Graph Data Structures and Algorithms from Scratch

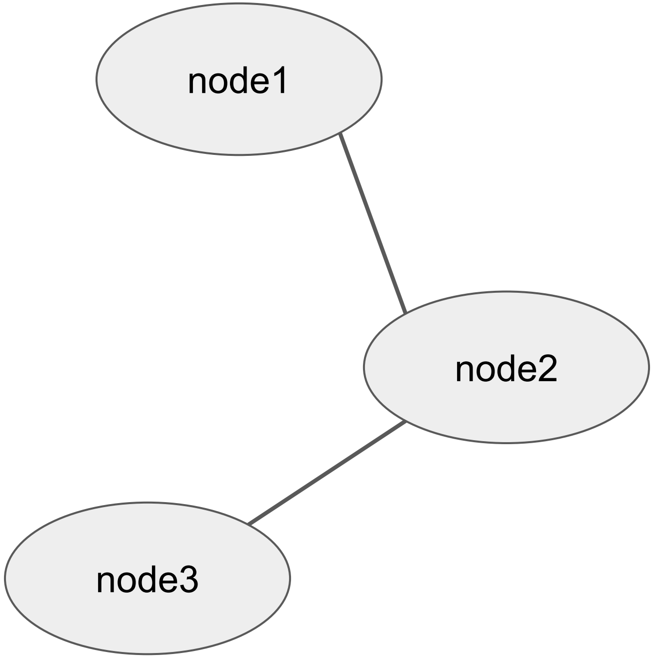
Agenda

- Graph Theory
- Graph Data Structures
- Graph Properties
- Graph Algorithms
- Advanced Graph Algorithms (bonus)
- 5-10 minute break every hour

Graph Theory

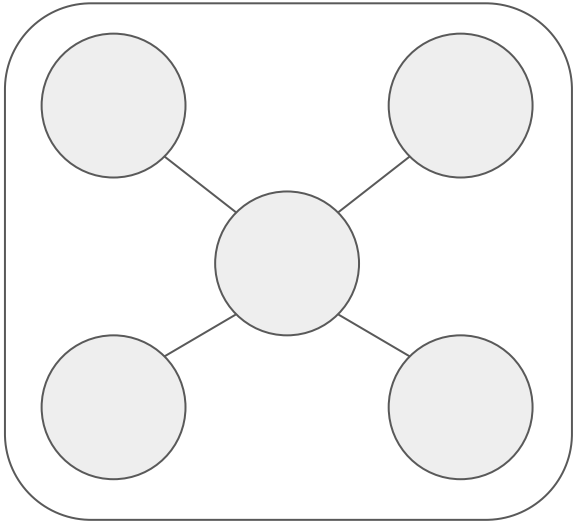
What Is A Graph?

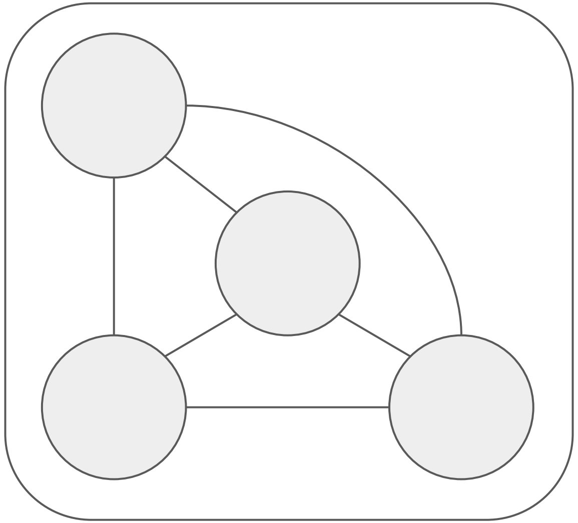
- Nodes/vertices
- Edges

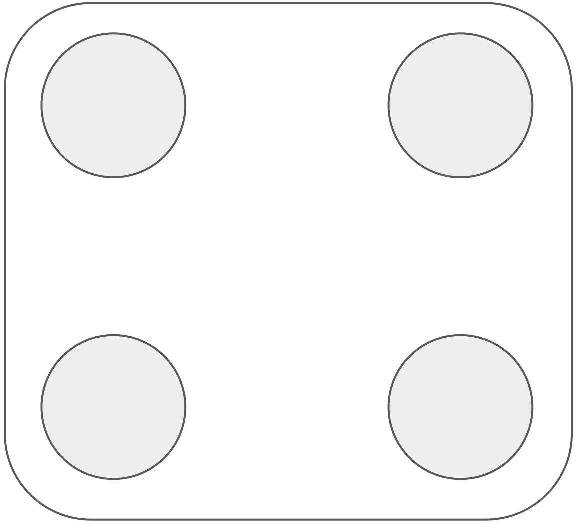


Exercise

How many nodes and edges in each of these graphs?

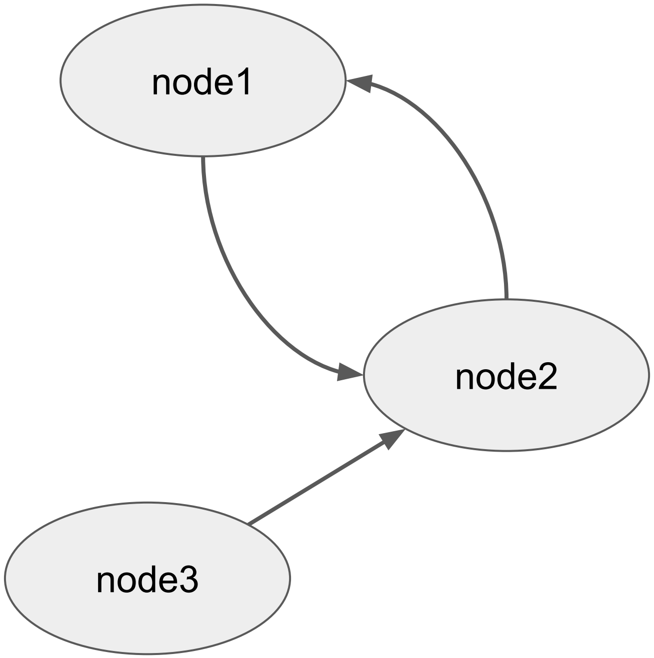






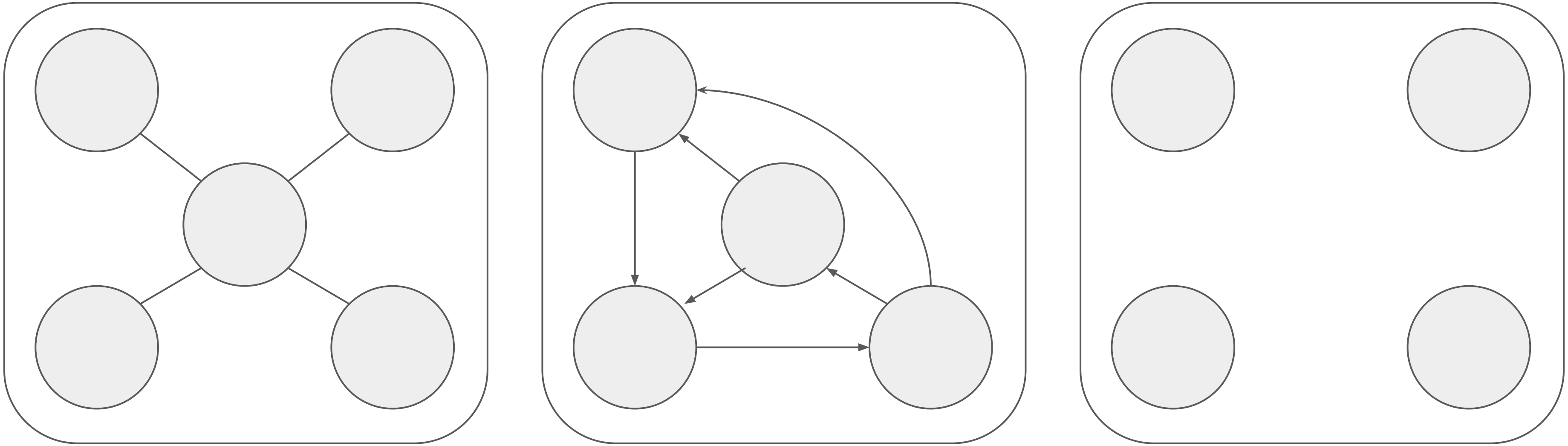
Directed Graphs

- Edges have a direction
- Can go both ways
- Sometimes called “arcs”



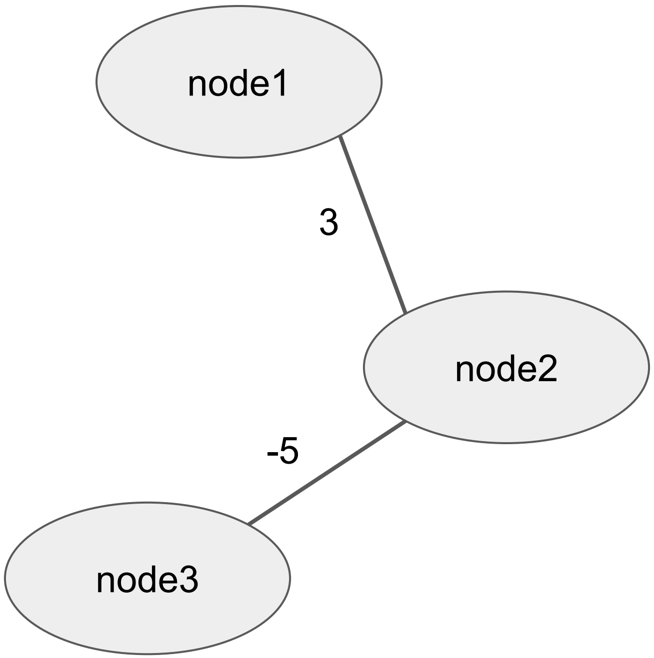
Exercise

Which of these graphs are directed?



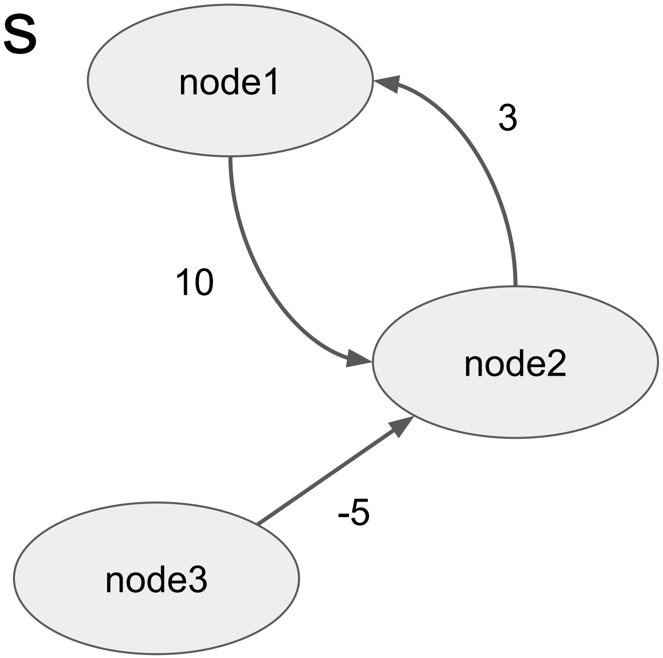
Weighted Graphs

- Each edge has a weight



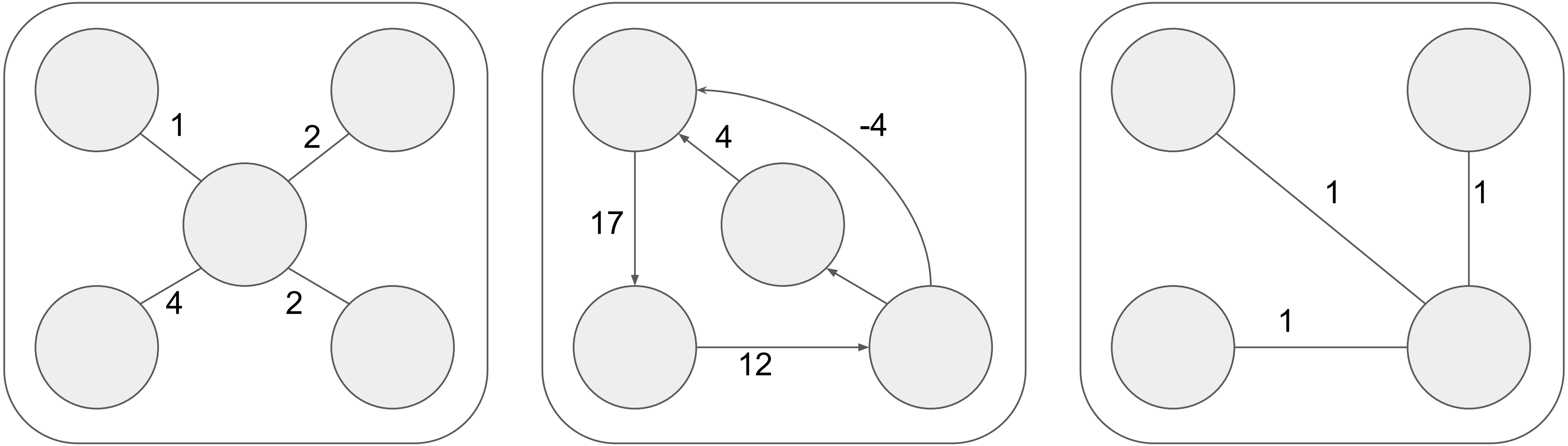
Directed Weighted Graphs

- Both of the above descriptors



Exercise

Which edge in these graphs has the highest weight? Lowest weight?



Naming Conventions

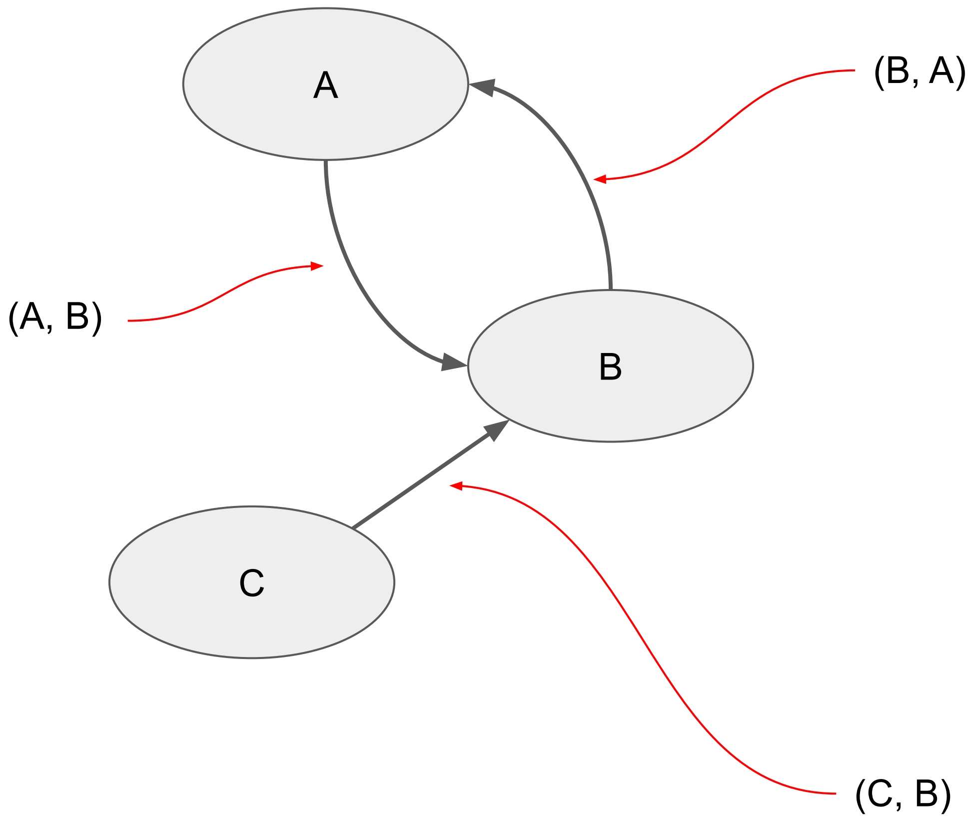
- A Graph G is an ordered pair (V, E)
- V is the set of vertices
- E is the set of edges
  - Edges are described as (u, v)
  - There being an edge from u to v is the same as saying v is a neighbor of u
- $\varnothing$ is the empty set; $(\varnothing, \varnothing)$ is the empty graph

Example

G = {V, E}

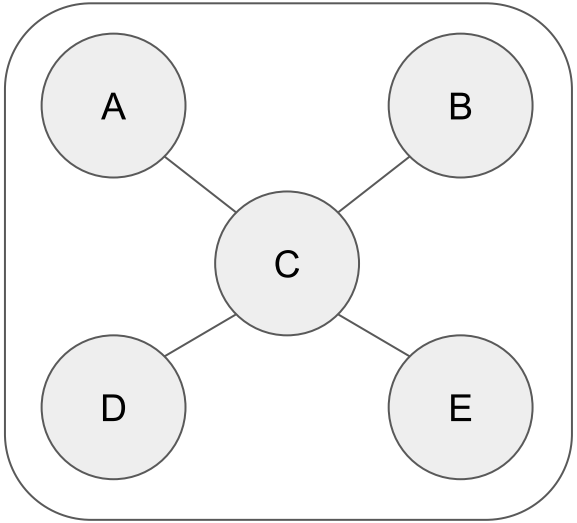
V = {A, B, C}

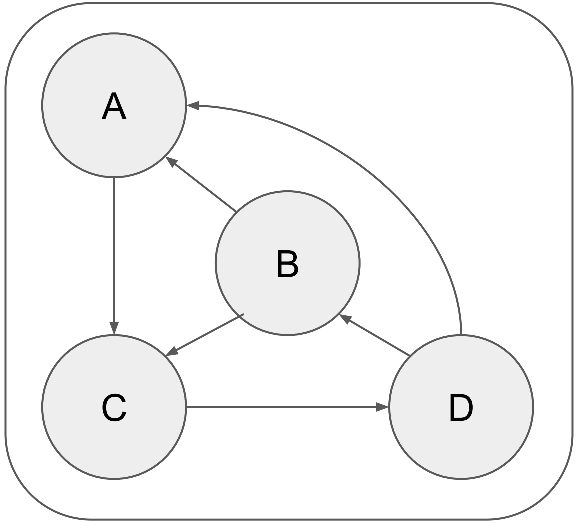
E = { (A,B), (B, A), (C, B) }

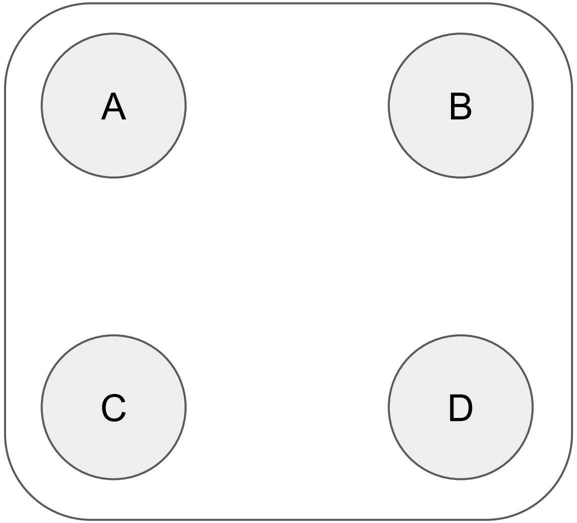


Exercise

Write down these graphs in mathematical notation.





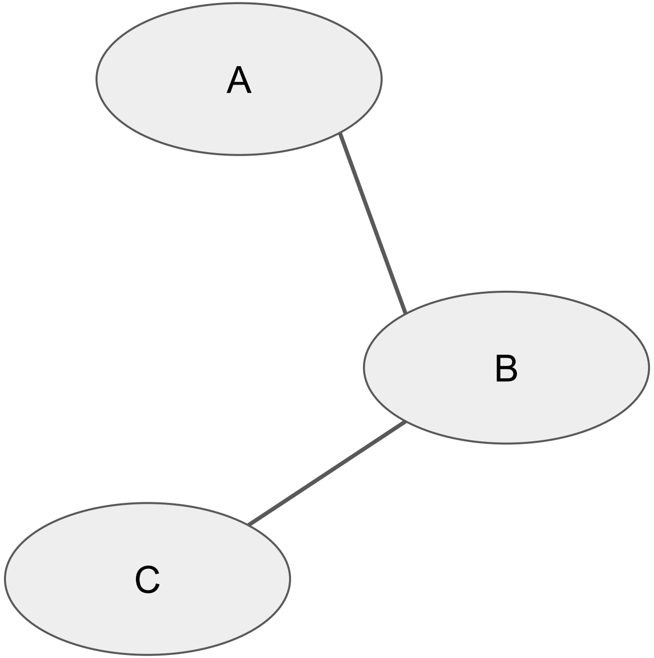


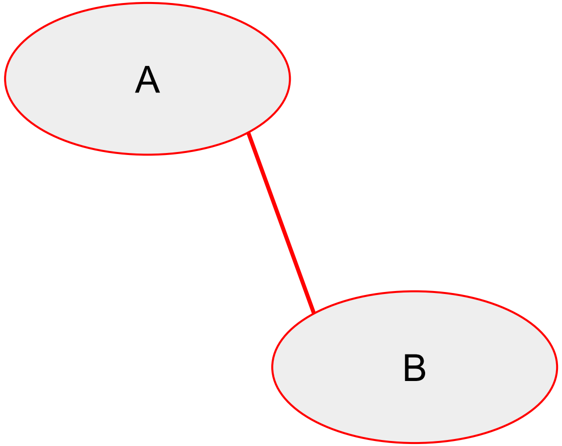
Exercise

If $G = (V, E)$ is an undirected graph, and (u, v) is in E, what else must be in E?

What Is A Subgraph?

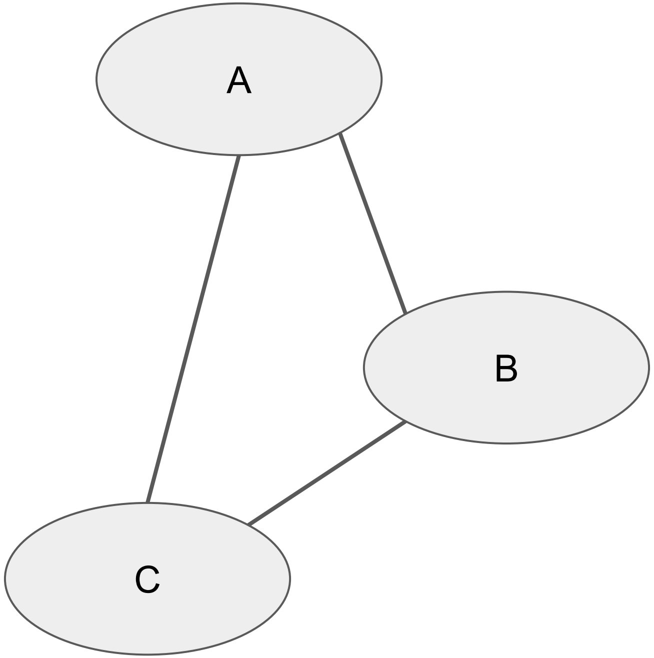
- H = (V', E')
- V' ⊆ V (subset)
- E' ⊆ E (subset)





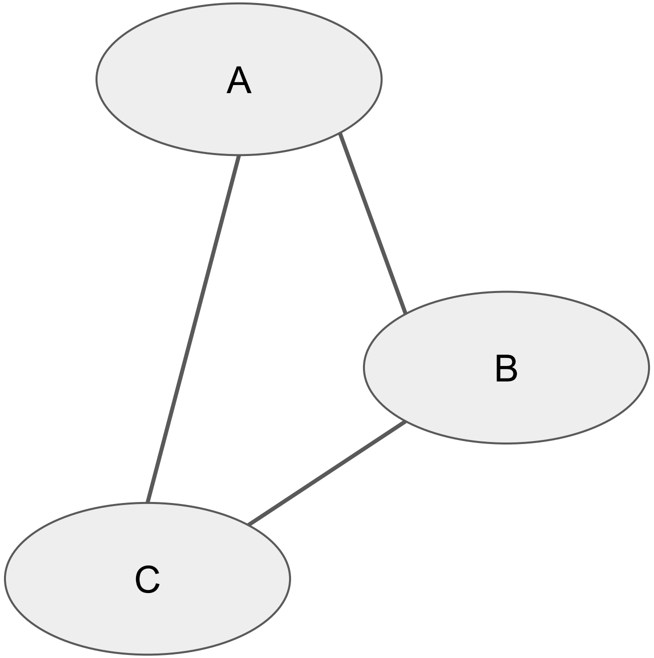
What Is A Path?

- A path from u to v is a subgraph with edges (u, v_0), (v_0, v_1), ..., (v_n, v)
- (A, B)
- (A, B), (B, C)
- (A, B), (B, C), (C, A)



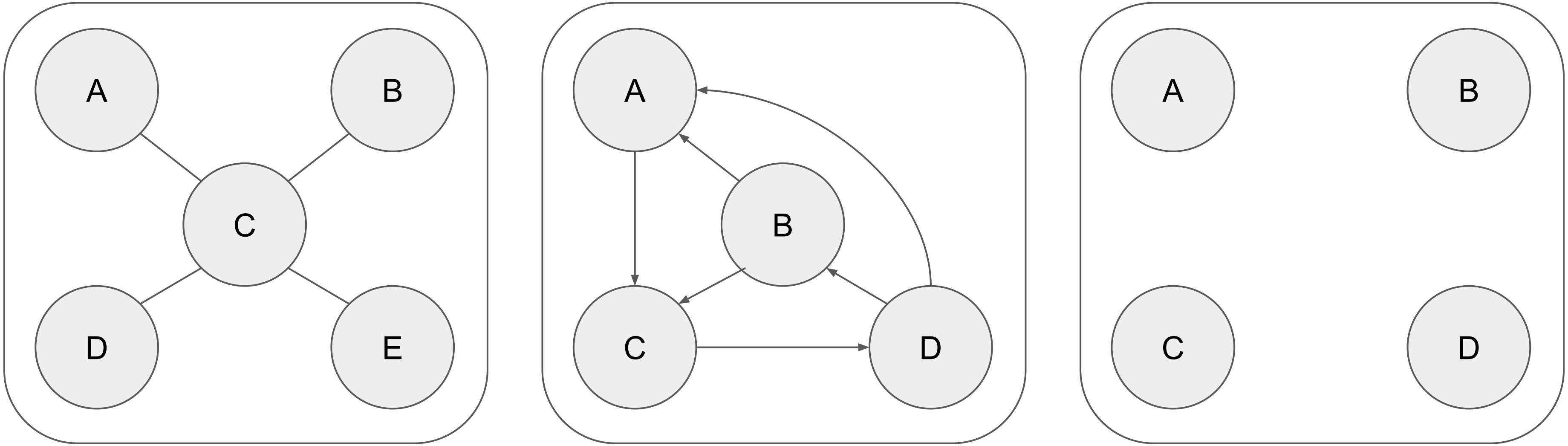
What Is A Cycle?

- A subgraph that has a path from v to v for all v in V
- Directed vs undirected



Exercise

Find a subgraph of these graphs that does not have a cycle

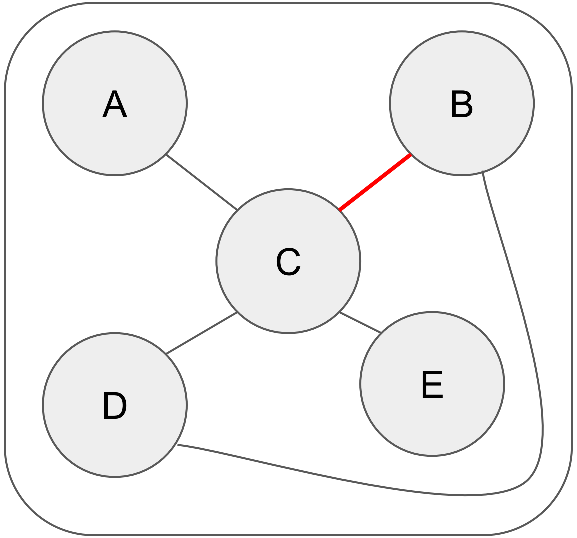


Graph Data Structures

Adjacency Matrix

- Make a |V| by |V| matrix, called A
- For each edge $(v_i, v_j) \in E$, set A[i][j] = 1

Adjacency Matrix Example



	A	B	C	D	E
A	0	0	1	0	0
B	0	0	1	1	0
C	1	1	0	1	1
D	0	1	1	0	0
E	0	0	1	0	0

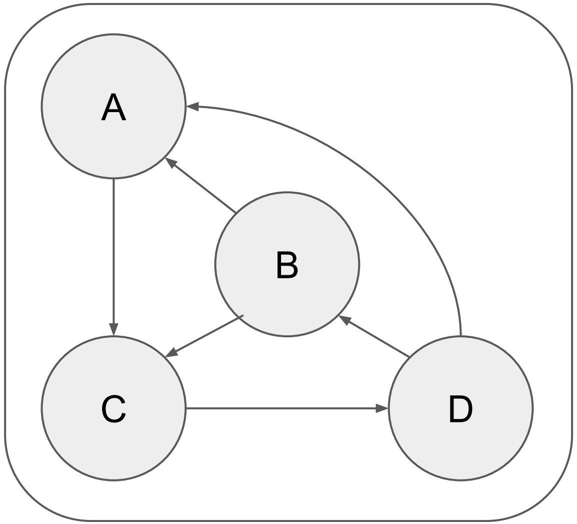
A[C, B] is 1 because C and B are adjacent because (C, B) $\in$ E

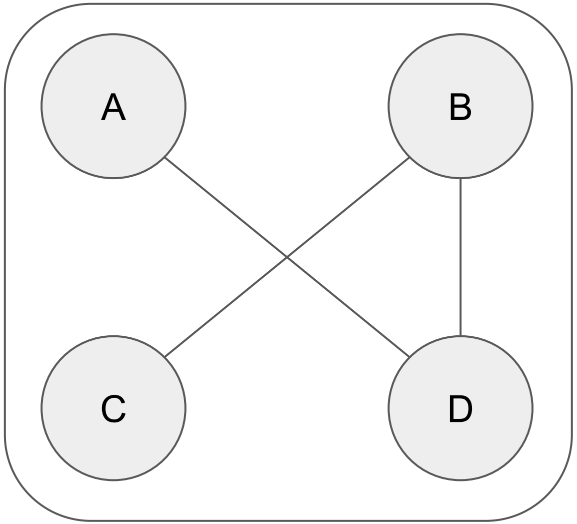
Adjacency Matrix Properties

- Diagonal
- Symmetry
- Matrix operations like exponentiation

Adjacency Matrix Exercise

Find the adjacency matrix for these graphs (NOTE: one is directed!)

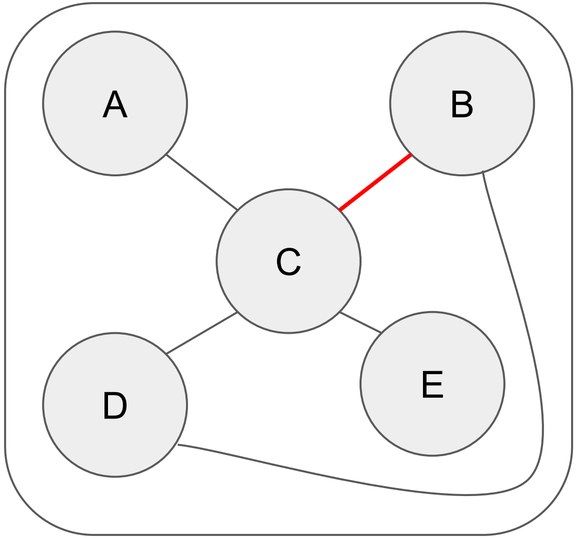




Adjacency List

- Make a list of |V| lists (usually linked lists)
- Add each edge (v_i, v_j) to the list A[j]

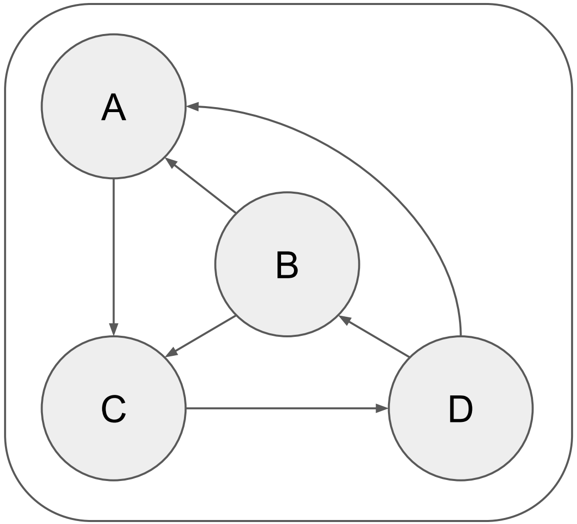
Adjacency List Example

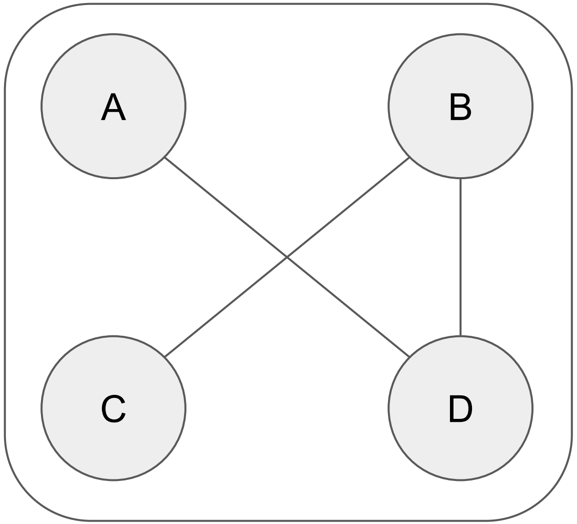



```
A  [C]
B  [C, D]
C  [A, B, D, E]
D  [B, C]
E  [C]
```

Exercise

Find the adjacency list for these graphs (NOTE: one is directed!)





Discussion

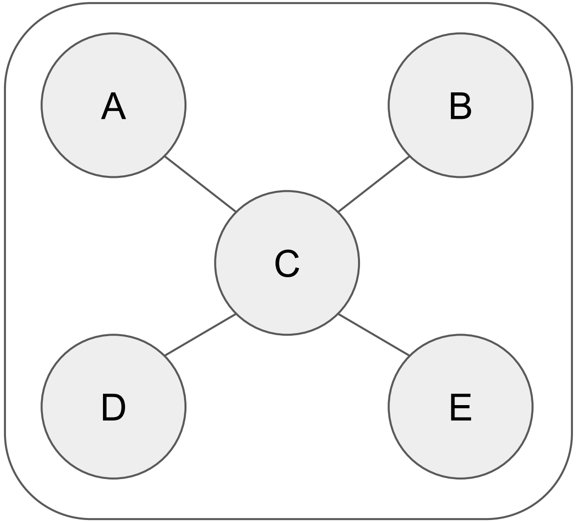
- Space complexity
- Time complexity
- Ease of use

Programming Exercise

Graph Properties

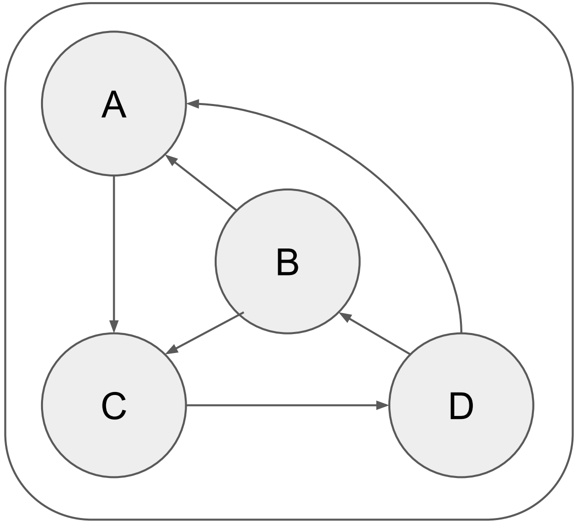
Degree

- The degree of a vertex is the number of edges containing it



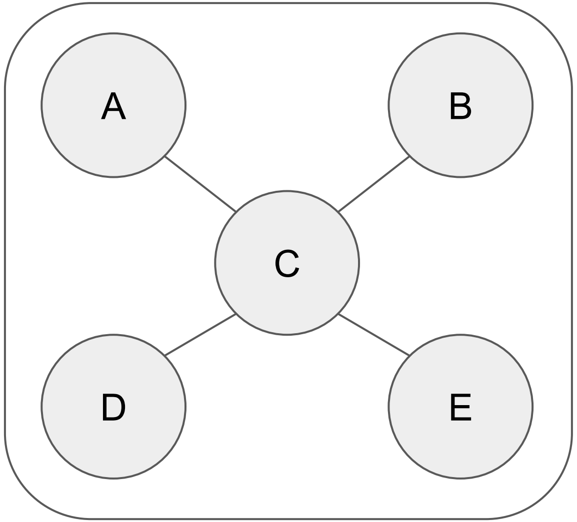
Degree

- In a directed graph *in-degree* is the number of edges coming in, and *out-degree* is the number of edges going out



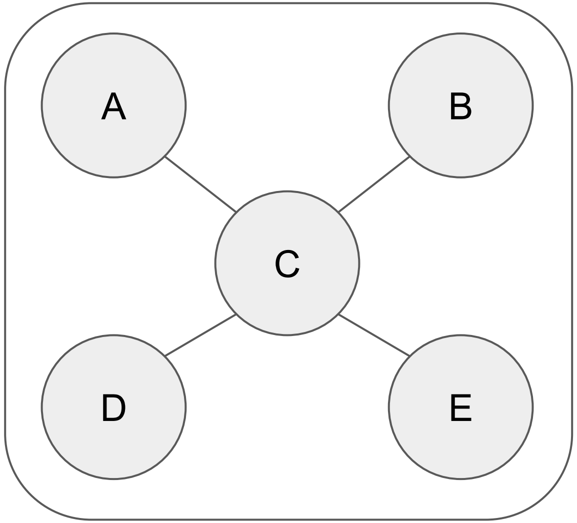
Degree

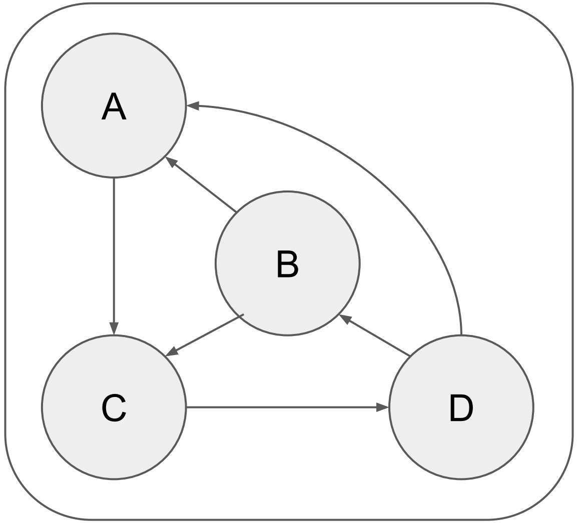
- The minimum and maximum degree of a graph are the degree of the node with the smallest or largest degree, respectively



Exercise

Find the degree of each node. For the directed graph, find in- and out-degree.

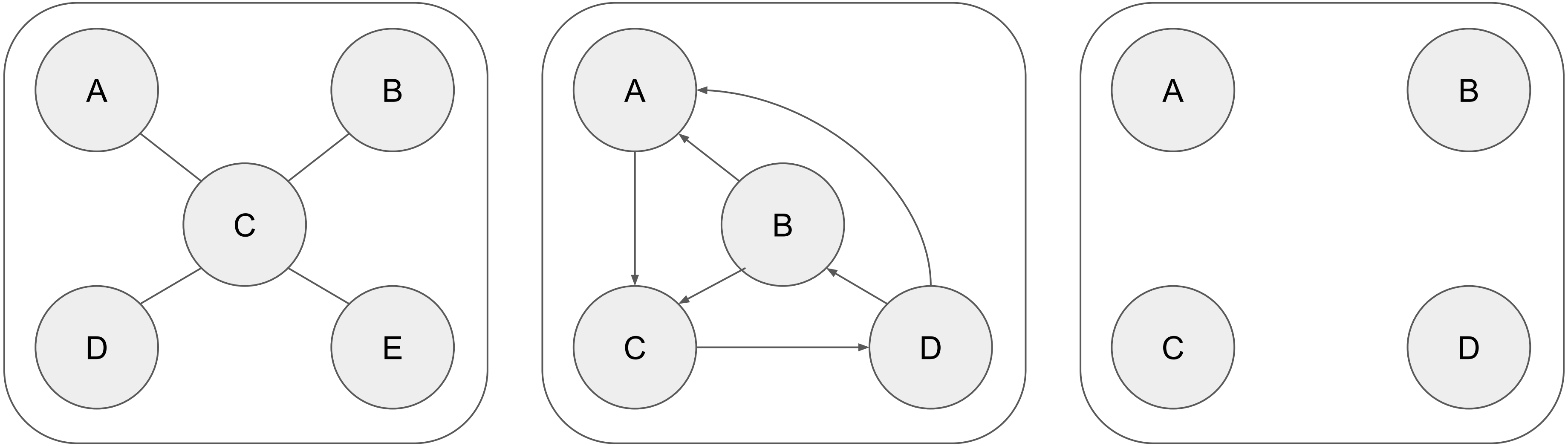




Programming Exercise

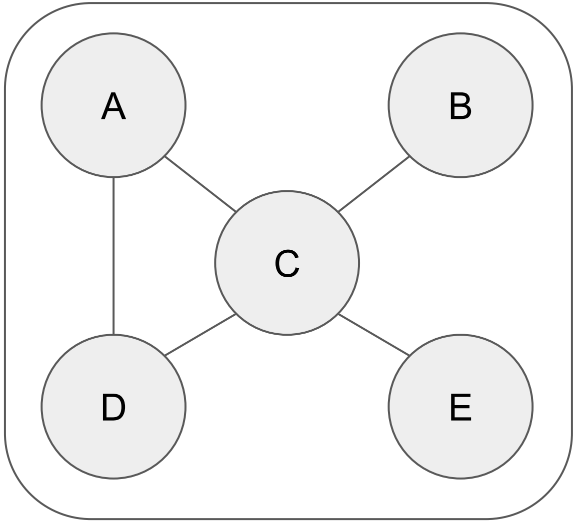
Exercise

Find the degree of each of these nodes.



Connectedness

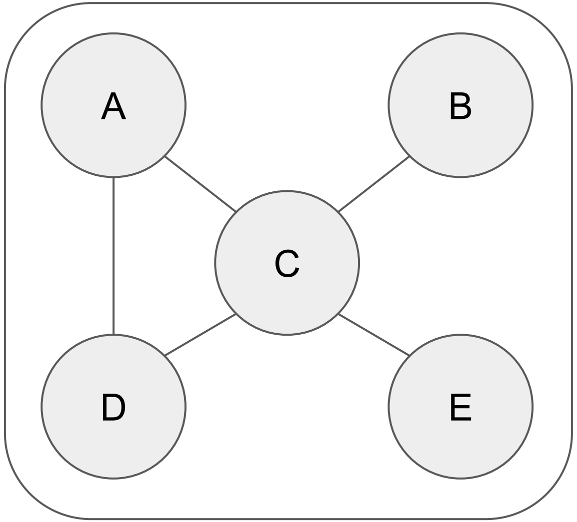
- A graph is connected if there is a path from every vertex to every other vertex.



Programming Exercise

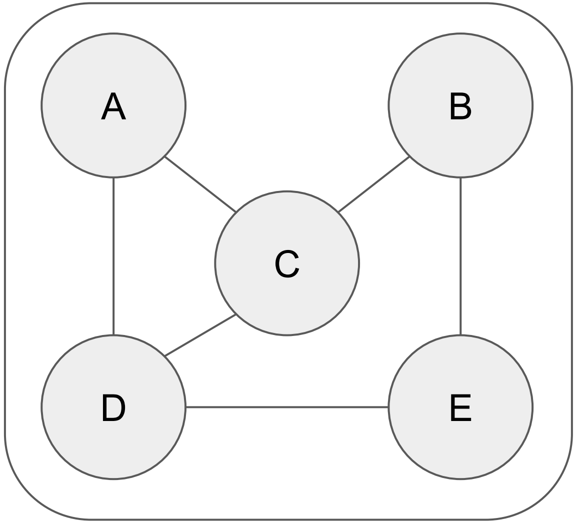
Cut edge

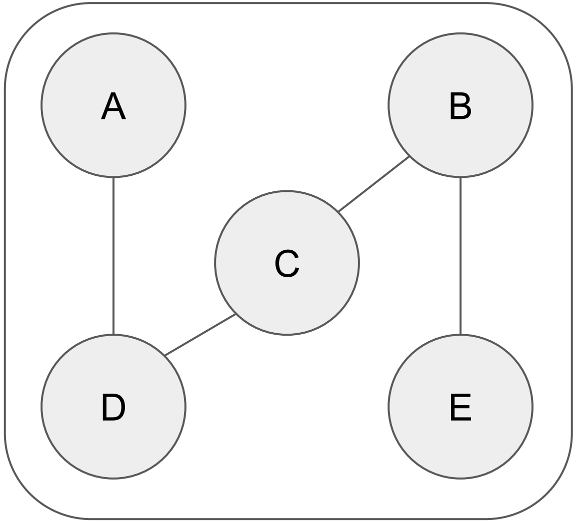
- An edge that, if removed, disconnects the graph.
- An edge that is not in a cycle.

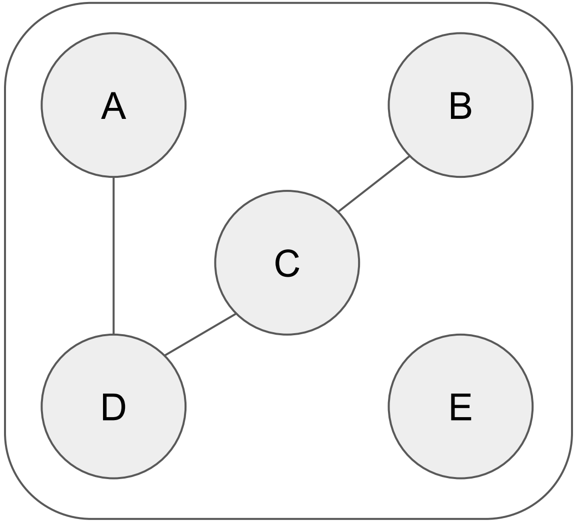


Exercise

- Are these graphs connected?
- Which edges are cut edges?

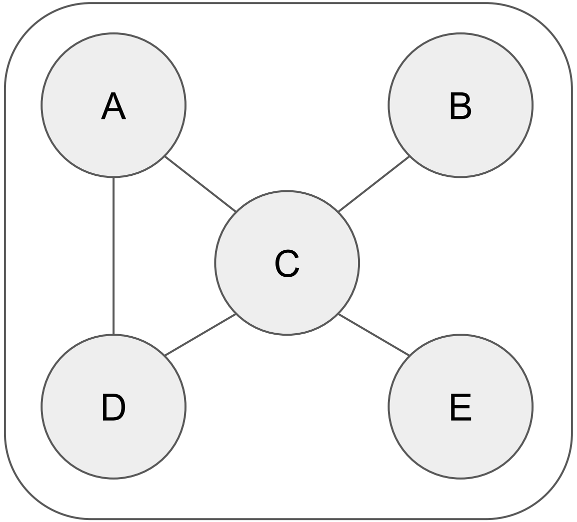






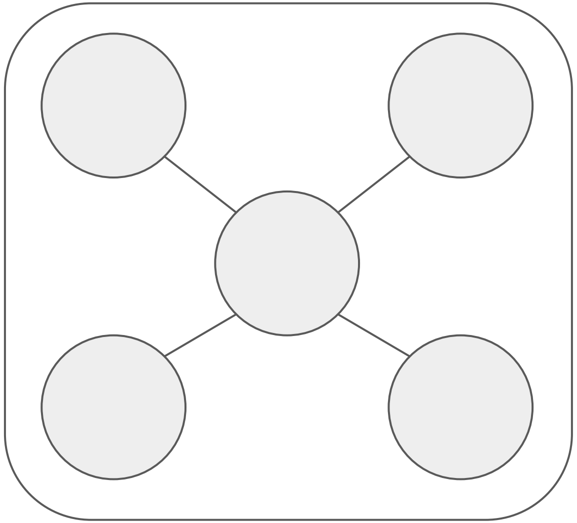
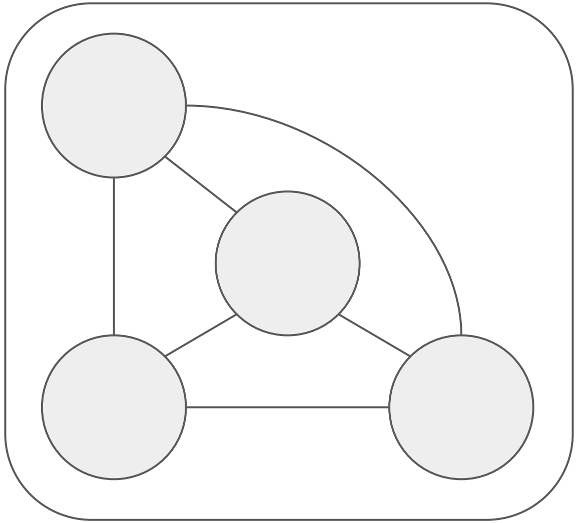
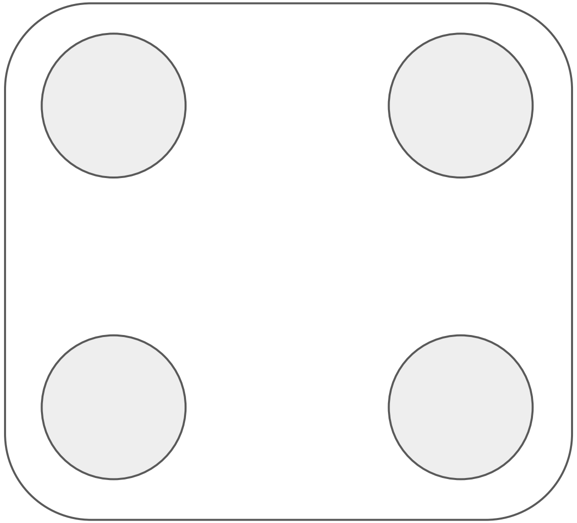
Completeness

- A graph is complete if there is an edge from every vertex to every vertex



Exercise

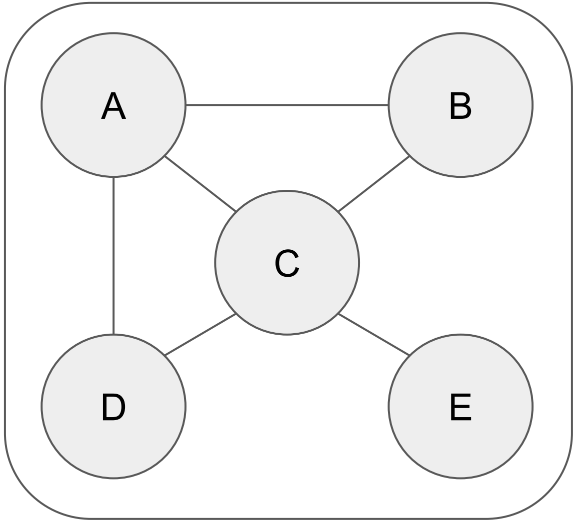
Which of these graphs are complete graphs?

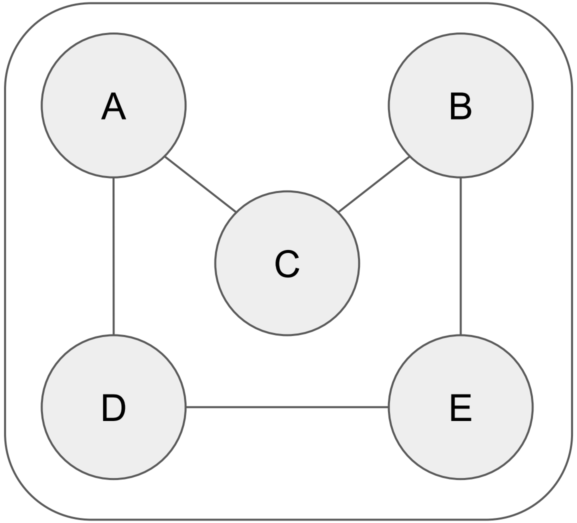




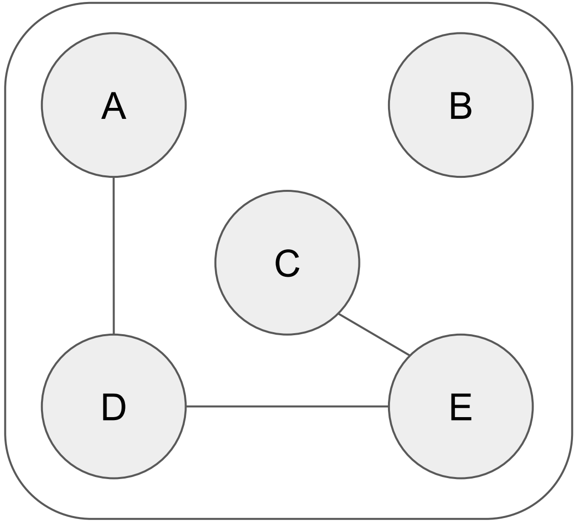
Programming Exercise

Cyclic

- A graph is cyclic if it has a cycle in it





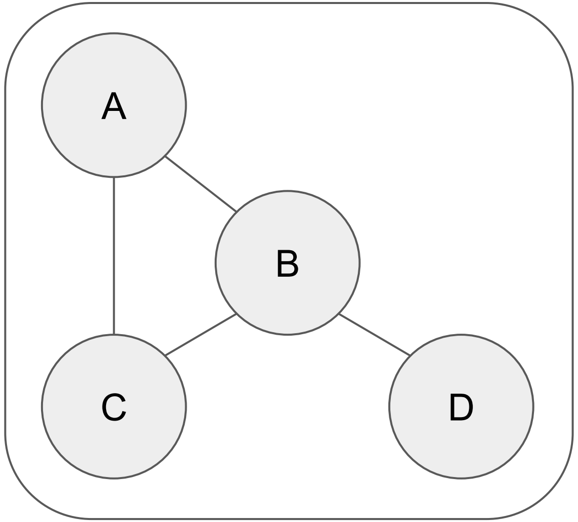


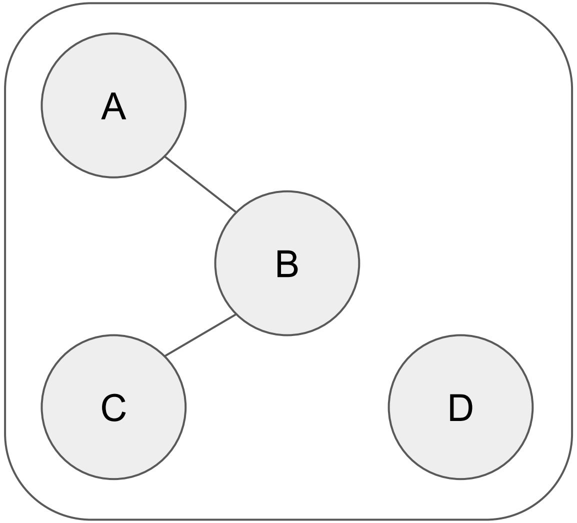
Exercise

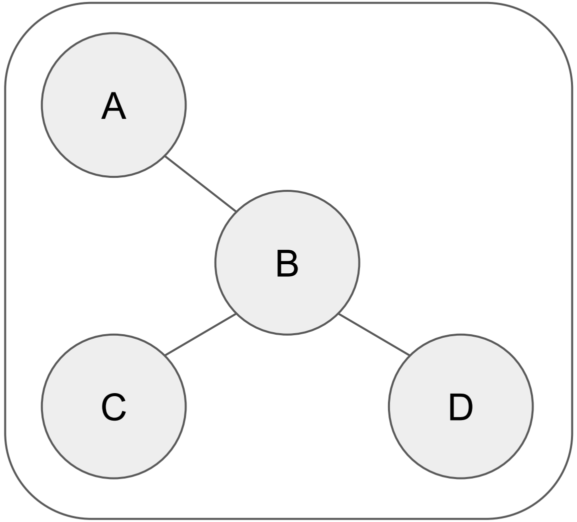
- Can a connected graph be acyclic?
- Can a complete graph be cyclic?
- Can an undirected graph where each node has degree 2 be acyclic?

Trees

- A graph is a tree if it has no cycles and is connected







Discussion

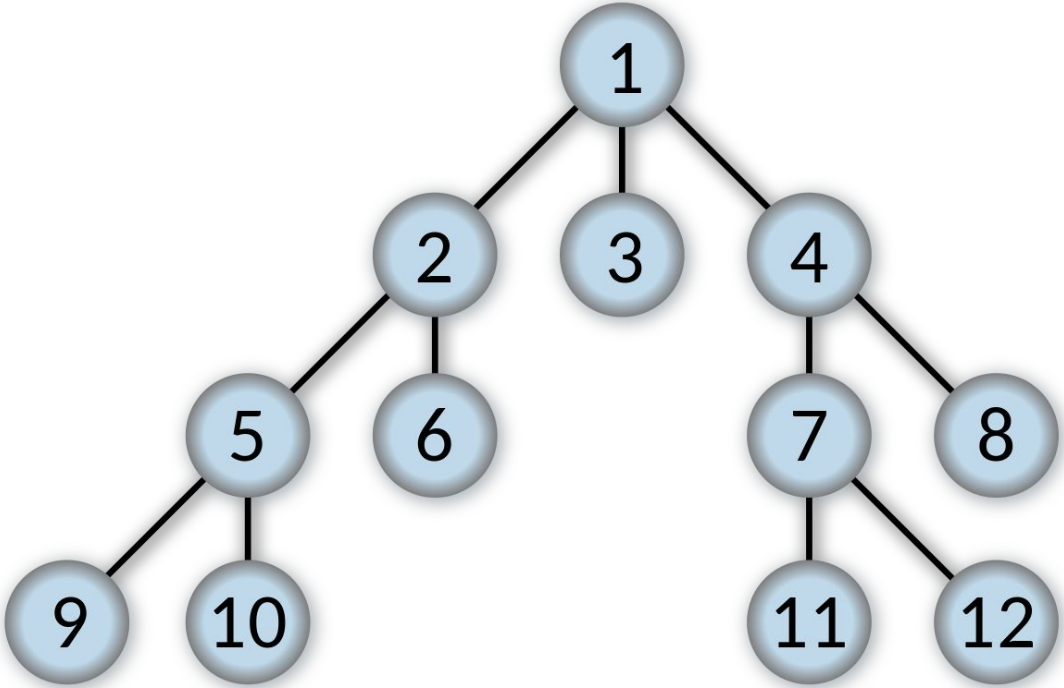
- Cut-edges represent a place where you can stop the spread of a disease
- Cut-edges represent a place where one power line going down takes out power
- In-degree was used for search result rankings
- High degree in an infection graph can represent a super-spreader
- Degree of social networks tells you how connected people are
- Out-degree of dependency graph shows number of dependencies
- When else are these things useful to use?

Graph Algorithms

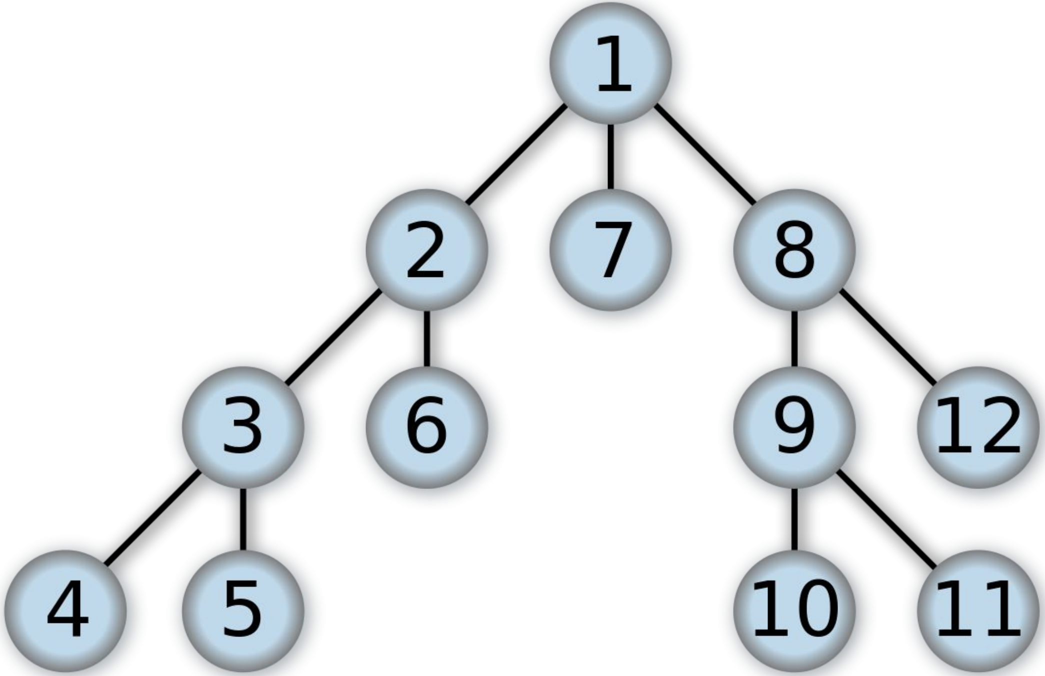
Search Algorithm Uses

- Finding files
- Navigation
- Currency conversion

Breadth-first Search



Depth-first Search

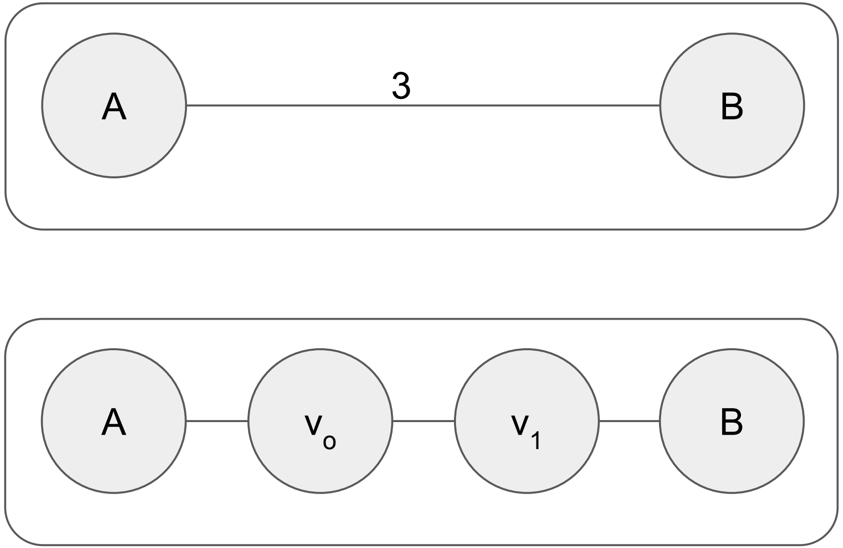


Difference

- BFS will always find the shortest path

BFS on weighted graphs

- “Shortest path” means something different
- Can convert weighted graphs to non-weighted
- Dijkstra’s algorithm

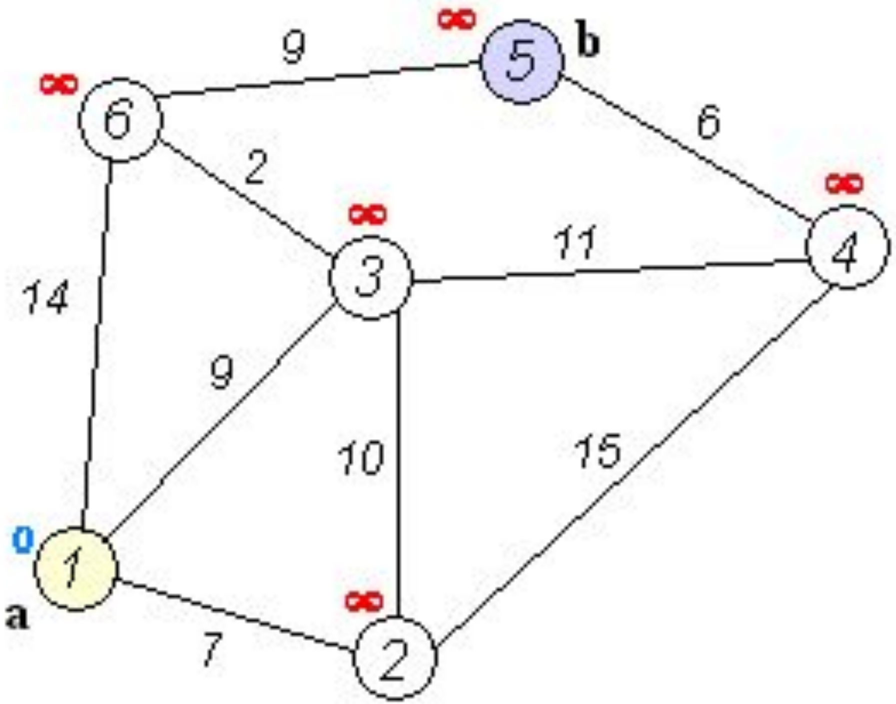


These two graphs are equivalent!

Dijkstra’s algorithm

- Keep track of shortest known distance to each node
- From the node with the shortest distance from the start point, re-calculate everything next to it
    - If this node has a shorter path, remember that
- Once you have found the target, trace the shortest path

Dijkstra’s algorithm

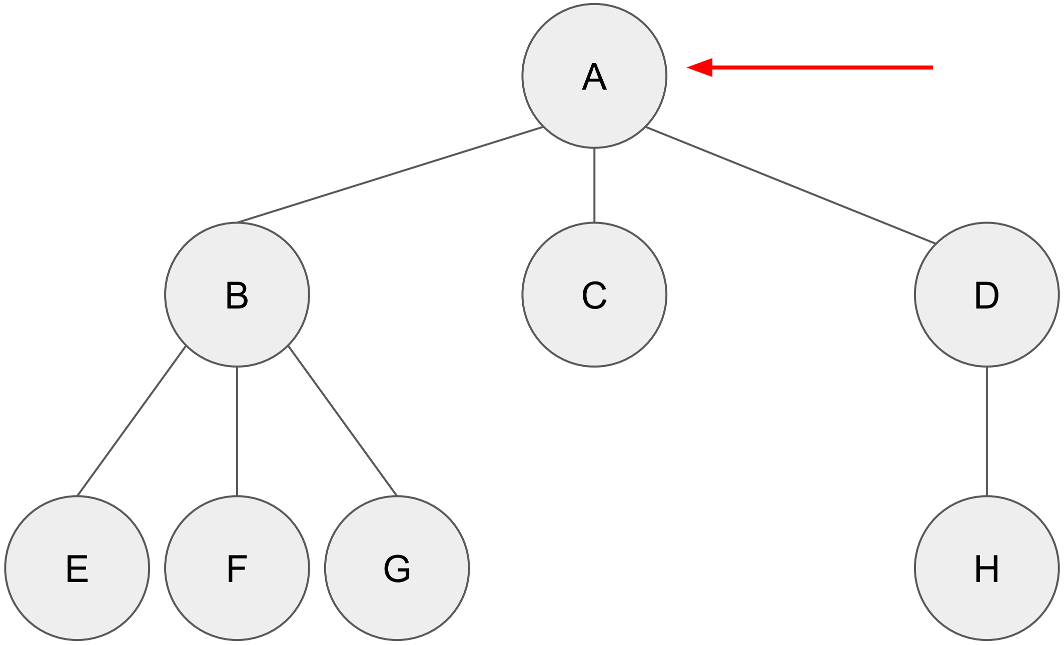


Generic search

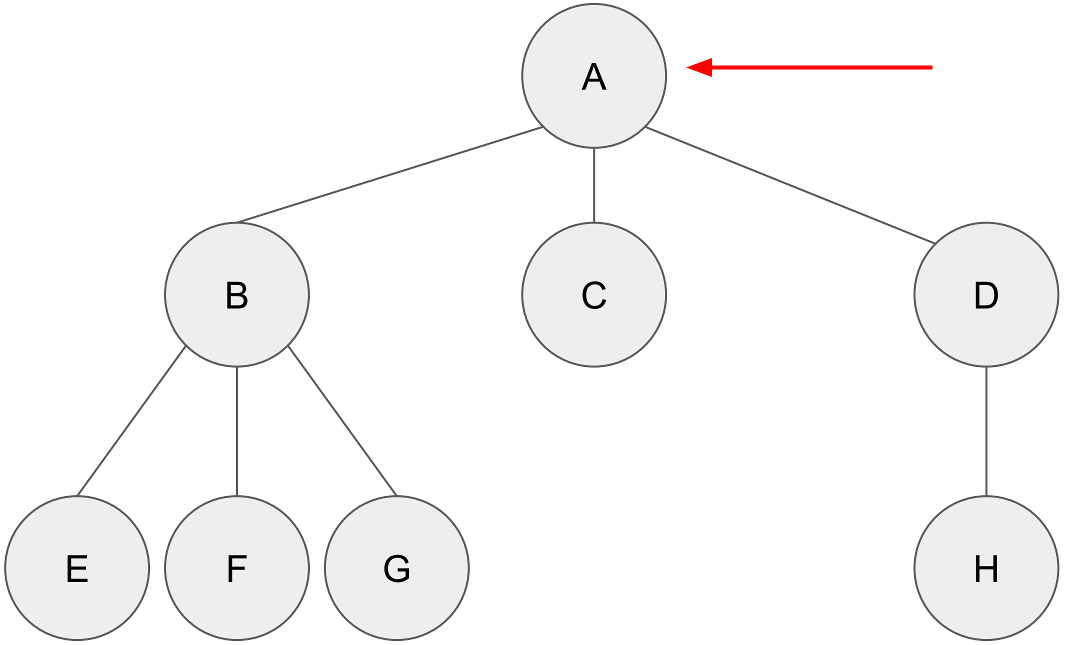
```
def search(graph, source, target):
    visited = set()
    to_visit = ToVisitDataStructure(source)
    while to_visit:
        current = get_next(to_visit)
        if current == target:
            return True
        if current not in visited:
            visited.append(current)
            neighbours = graph[current]
            for neighbour in neighbours:
                add_node(to_visit, neighbour)
    return False
```

Breadth First Search

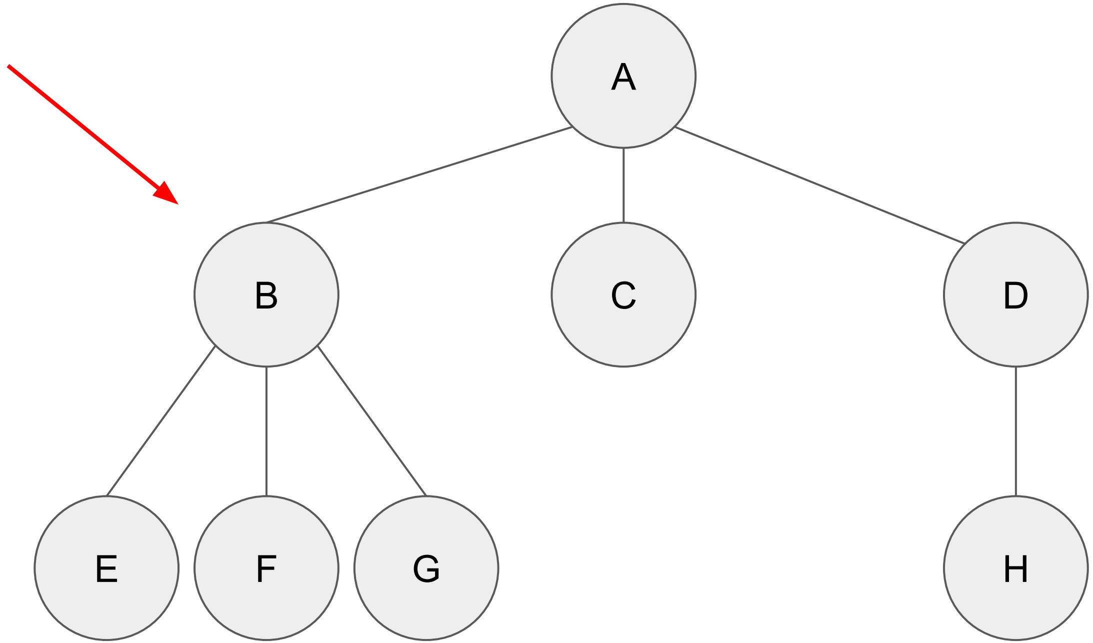
- Add to end, pop from front
- Queue



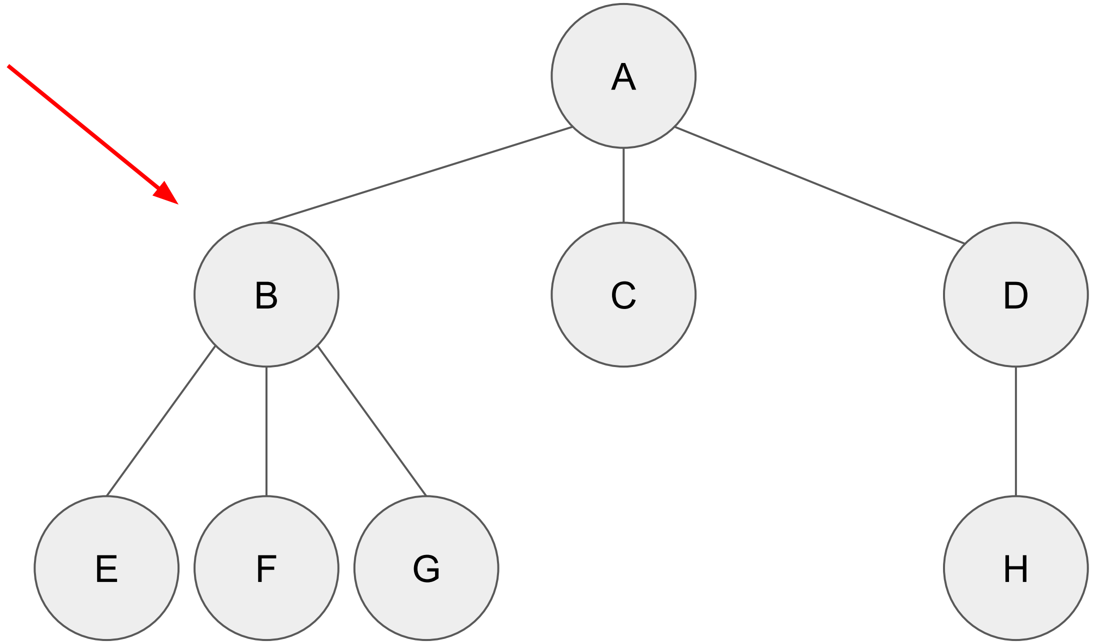

```
current  = A
to_visit = {}
```



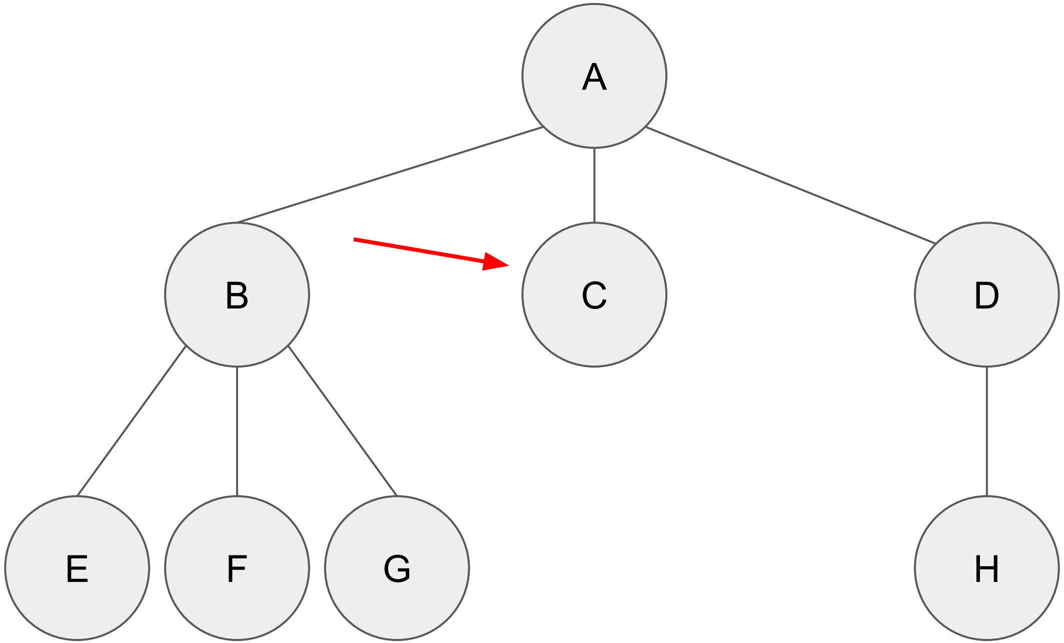

```
current  = A
to_visit = {B, C, D}
```




```
current  = B
to_visit = {C, D}
```



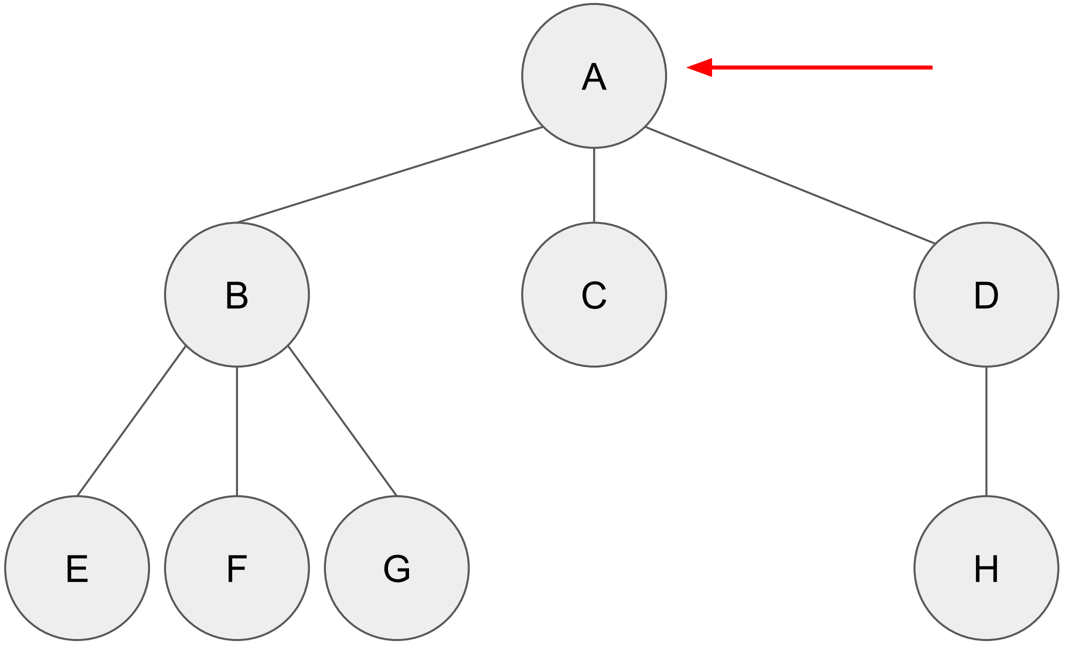

```
current  = B
to_visit = {C, D, E, F, G}
```



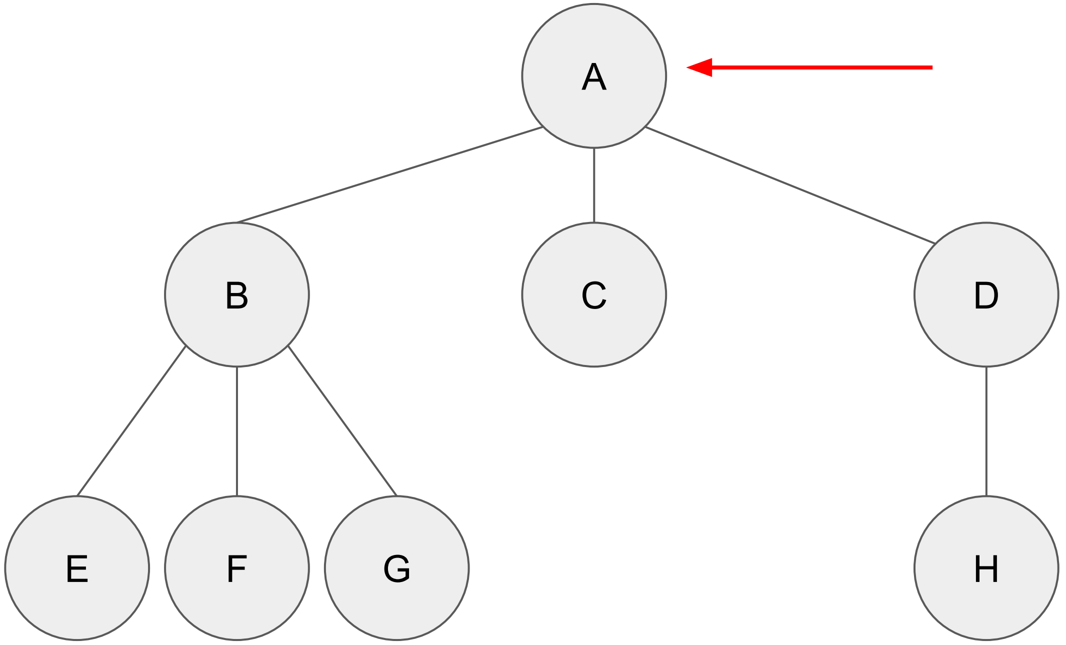

```
current  = C
to_visit = {D, E, F, G}
```

Depth First Search

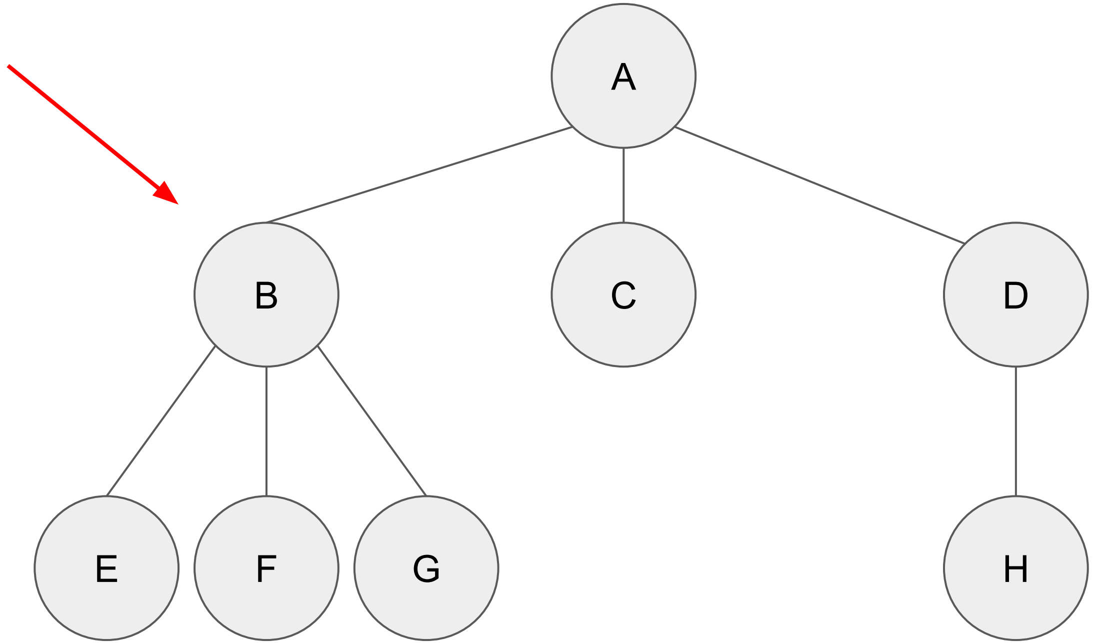
- Add to front, pop from front
- Stack



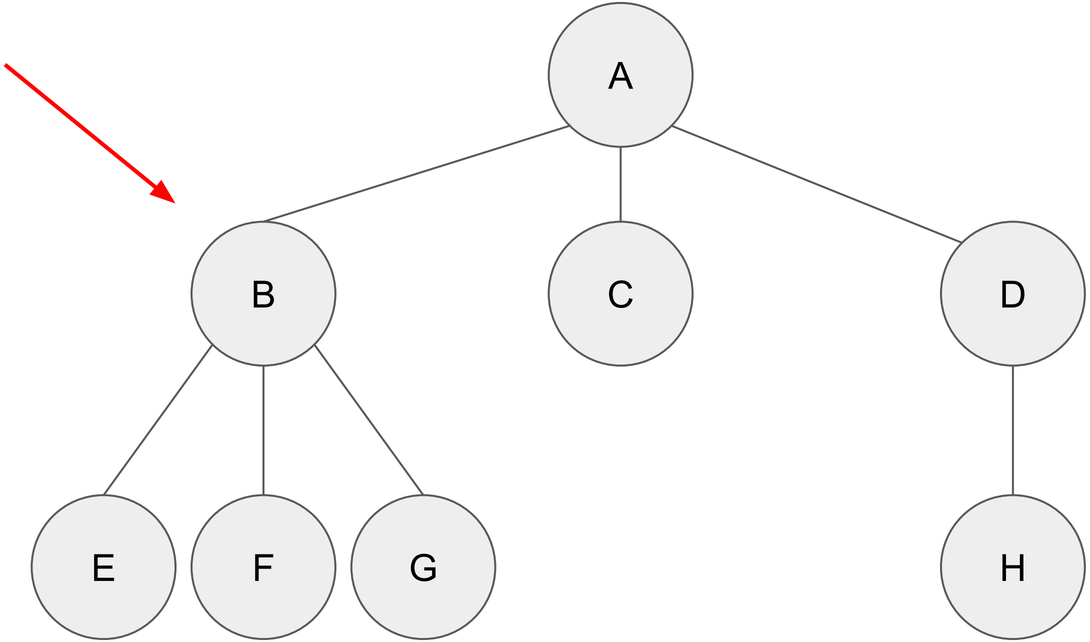

```
current  = A
to_visit = {}
```



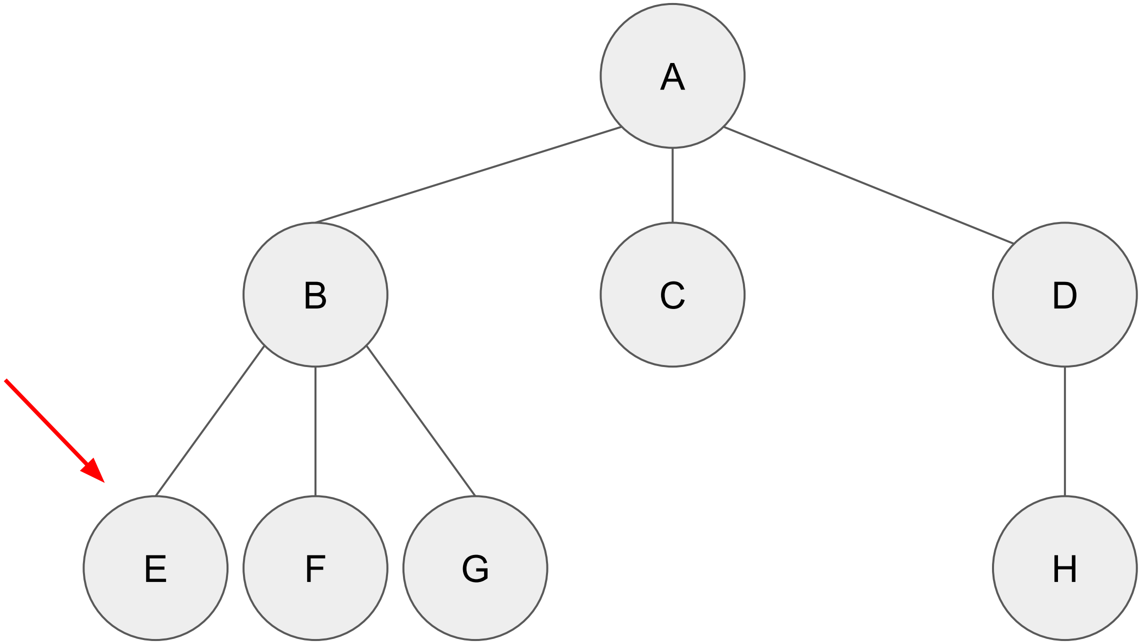

```
current  = A
to_visit = {B, C, D}
```




```
current  = B
to_visit = {C, D}
```




```
current  = B
to_visit = {E, F, G, C, D}
```




```
current  = E
to_visit = {F, G, C, D}
```

Programming Exercise

NetworkX

Creating graphs

```
import networkx as nx

G = nx.Graph()

G.add_node(1)

G.add_nodes_from([2, 3])

G.add_edge(1, 2)
```

Properties

```
>>> G.degree[1]

1

>>> G.degree[2] # the graph is undirected

1

>>> G.number_of_nodes()

3
```

Programming Exercise

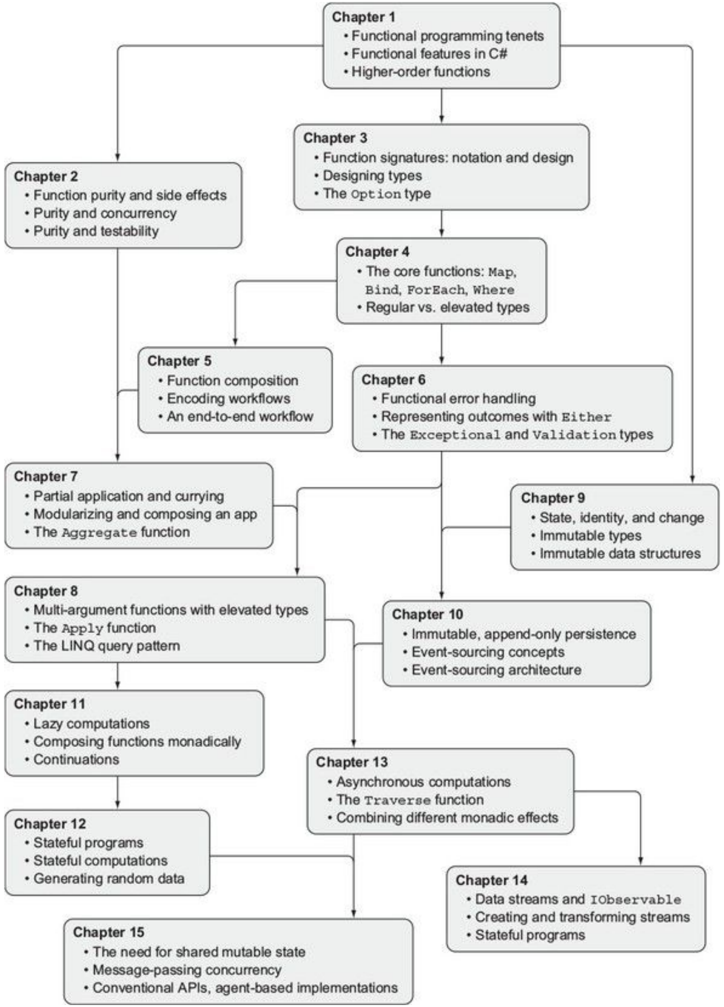
Advanced Graph Algorithm

Topological Sort

- Creates an ordering of the vertices in a directed graph
- Before a vertex appears all of its ancestors
    - A node v’s ancestor is any node that appears before v in any path

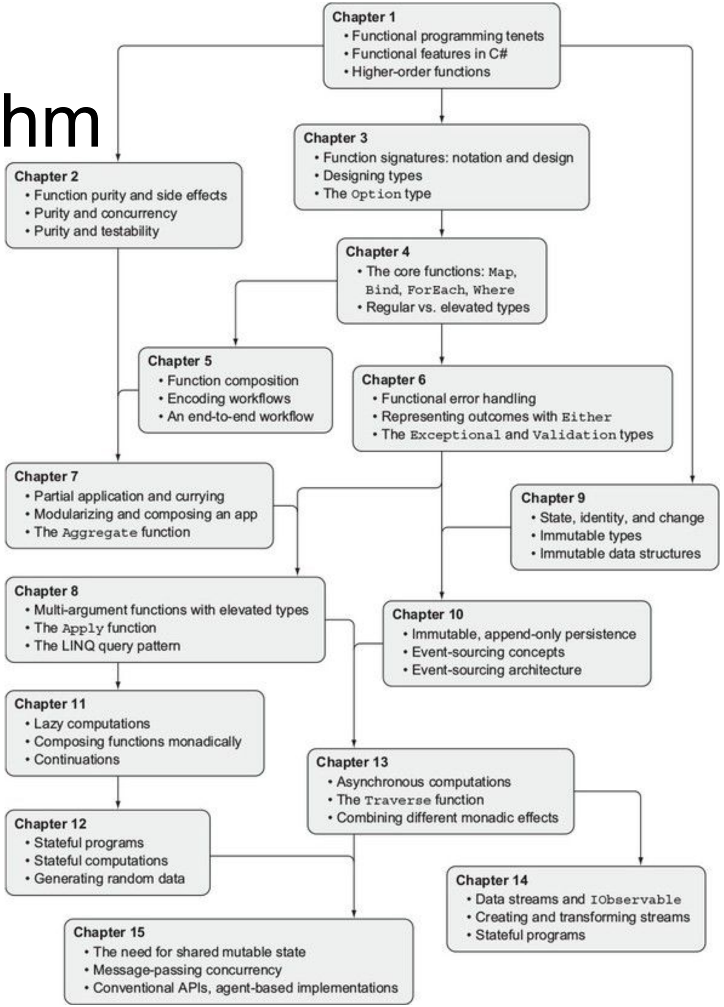
Topological Sort example

- What chapter must one begin their reading with?
- Does the relative order of chapters 2 and 3 matter?
- Does the relative order of chapters 4 and 10 matter?



Topological Sort: Kahn's Algorithm

- While there are still nodes left to process:
  - Initialize a queue
  - For all nodes with in-degree 0:
    - Add node to queue
    - Remove all edges from that node
    - Mark those nodes as processed



Programming Exercise

Recap

- Learned what graphs are
- Learned about properties of graphs, and how to calculate them
- Learned about how to search graphs
- Used networkx to implement the above
- Discussed an advanced graph algorithm

Q&A