Chapter 1

Basic notions in graph theory

1. Introduction

Many real-world and engineering problems involve graphs and graph-based problems. Graphs can been seen as a language for modeling and a tool for problem-solving. A graph *topology*, i.e. how entities and links are arranged, can induce some interesting properties. Let's consider some introductory examples to illustrate this. There is an abundance of literature concerning graph theory. In this course, students car refer to [1], [2] and [3].

Example 1.1 : modeling a friendship network

In everyday life, one can have many friends, and each of them can be friends with others, and so on. For instance, social networks promote the development of communities and the establishment of policies for data sharing. Such policies are generally based on friendship networks. For instance, we can prove that there can always be at least two people in any group of friends who have the same number of friends.

Example 1.2 : a road network model

A road network is a collection of towns connected by roadways. Visualizing such a network is important if we want to find a way when we are lost or if we want to discover the shortest path between two places. Using graph theory, it is also possible to build some interesting routes, such as the one that passes once through every town or road on a map.

Example 1.3 : a telecommunications network model

A computer network is made up of physical connections such optical fiber, radio waves, metallic cables, as well as a set of equipment including computers, hubs, switches, routers, repeaters, etc. To ensure that a message reaches its destination, it must be *routed* through a router to the appropriate output port. Particularly, we can look for the fastest rate at which data can be delivered across a network, or the most efficient path in a network (in terms of time, energy required, hops, etc.).

Example 1.4 : problem-solving models

In artificial intelligence, problem solving consists of determining a sequence of actions to reach a desired situation from a starting point. Each action made changes to the problem's state and this creates links between possible situations. The problem can be solved by searching pathways made up of these links.

Example 1.5 : programs' modeling

When learning to program, especially in imperative languages like C, we define a program as a sequence of actions. We often employ the metaphor of a cooking recipe to illustrate this.

However, a program may be defined from a different perspective. Actually, each action in a program changes the values of the variables. These values along with the variables' names are referred to as a "state". Consequently, an instruction can be viewed as a link connecting one state to another, and the program as a whole can be seen as a set of links between the possible states. This modeling enables the detection of program cycles (i.e., endless loops) as reasoning about programs (also known as formal semantics).

The examples above show the potential of graphs and how they may be applied to solve many problems. We will give a series of definitions and properties of graphs in order to solve typical real-world problems.

A graph may be loosely defined as a figurative representation consisting up of points, referred to as *nodes* or *vertices* (plural of *vertex*), interconnected by a series of straight or curved lines. The links usually represent a relationship between the vertices. If they are symmetric (the graph is called *undirected*), they are called *edges*; if they are asymmetric (the graph is called *directed*), they are called *arcs*.

2. Undirected graphs

Definition 1.1

An *undirected graph* is a pair (X, E) where X is a set of *vertices* and E is a set of *edges* (a set of links between the vertices in X).

An edge is written as an unordered pair (s, t). Typically, vertices are named, but edges can also be named in particular situations. If an edge connects a vertex to itself, it is called to as a *loop*.

The vertices of the graph in figure 1.1 are $\{a, b, c, d, e\}$ and edges $\{A = (a, b), B = (a, c), C = (b, b), E = (c, d), F = (c, d), G = (d, e), H = (b, d)\}$. In the graph of figure 1.2 , the vertices are $\{p, q, r, s\}$ and edges are $\{U = (p, r), V = (p, q), X = (p, s), Y = (q, s), Z = (r, s)\}$.

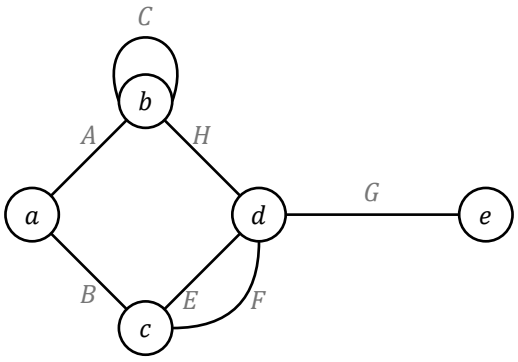


Figure 1.1 - an example of a multigraph

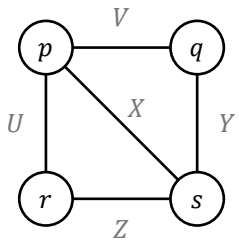


Figure 1.2 - an example of a simple graph

Remark : the graph's drawing is not distinctive. Different geometric diagrams may be used to represent the same graph. Edges can be either straight or curved lines. Note, however, that if a graph can drawn without any edge intersection then it is said to be *planar*.

An edge A is said to be *incident* to a vertex x if A starts at x. *Adjacent* vertices are those that are connected by an edge. Similarly, two edges are considered adjacent if they start from the same vertex. In figure 1.1 , vertices a and b are adjacent, edges A and B are also adjacent.

Usually, edges are associated with information, which we refer to as the edge's *weight* or *value* (sometimes called *label* too). A *valued graph* is a graph that has values on its edges.

2.1. Order, size and degrees of a graph

Definition 1.2

Let $G = (V, E)$ be an undirected graph. The number of vertices in V (denoted as order(G)) is called the order of G. The size of G (denoted as size(G)) is the total number of edges in E.

Consider a vertex x of G. We denote by $d_G(x)$ the number of edges that start at x, we call it *the degree* of x. The degree of G (denoted as $d(G)$) is the greatest degree of its vertices ($d(G) = \max_x d_G(x)$).

In figure 1.3 , the order of the graph is 7. The degrees of its vertices are given by the following table:

Vertex	a	b	c	d	e	f	g
Degree ($d_G(x)$)	4	2	2	3	2	1	0

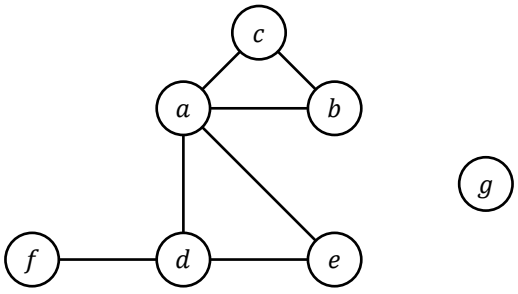


Figure 1.3 - types of vertices (g is isolated and f is pendant)

The degree of this graph is 4 (which is the degree of a). Vertex g is called an *isolated* vertex since its degree is null. Vertex f is a *pendant* vertex because it has degree 1. Note that the sum of all degrees is even, and is twice the graph's size. This is not a specific property of this graph as it is stated in the following theorem.

Theorem 1.1

Let $G = (X, E)$ be a undirected graph. The equality $\sum_{s \in X} d(s) = 2 \times \text{size}(G)$ is always satisfied. Hence, in a graph, the sum of all its degrees is always even.

Proposition 1.1

A straightforward application of this theorem suggests that there are an even number of vertices with odd degrees.

We can now answer the following question: is it feasible to have a group of five persons, each of whom has exactly three friends (within the group)?

If all of the vertices in a graph have the same degree, the graph is called *regular*. Let k be this degree, the graph is called k-regular. It is simple to demonstrate that the order of a k-regular graph is even if k is odd.

2.2. Terminology for particular kinds of graphs

Depending on the topology of the edges, different graph types exist. They may have specific properties.

Multigraph

In a multigraph, many edges may exists between two vertices. It can also have loops.

➠ The graph in figure 1.1 is a multigraph.

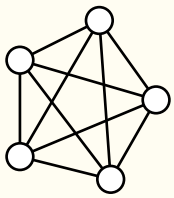
Simple graph

In a simple graph, one edge at most may exist between two vertices. There are no loops. A simple graph has the characteristic that at least two vertices have the same degree.

➠ The graph in figure 1.2 is a simple graph.

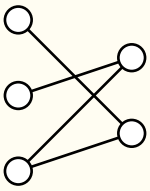
Complete graph

A complete graph is a simple graph in which each vertex is connected to every other vertex. A complete graph with order n is denoted K_n. Its size is given by $\frac{n(n-1)}{2}$. Actually, this a regular graph (more precisely, it is $(n-1)$ -regular).



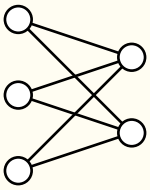
Bipartite graph

A bipartite graph's vertices may be divided into two disjoint subsets X and Y, with edges connecting only vertices from X to vertices from Y (there are no edges inside the same subset).



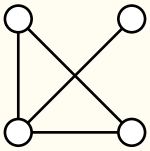
Complete bipartite graph

In a complete bipartite graph, every vertex from X is connected to every vertex from Y. It is denoted by $K_{m,n}$ where m is the cardinality of X and n is the cardinality of Y. The size of $K_{m,n}$ is $m \times n$. This graph is regular if and only if $m = n$.



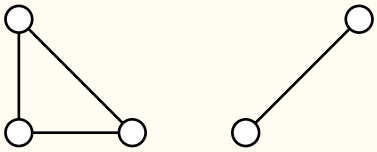
Connected Graph

A graph is a connected graph in which one may follow one or more edges from each vertex to any other vertex.



Non-connected Graph

A non-connected graph is a graph that has at least one vertex that cannot be reached from another vertex.



Bipartite graphs are used to solve several assignment problems by associating items from a source set with items from a target set. For example, the first set may consist of tasks, while the second one may consist of the required resources to complete the tasks. The edges describe how a resource is assigned to a task.

2.3. Planar graph

Definition 1.3

A *planar graph* is a graph that can be embedded in the plane with no edges intersecting.

Initially, the term "graph" was used because graphs are drawn. Their geometric properties can be used for modeling structural constraints, particularly in a 2D environment. It is essential to understand what is intended by *drawing* edges; an edge can be drawn using any continuous straight or curved line. Therefore, the characteristics of the graph are not defined by the form of an edge.

Planar graphs have the advantage of being concise when drawn. Their applications are numerous. In computer science, they are frequently utilized to model circuits and networks. They can be used, for example, to the design of *printed circuit boards* (PCBs) in which wires shouldn't cross. Additionally, because crossing edges can lead people to see incorrect vertices, planar networks are easier to comprehend.

Consider the graph in figure 1.4 , which contains two crossing edges. The same graph (in figure 1.5) may be re-drawn without any edge crossings. Thus, this graph is planar.

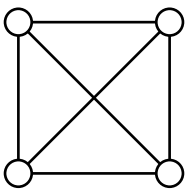


Figure 1.4 - this graph is being drawn with two crossing edges

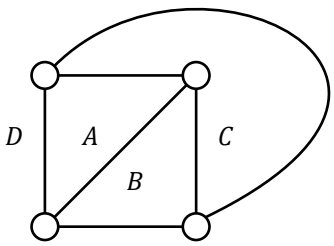


Figure 1.5 - the same graph be redrawn with no crossing edges

Definition 1.4

In a planar graph, a *face* is an area bounded by edges such that two arbitrary points may always be connected by an edge that does not intersect any other edge or vertex.

The graph in figure 1.5 has four faces: three with finite areas (A, B, and C) and one exterior face with an infinite area (D). A face's *borders* are determined by the edges that surround it. The number of edges surrounding a face defines its *degree*. Therefore, every face in figure 1.5 has a degree of three.

We can state two theorems about planar graphs.

Theorem 1.2

The sum of the degrees of the faces of a planar graph is twice its size.

Theorem 1.3

(Formula of planarity) Let G be a planar graph, where S is its order, A is its size, and R is the number of faces. Therefore, $S - A + R = 2$ (this formula is meaningless if G is not planar).

We can check these properties against the graph in figure 1.5 :
- The size of the graph is 6, the sum of the degrees of its faces is 3+3+3+3=12. Checked.
- S=4, A=6 and R=4 : 4-6+4=2. Checked.

We will use the last theorem to prove that the graph $K_{3,3}$ is not planar. In fact, this graph has an order of 6 and a size of 9. If it was planar, the number of faces would be $R = 2 + A - S = 2 + 9 - 6 = 5$, with four internal faces (F_1, F_2, F_3, and F_4) and one external face (F_5). Since the graph is simple, the degree of F_5 is at least equal to 3.

Given that G is bipartite, let X be the left vertices set and Y be the right vertices set. If we take one vertex from X, an internal face is formed by considering one vertex from Y, then another vertex from X (different from the first one since G is simple), and finally another vertex from Y to return to the initial vertex (and close the face). As a consequence, we easily see that each internal face has a minimum degree of four. By using the first theorem, we have:

$$\sum_{i=1}^{5} d(F_i) = 18$$

but we have just proven that:

$$\sum_{i=1}^{5} d(F_i) \geq 4 \times 4 + 3 = 19$$

Contradiction! Consequently, $K_{3,3}$ is not planar.

We can similarly prove that the graph K_5 is not planar. **Obviously, any graph that contain a non-planar subgraph inside, is itself non-planar**.

The graph in figure 1.6 is an example of a non-planar graph.

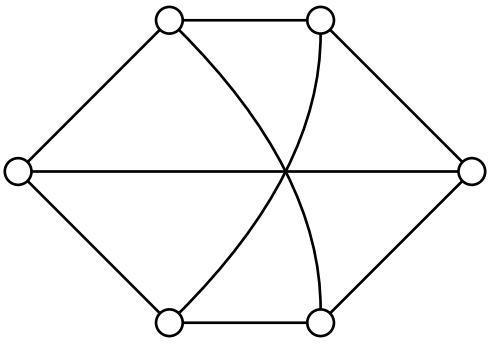


Figure 1.6 - a non-planar graph

2.4. Walks, cycles and cocycles

Definition 1.5

In a graph G, a *walk* from vertex x_0 to vertex x_k is a sequence of vertices $x_0, x_1,\ldots, x_k$ such that (x_{i-1}, x_i) is an edge for $i = 1..k$.

The walks $x_0, x_1,\ldots, x_k$ and $x_k, x_{k-1},\ldots, x_1, x_0$ (obtained by reversing the order of the vertices in the first walk) are the same. In reality, this definition is not quite precise for a multigraph. In this case, one should specify the edges walked from one vertex to another.

The sequence b, a, d, f, depicted in a thick line in figure 1.7 , is a walk (it is equivalent to f, d, a, b). The sequence is also a walk despite the fact that certain vertices are repeated several times. To differentiate them, we will say that a walk is *simple* if all of its vertices are distinct.

A *trail* is a walk in which all edges are distinct (no edge is repeated). For instance, a, c, b, a, d, f is a trail.

A *closed walk* is a walk in which the initial and final vertices are the same (otherwise, it is an open walk). A closed trail is called a *cycle*. In figure 1.7 , the cycles are a, b, c, a and a, e, d, a.

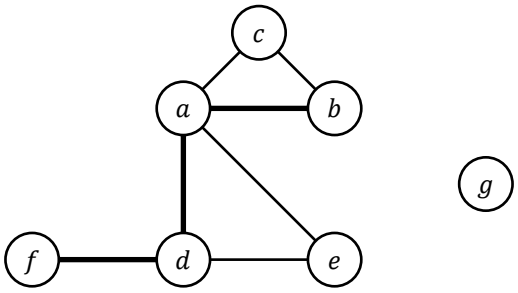


Figure 1.7 - example of a walk : b, a, d, f

In a graph $G = (X, E)$, let W be a subset of X ($W \subseteq X$). A *cocycle* is the set of edges with one endpoint in W and the other in $X - W$. If the cocyle is removed from G, the vertices of W become isolated from the rest of the graph. In figure 1.7 , if $W = \{a, e, d\}$ then the corresponding cocycle is $\{(a, c), (a, b), (f, d)\}$.

Walks allow the computation of useful metrics about a graph. First, we can define a walk's *length* as the number of edges it contains. The walk b, a, d, f has a length of 3. The *distance* between two vertices is the length of the shortest walk possible between them. The distance between b and d is 2. Finally, the *diameter* of a graph is the greatest distance between any two of its vertices (compare this to the diameter of a disk). The diameter of the considered example is 3.

Thanks to lengths and cycles, it is possible to state a theorem that characterizes bipartite graphs.

Theorem 1.4

A graph is bipartite if and only if it has no cycles of odd length.

Thanks to this theorem, we can prove that the graph in figure 1.7 is not bipartite.

2.5. Eulerian graph

Definition 1.6

An *Eulerian cycle* is a cycle that passes through all the edges of a graph exactly once. A graph with an Eulerian cycle is called an *Eulerian graph*. An open walk is *Eulerian* if it passes through all the edges exactly once. A non-Eulerian graph with an Eulerian walk is called a *semi-Eulerian graph*.

Another definition of an Eulerian graph is one in which the edges may be drawn without lifting the hand or retracing any edge, while still being able to return to the starting vertex.

The name "Eulerian" originates from the famous bridge problem in the town of Königsberg (now called Kaliningrad). The bridge map of this town is shown in figure 1.8 . The graph in figure 1.9 illustrates the various travel possibilities between points within the town. The problem consisted of determining if it was possible to cross all the bridges in the town, each bridge being traversed only once. Euler was the first to solve this problem.

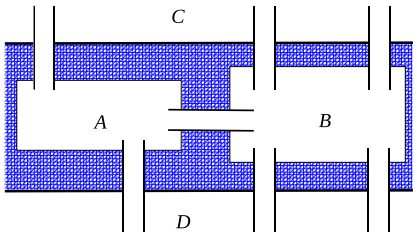


Figure 1.8 - the map of the bridges in Königsberg

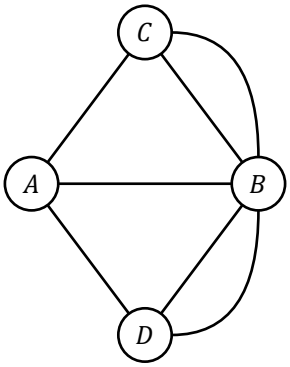


Figure 1.9 - the traversing graph of the problem of Königsberg

Euler even proved a theorem that defines the sufficient and necessary conditions for a graph to be Eulerian or semi-Eulerian.

Theorem 1.5

- A connected graph has an Eulerian cycle if and only if all of its vertices have even degrees.
- A connected graph has an Eulerian walk if and only if all of its vertices, except for two, have even degrees.

It is straightforward to see that the graph in figure 1.9 is not semi-Eulerian because it has 4 vertices that have odd degrees. Obviously, it is not an Eulerian graph.

Let's consider the envelope design problem shown in figure 1.10 . In this graph, the vertices a, b and c have even degrees, while the vertices d and e have odd degrees. By applying the Euler theorem, we can conclude that this graph is semi-Eulerian. An Eulerian walk for this graph is: $d, a, c, b, e, a, b, d, e$.

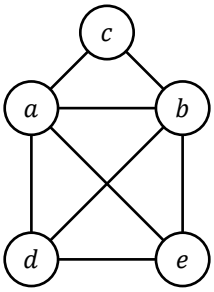


Figure 1.10 - the envelope design problem

Building Eulerian walks can be accomplished by using various algorithms. One of them is the Fleury algorithm which requires the graph to be, at least, semi-Eulerian.

Algorithm of Fleury

```
if the graph is Eulerian
    Start from any vertex
else
    Start from a vertex with an odd degree
Choose the next vertex depending on unvisited incident edges
if there are multiple choices
    Select a non-bridge
else
    Select the edge
Delete the chosen edge then go to the new vertex
repeat from 5. until there are no edges remaining
```

In order to determine whether an edge is a bridge, we can apply the following algorithm:

Algorithm to decide if an edge (u, v) is a bridge

```
if there is one adjacent vertex to u (in this case v)
    This is not a bridge
else
    a = number of reachable vertices from u
    b = number of reachable vertices from u if the edge (u, v) is removed
    if a > b
        This is bridge
    else
        This is not a bridge
```

2.6. Hamiltonian graph

Definition 1.7

A *Hamiltonian cycle* is a cycle that visits every vertex of a graph exactly once. A graph containing a Hamiltonian cycle is called a *Hamiltonian graph*.

A *Hamiltonian (open) walk* is a walk that visits every vertex of a graph exactly once. A non-Hamiltonian graph containing a Hamiltonian walk is called a *semi-Hamiltonian graph*.

Hamiltonian graphs have multiple applications. One such application is the traveling salesman problem, in which a salesman has to visit all towns in an area while keeping travel costs to a minimum.

There is no straightforward way to categorize Hamiltonian or semi-Hamiltonian graphs. However, we can note that:
- A graph containing a pendant vertex (a vertex with a degree of 1) cannot be Hamiltonian.
- In a Hamiltonian graph, all edges corresponding to vertices with degree 2 belong to a Hamiltonian cycle.
- The complete graph K_n is Hamiltonian.

Two theorems can, however, help determine if a graph is Hamiltonian in special cases. They establish only sufficient conditions for that.

Theorem 1.6

- Let G be a simple graph with order $n > 3$. If for every pair (x, y) of non-adjacent vertices, $d(x) + d(y) \geq n$, then G is Hamiltonian.
- (Straightforward application) For any graph G with order $n > 3$, if $d(x) \geq \frac{n}{2}$ for all vertex x, then G is Hamiltonian.

If we apply the first theorem to the graph in figure 1.10, we get the following table (yellow cells indicate non-adjacent vertices). Since the order of this graph is 5, it is thus a Hamiltonian graph.

Vertex	a $(d(a)=4)$	b $(d(b)=4)$	c $(d(c)=2)$	d $(d(d)=3)$	e $(d(e)=3)$
a					
b					
c				$d(c)+d(d)=5$	$d(c)+d(e)=5$
d					
e					

2.7. Partial graph and subgraph

Let $G=(X,E)$ be a graph. A *partial graph* of G is a graph $G'=(X,E')$ such that $E'\subset E$. In other words, to obtain a partial graph from G, some edges are taken off.

Let $A\subset X$ be a set of vertices. The induced *subgraph* by A, denoted as $(A,E(A))$, is a graph whose vertices are A and edges are those of E, with both endpoints in A.

A *clique* is a complete subgraph of G. Among all cliques, the *maximum clique* (having the biggest number of vertices) has interesting properties for some algorithms (for instance, graph coloration). In figure 1.11 , we can see a graph along with its cliques. There are two cliques: one with 3 vertices and the other, which is the largest, with 4 vertices.

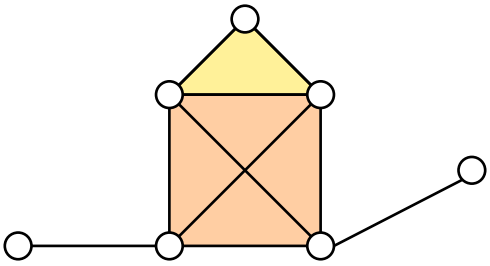


Figure 1.11 - a graph along with its cliques

In the context of a map containing the towns and roads in Algeria, a subgraph would represent a specific region or "Wilaya" along with all its associated towns and roads. On the other hand, a partial graph would exclusively show the national roads.

We can also define (connected) *components* using subgraphs. A component is a maximal subgraph in which there is a walk between every pair of vertices. In the non-connected graph shown in section 2.2 , there are two components. A connected graph has only one component.

A *stable set* is a set of vertices in which no edges connect any two vertices. For example, in a bipartite graph where vertices from a set X are linked to vertices from a set Y, X and Y are hence stable sets. The *stability number* of a graph G (denoted as $\alpha(G)$) is defined as the cardinality of its largest stable set. The stability number of the graph in figure 1.11 is 4.

2.8. Other operations on graphs

It is possible to create new graphs by performing different operations on a given graph.

Complementary graph

Let $G=(X,E)$ be a simple graph. The complementary graph of G is a graph with vertices in X, where two vertices are connected if and only if they were not connected in G. In other words, if the edges in the original graph represent a relation between vertices, the complementary graph represents the negation of this relation. The complementary graph of the one in figure 1.10 is given in figure 1.12 .

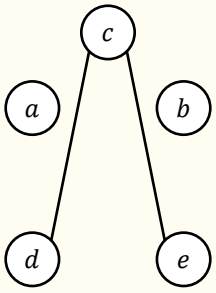


Figure 1.12 - the complementary graph of the one in figure 1.10

Dual graph

The dual graph is derived from a planar graph. It shows the adjacency between its faces. In other words, if G is a planar graph, then the vertices of the dual graphs are the faces of G. If two faces of G are adjacent, i.e., they share a common edge, then the dual graph has an edge that connects the two faces. The black graph in figure 1.13 is a planar graph, its dual graph is drawn in red.

This transformation is an *involution*. When applied twice, it yields the original graph. The dual graph converts cycles into cocycles and cocycles into cycles, respectively.

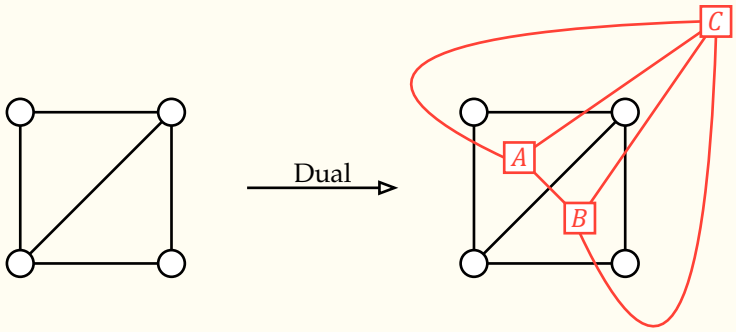


Figure 1.13 - example of a dual graph

Line graph

A line graph can be built for any graph. It represents the adjacency of edges in a given graph. In the line graph, each edge from the original graph becomes a vertex. If two edges in the first graph are adjacent (they have a shared vertex), their corresponding vertices in the line graph will be connected by an edge. The graph in figure 1.14 is the line graph of the one shown in figure 1.2 .

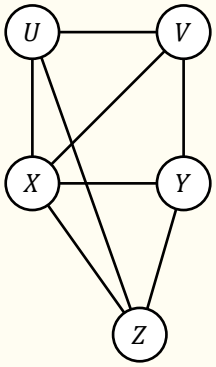


Figure 1.14 - the line graph obtained from the graph in figure 1.2

2.9. Non-graphical representation of a graph

A graph can be represented in ways other than graphical schemes. Matrix representations are particularly useful for computations or storage on IT systems.

For any graph, two matrices can be defined: the *adjacency matrix* and the *incidence matrix*. The adjacency matrix, which models the adjacency of vertices, is the most often used. It is a square matrix with n rows and columns, where n represents the graph's order. Let i be the index of vertex x and j be the index of vertex y. The element in the i-th row and j-th column of the adjacency matrix indicates the number of edges between x and y. Loops are counted twice.

It is easy to see that the adjacency matrix of an undirected graph is symmetric. The adjacency matrix of the graph in figure 1.1 (by using the vertices' order: a, b, c, d, e) is:

$$\begin{pmatrix} & a & b & c & d & e \\ a & 0 & 1 & 1 & 0 & 0 \\ b & 1 & 2 & 0 & 1 & 0 \\ b & 1 & 0 & 0 & 2 & 0 \\ d & 0 & 1 & 2 & 0 & 1 \\ e & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Some adjacency matrices have special shapes. For instance, in a complete graph, the adjacency matrix only contains 1s. In a bipartite graph, the adjacency matrix has a special form if vertices get conveniently rearranged (how?).

The incidence matrix is used to represent both vertices and edges. In this matrix, the rows correspond to vertices, while the columns correspond to edges. If an edge j is incident to vertex i, then the corresponding matrix element (at position (i, j)) is 1; otherwise, it is 0. When dealing with loops, we use the value 2. For the last adjacency matrix, the corresponding incidence matrix is:

$$\begin{pmatrix} & A & B & C & E & F & G & H \\ a & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ b & 1 & 0 & 2 & 0 & 0 & 0 & 1 \\ c & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ d & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ e & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Non-matrix representations are also possible. For example, we can use adjacency lists, in which linked lists are employed to store the graph. A first list is used to store the names of the vertices. For each entry, another linked list indicates the adjacent vertices. If there are many edges between two vertices, the adjacency lists repeat them as many times as there are edges between them. For the precedent example, the adjacency lists are:

Vertex	Adjacency list
a	b, c
b	a, b, b, d
c	a, d, d
d	b, c, c, e
e	d

2.10. Graph coloring

Coloring the vertices of a simple graph involves assigning a color (an abstract information) to each vertex such that adjacent vertices do not have the same color. While the intuitive solution is to use a different color for each vertex, the graph coloring problem tries to find the smallest number of colors needed to meet the coloring constraints. A graph coloring with k colors is equivalent to creating a partition of the vertices into k stable sets.

The term "coloring" traditionally arises from map colorization in printing. In fact, when dealing with a map featuring multiple countries, the goal is to assign each country a color in such a way that neighboring countries are not painted with the same color, while using the fewest number of colors possible.

The minimal number of colors needed to color a graph G is called the *chromatic number* (denoted as $\delta(G)$). In general, computing $\delta(G)$ is a very difficult problem. However, one can make interesting approximations by either computing upper and lower bounds of $\delta(G)$ or using approximate (heuristic) algorithms.

Let us begin with an interesting theorem that establishes an upper bound on the chromatic number for planar graphs.

Theorem 1.7

(called Four colors theorem) Any planar graph can be colored with at most 4 colors.

The inverse theorem is not true; a graph with a chromatic number less than 5 is not necessarily planar.

For an arbitrary graph G, we can define the following bounds for its chromatic number:

- **Upper bounds:**
  - If G is not complete, then $\delta(G) \leq r$, where r is the greatest degree of its vertices ($r = \max_x d(x)$). If G is complete (the graph K_n), then its chromatic number is n. For the graph in figure 1.11 , this upper bound is 4.
  - $\delta(G) \leq n + 1 - \alpha(G)$, where n is the order of G and $\alpha(G)$ is the stability number of G. For the considered graph, this bound is 7+1-3=5. Next, we take the least value among the upper bounds (in this case 4).
- **Lower bounds:**
  - The chromatic number of a graph G is always greater than the chromatic number of any of its subgraphs. This can be helpful if the number of subgraphs is not large.
  - The chromatic number is greater than or equal to the order of the maximum clique. In the given example, the order of the maximum clique is 4, hence $\delta(G) \geq 4$.

As stated earlier, computing the optimal value of the chromatic number is a hard problem. Besides upper and lower bounds, some approximate algorithms (heuristics) permit finding an acceptable value of the chromatic number while maintaining a low level of complexity (run time). One of these algorithms is the Welsh and Powell's algorithm. On average, this algorithm performs well; however, it can yield poor results for some graphs.

Algorithm of Welsh and Powell

```
Sort the vertices in descending order in function of their degrees (x_1..x_k)
Start with a given color c
while there are non-colored vertices
    for non colored x in x_1..x_k
        if there is no adjacent vertex of x with color c
            color(x) = c
    c =a new color
```

We will apply this algorithm to the graph in figure 1.11 (by using the vertices' names in figure 1.15 and the colors: red, yellow, green and blue). First, vertices are sorted in a descendant order in function of the degrees then we color the vertices (NA means not assigned). We can see here that the algorithm computes the optimal value of the chromatic number.

Vertex	***a***	***b***	***d***	***e***	***c***	***f***	***g***
Degree	4	4	4	4	2	1	1
Initialization	NA	NA	NA	NA	NA	NA	NA
Iteration 1	Red	NA	NA	NA	NA	Red	Red
Iteration 2	Red	Yellow	NA	NA	NA	Red	Red
Iteration 3	Red	Yellow	Green	NA	Green	Red	Red
Iteration 4	Red	Yellow	Green	Blue	Green	Red	Red

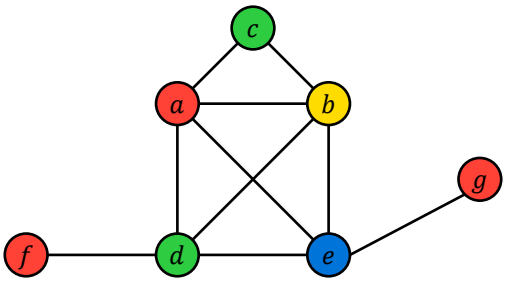


Figure 1.15 - example of graph coloring

Graph coloring allows for characterizing bipartite graphs. Actually, the chromatic number of a graph is 2 if and only if it is a bipartite graph. This is because in a bipartite graph, the edges connect a vertex from one set, X, to a vertex from a another set, Y, without any edges between vertices of the same set. Hence, the vertices of X can be colored with one color, and the vertices of Y can be colored with another color (as shown in figure 1.16 .

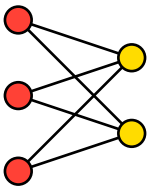


Figure 1.16 - coloring a bipartite graph

Remark : A similar problem can be defined for the edges. Edge coloring consists of assigning colors to the edges so that two adjacent edges cannot have the same color. To solve this problem, there is no need for a new algorithm; it is sufficient to build the line graph and then apply the same vertex coloring algorithms.

3. Directed graphs

Definition 1.8

A *directed graph* is defined by a pair (X, E) where X is a set of vertices and $E \subseteq X \times X$ is a set of *arcs* that model an asymmetric relation over the vertices X.

Directed graphs represent a one-way relationship between vertices, unlike undirected graphs. In an arc (x, y), x is called the source of the arc, and y is the target of the arc. Notice that the arc (x, y) is different of (y, x).

Almost all concepts regarding undirected graphs can be applied to directed graphs as well, albeit with some distinctions. For example, a directed graph can also be weighted.

Directed graphs are commonly known as p-graphs, where p represents the maximum number of arcs that can connect vertices. In this course, we will only consider 1-graphs. These are directed graphs where at most one arc can connect a vertex to another. This means that the considered graphs do not contain loops.

Example 1.16 : a directed graph

In a tournament, players compete in pairs. The following results were obtained:
- Player A beats B and D
- Player B beats C and D
- Player C beats A
- Player D beats C

This situation can be modeled by the graph in figure 1.17 .

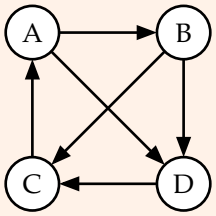


Figure 1.17 - the graph modeling the results of the tournament

3.1. Predecessors, successors and neighbors

In a directed graph $G = (X, E)$, the predecessor of a vertex x is any vertex y such that (y, x) is an arc (i.e., $(y, x) \in E$). The set of all predecessors of x is denoted as $\Gamma^-(x)$. In the graph in figure 1.17 , we have : $\Gamma^-(C) = \{B, D\}$ (players who have beaten C).

The successor of a vertex x is any vertex y such that (x, y) is an arc (i.e., $(x, y) \in E$). The set of all successors of x is denoted as $\Gamma^+(x)$. In the graph in figure 1.17 , we have : $\Gamma^+(C) = \{A\}$ (players who have been beaten by C).

The neighbors of a vertex x is the union of its predecessors and its successors $\Gamma(x) = \Gamma^+(x) \cup \Gamma^-(x)$. There, we have: $\Gamma(C) = \{A, B, D\}$.

3.2. The degrees of a directed graph

Obviously, a directed graph can still be characterized by its order (the number of vertices) and its size (the number of arcs). Each vertex can be further described by its degree, but it's important to distinguish between incoming arcs and outgoing arcs.

For a vertex x, the outdegree, denoted as $d^+(x)$, is defined as the cardinality of $\Gamma^+(x)$. The indegree, denoted as $d^-(x)$, is defined as the cardinality of $\Gamma^-(x)$. The degree of x ($d(x)$) is the sum of its indegree and its outdegree ($d(x) = d^+(x) + d^-(x)$).

For the graph in figure 1.17 , the degrees are summarized in the following table:

Vertex	A	B	C	D	Sum
$d^+(x)$	2	2	1	1	6
$d^-(x)$	1	1	2	2	6
$d_G(x)$	3	3	3	3	12

There is a relationship between the indegrees and outdegrees in a directed graph:

Theorem 1.8

Let $G = (X, E)$ be a directed graph. We have:$\sum_{x\in X} d_G^+(x) = \sum_{x\in X} d_G^-(x)$

As a result, the sum of indegrees and outdegrees is always even.

3.3. Paths and circuits

Since an arc has a direction, we cannot use the same definitions of walks and cycles as in undirected graphs.

Definition 1.9

Let $G = (X, E)$ be a directed graph, a *path* between the vertices x and y (in contrast to a walk) is a sequence of vertices $x_0, x_1,\ldots, x_k$ such that $x_0 = x$, $x_k = y$ and $(x_{i-1}, x_i) \in E$ for all i (in other words, (x_{i-1}, x_i) is an arc for all i).

It is crucial to consider the direction of arcs here because if $x_0, x_1,\ldots, x_k$ is a path, then $x_k, x_{k-1}, ..., x_0$ may not be a path. In figure 1.17 , the length of the path A, B, C is 2. The length of the path A is 0. If the beginning of a path coincides with its end, then we have a *circuit* ($x_0 = x_k$). In this considered graph, A, B, C, A is a circuit.

Consider a subset A of X. Let $w^+(A)$ represent all arcs with sources in A, and let $w^-(A)$ represent all arcs with targets in A. The *cocircuit* of A, denoted as $w(A)$, is the union of $w^+(A)$ and $w^-(A)$. This corresponds to the minimum set of arcs that need to be removed in order to isolate the vertices in A. In the considered graph, if $Z = \{A, B\}$ then its cocircuit is $w(Z) = \{(A, D), (B, C), (B, D), (C, A)\}$.

3.4. Non-graphical representations of directed graphs

A directed graph can be represented by the means of the adjacency matrix and the incidence matrix. The adjacency matrix is a square matrix of n rows and n columns (where n is the graph order). If there is an arc from x to y then we put 1 in the corresponding element of the adjacency matrix, otherwise it is 0.

The adjacency matrix of the graph in figure 1.17 is:

$$\begin{pmatrix} & A & B & C & D \\ A & 0 & 1 & 0 & 1 \\ B & 0 & 0 & 1 & 1 \\ C & 1 & 0 & 0 & 0 \\ D & 0 & 0 & 1 & 0 \end{pmatrix}$$

The adjacency matrix is usually asymmetric. The incidence matrix needs to distinguish between incoming and outgoing arcs. Positive values (usually +1) are used to indicate the targets of arcs, and negative values (usually −1) are used to indicate the sources of arcs. The incidence matrix of the considered graph is:

$$\begin{pmatrix} & 1 & 2 & 3 & 4 & 5 & 6 \\ A & -1 & 0 & 0 & 1 & -1 & 0 \\ B & 1 & -1 & 0 & 0 & 0 & -1 \\ C & 0 & 0 & 1 & -1 & 0 & 1 \\ D & 0 & 1 & -1 & 0 & 1 & 0 \end{pmatrix}$$

Adjacency lists can also be used to represent directed graphs. However only the successors of vertices are represented.

The adjacency matrix offers a significant advantage compared to other representations. Let M be the adjacency matrix of a graph. Firstly, remember that M gives the number of arcs between the vertices. In other words, M gives the number of paths of length 1 in the graph.

As M is a square matrix, we can calculate the matrix multiplication $M^2 = M \times M$. The element at position (i, j) of M^2 is given by the formula: $\sum_k M_{i,k}.M_{k,j}$. This calculates the number of paths from vertex i to vertex j through various vertices k. Therefore, it's evident to see that M^2 represents the number of paths of length 2 in the graph.

By generalizing, we get the matrix $M^n = \overbrace{M \times M ... \times M}^{n \text{ times}}$. It yields the number of paths of length n in the graph.

Power of M	Result	Power of M	Result
M	$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	M^2	$\begin{pmatrix} 2 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$
M^3	$\begin{pmatrix} 0 & 0 & 2 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$	M^4	$\begin{pmatrix} 1 & 2 & 0 & 2 \\ 0 & 1 & 2 & 2 \\ 2 & 0 & 1 & 0 \\ 0 & 0 & 2 & 1 \end{pmatrix}$

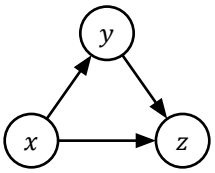
3.5. Transitive graph and transitive closure

A directed graph models a binary relation over a set of vertices. Some relational properties can be extended to graphs. In particular, the transitivity of a relation R means that if xRy and yRz then xRz. By extension, we can make the following definition:

Definition 1.10

A *transitive graph* is defined as follows: whenever there exists an arc from x to y, and another arc from y to z, it follows that there exists an arc from x to z.

A transitive graph can alternatively be defined as: if there exists a path of length $n > 1$ from x to y, then there exists an arc connecting directly x to y. This is illustrated in the following graph ($n = 2$):



For any intransitive graph G, it is possible to build a transitive graph G^* called the *transitive closure* of G. The idea is to add a minimum number of arcs to G so that it becomes transitive. The algorithm repeatedly applies the following principle: if there exists an arc from vertex x to vertex y and another arc from vertex y to vertex z, then add a new arc (if it does not exist) from x to z.

In figure 1.18 , an intransitive graph has been transformed into a transitive graph by adding arcs. The resulting graph (the combination of the black and red lines) represents the graph's transitive closure.

The transitive closure of a graph gives all of the paths that start at a particular vertex. This can be used to solve the reachability problem, which involves determining if a vertex can reach another vertex.

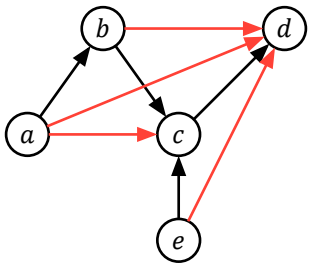


Figure 1.18 - the transitive closure of a graph (red arcs have been added to make the first graph transitive)

Many algorithms can be used to compute the transitive closure. One such method utilizes the adjacency matrix. Let M represent the adjacency matrix of a graph. We have already shown that the matrix M^i determines the number of

paths of length i within the graph. By computing successive powers of M (from 1 up to $n-1$, where n is the graph order), all possible paths are generated. It's worth noting that in a graph comprising n vertices, a non-cyclic path can have a maximum length of $n-1$. The following algorithm summarizes that:

Algorithm of building the transitive closure of a graph G

1 $n = \text{order}(G)$
2 M=Adjacency matrix of G
3 Compute $M^* = M \oplus M^2 \oplus \ldots \oplus M^{n-1}$

$\oplus$ is a boolean sum, i.e., $u \oplus v = 1$ if $u \neq 0$ or $v \neq 0$, otherwise it is 0. By applying this algorithm on the graph in figure 1.18 (only on the black part), we obtain:

M	M^2	M^3	M^4
$\begin{pmatrix} 0&1&0&0&0\\ 0&0&1&0&0\\ 0&0&0&1&0\\ 0&0&0&0&0\\ 0&0&1&0&0 \end{pmatrix}$	$\begin{pmatrix} 0&0&1&0&0\\ 0&0&0&1&0\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&0&1&0 \end{pmatrix}$	$\begin{pmatrix} 0&0&0&1&0\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0 \end{pmatrix}$	$\begin{pmatrix} 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0 \end{pmatrix}$

The matrix M^* is:

$$\begin{pmatrix} 0&1&1&1&0\\ 0&0&1&1&0\\ 0&0&0&1&0\\ 0&0&0&0&0\\ 0&0&1&1&0 \end{pmatrix}$$

This algorithm is inefficient since its complexity is $O(n^4)$. A more optimized version, called Warshall's algorithm, allows a complexity of $O(n^3)$:

Algorithm of Warshall

1 $n = \text{order}(G)$
2 M=Adjacency matrix of G
3 $R^{(0)} = M$
4 **for** $k = 1..n$
5 | **for** $i = 1..n$
6 | | **for** $j = 1..n$
7 | | | $R^{(k)}_{(i,j)} = R^{(k-1)}_{(i,j)} \vee \left(R^{(k-1)}_{(i,k)} \wedge R^{(k-1)}_{(k,j)}\right)$
8 $M^* = R^{(n)}$

By applying the Warshall algorithm on the previous graph, we obtain:

$R^{(0)}$	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$
$\begin{pmatrix} 0&1&0&0&0\\ 0&0&1&0&0\\ 0&0&0&1&0\\ 0&0&0&0&0\\ 0&0&1&0&0 \end{pmatrix}$	$\begin{pmatrix} 0&1&0&0&0\\ 0&0&1&0&0\\ 0&0&0&1&0\\ 0&0&0&0&0\\ 0&0&1&0&0 \end{pmatrix}$	$\begin{pmatrix} 0&1&1&0&0\\ 0&0&1&0&0\\ 0&0&0&1&0\\ 0&0&0&0&0\\ 0&0&1&0&0 \end{pmatrix}$	$\begin{pmatrix} 0&1&1&1&0\\ 0&0&1&1&0\\ 0&0&0&1&0\\ 0&0&0&0&0\\ 0&0&1&1&0 \end{pmatrix}$

$R^{(4)}$	$R^{(5)} = M^*$
$\begin{pmatrix} 0&1&1&1&0\\ 0&0&1&1&0\\ 0&0&0&1&0\\ 0&0&0&0&0\\ 0&0&1&1&0 \end{pmatrix}$	$\begin{pmatrix} 0&1&1&1&0\\ 0&0&1&1&0\\ 0&0&0&1&0\\ 0&0&0&0&0\\ 0&0&1&1&0 \end{pmatrix}$

3.6. Strongly connected graph and strongly connected components

A directed graph is *connected* if when its arcs are turned into edges, the resulting graph is connected. A directed graph is *strongly connected* if there is at least one path between any two vertices (x, y). In other words, each vertex may be reached from any other vertex.

A *strongly connected component* is a maximum subgraph that is strongly connected. A strongly connected graph has one strongly connected component that equals the graph itself.

If a graph is not strongly connected, it can be decomposed into a set of strongly connected components. To achieve that, we use a *marking algorithm*.

Algorithm for building the strongly connected components of a graph G

1. $k = 0$
2. Choose a vertex x **and** mark it by (+) **and** (-)
3. $k = k + 1$
4. Mark all the direct **or** indirect successors of x by (+)
5. Mark all the direct of indirect predecessors of x by (-)
6. Marked vertices by (+) **and** (-) define the strongly component C_k
7. Remove all the vertices **in** C_k from G
8. **repeat** from 2. **until** there are no vertices left

Let us apply this algorithm on the graph in figure 1.19 .

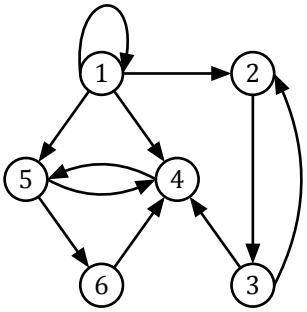


Figure 1.19 - an example of non-strongly connected graph

Thus, we obtain:

Vertex	1	2	3	4	5	6	**Component**
Marking	+-	+	+	+	+	+	$C_1 = \{1\}$
Marking		+-	+-	+	+	+	$C_2 = \{2, 3\}$
Marking				+-	+-	+-	$C_3 = \{4, 5, 6\}$

Once the strongly connected components are computed, it is possible to create a new graph called the reduced graph. In this graph, the vertices are the strongly connected components. The reduced graph has an edge from vertex i to vertex j if, in the original graph, there is an edge from a vertex x from the strongly connected component i to a vertex y from the strongly connected component j. In this considered example, the reduced graph is given by figure 1.20 .

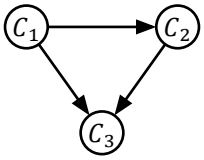


Figure 1.20 - the reduced graph of the graph

3.7. Graph scheduling (topological sorting)

Consider a directed, connected and acyclic graph. The scheduling of vertices in such a graph consists of organizing them into levels so that arcs always go from a lower level to an upper one. This kind of scheduling is also known as *topological sorting*.

Formally, topological sorting of a directed graph $G = (X, E)$ is equivalent to define a function $f : X \to \mathbb{N}^*$ that specifies the level of each vertex. The function f should respect the following constraints:

- For each arc $(x, y) \in E$, $f(x) < f(y)$.
- f should minimize $\max_x f(x)$ (i.e., the number of potential levels).

In order to make a a topological sorting, we use the predecessor function. This is done thanks to the following algorithm:

Algorithm of topological sorting of a graph G

```
1 rank = 1
2 A = the set of all vertices x such that Γ⁻(x) = ∅
3 for x in A
4 |   level(x) = rank
5 rank = rank + 1
6 Remove all vertices in A from G
7 repeat from 2. until there are no vertices left
```

We will apply this algorithm on the graph in figure 1.21 .

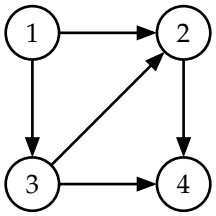


Figure 1.21 - a graph to be sorted topologically

This algorithm is applied as follows:

x	Γ^-	Action	x	Γ^-	Action	x	Γ^-	Action	x	Γ^-	Action
1	$\emptyset$	lev(1) = 1	1	-		1	-		1	-	
2	1,3		2	3		2	$\emptyset$	lev(2) = 3	2	-	
3	1		3	$\emptyset$	lev(3) = 2	3	-		3	-	
4	2,3		4	2,3		4	2		4	$\emptyset$	lev(4) = 4

The ranking of vertices into different levels (1 up to 4) is shown in figure 1.22 .

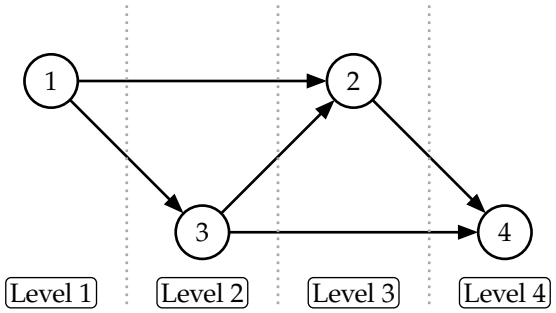


Figure 1.22 - ranking of the vertices

3.8. Searching graphs for circuits

Searching graphs for circuits has several applications. Consider a waiting graph, with vertices representing processes and arcs representing waiting relationships (a process is blocked while waiting for another process). Looking for circuits in such a graph enables the detection of deadlocks.

Many algorithms can be used to search for circuits, including the transitive closure and the Warshall algorithm. In this case, if the matrix diagonal has a non-null element at any iteration, then the graph contains a circuit. However, this method can only detect circuits and cannot build them.

We can also use topological sorting to discover circuits. In fact, if the topological sorting is carried out until the end, then the graph does not contain a circuit. However, if, at any given iteration, we discover that all vertices have at least one predecessor, then the graph certainly contains a circuit. This one can be built by applying the following algorithm:

Algorithm for detecting and building a circuit in a graph G

```
Apply topological sorting until all vertices have at least one predecessor or the end of the algorithm
if all vertices have been ranked
    The graph does not contain a circuit
else
    The graph contains a circuit
    Select one of the vertices and mark it
    Select a unmarked predecessor of the last mark vertex and mark it
    repeat step 7 until there are only marked vertices
    Let x_0, x_1, ..., x_0 be the sequence of vertices such that x_0 is the first vertex to appear twice
    The circuit is formed by reversing the order of this sequence (i.e., x_0, ..., x_1, x_0)
```

Let us apply this algorithm to the graph in figure 1.23 .

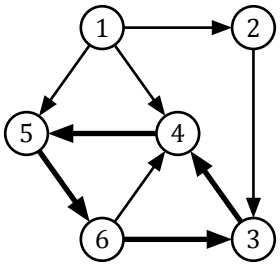


Figure 1.23 - an example of a graph with a circuit

x	Γ^-	Action	x	Γ^-	Action	x	Γ^-	Action
1	2		1	∅	lev(1)=2	1	-	There is a circuit
2	∅	lev(2) = 1	2	-		2	-	
3	2,6		3	6		3	6	
4	1,3,6		4	1,3,6		4	3,6	
5	1,4		5	1,4		5	4	
6	5		6	5		6	5	

We arbitrarily choose the vertex 3 and mark it, then we mark:
- Vertex 6 (since it is the only predecessor or 3)
- Vertex 5 (since it is the only predecessor or 5)
- Vertex 4 (since it is the only predecessor or 4)
- Vertex 3 (chosen arbitrarily here)

We obtain the sequence 3, 6, 5, 4, 3. Therefore, the circuit is 3, 4, 5, 6, 3.