# CSE 332
# Data Structures & Parallelism

## Graphs

*Melissa Winstanley*
*Spring 2024*

# Reminders

- Exercise 6 (hashing) due tomorrow 11:59pm
- P2 CP2 due Thursday, 11:59pm
- Exercise 7 (sorting) due next week

# Our schedule

- This week: graphs
- Next week: parallelism
  - Need to make sure you have what you need for P3!
- Last week of class: more graphs

Today: Graphs

- Intro & definitions

# Problem space

Problems where the dataset is best represented as **items** and the **relationships between them**

Examples:

- Friendships
- Family trees
- ...?

# Graphs

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

  $G = (V,E)$

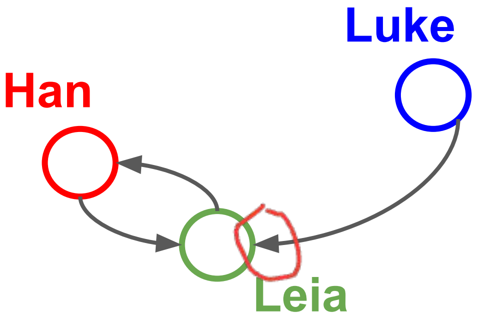
  - A set of vertices, also known as nodes

    $V = \{v_1, v_2, \ldots, v_n\}$

  - A set of edges

    $E = \{e_1, e_2, \ldots, e_m\}$

    - Each edge $e_i$ is a pair of vertices $(v_j, v_k)$
    - An edge "connects" the vertices
- Graphs can be directed or undirected



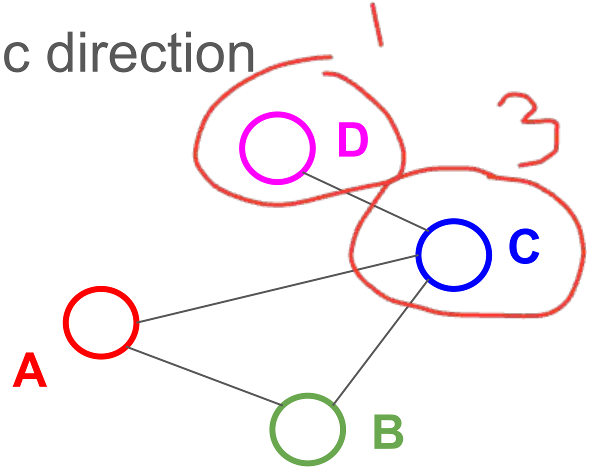

V = {Han, Leia, Luke}
E = { (Luke, Leia),
(Han, Leia),
(Leia, Han) }

# An ADT?

- Can think of graphs as an ADT with operations like `isEdge((v_j,v_k))` — $\texttt{isEdge}((\texttt{v}_\texttt{j},\texttt{v}_\texttt{k}))$
- But it is unclear what the “standard operations” are
- Instead we tend to develop algorithms over graphs and then use data structures that are efficient for those algorithms
- Many important problems can be solved by:
    1. Formulating them in terms of graphs
    2. Applying a standard graph algorithm
- To make the formulation easy and standard, we have a lot of *standard terminology* about graphs

# Undirected Graphs

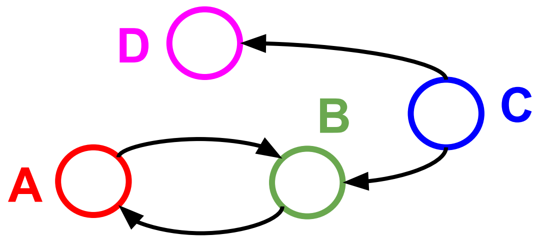
- In undirected graphs, edges have no specific direction
  - Edges are always “two-way”



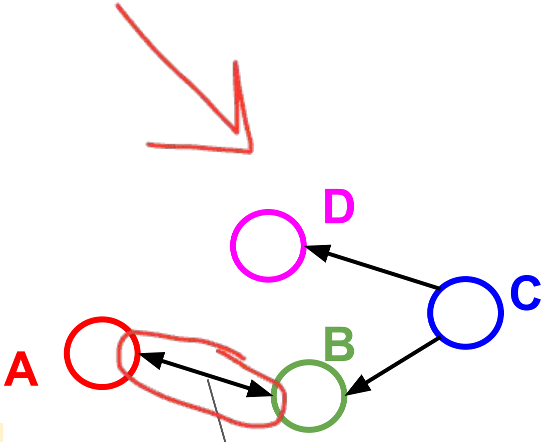

- Thus, `(u,v) ∈ E` implies `(v,u) ∈ E`.
  - Only one of these edges needs to be in the set; the other is implicit
- **Degree** of a vertex: number of edges containing that vertex
  - Put another way: the number of adjacent vertices

# Directed Graphs

- In directed graphs, edges have a direction




or

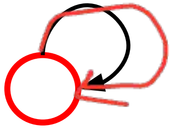



2 edges here

- Thus, `(u,v) ∈ E` does *not* imply `(v,u) ∈ E`.
    - Let `(u,v) ∈ E` mean `u → 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

C:
in 0
out 2

# 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 zero
- A graph does not have to be connected (in an undirected graph, this means we can follow edges from any node to every other node), even if every node has non-zero degree

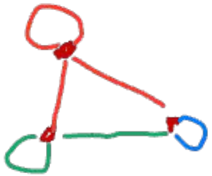
# Some graphs

What are the vertices and what are the edges? Self loops? Directed?

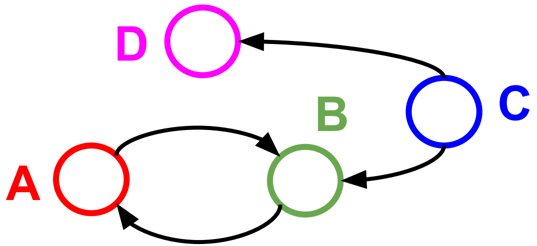
- Web pages with links
- Facebook friends
- “Input data” for the Kevin Bacon game
- Methods in a program that call each other
- Road maps (e.g., Google maps)
- Airline routes
- Family trees
- Course prerequisites
- …

Wow: Using the same algorithms for problems across so many domains sounds like “core computer science and engineering”

# More notation



$3+2+1$




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

- `|V|` is the number of vertices
- `|E|` is the number of edges
  - Minimum? $0$
  - Maximum for undirected? $\frac{v(v+1)}{2} - v$
  - Maximum for directed? $v^2 - v$

$O(v^2)$

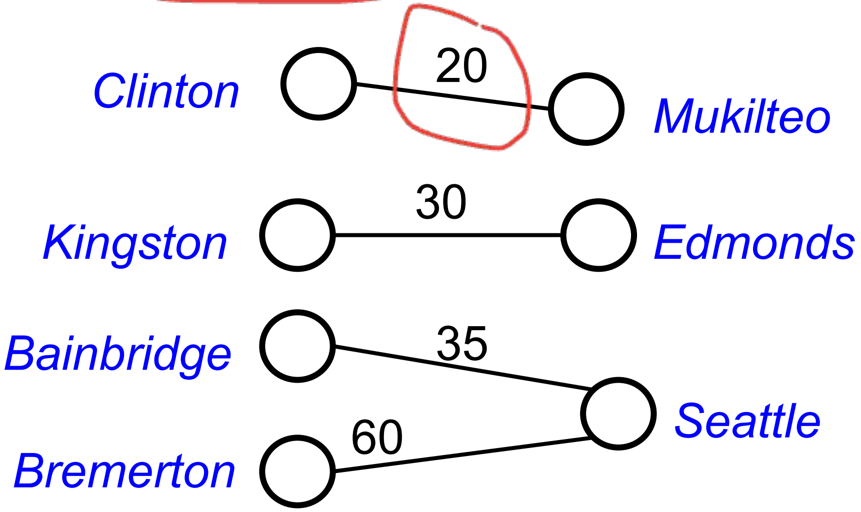
- If `(u,v)` ∈ `E`
  - Then `v` is a neighbor of `u`, i.e., `v` is adjacent to `u`
  - Order matters for directed edges
    - `u` is not adjacent to `v` unless `(v,u)` ∈ `E`

`V = {A, B, C, D}`
`E = {(C, B),`
`(A, B),`
`(B, A),`
`(C, D)}`

# Weighted graphs

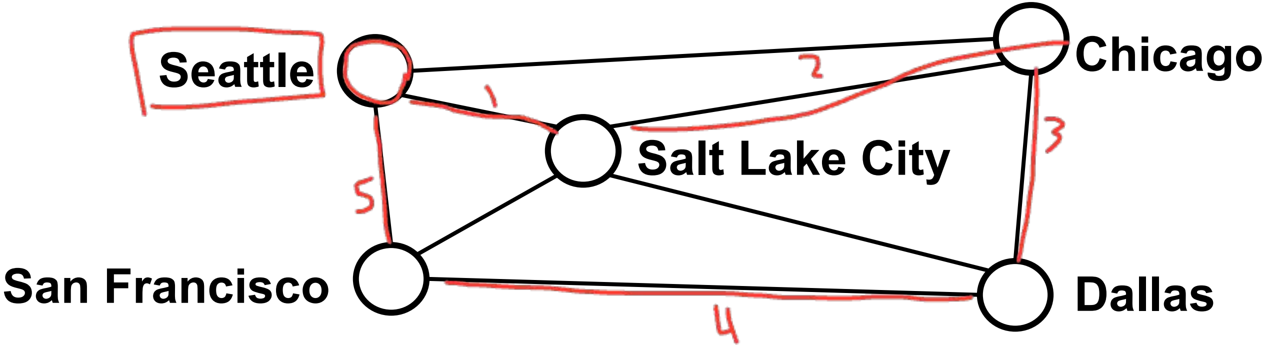
In a weighted graph, each edge has a weight a.k.a. cost

- Typically numeric (most examples will use ints)
- *Orthogonal* to whether graph is directed
- Some graphs allow *negative weights*; many don't



# 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 $0 \leq i < n$. Say "*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 path (that also happens to be a cycle):

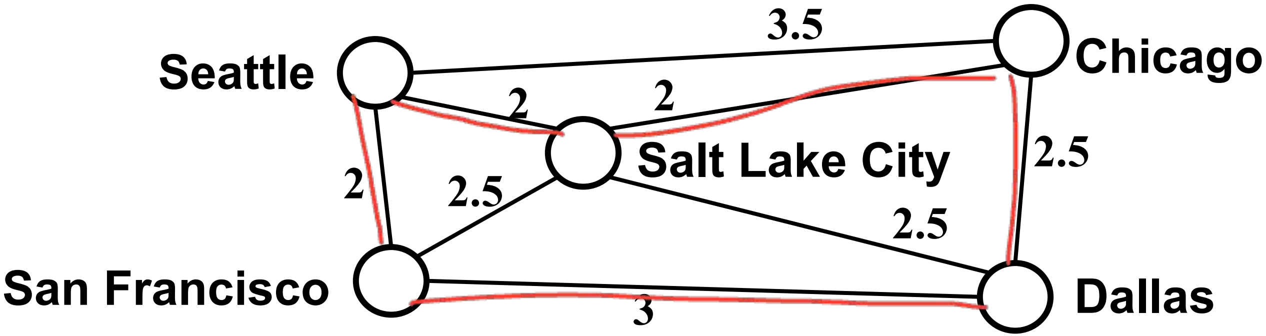
[Seattle, Salt Lake City, Chicago, Dallas, San Francisco, Seattle]

# Path Length and Cost

- Path length: Number of *edges* in a path (also called "unweighted cost")
- Path cost: Sum of the weights of each edge

Example where:

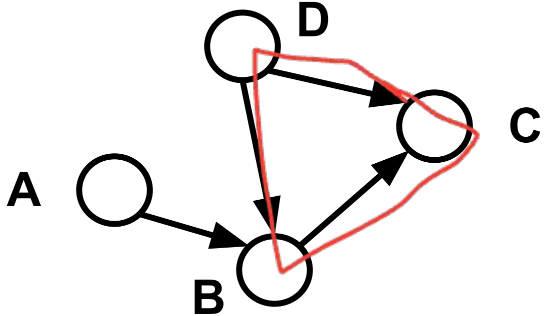
P= [Seattle, Salt Lake City, Chicago, Dallas, San Francisco]




length(P) = 4
cost(P) = 9.5

# Paths/cycles in directed graphs

Example:



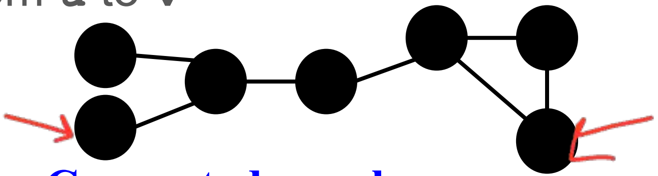

Directed? | What if undirected?

Is there a path from A to D? N | Y

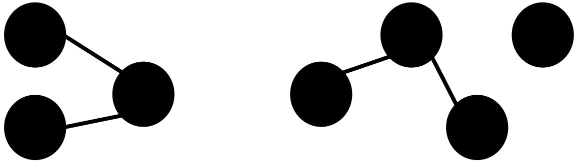
Does the graph contain any cycles? N | Y

# Undirected graph connectivity

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

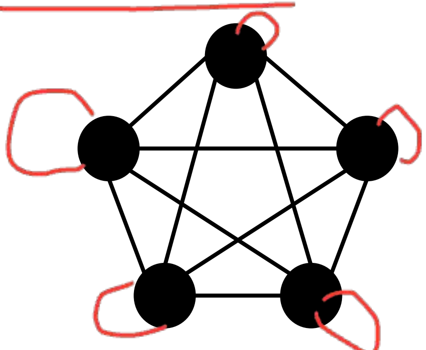


Connected graph



Disconnected graph

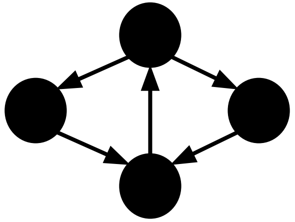
An undirected graph is complete, a.k.a. fully connected if for all pairs of vertices **`u,v`**, there exists an edge from **`u`** to **`v`**



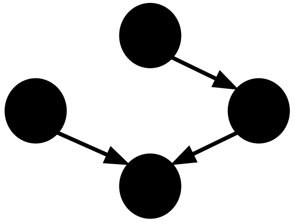
What's missing?

# Directed graph connectivity

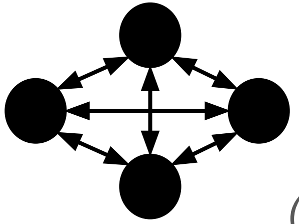
- A directed graph is strongly connected if there is a path from every vertex to every other vertex



- 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



*(plus self edges)*

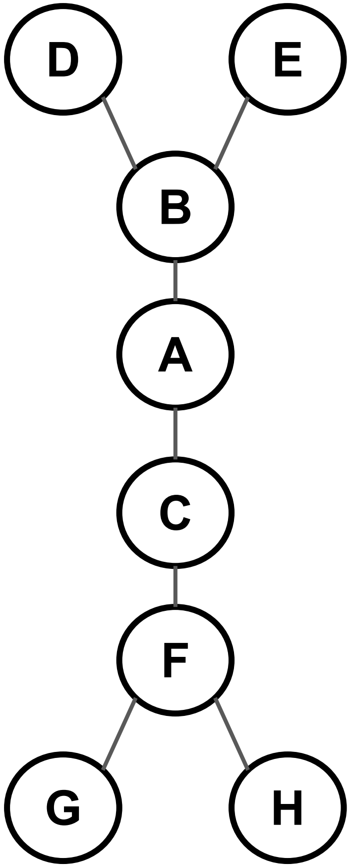
# Trees as graphs

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

- Undirected
- Acyclic
- Connected

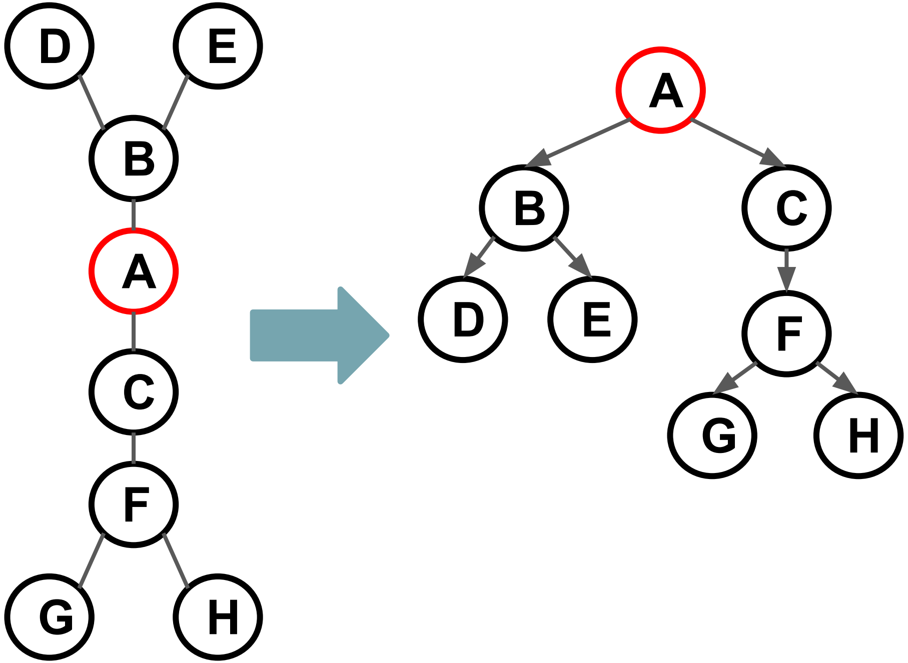
So all trees are graphs, but not all graphs are trees

How does this relate to the trees we know and love?...



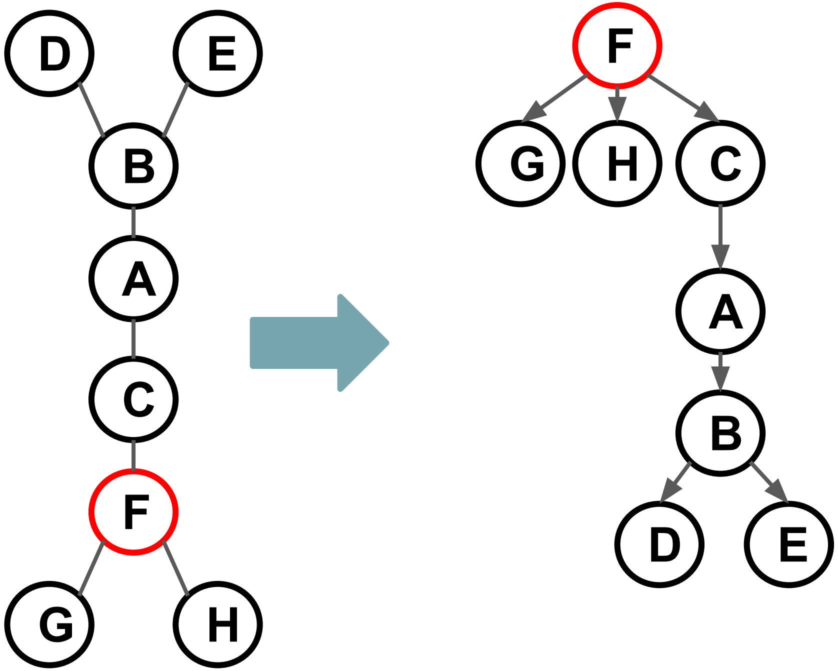
# Rooted Trees

- 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)



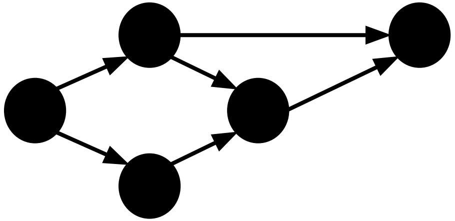
# Rooted Trees (Another example)

- 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)



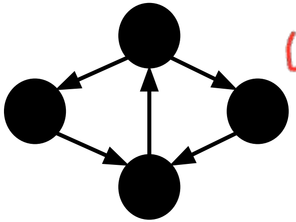
# Directed acyclic graphs (DAGs)

- 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:



Not a rooted directed tree, has a cycle (in the undirected sense)

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



cycle

# Density / sparsity

The range of numbers of edges is really wide

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

- One more fact: If an undirected graph is *connected*, then $|\mathbf{E}| \geq |\mathbf{V}| - 1$

- Because $|\mathbf{E}|$ is often much smaller than its maximum size, we do not always approximate as $|\mathbf{E}|$ as $\mathbf{O}(|\mathbf{V}|^2)$
  - This is a correct bound, it just is often not tight
  - **Dense**: If it is tight, i.e., $|\mathbf{E}|$ is $\mathbf{\Theta}(|\mathbf{V}|^2)$
    - More sloppily, dense means "lots of edges"
  - **Sparse**: $|\mathbf{E}|$ is $\mathbf{O}(|\mathbf{V}|)$
    - More sloppily, sparse means "most (possible) edges missing

# Examples again

Which might be dense? Which might be sparse?

- 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 prerequisites
- …

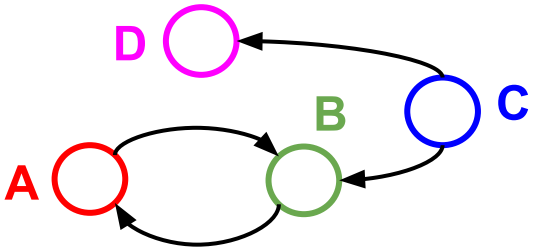
# What is the Data Structure?

- 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

V×V

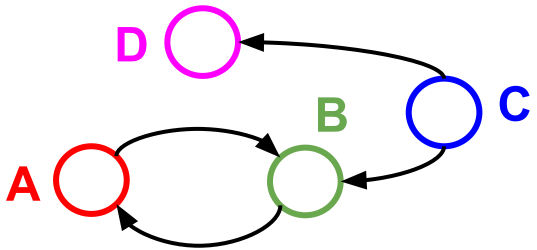
- Assign each node a number from `0` to `|V|-1`
- A `|V|x|V|` matrix (i.e., 2-D array) of Booleans (or 1 vs. 0)
  - If `M` is the matrix, then `M[u][v] == true` means there is an edge from `u` to `v`




| From \ To | 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 in terms of $\boxed{|V|}$ and $\boxed{|E|}$ 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: $O(1)$
- Space requirements: $V^2$
- Best for sparse or dense graphs?

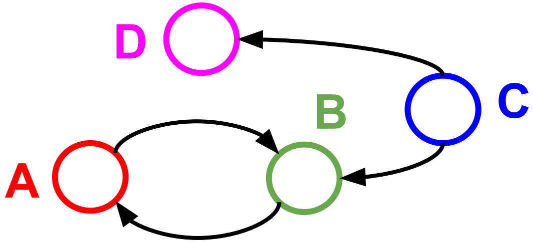


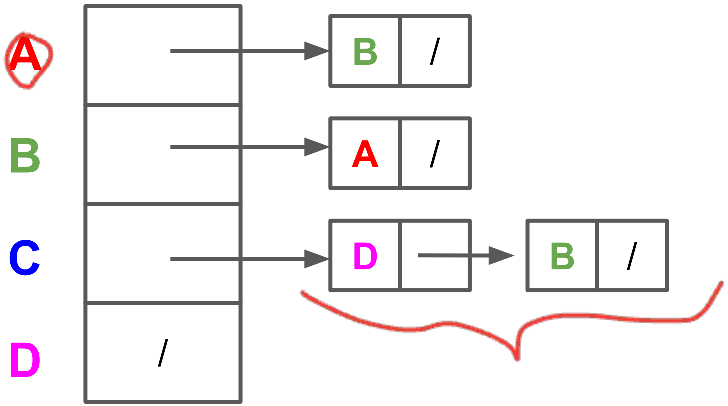

To

| From | A | B | C | D |
|---|---|---|---|---|
| A | F | T | F | F |
| B | T | F | F | F |
| C | F | T | F | T |
| D | F | F | ~~F~~ T | F |

# Adjacency List

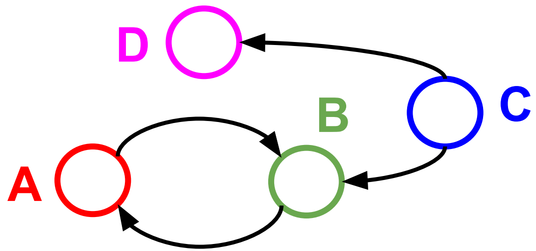
- Assign each node a number from `0` to `|V|-1`
- An array of length `|V|` in which each entry stores a list of all adjacent vertices (e.g., linked list)

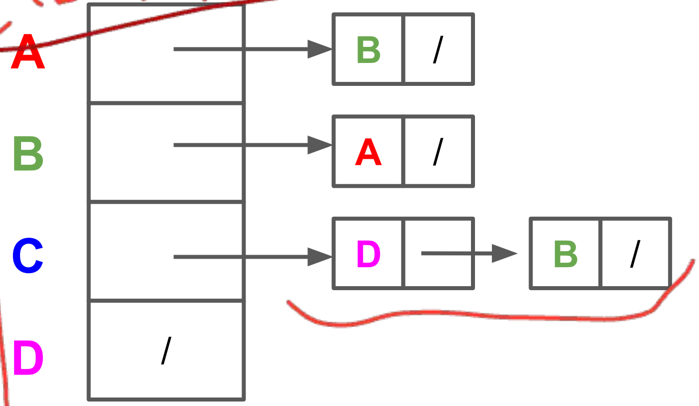


# Adjacency List

- $\boxed{D}$ = “out-degree of a vertex”
- Running time in terms of **|V|, |E|, D** to:
  - Get a vertex’s out-edges: $O(D)$
  - Get a vertex’s in-edges: $O(VD) = O(V+E)$
  - Decide if some edge exists: $O(D)$
  - Insert an edge: $O(D)$
  - Delete an edge: $O(D)$
- Space requirements: $V+E$
- Best for sparse or dense graphs?



# Which is better?

- It depends
- But in reality...
  - ...a lot of problems have sparse graphs...
    - Streets form grids
    - Airlines rarely fly to all possible cities
  - ...so you'll see lots of adjacency lists