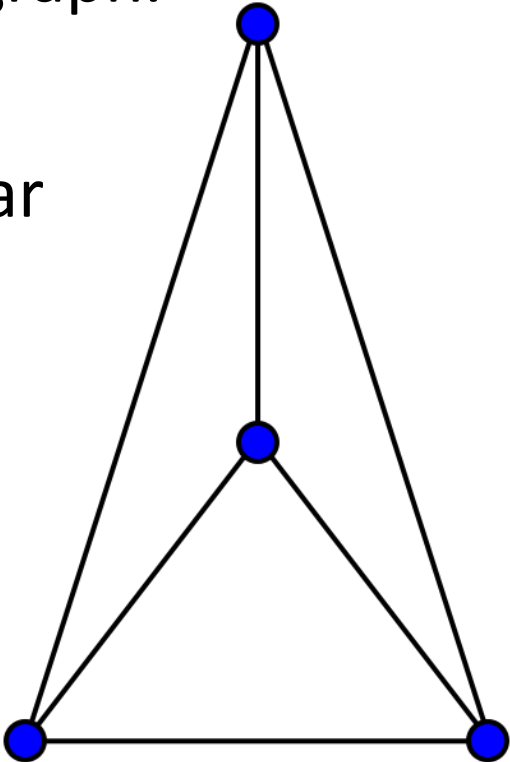


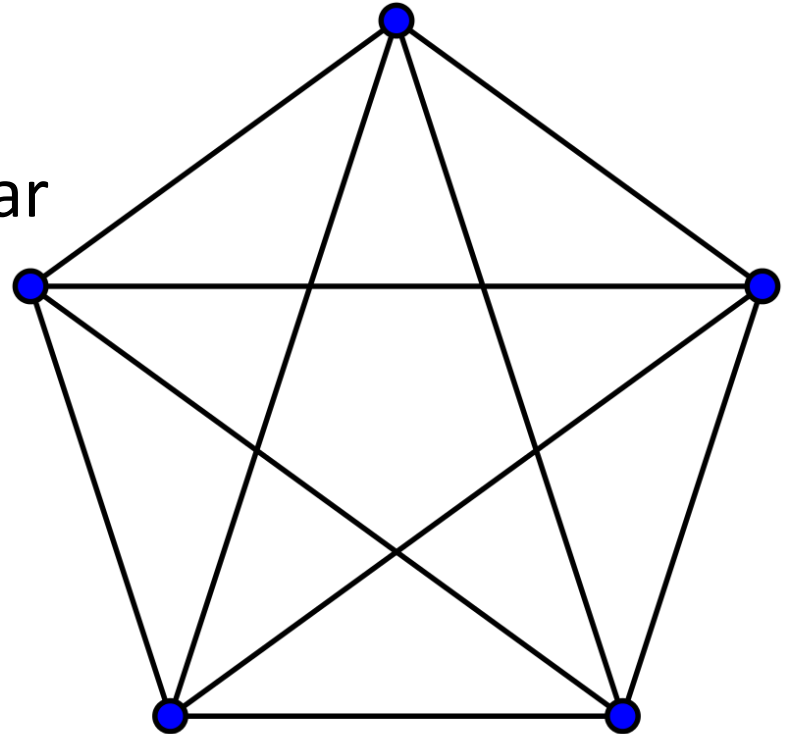
Planar & Euler Graphs

Planar Graphs

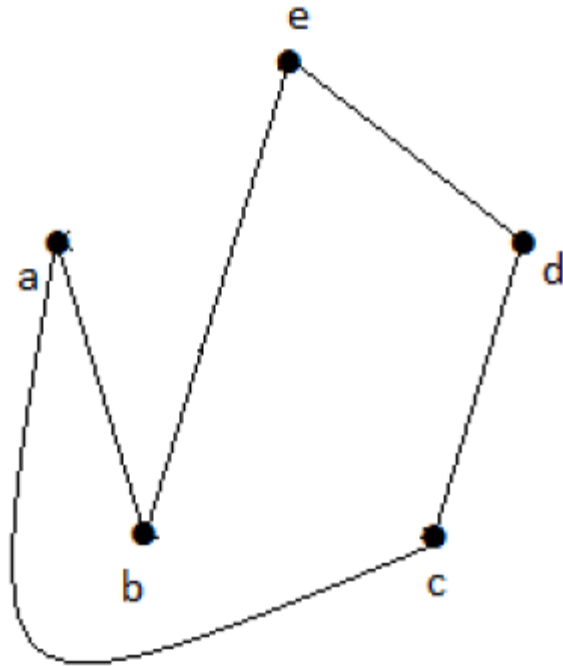
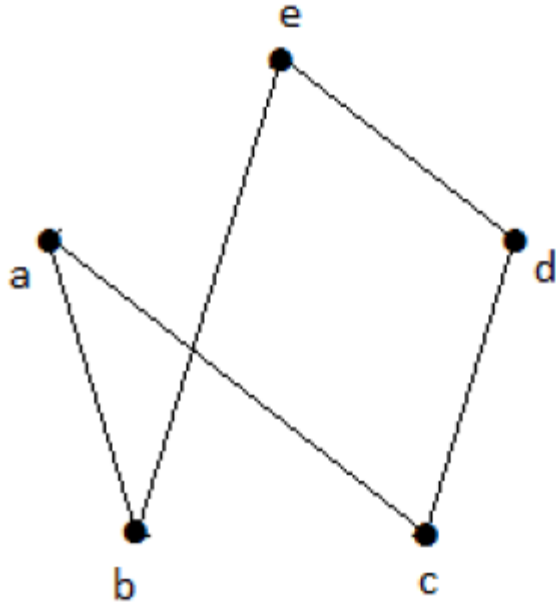
- If a graph G can be drawn on a plane (or a sphere) so that the edges only intersect at vertices, then it is planar.
- Such a drawing of a planar graph is a planar embedding of the graph.



Nonplanar



Contd...



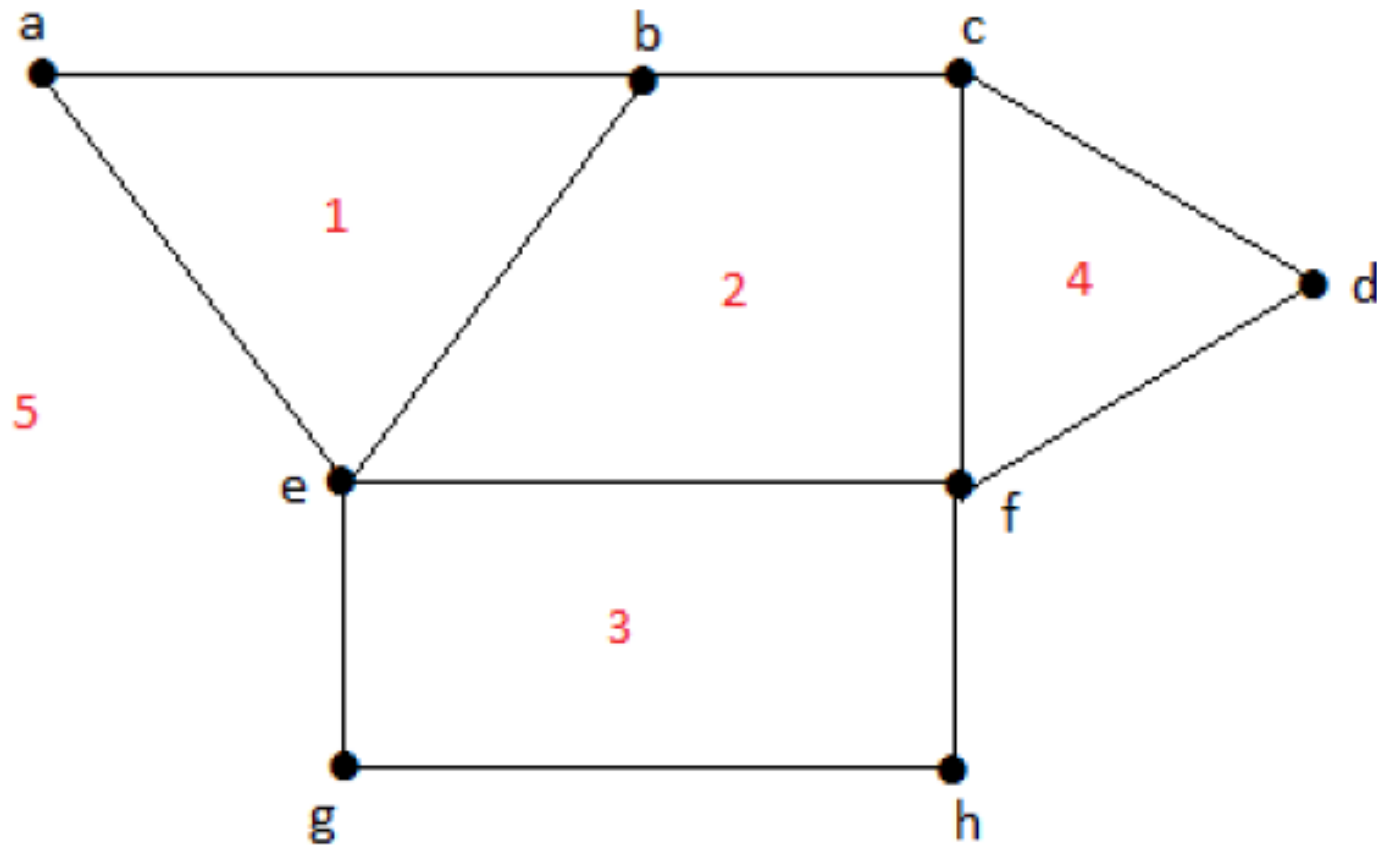
- A non-planar graph converted in to a planar graph.

Euler's formula

- For a finite, connected, planar graph, if
 - v is the number of vertices,
 - e is the number of edges, and
 - f is the number of faces (regions bounded by edges, including the outer, infinitely large region).
 - Then,

$$v - e + f = 2$$

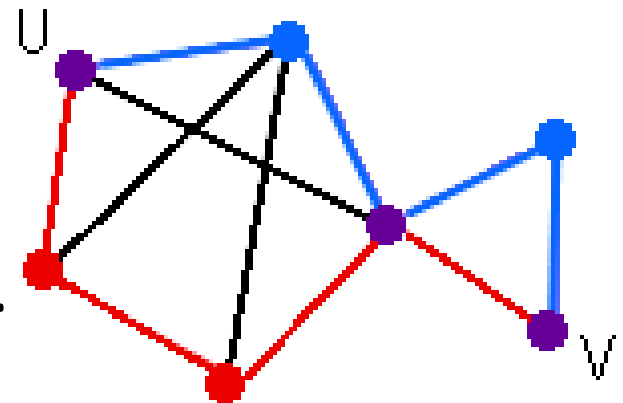
Example



$$8 - 11 + 5 = 2$$

Some Definitions

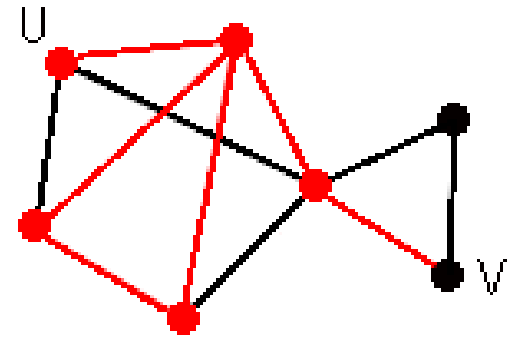
- Walk
 - An alternating sequence of vertices and connecting edges.
 - Can end on the same vertex on which it began or on a different vertex.
 - Can travel over any edge and any vertex any number of times.
- Path
 - A walk that does not include any vertex twice, except that its first and last vertices might be the same.



Contd...

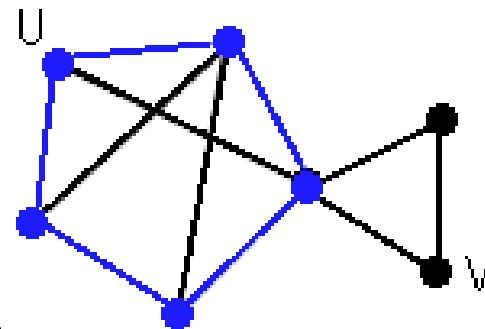
- Trail

- A walk that does not pass over the same edge twice.
- Might visit the same vertex twice, but only if it comes and goes from a different edge each time.



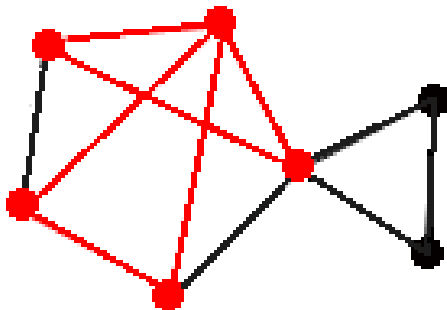
- Cycle

- Path that begins and ends on the same vertex.



- Circuit

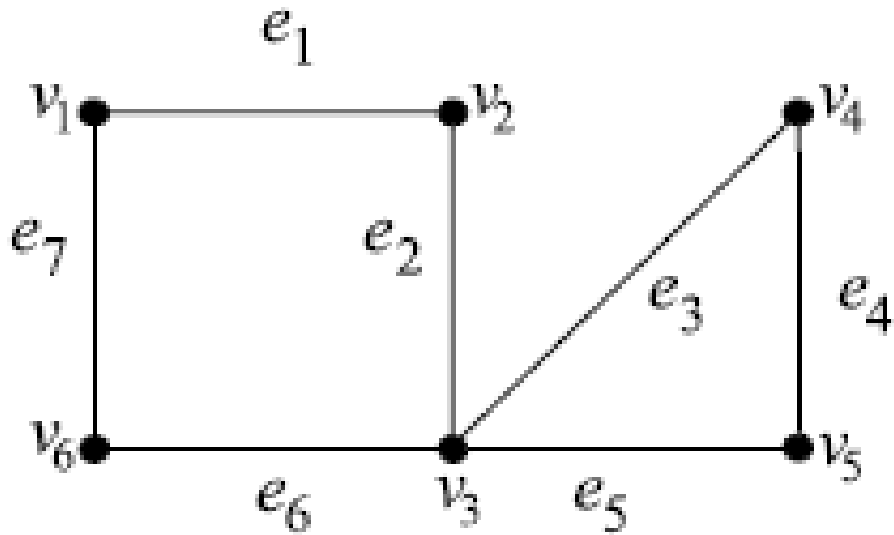
- Trail that begins and ends on the same vertex.



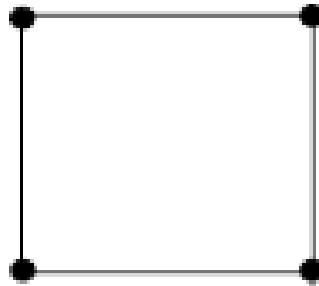
Euler Graphs

- An **Eulerian trail** (or **Eulerian path**) is a trail in a graph which visits every edge exactly once.
- An **Eulerian circuit** (or **Eulerian cycle**) is an Eulerian trail which starts and ends on the same vertex.
- A graph containing an Eulerian circuit is called **Euler graph**.
- A connected graph has an Eulerian path iff it has **at most two vertices of odd degree**.
- A connected graph G is an Euler graph iff **no vertex of G has odd degree**.

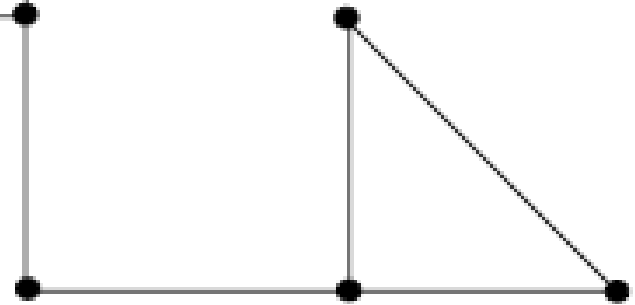
Example



Eulerian graph as
Euler circuit exists.



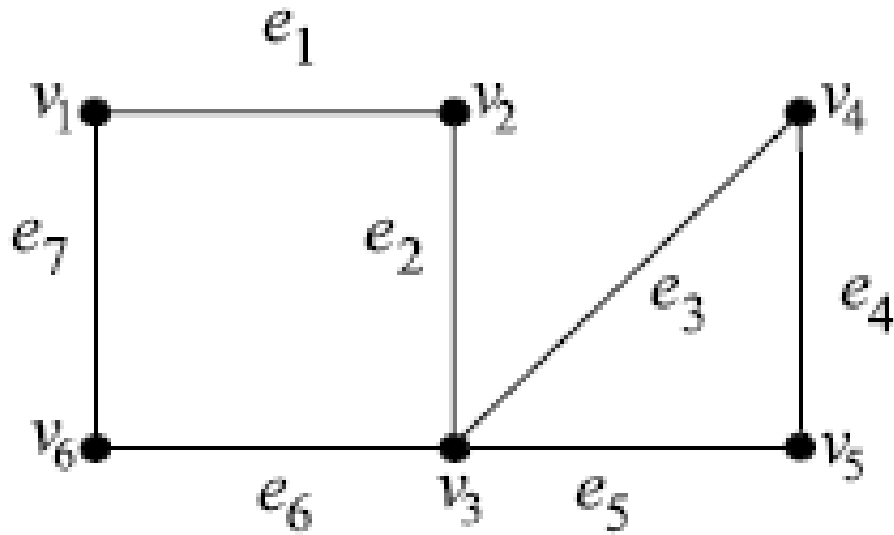
Non-Eulerian
graph as only
Euler path exists.



Fleury's algorithm

1. Start at a vertex of odd degree (Euler path), or, if the graph has none, start with an arbitrarily chosen vertex (Euler circuit).
2. Choose the next edge in the path to be one whose deletion would not disconnect the graph. (Always choose the non-bridge in between a bridge and a non-bridge.)
3. Add that edge to the circuit, and delete it from the graph.
4. Continue until the circuit is complete.

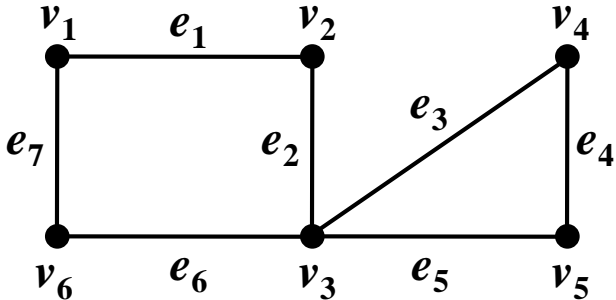
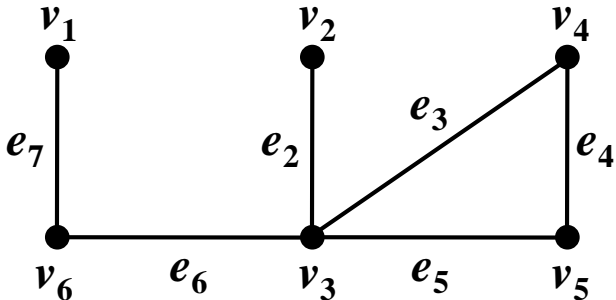
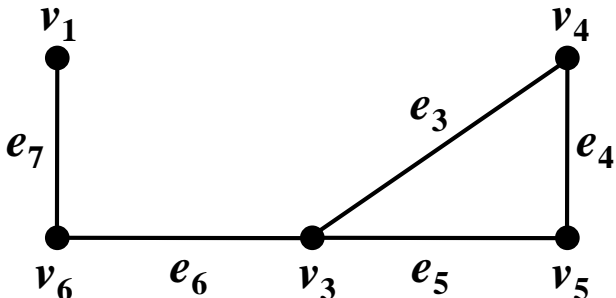
Example – 1



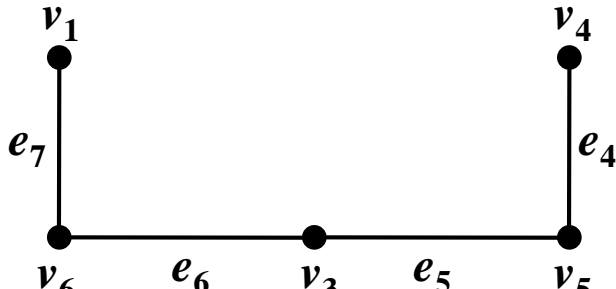
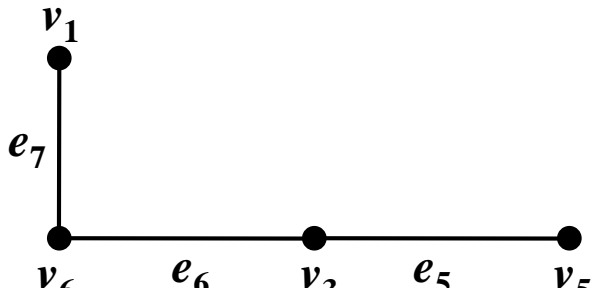
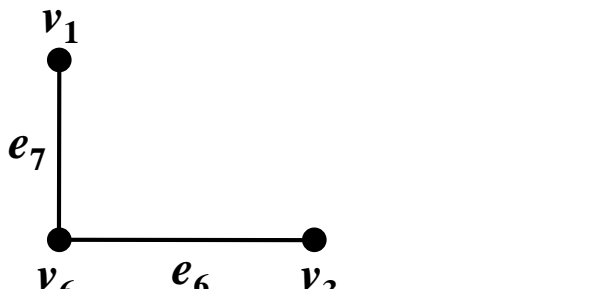
Vertex	Degree
v_1	2
v_2	2
v_3	4
v_4	2
v_5	2
v_6	2

- As degree of each vertex is even, thus Euler circuit exists.

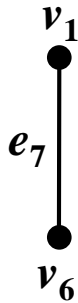
Contd...

Graph	Current Vertex	Trail
	v_1	NULL
	v_2	v_1v_2 or e_1
	v_3	$v_1v_2v_3$ or e_1e_2

Contd...

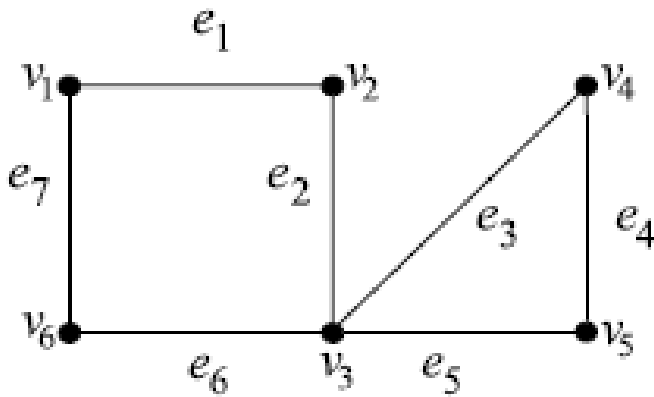
Graph	Current Vertex	Trail
	v_4 v_3v_6 or e_6 is a bridge and can't be selected.	$v_1v_2v_3v_4$ or $e_1e_2e_3$
	v_5	$v_1v_2v_3v_4v_5$ or $e_1e_2e_3e_4$
	v_3	$v_1v_2v_3v_4v_5v_3$ or $e_1e_2e_3e_4e_5$

Contd...

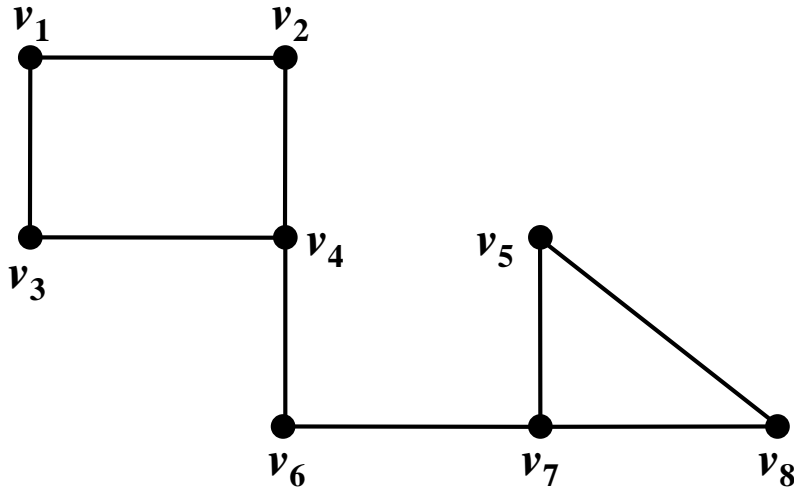
Graph	Current Vertex	Trail
	v_6	$v_1v_2v_3v_4v_5v_3v_6$ or $e_1e_2e_3e_4e_5e_6$
NULL	v_1	$v_1v_2v_3v_4v_5v_3v_6v_1$ or $e_1e_2e_3e_4e_5e_6e_7$

• Required Trail:

$v_1v_2v_3v_4v_5v_3v_6v_1$
or $e_1e_2e_3e_4e_5e_6e_7$



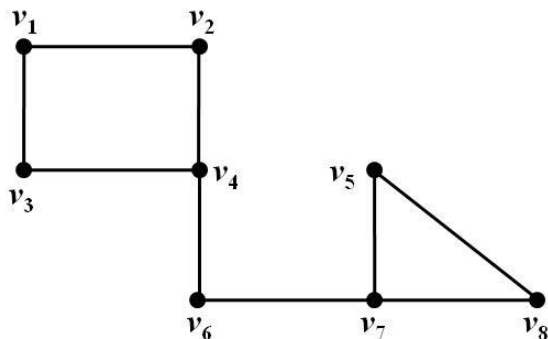
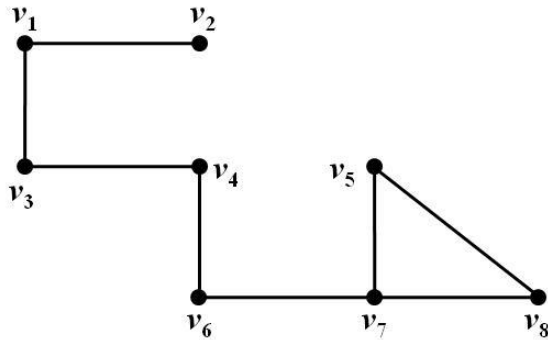
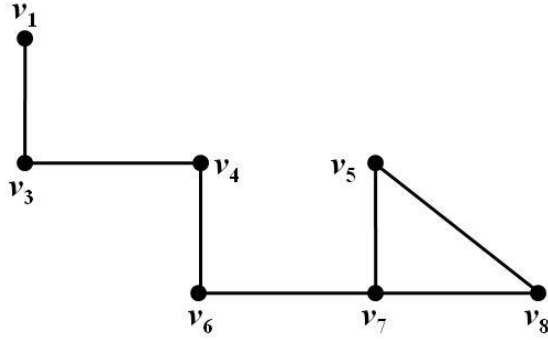
Example – 2



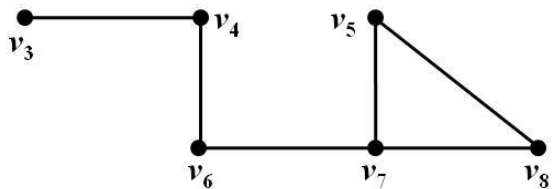
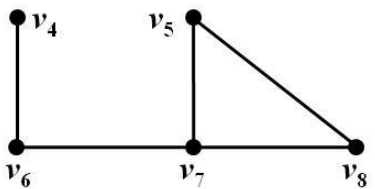
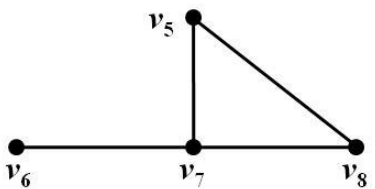
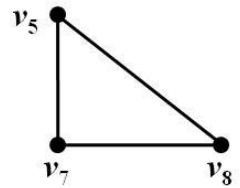
Vertex	Degree
v_1	2
v_2	2
v_3	4
v_4	3
v_5	2
v_6	2
v_7	3
v_8	2

- As degree of all the vertices is even except two (v_4 or v_7), thus Euler path exists.
- Start either from v_4 or v_7 .

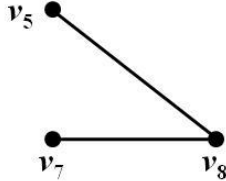
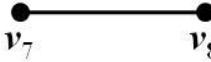
Contd...

Graph	Current Vertex	Trail
	v_4	NULL
	v_2 v_4v_6 is a bridge and can't be selected.	v_4v_2
	v_1	$v_4v_2v_1$

Contd...

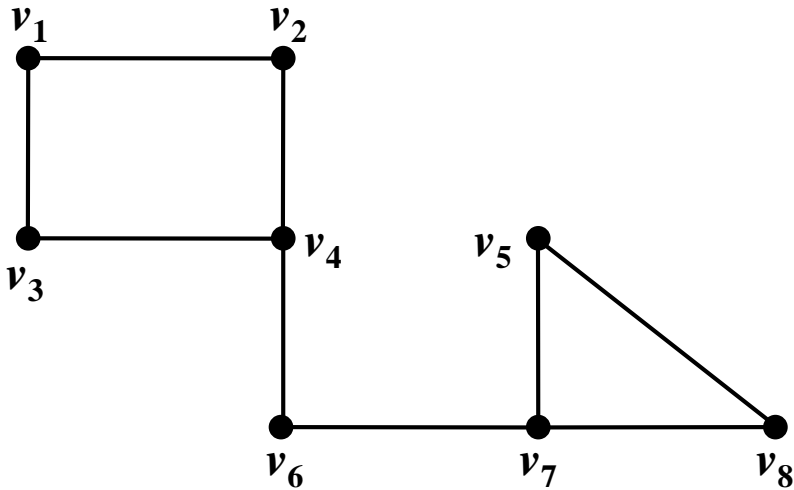
Graph	Current Vertex	Trail
	v_3	$v_4v_2v_1v_3$
	v_4	$v_4v_2v_1v_3v_4$
	v_6	$v_4v_2v_1v_3v_4v_6$
	v_7	$v_4v_2v_1v_3v_4v_6v_7$

Contd...

Graph	Current Vertex	Trail
	v_5	$v_4v_2v_1v_3v_4v_6v_7v_5$
	v_8	$v_4v_2v_1v_3v_4v_6v_7v_5v_8$
NULL	v_7	$v_4v_2v_1v_3v_4v_6v_7v_5v_8v_7$

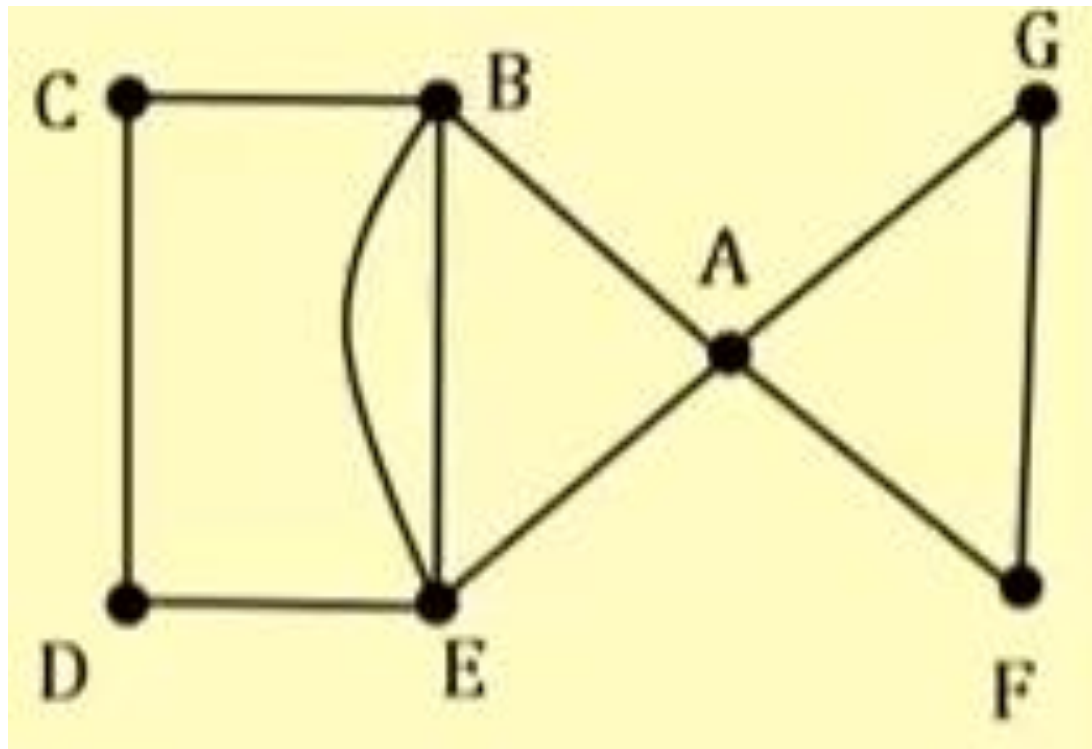
- Required Trail:

$v_4v_2v_1v_3v_4v_6v_7v_5v_8v_7$



Example – 3

- Find Euler circuit using Fleury's algorithm. Start at vertex A.



<https://www.youtube.com/watch?v=vvP4Fg4r-Ns>

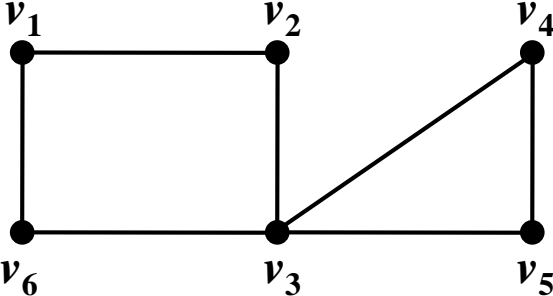
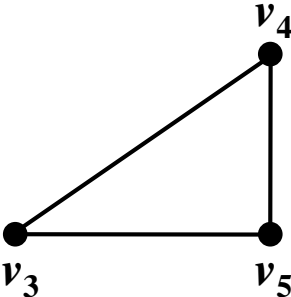
Contd...

- In Fleury's algorithm, the graph traversal is linear in the number of edges, i.e. $O(|E|)$, excluding the complexity of detecting bridges.
- With Tarjan's linear time bridge-finding algorithm, the time complexity is $O(|E|^2)$.
- With Thorup's dynamic bridge-finding algorithm, the time complexity gets improved to $O(|E|(\log |E|)^3 \log \log |E|)$.

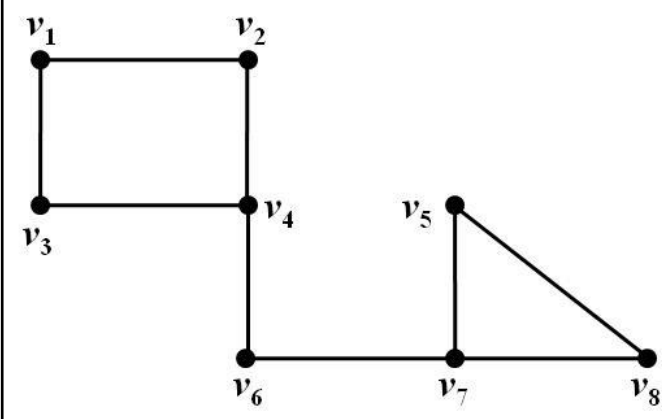
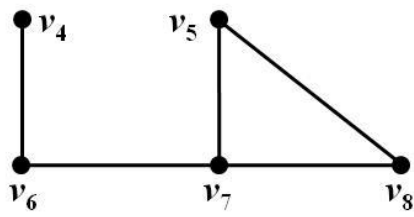
Hierholzer's algorithm

1. Choose any starting vertex v , and follow a trail of edges from that vertex until returning to v . The tour formed in this way is a closed tour, but may not cover all the vertices and edges of the initial graph.
2. As long as there exists a vertex u that belongs to the current tour but that has adjacent edges not part of the tour, start another trail from u , following unused edges until returning to u , and join the tour formed in this way to the previous tour.

Example – 1

Graph	Trail
	$v_1v_2\underline{v_3}v_6v_1$
	$v_1v_2v_3v_4v_5v_3v_6v_1$
Null	Required Trail: $v_1v_2v_3v_4v_5v_3v_6v_1$

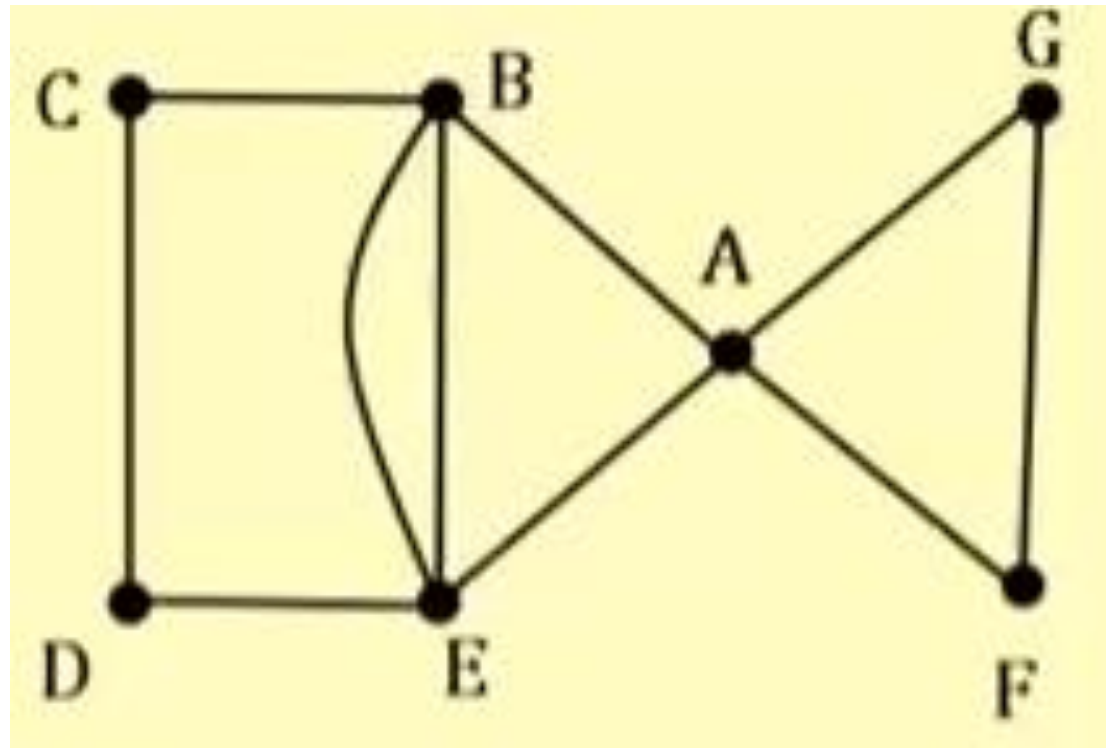
Example – 2

Graph	Trail
	$v_1 v_2 \underline{v_4} v_3$
	<p>Circuit starting and ending at vertex v_4 cannot be formed with the remaining edges.</p> <p>Algorithm terminates reporting graph is not Euler.</p>

Example – 3

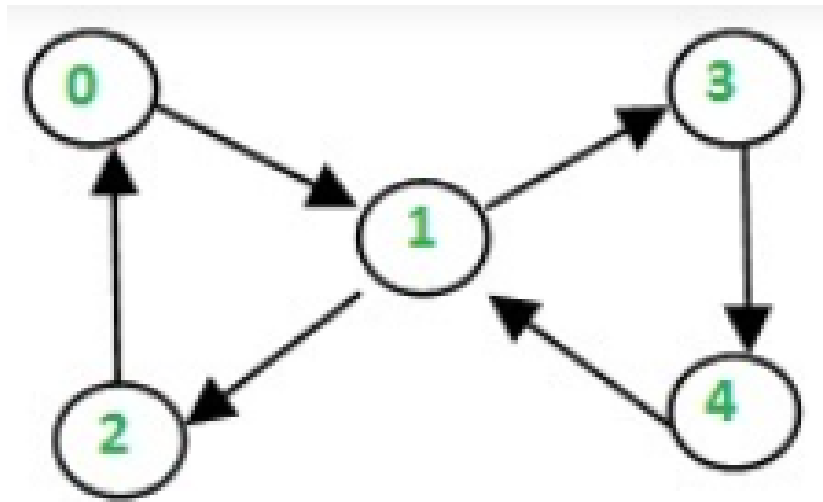
- Find Euler circuit using Hierholzer's algorithm. Start at vertex A.

- Solution:
 - ABCDEA
 - AGFABCDEA
 - AGFABEBCDEA



Example - 4

- Link: <https://www.geeksforgeeks.org/hierholzers-algorithm-directed-graph/>

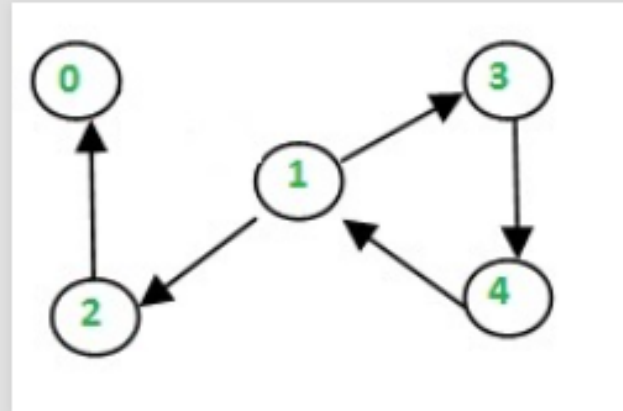


Contd..

Let's start our path from 0.

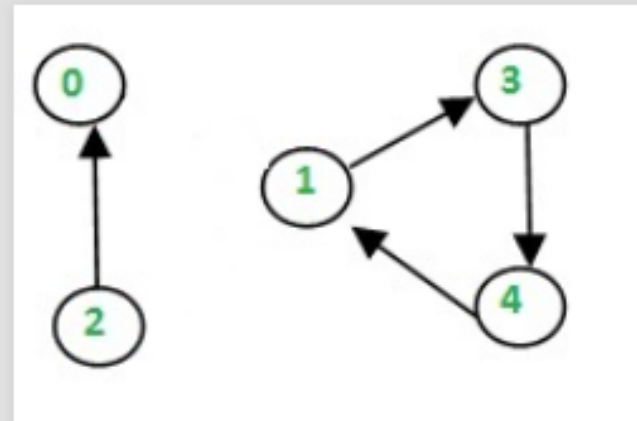
Thus, $\text{curr_path} = \{0\}$ and $\text{circuit} = \{\}$

Now let's use the edge $0 \rightarrow 1$



Now, $\text{curr_path} = \{0,1\}$ and $\text{circuit} = \{\}$

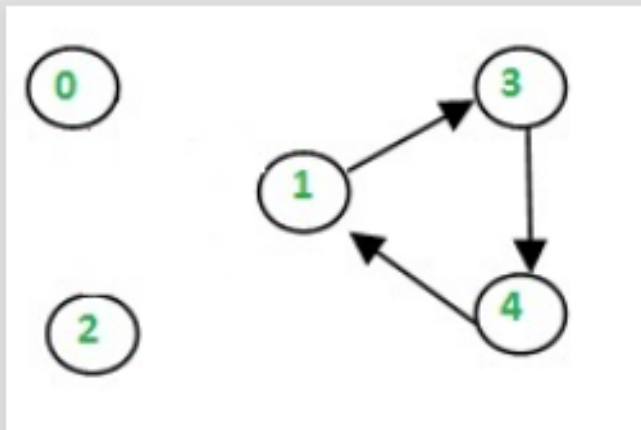
similarly we reach up to 2 and then to 0 again as



Contd..

Now, $\text{curr_path} = \{0,1,2\}$ and $\text{circuit} = \{\}$

Then we go to 0, now since 0 haven't got any unused edge we put 0 in circuit and back track till we find an edge



We then have $\text{curr_path} = \{0,1,2\}$ and $\text{circuit} = \{0\}$

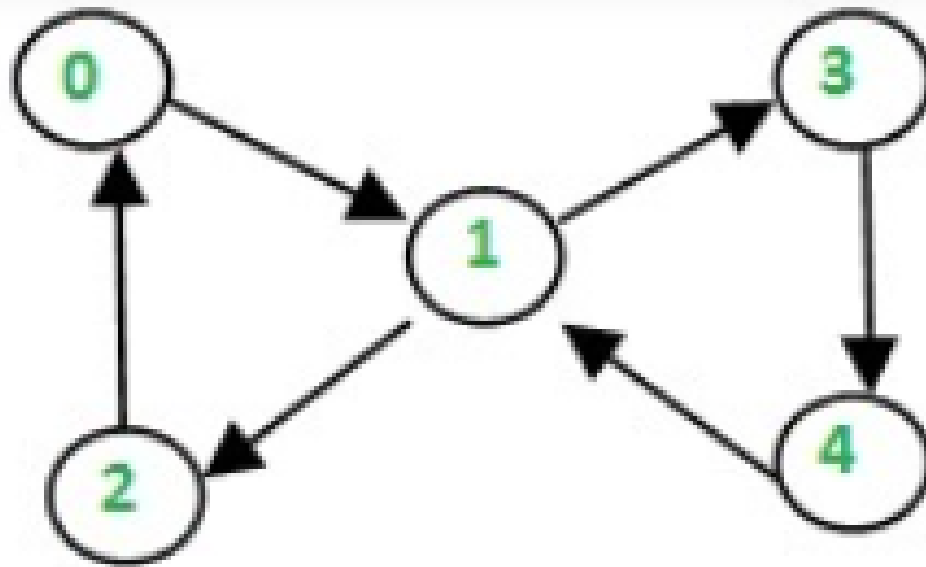
Similarly, when we backtrack to 2, we don't find any unused edge. Hence put 2 in the circuit and backtrack again.

$\text{curr_path} = \{0,1\}$ and $\text{circuit} = \{0,2\}$

Contd..

After reaching 1 we go to through unused edge 1->3 and then 3->4, 4->1 until all edges have been traversed.

The contents of the two containers look as:
`curr_path = {0,1,3,4,1}` and `circuit = {0,2}`

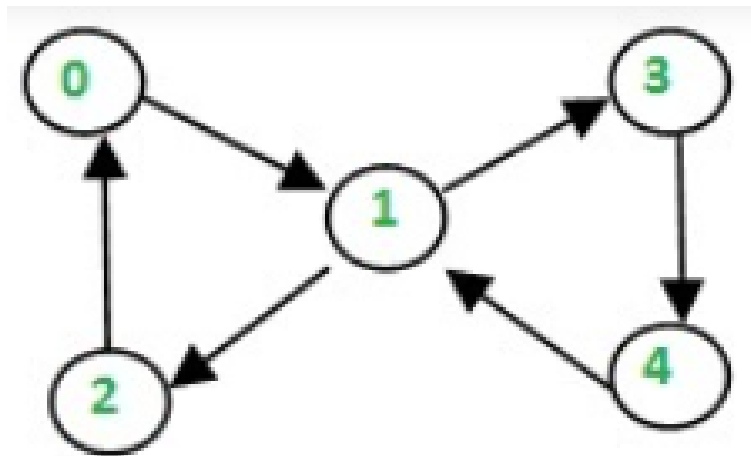


Contd..

now as all edges have been used, the curr_path is popped one by one into the circuit.

Finally, we've circuit = {0,2,1,4,3,1,0}

We print the circuit in reverse to obtain the path followed.



0 -> 1 -> 3 -> 4 -> 1 -> 2 -> 0

Contd...

- If doubly linked list is used to maintain
 - The set of unused edges incident to each vertex,
 - The list of vertices on the current tour that have unused edges, and
 - The tour itself.
- The individual operations of finding
 - Unused edges exiting each vertex,
 - A new starting vertex for a tour, and
 - Connecting two tours that share a vertex.
- May be performed in constant time, so the algorithm takes linear time, **$O(|E|)$** .

Thank you