CSE332: Data Abstractions

Lecture 15: Introduction to Graphs

Tyler Robison

2010 Summer

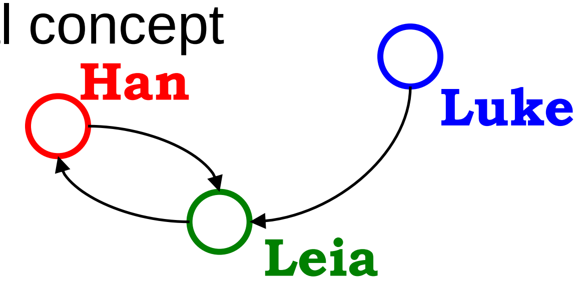
Graphs

- A graph is a formalism for representing relationships among items
  - Very general definition because very general concept
- A graph is a pair of sets

  G = (V,E)
  - A set of vertices, also known as nodes

    $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$
  - A set of edges

    $\mathbf{E} = \{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_m\}$
    - Each edge $\mathbf{e}_i$ is a pair of vertices
    - An edge "connects" the vertices
- Graphs can be directed or undirected



V = {Han,Leia,Luke}

E = {(Luke,Leia),
(Han,Leia),
(Leia,Han)}

Some graphs

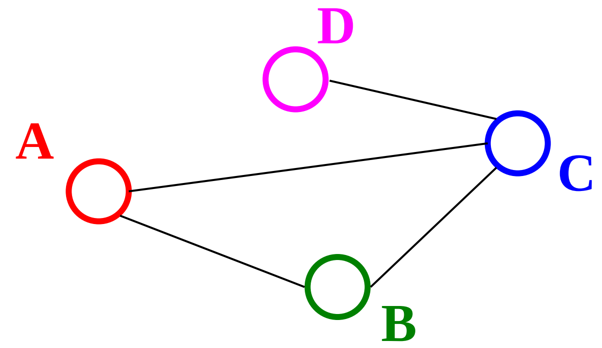
For each, what are the vertices and what are the edges?

- Web pages with links
- Facebook friends
- “Input data” for the Kevin Bacon game
- Methods in a program that call each other
- Road maps (e.g., Google maps)
- Airline routes
- Family trees
- Course pre-requisites
- …

Quite versatile & useful

Undirected Graphs

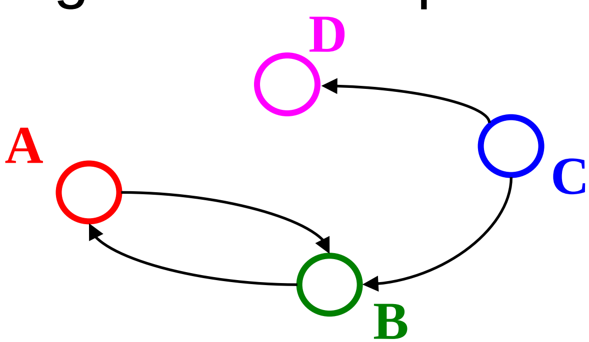
- In undirected graphs, edges have no specific direction
  - Edges are always “two-way”



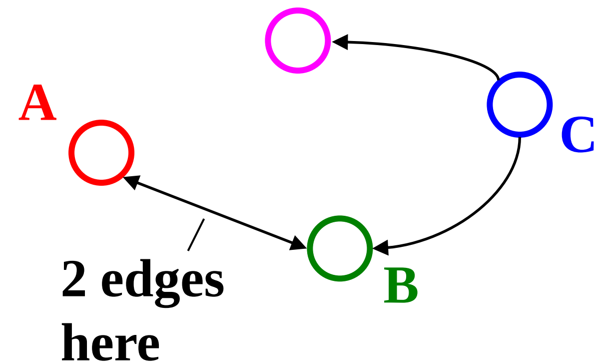
- Thus, **(u,v)** ∈ **E** implies **(v,u)** ∈ **E**.
  - Only one of these edges needs to be in the set; the other is implicit
- Degree of a vertex: number of edges containing that vertex
  - Put another way: the number of adjacent vertices

Directed graphs

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



or

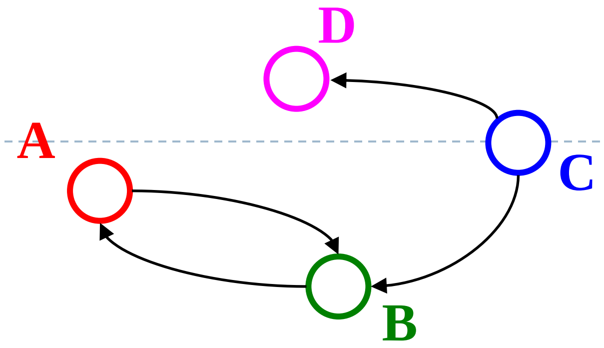


- Thus, **`(u,v)` ∈ `E`** does *not* imply **`(v,u)` ∈ `E`**.
  - Let **`(u,v)` ∈ `E`** mean u → v and call **`u`** the source and **`v`** the destination
- In-Degree of a vertex: number of in-bound edges, i.e., edges where the vertex is the destination
- Out-Degree of a vertex: number of out-bound edges, i.e., edges where the vertex is the source

Self-edges, connectedness, etc.

- 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 (in which case often implicit, but we will be explicit)
- A node can have a degree / in-degree / out-degree of zero
- (Undirected) Connected: We can follow edges from any node to get to any other node
  - Not necessarily connected, even if every node has non-zero degree

More notation



For a graph **G=(V,E)**:

- **|V|** is the number of vertices
- **|E|** is the number of edges
  - Minimum edges?
    - **0**
  - Maximum edges for undirected?
    - $|V||V+1|/2 \in O(|V|^2)$
  - Maximum edges for directed?
    - $|V|^2 \in O(|V|^2)$
      (assuming self-edges allowed, else subtract **|V|**)

- If $(u,v) \in E$
  - Then **v** is a neighbor of **u**,
    i.e., **v** is adjacent to **u**
  - Order matters for directed edges

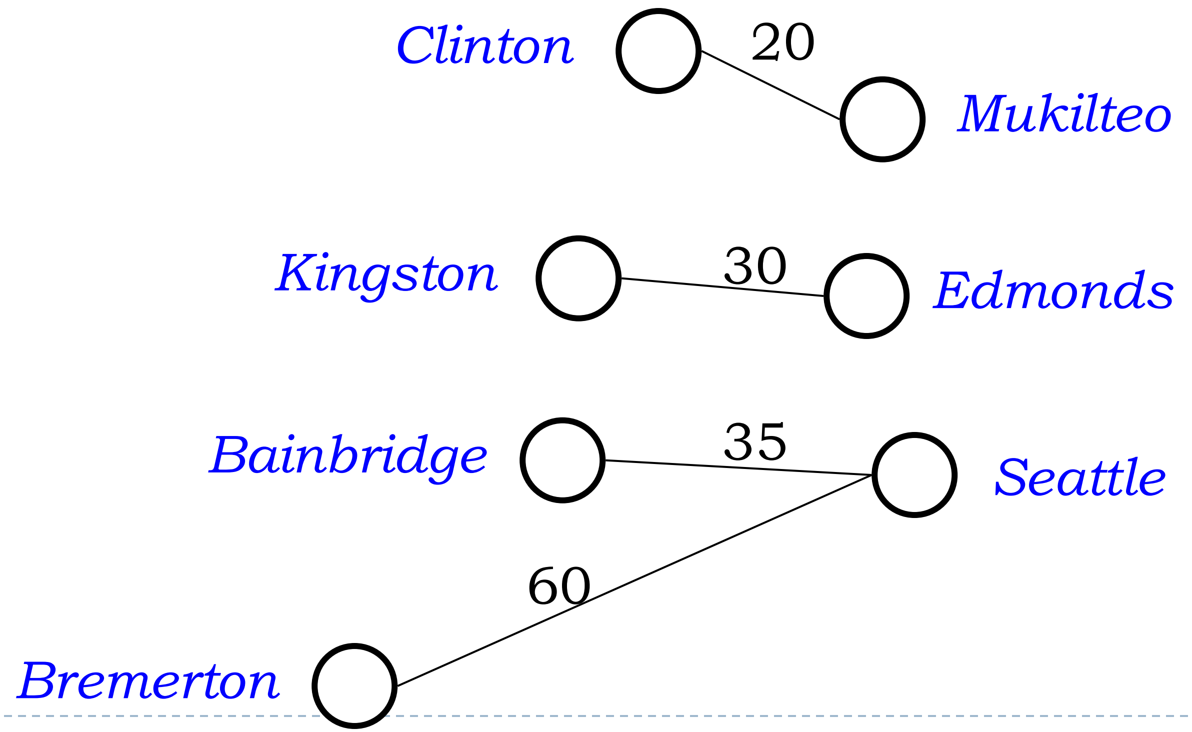
Examples again

Which would use directed edges? Which would have self-edges? Which could have 0-degree nodes?

- Web pages with links
- Facebook friends
- "Input data" for the Kevin Bacon game
- Methods in a program that call each other
- Road maps (e.g., Google maps)
- Airline routes
- Family trees
- Course pre-requisites
- …

Weighted graphs

- In a weighed graph, each edge has a weight a.k.a. cost
  - Typically numeric (most examples will use ints)
  - *Orthogonal* to whether graph is directed
  - Some graphs allow *negative weights*; many don't



Examples

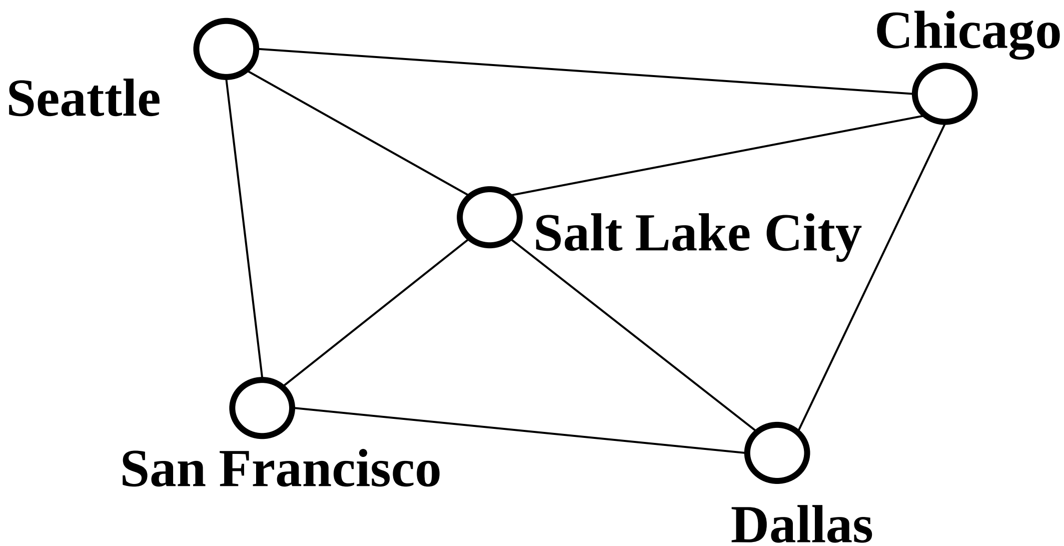
What, if anything, might weights represent for each of these? Do negative weights make sense?

- Web pages with links
- Facebook friends
- “Input data” for the Kevin Bacon game
- 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 $[\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_n]$ such that $(\mathbf{v}_i, \mathbf{v}_{i+1}) \in \mathbf{E}$ for all $0 \leq i < n$. Say *"a path from* $\mathbf{v}_0$ *to* $\mathbf{v}_n$*"*

- A cycle is a path that begins and ends at the same node ($\mathbf{v}_0 == \mathbf{v}_n$)



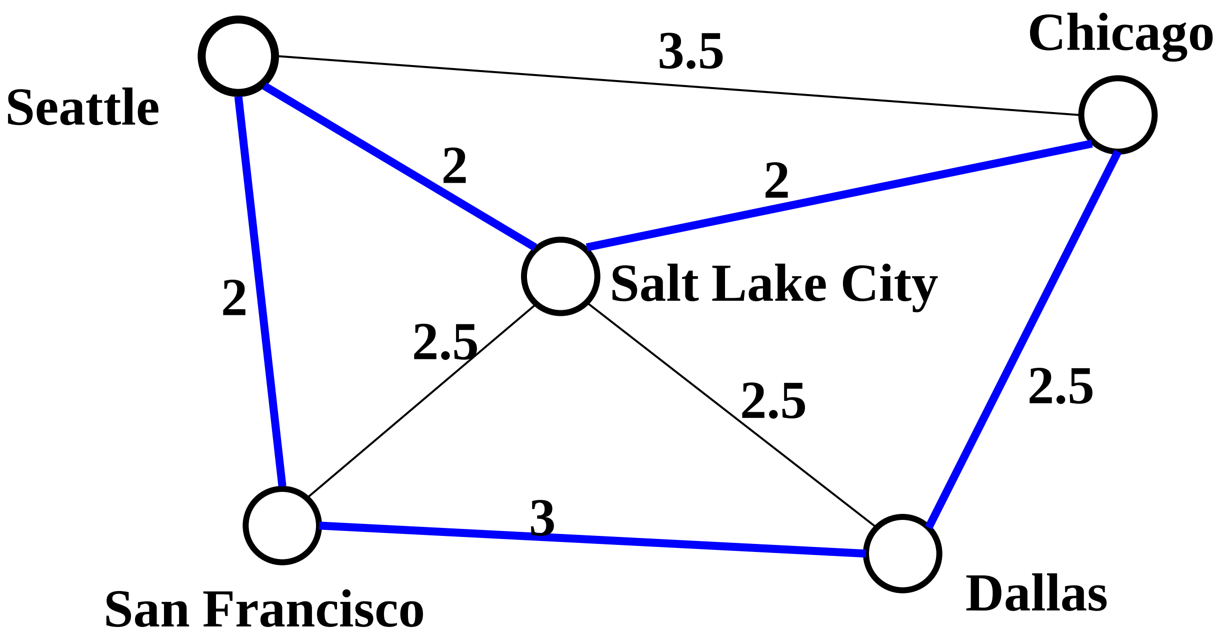
Example cycle: [Seattle, Salt Lake City, Chicago, Dallas, San Francisco, Seattle]

Path Length and Cost

- Path length: Number of *edges* in a path
- Path cost: sum of the weights of each edge

Example where

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



length(P) = 5

cost(P) = 11.5

Simple paths and cycles

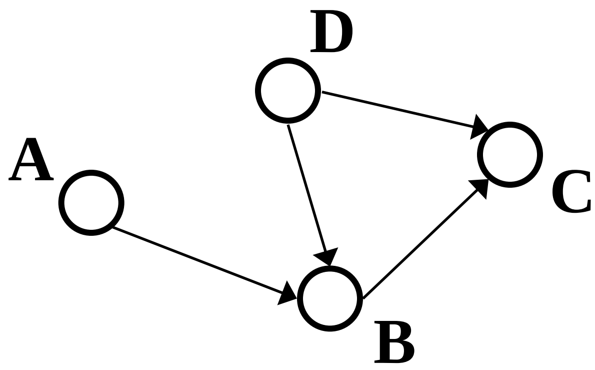
- A simple path repeats no vertices, except the first might be the last
  [Seattle, Salt Lake City, San Francisco, Dallas]
  [Seattle, Salt Lake City, San Francisco, Dallas, Seattle]

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

Paths/cycles in directed graphs

Example:

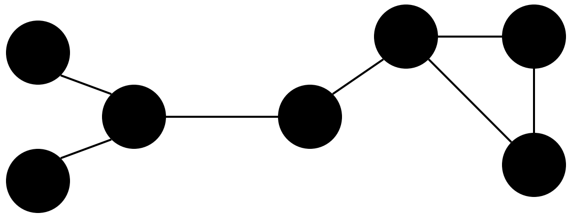


Is there a path from A to D?

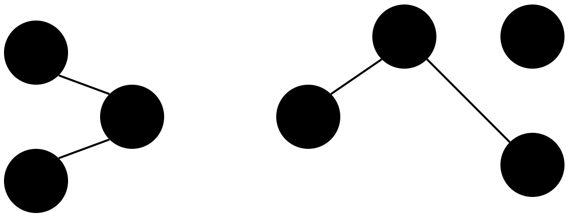
Does the graph contain any cycles?

Undirected graph connectivity

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

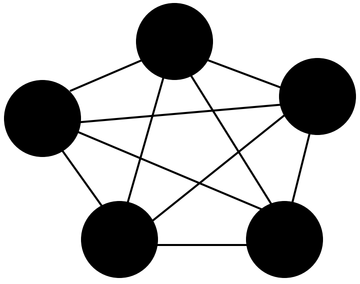


Connected graph



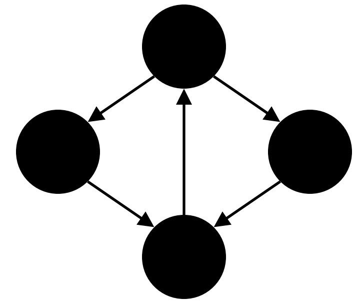
Disconnected graph

- 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`**

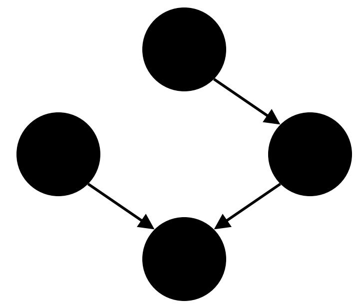


Directed graph connectivity

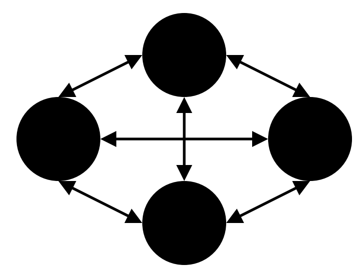
- A directed graph is strongly connected if there is a path from every vertex to every other vertex



- 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



Examples

For undirected graphs: connected? For directed graphs: strongly connected? weakly connected?

- Web pages with links
- Facebook friends
- “Input data” for the Kevin Bacon game
- Methods in a program that call each other
- Road maps (e.g., Google maps)
- Airline routes
- Family trees
- Course pre-requisites
- …

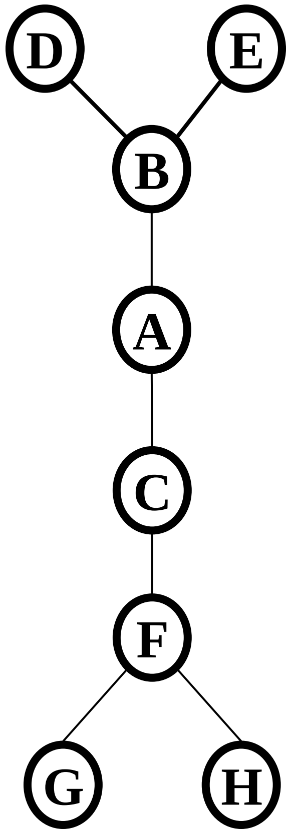
Trees as graphs

When talking about graphs, we say a tree is a graph that is:

- acyclic
- connected
- undirected

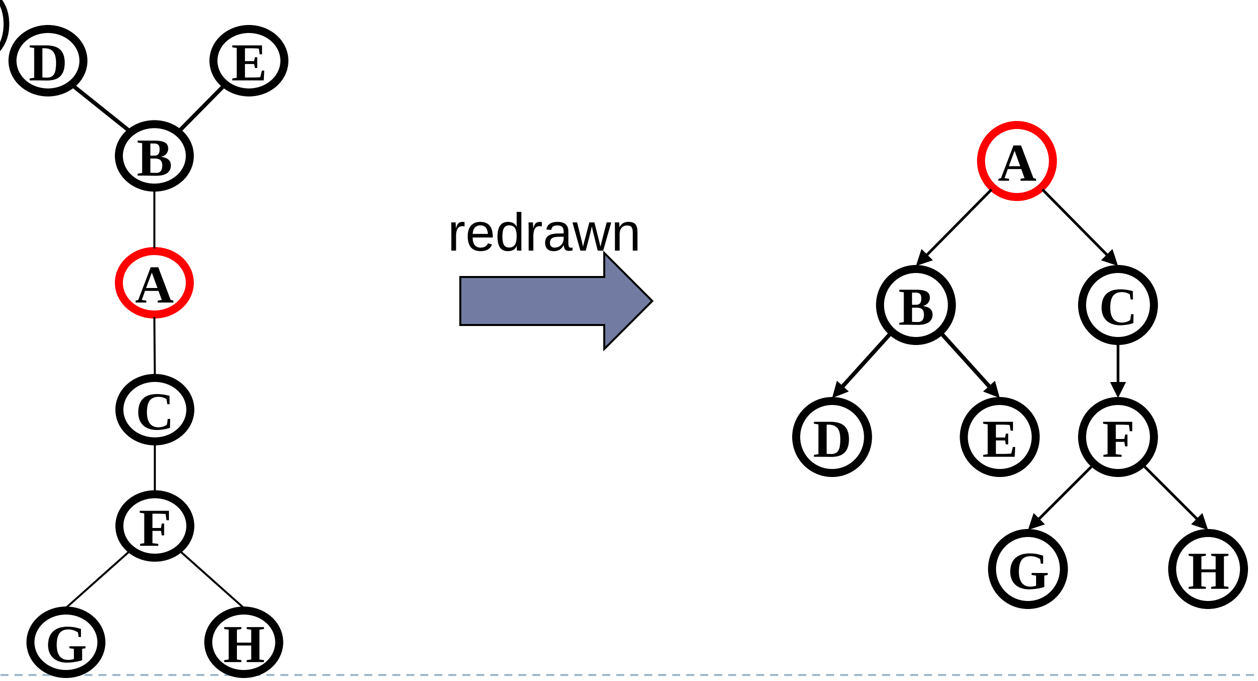
So all trees are graphs, but not all graphs are trees

Example:



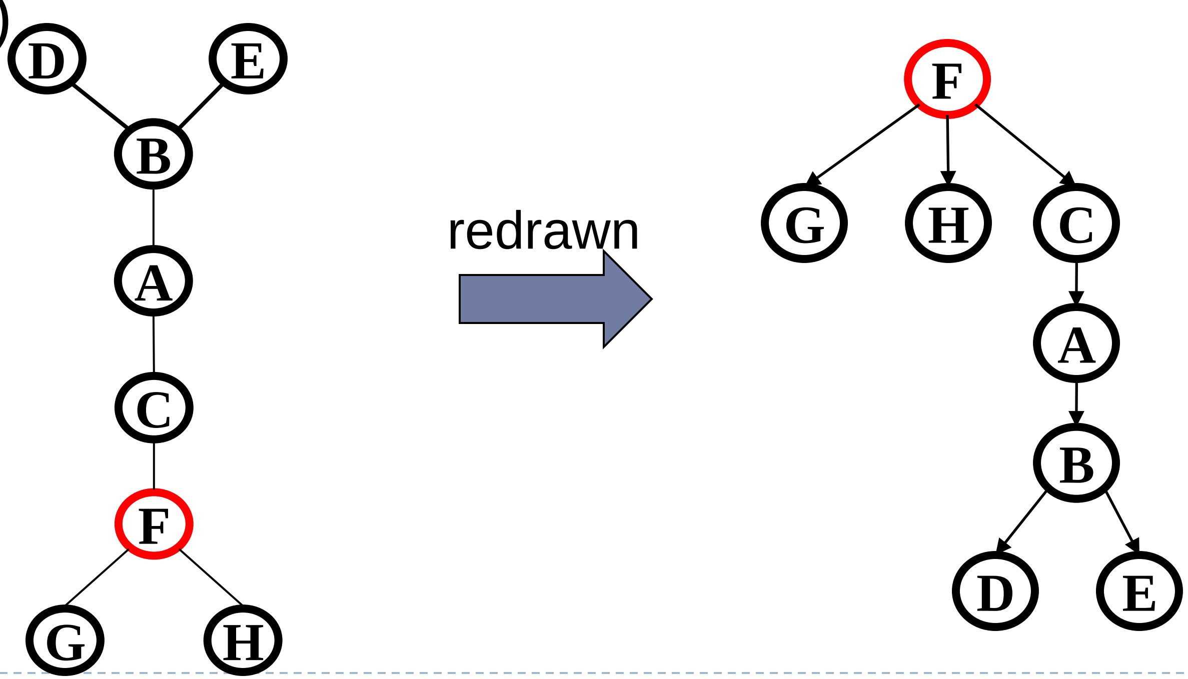
Rooted Trees

- We are more accustomed to rooted trees where:
  - We identify a unique ("special") root
  - We think of edges as directed: parent to children
- Given a tree, once you pick a root, you have a unique rooted tree (just drawn differently and with undirected edges)



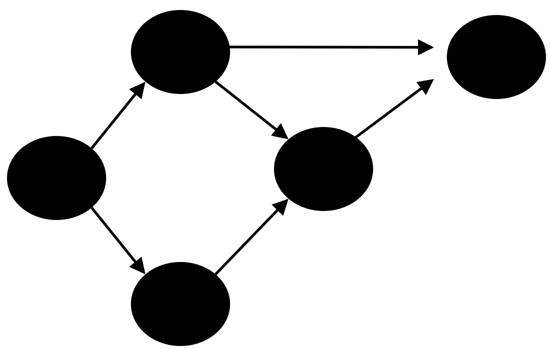
Rooted Trees

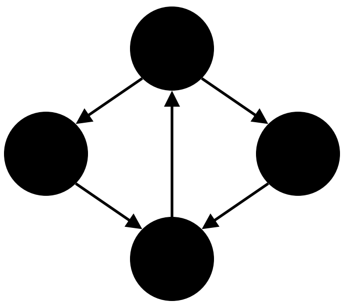
- We are more accustomed to rooted trees where:
  - We identify a unique ("special") root
  - We think of edges as directed: parent to children
- Given a tree, once you pick a root, you have a unique rooted tree (just drawn differently and with undirected edges)



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

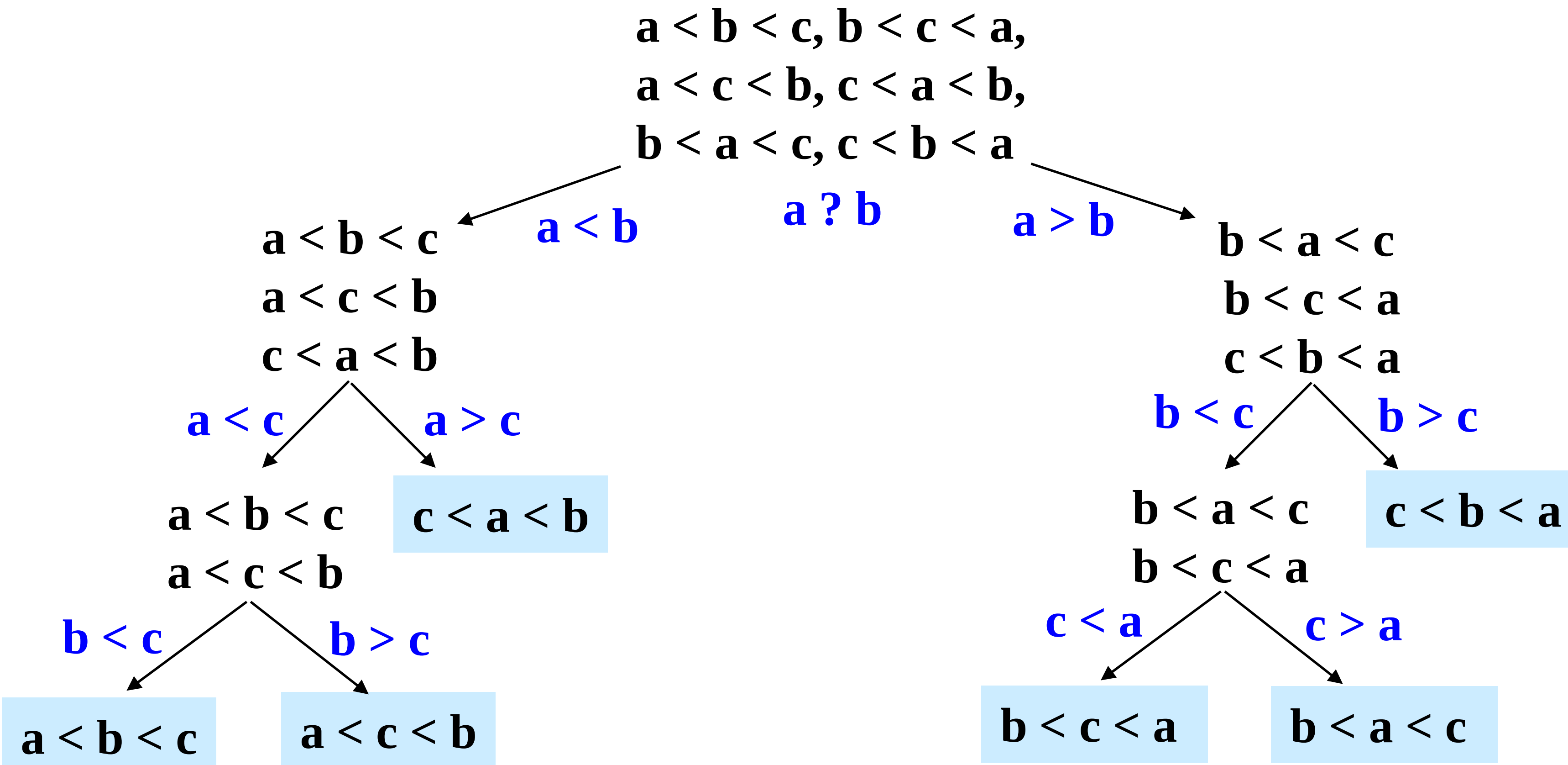




  - Every DAG is a directed graph
  - But not every directed graph is a DAG

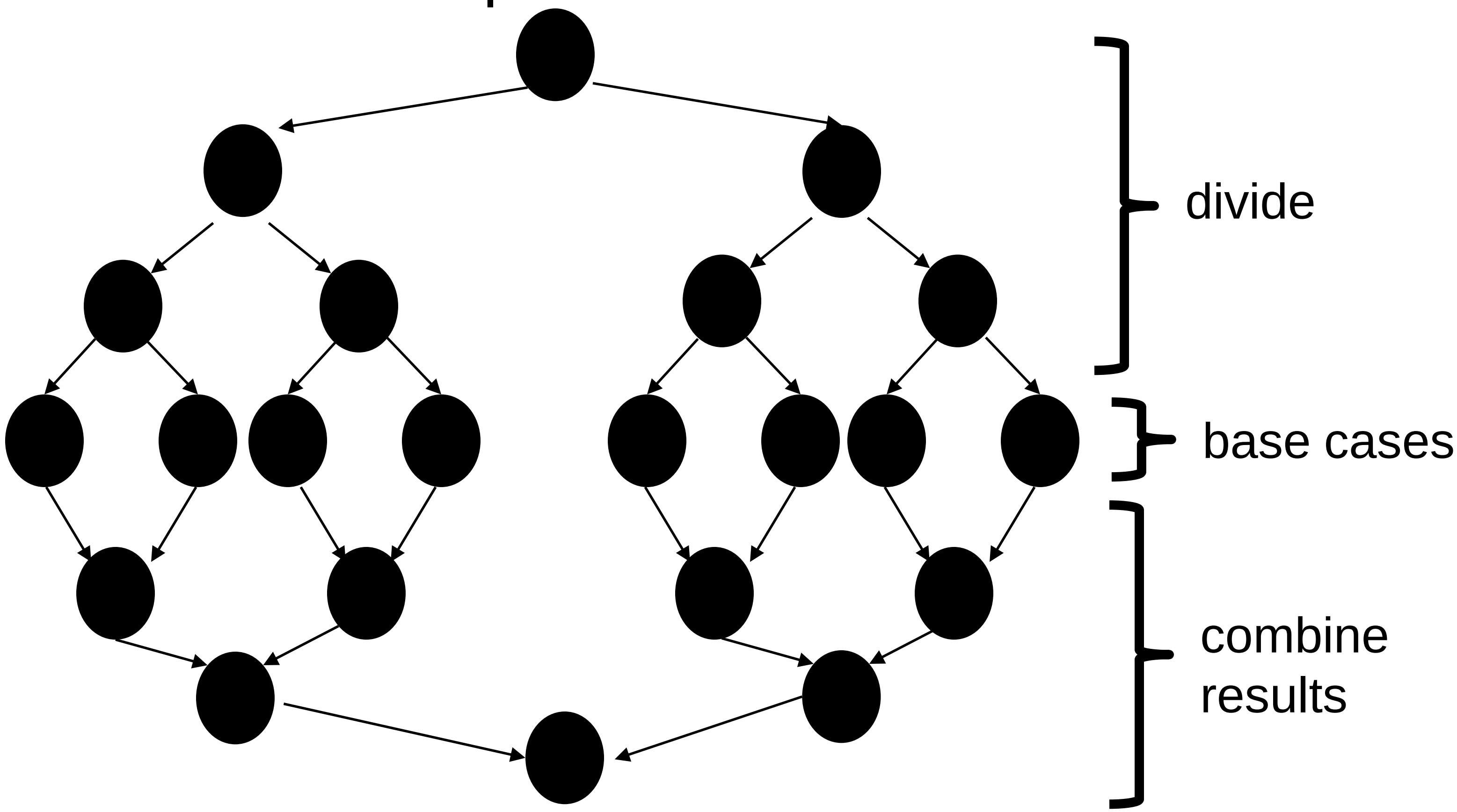
Problem Representation

- Decision Tree as rooted, directed tree
- Start at root; follow outcome of comparisons



Problem Representation

- Quick/MergeSort as a graph
- Nodes as conceptual states of data



Density / sparseness

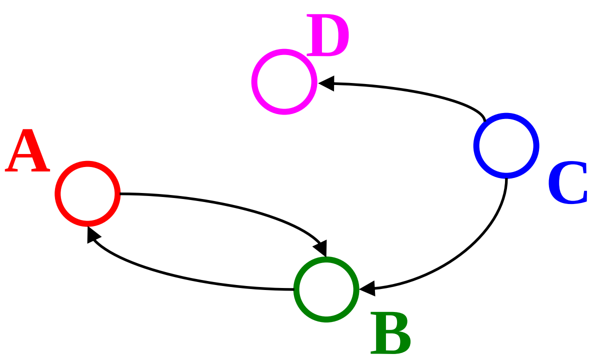
- Recall: In an undirected graph, $0 \leq |E| \leq |V|^2$
- Recall: In a directed graph: $0 \leq |E| \leq |V|^2$
- So for any graph, $|E|$ is $O(|V|^2)$
- One more fact: If an undirected graph is *connected*, then $|V|-1 \leq |E|$
- Because $|E|$ is often much smaller than its maximum size, we do not always approximate as $|E|$ as $O(|V|^2)$
  - This is a correct bound, it just is often not tight
  - If it is tight, i.e., $|E|$ is $\Theta(|V|^2)$ we say the graph is dense
    - More sloppily, dense means "lots of edges"
  - If $|E|$ is $O(|V|)$ we say the graph is sparse
    - More sloppily, sparse means "most possible edges missing"

Now the data structure

- Okay, so graphs are really useful for lots of data and questions we might ask like "what's the lowest-cost path from x to y"
- But we need a data structure that represents graphs
- Which data structure is "best" 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…
  - 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`**



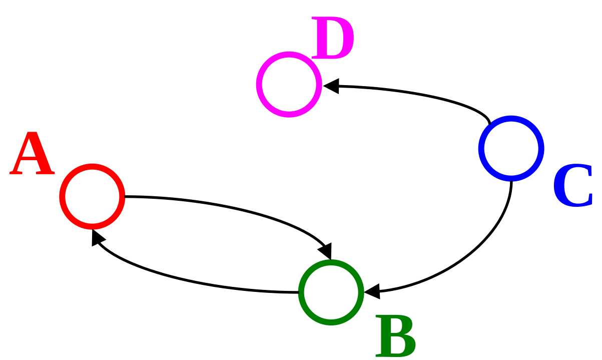
	A	B	C	D
A	F	T	F	F
B	T	F	F	F
C	F	T	F	T
D	F	F	F	F

Adjacency matrix properties

- Running time to:
  - Get a vertex's out-edges: $O(|V|)$
  - Get a vertex's in-edges: $O(|V|)$
  - Decide if some edge exists: $O(1)$
  - Insert an edge: $O(1)$
  - Delete an edge: $O(1)$

	A	B	C	D
A	F	T	F	F
B	T	F	F	F
C	F	T	F	T
D	F	F	F	F

- Space requirements:
  - $|V|^2$ bits



- Best for dense graphs

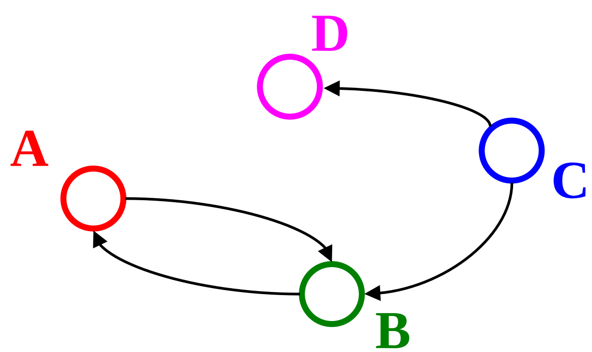
Adjacency matrix properties

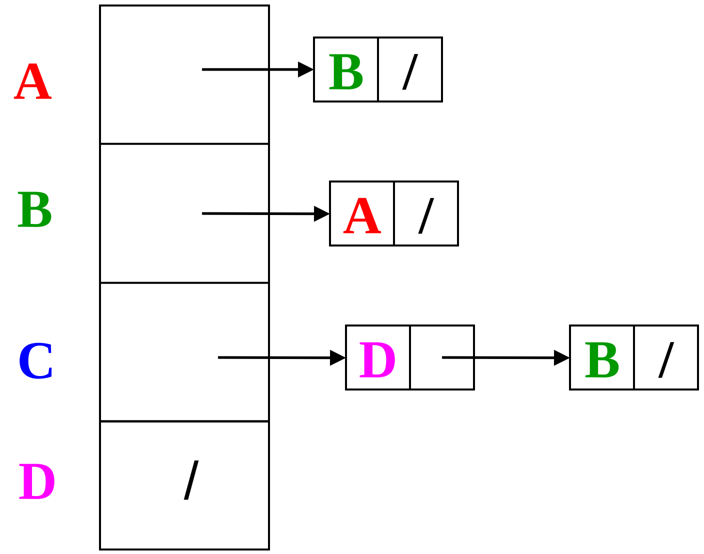
- How will the adjacency matrix vary if (un)directed?
  - Undirected: Will be symmetric about diagonal axis
- How can we adapt the representation for weighted graphs?
  - Instead of a boolean, store an int/double in each cell
  - Need some value to represent ‘not an edge’
    - Say -1 or 0

	A	B	C	D
A	F	T	F	F
B	T	F	F	F
C	F	T	F	T
D	F	F	F	F

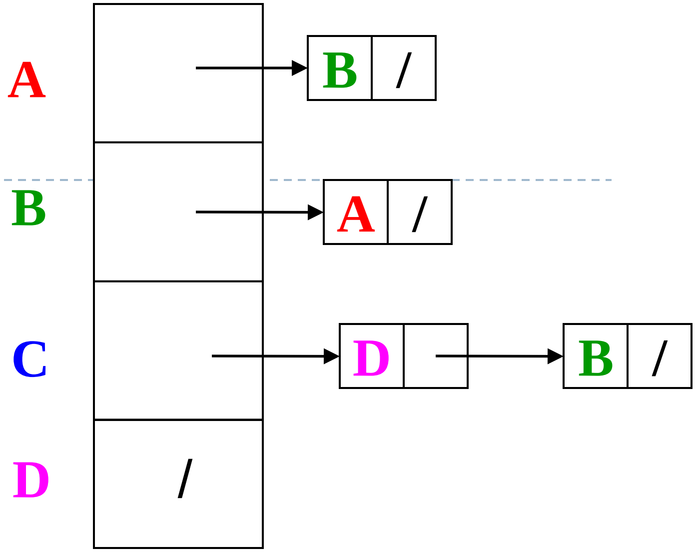
Adjacency List

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





Adjacency List Properties

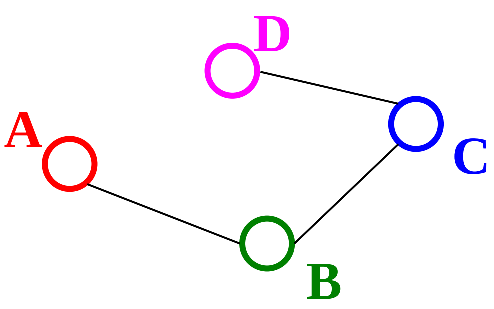


- Running time to:
  - Get all of a vertex's out-edges: $O(d)$ where d is out-degree of vertex
  - Get all of a vertex's in-edges: O(|E|) (but could keep a second adjacency list for this!)
  - Decide if some edge exists: $O(d)$ where d is out-degree of source
  - Insert an edge: $O(1)$
  - Delete an edge: $O(d)$ where d is out-degree of source
- Space requirements:
  - $O(|V|+|E|)$
- Best for sparse graphs: so usually just stick with linked lists

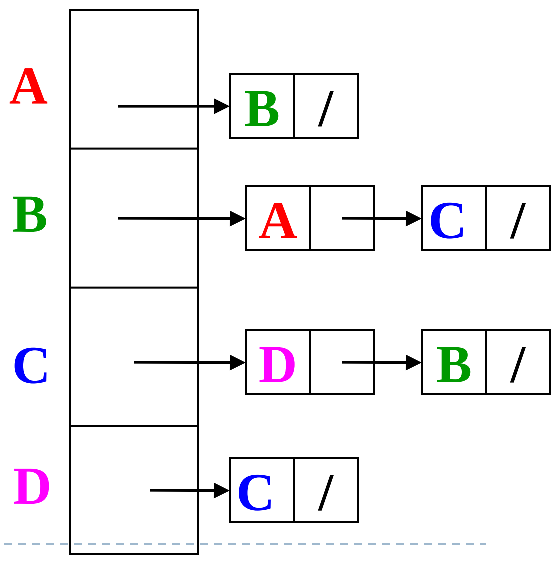
Undirected graphs

Adjacency matrices & adjacency lists both do fine for undirected graphs

- Matrix: Could save space; only ~1/2 the array is used
- Lists: Each edge in two lists to support efficient “get all neighbors”



	A	B	C	D
A	F	T	F	F
B	T	F	T	F
C	F	T	F	T
D	F	F	T	F



Next…

Okay, we can represent graphs

Now let's implement some useful and non-trivial algorithms

- Topological sort: Given a DAG, order all the vertices so that every vertex comes before all of its neighbors