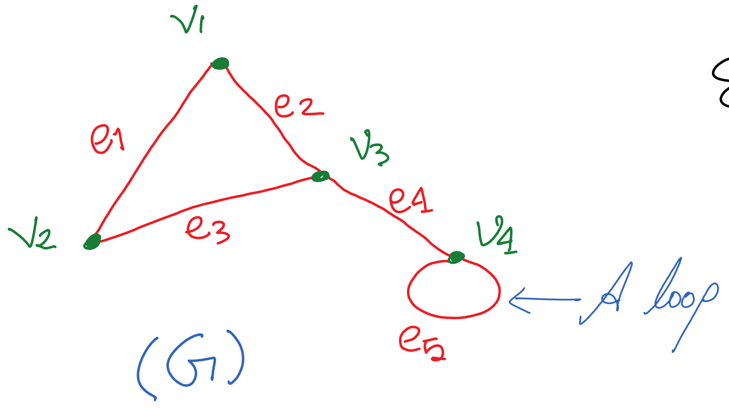
What is A Graph?

A graph G consists of two finite sets: a non-empty set V(G) of vertices and a set of E(G) of edges, where each edge is associated with a set consisting of either one or two vertices called its endpoints.



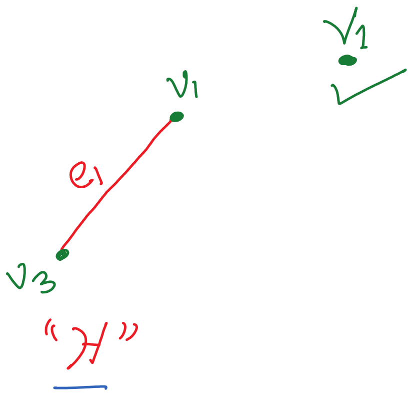
$$V(G) = \{v_1, v_2, v_3, v_4\}$$

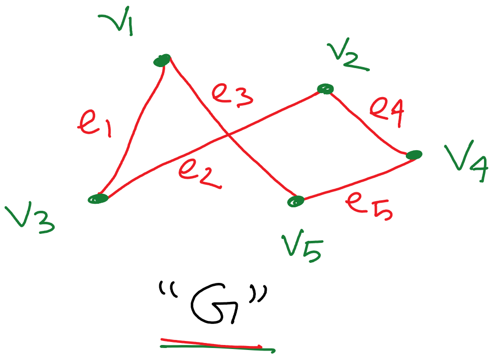
$$E(G) = \{e_1, e_2, e_3, e_4, e_5\}$$

Edges	End-points
e_1	$\{v_1, v_2\}$
e_4	$\{v_3, v_4\}$
e_5	$\{v_4\}$

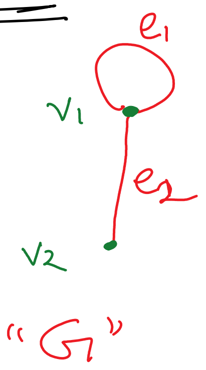
Subgraph A graph H is said to be a subgraph of a graph G if and only if every vertex in H is also a vertex in G, and every edge in H is also an edge of G and every edge in H has the same endpoints

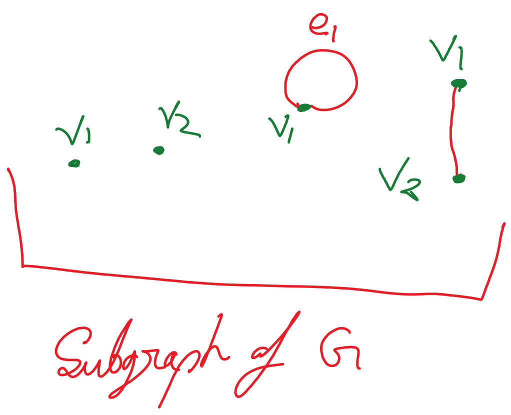
G and every edge in H has the same endpoints as it has in G.



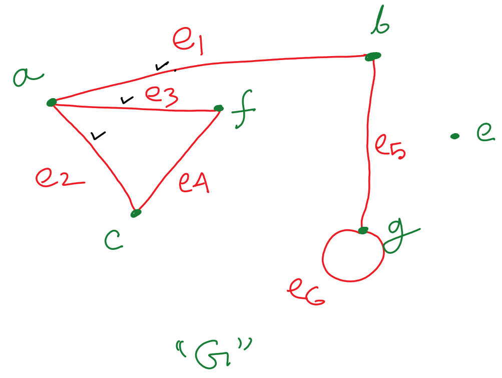


Example





Degree



① e_1, e_2, e_3 are connected to a

② The degree of a = 3

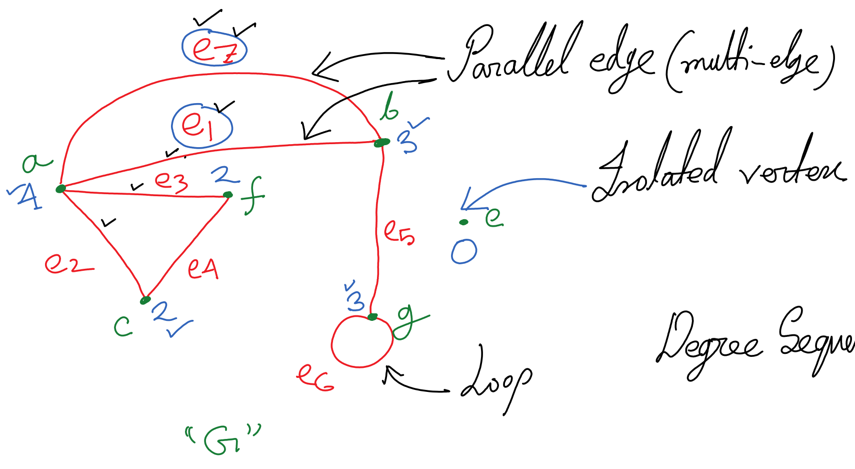
③ e_2 & e_4 are connected to C

④ The degree of c = 2

⑤ Degree of e = 0

⑥ Degree of G = 3 [⊛ A loop has 2 degree]

A degree of a vertex is the number of edges with that vertex as an endpoint

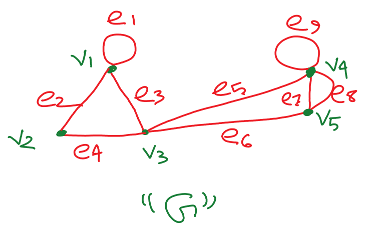


Degree Sequence of G = (0, 2, 2, 3, 3, 4)

Example

① Find all the loops → e_1, e_9

② Find all of the parallel edges → e_8, e_7



③ Find the degree of $V4 \rightarrow 5$

④ Find the overall (total) edges of $G \rightarrow 9$

⑤ Find the overall (total) degree of $G \rightarrow d(v_1) + d(v_2) + d(v_3) + d(V4) + d(v_5)$

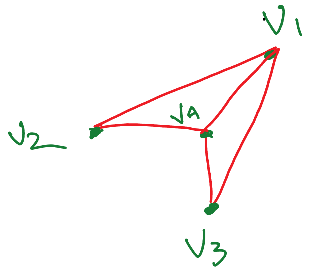
$= 4+2+4+5+3$

$= 18$

Sum of Degrees of Vertices Theorem

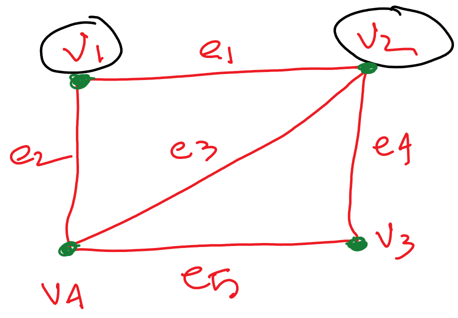
Number of vertices $= |V|$

Number of edges $= |E|$



Number of vertices = number of order $= 4$

Number of edges = number of size of the graph $= 5$

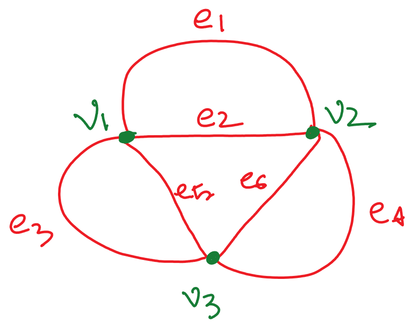


$|V| = 4$ ✓

$|E| = 5$ ✓

Total degree $= 2+3+2+3$

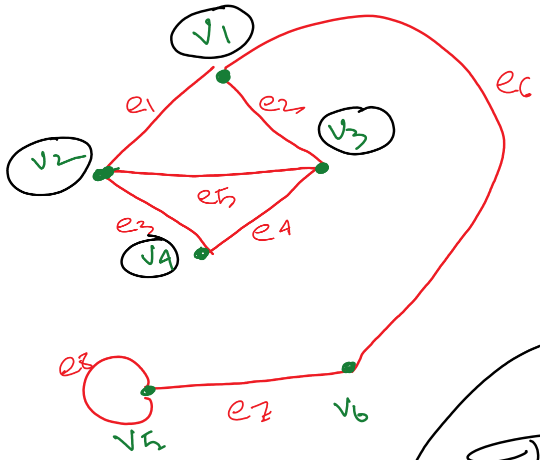
$= 10$



$|V| = 3$ ✓

$|E| = 6$ ✓

Total degree $= 4+4+4 = 12$



$|V| = \overset{\checkmark}{6}$

$|E| = \overset{\checkmark}{\underline{8}}$

$\text{Total degree} = 3+3+3+2+2+3$
$= \underline{16}$

$\checkmark\ \text{Total degree} = 2 \times |E|$

$\Rightarrow \checkmark\ |E| = \dfrac{\text{Total degree}}{2}$

Total number of degree of $G = 2|E|$

Let $\underline{G}$ be graph, that has n vertices $[n = 1, 2, 3, \ldots\ldots]$

$v_1, v_2, v_3, \ldots\ldots\ldots, v_n$

Let us consider that G has m edges $(e_1, e_2, e_3, \ldots\ e_m)$

If e_i is an edge, then vertices v_i and v_j, will have two degree. That means every edge adds two degrees to the graph.

✓ Corollary 1

⊛ Total degree of a graph is always <u>even</u>

It is because, <u>total degree, $G = 2|E|$</u>

Corollary 2

⊛ In any graph, there is an even number of vertices of odd degree

(E1) Construct a graph with four vertices of degrees 1,1,2,3

$|V| = 4$

Total degree $= 1+1+2+3 = \underline{7}$ (odd)

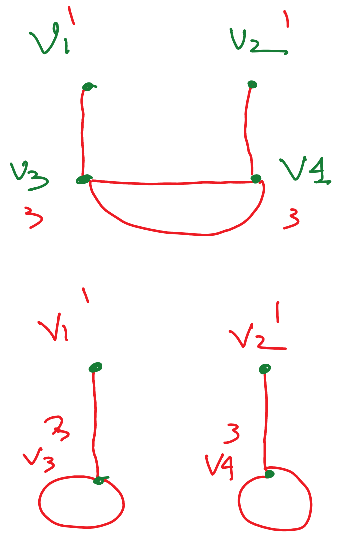
✓ Graph cannot be constructed

E2 Construct a graph with 4 vertices of degrees 1, 1, 3, 3

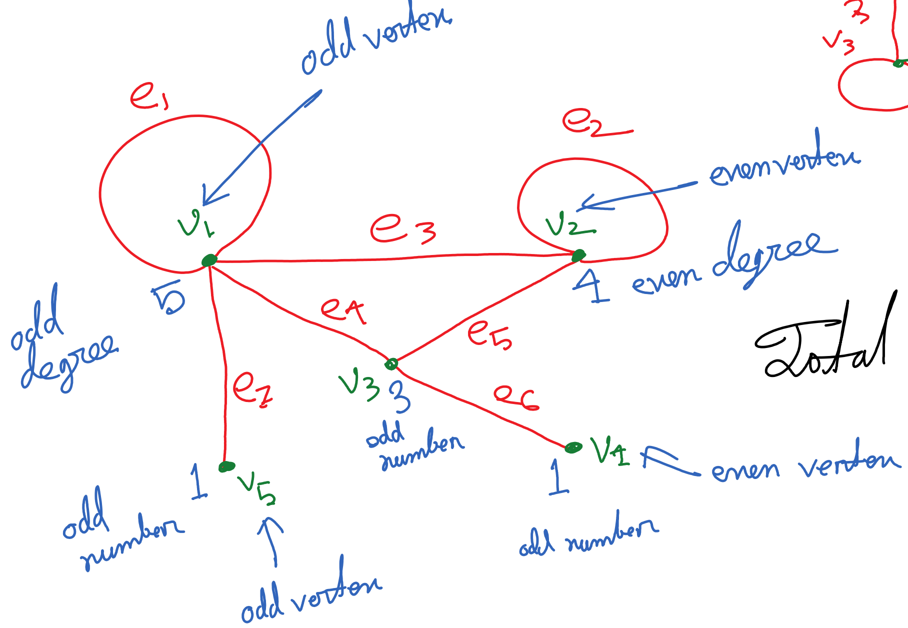
$|V| = 4$

Total degree $= 1+1+3+3 = 8$ (even)

$\therefore$ we can construct the graph.



E3



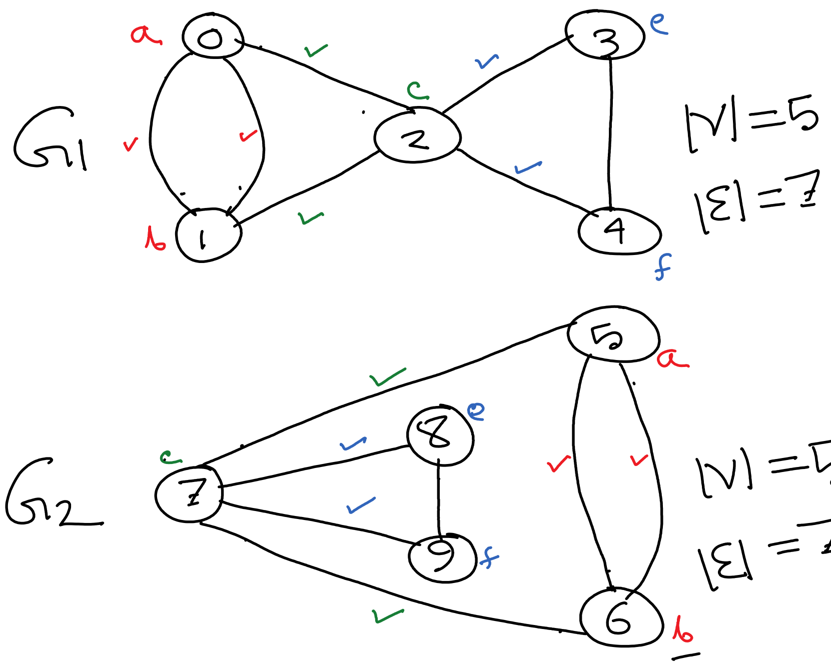
Total degree $= 14$

Minimum degree $\delta(G) = 1$ Maximum degree $\Delta(G) = 5$

$|E| = 7$ total degree $= 2 \times |E| = 14$

Isomorphism $(A \cong B)$

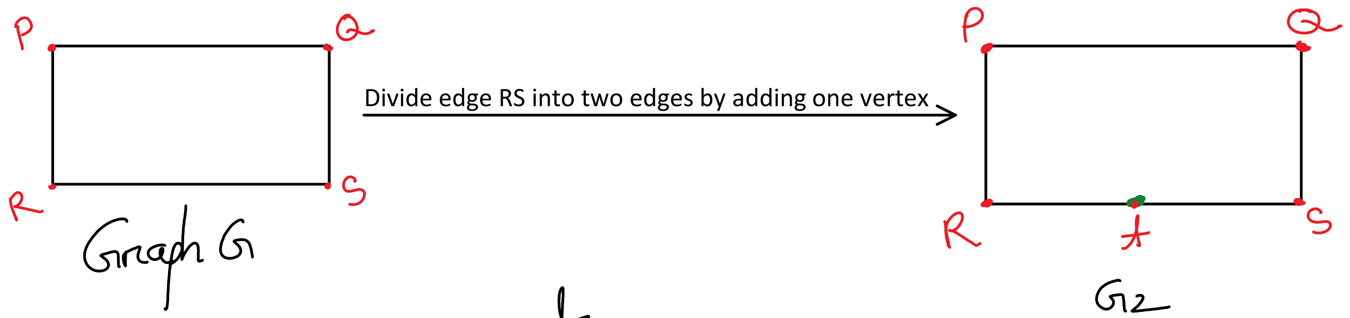
Two graphs G1 and G2 are isomorphic if there exists a matching between their vertices so that two vertices are connected by an edge in G1 if and only if corresponding vertices are connected by an edge in G2.

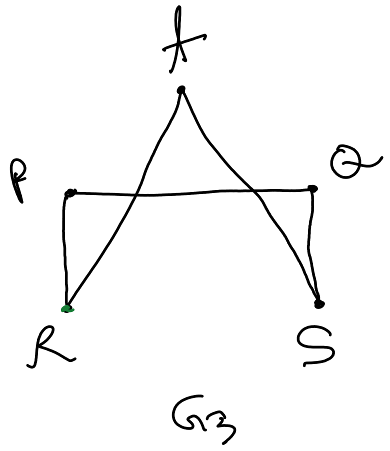


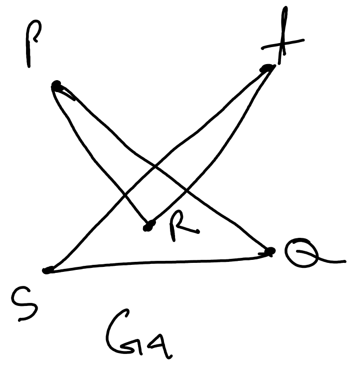
$\therefore G_1 \cong G_2$

Homomorphism

Two graphs G1 and G2 are said to be homomorphic, if each of these graphs can be obtained from the same graph G by dividing some edges of G with more vertices.





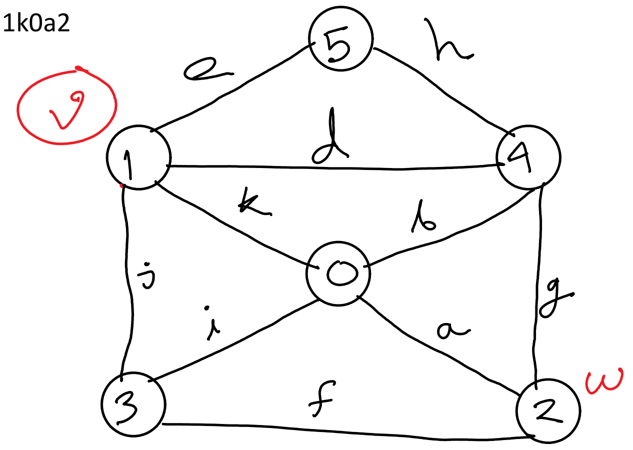


Here, graph G2, G3 and G4 are homomorphic to graph G

Walks, Trails, Paths and Circuits

A Walk: A walk from v to w is a finite alternating sequence of adjacent vertices of edges of G

A walk from V to W is 1e5h4g2, or it can be 1j3f2, or it can be 1k0a2
Length of Walk: Number of edges in the walk.
For example the length of 1e5h4g2 is 3
It is written as, Length of W = 3

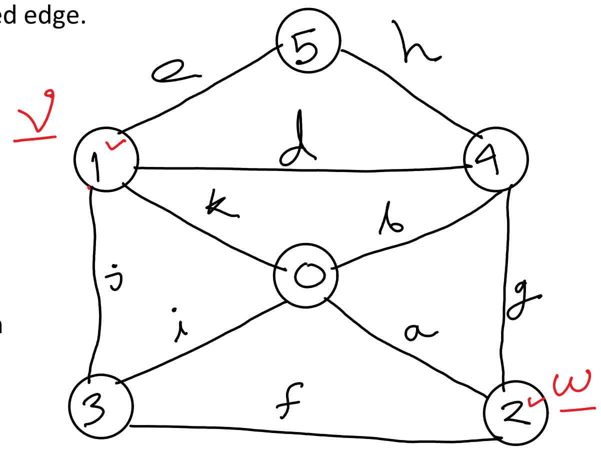


Open Walk: When the walk starts from one vertex and ends in a separate vertex, it is called an open walk. For example, the walk from V to W is an open walk.

Trivial Walk: If the length of walk is 0, it is called a trivial walk
For example the length of walk from V to V is 0.

Closed Walk: When the walk starts from a vertex and ends in the same vertex, it is called closed walk. For example - a walk 1e5h4d1 is a closed walk.

Trail: A trail from V to W is a walk from V to W that does not contain a repeated edge.
For example -
The walk from 1 to 0 = 1e5h4g2a0 is a trail.
However, if we walk back from 0 as 1e5h4g2a0a2 is not a trail.



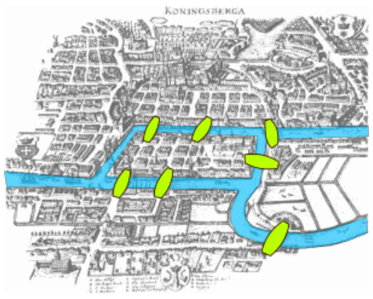
Path: A path from V to W is a trail that does not contain a repeated vertex.
For example -
The walk 1e5h4g2f3 is a path. Because there is no repeated vertex.

Distance: Distance from 1 to 2, denoted as d(1,2) is the smallest length of path between the two vertices.
For example -
the distance between d(1,2) = 2
Here the path is 1k0a2

$d(1,2) = 2$

Circuit: A circuit is a trail that contains at least one edge and starts and ends at the same vertex. If there is no repeating vertex, it is called a simple circuit.
For example -
The walk 1e5h4d1 is a circuit because it contains more than one edge and ends at the same vertex.

Euler Trails and Circuit



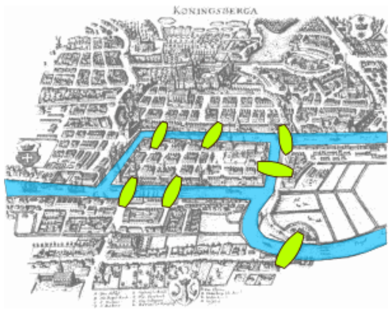
Euler is a mathematician and his research initiate the Graph Theory. He published a paper in 1736 which is the first paper in graph theory. The topic of this paper is also known as Königsberg bridge problem.

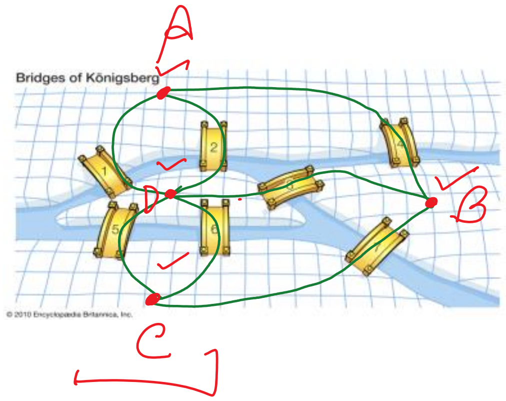
The Problem Statement:

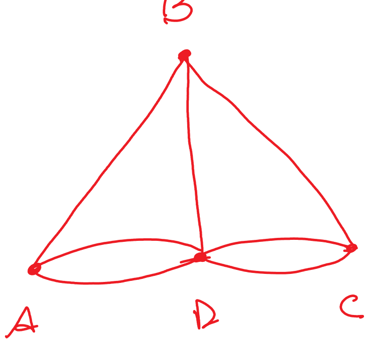
1. Start anywhere in the landmass.
2. Go through every bridge for once and then come back to the same place.
3. Is it possible?
4. It is not possible to do it.

Experimental Approach:

1. Start with the fundamental elements of a graph - (i) Vertices and (ii) Edges
2. Assign vertices to each landmass
3. Connect the landmasses so that they pass through the bridge
4. Can we start at a vertex and go to each vertex for once and come back to the same vertex?
5. No it is not possible.

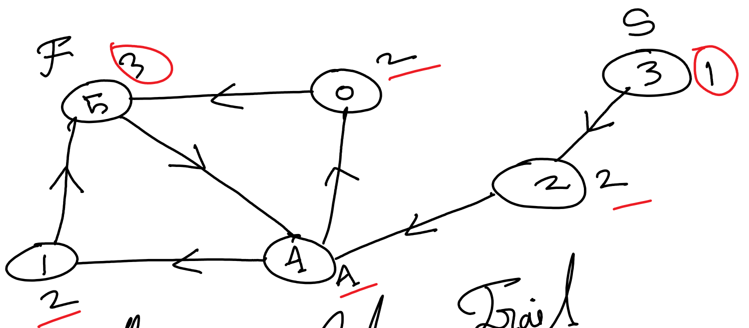




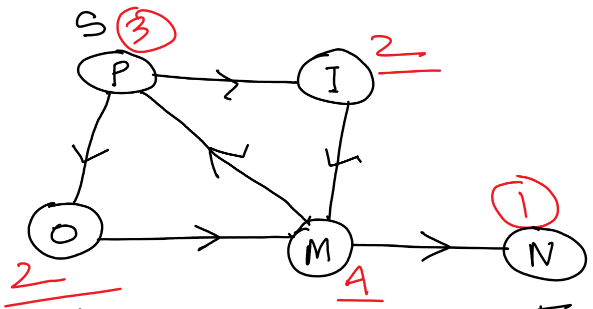


Euler's Trail:

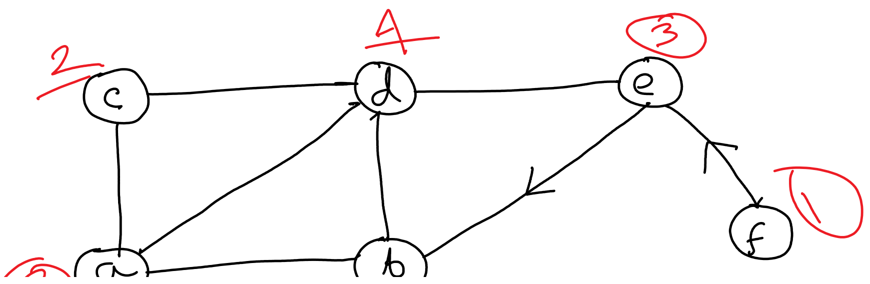
1. An Euler's trail is a trail that visits every edge exactly once
2. They are also called Euler's path
3. If all of the vertices have even degree except 2, then it is a Euler's trail.
4. We start in an odd vertex and stop at an odd vertex.

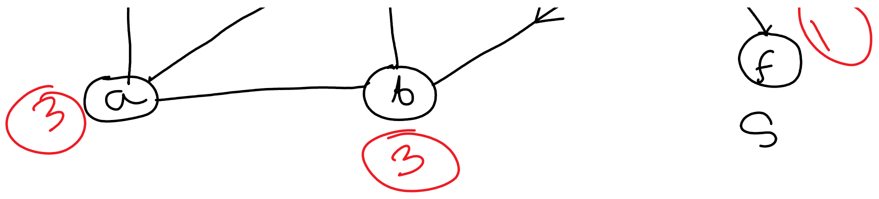


It is an Euler's Trail

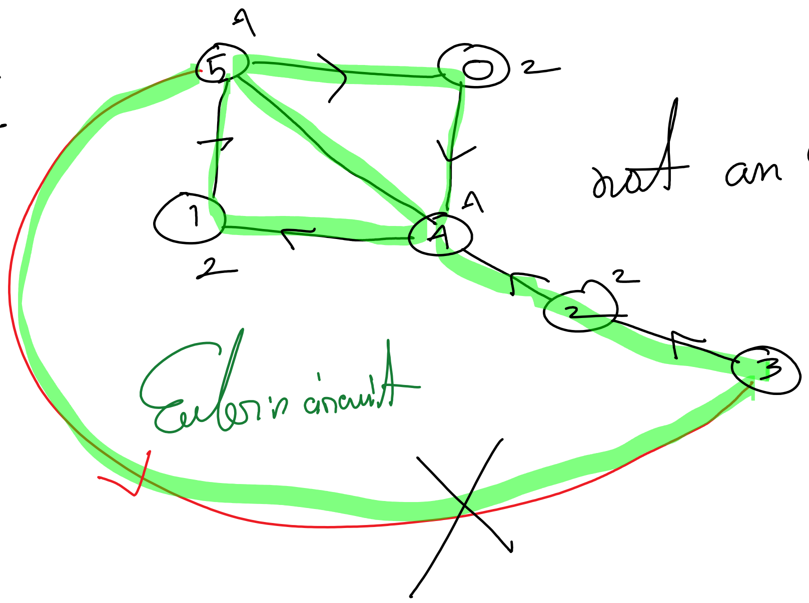


It is an Euler's Trail





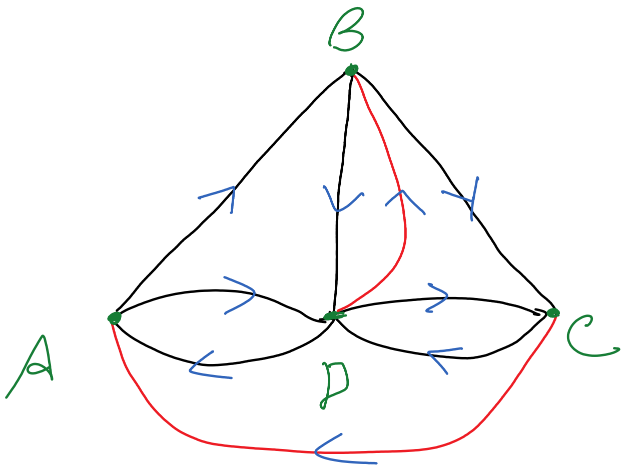
Euler's Circuit



not an Euler's circuit

Euler's Circuit

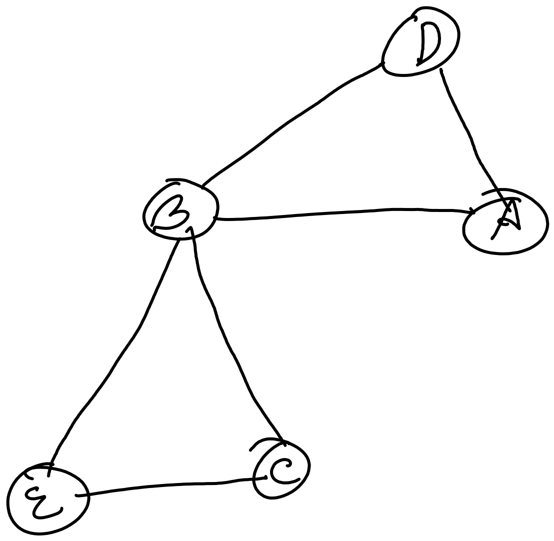
1. An Euler circuit is a circuit that uses every edge in a graph with no repeats. Being a circuit, it must start and end at the same vertex.
2. An Euler's circuit is possible only if all of the vertices have even degrees.
3. A connected graph G is Eulerian if and only if the degree of each vertex of G is even.
4. There cannot be any vertex which is not connected. That means a vertex with 0 degree is not allowed in Eulerian.



Fleury's Algorithm

The Fleury's algorithm is an algorithm gives us a systematic way to find the Euler's circuit. To apply the Fleury's algorithm, follow the steps:

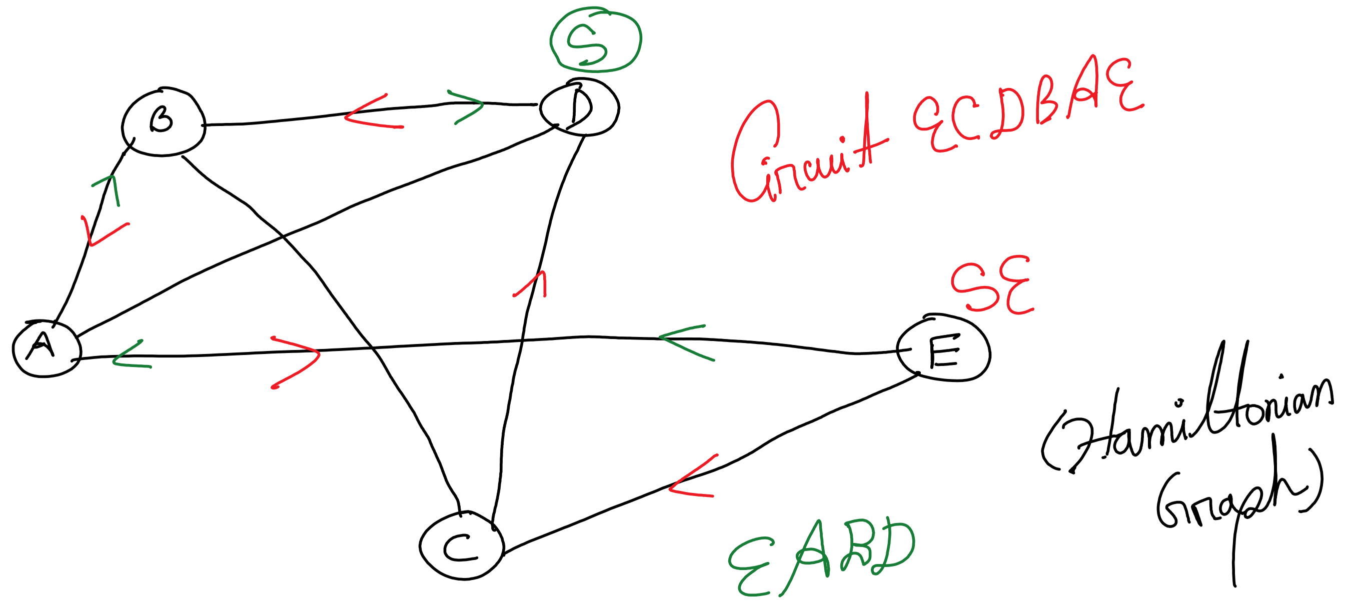
1. First of all check if the graph is connected. We can apply Fleury's algorithm only when the graph is fully connected.
2. Then check if all of the vertices have even degree. It is the condition for a graph to be Euler's circuit.
3. Create a replica of the existing graph.
4. Start at any vertex.
5. Move through an edge and remove that edge.
6. If this removal disconnect the unexplored part of the graph, bring back the edge. We cannot remove it.
7. Keeping traversing through the edges and removing them as long as the none of the removals disconnect the unexplored part of the graph.
8. If no edge is remaining at the end, then path we traversed is a Euler's circuit.

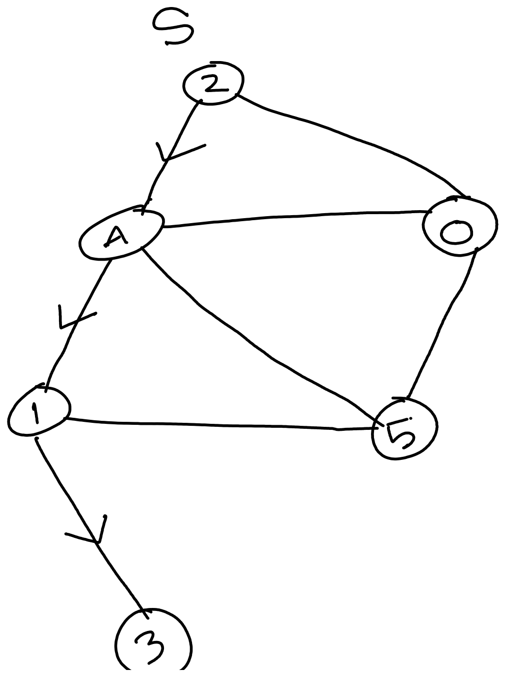


Hamiltonian Paths and Circuit

Hamiltonian Circuit:
A Hamiltonian circuit in a graph G is a circuit that contains each vertex in G exactly once except for the starting and ending vertex that appears twice. The procedure is:

1. Strat with any vertex
2. Visit every vertex for once
3. Check if it is possible to return to the starting vertex (starting vertex can be repeated)
4. If it is possible to return back to the starting vertex without repeating any vertex except the starting and the ending vertex, then it is a Hamiltonian circuit.
5. In Hamiltonian circuit, we are allowed to repeat edges.





3

Hamiltonian Path:
If we start at a vertex, visit every vertex for once and end at a different vertex, then we have a Hamiltonian path.

Ore's Theorem

✓① ✓② ✓③

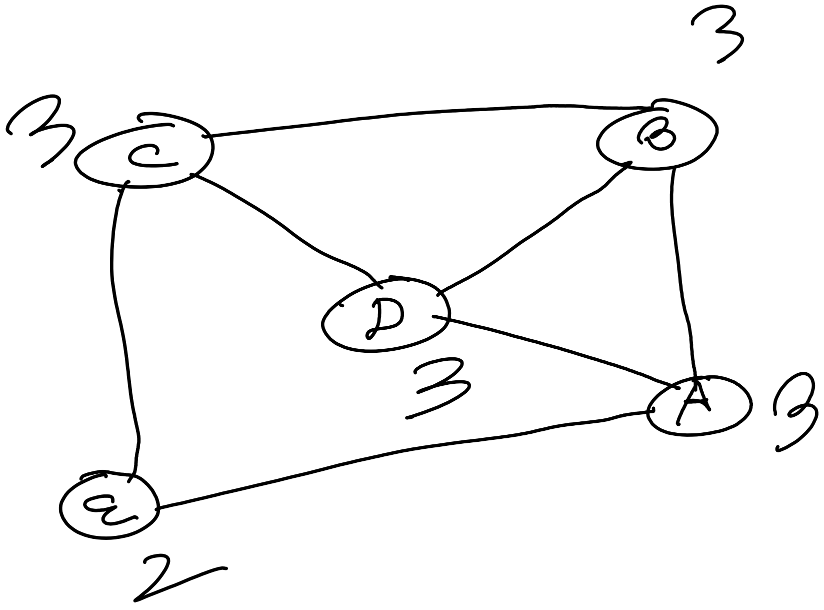
If a simple graph with n>=3 vertices and if deg(v)+deg(w)>=n for each pair of non-adjacent vertices v and w, then G is a Hamiltonian.

So, the conditions are:

1. The graph has to be simple. That means there cannot be any loop or parallel edge.
2. The number of vertices (n) has to be more than or equal to 3 (n>=3 vertices).
3. The sum of the non-adjacent vertices has to be more than or equal to number of vertices.

$n=5$

NAV	Sum of degrees
BE $\rightarrow$	5
AC $\rightarrow$	6
DE $\rightarrow$	5



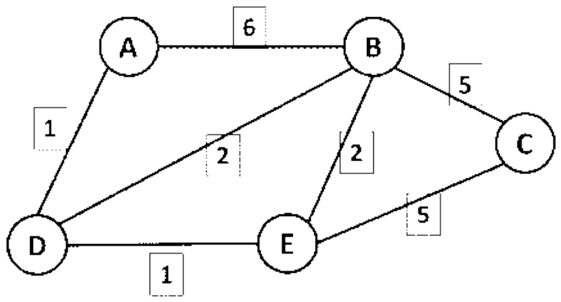
However, if these three conditions are not satisfied, it does not mean that the graph is not Hamiltonian.

The Shortest Path Problem (Dijkstra's Algorithm)

The algorithm
Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.
2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances through the current node. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B through A will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, the current value will be kept.
4. When we are done considering all of the unvisited neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

What do we need?
- A graph
- The Path cost of edge weight (numerical values)
- A table
- Two lists - Visited and Unvisited
- And of course the algorithm itself

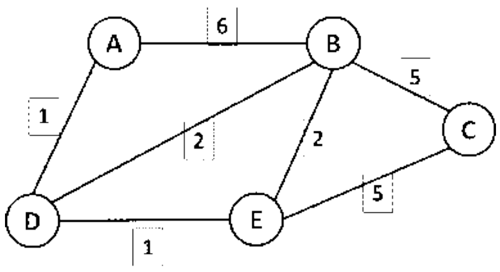


Vertex	Shortest Distance from the Starting Vertex	Previous Vertex
A		
B		
C		
D		
E		

Visited = []
Unvisited = [A, B, C, D, E]

Vertex	Shortest Distance from the Starting Vertex	Previous Vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

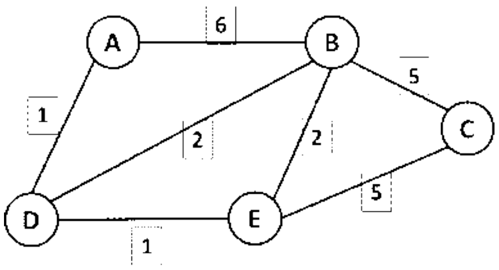
1. The distance from the starting vertex to starting vertex is 0. We are starting from A, therefore distance from A to A is 0.
2. Assign maximum distance to other vertices. The maximum distance is $\infty\ (infinity)$



3. Examine the unvisited neighbors from current vertex. Our current vertex is A.
4. Here B and D are the unvisited neighbors
5. Calculate the distance of the neighbors from the current vertex.
   a. Distance from A to B is 0+6 = 6
   b. Distance from A to D is 0+1 = 1
6. If the new distance is smaller than the current shortest distance, update the current shortest distance.
7. We went to B and D from A. Therefore A is the previous vertex. Add it in the table.
8. The node A has been visited. We won't come back here again.
9. Remove it from the unvisited list and add it to the visited list.

Vertex	Shortest Distance from the Starting Vertex	Previous Vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]
Unvisited = [B, C, D, E]

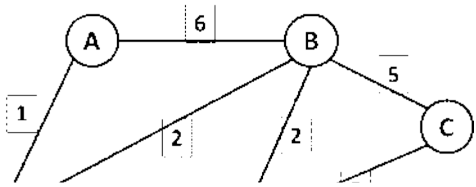


10. Again visit the unvisited vertex with shortest distance. In our example, it is D.
11. Then examine the unvisited neighbors. The neighbors of D are B and E. We won't consider A because it has been visited.
12. Calculate the distance of B and E from the starting vertex A through current vertex. Our current vertex is D:
    a. Distance of B from A through D = 1+2 = 3
    b. Distance of E from A through D = 1+1 = 2
13. If the new distance is smaller than the current shortest distance, update the current shortest distance.
    a. Current shortest distance of B is 6. The new distance is 3 which is smaller than 6. So update the distance by 3
    b. Current shortest distance of E is ∞. The new distance is 1. So update it
14. We have visited B and E through D. Therefore the previous vertex is D.
15. Then remove the B from the unvisited list and add it to the visited list.

Vertex	Shortest Distance from the Starting Vertex	Previous Vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

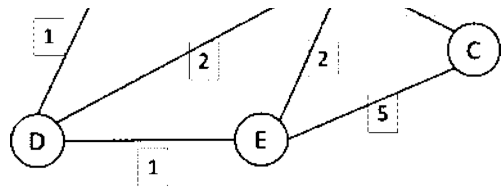
Visited = [A, D]
Unvisited = [B, C, E]

16. Again visit the unvisited vertex with shortest distance. In our example, it is E.
17. Then examine the unvisited neighbors. The neighbors of E are B and C. We won't consider D because it has been visited.
18. Calculate the distance of B and C from the starting vertex A through current vertex. Our current vertex is E:
    a. Distance of B from A through E = 2+2 = 4



won't consider D because it has been visited.

18. Calculate the distance of B and C from the starting vertex A through current vertex. Our current vertex is E:
    a. Distance of B from A through E = 2+2 = 4
    b. Distance of C from A through E = 2+5 = 7
19. If the new distance is smaller than the current shortest distance, update the current shortest distance.
    a. Current shortest distance of B is 3. The new distance is 4 which is larger than 3. So do not update the distance.
    b. Current shortest distance of E is ∞. The new distance is 7. So update it
20. We have visited C through E. Therefore the previous vertex is E.
21. Then remove the E from the unvisited list and add it to the visited list.



Vertex	Shortest Distance from the Starting Vertex	Previous Vertex
A	**0**	
B	3	**D**
C	7	**E**
D	1	**A**
E	1	**D**

Visited = [A, D, E]
Unvisited = [B, C]

Keep repeating the process