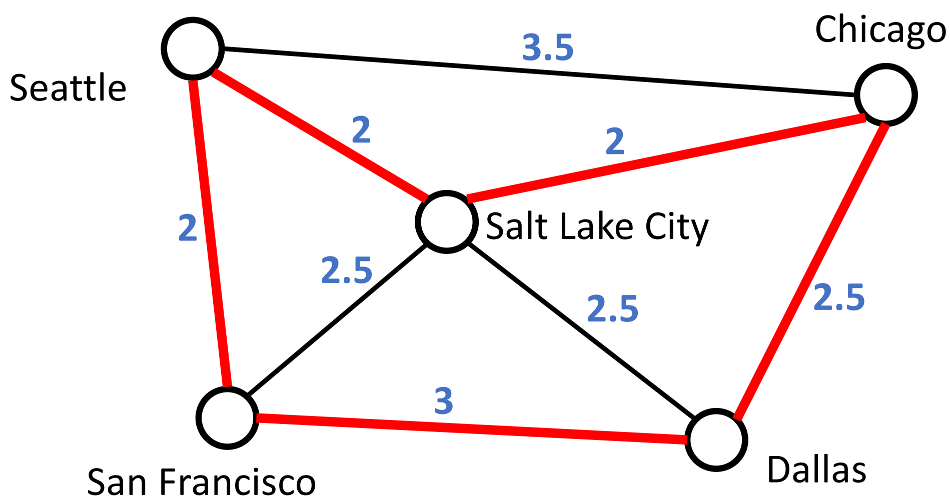
Path Length and Cost

Path length: Number of *edges* in a path

Path cost: Sum of *weights* of edges in a path

Example:

let **P** = [Seattle, Salt Lake City, Chicago, Dallas, San Francisco, Seattle]



length(**P**) =

cost(**P**) =

Simple Paths and Cycles

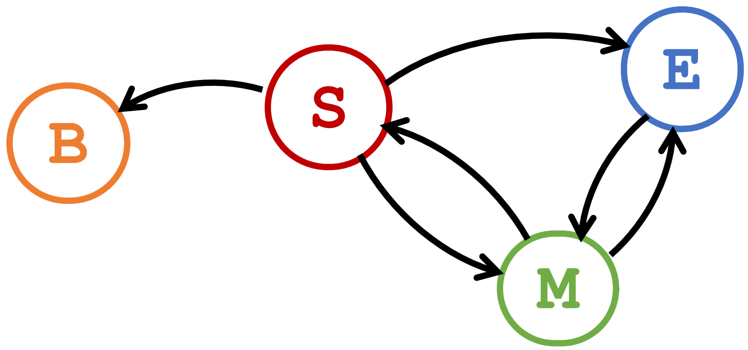
- A **simple path** repeats no vertices, except the first might be the last
  e.g. [Seattle, Salt Lake City, San Francisco, Dallas]
  [Seattle, Salt Lake City, San Francisco, Dallas, Seattle]

- Recall, a **cycle** is a path that ends where it begins
  e.g. [Seattle, Salt Lake City, San Francisco, Dallas, Seattle]
  [Seattle, Salt Lake City, Seattle, Dallas, Seattle]

- A **simple cycle** is a cycle and a simple path
  e.g. [Seattle, Salt Lake City, San Francisco, Dallas, Seattle]

Paths and Cycles in Directed Graphs

Example:

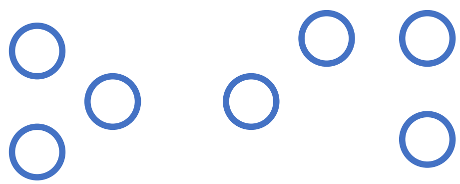


Is there a path from **B** to **M** ?

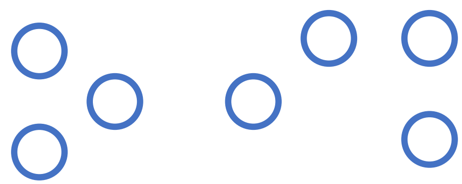
Does the graph contain any cycles?

Undirected-Graph Connectivity

- An undirected graph is **connected** if
  for all pairs of vertices `(u,v)`, there exists a *path* from `u` to `v`

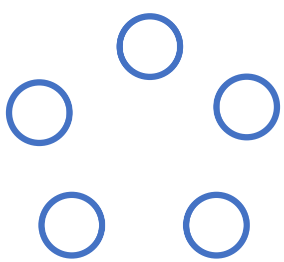


Connected graph



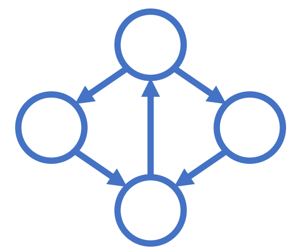
Connected graph

- An undirected graph is **complete**, a.k.a. **fully connected** if
  for *all* pairs of vertices `(u,v)`, there exists an *edge* from `u` to `v`

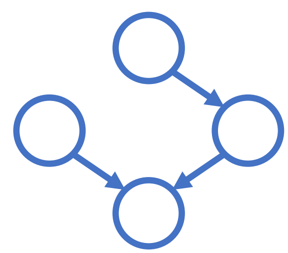


Directed-Graph Connectivity

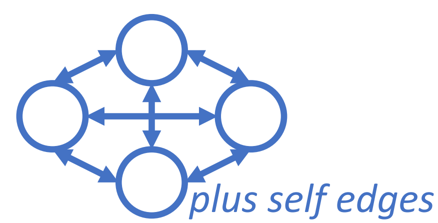
- A directed graph is **strongly connected** if there is a path from every vertex to every other vertex



- A directed graph is **weakly connected** if there is a path from every vertex to every other vertex *ignoring direction of edges*



- A **complete** a.k.a. **fully connected** directed graph has an edge from every vertex to every other vertex



Practice Time!

Let graph `G = (V, E)`

where

`V = {a, b, c, d}`

`E = {(a,b), (b,c), (a,c), (b,d)}`

How connected is `G`?

A. Disconnected

B. Weakly Connected

C. Strongly Connected

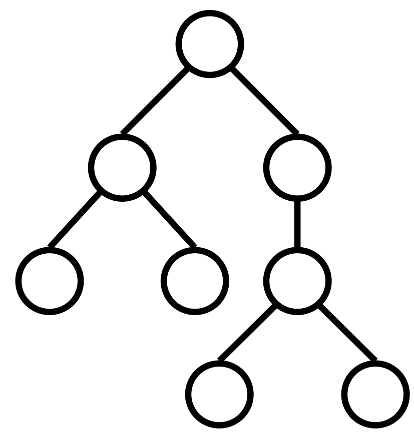
D. Complete / Fully Connected

Trees as Graphs

When talking about graphs,

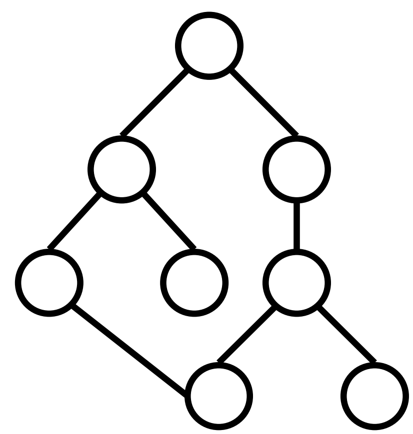
we say a **tree** is a graph that is:

- Connected
- Acyclic
  when you treat edges as undirected



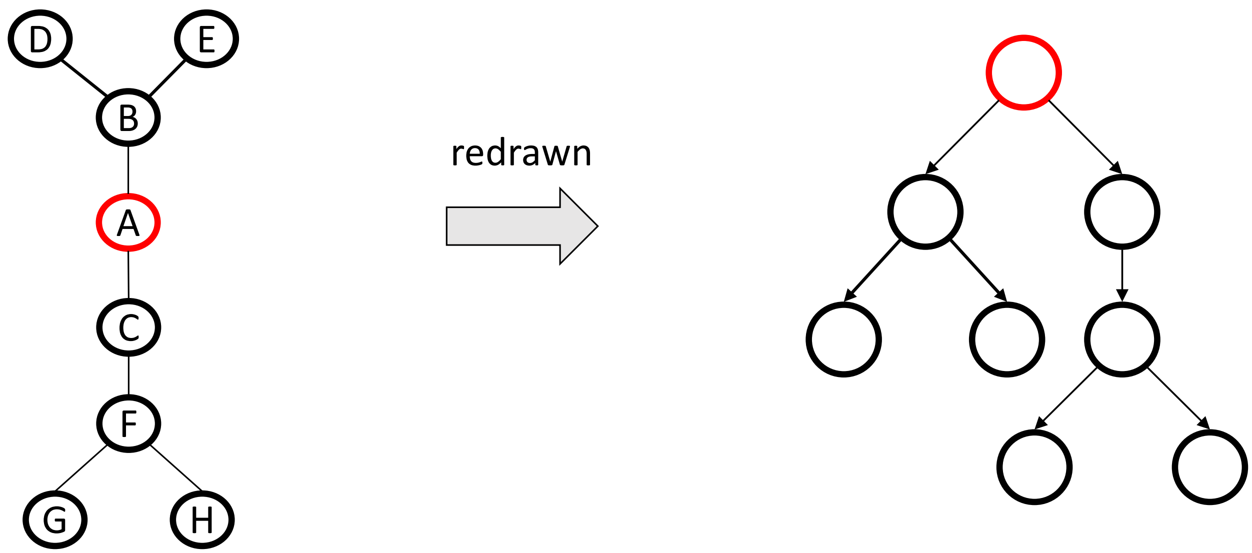
Note that

- Edges can be undirected
- All trees are graphs, but not all graphs are trees



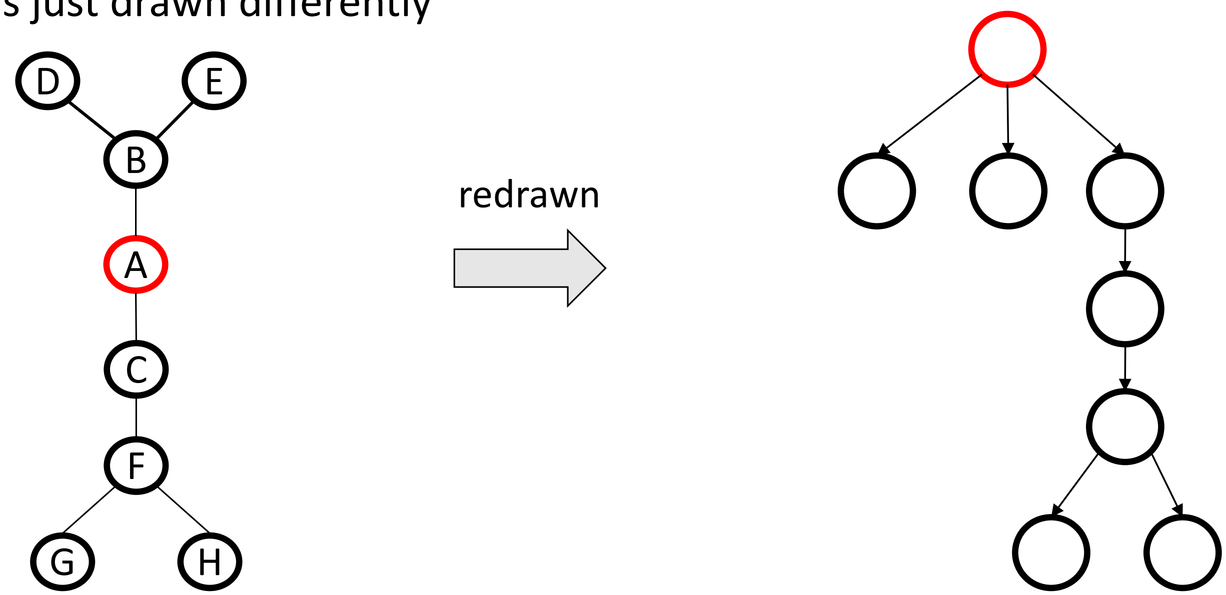
Rooted Trees

- We are more accustomed to **rooted trees** where:
  - We identify a unique root
  - We think of edges as directed: parent to children
- Given a tree, picking a root gives a unique rooted tree
  - The tree is just drawn differently



Rooted Trees

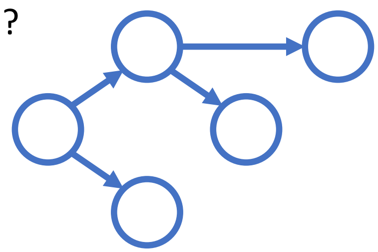
- We are more accustomed to **rooted trees** where:
  - We identify a unique root
  - We think of edges as directed: parent to children
- Given a tree, picking a root gives a unique rooted tree
  - The tree is just drawn differently



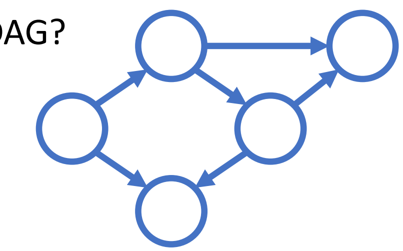
Directed Acyclic Graphs (DAGs)

- A **DAG** is a directed graph with no (directed) cycles
  - Every rooted directed tree is a DAG
  - But not every DAG is a rooted directed tree

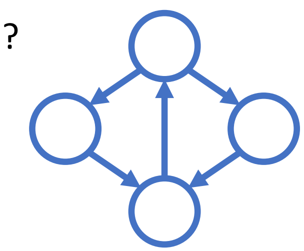
DAG?



DAG?



DAG?



  - Every DAG is a directed graph
  - But not every directed graph is a DAG

Examples

Which of our directed-graph examples do you expect to be a DAG?

- Web pages with links
- Methods in a program that call each other
- Airline routes
- Family trees
- Course pre-requisites

Density / Sparsity

- Recall: In an undirected graph, $0 \leq |\mathtt{E}| < |\mathtt{V}|^2$
- Recall: In a directed graph: $0 \leq |\mathtt{E}| \leq |\mathtt{V}|^2$
- So for any graph, $O(|\mathtt{E}|+|\mathtt{V}|)$ is

- Another fact: If an undirected graph is *connected*, then $|\mathtt{V}|-1 \leq |\mathtt{E}|$

- Because $|\mathtt{E}|$ is often much smaller than its maximum size, we do not always approximate $|\mathtt{E}|$ as $O(|\mathtt{V}|^2)$
  - This is a correct bound, it just is often not tight
  - If it is tight, i.e., $|\mathtt{E}|$ is $\theta(|\mathtt{V}|^2)$ we say the graph is **dense**
    - More sloppily, dense means
  - If $|\mathtt{E}|$ is $O(|\mathtt{V}|)$ we say the graph is **sparse**
    - More sloppily, sparse means “most possible edges

What is the Data Structure?

- So graphs are really useful for lots of data and questions
  - For example, “what’s the lowest-cost path from x to y”

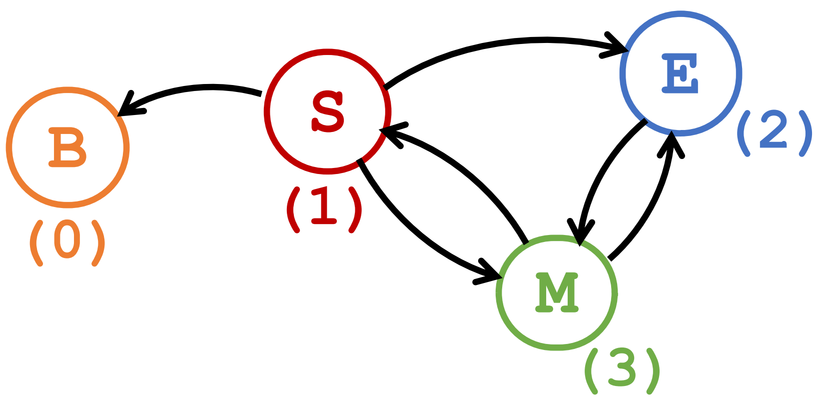
- But we need a data structure that represents graphs

- The “best one” can depend on:
  - Properties of the graph (e.g., dense versus sparse)
  - The common queries (e.g., “is `(u,v)` an edge?” versus “what are the neighbors of node **`u`**?”)

- So we’ll discuss the two standard graph representations
  - **Adjacency Matrix** and **Adjacency List**
  - Different trade-offs, particularly time versus space

Adjacency Matrix

- Assign each node a number from `0` to `|V|-1`
- A `|V|` x `|V|` matrix (i.e., 2-D array) of Booleans (or 1 vs. 0)
  - If `M` is the matrix, then `M[u][v] == true` means there is an edge from `u` to `v`

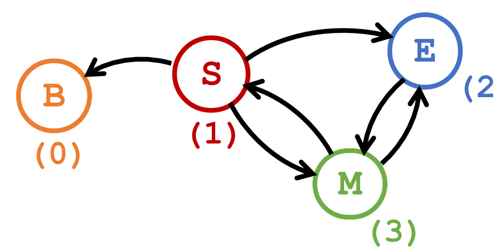


	0	1	2	3
0				
1				
2				
3				

Adjacency Matrix Properties

- Running time to:
  - Get a vertex's out-edges:
  - Get a vertex's in-edges:
  - Decide if some edge exists:
  - Insert an edge:
  - Delete an edge:
- Space requirements:
- Best for sparse or dense graphs?

	0	1	2	3
0	F	F	F	F
1	T	F	T	T
2	F	F	T	T
3	F	T	T	F



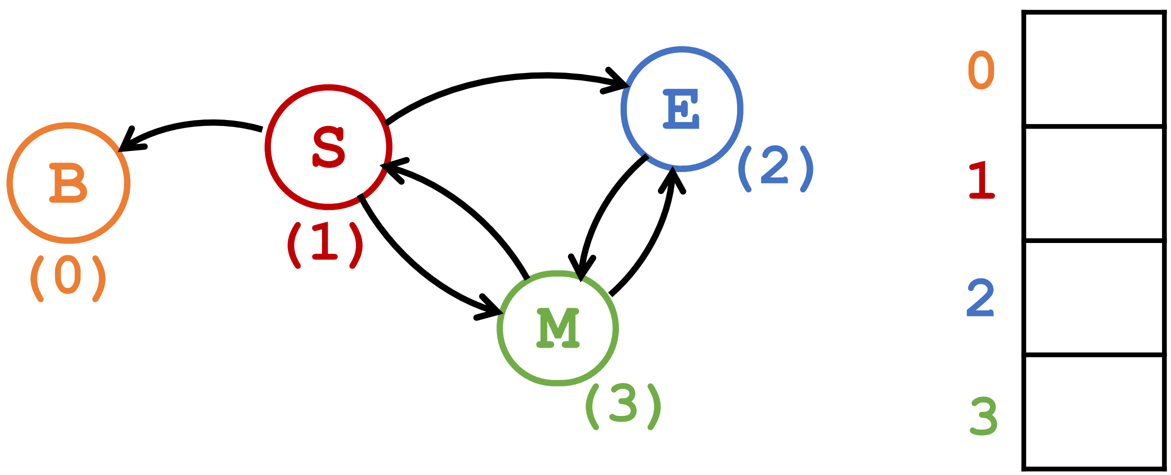
Adjacency Matrix Properties

- How will the adjacency matrix vary for an *undirected graph*?
  - Undirected will be symmetric around the diagonal

- How can we adapt the representation for *weighted graphs*?
  - Instead of a Boolean, store a number in each cell
  - Need some value to represent ‘not an edge’
    - In *some* situations, 0 or -1 works

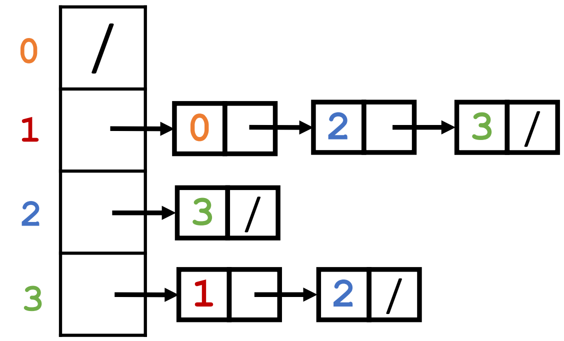
Adjacency List

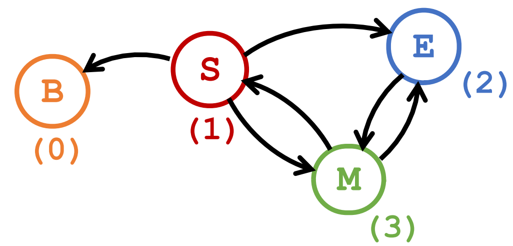
- Assign each node a number from `0` to `|V|-1`
- An array of length |V| in which each entry stores a list of all adjacent vertices (e.g., linked list)



Adjacency List Properties

- Running time to:
  - Get all of a vertex's out-edges:
    where *d* is out-degree of vertex
  - Get all of a vertex's in-edges:
    (but could keep a second adjacency list for this!)
  - Decide if some edge exists:
    where *d* is out-degree of source
  - Insert an edge:
    (unless you need to check if it's there)
  - Delete an edge:
    where *d* is out-degree of source
- Space requirements:
  - $O(|V|+|E|)$





Best for sparse or dense graphs?