République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Ibn Khaldoun–Tiaret

Faculty of Mathematics and Computer Science

Department of Computer Science

End-of-studies dissertation

In order to obtain the academic Master's degree

Specialization: Networks and Telecommunications

Presented by:

BOUZID Asmaa

MEBROUK Yacine

Theme

The automatic solver in graph theory "Application of Eulerian chain"

Publicly defended on : 11/07/2023

Jury:

President: Mr. DAHMANI Youcef (Professor)…..……Tiaret Ibn Khaldoun University

Supervisor: Mr. ALEM Abdelkader **(MAA)**…… Tiaret Ibn Khaldoun University

Examiner: Mr .BOUALEM Adda (MCA)..…… Tiaret Ibn Khaldoun University

Academic year 2022-2023


بسم الله الرحمن الرحيم

Thanks

First we would like to thank the Al mighty God, for having given us the strength to reach this level of study, and the patience to accomplish this modest work.

We would like to express our deepest thanks to our dear parents:

Who have never stopped encouraging and advising us, they have helped us a lot all along our way, thanks to their love, their understanding and their patience without ever taking their eyes off us or lowering their arms and their moral support and material, one would not really know towards them,

Our thanks go particularly to our supervisor **Mr** "**ALEM Abdelkader**". For his will ingress to accept to supervise us, for all the advice he gives us, his help, his remarks and his kindness.

We would also like to thank the members of the jury for the interest they have shown in our research by agreeing to examine our work and enrich it with their proposals

Finally, we address our sincere feelings of gratitude to all the people who have participated directly or indirectly in the realization of this work

Dedicate

I want to take a moment to express my heartfelt gratitude and love for each one of you. You have been an integral part of my life, and I am truly blessed to have you by my side, supporting and encouraging me every step of the way.

To myself, I am proud of the person I have become. I admire my resilience, strength, and the continuous growth I strive for. Through the ups and downs, I have stayed determined and remained true to who I am. I promise to continue nurturing and believing in myself, knowing that I am capable of achieving great things.

To my father BOUZID Djellali, you have been my guiding light, teaching me invaluable lessons about life, responsibility, and perseverance. Your unwavering support, love, and sacrifices have shaped me into the person I am today. I am grateful for your wisdom, and I carry your teachings with me wherever I go.

To my mother TOUMI Taoues , your unconditional love, warmth, and nurturing nature have been a source of comfort and strength. You have been my rock, providing endless support, encouragement, and understanding. Your love has shaped my character, and I am forever grateful for your presence in my life.

To my sisters Salima and Sabah, you are my best friends, confidantes, and companions. We have shared laughter, tears, and countless memories together. Your love, loyalty, and sisterhood mean the world to me. I cherish the bond we share and look forward to the adventures that await us.

To my brothers Abdelbasset and Mouhamed Nour El Islem, you are my pillars of strength and my partners in crime. Our bond is unbreakable, and the memories we have created are treasured. Your presence brings joy and laughter into my life, and I am grateful for your unwavering support and love.

With heartfelt appreciation and love,

ASMAA

Dedicate

To my dear parents, my unwavering pillars,

To you who have given me life and guided me through every step of my journey, I dedicate these words filled with gratitude and love. Your unwavering support and unconditional love have been the driving force in my life. Every day, I am grateful to have been blessed with such wonderful parents like you.

You have been my mentors, my confidants, and my greatest advocates. Your infinite patience and wisdom have taught me invaluable lessons. You have shown me the value of hard work, honesty, and compassion towards others. Your comforting presence and constant encouragement have given me the confidence to pursue my dreams.

To my beloved brothers,

May this message testify to the love and gratitude I feel towards both of you . Since our childhood, we have shared precious moments and memories that will forever be etched in my heart. You are my most loyal allies and companions on this journey called life.

We have shared laughter, tears, and experiences that have strengthened our bond as siblings. Your unwavering support and presence in my life are priceless gifts. I am proud of your achievements, and I will always be there for you, no matter what.

To my friends,

Tayeb Hatem and Kacem Mohamed , you are more than friends , you are my brothers my family in this life .Your unwavering support has helped me overcome challenges and celebrate victories .

To all of you, my loving parents, my beloved brothers, and my friends, I dedicate this message filled with gratitude and recognition. You are the pillars that support my life, and I can never thank you enough for everything you have done.

YACINE

Table of contents

List of tables

Figure list

General Introduction

General Introduction

The history of graph theory probably begins with Euler's work in the 18th century and finds its origins in the study of certain problems, such as the Königsberg bridge problem (the residents of Königsberg wondered whether it was possible to start from any point in the city, cross all bridges and return to the starting point without passing over the same bridge twice), the walking of a knight on a chessboard, and the map coloring problem..**(Eric Sigward, 2002)**

The objective of this dissertation is for the system to receive a typical graph exercise statement (Eulerian path) as input and return the solution as output. This solution is represented in a format close to a typical correction, allowing students to verify their solutions and enabling the teacher to automate part of the correction process.

In the first chapter, we present the fundamental aspects of graph theory that are relevant to our study and the problem of Eulerian paths.

In the second chapter, we introduce the Eulerian path. We then present the depth-first search algorithm and some LaTeX commands.

In the third chapter, we apply the algorithm to solve our problem and provide several obtained results.

Finally, we conclude with a general conclusion and some perspectives

CHAPTER I:

Graph theory rappel

1. INTRODUCTION

How can we lay cable at minimum cost and make it accessible from any phone? What is the shortest route from the capital to each state capital? How can n jobs be filled by n people? What is the maximum flow rate per unit time from source to sink in a network of pipes? How many layers of computer chips are needed so that wires on the same layer do not cross? How can a sports league season be organized with a minimum number of weeks? In what order should traveling salesmen visit cities to minimize travel time? Can all map regions be color-coded in four different colors so that adjacent regions are colored differently? **(DOUGLAS, 2001)**

These practical problems and many others involve graph theory. So, in this chapter we introduce the definitions and basic notions of graph theory, together with examples and illustrative figures.

1.1 History

It's fascinating to learn about the Königsberg Bridge Problem and its connection to the development of graph theory. The inability of the city's residents to walk a route that crossed each bridge exactly once is an example of what is now known as the Eulerian path problem. Euler's work on this problem laid the foundation for the study of graphs and their properties. Graph theory has since become an important field of study with applications in various areas such as computer science, social networks, and transportation systems. (Figure 1) **(John M, 2008)**

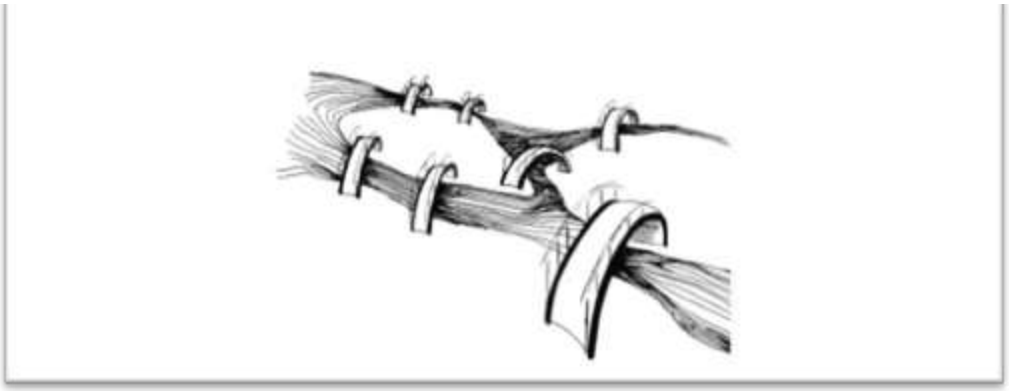


Figure 1: the bridges in Konigsberg **(John M, 2008)**

Yes, the Königsberg Bridge Problem is often cited as the birth of graph theory and serves as a basic example of a graph. In this problem, the city of Königsberg is represented as a graph with land masses as vertices and bridges as edges. The citizens' question of whether they could cross every bridge exactly once and return home is equivalent to finding an Eulerian path in the graph. This problem helped to establish the concept of a graph as a set of vertices connected by edges, which is now a fundamental concept in graph theory. .**(DOUGLAS, 2001)**

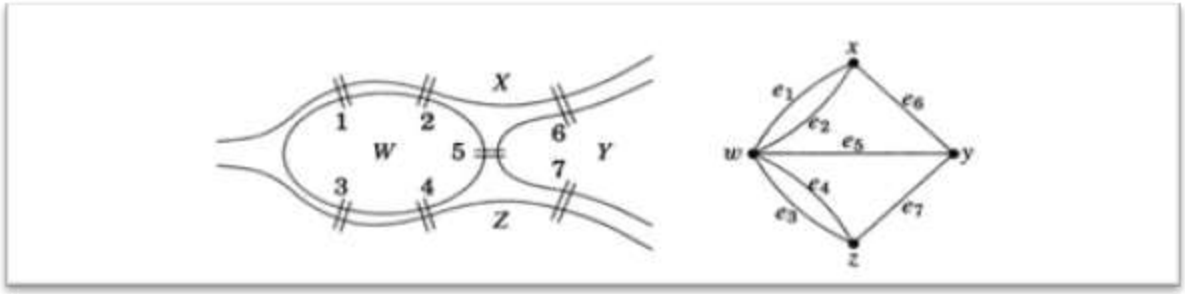


Figure 2: Konigsberg model (DOUGLAS, 2001)

The model on the right helps to illustrate why the desired traversal in the Königsberg Bridge Problem does not exist. The fact that each land mass must be involved in an even number of bridges is a necessary condition for the existence of an Eulerian path in the graph. However, this condition was not met in Königsberg, as some land masses were connected to an odd number of bridges.

As you mentioned, the Königsberg Bridge Problem becomes more interesting when we explore which configurations of bridges and land masses do have traversals. This problem is a classic example of graph theory and has led to many important results in the field. It also serves as a general model for discussing similar questions in other contexts. **(DOUGLAS, 2001)**

1.2. Graph definition

That's a great summary of the basic definitions of a graph. A graph G is a pair consisting of a non-empty set of points called vertices (V) and a set of edges (E), where each edge is a pair of vertices. If an edge connects a vertex to itself, it is called a loop. If there are multiple edges between the same pair of vertices, they are called multiple edges.

A simple graph is a graph with sigma. In a directed graph, denoted by G = (V, A), the edges are directed and are called arcs. This means that each arc has a direction and goes from one vertex (the tail) to another vertex (the head). Directed graphs are useful in modeling situations where the direction of the relationship between vertices is important, such as in a flow network or a social network. **(Abbes, 2021)**..

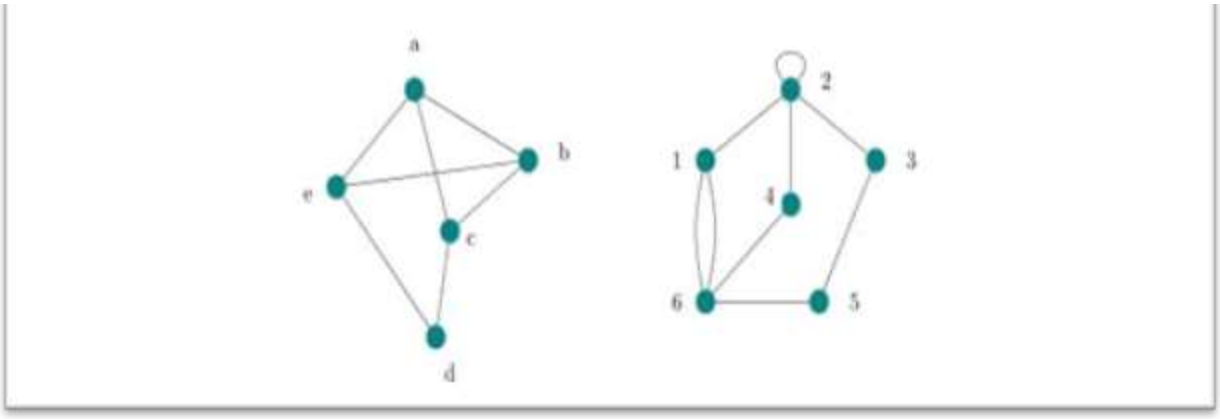


Figure 3: the graph on the left is simple, but the one on the right is not **(Abbes, 2021)**

1.3. The types of graphs

1.3.1 Oriented graph

A graph G = (V, E) consists of a finite set V = {v1, v2, ...} whose elements are called vertices, and a set E of ordered pairs of vertices, known as edges. An edge e from vertex x to vertex y can be denoted as e = (x, y), where x is the initial vertex and y is the terminal vertex.**(Didier Müller, 2004)**

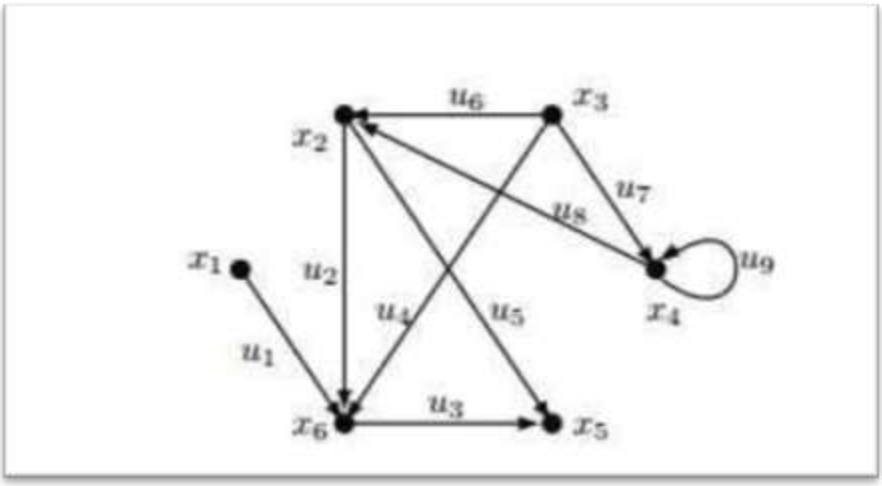


Figure 4: oriented graph (persu, 2023)

1.3.2 Graph no-oriented

We can define the graph Gr1 = (V1, E) by specifying its elements .Here, E = {e1, e3, ...,en} represents the set of edges, where each element e_i is called an edge. When an edge e connects two vertices, we say that the vertices are adjacent to or incident to e_i. **(Didier Müller, 2004)**

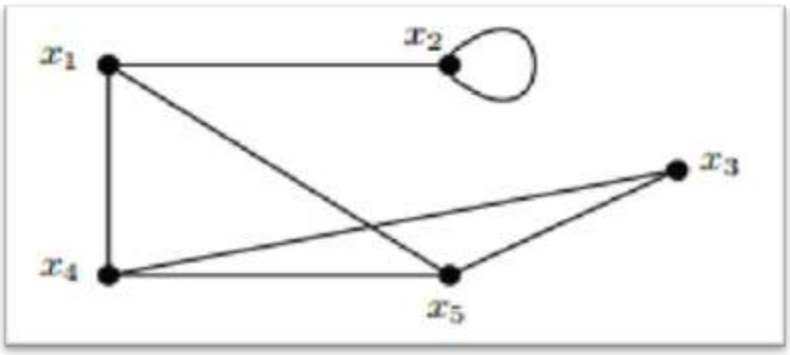


Figure 5: graph non oriented (persu, 2023)

1.3.3 Simple graph

In addition to edges connecting vertices to vertices, graphs have loops and multiple edges. A loop is an edge connecting a vertex to itself, and a sigma is an edge connecting a pair of identical vertices.

Graphs that allow loops and multiple edges are called polygraphs. A polygraph is a generalization of a simple graph, which does not allow loops or multiple edges.

A multigraph can have edges connecting vertices, loops on vertices, and multiple edges between pairs of the same vertices. This type of graph is useful for modeling specific situations, such as transportation networks or social networks. **(Didier Müller, 2004)**

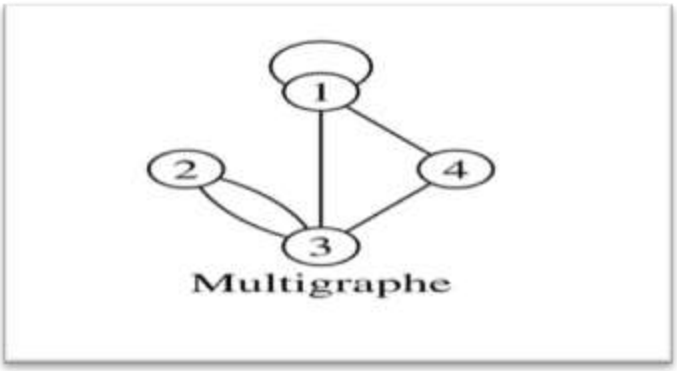


Figure 6: Multigraphe (Didier Müller, 2004)

1.3.4 Connexe graph

If from any vertex it is possible to connect all other units by following the unconnected unit: It consists of components and compresses the connection. The components are {1,2,3,4} and {5,6}. **(Didier Müller, 2004)**

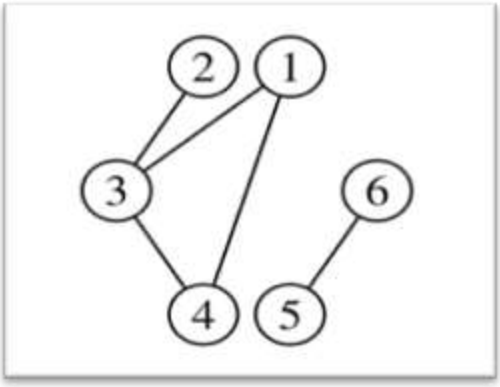


Figure 7: graph connexe (Didier Müller, 2004)

1.3.5 Complete graph

If each vertex of a graph is directly connected to all other vertices **(Didier Müller, 2004)**

V={1,2,3,4,5}

E= { {1,2} {1,3} {1,4} {4,5} {2,3} {2,4} {2,5} {3,4} {3,5} {4,5}

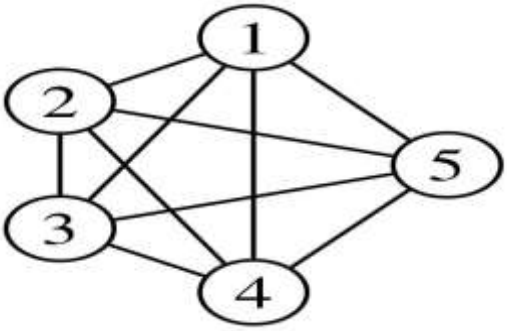


Figure 8: complete graph (Didier Müller, 2004)

1.3.6 Bipartite graph

If its vertices can be partitioned into two sets x and y, then every stop of the graph is connected by a vertex of x and a vertex of y. **(Didier Müller, 2004)**

x={1,3,5}

y={2,4}

v={1,2,3,4,5}

e={ {1,2} {1,4} {2,5} {3,4} {4,5} }

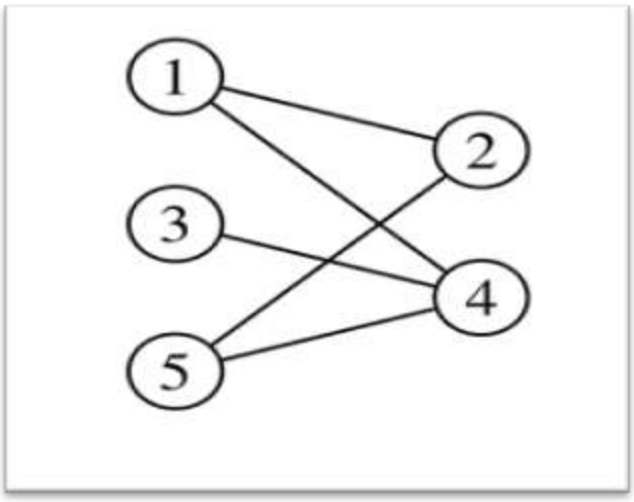


Figure 9: bipartite graph (Didier Müller, 2004)

1.3.6 Regular Graph

A regular graph is defined as a graph in which all vertices have the same degree. If the degree of each vertex is 'r', then the graph is referred to as a regular graph of degree 'r'. The null graph is an example of a regular graph with degree 0, as all vertices have no edges. On the other hand, the complete graph 'K' is a regular graph with degree 'n-1', where 'n' represents the total number of vertices in the graph.

Furthermore, for a regular graph 'G' with 'n' vertices and degree 'r', the number of edges in 'G' can be determined as 'n*r/2'.

This relationship arises from the fact that each edge is incident to two vertices, and the sum of the degrees of all vertices in the graph is twice the number of edges.

Regular graphs hold significant importance in graph theory and find applications in various fields such as chemistry, physics, computer science, and more. Graphs possess intriguing properties and have been extensively studied in mathematics. **(VASUDEV, 2006)**

1.3.7 Cycles

A cycle in graph theory, denoted by 'Cn' (where 'n ≥ 3'), is a graph consisting of 'n' vertices, labeled as 'V_1, V_2, ..., Vn', and 'n' edges connecting these vertices in a circular manner. The edges of the cycle are represented as '(V_1, V_2), (V_2, V_3), ..., (Vn_{-1}, V_1)', forming a closed loop.

To illustrate this concept, let's consider the examples of 'C_3', 'C_4', and 'C_5' cycles as depicted in the figure.

The 'C_3' cycle graph consists of three vertices, denoted as 'V_1, V_2, V_3', forming a triangle. The edges in this cycle are '(V_1, V_2), (V_2, V_3), (V_3, V_1)'. Mathematically, the 'C_3' cycle graph can be represented as follows:

$C_3 = (V, E)$

$V = \{V_1, V_2, V_3\}$

$E = \{(V_1, V_2), (V_2, V_3), (V_3, V_1)\}$

Similarly, the 'C_4' cycle graph is formed by introducing a fourth vertex to the cycle, resulting in a square-like structure. The edges in this cycle are '(V_1, V_2), (V_2, V_3), (V_3, V_4), (V_4, V_1)'. The mathematical representation of the 'C_4' cycle graph is:

$C_4 = (V, E)$

$V = \{V_1, V_2, V_3, V_4\}$

$E = \{(V_1, V_2), (V_2, V_3), (V_3, V_4), (V_4, V_1)\}$

Likewise, the 'C_5' cycle graph is constructed by adding a fifth vertex to the cycle, forming a pentagon-like structure. The edges in this cycle are '(V_1, V_2), (V_2, V_3), (V_3, V_4), (V_4, V_5), (V_5, V_1)'. The mathematical representation of the 'C_5' cycle graph is:

$C_5 = (V, E)$

$V = \{V_1, V_2, V_3, V_4, V_5\}$

$E = \{(V_1, V_2), (V_2, V_3), (V_3, V_4), (V_4, V_5), (V_5, V_1)\}$

In summary, a cycle graph 'C□' is composed of 'n' vertices and 'n' edges forming a closed loop. The mathematical representation of a cycle graph includes a set of vertices 'V' and a set of edges 'E', denoted as 'C□ = (V, E)'. The edges are defined by connecting consecutive vertices in the cycle, with the last vertex connected back to the first vertex, as shown in the formulas above for 'C_3', 'C_4', and 'C_5' graphs.**(VASUDEV, 2006)**

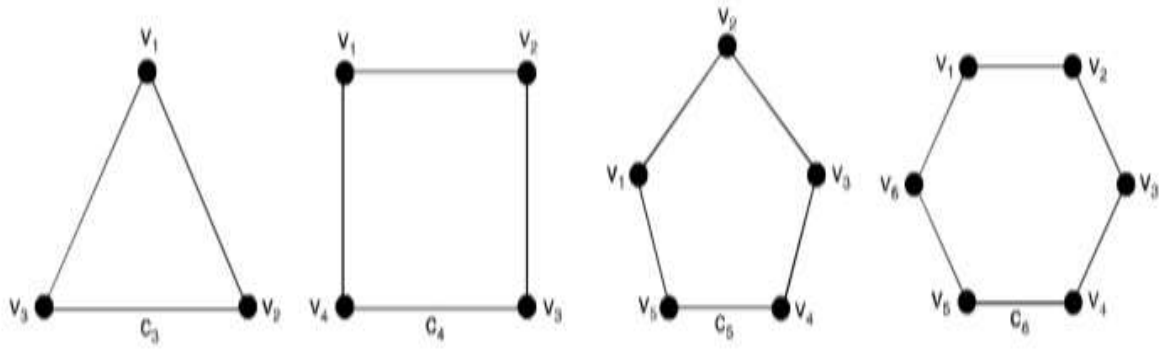


Figure 10: Cycles C_3, C4, C5 and C6 (VASUDEV, 2006)

1.3.8 Wheels

The wheel graph labeled as "Wn" is constructed by introducing an extra vertex to the cycle graph "Cn" (where "n ≥ 3"), and then adding new edges to connect this newly added vertex to every existing vertex in "Cn". This process results in the formation of a wheel-shaped graph.

To illustrate this concept, let's consider the examples of "W_4", "W_5", "W_6", and "W_7" wheels as depicted in Figure 1.

The "W_4" wheel graph consists of a cycle with four vertices, denoted as "C_4", forming a square. Additionally, a fifth vertex is added to the graph, which is connected to each of the four vertices in "C_4" by new edges. Mathematically, the "W_4" graph can be represented as follows:

$W_4 = (V, E)$

$V = \{v_1, v_2, v_3, v_4, v_5\}$

$E = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_1, v_5), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_5, v_2)\}$

Similarly, the "W_5" wheel graph is formed by connecting a fifth vertex to each vertex of the cycle graph "C_5", creating a pentagon-like structure. The mathematical representation of the "W_5" graph is:

$W_5 = (V, E)$

$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$

$E = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_1, v_5), (v_1, v_6), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_5, v_6), (v_6, v_2)\}$

Similarly, the "W_6" wheel graph includes a sixth vertex connected to each of the vertices in the cycle graph "C_6". The "W_7" wheel graph is formed by adding a seventh vertex and connecting it to every vertex in "C_7". The mathematical representations of "W_6" and "W_7" graphs follow a similar pattern as shown above for "W_4" and "W_5".

In summary, the "W□" wheel graph is constructed by extending the cycle graph "C□" with an additional vertex and connecting it to every existing vertex in "C□". This process creates a wheel-like structure with "n+1" vertices and "2n" edges. The mathematical representation of the "W□" wheel graph can be defined as "W□ = (V, E)", where V represents the set of vertices and E represents the set of edges connecting the vertices in the graph..**(VASUDEV, 2006)**

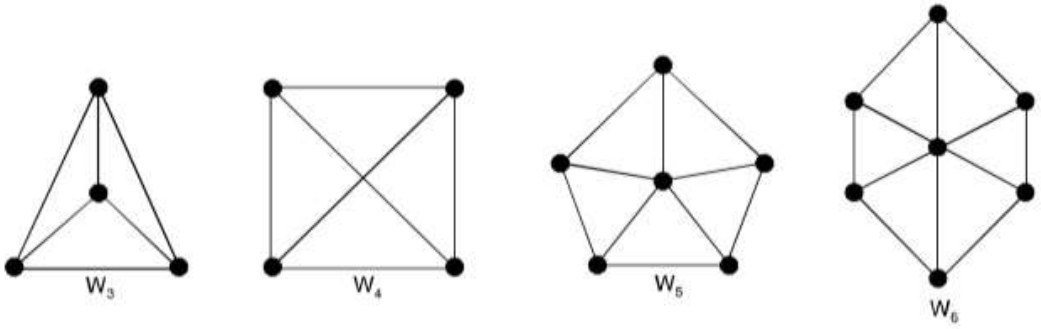


Figure 11: Wheels W_3, W_4, W_5 and W_6 (VASUDEV, 2006)

1.3.9 Platonic graph

A figure formed by the vertices and edges of five regular solids (platons): tetrahedron, octahedron, cube, dodecahedron, and icosahedron..**(VASUDEV, 2006)**

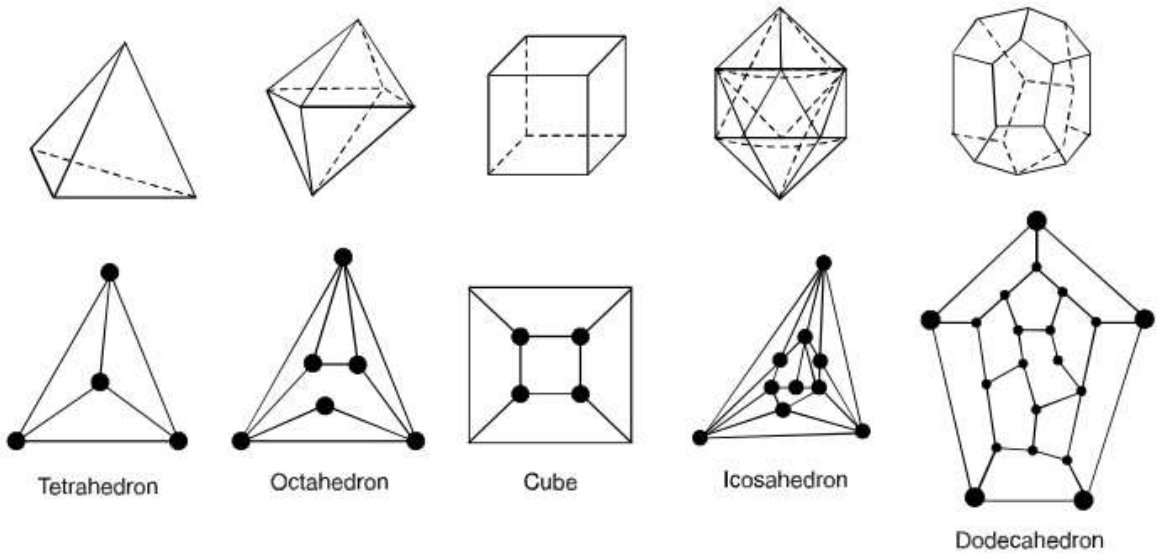


Figure 12 : Platonic graph (VASUDEV, 2006)

1.4. Eulerian path problem

Data : Graph G

Question: Is there a simple clear phrase for G that includes all edges of G? Such a path is called an Euler chain. If the Euler chain is closed (the first and last vertex of the path are the same), it is called an Euler circle. The routing problem of city streets is thus a problem of passing through each and only one of them, and thus the existence of Euler circles in a graph

where vertices are intersections of streets and stops are parts of streets at the crossroads **(Frederic Murnier)**.

1.5. Progression in a graph

The concepts of progression in a graph are : Cycles, chains, paths, tours.

1.5.1 Chain

In the context of graph theory, we define a chain in a graph as either a directed chain or an undirected chain. Let's consider a graph G, which is represented by the set of vertices V and the set of arcs (or edges) U. We can express this graph as G = (V, U).

A chain in G is an alternating sequence of vertices and arcs. Specifically, we represent the chain as U = V0, U1, V1, ..., where U0, U1, U2, ... denote the arcs, and V0, V1, V2, ... denote the vertices. The endpoints of each arc Ui are the vertices Vi-1 and Vi.

Furthermore, if a chain Ua connects the vertices V0 and Vk, we say that Ua is a chain of length k. The length of the chain represents the number of arcs or edges in the sequence.

Mathematically, we can express a chain U as:

U = (V0, U1, V1, U2, V2, ..., Uk-1, Vk)

By employing these definitions and mathematical representations, we can analyze and study chains in graphs, taking into account their directionality (in the case of directed chains) or their lack thereof (in the case of undirected chains). **(A. Bretto A, 2012)**

1.5.2 Cycle

Cycle is a closed path in a graph where the starting and ending vertex is the same, and all other vertices in the cycle are distinct. A cycle can be defined as a sequence of vertices V0, V1, ..., Vk-1, Vk, along with their corresponding edges U1, U2, ..., Uk and V0=Vk (directed or undirected) we say U is a cycle **(A. Bretto A, 2012)**.

1.5.3 Path

There exists an alternating sequence of vertices and stops y=V0,U1,V1,...,Vk such that 1=<i=<K in the graph (directed or undirected), provided that the vertices V i+1,Uk,Vk-1,Uk,Vk are its endpoints. ,Vk-1,Uk,Vk if and only if there exists an alternating sequence of vertices V i+1 such that 1=<i=<K in the graph (directed or undirected) and the stop y=V0,U1,V1,...,Vk is its endpoint. **(A. Bretto A, 2012)**

1.5.4 .Circuit

In the directed graph G= (V.U), y = V0 , U 1 ,V1,... , Vk-1 , U k ,Vk . y is a circuit of length k such that y = Vk-1 , U k ,Vk , k >0, V0 =Vk. y is a circuit of length k such that y = Vk-1 , U k ,Vk , k >0, V0 =Vk. **(A. Bretto A, 2012)**

1.6. Representation mode

1.6.1 Succession list

Consider a graph G = [X , U]. To represent a graph G by an inheritance list, we use

6.1.1. an array, called a dictionary, containing the vertices of the graph G.

6.2.2. a list of successors (or predecessors) matching each element of the dictionary. **(Fatima zohra tebbak)**

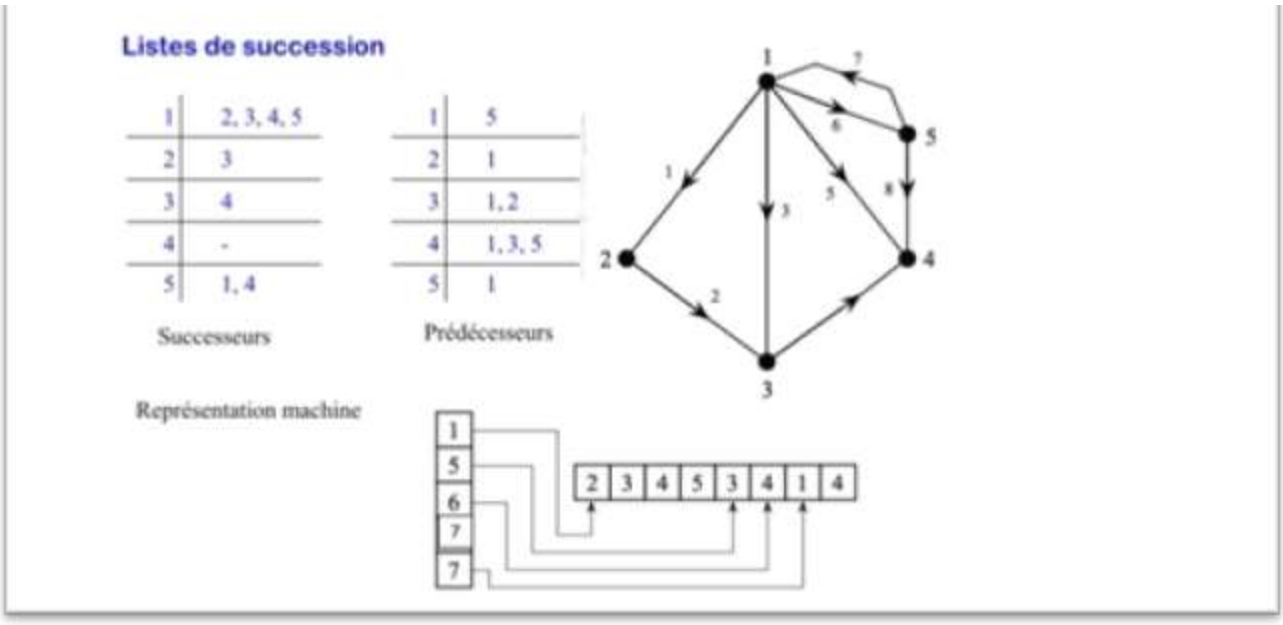


Figure 13: Succession list(Jean Charles Régin, 2016)

1.6.2 Adjacency matrix

Consider a graph G = [X , U] of degree N . The adjacency matrix of G is equal to the matrix A = (a) of dimension Nx N .

6.1.1.Directed graph The adjacency matrix of a directed graph is any matrix with zeros and ones.

6.2.2. The adjacency matrix of an undirected graph is a diagonal symmetric matrix _3 . **(Fatma zohra tebbak)**

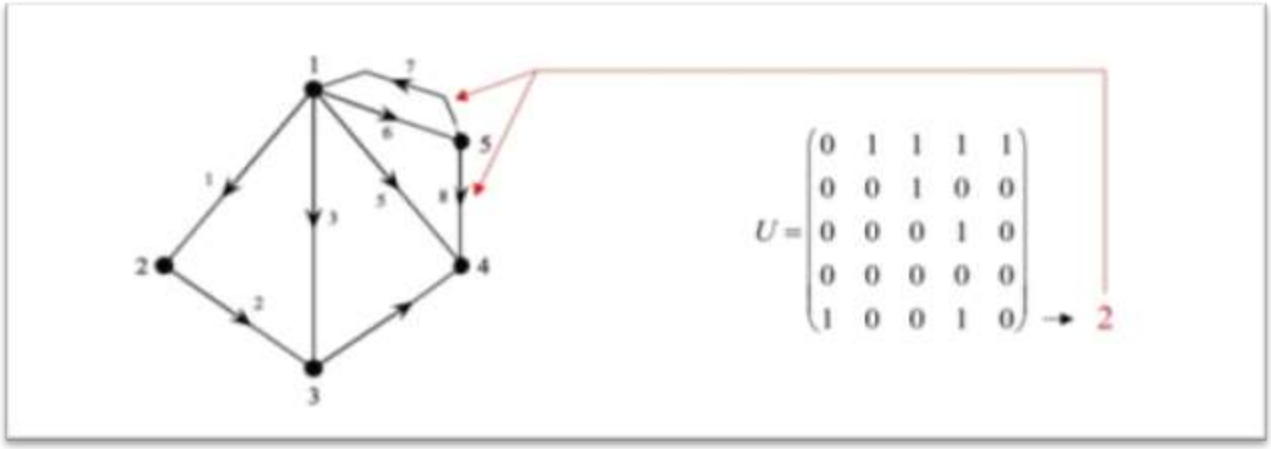


Figure 14: Adjacency matrix (Jean Charles Régin, 2016)

1.6 .3. Incidence matrix of

1.6.3.1 A directed graph

Consider the loop-free directed graph G = [X , U] of order N , with:

- X = { x_1 , ... xN } : set of vertices.
- U = { a , ... am): set of arcs.

We call incidence matrix of G , the matrix A = (a) of dimension Nx M , such that:

- 1 if x , is the initial end of uj.
- 1 if x , is the terminal end of u.
- 0 if x , is not an extremity of u.

1.6.3.2 An undirected graph

Consider an undirected loop graph G = [X , U] of degree N:

- X = {x_1, ... xn}: the set of vertices.

- U = {aam}: the set of edges.

The incidence matrix of G is denoted by the matrix A = (a) of dimension Nx M such that

- 1 if x is an extreme value of u.

- 0 otherwise. **(Fatima zohra tebbak)**

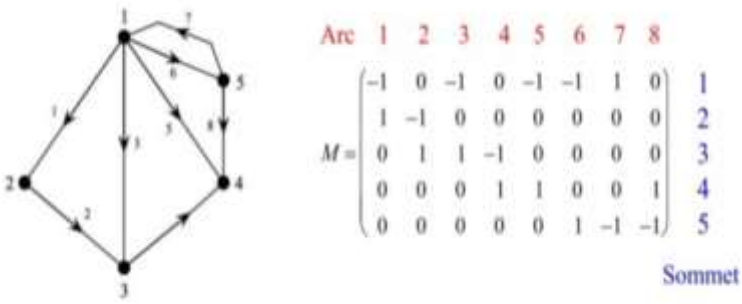


Figure 15: incidence matrix **(Jean, 2019)**

1.7. Operations on graphs

• As in the case of mathematical entities, it is convenient to consider a large graph as a combination of smaller graphs and to derive its properties from those of the smaller graphs. Since a graph is defined by a set of vertices and edges, it is natural to use the terminology of set theory to define relations between graphs. In particular, the union of two graphs G1=(V1,E1) and G2=(V2,E2) is another graph G3 (written G3=G1∪G2) with vertex set V3=V1ÂV3 and edge set E3=E1â Similarly, the intersection between G1and G2 of graphs G1 and G2 is a graph G4 consisting only of vertices and edges in both G1 and G2. The ring union of two graphs G1 and G2 (written G1G2) is a graph consisting of the vertex set V1Sm_C2 and an edge contained in either G1 or G2 but not in both The two graphs, their union, intersection and ring union are shown in Figure 13. As is clear from the definitions, the three operations just described are commutative. That is, as in the case of mathematical entities, it is convenient to consider a large graph as a combination of smaller graphs and derive its properties from those of the smaller graphs. Since a graph is defined by a set of vertices and edges, it is natural to use the terminology of set theory to define relations between graphs. In particular, the union of two graphs G1=(V1,E1) and G2=(V2,E2) is another graph G3 (written G3=G1∪G2) with vertex set V3=V1ÂV3 and edge set E3=E1â Similarly, the intersection between G1and G2 of graphs G1 and G2 is a graph G4 consisting only of vertices and edges in both G1 and G2. The ring union of two graphs G1 and G2 (written G1G2) is a graph consisting of the vertex set V1Sm_C2 and an edge contained in either G1 or G2 but not in both The two graphs, their union, intersection and ring union are shown in Figure 16. As is clear from the definitions, the three operations just described are commutative. That is.

- G1∪G2=G2∪G1,

- G1⋂G2=G2⋂G1,
- G1⊕G2=G2⊕G1.
- If G1 and G2 are edge disjoint, then G1⋂G2 is a null graph, andG1⊕G2=
- G1⋃G2.IfG1 and G2 are vertex disjoint, then G1⋂G2 is empty.For any graph .G,
- G⋃G=G⋂G=G, and
- G⊕G=a null graph.

If g is a subgraph of G, then G⊕g is, by definition, that subgraph of G which remains after all the edges in g have been removed from G. Therefore, G⊕g is written as G−g,whenever g⊆G. Because of this complementary nature, G⊕ g=G−g is often called the complement of g in G. Decomposition: A graph G is said to have been decomposed into two subgraphs g1 and g2 if : g1⋃g2=G, and g1⋂g2=a null graph.**(NARSINGH DEO, 1974)**

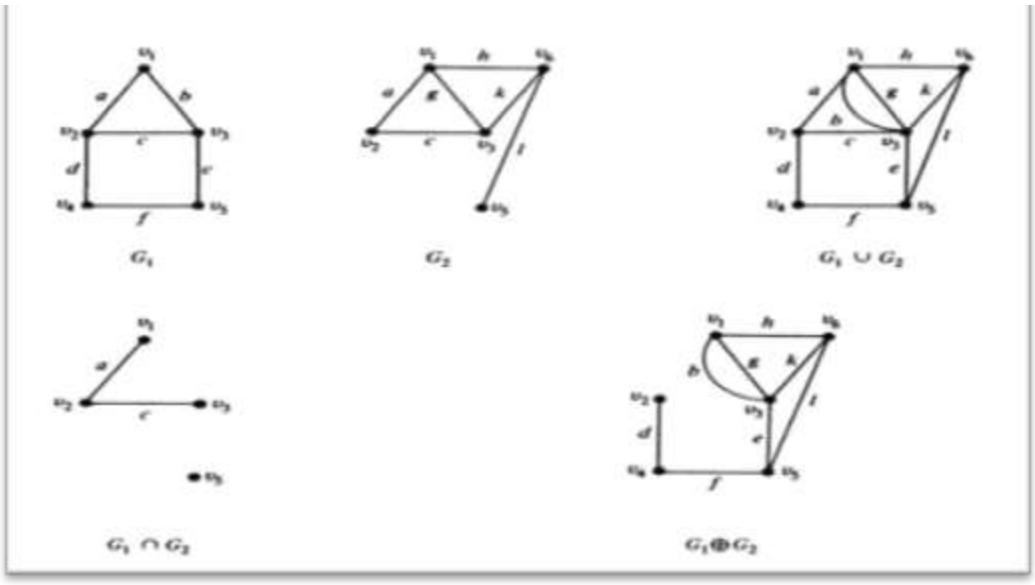


Figure 16: union, intersection, and ring sum of tow graphs.(NARSINGH DEO, 1974)

In other words, every edge of G resides on either gl or g2, but never on both. However, some vertices may reside on both gl and g2. In the decomposition, isolated vertices are ignored. m edges {el,e2,... ,em} can be decomposed into pairs of subgraphs g1,g2 in 2m-1-1 ways (why?). Although union, intersection, and ring sum are defined for a pair of graphs, these definitions can be extended in obvious ways to include any finite number of graphs. Similarly, a graph G can be decomposed into two or more subgraphs - subgraphs whose (paired) edges are disjoint and collectively contain all edges of G .**(NARSINGH DEO, 1974)**

1.8. Applications of graphs

Graph theory has found extensive use in disciplines such as engineering, physics, social sciences, biological sciences, linguistics, and many others.

In engineering, graphs provide a powerful tool for understanding and solving complex problems. They can be utilized to model networks, such as electrical circuits, communication systems, transportation routes, and logistical networks. By representing these systems as graphs, engineers can study their properties, optimize their design, and analyze their performance.

In physics, graphs play a crucial role in describing and studying interactions between particles, molecular structures, and energy flows. They are extensively employed in fields like quantum mechanics, statistical physics, and computational physics to analyze complex systems and visualize relationships between various components.

Social sciences benefit from graph theory by using graphs to represent social networks, where individuals or entities are represented as nodes, and their connections or interactions are represented as edges. This enables researchers to study social relationships, analyze information diffusion, and investigate the spread of influence within communities.

In biological sciences, graphs provide a valuable framework for modeling genetic networks, protein interactions, food webs, and neural networks. By employing graph theory, researchers can understand the structure and dynamics of biological systems, identify key components, and analyze the impact of perturbations or mutations.

Linguistics also makes extensive use of graphs to analyze language structures and relationships. Graphs can be employed to represent phonological patterns, syntactic structures, semantic networks, and language evolution. This allows linguists to study language properties, analyze linguistic phenomena, and develop computational models for natural language processing.

Beyond these specific fields, graph theory finds applications in diverse domains such as computer science, information technology, operations research, data mining, and optimization. Its wide-ranging utility stems from the fact that graphs can effectively represent and analyze relationships and dependencies in almost any system composed of discrete objects.

In summary, the simplicity and versatility of graph theory have made it an invaluable tool across various disciplines. By employing graphs to represent objects and their relationships, researchers and practitioners can gain insights, solve complex problems, and uncover patterns and structures in diverse fields, ranging from engineering and physics to social sciences, biological sciences, linguistics, and beyond...**(NARSINGH DEO, 1974)**

1.9. The Basic of graph theory

Let V(G) denote the vertex set of a graph G and E(G) denote the edge set. For notational convenience, instead of denoting edges as {u, v}, we denote them simply as uv. Given two vertices u and v, we say that u and v are adjacent if uv∈E. In this case, we say that u and v are endpoints of edge uv. If uv/∈ E, then u and v are nonadjacent. Furthermore, if an edge has vertex v as an endpoint, we say that v is incident to E.

We denote the neighborhood (or open neighborhood) of a vertex v by N(v). v: N(v)={x∈V|vx∈E}.**(John M, 2008)**

- A closed neighborhood of a vertex v, denoted N[u], is simply the set {v} U N(v). Similarly, a closed neighborhood of S denoted N[S] is defined to be SUN(S).
- The degree of u, denoted deg(v), is the number of edges associated to u. For simple graphs, this is the same as the cardinality of the (open) neighborhood of v.Δ(G)= max{deg(v) | v∈ V(G)}.

Similarly, the minimum degree of a graph G, denoted by 5(G), is defined to be

- δ(G)= min{deg(v) | v∈ V(G)}.**(John M, 2008)**

1.10. Terminology

10.1 Subgraph: H = (Y,B) is a subgraph of G = (X.A) if Y is subset of X and B is subset of A.

10.2 Order of a graph: the order of a graph is the number of vertices in that graph.

10.3 Path: a finite sequence of vertices connected by edges.

10.4 Simple path: a path that does not use the same edge twice

10.5 Eulerian path: a simple path that passes through all edges of a graph.

10.6 Hamiltonian path: a simple path that passes through all vertices of a graph once and only once.

10.7 Walk: a sequence of vertices connected by arcs in a directed graph.

10.8 Cycle: a path that returns to its starting point.

10.9 Eulerian cycle: a simple cycle that passes through all edges of a graph once and only once.

10.10 Hamiltonian cycle: a simple cycle that passes through all vertices of a graph once and only once.

10.11 Connected graph: a graph G is said to be connected if for every pair of vertices (x, y) in G, there exists a path from x to y.

10.13Tree: a connected graph without a simple cycle or loop.

10.13 Eulerian graph: a graph that has an Eulerian cycle.

10.14 Semi-Eulerian graph: a graph that has an Eulerian path.

10.15 Hamiltonian graph: a graph that has a Hamiltonian cycle.

10.16 Semi-Hamiltonian graph: a graph that has a Hamiltonian path.

10.17 Valued graph: a graph where real numbers are associated with the edges. In this presentation, we will only consider positive valuations.

10.18 Length of a path: the number of edges that make up the path.

10.19 Value of a path: the sum of the values of the edges (arcs) in a valued graph.

10.20 Distance between two vertices: the length of the shortest path connecting those two vertices.

10.21 Diameter of a graph: the maximum distance between the vertices of a graph.

10.22 Chromatic index: the minimum number of colors required to color the edges of a graph such that adjacent edges do not have the same color.

10.23 Chromatic number of a graph: the minimum number of colors required to color the vertices of a graph such that adjacent vertices do not have the same color.**(Eric Sigward, 2002)**

10.24 isolated and pendant vertices :

10.24.1 Isolated vertex : A vertex having no incident edge is called an isolated vertex. In other words, isolated vertices are those with a degree of zero.

10.24.2 Pendant or end vertex: A vertex of degree one is called a pendant vertex or an end vertex. In the above figure, V_{5} is a pendant vertex.

10.24.3 In-degree and out-degree: In a graph G, the out-degree of a vertex v_{i} of G, denoted by outdeg(v) or deg(v), is the number of edges beginning at v_{i}. The in-degree of vertex v_{p}, denoted by indeg(v) or deg(v), is the number of edges ending at v_{p}.

The sum of the in-degree and out-degree of a vertex is called the total degree of the vertex. A vertex with a total in-degree of zero is called a source, and a vertex with a total out-degree of zero is called a sink. This is because each edge has an initial vertex and a terminal vertex.

10. 25 the handshaking theorem :

If G = (V, E) is an undirected graph with e edges, then the sum of the degrees of the vertices in G is equal to 2e. In other words:

$$\sum_{v \text{ in } V} \deg(v) = 2e$$

The sum of the degrees of the vertices in an undirected graph is always even **(VASUDEV, 2006)**

10 .26 vertex degrees:

The degree, denoted as d(v), of a vertex v in a graph G is defined as the number of edges in G that are incident with v. When considering loops, each loop is counted as two edges incident with the vertex.

Additionally, we denote the minimum and maximum degrees of vertices in G as $\delta(G)$ and $\Delta(G)$, respectively.

To summarize:

- The degree of a vertex v in G: d(v)

- Minimum degree of vertices in G: $\delta(G)$ and Maximum degree of vertices in G: $\Delta(G)$ **(J.A Bondy, 1976)**

Conclusion

In this chapter, we explored the historical background of Eulerian chains and then proceeded to define various types of graphs. These included oriented graphs, non-oriented graphs, simple graphs, connex graphs, complete graphs, bipartite graphs, regular graphs, wheel graphs, cycle graphs, and Platonic graphs.

Next, we delved into different methods of graph representation, which encompassed the use of succession lists, adjacency matrices, and incidence matrices. We also discussed the Eulerian path problem, which pertains to finding a path that traverses each edge of a graph exactly once.

Furthermore, we introduced the concept of progressions within a graph, namely chains, cycles, paths, and circuits. These terms were used to describe specific sequences of vertices and edges within a graph.

Finally, we concluded the chapter by exploring various operations on graphs and discussing their applications in diverse fields. Additionally, we covered fundamental graph terminology to facilitate a comprehensive understanding of the subject matter.

Chapter II

Eulerian chain

2.1. Introduction

In this chapter, our focus will be on the Eulerian chain. We will begin by revisiting the definition of an Eulerian chain, which is a concept in graph theory.

After establishing the definition of Eulerian chain , we will delve deeper into the properties and characteristics of Eulerian chains. We will explore the conditions under which a graph can possess an Eulerian chain and examine the implications of these conditions. Understanding the properties of Eulerian chains is crucial for identifying and analyzing them in various graph structures.

Next, we will discuss the "First Depth Search Algorithm," which is commonly employed in the application and identification of Eulerian chains. This algorithm, also known as the depth-first search (DFS) algorithm, systematically explores the vertices and edges of a graph to construct and trace paths. We will explain the algorithm's steps and illustrate how it can be used to determine the existence of Eulerian chains in a given graph.

In addition to discussing the Eulerian chain and its application, we will also cover LaTeX commands. LaTeX is a typesetting system commonly used in academia and scientific research to create professional-looking documents, including mathematical formulas, graphs, and diagrams. We will present and explain relevant LaTeX commands that can be used to generate visual representations of graphs and mathematical equations in a document.

By the end of this chapter, we will have a comprehensive understanding of Eulerian chains, their properties, the algorithm used to identify them, and the LaTeX commands that can be employed to create visually appealing representations of graphs and mathematical formulas.

2.2. Definitions

2.1 Definition Eulerian chain: An Eulerian chain is a path in a graph that traverses each edge exactly once, allowing for repeated visits to vertices. This path starts and ends at different vertices. It is named after the famous mathematician Leonhard Euler, who made significant contributions to the field of graph theory.

An Eulerian chain is a specific type of path in a graph, which is a mathematical representation of a network or interconnected system. In an Eulerian chain, you traverse each edge of the graph exactly once, allowing for multiple visits to vertices or nodes.

The path starts at one vertex and ends at a different vertex, which means the chain forms a loop or a closed circuit within the graph. This property distinguishes an Eulerian chain from an Eulerian cycle, where the path starts and ends at the same vertex.

The concept of an Eulerian chain is named after Leonhard Euler, a renowned Swiss mathematician who lived in the 18th century. Euler made significant contributions to the field of graph theory, which deals with the study of graphs and their properties. Euler's work on graph theory laid the foundation for many fundamental concepts and algorithms used in modern mathematics and computer science.

.**2.2 Definition Eulerian cycle :** If this Eulerian chain is closed, we say that we have an eulerian cycle.**(meilleur en maths)**

2.3. Euler's theorems (admitted):

3.1 A connected graph admits an Eulerian chain if and only if the number of vertices of odd degree is 0 or 2.

An Eulerian chain is a path in a graph that traverses each edge exactly once. In order for a connected graph to admit an Eulerian chain, certain conditions must be met. One such condition is that the number of vertices with odd degrees in the graph must be either 0 or 2.

To understand why this is the case, let's examine the properties of Eulerian chains and the degrees of vertices in a graph.

First, a degree of a vertex in a graph refers to the number of edges incident to that vertex. In an undirected graph, each edge is incident to two vertices, so the sum of degrees of all vertices is twice the number of edges.

Now, let's consider the properties of an Eulerian chain. Since an Eulerian chain visits each edge exactly once, it means that each time we enter a vertex, we must have an exit point from that vertex. The only exception is the starting vertex, where we can enter without an immediate exit, and the ending vertex, where we can exit without going any further.

Based on this observation, we can conclude that for a connected graph to admit an Eulerian chain, the degrees of all vertices except for two (or zero) must be even. This is because each time we enter a vertex, we consume one of its incident edges, which decreases its degree by one. Similarly, each time we exit a vertex, we consume another incident edge, reducing its degree further.

If all vertices except for two (or zero) had even degrees, we would be able to enter and exit each vertex without getting stuck. However, if more than two vertices had odd degrees, we would encounter a situation where we couldn't exit a vertex without leaving some edges unvisited, or we would be unable to enter a vertex without revisiting an already traversed edge.

Therefore, a connected graph admits an Eulerian chain if and only if the number of vertices with odd degree is 0 or 2. If there are no vertices with odd degree, we can start and end at any vertex, traversing all edges. If there are exactly two vertices with odd degrees, we can start at one of them, traverse all edges, and end at the other odd-degree vertex.

3.2 A connected graph admits a Eulerian cycle if and only if the number of vertices of odd degree is 0 (all vertices have an even degree). To understand why a connected graph admits an Eulerian cycle if and only if the number of vertices with an odd degree is zero, let's first define some terms.

A connected graph is a graph where there is a path between any two vertices. In other words, you can reach any vertex from any other vertex in the graph.

An Eulerian cycle is a cycle in a graph that visits every edge exactly once and returns to the starting vertex. In other words, it is a closed walk that includes all the edges of the graph.

Now, let's examine why the number of vertices with odd degree is important in determining the existence of an Eulerian cycle.

First, let's assume that a connected graph has an Eulerian cycle. In this case, we can start at any vertex and traverse each edge exactly once, eventually returning to the starting vertex. Since we visit each edge once, every time we enter a vertex, we must leave it as well. Thus, every vertex along the cycle has an even degree.

Now, let's consider the converse. Suppose a connected graph has all vertices with even degrees. We want to show that there exists an Eulerian cycle in this graph.

To construct an Eulerian cycle, we can use the following algorithm:

Start at any vertex in the graph.

Traverse an arbitrary edge from the current vertex to an adjacent vertex.

Remove the traversed edge from the graph.

Repeat steps 2 and 3 until no edges remain in the graph.

Since each vertex in the graph has an even degree, we can always find an unvisited edge from the current vertex because whenever we enter a vertex, there must be an unused edge to leave it. Furthermore, since the graph is connected, we can keep traversing edges until we exhaust all of them.

By the end of this algorithm, we will have visited every edge exactly once, and we will return to the starting vertex, forming a closed walk. This closed walk is an Eulerian cycle.

Now, let's consider the case when a connected graph has one or more vertices with odd degrees.

If there is more than one vertex with an odd degree, it is impossible to form a closed walk that includes every edge exactly once because whenever we enter a vertex with an odd degree, we must leave it as well, but there won't be any unused edges to leave the vertex again.

If there is exactly one vertex with an odd degree, it is still impossible to form a closed walk that includes every edge exactly once because we would start and end at the vertex with the odd degree. Therefore, there would be an unused edge when we reach the starting vertex.

Hence, in a connected graph, the existence of an Eulerian cycle is guaranteed if and only if all vertices have even degrees, i.e., there are no vertices with an odd degree. **(meilleur en maths)**

2.4. Euler's algorithm

2.4.1. Case of an Eulerian cycle:

The graph is connected and all vertices have even degree.

1st Step :

We arbitrarily choose a vertex of the graph and we create a cycle containing edges of the graph at most once, If all the edges of the graph are contained once and only once in the cycle then we are finished because we have Eulerian cycle. Otherwise we go to the second step.

2nd Step :

We choose a vertex of the previous cycle for which it is possible to create a closed chain (cycle) of origins. Beginning and end of this vertex and containing edges of the graph at most once and not being contained in the previous cycle By grouping the two cycles, we obtain a new cycle (of length the sum of the two lengths of the previous rounds). If all the edges are contained once and only once in the new cycle then the new cycle is Eulerian . Otherwise, the second step is repeated with the new cycle.

2.4.2 Case of an Eulerian chain

The graph is connected and two vertices (and only two) have odd degree.

1st Step :

Choose a chain that contains an edge of the graph at most once and connects two vertices of odd degree.

If all edges of the graph are included in that chain, then the chain is Eulerian. Otherwise, proceed to the second step.

2nd Step :

Select a vertex of the previous chain that can form a closed chain (cycle) of origin and endpoints, and select the edge of the graph containing this selected vertex at most once, so that it is not included in the previous sequence. Eulerian Chain - Eulerian Cycle

Combining two chains yields a new chain (whose length is the sum of the two lengths of the previous chain). If every edge of the graph is contained in the new chain only once, then the chain is Eulerian. Otherwise, redo the second step with the new chain. **(meilleur en maths)**

2.5. Eulerian circuits

Let us return to our analysis of the Königsberg bridge problem. What the Königsbergers were looking for was a closed trace containing all the edges of the graph. As we have seen, a necessary condition for the existence of such a trace is that the degree of all vertices be even. It is also necessary that all edges belong to the same component of the graph.

The Swiss mathematician Leonhard Euler (pronounced "Euler") stated that these conditions are also sufficient conditions (DOUGLAS, 2001). In honor of his contributions, we bear his name on these graphs. Euler's paper, published in 1741, did not give a proof of the sufficiency

of the obvious necessary conditions; Hierholzer (DOUGLAS, 2001) gave the first complete proof. The graph we drew to model the city in Figure 02 did not appear in print until 1894.

2.5.1 Definition

An Eulerian circuit refers to a graph that contains a closed trail, which includes all of its edges. This closed trail is called a circuit if it is cyclic and does not specify a particular starting vertex. In essence, an Eulerian circuit or locus is a circuit that encompasses all edges of a given graph.

Furthermore, an even graph is defined as a graph in which all vertices have an even degree. Conversely, if the degree of a vertex is odd (even), then the number of vertices with an odd (even) degree is also odd (even).

It is worth noting that the concept of an Eulerian circuit can also be applied to graphs that include loops. In such cases, the definition of vertex degree is extended to account for loops, with each loop contributing a degree of 2 to the associated vertex. This extension maintains the parity of the degree, ensuring that the presence of a loop does not impact whether the graph has an Eulerian circuit, unless the loop consists of components with a single vertex.

To prove properties of Eulerian graphs, a lemma is commonly employed. This lemma relates to maximal paths within a graph. A maximal path refers to a path (sequence of vertices) within a graph that cannot be extended further to form a longer path. Since finite graphs cannot have infinitely extended paths, there will always exist a maximal (non-extendable) path within a finite graph.

By leveraging these concepts and employing the lemma, one can establish various features and properties of Eulerian graphs. These findings contribute to a deeper understanding of the structure and characteristics of graphs and facilitate the identification and analysis of Eulerian circuits in graph theory. (DOUGLAS, 2001).

2.6. Eulerian numbers

Consider the scenario where we need to install an organ with n pipes in a concert hall. Each pipe has a distinct length, and the pipes must be arranged in a row. We define "ascending" as two adjacent pipes where the left pipe is shorter than the right pipe, and "descending" as two adjacent pipes where the left pipe is taller than the right pipe.

If we arrange the pipes from shortest to tallest, there will be n - 1 possible arrangements. In this case, there are no ascending or descending patterns present. Similarly, if we arrange the pipes from tallest to shortest, we also obtain n - 1 possible arrangements, with no ascending or descending sequences.

Now, let's introduce the concept of a specific requirement set by the eccentric director of the concert hall. The director, for aesthetic or acoustic reasons, demands exactly k ascensions in the arrangement of the n pipes. The question arises: How many different ways can we install the organ while satisfying these conditions? The answer is given by the Eulerian number (Eulerian(n, k)).

To elaborate more abstractly, the Eulerian number represents the count of permutations of the integers {1, 2, ..., n} where exactly k numbers between 1 and n-1 satisfy the condition (i) < (i + 1). In other words, the Eulerian number calculates the number of arrangements that exhibit the desired k ascensions within the given constraints.

Mathematically, the Eulerian number (Eulerian(n, k)) can be computed using the formula:

Eulerian(n, k) = (n - k) * Eulerian(n - 1, k - 1) + (k + 1) * Eulerian(n - 1, k)

with the base cases:

Eulerian(n, 0) = 1 if n > 0

Eulerian(0, k) = 0 if k > 0

By recursively applying the Eulerian number formula and considering the base cases, we can determine the total number of valid arrangements that fulfill the director's requirement of exactly k ascensions in the arrangement of n pipes.

This mathematical concept finds applications in various scenarios, particularly in combinatorics and permutation analysis. It allows us to quantitatively explore and analyze different possibilities and configurations within a given set of constraints...**(John M, 2008)**

2.7. Eulerian digraphs

The following theorem characterizes Eulerian graphs.

Theorem: A connected graph D(V, A) is Eulerian if and only if indeg(v) = outdeg(v) for all vertices v(V).

Proof First, assume that D is Eulerian: while traversing C, each time it encounters a vertex v, it passes through an arc toward v and then through an arc away from v. Thus, indeg(v) = outdeg(v) for all vertices.

Let us assume that indeg(v) = outdeg(v) for all vertices v in D (Saidur, 2017).

Euler discovered the relation between the number of vertices, edges, and regions in a graph, and his discovery is often referred to as Euler's formula.

Theorem: (Euler's formula): if G is a connected planar graph with n vertices, q edges, and r domains, then

n - q + r = 2

Proof: Using induction on the number q of edges, if q = 0, then G must be K_1 . The result still holds in this case. Let us assume that the result holds for all connected planar graphs with fewer than q edges, and let G have q edges.

Case 1: Suppose G is a tree. Of course, the planar representation of a tree has only one domain, so r = 1.

Therefore, n - q + r = n - (n - 1) + 1 = 2, and the result holds.

Case 2: Suppose G is not a tree; let C be a cycle of G and e an edge of C. Consider the graph G - e. Compared to G, this graph has the same number of vertices, one less edge and one less domain:

n - (q - 1) + (r - 1) = 2

n - q + r = 2.

The result holds in both cases and induction is complete.. **(John M, 2008)**

2.8. Euler tours

An Euler tour refers to the path traced along all the edges of a graph. This path, known as the Eulerian locus, was first investigated by Euler himself. In his initial exploration of graph theory, presented in his 1736 paper, Euler demonstrated that it was impossible to cross each of the seven bridges in Königsberg only once while traversing the town. To illustrate this, a plan of Königsberg and the Pregel River was depicted in Figure 02.

Proving the impossibility of such a walk is essentially equivalent to establishing that the graph shown in Figure 15b does not possess Euler's trajectory.

In the context of graph theory, a tour of a graph refers to a closed walk, where each side of the graph is crossed at least once. Specifically, an Euler tour is a tour that traverses each edge of the graph exactly once, resulting in a closed Euler trail. Therefore, if a graph contains an Eulerian tour, it is classified as Eulerian.

Theorem: A nonempty connected graph is eulerian if and only if it has no vertices of odd degree.**(J.A Bondy, 1976)**

2.9. Euler's Formula

Euler discovered the relation between the number of vertices, edges, and regions in a graph, a discovery often referred to as Euler's formula.

Theorem: (Euler's formula): If G is a connected planar graph with n vertices, q edges and r domains, then

$$n - q + r = 2$$

Proof: Using induction on the number q of edges, if q = 0, then G must be K_1 . The result still holds in this case. Let us assume that the result holds for all connected planar graphs with fewer than q edges, and let G have q edges.

Case 1: Suppose G is a tree. Of course, the planar representation of a tree has only one domain, so r = 1.

Therefore, n - q + r = n - (n - 1) + 1 = 2, and the result holds.

Case 2: Suppose G is not a tree; let C be a cycle of G and e an edge of C. Consider the graph G - e. Compared to G, this graph has the same number of vertices, one less edge and one less domain:

$$n - (q - 1) + (r - 1) = 2$$
and
$$n - q + r = 2.$$

The result holds in both cases and induction is complete..**(John M, 2008)**

2.10. depth-first search on a graph

The depth-first search (DFS) algorithm plays a crucial role in finding Eulerian paths and circuits in graphs. An Eulerian path is a path in a graph that traverses each edge exactly once, while an Eulerian circuit is an Eulerian path that starts and ends at the same vertex. In this

long text, we will explore how DFS is utilized to recursively find Eulerian paths and circuits, and discuss the relationship between Eulerian graphs and this algorithm.

To begin, let's consider an Eulerian graph, which is a graph that contains an Eulerian circuit. In an Eulerian graph, every vertex is of even degree, meaning that the number of edges incident to each vertex is an even number. This property is essential for the existence of Eulerian circuits, as the even degree ensures that we can enter and exit each vertex without leaving any unvisited edges.

Now, let's discuss how DFS can be applied to find an Eulerian path or circuit in a graph. The basic idea is to traverse the graph using DFS, making sure to explore all possible edges while keeping track of the visited edges. Here's how the algorithm works:

1. Start the DFS from any vertex in the graph. Initialize an empty stack to keep track of the visited edges.

2. While traversing the graph using DFS, at each vertex, choose an unvisited edge and follow it to the next vertex.

3. Whenever an edge is visited, push it onto the stack.

4. If a vertex is reached where there are no unvisited edges, it means we have encountered a dead end. At this point, backtrack by popping edges from the stack until we find a vertex with unvisited edges.

5. Continue the DFS traversal until all edges have been visited.

At the end of this DFS traversal, the stack will contain the Eulerian path in reverse order. To obtain the correct order, we can simply reverse the stack.

However, there is a caveat when it comes to finding Eulerian circuits using DFS. If the graph contains multiple connected components, we need to ensure that every component is Eulerian. If any component is not Eulerian, it means that there is no Eulerian circuit in the graph.

To handle this scenario, we can modify the DFS algorithm as follows:

1. Start the DFS from any vertex in the graph.

2. While traversing the graph using DFS, at each vertex, choose an unvisited edge and follow it to the next vertex.

3. Whenever an edge is visited, push it onto the stack.

4. If a vertex is reached where there are no unvisited edges, it means we have encountered a dead end. At this point, backtrack by popping edges from the stack until we find a vertex with unvisited edges.

5. After the backtracking step, if there are still unvisited edges in the graph, choose a vertex from the current connected component that has unvisited edges and start a new DFS traversal from that vertex.

6. Repeat steps 2-5 until all edges have been visited in all connected components.

After this modified DFS traversal, if there are any remaining unvisited edges, it implies that the graph is not Eulerian and does not contain an Eulerian circuit. However, if all edges have been visited, we can combine the stacks obtained from each connected component to form the Eulerian circuit.

In summary, the DFS algorithm, when applied recursively, allows us to find Eulerian paths and circuits in graphs. By carefully traversing the graph, exploring all edges, and using backtracking to handle dead ends, we can systematically uncover the Eulerian structure of a graph. It is important to note that DFS alone cannot make a graph Eulerian, as the graph must satisfy the condition of even degrees for each vertex. DFS simply helps us navigate through the graph to identify and construct Eulerian paths and circuits when they exist.

The relationship between Eulerian graphs and the DFS algorithm lies in the fact that DFS provides a means to explore and uncover the Eulerian structure within a graph. By applying DFS, we can effectively traverse the graph and identify the necessary conditions for Eulerian paths and circuits. The algorithm's backtracking nature allows us to handle the intricacies of dead ends and disconnected components, ensuring that we cover all edges and vertices while finding the Eulerian path or circuit.

In conclusion, the DFS algorithm is an invaluable tool in graph theory, particularly when it comes to finding Eulerian paths and circuits. Its recursive nature, combined with careful edge selection and backtracking, enables us to efficiently navigate through a graph and construct Eulerian paths and circuits when the underlying graph satisfies the necessary conditions. DFS serves as a fundamental technique in uncovering the fascinating properties of Eulerian graphs. [NARSINGH DEO, 1974].

2.11. Introduction to LATEX

2.11. 1 What is LATEX?

LATEX, pronounced "lay-tek," is a typesetting system commonly used for creating high-quality documents, particularly those containing mathematical equations and scientific content. It was developed by Leslie Lamport in 1982 as a set of macros built on top of the TeX typesetting system, originally created by Donald E. Knuth.

The motivation behind the creation of LATEX was to simplify the process of typesetting and formatting documents, especially those with complex mathematical formulas. TeX, developed by Knuth in the late 1970s, was already a powerful typesetting system that focused on producing high-quality output. However, it required users to have in-depth knowledge of its intricate syntax and low-level commands to create documents. Lamport recognized the need for a more user-friendly and structured approach, leading to the development of LATEX.

LATEX provides a higher-level set of commands and macros that abstracts away many of the low-level details of TeX, making it more accessible to users. It introduces a document markup language, where users write their content using plain text combined with LATEX commands to specify the structure, formatting, and mathematical expressions within the document.

One of the significant advantages of LATEX is its ability to handle complex mathematical formulas and equations seamlessly. It offers an extensive range of mathematical symbols, operators, and formatting options, allowing users to express mathematical concepts with precision. The LATEX typesetting engine automatically handles the placement and alignment of equations, ensuring professional-looking output.

Moreover, LATEX excels in the typesetting of scientific and technical documents, such as research papers, theses, reports, and books. It provides extensive support for bibliographies, cross-referencing, tables of contents, indexes, and various other features commonly found in academic publications.

The LATEX system operates by compiling source files written in plain text with LATEX markup into a variety of output formats, including PDF, DVI (Device Independent), and others. Users typically write their documents in a text editor using LATEX syntax and

then compile the source code using a LATEX distribution such as TeX Live, MiKTeX, or MacTeX.

Due to its power, versatility, and the quality of its output, LATEX has become the standard typesetting system in many academic and scientific disciplines. It is widely used by researchers, scientists, mathematicians, engineers, and professionals who require precise and aesthetically pleasing document formatting, particularly when dealing with mathematical and technical content.

In summary, LATEX is a typesetting system built on top of TeX, designed to simplify the creation of complex documents and mathematical expressions. It provides a higher-level markup language, extensive mathematical capabilities, and produces professional-quality output. Its wide adoption in academic and scientific communities is a testament to its effectiveness and reliability for producing beautifully formatted documents. (**Linda Chan-Sun, 2004)**

2.11. 2 Principle

LATEX can be considered a high-level programming language because it relies on TeX, a low-level language. This means that the document you want to create must be written in a source file (e.g., my_file.tex), composed of a set of LATEX commands (tags), and compiled; the LATEX compiler will output a device-independent file (DVI) (my_file.dvi). This file can be converted to PostScript or PDF format for printing or output. Most LATEX commands begin with a "backslash," with required arguments enclosed in curly braces ({ and }) and optional arguments enclosed in square brackets ([and]). Example \documentclass[12pt]{report} **(frederic Gerdends, 1997)**

2.11. 3 Some compilation commands

2.11.3.1 Under UNIX/Linux

1. to compile the source file (file.tex): latex file.tex

In case of an error, the line with the error is indicated. The position of the error within the line is indicated by a new line. A brief description of the error is also displayed. The user is then free to use several commands:

- "?" : The help menu is displayed.

- "h" : a detailed description of the error that LaTeX stopped is displayed.

- "return" : forces the compile to continue.

- "s" : Displays the following error messages.

- "r" : Continue compiling without stopping.

- "q" : Continue compilation without displaying the message.

- "i" : insert something (e.g. forgotten tag) to continue compiling.

- "e" : Edit source files.

- "x" : Abort compilation.

- A number from 1 to 9, ignoring the next x characters in the source. 2.

2. 2. compile references: bibtex file.

3. to view DVI files: xdvi file.dvi.

4. to convert DVI file ←→ PS: dvips file.dvi.

To print a PS file: lp -d <my_printer> file.ps. 6.

Convert DVI file ←→ PDF: dvipdf file.dvi

7. to convert a LATEX source directly to PDF: pdflatex file.tex

Finally, the following steps are required to generate a complete PostScript document

- $ latex file.tex

- $ latex file.tex

- dvips file.dvi

The second step restores the cross-references and table of contents (without this recompilation, the "?) **(Leslie Lamport , 1994)**

2.11. 3.2 Under Windows

It was enough to install the freeware compiler MikTeX 1. Later, the first easily usable editor was WinEdt 2. It has the disadvantage of being shareware, so today freeware alternatives such

as TeXnicCenter3 and MeVa4 are preferred.Ghostview5 is a freeware software that allows viewing files in PostScript format. **(Leslie Lamport , 1994)**

2.11. 3.3 Under Mac OS X

Just install MacTeX6 and TexShop7 to get the freeware integrated environment. don't forget to change the default encoding (MacOSRoman) to Latin-1 in TexShop's preferences. since I wanted to use Emacs as well. every time I compiled. I looked for a freeware dvi/pdf viewer that would update directly. This is how I found TeXniscope8. **(Leslie Lamport , 1994)**

2.10.4 General structure of a LATEX document:

A very simple first example:

- % This is a comment
- % Header of any LaTeX document. Specifies the type of written document
- \documentclass[11pt,a4paper]{article}
- \begin{document} % marks the beginning of the text to be composed
- The body of the document....
- \end{document} % marks the end of the document

1) http://www.miktex.org
2) http://www.winedt.com/
3) http://www.toolscenter.org/
4) http://www.meshwalk.com/latexeditor/
5) http://www.cs.wisc.edu/~ghost/
6) http://www.tug.org/mactex/
7) http://www.uoregon.edu/~koch/texshop/
8) http://www2.ing.unipi.it/~d9615/homepage/texniscope.html

➢ LaTeX documents always begin with the documentclass command, which specifies the document class (braces). The most commonly used classes are article, report, letter, and book. The options for this command are declared within square brackets. The most common options are 10pt, 11pt, 12pt (to determine nominal font size), a4paper (to determine paper dimensions), french, twocolumn (for two-column text layout), and twoside (for two-sided writing).

- In practice, the header can define many settings, such as the package to be used (with the \usepackage command), command redefinition (see §5), title, bibliography style, and so on.
- The following example should be suitable for most applications:
- ¬ documentclass[11pt,twoside,a4paper]{article}
- ¬ %---include package (optional)---
- ==========
- ¬ usepackage[french]{babel} to specify that it is in French.
- ¬ usepackage{a4} paper size
- ¬ ¦ Specify PostScript font with [T1]{fontenc} %.
- ¬ % TrueType vectorial font, French quotation marks
- ¬ % to manage images % % usepackage{epsfig}
- ¬ usepackage{amsmath, amsthm} % Very good math mode
- ¬ ¦ package{amsfonts,amssymb} % for defining sets
- ¬ ¦ For the arrangement of figures
- ¬ ¦ usepackage{url} % for efficient management of URLs
- ¬ % Bibliographic style % Bibliographic style
- ¬ %--- for titles
- ¬ title{document title} author{author name
- ¬ author{Sebastien Varrette <\url{Sebastien.Varrette@imag.fr}>}
- ¬ ●begin{document}
- ¬ maketitle % write title
- ¬ tableofcontents write the table of contents.
- ¬ section{first section}
- ¬ subsection{first subsection} The body of the text...
- ¬ end{document}. **(Linda Chan- Sun, 2004)**

2.11. 5 Common Packages

10.5.1. **aeguill:** with the cyr option, includes the ae package to produce quality PDF documents by adding French quotes « and ».

10.5.2. **amsmath, amsthm, amsfonts, amssymb:** American Mathematical Society extensions that provide a set of commands for mathematical mode.

10.5.3. **babel:** adapts the names of chapters, dates, and other texts inserted by LATEX in the language passed as an option.

10.5.4. **Color:** for the use of colors.

10.5.5. **draftcopy:** prints the word "DRAFT" in the background of the page.

10.5.6. **epsfig:** for the management of graphics in eps format.

10.5.7. **fancybox:** adds several page framing commands.

10.5.8. **float:** improves the management of floating elements such as tables and figures.

10.5.9. **fontenc:** with the T1 option, allows the compiler to use the new font encoding format. This package should be used systematically.

10.5.10. **Graphics:** provides several commands for manipulating boxes and graphics.

10.5.11. **Import:** for managing subdirectories.

10.5.12. **Listing:** for optimized display of source codes.

10.5.13. **Minitoc:** allows the construction of a mini table of contents at the beginning of each chapter under the book and report classes.

10.5.14. **Multirow:** for multi-row table cells.

10.5.15. **Rotating:** for rotating tables, figures, and legends.

10.5.16. **url:** allows URLs to be displayed correctly. **(Linda Chan- Sun, 2004)**

2.12. The Basics:

2.11.1 Reserved Characters

There are a number of characters reserved by LATEX because they introduce a command. They are summarized in the following table. All other characters can be used freely.

- % Comment
- \ Command
- {...} Block of processing
- ~ Non-breaking space
- $ Mathematical mode
- & Table alignment marker
- # Macro parameter
- ^ and Exponentiation and subscripting

2.12.2 Special characters

The characters coded in ISO-8859-1 are understood by the compiler: common accented characters can be entered directly. However, there are a number of special characters summarized in Table 1. (Christian Rolland, 1999)

Table (01): Special characters in latex

ö	\"{o}	ȯ	\.{o}	ŏ	\u{o}	č	\v{c}
ő	\H{o}	o͡o	\t{oo}	ç	\c{c}	ọ	\d{o}
o̲	\b{o}	ō	\={o}				
œ, Œ	\oe, \OE	†	\dag	æ, Æ	\ae, \AE	‡	\ddag
å, Å	\aa, \AA	§	\S	ø, Ø	\o, \O	¶	\P
ł, Ł	\l, \L	©	\copyright	ß	\ss	£	\pounds
¿	?‘	ı	\i	¡	!‘	ȷ	\j
#	\#	$	\$	%	\%	&	\&
_	_	{	\{	}	\}	\	\textbackslash

2.12.3 Common commands

Table (02) : Common commands in latex

‚documentclass{...}	Définit la classe de document
‚usepackage{...}	Charge un package
‚title{...}	Description du titre du document
‚author{...}	Description de l'auteur
‚date{...}	Date de rédation
‚maketitle	Ecrit le titre
‚tableofcontents	Ecrit la table des matières
‚listoffigures	Ecrit la liste des figures
‚listoftables	Ecrit la liste des tableaux
‚TeX, \LaTeX, \LaTeXe	TEX, LATEX, LATEX 2_ε
‚verb!...!	Mode `verbatim` en ligne – voir §2.7.4
‚begin{*env*}...\end{*env*}	Délimitation du bloc d'environnement *env*

2.13. Saces and line breaks in the source file

Care should be taken with the use of commands because LATEX ignores a space immediately following the command when it is inserted into the text. Consider the following two examples:

- \LaTeX is great. LATEX is great.

- \LaTeX\ is great. LATEX is great.

In addition, LATEX considers carriage returns, tabs, and a succession of empty spaces as a single empty space. Thus, by typing:

- This is a test of spaces.
- This is an example of a line break.\\
- This is the beginning of a new paragraph.

we obtain:

- This is a test of spaces.
- This is an example of a line break.
- This is the beginning of a new paragraph. **(Christian Rolland, 1999)**

2.14. Hierarchical structure of the document

The structure of a LATEX document is based on the use of chapter commands. These are fully controlled by LATEX (especially with regard to indentation and numbering). The possibilities for subdivisions are summarized below, some of which are only available in certain styles.

\part{} % part

- \chapter{} % chapter
- \section{} % section
- \subsection{} % subsection
- \subsubsection{} % subsubsection (subsection level 2)
- \paragraph{} % paragraph (subsection level 3)
- \subparagraph{} % subparagraph (subsection level 4)
- \appendix % signals the beginning of the appendices

Don't forget to put the chapter, section title, etc., in braces.

You may wish to remove the numbering suggested by LATEX. To do this, simply place a "*" character before the section title. Thus, in the following example, the command section*{Acknowledgements} would remove the section numbering. **(Thomas Nemeth, 2000)**

2.14.1 Inserting files

When writing long documents, working with a single file is not fun. In fact, the time savings can be long and navigation is not always easy. Therefore, we have prepared the following command:

- input{file} is replaced by the contents of file.tex;

- include{file} inserts file.tex; include{file} inserts file.tex; include cannot be written in the header, but includeonly{file1,file2,...} . .

Note that the import package provides the command␣subimport{dir/}{file} or␣subincludefrom{dir/}{file} to include file.tex relative to the subdirectory dir/. **(Thomas Nemeth, 2000)**

Conclusion

In conclusion, this chapter has provided a thorough exploration of the concept of Eulerian paths and cycles in graphs. We have defined and discussed the properties of Eulerian chains, which traverse each edge of a graph exactly once, and Eulerian cycles, which are closed paths visiting every vertex and edge exactly once. Euler's theorems have been introduced as fundamental conditions for the existence of Eulerian paths and cycles in a graph.

The chapter has delved into the Eulerian algorithm, a systematic procedure for identifying and constructing Eulerian paths and cycles in graphs. This algorithm plays a crucial role in determining whether a given graph possesses an Eulerian circuit. Moreover, we have covered essential concepts related to Eulerian circuits, including Eulerian numbers, Eulerian graphs, Eulerian formulas, Eulerian tours, and Eulerian formulas. These concepts provide deeper insights into the properties and characteristics of Eulerian graphs.

Furthermore, we have discussed the significance of the depth-first search (DFS) algorithm in graph traversal. DFS serves as a powerful tool for systematically exploring vertices and edges in a graph. Its application extends beyond Eulerian paths and cycles, finding utility in various other graph-related tasks and research projects. Additionally, we have mentioned the utilization of LaTeX commands for formatting mathematical expressions and equations in our thesis, ensuring precise and professional representation of mathematical content.

By covering these topics comprehensively, we have acquired a solid understanding of Eulerian paths and cycles, along with the associated theorems, algorithms, and fundamental concepts. The knowledge and techniques presented in this chapter lay a robust foundation for the subsequent sections of our thesis. In these upcoming sections, we will delve further into the analysis and application of Eulerian graphs, exploring their significance in diverse domains such as network analysis, graph theory, and optimization problems.

The study of Eulerian paths and cycles opens up avenues for solving complex problems in various fields, including transportation planning, circuit design, and data analysis. The ability to identify and construct Eulerian paths and cycles provides insights into the connectivity and structure of graphs, offering valuable information for optimizing network

flows, designing efficient algorithms, and understanding relationships within complex systems.

As we progress in our research, we will leverage the knowledge gained in this chapter to investigate specific applications of Eulerian graphs and their associated algorithms. This will involve exploring real-world datasets, applying graph theory principles, and employing computational techniques to solve complex problems. By building upon the foundations established here, we will contribute to the advancement of knowledge in the field of Eulerian graphs and their practical implications.

In summary, this chapter has presented a comprehensive overview of Eulerian paths and cycles, Euler's theorems, the Eulerian algorithm, and the depth-first search algorithm. We have discussed the significance of LaTeX for mathematical formatting and emphasized the relevance of these concepts in our thesis research. Armed with this knowledge, we are well-equipped to delve deeper into the analysis, applications, and advancements in Eulerian graph theory in the subsequent sections of our thesis.

Chapter 3:

Implementation of an Eulerian chain application.

Chapter 3: Implementation of an Eulerian chain application

Introduction

This chapter marks the conclusion of our final year project focused on the development of an automatic solver based on graph theory, specifically the Eulerian path problem. Our goal was to create a system capable of receiving a typical exercise statement on Eulerian paths as input and providing a solution that closely resembles a typical correction. This solution would enable students to verify their own answers, while allowing teachers to automate a part of the correction process.

In the previous chapters, we extensively explored graph theory and the fundamental concept of Eulerian paths.

Our system is based on a set of rules and heuristics specifically designed to identify and solve Eulerian paths in a given graph. When an exercise statement is submitted to the system, it analyzes the provided information and applies appropriate techniques to generate a solution. This solution is then presented in a form similar to a typical correction, allowing users to compare their own results with the proposed solution.

One major advantage of our automatic solver is its ability to handle a wide range of typical Eulerian path exercises. Whether it's simple graphs or more complex ones, the system is capable of providing accurate and reliable solutions. Additionally, it offers the necessary flexibility to accommodate various exercise-specific constraints, such as connectivity restrictions or edge weights.

By automating a part of the correction process, our system allows teachers to save valuable time. They can focus on more pedagogical tasks, such as analyzing common errors and providing personalized feedback to students. At the same time, learners benefit from an additional tool to assess their knowledge and practice autonomously.

In this final chapter, we will present the results of our experiments and evaluate the performance of our automatic solver. We will also discuss the limitations of our system and prospects for future research and improvements. Finally, we will conclude by highlighting the potential impact of our work on the teaching and learning of graph theory, emphasizing the advantages of automation in the correction process of Eulerian path exercises.

3.1. Interface

3.1.1 Step01

This interface has the following options:

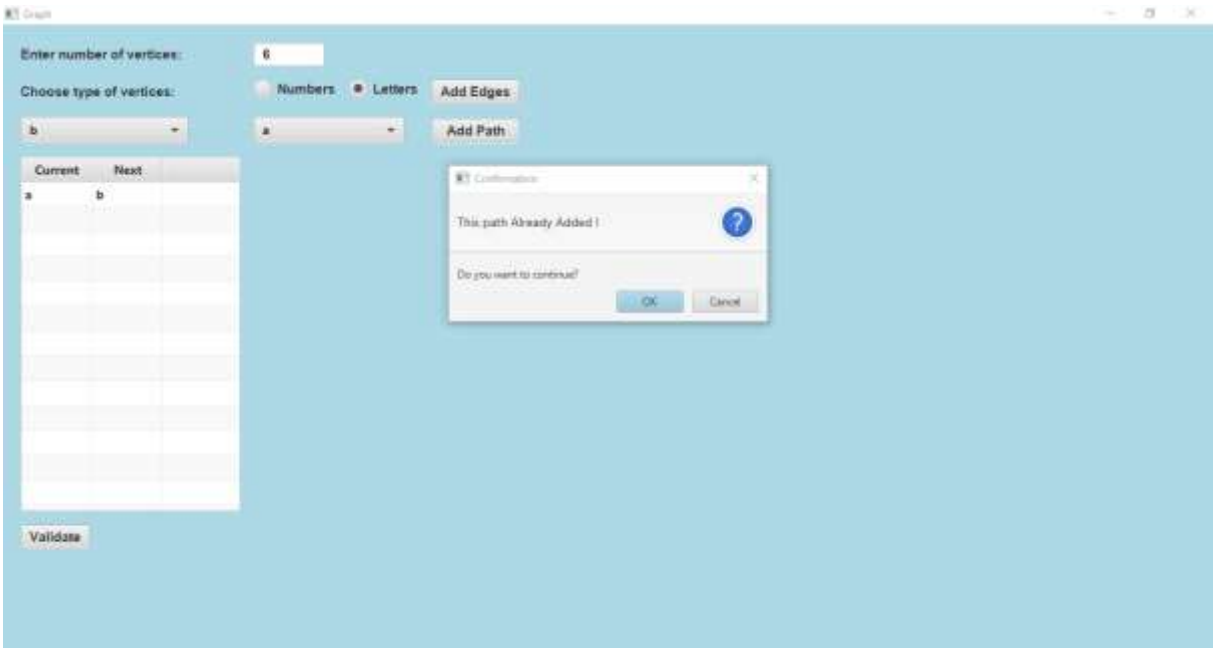
1.1Enter the number of vertices.

1.2Choose a type (numbers or letters)

. 1.3Add edges.

3.1.2 Step02

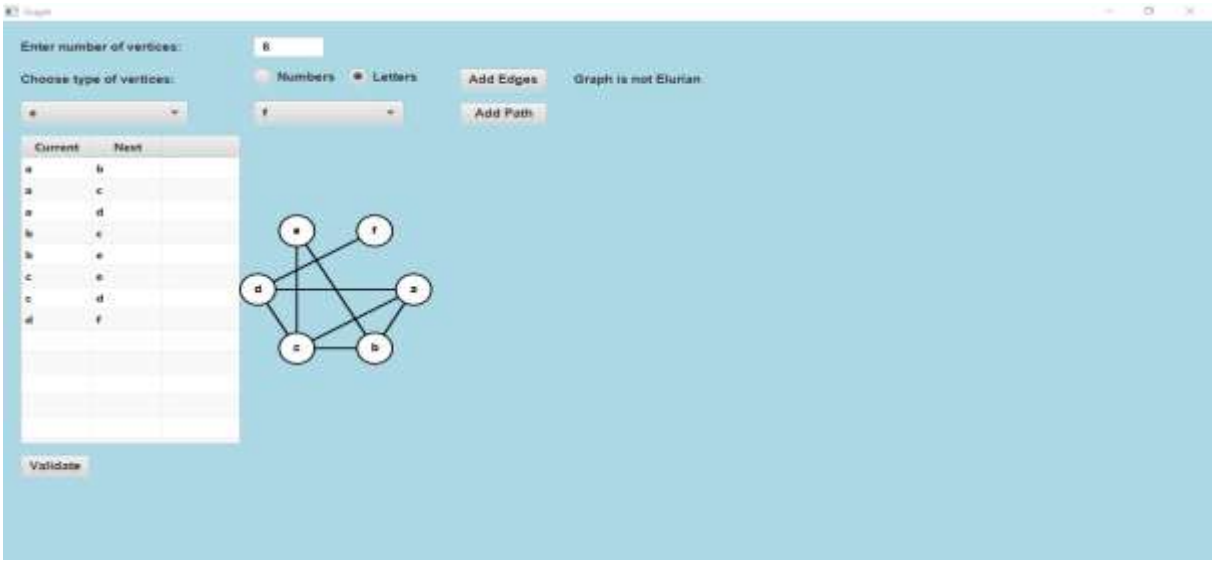
When we enter path **a_b,** After we enter **b_a** it shows this notification.

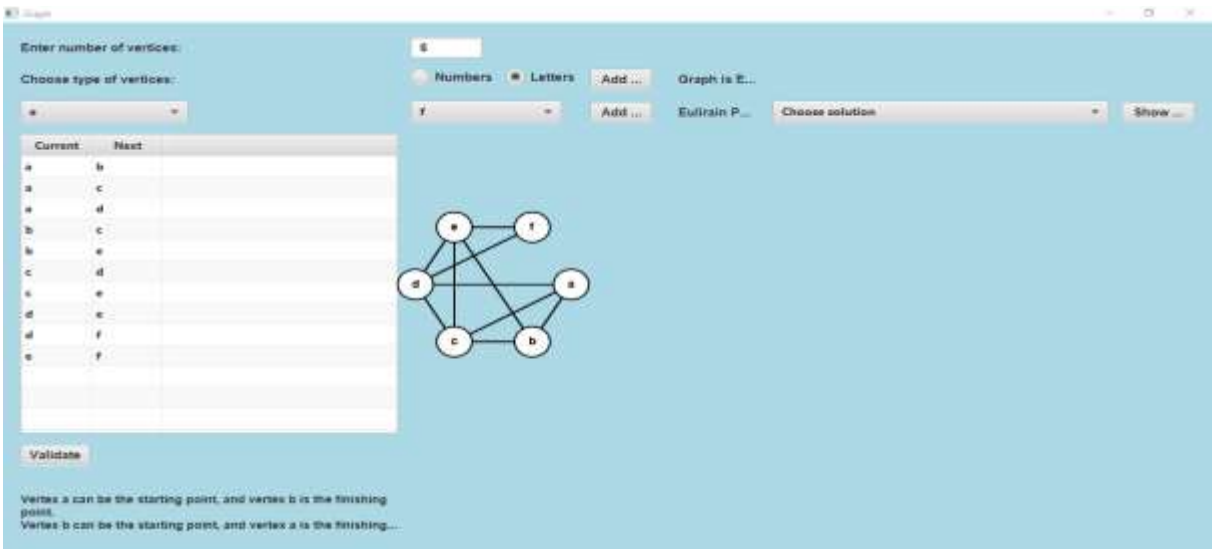


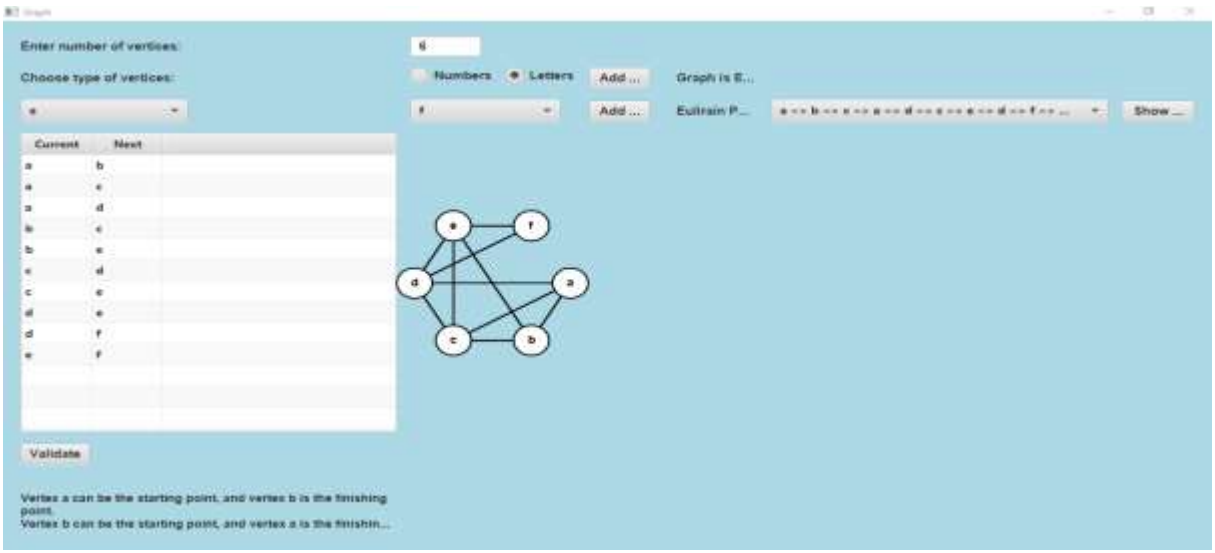
3.1.3 Step03

When we enter number of vertices and we choose type of vertices it showed us this interface named select vertices types

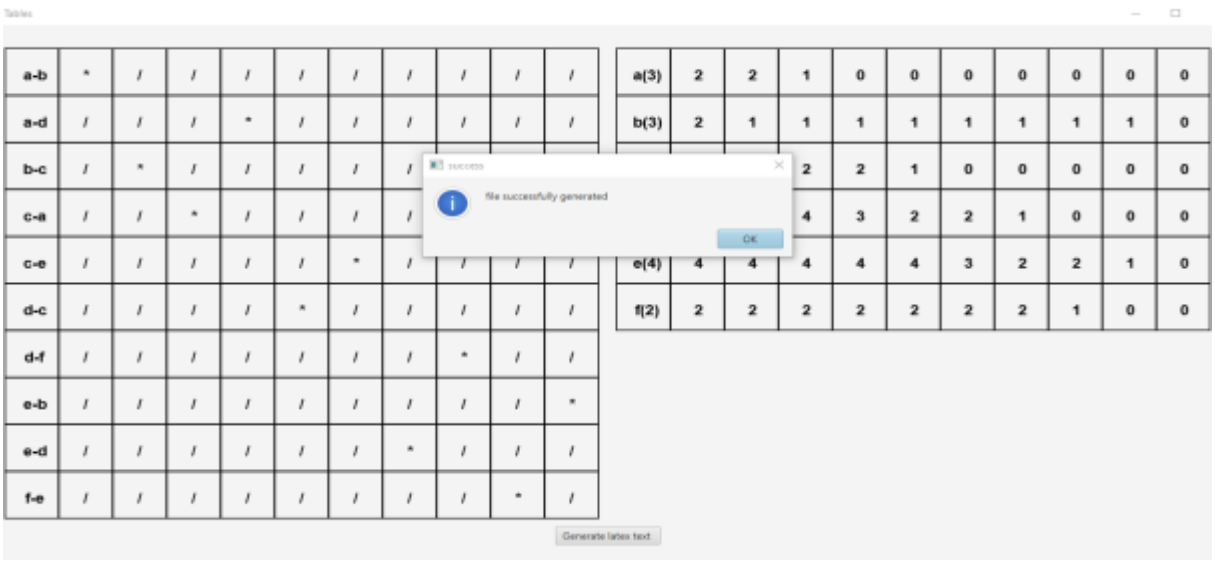
3.1.4Example 01: When we enter the data of graph.



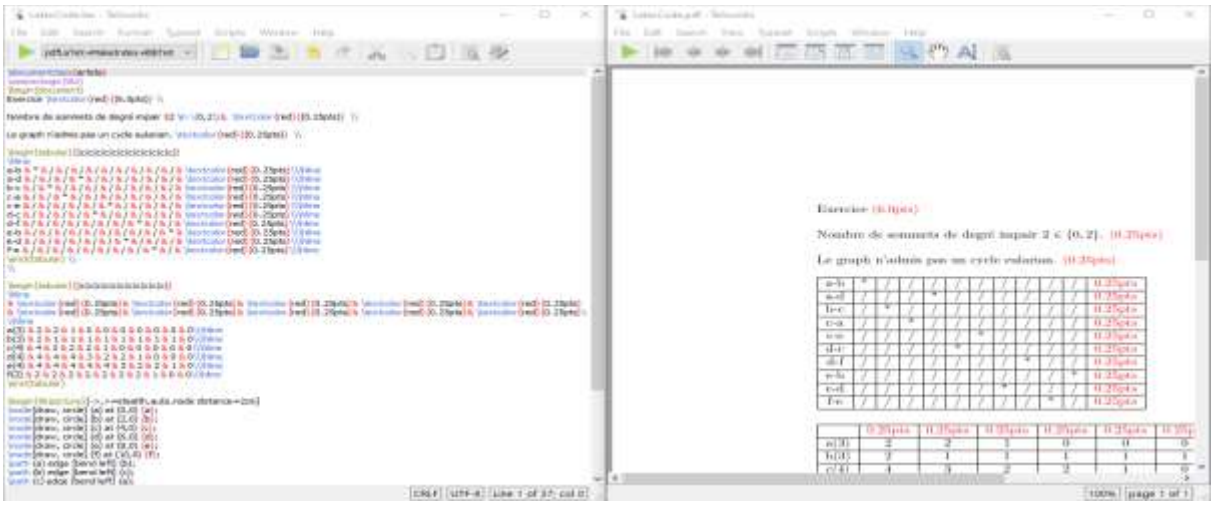




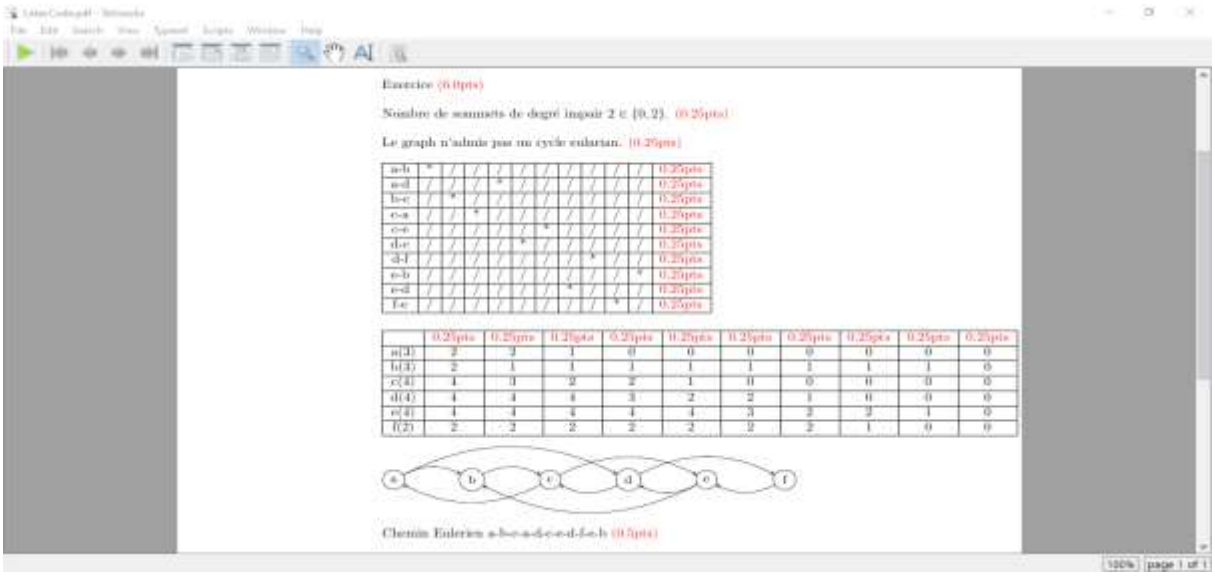
3.1.5 Table of degree and table of path



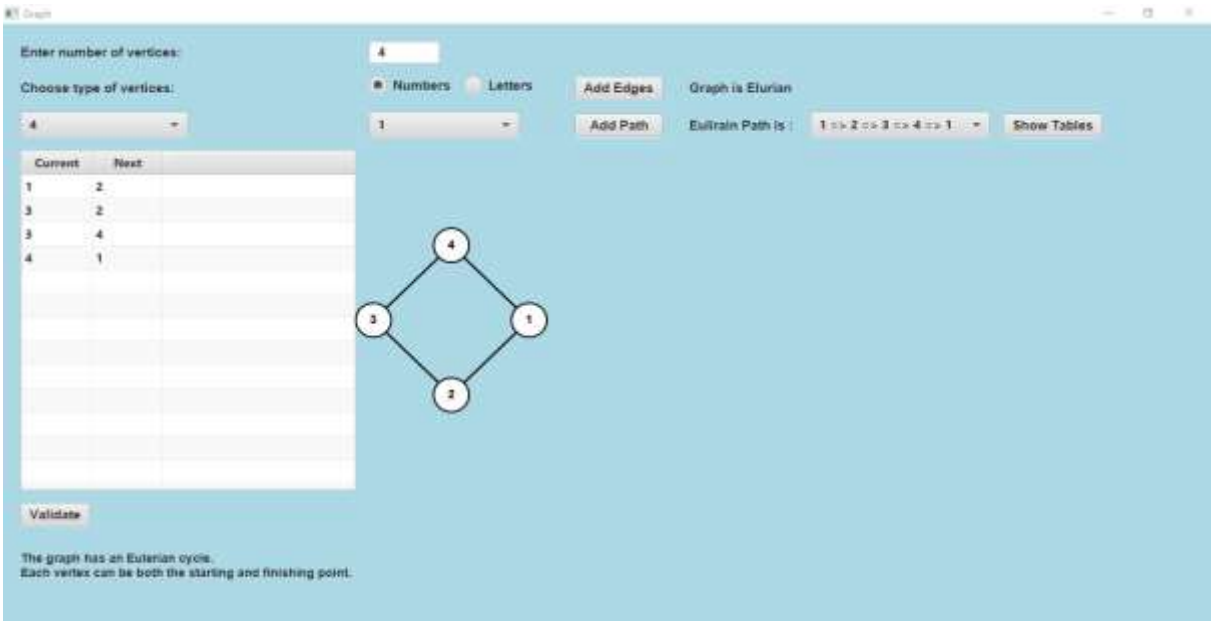
3.1.6 Display in latex



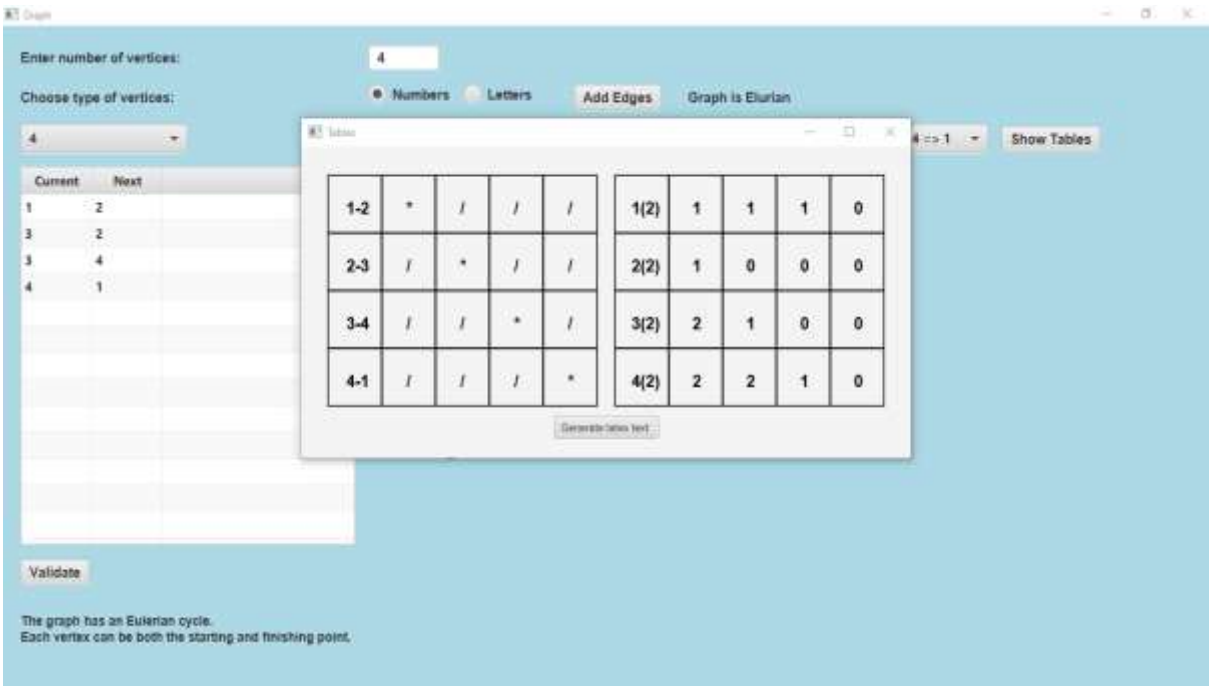
Correction of example 01



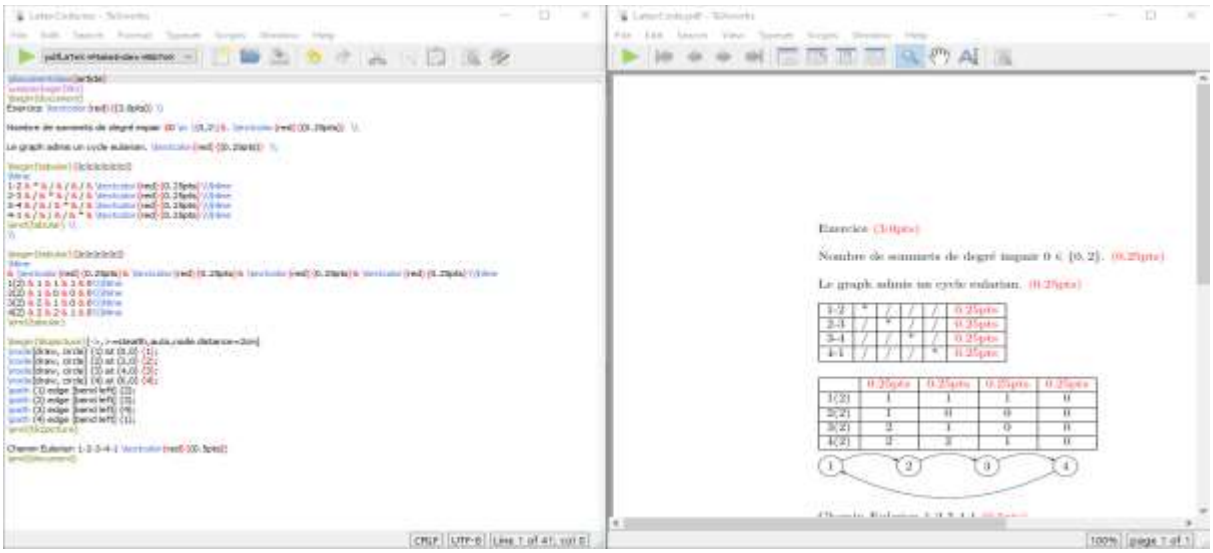
3.1.7 Example 02



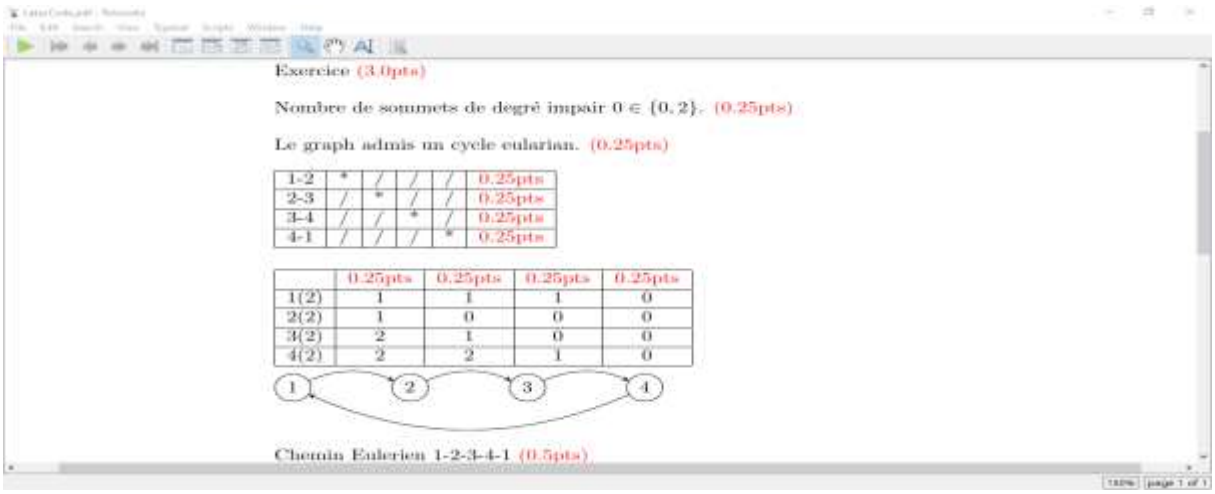
3.1.8 Example 02: Table of path



3.1.9 display of example 02



3.1.10 Correction of example 02



Exercice (3.0pts)

Nombre de sommets de degré impair $0 \in \{0, 2\}$. (0.25pts)

Le graph admis un cycle eulerian. (0.25pts)

1-2	*	/	/	/	0.25pts
2-3	/	*	/	/	0.25pts
3-4	/	/	*	/	0.25pts
4-1	/	/	/	*	0.25pts

	0.25pts	0.25pts	0.25pts	0.25pts
1(2)	1	1	1	0
2(2)	1	0	0	0
3(2)	2	1	0	0
4(2)	2	2	1	0

Chemin Eulerien 1-2-3-4-1 (0.5pts)

3.2 Conclusion

In conclusion, the final chapter of this research project centered around leveraging the Java programming language to address a specific objective: designing a system that can take a standard graph exercise statement, such as an Eulerian path problem, as input and generate a solution as output. The solution generated by the system closely resembles a typical correction, thereby enabling students to validate their own answers and allowing teachers to automate a portion of the correction process. Through the development of this application, we have successfully bridged the gap between theoretical graph concepts and their practical implementation.

The application of Java has played a pivotal role in creating an efficient and reliable system. Java's robustness, versatility, and extensive library support have proven to be instrumental in implementing complex graph algorithms and data structures required for solving such exercises. By leveraging Java's features, we have been able to develop a system that can accurately solve graph exercises, offering immediate feedback to students regarding the correctness of their work.

The benefits of this system extend beyond student feedback. The automation of the correction process empowers teachers to streamline their evaluation procedures, saving them valuable time and effort. Additionally, by providing a reliable and standardized solution, the system assists teachers in maintaining consistency and objectivity when assessing students' work.

In summary, the application developed in this research project, utilizing the Java programming language, has successfully addressed the objective of designing a system to solve graph exercises. By offering immediate feedback to students and enabling teachers to automate part of the correction process, the system enhances the learning experience. Java's robustness and extensive libraries have played a crucial role in implementing complex graph algorithms, bridging the gap between theoretical concepts and their practical application. The impact of this system extends beyond individual students, empowering teachers to efficiently evaluate and provide feedback on graph exercises.

General Conclusion

General Conclusion

This research paper delves into the automatic solution of graph theory problems, with a specific focus on the Euler path problem. The paper begins by providing essential definitions related to graph theory, including the history of the field, the concept of a graph, various types of graphs, and an explanation of the Euler path problem. Additionally, the paper covers the notion of graph progression, different methods of representing graphs, techniques for manipulating graphs, applications of graphs in various fields, and fundamental terminology associated with graphs.

The main objective of the study is to find an Euler path in a given graph. To achieve this, the system takes a typical exercise statement related to Euler paths as input and produces a solution as output. The generated solution is presented in a format similar to a standard correction, enabling students to validate their own solutions and allowing teachers to automate part of the correction process.

The choice of the Java programming language for implementing the solution is justified due to its robustness, portability, and security features. Java provides a reliable and versatile development platform for building software applications. In this study, a depth-first search algorithm was employed to explore all accessible nodes from the starting node, enabling the identification of an Euler path in the graph.

To ensure professional and high-quality document production, LaTeX was utilized as a typesetting system. LaTeX offers advanced typographical features and is widely used for generating scientific and technical documents.

In summary, this research paper focuses on automating the solution of graph theory problems, specifically the Euler path problem. By adopting the Java programming language and utilizing a depth-first search algorithm, the system can effectively find an Euler path in a graph. The use of LaTeX contributes to the production of professional documents. This research bridges the gap between theoretical concepts and practical implementation, providing a valuable tool for students to validate their solutions and teachers to streamline the correction process.

Reference Lists

References

- A. Bretto A. faisait. F, 2012, Hennecant Elements de theorie des graphes Editeur SPRINGER, collection Iris.
- Abbes Amel, 2021, Mémoire de fin d'étude Master, Thème Le problème de la coloration. par liste sur quelques classes des graphes.
- C. VASUDEV, 2006, Graph Theory with Applications **ISBN 978-81-224-2413-3.**
- Christian Rolland, 1999, la tex par pratique , O'Reilly,. ISBN 2841770737.
- Didier Müller, 2004, Introductions à la théorie des graphes, Cahier NO6, Commission Romande de mathématique, France.
- DOUGLAS B.West, 2001, introduction to graph Theory, Second edition, University of Illinois, Urbana.
- Eric Sigward, 2002, Introduction à ls théorie des graphes, ac-mancy-metz.fr.
- Fatima zohra tebbak, Introduction à la modélisation, Support de cours, déesses Préparatoires Ecole Supérieure en Greniez Electronique et Energétique d'Oran.
- frederic Gerdends, Guide Later, 1997, http://www. spi ens. Fr/" berg / 1999 Fintro/ spiens. Latex.
- Frederic Murnier, Andras Sebô, livre parcours coupes.
- **Frederic Murnier**: site web - [http:// persu.usthb. dz - mboukola / Cours TG. Pdf, Consulté le 25/04/2023]
- J.A BOndy and U.S.R Murty, 1976, Graph Theory with Application, Departement of Combinatorics and Optimization, University of Unterloo, canada.
- Jean Charles Régin, Arnoud Malapert, 2016, Theorie des graphes.

- John M, Harris, Jeffry L Hirst, Michael J. Mossinghoff, 2008, combinatorics and Graph Theory, Second edition.
- Leslie Lamport , 1994, A Document Preparation System: Latex Addition -Wesley 2nd edition,. ISBN 20-201-52983-1
- Linda Chan- Sun, 2004, Tutorial Latex http://www. Supinfo-projects.com/fr/2004/latex.
- Md. Saidur Rahman, 2017, Basic Graph Theory.
- meilleur en maths, Chaine eulériennes – Cycles eulériens spy right ® muillen Meilleur en math, maths.com .
- NARSINGH DEO, 1974, Graph Theory with Applications to Engineering B, Computer Science.
- ROBIN J.WILSON, 1972, Introduction to Groph Theory fourth Edition.
- Thomas Nemeth, 2000, Cours document sur l'utilisation de latex, http://www.comment camarche.net/ccmdoc/index.php3 mot-latex.

Résumé :

La théorie des graphes est un domaine de recherche actif depuis 200 ans. Le plus ancien article connu sur la théorie des graphes a été écrit par euler en 1736 pour résoudre le problème des ponts de konisberg.

Dans ce travail, nous avons étudié le solveur automatique en théorie des graphes "chaîne euleurienne" . Après avoir expliqué quelques notions de base sur les graphes et le problème de cette chaîne . Nous avons cité les relations fondamentales à la chaine euleurienne , la commande latex et nous avons ensuite présenté le " premier algorithme de recherche en profondeur " et ces deux derniers que nous avons utilisé dans l'application .

Enfin, nous avons présenté une application à la chaîne euleurienne.

Mots-clés : graphe, chaîne euleurienne , latex , première recherche en profondeur .

الملخص

كانت نظرية الرسم البياني مجالًا نشطًا للبحث لمدة 200عام .أقدم ورقة معروفة عن نظرية الرسم البياني كتبها أويلر عام 1736لحل مشكلة جسر كونيسبرج.

في هذا العمل ، درسنا الحل التلقائي في نظرية الرسم البياني "سلسلة euleurian".تتجنب أوبر شرح المفاهيم الأساسية حول الرسوم البيانية والمشكلات التي تواجهها .هناك العديد من العلاقات الأساسية في سلسلة euleurian، أمر اللاتكس ولدينا ، بما في ذلك "خوارزمية رئيسية للبحث في العمق "وهاتين الأخيرين التي استخدمناها في التطبيق.

أخيرًا ، لا يتعين علينا تقديم طلب إلى سلسلة .euleurian

الكلمات الرئيسية: الرسم البياني ، سلسلة euleurian، اللاتكس ، البحث المتعمق الأول

.

Abstract

Graph theory has been an active field of research for 200 years. The oldest known paper on graph theory was written by Euler in 1736 to solve the Konigsberg bridge problem.

In this work, we studied the automatic solver in graph theory " Eulerian chain".

After explaining some basics about graphs and the problem of this chain. We cited the fundamental relations to the Eulerian chain, the latex command and then we presented the "first depth-search algorithm" and these last two that we used in the application.

Finally, we presented an application to the Eulerian chain.

Keywords: graph, Eulerian, latex, first in-depth search .

Annexes

Annexe 01

Java is a popular general-purpose programming language that was created by James Gosling and his team at Sun Microsystems (which is now owned by Oracle Corporation). It was first released in 1995 and has since become one of the most widely used programming languages in the world.

Annexe 02

LaTeX is a typesetting system and markup language that is widely used for creating professional-looking documents, particularly in the fields of academia, mathematics, and science. It was created by Leslie Lamport in the 1980s as an extension of the TeX typesetting system developed by Donald Knuth.