Lecture 5: Graphs & their Representation

Why Do We Need Graphs

Graph Algorithms: Many problems can be formulated as problems on graphs and can be solved with graph algorithms. To learn those graph algorithms, we need basic knowledge of graphs.

Graphs: A graph describes some relation among a set of objects.

Examples of Applications: Communication and transportation networks, logic circuits, computer aided designs, etc.

Graphs – Continued

There are various types of graphs.

Two basic types are

undirected graphs (usually just called graphs)

directed graphs (sometimes called digraphs)

Undirected graphs

Definition: An (undirected) graph $G = (V, E)$ consists of two sets (components), a set V of vertices (nodes) and a set E of edges.

E is a set of 2-subsets of V,
that is, each edge is of the form $\{v, w\}$.

Example of an (undirected) graph:
$V = \{1, 2, 3, 4, 5, 6\}$.
$E = \{\{1, 2\}, \{2, 5\}, \{1, 5\}, \{3, 6\}\}$

Remark: In this course, unless otherwise stated, all graphs dealt with do not have self-loops (that is, $v \neq w$) and do not have parallel edges (that is, all edges are distinct).

Graphs – Continued

Definition: A directed graph $G = (V, E)$ consists of two sets (components), a set V of vertices (nodes) and a set E of edges.

E is a subset of $V \times V$,
that is, each edge is of the form (v, w),
a directed edge from v to w.

Example of a digraph:
$V = \{1, 2, 3, 4, 5, 6\}$.
$E = \{(1, 2), (2, 4), (4, 1), (2, 5), (4, 5), (5, 4), (6, 3)\}$.

Remark: Again, unless otherwise stated, all graphs dealt with do not have self-loops and do not have parallel edges.

Representation

We will consider various (equivalent) representations of graphs:

- Set description (as in definition).
  Mathematical

- Pictorial representation.
  For explanations

- Adjacency matrix representation.
  In computer

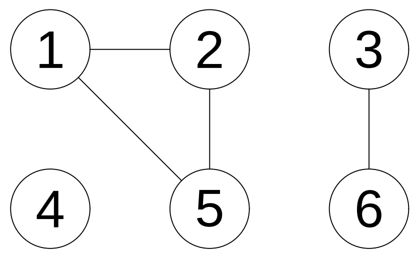
- Adjacency list representation.
  In computer

Pictorial Representation

Example, (undirected) graph:

$V = \{1, 2, 3, 4, 5, 6\}$.

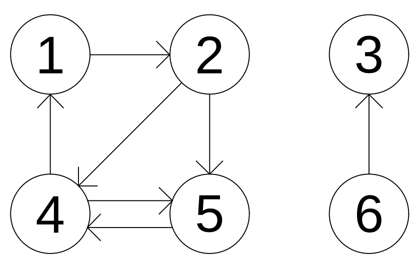
$E = \{\{1, 2\}, \{2, 5\}, \{1, 5\}, \{3, 6\}\}$



Example, digraph:

$V = \{1, 2, 3, 4, 5, 6\}$.

$E = \{(1, 2), (2, 4), (4, 1), (2, 5), (4, 5), (5, 4), (6, 3)\}$.



Matrix Representation

Adjacency Matrix of Digraphs: Let $G = (V, E)$ be a directed graph, where V is indexed by $\{1, 2, ..., n\}$. Its $n \times n$ *adjacency matrix* of G is defined by

$$A[v, w] = \begin{cases} 1 & \text{if } (v, w) \in E, \\ 0 & \text{otherwise} \end{cases}$$

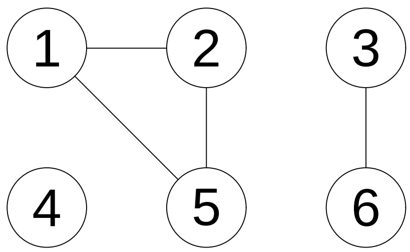
Adjacency Matrix of Graphs: Let $G = (V, E)$ be a graph, where V is indexed by $\{1, 2, ..., n\}$. The $n \times n$ *adjacency matrix* of G is defined by

$$A[v, w] = \begin{cases} 1 & \text{if } \{v, w\} \in E, \\ 0 & \text{otherwise} \end{cases}$$

Remark: The adjacency matrix of a digraph might not be symmetric, while that of undirected graphs must be symmetric.

Matrix Representation – Continued

Example, undirected graph: The adjacency matrix of

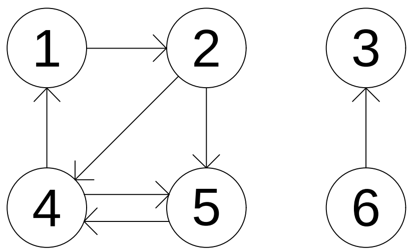


is

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Matrix Representation – Continued

Example, directed graph: The adjacency matrix of



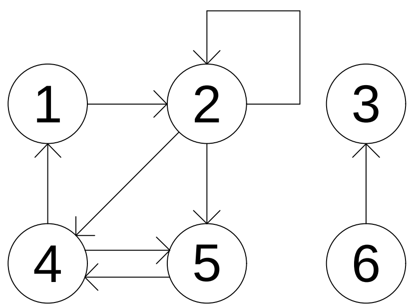
is

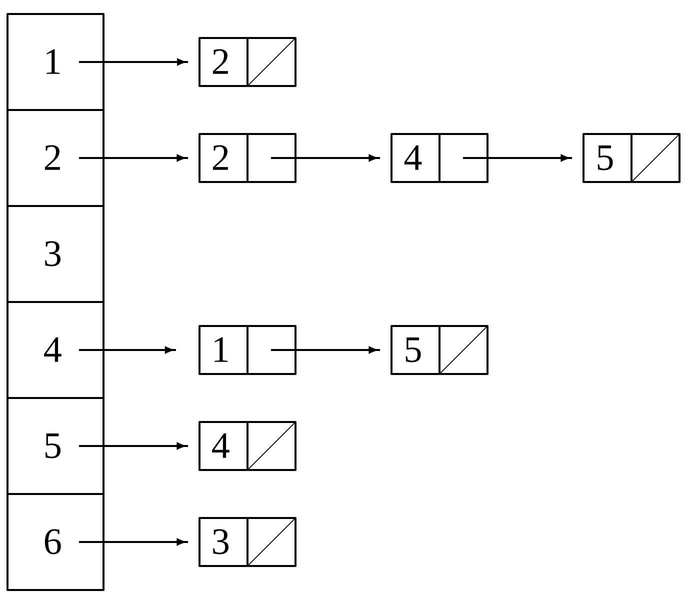
$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Adjacency List

An **Adjacency List** is an array $Adj[1...n]$ of pointers, where $Adj[u]$ points to a linked list containing the vertices u such that $\{u, v\}$ (undirected) or (u, v) (directed) is an edge.

Example (Attention: A loop at vertex 2).





The Best Representation

Which is better to use, adjacency matrics or adjacency lists?

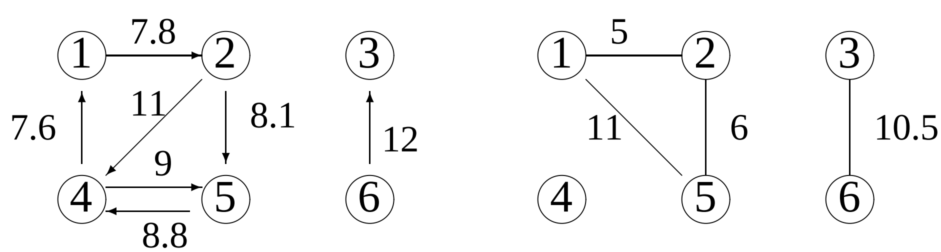
The answer depends upon what operations your algorithm needs to perfom

Operation	Adjacency List	Adjacency Matrix
Initialization (space needed)	$\Theta(\|E\| + \|V\|)$	$\Theta(\|V\|^2)$
Scan all Edges once	$\Theta(\|E\| + \|V\|)$	$\Theta(\|V\|^2)$
Check if $(u, v) \in E$	$\Theta(\|V\|)$	$O(1)$

Weighted Graphs

Definition: A *weighted graph (digraph)* is a graph in which there is a number associated with each edge called the *weight* of the edge.
This weight could represent many things, e.g., the length of a road connecting u to v, the cost of shipping an item from u to v, the capacity of a pipeline connecting u to v, etc..

Examples:



For weighted graphs (and digraphs), we can store the weights in a matrix similar to the adjacency matrix.

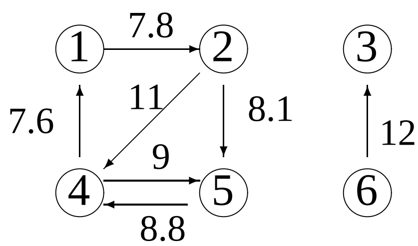
For example, for an edge $\{v, w\}$ (or (v, w)), define

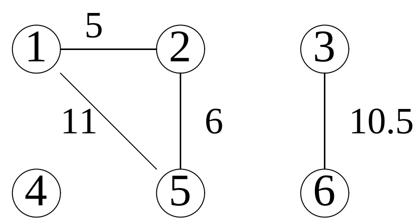
$$A[v, w] = W(v, w),$$

the weight of the edge.

Otherwise we define $A[v, w]$ to be ∞, which is larger than any number.

The *adjacency list* data structure can similarly be augmented to store weights.





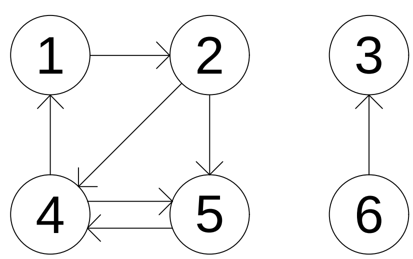
$$\begin{bmatrix} \infty & 7.8 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 11 & 8.1 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty \\ 7.6 & \infty & \infty & \infty & 9 & \infty \\ \infty & \infty & \infty & 8.8 & \infty & \infty \\ \infty & \infty & 12 & \infty & \infty & \infty \end{bmatrix}.$$

$$\begin{bmatrix} \infty & 5 & \infty & \infty & 11 & \infty \\ 5 & \infty & \infty & \infty & 6 & \infty \\ \infty & \infty & \infty & \infty & \infty & 10.5 \\ \infty & \infty & \infty & \infty & \infty & \infty \\ 11 & 6 & \infty & \infty & \infty & \infty \\ \infty & \infty & 10.5 & \infty & \infty & \infty \end{bmatrix}.$$

Incidence

Incident from & to: In a directed graph $G = (V, E)$, an edge (u, v) is *incident from* or *leaves* u, and is *incident to* or enters v.

Example: In the digraph

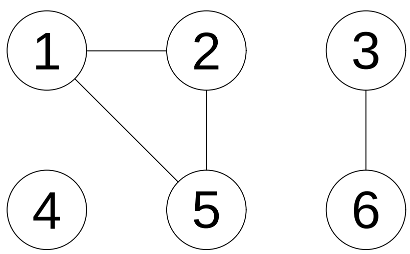


the edge $(1, 2)$ is incident from 1 and is incident to 2.

$\boxed{\textbf{Incident}}$

Incident on: In an undirected graph $G = (V, E)$, an edge $\{u, v\}$ is said to be *incident on* both u and v.

Example: In the undirected graph

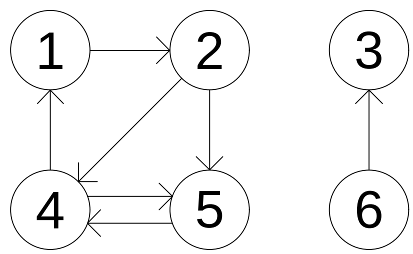


the edge $\{3, 6\}$ is incident on both 3 and 6.

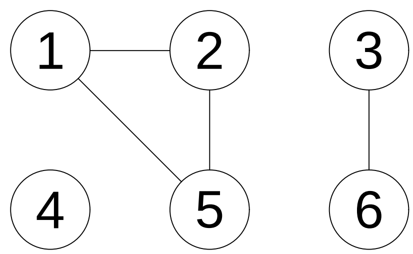
Adjacent

A vertex v is *adjacent to* u if there is an edge from u to v.

For example, in the digraph below, 2 is adjacent to 1.



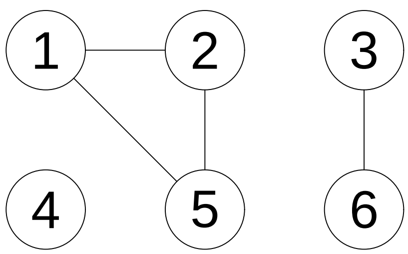
In the undirected graph below, 1 and 2 are adjacent vertices.



Degree of Vertices and Graphs

Degrees: The degree of a vertex in an undirected graph is the number of edges incident on it, with a loop being counted twice. The (total) degree is the sum of the degrees of all vertices.

Example: For the undirected graph



we have

$$\deg(1) = \deg(2) = \deg(5) = 2,$$
$$\deg(3) = \deg(6) = 1, \ \deg(4) = 0.$$

So $\deg(G) = \sum_{i=1}^{6} \deg(i) = 8$.

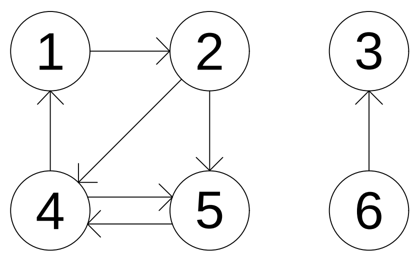
Degree Formula: For a graph $G = (V, E)$,

$$\deg(G) = \sum_{v \in V} \deg(v) = 2|E|.$$

Degree of Vertices and Graphs – Continued

In a digraph, the *outdegree* of a vertex is the number of edges leaving it, and the *indegree* of a vertex is the number of edges entering it. The *degree* of a vertex in a directed graph is its indegree plus outdegree.

Example: For the directed graph



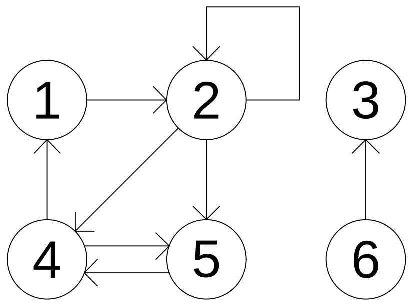
we have

$$\text{indegree}(1) = \text{outdegree}(1) = 1,$$
$$\text{indegree}(2) = 1, \ \text{outdegree}(2) = 2,$$
$$\text{indegree}(5) = 2, \ \text{outdegree}(5) = 1.$$

Degree Formula: For a digraph $G = (V, E)$,

$$\sum_{v \in V} \text{indegree}(v) = \sum_{v \in V} \text{outdegree}(v) = |E|.$$

Paths



Path: A *path* in a directed graph is a sequence of vertices

$$\langle v_0, v_1, \cdots, v_k \rangle$$

such that

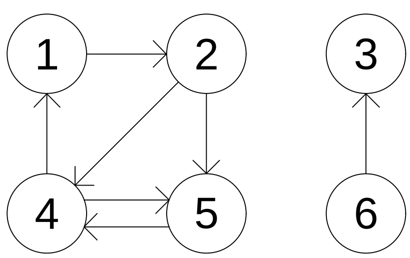
$$(v_{i-1}, v_i) \in E$$

for $i = 1, 2, .., k$.

The *length* of the path is the number of edges, k.

We say that w is *reachable* from u if there is a path from u to w. A path is *simple* if all vertices are distinct.

Example: In the digraph

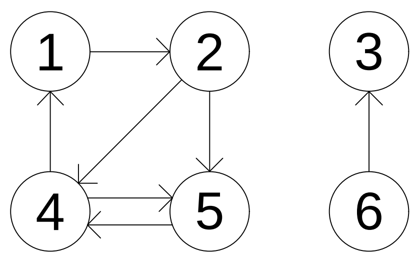


$\langle 2, 4, 5, 4 \rangle$ is a path of length 3. It is not simple.

Cycles

Cycle: In a directed graph, a *cycle* is a path $\langle v_0, v_1, \cdots, v_k \rangle$ in which $v_0 = v_k$.

For example, $\langle 1, 2, 5, 4, 1 \rangle$, $\langle 5, 4, 5 \rangle$, $\langle 1, 2, 4 \rangle$, are cycles in the graph

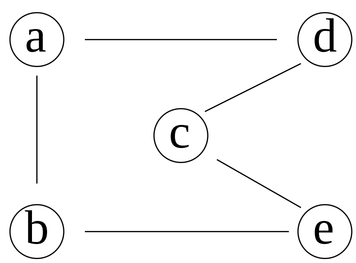


Paths and Cycles in Undirected Graphs are similarly defined.

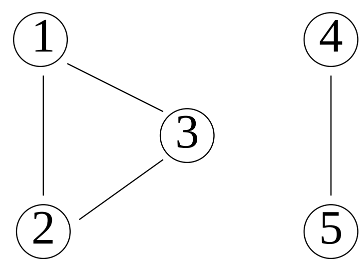
A cycle $\langle v_0, v_1, \cdots, v_k \rangle$ is simple if the path $\langle v_0, v_1, \cdots, v_{k-1} \rangle$ is simple.

Special Graphs

Connected Graph: An undirected graph is *connected* if every pair of vertices is connected by a path.

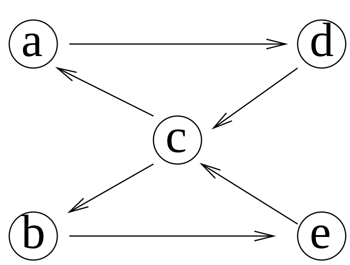


connected

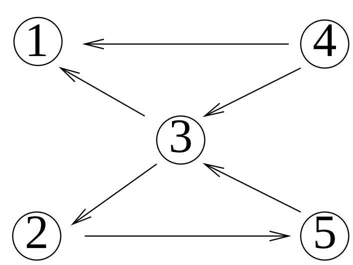


unconnected

Strongly Connected Digraph: A directed graph is *strongly connected* if for any pair of vertices one can be reachable from the other.



strongly connected

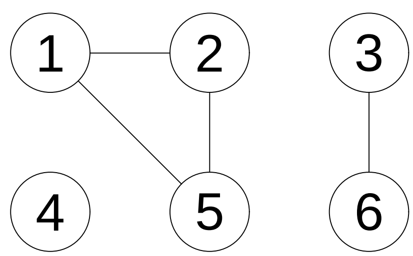


not strongly connected

Special Graphs – Continued

Acyclic Graphs: A graph with no cycle is *acyclic*.
DAG = directed acyclic graph.

Forest: An acyclic, undirected graph is a *forest*. For example, the undirected graph below is not a forest, as it has cycles.

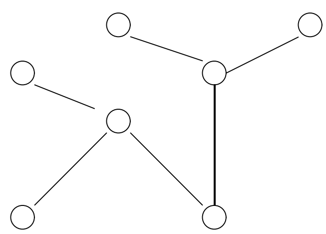


Tree: An acyclic, connected undirected graph is a tree.

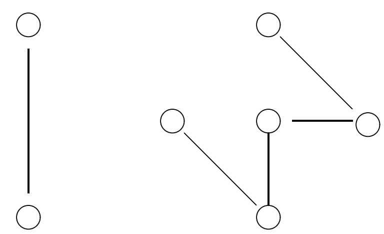
Comments on Forests and Trees: Any forest must be composed of a number of trees.

Special Graphs – Trees

Trees: A tree is a connected, acyclic, undirected graph. For example, Graph (a) is a tree, while (b) is a forest:



(a) A tree



(b) A forest

Properties of Trees: Let $G = (V, E)$ be an undirected graph. The following statements are equivalent:

(1) G is a tree.

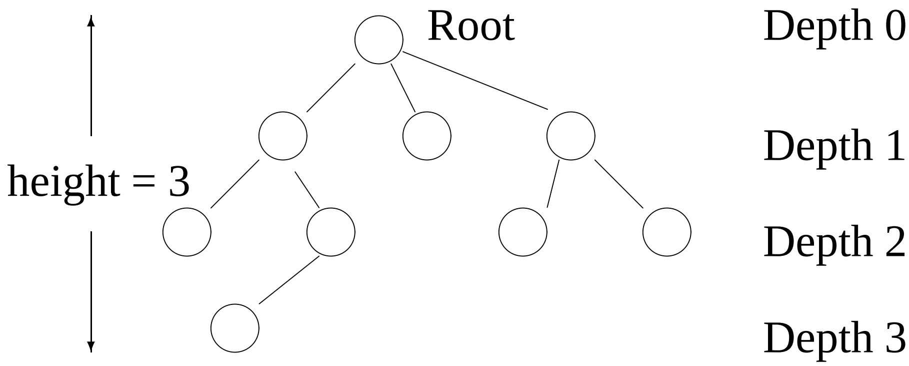
(2) Any two vertices in G are connected by a unique simple path.

(3) G is connected, and $|E| = |V| - 1$.

(4) G is acyclic, and $|E| = |V| - 1$.

Rooted Trees and Ordered Trees

Rooted Tree: A *rooted tree* is a tree in which one of the vertices is distinguished from the others. The distinguished vertex is called the *root* of the tree.



Rooted tree

Some Basic Notions: Height, depth (level), nodes (internal vertices), leaves, children.

Ordered Tree: An ordered tree is a tree in which the children of each node are ordered.

Binary Trees

Binary Tree: In a binary tree every node has at most two children.

We distinguish between left children
and right children.