Subgraphs, Paths and Connectivity

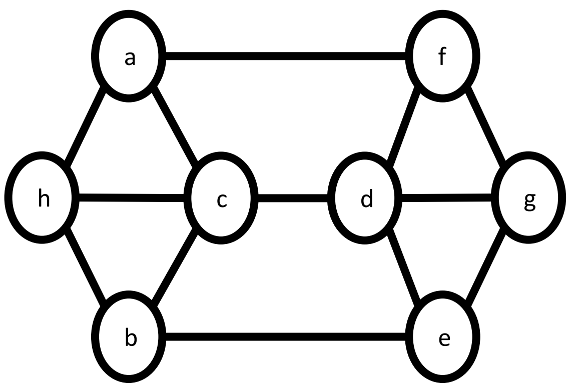
Lecture 25
By Marina Barsky

Subgraphs

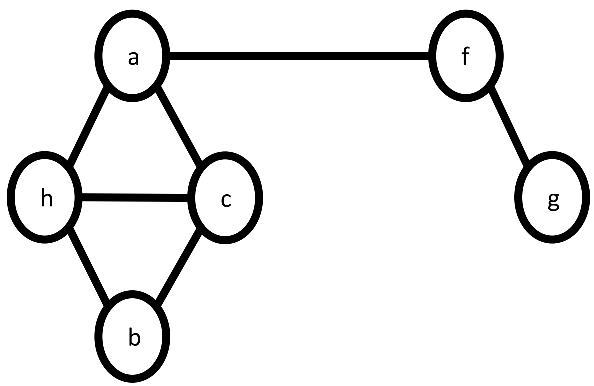
A ***subgraph*** of a graph is obtained by deleting any subset of vertices and edges.

- If a vertex is deleted, then all of its incident edges disappear

A subgraph is ***spanning*** if it includes **all** of the vertices (only some edges are deleted).

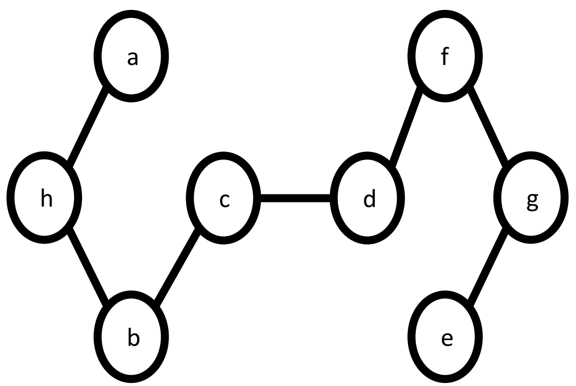


A graph.



A non-spanning subgraph.

An induced subgraph G[{a, b, c, h, f, g}].



A spanning subgraph.

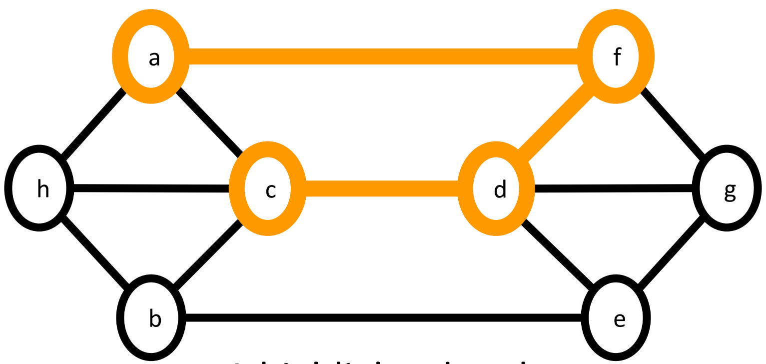
An ***induced subgraph*** is obtained by deleting any subset of vertices.
It is denoted by G[U] where U is the set of vertices that are not deleted.

Paths

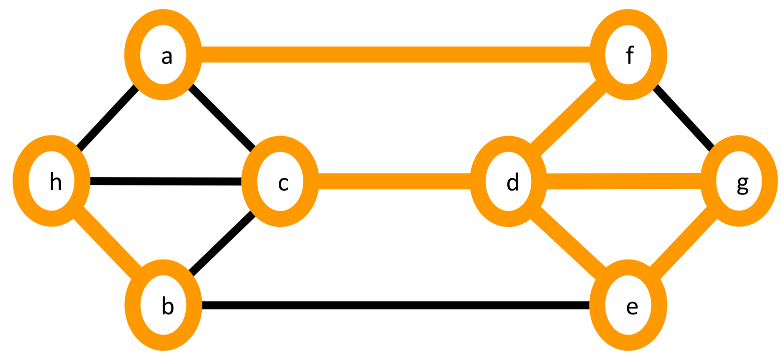
A ***path*** of length k is an alternating sequence of vertices and edges:

$$v_1,\ (v_1, v_2),\ v_2,\ (v_2, v_3),\ v_3,\ \ldots,\ v_k,\ (v_k, v_{k+1}),\ v_{k+1} \text{ where } v_i \neq v_j \text{ if } i \neq j.$$

In other words, there are k+1 vertices and k edges, the vertices are distinct, and each edge connects consecutive vertices on the path.



A highlighted path
a, (a,f), f, (f,d), d, (d,c), c



This is not a path since it is disconnected and also d appears multiple times.

The ***length of a path*** is the number of traversed edges.
A path from u to v is a ***shortest path*** if there is no shorter path from u to v.
For example, there are two shortest paths from f to e above.

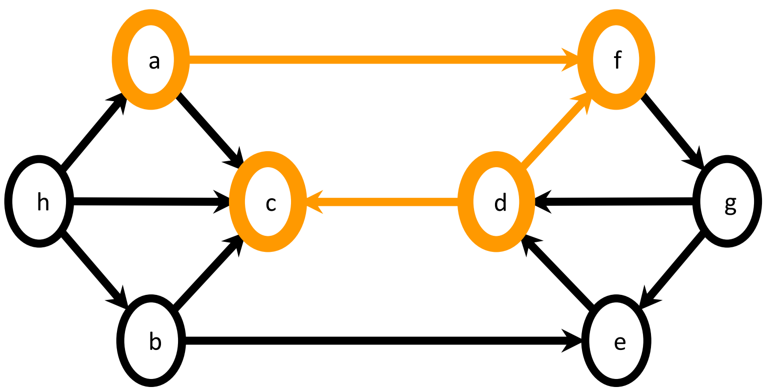
Cycles

- A cycle (sometimes called a circuit) in a graph is a path where the first vertex is the same as the last one
- All the edges in a cycle are distinct

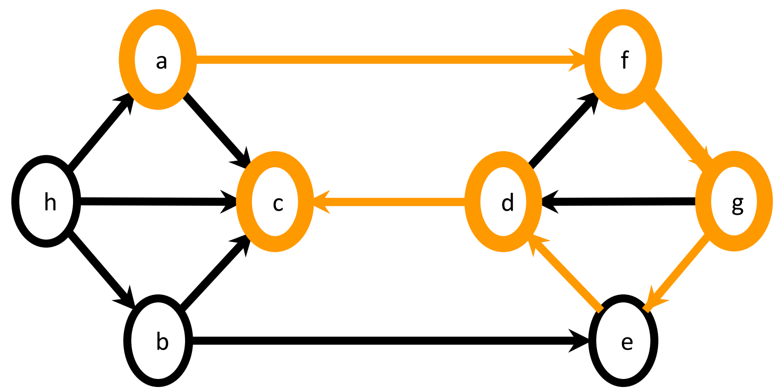
Directed Paths

In a directed graph each edge is oriented in one of two ways with respect to a path:

- The edge is *forward* if it has the form v_i, (v_i, v_{i+1}), v_{i+1}.
- The edge is *backward* if it has the form v_i, (v_{i+1}, v_i), v_{i+1}.



A highlighted path
a, (a,f), f, (f,d), d, (d,c), c
where (f,d) is the only backwards edge.

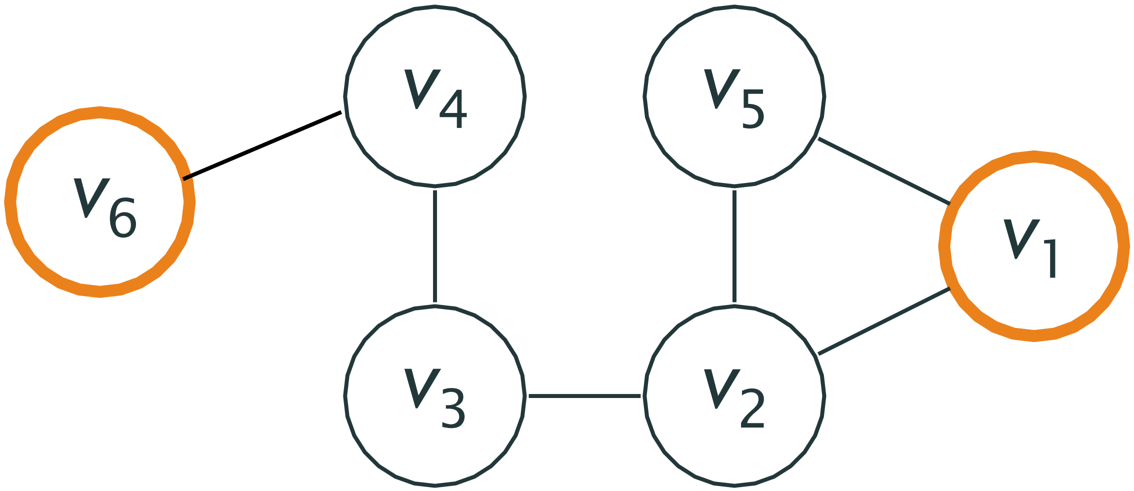


A directed path from a to c.

A path is a *directed path* if every edge is a forward edge.

Connectivity in undirected graphs

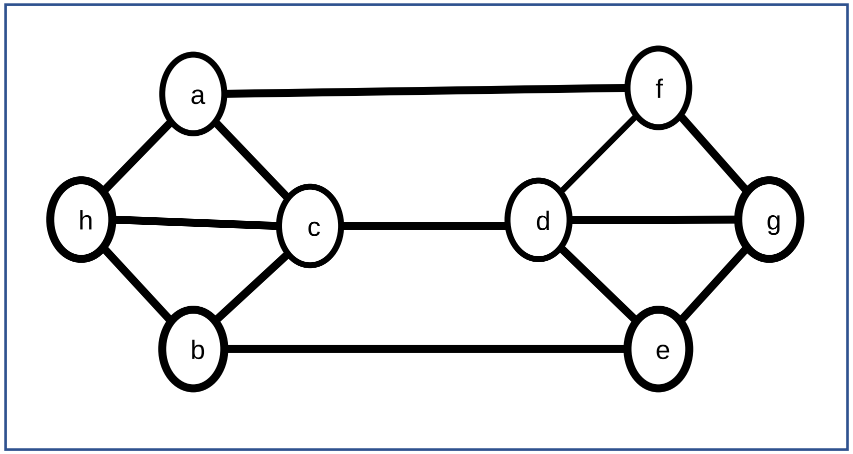
- Two vertices are connected, if there is a path between them
- The definition is transitive: if *u* and *v* are connected and *v* and *w* are connected, then *u* and *w* are connected as well



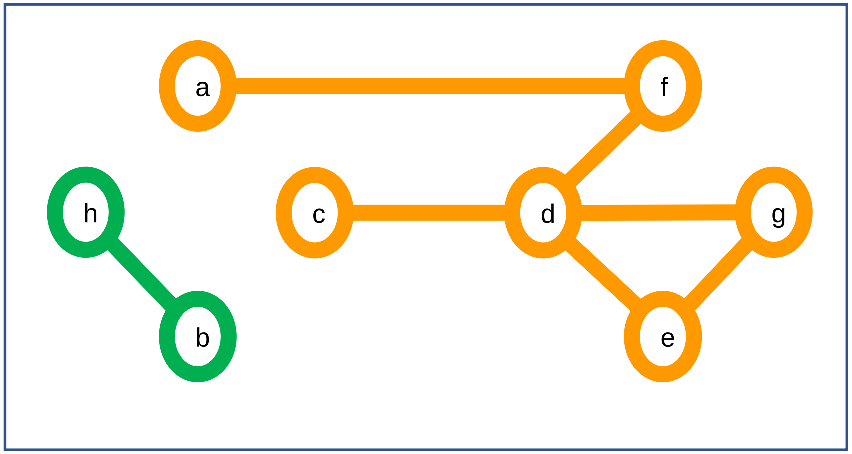
v_1 and v_6 are connected.

Connected graph

- A graph is connected, if any two of its nodes are connected. In other words, there is a path between any pair of nodes



This graph is connected.

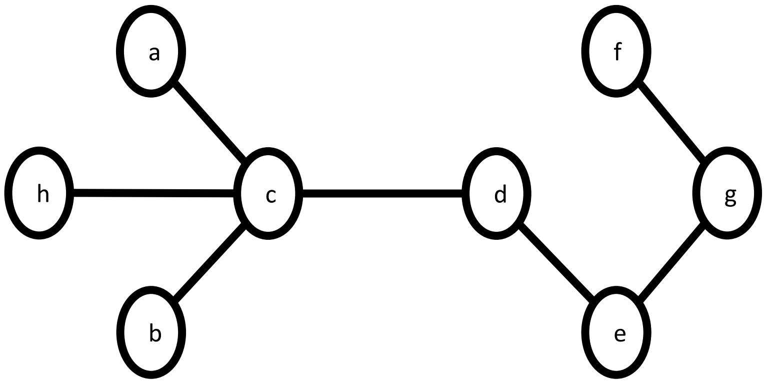


This graph is not connected.

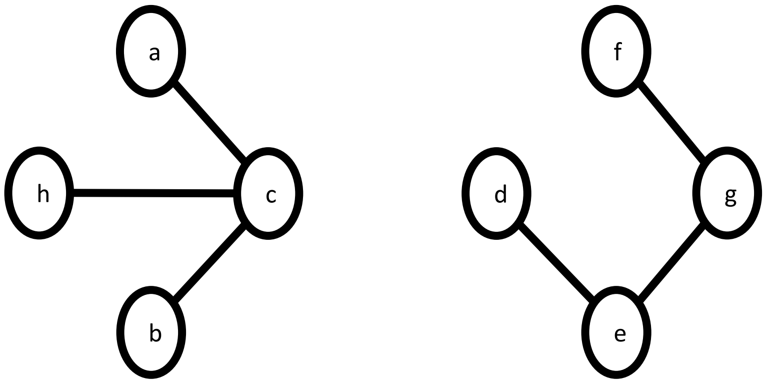
Trees and Forests

Trees and Forests

A ***tree*** is a connected **acyclic** graph. That is, each node is connected to some other node, and there are no cycles.
A *forest* is an acyclic graph (i.e. its connected components are trees.)
A *leaf* is a vertex of degree one, and the other vertices are *internal nodes*.



A tree with four leaves and four internal vertices.



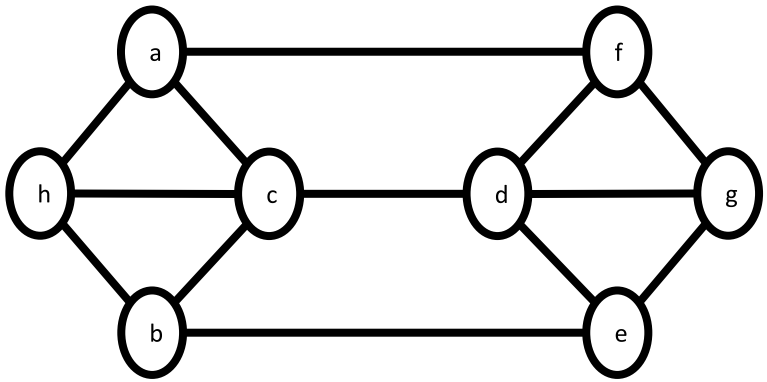
A forest with two component trees.

Lemmas:

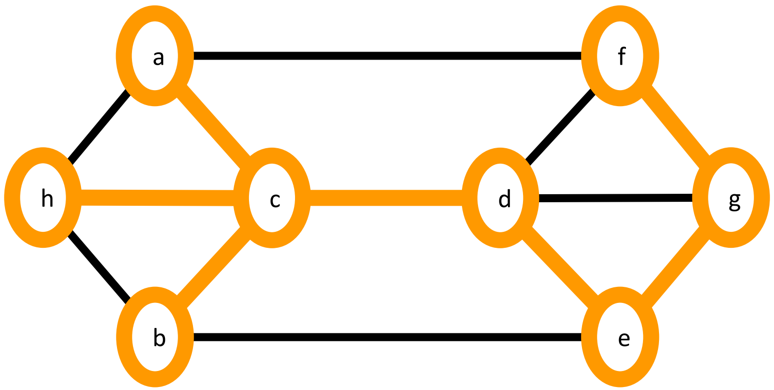
- A tree on *n* vertices has *n*-1 edges.
- A forest with *n* vertices and *c* components has *n*-*c* edges.
- There is a unique path between any two vertices within a tree.

Spanning Trees

A ***spanning tree*** is a subgraph that is spanning and is a tree.



A connected graph.

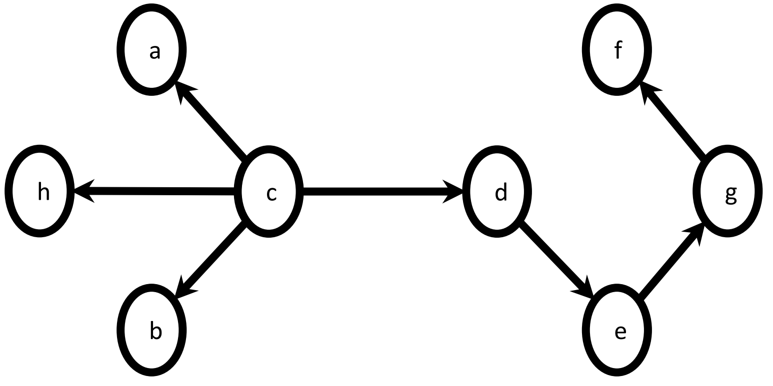


A spanning tree of the graph.

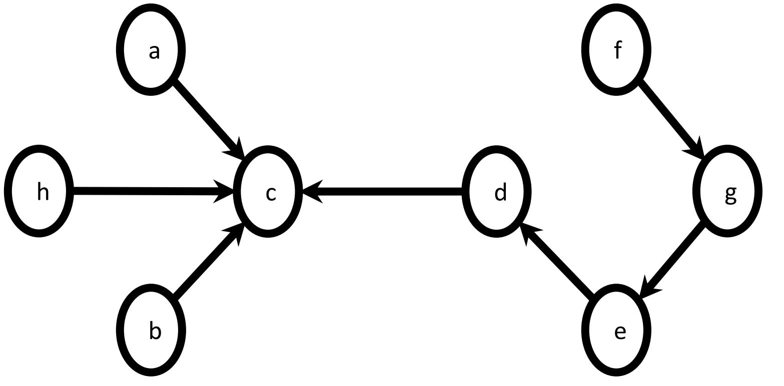
Lemma:
A graph is connected if and only if it has a spanning tree.

Rooted Trees

A ***rooted tree*** has a specified *root vertex*.
Every edge joins a *parent* and a *child* vertex, where the parent is closer to the root.



A rooted tree from vertex c.
Edges are directed outward from the root (i.e. parent to child).



A rooted tree from vertex c.
Edges are directed inward to the root (i.e. child to parent).

Sometimes we direct edges *outward* from the root or *inward* to the root.
When rooted trees are drawn the root is typically placed at the top and every parent is placed above its children.

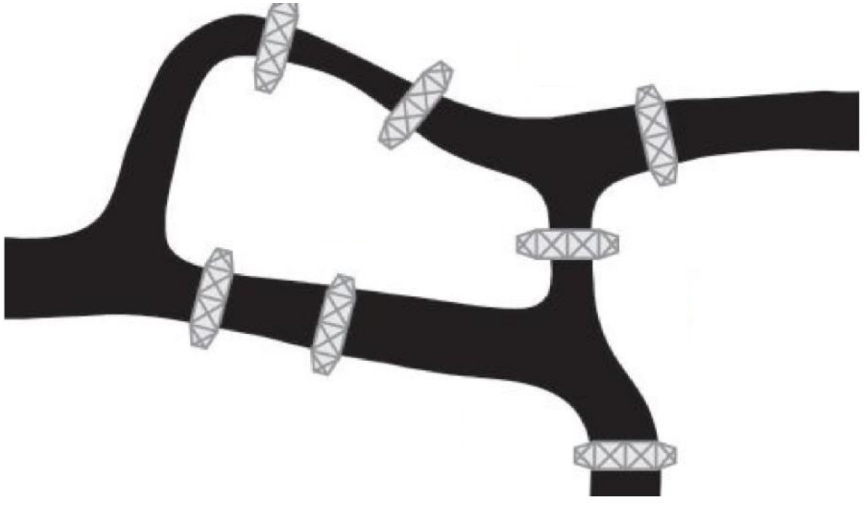

Leonhard Euler
1707 - 1783

Modeling with graphs and paths

Seven bridges of Königsberg

Euler's dilemma:

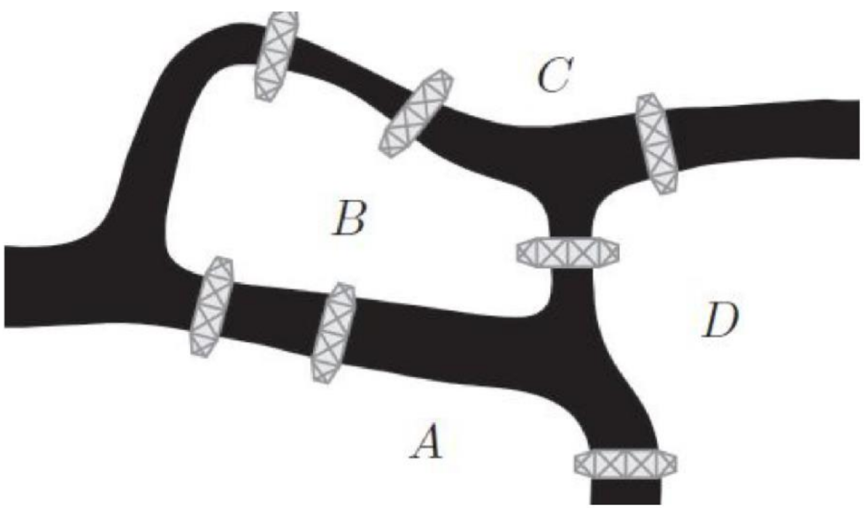
Can I take a walk and cross each bridge exactly once?



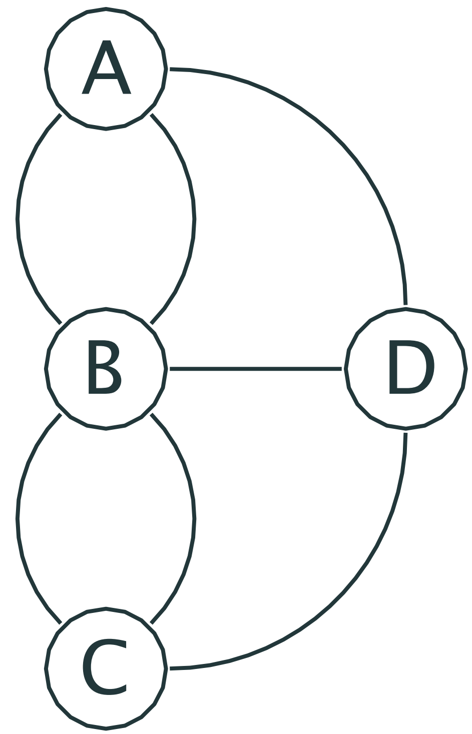
Seven bridges of Königsberg

Eulerian path problem

Is there a path which visits **every edge** of the graph exactly once?

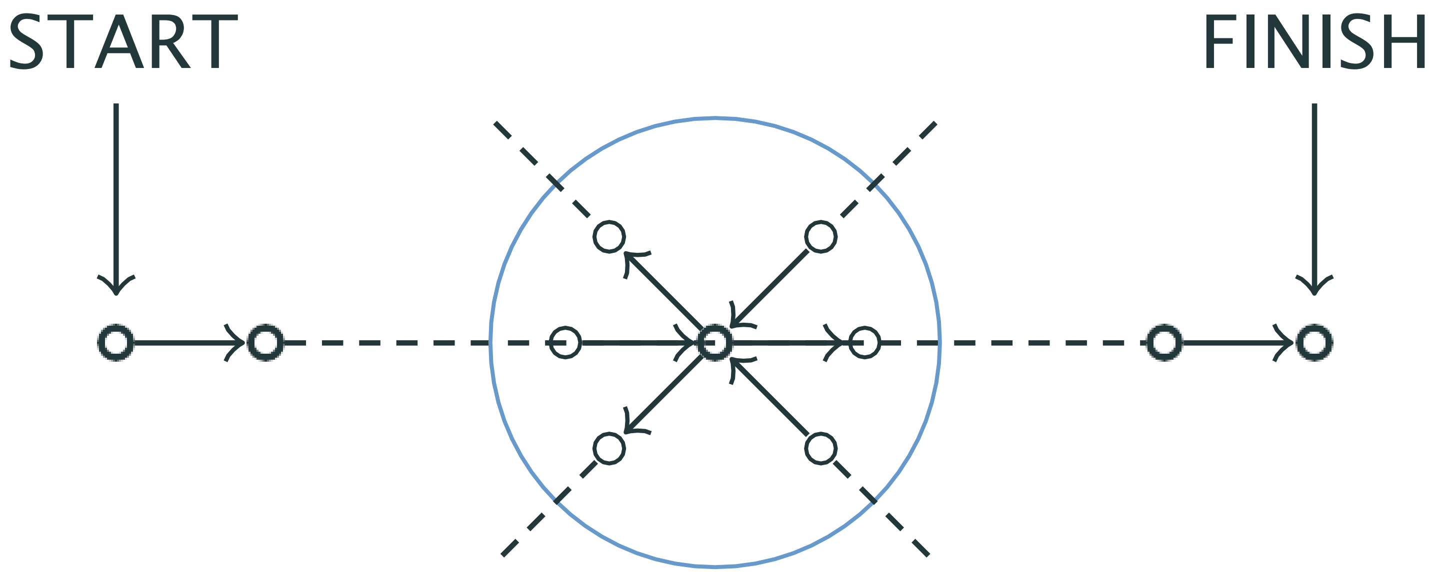


Seven bridges of Königsberg



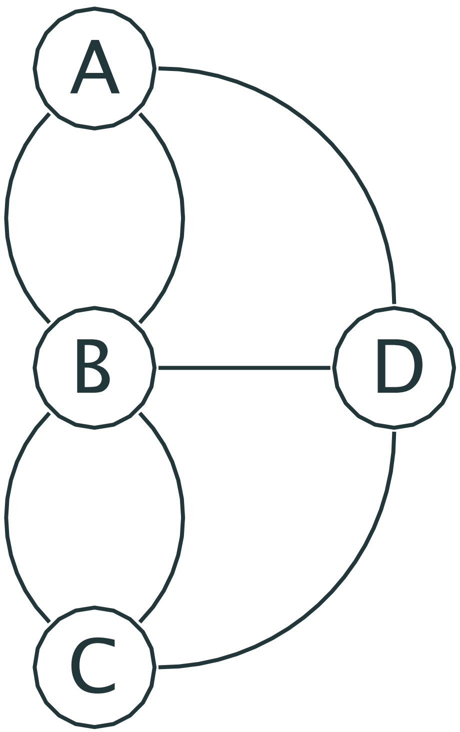
Modeled as Graph

Eulerian Path

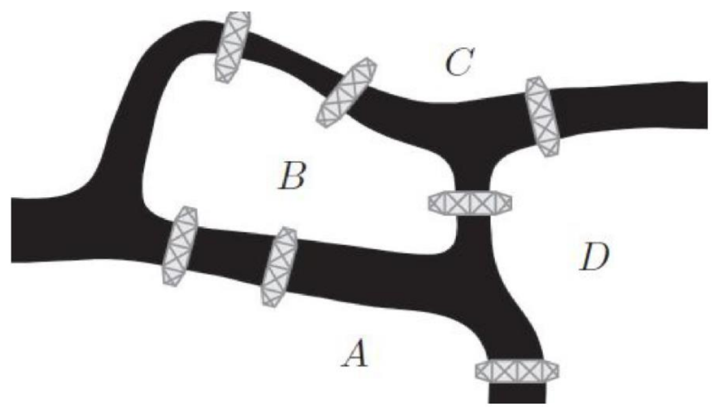


Necessary condition: all but START and FINISH vertices must have **even** degrees. Why?

Seven bridges of Königsberg



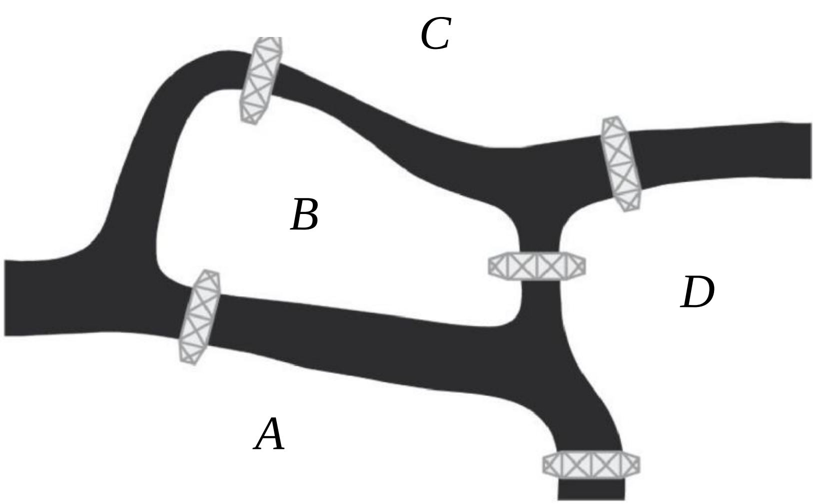
Is there an Eulerian Path through these seven bridges?



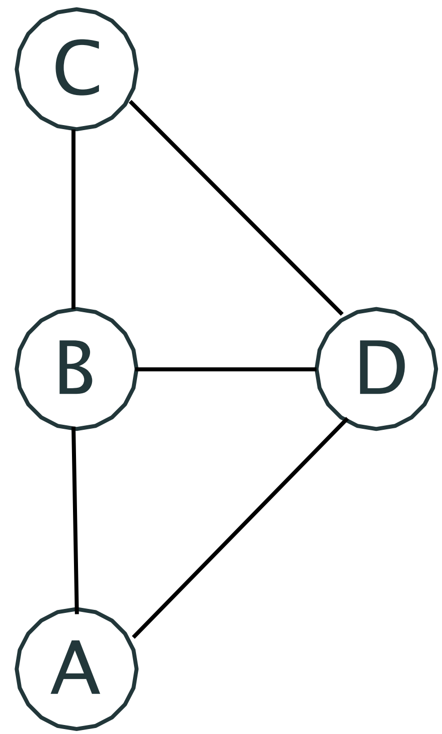
Königsberg, 17-th century

Five Bridges of Kaliningrad

Is there an Eulerian Path through these five bridges?



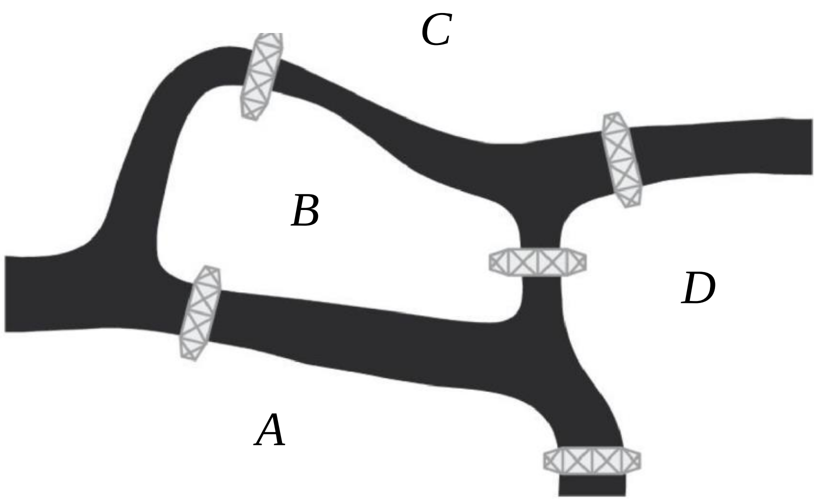
Königsberg (Kaliningrad), 21-th century



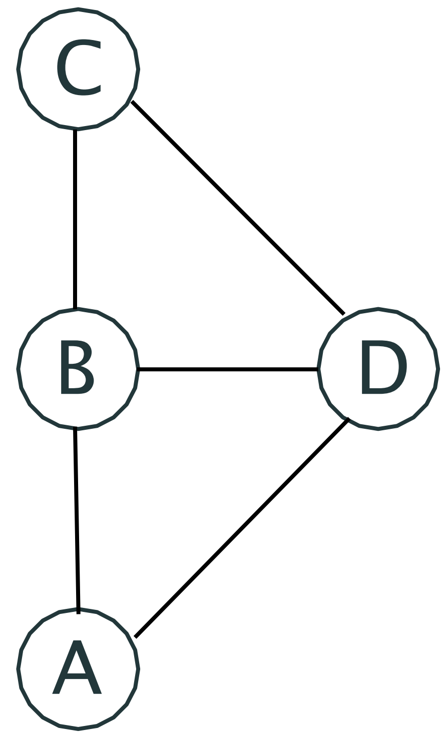
Five Bridges of Kaliningrad

B and D have odd degree

If there exists an Eulerian path, B and D must be START and FINISH



Königsberg (Kaliningrad), 21-th century



Eulerian Cycle

An Eulerian cycle (circuit) visits every edge **exactly once** and returns to the starting vertex.

- A cycle must have the same starting and ending vertex
- While in a path the starting and ending node should not necessarily be the same (but they might be the same). So the cycle is a special case of a path.

Criteria for Eulerian Cycle (Path)

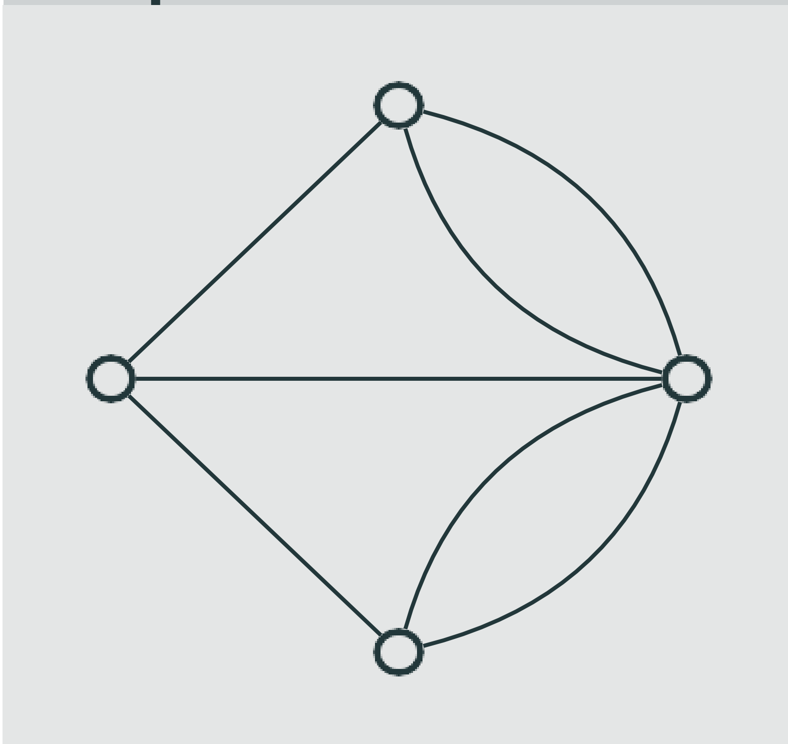
Theorem

A **connected** *undirected* graph contains an Eulerian cycle, **if and only if** the degree of every node is even.

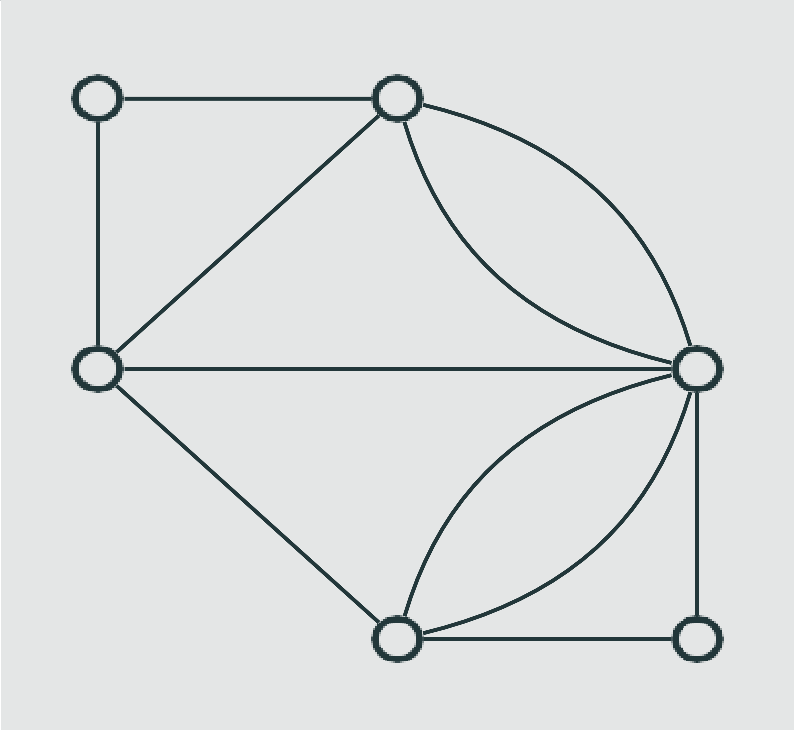
Note: every cycle is also a path, so if we have an Eulerian cycle, we also have an Eulerian path

But if we only want a path which is not a cycle, then exactly 2 vertices (namely start and end) are allowed to have odd degrees.

Graph A



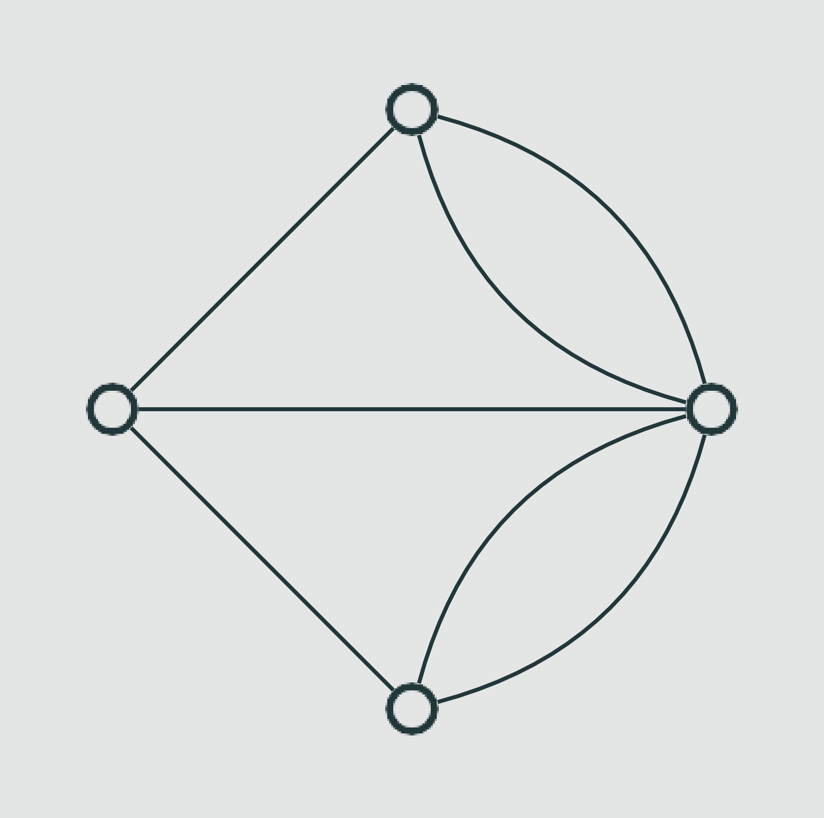
Graph B



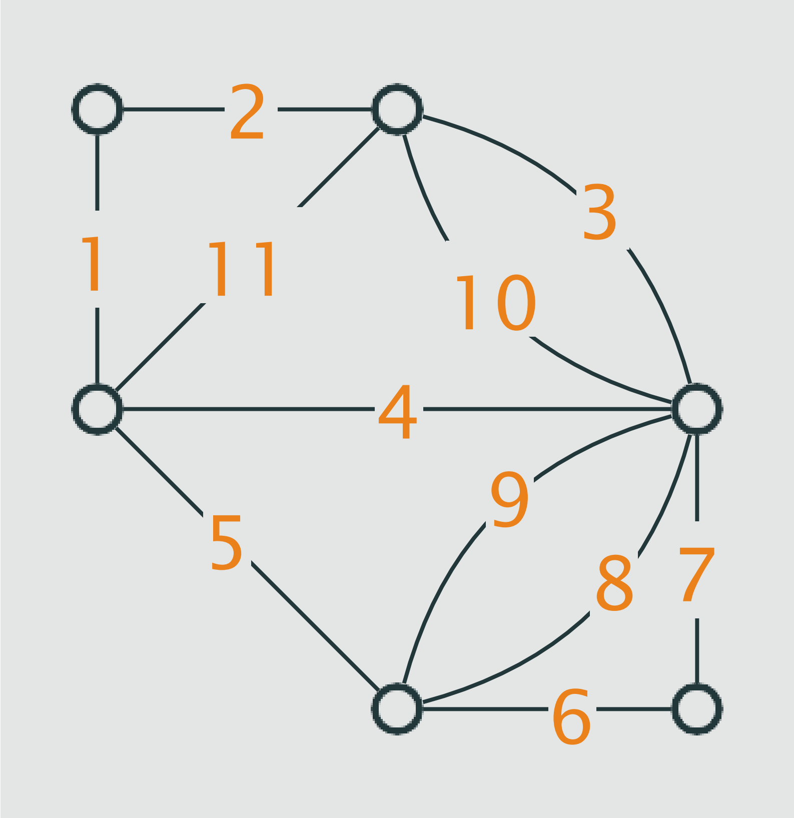
Which graph is an Eulerian graph (contains Eulerian cycle)?

A. Graph A
B. Graph B
C. Both A and B
D. Neither A nor B

Non-Eulerian graph

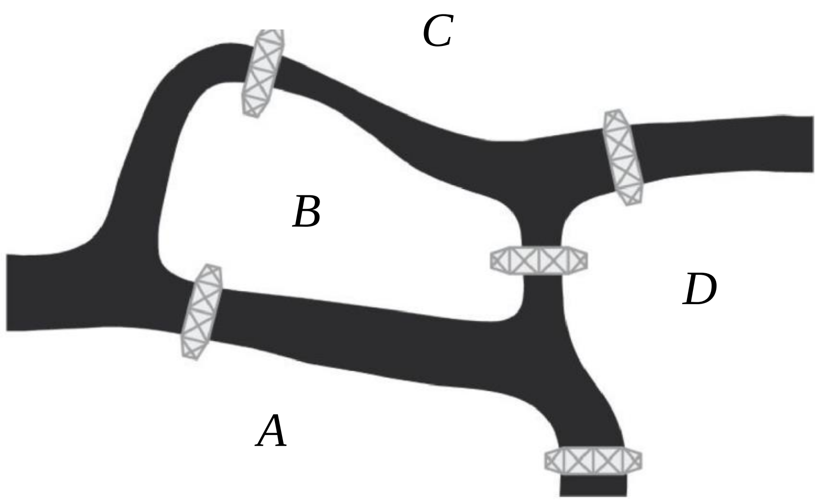


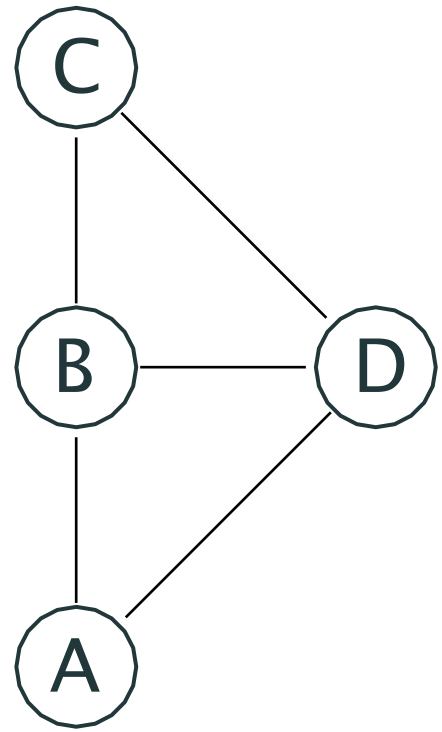
Eulerian graph



Eulerian path (cycle)

Algorithm for finding Eulerian Cycle (Path)





The theorem about the existence of an Eulerian cycle can be transformed into an efficient algorithm for constructing it

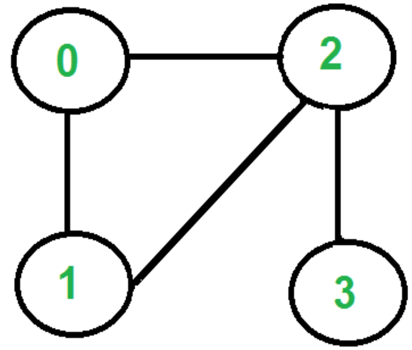
Eulerian Path Algorithm

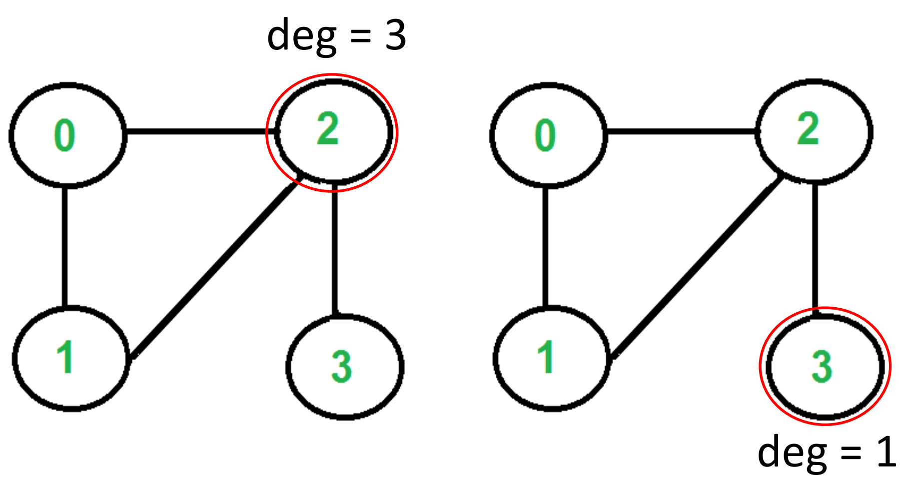
- If there are no odd-degree vertices, start anywhere
  If there are 2 odd-degree vertices, start at one of them.

- Out of the current vertex follow any edge
  - If you have a choice between a *bridge* and a *non-bridge*, always **choose the non-bridge**: “don’t burn bridges“ so that you can come back to a vertex and traverse remaining edges
  - Remove each followed edge (or mark as processed)

- Stop when you run out of edges

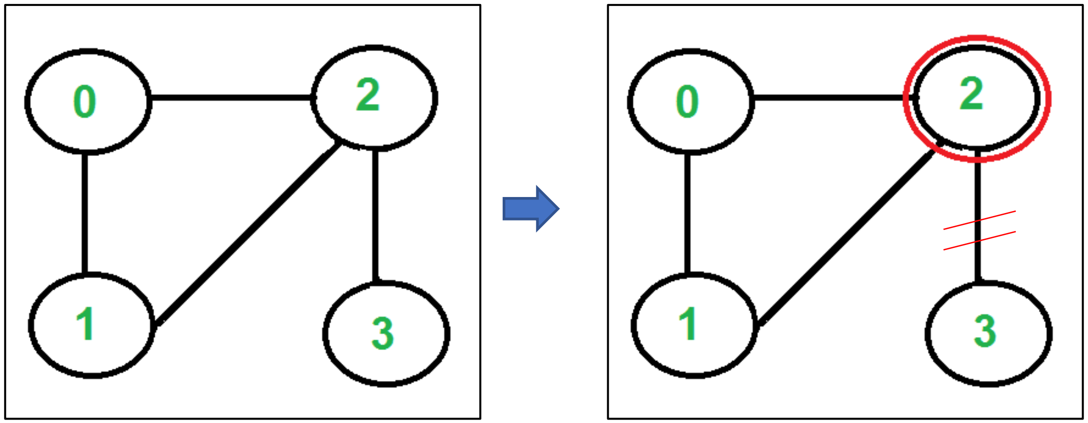
Example





Two vertices with odd degree – choose any of them to start

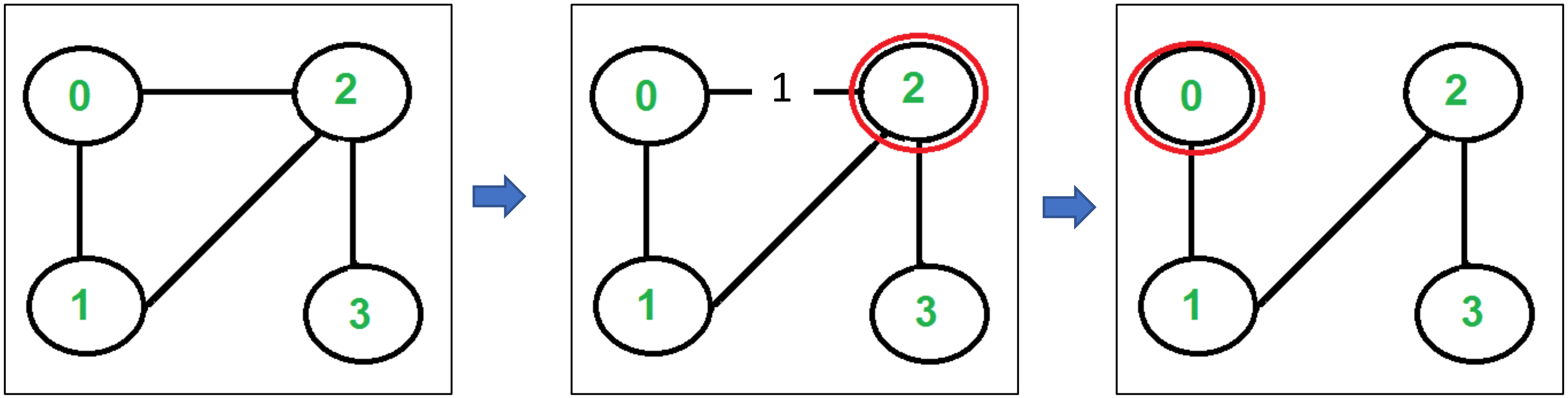
Example: where to go first?



Do not go there:
(2,3) is a bridge

Eulerian Path:

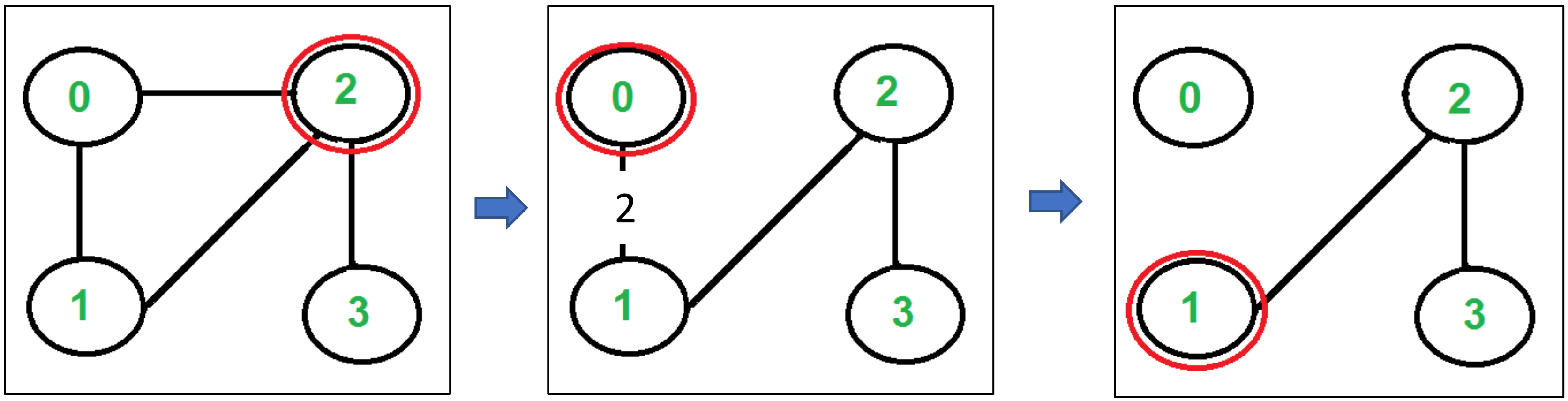
Example: step 1



Move along (2,0) and then delete edge (2,0)

Eulerian Path: (2,0)

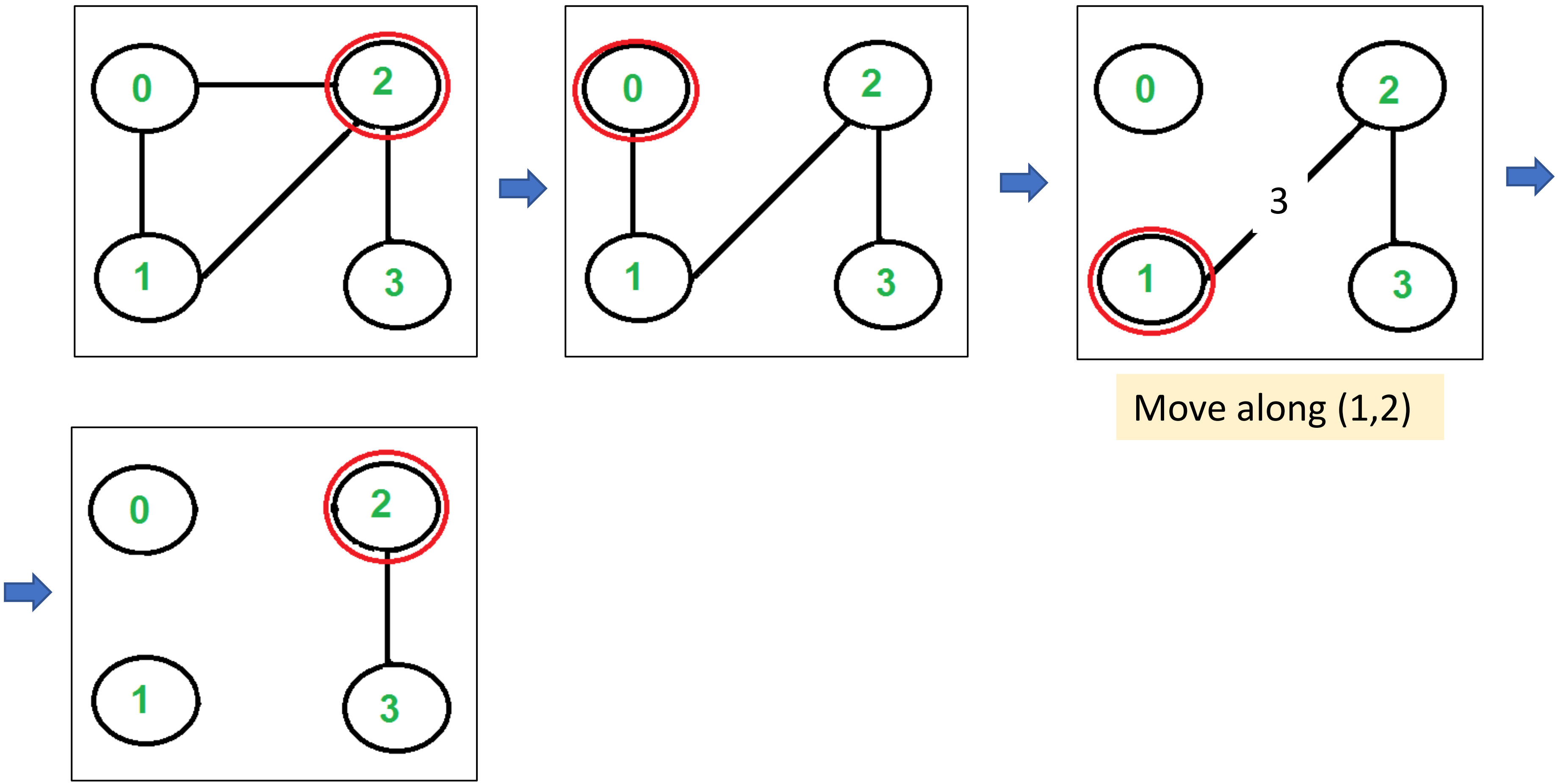
Example: step 2



Move along (0,1) and then delete edge (0,1)

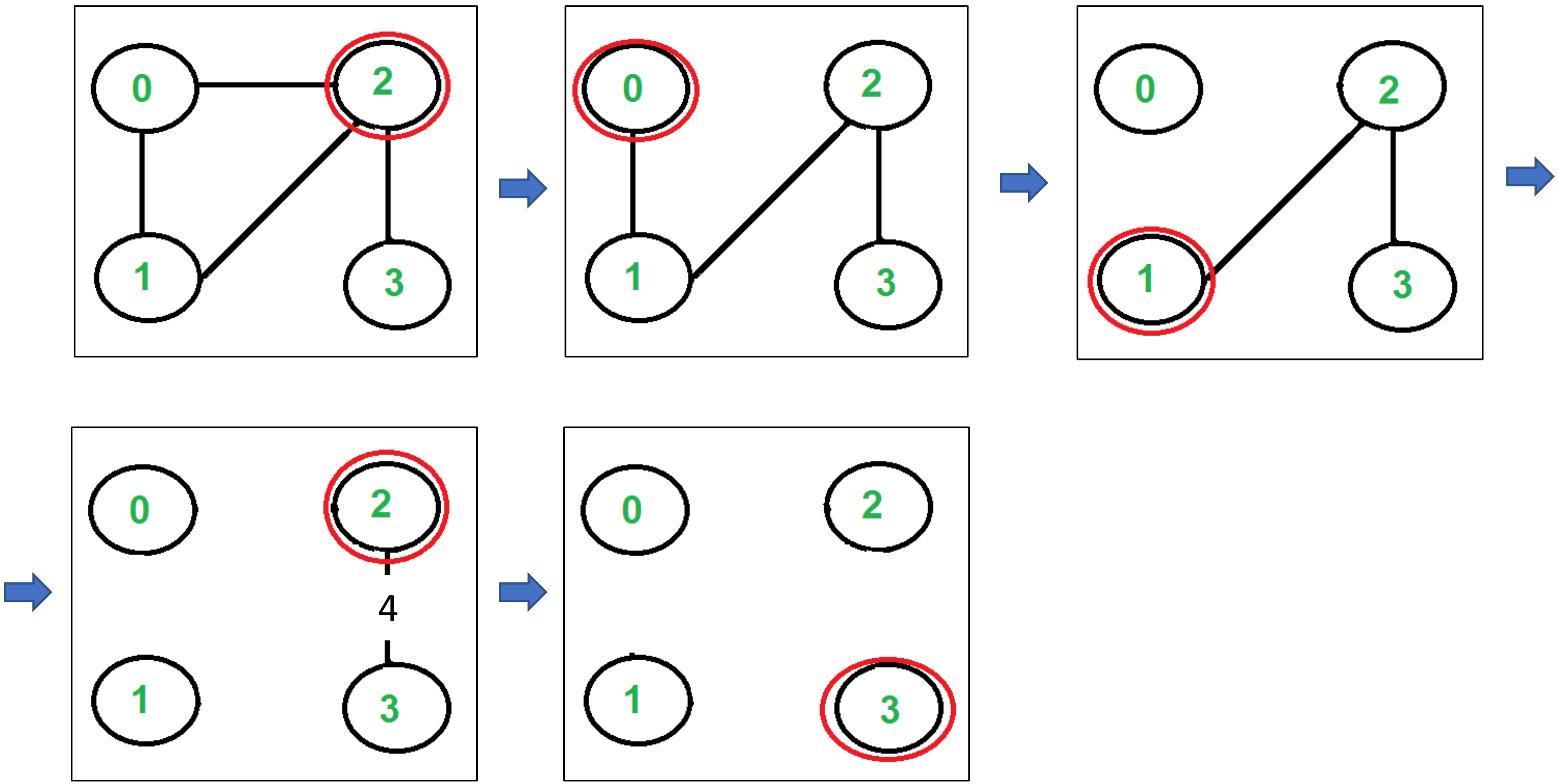
Eulerian Path: (2,0), (0,1)

Example: step 3



Eulerian Path: (2,0), (0,1), (1,2)

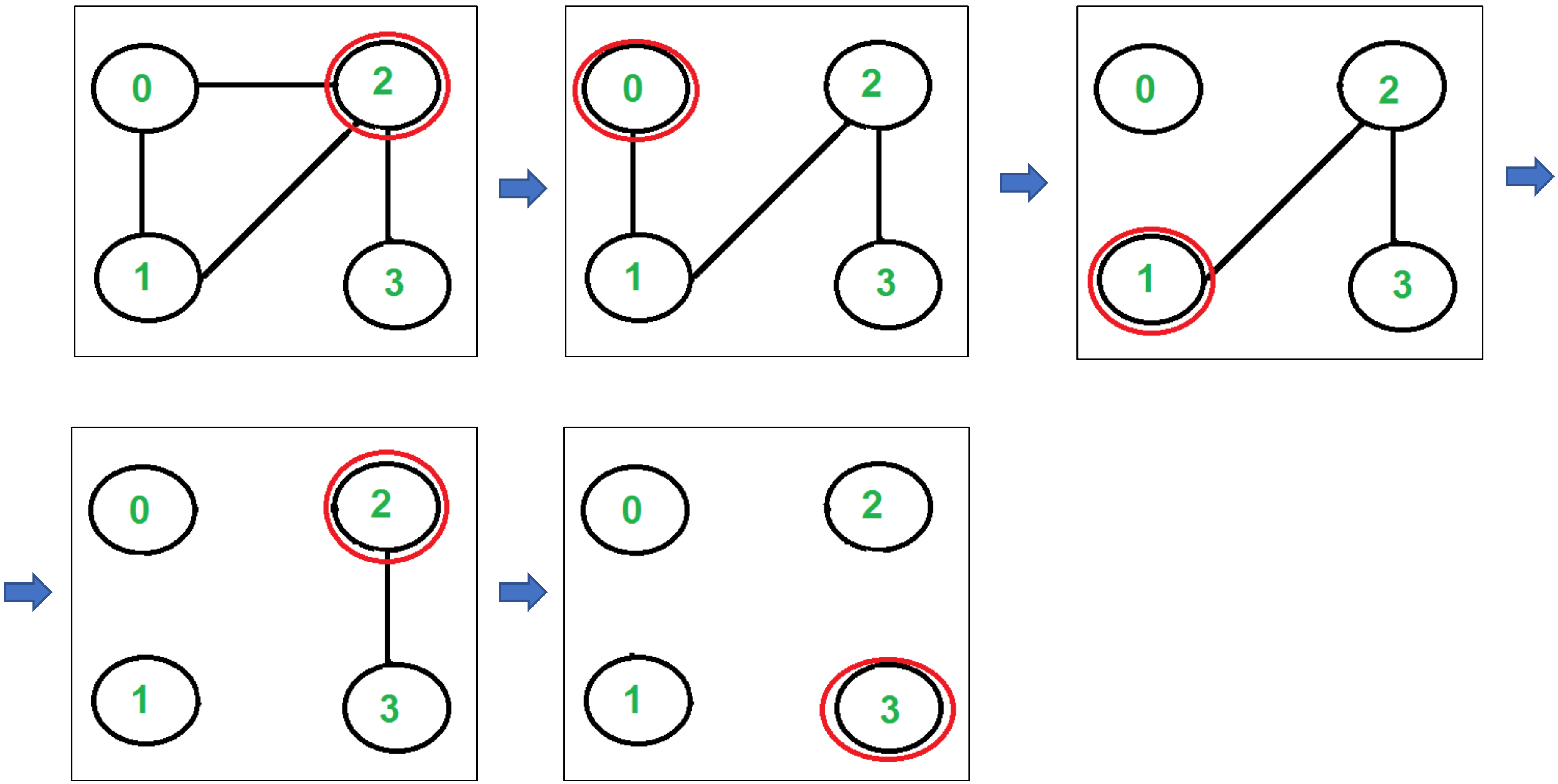
Example: step 4



Move along (2,3)

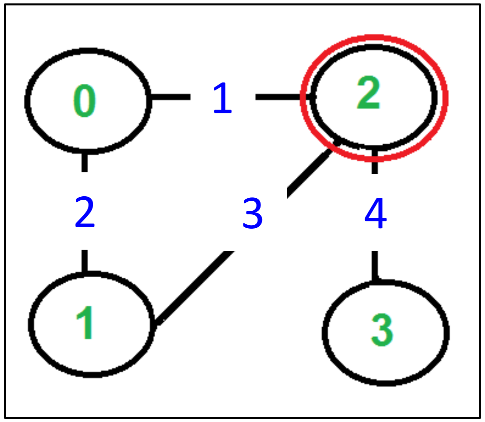
Eulerian Path: (2,0), (0,1), (1,2), (2,3)

Example: the end



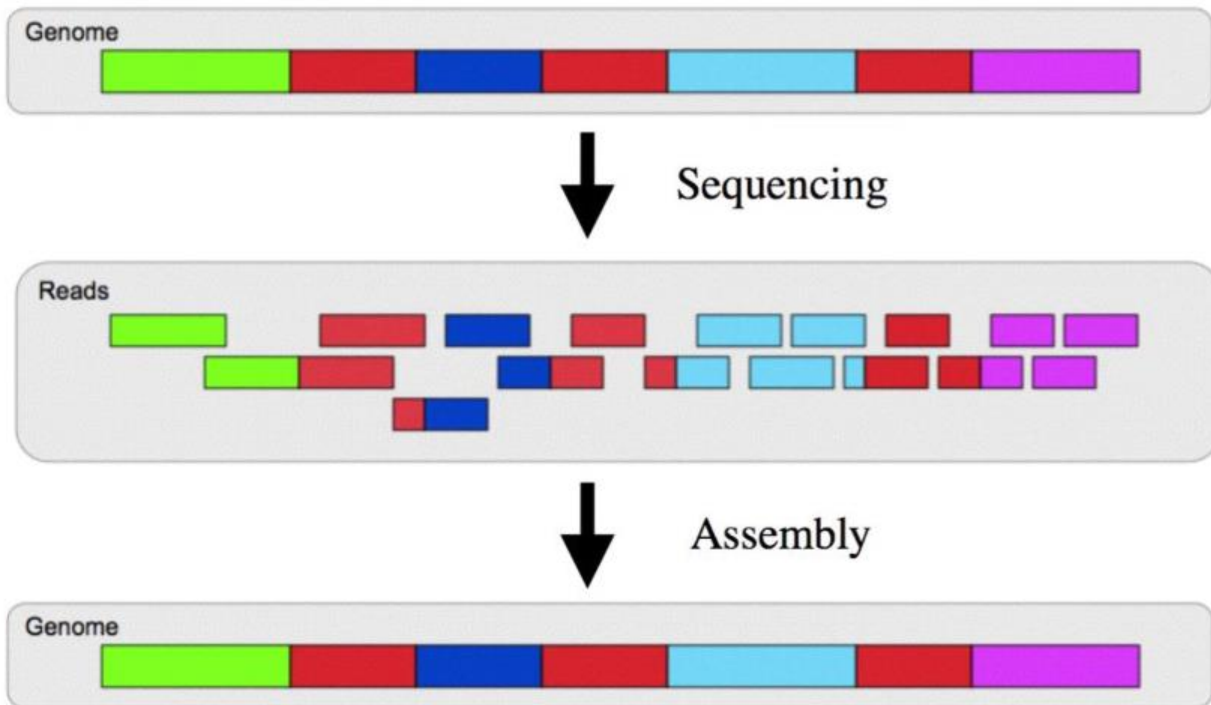
Eulerian Path: (2,0), (0,1), (1,2), (2,3)

Example: the end



Eulerian Path: (2,0), (0,1), (1,2), (2,3)

Genome Assembly problem



Genome Assembly problem: toy example

Find a string whose all substrings of length 3 are:

AGC, ATC, CAG, CAT, CCA, GCA, TCA, TCC.

How is this related to paths in graphs?..

All Substrings of Length 3

```
DISCRETE
DIS
 ISC
  SCR
   CRE
    RET
     ETE
```

Every two neighbor 3-substrings have
a common part of length 2, called an overlap

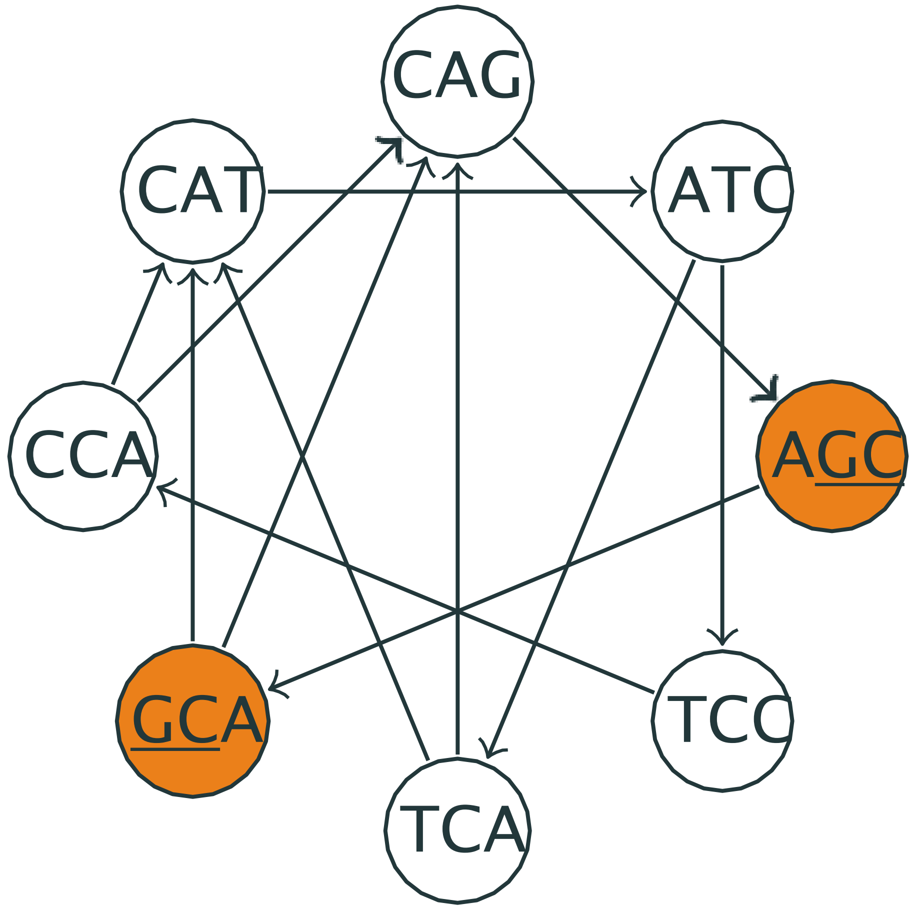
Finding a Permutation

- Goal: Find a string whose all substrings of length 3 are AGC, ATC, CAG, CAT, CCA, GCA, TCA, TCC

- Hence, we need to order these 3-substrings such that the overlap between any two consecutive substrings is equal to 2

Overlap Graph

AGC
ATC
CAG
CAT
CCA
GCA
TCA
TCC



There is an edge from s_1 to s_2 if $s_1[2:3]=s_2[1:2]$

Different approach
(De Bruijn; Pevzner, Tang, Waterman)

State-of-the-art genome assemblers

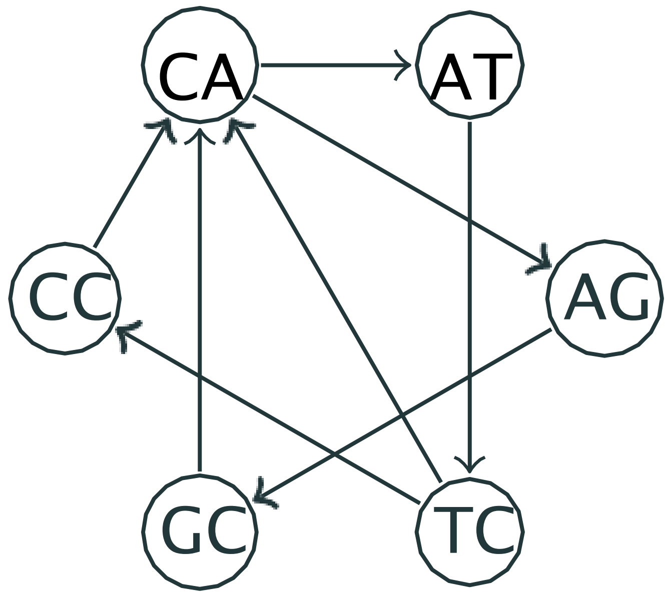
- In the overlap graph, each node corresponds to the input substring
- Let's instead represent each edge by the same substring, broken into 2 nodes (overlaps):

  E.g., represent the string CAT as an edge

  CA $\rightarrow$ AT

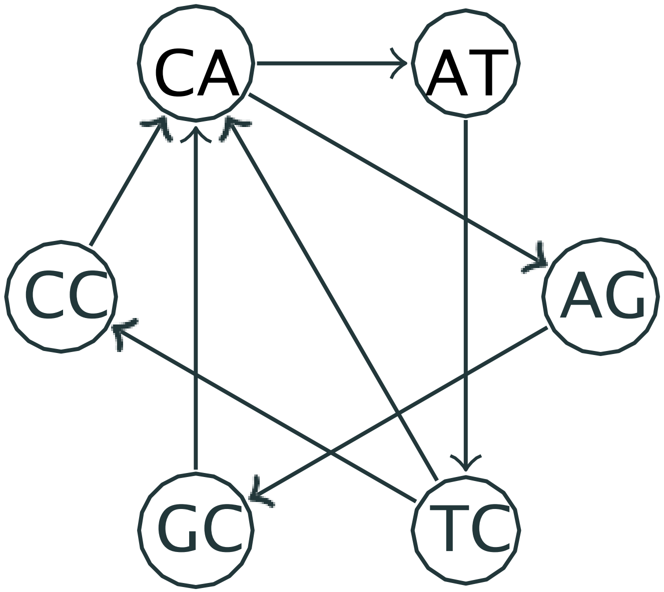
De Bruijn Graph

AGC, ATC, CAG, CAT, CCA, GCA,
TCA, TCC



De Bruijn Graph

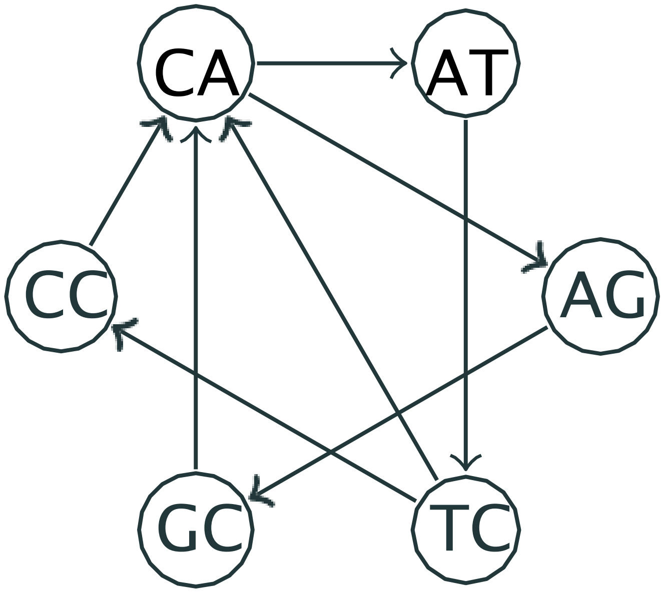
AGC, ATC, CAG, CAT, CCA, GCA,
TCA, TCC



now, we need to find an order of edges

De Bruijn Graph

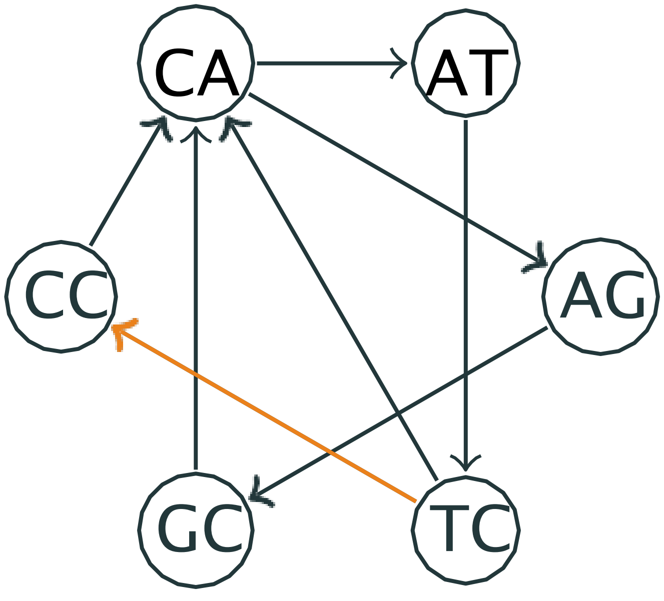
AGC, ATC, CAG, CAT, CCA, GCA,
TCA, TCC



that is, an Eulerian path

De Bruijn Graph

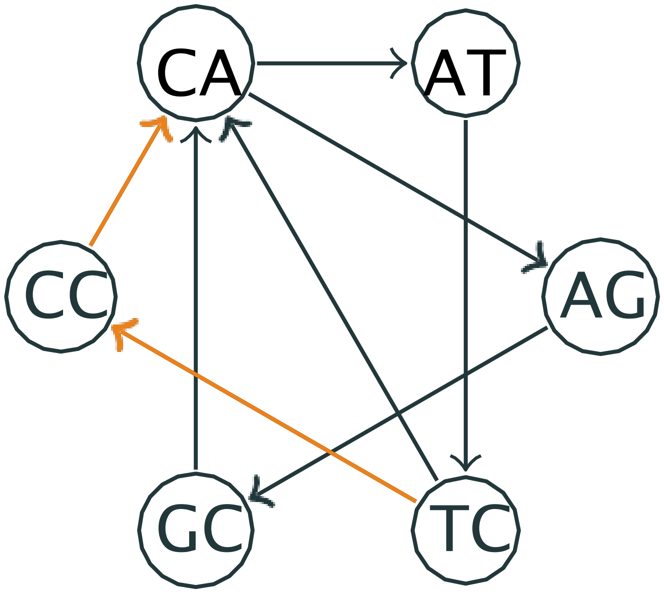
AGC, ATC, CAG, CAT, CCA, GCA, TCA, TCC



TCC

De Bruijn Graph

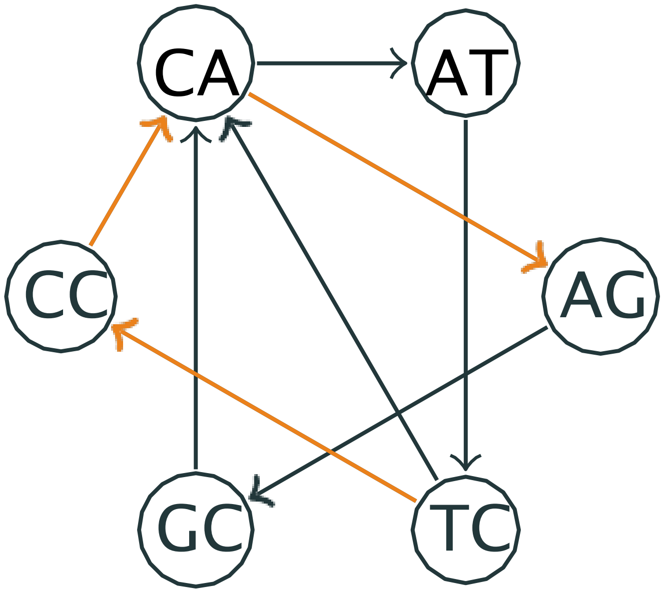
AGC, ATC, CAG, CAT, CCA, GCA,
TCA, TCC



TCCA

De Bruijn Graph

AGC, ATC, CAG, CAT, CCA, GCA,
TCA, TCC



TCCAG

Group activity DeBruijn Graph

- Imagine that you are given a **large** set of 3-letter strings which represent all possible different substrings of the large "genome" string:

  him, eno, ome, chi, nom, mpg, pge, gen, imp

- Recover the whole "genome" sequence by building a graph model of the problem.

- Draw the graph and explain which algorithm you used on this model to recover the original "genome"