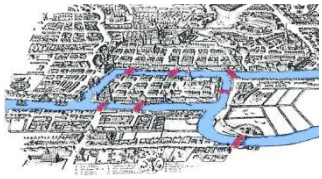
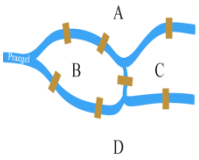
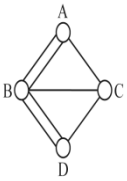
Chapter I *Basic Definitions*

Introduction

The history of graph theory begins with the work of Euler in the 18th century to study certain problems, such as that of the 7 bridges of Königsberg (the inhabitants of Königsberg, a city in Russia, wondered if it was possible, starting from any district of the city, to cross all the bridges without passing twice by the same one and to return to their starting point).







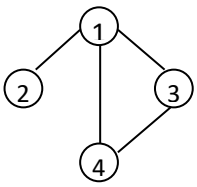
Generally speaking, a graph allows us to represent the structure and connections of a set of elements by expressing the relationships between them.

Graph theory is used in several fields such as: communication networks, road networks, interaction of various animal species, electrical circuits, social networks, chemistry, language theory, etc.

Example problems: Exam timetable

We have 4 subjects with the following interactions: subjects 1 and 2 have students in common as well as subjects 1 and 3 and subjects 1 and 4 and subjects 3 and 4.

The problem is how to organize the exams in a minimum of time, taking into account the subjects with students in common.



I- Definitions:

1- Intuitive definition:

A graph is a set of points connected (or not) by segments.

- The points are called “vertices” or “nodes” or “summits”
- Segments can be:

  - Without direction: these are called “**edges**” and the graph is “**undirected**”. Example, a road network with two-way roads (see G1).

  - With direction (arrows): they are called “**arcs**” and the graph is called an “**directed**” graph. Example, a road network whose roads are one-way (see G2).

2- Mathematical definition:

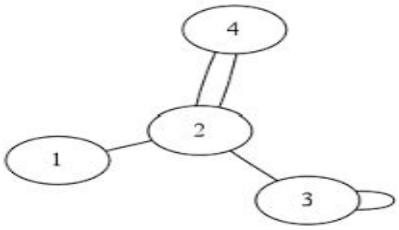
a) **Undirected graph:** a graph G is defined by **G=(X,U)** such that:

X: the set of vertices

U: the set of edges. Edges are **unordered** pairs of vertices.

Example: let G1 = (X,U) such that X = {1,2,3,4} and

U={u1,u2,u3,u4,u5} with u1=[1,2], u2=[2,3],u3=[3,3],u4=[2,4],u5=[4,2]



G1

b) **Directed graph:** A directed graph is defined by **G=(X,U)** such that:

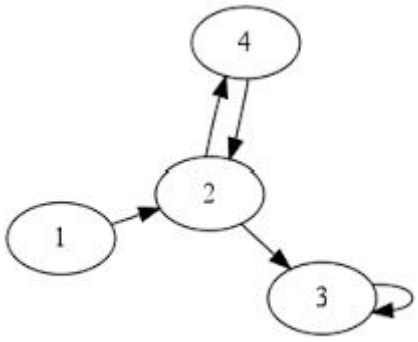
X: the set of vertices

U: the set of arcs. Arcs are **ordered** pairs of vertices.

Example: let G2 = (X,U) such that

X = {1,2,3,4}

U={u1,u2,u3,u4,u5}with: u1=(1,2),u2=(2,3),u3=(3,3),u4=(2,4),u5=(4,2)



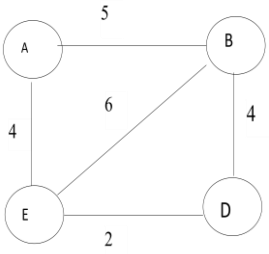
(G2)

c) **Valued graph**: a valued graph (directed or undirected) is defined by G=(X,U,V) such that:

- X: the set of vertices

- U: the set of arcs/edges.
- V: function that assigns a value to each arc/edge.

G3



II- Basic Concepts

Undirected graph	Directed graph
Vertex / Summit	Vertex / Summit
Edge	Arc
Chain	Path
Cycle	Circuit

1-**Adjacent vertices**: In an undirected (resp. directed) graph, two vertices are adjacent if there is an edge (resp. arc) that connects them.
Example: in the 2 graphs G1 and G2, the two vertices 1 and 2 are adjacent and 1 and 3 are not adjacent.

2- **Edges (arcs) are adjacent** if they have at least one vertex in common (eg, u1 and u2).

3- **An edge (arc) u is incident** to a vertex **x** if **x ∈ u** (x is an endpoint of u).

4-**Loop:** is an edge (arc) whose two ends are identical (connecting the same vertex twice) (eg, u3).

5- **Multiple edges (arcs):** More than one edge (arc) connecting the same pair of vertices.

6- **The order of a graph** is the total number of vertices it contains (the order of G1 and G2 is 4).

7- **The degree of a vertex** is the number of edges (arcs) that connect it to other vertices in the graph. The loop counted 2 times (eg, in G1 deg(1)=1, deg(2)=4, deg(3)=3…).

8- **The degree of a graph** is the max of the degrees of the vertices it contains (the degree of G1 and G2 is 4).

9- **The handshake property**: the sum of the degrees of all vertices in a graph is equal to twice the total number of edges (arcs).

$$\sum d\,(x) = 2 \times card(U)$$

10- **Chain (path)**: A sequence of vertices connected by edges (arcs). The length of the chain (path) is the number of edges (arcs) visited to get from the first vertex in the chain (path) to the last.

11- **Elementary chain (path)**: we do not pass through the same vertex more than once.

12- **Simple chain (path)**: we do not pass more than once over the same edge (arc).

13- **Closed chain (path):** the starting vertex and the ending vertex are identical.

14- **Cycle (circuits):** simple chain (path) closed.

15- **Eulerian chain (path)** : is a chain (path) satisfying the following conditions :

a. It contains all the edges (arcs) of the graph ;

b. Each edge (arc) is described only once time.

16- **Eulerian cycle (circuit):** A Eulerian cycle (circuit) is a Eulerian chain (path) whose starting vertex and arrival vertex are the same.

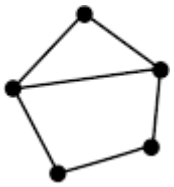
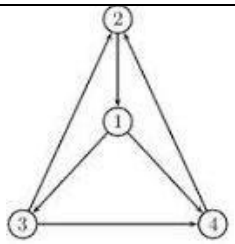
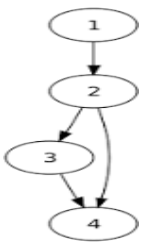
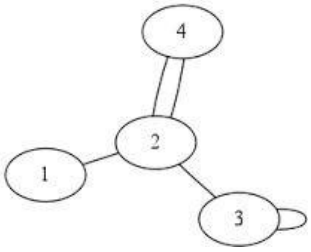
17- **Hamiltonian chain (path) :** is a chain (path) satisfying the following conditions:

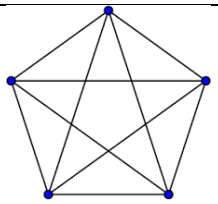
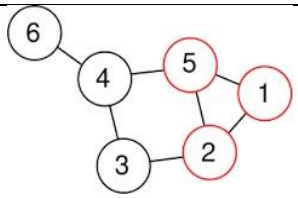
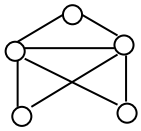
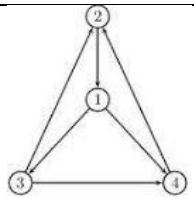
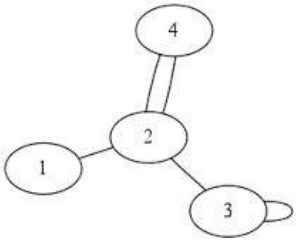
a. It contains all the vertices of the graph ;

b. Each vertex is described only once (it only consists of vertices that are different).

18- **Hamiltonian cycle (circuit):** A Hamiltonian cycle (circuit) is a Hamiltonian chain (path) whose starting vertex and arrival vertex are the same.

III- Some types of graphs:

Graph	Definition	Example
Planar graph	Can be drawn on the plane without crossing edges (arcs)	
Regular graph	All vertices have the same degree.	
Simple graph	Does not contain loops and multiple edges (arcs).	
Multi-graph	Contains loops and/or multiple edges (arcs).	

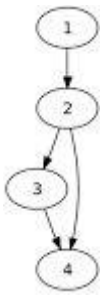
Complete graph	Is a simple graph whose vertices are all adjacent (it has an edge (arc) between any pair of summits).	
Click In A graph	A complete subgraph	
Eulerian graph	It contains an Eulerian cycle (circuit). So, it is a graph that can be drawn without ever lifting the pencil and without going two times by the same edge (arc).	
Hamiltonian graph	it contains an Hamiltonian cycle (circuit)	
Connected graph	There exists a chain (path) between any pair of vertices.	

IV- Matrices associated with a graph

The goal of matrices is to create a **computer representation** of the graph so that it can be processed **automatically** .

1- The Adjacency Matrix

The adjacency matrix associated with a graph G(X,U) of order n is an nxn matrix, such that the element $m_{i,j}$ is defined:



Sommet / sommet	1	2	3	4
1	0	1	0	0
2	0	0	1	1
3	0	0	0	1
4	0	0	0	0

$$m_{i,j} = \begin{cases} \text{number of arcs/edges connecting (xi and xj)} \\ 0 \text{ if } (xi, xj) \notin U \end{cases}$$

Other examples to see during the course

RQ:

1. In the case of a simple graph, the matrix is Boolean and the diagonal is zero.
2. The matrix associated with an undirected graph is always a symmetric matrix.
3. The degree of a vertex can be found from the adjacency matrix of a graph.
   - For an undirected graph without loops, it is sufficient to sum the coefficients on the row (or column) corresponding to the vertex,
   - For a directed graph, we do the sum on the line and on the column.

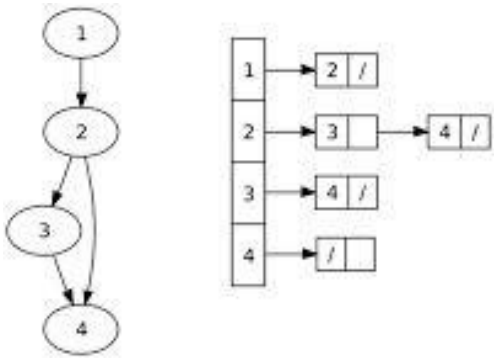
     The line represents the positive degrees d+ (outgoing arcs)

     The column represents negative degrees d- (incoming arcs)

4. From the adjacency matrix we can find the chains (paths) of length N between any pair of vertices by calculating M^N . The coefficient located at the intersection of the i^{th} row and the j th column of M^N is equal to the number of chains (paths) connecting vertex i of the graph to vertex j.
5. The trace of a matrix M^N represents the number of circuits (cycles) of length N.

2- The Adjacency List .

Another idea to represent matrices in memory: adjacency lists. In this type of structure, we have an array of lists of vertices. The array has as many cells as there are vertices. Each cell points to a list of vertices. This list represents the successors of the vertex in question.



3- **The Incidence Matrix:** Let G be a directed graph which has n vertices numbered from 1 to n, and m arcs numbered from 1 to m.

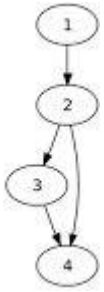
The incidence matrix of the graph is called the matrix $MI=(a_{i,j})$ including n rows and m columns such that:

- $a_{i,j}$ is worth +1, if the arc numbered j admits the vertex i as origin;
- $a_{i,j}$ is -1, if the arc numbered j admits the vertex i as arrival;
- $a_{i,j}$ is 2 , if the arc numbered j is a loop on the vertex i

- ai,j is 0 in the other cases .

a1: (1,2) a2: (2,3)

a3: (2,4) a4: (3,4)



Edge (arc) / summit	a1	a2	a3	a4
1	1	0	0	0
2	-1	1	1	0
3	0	-1	0	1
4	0	0	-1	-1

RQ .

- If the graph is undirected, there is no longer any notion of origin and arrival of an edge. We therefore put +1 in place of +1 or -1, and we put 0 elsewhere.

V- Connectivity:

A **connected graph** is a graph such that for any pair of vertices ***x*** and ***y*** there exists a **chain** or **path** connecting ***x*** and ***y*** .

The notion of connectivity is an essential property of graphs, especially for network problems (**each vertex is accessible from any other vertex**).

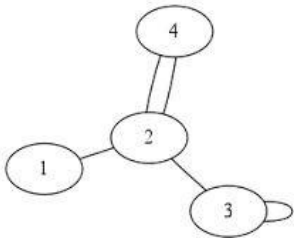
V.1- Simple connectivity: a graph is connected if and only if:

$$\forall\, i\,, j\ \in X \begin{cases} i = j \\ \exists\ a\ \textbf{chain}\ connecting\ i\ et\ j \end{cases} or$$

Between any pair of vertices it is possible to find a **chain between these 2 vertices.**

RQ: if a graph is not **connected** it admits at least 2 connected components.

Example: G is connected



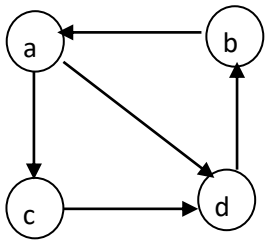
V.2- Strong connectivity: a graph is strongly connected if and only if:

$$\forall\, i\,, j\ \in X \begin{cases} i = j \\ \exists\ a\ \textbf{path connecting}\ i\ et\ j \end{cases} or$$

Between any pair of vertices there is a **circuit that passes through these 2 vertices.**

RQ: if a graph is not **strongly** connected it admits at least 2 strongly connected components.

Example: G is connected

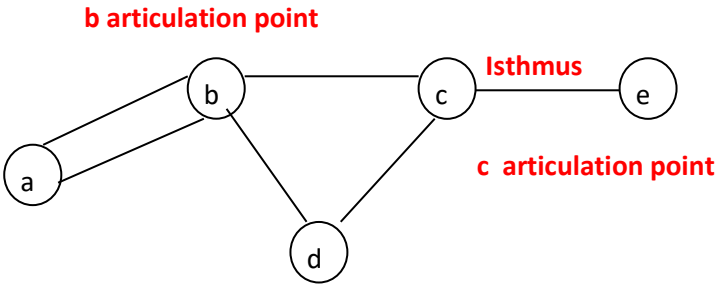


RQ: - the connectedness relation (simple or strong) is an equivalence relation.

- A graph is connected (or strongly connected) if the number of its connected (or strongly connected) components is equal to **1**.

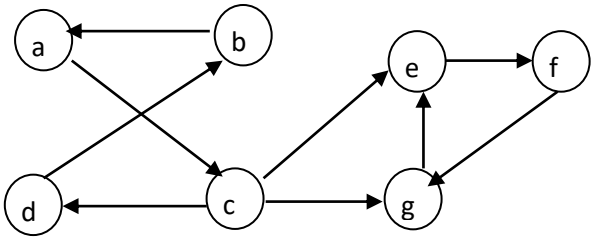
V.3- Definitions:

- **Articulation point of a graph:** is a vertex whose deletion increases the number of connected components.
- **Isthmus:** is an edge whose deletion increases the number of connected components.
- **Articulation set E ⊂X** of a connected graph **G** is a subset of vertices whose deletion gives a graph **G'** which is not connected.



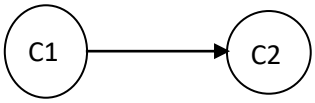
V.4- Reduced graph: let G=(X,U) be a directed graph, we call reduced graph G_R whose vertices are $C_1, C_2, \ldots$ the strongly connected components of **G** and there exists an arc between **Ci** and **Cj** iff there exists at least one arc between a vertex of **Ci** and a vertex of **Cj** in **G** .

Example :



This graph is not strongly connected but it contains 2 strongly connected components

C1 = { a,b,c,d } and **C2 = { e,f,g }** and gives the following reduced graph:



VI- Traversing a graph:

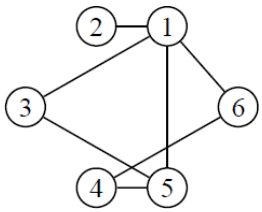
Using a graph as a model, we need an examination of the vertices. We can think of this examination as a walk along the edges/arcs during which we visit the vertices.
Most often, a graph traversal is a tool to study a global property of the graph:

- is the graph connected ?
- is the graph bipartite ?
- is the directed graph connected ?
- what are the articulation vertices ?

A graph traversal is any deterministic process that allows one to choose, from the visited vertices, the next vertex to visit. The problem consists of determining an order on the visits of the vertices.
A path is a sequence S of vertices such that:

- s is the first vertex of S
- Each vertex appears once and only once in S
- Every vertex except the root is adjacent to a vertex placed before it in the list.

Example :
6 1 4 2 3 5 is a path from 6



To explore a graph, there are two main traversal strategies:

- Depth-first browse
- The width first browse

Depth-first browse

It consists, starting from a given vertex, of following a chain (path) as far as possible. Once the path is finished or when we come back to a vertex already explored, we go back to take all the chains (paths) previously ignored. This type of path uses a stack. It works according to the following algorithm:

```
Begin
mark(s) ← 1;
For all successors si of s do
 If not mark(s i ) then
  Depth_path (s i );
 End IF
End For
End
```

Width path First

It consists of exploring the vertices of the graph level by level, starting from a given vertex. This type of traversal uses a queue instead. It works according to the following algorithm:

Graph_Width_Traversal Algorithm

```
Procedure Width_path (s0: vertex)
Data: G: graph
Variable : s, suc: vertex, F: vertex queue
Begin
Thread(s0,F);
mark (s0) ←1
While Non_File_empty(F) do
      s← scroll(F);
     For any adjacent juice to be do
          If not marked (suc) then
             mark (suc) ←1
             thread(suc,F);
          End IF
    End For
End While
END
```