Graph Traversal

10.020 Data Driven World

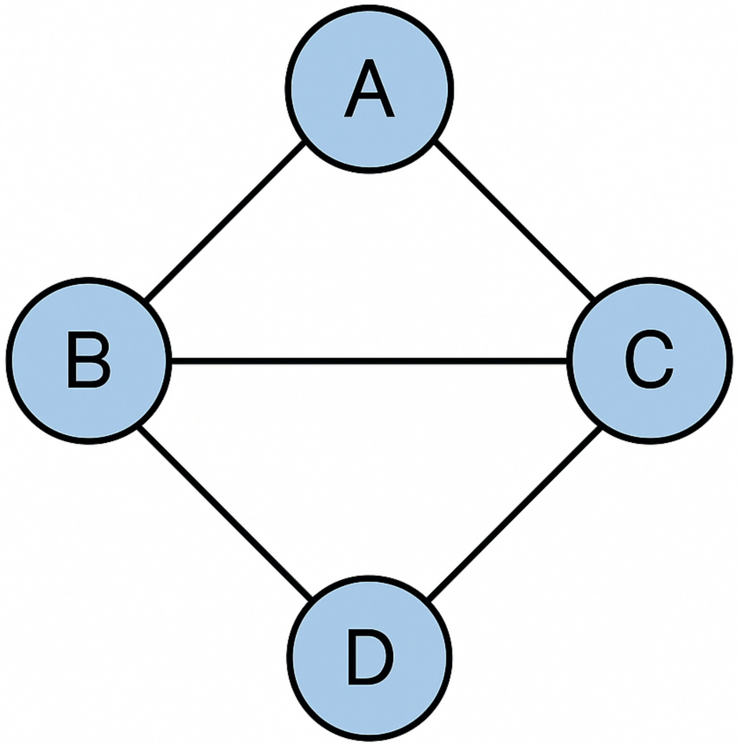
Natalie Agus

Learning Objectives

- Define graph, vertices, edges and weights.
- Use **Dictionary** and **OOP** to represent graph.
    - Represent graphs using **adjacency-list** representation or adjacency-matrix representation.
    - Differentiate **directed** and **undirected** graphs.
- Define paths.
- Create a **Vertex** class and a **Graph** class.
- **Extend** class Vertex and Graph for **graph traversal algorithm**
- Explain and implement **breadth first search**
- Explain and implement **depth first search.**

Graph

Basic Elements

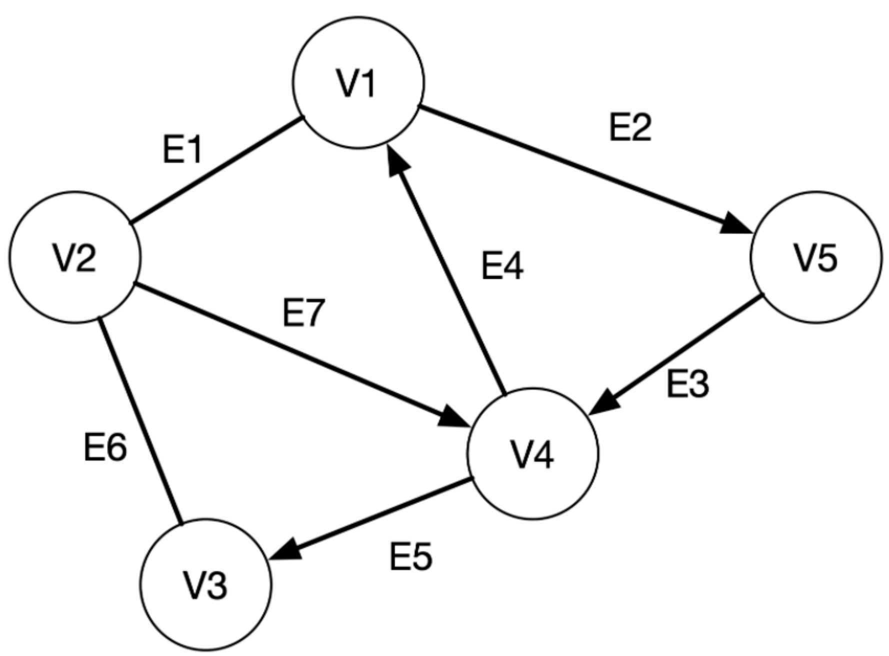


- Graph is a **data structure:**
  - Just like queue, stack, deque, heap
  - Non-linear data structure
- A graph has **nodes** **(vertices)** connected by **edges**
- Application:
  - Model networks (nodes are entities, edges are connections): social network, network routing, recommendation system
  - Game level maps: **pathfinding**
  - **Navigation**

Graph

Directions

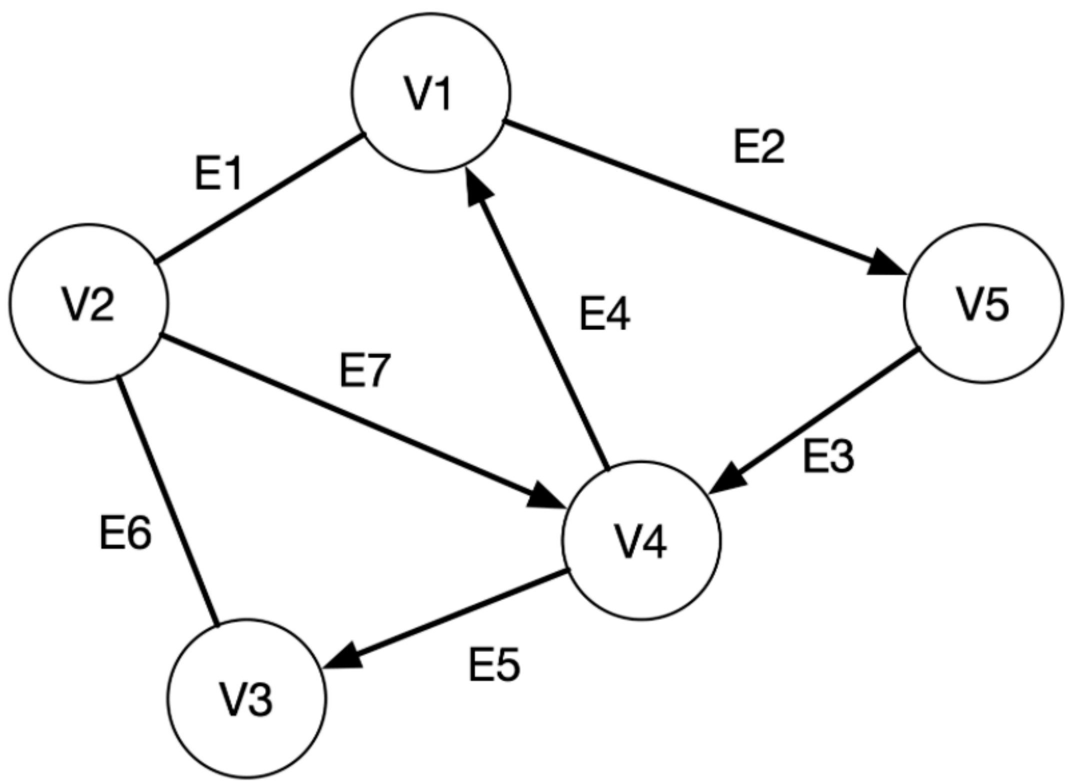
- It can be **unidirectional** or **bidirectional**
  - **Directed vs undirected graphs**
- Tree is a form of graph that does not form a **cycle**



Represent Graph in Code

Adjacency Matrix

- **Cons**: might result in sparse matrix

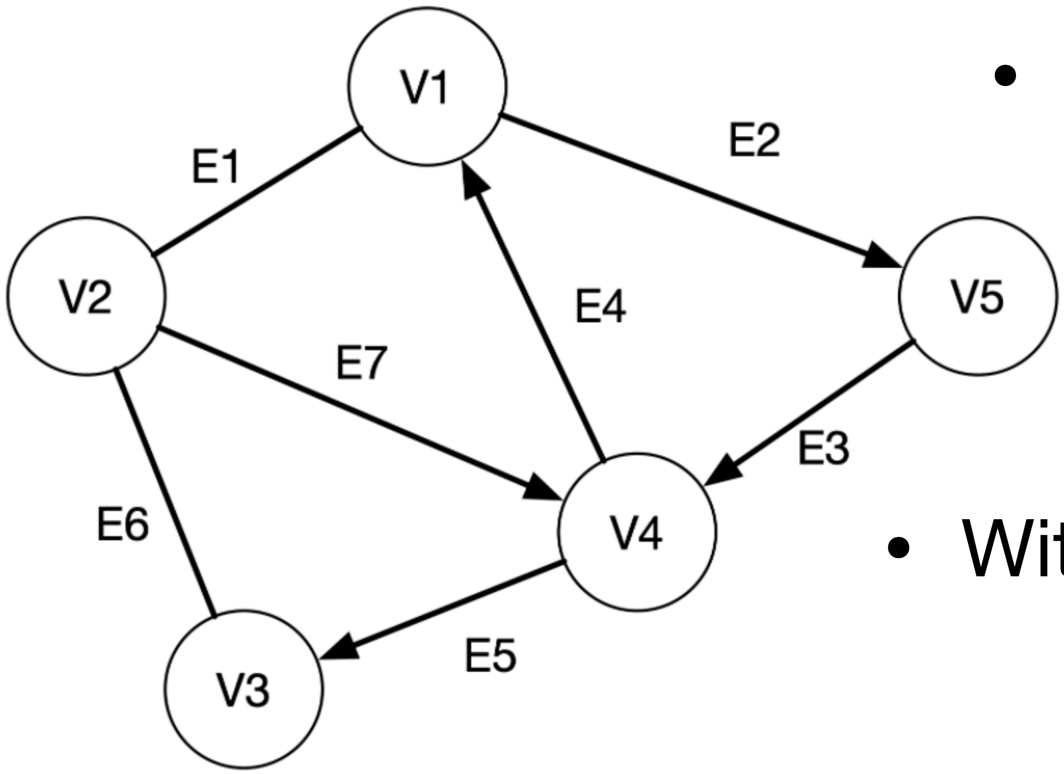


	V1	**V2**	**V3**	**V4**	**V5**
V1		1			1
V2	1		1	1	
V3		1			
V4	1		1		
V5				1	

Represent Graph in Code

Adjacency List

- Suitable if the number of edges is not large



- Without cost

```
graph1 = {'V1': ['V2', 'V5'],
          'V2': ['V1', 'V3', 'V4'],
          'V3': ['V2'],
          'V4': ['V1', 'V3'],
          'V5': ['V4']}
```

- With cost

```
graph1 = {'V1': {'V2': 1, 'V5': 1},
          'V2': {'V1': 1, 'V3': 1, 'V4': 1},
          'V3': {'V2': 1 },
          'V4': {'V1': 1, 'V3': 1},
          'V5': {'V4': 1}}
```

Represent Graph in Code

Using OOP

- It's a **has-a** relationship (composition)
  - A graph has a list of **Vertices**

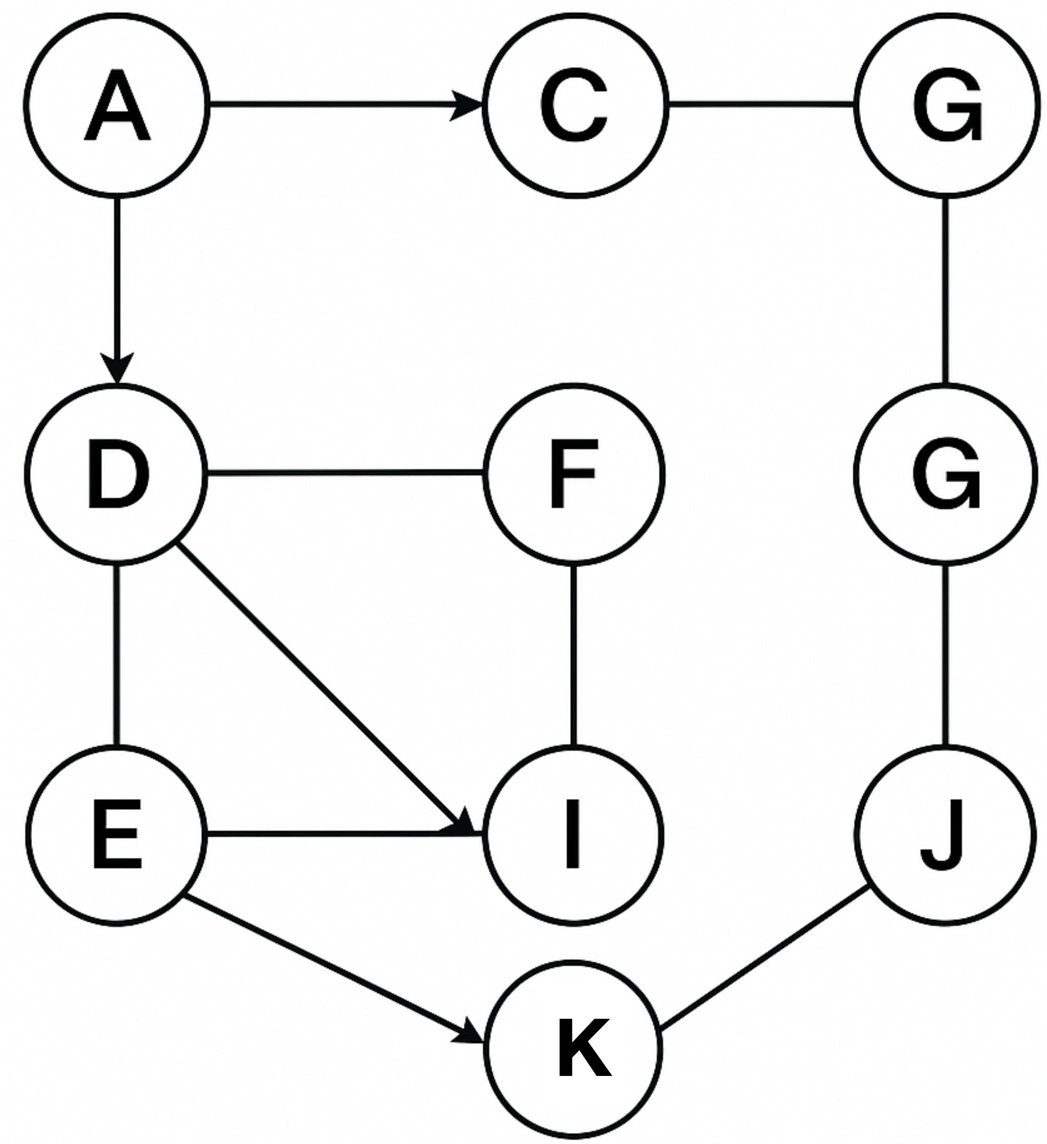
Graph
vertices
add_vertex(id) get_vertex(id) add_edge(start_id, end_id, weight) get_neighbours(id) get_num_vertices()

Vertex
id neighbours
add_neighbour(neighbour_vertex,weight) get_neighbours() get_weight(neighbour_vertex)

Graph Traversal

Given a starting node, how do we "walk" the graph?

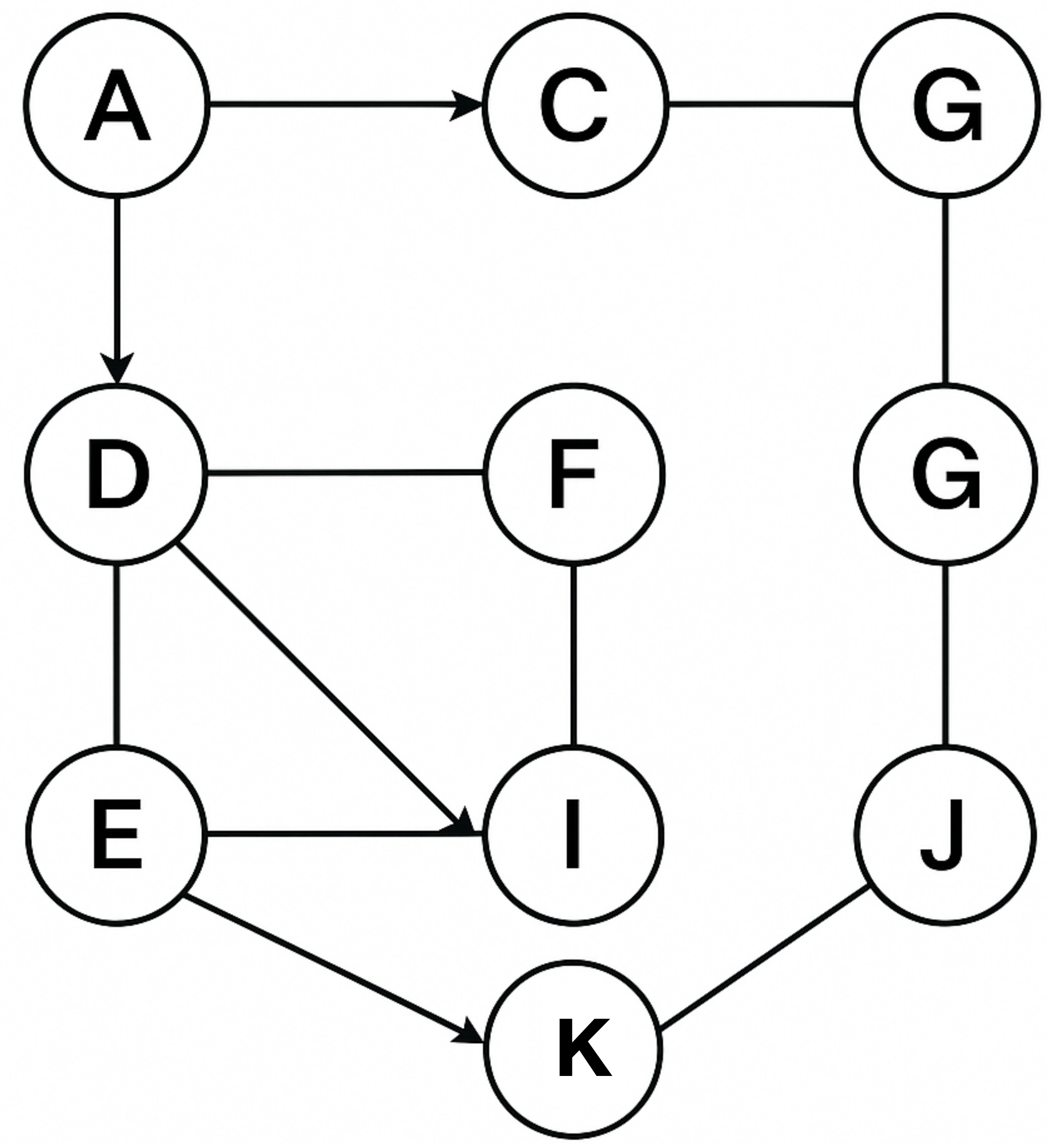
- **Breadth-first** search
- **Depth-first** search
- You need to know:
  - **Starting** node
  - **Neighbours** of each node (directional)
  - Edge costs (optional)



Breadth-first Search

Given a starting node, how do we "walk" the graph?

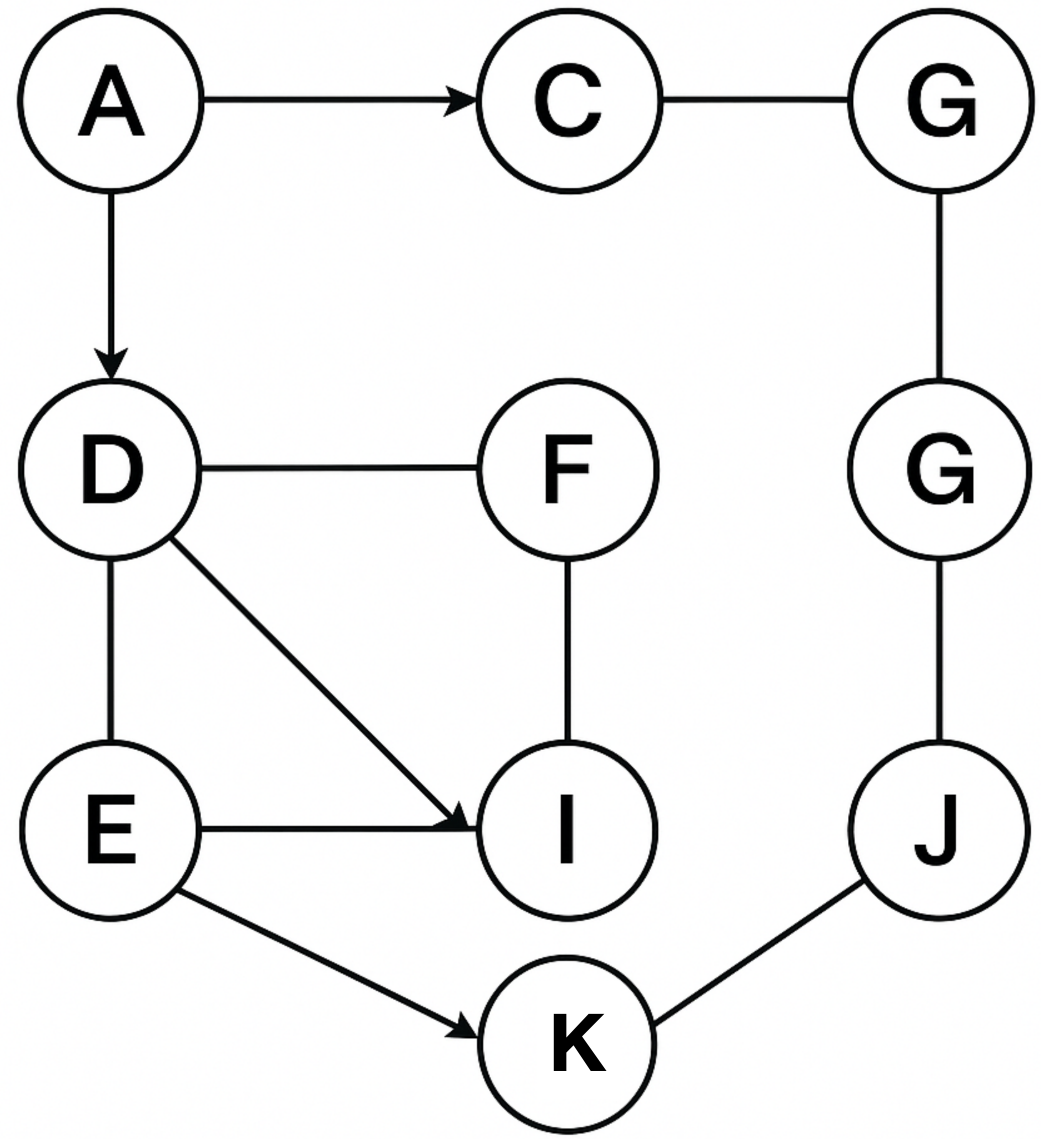
- **Always** queue nodes in the order they are discovered
  - **Useful data structure:** Queue
  - **2-Colour concept**: white (new), black (done)
  - **If weight is constant,** BFS can be used to find **shortest path**



Depth-first Search

Given a starting node, how do we "walk" the graph?

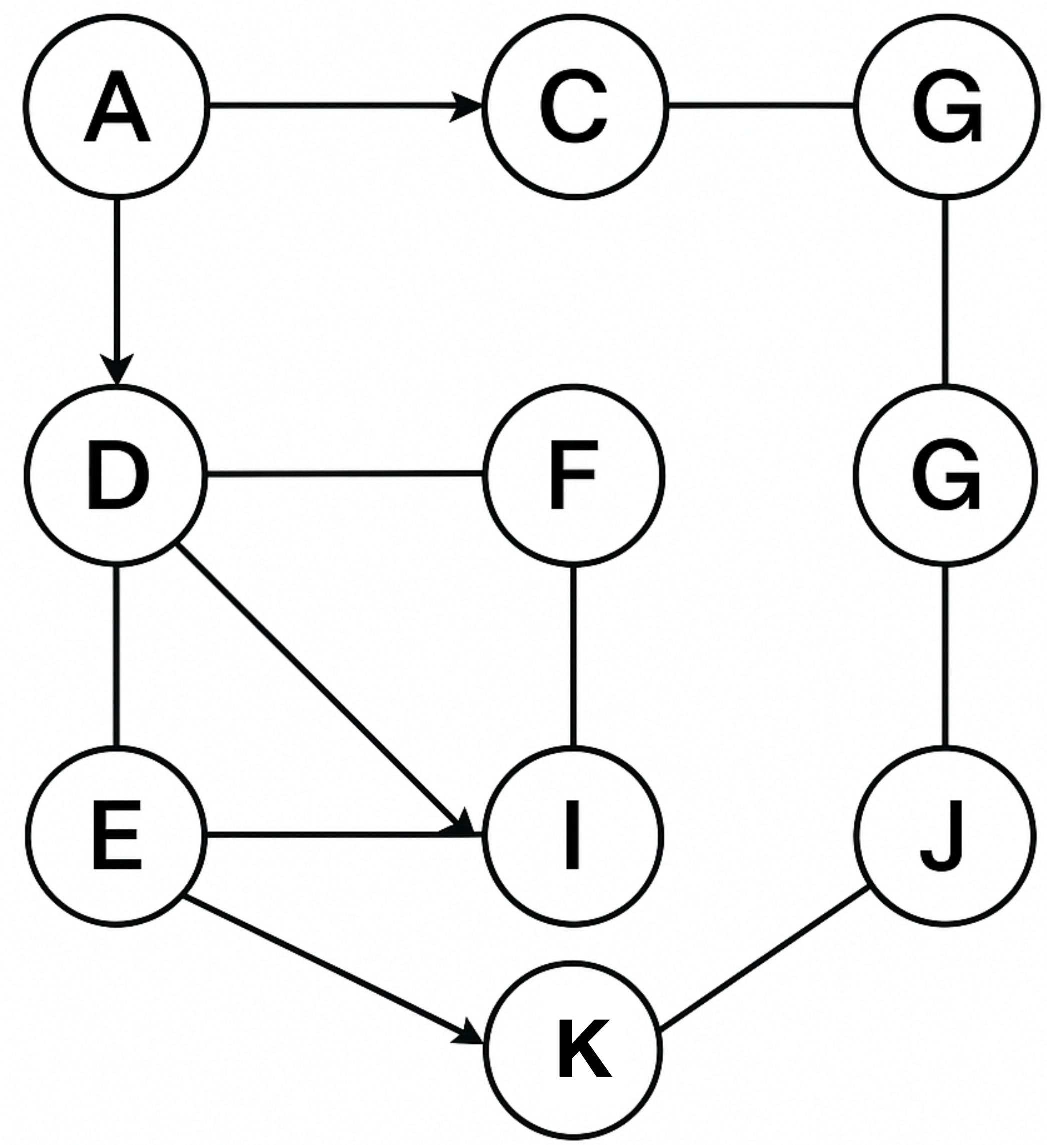
- **Always** stack nodes to explore as deep as possible before **backtracking**
  - **Useful data structure:** Stack
  - **3-Colour concept**: white (new), grey (in-transit, visiting), black (visited, left for good)



Depth-first Search

Cycle Detection

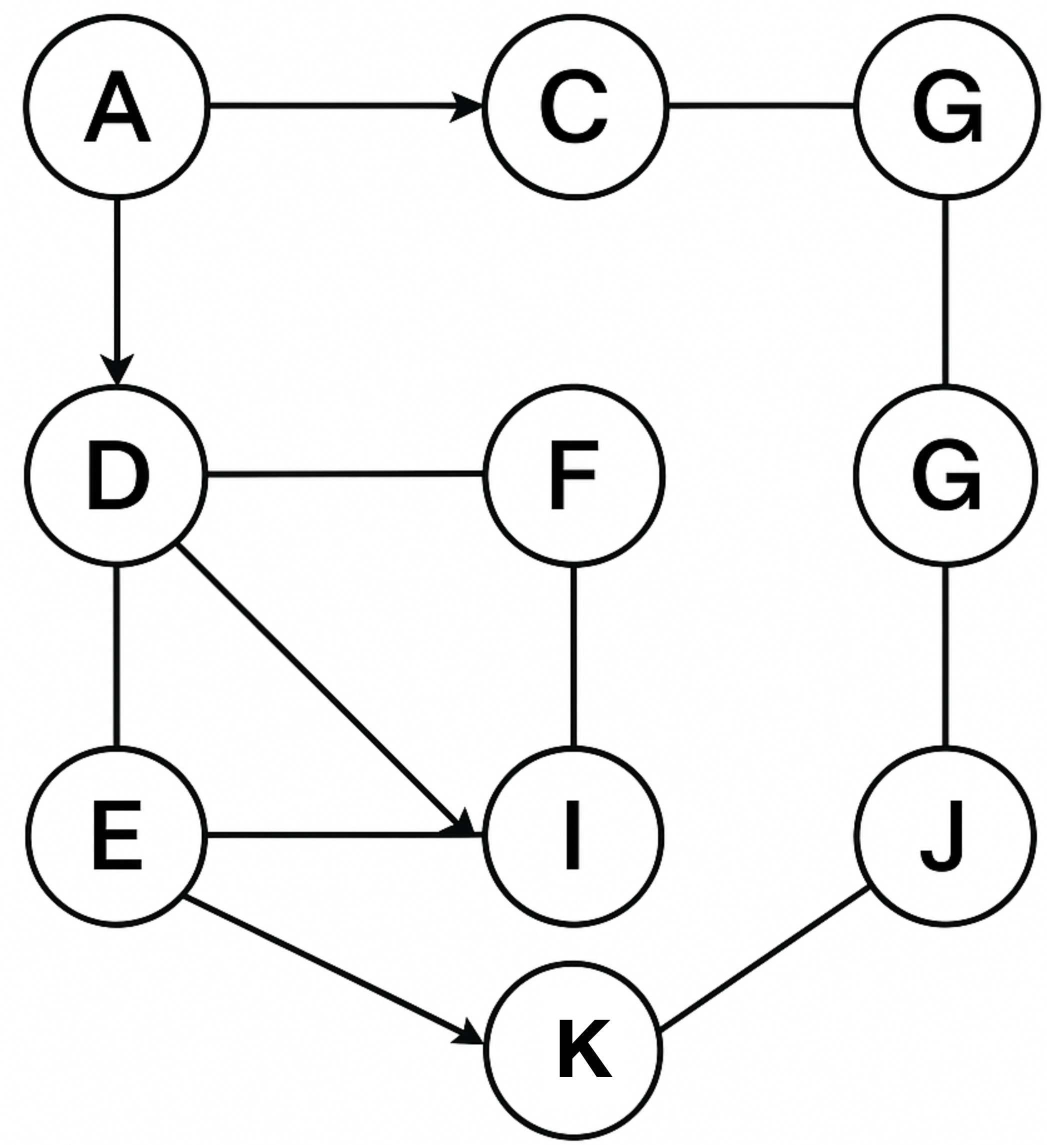
- Meeting an *already explored* node means a cycle is present



Depth-first Search

Topological Sort

- Do DFS, then get order by **finishing** time from largest to smallest



Can BFS detect Cycles?

DFS can detect cycles, what about BFS?

- BFS has no "in-visit" state, it's either *not visited* or *visited*
  - *What if we say: if I saw a vertex that's already visited, that means there's a cycle!*
- In undirected graphs (**bidirectional**)?
- In directed graphs (**unidirectional**)?

Graph Traversal

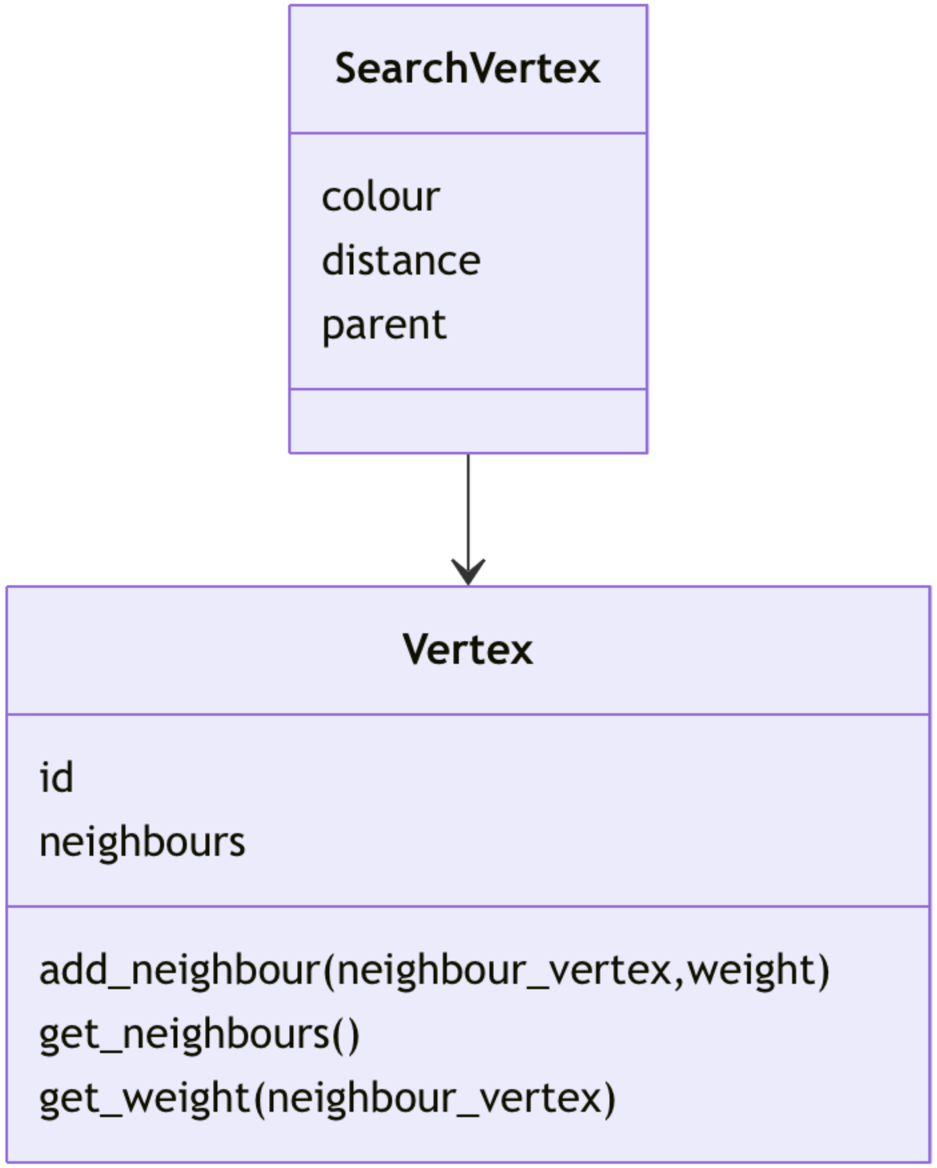
Applications

- **Breadth-first** search
  - Shortest path in public-transport system (edges of **equal** weights)
  - Web crawling: discover all reachable pages from a starting page (level-by-level)
- **Depth-first** search
  - Finding file in nested folders
  - Solving Sudoku or puzzles
  - Detecting **cycles** in dependencies
  - Course prereq solution (**topological sort**)

Inheritance

SearchVertex and Vertex

- Inheritance allows us to **create** a **new** class **without duplicating** all the other parts that is the same as classes we already have
- This is an **is-a** relationship
    - SearchVertex is-a Vertex



Learning Objectives

- Define graph, vertices, edges and weights.
- Use **Dictionary** and **OOP** to represent graph.
    - Represent graphs using **adjacency-list** representation or adjacency-matrix representation.
    - Differentiate **directed** and **undirected** graphs.
- Define paths.
- Create a **Vertex** class and a **Graph** class.
- **Extend** class Vertex and Graph for **graph traversal algorithm**
- Explain and implement **breadth first search**
- Explain and implement **depth first search.**