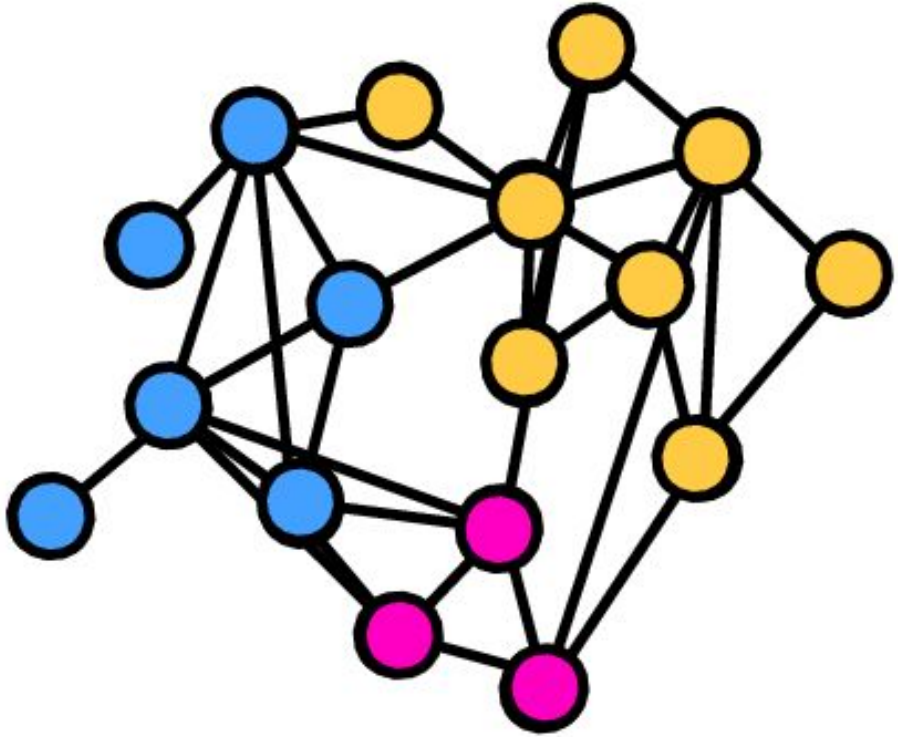
Graphs



Lecture Flow

1) Pre-requisites
2) Definition of Graphs
3) Types of Graphs
4) Graph terminologies
5) Checkpoint
6) Graph representations
7) Graphs & Trees
8) Receiving Inputs on Graph Problems
9) Practice questions
10) Quote of the day

Pre-requisites

- Arrays / Linked list
- Matrices
- Dictionaries
- Time and space complexity analysis

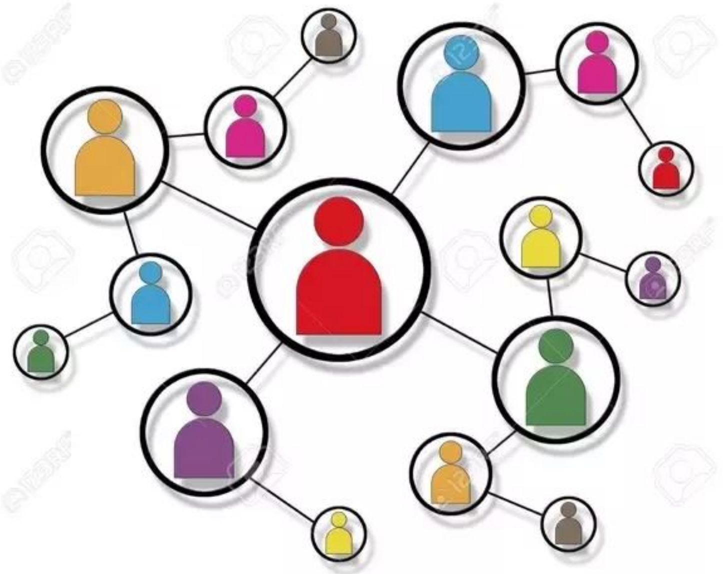
What is a Graph?

Definition

- A way to represent **relationships** between objects.
- A collection of **nodes** that have data and are connected to other nodes.

Example: Friendship Graph

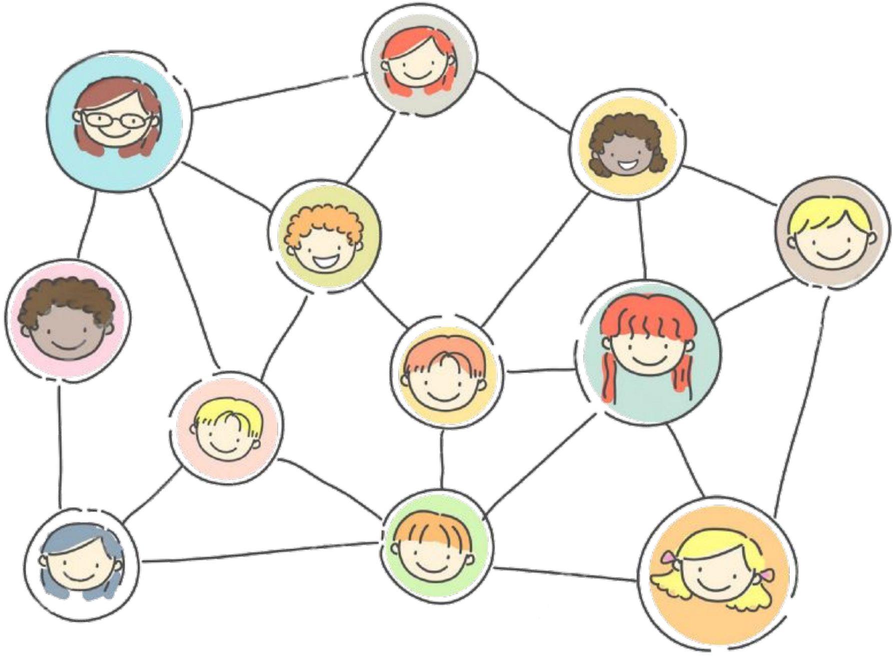
- Nodes or vertices: The objects in graph
    - These people our case
- Edges: the relation between nodes.
    - The friendship between them (the lines)



What are the types of graphs?

Undirected Graph

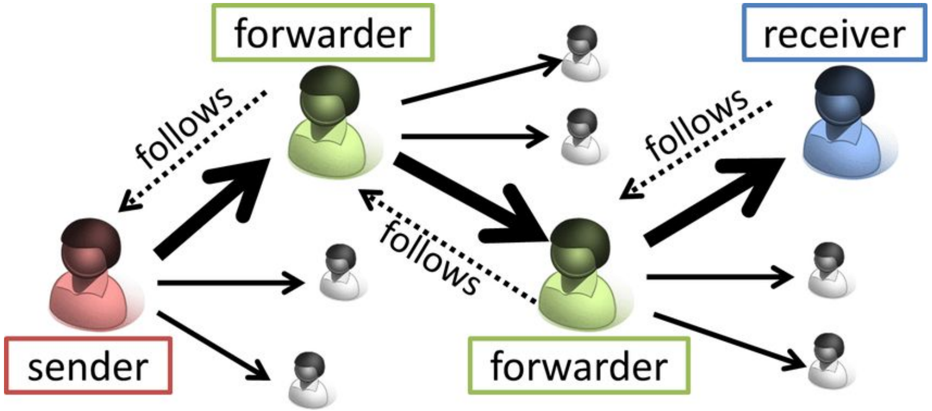
- Facebook friendship
- If Alice is friends with Bob, Bob is also friends with alice



Directed Graph

- Twitter's "following" relation

- If person A follows person B, that does not mean that person B follows person A.

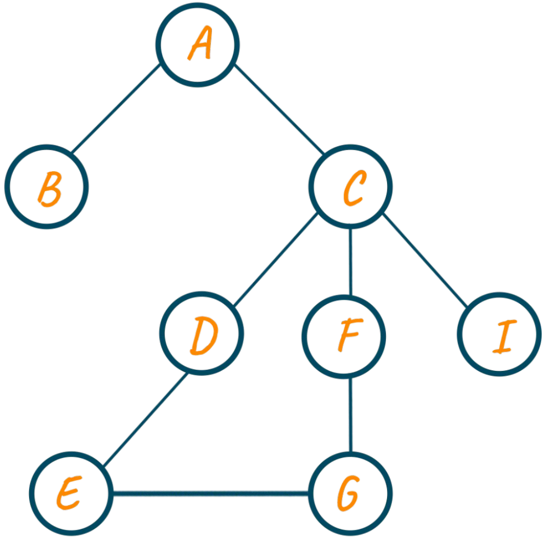


Let’s say we want to add a weight parameter which represents the strength of the friendship.

How can we do that?

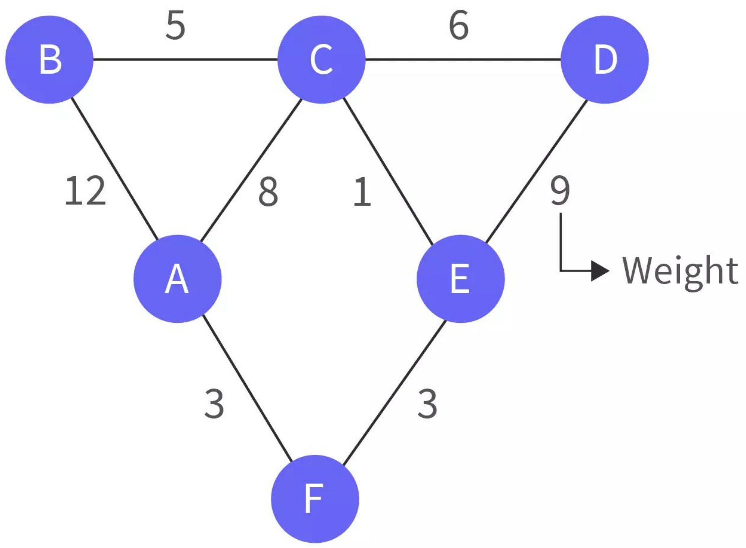
Unweighted Graph

All edges have the **same value.**



Weighted Graph

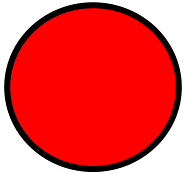
Weighted graphs assign **numerical values** to edges.



Graph Terminologies

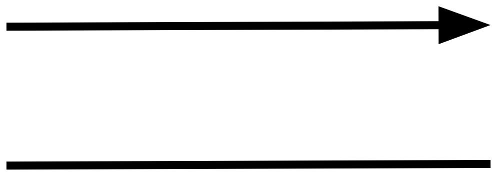
Terminology: Node / Vertex

- Represent entities (e.g., people, devices).
- Connected to other nodes by edges.



Terminology: Edge

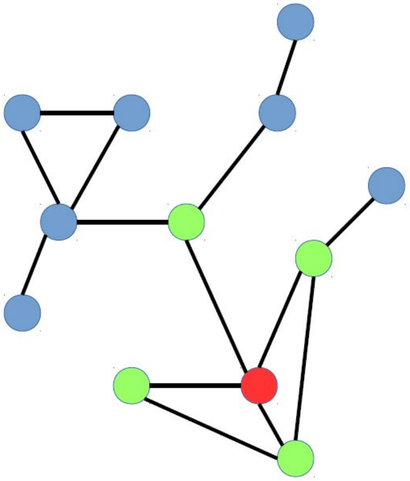
- Connection between two nodes.
- Represented by lines or arrows.
- model relationships in various systems.



Terminology: Neighbors

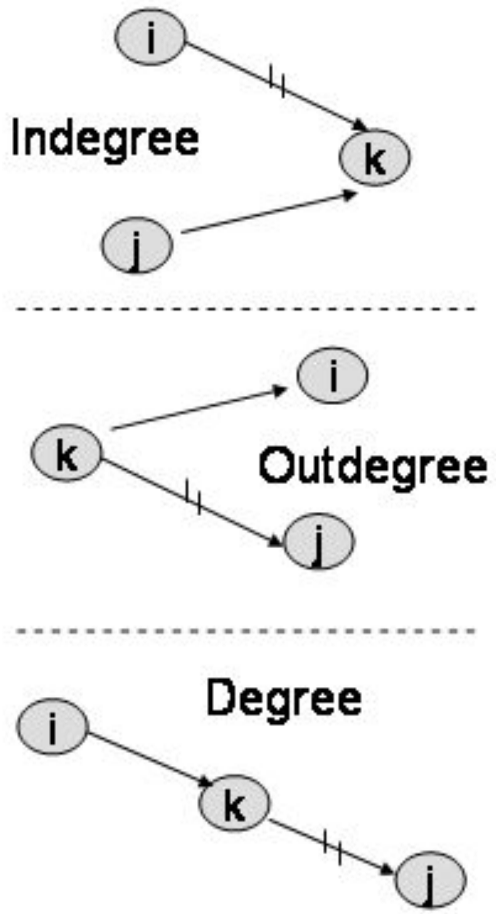
- Two nodes are **neighbors** or **adjacent** if there is an edge **between** them

Graph Neighborhood



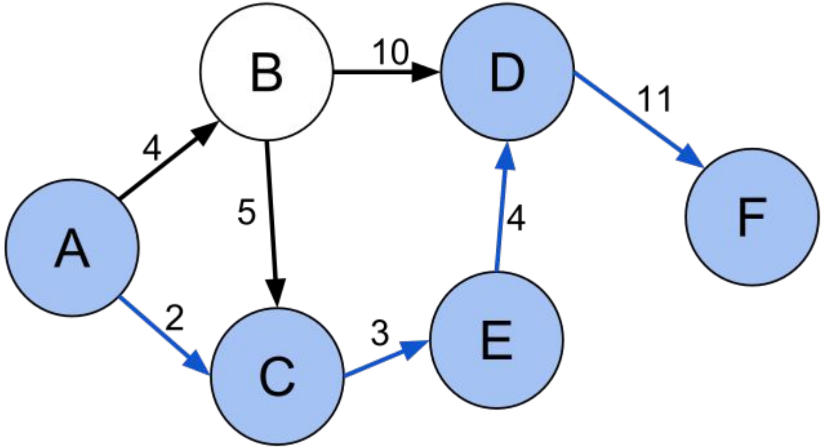
Terminology: Degree

- Node degree = number of neighbors.
- In directed graphs:
  - Indegree = edges ending at node.
  - Outdegree = edges starting at node.



Terminology: Path

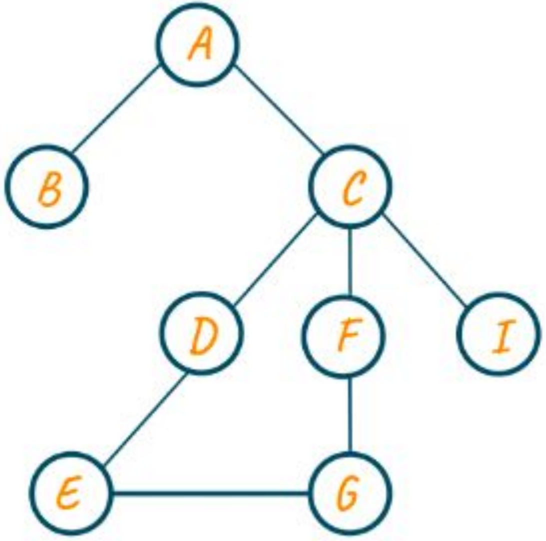
- A **path** leads from one node to another.
- The length of a path is the number of edges in it.
- Let's consider the path between node A and node F



Terminology: Cycle

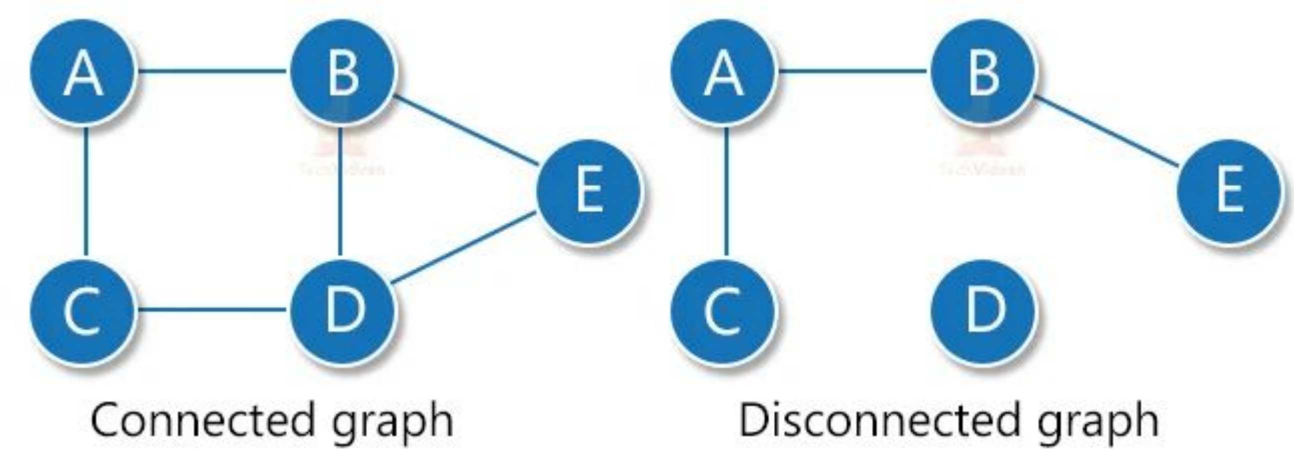
A path is a **cycle** if the first and the last node of the path is the same.

Cycle = C-D-E-G-F-C



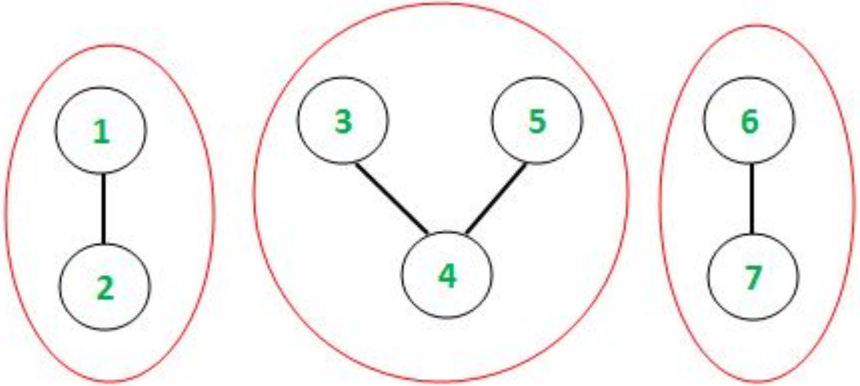
Terminology: Connectivity

A graph is **connected** if there is a path between any two nodes



Terminology: Components

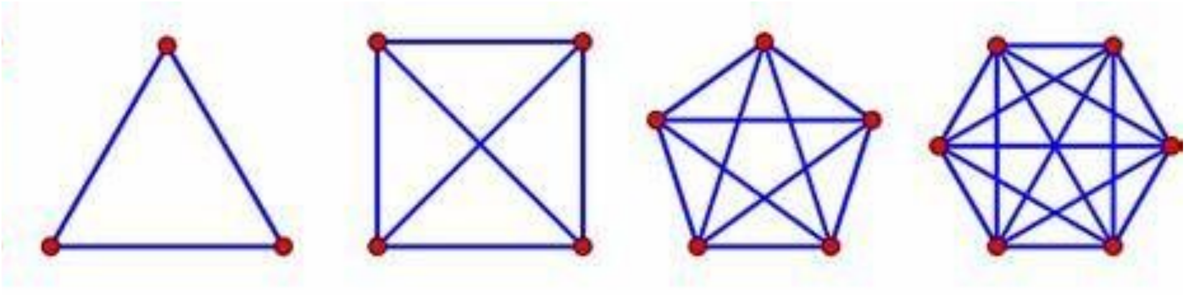
The connected parts of a graph are called its **components.**



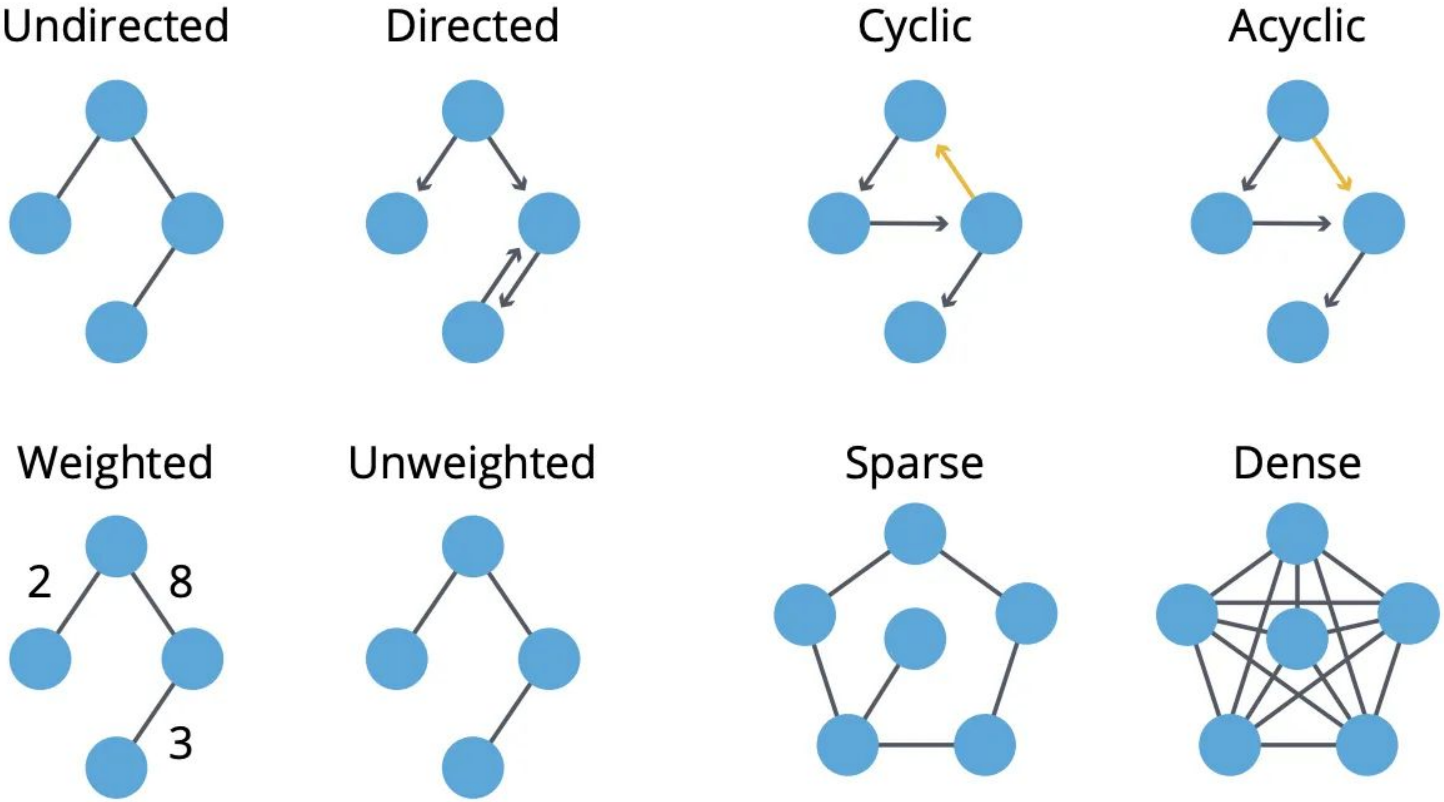
The **counts** of connected components are - **2, 3 and 2**

Terminology: Complete Graph

A complete graph is a graph in which **each pair of node** is connected by an edge.



Summary


Undirected
Directed
Cyclic
Acyclic
Weighted
2
8
3
Unweighted
Sparse
Dense

Common Terminologies

Node: _____?

Edge: _____?

Path: _____?

Cycle: _____?

Connectivity: _____?

Components: _____?

Common Terminologies

Node: represents elements

Edge: is like a line connecting two points or nodes

Path: list of edges that connects nodes

Cycle: if start and end of a path is the same

Connectivity: if there is a path between any two nodes of the graph

Components: connected part of a graph

Common Terminologies

Tree: _____?

Neighbours(adjacent): _____?

Degree: _____?

Indegree: _____?

Outdegree: _____?

Complete Graph: _____?

Traverse: _____?

Common Terminologies

Tree: is undirected, connected graph consists of n nodes and n-1 edges

Neighbours(adjacent): two nodes are neighbors if there is an edge between them

Degree: is number of neighbours a node has

Indegree: number of edges that end at the node

Outdegree: number of edges that start at the node

Complete Graph: every node has n-1 degree (an edge from every node to every other node)

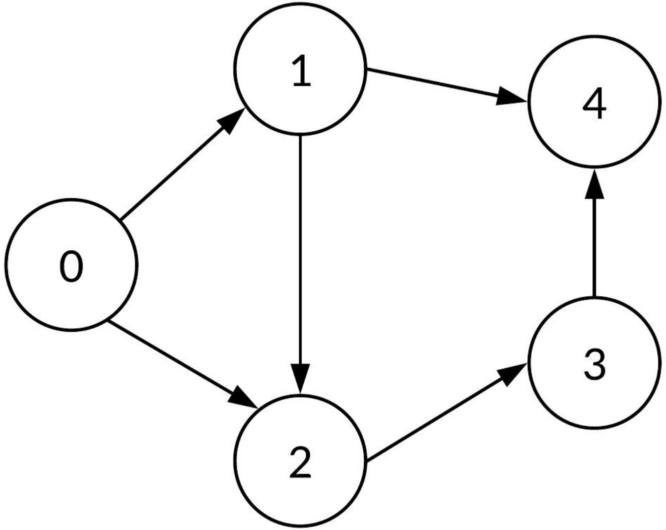
Traverse: going through the graph using edges

Graph Representations

Graph Representations

- Adjacency Matrix
- Adjacency List
- Edge List
- Grids as Graphs

Graph Representation: Adjacency matrix



Adjacency Matrix

	0	1	2	3	4
0	0	1	1	0	0
1	0	0	1	0	1
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	0	0	0

Advantages and disadvantages of Adjacency Matrix

Graph Representation: Adjacency matrix

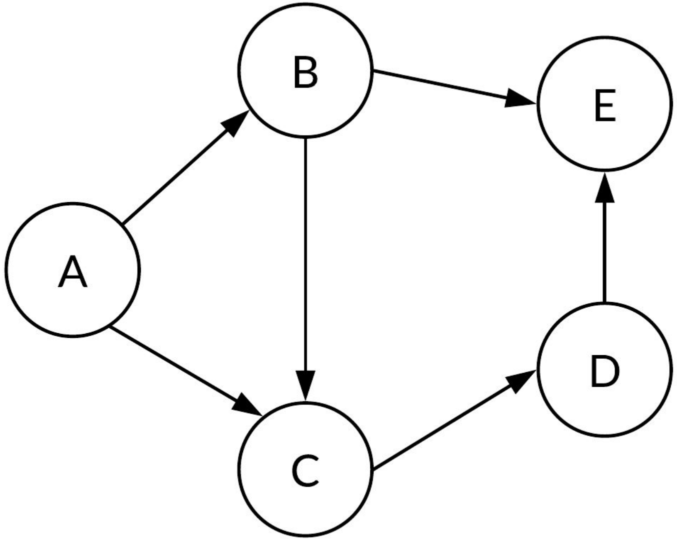
Advantages:

- To represent dense graphs.
- Edge lookup is fast

Disadvantages:

- It takes more time to build and consume more memory (**O(N**2)** for both cases)
- Finding neighbors of a node is costly

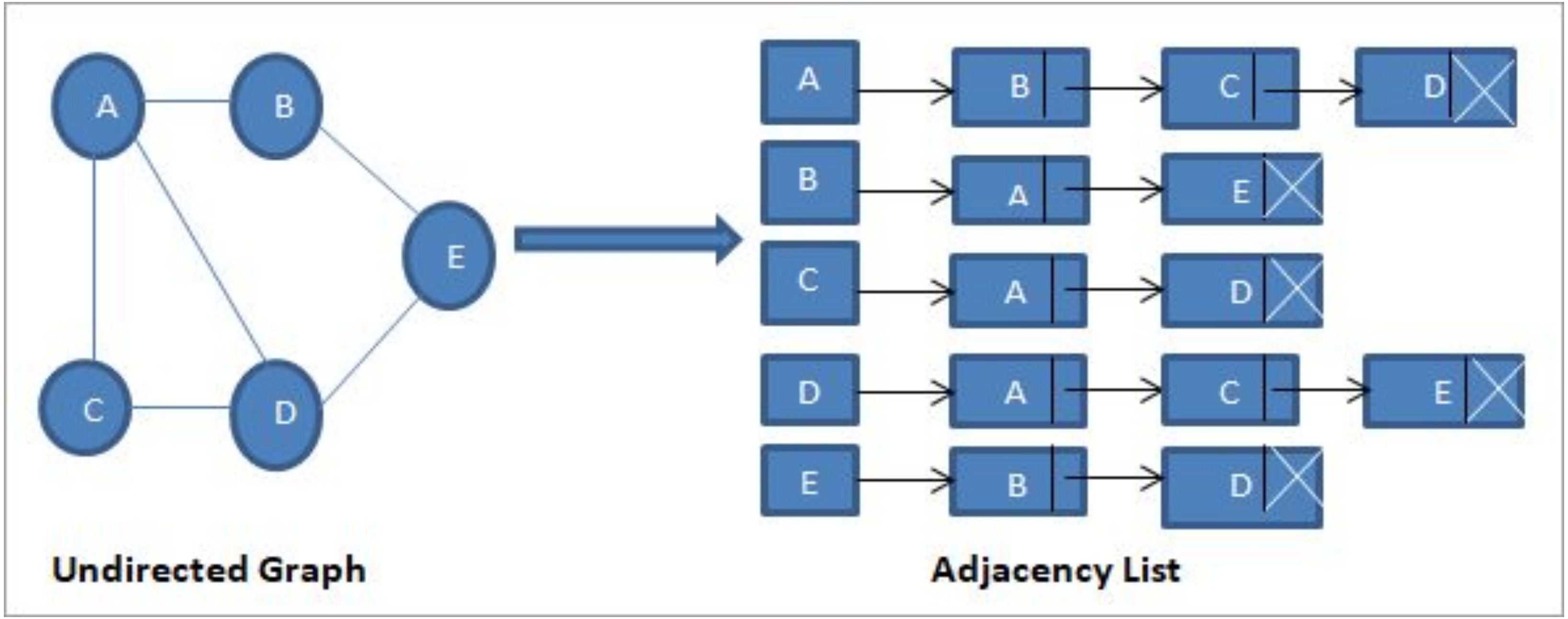
Graph Representation: Adjacency List using List




```
Adjacency list:

graph = {
'A': ['B', 'C'],
'B': ['C', 'E'],
'C': ['D'],
'D': ['E'],
}
```

Graph Representation: Adjacency List using Linked List



Advantages and Disadvantages of Adjacency List

Graph Representation: Adjacency list

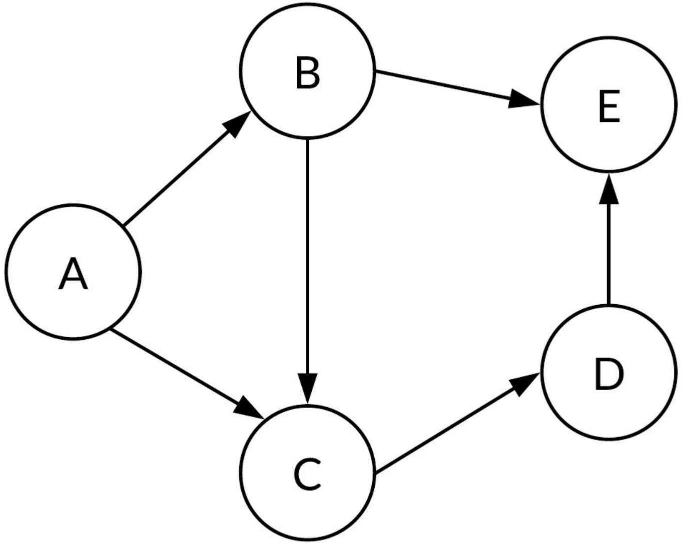
Advantages:

- It uses less memory
- Neighbours of a node can be found pretty fast
- Best for sparse graphs

Disadvantages:

- Edge look up is slow

Graph Representation: Edge list




```
Edge list:

graph = [('A', 'B'),
('A', 'C'),
('B', 'C'),
('B', 'E'),
('C', 'D'),
('D', 'E')
]
```

Advantages and Disadvantages of Edge List

Graph Representation: Edge list

Advantages:

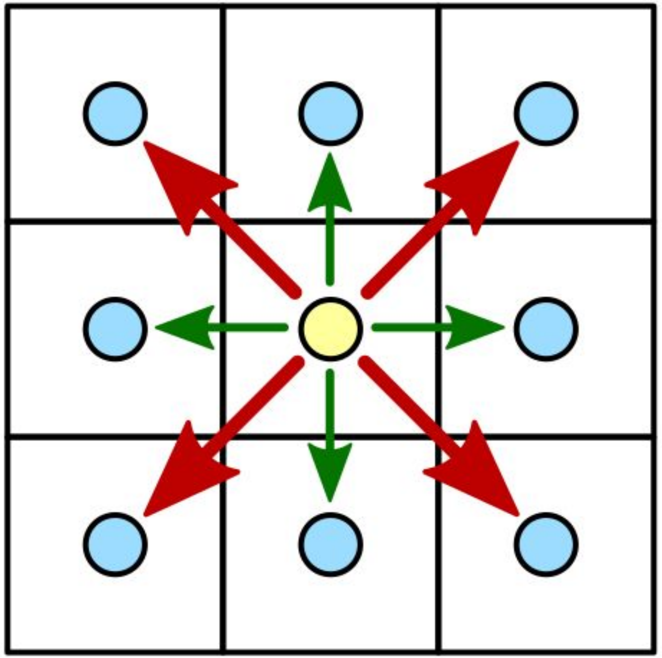
- It uses less memory.
- Easy to represent

Disadvantages:

- Edge look up is slow
- Hard to traverse

Graph Representation: Grids as Graph

- Matrix cells = **nodes**
- **Edges** between adjacent cells:
    - **4 perpendicular**
    - **2 diagonal/antidiagonal.**



Graph Representation: Grids

- ## Direction vectors

The standard square neighborhood

(-1,-1)	(-1, 0)	(-1,+1)
(0, -1)		(0, +1)
(+1,-1)	(+1, 0)	(+1,+1)

Only vertical and horizontal neighbors

	(-1, 0)	
(0, -1)		(0, +1)
	(+1, 0)	

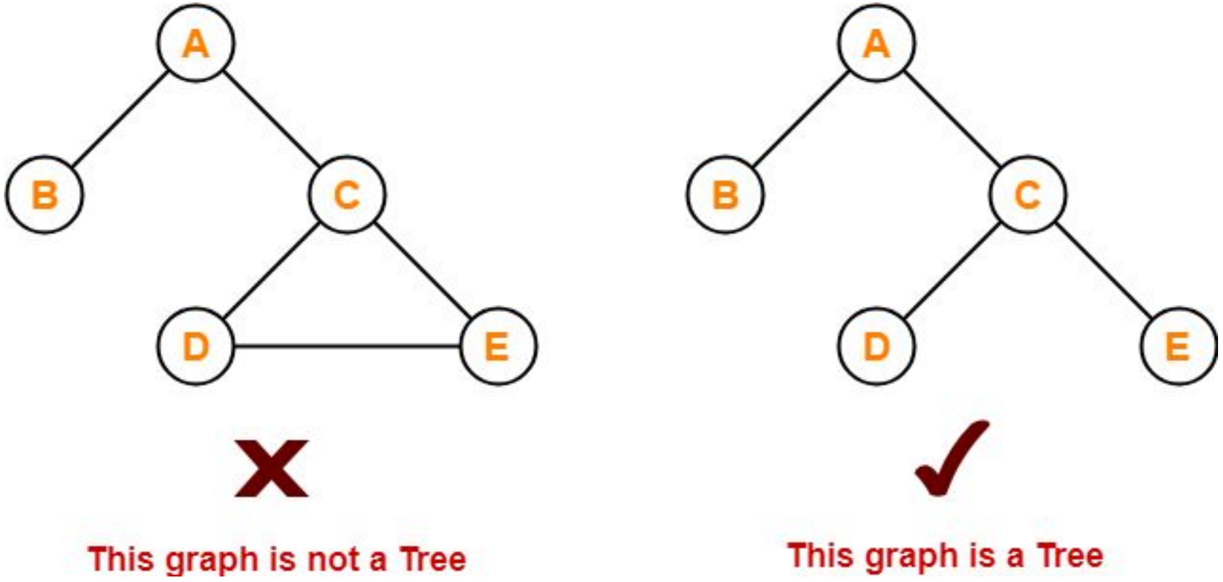
The chess knight neighborhood

	(-2,-1)		(-2,+1)	
(-1,-2)				(-1,+2)
(+1,-2)				(+1,+2)
	(+2,-1)		(+2,+1)	

Graph and Tree

Tree

- A tree is a **connected** and **acyclic graph.**
- A tree has a **unique** path between any two vertices.
- How many edges does a tree have?



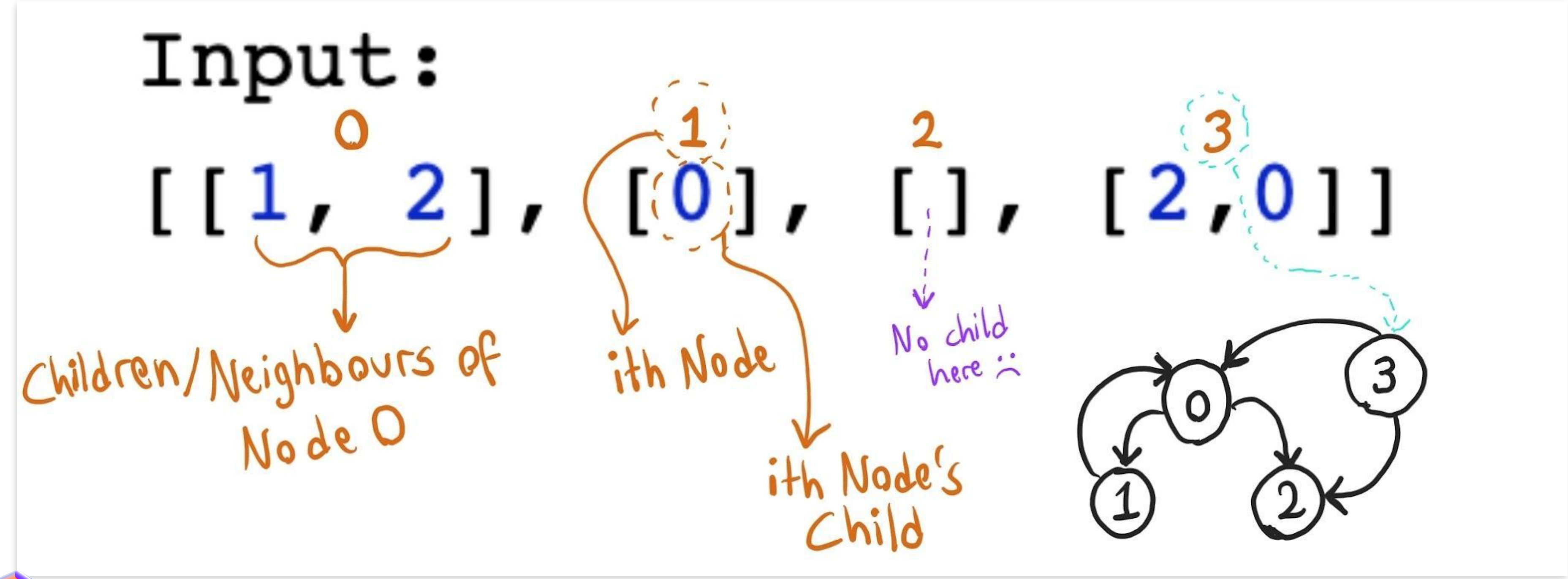
Receiving Inputs on Graph Problems

Adjacency List Inputs

Receiving Inputs: Directed Unweighted Graphs - AL

```
Input:
[[1, 2], [0], [], [2, 0]]
```

Receiving Inputs: Directed Unweighted Graphs - AL

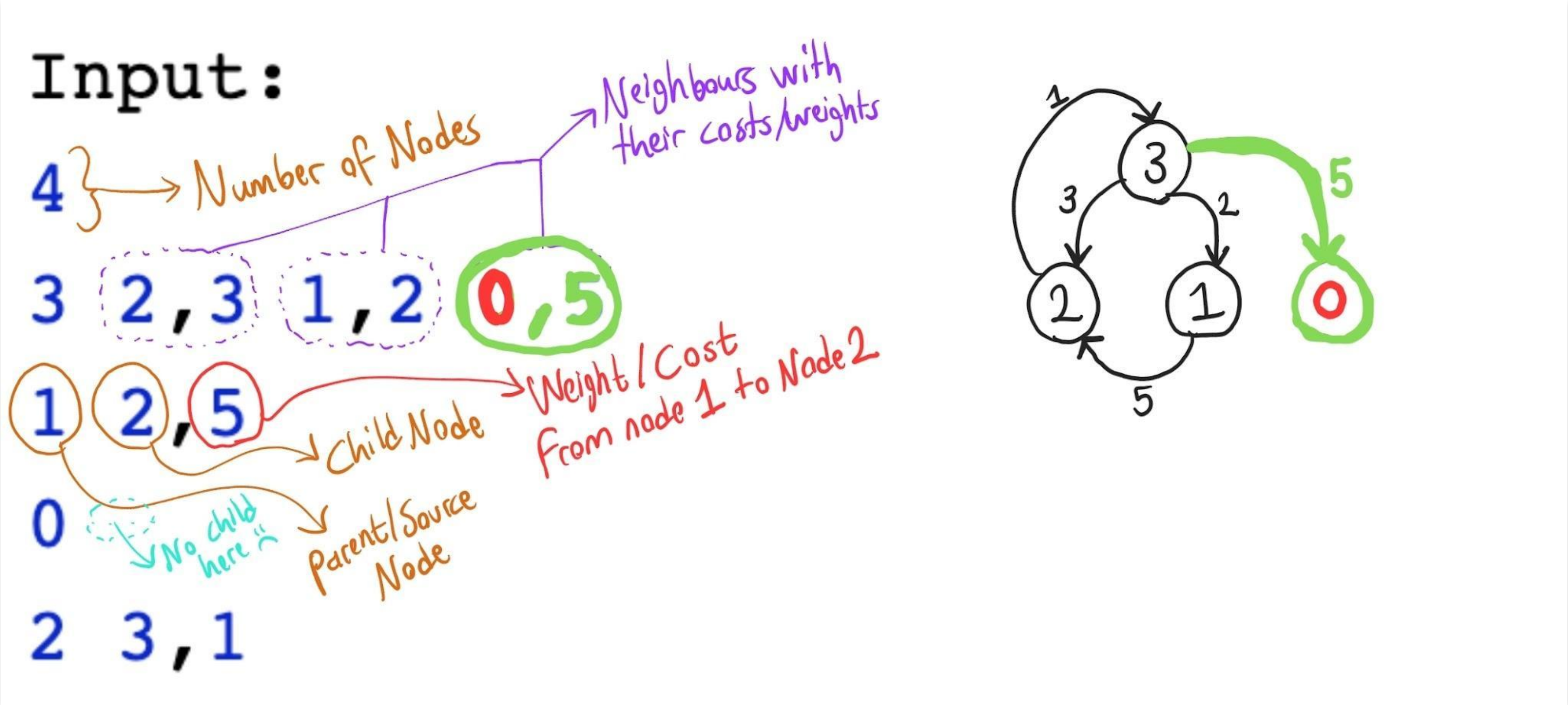


Think of ways to implement this

Receiving Inputs: Directed Weighted Graphs - AL

```
Input:
4
3 2,3 1,2 0,5
1 2,5
0
2 3,1
```

Receiving Inputs: Directed Weighted Graphs - AL



Receiving Inputs: DWG Complexity Analysis - AL

- Time Complexity: O(n + m)
- Space Complexity: O(2m)

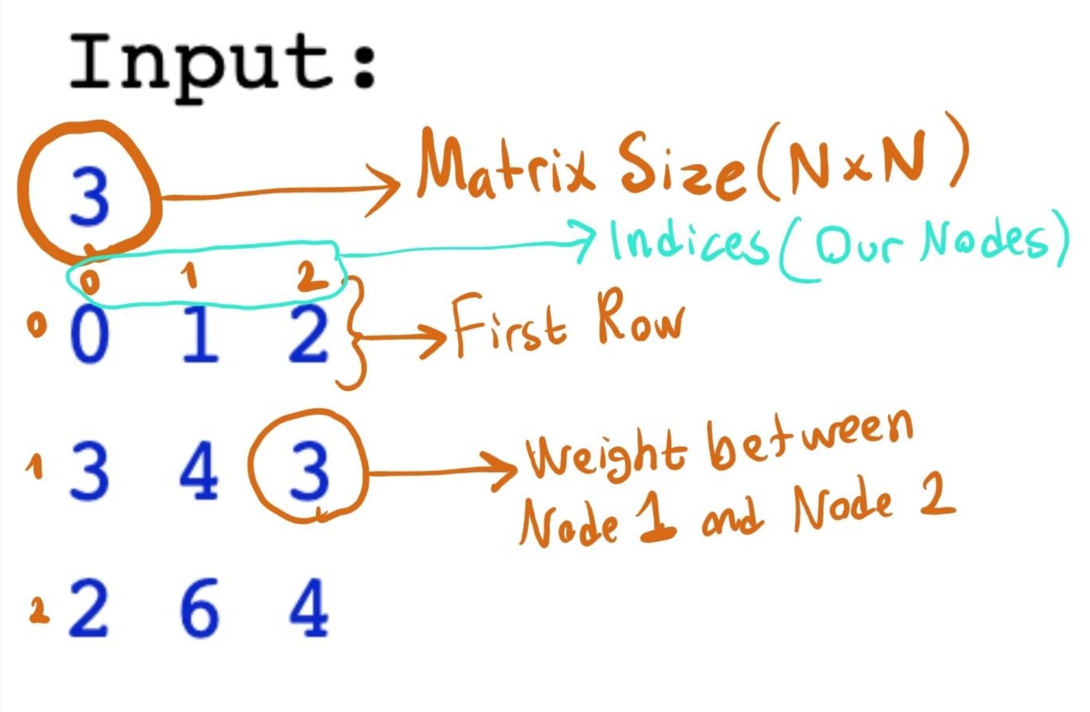
n = number of nodes
m = number of edges

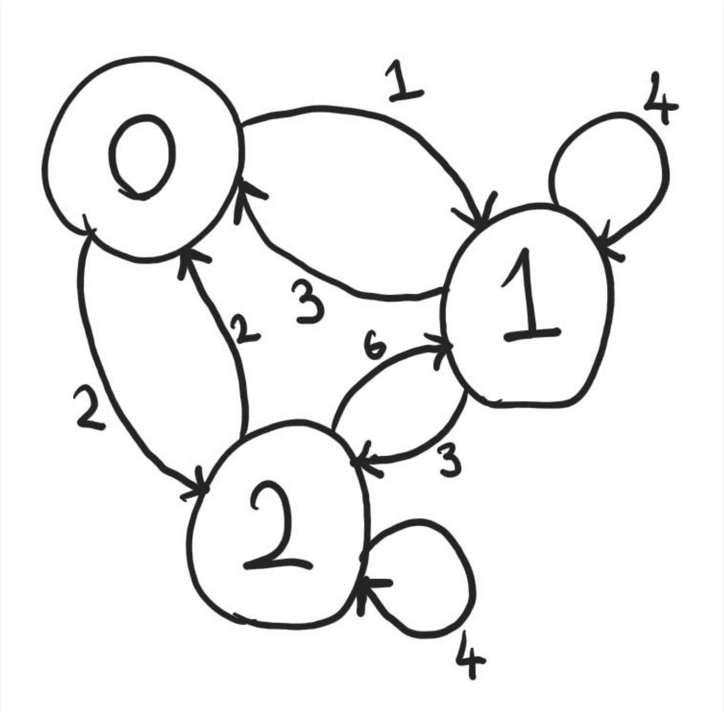
Adjacency Matrix Inputs

Receiving Inputs: Directed Weighted Graphs - AM

```
Input:
3
0 1 2
3 4 3
2 6 4
```

Receiving Inputs: DWG Illustration - AM





Receiving Inputs: DWG Complexity Analysis - AM

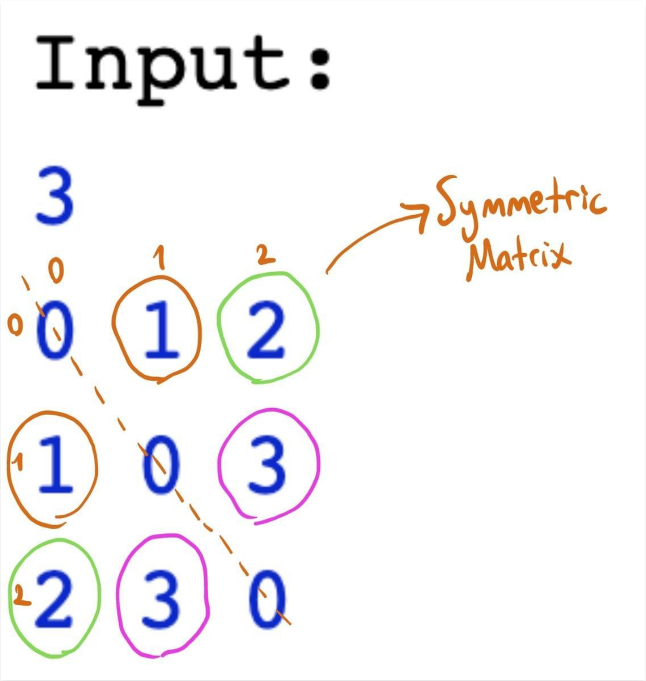
- Time Complexity: **O(n^2)**
- Space Complexity: **O(n^2)**

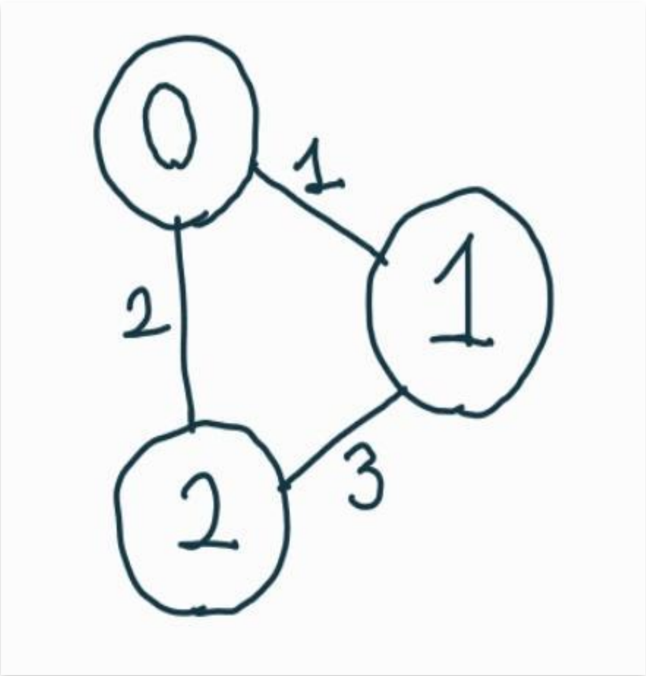
n = number of nodes(matrix length)

Receiving Inputs: Undirected Weighted Graphs - AM

```
Input:
3
0 1 2
1 0 3
2 3 0
```

Receiving Inputs: UDWG Illustration - AM





Receiving Inputs: UDWG Complexity Analysis - AM

- Time Complexity: **O(n^2)**
- Space Complexity: **O(n^2)**

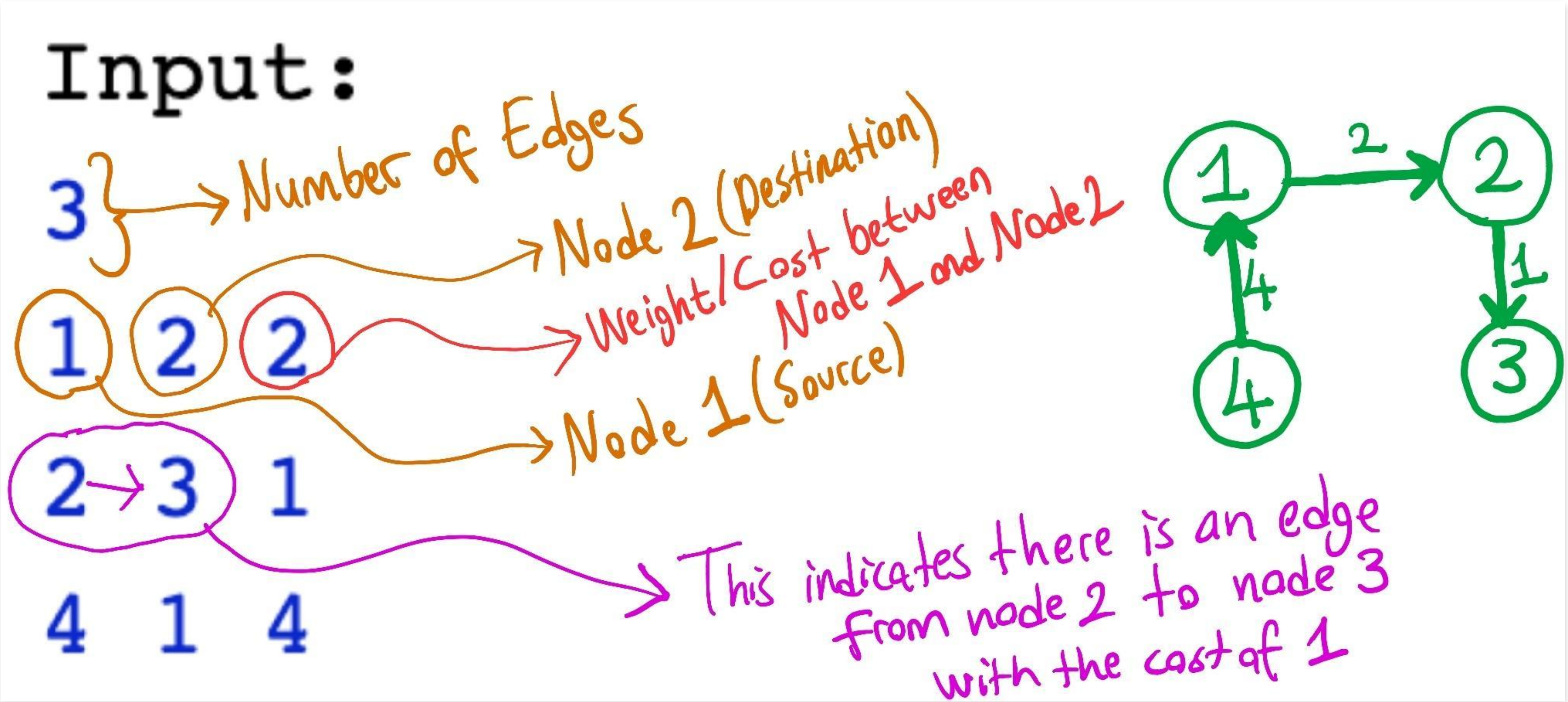
n = number of nodes (matrix length)

Edge List Inputs

Receiving Inputs: Directed Weighted Graphs - EL

```
Input:
3
1 2 2
2 3 1
4 1 4
```

Receiving Inputs: DWG Illustration - EL



Receiving Inputs: DWG Complexity Analysis - EL

- Time Complexity: **O(n)**
- Space Complexity: **O(n)**

n = number of edges

For Multiple Graph Problems (Generic Templates)

Generic template - I

```python
t = int(input()) # number of test cases

for _ in range(t):
    # number of nodes and edges
    n, m = map(int, input().split())
    graph = [[] for _ in range(n)]
    for j in range(m):
        u, v = map(int, input().split())
        graph[u - 1].append(v - 1)
        graph[v - 1].append(u - 1)

    # do something with the graph
```

Which kind of input is the graph?

OR

Generic template - II

```python
from collections import defaultdict

t = int(input()) # number of test cases
for _ in range(t):
    # number of nodes and edges
    n, m = map(int, input().split())
    graph = defaultdict(list)
    for j in range(m):
        u, v = map(int, input().split())
        graph[u].append(v)
        graph[v].append(u)
```

Tip - To make your inputs faster...

```python
import sys
input = sys.stdin.readline
```

- This replaces the input() function with sys.stdin.readline() which is faster.

Common Pitfalls

Common Pitfalls

- Not considering cycles in the graph.
- Not checking whether the graph is directed or undirected
- Not understanding input format well
- Falling in to infinite loop (because of cycles)

Types of Graph Questions

- Graph questions can be classified into different categories based on the problem requirements.
- Some common types of graph questions include:
    - Shortest path: find the shortest path between two vertices.
    - Connectivity: determine if there is a path between two vertices.
    - Cycle detection: detect cycles in the graph.
    - Topological sorting: order the vertices in a directed acyclic graph.

Approaches to Solving Graph Problems

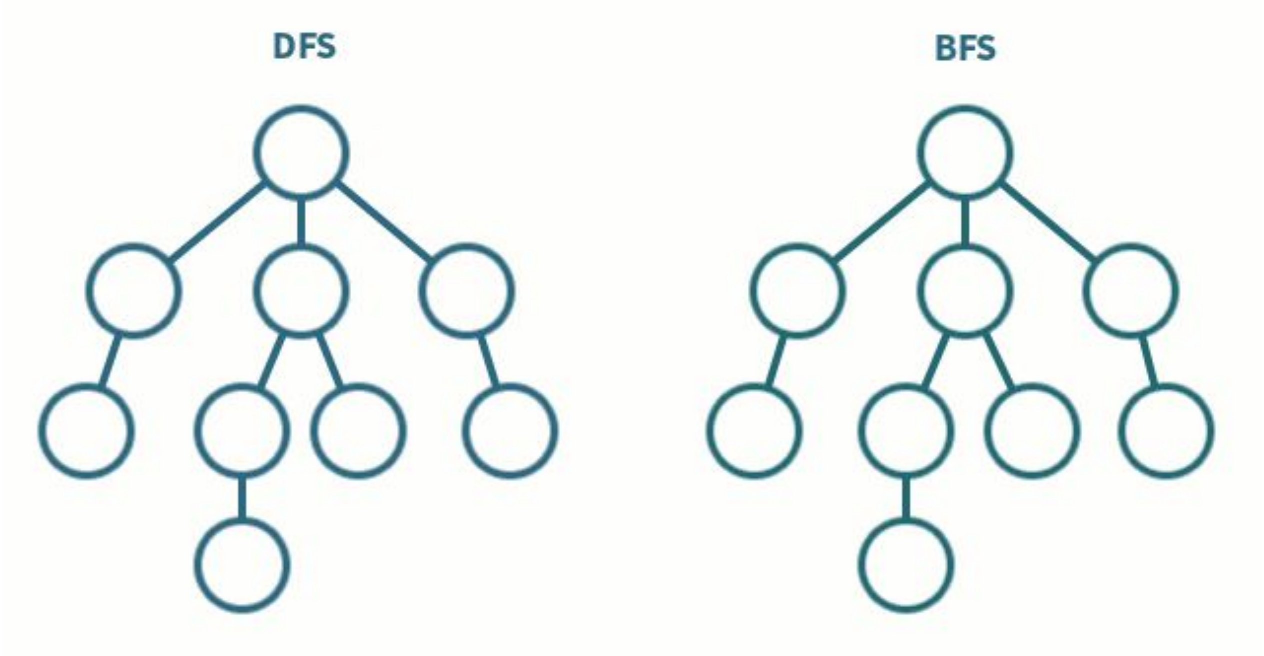
There are several approaches to solving graph problems, including:

- Breadth-first search (BFS)
- Depth-first search (DFS)
- Dijkstra's algorithm
- Bellman-Ford algorithm
- Kruskal's algorithm
- Floyd-Warshall algorithm

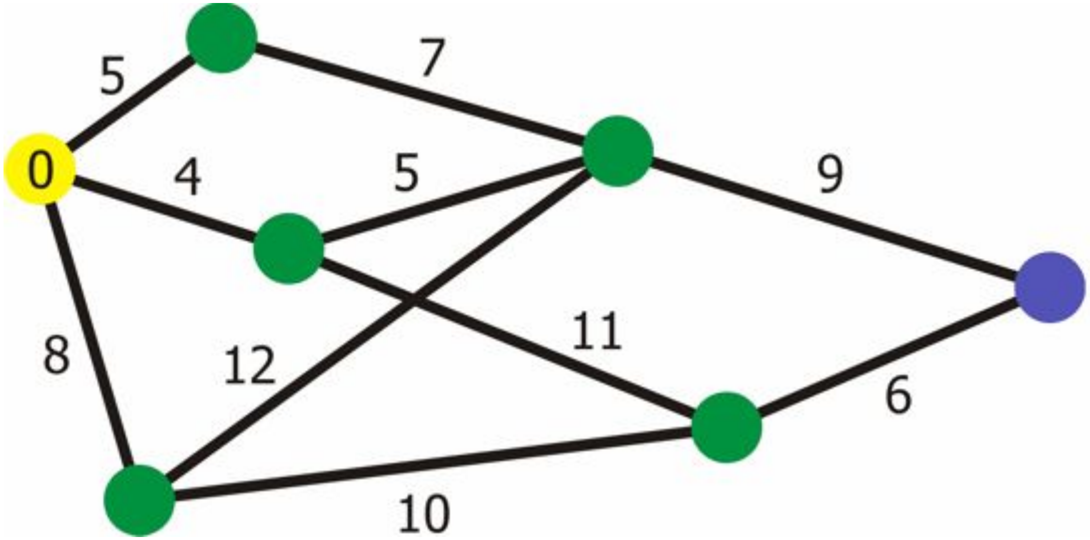
The choice of algorithm depends on the **problem's requirements.**

What is next?

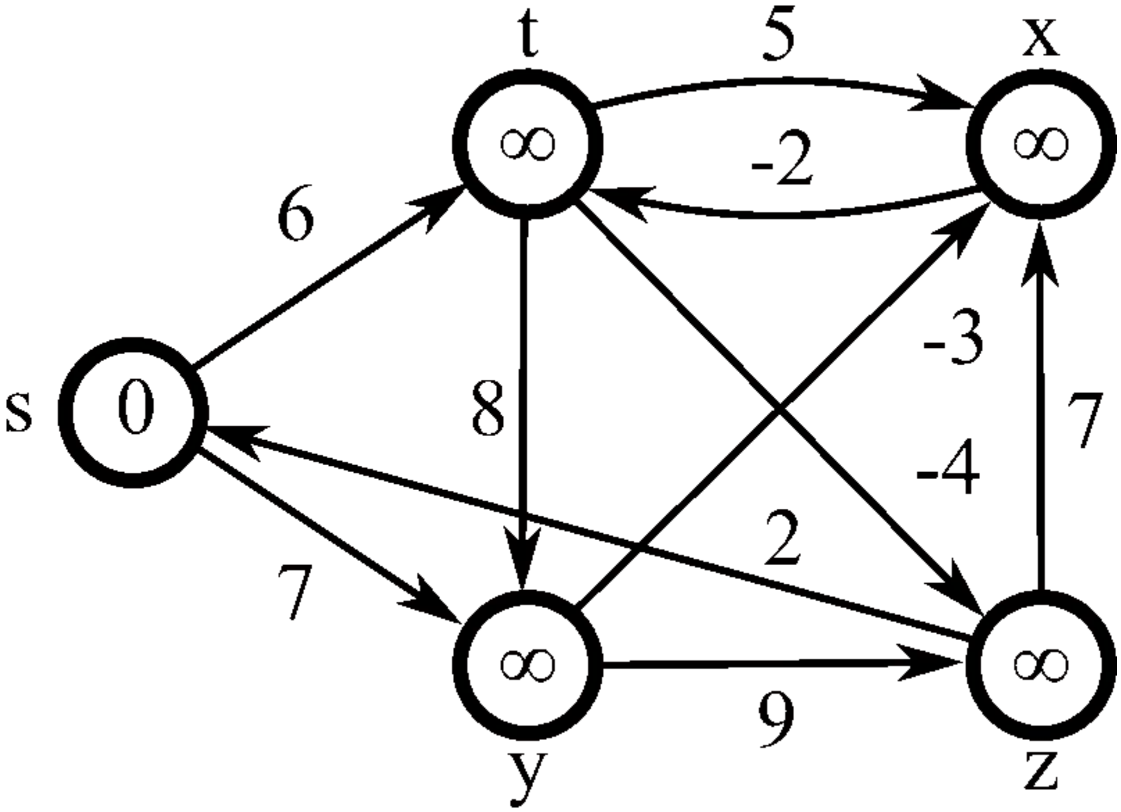
Graph Traversal: DFS and BFS



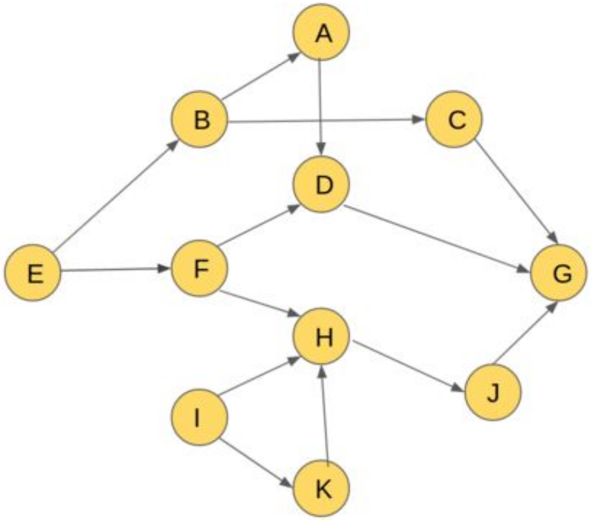
Shortest path: Dijkstra



Shortest path: Bellman Ford



Topological Sort



RESULT :

Exercise problems...

1. Operations on graph
2. Cities and roads
3. From adjacency matrix to adjacency list
4. From adjacency list to adjacency matrix
5. Regular graph
6. Sources and sinks

For more graph representation Problems: Link

Quote of The Day

"You can always recognize truth by its beauty and simplicity."

- Richard P. Feynman