Fundamental Data Structures

I. Intro to Data Structures

- A data structure is a specific way to store & organize data to make operations like insertion, deletion, searching, & updating faster
    - Ex: List, stack, queue, tree, graph, etc.
- No single data structure works well for all purposes
    - It's important to know the strengths and limitations of each data structure

Common DS operations

Insertion
Deletion
Searching
Updating

II. Linear Data Structures

- In the linear data structure, the data is organized in a linear order

III Array

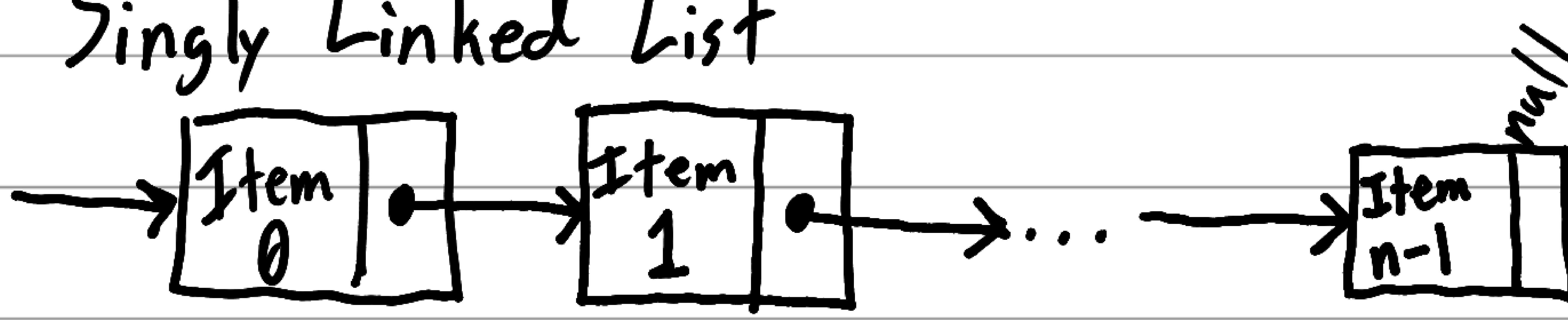
It provides constant access time to an item regardless of the position

Vectors (C++) and ArrayList (Java) recommended

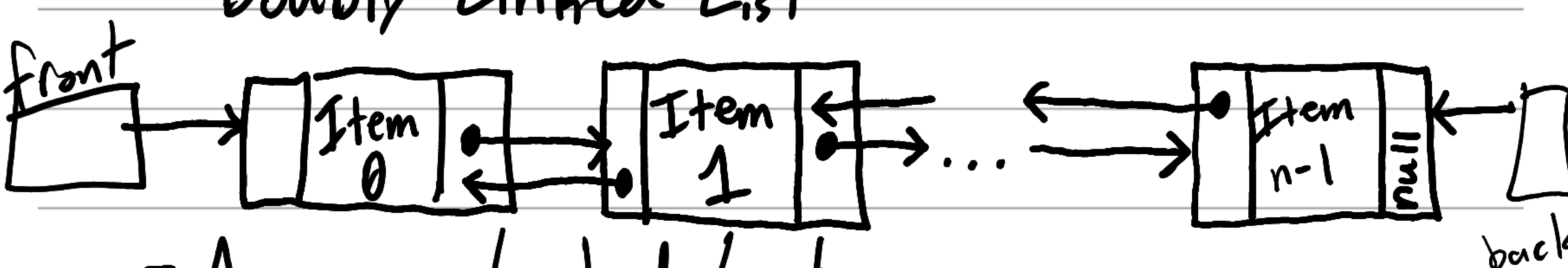
IV. Linked List

A linked list consists of nodes where each node contains a data part & a link part

Singly Linked List



Doubly Linked List



- Array vs. Linked List

Array

Pros	Cons
· lookup time (01)	· Can have empty memory (indicies)
· Insertion	· Shifts to insert or delete
· One chunk of memory	

Linked List

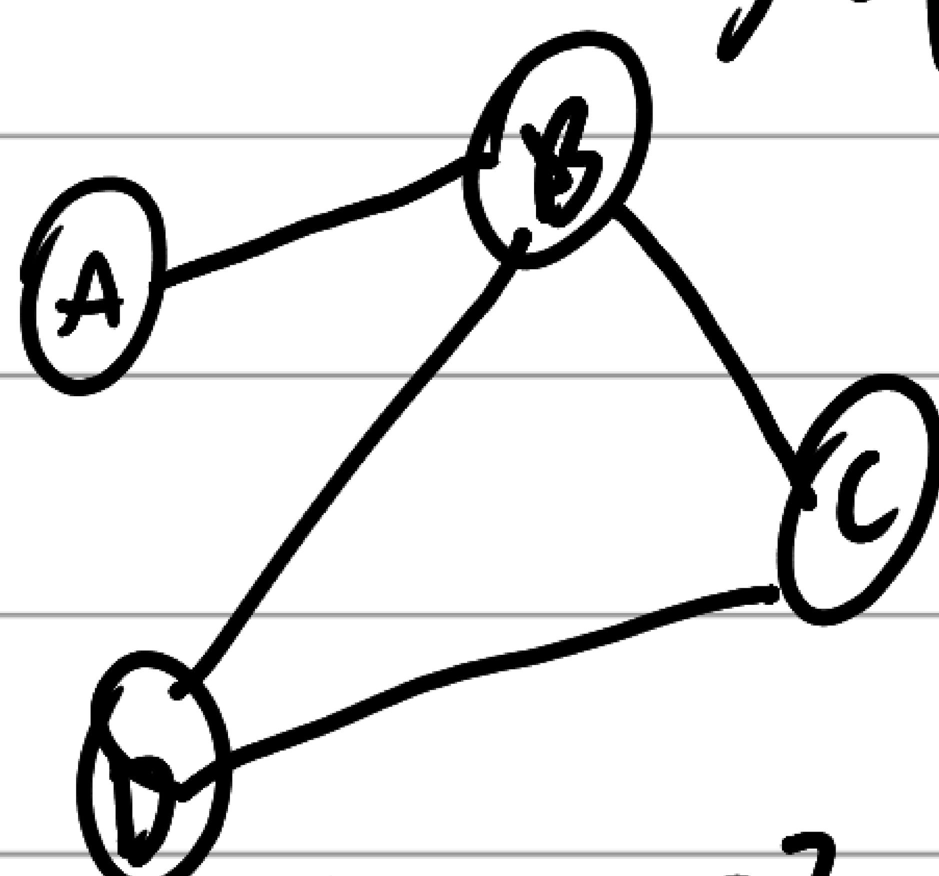
Pros	Cons
· No empty memory	· More memory - Pointer
·	· Access time
	· Insertion/deletion - shifts needed

- Stack
  - Last in First out (LIFO)
- Queue
  - First in First out (FIFO)

V. Introduction to Graphs

- Composed of nodes & edges

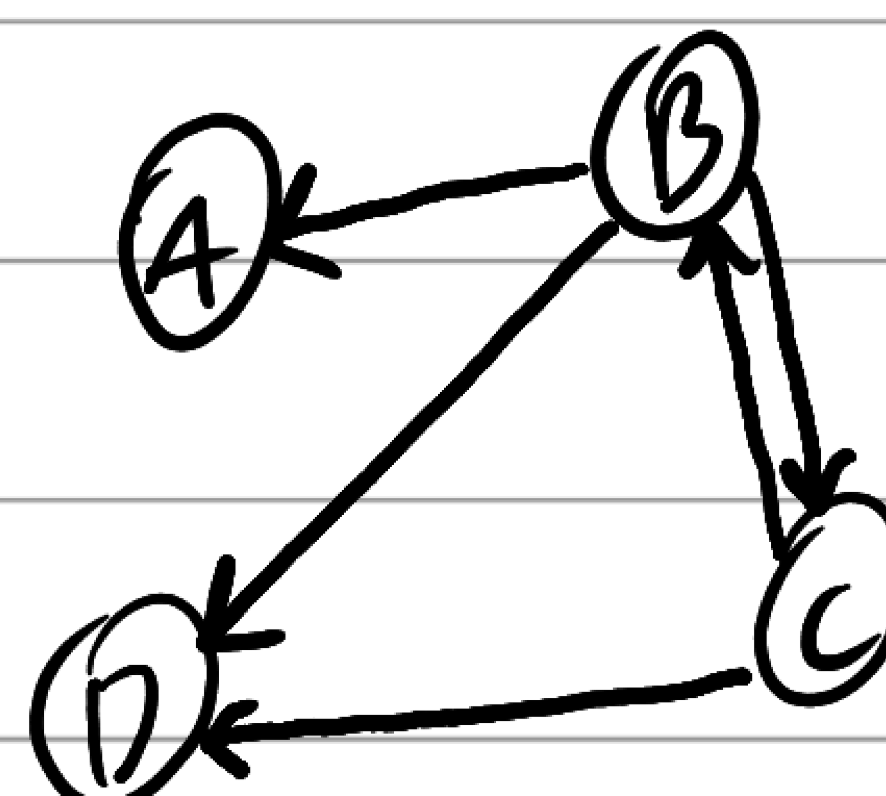
Undirected graph



$V = \{A, B, C, D\}$

$E = \{(A,B), (B,C),$
$(B,D), (C,D)\}$

Directed Graph

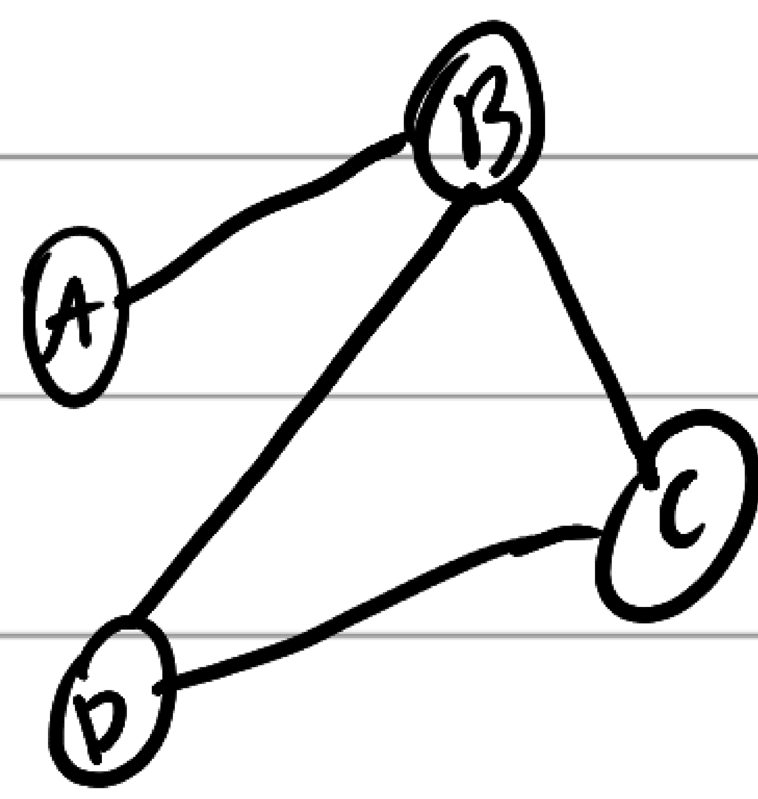


$V = \{A, B, C, D\}$

$E = \{(B,A), (B,C), (B,D),$
$(C,B), (C,D)\}$

- Use alphabetical order (ascending), for undirected graph edges

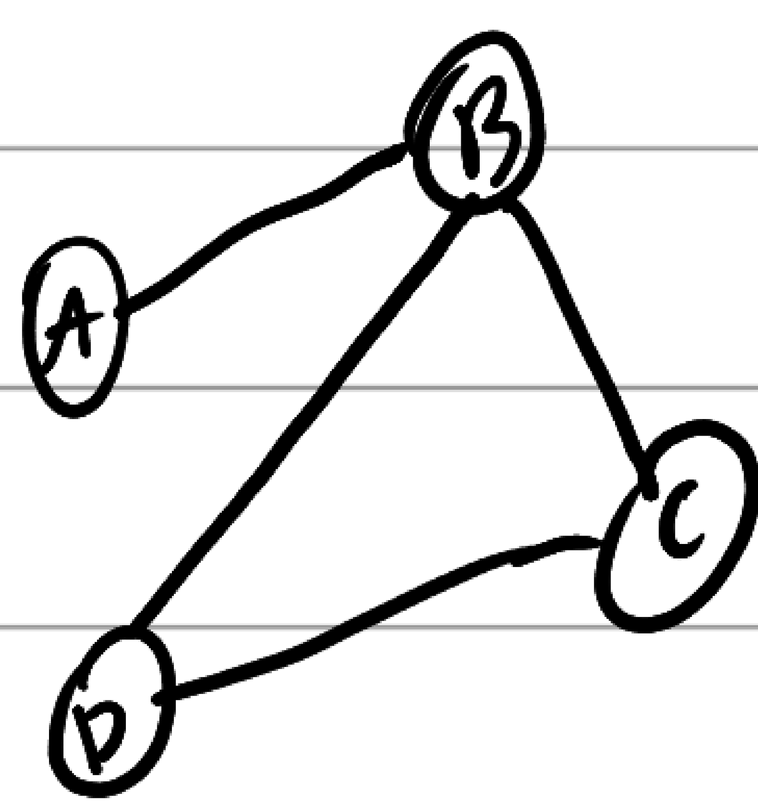
- Graph Representation: Adjacency Matrix
  - 2D-Array

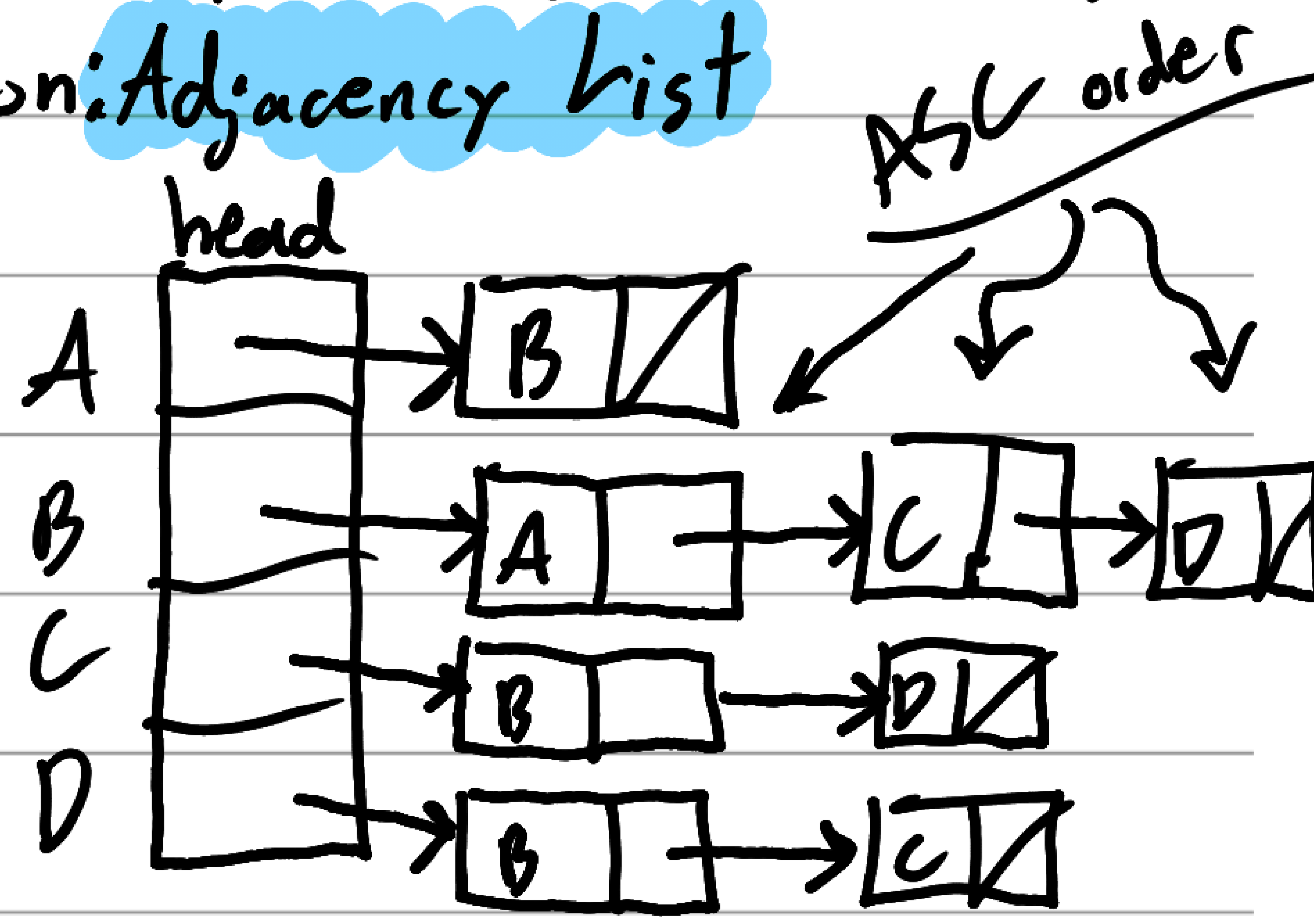


$$
\begin{array}{c|cccc|}
 & A & B & C & D \\
\hline
A & 0 & 1 & 0 & 0 \\
B & 1 & 0 & 1 & 1 \\
C & 0 & 1 & 0 & 1 \\
D & 0 & 1 & 1 & 0 \\
\hline
\end{array}
$$

  - Maps edges of a graph (1 = edge, 0 = no edge)

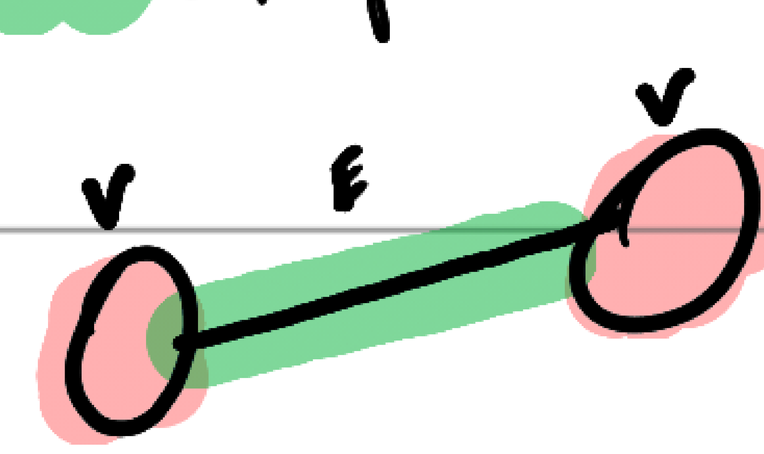
- Graph Representation: Adjacency List





Set: Unordered collection of distinct items (can be empty)

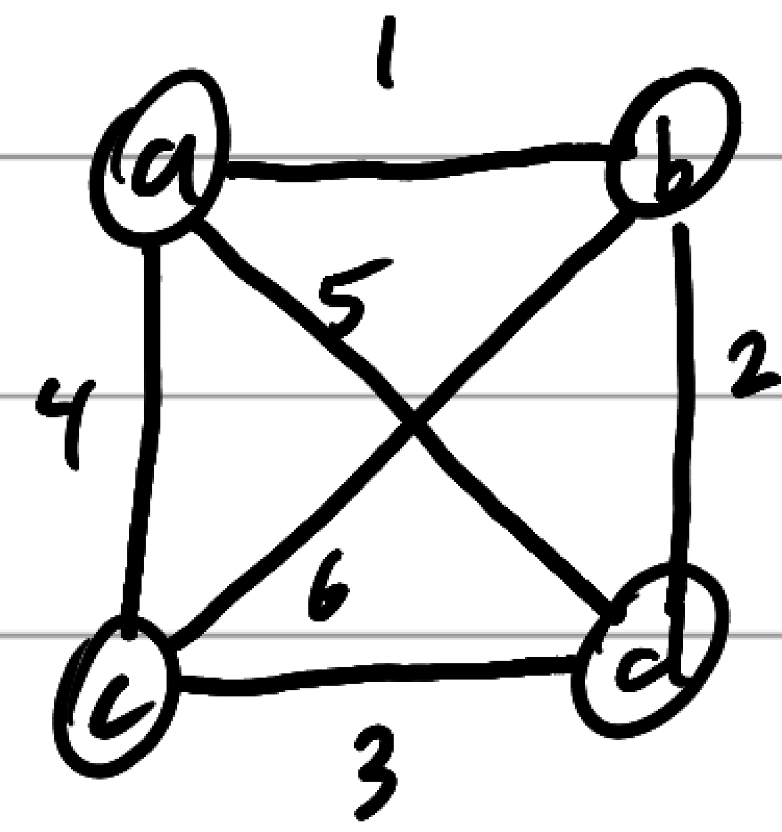
- Graph Definition
  - A graph, $G = \langle V, E \rangle$, is composed of a pair of two sets: a finite nonempty set V of items called vertices & a set E of pairs of these items called edges



- $|V|$ and $|E|$ represent total number of vertices and edges respectively

$|V| = 4$

$0 \le |E| \le 6$



- Max number of edges (Undirected)

$$\frac{(|V|-1) \times |V|}{2}$$

$$\frac{\overset{(|V|-1)}{3} \cdot \overset{|V|}{4}}{2} \rightarrow \overset{|E|}{6}$$

- Max number of edges (Directed)

$$\frac{(|V|-1) \times |V|}{2} \times 2$$

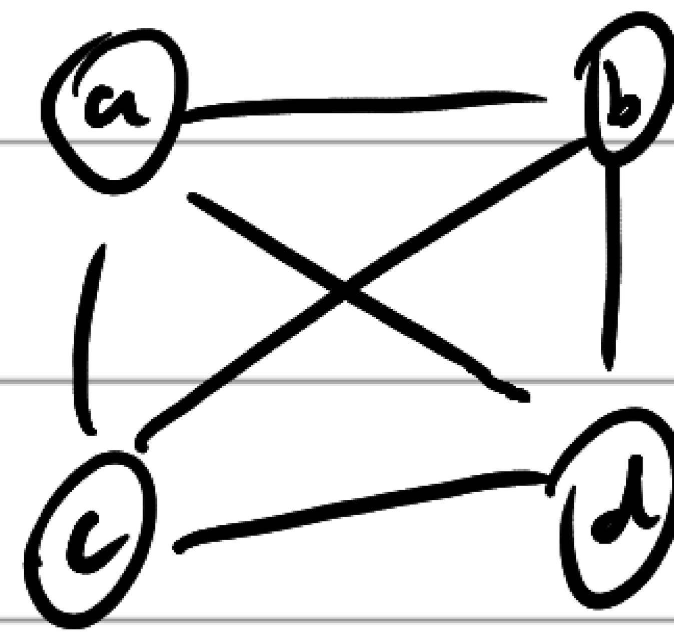
$$\frac{3 \cdot 4}{2} \times 2 = 12$$

☆ . Useful Formula for CS

$$\sum_{i=1}^{n} = 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$

- Complete Graph

- Every pair of vertices in the graph is connected by an edge

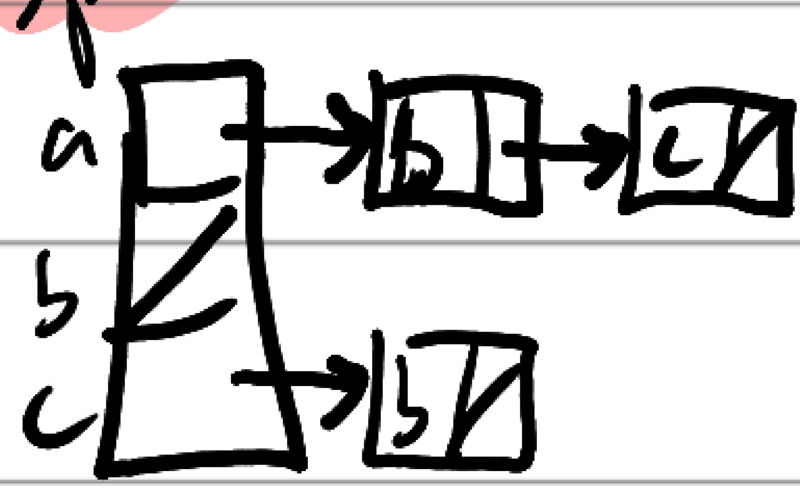


- Dense & Sparse

- Dense Graph: The number of edges is close to the maximal number of edges
- Sparse Graph: Opposite of dense graph

<u>Space consideration</u>

Adj List better for Sparse Graph

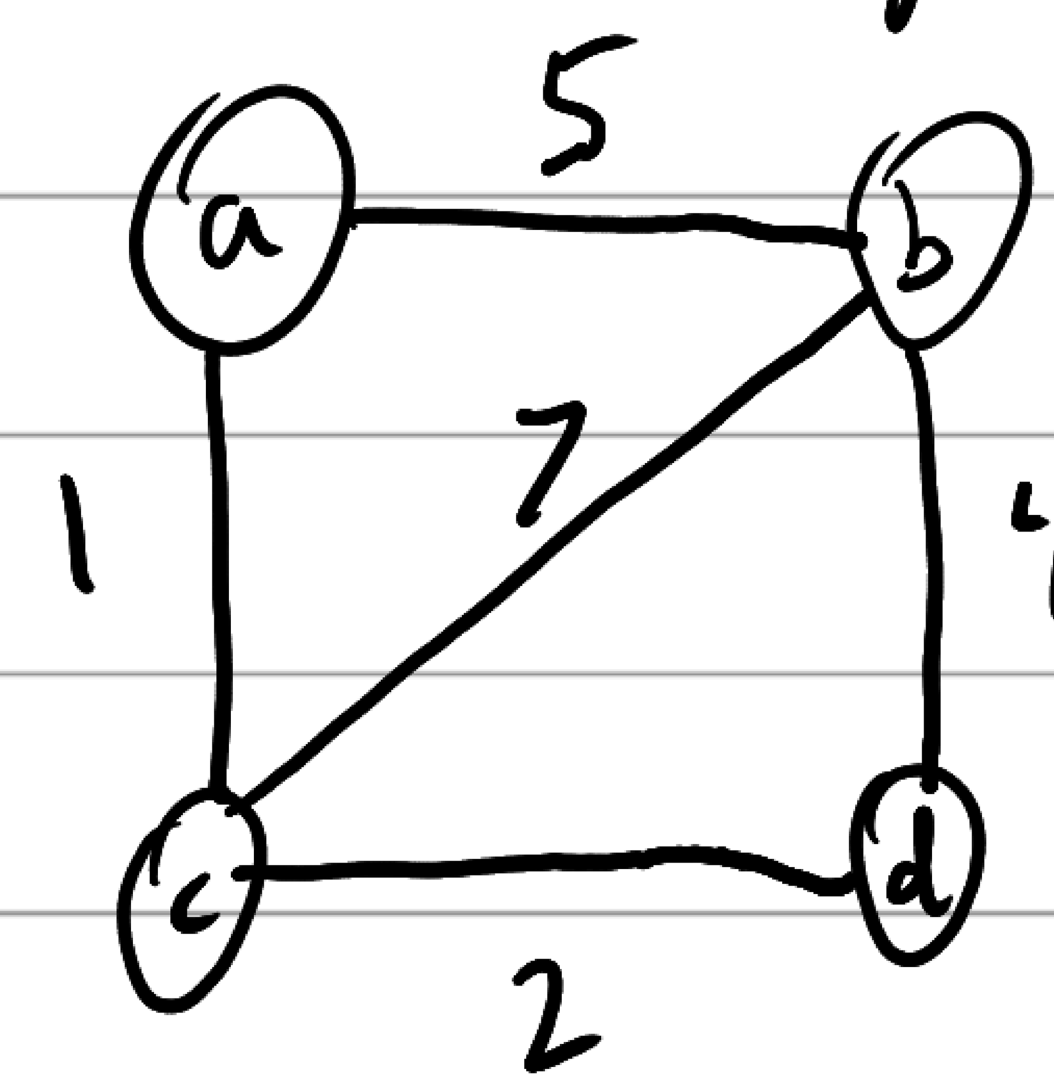


Adj Matrix better for Dense Graph

$$\begin{array}{c|c} & \begin{matrix} a & b & c \end{matrix} \\ \begin{matrix} a \\ b \\ c \end{matrix} & \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

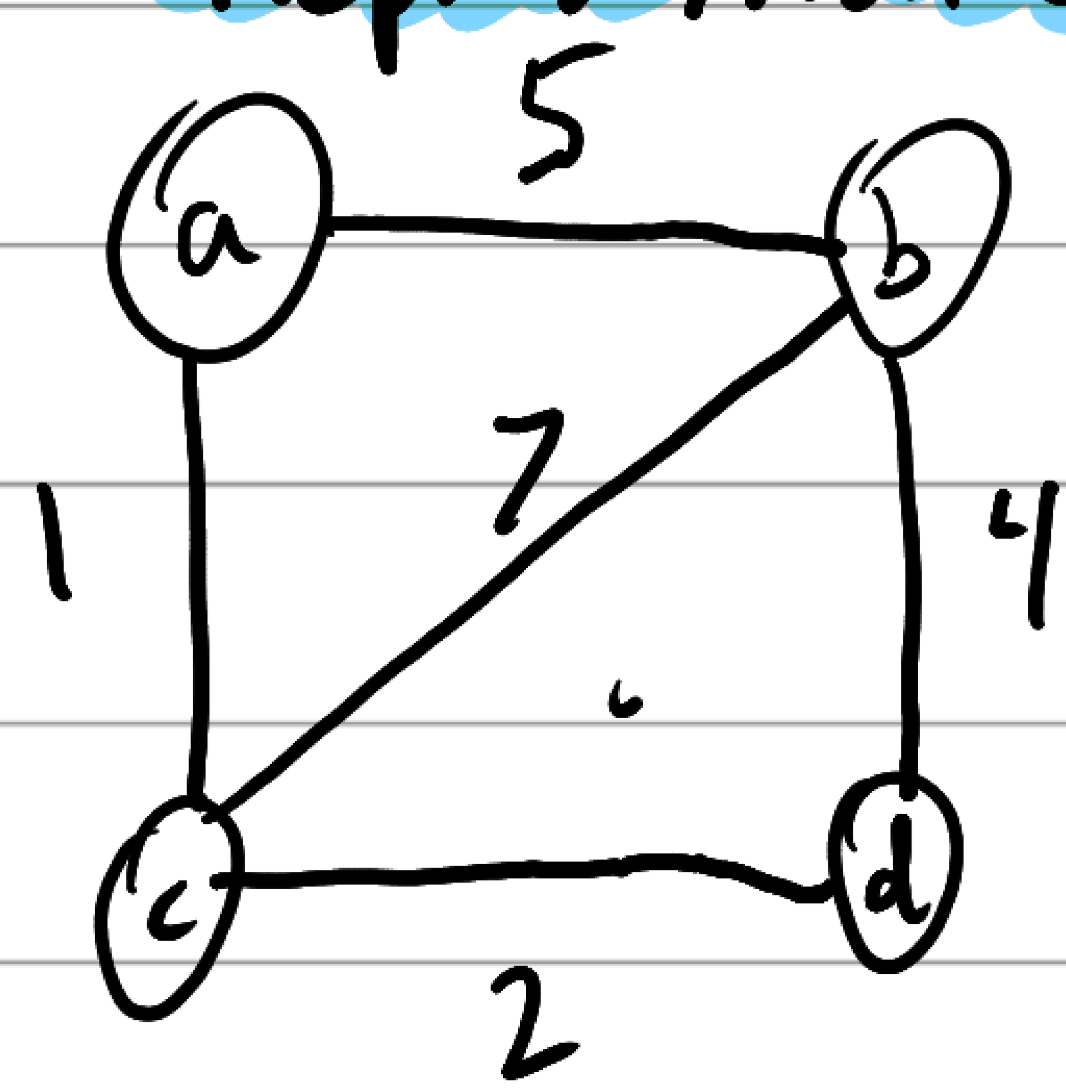
VI. Weighted Graphs

- A weighted undirected graph & a weighted directed graph are graphs w/ numbers (costs) assigned to their edges



- Representation: Adjacency Matrix

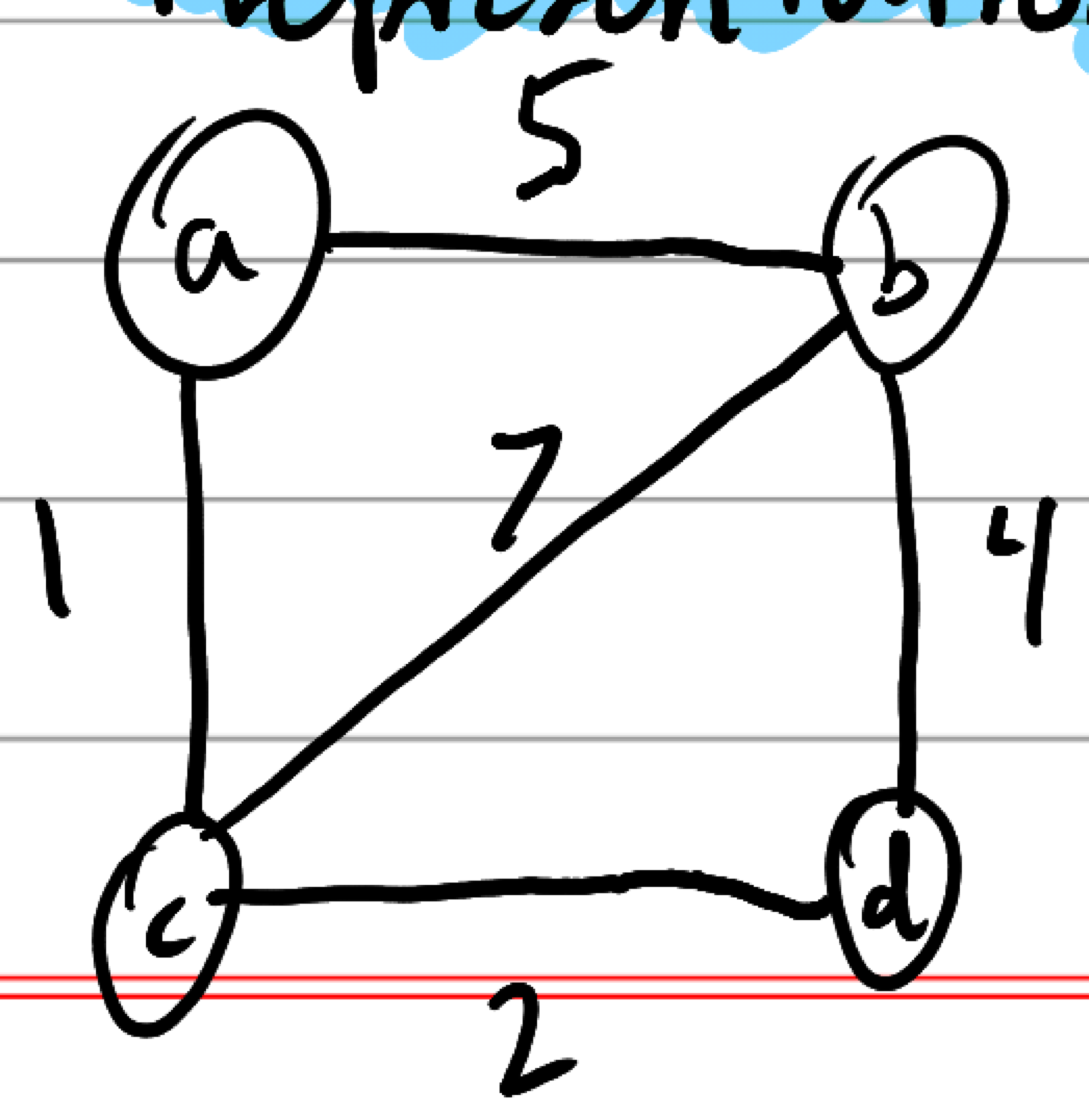
Use 0's for class

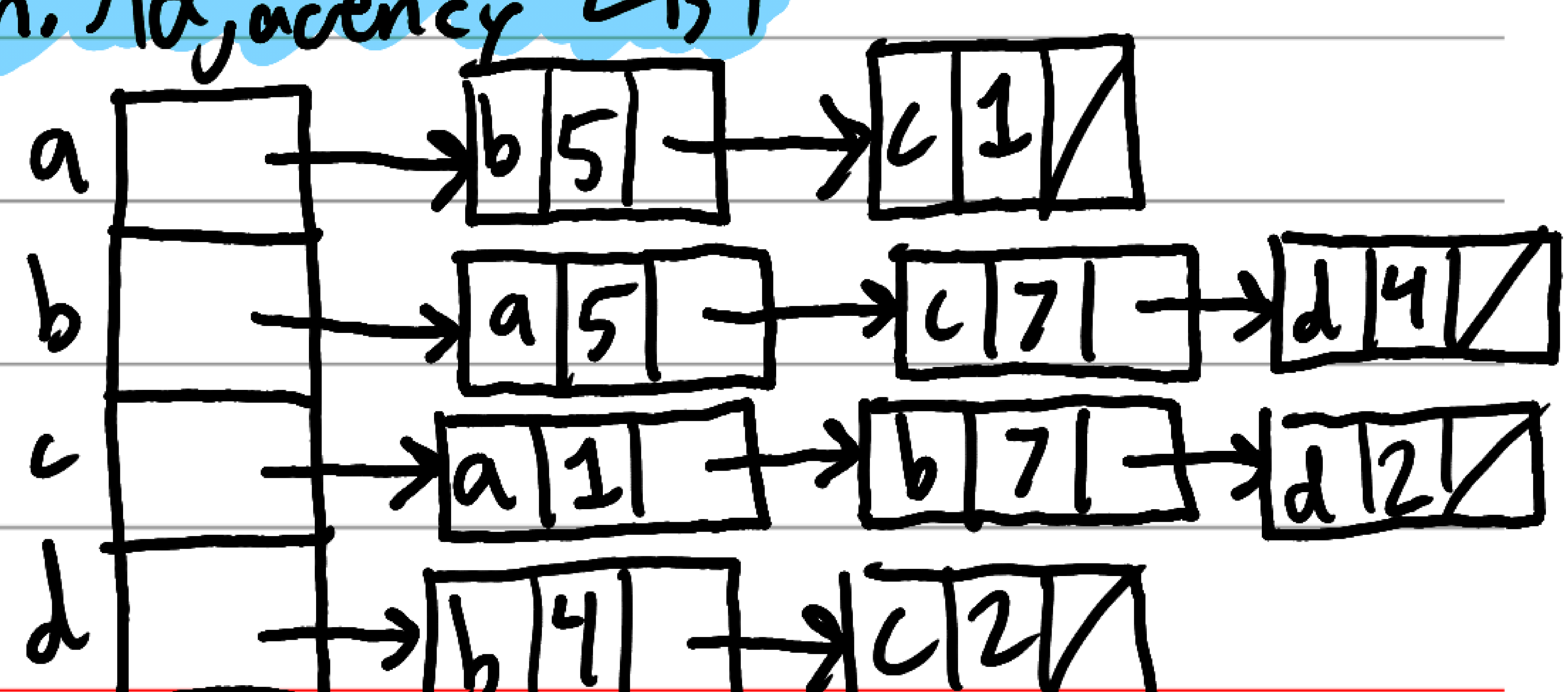


$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} \infty & 5 & 1 & \infty \\ 5 & \infty & 7 & 4 \\ 1 & 7 & \infty & 2 \\ \infty & 4 & 2 & \infty \end{bmatrix} \end{array}$$

main diagonal can be made 0's

- Representation: Adjacency List





VII. Graph Vocabulary

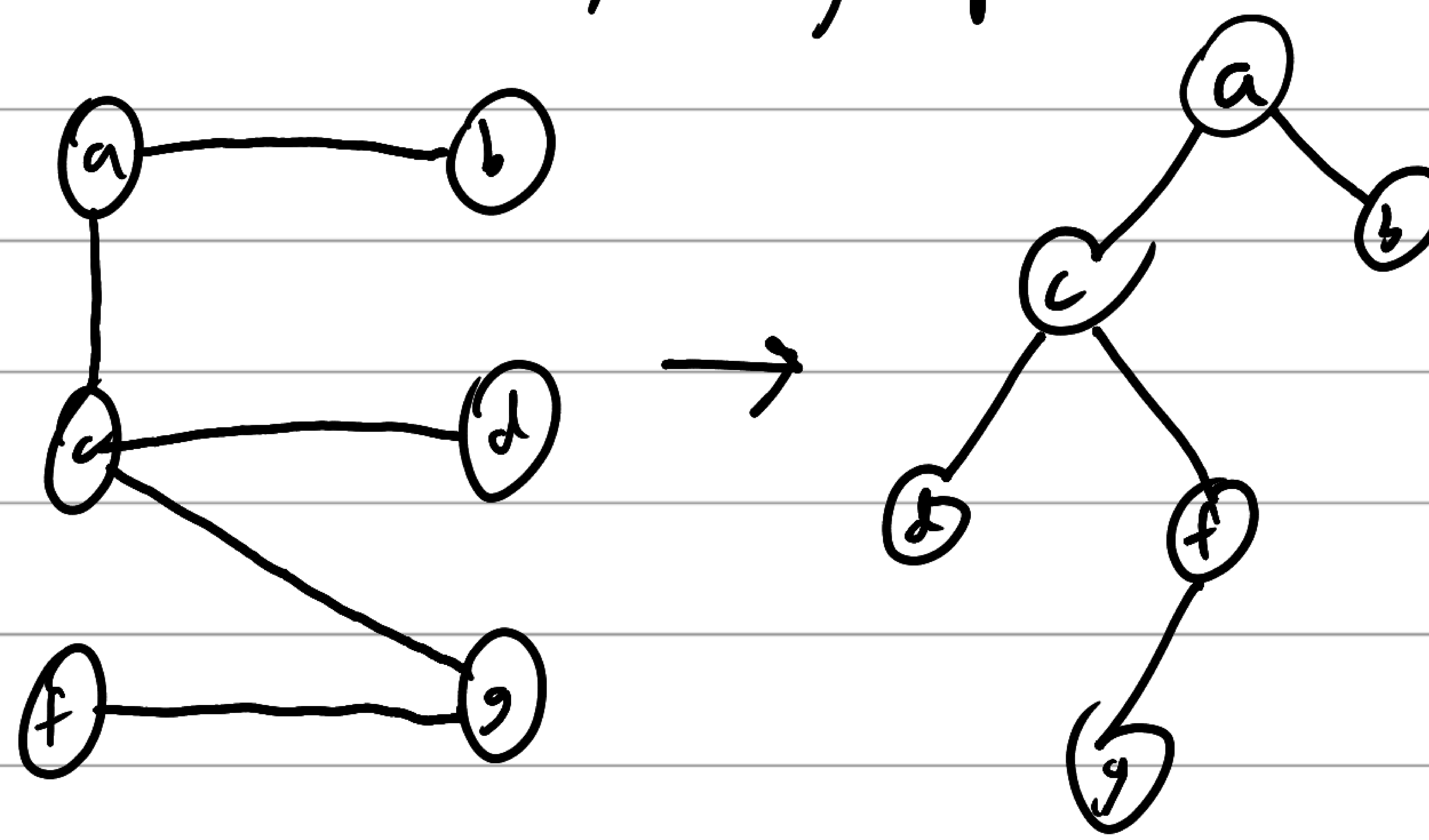
Length
Simpath
Connected
Conn compon
Cycle
Acyclic cy

- **Path:** Sequence of edges connecting a sequence of vertices representing route from origin to destination node
- **Length:** Number of edges or sum of edge weights traversed to connect a sequence of vertices
- **Connected:** Graph where a path exists between every pair of vertices, meaning all nodes reachable from each other.
- Connected Component:
- **Cycle:** A path of edges & vertices where a vertex is reachable from itself, start & end at same node w/o repeating edges
- **Acyclic:** A graph containing no cycles

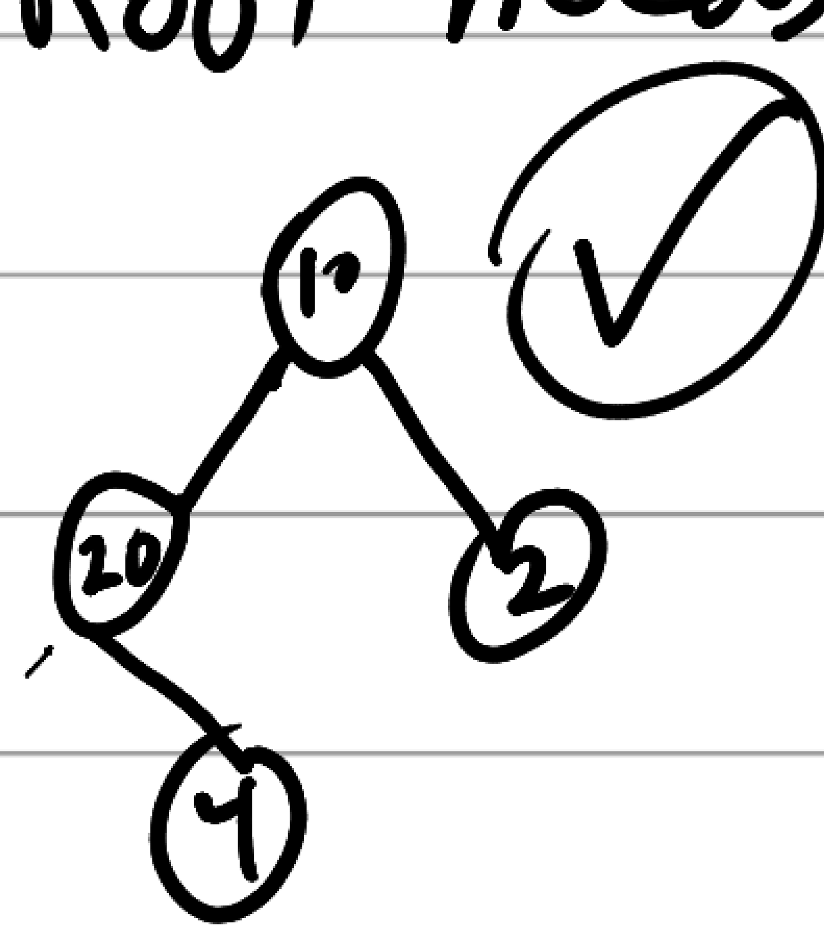
VIII. Trees

- Note that tree is actually a graph
    - Connected, acyclic graph

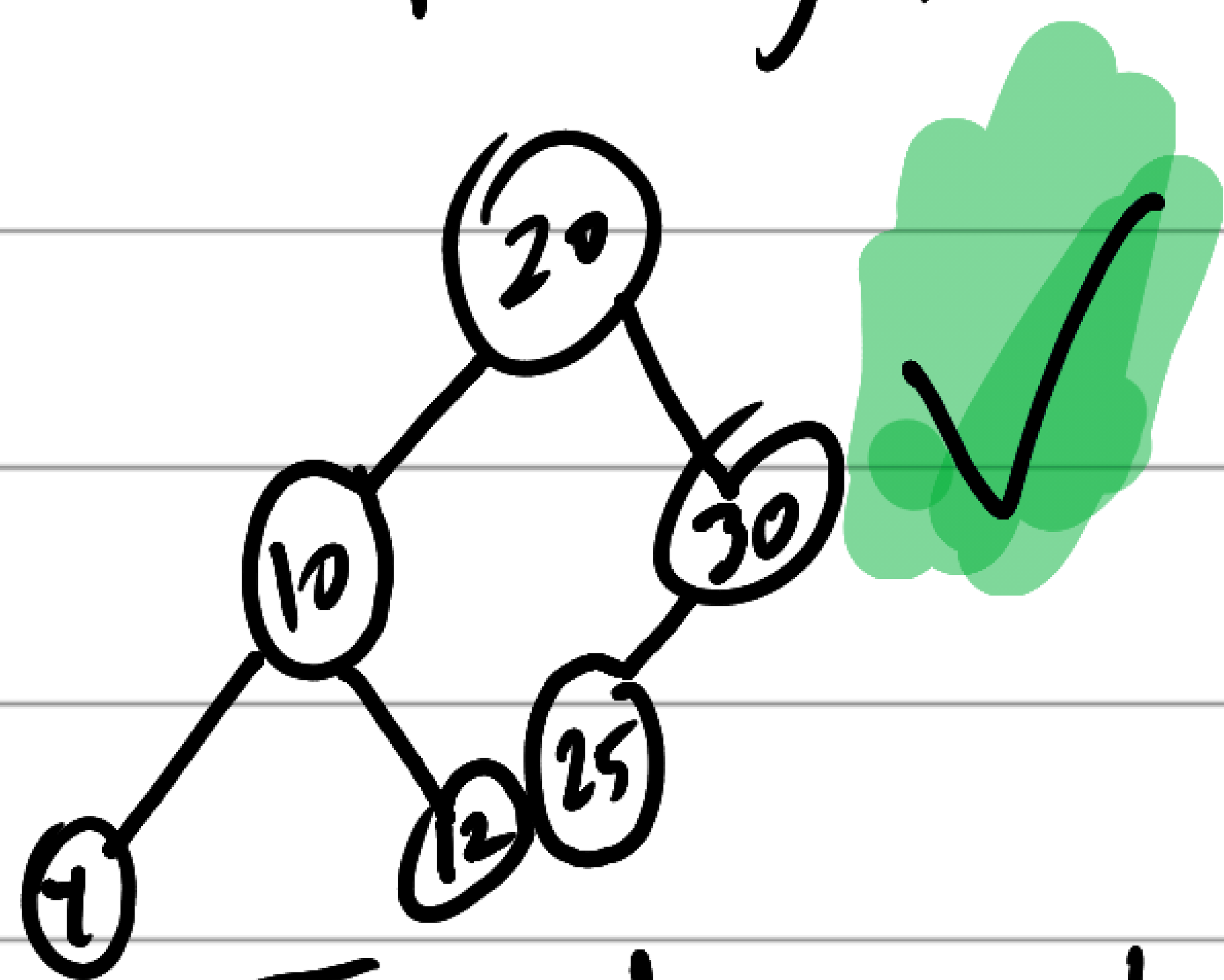
Ex.

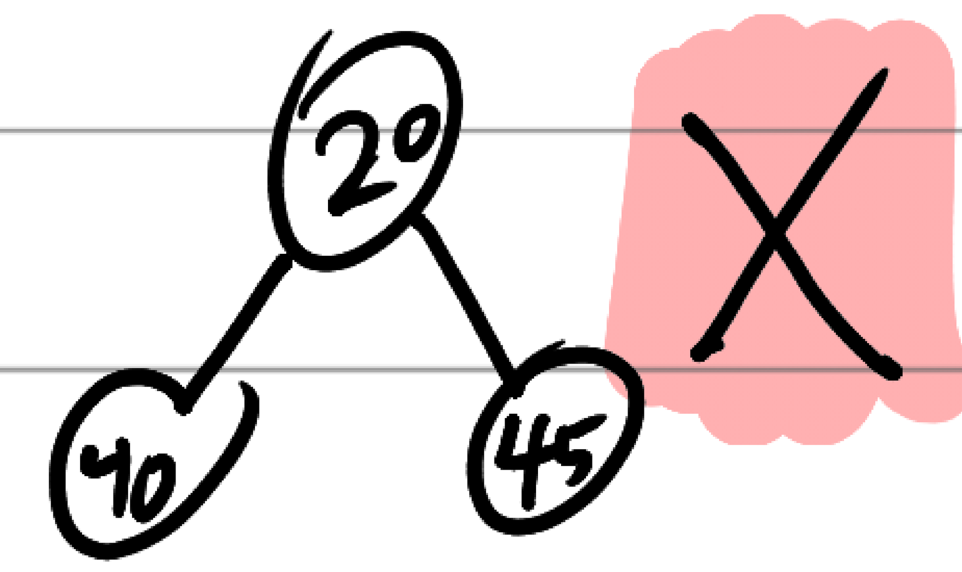


- Different kinds of trees
    - Main Interest: Binary Tree & Binary Search Tree
    - Free Tree, Rooted Tree, Ordered Tree
- Binary Tree
    - Root needs left & right child



- Binary Search Tree (BST)
  - All left children are less than parent
  - All right children are more than parent





- Implementation: Binary Tree