Day 8
Graph theory
& Traversal problem

Lecturer: Msc. Minh Tan Le

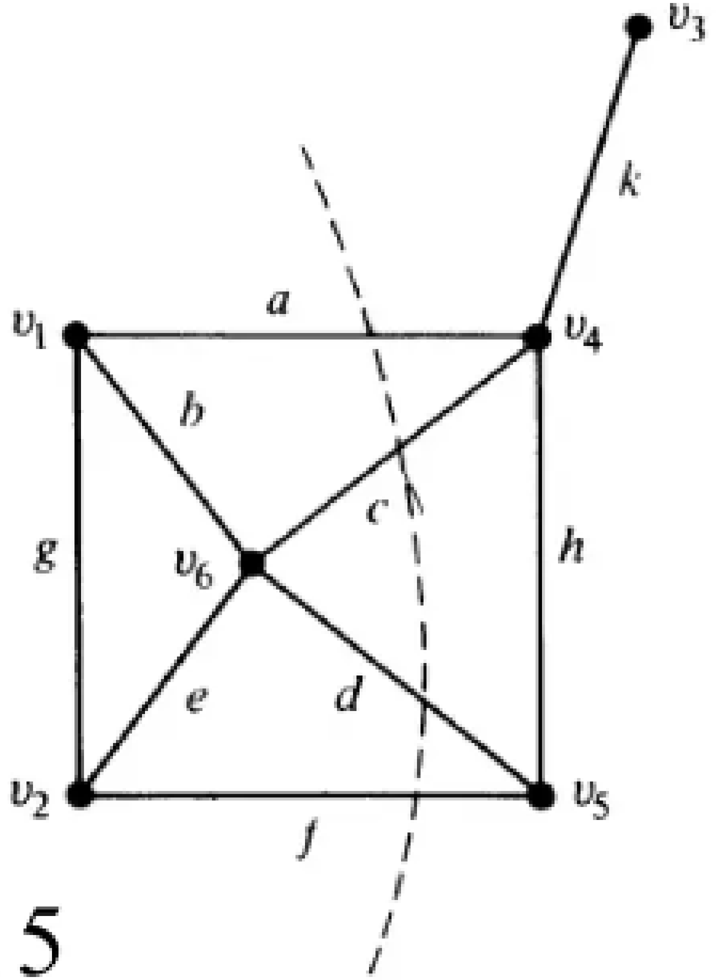
Part I. Concepts

Connectivity

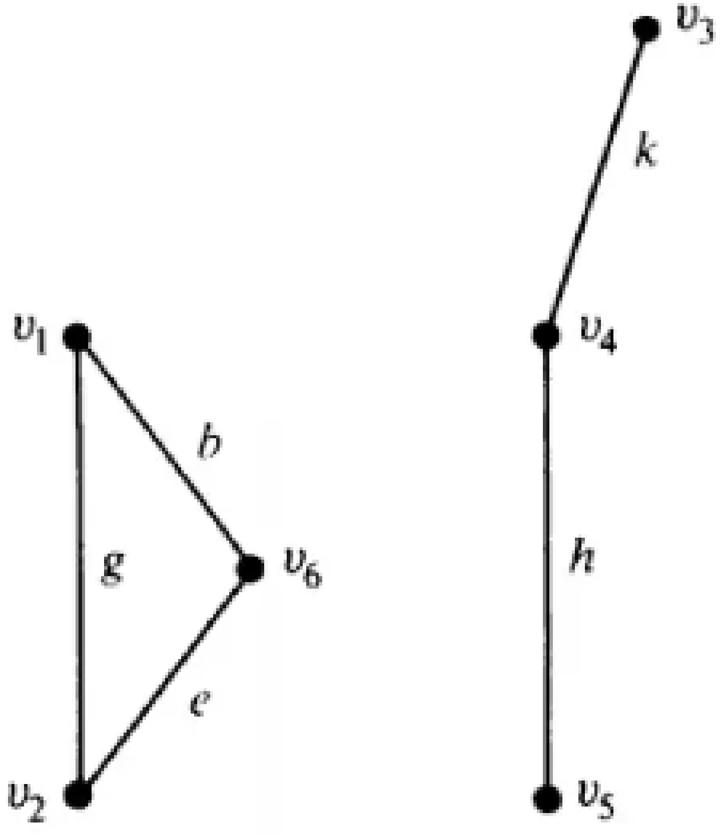
- **Connected graph (undirected)**: There exists an path between any 2 vertices.
  - **Strongly connected (directed)**: a can go to b and other way around.
  - **Weakly connected (directed)**: Replacing all arcs with undirected edges resulting a connected graph.
- **Disconnected graph**: Not connected graph.
  - **Components**: Connected subgraph that is isolated.

In undirected graph…

- **Cut-set (or cut)** is a **subset of edges** that when removed, a disconnected graph is created.
- **Cut point** is **a vertex** that when removed from connected graph resulting a disconnected graph.
- **Bridge** is **an edge** that when removed from connected graph resulting a disconnected graph.



Rank 5

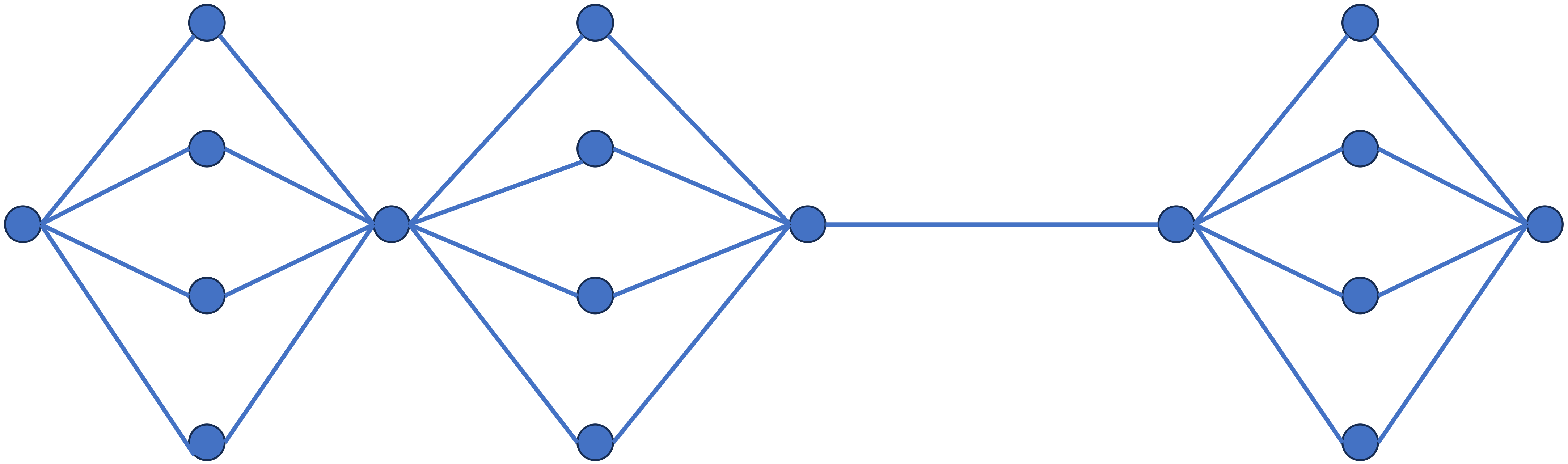


Rank 4

Subgraph

- Given $G = (V, E)$, a subgraph of G is $H(W, F)$ that:

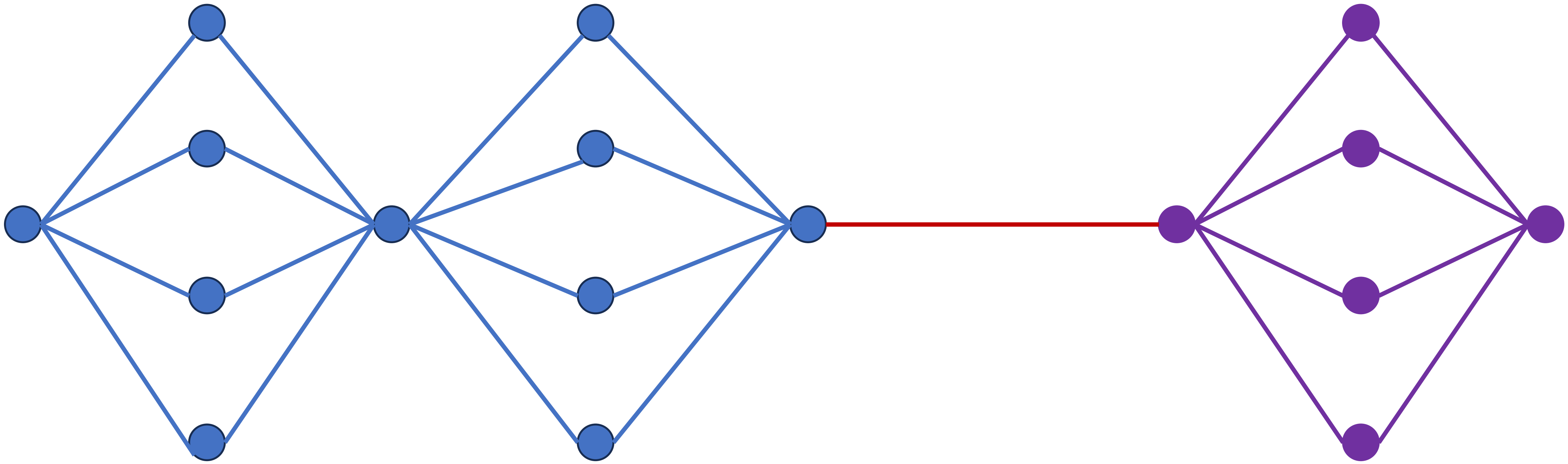
$$W \subset V, F \subset E$$



Subgraph

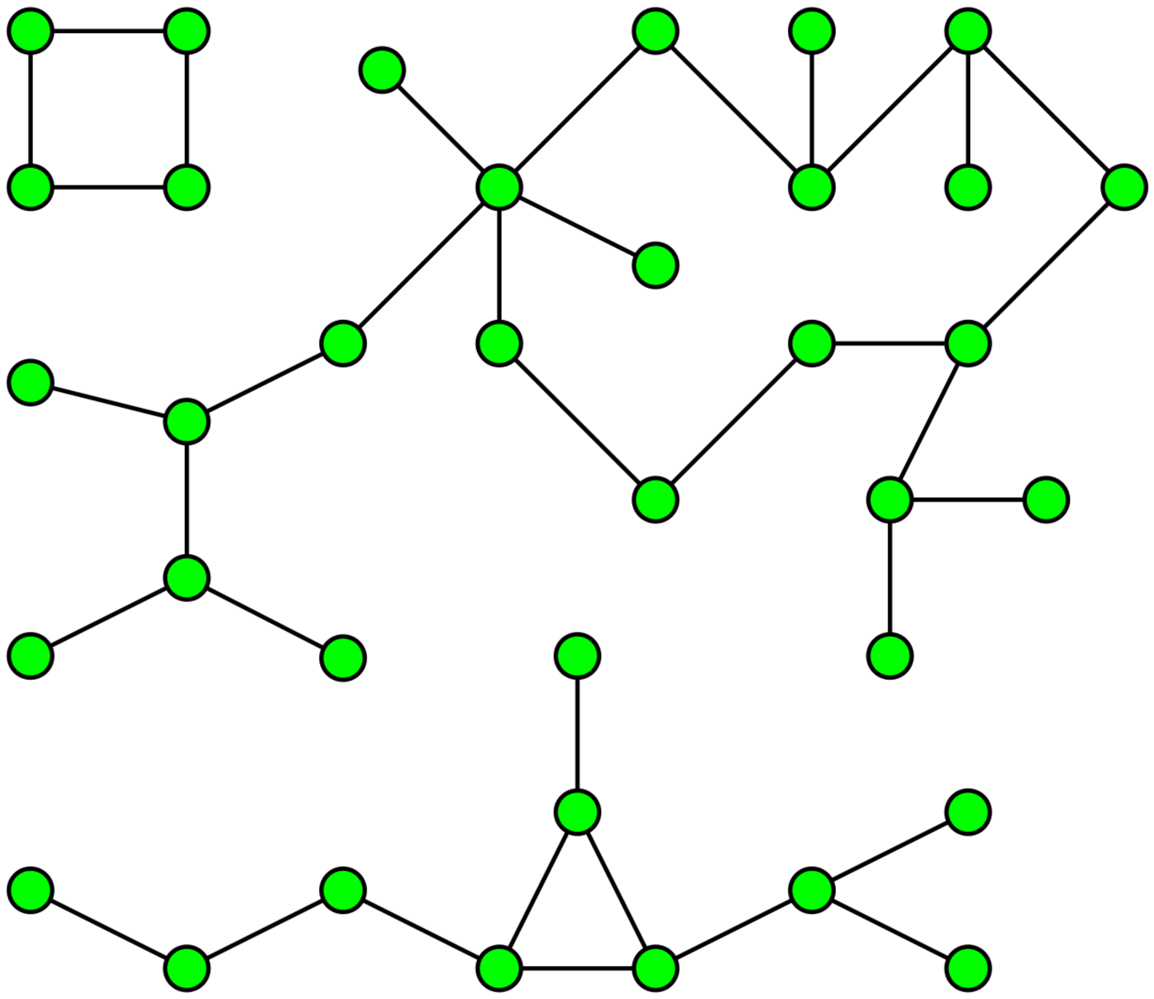
- Given $G = (V, E)$, a subgraph of G is $H(W, F)$ that:

$$W \subset V, F \subset E$$



Components

- Sometimes called connected components.
- They are subgraphs, but not part of any larger connected subgraph.

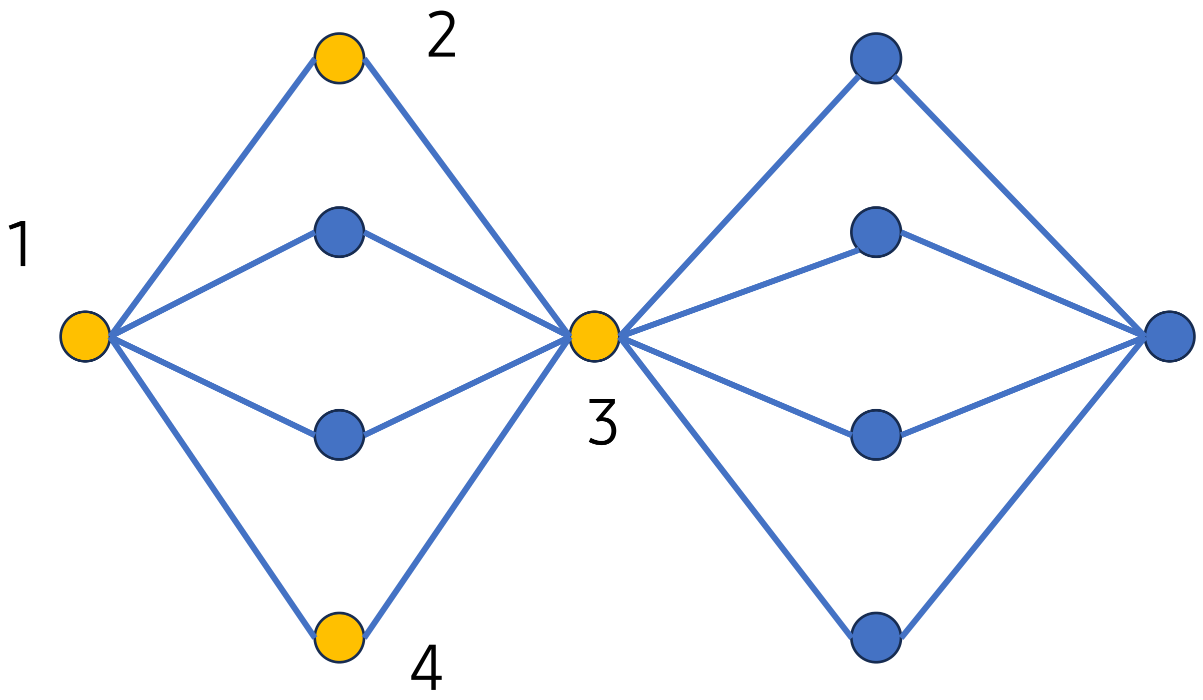


Part II: Traveling

- Passing each node so as they are all visited once.
  - Graph traversal: At least or exactly once.
  - Tree traversal: Exactly once.
- Applications: Searching, scanning, optimizing,...
- Concepts: Type of cycles, type of paths, type of graphs
- Types of graphs determined by rules

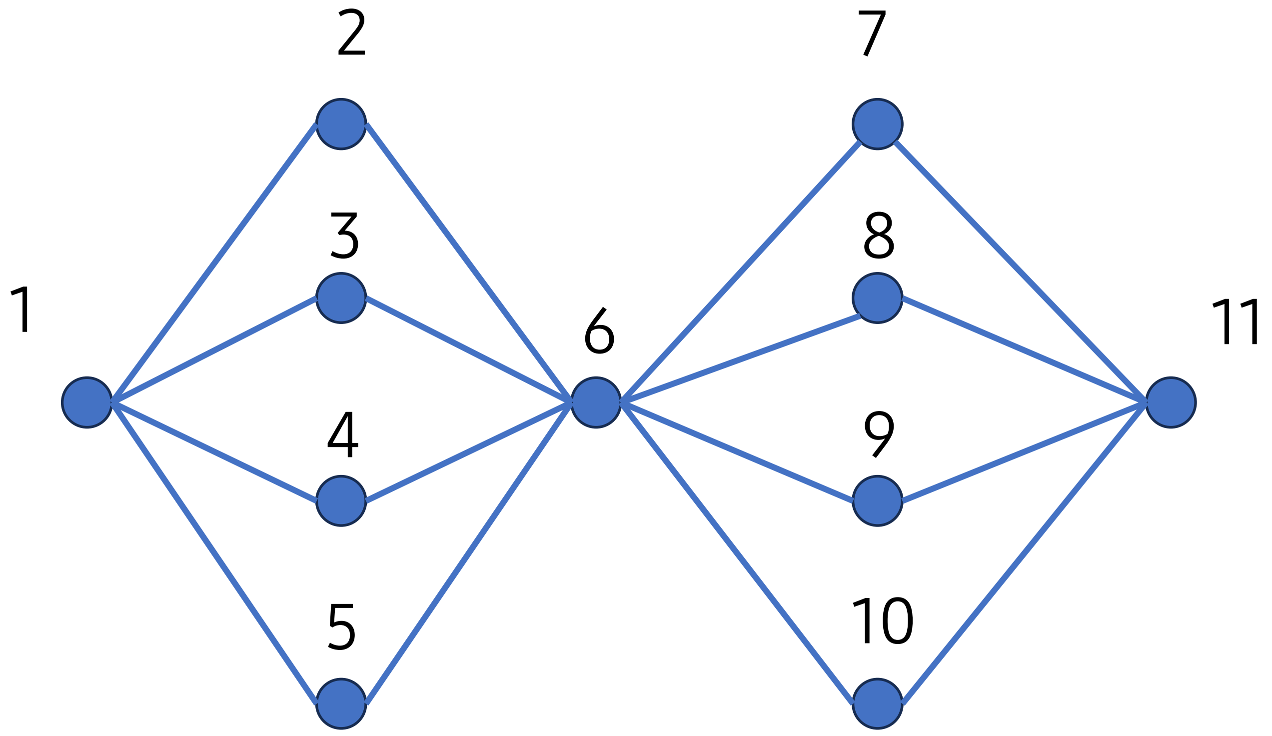
Cycle

- Definition: A path that has the same start & end node.
- Synonyms: Circuit, tour, closed walk.



Type of cycles

- Euler cycle: Must visit all edges, each is met exactly once.
  - Vertex revisiting is allowed
- Hamilton cycle: Must visit all vertices, each is met once.
  - No edge revisiting



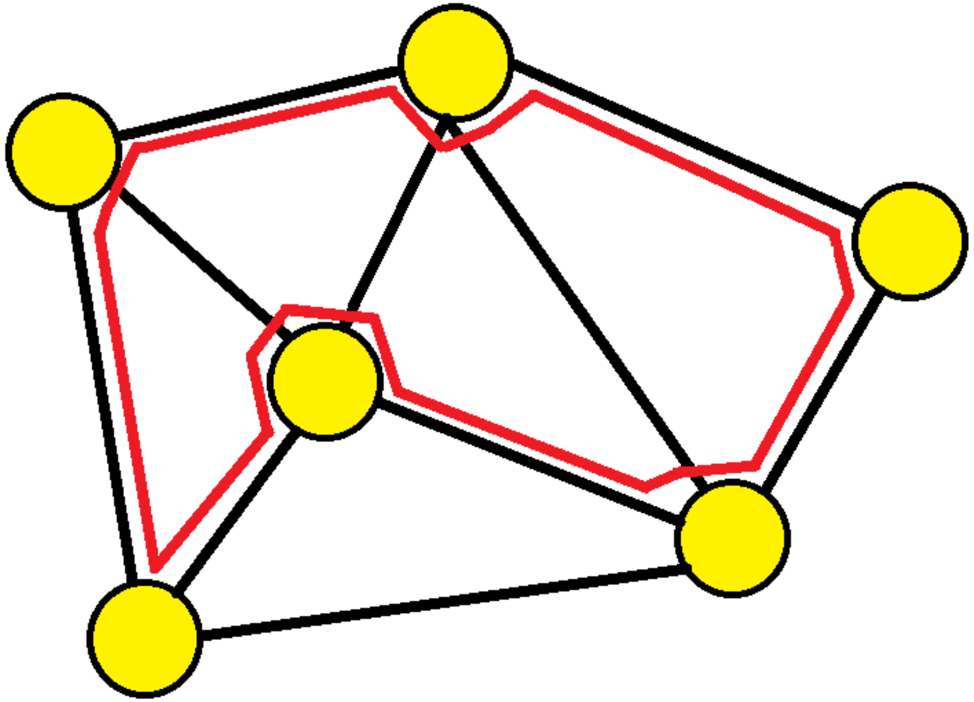
Paths

- Euler path: Must visit all **edges**, each is met exactly once.
  - Vertex revisiting is allowed
  - No need to end where we started
- Hamilton path: Must visit all **vertices**, each is met once.
  - No edge revisiting
  - No need to end where we started

Categorize Graphs by cycles/paths

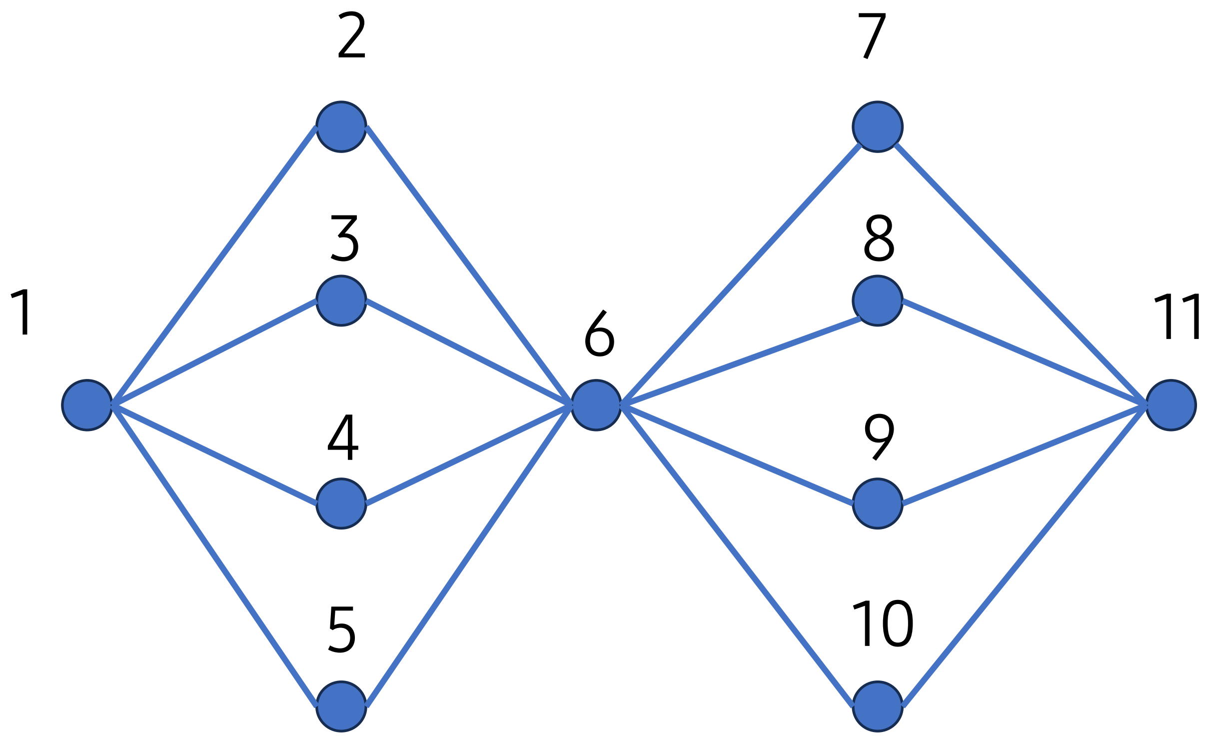
- Eulerian graph: A **connected graph** that has Euler **cycle**.
  - No odd-degree vertex.
- Semi-Eulerian graph: A **connected graph** that has Euler **path**, but **not a cycle**.
  - Undirected: Exactly 2 odd-degree vertices.
  - Directed: Exactly 2 vertices u, v satisfying: $\begin{cases} \deg^+(u) = \deg^-(u) + 1 \\ \deg^-(v) = \deg^+(v) + 1 \end{cases}$

- Hamiltonian graph: A graph that has Hamilton **cycle**.
- Semi-Hamiltonian graph: A graph that has Hamilton **path**, but **not a cycle**.



Part III. Determination

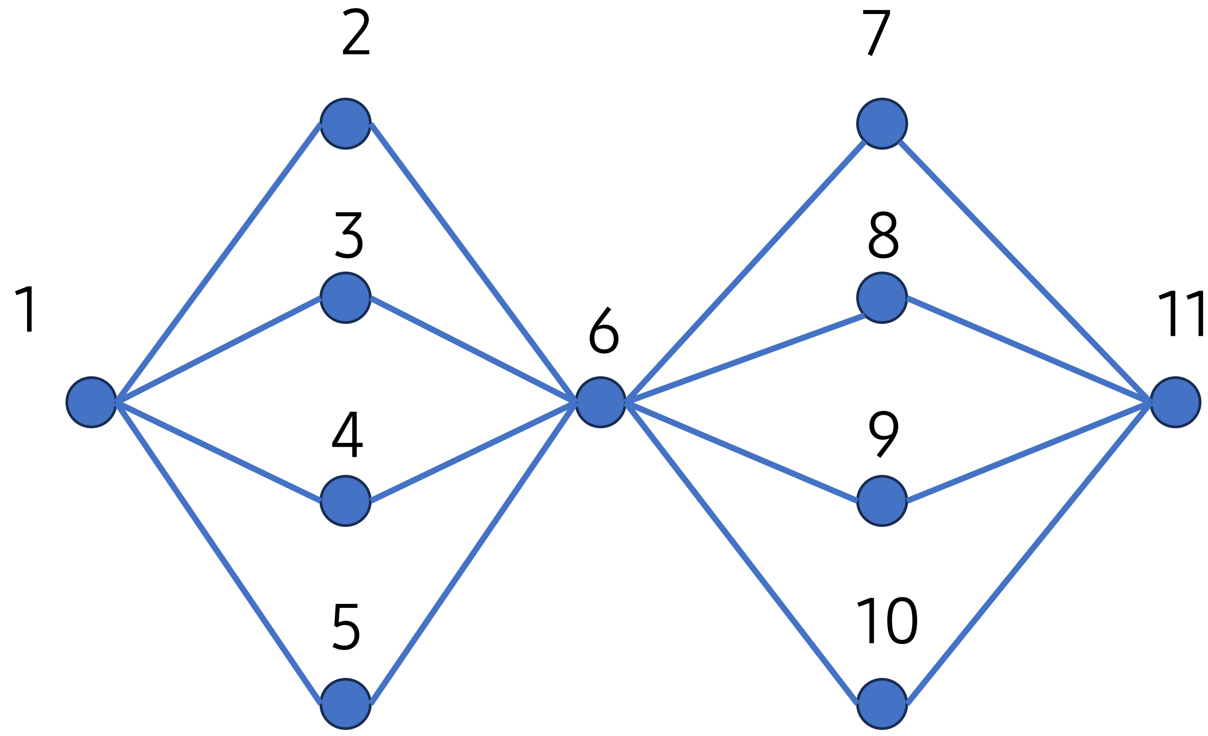
- How to test whether a graph is:
  - Eulerian graph
  - Semi-Eulerian graph
  - Hamilton graph
  - Semi-Hamilton graph



Is this Eulerian graph?

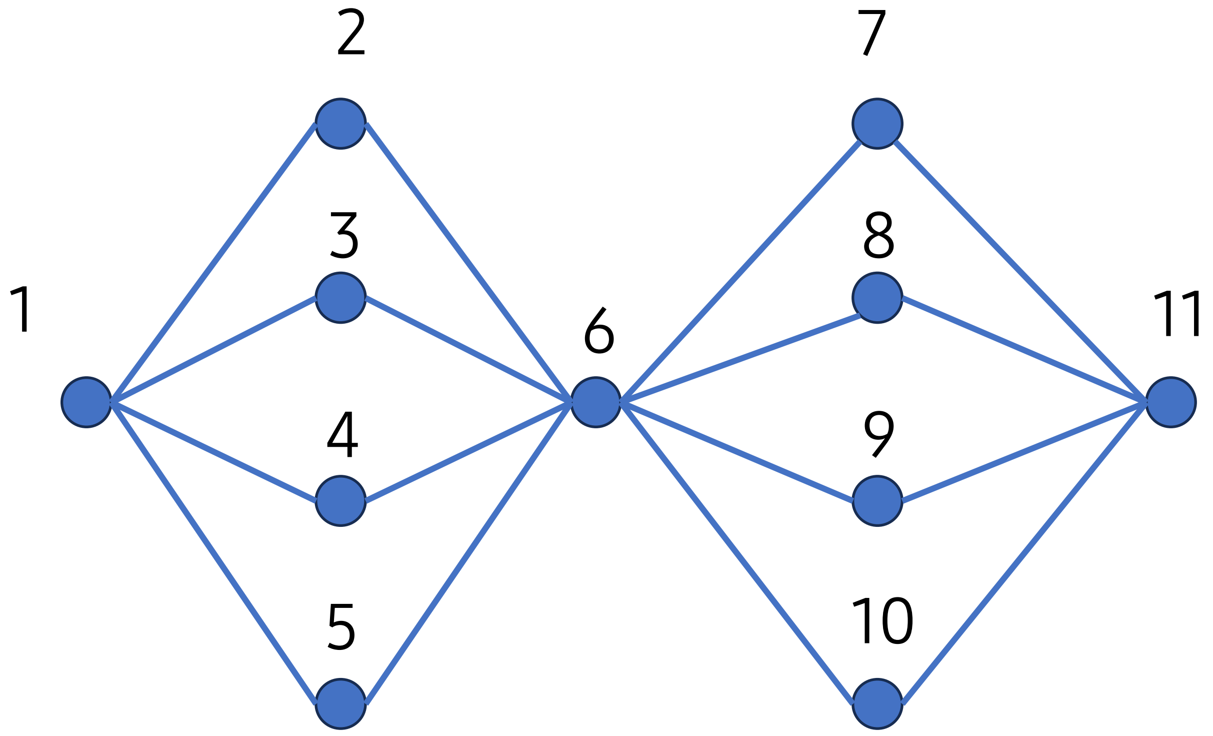
1. Is it connected?
2. Are all vertex degrees even?
3. Find the **cycle**. Remember:

   Euler is **edge**!



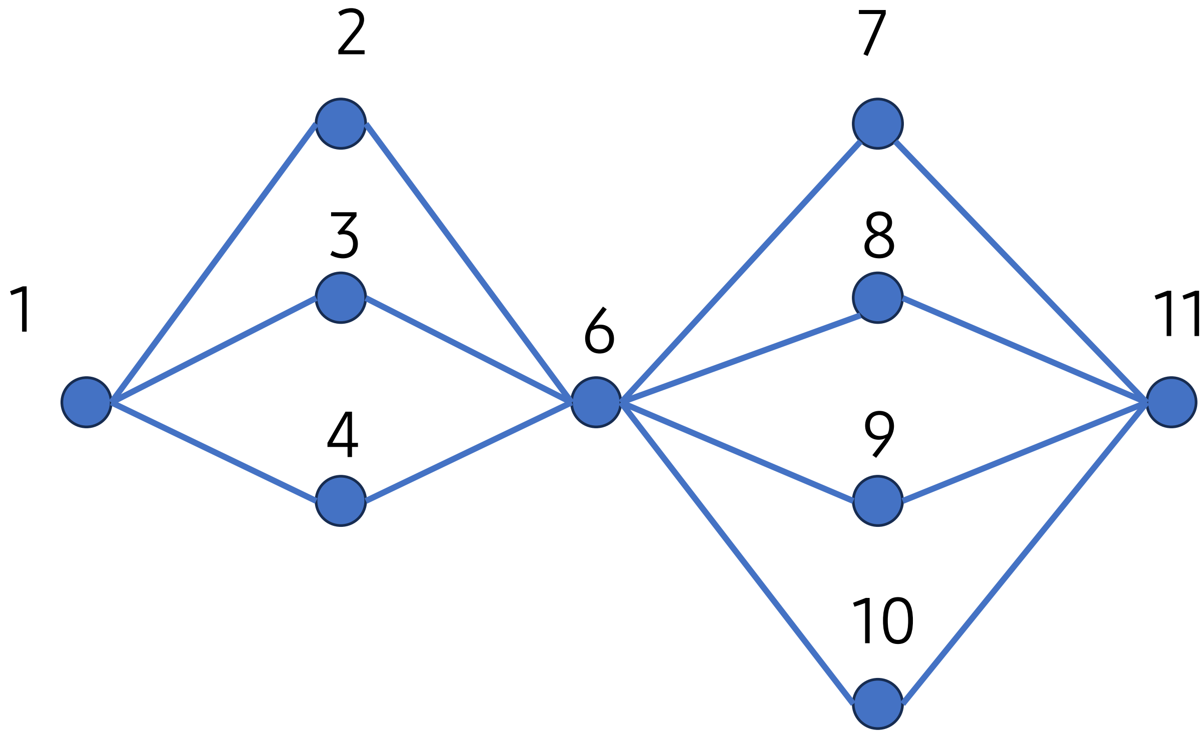
Semi-Eulerian graph

1. Is it connected?
2. Does it have 2 odd-degree vertices?
3. Find the **path**. Remember: Euler is **edge**!

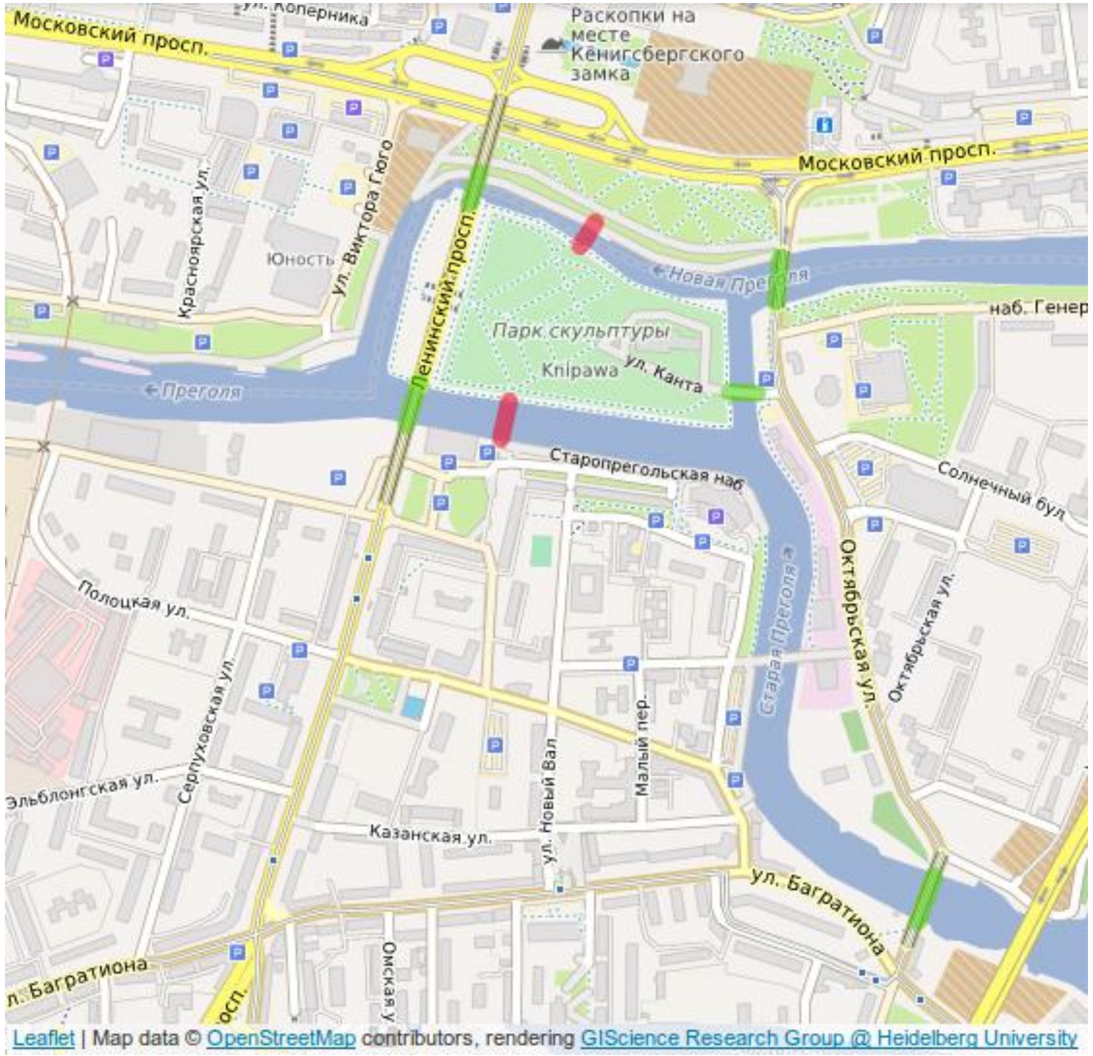


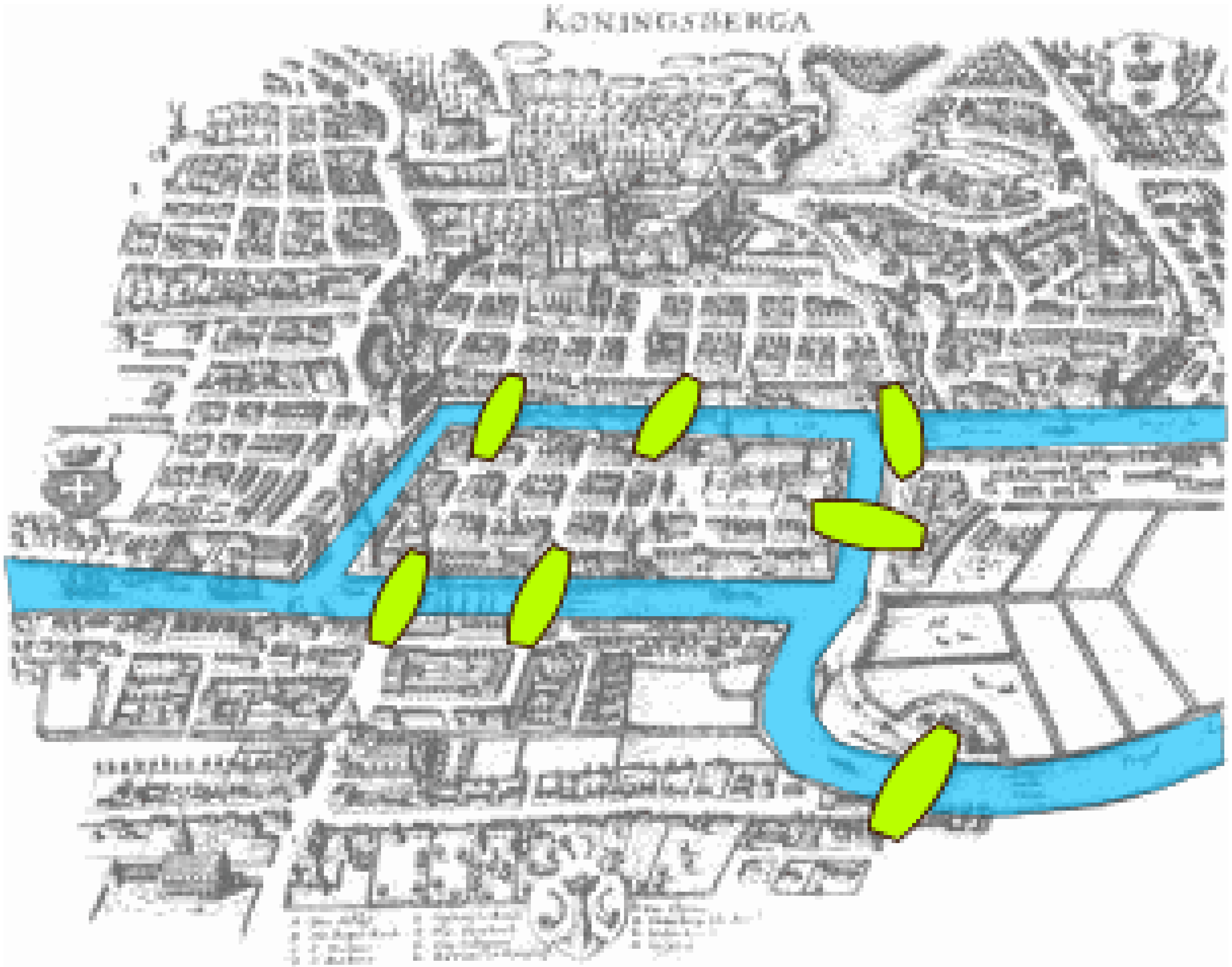
Semi-Eulerian graph

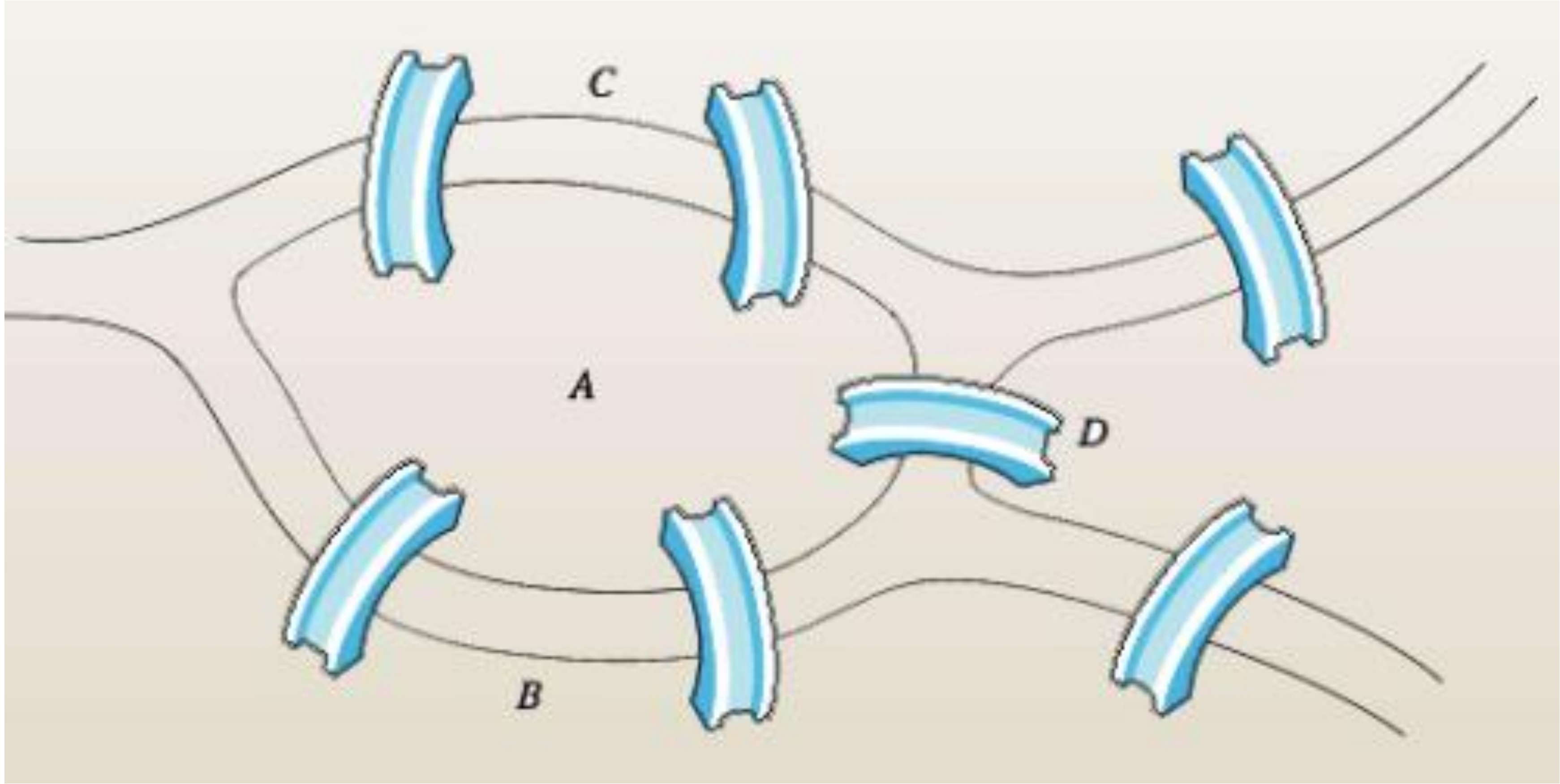
1. Is it connected?
2. Does it have 2 odd-degree vertices?
3. Find the **path**. Remember: Euler is **edge**!

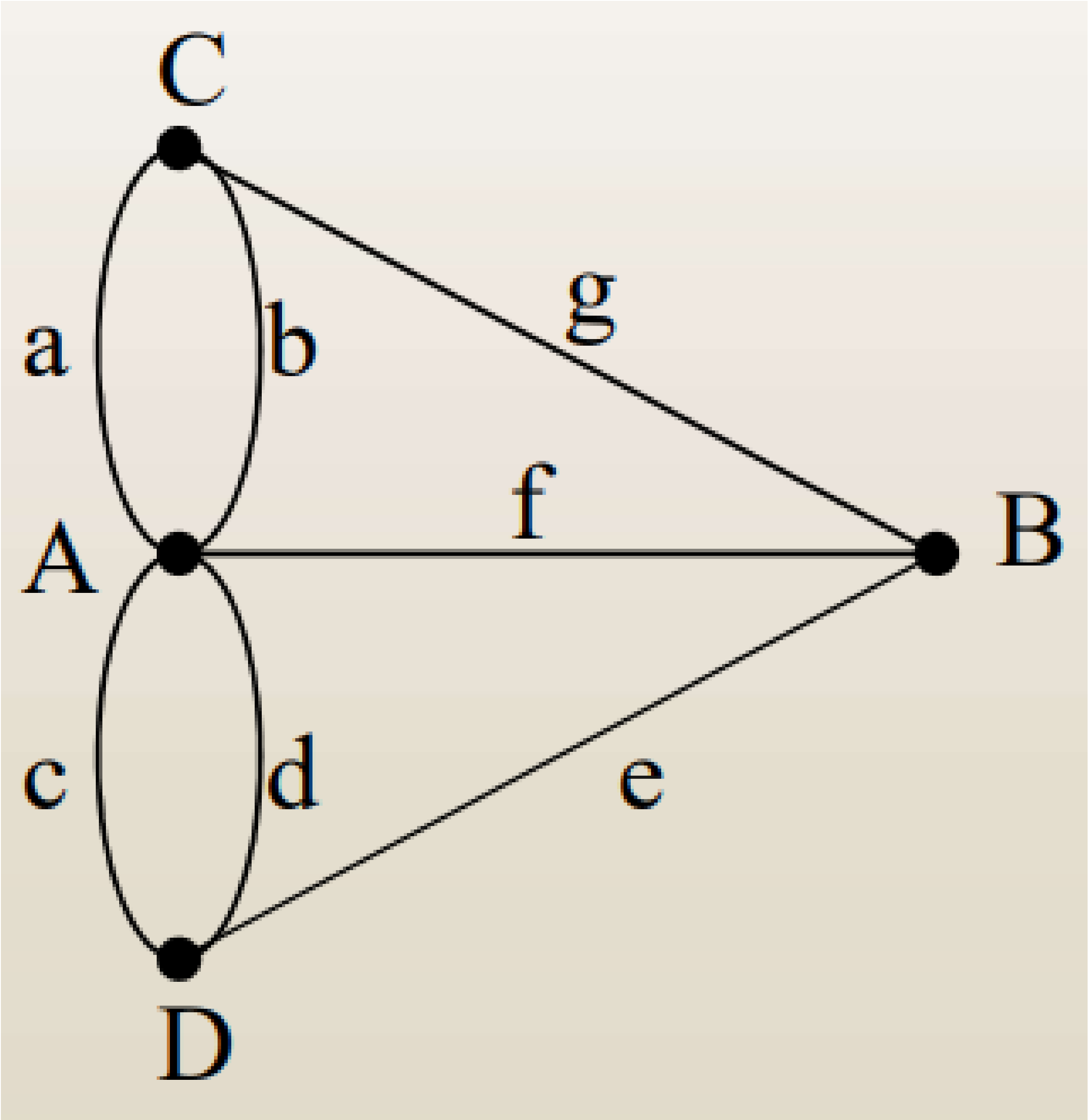


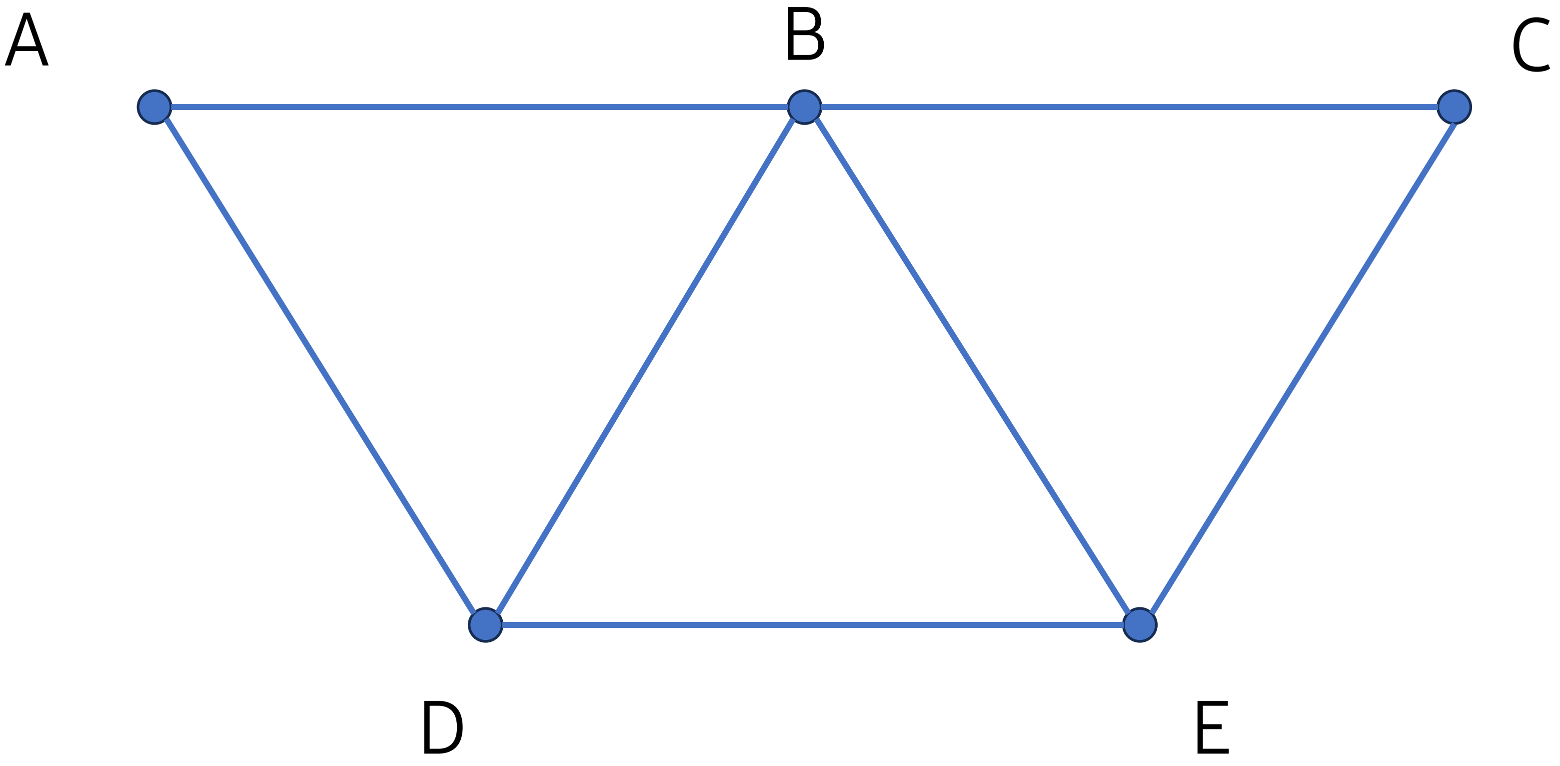
Seven Bridges of Königsberg

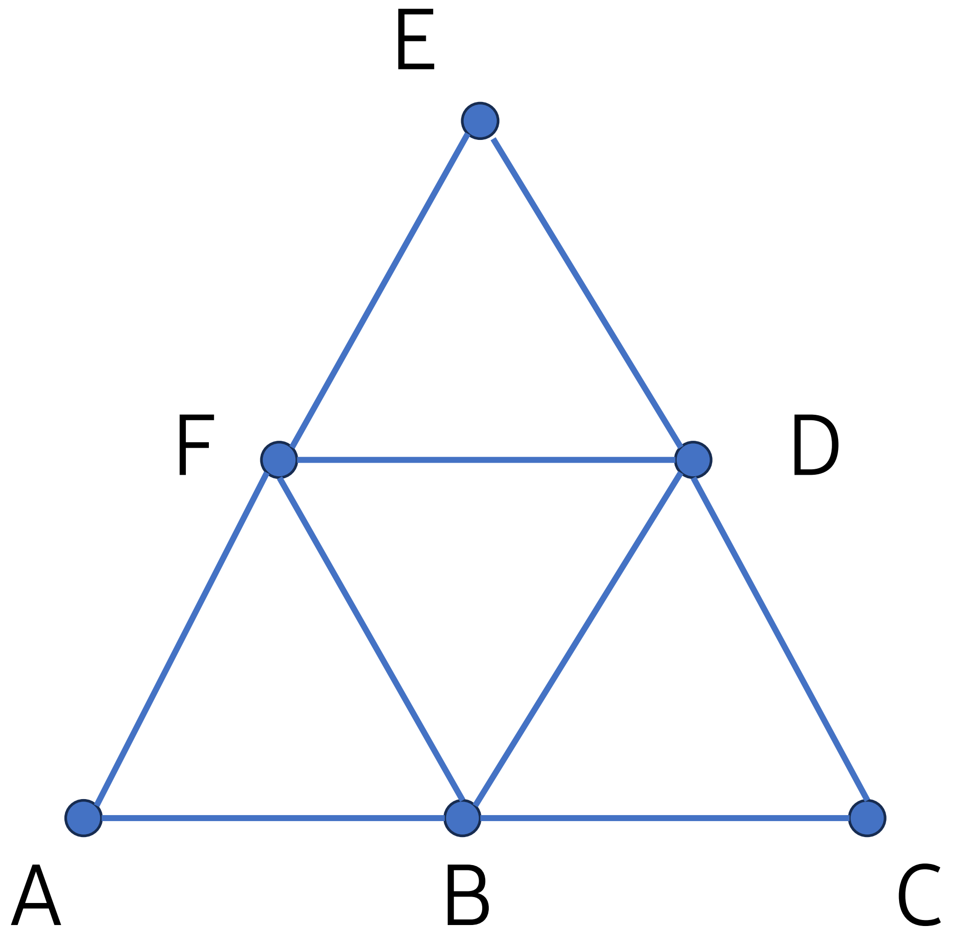



KONINGSBERGA


C
A
D
B

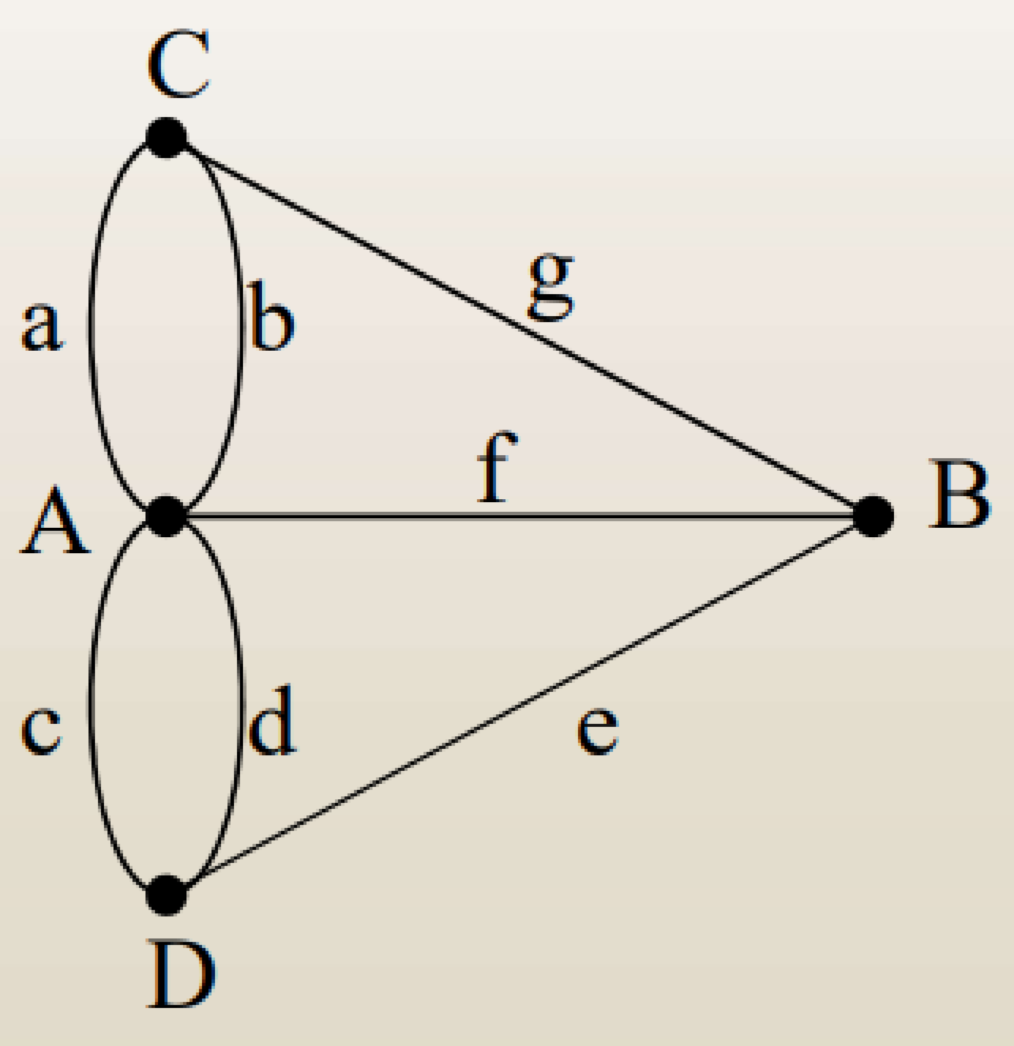

C
a
b
g
f
A
B
c
d
e
D


A
B
C
D
E



a. Is A -> B -> C -> D -> E -> F -> A an Euler circuit?
b. Does the graph have an Euler circuit?

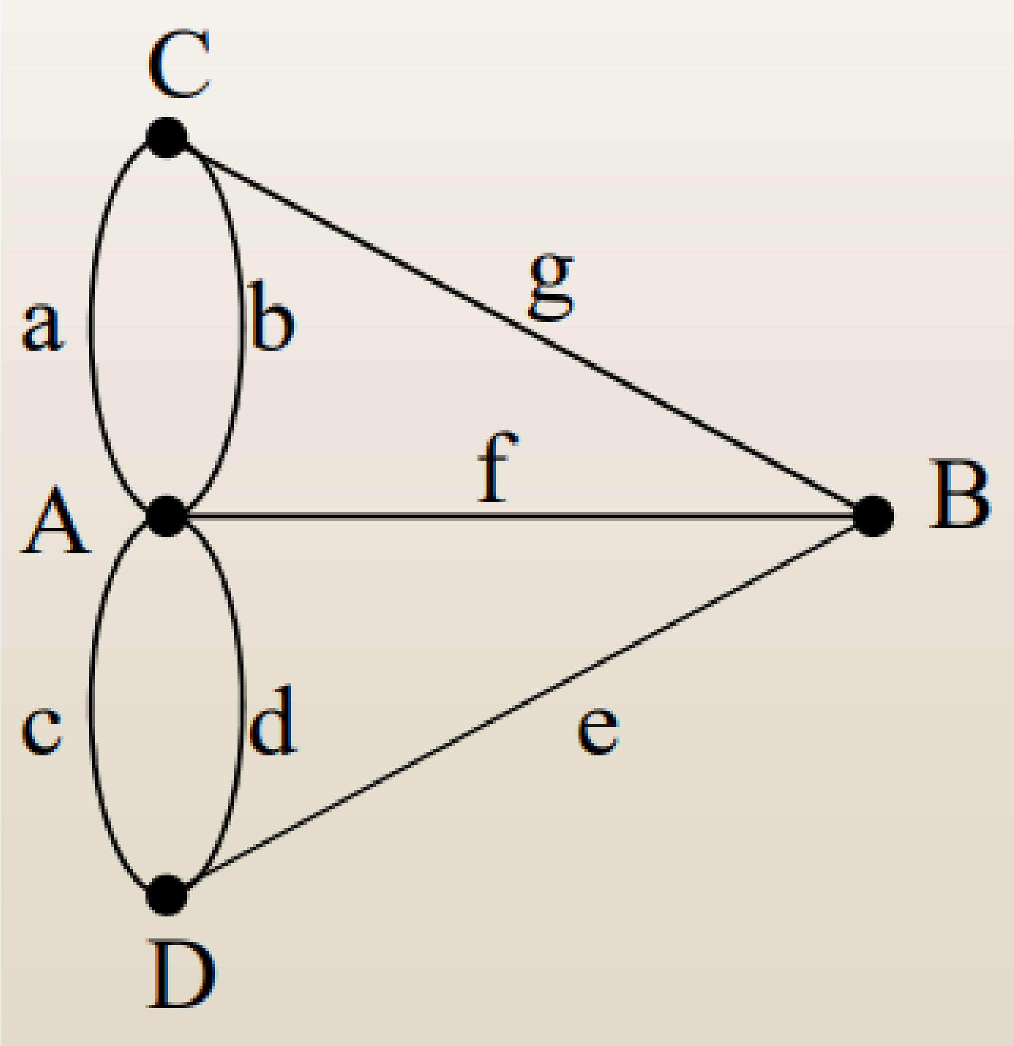
Hamilton graph



Hamiltonian graph

1. Is it connected?
2. Does it have any Hamilton cycle/circuit?

Remember: Hamilton is all about **nodes**.



Semi-Hamiltonian graph

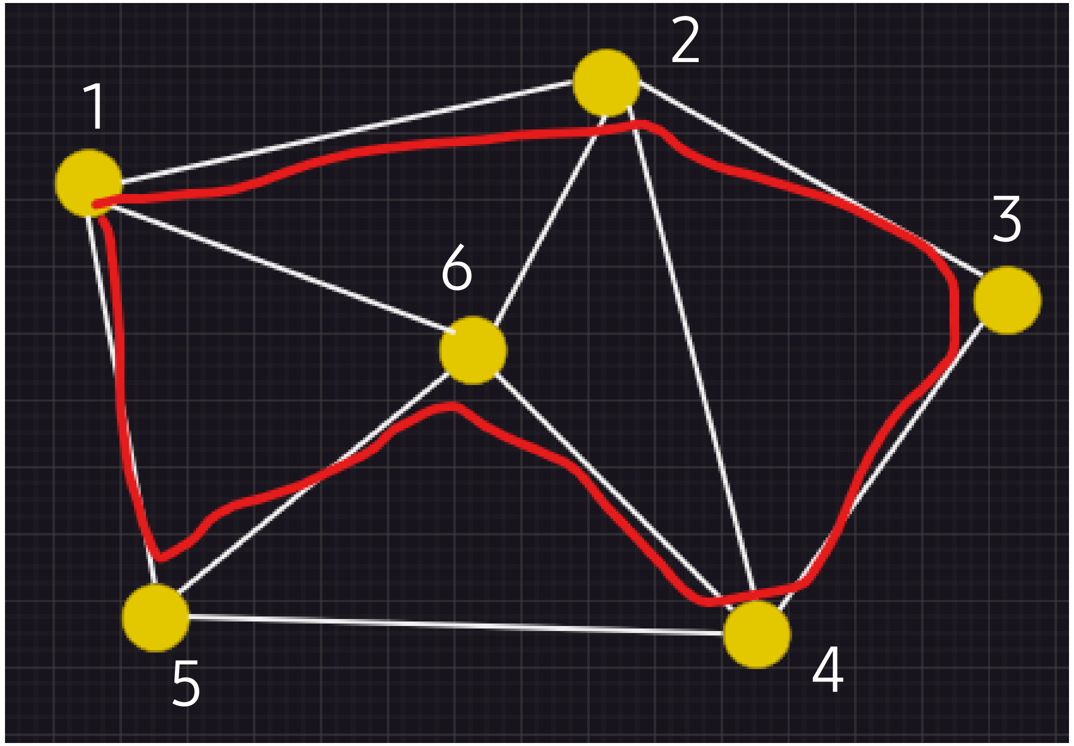
1. Is it Hamiltonian graph? If not, go to step 2.
2. Is it connected?
3. Does it have any Hamilton path?

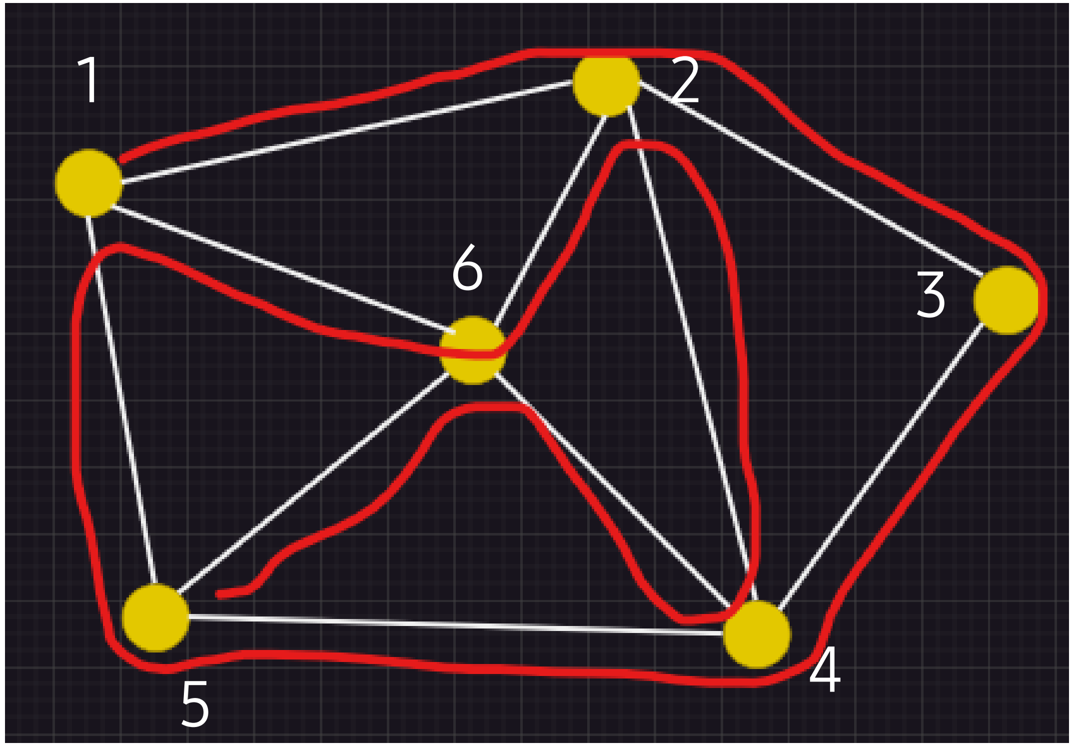
Sadly, there is no easy way to test whether it's a (semi-) Hamiltonian graph or not. 😰

Conjecture!

- A complete graph K_n is Hamiltonian if $n \geq 3$.
- A connected graph with at least 3 vertices which does not contain a $K_{1,3}$ (subgraph) is Hamiltonian.
- Lemma: If C is an Hamiltonian cycle in $G(V, E)$, then $c(G - S) \leq c(C - S) \leq |S|$
  - $S \subseteq V$
  - $G - S$: Subgraph of G in which no nodes of S exists.
  - c(): Number of (connected) component.

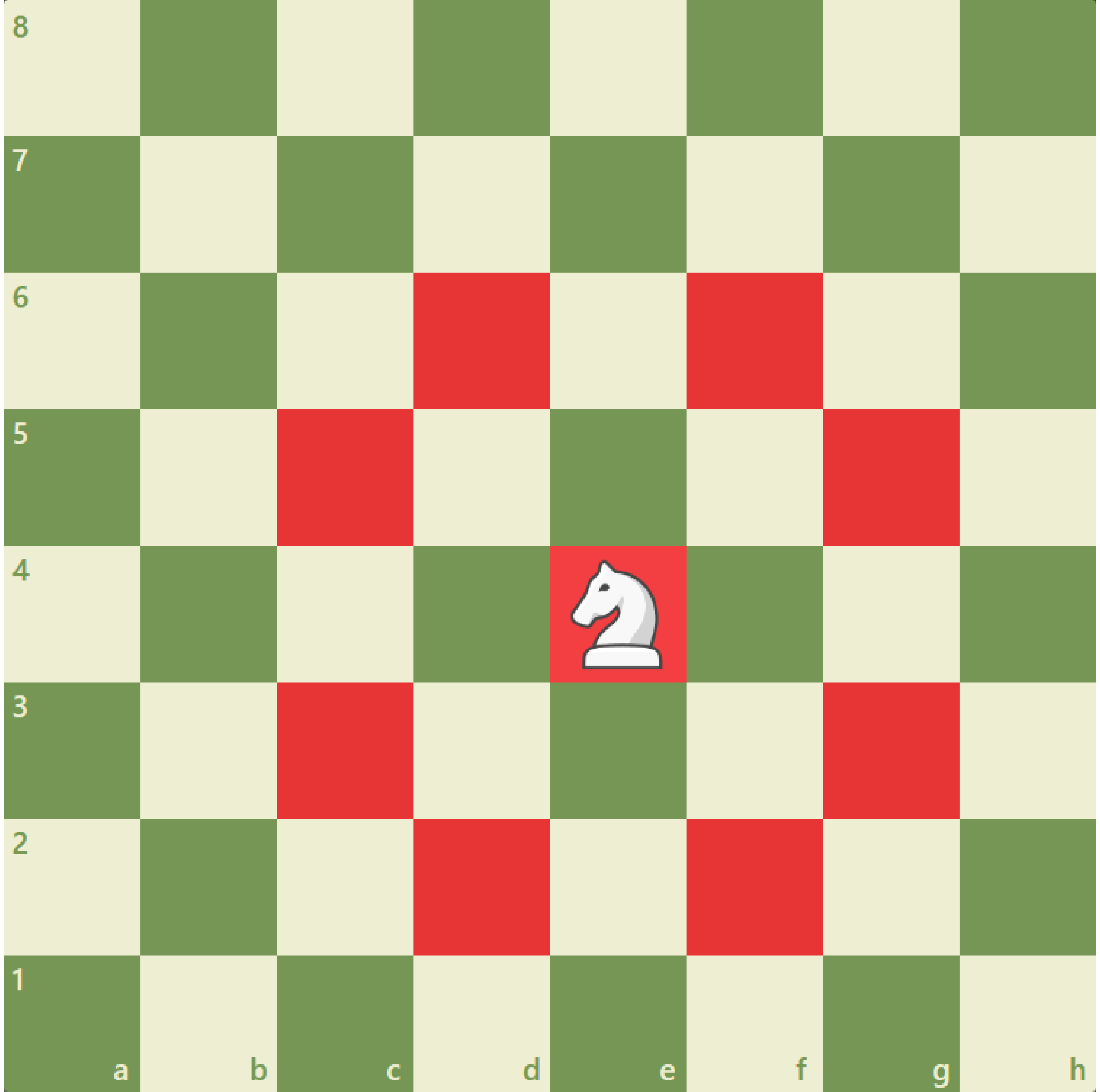
Non-semi graph is easier than semi one

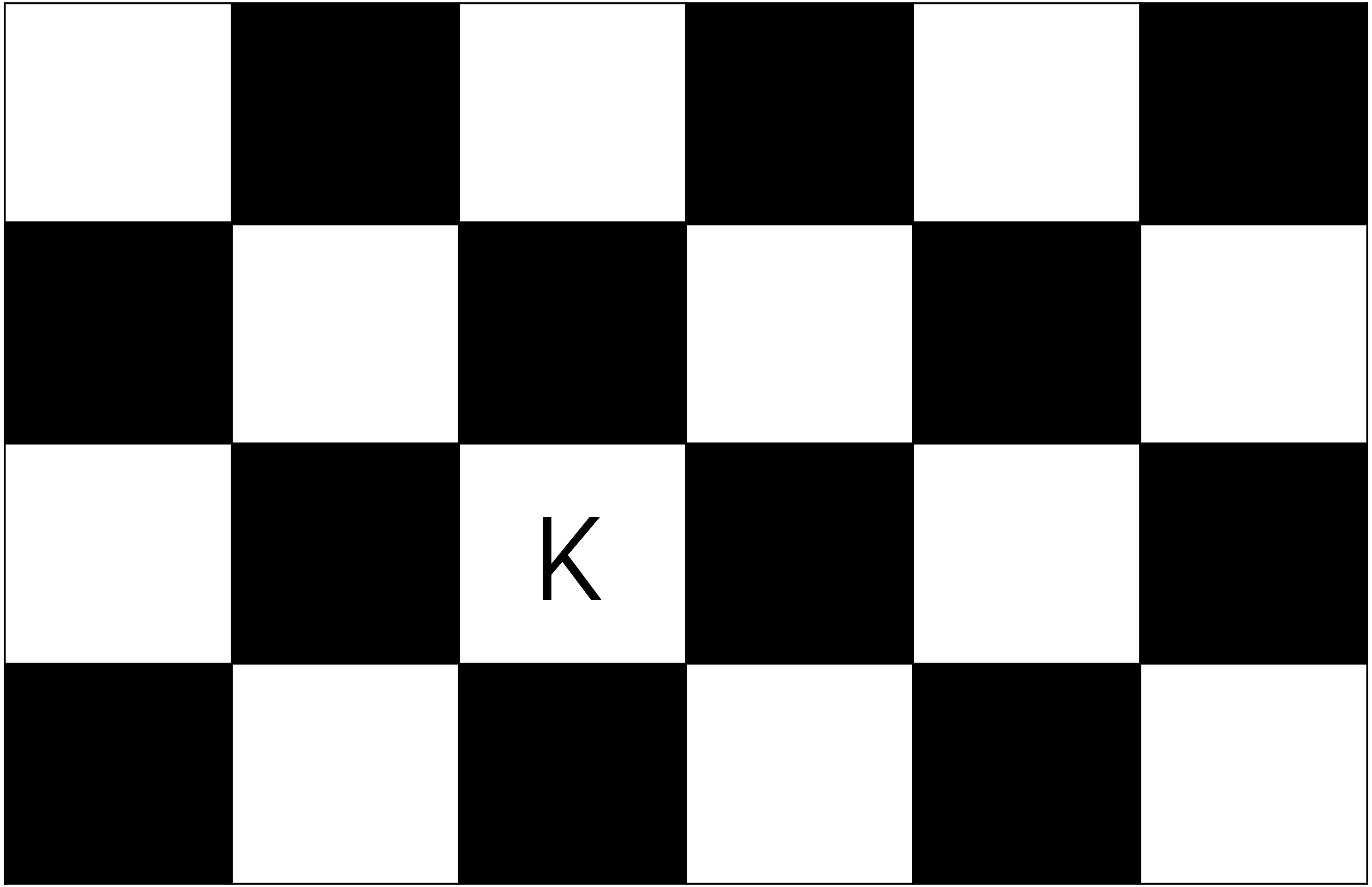


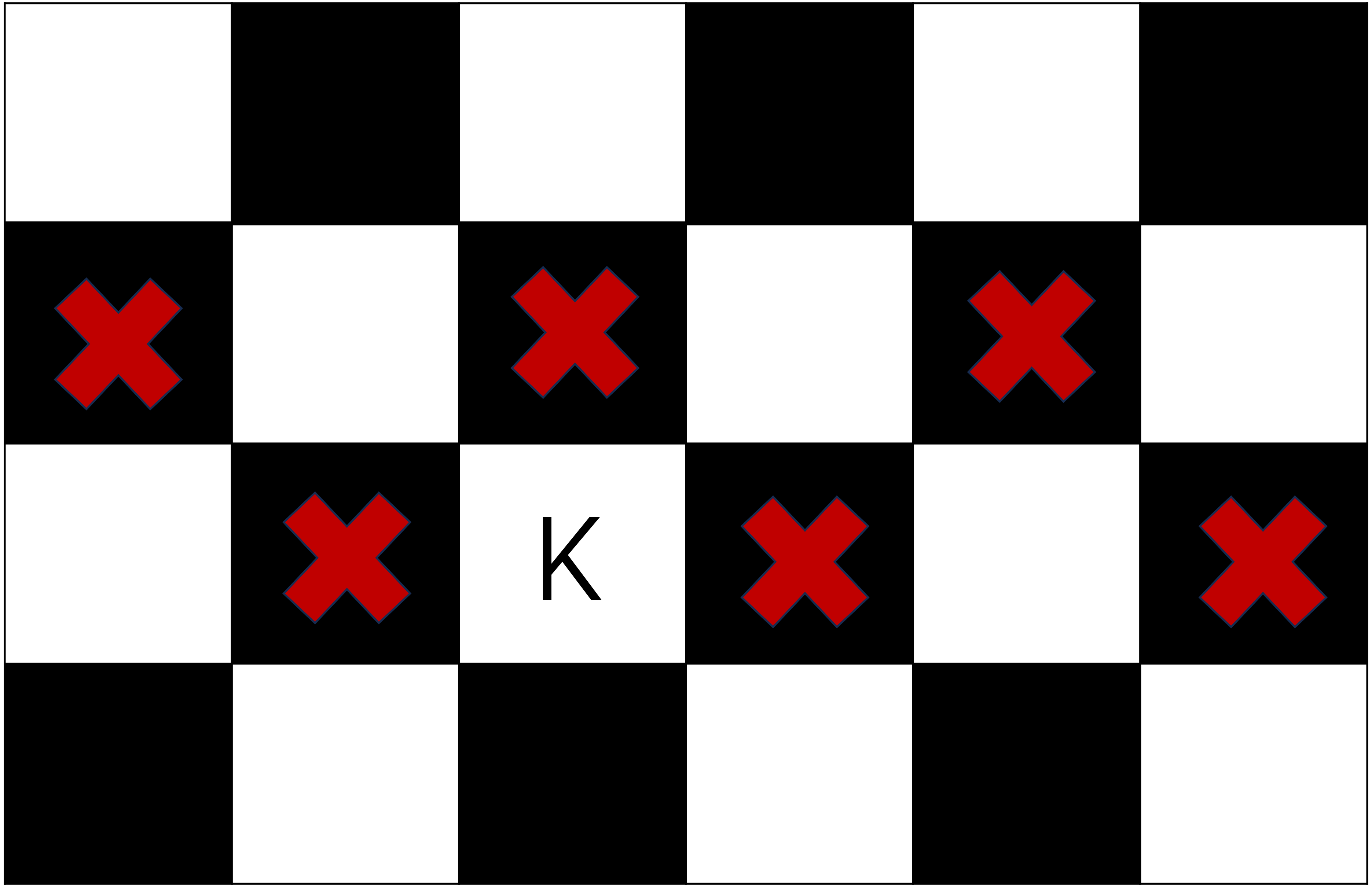


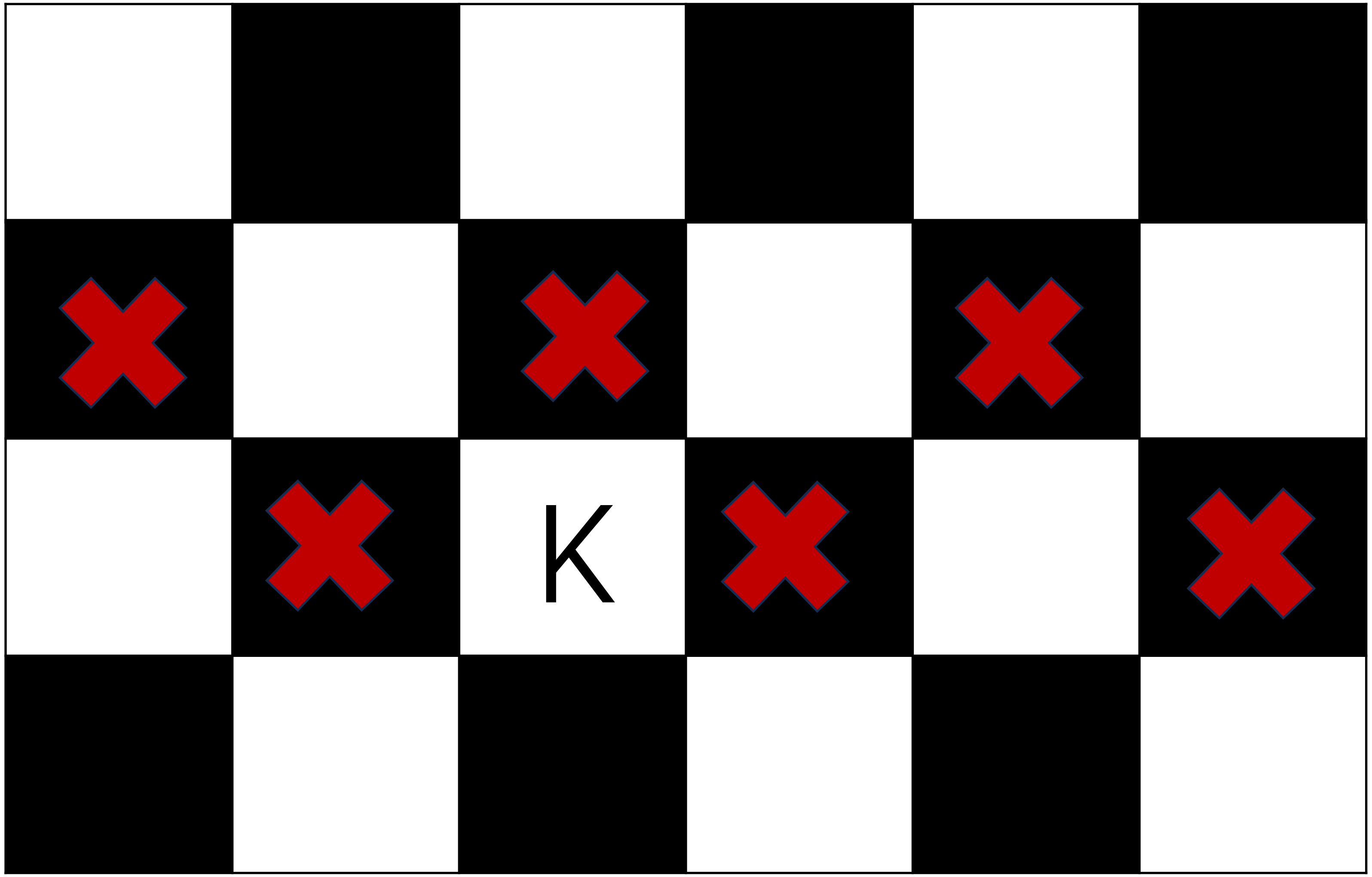
Exercise

On a chessboard, a knight can move from one square to another if they differ by 1 in one coordinate and 2 by another. A knight-tour is a path that a knight visiting every single square exactly once and return to the starting square. Show that a 4-by-n chessboard contains no knight-tour for all n.


8
7
6
5
4
3
2
1
a b c d e f g h

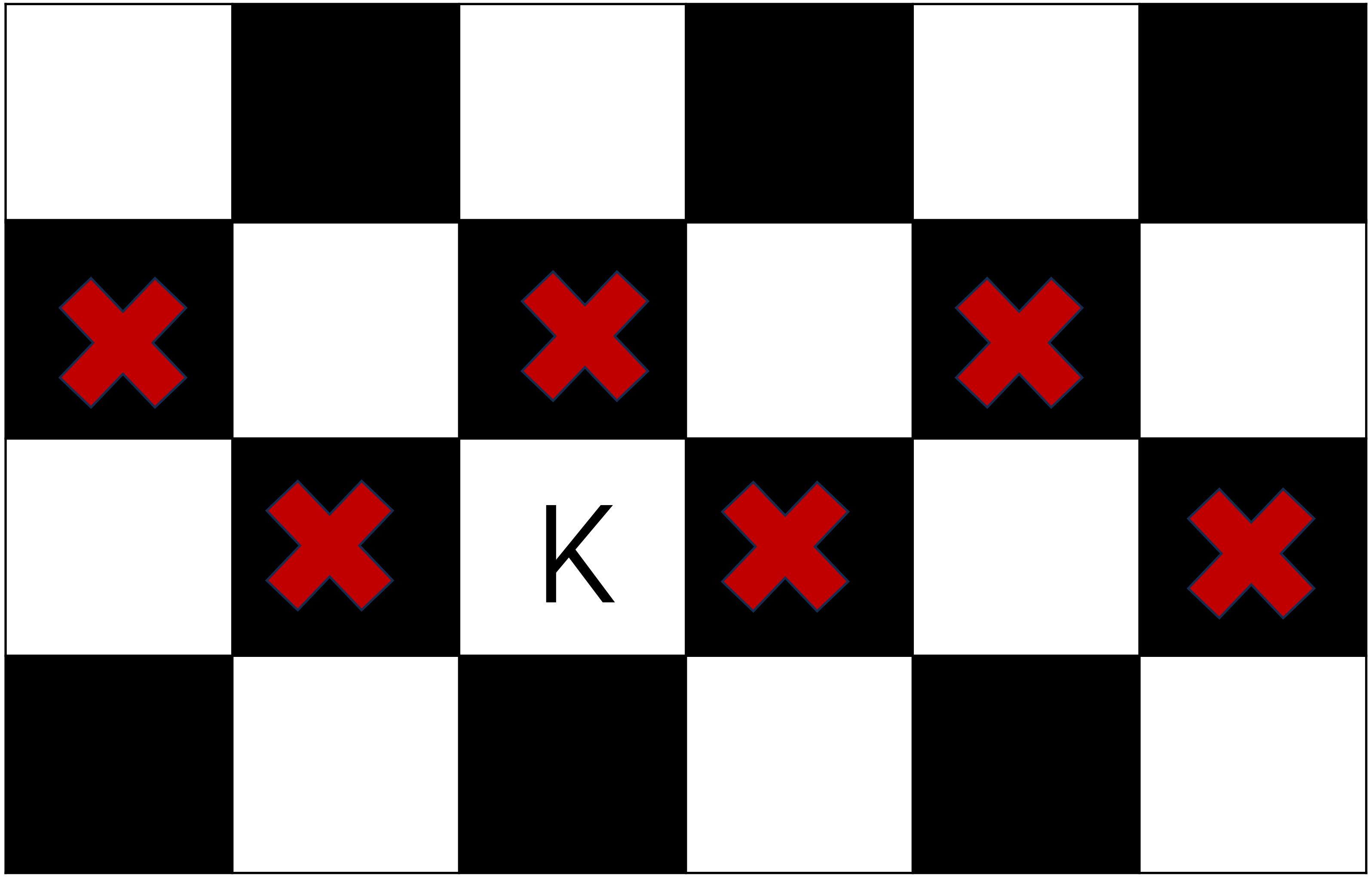

K


K



Let n be the number of disconnected white node.

=> Number of component is at least $n + 1$.



$$c(G - S) \geq n + 1$$

$$c(G - S) > n$$

We also have:

$$|S| = n$$

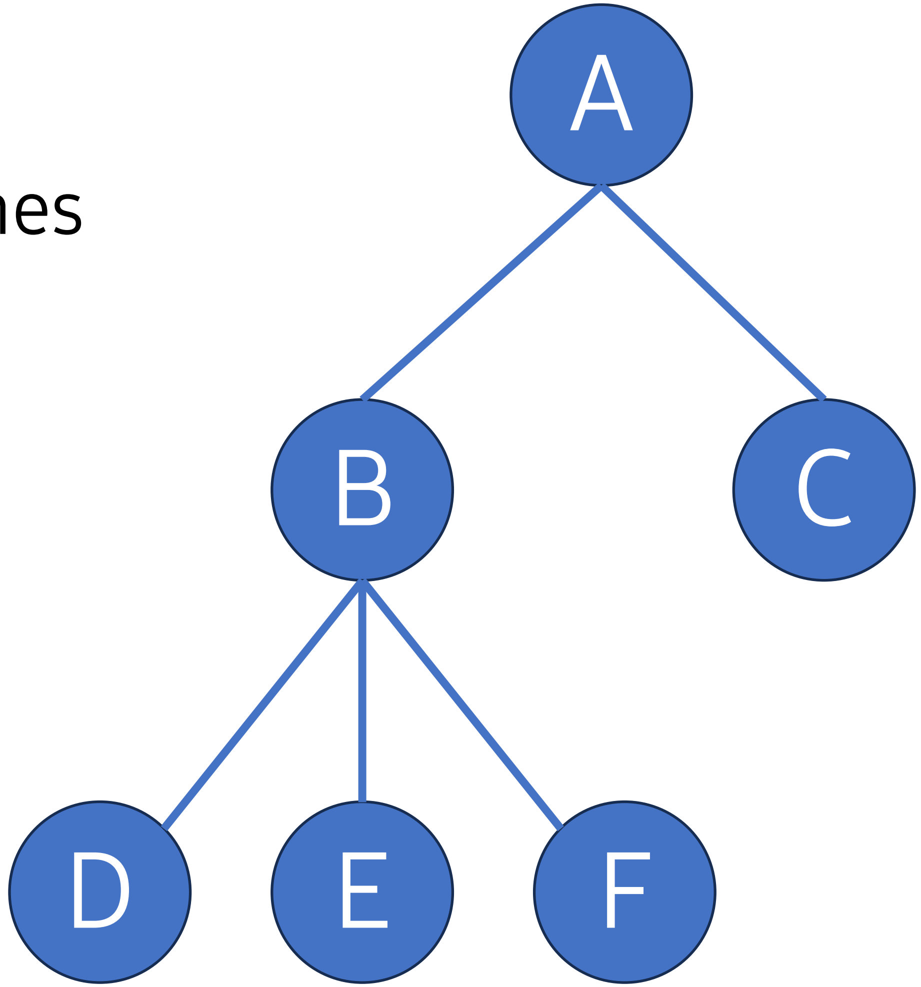
So:

$$c(G - S) > |S|$$

IV. Graph traversal algorithms

Trees

- There are root, leaves (nodes), branches (arcs).
- No cycle exists: While traveling, you <u>can't go back to the previous node</u>.
- Trees are undirected and connected.

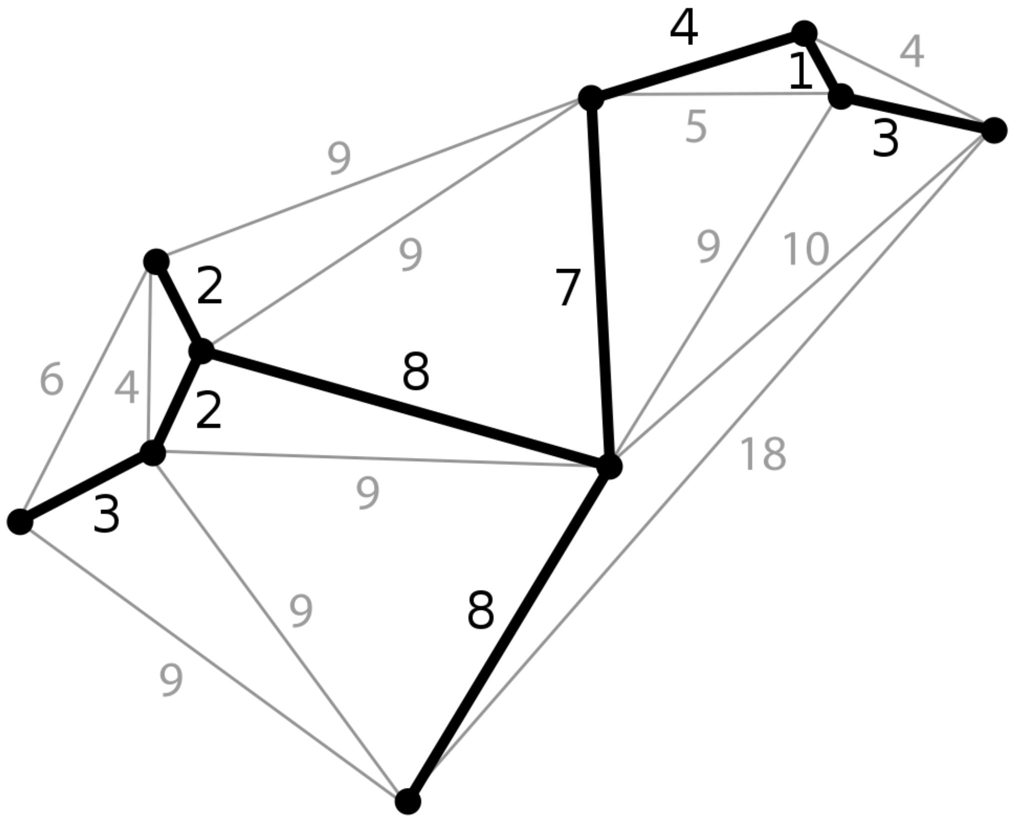


Equivalent properties

- G is a tree.
- G does not contain cycle and has $n - 1$ edges.
- G is connected and has $n - 1$ edges.
- G is connected and all edges are bridges.
- There is exactly 1 edge between any pair of nodes.
- G does not contain cycle, but after adding one edge, we have exactly 1 cycle.

Spanning tree

- A spanning tree is a tree subgraph which contains all vertices in the bigger graph.
- A spanning tree becomes minimum if sum of all (edge) weights is minimum.

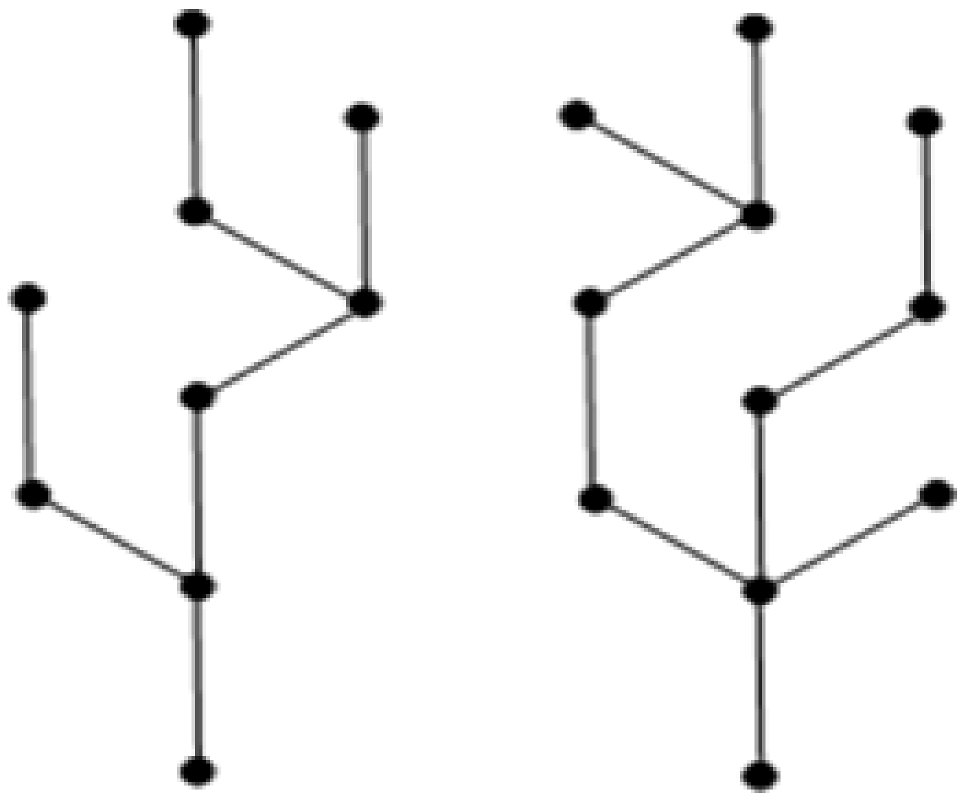


Other types

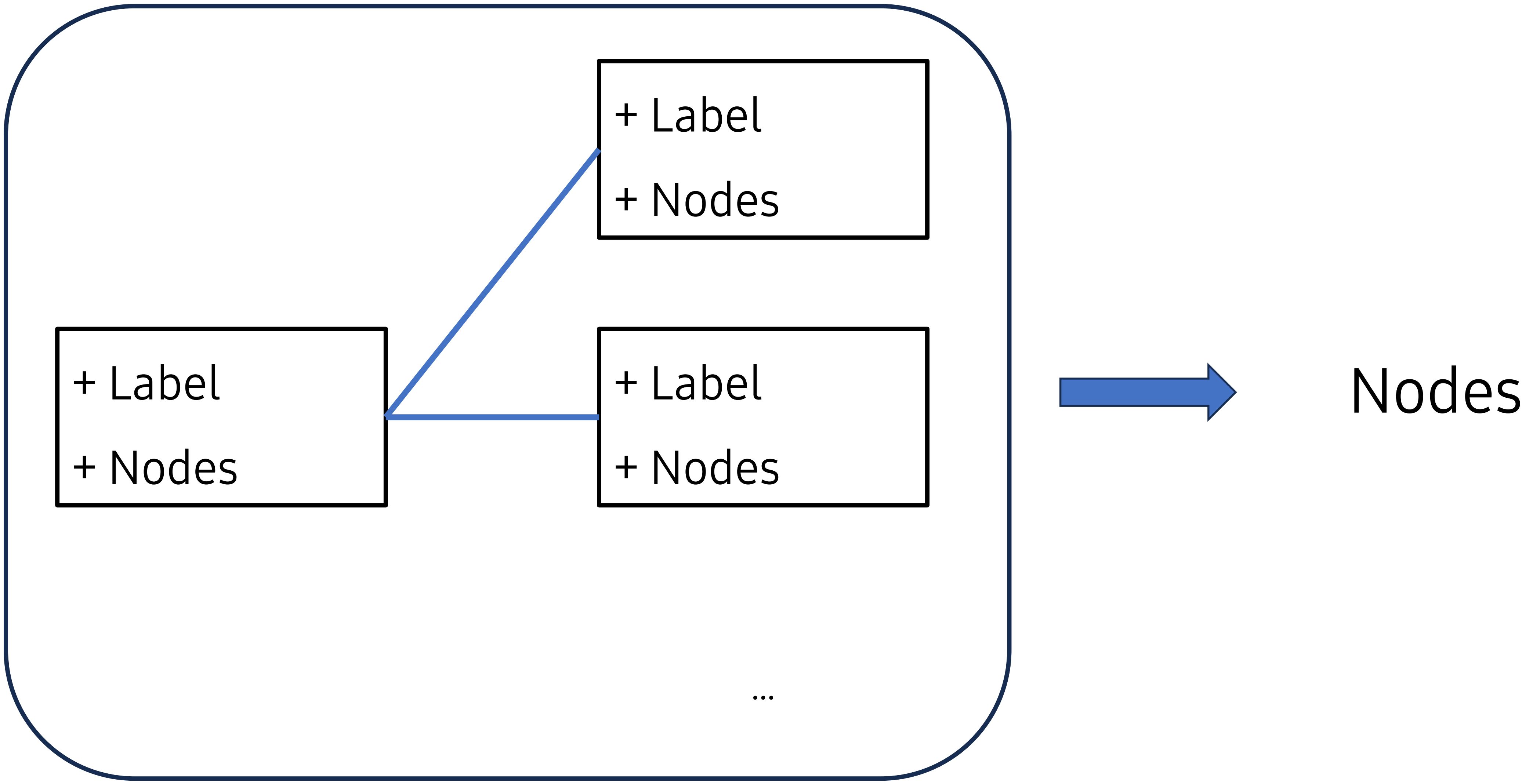
- Rooted tree
- Directed tree
- Binary tree
- Heap tree
- …

Forest

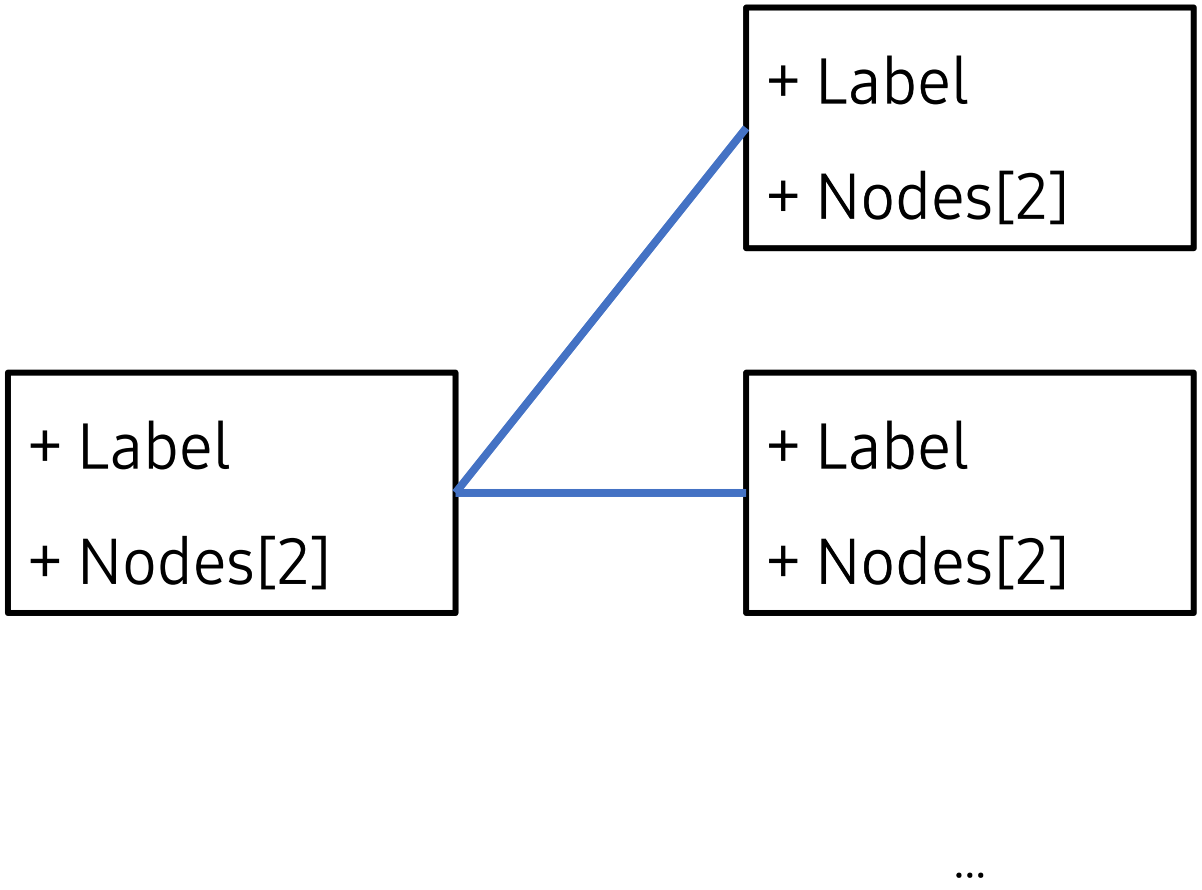
- A bunch of trees that are components is a forest.
- There’s no cycle in forest.



Normal graph data structure



Tree data structure



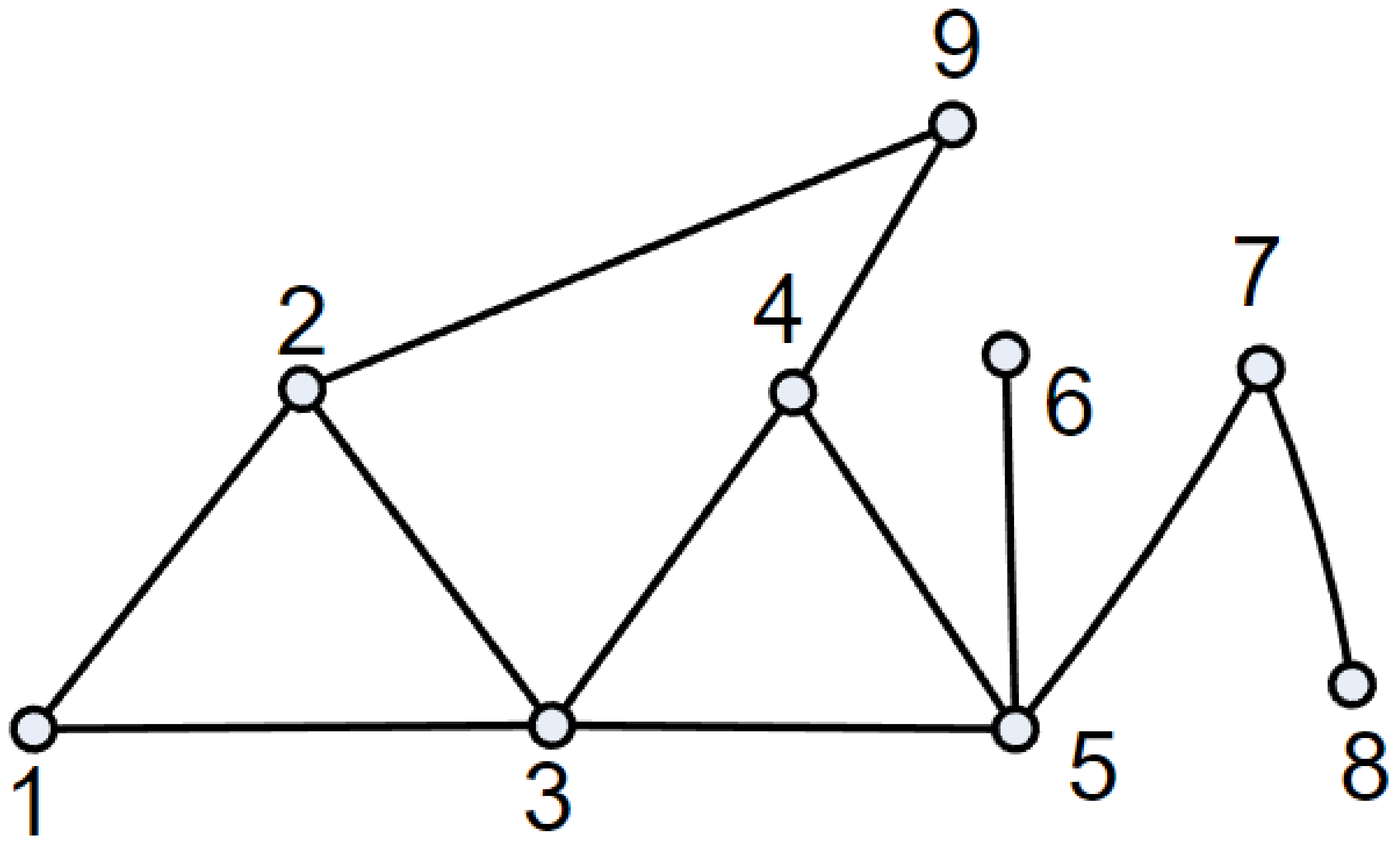
Goals

- Scanning
- Searching
- Path finding
- Connectivity check
- Minimum spanning tree search

Algorithms

Algorithms	Scanning	Searching	P.Finding	Conn. check	Min Spanning tree search
DFS		✓	✓ (A to B)	✓	
BFS		✓	✓ (A to B)	✓	
Fleury			✓ (Euler)		
Dijkstra	✓ (A to B)		✓ (shortest)		
Ford-Bellman			✓ (shortest)		
Floyd			✓ (shortest)		
Kruskal					✓ (edges)
Jarnik-Prim					✓ (vertices)

Depth–first search vs. Breadth–first search



Depth-first search

$vs = \emptyset$: A set of visited vertices.

1. Choose a vertex v as a starting position.
2. Visit v, add v to vs if you haven't.
3. Find vertix u that is **not in** vs and connects to v.
4. If u exists, set $v = u$ and go to step 2.
5. If not,

    if $prev\_v$ <u>is the starting position</u> or vs <u>contains all vertices</u>:

        Stop.

    set $v = prev\_v$ and go to step 3.

Breadth-first search

$vs = \emptyset$: A set of visited vertices.

1. Choose a vertex v as a starting position.
2. Visit v, add v to vs if you haven't.
3. Find **all vertices** us that are **not in** vs and connects to v.
4. If $\underline{us = \emptyset}$ or $\underline{vs \text{ contains all vertices}}$, stop.
5. For each $u \in us$, set $v = u$ and go to step 2.

DFS vs. BFS

Exercises

- Implement graphs and trees using struct in C++.
  - Define those in 2 header files.
  - Tree must have dfs and bfs functions that search for string label and print each visiting step while searching.
  - Executation codes (such as main) are in one cpp file.