Graph Structures

A Review of Basic Concepts

Basic Graph Definitions

A graph is a mathematical object that is used to model different situations ,objects and processes:

Linked list

Tree (partial instance of graph)

Flowchart of a program

City map

Electric circuits

Course curriculum

Vertices and Edges

Definition: A graph is a collection (nonempty set) of vertices and edges

Vertices: can have names and properties

Edges: connect two vertices,

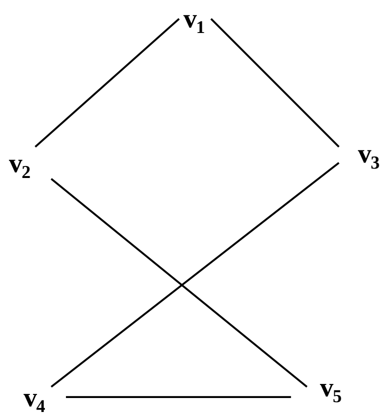
can be labeled,

can be directed

Adjacent vertices: there is an edge between them

BASIC DEFINITIONS:Example

- *Graph*: G = {V, E}
- $V = \{v_1, v_2, \ldots, v_n\}$, $E = \{e_1, e_2, \ldots e_m\}$, $e_{ij} = (v_i, v_j, l_{ij})$
- A set of *n* nodes/vertices, and *m* edges/arcs as pairs of nodes: *Problem sizes: n, m*
- When l_{ij} is present, it is a *label*, if it is a number it is the *weight* of an edge.



$V = \{v_1, v_2, v_3, v_4, v_5\}$
$E = \{e_1 = (v_1, v_2), (v_3, v_1), (v_3, v_4), (v_5, v_2), e_5 = (v_4, v_5)\}$

degree(v_1)=2, degree(v_2)=2, … :number of arcs

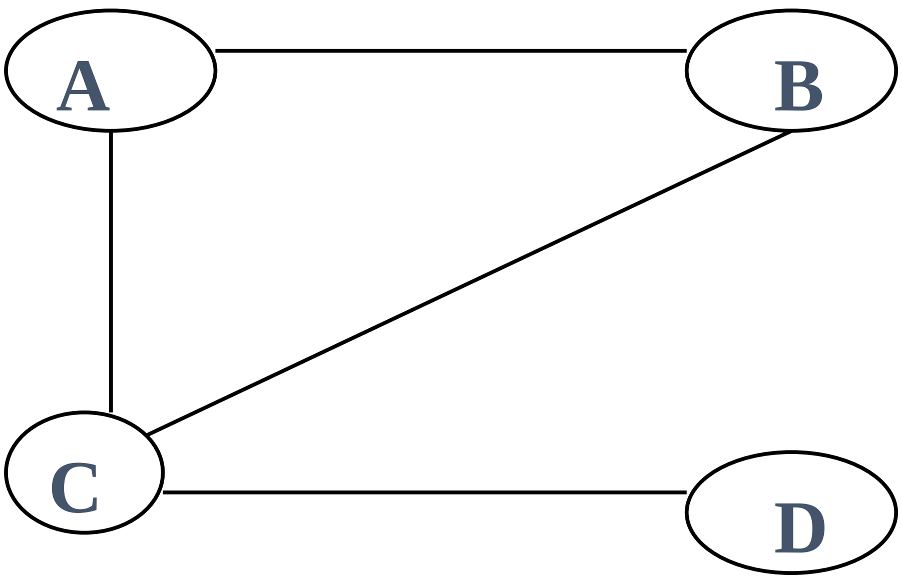
For directed graph: *indegree*, and *outdegree* may differ

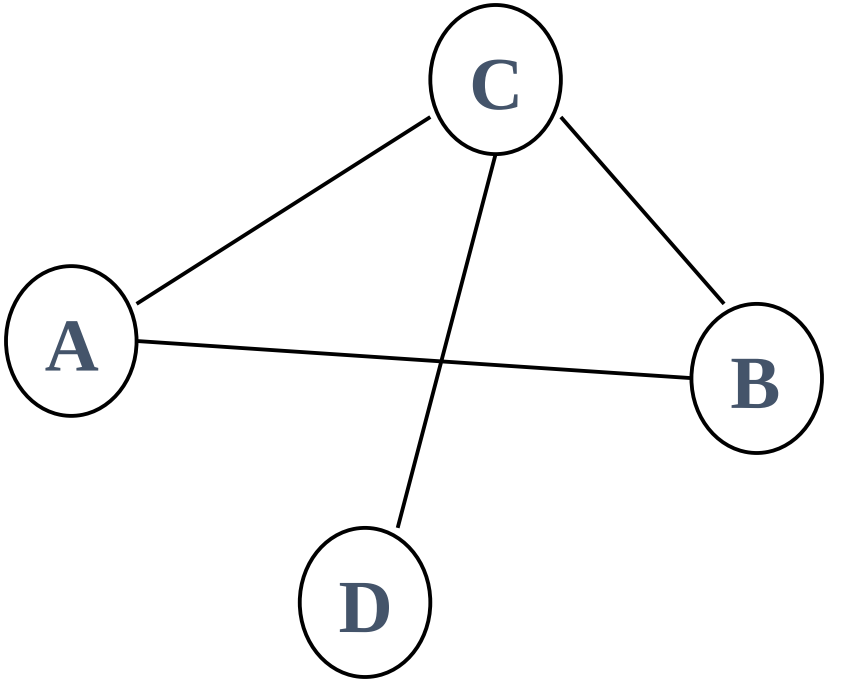
More Examples

Graph1:Undirected

Vertices: A,B,C,D

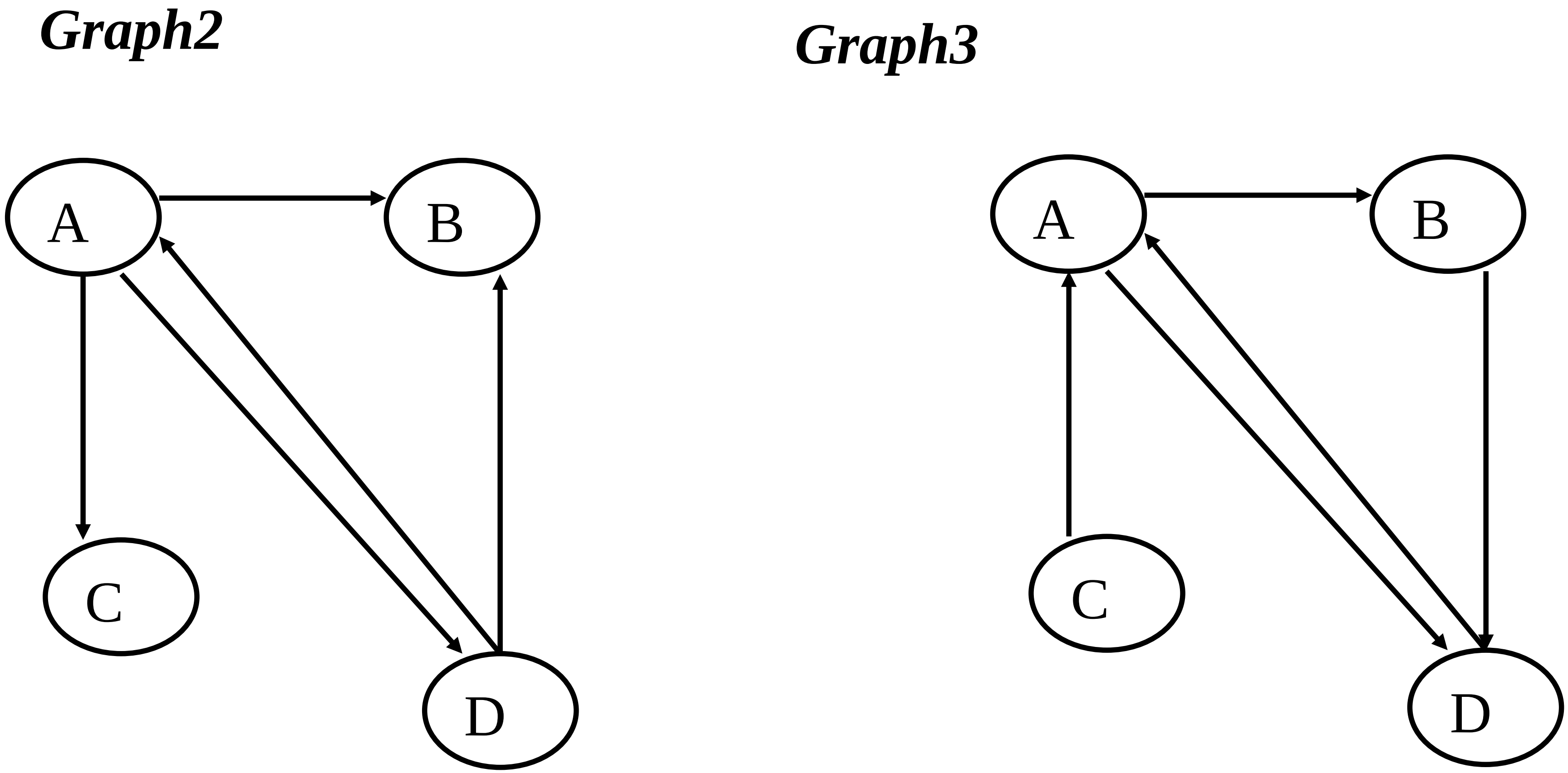
Edges: AB, AC, BC, CD





Two ways to draw the same graph

Directed and undirected graphs



These two are different directed graphs

More definitions : Path

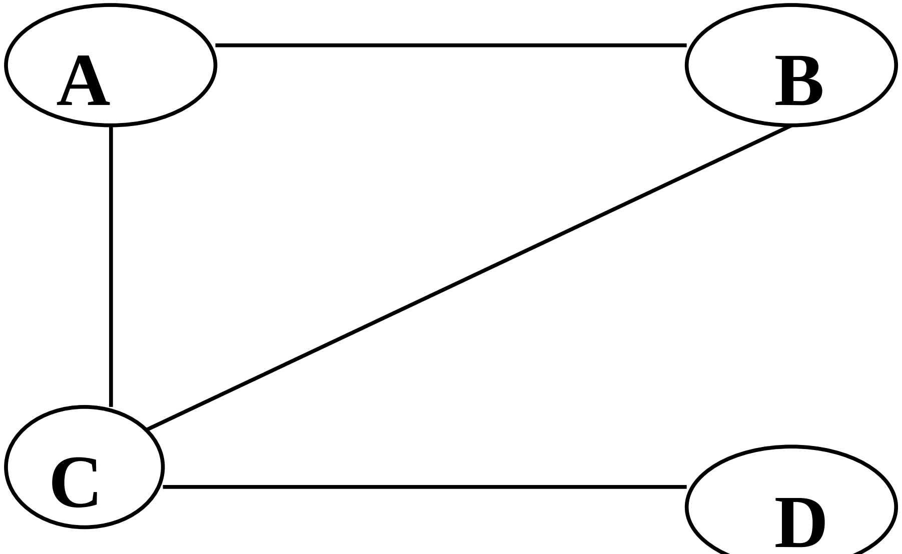
A list of vertices in which successive vertices are connected by edges

A B C

B A C D

A B C A B C A B C D

B A B A C



More definitions : Simple Path

No vertex is repeated.

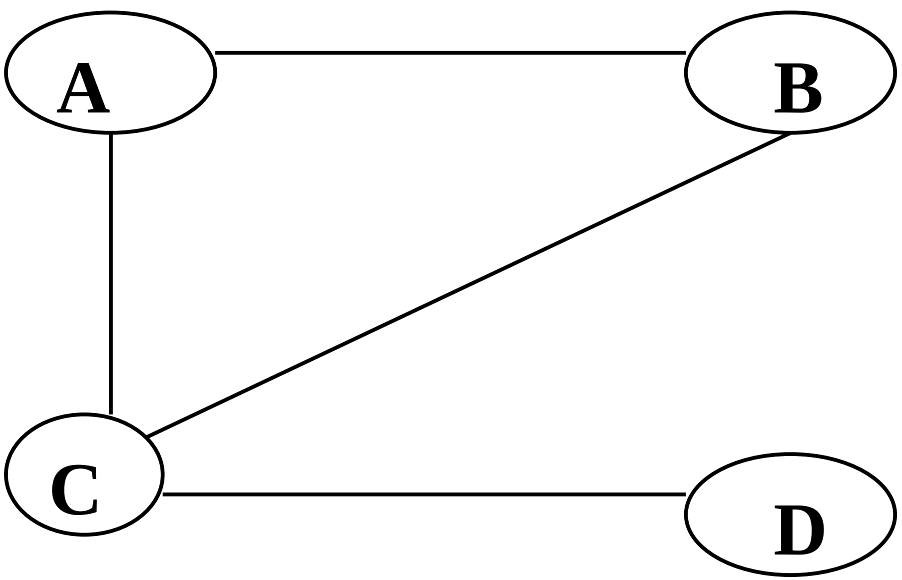
A B C D

D C A

D C B

A B

A B C



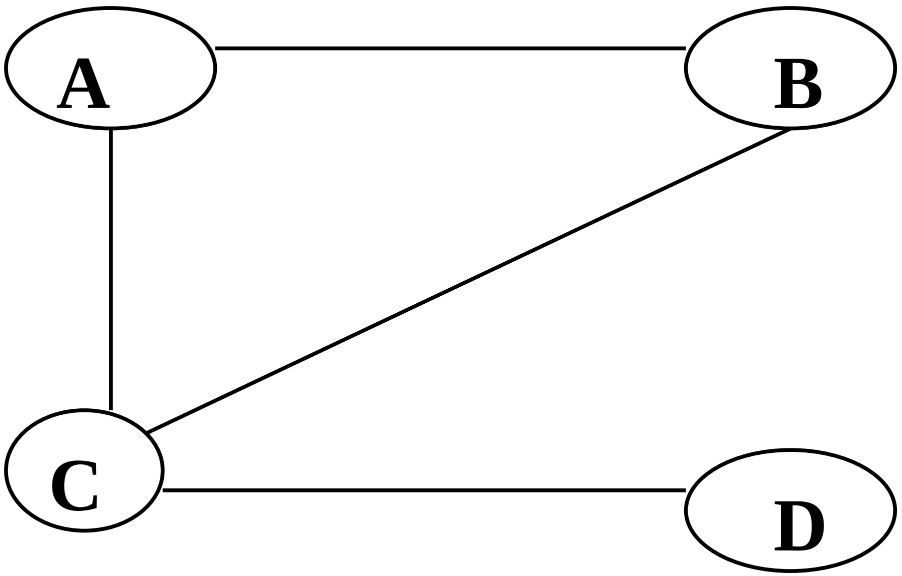
More definitions : Cycle

Simple path with distinct edges, except that the first vertex is equal to the last

A B C A

B A C B

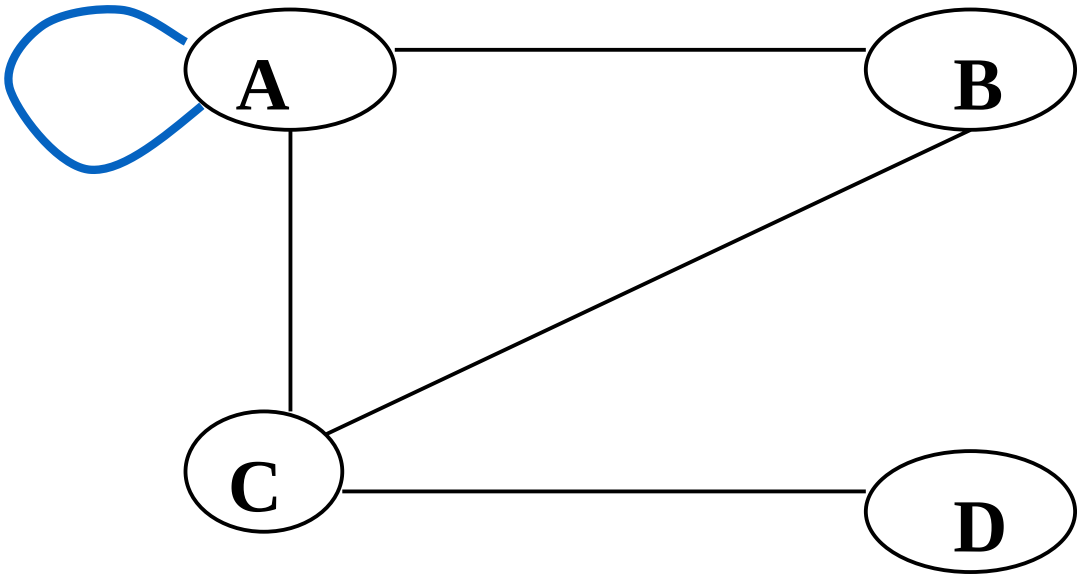
C B A C



A graph without cycles is called acyclic graph.

More definitions : Loop

An edge that connects the vertex with itself

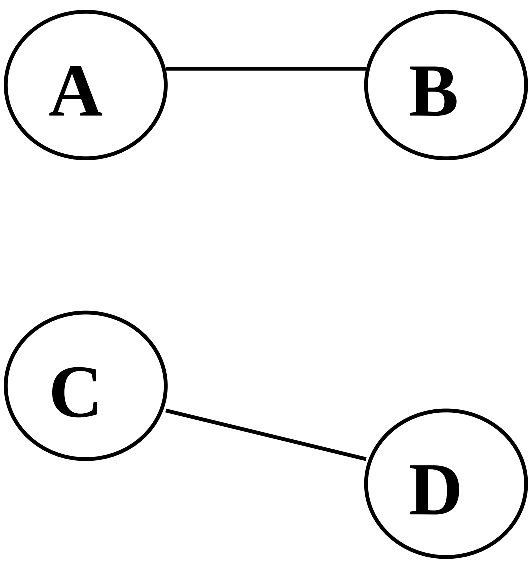


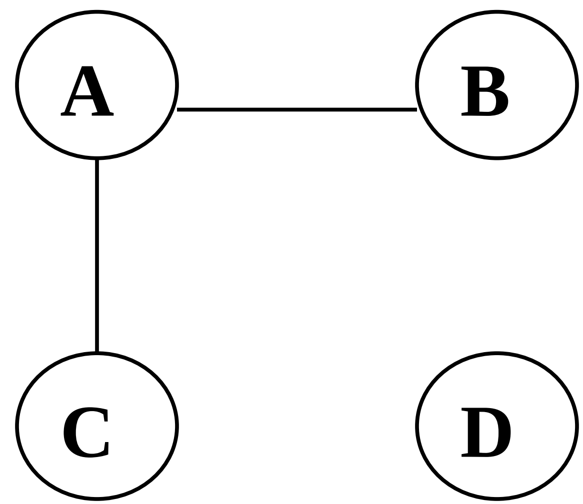
Connected and Disconnected graphs

Connected graph: There is a path between each two vertices

Disconnected graph : There are at least two vertices not connected by a path.

Examples of disconnected graphs:

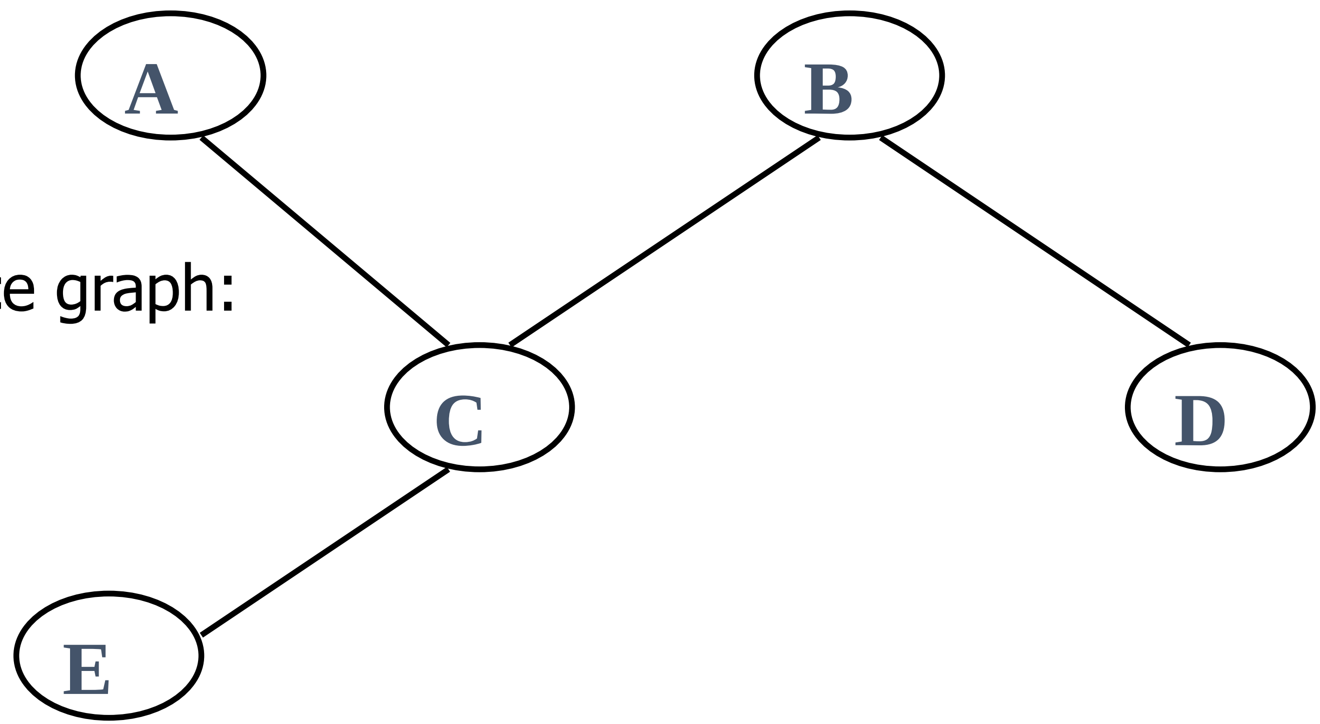




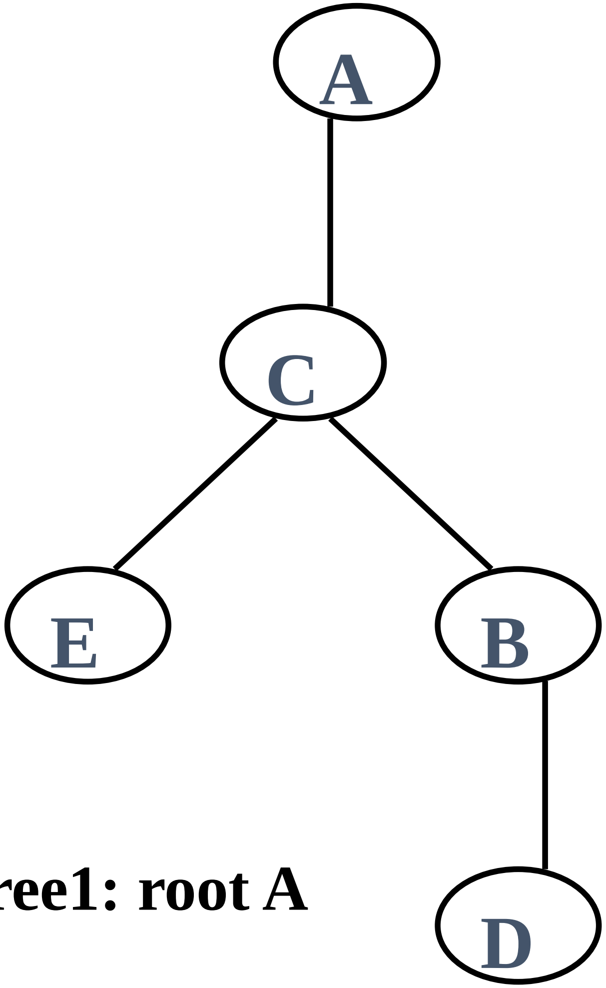
Graphs and Trees

Tree: an undirected graph with no cycles, and a node chosen to be the root

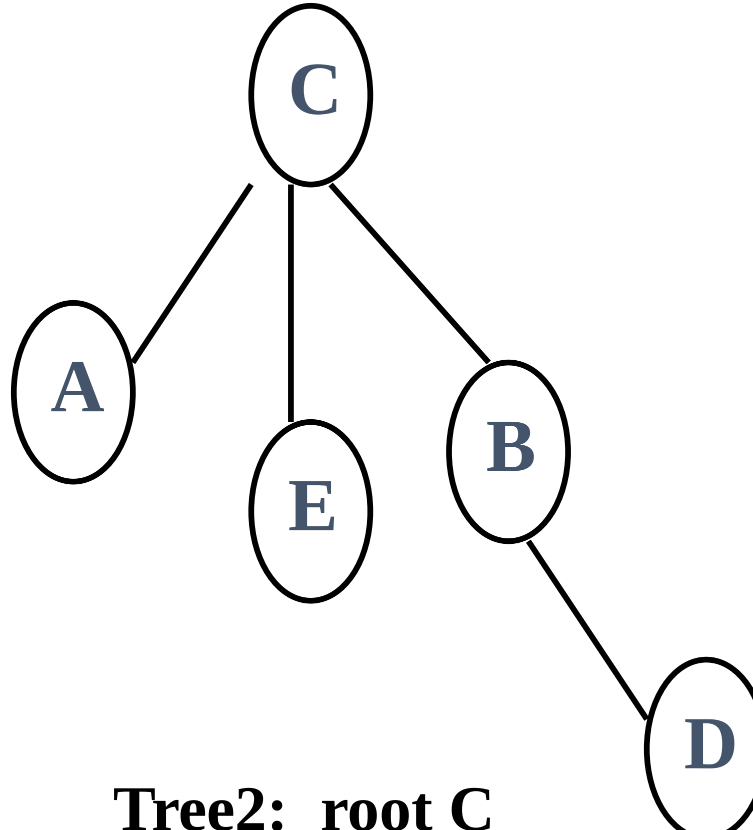
Source graph:



Graphs and Trees



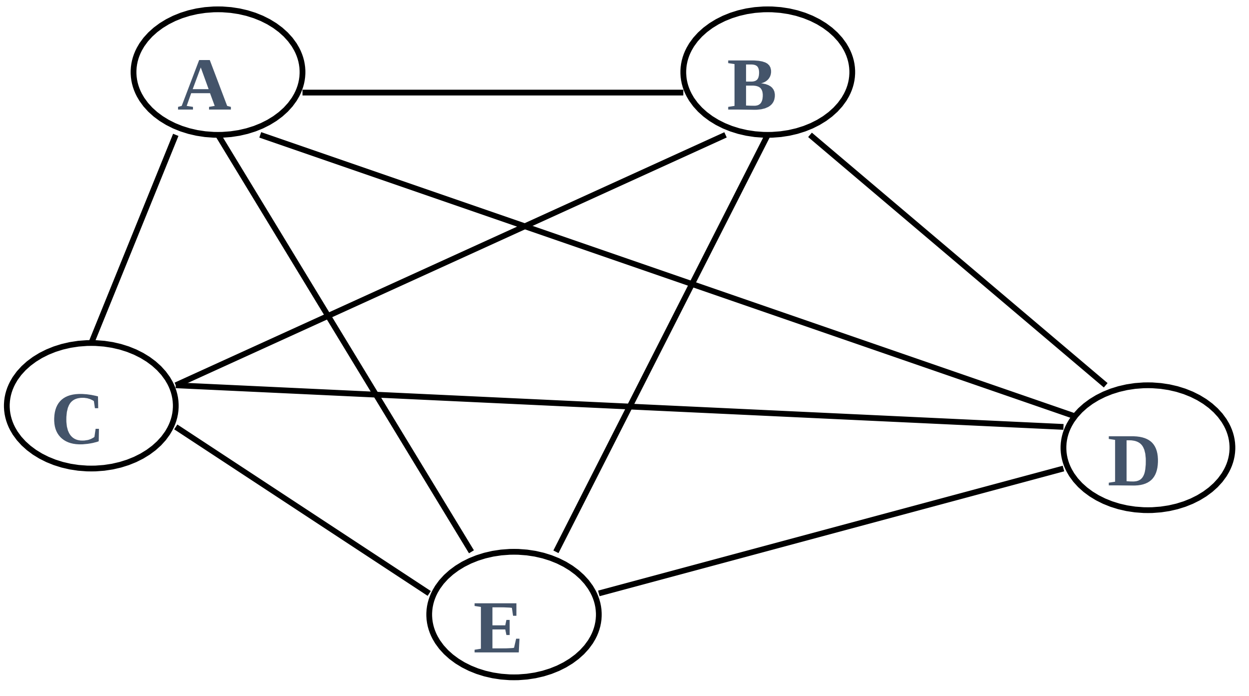
Tree1: root A



Tree2: root C

Complete graphs

Graphs with all edges present – each vertex is connected to all other vertices



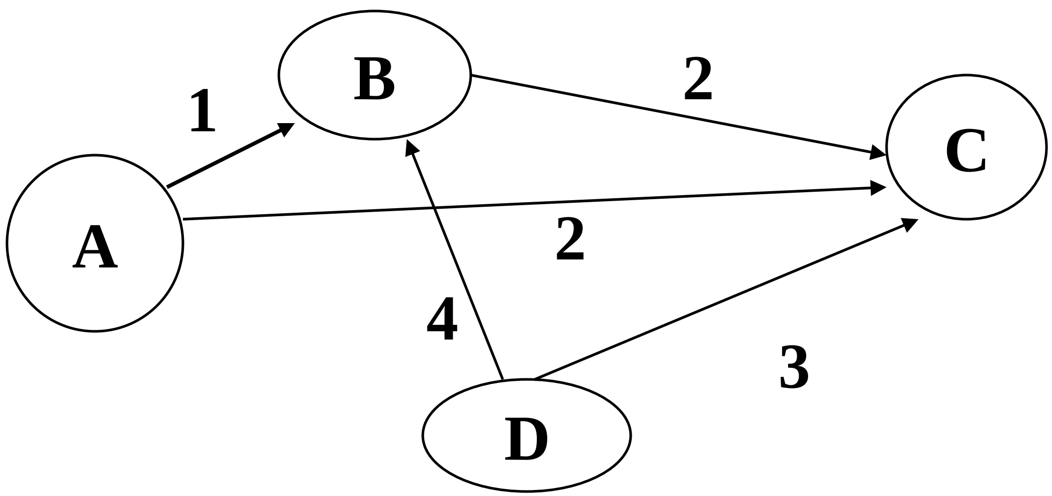
A complete graph

Sparse graphs: relatively few of the possible edges are present

Weighted graphs and Networks

Weighted graph – weights are assigned to each edge (e.g. road map)

Networks: directed weighted graphs



Graph Representation Methods

- Adjacency matrix
- Adjacency lists

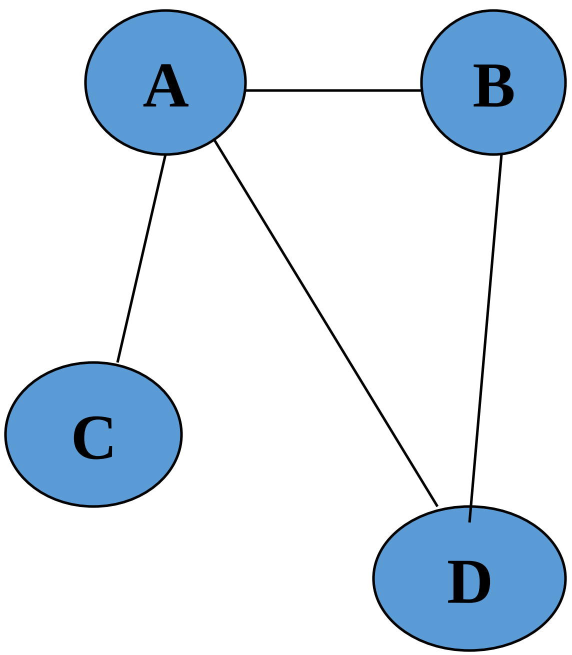
Adjacency Matrix: Undirected Graphs

Vertices: A,B,C,D

Edges:AC, AB, AD, BD

The matrix is symmetrical

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



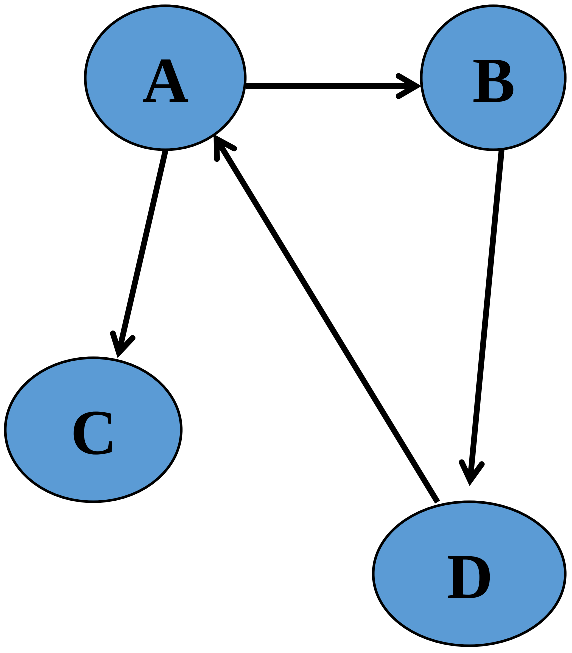
Adjacency Matrix: Directed Graphs

Vertices: A,B,C,D

Edges: AC, AB, BD, DA

The matrix is not symmetrical

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

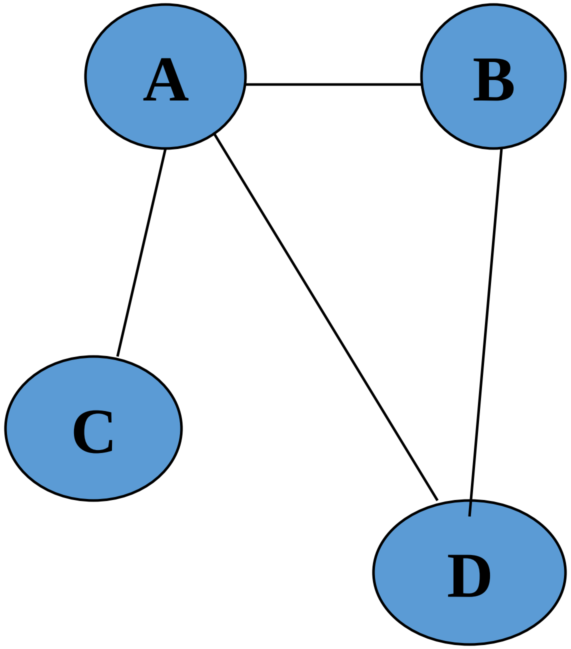


Adjacency lists:undirected graphs

Vertices: A,B,C,D

Edges:AC, AB, AD, BD

Vertex	lists
A	B C D
B	A D
C	A
D	A B

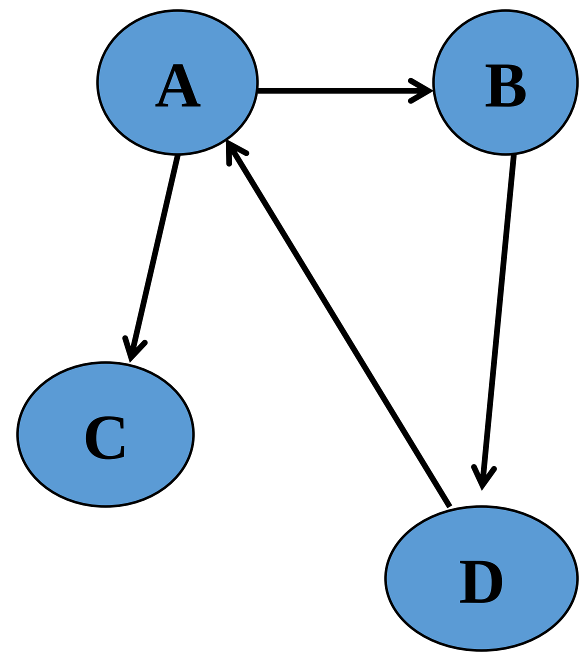


Adjacency lists: directed graphs

Vertices: A,B,C,D

Edges:AC, AB, BD, DA

Vertex	lists
A	B C
B	D
C	/
D	A



Graph Search Methods

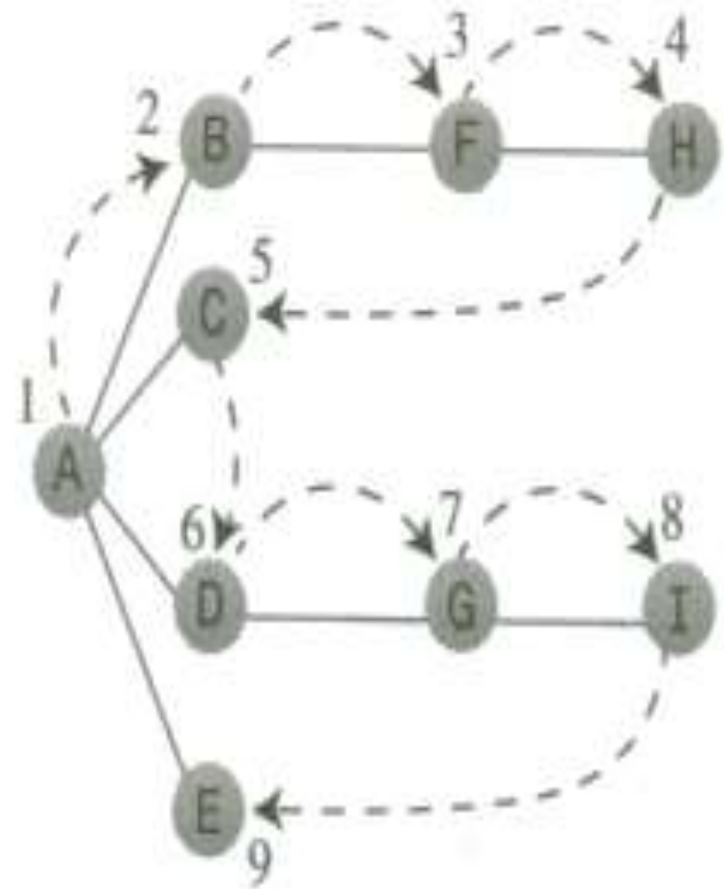
- Fundamental operation: find out which vertices can be reached from a specified vertex.
- An algorithm that provides a systematic way to start at a specified vertex and then move along edges to other vertex.
- Every vertex that is connected to this vertex has to be visited at the end.
- Two common approaches :
  - depth-first (DFS) and
  - breadth-first search (BFS).

DFS

- Can be implemented with a stack to remember where it should go when it reaches a dead end.
- DFS goes far way from the starting point as quickly as possible and returns only if it reaches a dead end.
- Used in in most simulations of games

DFS

- Steps:
  - -- start with a vertex
  - -- visit it
  - -- push it on a stack and mark it
  - -- go to any vertex adjacent to it that hasn't yet been visited
  - -- if there are no unvisited nodes, po a vertex off the stack till you come across a vertex that has unvisited adjacent vertices



DFS Complexity

- Time Complexity
  - Adjacency Lists
    - Each node is marked visited once: O(|V|)
    - Each node is checked for each incoming edge O (|V + E|)
  - Adjacency Matrix
    - Have to check all entries in matrix: $O(|V|^2)$
- Space Complexity
    - Worst case: all nodes are put on stack (if graph is linear)
    - O(n)

Breadth-First Search: BFS

- Implemented with a queue.
- Stays as close as possible to the starting point.
- Visits all the vertices adjacent to the starting vertex
- Continue with unvisited neighbors of the visited vertices

BFS

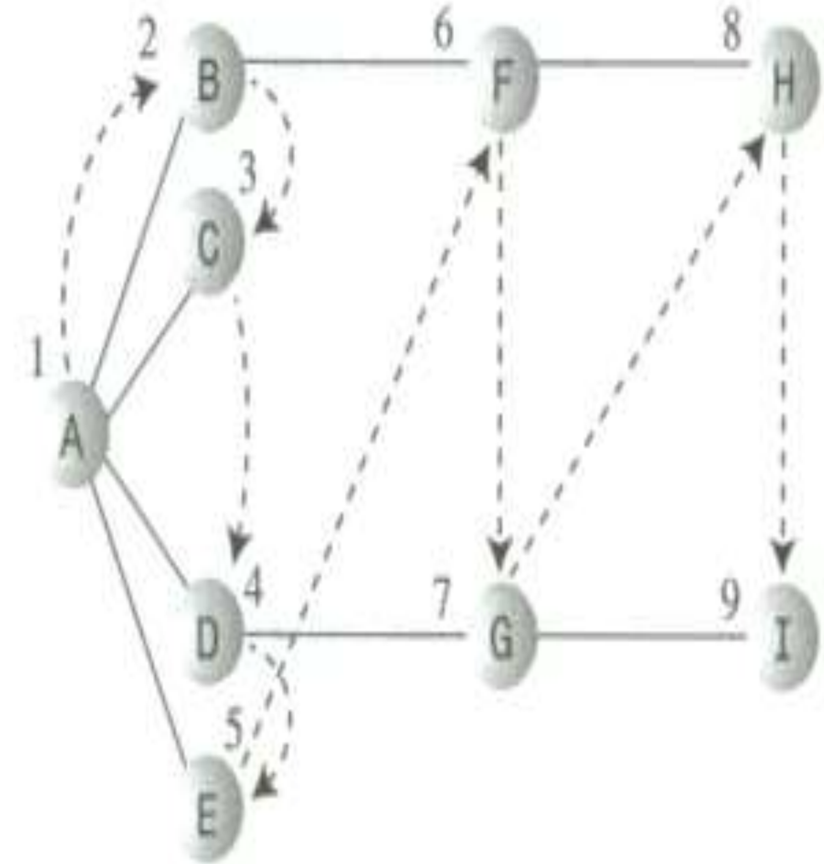
- Steps:

-- start with visiting a starting vertex

-- visit the next unvisited adjacent vertex, mark it and insert it into the queue.

-- if there are no unvisited vertices, remove a vertex from the queue and make it the current vertex.

-- continue until you reach the end.



BFS Complexity

- Memory required: Need to maintain a queue, which contains a list of all vertices we need to explore.
    - Worst case: all nodes put on queue (if all are adjacent to first node)

    O(n)
- Runtime :Adjacency List
    - O(|E|) to scan through adjacency list and O(|V|) to visit each vertex.

  →O(|V+E|) This is considered linear time in the size of G.
- Runtime : Adjacency Matrix
    - Scanning each row for checking the connectivity of a Vertex is O(n).

  →Have to check all entries in matrix: $O(n^2)$