AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

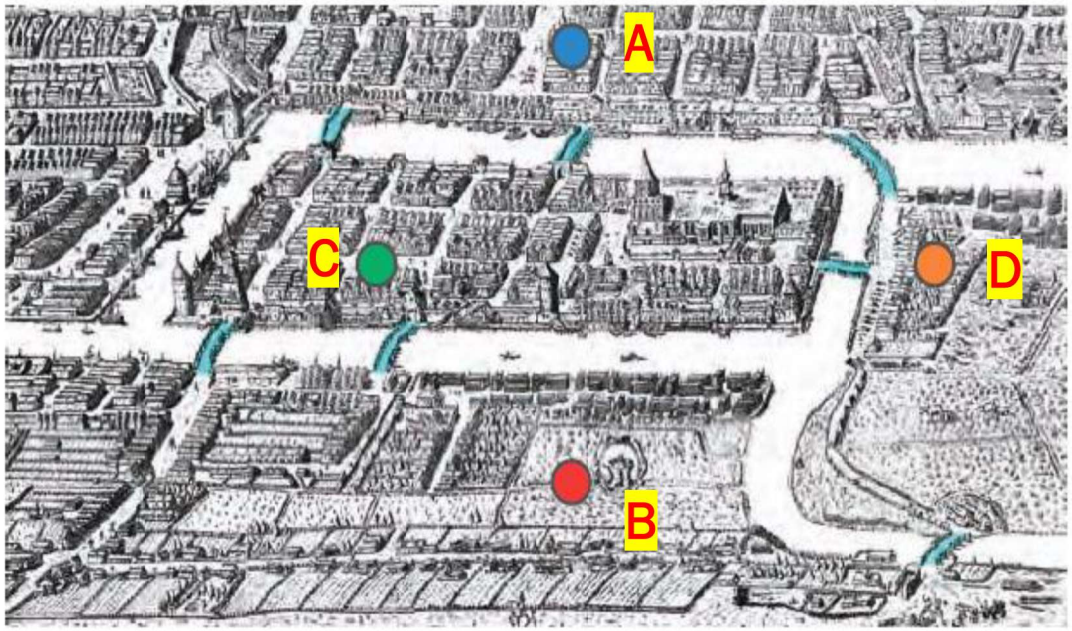
WALKING IN DE BRUIJN GRAPHS

22BIO211: Intelligence of Biological Systems - 2

Dr. Manjusha Nair M
Amrita School of Computing, Amritapuri

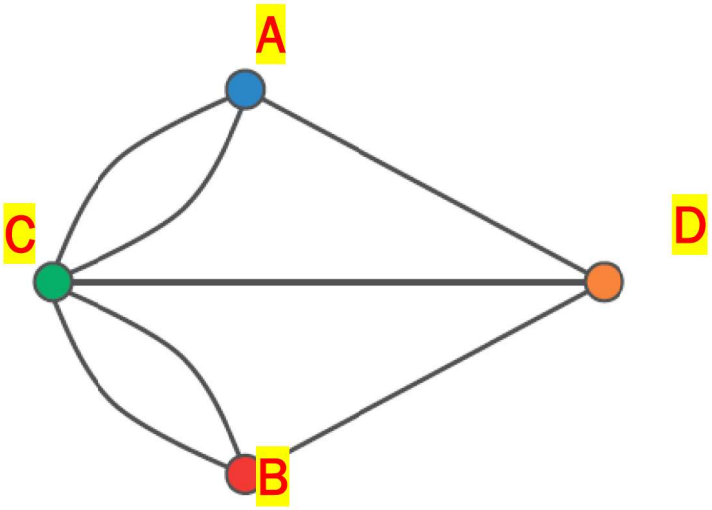
Email : manjushanair@am.amrita.edu
Contact No: 9447745519

The Seven Bridges of Königsberg



Prussian city of Königsberg: 1735

Two *banks* of the Pregel River: A , B
Two river islands: C,D



Seven bridges connected these four different parts of the city : become Edges of the graph

Is it possible to set out from my house, cross each bridge exactly once, and return home?

Eulerian paths

String Reconstruction Problem : reduces to finding a path in the de Bruijn graph that visits every edge exactly once

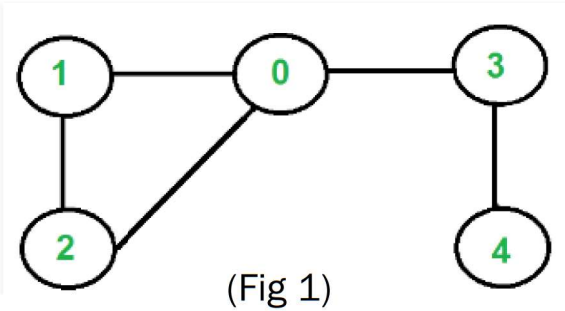
An ***Euler path***, in a graph or multigraph, is a walk through the graph which uses every edge exactly once.

Eulerian Path Problem:
Construct an Eulerian path in a graph.

Input: A directed graph.
Output: A path visiting every edge in the graph exactly once (if such a path exists).

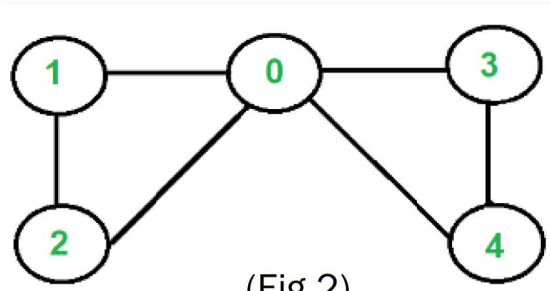
Not all graphs have Eulerian walks.

Eulerian Paths/Cycles : Undirected Graph



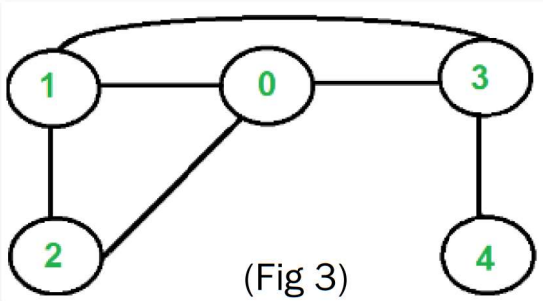
(Fig 1)

Eulerian path exist: 4-3-0-1-2-0
No Eulerian cycle



(Fig 2)

Eulerian path exist
Eulerian cycle exist : 4-3-0-1-2-0-4



(Fig 3)

Not Eulerian

Note that only one vertex with odd degree is not possible in an undirected Eulerian graph (Fig 3)

Eulerian Paths/Cycles : Undirected Graph

A graph is called **Eulerian** if it has an Eulerian Cycle and called **Semi-Eulerian** if it has an Eulerian Path.

An undirected graph has Eulerian Path if

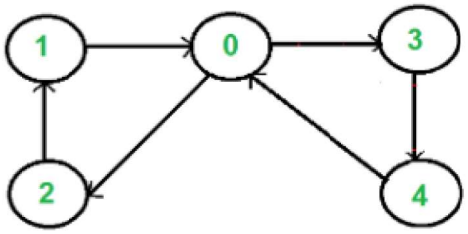
- zero or two vertices (at most two vertices) have odd degree and all other vertices have even degree. (Fig 1)

An undirected graph has Eulerian cycle

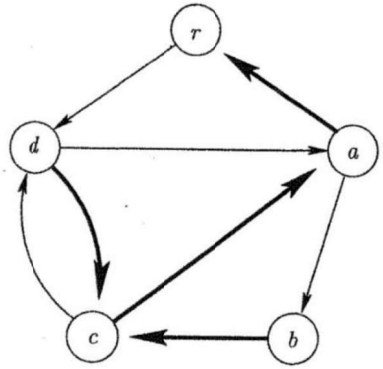
- if All vertices have even degree (Fig 2).

Note that only one vertex with odd degree is not possible in an undirected Eulerian graph (Fig 3)

Eulerian paths/cycles : Directed graph



Eulerian cycle : 1-> 0-> 3-> 4-> 0-> 2->1



r→d→a→b→c→d→c→a→r

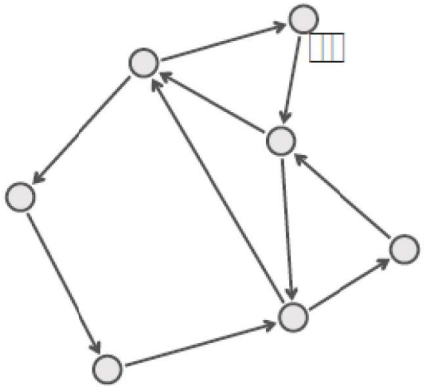
Node is **balanced** if indegree equals outdegree .
Node is **semi-balanced** if indegree differs from outdegree by 1

A directed, connected graph is Eulerian if and only if it has at most 2 semi-balanced nodes (starting and finishing nodes) and all other nodes are balanced

Euler Theorem

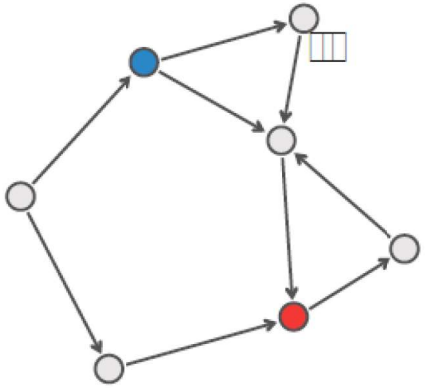
- Any Eulerian graph must be balanced
- Any Eulerian Graph must be strongly connected
- **Euler's Theorem:** Every balanced, strongly connected directed graph is Eulerian.

Some Example Graphs



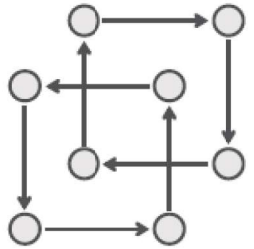
Balanced

$I_N(v) = 1$ and $O_{UT}(v) = 2$



$I_N(w) = 2$ and $O_{UT}(w) = 1$

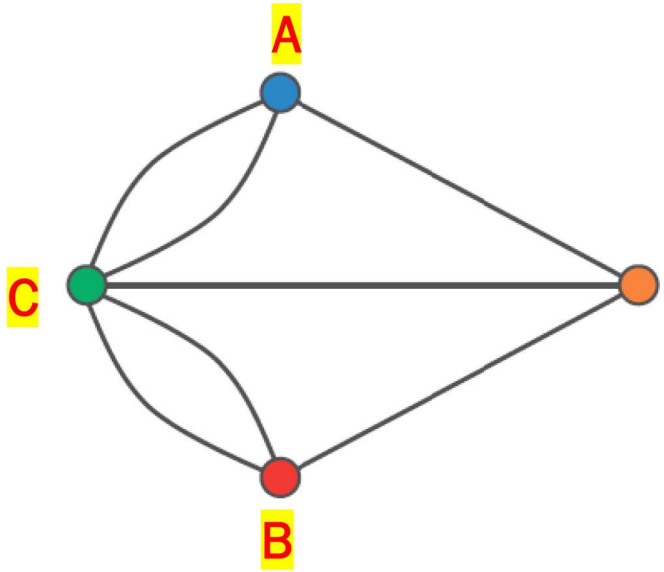
Unbalanced



Balanced but disconnected

Euler's Solution to Königsberg Problem

Note that an Eulerian cycle in Königsberg would immediately provide the residents of the city with the walk they had wanted.



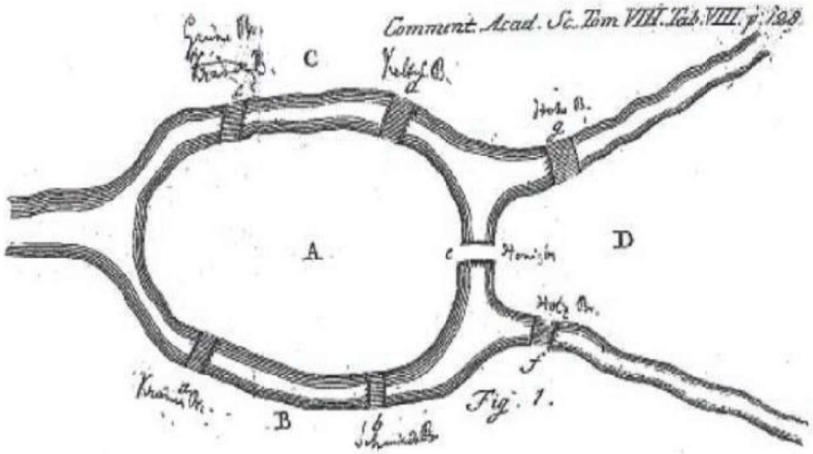
Vertices (Cities)	Degree
A	3
B	3
C	5
D	3

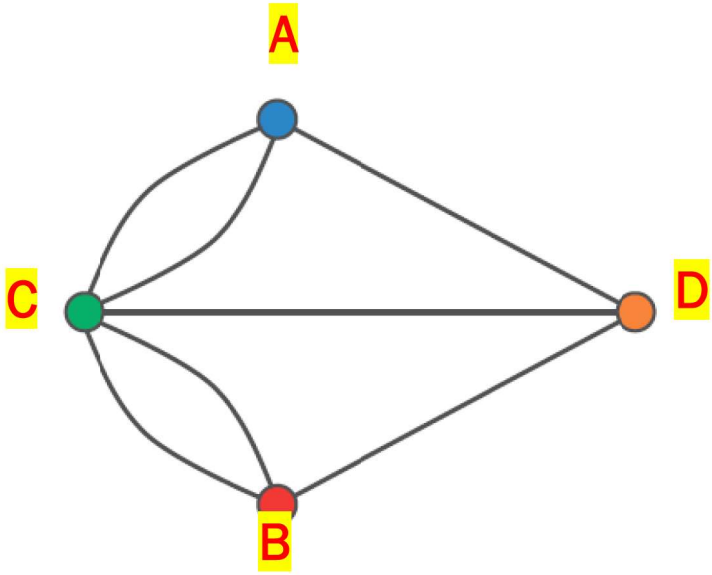
No walk can cross each bridge exactly once.

The graph Königsberg is not Eulerian

The Seven Bridges of Königsberg

Euler's presented his solution of the Bridges of Königsberg Problem to the Imperial Russian Academy of Sciences in St. Petersburg in 1735.



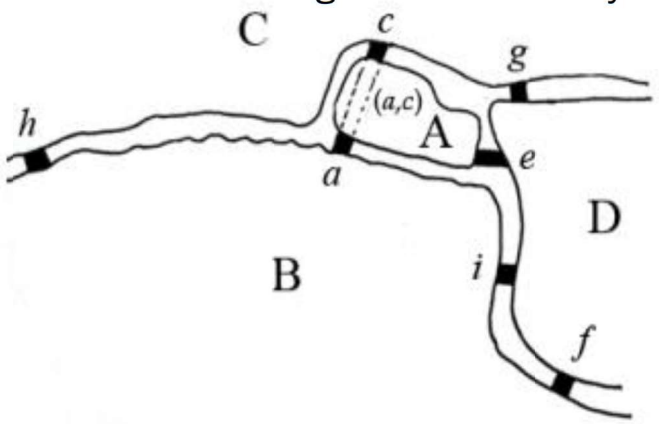


Euler's illustration of Königsberg, Prussian city of Königsberg: 1735

The Seven Bridges of *Kaliningrad*

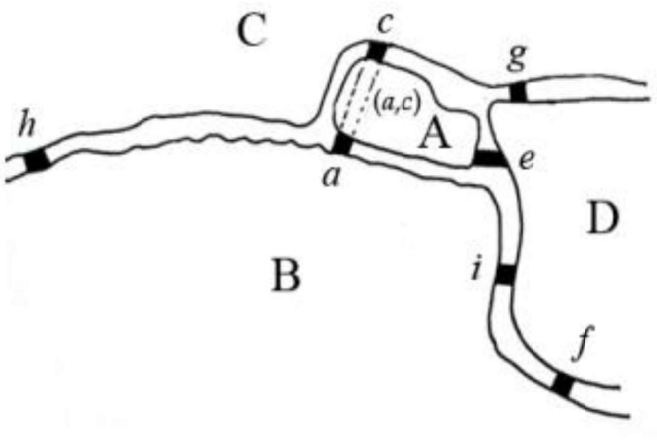
- Königsberg was largely destroyed during World War II, its ruins were captured by the Soviet army.
- The city was renamed Kaliningrad in 1946 in honor of Soviet revolutionary Mikhail Kalinin.

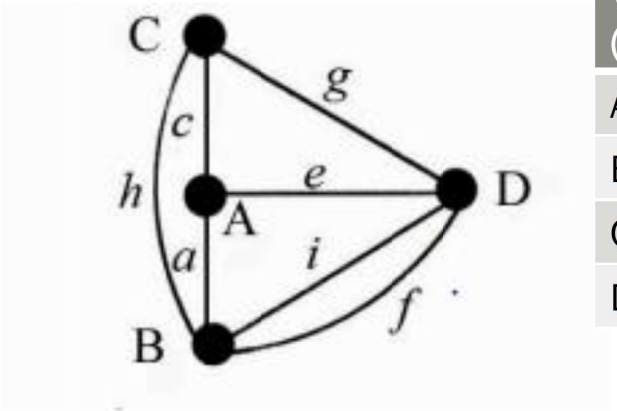
The Seven Bridges of modern-day Kaliningrad



The Seven Bridges of *Kaliningrad*

The Seven Bridges of modern-day Kaliningrad





Vertices (Cities)	Degree
A	3
B	4
C	3
D	4

Eulerian Path : C -> B -> A -> C -> D -> B -> D -> A

- City of Kaliningrad still does not contain an Eulerian cycle.
- However, this graph does contain an Eulerian path
- Which means that Kaliningrad residents can walk crossing every bridge exactly once, but cannot do so and return to where they started.

Checking Eulerian Graphs

```
class Graph:
    def __init__(self):
        self.graph = defaultdict(list)
    def add_edge(self, u, v):
        self.graph[u].append(v)
```

```
def CheckEulerian(self):
    # Check if each node has equal in-degree and out-degree
    in_degree = out_degree = defaultdict(int)
    for u in self.graph:
        out_degree[u] = len(self.graph[u])
        for v in self.graph[u]:
            in_degree[v] += 1
    for node in in_degree:
        if in_degree[node] != out_degree[node]:
            return False
    return True
```

Constructing Eulerian Cycle

```
EULERIANCYCLE(Graph)
    form a cycle Cycle by randomly walking in Graph (don't visit the same edge twice!)
    while there are unexplored edges in Graph
        select a node newStart in Cycle with still unexplored edges
        form Cycle' by traversing Cycle (starting at newStart) and then randomly walking
        Cycle ← Cycle'
    return Cycle
```

Linear Time Algorithms exist

Hierholzer's algorithm

- Choose any starting vertex v, and follow a trail of edges from that vertex until returning to v.
  - *It is not possible to get stuck at any vertex other than v, because the even degree of all vertices ensures that, when the trail enters another vertex w there must be an unused edge leaving w.*
  - *The tour formed in this way is a closed tour, but may not cover all the vertices and edges of the initial graph.*
- 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.
- Since we assume the original graph is connected, repeating the previous step will exhaust all edges of the graph.

Constructing Eulerian Cycle

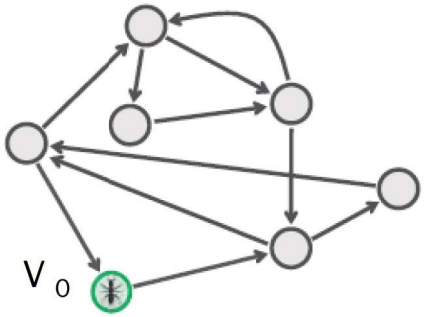


Fig A

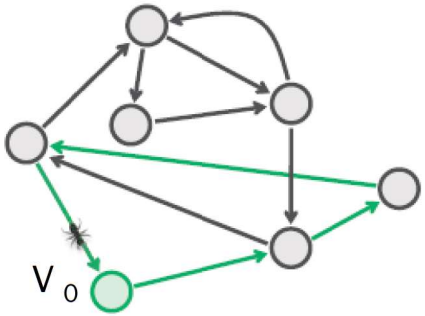


Fig B

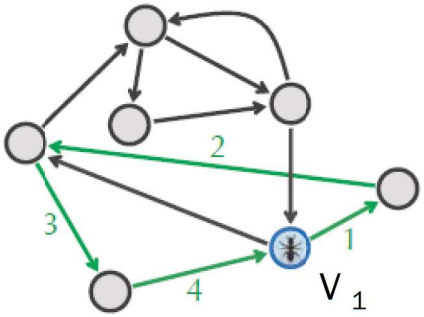


Fig C

Cycle 1

Cycle 2

Cycle 3

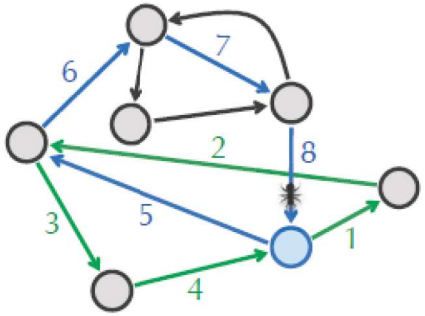


Fig D

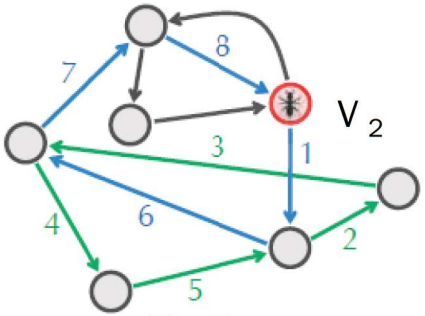


Fig E

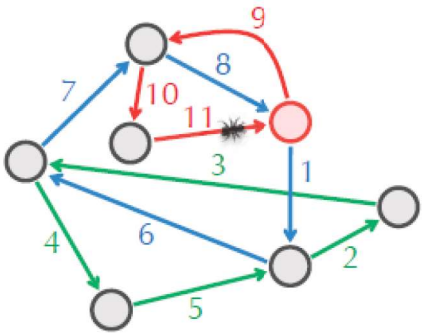


Fig F

Generate larger and larger cycles at each iteration, and so we are guaranteed that sooner or later some Cycle $_m$ will traverse all the edges in Graph

Hierholzer's algorithm

- Construct an Eulerian cycle by recursively traversing the graph.

```
def EulerianCycle(self, u, visited, cycle):
    while self.graph[u]:
        v = self.graph[u].pop()
        self. EulerianCycle(v, visited, cycle)
    cycle.append(u)
```

- Start from an arbitrary node and explore all edges incident to that node.

```
# Start from any node,
def FindEulerianCycle(self):
    if not self.CheckEulerian():
        print("No Eulerian cycle exists in the given graph.")
        return None
    cycle = []
    start_node = next(iter(self.graph))
    self.EulerianCycle(start_node, set(), cycle)
    return cycle[::-1]
```

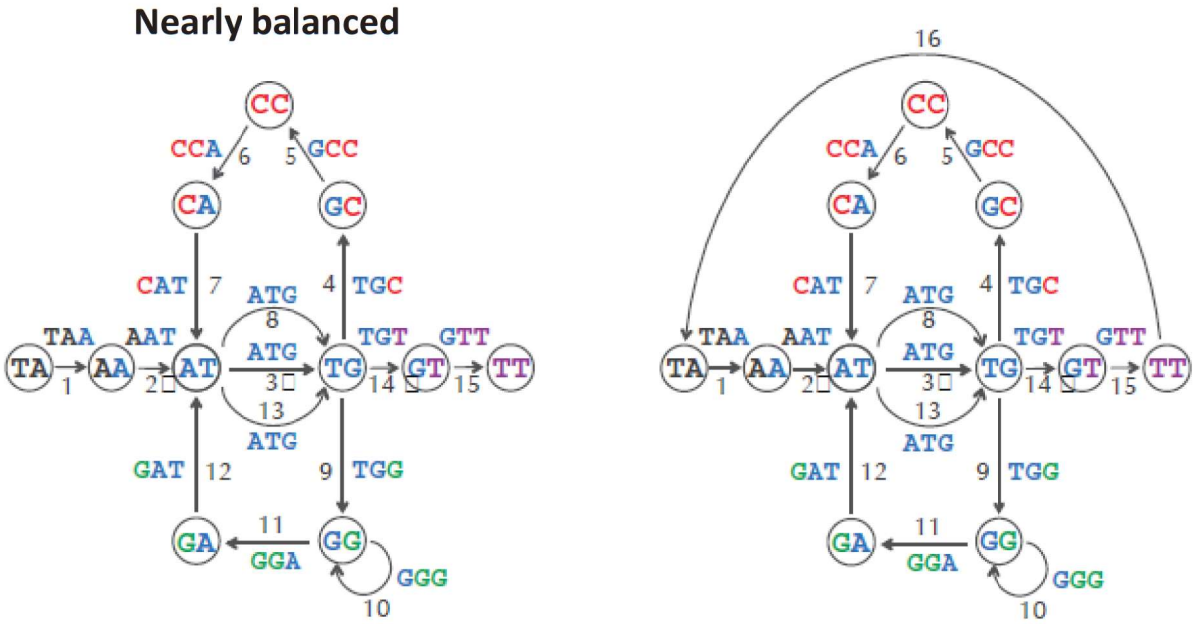
Hierholzer's algorithm

- # Example Usage:

```
g = Graph()
g.add_edge(0, 1)
g.add_edge(1, 2)
g.add_edge(2, 0)
g.add_edge(2, 3)
g.add_edge(3, 0)
eulerian_cycle = g.FindEulerianCycle()
if eulerian_cycle:
    print("Eulerian Cycle:", eulerian_cycle)
```

-

From Eulerian Cycle to Eulerian Path



Transforming an Eulerian path (left) into an Eulerian cycle (right) by adding an edge

Adding an edge between unbalanced nodes makes the graph balanced and strongly connected

Summary

- Euler Theorem
- The Seven Bridges of *Koningsberg*
- The Seven Bridges of *Kaliningrad*
- Constructing Eulerian Cycle
- From Eulerian Cycle to Eulerian Path