Graph Algorithms (Spring 2026)

1. Basics of Graph Theory

Teacher: Arnaud Casteigts *Assistant: Himika Das*

In this first lesson, we give basic definitions of graph theory. Keywords: undirected graphs, directed graphs, paths, connectivity, connected components, adjacency matrix, adjacency lists, regular graphs, bipartite graphs, planar graphs.

1.1 Basic definitions

1.1.1 Undirected graphs

An **undirected graph** $G = (V, E)$ (or simply a graph) is made of a set of **vertices** V (also called nodes) and a set of **edges** E connecting these vertices (also called links). Each edge is defined as a *pair* of vertices. For example, the following graph corresponds to $V = \{a, b, c, d\}$ and $E = \{\{a, b\}, \{b, c\}, \{a, c\}, \{c, d\}\}$. We write n for $|V|$ and m for $|E|$. Here, $n = 4$ and $m = 4$.

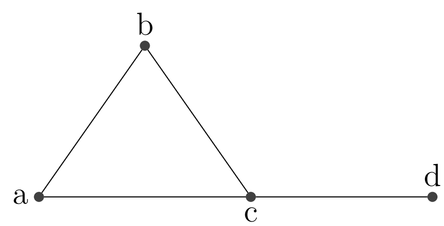


Figure 1: A graph G_1.

In general, a graph may also have **loops** (edge from a vertex to itself), or **multiple edges** (several edges between the same pair of vertices). Unless otherwise mentioned, we will consider only **simple graphs**, i.e. graphs without loops nor multiple edges, like G_1. The rest of the definitions below consider simple graphs.

The **degree** of a vertex u, noted $d(u)$ is the number of edges incident to u, namely $d(v) = |\{e \in E \mid v \in e\}|$. Here, $d(a) = 2$ and $d(c) = 3$. The **neighbors** of u are all the vertices that share an edge with u. Thus, the number of neighbors of a vertex corresponds to its degree. If we have an edge $e = \{u, v\}$, then u and v are the **endpoints** of e.

A **walk** from u to v is a sequence of vertices $u = u_1, u_2, ..., u_k = v$ such that for all $i < k$, $\{u_i, u_{i+1}\} \in E$. For example, (a, b, c, d) is a walk in G_1, and so is (a, c, b, a, c, d). A **path** is a walk that does not repeat any intermediate vertex. A path from a vertex to itself is called a **cycle**, for example (a, b, c, a).

A graph is **connected** if a path exists between every pair of vertices. Otherwise, it can be partitionned into a set of **connected components**. A graph is **complete** if an edge exists between every pair of vertices. Connectivity and completeness should not be mistaken; for example, G_1 is connected, but is not complete.

Finally, the **length** of a walk/path/cycle is the number of *edges* it uses. The **distance** between two vertices is the length of the shortest path between them (∞ if no path exists), and the **diameter** of the graph is the largest distance over all pairs of vertices (G_1 has diameter 2, which is the distance between a and d, or the distance between b and d).

In this class, whenever we use the term "graph" without adjective, we mean a simple and undirected graph.

1.1.2 Directed graphs

A **directed graph** $G = (V, A)$ (also called a digraph) is made of a set of vertices V and a set of **arcs** (or directed edges), where each arc is defined by an *ordered pair* of vertices. For example, the graph on fig. 2 corresponds to $V = \{a, b, c, d\}$ and $A = \{(b, a), (a, c), (b, c), (c, b), (c, d)\}$.

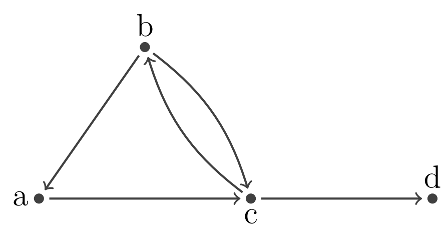


Figure 2: A directed graph G_2.

Simple digraphs are defined analogously to simple graphs (no loop nor multiple arcs). Two arcs with different directions are not the same, so a simple digraph can have such arcs, like in G_2 between b and c.

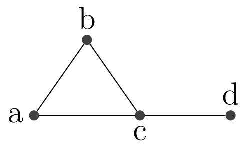
The **in-degree** of a vertex u, noted $d^-(u)$ is the number of incoming arcs to u. Its **out-degree** $d^+(u)$ is the number of outgoing arcs from u. For example, in G_2, we have $d^-(b) = 1$ and $d^+(b) = 2$. **In-neighbors** and **out-neighbors** are defined analogously. They are also called **predecessors** and **successors**.

A **path** from u to v in a digraph is defined in the same way as in undirected graph, except that it must satisfy $(u_i, u_{i+1}) \in A$, direction matters! In digraphs, the reachability relation is not symmetric, e.g. in G_2, a path exists from a to d, but not from d to a.

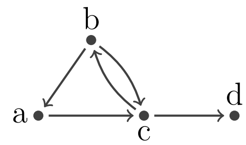
A digraph is **strongly connected** if there exists a path from every vertex to every other vertex. It is **weakly connected** if a path would exist if we ignored the direction of the arcs. For example, G_2 is weakly connected, but it is not strongly connected.

1.1.3 Data structures

Graphs can be represented in various ways. A common one is as an **adjacency matrix**, which is a $n \times n$ matrix that encodes the edges (resp. non-edges) as 1 (resp. 0).



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



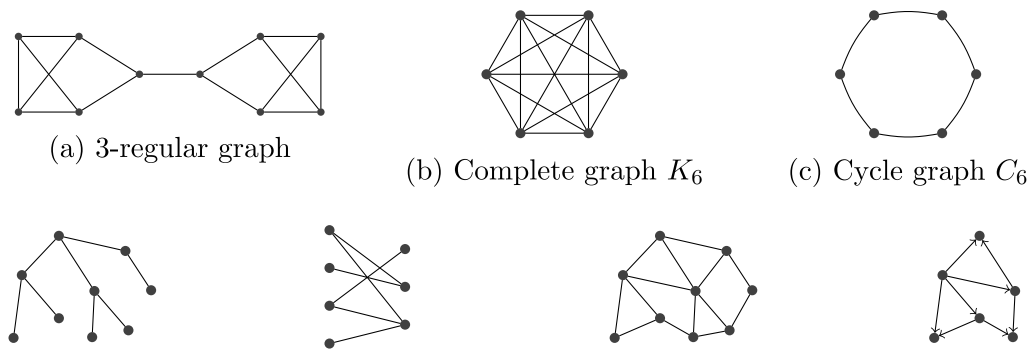
$$= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Another common data structure for graphs is an **adjacency lists**, which consists of a list of neighbors for each node. Adjacency lists are more *space* efficient than adjacency matrices, especially in sparse graphs (graphs with few edges), because the non-edges are not explicitly stored. However, adjacency matrices are faster because their values can be accessed directly. They are also convenient for algebraic manipulation, for example, if M is the adjacency matrix of some graph, then M^k represents something specific about the graph (to be discussed in exercises); the eigenvalues of M also give information about how the graph is organized. Most graph libraries (like `networkx` in Python) rely on adjacency lists. We will often use higher level primitives that mask the underlying data structure, discussing the cost when relevant.

1.1.4 Some classes of graphs

Here are specific classes of graphs that are well known (illustrated in Figure 3):

- Regular graph: all vertices have the same degree.
- Complete graph: all pairs of vertices share an edge.
- Cycle graph: consists of a single cycle.
- Tree: contains no cycles and is connected.
- Bipartite graph: V can be split in two parts, such that all the edges are between these parts.
- Planar graph: can be drawn on the plane without crossing edges.
- Directed acyclic graph (DAG): directed graph that contains no cycle



(a) 3-regular graph (b) Complete graph K_6 (c) Cycle graph C_6

(d) Tree (e) Bipartite graph (f) Planar graph (g) DAG

Figure 3: Some common classes of graphs.

2 Basic lemmas

Graph theory is often about proving properties. Let us start with a few basic properties. Namely, (1) in a graph, the sum of all degrees is always even; (2) in a digraph the sum of in-degrees is always equal to the sum of out-degrees; and (3) in a graph, there is at least two nodes that have the same degree.

Lemma 2.1 ("Handshaking")**.** *For all graphs* $G = (V, E)$, $\sum_{v \in V} d(v)$ *is even.*

Proof. Every edge increases the sum by 2 (1 on each side). □

Lemma 2.2 ("Footkicking")**.** *For all digraphs* $G = (V, A)$, $\sum_{v \in V} d^-(v) = \sum_{v \in V} d^+(v)$.

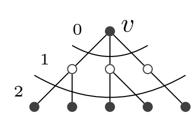
Proof. Every arc increases both sums by 1. □

Lemma 2.3 (Same degree)**.** *Let* G *be a graph. At least two nodes of* G *have the same degree.*

Proof. (By contradiction) The smallest possible degree is 0 and the largest possible degree is $n - 1$, so there are n possible values. If all the nodes have different degrees, then each value must be used. In particular, one node must have degree 0 (isolated node) and one node must have degree $n - 1$ (neighbor with all the others), which is impossible in the same graph. □

Lemma 2.4. *Trees are bipartite graphs.*

Proof. Let $G = (V, E)$ be a tree. We will split V into V_1 and V_2 such that all the edges are between V_1 and V_2. For this, choose an arbitrary node $v \in V$. Put all nodes whose distance to v is even into V_1 and the others into V_2 (see the picture). Because G is a tree, the distance from the endpoints of any edge to v must differ by one, thus one is in V_1, the other in V_2.



□