Ministry of Higher Education and Scientific Research
Djilali Bounaâma University, Khemis Miliana
Faculty of Matter Sciences and Computer Science
Computer Science Department

GRAPH THEORY

Course and Tutorial Sessions

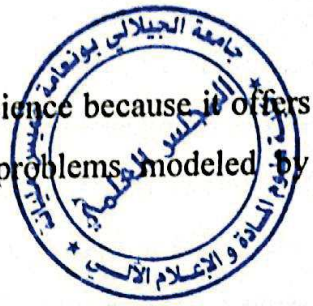
L2 Computer Science

By

Dr. Abdesselam KALI

November , 2025

Preface



Graph theory has become an essential part of computer science because it offers many techniques and algorithms for solving complex problems modeled by graphs.

This textbook reflects seventeen years of teaching experience in the Department of Mathematics and Computer Science at Djilali Bounaâma University, Khemis-Miliana. It is designed for second-year undergraduate students majoring in Computer Science. The goal is not to make students specialists in graph theory, but to show how graph properties can be used effectively to approach practical problems through mathematical reasoning. Each chapter includes exercises with detailed solutions to support classroom learning and tutorial sessions.

The material follows the official syllabus outlined in the most recent *Canevas* for the 2018–2019 academic year. The theory is introduced step by step, with emphasis on two main aspects: modeling problems with graphs and applying algorithms to solve them. The presentation is intentionally kept accessible rather than overly formal. Mathematical notation is simplified, and terminology is limited to what is necessary, ensuring clarity without oversimplification.

The primary bibliographic source used in preparing this textbook is **Claude Berge's** *Graphes et hypergraphes* (Bordas, 1973, 300 pages), which remains one of the foundational references in the field. Other works cited in the bibliography section have also been used as supporting material to enrich and complement the content presented here.

Students are expected to have a basic background in mathematics and algorithms to fully benefit from this course. The recommended weekly schedule includes

1 hour and 30 minutes of lectures and 1 hour and 30 minutes of tutorials. Assessment consists of continuous evaluation and a final exam. The module carries a coefficient of 2 and is worth 4 academic credits.

We would like to express our sincere gratitude to everyone who contributed to the preparation of this work. Our special thanks go to Dr. **Sakri Redha** Associate Professor (Class A), Department of Urban Hydraulics, National Higher School of Hydraulics (ENSH), and to Dr. **Bensaid Chaima** Associate Professor (Class A), Computer Science Department, Faculty of Matter Sciences and Computer Science, Djilali Bounaâma University, Khemis-Miliana, for their careful review and valuable suggestions. We are also thankful to our colleagues with whom we have shared many years of teaching mathematics in the Department of Mathematics and Computer Science.

Historical Introduction

Graph theory provides a formal framework for representing structures and relationships by modeling them as points, referred to as *vertices*, and by connections between these points, referred to as *arcs* or *edges*, depending on whether they are directed or undirected. Over time, it has become an indispensable theoretical and practical tool in the modeling of a wide variety of problems, such as those arising in communication networks, interactions among animal species, electrical circuits, the knight's tour on a chessboard, and the map-coloring problem, among others.

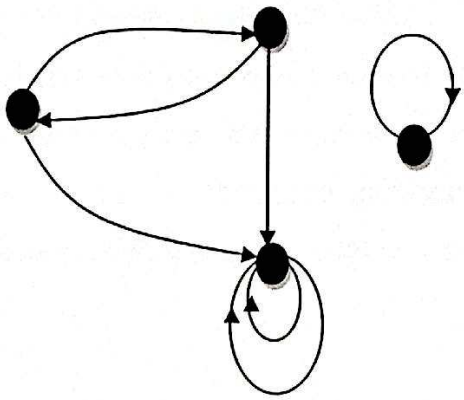


Figure 1. Example of a graph.

The first known problem involving the use of a graph is the *Seven Bridges of Königsberg*, a city now known as Kaliningrad. According to historical accounts, during a visit to the city in 1736, Leonhard Euler sought to determine a circular route that would start from any point, cross each of the seven bridges exactly once, and return to the starting point. This problem, which is now known as the

search for an Eulerian cycle in a graph, marked the origin of graph theory as a distinct field of mathematics.

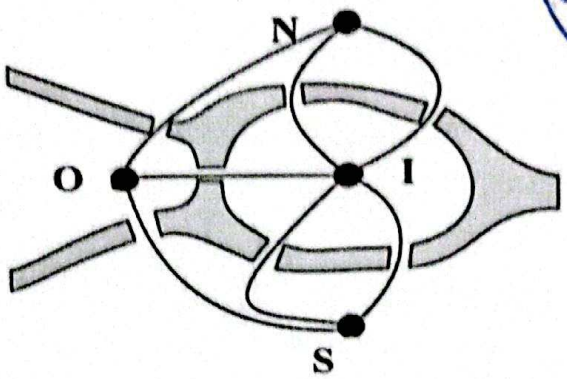


Figure 2. The Seven Bridges of Königsberg problem.

Euler proposed representing the problem as a graph. The different areas of the city were identified by their geographical positions: **N** (north), **S** (south), **O** (west), and **I** (island). Each bridge was then represented by an edge connecting the corresponding regions.

In 1847, Kirchhoff introduced the concept now known as the *spanning tree* to analyze the electrical current in each branch of a circuit. Kirchhoff used graphs to represent an electrical circuit with its resistors, capacitors, and other components, and demonstrated that to determine the current in each branch, it was not necessary to consider every cycle in the graph individually. Instead, a spanning tree could be used. This approach has since become a standard method in circuit analysis.

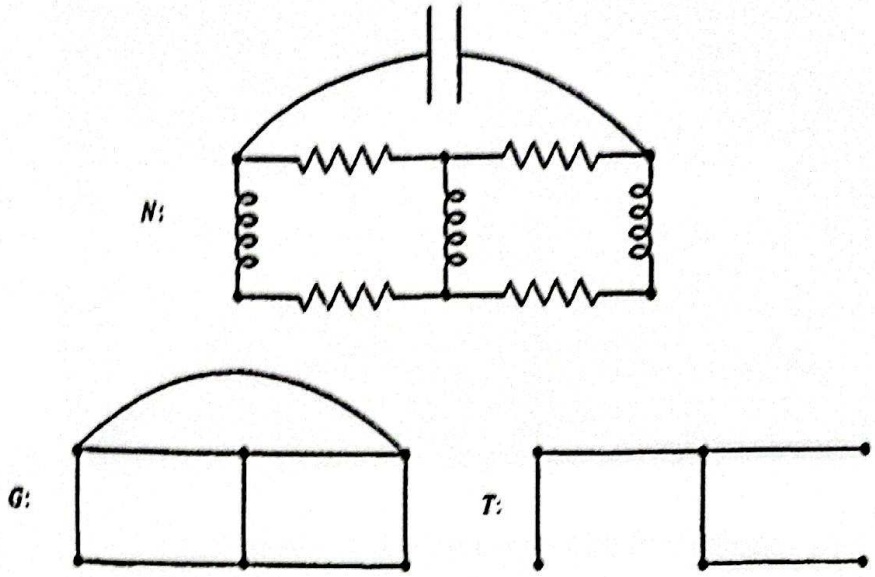


Figure 3. An electrical circuit N, its underlying graph G, and a spanning tree T.

In 1852, Francis Guthrie, while studying the coloring of a map of the regions of England, conjectured that any geographical map could be colored using only four different colors in such a way that any two adjacent regions would always receive distinct colors. The *Four Color Conjecture* is a problem in graph theory, since each map can be represented by a graph in which the countries (including the outer region) correspond to vertices, and two vertices are connected by an edge whenever the corresponding countries share a common border. Such a graph can clearly be drawn on a plane without any intersecting edges. Therefore, if it is possible to color the vertices of every such "planar" graph with four or fewer colors so that adjacent vertices have different colors, the Four Color Conjecture would be proven. This conjecture was finally demonstrated in 1976 by two American mathematicians, Kenneth Appel and Wolfgang Haken.

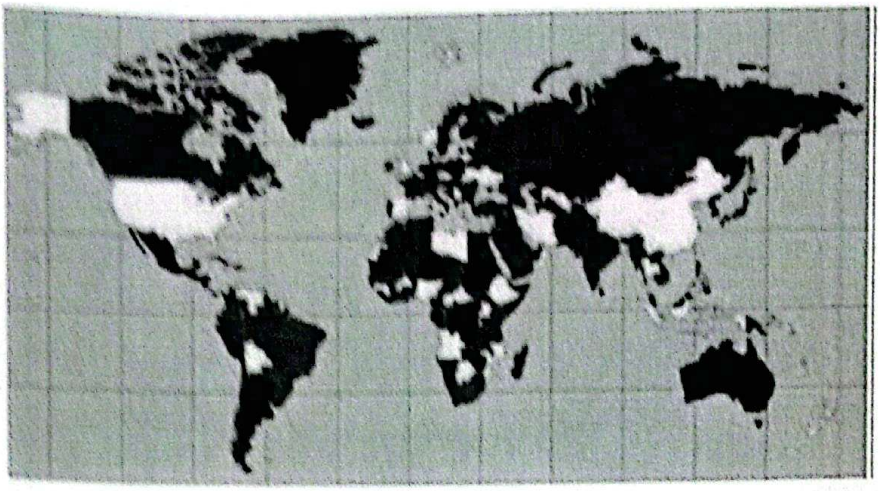


Figure 4. Coloring a map with four colors.

In 1857, Cayley undertook the enumeration of all isomers of the alkane C_nH_{2n+2}. He formulated the problem in terms of graphs: to find all trees with p vertices in which each vertex has a degree of either 1 or 4.

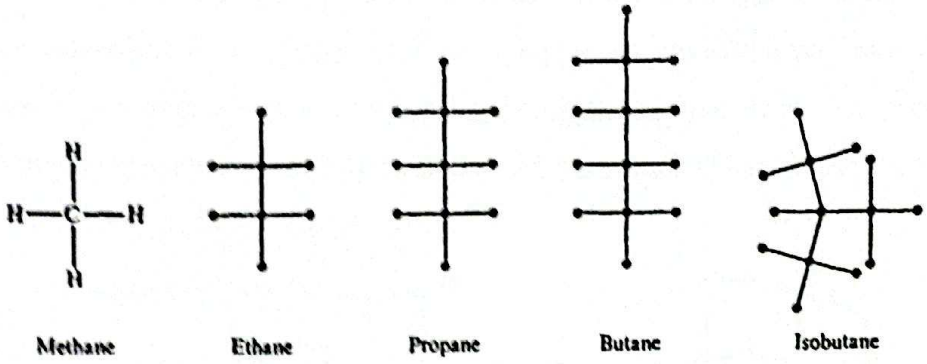


Figure 5. The smallest alkane isomers.

In 1869, Jordan later rediscovered "trees," which he defined as purely mathematical objects.

In 1859, William Hamilton invented a game based on a dodecahedron with 20 points, each representing a city. The player had to find a closed path that passes through each city exactly once, a problem known in graph theory as *the Hamiltonian circuit problem.*

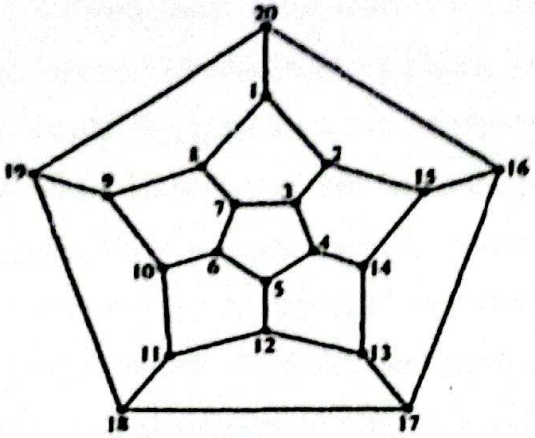


Fig. 6: The Hamiltonian game.

In the 20th century and in the present one, there have already been many rediscoveries of graph theory, of which we can only briefly mention a few in this chronological account.

In 1936, the psychologist Lewin proposed representing an individual's "life space" by means of a "planar" map. In such a map, the regions represent the various activities of a person, such as their work environment, home, and hobbies. This viewpoint led to another psychological interpretation of a graph, in which people are represented by points and interpersonal relationships by lines.

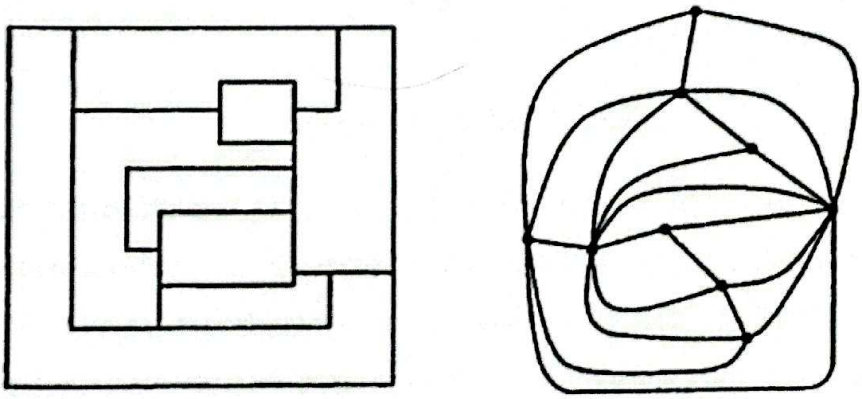


Figure 7. A map and its corresponding graph.

In theoretical physics, and particularly in the study of the statistical mechanics of molecules, Uhlenbeck represented molecules as points, where two "adjacent" points indicate, for example, magnetic attraction or repulsion with the nearest neighbor. In a similar interpretation by Lee and Yang, the points represent small cubes in Euclidean space, where each cube may or may not be occupied by a molecule. Two points are considered "adjacent" whenever both corresponding cubes are occupied. Another aspect of physics uses graph theory primarily as a visual tool. Feynman introduced diagrams in which the points represent physical particles and the lines represent the trajectories of these particles after collisions.

In probability theory, a Markov chain is represented by a *directed graph*, in which the states of the chain are depicted as points, and a directed line from one point to another indicates the transition probability from the state represented by the starting point to the next state represented by the ending point.

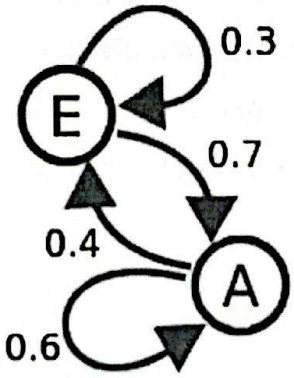


Figure 8. Two-state Markov chain.

A similar representation of a directed graph appears in numerical analysis. For example, consider a sparse square matrix associated with a directed graph as follows: the vertices represent the indices of the rows and columns of the given matrix, and there is a directed edge from vertex ***i*** to vertex ***j*** whenever the matrix element with indices ***(i, j)*** is nonzero. The similarity between this approach and that of Markov chains is immediately apparent.

However, graph theory truly began to develop during the Second World War, more precisely in England around 1940, under the name "*Operations Research.*" The Allied General Staff, aiming to increase the efficiency of its operations, entrusted this task to the physicist Patrick Blackett. The objective was to determine the best rotation schedules for aircraft crews, the optimal placement of radar stations, and later, the organization of transatlantic convoys.

Thus, it was at the end of the 20th century and the beginning of the 21st century that graph theory experienced its true development, driven by the growing use of networks in everyday life, networks whose operation must be constantly optimized. These include road transportation networks, water and electricity distribution systems, data transmission networks (landline, GSM, Wi-Fi, etc.), and information networks (databases, the web, social networks, and others). All of these rely on a theoretical approach based on graph theory.

In such representations, the vertices (points) of a graph correspond to physical locations where certain goods can be stored or shipped, while a directed edge between two points, labeled with a positive number, represents a channel for the transmission of goods and a capacity indicating the maximum possible quantity that can be transported in a single trip.

Recent research in graph theory is often carried out by computer scientists, due to the growing importance of its algorithmic aspects. The terminology of graph theory, along with some basic concepts and fundamental theorems, will be introduced in **Chapter 1**.

Module Content

Chapter IV. Pathway problems

Chapter V. Hamiltonian and Eulerian Problems

Chapter VI. Colouring

Chapter I: Basic definitions

As noted in the Introduction, **graph theory** is a branch of mathematics that deals with problems using specific types of diagrams called graphs. Graph theory has many applications across a wide range of fields, including operations research, physics, chemistry, computer science, and other scientific disciplines. In this chapter, we introduce some fundamental concepts of graph theory and provide a variety of examples. We also present several elementary results.

1. "Intuitive" definition of a graph:

Intuitively speaking, a graph is a diagram consisting of a finite number of points, called *vertices*, and a finite number of arrows, called *arcs*, that connect certain pairs of these points.

Example:

In the graph below:

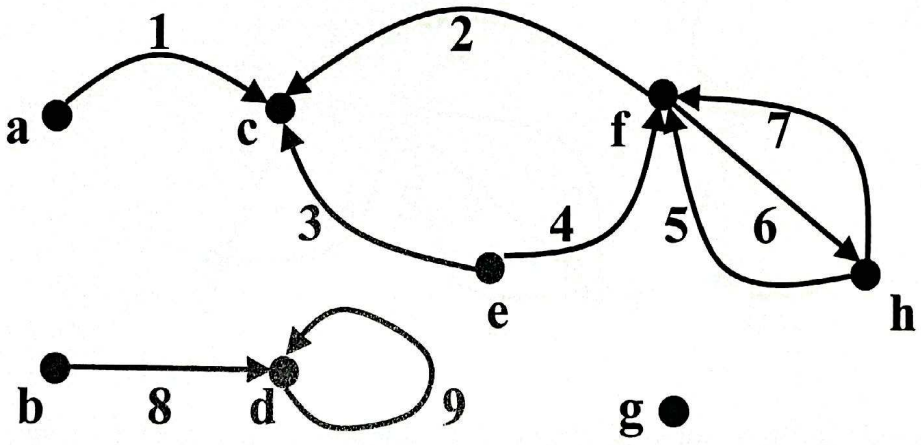


Figure 1. An example of a graph.

- **X = { a, b, c, d, e, f, g, h}** is the set of vertices, and
- **U={1, 2, 3, 4, 5, 6, 7, 8, 9}** is the set of arcs.

2. Mathematical definition of a graph:

Formally, a graph **G** is defined to be a pair **(X , U)**, where **X** is a finite *set* of nodes called *vertices,* and **U** is a set of pairs of vertices **(x , y)** of elements in **X** called *arcs.*

Remark:

The position of the vertices and the shape of the arcs in a graph drawing are not important; what matters is how the vertices are joined by arcs.

The two graphs shown in **Figures 1** and **2** have the same structure. We say that they are *isomorphic*, as defined below:

Two graphs $\mathbf{G_1=(X_1 , U_1)}$ and $\mathbf{G_2=(X_2 , U_2)}$ are said to be *isomorphic* if there exists a bijection **f** from $\mathbf{X_1}$ to $\mathbf{X_2}$ such that, for every pair **(x , y)** of elements in $\mathbf{X_1}$, **(x , y)** is an arc in $\mathbf{U_1}$ if and only, if **(f(x) , f(y))** is an arc in $\mathbf{U_2}$.

For example: **(a , c)** is an arc of $\mathbf{G_1}$, and **(f(a) , f(c))=(b , c)** is an arc of $\mathbf{G_2}$. (b , d) is an arc of $\mathbf{G_1}$, and **(f(a) , f(c))=(g , h)** is an arc of $\mathbf{G_2}$, and so on.

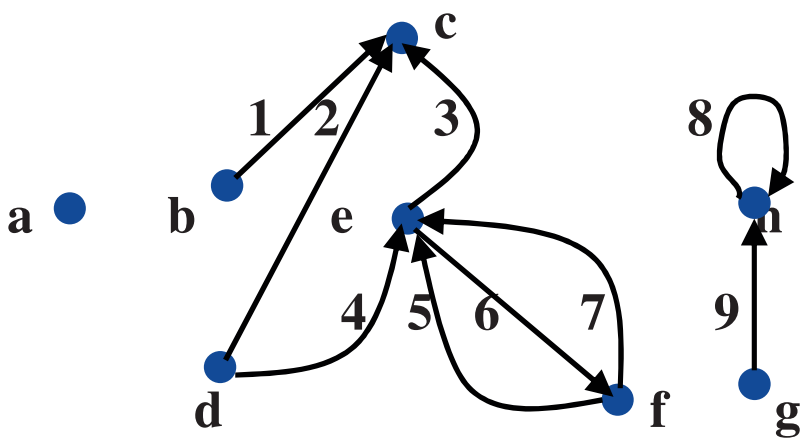


Figure 2. Graph $\mathbf{G_2}$ isomorphic to the graph $\mathbf{G_1}$ in the previous figure.

Notation:

In **Figure 2, a**rc **6** goes from vertex **e** to vertex **f**. It is said to be in the form **(e , f)**, and by convention, we write **6 = (e , f)**. Note that while the form

(e , f) is enough to uniquely identify arc **6**, the form **(f , e)** is not enough to uniquely identify arc **5**, because **7 = (f , e)** as well.

3. Order, orientation and multiplicity:

3.1. Order and Size of a Graph:

The *order* of a graph **G**, denoted by **|G|**, is the number of its vertices, and its *size*, denoted by **‖G‖**, is the number of its arcs.

Example:

The graph in the **Figure 1**, "an example of a graph", has an order of **8** and size of **9**, that is, **|G| = 8** and **‖G‖ = 9.**

3.2. Orientation:

In the graph **G = (X , U)** shown in **Figure 1** or **2**, each arc **u = (x , y)** represents a directed line (an arrow) from vertex **x** to vertex **y**. A graph of this type is called a *directed graph.* If the connection between **x** and **y** is represented by a line without any direction it is called an *edge*. The graph is then called an *undirected graph.*

In what follows, the term *graph* will refer to a *directed graph*, while an *undirected graph* will be referred to as a multigraph.

Example:

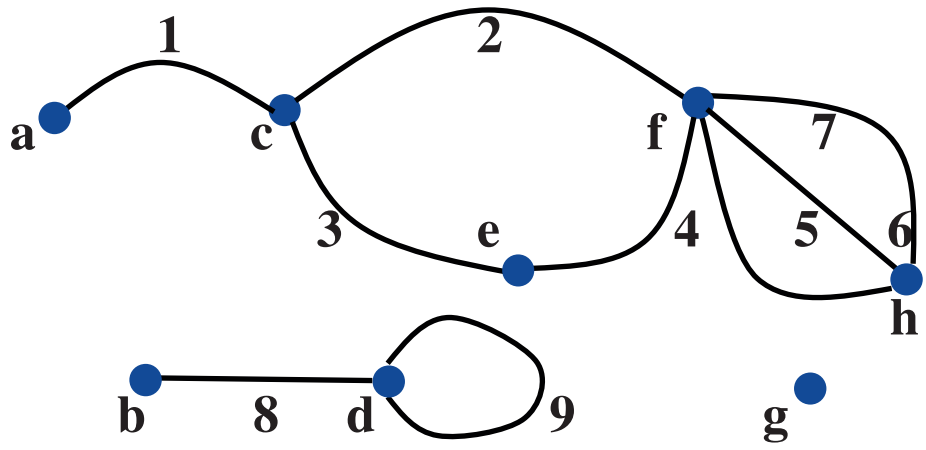


Figure 3. Multigraph associated with the graph in Figure 1.

Notation:

In a multigraph **G = (X , U)**, the notation **{x , y}**, where **x∈X** and **y∈X**, is typically used to represent an *edge*.

For example, in the multigraph shown in **Figure 3**, the edge **1** is denoted by **{a , c}**, that is, **1 = {a , c}**.

Remark:

As noted by C. Berge (1973), it may appear convenient to distinguish between two separate theories: one for *directed* graphs and another for *undirected* graphs. However, this distinction is not strictly necessary. In reality, All graphs are directed, but sometimes the direction need not be specified. To apply a directed concept in a undirected graph we will actually consider the associated directed graph obtained by replacing each edge with two arcs in opposite directions. Conversely, an undirected concept can be applied to a directed graph by ignoring the orientation of the arcs.

3.3. Multiplicity:

Loop and Endpoints:

In a graph, an arc of the form **(x , x)** is called a *loop*. For an arc **(x , y)**, **x** and **y** are called the *endpoints* of the arc. Specifically, vertex **x** is its *initial endpoint,* and vertex **y** is its *terminal endpoint.*

Example:

In the graph below: arc **9 = (d , d)** is a loop. Vertex **b** is the initial endpoint of arc **8** and vertex **d** is its terminal endpoint.

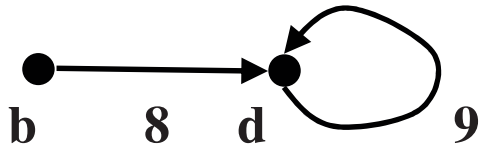


Figure 4. An example of a loop and endpoints.

Multiplicity of a pair x, y:

The multiplicity of a pair **x** , **y** is the number of arcs, having **x** as the initial endpoint and **y** as the final endpoint.

Notation:

The multiplicity of a pair **x** , **y** is denoted as $\mathbf{m_G^+(x,y)}$.

We define $\mathbf{m_G^-(x,y) = m_G^+(y,x)}$ and $\mathbf{m_G(x,y) = m_G^+(x,y) + m_G^-(x,y)}$.

If $\mathbf{x \neq y}$, then $\mathbf{m_G(x,y)}$ denotes the number of arcs with one endpoint at **x** and the other at **y**.

If $\mathbf{x = y}$, then $\mathbf{m_G(x,y)}$ is equal to twice the number of loops attached to vertex **x**.

Example:

In the graph shown in **Figure 1**: $\mathbf{m_G^+(h,f) = 2}$, $\mathbf{m_G^-(c,f) = 1}$, $\mathbf{m_G(d,d) = 2}$.

4. Relations between the elements of a graph:

4.1. Relations between vertices:

Neighboring vertices:

In a graph, vertex **x** is a *successor* of vertex **y** if there is an arc in the form **(y , x)**.

Vertex **x** is a *predecessor* of vertex **y** if there is an arc in the form **(x , y)**.
Vertex **x** is a *neighbor* of vertex **y** if **x** is either a successor or a predecessor of **y**.

The term *adjacent vertices* is also used to refer to neighboring vertices.

Notations:

The set of successors of vertex **x** is denoted as $\Gamma_G^+(\mathbf{x})$.

The set of predecessors of vertex x is denoted as $\Gamma_G^-(\mathbf{x})$.

The set of neighbors of vertex x is denoted as $\Gamma_G(\mathbf{x}) = \Gamma_G^+(\mathbf{x}) \cup \Gamma_G^-(\mathbf{x})$.

Example:

In the graph shown in **Figure 1**:

$\Gamma_G(\mathbf{f}) = \Gamma_G^+(\mathbf{f}) \cup \Gamma_G^-(\mathbf{f}) = \{\mathbf{c},\mathbf{h}\} \cup \{\mathbf{e},\mathbf{h}\} = \{\mathbf{c},\mathbf{e},\mathbf{h}\}$.

Isolated vertex:

It is possible that $\Gamma_G(\mathbf{x}) = \varnothing$ (the empty set). If $\Gamma_G(\mathbf{x}) = \varnothing$, **x** is called an *isolated vertex.*

Example:

In the graph in the **Figure 1**: **g** is an *isolated vertex* since $\Gamma_G(\mathbf{g}) = \varnothing$.

Pendant vertex:

A *pendant vertex* is a vertex that has only one neighbor.

Example:

In the graph shown in **Figure 1**: **b** is a pendent vertex since $\Gamma_G(\mathbf{b}) = \{\mathbf{d}\}$.

Degree of a vertex:

In a multigraph, the *degree* of a vertex is the number of edges that have the vertex as an endpoint. If the vertex is connected to a loop, the loop is counted twice. A vertex has degree **0** if it is an isolate vertex, and degree **1** if it is a pendant vertex.

Notation:

The degree of a vertex **x** in the graph **G** is denoted by $\mathbf{d}_G(\mathbf{x})$.

Example:

In the graph shown in **Figure 3**, an example of a multigraph, the degrees of the vertices are as follows:

X	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**
$\mathbf{d}_G(\mathbf{x})$	**1**	**1**	**3**	**3**	**2**	**5**	**0**	**3**

Vertex **g** is an isolated vertex, while vertices **a** and **b** are both pendant vertices.

In a graph, two types of degrees are associated with each vertex. Let **x** be a vertex in a graph **G**: the number of arcs of the form **(x , y)**, going from **x**, is denoted by $\mathbf{d}_{\mathbf{G}}^{+}(\mathbf{x})$ and is called the *outer demi-degree* of **x**. Similarly, the number of arcs of the form **(y , x)**, going to **x** is denoted by $\mathbf{d}_{\mathbf{G}}^{-}(\mathbf{x})$ and is called the *inner demi-degree* of **x**.

The *total degree* of vertex **x** is given by $\mathbf{d}_{\mathbf{G}}(\mathbf{x}) = \mathbf{d}_{\mathbf{G}}^{+}(\mathbf{x}) + \mathbf{d}_{\mathbf{G}}^{-}(\mathbf{x})$ where each loop is counted twice.

Example:

In the graph shown in **Figure** 1, the degrees of the vertices are as follows:

X	**a**	**b**	**c**	**d**	**E**	**f**	**g**	**h**
$\mathbf{d}_{\mathbf{G}}^{+}(\mathbf{x})$	**1**	**1**	**0**	**1**	**2**	**2**	**0**	**2**
$\mathbf{d}_{\mathbf{G}}^{-}(\mathbf{x})$	**0**	**0**	**3**	**2**	**0**	**3**	**0**	**1**
$\mathbf{d}_{\mathbf{G}}(\mathbf{x})$	**1**	**1**	**3**	**3**	**2**	**5**	**0**	**3**

Similarly, vertex **g** is an isolated vertex, while vertices **a** and **b** are both pendant vertices.

4.2. Relations between arcs and vertices:

Arc incident to a vertex:

In a graph, if a vertex **x** is the initial endpoint of an arc **u**, then **u** is said to be *incident out of* vertex **x**. Conversely, if **x** is the terminal endpoint of an arc **u**, then **u** is said to be *incident into* vertex **x**. In both cases, we say that the arc **u** is *incident to* vertex **x**. We also use the term *incoming arc* for incident inward and *outgoing arc* for incident outward. A *pendant arc* is incident to a *pendant vertex.*

Arc incident to a set of vertices:

In a graph **G = (X , U)**, let $\mathbf{A} \subset \mathbf{X}$. If the initial endpoint of an arc **u** belongs to **A** but its terminal endpoint does not, then **u** is said to be an arc *incident out of the set* **A**. Conversely, if the final endpoint of an arc **u** belongs to **A** but its initial endpoint does not, then **u** is said to be an arc *incident into the set* **A**.

Notation:

If **u** is incident out of **A**, we write: $\mathbf{u} \in \omega_G^+(\mathbf{A})$.

If **u** is incident into **A**, we write: $\mathbf{u} \in \omega_G^-(\mathbf{A})$.

The set of arcs incident to the set **A** is denoted as $\omega_G(\mathbf{A}) = \omega_G^+(\mathbf{A}) \cup \omega_G^-(\mathbf{A})$.

Example:

In the graph shown in **Figure 1**, if **A={c,e,f}** then $\omega_G(\mathbf{A}) = \omega_G^+(\mathbf{A}) \cup \omega_G^-(\mathbf{A}) = \{6\} \cup \{1,5,7\} = \{1,5,6,7\}$.

Adjacent arcs, adjacent edges:

Two arcs (or two edges) are said to be *adjacent* if they have at least one common endpoint.

Example:

In the graph of **Figure 4**, arc **8** and arc **9** are adjacent because **d** is a common vertex.

4.3. Graph qualifiers:

p-Graph:

If, in a graph **G**, the number of arcs from a vertex **x** to another vertex **y** does not exceed a number **p**, the graph is called a **p**-graph, and **p** is referred to as the *multiplicity* of the graph.

Example:

The graph in **Figure 1**, is a **2**-graph.

Simple Graph:

A multigraph is *simple* if it contains no loops and no more than one edge between any pair of distinct vertices.

Example:

The following graph is simple.

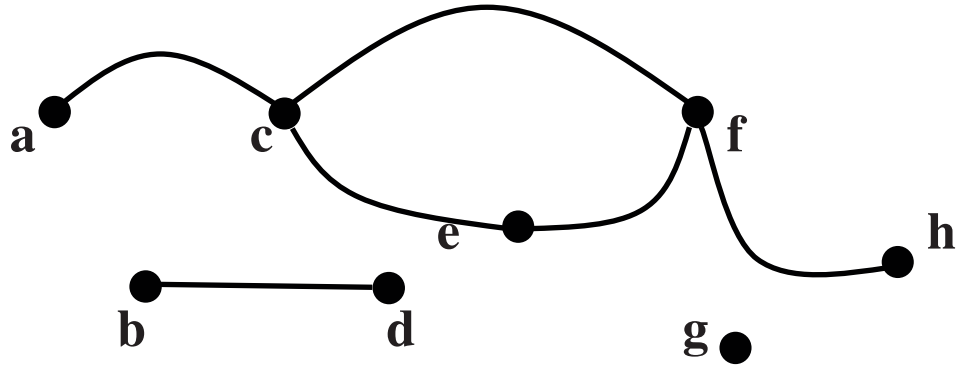


Figure 5. An example of a simple graph.

Regular graph:

If in a graph or in a multigraph each vertex has the same degree, this graph is said to be *regular.*

Example:

The graph in **Figure 1**: is not regular.

Symmetric or antisymmetric graph:

A graph is *symmetric* if for every pair of vertices **x** and **y**, there are as many arcs of the form **(x , y)** as there are arcs of the form **(y , x)**.

A directed graph is *antisymmetric* if for every arc of the form **(x , y)**, there is no other arc of the form **(x , y)** or of the form **(y , x)**.

Example:

Symmetric **Antisymmetric**

Figure 6. Symmetric and antisymmetric graphs.

Complete Graph, Clique:

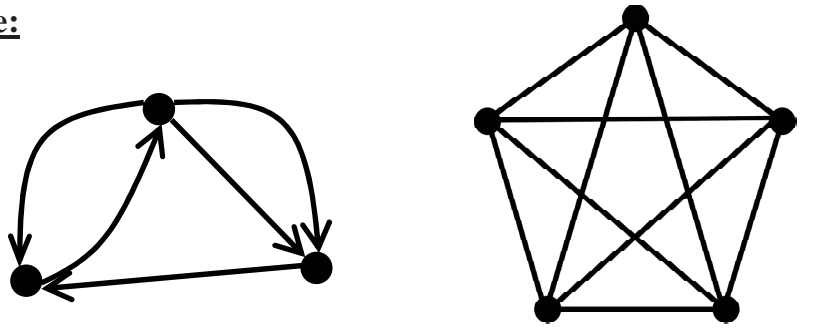
A graph is said to be *complete* if, for every pair of distinct vertices **x** and **y**, there is at least one arc of the form **(x , y)** or **(y , x)**. Similarly, a multigraph is said to be *complete* if every pair of distinct vertices is connected by an edge.

A *simple complete* graph of order **n** is called an **n**-clique.

Notation:

An **n**-clique is denoted as $\mathbf{K_n}$.

Example:



Complete graph **Simple and Complete multigraph**

Figure 7. Complete graph and the **5**-clique $\mathbf{K_5}$.

Bipartite graph, Complete bipartite graph:

A graph or a multigraph is *bipartite* if its vertex set can be partitioned into two classes $\mathbf{X_1}$ and $\mathbf{X_2}$ such that two vertices in the same class are never neighbors.

Notation:

A bipartite graph can be denoted as $\mathbf{G} = (\mathbf{X_1}, \mathbf{X_2}, \mathbf{U})$.

A simple complete bipartite graph with classes $|\mathbf{X}_1| = \mathbf{p}$ and $|\mathbf{X}_2| = \mathbf{q}$ is denoted as $\mathbf{K}_{\mathbf{p},\mathbf{q}}$.

Example:

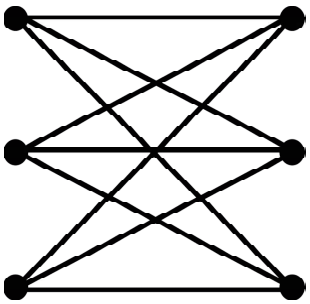


Figure 8. Simple complete bipartite graph $\mathbf{K}_{3,3}$.

Stable set:

In a graph or a multigraph, a set **S** of vertices is called a *stable set* if no arc or edge joins two distinct vertices in **S**.

Example:

In the graph below, the subset {a , d , e} is stable.

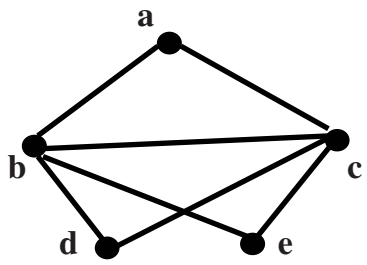


Figure 9: A set stable.

Complement graph:

Given a simple graph or multigraph **G = (X, U)** of order **n**, its *complement* graph **G'=(X , U')** is of the same order **n** and is defined as follows:

An arc (edge) in **U** does not belong to **U'**, and vice versa.

Note that the union $\mathbf{G} \cup \mathbf{G}'$ forms a clique $\mathbf{K}_{\mathbf{n}}$.

Example:

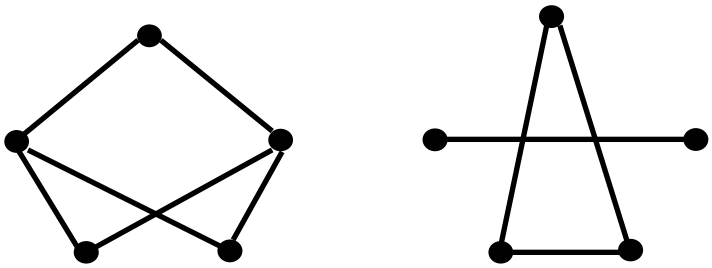


Figure 10: Graph and its complement.

Subgraph of G generated by A⊂X:

Let **G = (X , U)** be a graph or a multigraph, and let **A⊂X**.

The *subgraph* of **G** generated by **A** is the graph $\mathbf{G_A}$, whose vertices are the elements of **A** and whose arcs (edges) are the arcs (edges) of **G** that have both endpoints in **A**.

Partial graph of G generated by V⊂ U:

Let **G = (X , U)** be a graph or a multigraph, and let **V⊂U**.

The *partial graph* generated by **V** is the graph **G' = (X , V)** with the same set **X** of vertices as **G**, and whose arcs (edges) are the arcs (edges) of **V**. Removing from **G** the arcs (edges) in **U\V**.

Partial subgraph of G:

Let **G = (X , U)** be a graph or a multigraph.

A *partial subgraph* of **G** is a subgraph of a partial graph of **G** or a partial graph of a subgraph of **G**.

Example:

The graph in **Figure** 4, where **A={b , d},** is a subgraph of the graph in the **Figure 1**.

The graph in **Figure 11**, where **V={8}**, is a partial graph of the graph in **Figure** 4.

Therefore, the graph in **Figure 11** given below is a partial subgraph of the graph in **Figure 1**.

Figure 11. A partial graph of the graph in Figure 4 and a partial subgraph of the graph in Figure 1.

5. Matrices Associated with a Graph:

5.1. Vertex-arc incidence matrix:

Let **G** be a loopless graph of order **n** and size **m**.

The *vertex-arc incidence* matrix of **G** is a matrix $\mathbf{A}=\left(\mathbf{a}_{ij}\right)$ of size $\mathbf{n}\times\mathbf{m}$, where each row **i** corresponds to the vertex $\mathbf{x}_i$ of **G**, and each column **j** corresponds to the arc $\mathbf{u}_j$ of **G**. The matrix **A** is defined as follows:

- $\mathbf{a}_{ij}=+1$ if vertex $\mathbf{x}_i$ is the initial endpoint of arc $\mathbf{u}_j$,
- $\mathbf{a}_{ij}=-1$ if vertex $\mathbf{x}_i$ is the terminal endpoint of arc $\mathbf{u}_j$,
- $\mathbf{a}_{ij}=0$ in all other cases.

Note that the number of entries equal to +1 (respectively, -1) in row **i** gives the outer demi-degree $\mathbf{d}_G^+\left(\mathbf{x}_i\right)$ (respectively, the inner demi-degree $\mathbf{d}_G^-\left(\mathbf{x}_i\right)$) of the corresponding vertex $\mathbf{x}_i$.

Example:

The vertex-arc incidence matrix of the graph in **Figure 1**, with the arc **9** (the loop) removed is the following:

A	1	2	3	4	5	6	7	8
a	+1	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	+1
c	-1	-1	-1	0	0	0	0	0
d	0	0	0	0	0	0	0	-1
e	0	0	+1	+1	0	0	0	0
f	0	+1	0	-1	-1	+1	-1	0
g	0	0	0	0	0	0	0	0
h	0	0	0	0	+1	-1	+1	0

We can observe, for example, that: $\mathbf{d_G^+(f)=2}$ and $\mathbf{d_G^-(f)=3}$.

5.2. Adjacency matrix or vertex-vertex incidence matrix:

The *adjacency matrix* (also called the *vertex-vertex incidence matrix*) of a graph **G** of order **n** is a square matrix $\mathbf{A=(a_{ij})}$ of size $\mathbf{n\times n}$, where each row and each column corresponds to a vertex of **G**. The element $\mathbf{a_{ij}}$ represents the number of arcs from vertex $\mathbf{x_i}$ to vertex $\mathbf{x_j}$.

Unlike the vertex-arcs incidence matrix, loops can be represented using this matrix.

In this matrix, the sum of the entries in row **i** gives the outer demi-degree $\mathbf{d_G^+(x_i)}$ of the corresponding vertex $\mathbf{x_i}$, while the sum of the entries in column **j** gives the inner demi-degree $\mathbf{d_G^-(x_j)}$ of the corresponding vertex $\mathbf{x_j}$.

Example:

The adjacency matrix associated with the graph shown in **Figure** 1 is the following:

A	a	b	c	d	e	f	g	h
a	0	0	1	0	0	0	0	0
b	0	0	0	1	0	0	0	0
c	0	0	0	0	0	0	0	0
d	0	0	0	1	0	0	0	0
e	0	0	1	0	0	1	0	0
f	0	0	1	0	0	0	0	1
g	0	0	0	0	0	0	0	0
h	0	0	0	0	0	2	0	0

We can observe, for example, that: $\mathbf{d}_{\mathbf{G}}^{+}(\mathbf{d}) = 1$ and $\mathbf{d}_{\mathbf{G}}^{-}(\mathbf{d}) = 2$.

5.3. Condensed form of sparse matrices:

It is immediately noticeable that both incidence matrices are *sparse matrices*; many terms are zero. Therefore, it is possible to write each of the matrices in *compressed form*, that is, to "locate" the non-zero terms by their position in the matrix or by using the IFV matrix (Initial Vertex Final Vertex) if the arcs are numbered.

Example:

The compressed form of the vertex-vertex incidence matrix of the graph in **Figure 1** is as follows:

x	y	$\mathbf{m}_{\mathbf{G}}^{+}(\mathbf{x}, \mathrm{y})$
a	**c**	**1**
b	**d**	**1**
d	**d**	**1**
e	**c**	**1**
e	**f**	**1**
f	**c**	**1**
f	**h**	**1**
h	**f**	**2**

Since the arcs in **Figure 1** are numbered from **1** to **9**, we can write the **IFV** matrix:

Arc	IV	TV
1	a	c
2	f	c
3	e	c
4	e	f
5	h	f
6	f	h
7	h	f
8	b	d
9	d	d

6. Vocabulary related to connected graph

6.1. Chain, path, length:

A *chain* is a sequence $\mu = (\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_q)$ of adjacent arcs, i.e., each arc in the sequence has one endpoint in common with its predecessor and the other endpoint in common with its successor. The number **q > 0** of arcs in the sequence is the *length* of the chain.

In 1-graph, a chain is completely determined by the sequence of vertices it passes through. If the chain passes through vertices $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{q+1}$, we may write it as: $\mu = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{q+1})$. Here, vertex $\mathbf{x}_1$ is called the *initial endpoint* and vertex $\mathbf{x}_{q+1}$ is the *terminal endpoint* of chain μ.

A chain that does not visit any vertex more than once is called *elementary*.

A chain that does not use the same edge twice is called *simple.*

A *path* of *length* **q > 0** is a chain $\mu = (\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_q)$ in which the terminal vertex of arc $\mathbf{u}_i$ is the initial vertex of arc $\mathbf{u}_{i+1}$ for all **i < q.**

Example:

In the graph below:

The sequence **(1,4,5,6)** is a simple and elementary chain of length **3**.

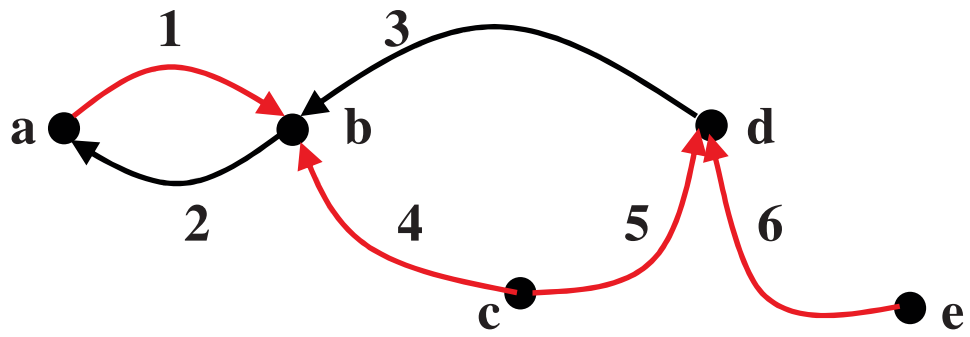


Figure 12. An example of a chain.

In the following graph: the sequence **(5,3,2,1)** is a simple path of length **3**, but it is not elementary, since it may revisit the vertex **b**.

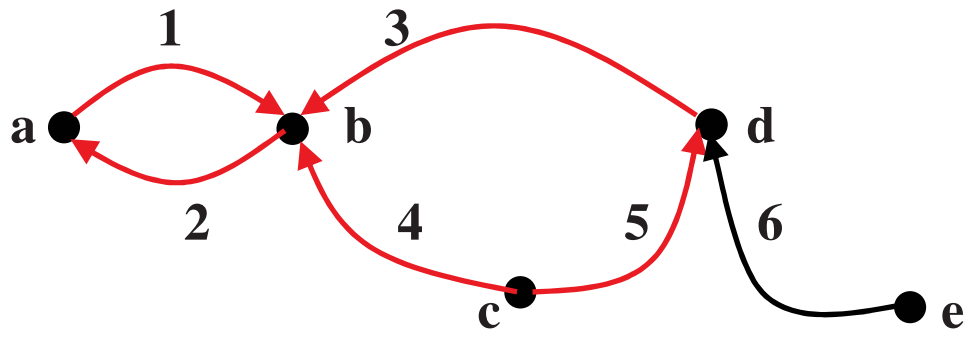


Figure 13. An example of a path.

6.2. Connected graph:

A graph is said to be *connected* if, for every pair of distinct vertices **x** and **y**, there exists a chain that links **x** to **y**.

A *connected component* of a graph is a maximal subset of vertices such that every pair of distinct vertices in the subset is connected by a chain, and no vertex outside the subset is connected by a chain to any vertex within it.

A graph is said to be *strongly connected* if, for every pair of distinct vertices **x** and **y**, there exists a path from **x** to **y** and another path from **y** to **x**.

A *strongly connected component* of a graph is a maximal subset of vertices in which every pair of distinct vertices **x** and **y** of the subset satisfies the condition that there is a path from **x** to **y** and a path from **y** to **x**. Moreover, for any vertex **z** outside this subset, there is no path from **z** to any vertex **x** in the subset, or no path from **x** to **z**.

Example:

The graph below is not connected and has three connected components: {**a**, **c**, **e**, **f**, **h**}, {**b**, **d**} and {**g**}.

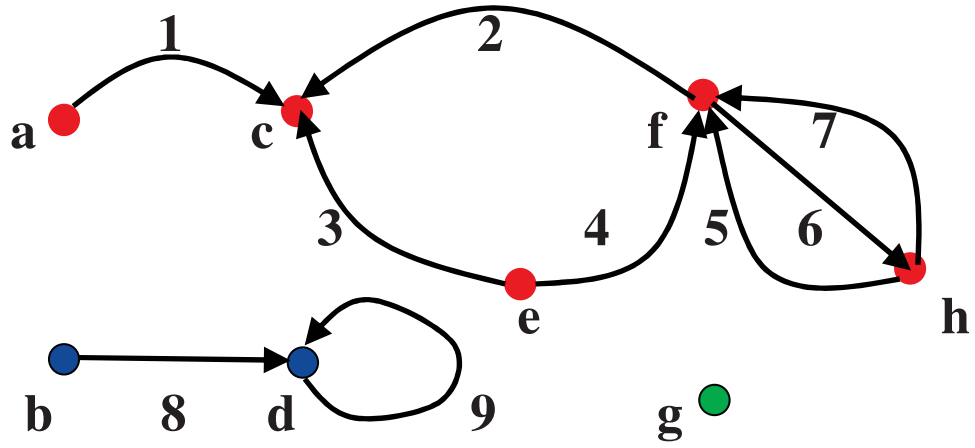


Figure 14. Connected components.

The graph below is strongly connected.

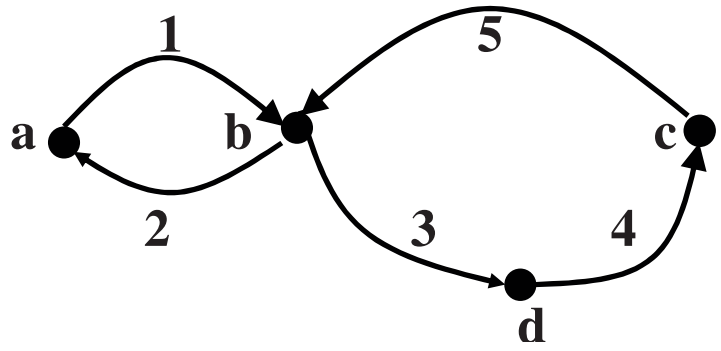


Figure 15. Strongly connected graph.

6.3. Cycle and circuit:

A *cycle* is a simple chain whose two endpoints coincide. A loop is a cycle of length 1.

A *circuit* is a cycle $\mu = (u_1, u_2, ..., u_q)$ such that for all **i < q** the terminal vertex of arc u_i is the initial vertex of arc u_{i+1}.

Notation:

We denote by μ^+ the set of arcs of the cycle oriented in the traversal direction and by μ^- the set of the other arcs of the cycle.

If the arcs are numbered **1**, **2**, **…**, **m**, every cycle can be associated with a vector $\vec{\mu} = (\mu_1, \mu_2, ..., \mu_m)$ in $\mathbb{R}^m$, with:

$$\mu_i = \begin{cases} +1 \text{ if } i \in \mu^+ \\ -1 \text{ if } i \in \mu^- \\ 0 \text{ if } i \notin \mu^+ \cup \mu^- \end{cases}.$$

Example:

In the graph below: **(3,4,5)** is a cycle of length **3**. The associated vector is $\vec{\mu} = (0,0,1,-1,1,0)$.

(1 , 2) is a circuit of length **2**. The associated vector is $\vec{\mu} = (1,1,0,0,0,0)$.

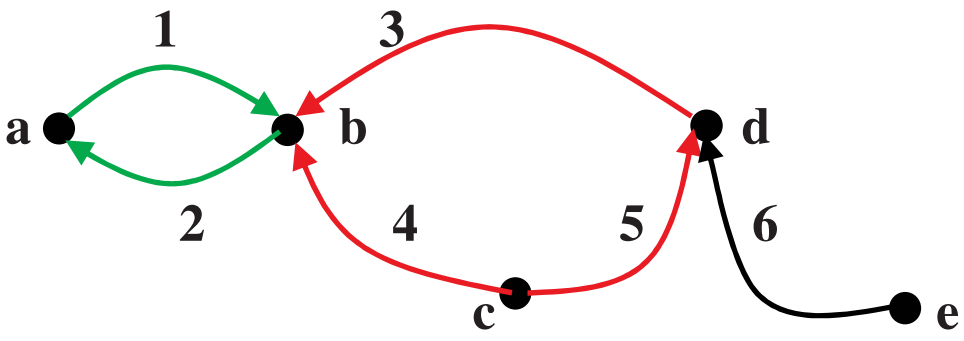


Figure 16. An example of a cycle and a circuit.

6.4. Cocycle and Cocircuit:

Let **G = (X , U)** a graph and $\mathbf{A} \subset \mathbf{X}$. The set of arcs incident to the set **A**, be denoted as $\omega_G(\mathbf{A}) = \omega_G^+(\mathbf{A}) \cup \omega_G^-(\mathbf{A})$ (see § 4.2, incident arc).

A *cocycle* is a non-empty set of arcs of the form $\omega_G(\mathbf{A})$, partitioned into two classes $\omega_G^+(\mathbf{A})$ and $\omega_G^-(\mathbf{A})$.

A **cocircuit** is a cocycle, either $\omega_G(\mathbf{A}) = \omega_G^+(\mathbf{A})$ or $\omega_G(\mathbf{A}) = \omega_G^-(\mathbf{A})$, in which all arcs are oriented in the same direction, i.e. into set **A,** or out of set **A.**

Notation:

If the arcs are numbered **1**, **2**, **…**, **m**, every cocycle can be associated with a vector $\vec{\omega} = (\omega_1, \omega_2, ..., \omega_m)$ in $\mathbb{R}^m$, with:

$$\omega_i = \begin{cases} +1 \text{ if } i \in \omega_G^+(\mathbf{A}) \\ -1 \text{ if } i \in \omega_G^-(\mathbf{A}) \\ 0 \text{ if } i \notin \omega_G(\mathbf{A}) \end{cases}$$

Example:

In the following graph $:\{2\} \cup \{1,6\}$ is a cocycle with $\mathbf{A} = \{\mathbf{b}, \mathbf{c}, \mathbf{d}\}$. The associated vector is $\vec{\omega} = (-1,1,0,0,0,-1)$.

$\{6\}$ is a cocircuit with $\mathbf{A} = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$. The associated vector is $\vec{\omega} = (0,0,0,0,0,-1)$.

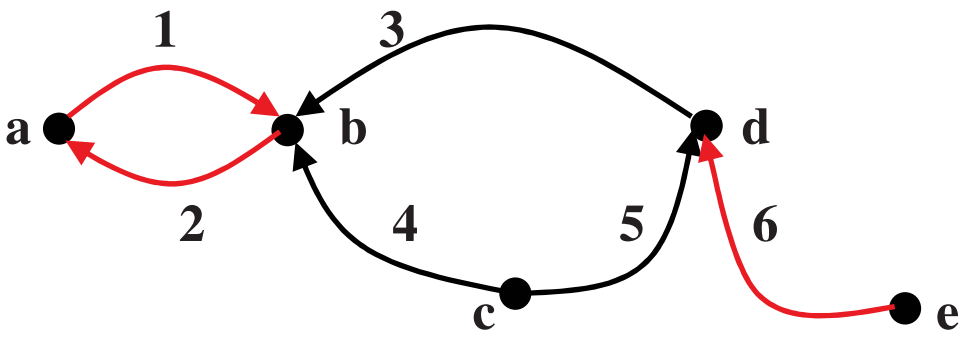


Figure 17. An example of a cycle and a circuit

An *elementary cocycle* is composed of the set of arcs joining two connected subgraphs $\mathbf{A}_1$ and $\mathbf{A}_2$, such that:

$$\begin{cases} \mathbf{A}_1 \neq \varnothing,\ \mathbf{A}_2 \neq \varnothing \\ \mathbf{A}_1 \cap \mathbf{A}_2 = \varnothing \\ \mathbf{A}_1 \cup \mathbf{A}_2 = \mathbf{C} \end{cases}.$$

where **C** is a connected component.

Example:

In the graph below: **{3,5}** is an elementary cocycle with $\mathbf{A}_1 = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ and $\mathbf{A}_2 = \{\mathbf{d}, \mathbf{e}\}$.

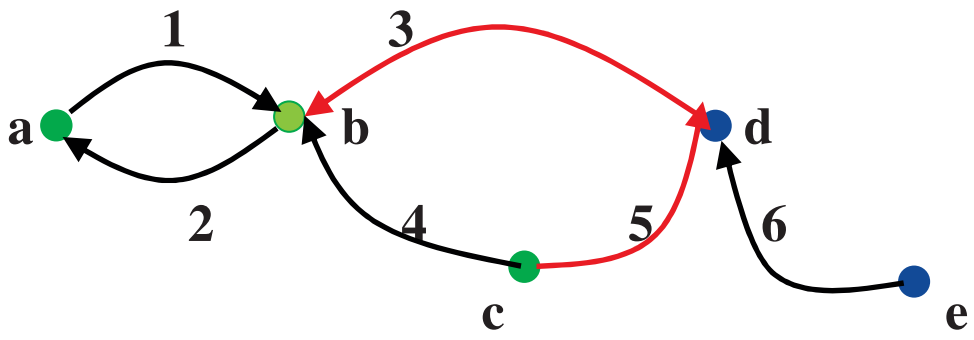
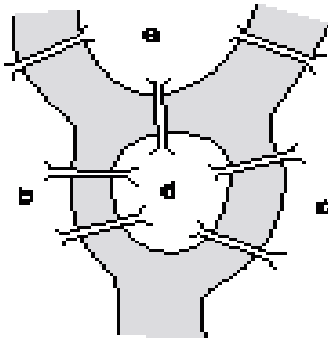


Figure 18. An example of an elementary cocycle.

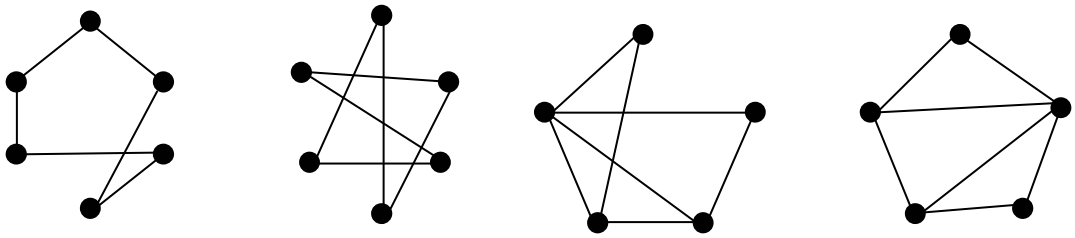
Tutorial Session (1)

Exercise 1: The Königsberg Bridge Problem

In the 18th century, the city of Königsberg consisted of two islands and seven bridges, as shown in the plan below. The residents wanted to take a walk crossing each bridge exactly once. Draw the graph associated with the city's layout.



Exercise 2: Determine if the following drawings represent the same graph.



G1 G2 G3 G4

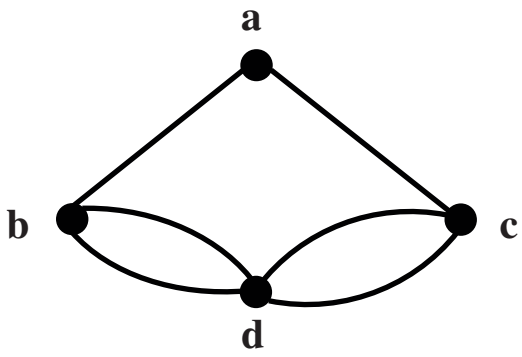
Exercise 3: Draw the following graphs:

1. The vertices are the faces of a cube. Two vertices are connected if the corresponding faces share an edge of the cube.
2. The vertices of the graph are all the two-element subsets of {1, 2, 3, 4}, and two vertices are connected if their intersection is non-empty.
3. Graph associated with the situation: Three countries each send two spies to a conference. The spies do not know each other, and each spy must make contact with all the spies from the other countries.

Tutorial Session 1 (Solution)

Exercise 1

Let's translate the Königsberg Bridge problem using a graph. Each region of the city is represented by a vertex, and a bridge between two regions is represented by an edge between the corresponding vertices. The graph associated with the city plan is the following graph:



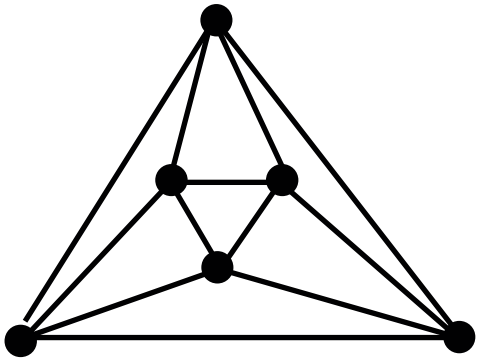
Exercise 2

Graphs **G1** and **G2** both represent the same graph, which is a cyclic graph (a cycle) of order **6**.

Graphs **G3** and **G4** both represent the same graph, in which one vertex has degree **4**, two vertices each have degree **3**, and the remaining two vertices each have degree **2**.

Exercise 3

All three situations correspond to the graph of the octahedron:



Tutorial Session (2)

Exercise 1:

In a graph $G = (X, U)$, the following four properties are always satisfied:

1. $\sum_{x \in X} d_G^-(x) = \sum_{x \in X} d_G^+(x)$.
2. $\sum_{x \in X} d_G(x) = 2|U|$.
3. $\sum_{x \in X} d_G(x)$ is an even number.
4. There is an even number of vertices with odd degree.

Exercise 2:

A football league contains 7 clubs. Due to time constraints, it is decided that each club will only play half of the possible matches. How can the tournament be organized?

Exercise 3:

How can you draw 5 segments on a sheet of paper so that each segment intersects exactly 3 others?

Exercise 4:

Is it possible to construct a simple graph $G = (X, U)$ with 4 vertices and 7 edges?

Exercise 5:

In a group of twenty children, is it possible that seven of them each have exactly three friends, nine of them have exactly four friends, and four of them have exactly five friends?

Tutorial Session 2 (Solution)

Exercise 1:

1. Each arc of the graph gives the in-degree once and the out-degree once. Therefore, $\sum_{x \in X} d_G^-(x) = \sum_{x \in X} d_G^+(x)$.

2. When summing the degrees of the vertices, an arc is counted twice, once for each endpoint. Therefore, $\sum_{x \in X} d_G(x) = 2|U|$.

3. This follows immediately from (2).

4. We have: $\sum_{x \in X} d_G(x) = \sum_{\substack{x \in X \\ d_G(x) \bmod 2 = 0}} d_G(x) + \sum_{\substack{x \in X \\ d_G(x) \bmod 2 = 1}} d_G(x)$.

Since the first sum is even, the second sum must also be even.

Since each term of the second sum is odd, this sum must contain an even number of terms.

Exercise 2:

The essential idea is to represent the problem with a graph **G = (X, U)**: each club is represented by a vertex, and an edge connects two clubs that play a match against each other. After testing all possible scenarios, we conclude that it is not possible for all teams to play the same number of matches. But what happens if the size of the data is large? With 15 clubs, for example, it is quite difficult to start working on this problem. A good idea is to use the mathematical properties of graphs that we have studied:

If the problem has a solution, then we have: $\forall x \in X : d_G(x) = 3,$ since each team must play 3 matches. If we count the total number of participations, we get: $\sum_{x \in X} d_G(x) = 3 \times 7 = 21,$ which is absurd (the sum of the degrees must be an even number). Organizing such a tournament is not possible. More generally, we see

that, for the same reason, if there is an odd number of teams, it is not possible for them all to play an odd number of matches.

Exercise 3:

If we think of representing each segment by a vertex, and connecting two vertices if the corresponding segments intersect, we see, exactly as in the previous exercise, that the proposed exercise is impossible.

Exercise 4:

The graph G is simple, so each vertex in the graph is adjacent to at most 3 other vertices. In other words, $\forall \mathbf{x} \in \mathbf{X} : \mathbf{d_G}(\mathbf{x}) \leq 3$. The sum of the degrees of the graph satisfies the inequality $\sum_{\mathbf{x} \in \mathbf{X}} \mathbf{d_G}(\mathbf{x}) = 3 \times 4 = 12 = 2 \times 6.$ Consequently, $\|\mathbf{G}\| \leq 6$, this is absurd. Therefore, it is not possible to construct a simple graph with 4 vertices and 7 edges.

Exercise 5:

The situation is impossible, at least if we assume that friendship is reciprocal. Indeed, by drawing the friendship graph, we see that the assumption implies there are 11 vertices of odd degree, which cannot happen.

Tutorial Session (3)

Exercise 1:

Let $G = (X, U)$ be a graph. Let $\delta(G)$ and $\Delta(G)$ represent the minimum and maximum degrees of its vertices, respectively. Show that:

$$\delta(G) \leq \frac{2|U|}{|X|} \leq \Delta(G).$$

Exercise 2:

Let $G = (X, U)$ be an r-regular graph. Show that:

$$|U| = \frac{r|X|}{2}.$$

Exercise 3:

Draw the clique $\boldsymbol{K_n}$ for $n = 2,3,4,5$. In general, how many edges does K_n have?

Exercise 4:

Draw all simple graphs of orders **3, 4, 5, 6** and **7** in which all vertices have degree 2. Deduce the number of possible different simple graphs of order n that we can draw.

Exercise 5:

Let $\boldsymbol{G}$ be a simple graph of order $\boldsymbol{2p}$, where each vertex has a degree of at least $\boldsymbol{p}$. Show that this graph is connected.

Exercise 6:

Let $\boldsymbol{G = (X, U)}$ be a simple bipartite graph. Show that:

$$\boldsymbol{|U| \leq \frac{|X|^2}{4}}.$$

Deduce that there exists a vertex x in G such that:

$$\boldsymbol{d_G(x) \leq \frac{|X|}{2}}.$$

Exercise 7:

Find the smallest number of vertices required to construct a simple graph with at least 1000 edges.

Tutorial Session 3 (Solution)

Exercise 1:

We know that: $\forall \mathbf{x} \in \mathbf{X}: \ \delta(\mathbf{G}) \le \mathbf{d}_{\mathbf{G}}(\mathbf{x}) \le \Delta(\mathbf{G})$.

By summing these inequalities over all vertices **x**, we obtain: $\sum_{\mathbf{x}\in\mathbf{X}} \delta(\mathbf{G}) \le \sum_{\mathbf{x}\in\mathbf{X}} \mathbf{d}_{\mathbf{G}}(\mathbf{x}) \le \sum_{\mathbf{x}\in\mathbf{X}} \Delta(\mathbf{G})$.

Since $\delta(\mathbf{G})$ and $\Delta(\mathbf{G})$ are independent of **x**, and $\sum_{\mathbf{x}\in\mathbf{X}} \mathbf{d}_{\mathbf{G}}(\mathbf{x}) = 2|\mathbf{U}|$, we get:

$\delta(\mathbf{G}) \times |\mathbf{X}| \le 2|\mathbf{U}| \le \Delta(\mathbf{G}) \times |\mathbf{X}|$.

Dividing by $|\mathbf{X}|$, we finally obtain: $\delta(\mathbf{G}) \le \frac{2|\mathbf{U}|}{|\mathbf{X}|} \le \Delta(\mathbf{G})$.

Exercise 2:

We have $\sum_{\mathbf{x}\in\mathbf{X}} \mathbf{d}_{\mathbf{G}}(\mathbf{x}) = 2|\mathbf{U}|$. Moreover, **G** is **r**-regular, that is: $\forall \mathbf{x} \in \mathbf{X}: \mathbf{d}_{\mathbf{G}}(\mathbf{x}) = \mathbf{r}$.

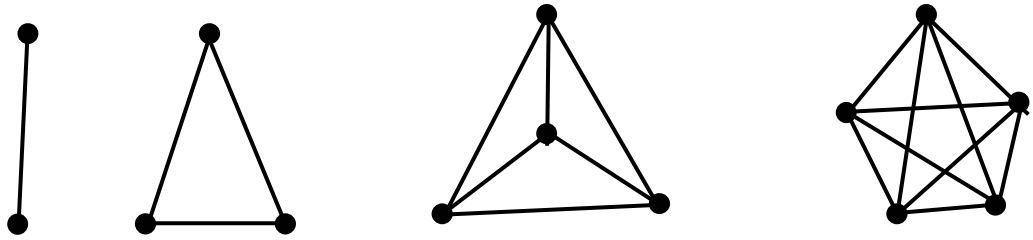
By substitution, we obtain: $\sum_{\mathbf{x}\in\mathbf{X}} \mathbf{r} = 2|\mathbf{U}|$.

Hence, $\mathbf{r}|\mathbf{X}| = 2|\mathbf{U}|$.

Therefore, $|\mathbf{U}| = \frac{\mathbf{r}|\mathbf{X}|}{2}$.

Exercise 3:

The simple and complete graphs are shown below. Since the graph $\mathbf{K_n}$ contains all possible edges, it has as many edges as there are ways to choose 2 elements from **n**, that is: $\mathbf{C}_{\mathbf{n}}^{2} = \frac{\mathbf{n}(\mathbf{n}-1)}{2}$.



Exercise 4:

A little reflection shows that if a graph has all its vertices of degree 2, then, starting from any vertex, one returns to it in a uniquely determined way after traversing a polygon. One might think that this completely determines the graph as soon as its order is known, but this is not the case. Indeed, it is not specified that the graph is connected; therefore, a graph of order 6 may consist of a hexagon or of two triangles.

However, in a simple graph where every vertex has degree 2, each connected component contains at least three vertices (the given vertex and the two distinct vertices adjacent to it). Hence, for orders 3, 4, and 5, there can be only one connected component, and the graph is uniquely determined. Starting from order 6, there are as many possible graphs as there are ways to write the order as a sum of integers greater than or equal to 3.

Exercise 5:

Consider a graph of order **2p** in which every vertex has a degree greater than or equal to **p**. Suppose the graph is not connected. Then there exist two distinct vertices **a** and **b** such that no chain connects them. In particular, they are not adjacent. There are at least **p** vertices adjacent to **a**, and **p** vertices adjacent to **b**. These vertices must all be distinct; otherwise, there would exist a chain of length 2 joining **a** and **b**. They are also distinct from **a** and **b** themselves. But then there would be **p + p + 2 = 2p + 2** distinct vertices, which is impossible.

Exercise 6:

In addition to being simple and bipartite, suppose that the graph is also complete. If we can prove that the inequality $|\mathbf{U}| \leq \frac{|\mathbf{X}|^2}{4}$ holds for a complete graph, then it will also hold for all other graphs. This is because a complete graph has the maximum possible number of edges. In bipartite graph the vertex set is divided into two disjoint classes X_1 and X_2 with $|\mathbf{X}_1| = \mathbf{p}$ and $|\mathbf{X}_2| = \mathbf{q}$. Because it is simple

and complete, then $|\mathbf{U}| = \mathbf{pq}$. Consequently, the inequality $|\mathbf{U}| \le \frac{|\mathbf{X}|^2}{4}$ is equivalent to $\mathbf{pq} \le \frac{(\mathbf{p}+\mathbf{q})^2}{4}$ which can be rewritten as $(\mathbf{p}-\mathbf{q})^2 \ge 0$. Since $(\mathbf{p}-\mathbf{q})^2 \ge 0$ is true, the inequality $|\mathbf{U}| \le \frac{|\mathbf{X}|^2}{4}$ also holds.

From Exercise 1, we have already shown that $\delta(\mathbf{G}) \le \frac{2|\mathbf{U}|}{|\mathbf{X}|}$. If we choose **x** as a vertex with minimum degree in the graph, then $\mathbf{d}_{\mathbf{G}}(\mathbf{x}) = \delta(\mathbf{G}) \le \frac{2|\mathbf{U}|}{|\mathbf{X}|}$. Using the result from the first question of this exercise, we obtain $\mathbf{d}_{\mathbf{G}}(\mathbf{x}) = \delta(\mathbf{G}) \le \frac{2|\mathbf{U}|}{|\mathbf{X}|} \le \frac{2\frac{|\mathbf{X}|^2}{4}}{|\mathbf{X}|} = \frac{|\mathbf{X}|}{2}$.

Exercise 7:

In a simple graph with **n** vertices, the maximum number of edges is when the graph is complete, i.e., a clique. In this case number of edges is $\frac{\mathbf{n}(\mathbf{n}-1)}{2}$. We want this to be at least 1000**.** So we must have $\frac{\mathbf{n}(\mathbf{n}-1)}{2} \ge 1000$. We can test integer values:

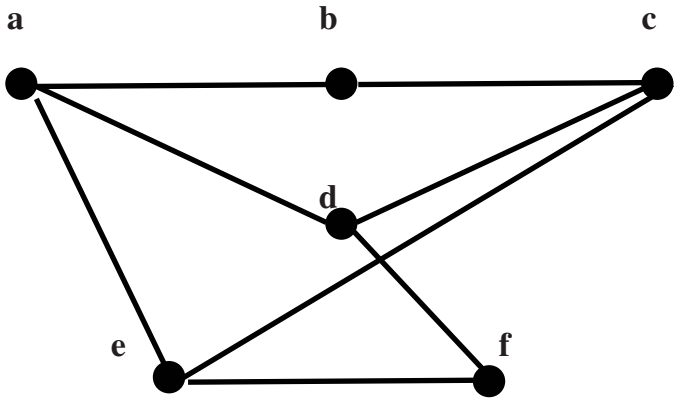
- For **n = 45**, we have only 990 edges (< 1000).
- For **n = 46**, we have **1035 edges** (≥ 1000).

We conclude that the smallest number of vertices required is **n = 46**.

Tutorial Session (4)

Exercise 1:

Let **G** the following multigraph:



1. Is the graph **G**:

   a) Simple b) Regular c) Symmetric d) Antisymmetric e) Complete

   f) Clique g) Bipartite h) Complete bipartite i) Connected j) Planar?

2. Determine:

   a) Subgraph of **G** generated by **{a, d, c, e}.**

   b) Partial graph of **G** generated by **{ {a, b}, {c,d} , {d,f} }**.

   c) Partial subgraph of **G**.

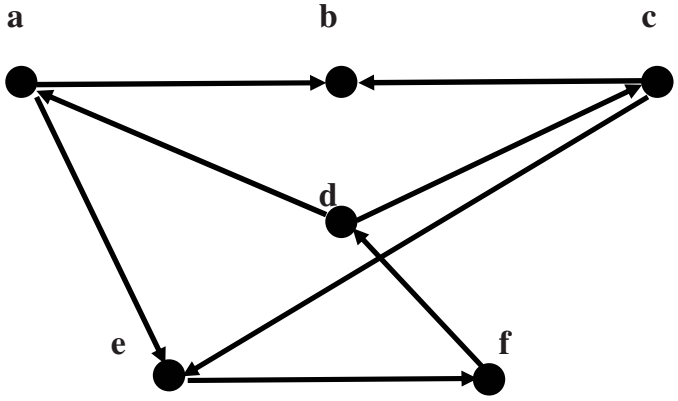
   d) Complement graph of **G** (if it exists).

   e) Dual graph (if it exists).

   f) Stable set of **G**.

Exercise 2:

Let **G** the following graph:

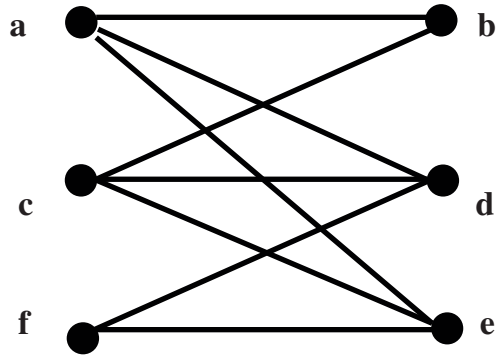


1. Determine:
   - **a)** The order and the size of the graph **G**.
   - **b)** Determine the multiplicity of the graph **G**.
   - **c)** The outer demi-degree, inner demi-degree and the total degree for each vertex of the graph **G**.
   - **d)** $\Gamma_G^+(\mathbf{d})$ the set of successors of vertex **d** , $\Gamma_G^-(\mathbf{d})$ the set of predecessors of vertex **d** and $\Gamma_G(\mathbf{d})$ the set of neighbors of vertex **d**.
   - **e)** $\mathbf{m}_G^+(\mathbf{a},\mathbf{b})$ the multiplicity of $(\mathbf{a},\mathbf{b})$.
   - **f)** $\omega_G(\{\mathbf{a},\mathbf{d},\mathbf{c}\})$ the set of arcs incident to the set **{a,d,c}**.
   - **g)** A path between vertices **f** and **e**. Is it a simple path? Is it an elementary path?
   - **h)** A circuit.
   - **i)** A cocycle.
2. Represent the graph using : a) vertex-vertex incidence matrix b) Vertex-arc incidence matrix.
3. Give the condensed forms of the vertex-vertex incidence matrix.
4. Is **G** a simply connected graph? b) Is it a strongly connected graph?

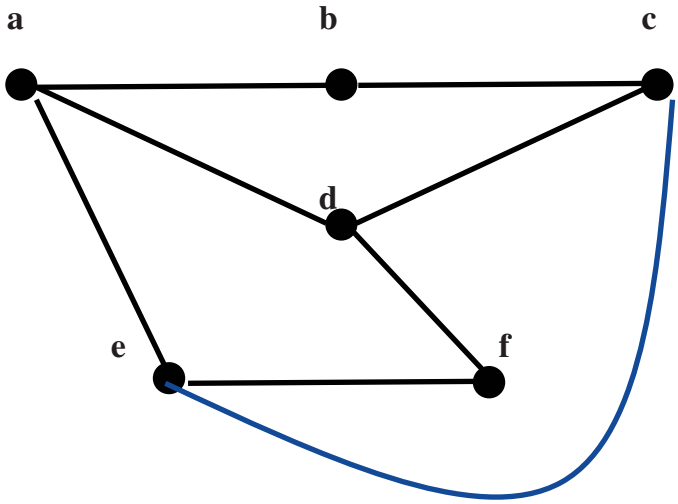
Tutorial Session 4 (Solution)

Exercise 1:

1. a) G is simple because it contains no loops or multiple edges.
b) The graph is not regular because the vertices do not have the same degree.
c) The graph is symmetric because, for each arc **(x,y)** , there is an arc in the form **(y,x)**; each edge **{x,y}** represents two arcs with opposite sense.
d) The graph is not antisymmetric because, if there is an arc **(x,y)**, there is other arc of the form **(y, x).**
e) G is not complete because vertices **b** and d are not adjacent.
f) G is not a clique because it is not complete.
g) G is bipartite because its set of vertices can be divided into two classes **X1={a,c,f}** and **X2={b,d,e}** such that no two vertices in the same class are adjacent.

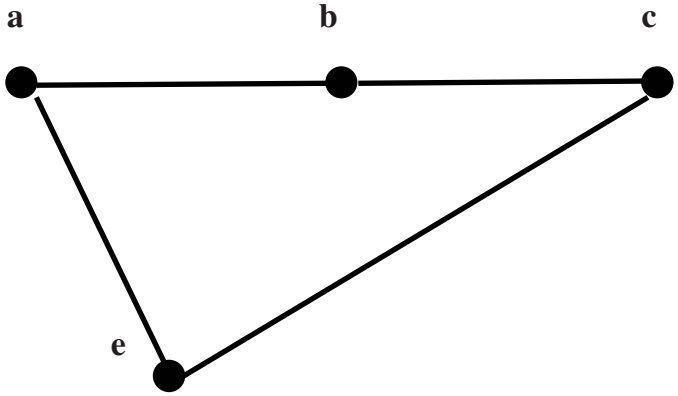


h) G is not complete bipartite because vertex **f** is not adjacent to vertex **b**.
i) G is connected because for every pair of distinct vertices **x** and **y**, there exists a chain that links **x** to **y.**
j) G is planar because it can be drawn on a plane so that no two edges intersect except at their endpoints.

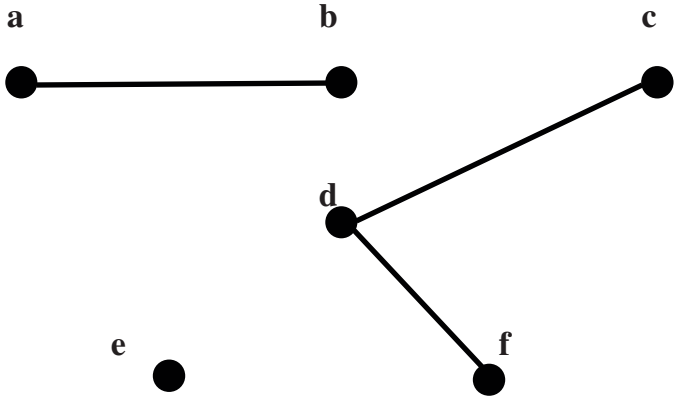


2.

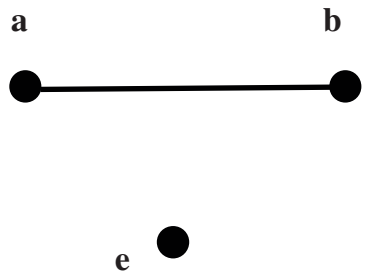
a) Subgraph of **G** generated by **{a, d, c, e}:**



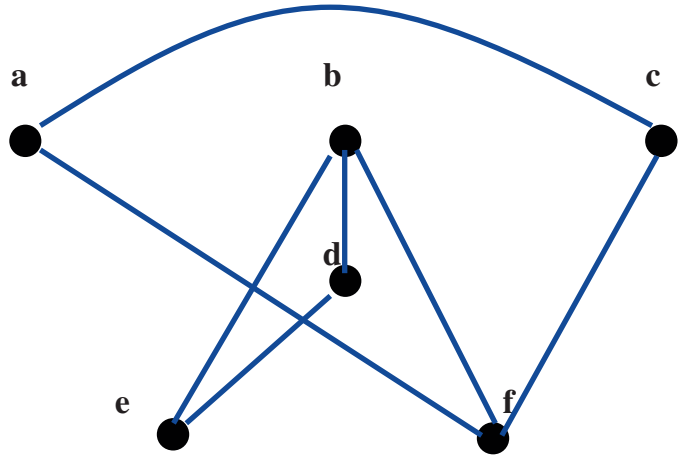
b) Partial graph of **G** generated by **{ {a, b}, {c,d} , {d,f} }**:



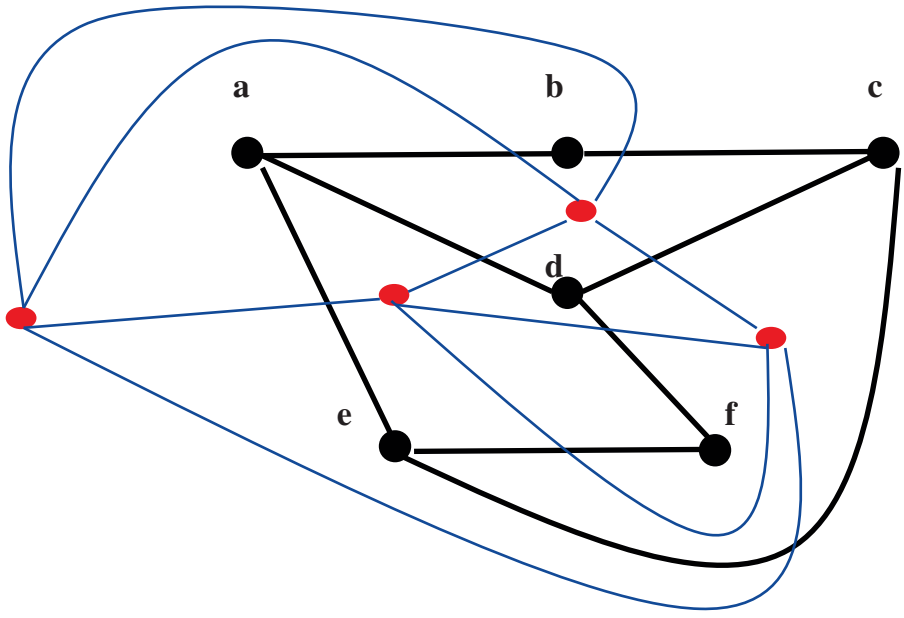
c) Partial subgraph of **G**:



d) Complement graph of **G** (if it exists): **G** is a simple graph; therefore, the complement of **G** exists.



e) Dual graph (if it exists): **G** is planar graph; therefore, the dual of **G** exists.



f) Stable set of **G**: **{a,c,f}** and **{b,d,e}**.

Exercise 2:

1.

a) The order: **| G | = 6**. The size: **||G|| = 8**.

b) The multiplicity: **G** is a **1**-graph .

c) The outer demi-degree, inner demi-degree and the total degree for each vertex of the graph **G**:

Vertex x	**a**	**b**	**c**	**d**	**e**	**f**
$\mathbf{d}_G^+(\mathbf{x})$	2	0	2	2	1	1
$\mathbf{d}_G^-(\mathbf{x})$	1	2	1	1	2	1
$\mathbf{d}_G(\mathbf{x})$	3	2	3	3	3	2

d) The set of successors of vertex **d:** $\Gamma_G^+(\mathbf{d}) = \{\mathbf{a},\mathbf{c}\}$.

The set of predecessors of vertex **d**: $\Gamma_G^-(\mathbf{d}) = \{\mathbf{f}\}$.

The set of neighbors of vertex **d:**

$\Gamma_G(\mathbf{d}) = \Gamma_G^+(\mathbf{d}) \cup \Gamma_G^-(\mathbf{d}) = \{\mathbf{a},\mathbf{c}\} \cup \{\mathbf{f}\} = \{\mathbf{a},\mathbf{c},\mathbf{f}\}$.

e) The multiplicity of $(\mathbf{a},\mathbf{b})$: $\mathbf{m}_G^+(\mathbf{a},\mathbf{b}) = \mathbf{1}$.

f) The set of arcs incident to the set **{a,d,c}**:

$\omega_G(\{\mathbf{a},\mathbf{d},\mathbf{c}\}) = \omega_G^+(\{\mathbf{a},\mathbf{d},\mathbf{c}\}) \cup \omega_G^-(\{\mathbf{a},\mathbf{d},\mathbf{c}\})$

$= \{(\mathbf{a},\mathbf{b}),(\mathbf{a},\mathbf{e}),(\mathbf{c},\mathbf{b}),(\mathbf{c},\mathbf{e})\} \cup \{(\mathbf{f},\mathbf{d})\}$

$\{(\mathbf{a},\mathbf{b}),(\mathbf{a},\mathbf{e}),(\mathbf{c},\mathbf{b}),(\mathbf{c},\mathbf{e}),(\mathbf{f},\mathbf{d})\}$.

g) A path between vertices **f** and **e:** .**(f,d,c,e)**. The path is simple (does not use the same edge twice). Is it also an elementary path (it does not visit any vertex more than once).

h) A circuit: **(a,e,f,d,a)**.

i) A cocycle: $\omega_G(\{\mathbf{a},\mathbf{d},\mathbf{c}\}) = \{(\mathbf{a},\mathbf{b}),(\mathbf{a},\mathbf{e}),(\mathbf{c},\mathbf{b}),(\mathbf{c},\mathbf{e}),(\mathbf{f},\mathbf{d})\}$.

2. **a)** vertex-vertex incidence matrix:

	a	**b**	**c**	**d**	**e**	**F**
a	0	**1**	0	0	**1**	0
b	0	0	0	0	0	0
c	0	**1**	0	0	**1**	0
d	**1**	0	**1**	0	0	0
e	0	0	0	0	0	**1**
f	0	0	0	**1**	0	0

b) Vertex-arc incidence matrix:

	(a,b)	**(a,e)**	**(c,b)**	**(c,e)**	**(d,a)**	**(d,c)**	**(e,f)**	**(f,d)**
a	**1**	**1**	0	0	**-1**	0	0	0
b	**-1**	0	**-1**	0	0	0	0	0
c	0	0	**1**	**1**	0	**-1**	0	0
d	0	0	0	0	**1**	**1**	0	**-1**
e	0	**-1**	0	**-1**	0	0	**1**	0
f	0	0	0	0	0	0	**-1**	**1**

3. The condensed forms of the vertex-vertex incidence matrix:

a) Compressed form of the vertex-vertex incidence matrix:

x	**y**	$\mathbf{m}_{\mathbf{G}}^{+}(\mathbf{x}, \mathrm{y})$
a	**b**	**1**
a	**e**	**1**
c	**b**	**1**
c	**e**	**1**
d	**a**	**1**
d	**c**	**1**
e	**f**	**1**
f	**d**	**1**

b) The IFV matrix:

Arc	IV	TV
1	**a**	**b**
2	**a**	**e**
3	**c**	**b**
4	**c**	**e**
5	**d**	**a**
6	**d**	**c**
7	**e**	**f**
8	**f**	**d**

4. **a) G** is a simply connected graph because, for every pair of distinct vertices **x** and **y**, there exists a chain connecting **x** to **y**.

b) G is not a strongly connected graph because there is a path from **a** to **b**, but no path from **b** to **a**.

Chapter II: Cycles

In this chapter, we explore important concepts related to cycles and their fundamental role in graph theory. We begin by studying the decomposition of cycles and cocycles, the Arc Colouring Lemma proposed by Minty (1960), and the notions of cycle and cocycle bases.

We then introduce the cyclomatic and cocyclomatic numbers, which measure the structure and connectivity of a graph. Next, we address the concept of planarity, where we learn how to determine whether a graph can be drawn on a plane without edge intersections. This section includes Euler's formula, Kuratowski's theorem (1930), and the idea of a dual graph. Finally, we study trees, forests, and arborescences, focusing on their definitions, main properties, and the construction of maximum (or spanning) trees, which are key structures in many applications of graph theory.

1. Cyclomatic and cocyclomatic number

1.1. Decomposition of cycles and cocycles into elementary sums

Decomposition of cycle into elementary cycles:

A cycle is said to be *elementary* if and only if it is minimal, meaning that it contains no other cycle as a proper subset. A cycle can be decomposed into a sum of pairwise arc-disjoint elementary cycles. This decomposition is achieved by traversing the cycle and extracting an elementary cycle each time a previously visited vertex is encountered. The procedure is formalized and presented as an explicit algorithm.

Algorithm:

Input: A graph and a cycle represented as a sequence of vertices, e.g., $(v_1, v_2, \ldots, v_k)$.

Output: A list of elementary cycles that are pairwise arc-disjoint.

Steps:

1. Initialize an empty list **ElementaryCycles = []**.
   Initialize an empty set **Visited = {}**.
2. Traverse the cycle from the first vertex to the last:
   - If v_i is not in **Visited**, add it to **Visited**.
   - If v_i is already in **Visited**:
     - An elementary cycle is detected. It starts from the previous occurrence of v_i to the current position.
     - Extract this subsequence as a new cycle and add it to **ElementaryCycles**.
     - Remove the vertices of this cycle from **Visited**.
3. Return the list **ElementaryCycles**.

Example:

Consider the cycle **(f,h,f,c,e,f)** in the graph below. This cycle is *not elementary*, as it contains repeated vertices.

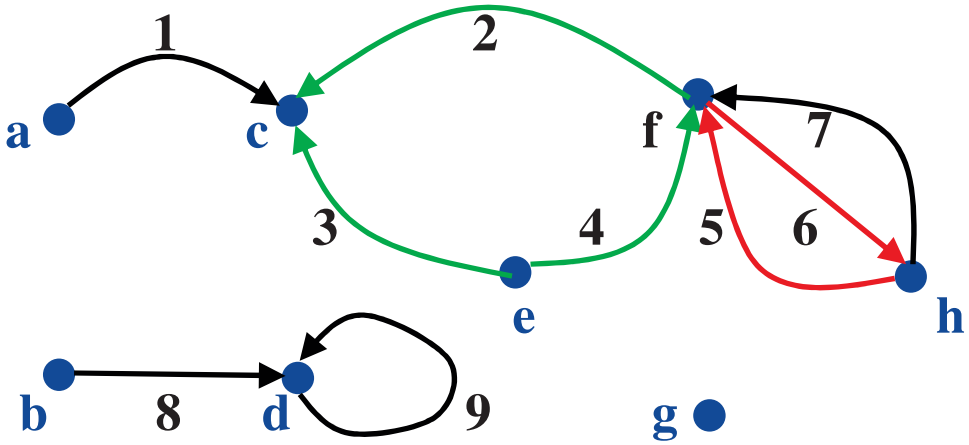


Figure 1. Decomposition of cycle into elementary cycles.

Traverse the cycle from the first occurrence of **f** to its second appearance: **(f,h,f)**, this is an *elementary cycle*.

The remaining chain is **(f,c,e,f)**, which we traverse again. From the next occurrence of **f** to the final one: **(f,c,e,f)**, this is another *elementary cycle*.

The cycle **(f,h,f,c,e,f)** is decomposed into the following elementary cycles:

(f,h,f) and **(f,c,e,f)**.

Let μ be the vector associated with the cycle **(f,h,f,c,e,f)**, which corresponds to the sequence of arcs **(6,5,2,3,4)**. We have $\mu = (0,1,-1,1,1,0,0,0)$.

The vectors associated with the elementary cycles are: $\mu_1 = (0,0,0,0,1,1,0,0,0)$ corresponding to **(6,5)** or **(f,h,f)**. $\mu_2 = (0,1,-1,1,0,0,0,0,0)$ corresponding to **(2,3,4)** or **(f,c,e,f)**.

Thus, we have the decomposition: $\mu = \mu_1 + \mu_2$.

Decomposition of cocycle into elementary cocycles:

A cocycle can be decomposed into a sum of elementary cocycles that are pairwise arc-disjoint.

The following algorithm decompose a cocycle into disjoint elementary cocycles.

Algorithm:

Input:

A graph **G=(X,U),** and a cocycle $\omega_G(A)$ associated with a vertex subset $A \subset X$.

Output:

A decomposition of $\omega_G(A)$ into pairwise arc-disjoint elementary cocycles.

Steps:

1. Identify the connected components $A_1, A_2, ..., A_k$ of the subgraph generated by the subset A.
2. For each **i=1,…,k,** determine the connected component C_i of the graph G that contains A_i.
3. For each **i=1,…,k,** determine the connected components $C_{i1}, C_{i2}, ..., C_{im_i}$ of the subgraph generated by $C_i - A_i$.
4. For each **i=1,…,k,** and for each **j=1,…,m_i,** construct the cocycle $\omega_G^{(i,j)} = -\omega_G(C_{ij})$.

[$\omega_G^{(i,j)}$ is an elementary cocycle since $\omega_G(C_{ij})$ joins the connected subgraphs C_{ij} and $A_i \cup C_{i1} \cup C_{i2} \cup ... C_{im_i}$. Furthermore, $\omega_G^{(i,j)}$ **(for i=1,…,k,** and **j=1,…,m_i)** are pairwise arc-disjoint.]

5. Return the list of all elementary cocycles $\omega_G^{(i,j)}$ as a decomposition of $\omega_G(A)$ into pairwise arc-disjoint elementary cocycles, i.e., $\omega_G(A) = \sum_{\substack{1\le i\le k \\ 1\le j\le i_m}} \omega_G^{(i,j)} = -\sum_{\substack{1\le i\le k \\ 1\le j\le i_m}} \omega_G(C_{ij})$.

Example:

In the graph **G = (X , U)** shown below, let us consider a cocycle $\omega = (1,2,3,4,5)$ associated with the subset $A = \{a,b,c,d\} \subset X$.

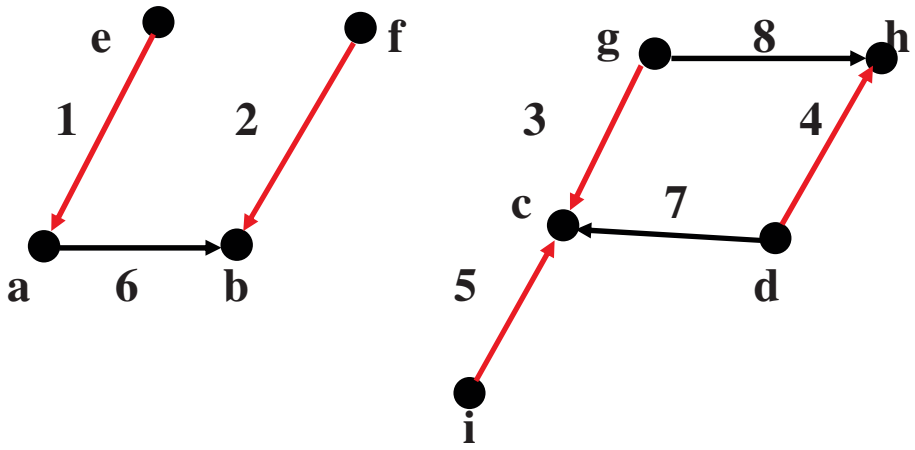


Figure 2: Example of a cocycle

To decompose the cocycle ω into elementary cocycles, we proceed with the following steps:

The subgraph generated by **A** has two connected components, $A_1 = \{a,b\}$ and $A_2 = \{c,d\}$, which are respectively included in $C_1 = \{a,b,e,f\}$ and $C_2 = \{c,d,g,h,i\}$, the two connected components of the graph.

The subgraph generated by $C_1 - A_1 = \{e,f\}$ has two connected components, $C_{1,1} = \{e\}$ and $C_{1,2} = \{f\}$. Similarly, the subgraph generated by $C_2 - A_2 = \{g,h,i\}$ has two connected components, $C_{2,1} = \{g,h\}$ and $C_{2,2} = \{i\}$.

For the vectors associated with the elementary cocycles, we obtain:

$$\omega_G(A) = \omega_G(A_1) + \omega_G(A_2)$$
$$= -\omega_G(C_{11}) - \omega_G(C_{12}) - \omega_G(C_{21}) - \omega_G(C_{22}).$$

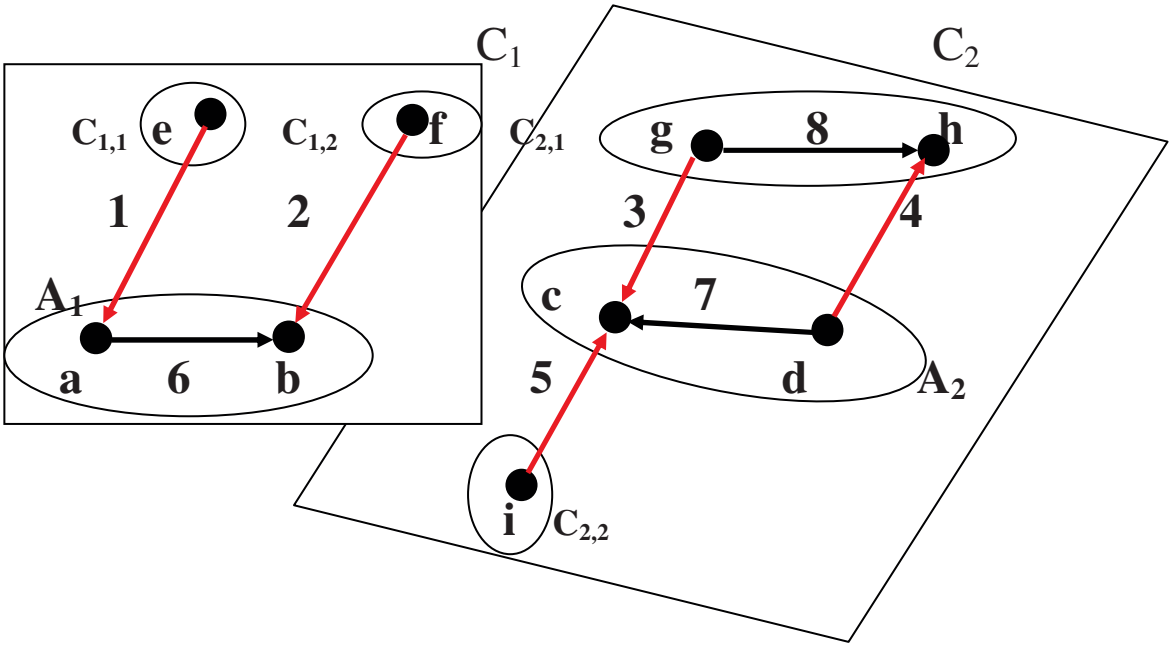


Figure 3: Illustration of the decomposition of a cocycle into elementary cocycles

1.2. <u>Arc colouring lemma (Minty 1960):</u>

Consider a graph with arcs numbered **1**, **2**, …, **m**, and colored either red, green, or black. Arc **1** is assumed to be colored black. Exactly one of the following conditions holds:

(1) there is an elementary cycle containing arc **1** and only red and black arcs with the property that all black arcs in the cycle have the same direction.

(2) there is an elementary cocycle containing arc **1** and only green and black arcs, with the property that all black arcs in the cocycle have the same direction.

***Consequence*:** Each arc belongs either to an elementary circuit or to an elementary cocircuit, but no arc belongs to both.

The following algorithm decides whether arc **1** belongs to an elementary cycle or cocycle:

Algorithm:

Input:

- A graph with arcs colored black, red, or green.
- Arc $1=(\mathbf{b},\mathbf{a})$, and it is colored black.

Steps:

1. Initialization:

- Let $\mathbf{M}$ be the set of marked vertices.
- Initialize $\mathbf{M}=\{\mathbf{a}\}$.

2. Marking Process:

Repeat until no new vertices can be added to $\mathbf{M}$:

For each vertex $\mathbf{x}\in\mathbf{M}$, do:

- For each unmarked vertex $\mathbf{y}$:
  - If there is a black arc $(\mathbf{x},\mathbf{y})$, then mark $\mathbf{y}$ (add $\mathbf{y}$ to $\mathbf{M}$).
  - If there is a red arc $(\mathbf{x},\mathbf{y})$ **or** $(\mathbf{y},\mathbf{x})$, then mark $\mathbf{y}$ (add $\mathbf{y}$ to $\mathbf{M}$).

3. Termination and Case Analysis:

- If $\mathbf{b}\in\mathbf{M}$, then:
  - The path used to mark $\mathbf{b}$ forms an elementary red-and-black cycle with all black arcs in the same direction.
  - Return: "Arc **1** belongs to an elementary cycle."
- Else:
  - Let $\mathbf{A}=\mathbf{M}$.
  - The arcs between $\mathbf{A}$ and its complement form a green-and-black cocycle with all black arcs oriented into $\mathbf{A}$.
  - Return: "Arc **1** belongs to an elementary cocycle."

Example:

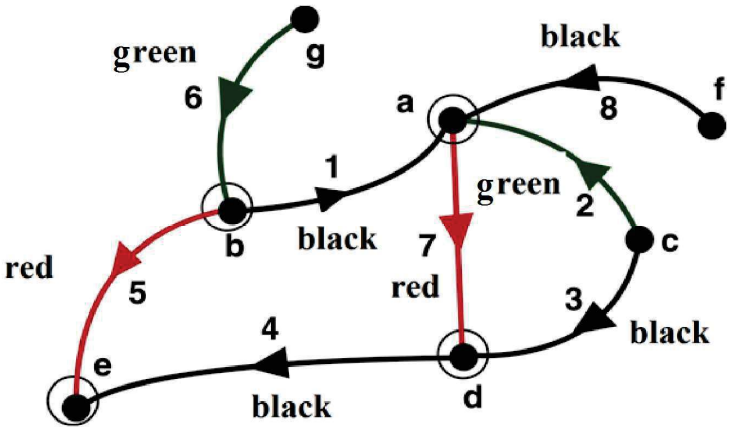


Figure 4: Example of a black and red cycle with all black arcs oriented in the same direction

Let the black arc be **1 = (b, a)**, and suppose that vertex **a** is initially marked.

Starting from vertex **a**, there is no black arc of the form **(a, x)**. However, we find a red arc **7 = (a, d)**. According to the marking rule, since there is a red arc from **a** to **d**, we mark vertex **d** and add it to the set **M** of marked vertices.

From vertex **d,** we find a black arc **4 = (d, e)**. Hence, we mark vertex **e** and add it to the set **M** of marked vertices.

From vertex **e,** there exists a red arc **5 = (b, e)** that connects back arc **1** to the marked vertex **e**. We marked vertex **b**. This completes a closed path **(1, 7, 4, 5)** that alternates between red and black arcs.

The path **(1, 7, 4, 5)** forms an elementary red-and-black cycle in which all black arcs are oriented in the same direction.

Therefore, arc **1** belongs to an elementary cycle.

Example:

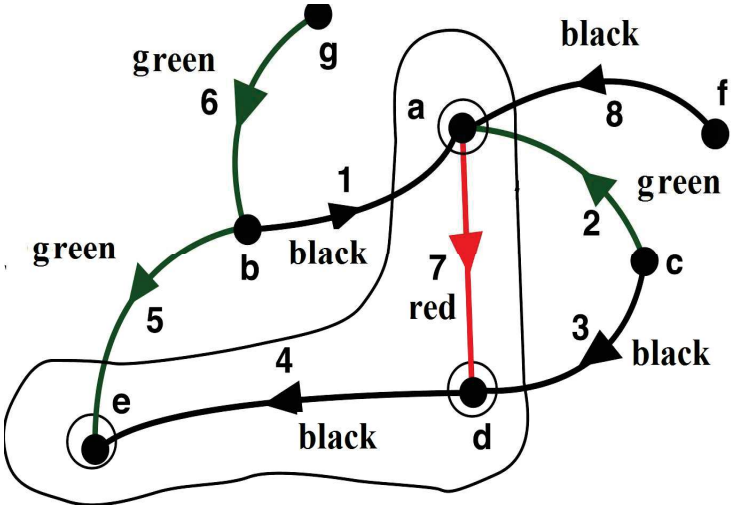


Figure 5: Example of a black and green cocycle with all black arcs oriented in the same direction

Contrary to the previous example, in this case, when the process reaches vertex **e**, there is no black arc of the form **(e, x)** and no red arc of the form **(e, x)** or **(x, e)**. Therefore, the marking process terminates at this point, and we conclude that there is no elementary red-and-black cycle to which arc **1** belongs. However, we can identify that arc 1 is part of an elementary black-and-green cocycle **{1, 2, 3, 5, 8}**, in which all black arcs are oriented in the same direction.

1.3. Cycle basis and cocycle basis:

Definitions:

The cycles $\mu_1, \mu_2, \dots, \mu_k$ are said to be *independent* if and only if a linear combination of their associated vectors is zero only when each coefficient is zero.

A *fundamental cycle basis* is a set of independent cycles such that any cycle can be expressed as a linear combination of these. The *cyclomatic number* of the graph is equal to the dimension of the cycle basis.

Similarly, cocycles $\omega_1, \omega_2, \dots, \omega_k$ are independent if and only if a linear combination of their associated vectors is zero only when each coefficient is zero.

A *fundamental cocycle basis* is a set of independent cocycles such that any other cocycle can be expressed as a linear combination of these. The *cocyclomatic number* of the graph is equal to the dimension of the cocycle basis.

Notations:

The cyclomatic number of a graph G is denoted as $\upsilon(\mathbf{G})$, and the cocyclomatic number is denoted as $\lambda(\mathbf{G})$.

Example:

Consider the graph G in **Figure 6**:

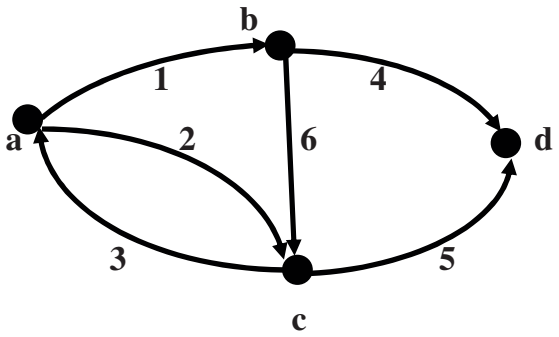


Figure 6: cyclomatic and cocyclomatic numbers**.**

The elementary cycles of the graph are :

$\mu_1 = (1,6,2) = (\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{a})$,

$\mu_2 = (1,6,3) = (\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{a})$,

$\mu_3 = (2,3) = (\mathbf{a},\mathbf{c},\mathbf{a})$,

$\mu_4 = (1,4,5,2) = (\mathbf{a},\mathbf{b},\mathbf{d},\mathbf{c},\mathbf{a})$,

$\mu_5 = (6,5,4) = (\mathbf{a},\mathbf{c},\mathbf{d},\mathbf{b})$,

$\mu_6 = (1,4,5,3) = (\mathbf{a},\mathbf{b},\mathbf{d},\mathbf{c},\mathbf{a})$.

These cycles are not independent, since we have, for example: $\mu_1 - \mu_2 + \mu_3 = 0$, but The cycles μ_1, μ_3, μ_4 form a cycle basis, and therefore, $\upsilon(\mathbf{G}) = 3$.

For the graph in **Figure 6**, the elementary cocycles are:

$\omega(\{\mathbf{a}\}) = \{1,2,3\}$,

$\omega(\{\mathbf{a},\mathbf{b}\}) = \{6,2,3,4\}$,

$\omega(\{\mathbf{a},\mathbf{c}\}) = \{6,1,5\}$,

$\omega(\{\mathbf{a},\mathbf{b},\mathbf{c}\}) = \{4,5\}$,

$\omega(\{\mathbf{a},\mathbf{b},\mathbf{d}\}) = \{6,2,3,5\}$,

$\omega(\{\mathbf{a},\mathbf{c},\mathbf{d}\}) = \{6,1,4\}$,

Obviously, these cocycles are not independent. To form a basis, one could take, for example, $\omega(\{\mathbf{a}\})$, $\omega(\{\mathbf{a},\mathbf{b},\mathbf{c}\})$ and $\omega(\{\mathbf{a},\mathbf{b},\mathbf{d}\})$. Hence $\lambda(\mathbf{G}) = 3$.

We have the following result:

Theorem:

Let a graph **G** have **n** vertices, **m** arcs, and **p** connected components. Then: $\upsilon(\mathbf{G}) = \mathbf{m} - \mathbf{n} + \mathbf{p}$ and $\lambda(\mathbf{G}) = \mathbf{n} - \mathbf{p}$.

Example:

Taking **Figure 6,** since $\upsilon(\mathbf{G}) = 6 - 4 + 1 = 3$ and $\lambda(\mathbf{G}) = \mathbf{n} - \mathbf{p} = 4 - 1 = 3$, the outcome is consistent with the results obtained in previous Example**.**

2. Planarity

2.1. Planar Graph

A graph **G** is said to be *planar* if it can be drawn on a plane such that its vertices are represented by distinct points and its arcs (or edges) are represented by simple curves that do not intersect except at their endpoints. A drawing of **G** that satisfies these conditions is called a *topological planar representation* of the graph.

Example: The following graph is planar:

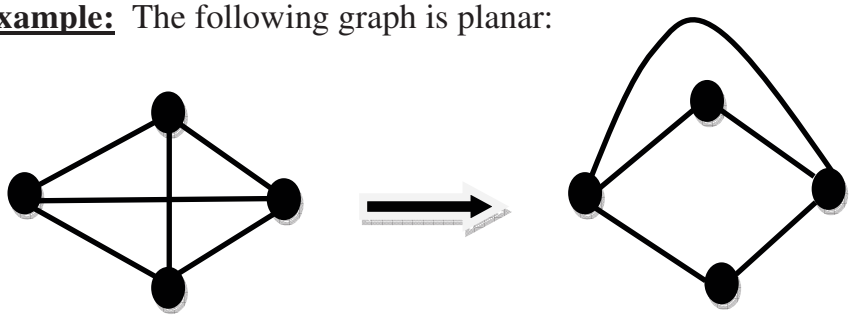


Figure 7: Topological representation of a planar graph**.**

Example:

Three factories, labeled **a**, **b** and **c**, receive water from point **d**, gas from point **e**, and electricity from point **f** through underground supply lines. The question is whether it is possible to arrange the three factories and the three utility stations so that no two supply lines intersect except at their endpoints.

It can be shown that while eight supply lines can be drawn without crossings, the ninth line must necessarily intersect at least one of the others. Therefore, the graph $\mathbf{K}_{3,3}$ is not planar.

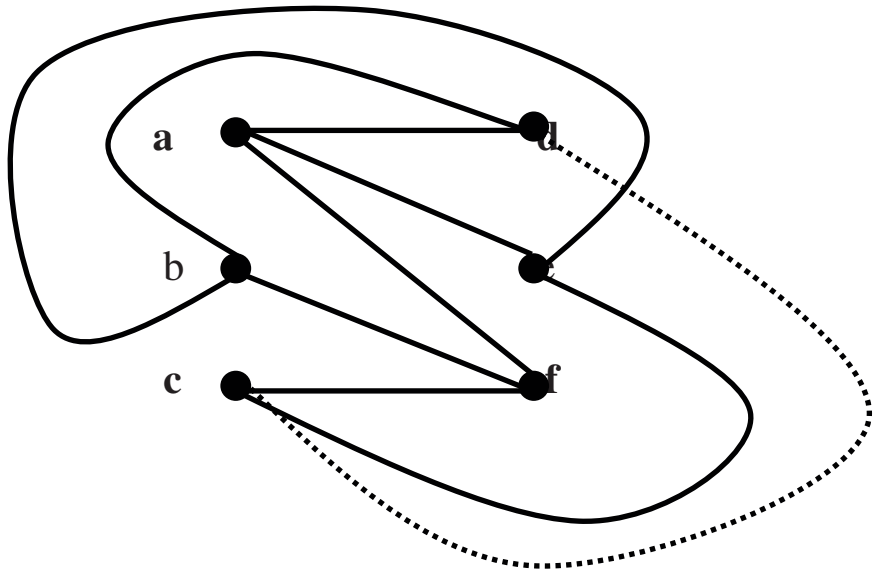


Figure 8: Problem of three factories and three utilities.

Let **G** be a topological representation of a planar graph.

A *face* of **G** is a region of the plane bounded by edges such that any two points within the region can be connected by a continuous curve that does not pass through any vertex or edge of **G**.

The *boundary* of a face is the set of edges that surround it.

The *contour* of a face is defined as an elementary cycle formed by the edges of its boundary that encloses the face **z** in its interior. Note that there is exactly one

unbounded face, which has no contour, whereas all other faces are *bounded* and possess exactly one contour.

Two faces are said to be *adjacent* if their boundaries share a common edge (if they meet only at a vertex, they are not considered adjacent).

Example:

The following figure gives a geographic map corresponds to a topological planar multigraph whose edges are the borders between countries:

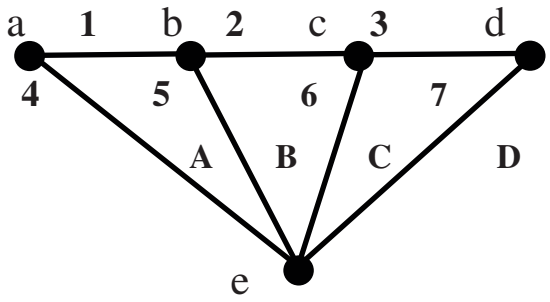


Figure 9: a topological planar graph

Faces A, B, and C are bounded, while D is the unbounded face. The edges 1, 4, and 5 form the boundary of face A. Faces A and B are adjacent.

We have the following results:

Theorem:

In a topological planar graph, the contours of the different bounded faces constitute a cycle basis.

2.2. Euler's formula

If a connected topological planar graph has **n** vertices, **m** arcs and **f** faces, then: $\mathbf{n}-\mathbf{m}+\mathbf{f}=2.$

Example:

In the graph shown by **Figure 9**, we have $\mathbf{n}-\mathbf{m}+\mathbf{f}=5-7+4=2.$

One consequence of Euler's Formula is:

Corollary:

A simple planar graph **G** has a vertex **x** of degree $\mathbf{d}_{\mathbf{G}}(\mathbf{x}) \leq 5$.

2.3. Kuratowski's theorem (1930)

Note first that $\mathbf{K}_5$ and $\mathbf{K}_{3,3}$ are not planar (see Tutorial Session 4).

A multigraph **G** is planar if, and only if, it contains no partial subgraph of type $\mathbf{K}_5$ or type $\mathbf{K}_{3,3}$.

2.4. Dual graph:

Let **G** be a planar graph that is connected and has no isolated vertices. The *dual graph* of **G**, denoted by **D**, is defined as follows:

Each vertex of **D** corresponds to a distinct face of **G**. An edge connecting two vertices in **D** represents an edge in **G** that is shared by the two corresponding faces. Graph **D** is also planar, connected and without isolated vertices.

See **Figure 10**. Graph **D** is called the *topological dual* of G. Note that:

(1) The topological dual of **D** is **G**.

(2) A loop in **G** corresponds to a pendant edge in D, and vice versa.

Example:

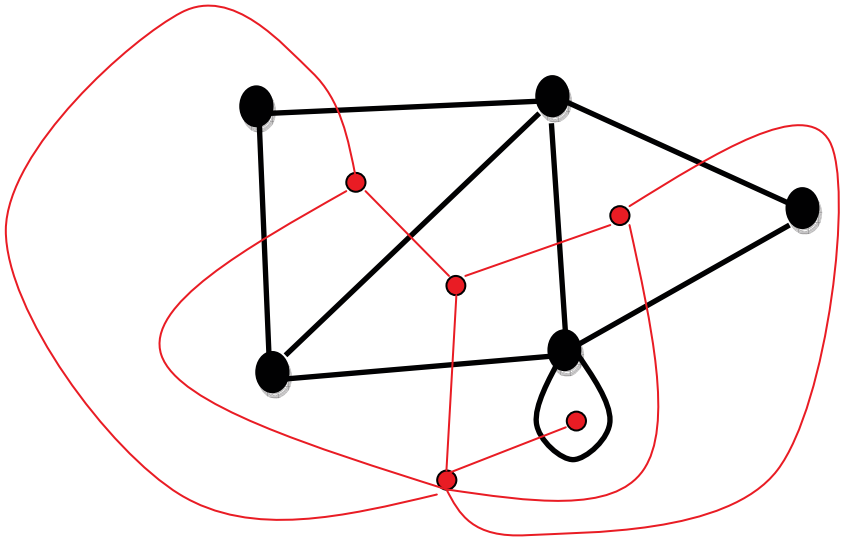


Figure 10: An example of a topological dual graph.

3.Tree, forest and arborescence

3.1. Tree and forest:

A *tree* is defined as a connected graph containing no cycles.

A graph that is acyclic but not necessarily connected is referred to as a *forest*.

Example:

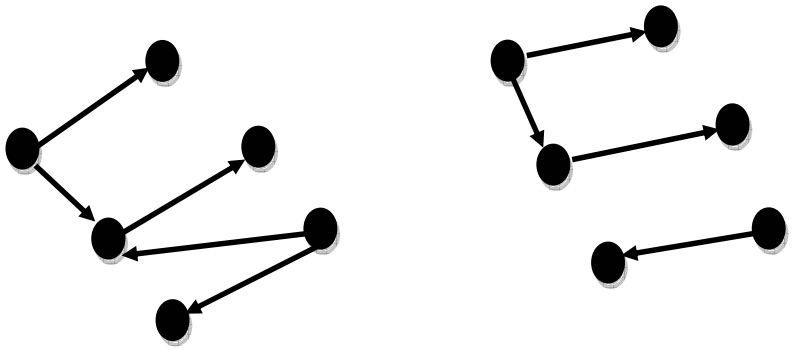


Figure 11: Example of a tree and forest.

The following properties are equivalent and each characterizes a tree:

Properties:

Let **H=(X,U)** be a graph of order **|X|=n>2**. The following statements are equivalent and characterize a tree:

1. **H** is connected and contains no cycles.
2. **H** contains **n-1** arcs and has no cycles.
3. **H** is connected and contains exactly **n-1** arcs.
4. **H** has no cycles, and adding any arc to **H** creates exactly one cycle.
5. **H** is connected, and removing any arc disconnects the graph.
6. Every pair of vertices of H is Connected by one and only one chain.

3.2. Arborescence:

In a graph **G,** a vertex **x** is called a *root* if all the vertices of **G** can be reached by paths starting from **x**.

An *arborescence* is defined as a tree that has a root.

Example:

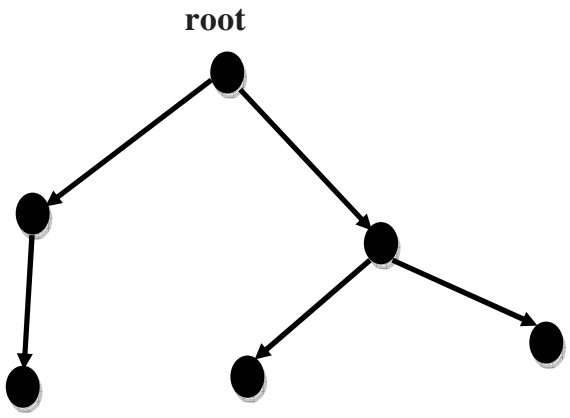
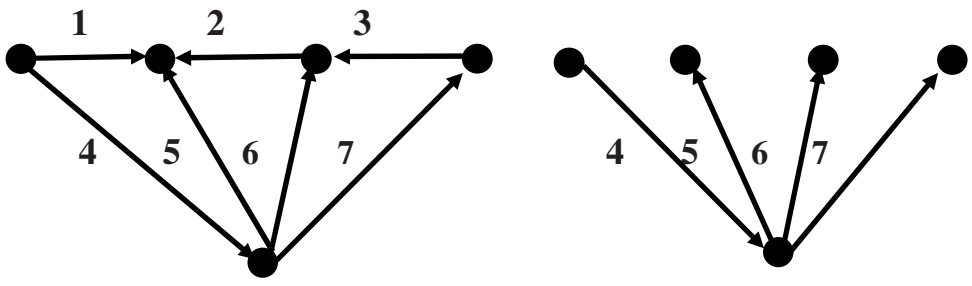


Figure 12: Example of an arborescence.

3.2.Maximum (or spanning) tree:

From any connected graph, one can obtain a partial graph that forms a tree. A *spanning tree* of a connected graph **G** is a partial graph that includes all the vertices of **G** and forms a tree. Equivalently, a spanning tree can be defined as a minimal connected partial graph of **G** or, equivalently, as a maximal acyclic partial graph of **G**. If **G** has **n** vertices, every spanning tree of **G** has exactly **n-1** arcs.

Example:



Graph **G** Spanning tree **H**

Figure 12: The partial graph of G generated by the set of arcs **{4,5,6,7}** form a spanning tree **H**.

A spanning tree can be constructed by the following algorithm:

Algorithm:

Input: A connected graph $\mathbf{G} = (\mathbf{X}, \mathbf{U})$.

Output: A spanning tree $\mathbf{T} = (\mathbf{X}, \mathbf{U_T})$.

Steps:

1. **Initialization:**

   Set $\mathbf{U_T} = \{\mathbf{u_0}\}$, where $\mathbf{u_0}$ is any arc (edge) of **G**.

2. **Iteration:**

   Repeat the following process:

- Find an arc $\mathbf{u_i} \in \mathbf{U} \setminus \mathbf{U_T}$ such that adding $\mathbf{u_i}$ to $\mathbf{U_T}$ does not create a cycle in the partial graph $(\mathbf{X}, \mathbf{U_T} \cup \{\mathbf{u_i}\})$.
- Add $\mathbf{u_i}$ to $\mathbf{U_T}$.

3. **Termination:**

   When no further arc can be added without forming a cycle, stop.

   The resulting partial graph $\mathbf{T} = (\mathbf{X}, \mathbf{U_T})$ is a spanning tree of **G**.

Example:

Consider the connected graph $\mathbf{G} = (\mathbf{X}, \mathbf{U})$:

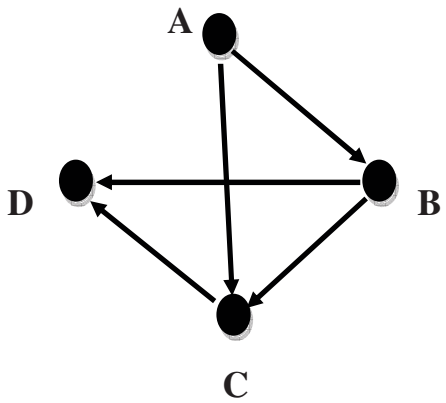


Figure 13: A connected graph.

Choose any arc to star: Let $\mathbf{u}_0 = (\mathbf{A},\mathbf{B})$, then $\mathbf{U}_T = \{(\mathbf{A},\mathbf{B})\}$.

Add a new arc that does not form a cycle: Choose $\mathbf{u}_1 = (\mathbf{A},\mathbf{C})$.

Adding $(\mathbf{A},\mathbf{C})$ (does not form a cycle). Now $\mathbf{U}_T = \{(\mathbf{A},\mathbf{B}),(\mathbf{A},\mathbf{C})\}$.

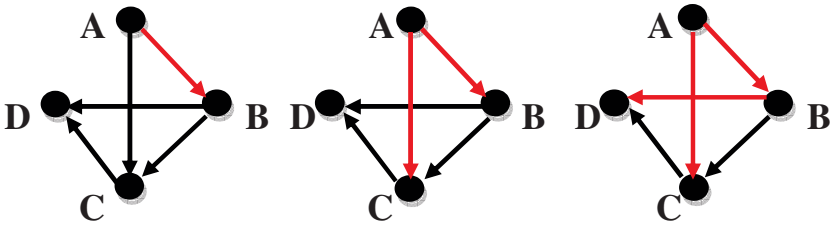
Continue adding: **Choose** $\mathbf{u}_2 = (\mathbf{B},\mathbf{D})$. Adding $(\mathbf{B},\mathbf{D})$ (does not form a cycle).

Now $\mathbf{U}_T = \{(\mathbf{A},\mathbf{B}),(\mathbf{A},\mathbf{C}),(\mathbf{B},\mathbf{D})\}$.

Stop when no more edges can be added without creating a cycle**:** Remaining edges: $(\mathbf{B},\mathbf{C})$ and $(\mathbf{C},\mathbf{D})$.

Adding either of them creates a cycle. So the process stops.

Result: The spanning tree is $\mathbf{T} = (\mathbf{X},\{(\mathbf{A},\mathbf{B}),(\mathbf{A},\mathbf{C}),(\mathbf{B},\mathbf{D})\})$.



$\mathbf{U}_T = \{(\mathbf{A},\mathbf{B})\}$ $\mathbf{U}_T = \{(\mathbf{A},\mathbf{B}),(\mathbf{A},\mathbf{C})\}$ $\mathbf{U}_T = \{(\mathbf{A},\mathbf{B}),(\mathbf{A},\mathbf{C}),(\mathbf{B},\mathbf{D})\}$

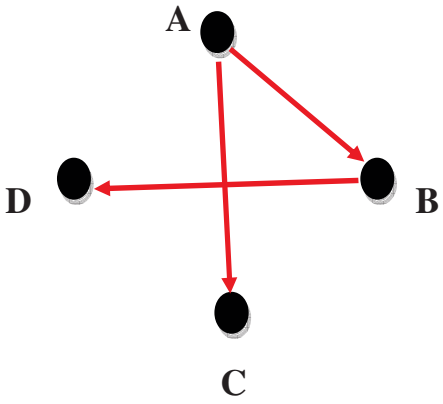


Figure 14: A spanning tree.

Cotree:

Given a connected graph $\mathbf{G} = (\mathbf{X}, \mathbf{U})$ and one of its spanning trees $\mathbf{T} = (\mathbf{X}, \mathbf{U_T})$.

The **cotree** associated with **T** is the partial graph $\mathbf{T^*} = (\mathbf{X}, \mathbf{U_{T^*}})$ where $\mathbf{U_{T^*}} = \mathbf{U} \setminus \mathbf{U_T}$ is the set of edges of **G** not belonging to the spanning tree.

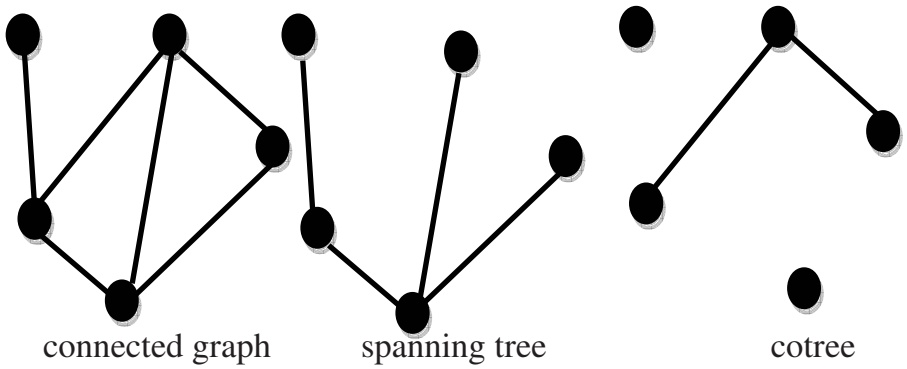
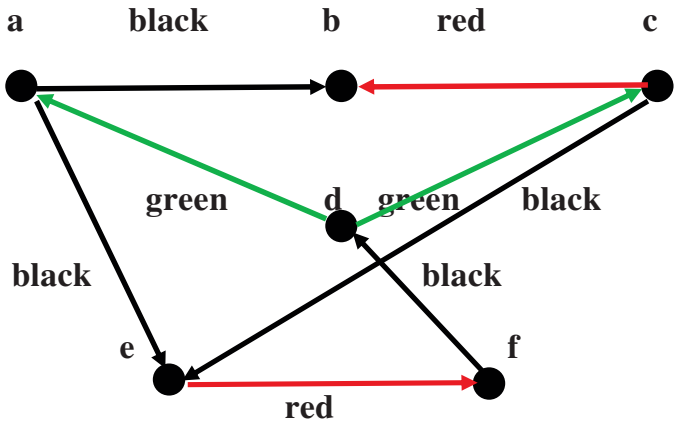


Figure 15: A cotree.

Tutorial Session (5)

Exercise 1:

Let **G** the following graph:



1. Decompose the cycle **(a , d , c , e , f , d , a)** into elementary cycles.
2. Decompose the cocycle $\omega(\{a,d,f\})$ into elementary cocycles.
3. Determine whether the arc **(a , b)**, colored in black, belongs to an elementary circuit or to an elementary cocircuit.
4. Calculate the cyclomatic number $\nu(G)$.
5. How many edges must be removed from graph **G** to transform it into a tree?
6. Give the spanning tree **T** of **G**.
7. Give the cotree **T*** associated with **T**.
8. Give a cycle basis for the graph **G**.
9. Give the vectors associated with each cycle of the basis.
10. Calculate the cocyclomatic number $\lambda(G)$.
11. Give a cocycle basis for the graph **G**.
12. Give the vectors associated with each cocycle of the basis.

Exercise 2:

Show that the multigraphs $K_{3,3}$ and K_5 are not planar.

Exercise 3:

Represent the arithmetic expression $\left(\mathbf{e}^{\mathbf{x}}-\mathbf{y}\right)*\left(\mathbf{c}+\mathbf{d}/\mathbf{z}\right)+4$ in the form of a tree, respecting the rules of operator precedence.

Tutorial Session 5 (Solution)

Exercise 1:

1. The decomposition is achieved by traversing the cycle and extracting an elementary cycle each time a previously visited vertex is encountered. The first elementary cycle obtained is (**d , c , e , f , d**). The remaining cycle, (**a , d , a**), is also an elementary cycle.

2. To decompose the cocycle $\omega(\{\mathbf{a},\mathbf{d},\mathbf{f}\})$ into elementary cocycles, we proceed with the following steps:

   The subgraph generated by $\mathbf{A}=\{\mathbf{a},\mathbf{d},\mathbf{f}\}$ has two connected components, $\mathbf{A}_1=\{\mathbf{a}\}$ and $\mathbf{A}_2=\{\mathbf{d},\mathbf{f}\}$, both are included in $\mathbf{C}=\{\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d},\mathbf{e},\mathbf{f}\}$.

   The subgraph generated by $\mathbf{C}-\mathbf{A}_1=\{\mathbf{b},\mathbf{c},\mathbf{d},\mathbf{e},\mathbf{f}\}$ has one connected components $\mathbf{C}_1=\{\mathbf{b},\mathbf{c},\mathbf{d},\mathbf{e},\mathbf{f}\}$. Similarly, the subgraph generated by $\mathbf{C}-\mathbf{A}_2=\{\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{e}\}$ has one connected component, $\mathbf{C}_2=\{\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{e}\}$.

   For the vectors associated with the elementary cocycles, we obtain:

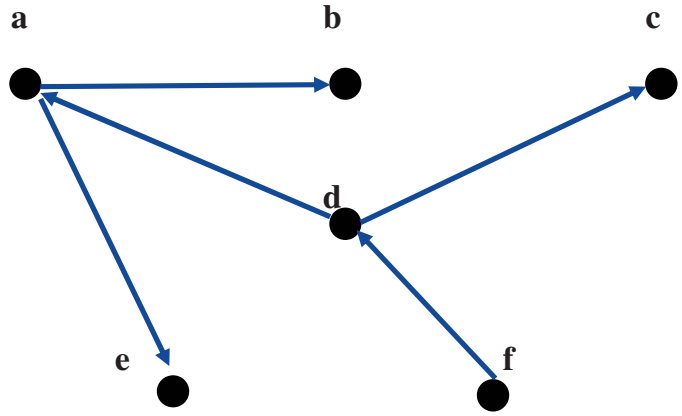
$$\begin{aligned}\omega_G(\mathbf{A}) &= \omega_G(\mathbf{A}_1)+\omega_G(\mathbf{A}_2)\\ &= -\omega_G(\mathbf{C}_1)-\omega_G(\mathbf{C}_2).\end{aligned}$$

3. When the marking process reaches vertex **d**, there is no black arc of the form (**d , x**) and no red arc of the form (**d , x**) or (**x , d**). Therefore, the marking process terminates at this point, and we conclude that there is no elementary red-and-black cycle to which arc (**a , b**) belongs. However, we can identify that arc (**a , b**) is part of an elementary black-and-green cocycle **{ (a , b) , (d , a) , (a , e) }**, in which all black arcs are oriented in the same direction.

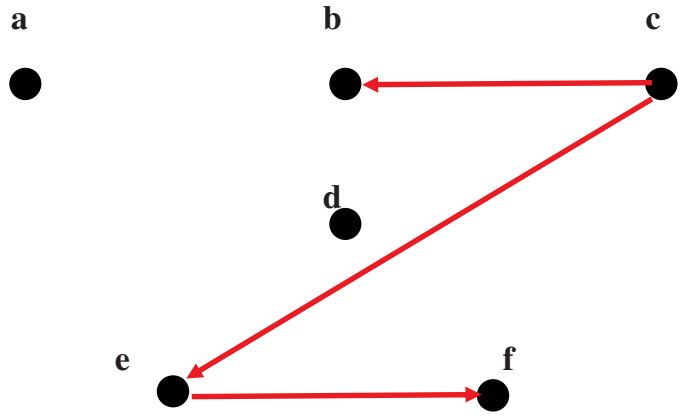
4. The cyclomatic number : $\nu(\mathbf{G})=\mathbf{m}-\mathbf{n}+\mathbf{p}=8-6+1=3.$

5. Let **n** and **m** be, respectively, the order and size of tree **T**. Tree **T** must have: **m = n - 1 = 6 - 1 = 5** edges. But graph **G** has **8** edges. So **3** edges must be removed.

6. A spanning tree **T** of **G**:



7. The cotree **T*** associated with **T**:



8. A cycle basis for the graph **G**:

Cycles : $\mathbf{C_1 = (\ a\ ,\ b\ ,\ c\ ,\ d\ ,\ a\ )}$, $\mathbf{C_2 = (\ d\ ,\ c\ ,\ e\ ,\ a\ ,\ d\ )}$ and $\mathbf{C_3 = (\ a\ ,\ e\ ,\ f\ ,\ d\ ,\ a\ )}$ form a cycle basis because there are three of them, and they are independent since each contains an arc of the cotree that the others do not.

9. Vectors associated with each cycle of the basis:

	(a,b)	**(a,e)**	**(c,b)**	**(c,e)**	**(d,a)**	**(d,c)**	**(e,f)**	**(f,d)**
$\mathbf{C_1}$	1	0	-1	0	1	-1	0	0
$\mathbf{C_2}$	0	-1	0	1	-1	1	0	0
$\mathbf{C_3}$	0	1	0	0	1	0	1	1

10. The cocyclomatic number: $\lambda(\mathbf{G}) = \mathbf{n} - \mathbf{p} = 6 - 1 = 5$.

11. A cocycle basis for the graph **G**:

Cocycles :

$\mathbf{CO}_1 = \omega(\{\mathbf{a},\mathbf{c},\mathbf{d},\mathbf{e},\mathbf{f}\}) = \{(\mathbf{a},\mathbf{b}),(\mathbf{c},\mathbf{b})\}$

$\mathbf{CO}_2 = \omega(\{\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d},\mathbf{f}\}) = \{(\mathbf{a},\mathbf{e}),(\mathbf{c},\mathbf{e}),(\mathbf{e},\mathbf{f})\}$

$\mathbf{CO}_3 = \omega(\{\mathbf{a},\mathbf{b},\mathbf{d},\mathbf{e},\mathbf{f}\}) = \{(\mathbf{d},\mathbf{c}),(\mathbf{c},\mathbf{b}),(\mathbf{c},\mathbf{e})\}$

$\mathbf{CO}_4 = \omega(\{\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d},\mathbf{e}\}) = \{(\mathbf{f},\mathbf{d}),(\mathbf{e},\mathbf{f})\}$

$\mathbf{CO}_5 = \omega(\{\mathbf{a},\mathbf{b},\mathbf{e}\}) = \{(\mathbf{d},\mathbf{a}),(\mathbf{c},\mathbf{e}),(\mathbf{e},\mathbf{f}),(\mathbf{c},\mathbf{b})\}$

form a cocycle basis because there are five of them, and they are independent since each contains an arc of the tree **T** that the others do not.

12. The vectors associated with each cocycle of the basis:

	(a,b)	**(a,e)**	**(c,b)**	**(c,e)**	**(d,a)**	**(d,c)**	**(e,f)**	**(f,d)**
CO_1	1	0	1	0	0	0	0	0
CO_2	0	1	0	1	0	0	-1	0
CO_3	0	0	-1	-1	0	1	0	0
CO_4	0	0	0	0	0	0	1	-1
CO_5	0	0	-1	-1	-1	0	1	0

Exercise 2:

a) Suppose that the multigraph $\mathbf{K}_{3,3}$ (representing the 3 houses and 3 factories problem) is planar. Then Euler's formula must hold, so: $\mathbf{f} = 2 - \mathbf{n} + \mathbf{m} = 2 - 6 + 9 = 5$.

Each face must have at least 4 edges on its boundary; otherwise, two vertices from the same set would be connected. The number of arcs in the bipartite incidence graph between faces and edges therefore gives: $2\mathbf{m} = 18 > 4\mathbf{f} = 20$. The contradiction in this result proves that $\mathbf{K}_{3,3}$ is not planar.

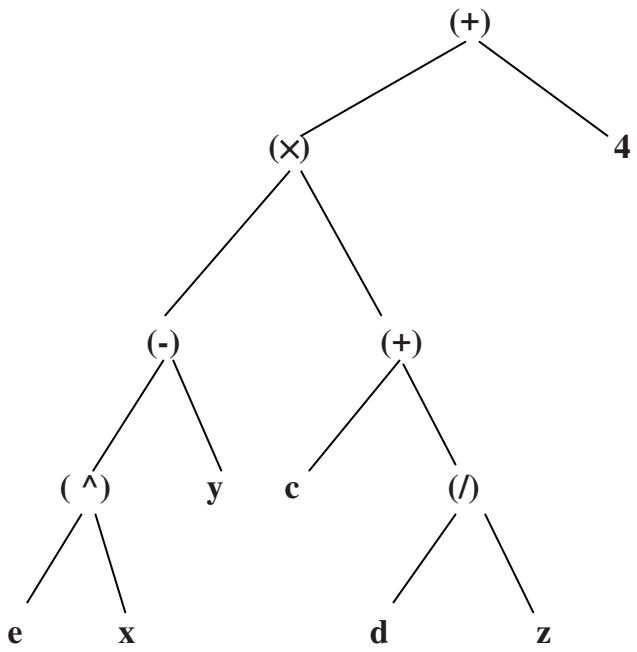
b) Now suppose that multigraph $\mathbf{K}_5$ is planar. Then Euler's formula must hold, so: $\mathbf{f} = 2 - \mathbf{n} + \mathbf{m} = 2 - 5 + 10 = 7$. Each face must have at least 3 edges on its boundary.

The number of arcs in the bipartite incidence graph between faces and edges therefore gives: $2\mathbf{m} = 20 > 3\mathbf{f} = 21$. The contradiction in this result proves that $\mathbf{K}_5$ is not planar.

Exercise 3:

The arithmetic expression $\left(\mathbf{e}^{\mathbf{x}} - \mathbf{y}\right) * \left(\mathbf{c} + \mathbf{d} / \mathbf{z}\right) + 4$ is evaluated according to operator precedence: Parentheses → Exponentiation → Division → Multiplication → Addition/Subtraction.

The main operator is **'+'** (evaluated last). The left part $\left(\mathbf{e}^{\mathbf{x}} - \mathbf{y}\right) * \left(\mathbf{c} + \mathbf{d} / \mathbf{z}\right)$ has **'×'** as its root. Inside it: $\left(\mathbf{e}^{\mathbf{x}} - \mathbf{y}\right)$ uses **'–'**, with $\mathbf{e}^{\mathbf{x}}$ (power) on the left and **y** on the right. $\left(\mathbf{c} + \mathbf{d} / \mathbf{z}\right)$ uses **'+'**, where **d / z** (division) is computed before addition. Finally, the result of multiplication is added to **4**.



Chapter III: Flow problems

1. Definitions

1.1. Transportation network:

A transportation network is a graph without loops in which each arc $\mathbf{u}$ is assigned a positive number $\mathbf{C}(\mathbf{u})$, called the *capacity* of arc $\mathbf{u}$. This network contains one vertex without predecessors, called the *source* or *entry* of the network, and another vertex without successors, called the *sink* or *exit* of the network. The network is denoted by $\mathbf{R} = (\mathbf{X}, \mathbf{U}, \mathbf{C})$.

Example:

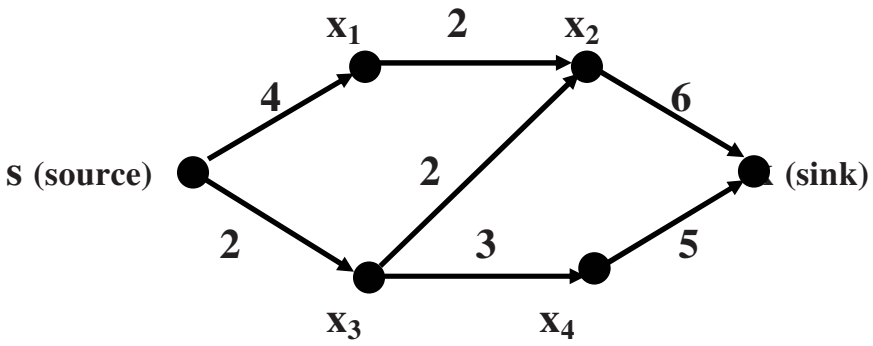


Figure 1: Transportation network.

Flow in a transportation network:

A flow $\mathbf{F}$ in a transportation network assigns to each arc $\mathbf{u}$ a value $\mathbf{F}(\mathbf{u})$, which represents the amount of flow that moves through this arc from the source to the sink.

A flow is *conservative* if it follows Kirchhoff's law at every vertex: "The total flow entering a vertex is equal to the total flow leaving that vertex."

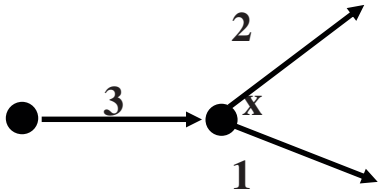


Figure 2: Kirchhoff's law**.**

In the graph shown in Figure 2, the amount of flow entering vertex **x** is equal to the sum of the flow amounts leaving **x**. The value of the flow is **2**. Kirchhoff's law is therefore satisfied at vertex **x**.

Compatible flow:

A flow is *compatible* in a network if, for every arc $\mathbf{u}=(\mathbf{x},\mathbf{y})$, we have $0 \le \mathbf{F}(\mathbf{u}) \le \mathbf{C}(\mathbf{u})$. In other words, for each arc **u**, the flow passing through it does not exceed its capacity.

In what follows, we will use the following representation:

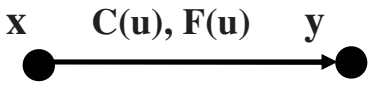


Figure 2: Representation "Capacity, Flow".

Example:

Consider the following network:

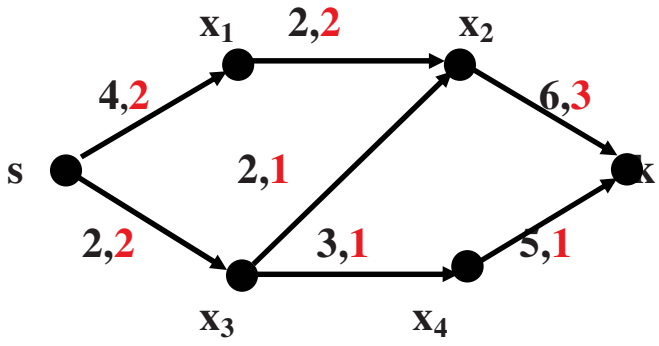


Figure 3: Compatible flow.

In the network shown in Figure 3, the flow passing through each arc does not exceed its capacity. Therefore, this flow is compatible.

Complete flow:

A flow is *complete* if, for every path from the source to the sink, there is at least one *saturated* arc, that is, an arc for which the flow equals its capacity $\mathbf{F(u) = C(u)}$.

Example:

In the network shown in Figure 3, there are three paths leading from **s** to **k**, and in each of them, at least one arc is saturated. Therefore, the flow is *complete*.

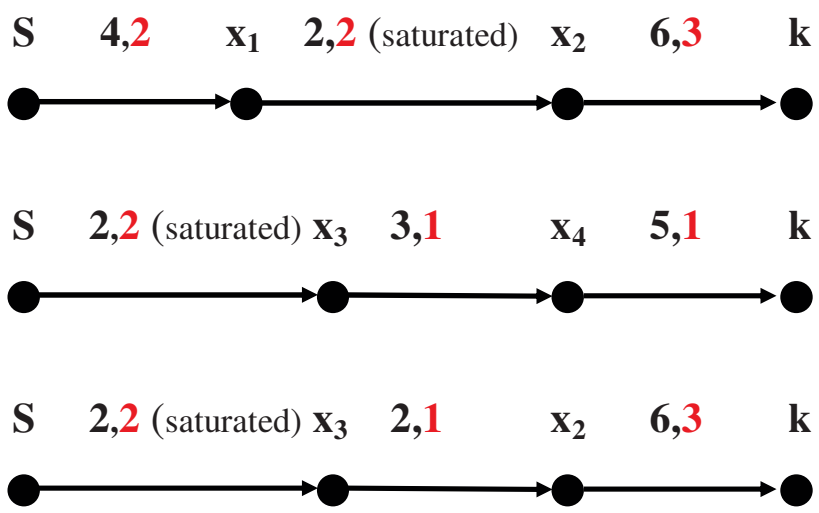


Figure 3: Complete flow**.**

2.2. Ford-Fulkerson theorem

Cut between two vertices and its capacity:

In a transportation network $\mathbf{R = (X, U, C)}$, a *cut* between **a** and **b** is defined to be a set of arcs of the form $\boldsymbol{\omega}^{+}\mathbf{(A)}$ with $\mathbf{a \in A}$ and $\mathbf{b \notin A}$.

The *capacity of a cut* is defined to be the sum of the capacities of its arcs, i.e.,

$$\mathbf{C(\omega^{+}(A)) = \sum_{u \in \omega^{+}(A)} C(u)}.$$

Example:

Consider the following flow network:

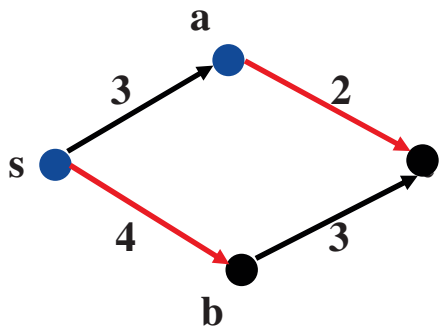


Figure 3: Cut and its capacity.

Let's create a cut between **s** and **t**.

Consider the set $\mathbf{A}=\{\mathbf{s},\mathbf{a}\}$, in which $\mathbf{a}\in\mathbf{A}$ and $\mathbf{t}\notin\mathbf{A}$. The set of arcs $\boldsymbol{\omega}^{+}(\mathbf{A})=\{(\mathbf{s},\mathbf{b}),(\mathbf{a},\mathbf{t})\}$ forms a cut between **s** and **t**.

In this example, the arc **(s,b)** has a capacity of **4,** and the arc $(\mathbf{a},\mathbf{t})$ has a capacity of **2**.

Therefore, the capacity of the cut is: $\mathbf{C}(\mathbf{s},\mathbf{b})+\mathbf{C}(\mathbf{a},\mathbf{t})=4+2=6.$

Maximum flow theorem (Ford, Fulkerson [1957])

In a transportation network, the maximum flow that can go from the source to the sink is equal to the minimum capacity of a cut that separates the source from the sink.

2.3. Ford-Fulkerson algorithm:

The maximum flow problem consists in finding the largest amount of flow that can be sent from the source **s** to the sink **k**, while taking into account the transport capacities of the arcs and the available quantity at the source **s**. The most well-known algorithm for solving the maximum flow problem is the Ford–Fulkerson algorithm. The idea of the Ford–Fulkerson algorithm is to start with a compatible flow in the network (The simplest one being the zero flow) and then to improve it step by step until a complete flow is reached. A chain along which the flow can be increased is called an *augmenting chain.*

This is a chain where: oriented in the traversal direction and by μ^- the set of the other arcs of the cycle.

- the forward arcs (oriented in the traversal direction) have not yet reached their capacity, and
- the backward arcs (oriented in the opposite direction of the traversal) carry a positive flow that can be reduced.

In other words, a chain μ is said to be augmenting if:

- for every forward arc $\mathbf{u} \in \mu^+ : \mathbf{F}(\mathbf{u}) < \mathbf{C}(\mathbf{u})$,
- for every backward arc $\mathbf{u} \in \mu^- : \mathbf{F}(\mathbf{u}) > 0$.

The flow along this chain μ can be increased by the amount: $\mathbf{y} = \min\left\{\left\{\mathbf{C}(\mathbf{u}) - \mathbf{F}(\mathbf{u}) \mid \mathbf{u} \in \mu^+\right\}, \left\{\mathbf{F}(\mathbf{u}) \mid \mathbf{u} \in \mu^-\right\}\right\}$.

To improve the flow, we:

- add **y** to the flow of all forward arcs in the chain, and
- subtract **y** from the flow of all backward arcs in the chain.

Example:

Consider the following transportation flow network:

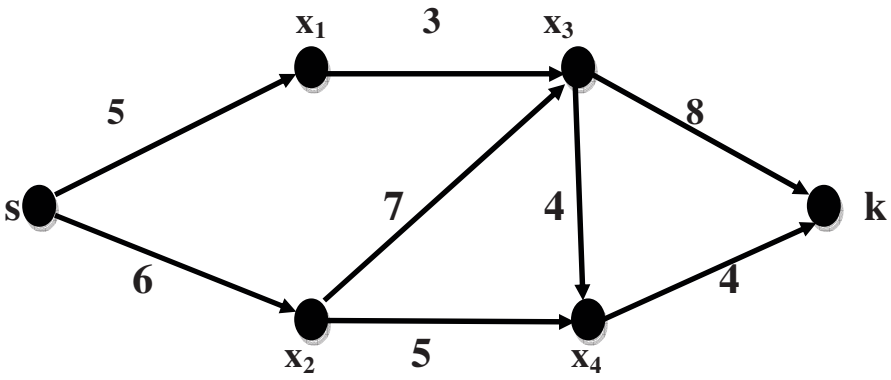
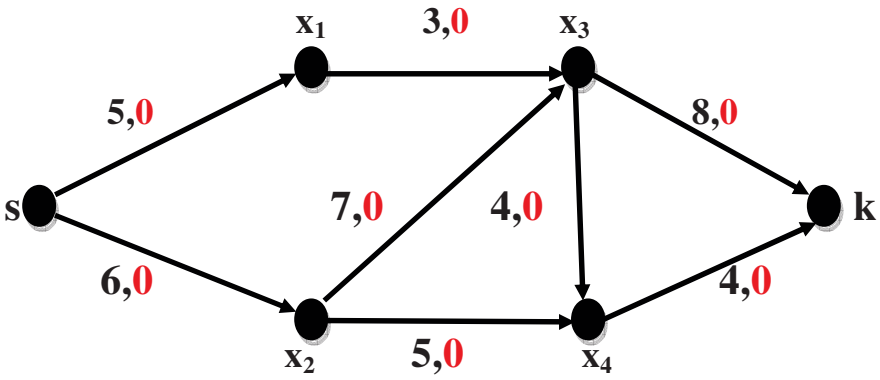
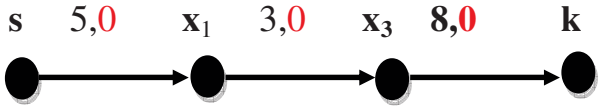


Figure 4: Transportation flow network.

Initialize all arc flows to zero:

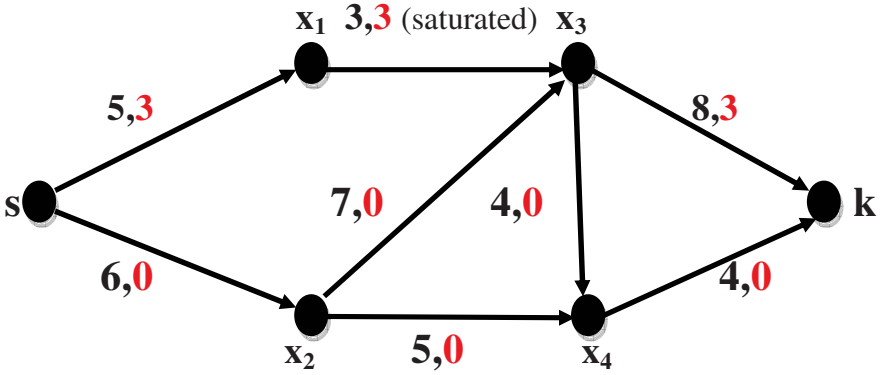


Find an augmenting chain between the source **s** and the sink **k** along which the flow can be increased. For example, we select the following chain:

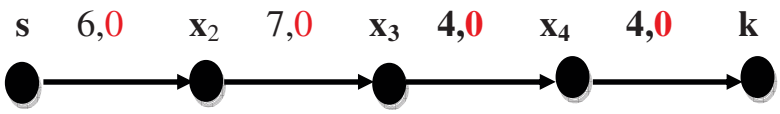


Calculate: $\mathbf{y} = \min\{\mathbf{C}(\mathbf{u}) - \mathbf{F}(\mathbf{u}) \mid \mathbf{u} \in \mu^+\} = \min\{5-0, 3-0, 8-0\} = 3$.

We improve the flow along this chain by adding the quantity 3 to the flow of the arcs in μ^+. The flow of the arcs that do not belong to the chain remains unchanged.

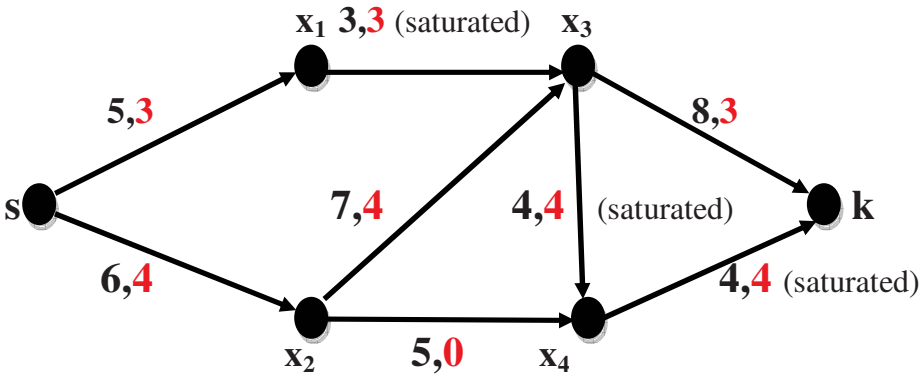


Now, find an augmenting chain in the residual network. We select:

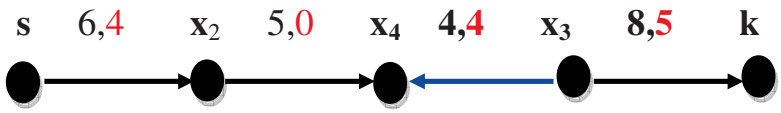


Calculate: $\mathbf{y} = \min\{\mathbf{C}(\mathbf{u}) - \mathbf{F}(\mathbf{u}) \mid \mathbf{u} \in \mu^+\} = \min\{6-0, 7-0, 4-0, 4-0\} = 4$.

We increase the flow by 4 on the arcs of the chain, and all other arcs keep the same flow.



Select the augmenting chain:



Calculate:

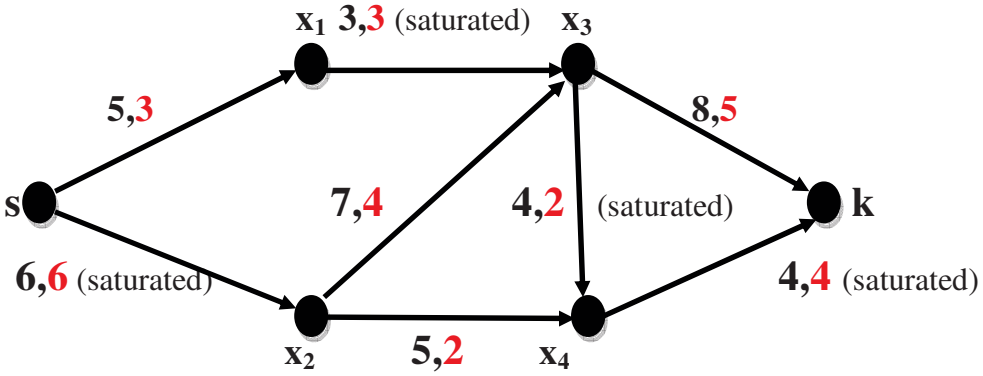
$\mathbf{y}_1 = \min\{\mathbf{C}(\mathbf{u}) - \mathbf{F}(\mathbf{u}) \mid \mathbf{u} \in \mu^+\} = \min\{6-4,\ 5-0,\ 8\text{-}5\} = 2$

and

$\mathbf{y}_2 = \min\{\mathbf{F}(\mathbf{u}) \mid \mathbf{u} \in \mu^-\} = 4.$

Then $\mathbf{y} = \min\{\mathbf{y}_1\ ,\ \mathbf{y}_2\} = \min\{2\ ,\ 4\} = 2.$

We increase the flow by 2 on the forward arcs (μ^+) and decrease it by 2 on the backward arcs (μ^-).



There is no augmenting chain in the residual graph. This means that the Ford–Fulkerson method has finished. Since the maximum flow is equal to the total flow leaving the source, in this example the maximum flow is **6+3 = 9**.

3. **Search for a compatible flow**

If the graph **G** is not a transportation network, the existence of a compatible flow is not guaranteed. The following algorithm search for the existence of such flow:

Algorithm:

1) Start with the zero flow.

2) Search for an arc **u** such that $\mathbf{F(u) < C(u)}$. If no such arc exists END. Otherwise, set **s** as the terminal endpoint of **u**, and **k** as the initial endpoint of **u**.

3) Search for a chain from **s** to **k** such that:

the arcs in the positive direction are not saturated, and the arcs in the negative direction have a flow strictly greater than their lower bound.

If such a chain does not exist END.

4) Use this chain to improve the flow, by adjusting it to satisfy the bound constraint on arc **u**, then return to step (2).

Chapter IV. Pathway problems

This chapter is devoted to the study of *pathway problems* in graph theory, which consist of determining specific structures or properties within a graph by following its vertices and edges according to well-defined rules. These problems are fundamental in various applications such as network design, communication systems, transportation, and optimization.

In the first section**,** we focus on the search for connected components, which aims to identify all subsets of vertices where each pair of vertices is connected by a path. The Trémeaux–Tarjan algorithm is introduced as an efficient method for detecting both connected and strongly connected components in undirected and directed graphs, respectively.

The **second section** deals with the problem of **finding the shortest path** between vertices in a weighted graph. After presenting the main conditions of the problem, we describe the **Moore–Dijkstra algorithm**, which provides an optimal and systematic way to compute the minimum distance from a given source vertex to all others.

Finally, the **third section** addresses the construction of a **spanning tree of maximum weight** in a connected graph. The objective is to select a subset of edges that connects all vertices while maximizing the total weight. For this purpose, the **Kruskal algorithm (1956)** is presented as a classical and effective approach.

Through these three parts, the chapter offers a unified understanding of fundamental graph traversal and optimization techniques that form the basis of many modern computational applications.

2. Search for connected components

1.1. Presentation of objectives:

The study of graph connectivity represents one of the most fundamental topics in graph theory. In many applications, it is sufficient to restrict attention to a single strongly connected component, since any graph can be analyzed component by component.

A variety of algorithms have been proposed to determine the set of vertices that belong to the same strongly connected component. In this chapter, we present a simple algorithm designed to identify the strongly connected component containing a specified vertex, following the works of Trémeaux (1882) and Tarjan (1972).

1.2. Trémeaux-Tarjan algorithm:

Principle of the Algorithm:

Let **G** be a directed graph. The main idea of the Trémeaux–Tarjan algorithm is to perform a *depth-first traversal* of **G** starting from a given vertex **x**, exploring both in the *forward direction* (following the orientation of the arcs) and in the *reverse direction* (following the arcs in the opposite orientation. The arcs traversed during the depth-first search form a tree.

The depth-first search (DFS) is implemented by managing the list of vertices to be visited as a stack (LIFO – Last In, First Out) structure. At each step, the algorithm considers the most recently discovered (marked) vertex at the top of the stack. From this vertex, it attempts to visit one of its *unmarked successors* (in the case of forward traversal) or *unmarked predecessors* (in the case of backward traversal). Each time such a vertex is found, it is marked and pushed onto the stack, and the search continues deeper into the graph. If no unmarked successor (or predecessor) exists, the algorithm *backtracks* by removing the top vertex from the stack and returning to the previous vertex. This process continues until no further vertex can be marked.

The first traversal identifies the set of vertices reachable from **x**, while the second traversal determines the set of vertices that can reach **x**. The intersection of these two sets forms the collection of vertices that are both reachable from and can

reach **x**; this set corresponds to the strongly connected component containing the vertex **x**.

Algorithm:

Input: A directed graph $\mathbf{G} = (\mathbf{X}, \mathbf{U})$.

Output: The number $\mathbf{k}$ of strongly connected components of $\mathbf{G}$, and the list $\mathbf{C}_1, \mathbf{C}_2, ..., \mathbf{C}_k$ of these components.

Step 1 – Initialization

1. Initialize $\mathbf{k} = 0$ and create an empty stack $\mathbf{S}$.
2. Choose an arbitrary unmarked vertex $\mathbf{x}$ of $\mathbf{X}$.

Step 2 – Forward Depth-First Search (DFS)

1. Mark $\mathbf{x}$ with a (**+**) sign and push it onto stack $\mathbf{S}$.
2. While $\mathbf{S}$ is not empty, do:
   a) Let $\mathbf{v}$ be the vertex on the top of the stack.
   b) If there exists an unmarked successor $\mathbf{u}$ of $\mathbf{v}$ in $\mathbf{G}$, then mark $\mathbf{u}$ with a (**+**) sign and push $\mathbf{u}$ onto stack $\mathbf{S}$.
   c) Otherwise, pop $\mathbf{v}$ from stack $\mathbf{S}$.
3. Let **ForwardSet** = {all vertices marked with (**+**)}
4. Unmark all vertices.

Step 3 – Backward Depth-First Search (DFS)

1. Mark $\mathbf{x}$ with a (**–**) sign and push it onto stack $\mathbf{S}$.
2. While $\mathbf{S}$ is not empty, do:
   a. Let $\mathbf{v}$ be the vertex on the top of the stack.
   b. If there exists an unmarked predecessor $\mathbf{u}$ of $\mathbf{v}$ in $\mathbf{G}$, then mark $\mathbf{u}$ with a (**–**) sign and push $\mathbf{u}$ onto stack $\mathbf{S}$.
   c. Otherwise, pop $\mathbf{v}$ from stack $\mathbf{S}$.
3. Let **BackwardSet** = {all vertices marked with (**–**)}.
4. Unmark all vertices.

Step 4 – Identification of the Strongly Connected Component

1. Increment $\mathbf{k} = \mathbf{k} + 1$.
2. Define $\mathbf{C}_k$ **= ForwardSet • BackwardSet.**
3. Remove the vertices of $\mathbf{C}_k$ from $\mathbf{X}$, and let $\mathbf{X} = \mathbf{X} \setminus \mathbf{C}_k$.

4. If $\mathbf{X}$ is not empty, let $\mathbf{G}'$ be the subgraph of $\mathbf{G}$ induced by the remaining vertices. Set $\mathbf{G} = \mathbf{G}'$ and repeat from Step 2.
5. Otherwise, terminate.

Step 5 – Output

The number of strongly connected components of $\mathbf{G}$ is $\mathbf{k}$.
Each set $\mathbf{C_i}$, $\mathbf{i} = 1,\ldots, \mathbf{k}$, corresponds to the set of vertices forming a strongly connected component of $\mathbf{G}$.

Example:

We apply the previous marking algorithm to determine the strongly connected components of the following graph:

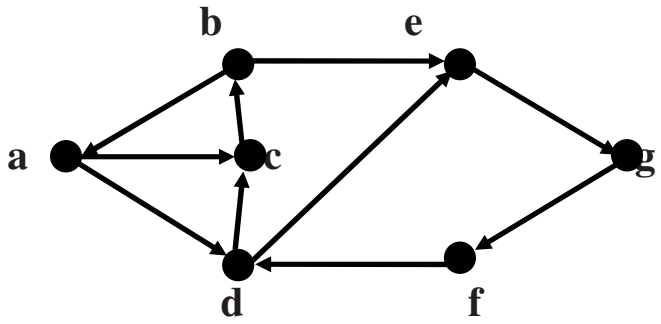


Figure 1: Search for strongly connected components.

We begin by setting $\mathbf{k} = 0$ and $\mathbf{S} = <>$.
From the set $\mathbf{X}$, we select vertex $\mathbf{a}$, mark it with the symbol (+), and push it onto the stack $\mathbf{S}$. Thus, $\mathbf{S} = < \mathbf{a} >$.
Since $\mathbf{S}$ is not empty, the vertex on the top of the stack is $\mathbf{a}$. Vertex $\mathbf{a}$ has two unmarked successors, $\mathbf{c}$ and $\mathbf{d}$. During the forward depth-first traversal, we arbitrarily choose vertex $\mathbf{d}$, mark it with (+), and push it onto the stack. We then obtain $\mathbf{S} = < \mathbf{a}, \mathbf{d} >$. Note that there is no particular preference when choosing between $\mathbf{c}$ and $\mathbf{d}$.
Now, $\mathbf{d}$ is at the top of the stack and has one unmarked successor. We mark this successor with (+) and push it onto the stack, giving $\mathbf{S} = < \mathbf{a}, \mathbf{d}, \mathbf{e} >$. Similarly, for vertex $\mathbf{e}$, we obtain $\mathbf{S} = < \mathbf{a}, \mathbf{d}, \mathbf{e}, \mathbf{f} >$.

At this stage, vertex **f** has no unmarked successors, so we pop **f** from the stack, yielding $\mathbf{S} =< \mathbf{a},\mathbf{d},\mathbf{e} >$.

For the same reason, we pop vertex **e**, obtaining $\mathbf{S} =< \mathbf{a},\mathbf{d} >$. Returning to vertex **d**, we find that it has one remaining unmarked successor, **c**. We then push **c** onto the stack, resulting in $\mathbf{S} =< \mathbf{a},\mathbf{d},\mathbf{c} >$.

The forward depth-first traversal continues as follows: $\mathbf{S} = \langle \mathbf{a},\mathbf{d},\mathbf{c},\mathbf{b} \rangle$, $\mathbf{S} = \langle \mathbf{a},\mathbf{d},\mathbf{c} \rangle$, then $\mathbf{S} = \langle \mathbf{a},\mathbf{d} \rangle$, $\mathbf{S} = \langle \mathbf{a} \rangle$, and finally $\mathbf{S} = \langle\ \rangle$.

The process terminates, and we obtain the **ForwardSet = X**.

During the backward depth-first traversal, the stack evolves as follows:

$\mathbf{S} = \langle\ \rangle$, $\mathbf{S} = \langle \mathbf{a} \rangle$, $\mathbf{S} = \langle \mathbf{a},\mathbf{b} \rangle$, $\mathbf{S} = \langle \mathbf{a},\mathbf{b},\mathbf{c} \rangle$, $\mathbf{S} = \langle \mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d} \rangle$, $\mathbf{S} = \langle \mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d},\mathbf{f} \rangle$, $\mathbf{S} = \langle \mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d},\mathbf{f},\mathbf{e} \rangle$, then $\mathbf{S} = \langle \mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d},\mathbf{f} \rangle$, $\mathbf{S} = \langle \mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d} \rangle$, $\mathbf{S} = \langle \mathbf{a},\mathbf{b},\mathbf{c} \rangle$, $\mathbf{S} = \langle \mathbf{a},\mathbf{b} \rangle$, $\mathbf{S} = \langle \mathbf{a} \rangle$, and finally $\mathbf{S} = \langle\ \rangle$.

Thus, the **BackwardSet = X**.

We obtain the first strongly connected component:

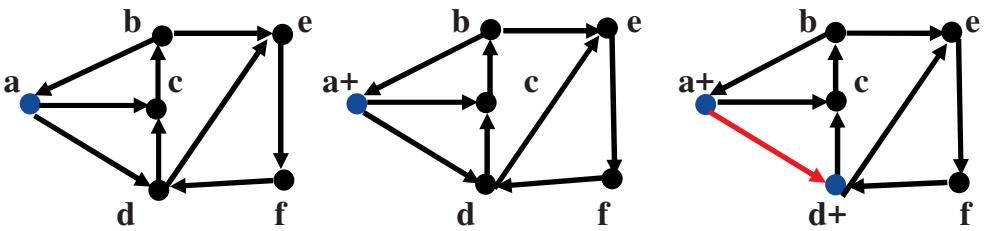
$\mathbf{C}_1 =$ ForwardSet $\cap$ BackwardSet = X.

After removing from **X** the vertices belonging to C_1, we obtain: $X = X \backslash C_1 = \varnothing$.

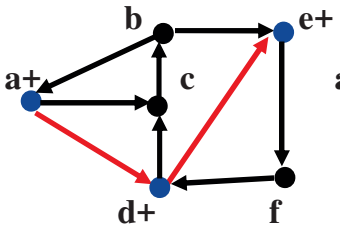
The process is thus completed. The graph **G** contains one strongly connected components. It follows that the graph is strongly connected.

The following figures present, in a stepwise manner, the marking procedure and the associated stack configurations observed during the forward and backward depth-first traversals:

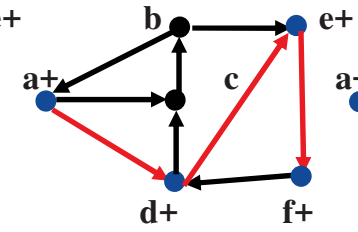
1) **<u>Forward depth-first traversals:</u>**



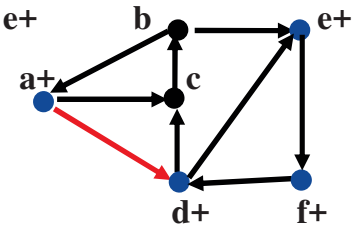
$\mathbf{S} = \langle\ \rangle$ $\mathbf{S} = \langle \mathbf{a} \rangle$ $\mathbf{S} = \langle \mathbf{a},\mathbf{d} \rangle$



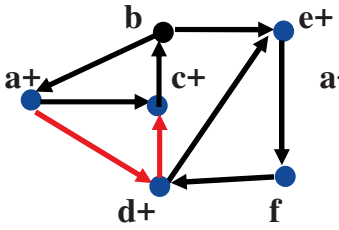
$S = \langle a, d, e \rangle$



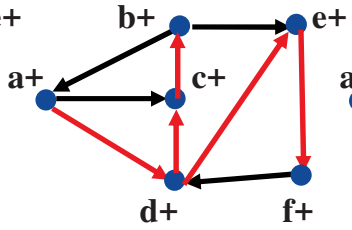
$S = \langle a, d, e, f \rangle$



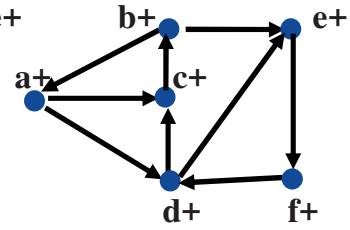
$S = \langle a, d \rangle$



$S = \langle a, d, c \rangle$

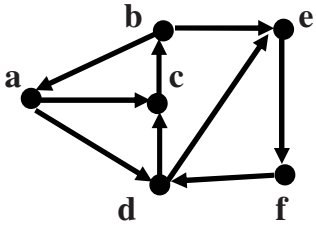


$S = \langle a, d, c, b \rangle$

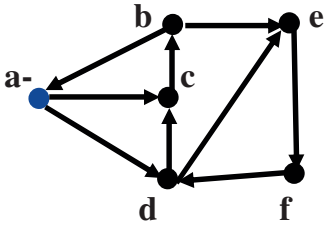


$S = \langle \ \rangle$

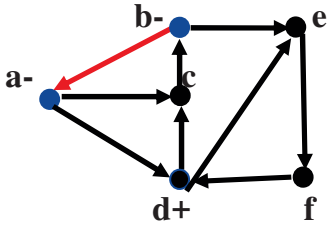
2) **Backward depth-first traversals:**



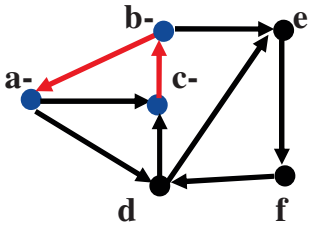
$S = \langle \ \rangle$



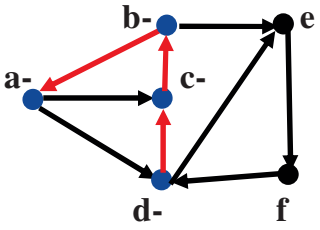
$S = \langle a \rangle$



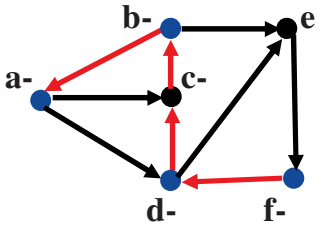
$S = \langle a, b \rangle$



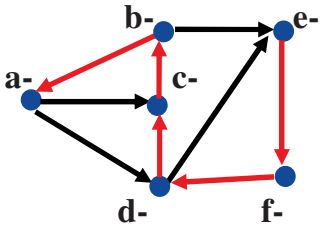
$S = \langle a, b, c \rangle$



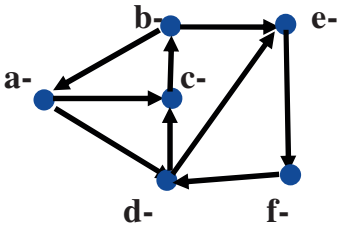
$S = \langle a, b, c, d \rangle$



$S = \langle a, b, c, d, f \rangle$



$S = \langle a, b, c, d, f, e \rangle$



$S = \langle \ \rangle$

2. Finding the shortest path

2.1. Presentation of the conditions

Let $\mathbf{G}$ be a graph in which each arc $\mathbf{u}$ is assigned a length $\mathbf{C(u)} \geq 0$.

Given two vertices $\mathbf{s}$ and $\mathbf{k}$ of $\mathbf{G}$, the goal is to find a path μ from $\mathbf{s}$ to $\mathbf{k}$ such that the sum of the lengths of its arcs is as small as possible.

The path μ is then called the *shortest path* from $\mathbf{s}$ to $\mathbf{k}$.

2.2 Moore-Dijkstra algorithm

This algorithm is applied to determine a *shortest-distance spanning tree* on a transport network $\mathbf{R = (X, U, C)}$, where the arc lengths are non-negative, that is, $\mathbf{C(u)} \geq 0$ for every $\mathbf{u} \in \mathbf{U}$.

Principle of the algorithm:

The idea of the Moore (1957) and Dijkstra (1959) algorithm is to progressively compute the shortest-distance spanning tree from a given starting vertex $\mathbf{s}$ to any other vertex $\mathbf{k}$. A distinctive feature of this algorithm is that the distances are introduced in increasing order.

Algorithm:

Input:

A a transport network $\mathbf{R = (X, U, C)}$, where $\mathbf{C(u)} \geq 0, \quad \forall u \in U$.

A source vertex $\mathbf{s} \in \mathbf{X}$.

Output:

The shortest paths from $\mathbf{s}$ to every other vertex of the graph.

Step 1:

1. Assign to each vertex $\mathbf{x} \in \mathbf{X}$ a label $\mathbf{d(x)}$ representing the current shortest known distance from from the starting vertex $\mathbf{s}$ to $\mathbf{x}$.
2. Set $\mathbf{d(s)} = 0$ and $\mathbf{d(x)} = +\infty$ for all other vertices.
3. Let $\mathbf{S}$ be the set of *unvisited* vertices, initially $\mathbf{S} = \{\mathbf{s}\}$.
4. Let $\mathbf{A}$ be the list of vertices included in the shortest-path spanning tree, initially $\mathbf{A} = \varnothing$.
5. Let α the the latest introduced vertex in $\mathbf{A}$. Set $\alpha = \mathbf{s}$.

Step 2:

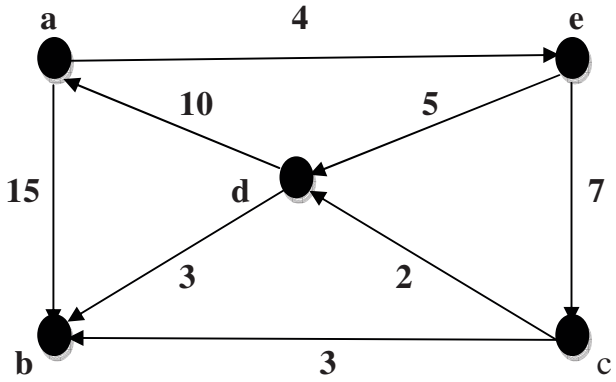
Examine all arcs $\mathbf{u}=(\alpha,\mathbf{y})$ such that the initial vertex is α and the terminal vertex $\mathbf{y}$ is not yet included in the set $\mathbf{A}$. If the condition $\mathbf{d}(\alpha)+\mathbf{C}(\mathbf{u})<\mathbf{d}(\mathbf{y})$ is satisfied, update the label of vertex $\mathbf{y}$ by setting $\mathbf{d}(\mathbf{y})=\mathbf{d}(\alpha)+\mathbf{C}(\mathbf{u})$.

Step 3:

Choose a vertex $\mathbf{z}\notin\mathbf{A}$ such that $\mathbf{d}(\mathbf{z})=\min\{\mathbf{d}(\mathbf{y}):\ \mathrm{y}\notin\mathrm{A}\}$.

1. If $\mathbf{d}(\mathbf{z})=\infty$, stop. The vertex $\mathbf{s}$ is not a root in $\mathbf{R}$.
2. If $\mathbf{d}(\mathbf{z})<\infty$, set $\alpha=\mathbf{z}$ and $\mathbf{A}:=\mathbf{A}\cup\{\alpha\}$.
3. If $\mathbf{A}=\mathbf{X}$, stop. $\mathbf{A}$ defines the shortest-path tree rooted at $\mathbf{s}$.
4. If $\mathbf{A}\neq\mathbf{X}$, return to Step 2.

Example:



Initialization:

Vertex **x**	**a**	**b**	**c**	**d**	**e**
d(x)	0	∞	∞	∞	∞

$\mathbf{A}=\{\mathbf{a}\}$, $\mathbf{T}=\varnothing$, and $\alpha=\mathbf{a}$.

1) Update of distances

We examine the two arcs **(a,b)** and **(a,e)**:

d(a) + C(a,b) = 15 < d(b) = ∞ ⇒ d(b) = 15.

d(a) + C(a,e) = 4 < d(e) = ∞ ⇒ d(e) = 4.

Vertex **x**	**a**	**b**	**c**	**d**	**e**
d(x)	0	15	∞	∞	4

Selection

d(e) = min { d(b), d(c), d(d), d(e)}

Update

α = e, A = {a, e}, $\mathbf{T} = \{(\mathbf{a},\mathbf{e})\}$.

2) Update of distances

We examine the two arcs **(e,c)** and **(e,d)**:

d(e) + C(e,c) = 11 < d(c) = ∞ ⇒ d(c) = 11.

d(e) + C(e,d) = 9 < d(d) = ∞ ⇒ d(d) = 9.

Vertex **x**	**a**	**b**	**c**	**d**	**e**
d(x)	0	15	11	9	4

Selection

d(d) = min{d(b), d(c), d(d)}.

Update

α = 4, A={a, e, d}, $\mathbf{T} = \{(\mathbf{a},\mathbf{e}),(\mathbf{e},\mathbf{d})\}$.

3) Update of distances

We examine arc **(d,b)**:

d(d) + C(d,b) = 12 < d(b) = 15 ⇒ d(b) = 12.

Vertex **x**	**a**	**b**	**c**	**d**	**e**
d(x)	0	12	11	9	4

Selection

$d(c) = \min\{d(b), \pi(c)\}$.

Update

$\alpha = c$, $A=\{a,e,d,c\}$, $T = \{(a,e),(e,d),(e,c)\}$.

4) Update of distances

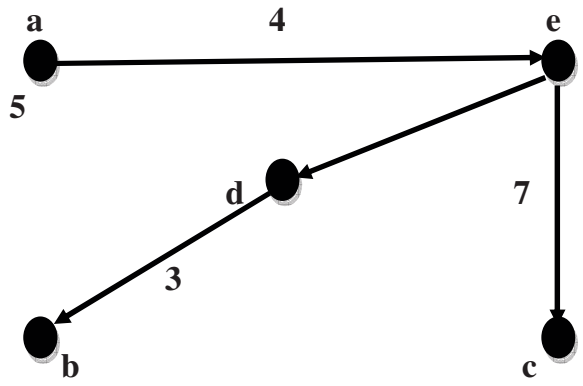
We examine arc **(c,b)**:

$d(c)+C(c,b) = 14 > d(b) = 12$.

Update

$\alpha = b$, $A = \{a,e,d,c,b\}$, $T = \{(a,e),(e,d),(e,c),(d,b)\}$.

The shortest-path tree originating from vertex **a** is



3. Search for a spanning tree of maximum weight in a graph

3.1. Presentation of objectives

Each edge of the graph $G = (X, U)$ is assigned a value (or weight). The minimum spanning tree problem consists in finding a subgraph that is a tree, such that the sum of the weights of its edges is minimal.

3.2. Kruskal's Algorithm (1956):

Principle of the algorithm:

The idea of Kruskal's algorithm is first to sort the edges in increasing order of their weights. Then, the algorithm gradually constructs the tree $\mathbf{A}$ by adding edges one by one in that order.

An edge is added only if its inclusion does not create a cycle in $\mathbf{A}$. Otherwise, the algorithm skips that edge and continues with the next one in the sorted list.

Algorithm

Input:

Weighted multigraph $\mathbf{G}=(\mathbf{X},\mathbf{U},\mathbf{C})$.

Output:

A tree or forest $\mathbf{A}=(\mathbf{V},\mathbf{W})$ of minimum total weight.

Step 1

Sort and number the edges of $\mathbf{G}$ in non-decreasing order of their weights: $\mathbf{C}(\mathbf{u}_1)\le\mathbf{C}(\mathbf{u}_2)\le\ldots\le\mathbf{C}\le(\mathbf{u}_\mathbf{m})$

Step 2

$\mathbf{W}=\varnothing$, $\mathbf{k}=0$.

Step 3

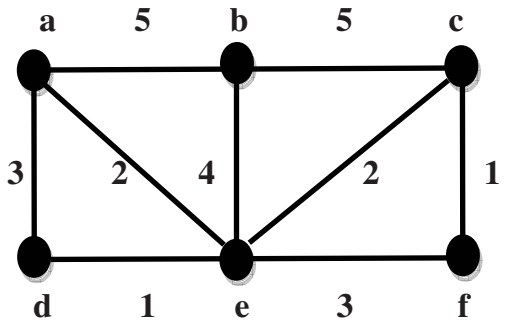
While $\mathbf{k}<\mathbf{m}$ and $|\mathbf{W}|<\mathbf{m}-1$ do:

If $\mathbf{u}_{\mathbf{k}+1}$ does not create a cycle with the edges in $\mathbf{W}$, then:

- $\mathbf{W}=\mathbf{W}\cup\{\mathbf{u}_{\mathbf{k}+1}\}$.
- $\mathbf{k}=\mathbf{k}+1$.

Example:

Let the following Weighted multigraph $\mathbf{G}=(\mathbf{X},\mathbf{U},\mathbf{C})$:



Sorted edges by increasing weight:

C(d,e)≤C(c,f)≤C(a,e)≤C(c,e)≤C(a,d)≤C(e,f)≤C(b,e)≤C(a,b)≤C(b,c)

Construction process:

W = ∅

W = {(d,e)}

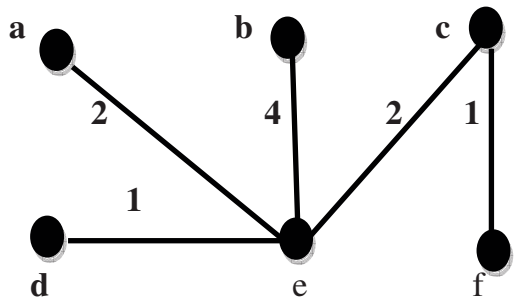
W = {(d,e), (c,f)}

W = {(d,e), (c,f), (a,e)}

W = {(d,e), (c,f), (a,e), (c,e)}

W = {(d,e), (c,f), (a,e), (c,e), (b,c)}

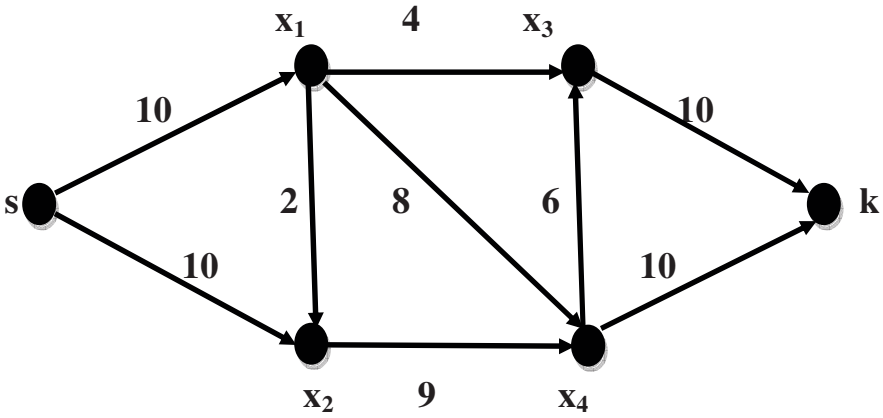
The final set W represents the minimum spanning tree of (G):



Tutorial Session (6)

Exercise 1:

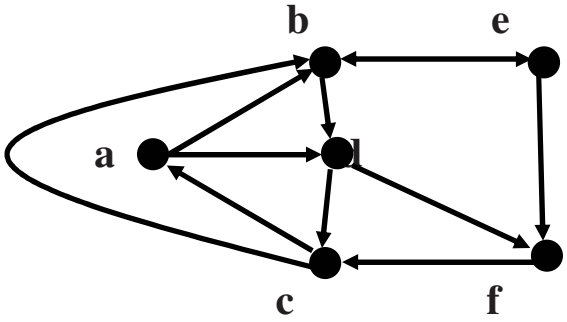
Consider the following transportation flow network:



Find the maximum flow that can be sent from the source **s** to the sink **k**

Exercise 2:

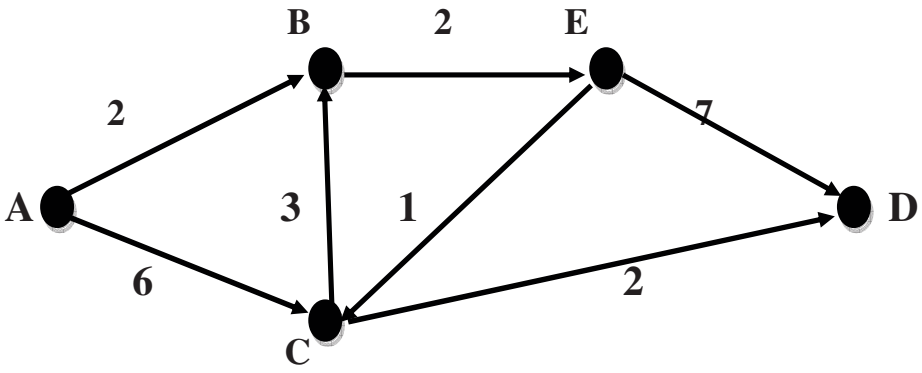
Let the following graph $\mathbf{G}=(\mathbf{X},\mathbf{U})$



Using Trémeaux-Tarjan algorithm, determine the strongly connected components of $\mathbf{G}$.

Exercise 3:

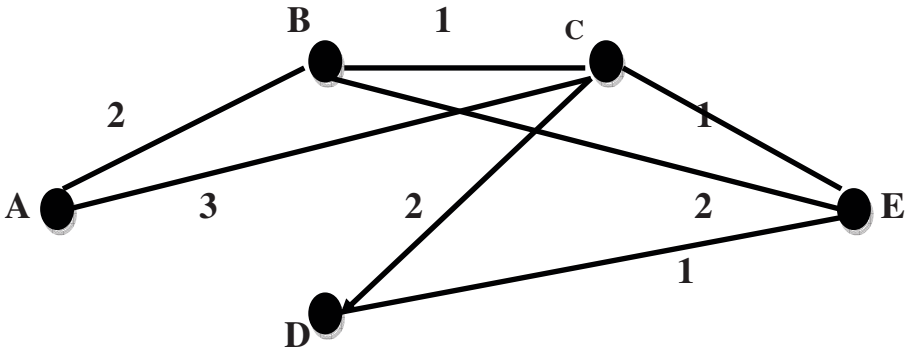
Consider the following transportation flow network:



Find the shortest paths from **A** to every other vertex of the graph.

Exercise 4:

Consider the following transportation flow network:

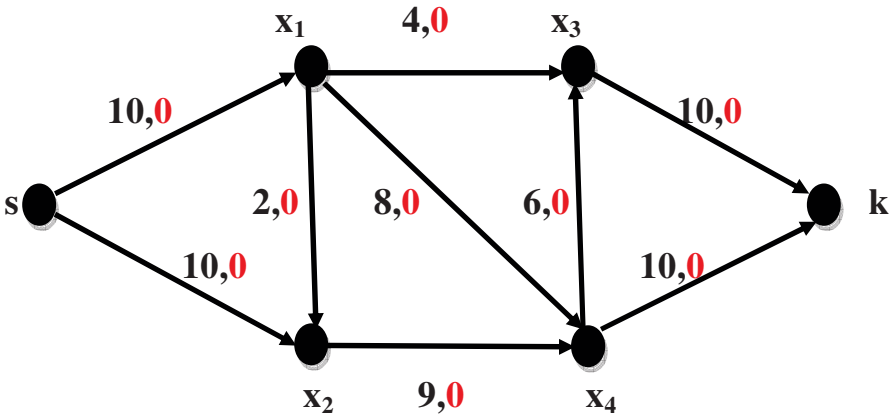


Find a spanning tree of maximum weight.

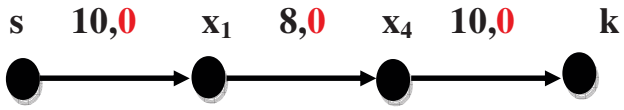
Tutorial Session 6 (Solution)

Exercise 1:

Initialize all arc flows to zero:

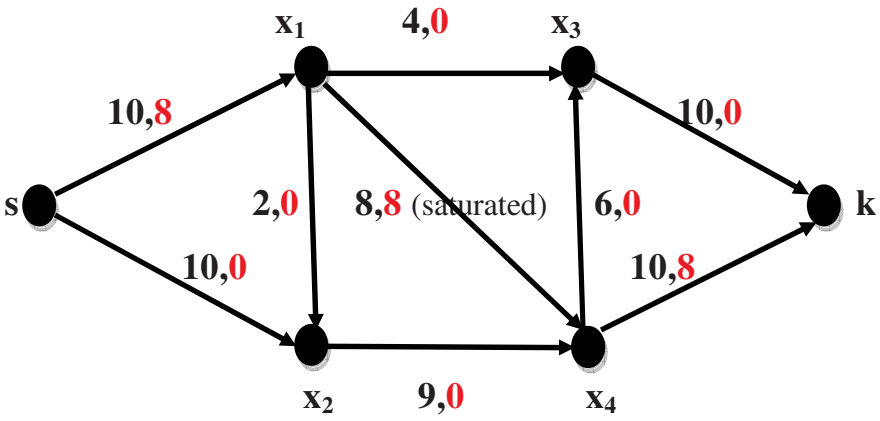


Find an **augmenting chain** between the source **s** and the sink **k** along which the flow can be increased. For example, we select the following chain:

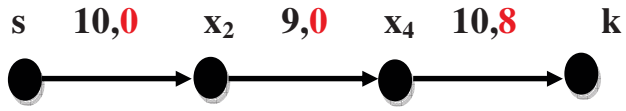


Calculate: $\mathbf{y} = \min\{\mathbf{C}(\mathbf{u}) - \mathbf{F}(\mathbf{u}) \mid \mathbf{u} \in \mu^+\} = \min\{10-0, 8-0, 10-0\} = 8$.

We improve the flow along this chain by adding the quantity **8** to the flow of the arcs in μ^+. The flow of the arcs that do not belong to the chain remains unchanged.

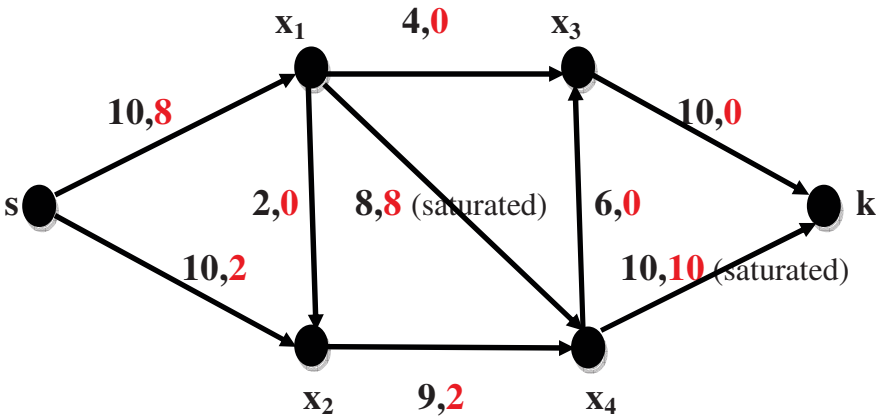


Now, find an augmenting chain in the residual network. We select:

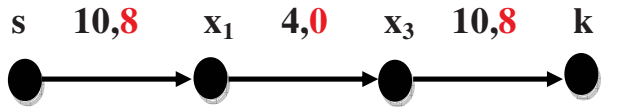


Calculate: $\mathbf{y} = \min\{\mathbf{C}(\mathbf{u}) - \mathbf{F}(\mathbf{u}) \mid \mathbf{u} \in \mu^+\} = \min\{10-0, 9-0, 10-8\} = 2$.

We increase the flow by **2** on the arcs of the chain, and all other arcs keep the same flow.

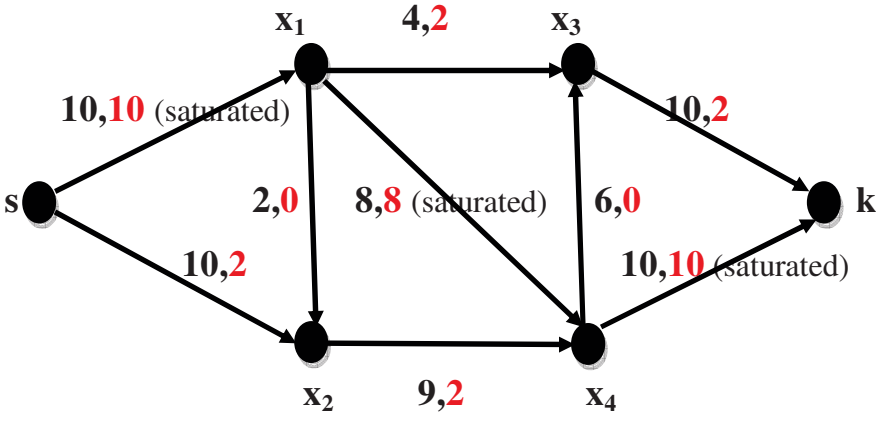


Select the augmenting chain:

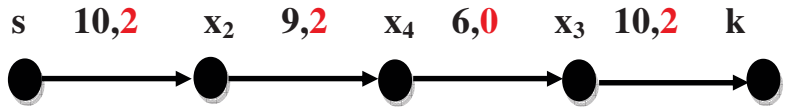


Calculate: $\mathbf{y} = \min\{\mathbf{C}(\mathbf{u}) - \mathbf{F}(\mathbf{u}) \mid \mathbf{u} \in \mu^+\} = \min\{10-8, 4-0, 10-0\} = 2$.

We increase the flow by **2** on the arcs of the chain:

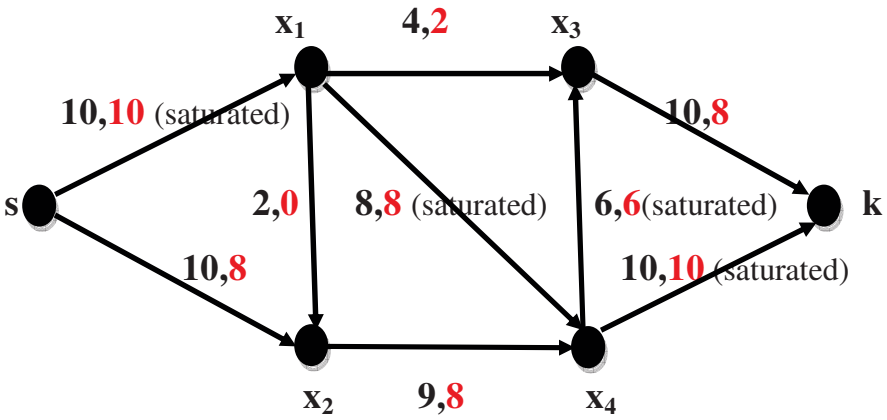


Select the augmenting chain:

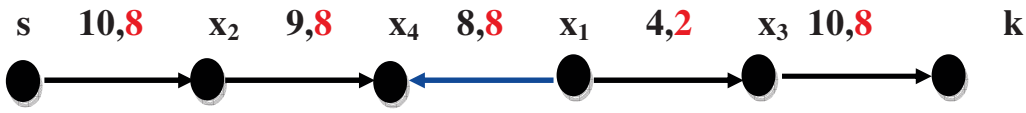


Calculate: $\mathbf{y} = \min\{\mathbf{C}(\mathbf{u}) - \mathbf{F}(\mathbf{u}) \mid \mathbf{u} \in \mu^+\} = \min\{10-2, 9-2, 6-0, 10-2\} = 6$.

We increase the flow by **6** on the arcs of the chain:



Select the augmenting chain:



Calculate:

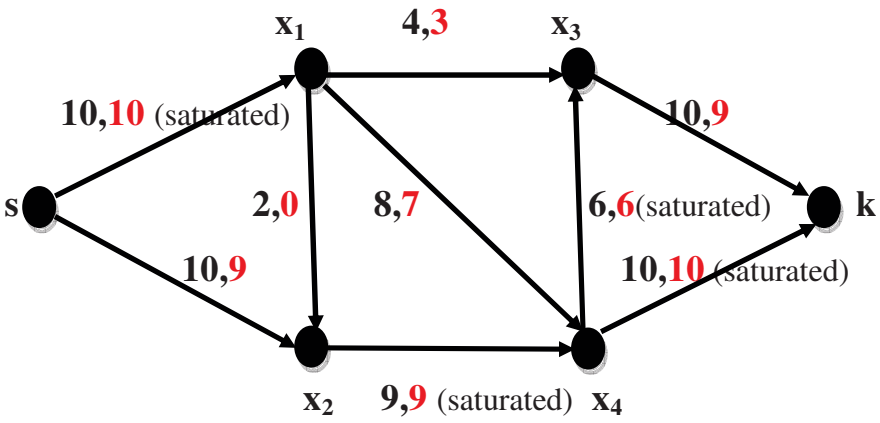
$\mathbf{y}_1 = \min\{\mathbf{C}(\mathbf{u}) - \mathbf{F}(\mathbf{u}) \mid \mathbf{u} \in \mu^+\} = \min\{10-8,\ 9-8,\ 4\text{-}2,\ 10\text{-}8\} = 1$

and

$\mathbf{y}_2 = \min\{\mathbf{F}(\mathbf{u}) \mid \mathbf{u} \in \mu^-\} = 8.$

Then $\mathbf{y} = \min\{\mathbf{y}_1\ ,\ \mathbf{y}_2\} = \min\{1\ ,\ 8\} = 1.$

We increase the flow by **1** on the forward arcs (μ^+) and decrease it by **1** on the backward arcs (μ^-).

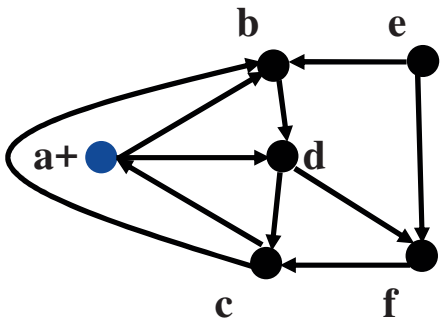


There is no augmenting chain in the residual graph. This means that the Ford–Fulkerson method has finished. Since the maximum flow is equal to the total flow leaving the source, in this example the maximum flow is **10+9 = 19**.

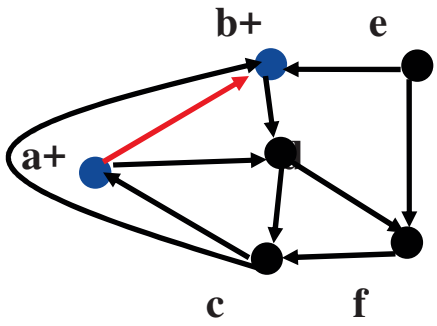
Exercise 2:

The following figures present, in a stepwise manner, the marking procedure and the associated stack configurations observed during the forward and backward depth-first traversals:

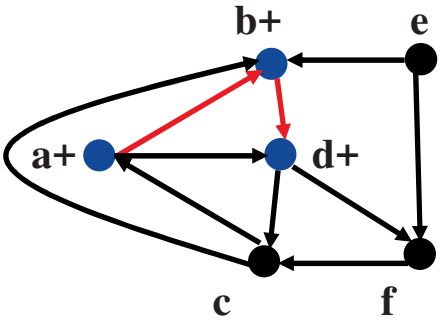
1) **Forward depth-first traversals:**



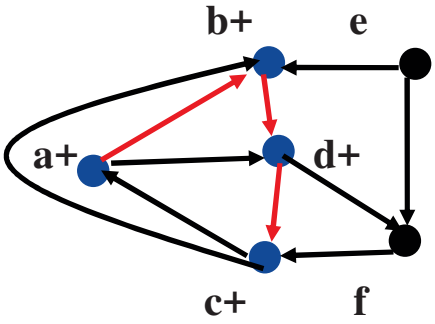
S = < a >



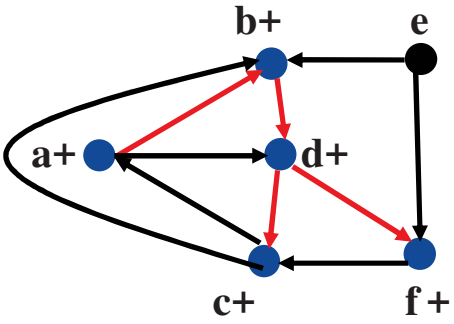
S = < a,b >



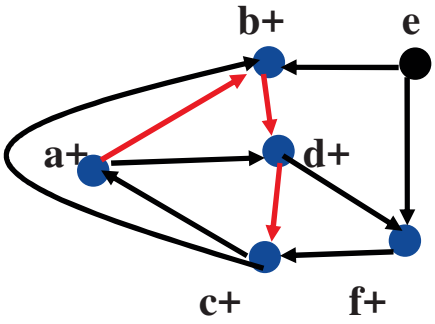
S = < a,b,d >



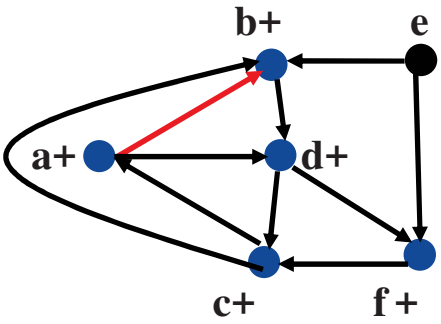
S = < a,b,d,c >



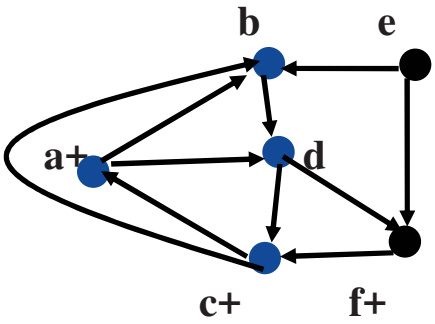
S = < a,b,d,f >



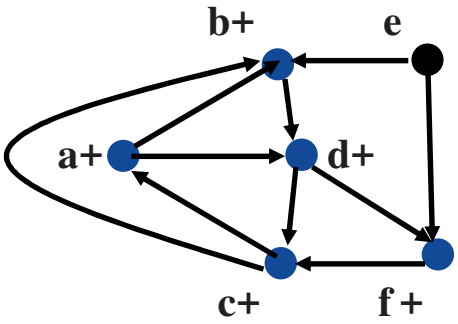
S = < a,b,d >



S = < a,b >



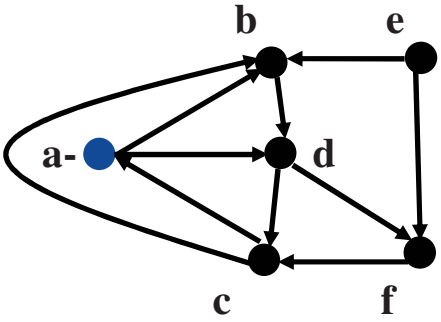
S = < a >



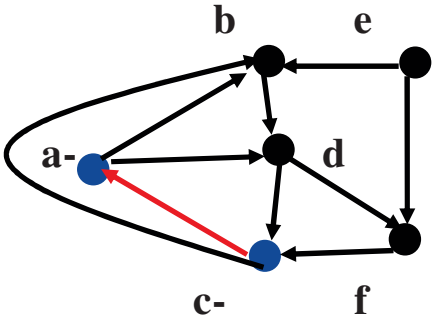
S = < >

Thus, ForwardSet={ a, b, c, d, f }.

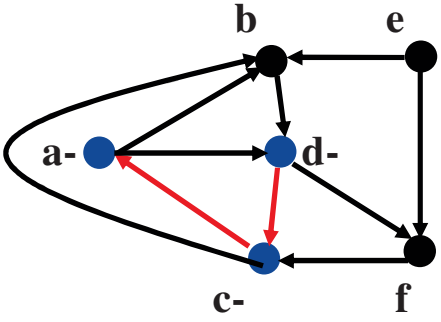
2) **Backward depth-first traversals:**



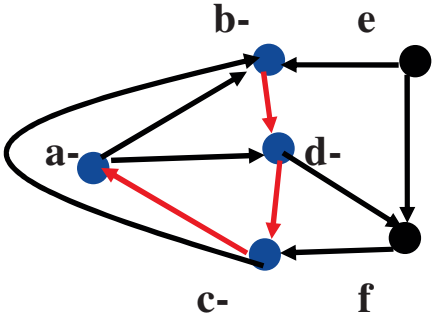
S = < a >



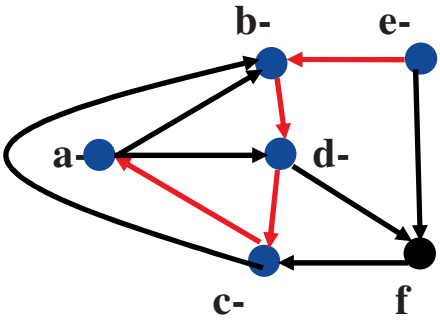
S = < a,c >



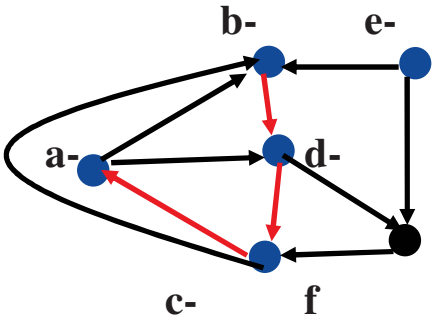
S = < a,c,d >



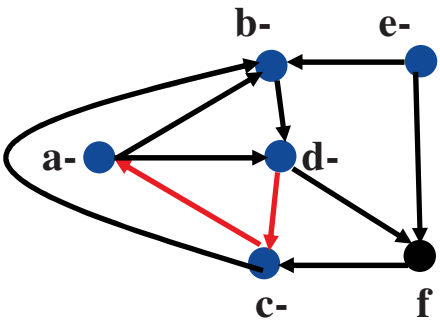
S = < a,c,d,b >



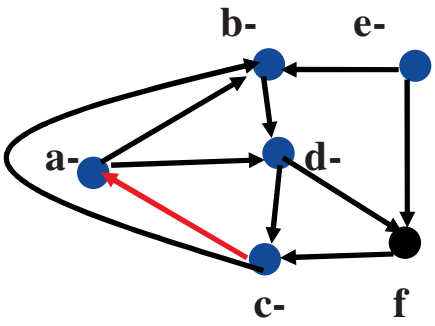
S = < a,c,d,b,e >



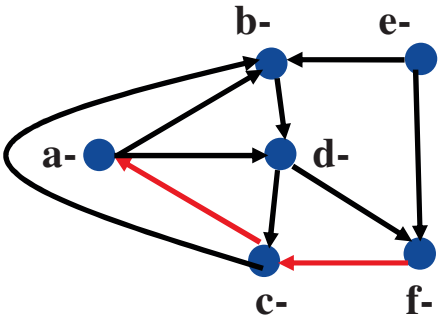
S = < a,c,d,b >



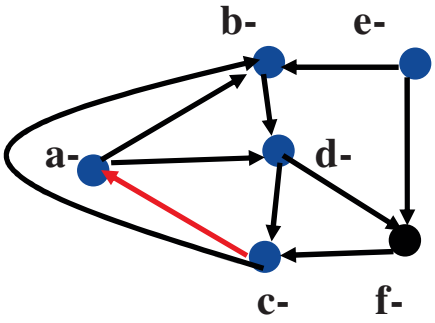
S = < a,c,d >



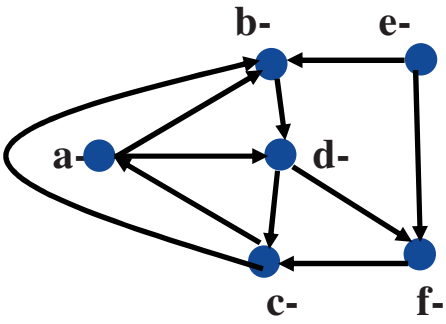
S = < a,c >



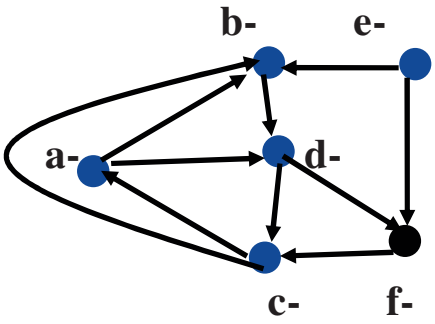
S = < a,c,f >



S = <a,c >



S = < a >



S = < >

Thus, the **BackwardSet ={ a, b, c, d, e, f }**.

Therefore, the first strongly connected component is:

$\mathbf{C_1}$ = ForwardSet ∩ BackwardSet

= **{ a, b, c, d, f }∩{ a, b, c, d, e, f }**

= **{ a, b, c, d, f }**.

We remove from **X** the vertices belonging to $\mathbf{C_1}$, which gives $\mathbf{X=X\setminus C_1=\{e\}}$.

All marks are then erased. We mark vertex **e** with the signs (+) and (–). Since there are no other vertices to be marked, we obtain $\mathbf{C_2=\{e\}}$.

Next, we remove from **X** the vertices of $\mathbf{C_2}$, resulting in $\mathbf{X=X\setminus C2=\emptyset}$. The process is therefore complete.

The number of strongly connected components of **G** is two, namely:

$\mathbf{C_1=\{a,b,c,d,f\}}$ and $\mathbf{C_2=\{e\}}$.

Exercise 3:

Initialization:

Vertex **x**	**A**	**B**	**C**	**D**	**E**
d(x)	0	∞	∞	∞	∞

S={A}, **T=∅**, $\alpha = \mathbf{A}$.

1) Update of distances

We examine the two arcs **(A,B)** and **(A,C)**:

d(A) + C(A,B) = 2 < d(B) = ∞ ⇒ d(B) = 2.

d(A) + C(A,C) = 6 < d(C) = ∞ ⇒ d(C) = 6.

Vertex **x**	**A**	**B**	**C**	**D**	**E**
d(x)	0	2	6	∞	∞

Selection

d(B) = min { d(B), d(C), d(D), d(E)}=2

Update

S = {A, B}, T={(A,B)}, α = b.

2) <u>Update of distances</u>

We examine the arc **(B,E)**:

$d(B) + C(B , E) = 4 < d(E) = \infty \Rightarrow d(E) = 4.$

Vertex **x**	**A**	**B**	**C**	**D**	**E**
d(x)	0	2	6	∞	4

<u>Selection</u>

$d(E) = \min\{d(C), d(D), d(E)\}.$

<u>Update</u>

$S=\{A, B, E\}, T=\{(A,B), (B,E)\}, \alpha = E,$.

3) <u>Update of distances</u>

We examine the two arcs **(E,C)** and **(E,D)**:

$d(E) + C(E,C) = 5 < d(C) = 6 \Rightarrow d(C) = 5.$

$d(E) + C(E,D) = 11 < d(D) = \infty \Rightarrow d(D) = 11.$

Vertex **x**	**A**	**B**	**C**	**D**	**E**
d(x)	0	2	5	11	4

<u>Selection</u>

$d(C) = \min\{d(C) , \pi(D)\}$.

<u>Update</u>

$A=\{A, B, E,C\}, T=\{(A,B), (B,E), (E,C)\}, \alpha = C$.

4) <u>Update of distances</u>

We examine the arc **(C,D)**:

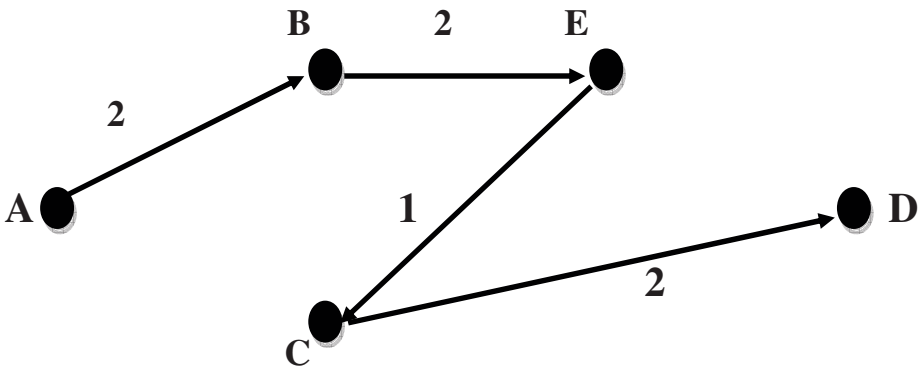
$d(C)+C(C,D) = 7 < d(D) = 11 \Rightarrow$ d(D) = 7.

Vertex **x**	**A**	**B**	**C**	**D**	**E**
d(x)	0	2	5	7	4

<u>Update</u>

A={A, B, E,C, D}, T={(A,B), (B,E), (E,C), (C,D)}.

The shortest-path tree originating from vertex **A** is:



<u>Exercise 3:</u>

Sorted edges by increasing weight:

C(B,C)≤C(D,E)≤C(D,E)≤C(C,D)≤C(B,E)≤C(A,B)≤C(A,C)

Construction process:

W = ∅

W = {(B,C)}

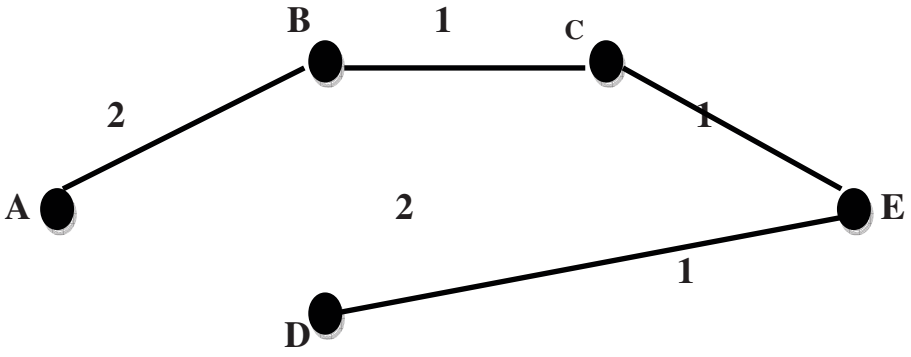
W = {(B,C), (C,E)}

W = {(B,C), (C,E), (E,D)}

W = {(B,C), (C,E), (E,D)}

W = {(B,C), (C,E), (E,D), (A,B)}

The final set W represents the minimum spanning tree of (G):



Chapter V: Hamiltonian and Eulerian Problems

In graph theory, two fundamental types of traversal problems are the *Hamiltonian* and *Eulerian* problems. Both deal with finding specific paths or cycles that visit the vertices or edges of a graph under certain conditions.

A Hamiltonian cycle is a closed path that visits each vertex of the graph exactly once before returning to the starting vertex. Determining whether such a cycle exists is a central and challenging question in graph theory, often associated with complex combinatorial structures. In this chapter, we present the main definitions and discuss both necessary and sufficient conditions for the existence of Hamiltonian paths and cycles.

On the other hand, an Eulerian cycle is a closed path that traverses every edge of the graph exactly once. The study of Eulerian graphs dates back to the 18th century with Euler's solution to the famous *Königsberg bridge problem.* We will recall the definitions, establish the necessary and sufficient conditions for the existence of an Eulerian cycle, and describe a simple local algorithm for its construction.

Finally, the chapter concludes with a discussion of the relations between Hamiltonian and Eulerian problems, highlighting the similarities and fundamental differences between these two classic graph-theoretic concepts.

1. Hamiltonian Problem

1.1. Definitions:

Let $\mathbf{G} = (\mathbf{X}, \mathbf{U})$ be a connected graph of order $\mathbf{n}$.

A *Hamiltonian path* (or *Hamiltonian chain*) is a path (or chain) that passes exactly once through each vertex of $\mathbf{G}$.

A Hamiltonian path is therefore an *elementary path* of length $\mathbf{n}-1$.

A *Hamiltonian circuit* (or *Hamiltonian cycle*) is a circuit (or cycle) that passes exactly once through each vertex of $\mathbf{G}$.

A Hamiltonian cycle is thus an *elementary cycle* of length $\mathbf{n}$.

An undirected graph is called *Hamiltonian* if it contains a Hamiltonian cycle. Similarly, a directed graph is called Hamiltonian if it contains a Hamiltonian circuit.

No polynomial-time solution is known for finding a Hamiltonian cycle. This problem is therefore very difficult. Moreover, there is no condition that is both necessary and sufficient for the existence of a Hamiltonian cycle. In addition, while many theorems concern the existence of a Hamiltonian cycle in a simple graph, only a few theorems address the existence of a Hamiltonian circuit in a directed graph.

Example:

The following graph has a Hamiltonian cycle: $(\mathbf{d},\mathbf{b},\mathbf{e},\mathbf{a},\mathbf{c},\mathbf{d})$.

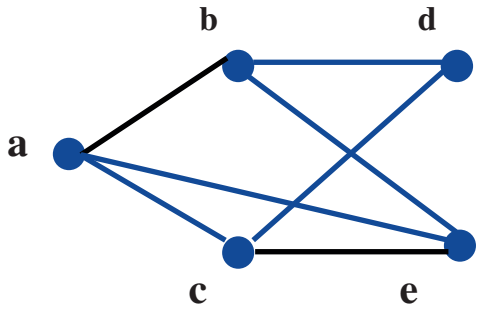


Figure 1: Hamiltonian cycle.

On the other hand, the following graph does not have a Hamiltonian cycle but has a Hamiltonian chain $(\mathbf{b},\mathbf{a},\mathbf{c},\mathbf{e},\mathbf{d})$.

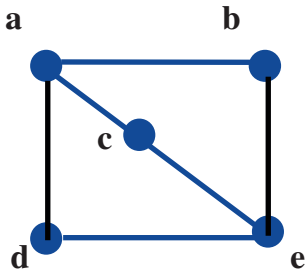


Figure 2: Hamiltonian chain.

1.2. Necessary condition for the existence of a Hamiltonian cycle:

Although no necessary and sufficient condition is known for determining whether a graph is Hamiltonian, several useful sufficient conditions provide guidance:

- If a multigraph **G** contains a vertex of degree 1, it cannot be Hamiltonian.
- For any simple multigraph **G** of order $\mathbf{n} \geq 3$, if every vertex has degree at least $\frac{\mathbf{n}}{2}$, then **G** is Hamiltonian (Dirac's Theorem).
- Every complete graph with at least three vertices is Hamiltonian.
- For an undirected graph to be Hamiltonian, it must be 2-connected; that is, the graph remains connected after the removal of any single vertex. The graph in the following figure is the smallest 2-connected non-Hamiltonian graph.

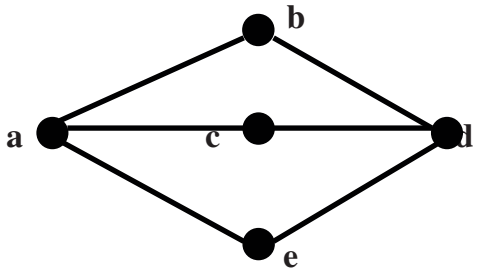


Figure 3: The smallest 2-connected non-hamiltonian graph.

- Every Hamiltonian graph contains a spanning subgraph of degree 2, meaning a partial graph that includes all the vertices of the original graph and in which each vertex is incident to exactly two edges (forming one or more cycles).

However, the existence of such a partial graph does not guarantee Hamiltonicity, as illustrated by the *Petersen graph*:

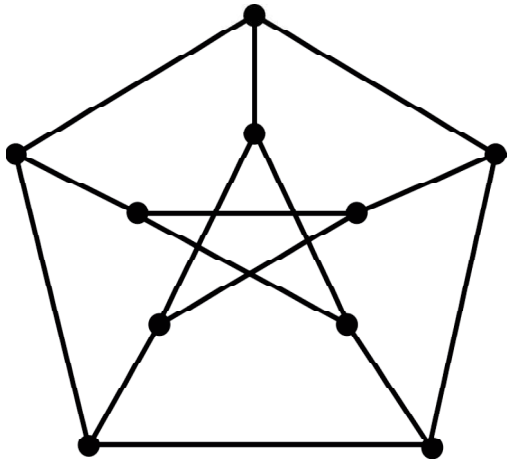


Figure 3: The Petersen graph is a well-known example of a non-Hamiltonian graph.

1.3. Sufficient condition for the existence of a Hamiltonian circuit:

Meyniel's Theorem (1973):

Let $\mathbf{G}$ be a loopless, strongly connected 1-graph of order $\mathbf{n}$. If, for every pair of non-adjacent vertices $\mathbf{x}$ and $\mathbf{y}$, the sum of their degrees satisfies $\mathbf{d}_G(\mathbf{x}) + \mathbf{d}_G(\mathbf{y}) \geq 2\mathbf{n} - 1$, then $\mathbf{G}$ contains a Hamiltonian circuit.

Corollary (Ghouila-Houri Theorem, 1960):

If a loopless, strongly connected 1-graph $\mathbf{G}$ of order $\mathbf{n}$ satisfies the condition $\mathbf{d}_G(\mathbf{x}) \geq 2\mathbf{n}$ for every vertex $\mathbf{x}$, then the graph contains a Hamiltonian circuit.

1.4. Sufficient condition for the existence of a Hamiltonian cycle:

Dirac's Theorem (1952):

A simple graph of order $\mathbf{n}$ that satisfies $\mathbf{d}_G(\mathbf{x}) \geq \frac{\mathbf{n}}{2}$ for every vertex $\mathbf{x}$, admits a Hamiltonian cycle.

This theorem is a restriction of Ore's theorem (1961).

Ore's Theorem (1961):

A simple graph of order $\mathbf{n}$ such that every pair of non-adjacent vertices $\mathbf{x}$ and $\mathbf{y}$ satisfies $\mathbf{d_G(x) + d_G(y) \geq 2n}$ admits a Hamiltonian cycle.

Tutte's Theorem (1956):

If a planar graph is 4-connected, then it contains a Hamiltonian cycle.

2. Eulerian Problem:

The Eulerian problem is one of the oldest combinatorial problems, as demonstrated by Euler's solution to the Seven Bridges of Königsberg problem (now Kaliningrad).

2.1. Definitions:

Let $\mathbf{G}$ be a connected multigraph of order $\mathbf{n}$. An *Eulerian chain* (or *Eulerian cycle*) is a chain (or cycle) that uses each edge of $\mathbf{G}$ exactly once. An Eulerian chain is therefore simple. A multigraph is called *Eulerian* if it contains an Eulerian cycle.

2.2. Necessary and sufficient condition for the existence of an Eulerian chain:

Euler's Theorem (1766):

A multigraph admits an Eulerian chain if and only if it is connected (except for isolated vertices) and if the number of vertices of odd degree is 0 or 2.

Corollary:

A multigraph admits an Eulerian cycle if and only if it is connected (except for isolated vertices) and has no vertices of odd degree.

2.3. Local algorithm to construct an Eulerian cycle:

Principle of the algorithm:

This algorithm aims to traverse **"in a single stroke"** all the edges of a connected multigraph in which all vertices have even degree.

Three traversal rules:

- Start from any vertex $\mathbf{a}$, and follow a chain without ever using the same edge twice.
- Upon reaching a vertex $\mathbf{x} \neq \mathbf{a}$ at the ***k***-th step, never take an edge that is a *bridge* in the graph $\mathbf{G_k}$ formed by the edges not yet used (except if $\mathbf{x}$ is an isolated vertex in $\mathbf{G_k}$). A bridge (also called an isthmus or cut-edge) is an edge whose removal increases the number of connected components of the graph.
- If you return to the starting vertex $\mathbf{a}$, continue along any unused edge if one exists; otherwise, stop.

Example:

Consider the following graph:

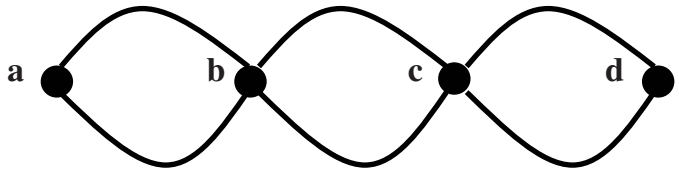


Figure 4: An Eulerian cycle

During traversal, some unused edges may become *bridges* in the partial graph formed by the remaining edges. When this happens, the algorithm avoids to use them by taking another way.

For example, we start at vertex **b**, so the current chain is $(\mathbf{b})$. We choose an unused edge from **b**, say **{b,c}**, obtaining the chain $(\mathbf{b},\mathbf{c})$. At this point, edge $\{\mathbf{c},\mathbf{b}\}$ becomes a bridge in the remaining graph, so we avoid using it for now. Instead, we select another edge, for example $\{\mathbf{c},\mathbf{d}\}$, and continue following the chain $(\mathbf{d},\mathbf{c},\mathbf{b},\mathbf{a},\mathbf{b})$ traversing unused edges without difficulty.

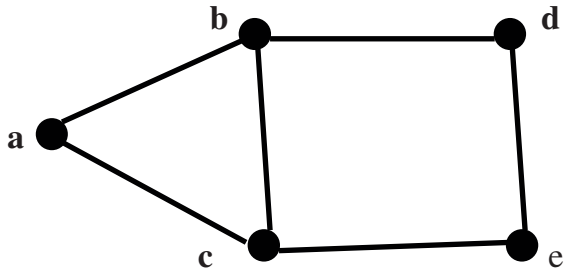
Finally, when we return to the starting vertex **b** and no unused edges remain, the algorithm stops. The resulting Eulerian cycle is $(\mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{c}, \mathbf{b}, \mathbf{a}, \mathbf{b})$.

2.4. Relation between Eulerian and Hamiltonian problem:

An Eulerian graph without subdivisions (i.e., without edges that have been divided by inserting extra vertices) is Hamiltonian, but the converse is obviously false.

Example:

Consider the following graph:



There is a cycle visiting all vertices exactly once: $(\mathbf{a},\mathbf{b},\mathbf{d},\mathbf{e},\mathbf{c},\mathbf{a})$. Therefore the graph is Hamiltonian. However, vertex **b** has degree 3 (odd). Thus the graph is not Eulerian.

This clearly shows that **Eulerian ≠ Hamiltonian**, but some Eulerian graphs without subdivisions can also be Hamiltonian.

Chapter VI. Colouring

Graph colouring is one of the fundamental topics in graph theory. It consists of assigning colours to certain elements of a graph (its vertices or its edges) subject to specific constraints. The main objective is to perform this colouring using the smallest possible number of colours while ensuring that adjacent elements do not share the same colour.

This chapter presents the essential concepts and results related to graph colouring.

In the first section, we introduce the basic definitions of vertex and edge colouring and the associated notions such as chromatic number and chromatic index.

The second section focuses on vertex colouring, which consists of partitioning the set of vertices into stable subsets.

The third section deals with arc (edge) colouring, where colours are assigned to the edges of the graph so that adjacent edges receive different colours.

The fourth section recalls some important propositions and theorems, including bounds on the chromatic number.

The fifth section presents the famous Four Colour Theorem, which applies to planar graphs and states that any planar map can be coloured with at most four colours.

Finally, the sixth section introduces the notion of perfect graphs, an important class of graphs in which the chromatic number equals the size of the largest clique for every induced subgraph.

1. **Definitions**:

Two types of colouring are defined: *vertex colouring* and *edge colouring*. The vertex colouring (respectively edge colouring) of a multigraph $\mathbf{G}$ corresponds to assigning a colour to each vertex (respectively each edge) in such a way that two adjacent vertices (respectively edges) do not share the same colour.

We call the *chromatic number* $\gamma(\mathbf{G})$ (respectively the *chromatic index* $\mathbf{q}(\mathbf{G})$) the minimum number of distinct colours required to perform a vertex colouring (respectively an edge colouring) of $\mathbf{G}$.

2. Vertex Colouring:

Let $\mathbf{G} = (\mathbf{X}, \mathbf{U})$ a multigraph. A subset of vertices of $\mathbf{X}$ is called a *stable set* if it contains only vertices that are pairwise non-adjacent.

The stability number $\alpha(\mathbf{G})$ is the maximum cardinality of a stable subset. A vertex colouring is therefore a partition of the vertex set into stable subsets.

$\mathbf{G}$ is said to be ***p**-chromatic* (or $\mathbf{q}$-colourable) if its vertices admit a colouring with $\mathbf{p}$ colours.

Property:

We can prove that: $\gamma(\mathbf{G}) \geq \dfrac{|\mathbf{X}|}{\alpha(\mathbf{G})}$.

Example:

In the graph shown below, the subset $\{\mathbf{a}, \mathbf{b}, \mathbf{e}\}$ forms a stable set of maximum cardinality. The minimum number of distinct colours required to achieve a proper vertex colouring of the graph is 3.

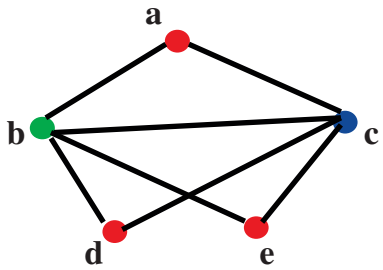


Figure 1: A minimum set stable and vertex colouring.

We have, $\gamma(\mathbf{G}) = 3 \geq \dfrac{|\mathbf{X}|}{\alpha(\mathbf{G})} = \dfrac{5}{3}$.

Brélaz (DSATUR) Algorithm (1979)

The **DSATUR** (Degree of Saturation) algorithm is a heuristic method for vertex colouring that aims to use the smallest possible number of colours.

Main idea:

At each step, the algorithm selects the *most constrained* uncoloured vertex, that is, the vertex whose neighbours already have the largest number of distinct colours (called its saturation degree). If several vertices have the same saturation degree, the one with the highest degree in the graph is chosen.

Principle:

1. Start by colouring the vertex with the highest degree using the first colour.
2. Compute the saturation degree of all uncoloured vertices.
3. Select the uncoloured vertex with the highest saturation degree (break ties by choosing the vertex with the highest degree).
4. Assign to this vertex the *smallest colour* not used by its neighbours. (The smallest colour refers to the first colour, in the ordered list of already used colours, that is not assigned to any neighbour of the vertex under consideration.

   This ensures that colours are reused whenever possible, avoiding unnecessary introduction of new colours and leading to a more efficient colouring of the graph).
5. Update the saturation degrees of adjacent uncoloured vertices.
6. Repeat steps 3–5 until all vertices are coloured.

Example:

Consider the following graph $\mathbf{G} = (\mathbf{X}, \mathbf{U})$

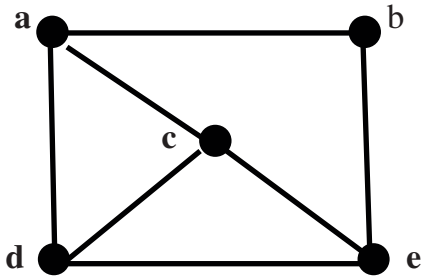


Figure 2: Vertex Colouring.

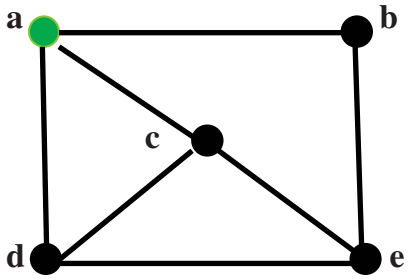
Step 1: Computation of vertex degrees

The degrees of the vertices are:

$\mathbf{d}_G(\mathbf{a}) = 3,\ \mathbf{d}_G(\mathbf{b}) = 2,\ \mathbf{d}_G(\mathbf{c}) = 3,\ \mathbf{d}_G(\mathbf{d}) = 3,\ \mathbf{d}_G(\mathbf{e}) = 3.$

Step 2: Initial colouring

Among the vertices of maximum degree, **a**, **c**, **d**, **e**, we select vertex **a** to start. Assign **Colour 1** (**green**) to vertex **a**.



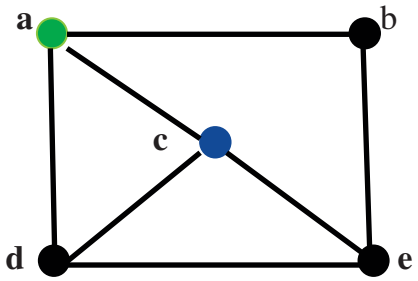
Step 3: Computation of saturation degrees

The saturation degree of a vertex is the number of distinct colours used in its neighbourhood. After colouring vertex **a**, its neighbours **b**, c, d each have one coloured neighbour, hence their saturation degree is 1.

Vertex **e** is not adjacent to **a**, so its saturation degree is 0.

Step 4: Selection of the next vertex

The vertices **b**, **c**, **d** have the highest saturation degree (equal to 1). To break the tie, we select the vertex with the highest degree. Choose vertex **c** (degree 3). Its neighbour **a** already uses Colour 1; therefore, assign Colour 2 (**blue**) to **c**.

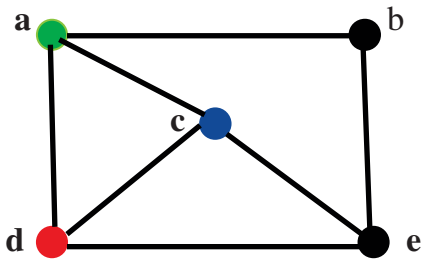


Step 5: Update of saturation degrees

- Vertex **d**, adjacent to both **a** (Colour 1) and **c** (Colour 2), now has two distinct colours in its neighbourhood, so its saturation degree becomes 2.
- Vertex **e**, adjacent to **c**, has one colour in its neighbourhood, so its saturation degree is 1.
- Vertex **b** remains with saturation degree 1.

Step 6: Next vertex selection

Vertex **d** has the highest saturation degree (2). Its neighbours are **a** (Colour 1), **c** (Colour 2), and **e** (uncoloured). The smallest available colour not used by its neighbours is **Colour 3**, thus assign Colour 3 (**red**) to vertex **d**.



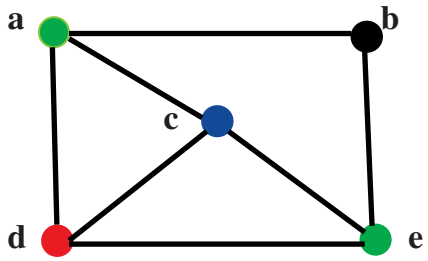
Step 7: Update of remaining vertices

- Vertex **b**: neighbours **a** (Colour 1) and **e** (uncoloured), then the saturation degree = 1.

- Vertex **e**: neighbours **b** (uncoloured), **c** (Colour 2), and **d** (Colour 3) then the saturation degree = 2.

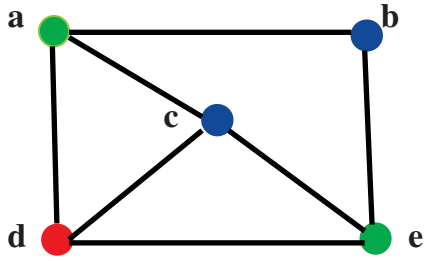
Step 8: Next vertex

Vertex **e** has the highest saturation degree (2). Its neighbours **c** and **d** use Colours 2 and 3 respectively, hence the smallest available colour is **Colour 1** (**green**). Assign Colour 1 to vertex **e**.



Step 9: Final vertex

Finally, only vertex **b** remains uncoloured. Its neighbours **a** and **e** both have Colour 1. Therefore, assign the next available colour: Colour 2 (**blue**) to **b**.



Step 10: Final colouring

The final vertex colouring is: $\mathbf{a}(1)$, $\mathrm{b}(2)$, $\mathrm{c}(2)$, $\mathrm{d}(3)$, $\mathrm{d}(1)$.

No two adjacent vertices share the same colour, and the colouring uses three colours in total. Thus, the *chromatic number* $\boldsymbol{\gamma}(\mathbf{G}) \leq 3$

3. **Edge Colouring:**

Let us first recall the notion of a *matching*. Let $\mathbf{G}$ be a multigraph without loops. A matching is defined to be a set $\mathbf{E}$, of edges such that no two edges of $\mathbf{E}$ are adjacent.

The *edge colouring* of $\mathbf{G}$ is the smallest integer $\mathbf{q}(\mathbf{G})$ having the following property: it is possible, with $\mathbf{q}(\mathbf{G})$ colours, to colour the edges of $\mathbf{G}$ so that two adjacent edges do not share the same colour.

In other words, an edge colouring is a partition of the set of edges into classes that are *matchings*.

The following edge colouring algorithm is a simple and efficient heuristic that performs well for small or practical graphs, although it does not always guarantee the use of the minimum possible number of colours.

Algorithm:

Input: A multigraph $\mathbf{G}=(\mathbf{X},\mathbf{U})$.

Output: A colouring of the edges of $\mathbf{G}$ such that no two adjacent edges share the same colour.

Step 1: List all edges of the graph in any order.

Step 2: Start with the first edge and assign it the first colour (Colour 1).

Step 3: For each next edge $\mathbf{e}$ in the list:

- Look at all edges adjacent to $\mathbf{e}$ that are already coloured.
- Choose the smallest colour (the first colour in order) that is not used by any adjacent edge.
- Assign that colour to $\mathbf{e}$.

Step 4:

Repeat Step 3 until all edges are coloured.

Example:

Consider the graph

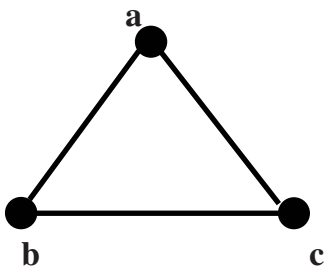


Figure 3: Edges Colouring.

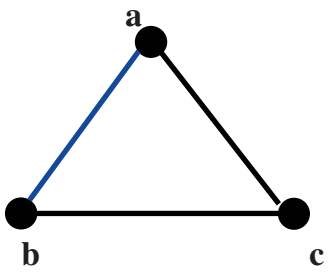
This is a *triangle graph* (a clique $\mathbf{K}_3$).

Step 1: List the edges

We list the edges in order : $\mathbf{e}_1 = \{\mathbf{a}, \mathbf{b}\}$, $\mathbf{e}_2 = \{\mathbf{b}, \mathbf{c}\}$, $\mathbf{e}_3 = \{\mathbf{c}, \mathbf{a}\}$.

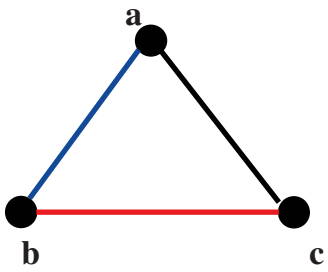
Step 2: Colour the first edge

Assign Colour 1 (blue) to edge $\mathbf{e}_1 = \{\mathbf{a}, \mathbf{b}\}$.



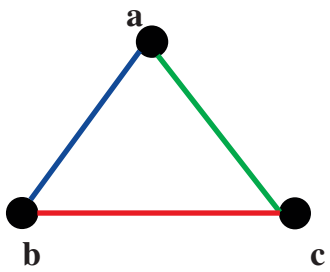
Step 3: Colour the next edge

Edge $\mathbf{e}_2 = \{\mathbf{b}, \mathbf{c}\}$ is adjacent to $\mathbf{e}_1$ (they share vertex $\mathbf{e}$. So, we cannot reuse Colour 1. Assign Colour 2 (**red**) to $\mathbf{e}_2$.



Step 4: Colour the last edge

Edge $\mathbf{e}_3 = \{\mathbf{c}, \mathbf{a}\}$ is adjacent to both $\mathbf{e}_1$ and $\mathbf{e}_2$ at $\mathbf{a}$ and $\mathbf{c}$. So, it cannot use Colour 1 or Colour 2. Asign Colour 3 (**green**) to $\mathbf{e}_3$.



The final edge colouring is: $\mathbf{e}_1(1),\ \mathrm{e}_2(2),\ \mathrm{e}_3(3)$.

No two adjacent edges share the same colour, and the colouring uses three colours in total. Thus, the the *chromatic index* $\mathbf{q}(\mathbf{G}) \le 3$.

4. **Proposals:**

For a multigraph $\mathbf{G} = (\mathbf{X}, \mathbf{U})$, we have the following two properties:

Properties:

4. $\mathbf{q}(\mathbf{G}) \ge \max\limits_{\mathbf{x} \in \mathbf{X}} \left(\mathbf{d}_{\mathbf{G}}(\mathbf{x})\right)$

5. $\boldsymbol{\gamma}(\mathbf{G}) \le \max\limits_{\mathbf{x} \in \mathbf{X}} \left(\mathbf{d}_{\mathbf{G}}(\mathbf{x})\right) + 1$

Brooks' Theorem (1941):

Let $\mathbf{G}$ be a connected simple graph with maximum degree $\mathbf{p}$. Then $\mathbf{G}$ is $\mathbf{p}$-colourable, unless

(1) $\mathbf{p} \ne 2$, and $\mathbf{G}$ is a $(\mathbf{p}+1)$-clique, or

(2) $\mathbf{p} = 2$, and $\mathbf{G}$ is an odd cycle.

Berge's Theorem (1970):

The chromatic index of a bipartite multigraph $\mathbf{G}$ with maximum degree $\mathbf{p}$ is $\mathbf{q}(\mathbf{G}) = \mathbf{p}$.

5. The Four-Colour Theorem

Every planar graph is 4-chromatic.

The Four-Colour Theorem states that it is possible, using only four distinct colours, to colour any map divided into connected regions in such a way that two adjacent regions—that is, regions sharing a common boundary (and not merely a single point)—always receive different colours.

The theorem can be expressed in several equivalent forms, such as the colouring of the faces of a polyhedron or the colouring of the vertices of a planar graph.

6. Perfect Graph

Let $\mathbf{G}=(\mathbf{X},\mathbf{U})$ be a multigraph, and let $\mathbf{w}(\mathbf{G})$ be the number of vertices in the clique subgraph of $\mathbf{G}$ that has the largest possible size.

If for every subgraph $\mathbf{S}$ of $\mathbf{G}$, we have $\boldsymbol{\gamma}(\mathbf{S})=\mathbf{w}(\mathbf{S})$, then $\mathbf{G}$ is called a *perfect graph.*

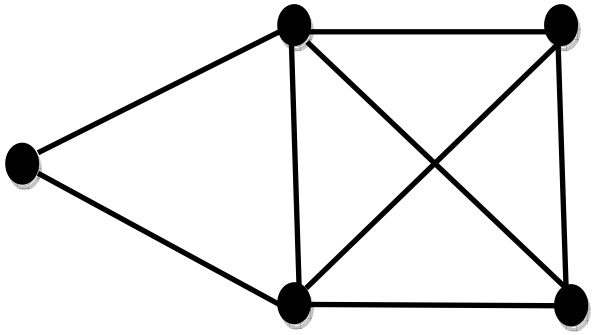
Examples:

- A clique $\mathbf{K_n}$ is a perfect graph. The largest clique is the graph itself, so $\mathbf{w}(\mathbf{K_n})=\mathbf{n}$. Since all vertices are adjacent, each vertex must have a different colour, so the chromatic number is also $\boldsymbol{\gamma}(\mathbf{K_n})=\mathbf{n}$. Hence, for every subgraph of $\mathbf{K_n}$, the chromatic number equals the size of its largest clique, which satisfies the definition of a perfect graph.
- A bipartite graph is also perfect, because it can always be coloured with two colours, and its largest clique has size 2.

Tutorial Session (7)

Exercise 1:

Show that the following graph admits an Eulerian chain.

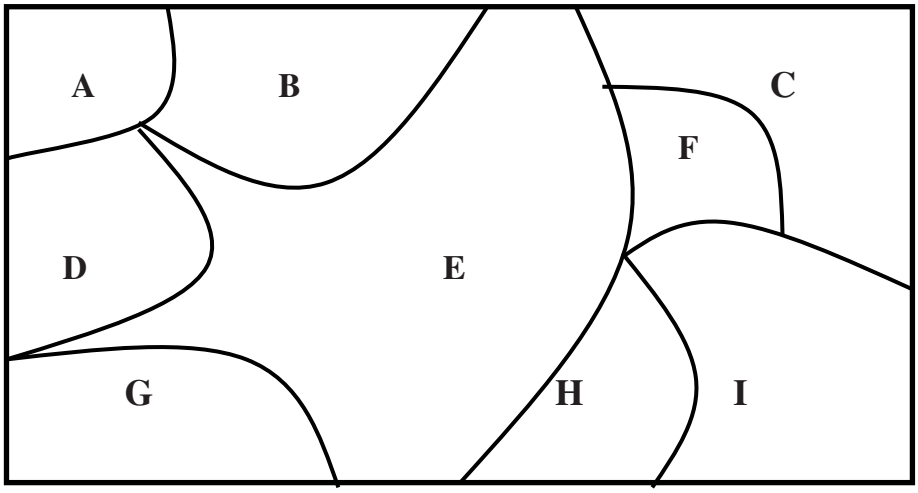


Exercise 2:

Does the Königsberg Bridge Problem have a solution?

Exercise 3:

How many colors are needed to color the map below, knowing that two neighboring countries must not be colored with the same color? A point is not considered as a border.



Exercise 4:

Three teachers **P1**, **P2**, **P3** must teach, next Monday, a certain number of class hours to three classes **C1**, **C2**, **C3**:

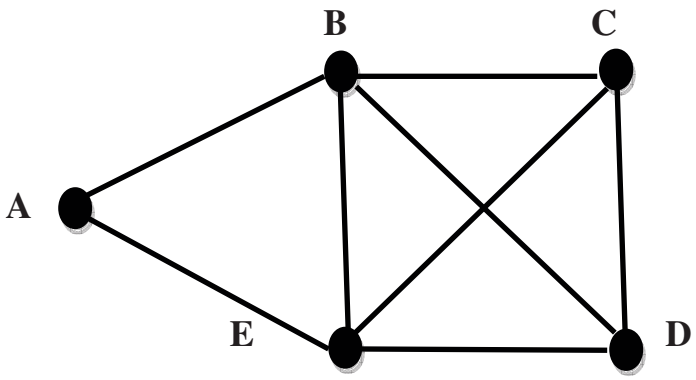
- **P1** must teach **C1** for **2** hours and **C2** for **1** hour.
- **P2** must teach **C1** for **1** hour, **C2** for **1** hour, and **C3** for **1** hour.
- **P3** must teach **C1** for **1** hour, C2 for **1** hour, and **C3** for 2 hours.

1. Model this problem as a graph and specify its type.
2. Determine the number of time slots required.
3. Using this graph, propose a possible timetable.

Tutorial Session 7 (Solution)

Exercise 1:

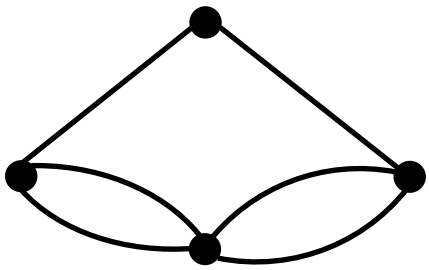
The following graph,



admits an Eulerian chain connecting vertices **C** and **D**, since the graph is connected and all vertices except **C** and **D** have even degrees. One possible Eulerian chain is: **(C, D, E, C, B, A, E, B, D).**

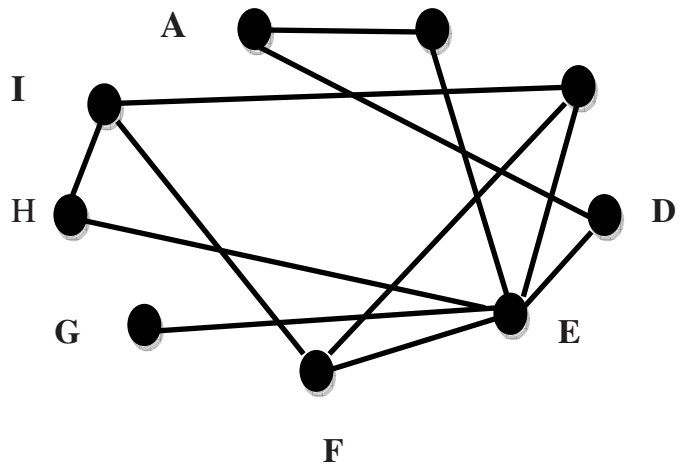
Exercise 2:

The Königsberg Bridge Problem has no solution, since the corresponding graph contains vertices of odd degree.



Exercise 3:

We begin by modeling the problem as a graph **G = (X , U)**. Each vertex **x** in **X** represents a country, and an edge **u** in **U** connects two vertices **x** and **y** if the corresponding countries are neighbors.



The goal is to colour the vertices so that no two adjacent vertices have the same colour.

The degrees of the vertices are:

$\mathbf{d_G(A) = 2},\ \mathbf{d_G(B) = 2},\ \mathbf{d_G(C) = 3},\ \mathbf{d_G(D) = 2},\ \mathbf{d_G(E) = 6},$

$\mathbf{d_G(F) = 3},\ \mathbf{d_G(G) = 1},\ \mathbf{d_G(H) = 2},\ \mathbf{d_G(I) = 3}.$

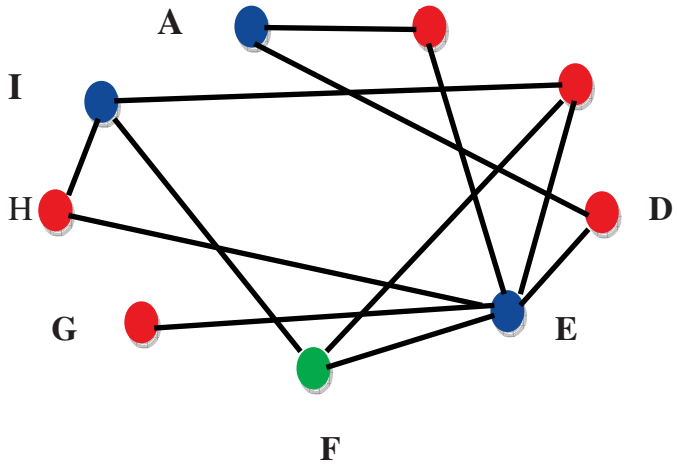
Vertices **E** has the maximum degree **6**. Assign **Colour 1** (**blue**) to vertex **E**.

After colouring vertex **E**, its neighbours **B**, **C**, **D**, **F**, **G**, **H** each have one coloured neighbour, hence their saturation degree is 1. To break the tie, we select the vertex with the highest degree. Choose vertex **C** (degree **3**). Its neighbour **E** already uses Colour 1; therefore, assign Colour 2 (**red**) to **C**.

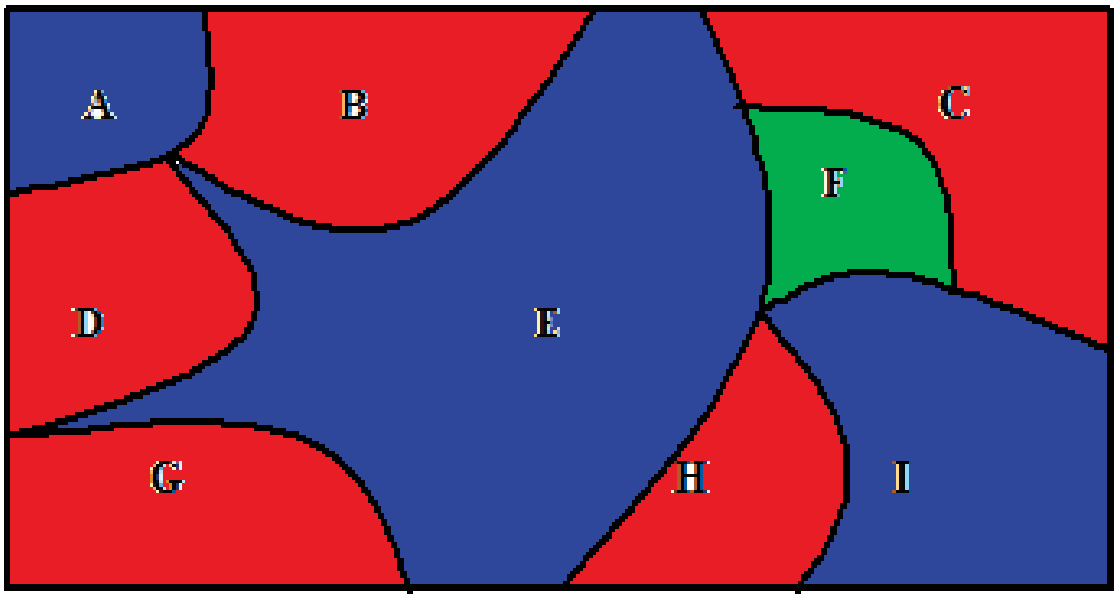
Vertex **F**, adjacent to both **E** (Colour 1) and **C** (Colour 2), has the highest saturation degree (2), thus assign Colour 3 (**green**) to vertex **F**.

Now, vertex **I** has the highest saturation degree (2). Its neighbours **C** and **F** use Colours 2 and 3 respectively, hence the smallest available colour is **Colour 1** (**blue**). Assign Colour 1 to vertex **I**.

Vertex **H** has the highest saturation degree (2). Its neighbours **E** and **I** use Colours 1, hence the smallest available colour is **Colour 2** (**red**). Assign **Colour 2** to vertex **H**. Using the same principle, we obtain the final coloring of both the graph and the map.

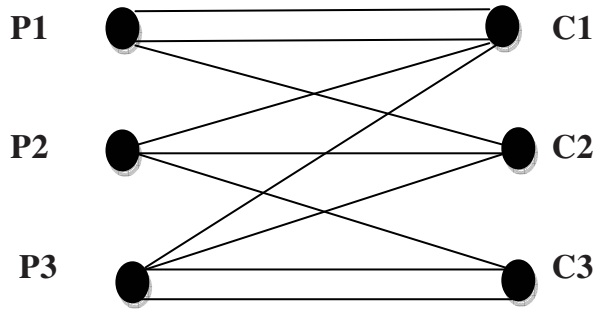


Thus, the *chromatic number* $\gamma(\mathbf{G}) \leq 3$.



Exercise 4:

We model the problem using a ***bipartite graph*** with two sets of vertices: **teachers (P)** and **classes (C)**. Each edge connects a teacher to a class, representing a teaching hour.



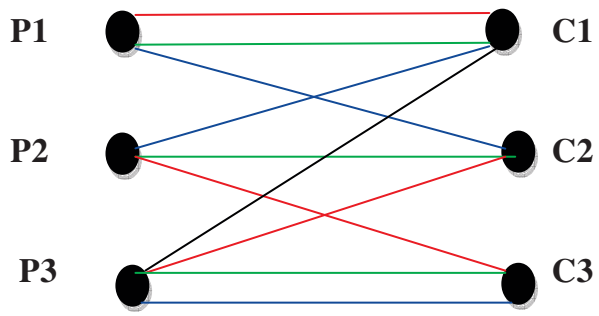
The **maximum number of time slots** needed equals the **maximum number of teaching hours** for any teacher, which corresponds to the **maximum degree** among the teacher vertices.

Since $\mathbf{Max(d(P_i)) = d(P_3) = 4}$, we need **4 time slots** in total.

We use the following color convention for the schedule:

- 1st hour → **red**
- 2nd hour → **green**
- 3rd hour → **blue**
- 4th hour → **black**

The coloring is assigned step by step, starting with the first hour, then the second, third, and fourth, as shown in the final graph.



Bibliography

1. Belharrat, N. (2003). *Théorie des graphes*. Pages Bleues & Lignes.
2. Berge, C. (1973). *Graphs and hypergraphs*. North-Holland Publishing.
3. Diestel, R. (2000). *Graph theory* (2nd ed.). Springer-Verlag.
4. Gibbons, A. (1985). *Algorithmic graph theory*. Cambridge University Press.
5. Harary, F. (2018). *Graph theory*. CRC Press, Taylor & Francis Group.
6. Kadri, O. (2021). *Théorie des graphes* [Polycopie]. Université Mostefa Benboulaïd Batna 2.
7. Kaufmann, M. (n.d.). *Des points, des flèches : la théorie des graphes*. Dunod, Sciences Poche. (Épuisé).
8. Mohar, B., & Thomassen, C. (2001). *Graphs on surfaces*. Johns Hopkins University Press.
9. Nguyen Huy Xuong. (1997). *Mathématiques discrètes et informatique*. Masson.
10. Pelle, S. (2005). *Géomatique : La théorie des graphes* [Polycopie]. École Nationale des Sciences Géographiques.
11. Sache, A. (1974). *La théorie des graphes*. Que Sais-Je? (Réédition prévue en 2004 chez Cassini).