Paths and Circuits

Table of Contents

[Euler Graphs 2](#_Toc138882774)

[Operations on Graphs 4](#_Toc138882775)

[Union 4](#_Toc138882776)

[Intersection 5](#_Toc138882777)

[Ring Sum 5](#_Toc138882778)

[Decomposition 6](#_Toc138882779)

[Deletion 7](#_Toc138882780)

[Fusion 7](#_Toc138882781)

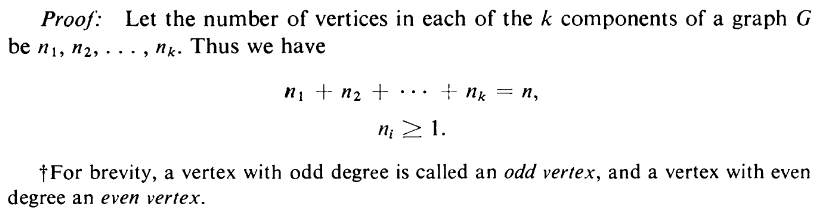
[Arbitrarily Traceable Euler Graphs 8](#_Toc138882782)

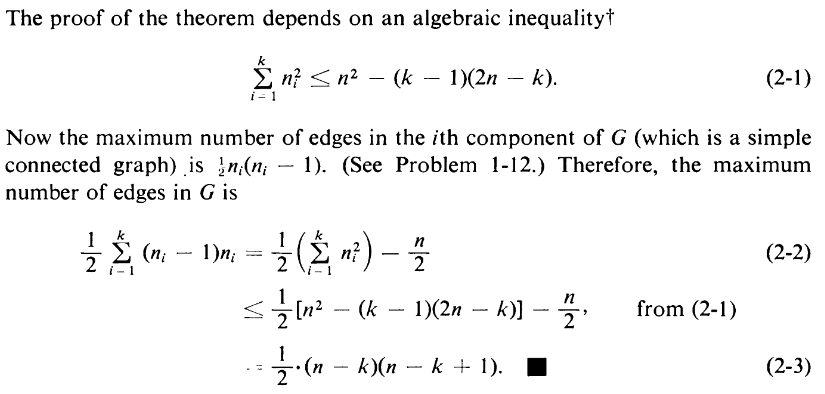
[Hamiltonian Circuit 10](#_Toc138882783)

[Travelling Salesman Problem 11](#_Toc138882784)

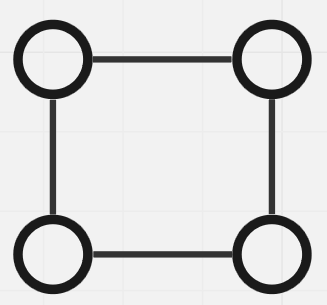
Theorem: A simple graph with vertices and components will have at most edges.

Proof:



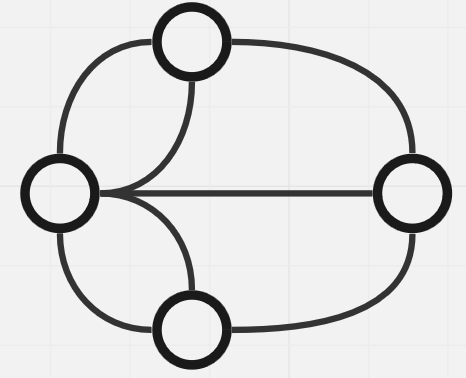


## Euler Graphs



An **Euler Graph**, such as the one shown above, is a graph in which it is possible to perform a **closed walk** visiting all of the vertices and using each edge **exactly once**. The circuit formed using this walk is called an **Euler Line**.

A famous example of a graph that is not an Euler Graph is the Konnigsburg Bridge graph, shown below.



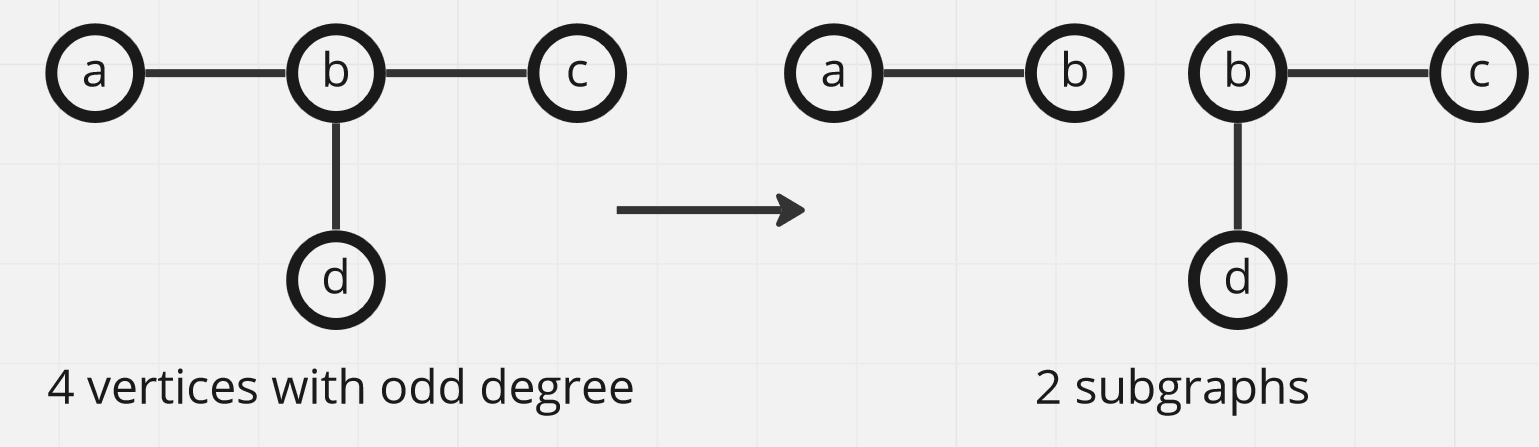
For a graph to be an Euler Graph, it must meet two requirements:

1. The graph must be **connected**.
2. All vertices must have an **even degree**.

If the graph is not connected, we cannot visit every vertex using a single path. If the graph does not have an even degree, at some point we will enter a vertex from which we cannot exit since it takes one edge to enter and one to leave.

An alternative form of an Euler Graph is also acknowledged in many places that uses an **open walk** which forms a **unicursal line**. In this case, the graph must have exactly **2 vertices** with **odd degree**.

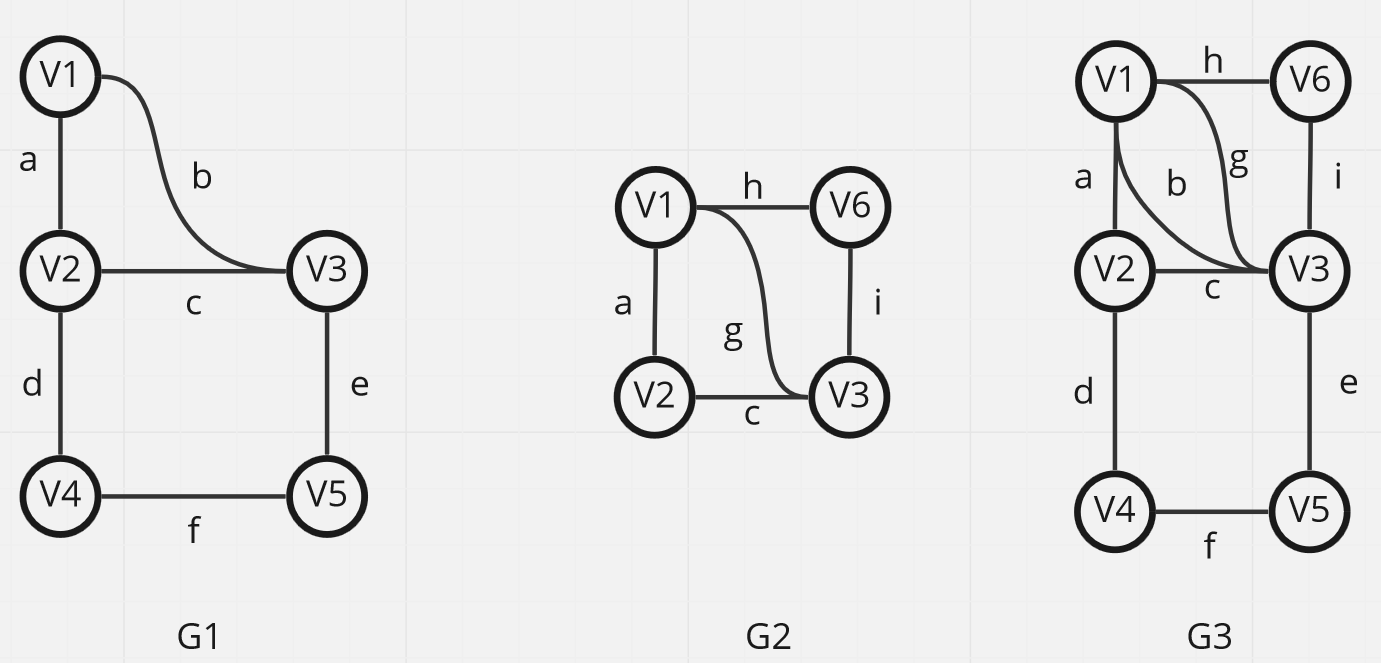
Theorem: A connected graph with odd degree vertices has edge-disjoint subgraphs that are unicursal.



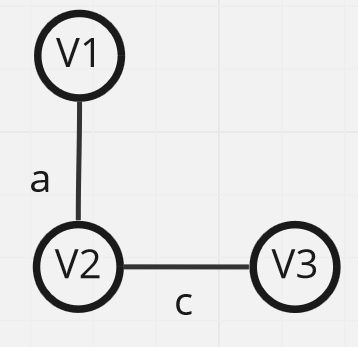
Each of the two subgraphs above are unicursal. By adding just one edge, we can turn them both into Euler graphs.

## Operations on Graphs

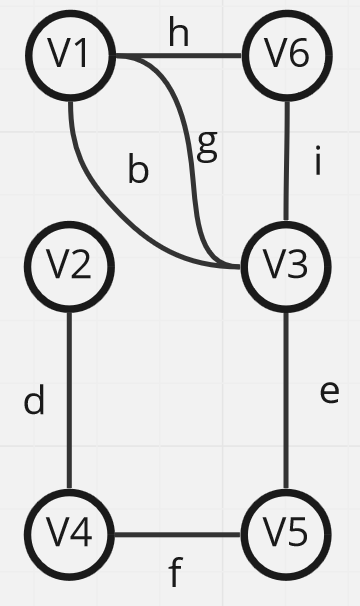
### Union



### Intersection

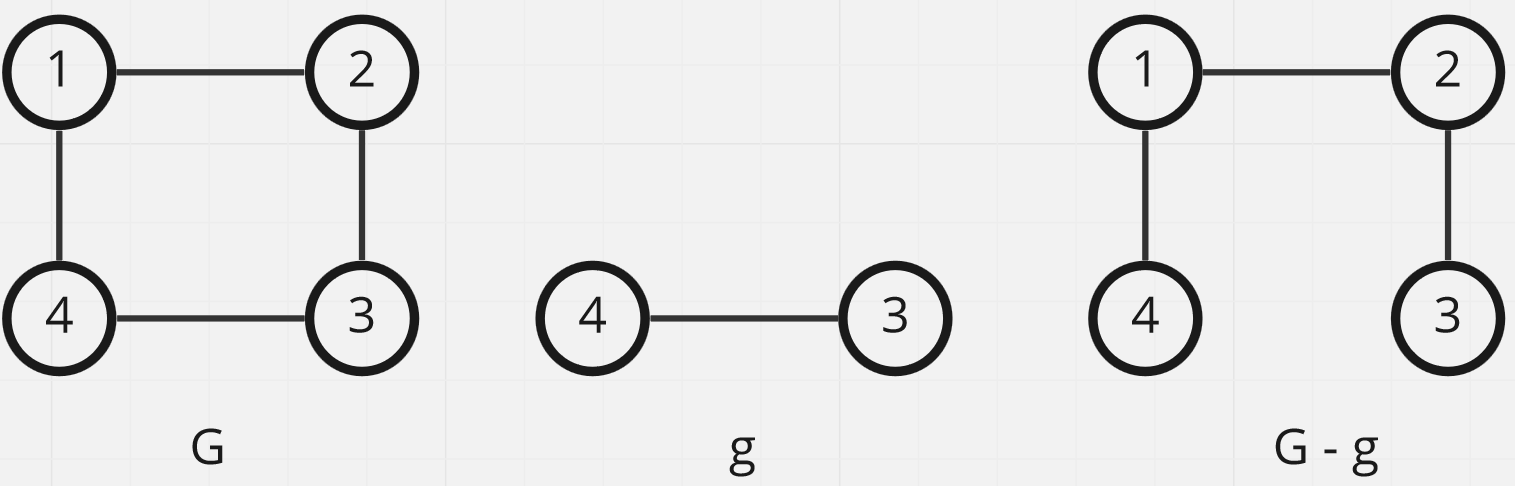


### Ring Sum



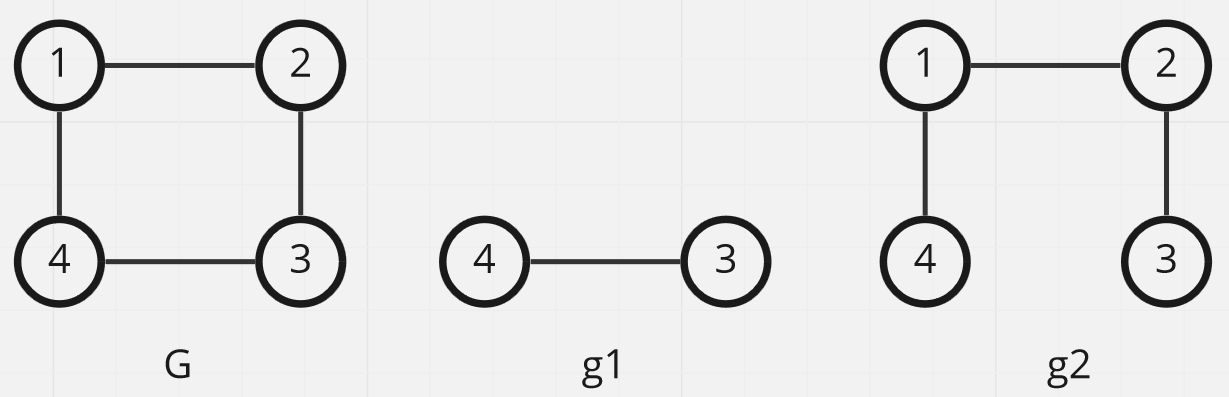
Null Graph

(selected edges removed)



### Decomposition

Decomposing a graph results in two graphs, and , where and Null Graph.



### Deletion

Deleting Edges: The edges are removed from the graph.

Delete Vertices: The vertices and the edges incident on the vertices are both removed.

### Fusion

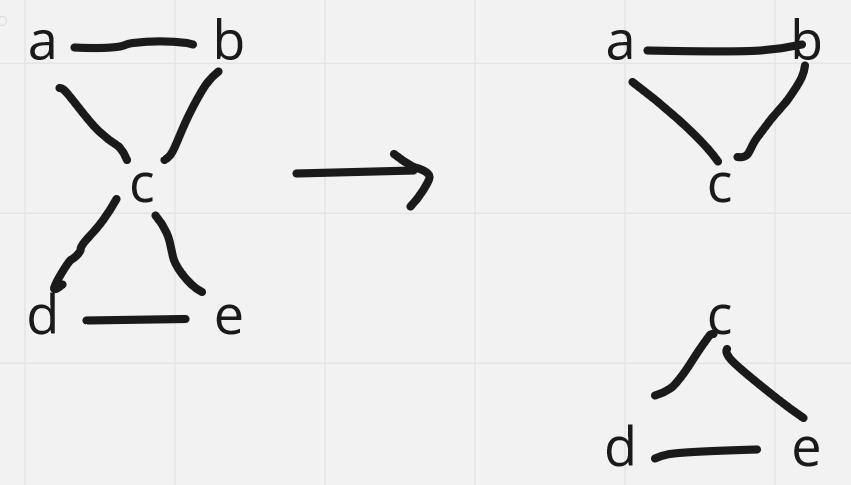
Fusing two vertices results in one less vertex but the same number of edges.



Theorem: A connected graph, , is an **Euler graph** if and only if it can be decomposed into **circuits**.

Proof:

Suppose we have a graph that can be decomposed into **edge-disjoint circuits**. By the definition of a circuit, this means that the **degree** of each vertex in the edge-disjoint circuits is exactly 2. By the definition of being edge-disjoint, it means the circuits have **no common edges**.

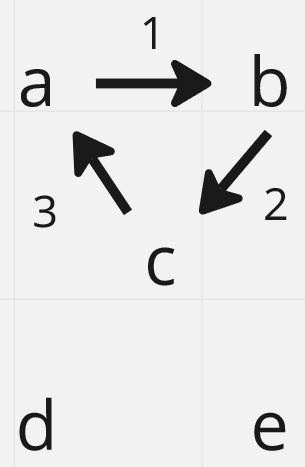


If we take the **union** of all of the circuits, the degree of each of the vertices must remain even since the sum of even degrees can never be odd. The resulting graph is connected and each vertex has an even degree, thus making it an Euler graph.

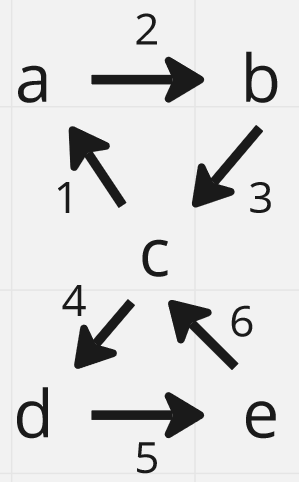
On the other hand, suppose we have an Euler graph, meaning the graph is connected and each vertex has an even degree. If we identify and remove circuits from this graph one by one, the vertex degrees must always decrease by exactly 2. Thus, an Euler graph can be decomposed into edge-disjoint circuits.

## Arbitrarily Traceable Euler Graphs

Suppose we are given an Euler graph and want to form an Euler line. At each vertex, we can pick an unvisited edge to traverse. However, we need to be careful while doing this since we cannot always blindly pick an edge. The situation below shows how this might not work.



However, there are cases where it is possible to blindly pick and still get an Euler line. As shown below, if we start from vertex , we are guaranteed to get an Euler line.



This Euler graph is said to be **arbitrarily traceable** from vertex .

Theorem: An Euler Graph is arbitrarily traceable from a vertex, , if and only if is a part of every edge-disjoint circuit that can be decomposed from the graph.

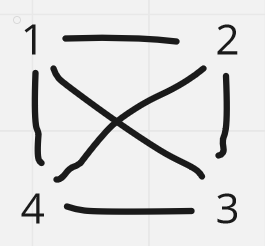
## Hamiltonian Circuit

For a **connected graph**, if it is possible to form a **closed walk** while visiting **each vertex exactly once** (except the starting vertex) and also **without repeating any edges**, the circuit formed is called a **Hamiltonian Circuit**. The path formed by the circuit is called a **Hamiltonian Path**. Note that it is not necessary to visit every edge in this case.

Thus, a Hamiltonian Circuit is a subgraph of the original graph, and a Hamiltonian Path is a subgraph of the Hamiltonian Circuit. Every Hamiltonian Circuit has a Hamiltonian Path, but not every Hamiltonian Path is part of a Hamiltonian Circuit. The latter case can occur if it is not possible to return to the starting vertex, as shown below.



A **complete graph** is guaranteed to contain a Hamiltonian Circuit.



Theorem: If a **complete graph** has vertices, where is odd and , it has **edge-disjoint Hamiltonian Circuits**.

The above theorem provides the solution to the **Seating Problem**.

Dirac’s Theorem: For a simple graph , for all of its vertices, if , there may be a Hamiltonian Circuit present.

The above theorem provides a **sufficient condition** for a Hamiltonian Circuit being present. If the conditions are satisfied, it is possible that an Hamiltonian Circuit is present, but not guaranteed. If the conditions are not satisfied, there is no possibility of a Hamiltonian Circuit being present.

### Travelling Salesman Problem

The **Travelling Salesman Problem** is a famous problem that requires us to find a path by which a salesman can visit every city in a map. Thus, it is essentially asking us to find a Hamiltonian Circuit. However, there can be multiple Hamiltonian Circuits. There is a cost associated with using each edge, and the actual problem is to find the Hamiltonian Circuit with the least overall cost.

If the graph is **fully connected**, then we can visit cities from the first city, cities from the second city and so on. Thus, there are Hamiltonian circuits. This considers both the clockwise and anti-clockwise paths when counting circuits, so the final value is .