Eulerian Cycle (2A)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice and Octave.

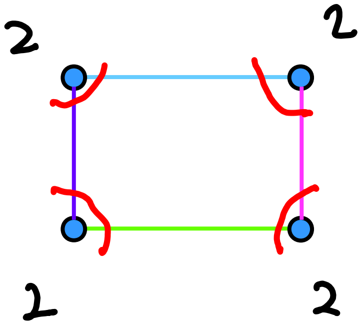
Euler Cycle

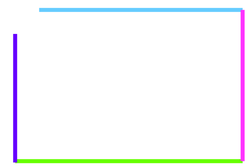
visits every **edge** exactly once

the existence of **Eulerian cycles**

all **vertices** in the graph have an **even** degree

connected graphs with **all vertices** of **even** degree have an **Eulerian cycles**





https://en.wikipedia.org/wiki/Eulerian_path

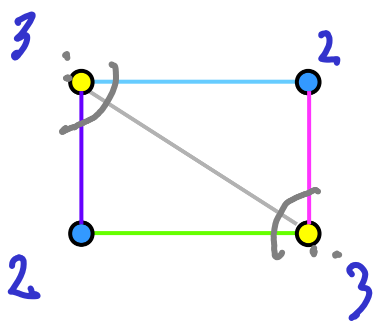
Euler Path

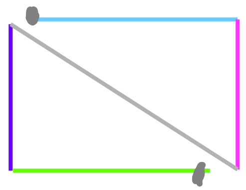
visits every **edge** exactly once

the existence of **Eulerian paths**

all the **vertices** in the graph have an **even** degree

except only **two** vertices with an **odd** degree



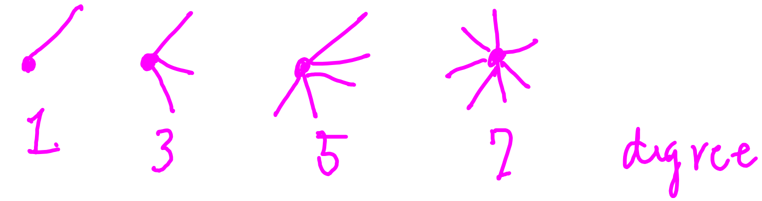


An **Eulerian path** starts and ends at different vertices
An **Eulerian cycle** starts and ends at the same vertex.

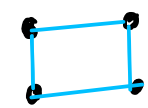
https://en.wikipedia.org/wiki/Eulerian_path

Conditions for Eulerian Cycles and Paths

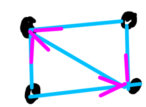
An odd vertex = a vertex with an odd degree
An even vertex = a vertex with an even degree



# of **odd** vertices	Eulerian **Path**	Eulerian **Cycle**
0	No	**Yes**
2	**Yes**	No
4,6,8, …	No	No
1,3,5,7, …	No such graph	No such graph



no odd vertex



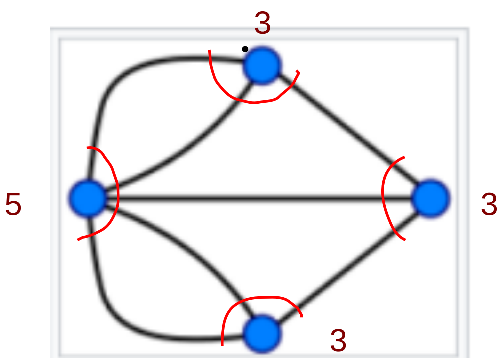
2 odd vertices

vertices with degree 3

If the graph is connected

http://people.ku.edu/~jlmartin/courses/math105-F11/Lectures/chapter5-part2.pdf

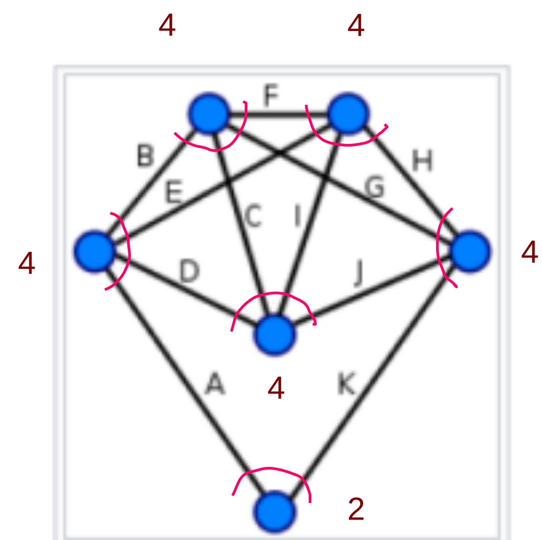
Odd Degree and Even Degree



The Königsberg Bridges multigraph. This multigraph is not Eulerian, therefore, a solution does not exist.

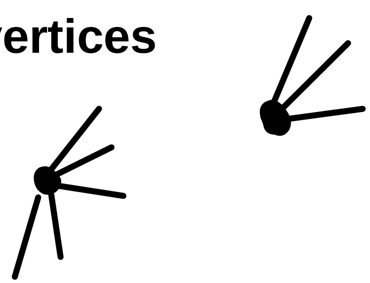
0 .... E. Cycle
2 .... E. Path
4
6
8
⋮
E.C X
E.P X

# of odd vertices degree odd



Every vertex of this graph has an even degree. Therefore, this is an Eulerian graph. Following the edges in alphabetical order gives an Eulerian circuit/cycle.

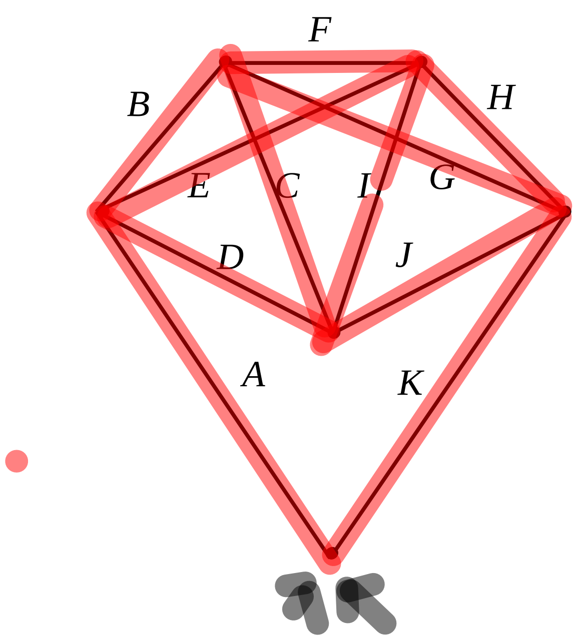
All <u>odd</u> degree vertices



All <u>even</u> degree vertices

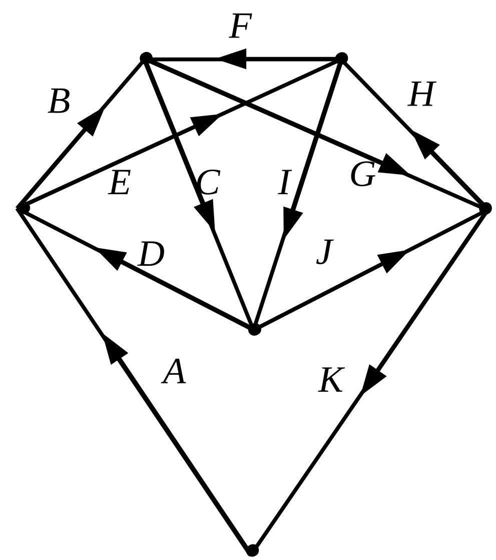
https://en.wikipedia.org/wiki/Eulerian_path

Euler Cycle Example



ABCDEFGHIJK

a path denoted by
the edge names



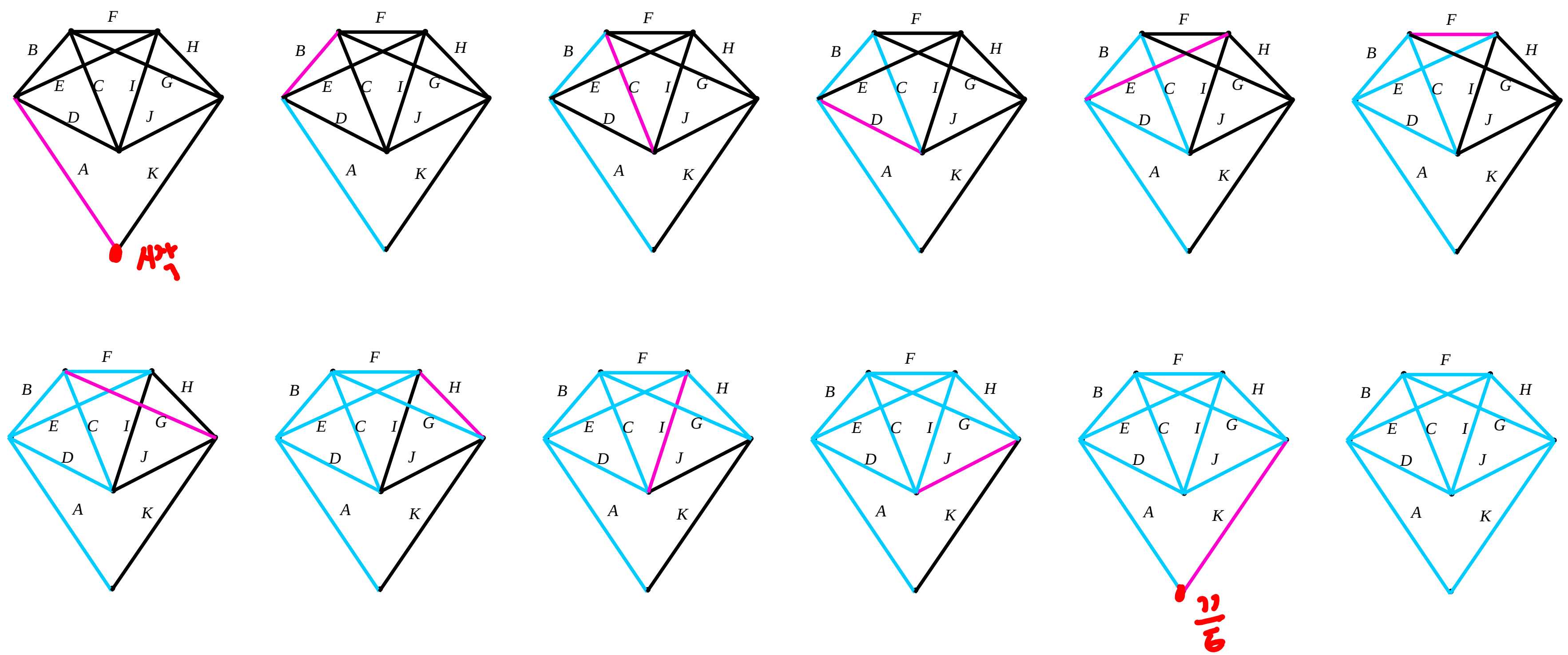
All <u>even</u> degree vertices

Eulerian Cycles

en.wikipedia.org

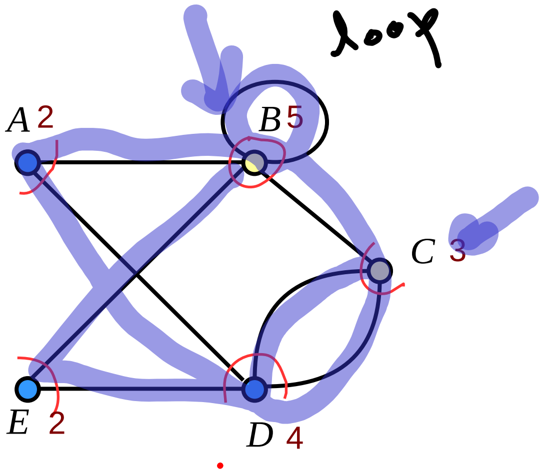
Euler Cycle Example

ABCDEFGHIJK

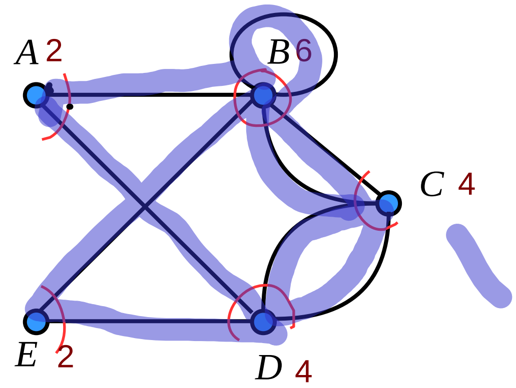


en.wikipedia.org

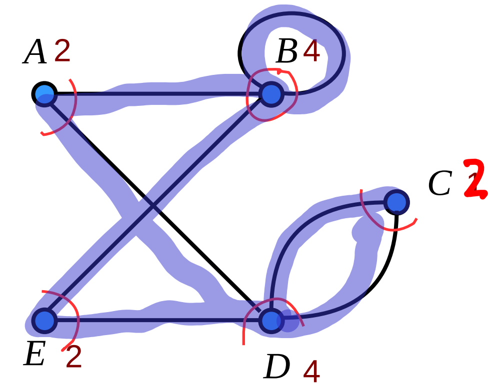
Euler Path and Cycle Examples



Eulerian Path
1. BBADCDEBC
2. CDCBBADEB



Euerian Cycle
1. CDCBBADEBC

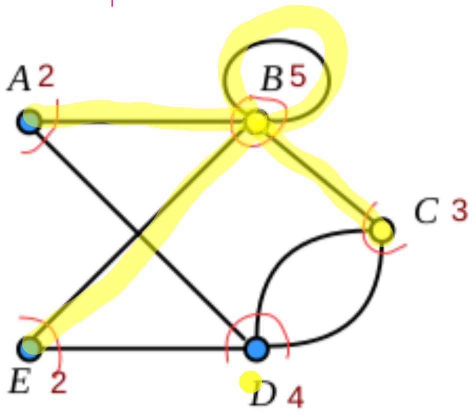


Euerian Cycle
2. CDEBBADC

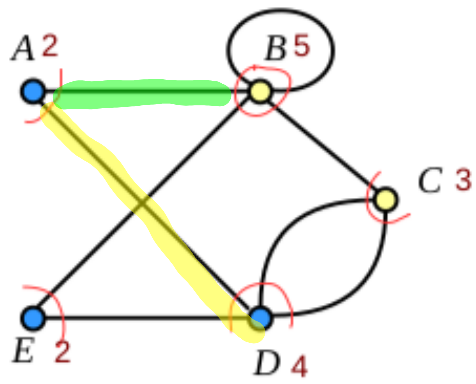
a path denoted by
the vertex names

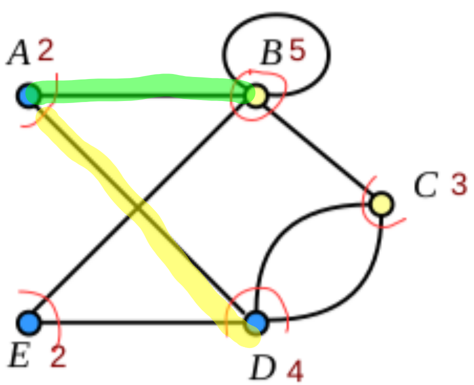
http://people.ku.edu/~jlmartin/courses/math105-F11/Lectures/chapter5-part2.pdf

5 choices

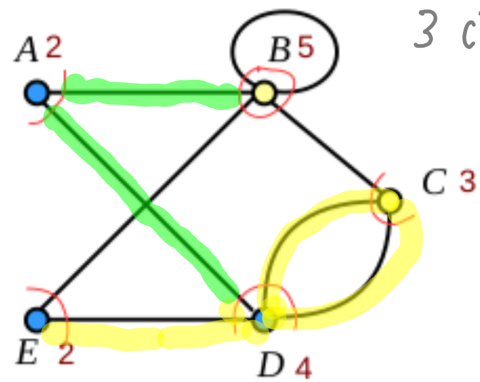


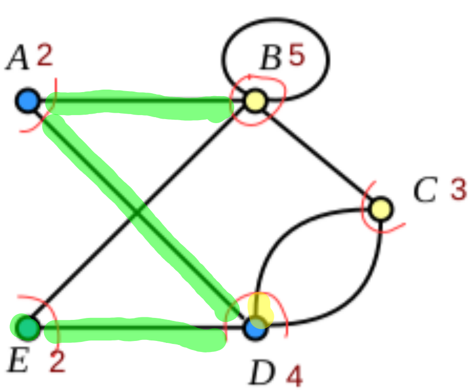
1 choice



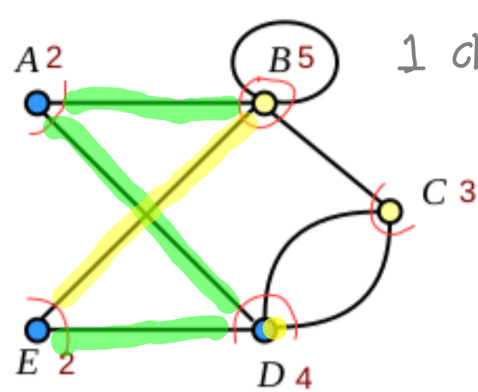


3 choices

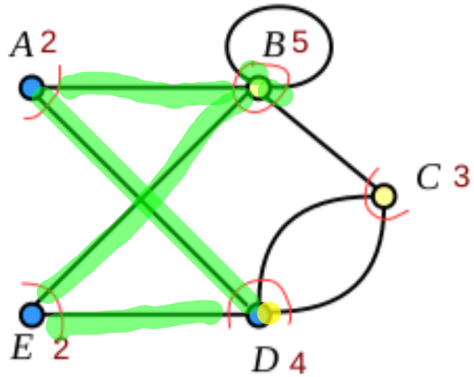


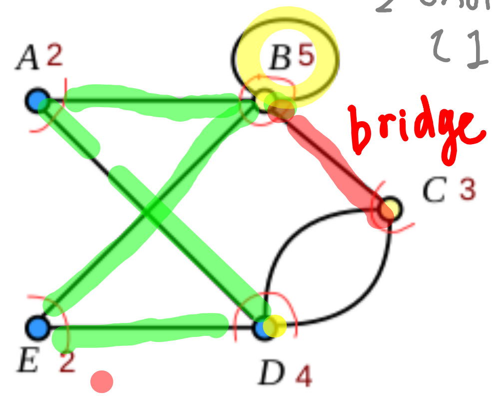


1 choice



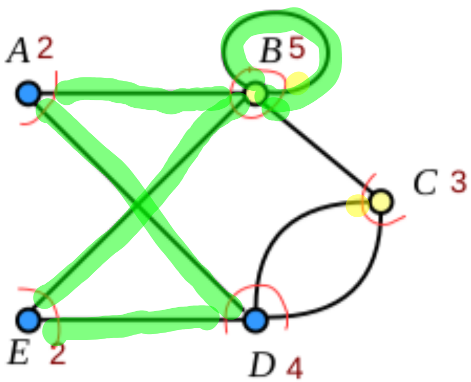
BADE

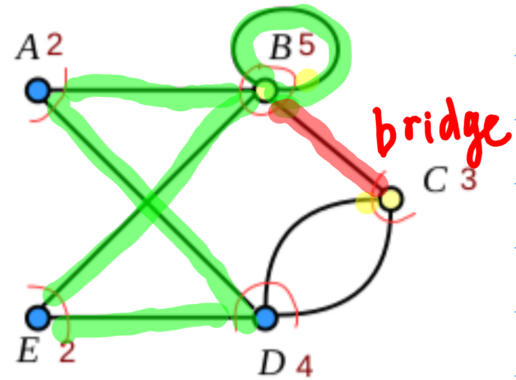




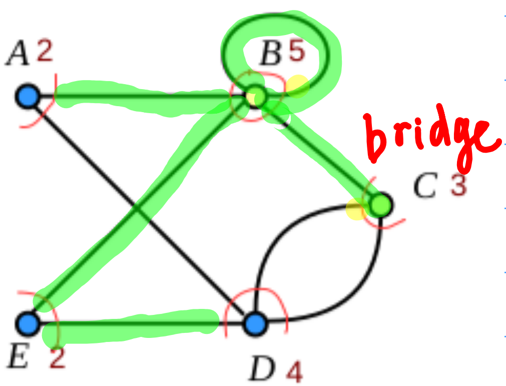
2 choices
(1 choice is a bridge)

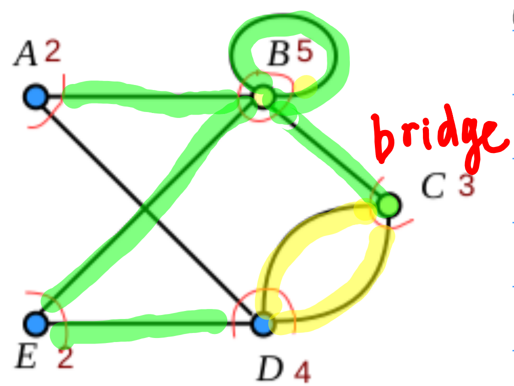
BADEBBCDC



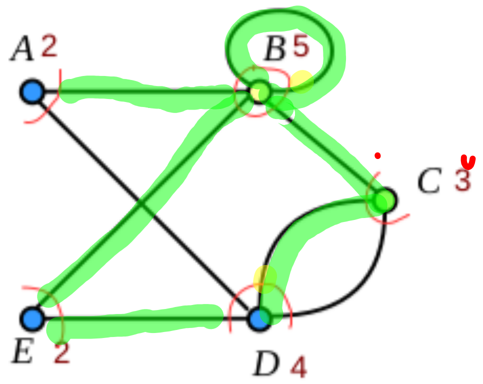


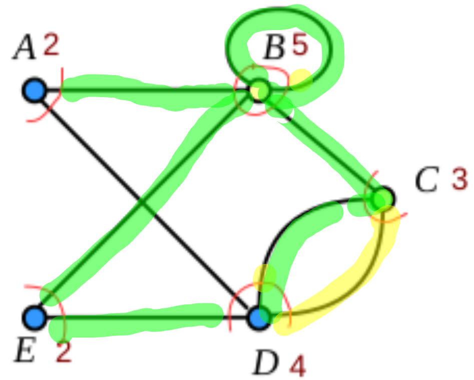
1 choice



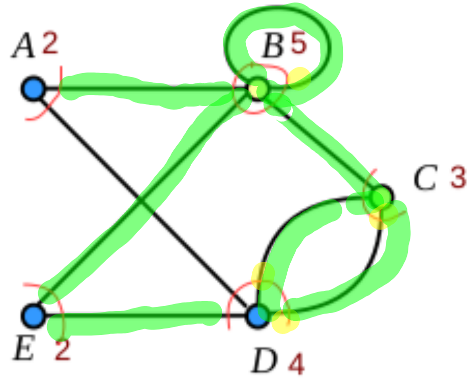


2 choices





1 choices



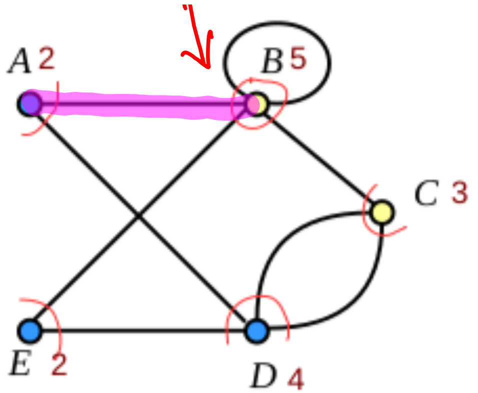
BADEBBCDC

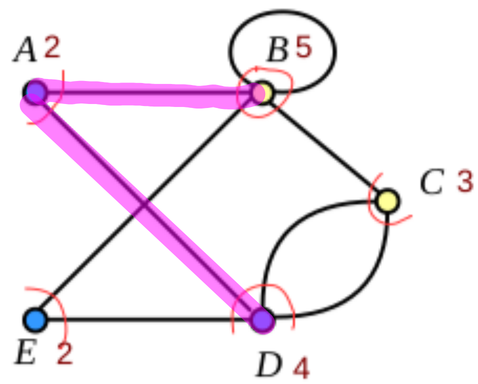
could be many Eulerian paths (not unique)

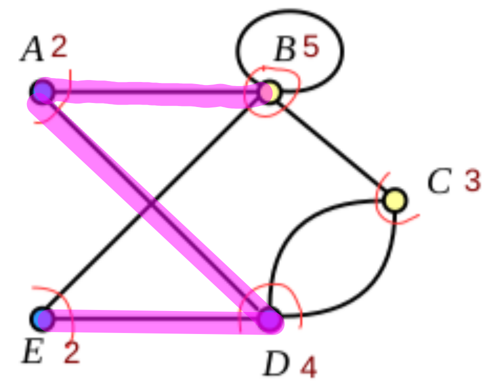
Eulerian Path

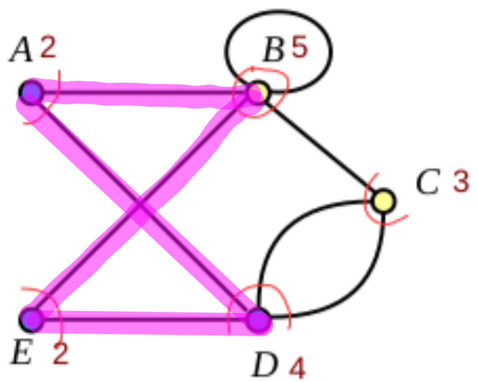
1. BBADCDEBC
2. CDCBBADEB

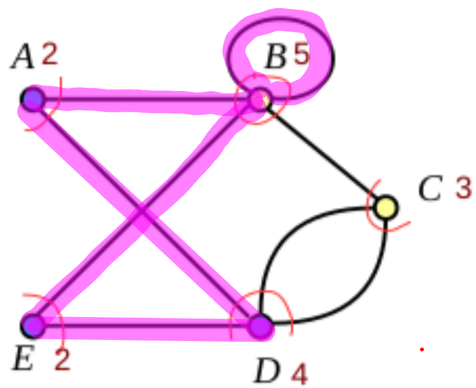
BADEBBCDC

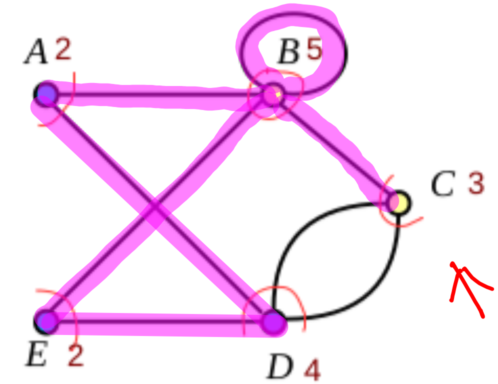




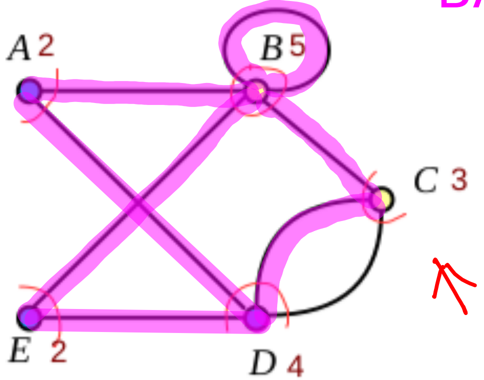


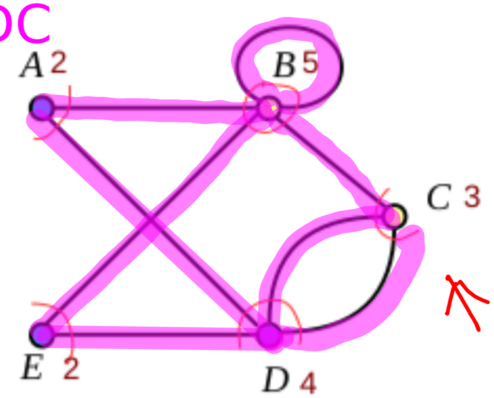






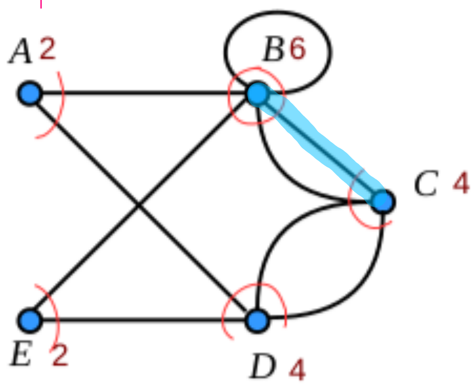
BADEBBCDC

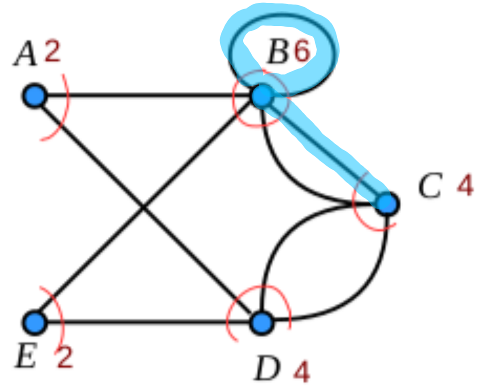


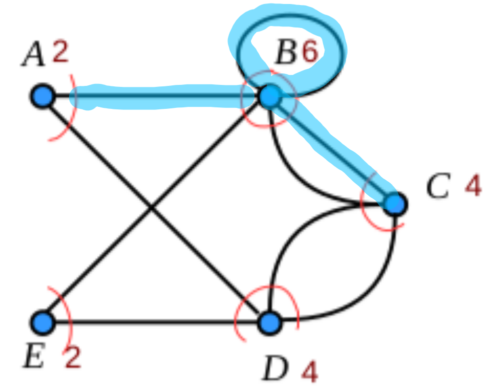


Euerian Cycle
1. CDCBBADEBC

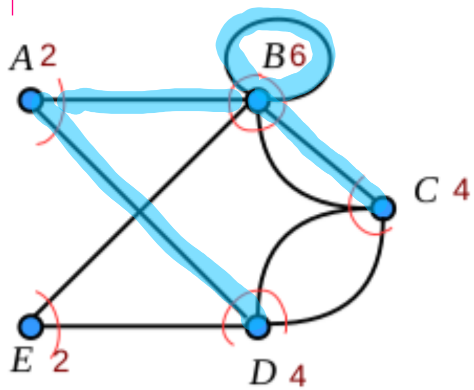
could be many Eulerian cycles (not unique)

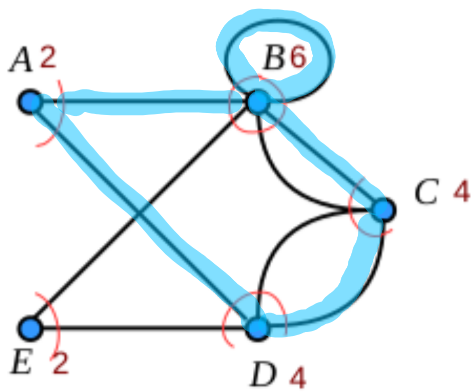


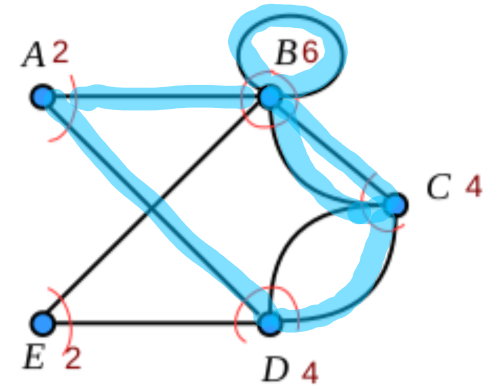


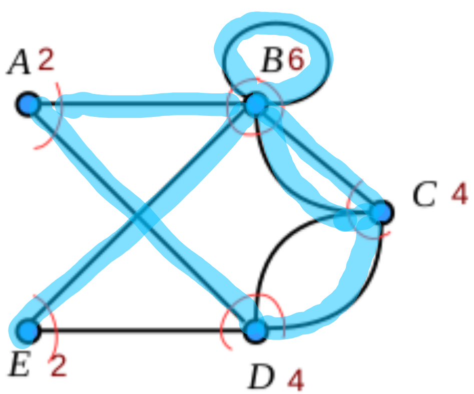


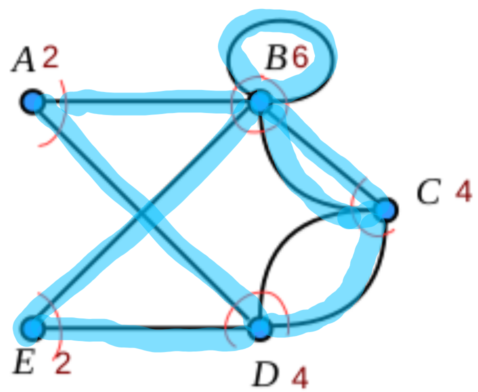
CBBABCDBEDC

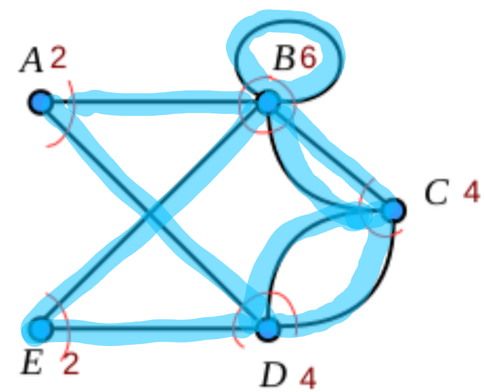


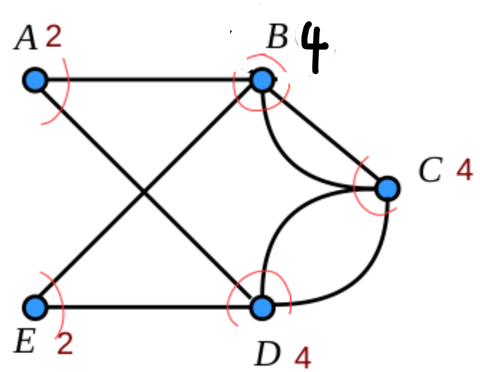


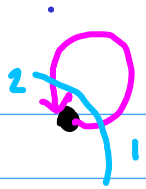












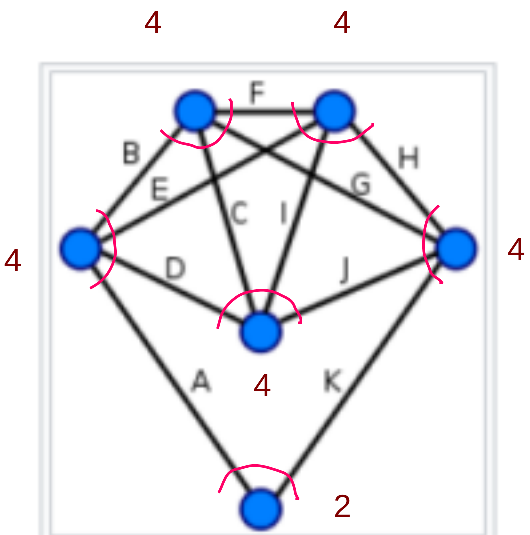
loop

Eulerian Cycles of Undirected Graphs

An **undirected** graph has an **Eulerian** **cycle**
if and only if every **vertex** has **even degree**,
and all of its **vertices** with **nonzero degree**
belong to a **single** **connected** **component**.

An **undirected** graph can be
decomposed into **edge-disjoint cycles**
if and only if all of its **vertices** have **even degree**.

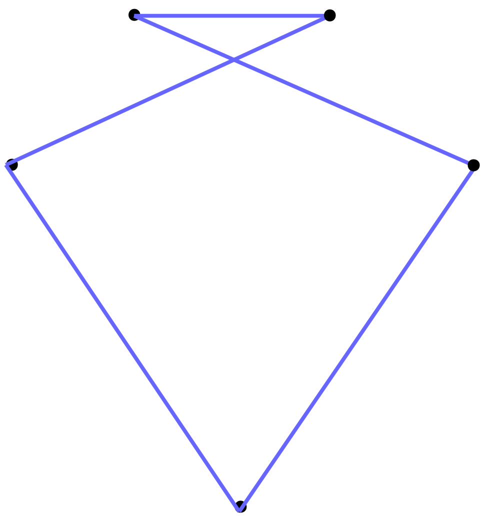
So, a graph has an **Eulerian** **cycle**
if and only if it can be decomposed
into **edge-disjoint cycles**
and its **nonzero-degree** vertices
belong to a **single connected component**.

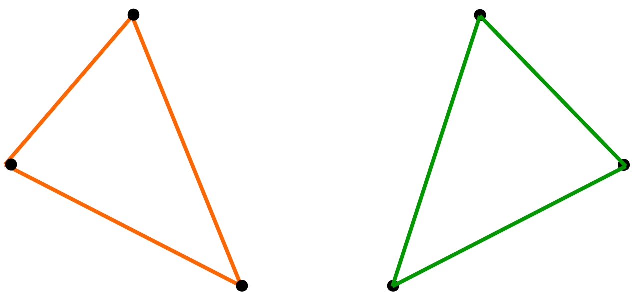


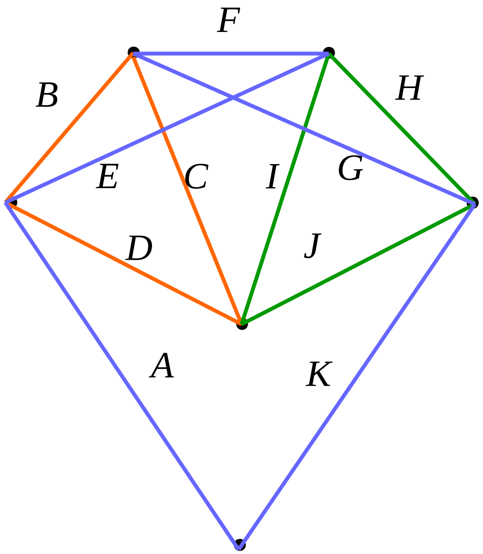
Every vertex of this graph has an even degree. Therefore, this is an Eulerian graph. Following the edges in alphabetical order gives an Eulerian circuit/cycle.

https://en.wikipedia.org/wiki/Eulerian_path

Edge Disjoint Cycle Decomposition







All even vertices ⇔ **Euerian Cycle** ⇔ **Edge Disjoint Cycles**

Eulerian Paths of Undirected Graphs

An undirected graph has an **Eulerian trail**
if and only if exactly **zero** or **two vertices** have **odd degree**,
and all of its vertices with **nonzero degree**
belong to a **single connected component**.

https://en.wikipedia.org/wiki/Eulerian_path

Eulerian Cycles of DiGraphs

A directed graph has an **Eulerian cycle**
if and only if every vertex has **equal in degree** and **out degree**,
and all of its vertices with nonzero degree
belong to a single strongly connected component.

Equivalently, a directed graph has an Eulerian cycle
if and only if it can be decomposed
into **edge-disjoint directed cycles**
and all of its vertices with nonzero degree
belong to a single strongly connected component.

https://en.wikipedia.org/wiki/Eulerian_path

Eulerian Paths of DiGraphs

A directed graph has an **Eulerian path**
if and only if **at most one** vertex has (out-degree) − (in-degree) = 1,
at most one vertex has (in-degree) − (out-degree) = 1,
every other vertex has equal in-degree and out-degree,
and all of its vertices with nonzero degree belong to a single connected component of the underlying undirected graph.

https://en.wikipedia.org/wiki/Eulerian_path

Seven Bridges of Königsberg

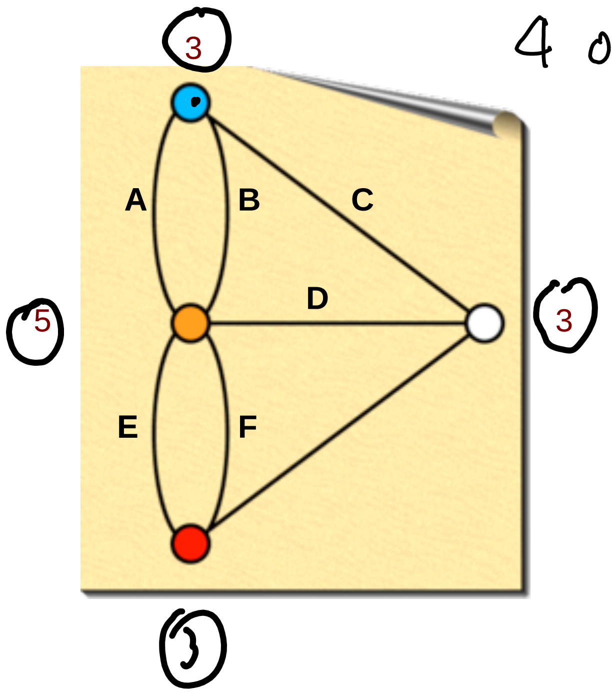
The problem was to devise a walk through the city that would cross each of those bridges once and only once.

https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

Seven and Eight Bridges Problems

7 bridges problem

4 odd vertices

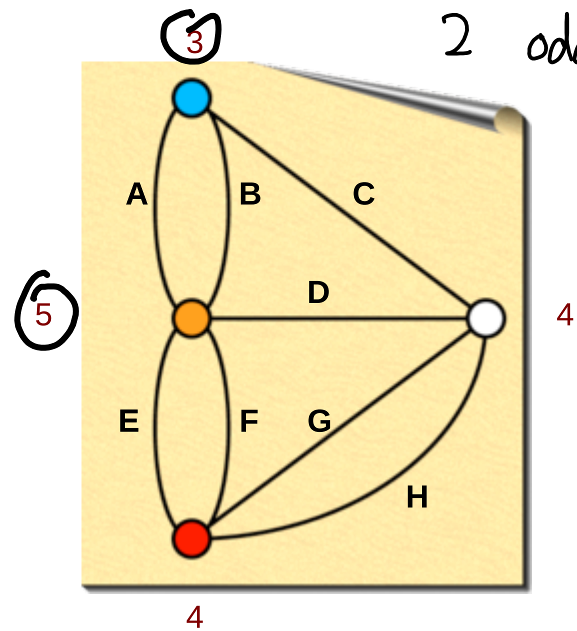


.E. Path X

E cycle X

8 bridges problem

2 odd vertices



Eulerian Path

AEHGFDCB

https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

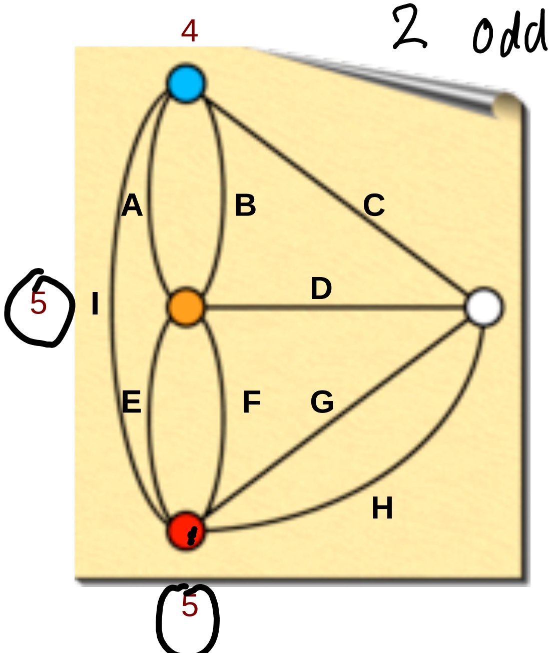
Nine and Ten Bridges Problems

check how many odd vertices!

odd degree

9 bridges problem

2 odd vertices

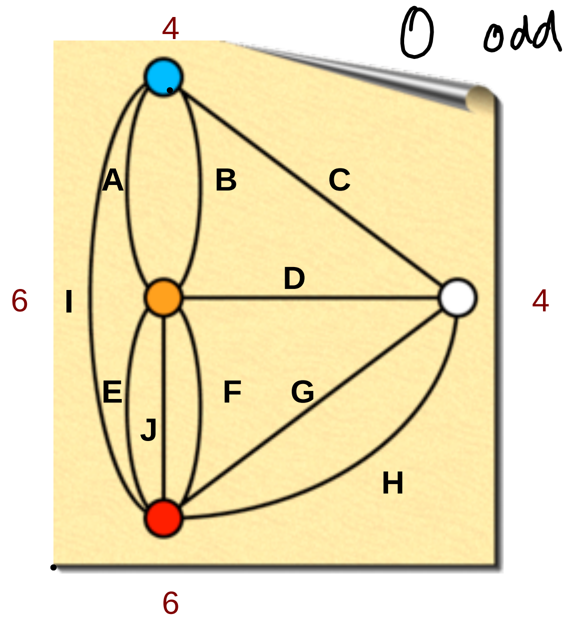


Eulerian Path

EHGFDCBAI

10 bridges problem

0 odd vertex

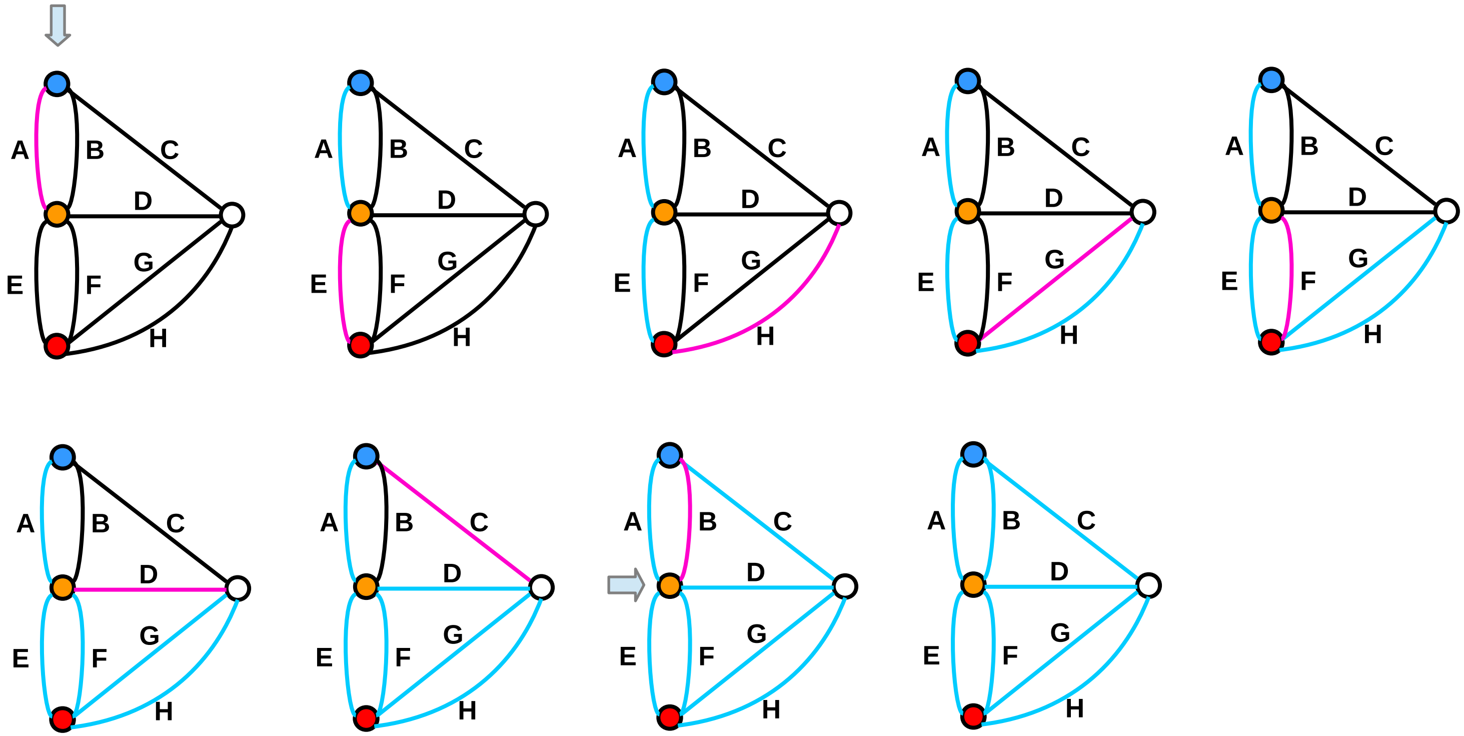


Eulerian Cycle

AEHGFDCBJI

https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

8 bridges – Eulerian Path

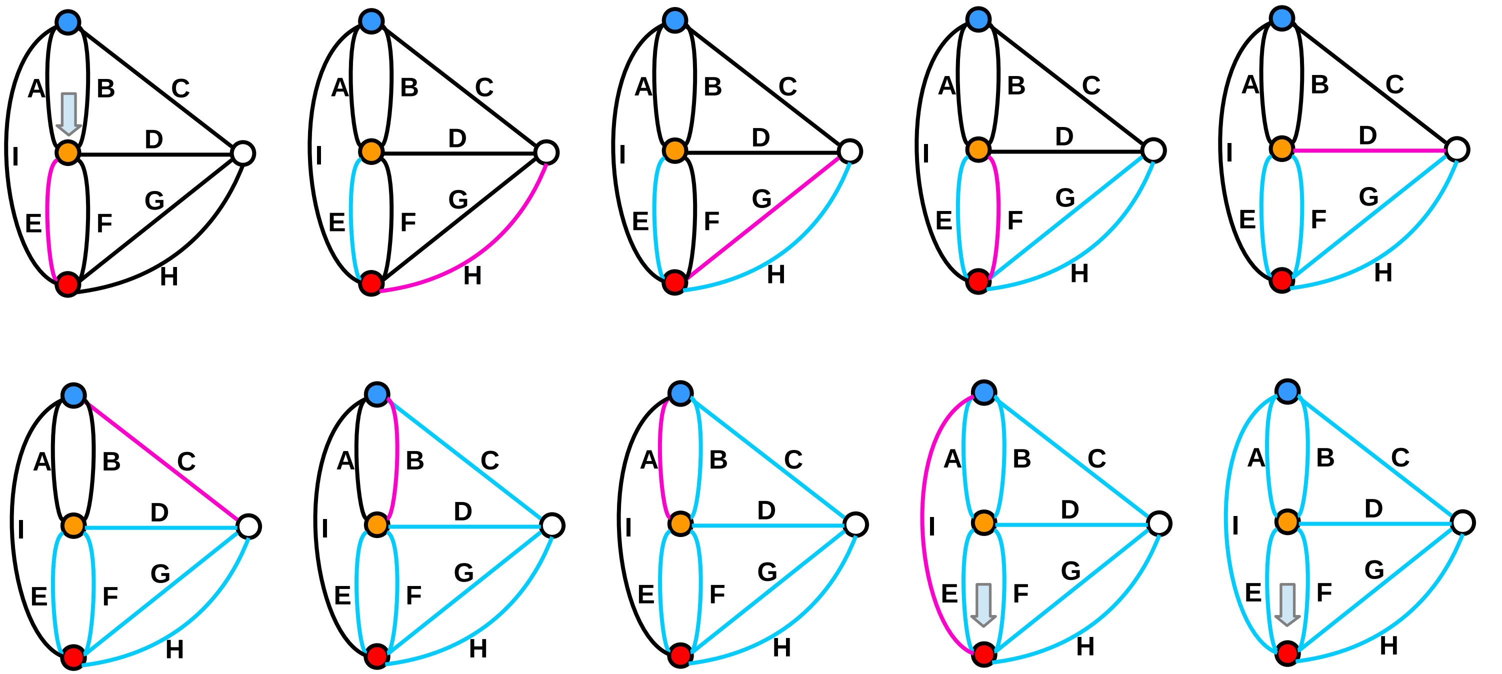


Eulerian Path

AEHGFDCB

https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

9 bridges – Eulerian Path

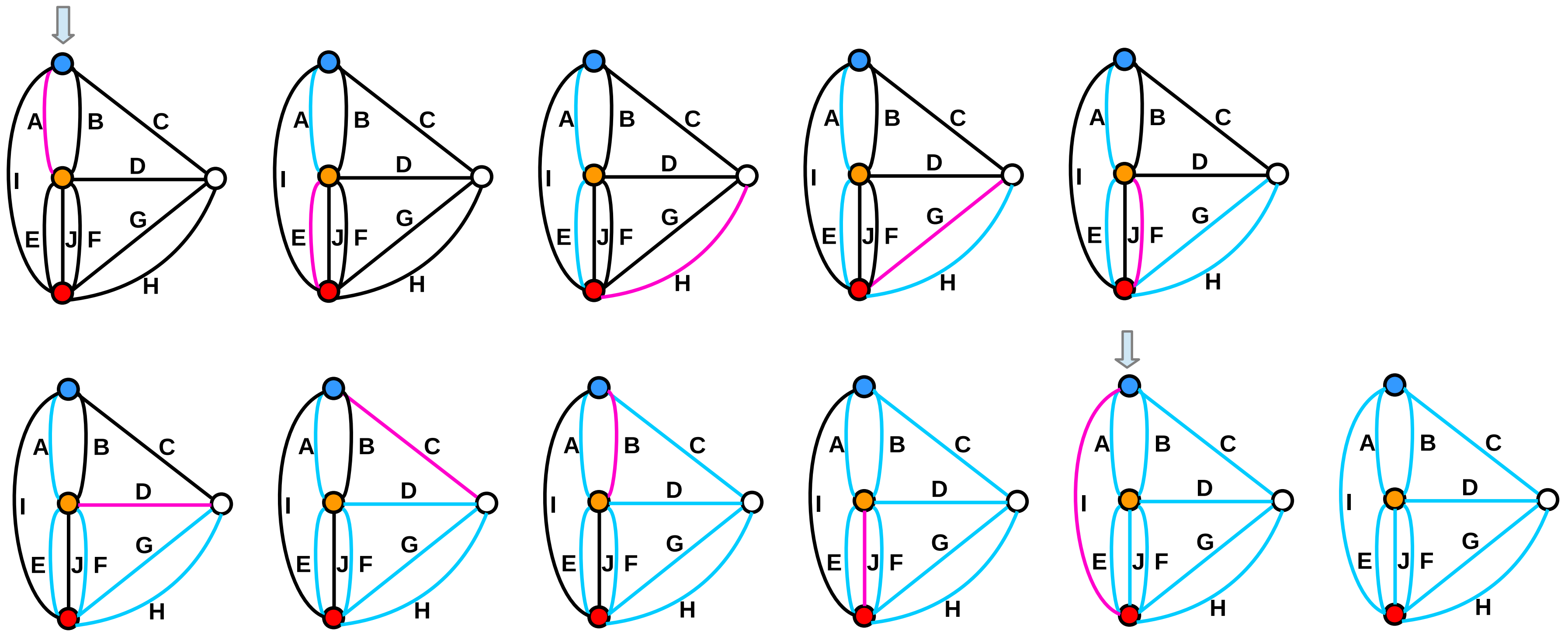


Eulerian Path

EHGFDCBAI

https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

10 bridges – Eulerian Cycle



Eulerian Cycle

AEHGFDCBJI

https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

Fleury's Algorithm

To find an Eulerian <u>path</u> or an Eulerian <u>cycle</u>:

1. make sure the graph has either **0** or **2 odd** vertices

2. if there are **0 odd** vertex, start <u>anywhere</u>.
   If there are **2 odd** vertices, start at one of the <u>two</u> <u>vertices</u>

3. follow edges one at a time.
   If you have a choice between a **bridge** and a **non-bridge**,
   Always <u>choose</u> the **non-bridge**

4. stop when you run out of edge

http://people.ku.edu/~jlmartin/courses/math105-F11/Lectures/chapter5-part2.pdf

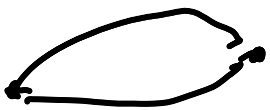
Bridges

A bridge edge

Removing a single edge from a connected graph
can make it disconnected

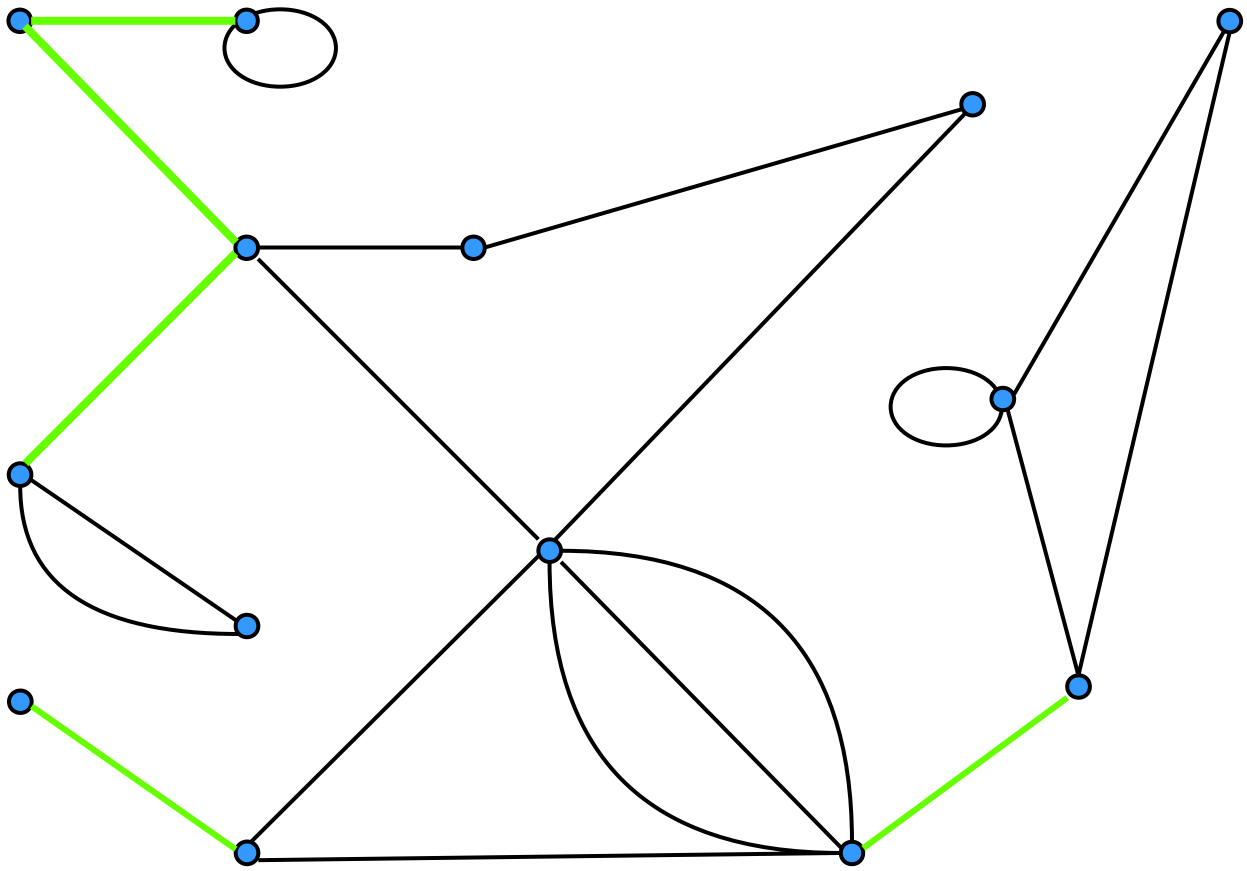
Non-bridge edges

Loops cannot be bridges 

Multiple edges cannot be bridges 

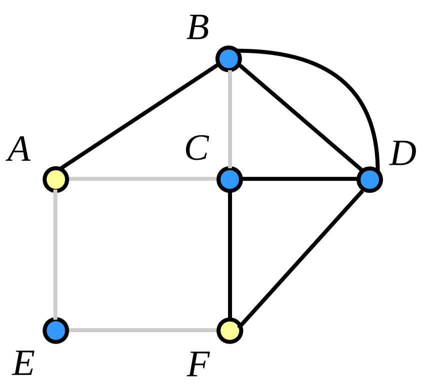
http://people.ku.edu/~jlmartin/courses/math105-F11/Lectures/chapter5-part2.pdf

Bridge examples in a graph

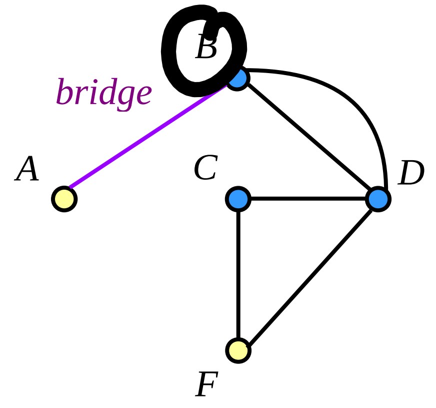


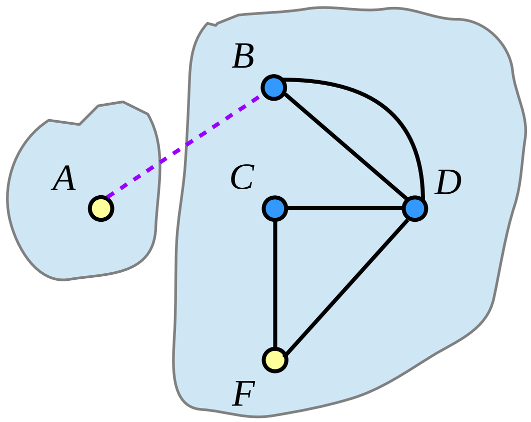
http://people.ku.edu/~jlmartin/courses/math105-F11/Lectures/chapter5-part2.pdf

Bridges must be avoided, if possible

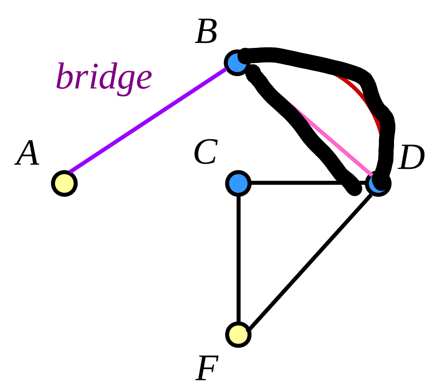


FEACB



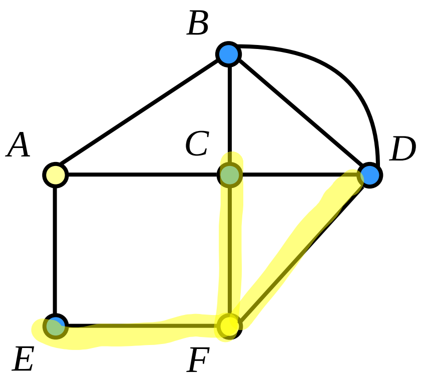


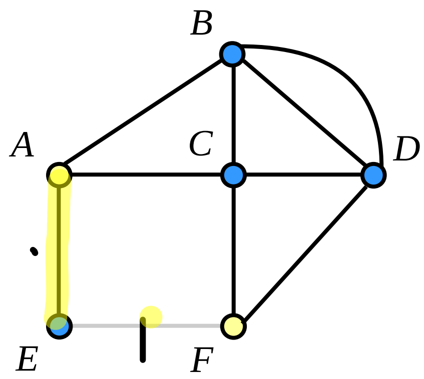
If there exists other choice other than a bridge
The bridge must not be chosen.



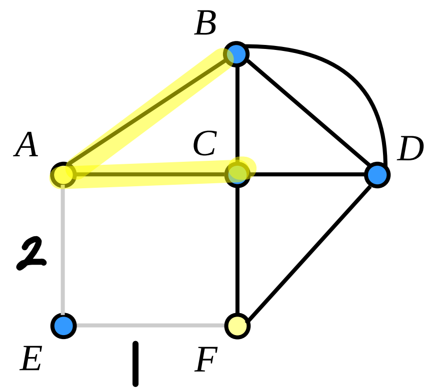
http://people.ku.edu/~jlmartin/courses/math105-F11/Lectures/chapter5-part2.pdf

Fleury's Algorithm (1)

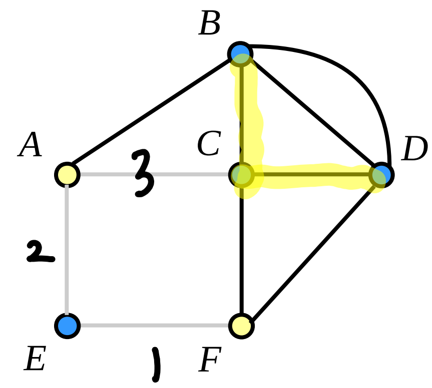




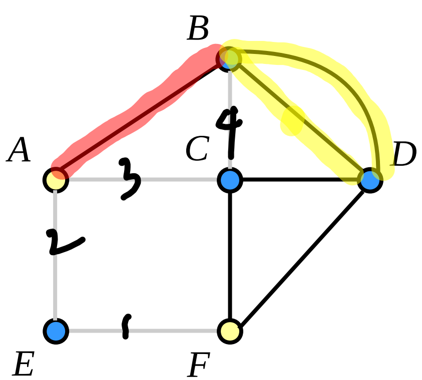
FE



FEA



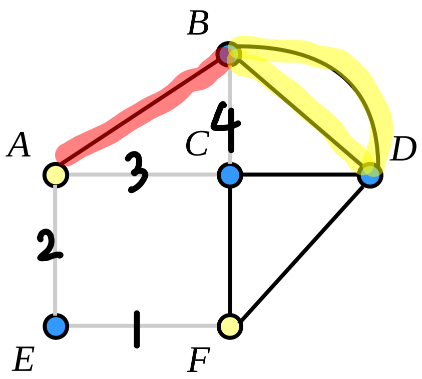
FEAC



FEACB

http://people.ku.edu/~jlmartin/courses/math105-F11/Lectures/chapter5-part2.pdf

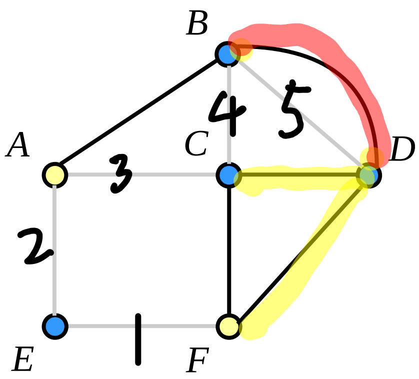
Fleury's Algorithm (2)



FEACB

BA : *bridge*

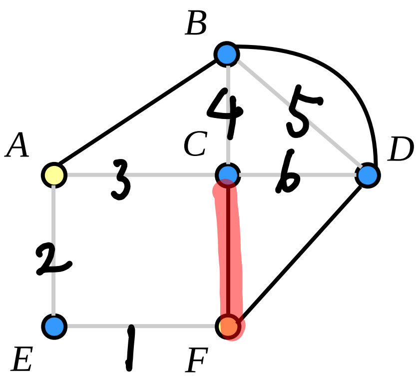
BD : *chosen*



FEACBD

DB : *bridge*

DC : *chosen*

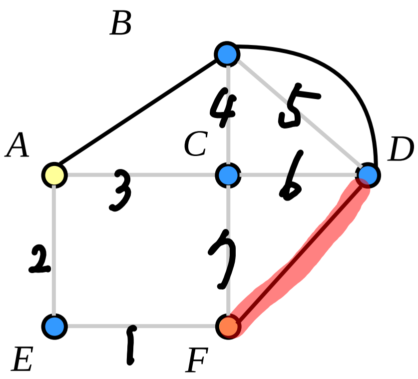


FEACBDC

CF : *bridge*

CF : *chosen*

no other choice



FEACBDCF

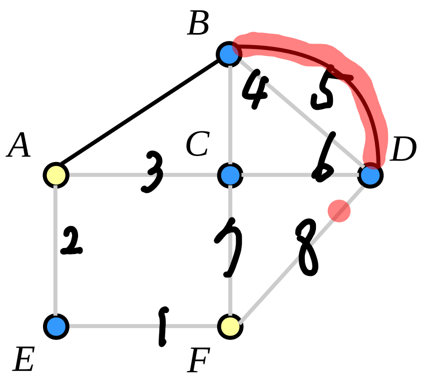
FD : *bridge*

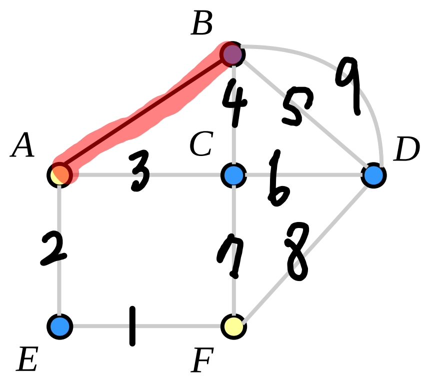
FD : *chosen*

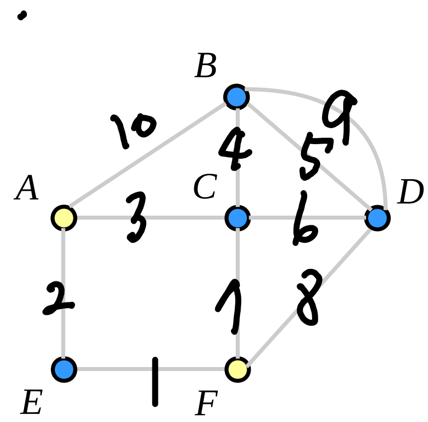
no other choice

http://people.ku.edu/~jlmartin/courses/math105-F11/Lectures/chapter5-part2.pdf

Fleury's Algorithm (3)







$FEACBDCFD$

DB: *bridge*

DB: *chosen*

no other choice

$FEACBDCFDB$

BA: *bridge*

BA: *chosen*

no other choice

edge names

F $\underbrace{1-2-3-4-5-6-7-8-9-10}$ A

E. path.

http://people.ku.edu/~jlmartin/courses/math105-F11/Lectures/chapter5-part2.pdf

References

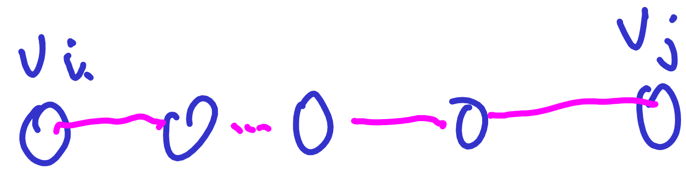
[1] http://en.wikipedia.org/
[2]

Hamiltonian Cycle (3A)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice and Octave.

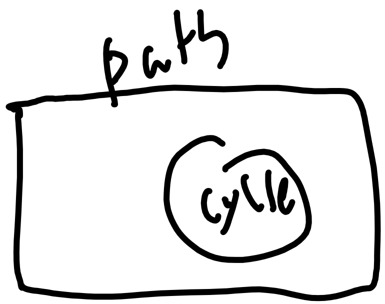
Hamiltonian Cycles

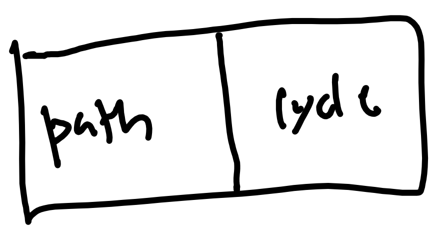


A Hamiltonian path is a path
in an undirected or directed graph
that visits **each vertex** exactly **once**.

A Hamiltonian cycle is
a Hamiltonian path that is a cycle.

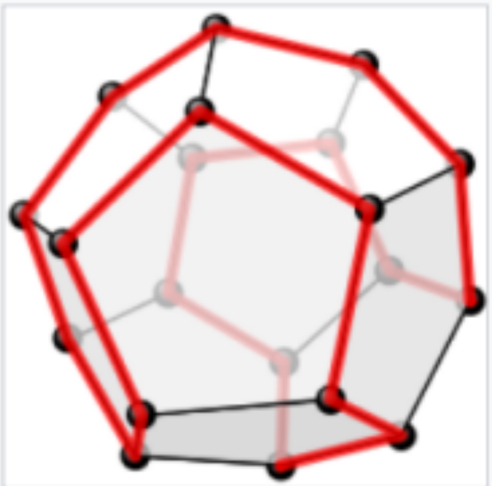
the Hamiltonian path problem is NP-complete.



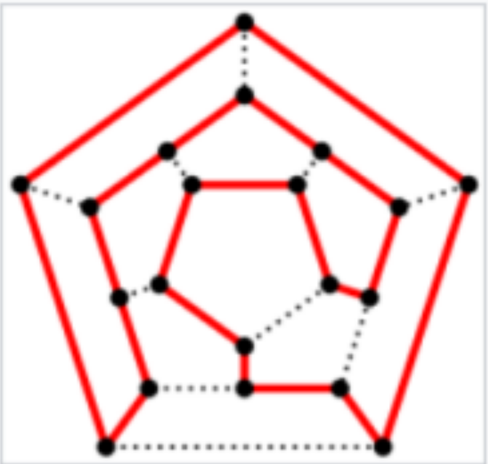


https://en.wikipedia.org/wiki/Hamiltonian_path

Hamiltonian Cycles



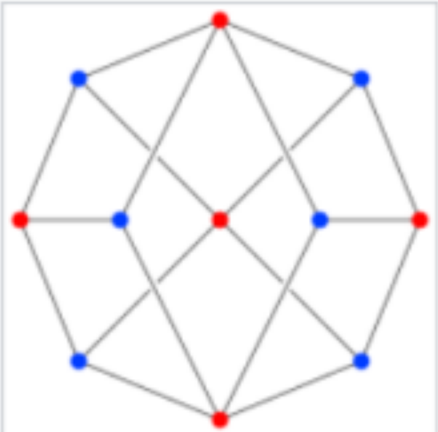
One possible Hamiltonian cycle through every vertex of a dodecahedron is shown in red – like all platonic solids, the dodecahedron is Hamiltonian



The above as a two-dimensional planar graph

https://en.wikipedia.org/wiki/Hamiltonian_path

Hamiltonian Cycles



The Herschel graph is the smallest possible polyhedral graph that does not have a Hamiltonian cycle.

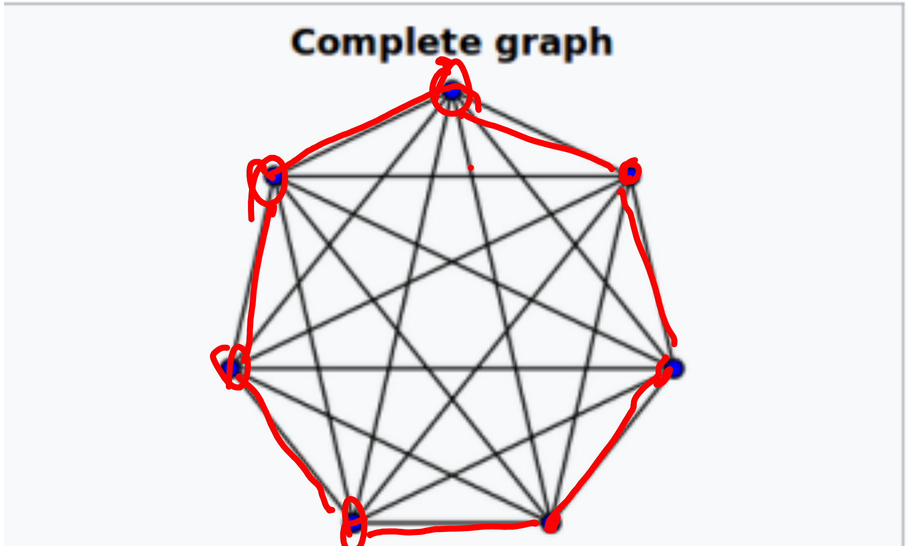
https://en.wikipedia.org/wiki/Hamiltonian_path

Hamiltonian Cycles

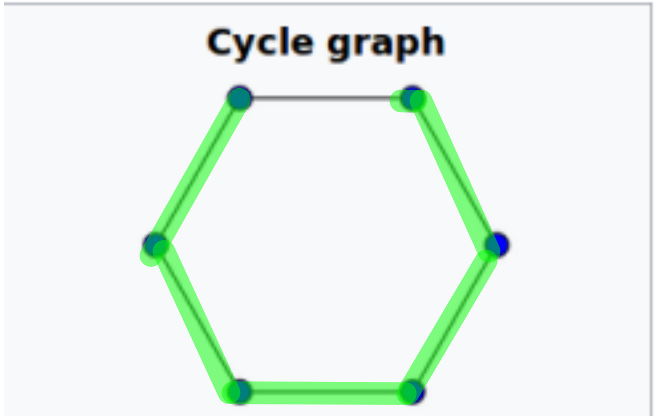
- a **complete graph** with more than two vertices is Hamiltonian
- every **cycle graph** is Hamiltonian
- every **tournament** has an odd number of Hamiltonian paths
- every **platonic solid**, considered as a graph, is Hamiltonian
- the **Cayley graph** of a finite **Coxeter** group is Hamiltonian

https://en.wikipedia.org/wiki/Hamiltonian_path

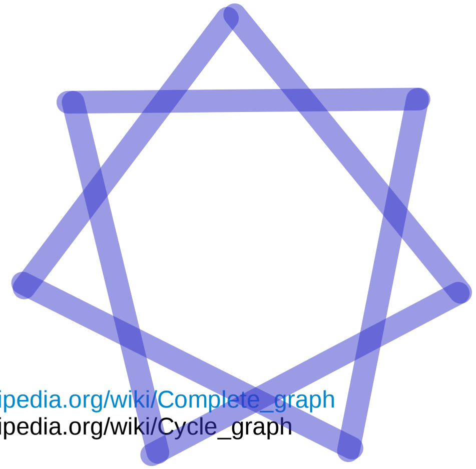
Complete Graphs and Cycle Graphs



K_7, a complete graph with 7 vertices

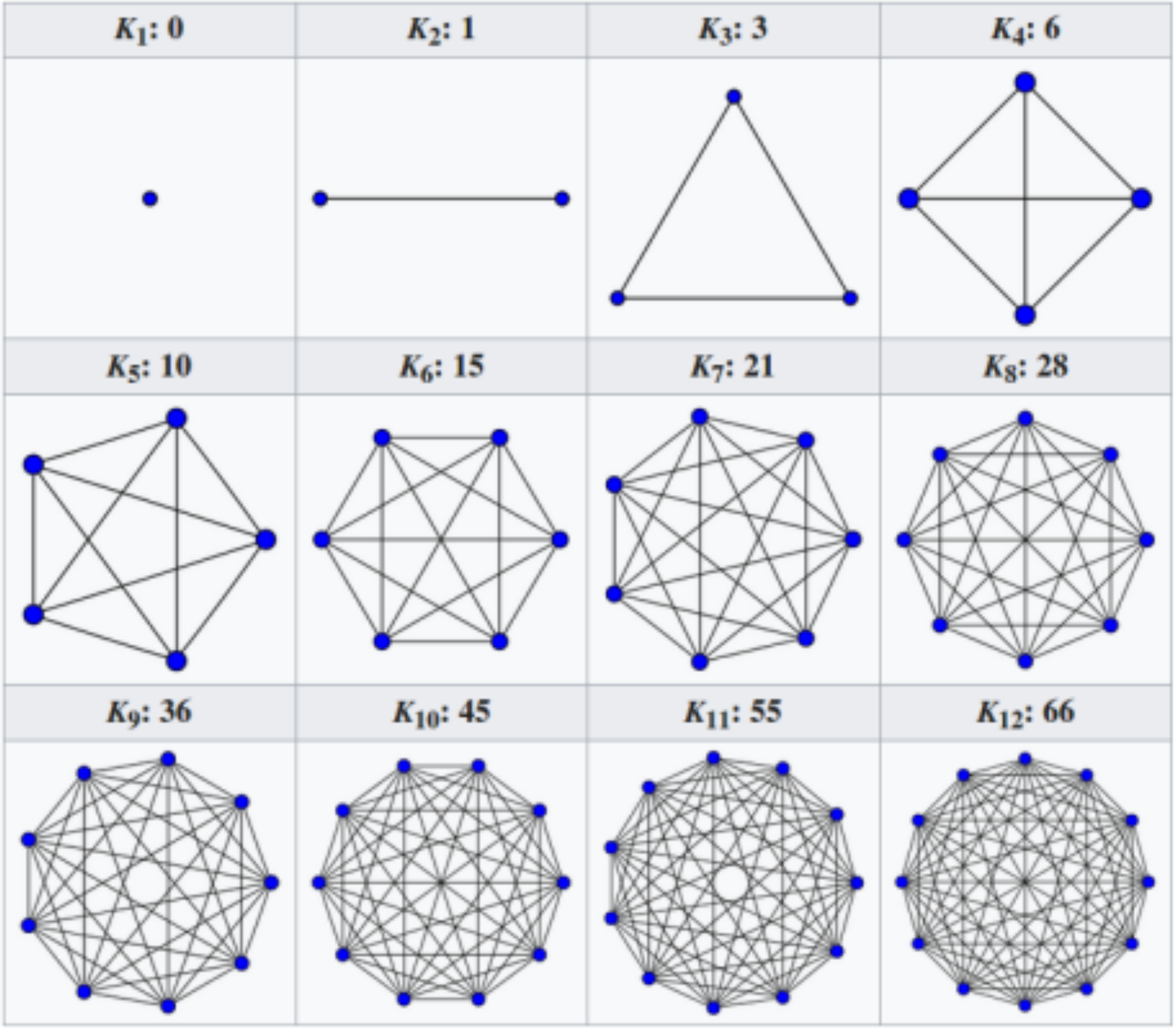


A cycle graph of length 6



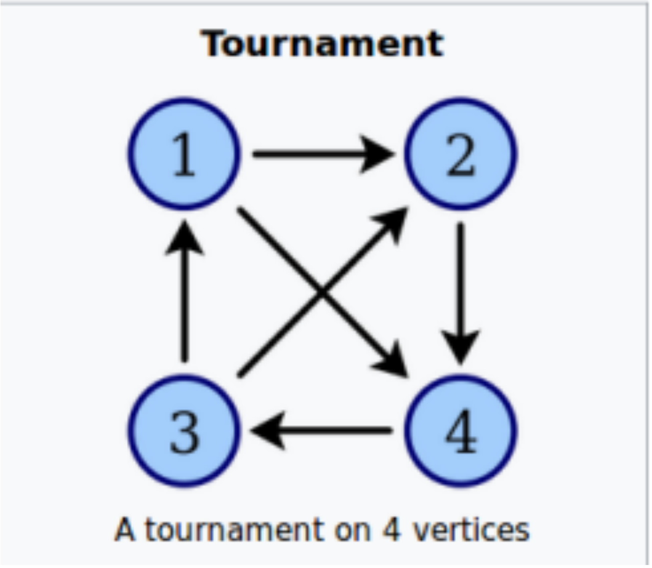
https://en.wikipedia.org/wiki/Complete_graph
https://en.wikipedia.org/wiki/Cycle_graph

Complete Graphs

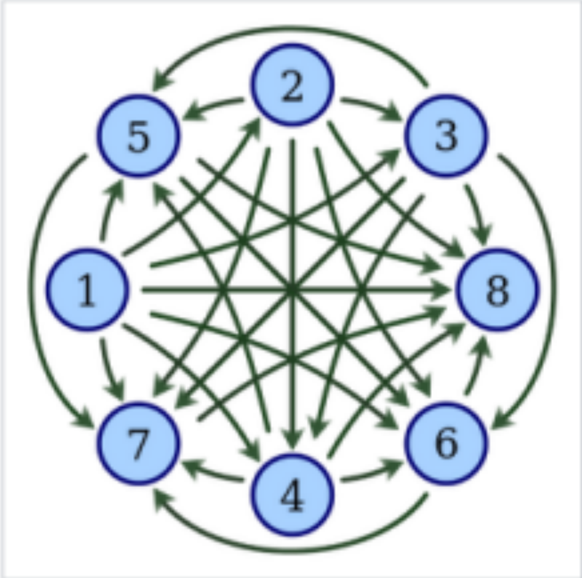


https://en.wikipedia.org/wiki/Complete_graph

Tournament Graphs



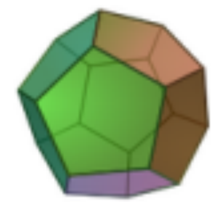
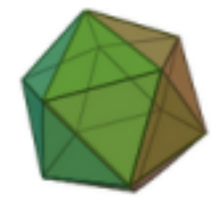
A tournament on 4 vertices



A transitive tournament on 8 vertices.

https://en.wikipedia.org/wiki/Tournament_(graph_theory

Platonic Solid Graphs

Tetrahedron	Cube	Octahedron	Dodecahedron	Icosahedron
Four faces	Six faces	Eight faces	Twelve faces	Twenty faces
				
(Animation)	(Animation)	(Animation)	(Animation)	(Animation)
(3D model)	(3D model)	(3D model)	(3D model)	(3D model)

https://en.wikipedia.org/wiki/Platonic_solid

Hamiltonian Cycles – Properties (1)

Any **Hamiltonian cycle** can be converted
to a **Hamiltonian path** by removing one of its edges,

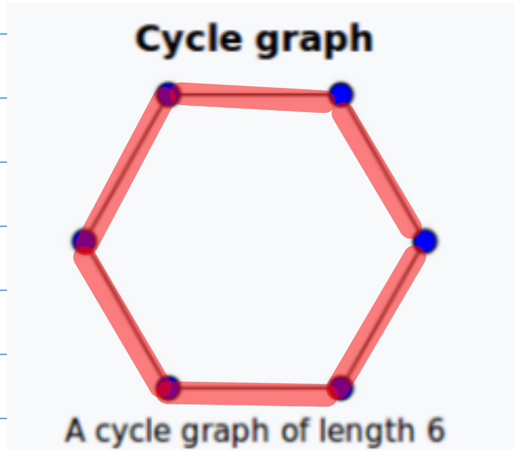
but a **Hamiltonian path** can be extended to
Hamiltonian cycle only if its endpoints are adjacent.

All **Hamiltonian graphs** are **biconnected**, but a
biconnected graph need not be Hamiltonian .

https://en.wikipedia.org/wiki/Hamiltonian_path

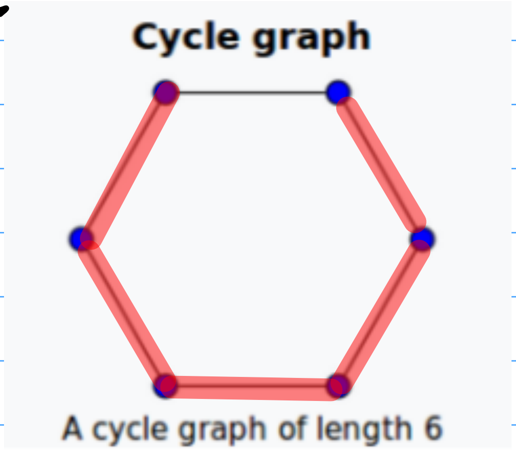
H. cycle $\longrightarrow$ H. Path

$\cancel{\longleftarrow}$ 일반적으로



이예제도

OK.



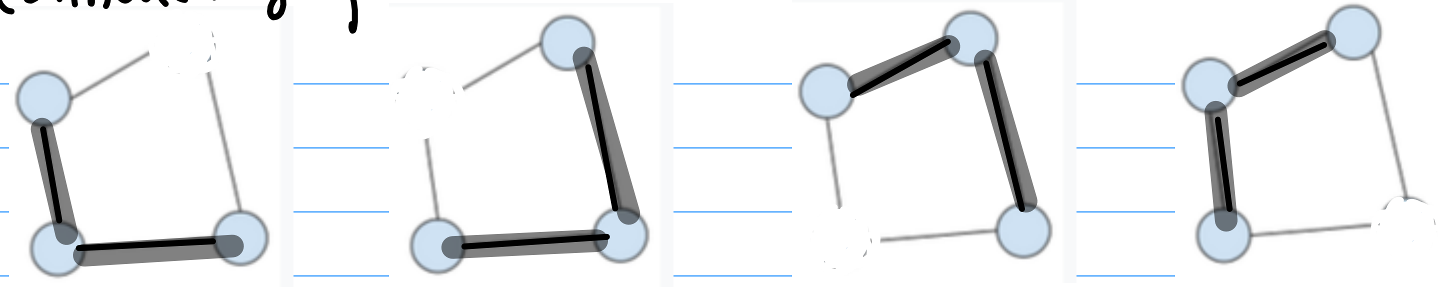
Biconnected Graph

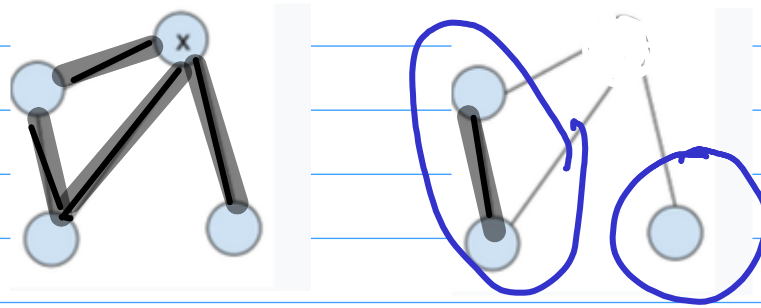
a biconnected graph is a connected and "nonseparable" graph, meaning that if any one **vertex** were to be removed, the graph will remain connected.

a biconnected graph has no articulation vertices.

The property of being **2-connected** is equivalent to **biconnectivity**, with the caveat that the complete graph of two vertices is sometimes regarded as biconnected but not 2-connected.

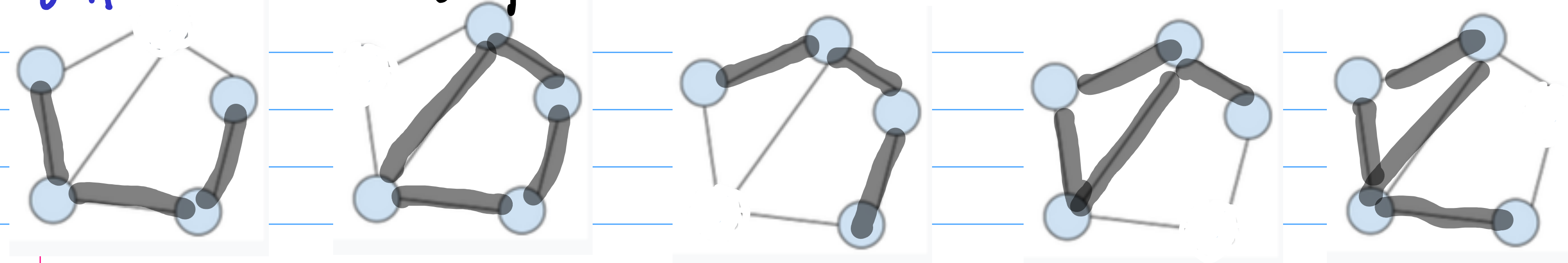
https://en.wikipedia.org/wiki/Biconnected_graph all connected graph

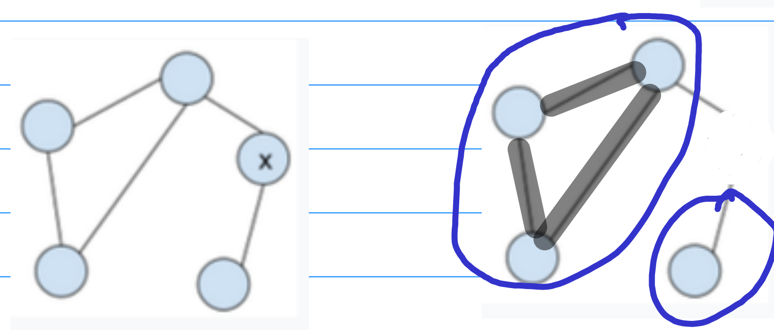




2 components (disconnected)

all connected graph



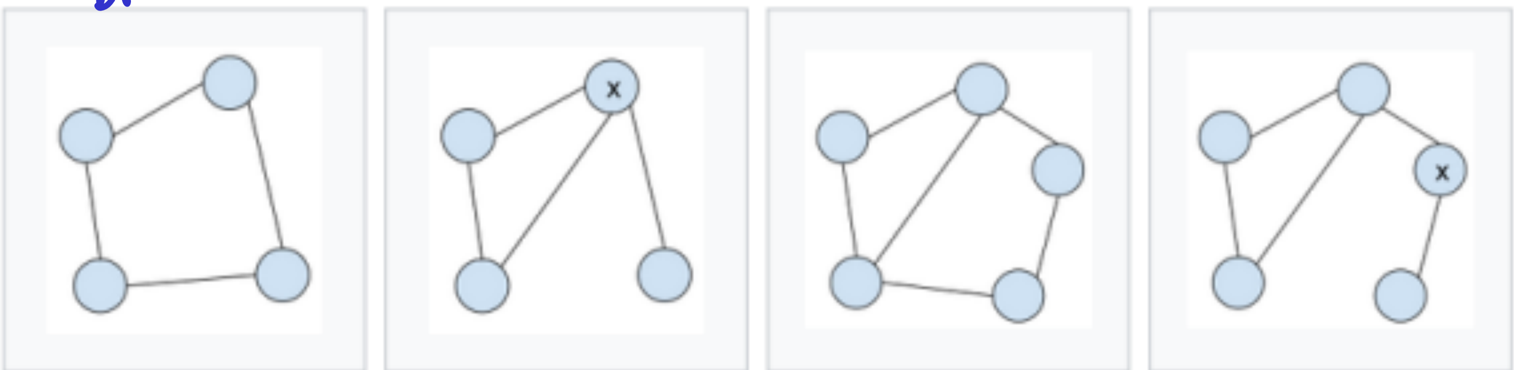


2 components (disconnected)

Biconnected Graph Examples

biconnect

biconnect



A biconnected graph on four vertices and four edges

A graph that is not biconnected. The removal of vertex x would disconnect the graph.

A biconnected graph on five vertices and six edges

A graph that is not biconnected. The removal of vertex x would disconnect the graph.

https://en.wikipedia.org/wiki/Biconnected_graph

Hamiltonian Cycles – Properties (2)

An **Eulerian graph** G :
a **connected** graph in which every **vertex** has **even degree** → E. Cycle

An **Eulerian graph** G necessarily has an **Euler path**,
a closed walk passing through each **edge** of G exactly **once**.

This **Eulerian path** corresponds to a **Hamiltonian cycle** in the **line graph** L(G), so the line graph of every **Eulerian graph** is **Hamiltonian**.

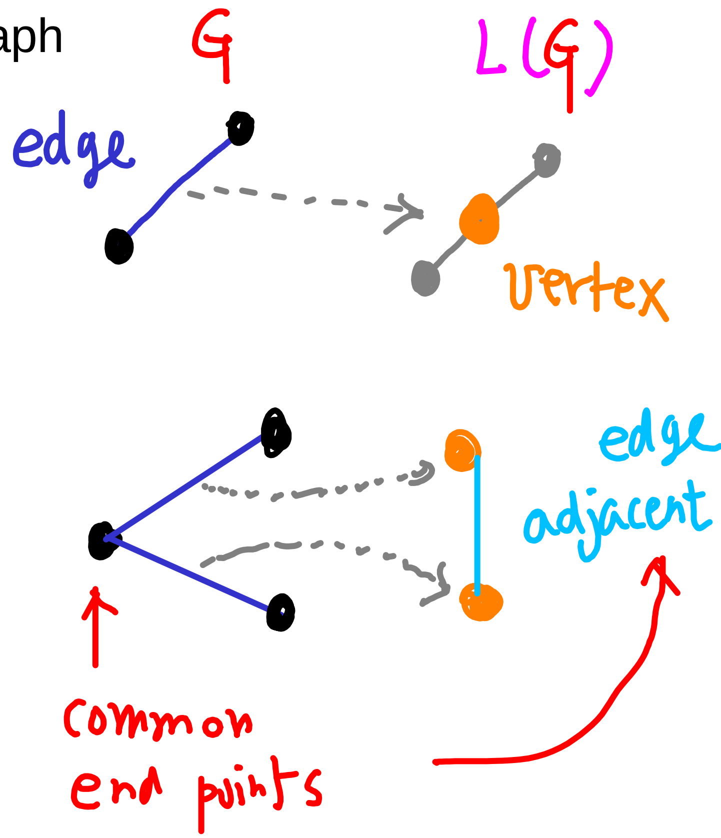
Line graphs may have other Hamiltonian cycles that do not correspond to Euler paths.

The **line graph** L(G) of every **Hamiltonian graph** G is itself **Hamiltonian**, regardless of whether the graph G is **Eulerian**.

https://en.wikipedia.org/wiki/Hamiltonian_path

Line Graphs

In the mathematical discipline of graph theory, the line graph of an undirected graph G is another graph L(G) that represents the adjacencies between edges of G.



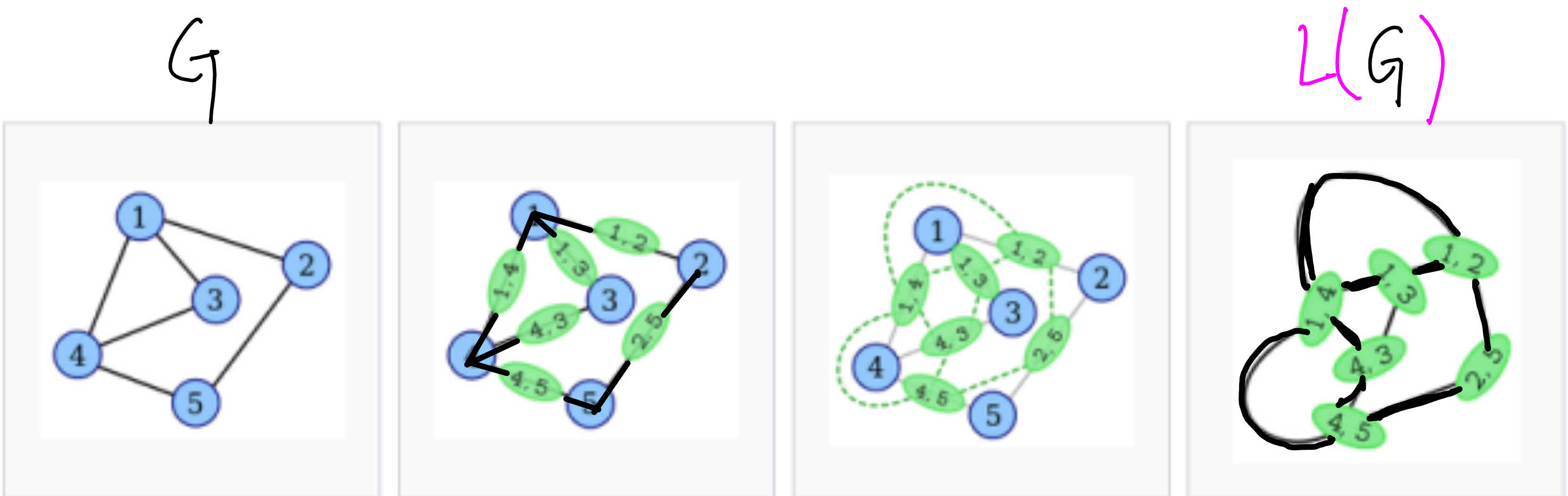
Given a graph G, its line graph L(G) is a graph such that

- each **vertex** of L(G) represents an **edge** of G; and
- two vertices of L(G) are **adjacent** if and only if their corresponding edges share a **common endpoint** ("are incident") in G.

That is, it is the **intersection graph** of the edges of G, representing each edge by the set of its two endpoints.

https://en.wikipedia.org/wiki/Line_graph

Line Graphs Examples



Graph G

Vertices in L(G) constructed from edges in G

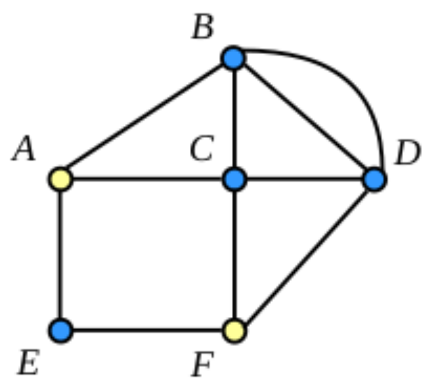
Added edges in L(G)

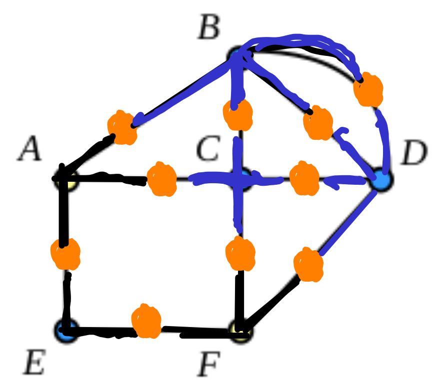
The line graph L(G)

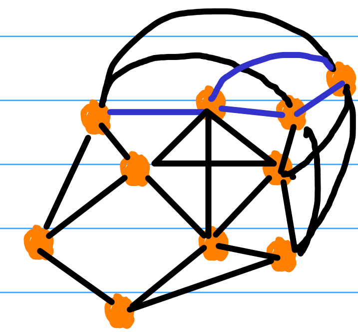
https://en.wikipedia.org/wiki/Line_graph

E. Path

Hamilton cycle





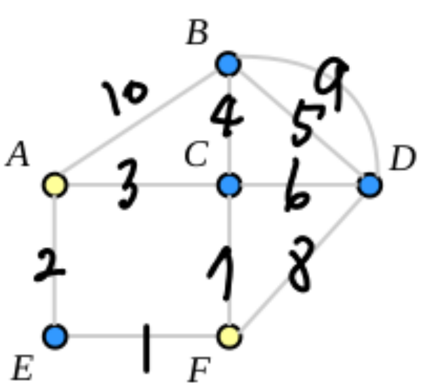


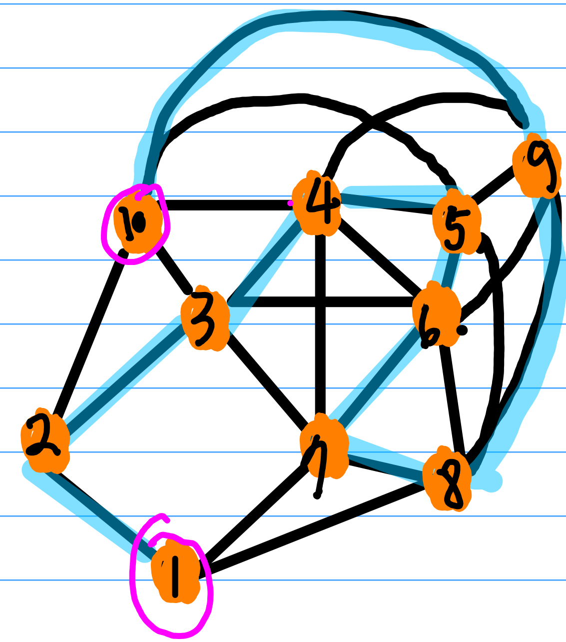
G

L(G)

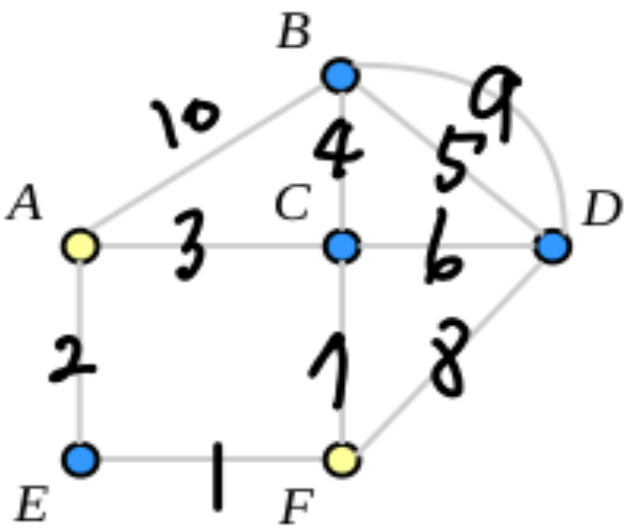
모든 edge
10개

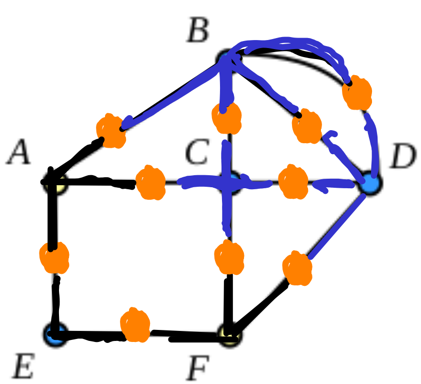
모든 vertex
10개

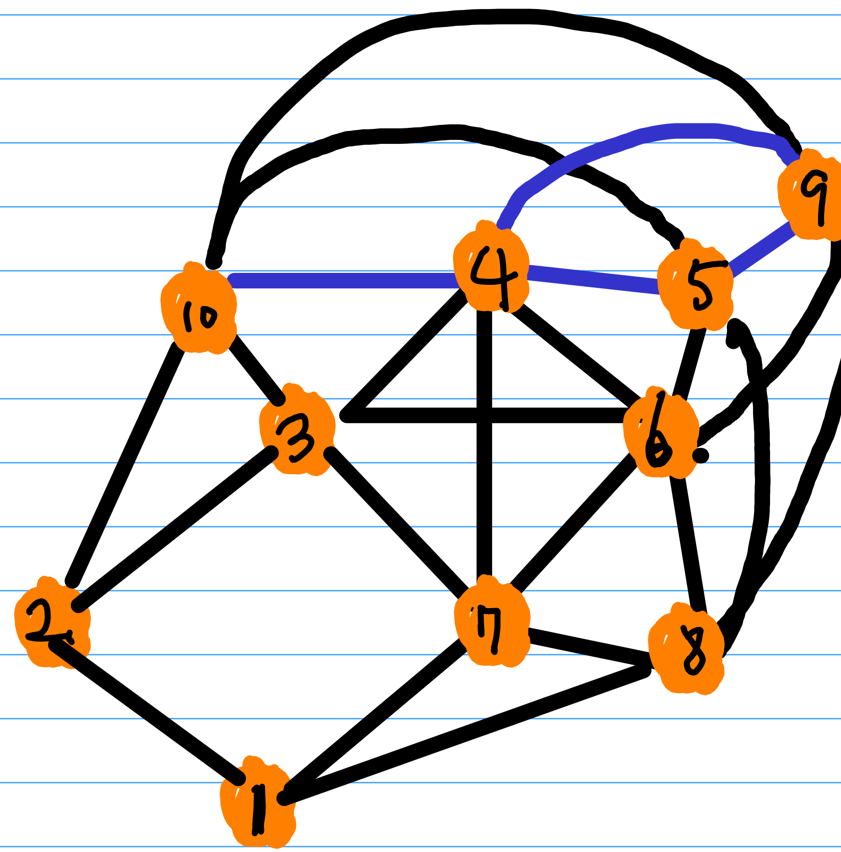


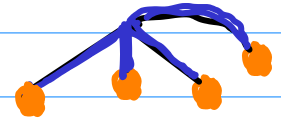


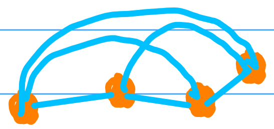
12345678910

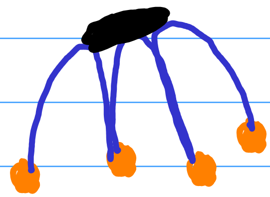

B
10
4
9
5
A
3
C
6
D
2
1
8
E
1
F

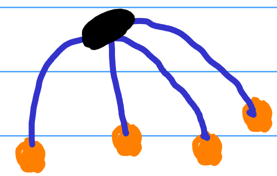

B
A
C
D
E
F

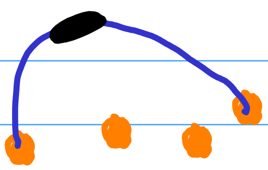

9
4
5
10
3
6
2
7
8
1











Hamiltonian Cycles – Properties (3)

A tournament (with more than two vertices) is Hamiltonian if and only if it is **strongly connected**.

The number of different Hamiltonian cycles
in a complete undirected graph on n vertices is $(n - 1)! / 2$
in a complete directed graph on n vertices is $(n - 1)!$.

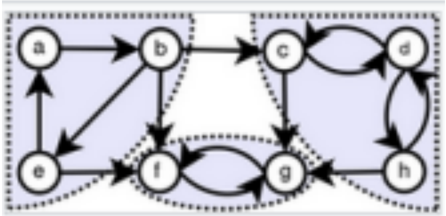
These counts assume that cycles that are the same apart from their starting point are not counted separately.

https://en.wikipedia.org/wiki/Hamiltonian_path

Strongly Connected Component

a directed graph is said to be **strongly connected** or **diconnected** if every **vertex** is reachable from every other **vertex.**

The **strongly connected components** or **diconnected components** of an arbitrary directed graph form a **partition** into **subgraphs** that are themselves **strongly connected**.



Graph with strongly connected components marked

https://en.wikipedia.org/wiki/Hamiltonian_path

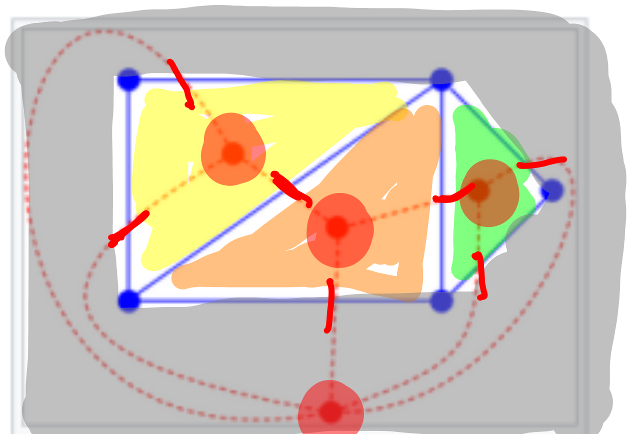
Dual Graph

the dual graph of a plane graph G is a graph that has a **vertex** for each **face** of G.

The dual graph has an **edge** whenever two **faces** of G are separated from each other by an **edge**,

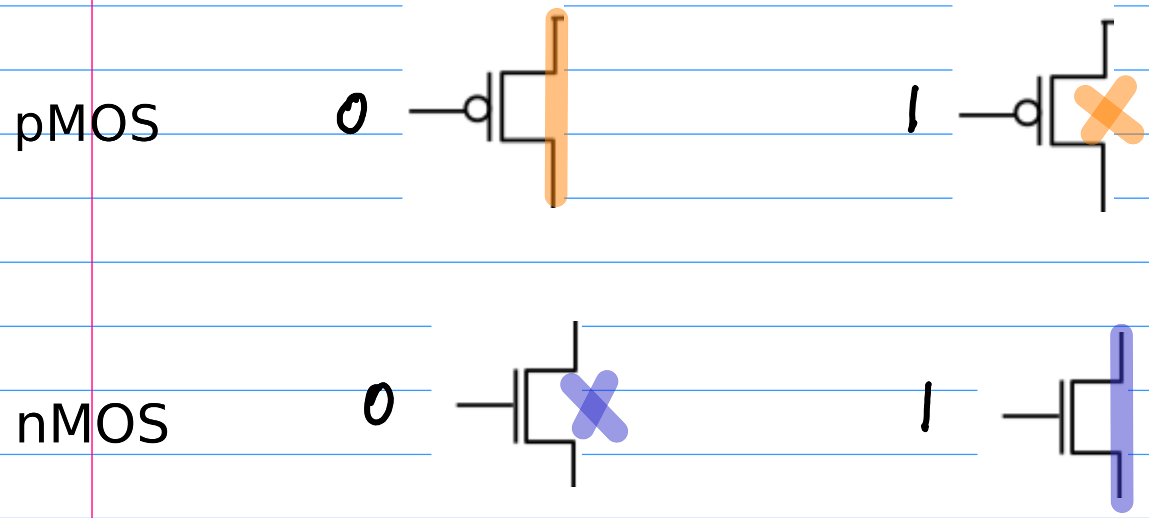
and a **self-loop** when the same **face** appears on both sides of an **edge**.

each **edge e** of G has a corresponding **dual edge**, whose endpoints are the **dual vertices** corresponding to the **faces** on either side of **e**.



The red graph is the dual graph of the blue graph, and *vice versa*.

https://en.wikipedia.org/wiki/Hamiltonian_path

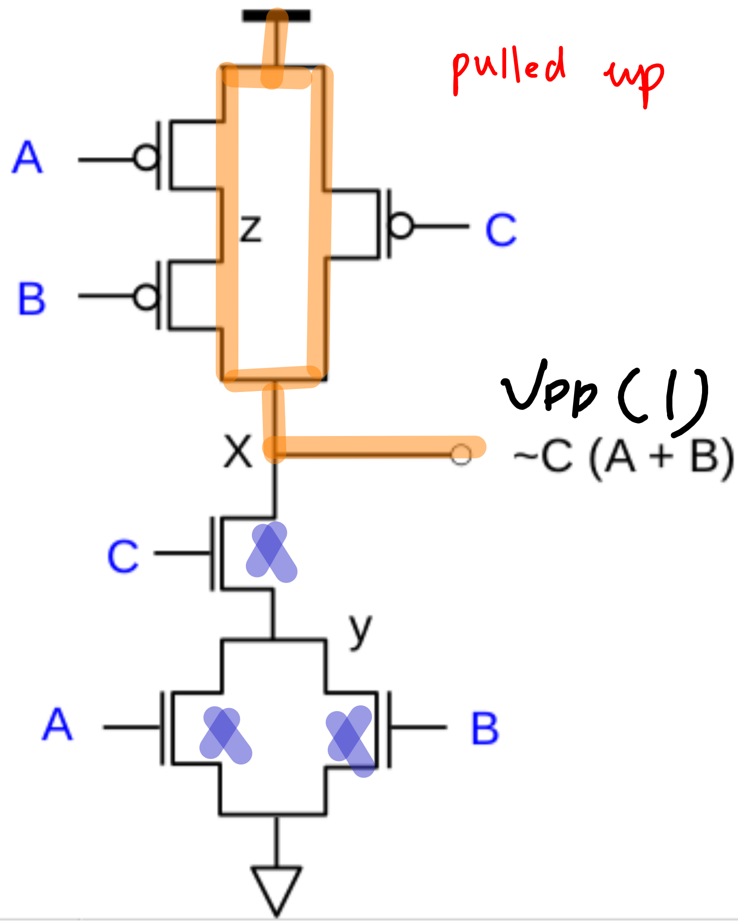


0 → 0V – L — GND

1 → 5V – H — V_{DD}

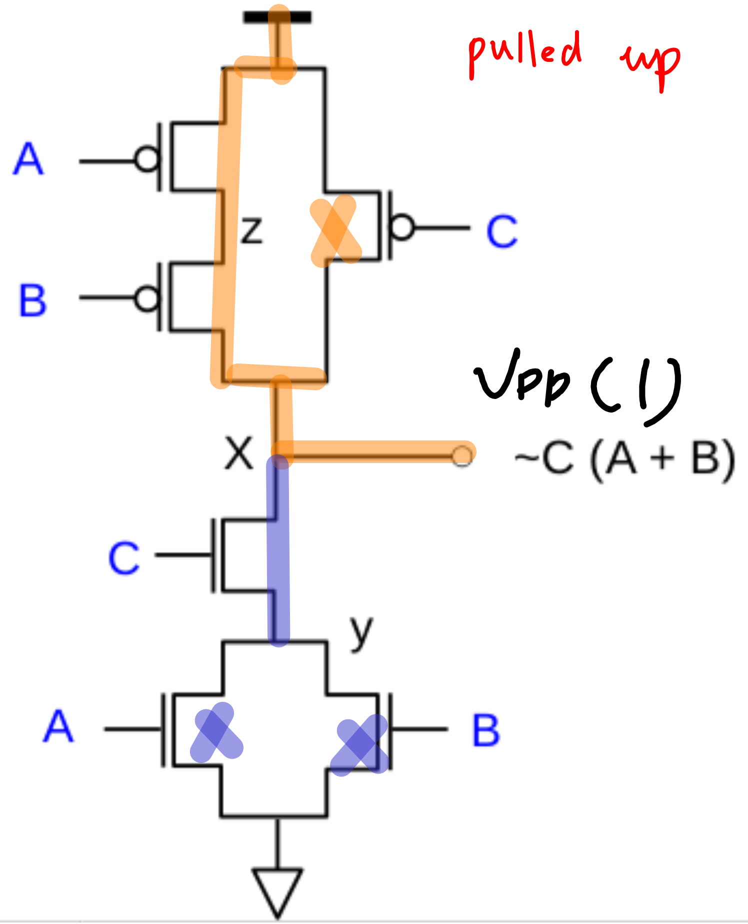
⑥

A	B	C	A+B	C(A+B)	~C(A+B)
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	0



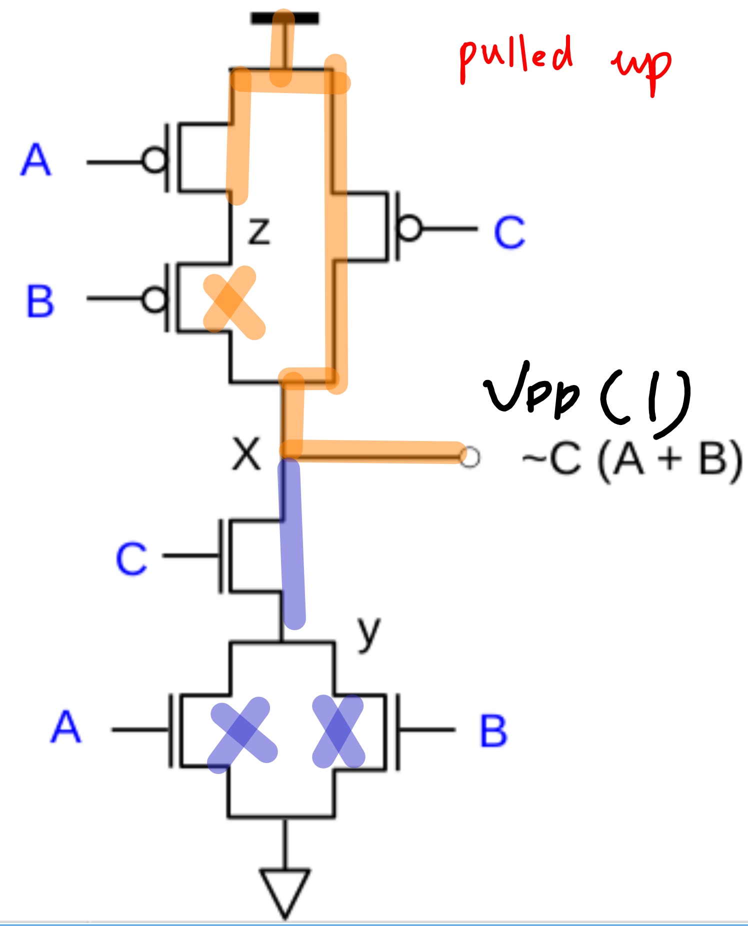
(i)

A	B	C	A+B	C(A+B)	~C(A+B)
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	0



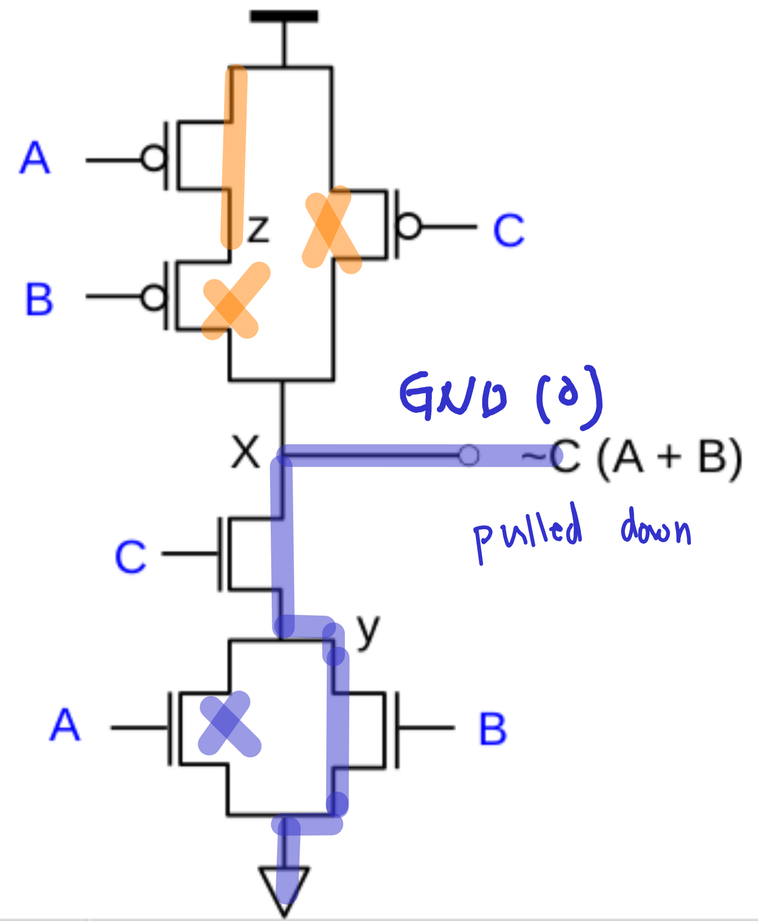
(2)

A	B	C	A+B	C(A+B)	~C(A+B)
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	0



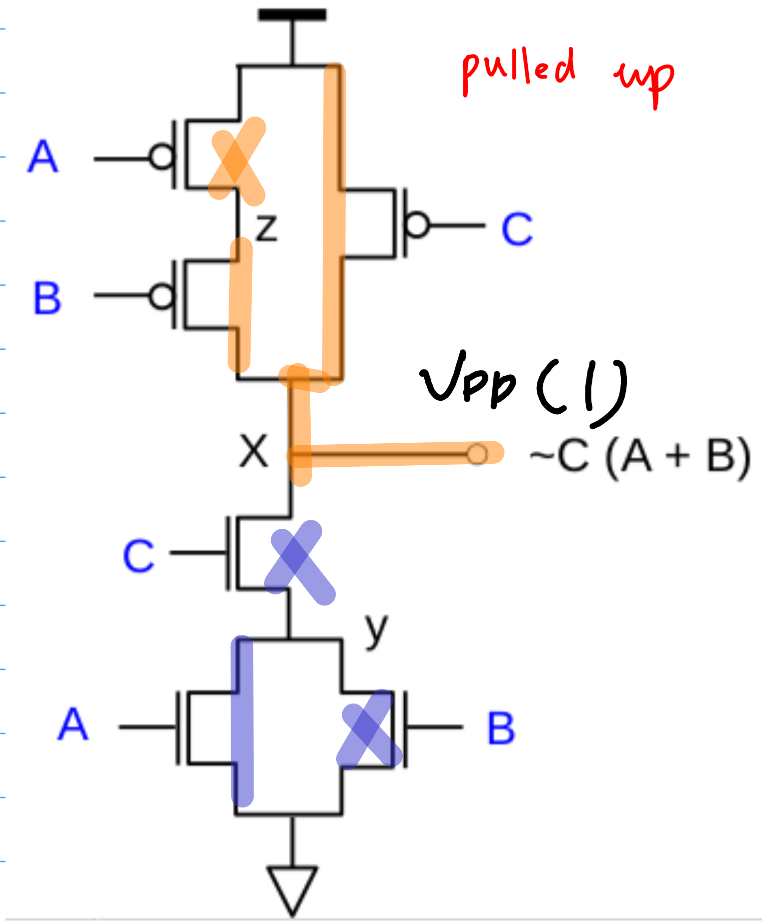
3

A	B	C	A+B	C(A+B)	~C(A+B)
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	0



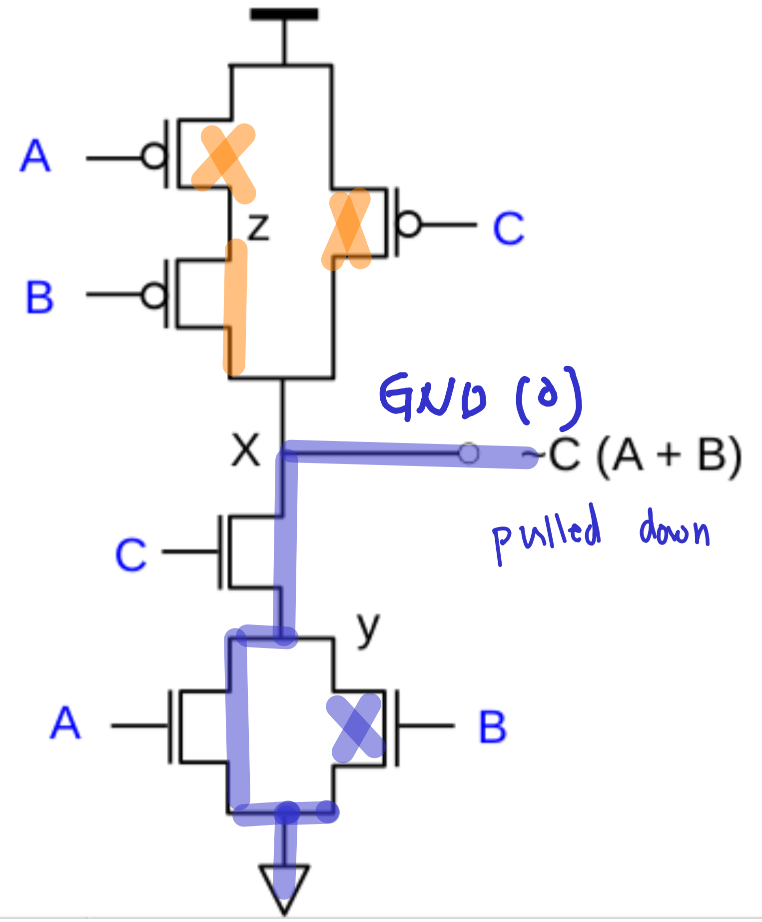
4

A	B	C	A+B	C(A+B)	~C(A+B)
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	0



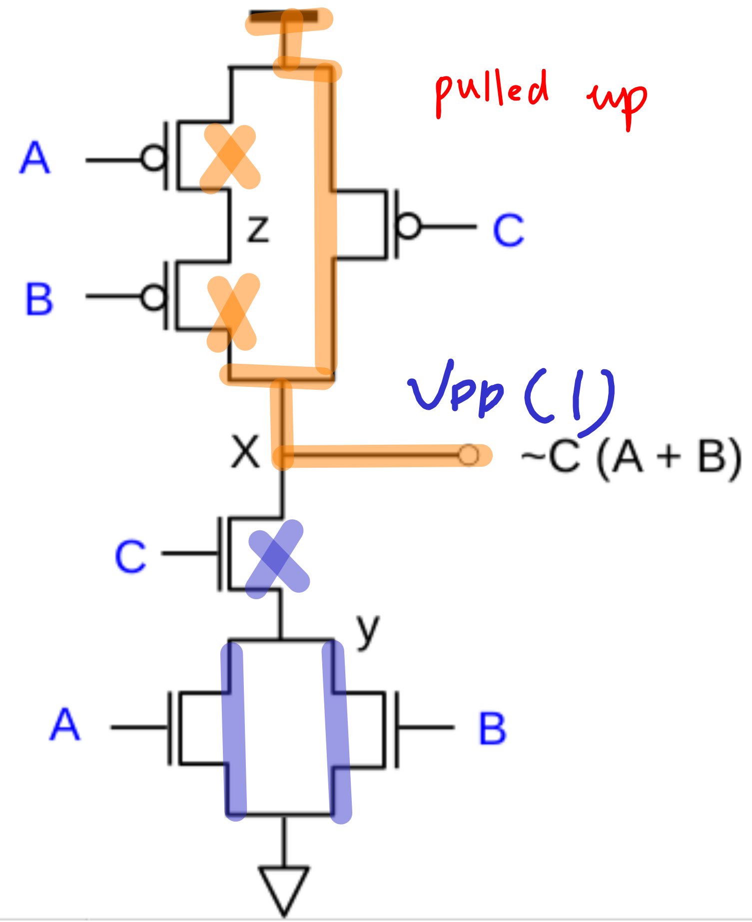
5

A	B	C	A+B	C(A+B)	~C(A+B)
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	0



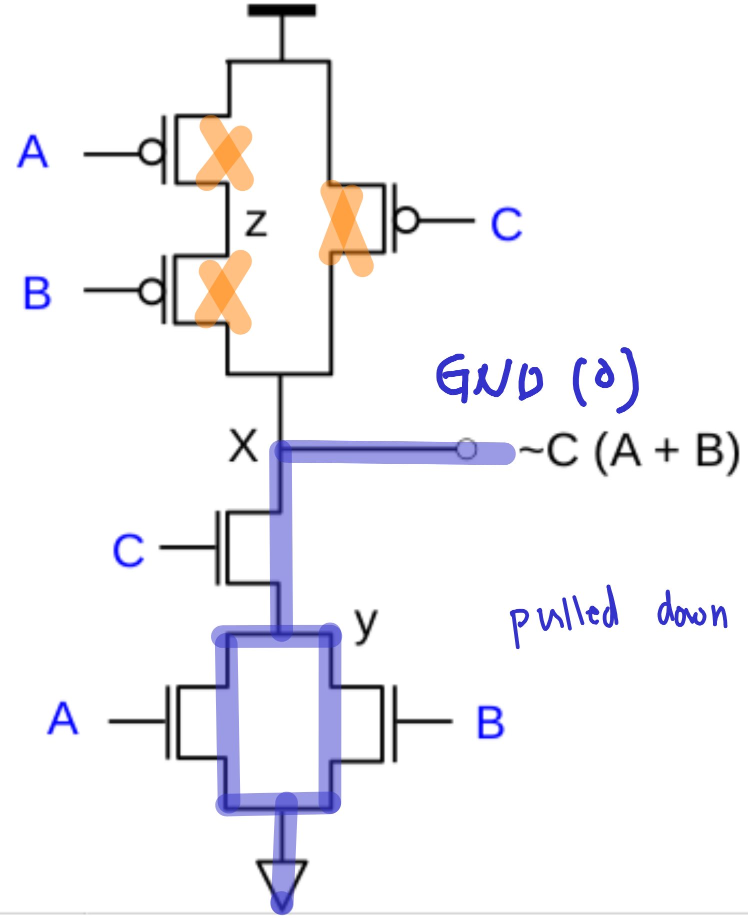
(6)

A	B	C	A+B	C(A+B)	~C(A+B)
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	0



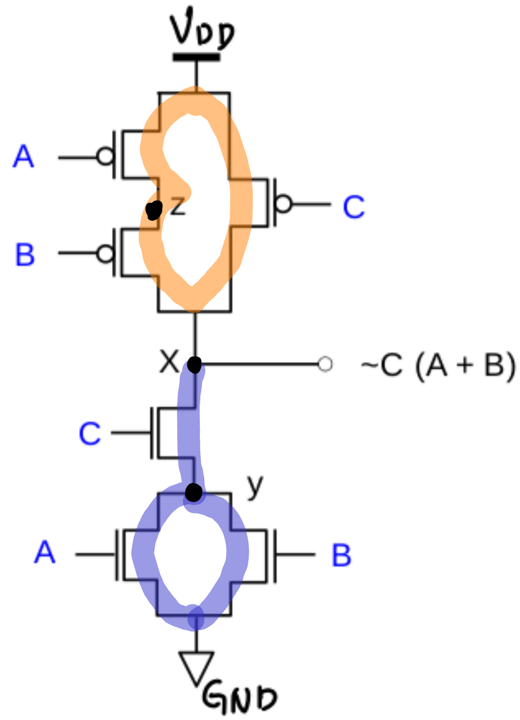
⑨

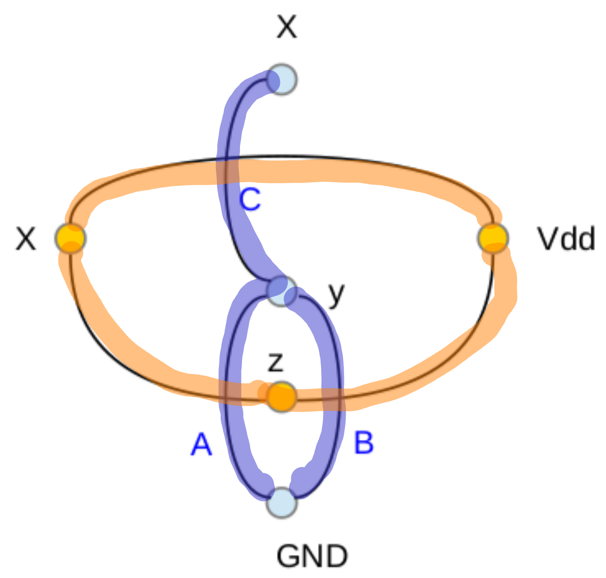
A	B	C	A+B	C(A+B)	~C(A+B)
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	0

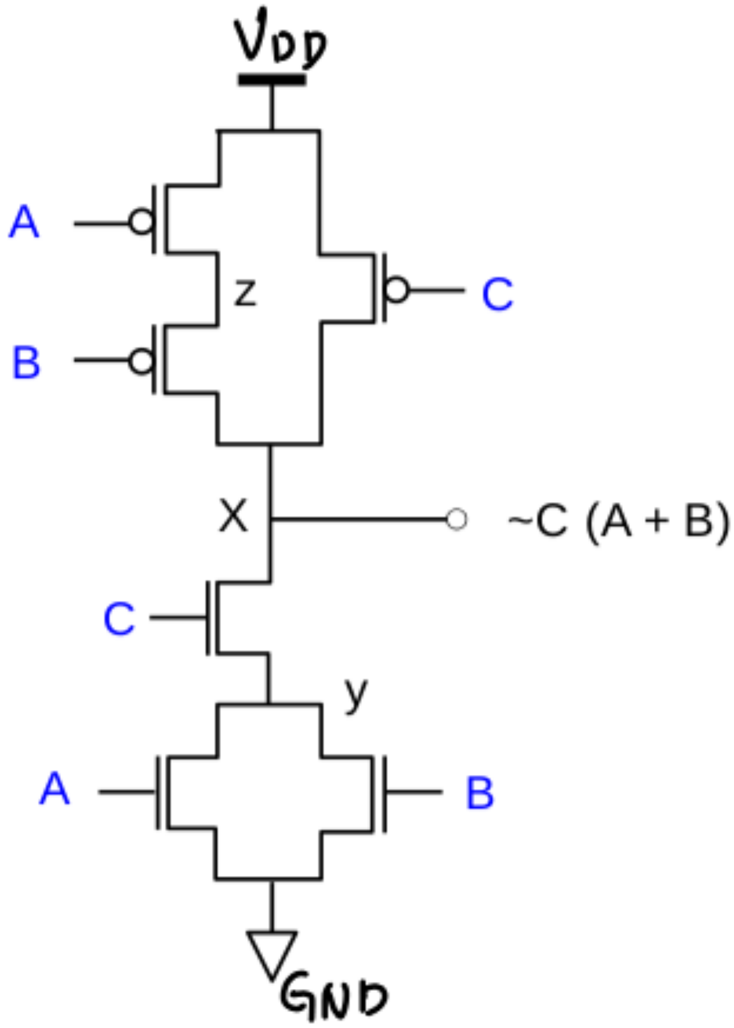


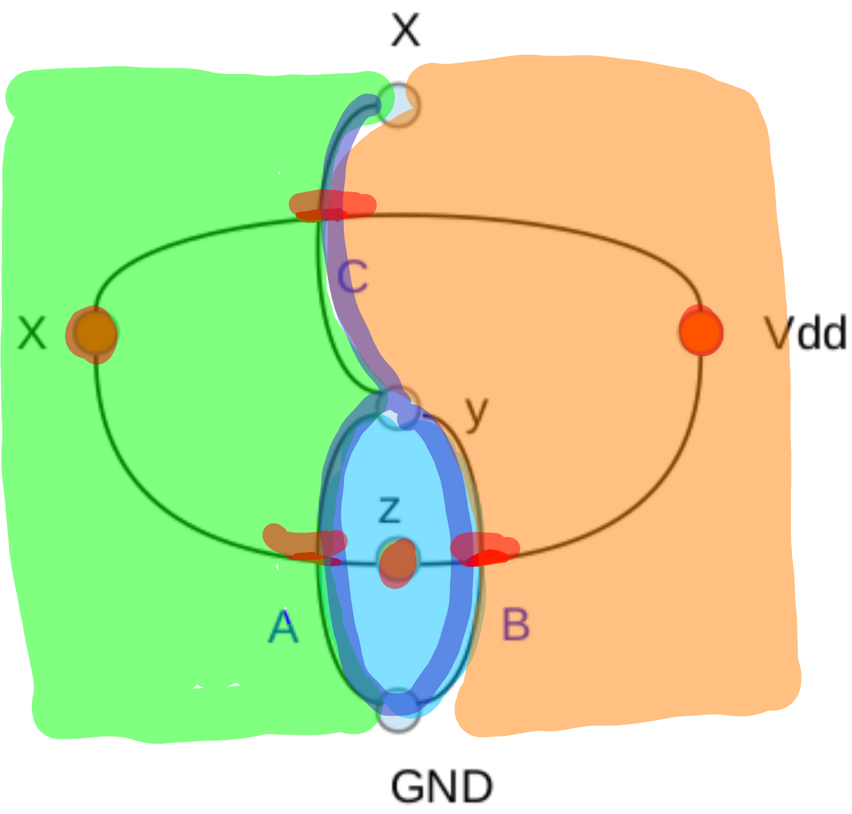
GND (0)

pulled down

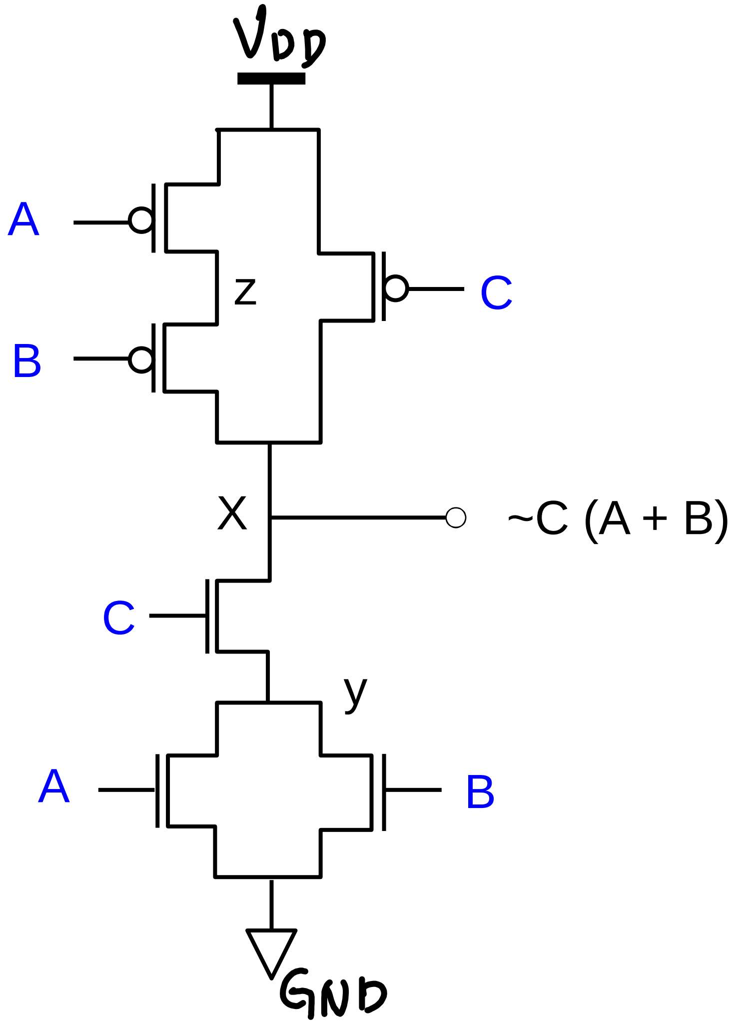

VDD
A
z
C
B
X
~C (A + B)
C
y
A
B
GND

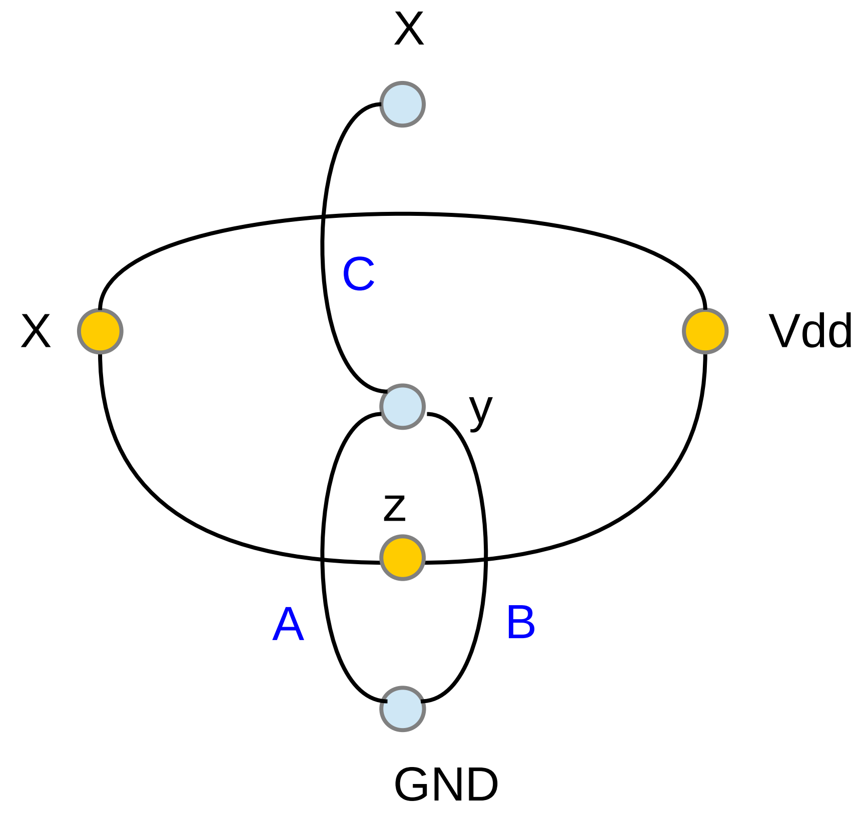

X
C
X
Vdd
y
z
A
B
GND


VDD
A
z
C
B
X
~C (A + B)
C
y
A
B
GND


X
C
X
Vdd
y
z
A
B
GND

Dual Graph





https://en.wikipedia.org/wiki/Hamiltonian_path

References

[1] http://en.wikipedia.org/
[2]

Minimum Spanning Tree (5A)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice and Octave.

Minimum Spanning Tree

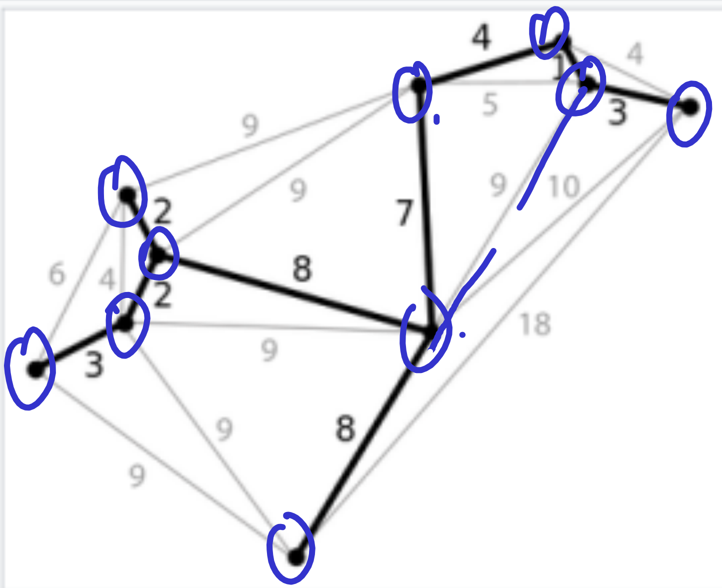
a **subset** of the **edges** of a connected, edge-weighted (un)directed graph that connects **all** the **vertices** together, without any **cycles** and with the **minimum** possible total edge **weight**.

a spanning tree whose sum of edge weights is as small as possible.

More generally, any edge-weighted undirected graph (not necessarily connected) has a minimum spanning **forest**, which is a **union** of the minimum spanning **trees** for its connected components.

https://en.wikipedia.org/wiki/Minimum_spanning_tree

Types of Shortest Path Problems



A planar graph and its minimum spanning tree. Each edge is labeled with its weight, which here is roughly proportional to its length.

tree loop X

cycle X

10 vertices

3+2+2+8+7+8+4+1+3

https://en.wikipedia.org/wiki/Minimum_spanning_tree

Properties (1)

Possible multiplicity
If there are **n vertices** in the graph,
then each spanning tree has **n−1 edges**.

Uniquenss
If each edge has a distinct weight
then there will be only one, unique minimum spanning tree.
this is true in many realistic situations

Minimum-cost subgraph
If the weights are positive, then a minimum spanning tree is in fact a minimum-cost subgraph connecting **all vertices**, since subgraphs containing cycles necessarily have more total weight.

https://en.wikipedia.org/wiki/Minimum_spanning_tree

Properties (2)

Cycle Property

For any **cycle C** in the graph, if the weight of an **edge e** of **C** is larger than the individual weights of all other **edges** of **C**, then this edge cannot belong to a MST.

Cut property

For any **cut C** of the graph, if the weight of an **edge e** in the **cut-set** of **C** is strictly smaller than the weights of all other edges of the **cut-set** of **C**, then this edge belongs to all MSTs of the graph.

https://en.wikipedia.org/wiki/Minimum_spanning_tree

Properties (3)

Minimum-cost edge

If the minimum cost **edge e** of a graph is unique, then this edge is included in any MST.

Contraction

If **T** is a **tree** of **MST edges**, then we can contract **T** into a single vertex while maintaining the invariant that the MST of the contracted graph plus T gives the MST for the graph before contraction.

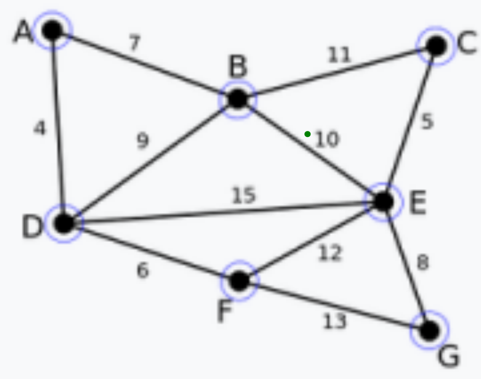
https://en.wikipedia.org/wiki/Minimum_spanning_tree

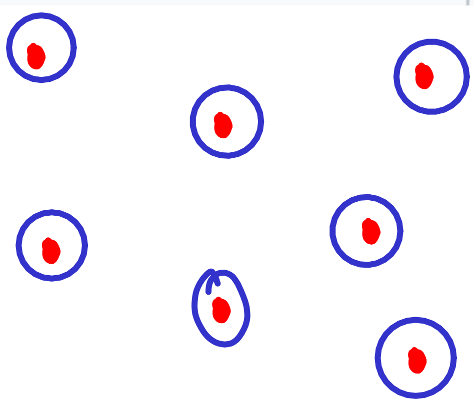
Borůvka's algorithm

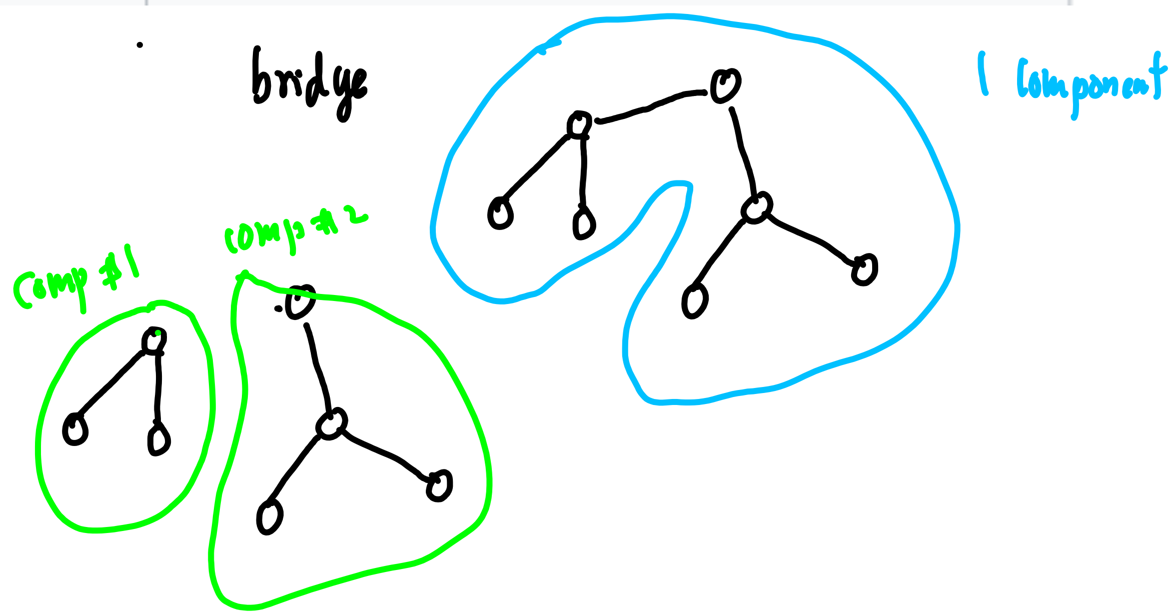
Input: A graph G whose edges have distinct weights
Initialize a forest **F** to be a set of one-vertex trees,
  one for each vertex of the graph.
While F has more than one component:
   Find the connected components of F and
   label each vertex of G by its component
   Initialize the cheapest edge for each component to "None"
   For each edge **uv** of **G**:
     If u and **v** have different component labels:
       If uv is <u>cheaper</u> than the <u>cheapest</u> edge
         for the component of **u**:
         Set **uv** as the <u>cheapest</u> edge for the component of **u**
       If uv is <u>cheaper</u> than the <u>cheapest</u> edge
         for the component of **v**:
         Set **uv** as the <u>cheapest</u> edge for the component of **v**
   For each component whose <u>cheapest</u> edge
     is not "None":
     Add its <u>cheapest</u> edge to **F**
Output: **F** is the minimum spanning forest of **G**.

https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm

Borůvka's algorithm examples (1)

Image	components	Description
	{A} {B} {C} {D} {E} {F} {G}	This is our original weighted graph. The numbers near the edges indicate their weight. Initially, every vertex by itself is a component (blue circles).

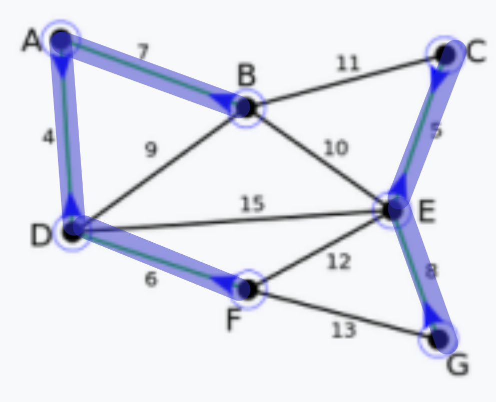


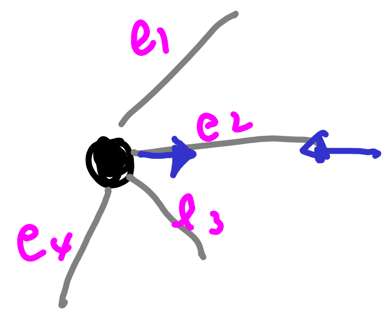


https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm

Borůvka's algorithm examples (2)

(2) connected components

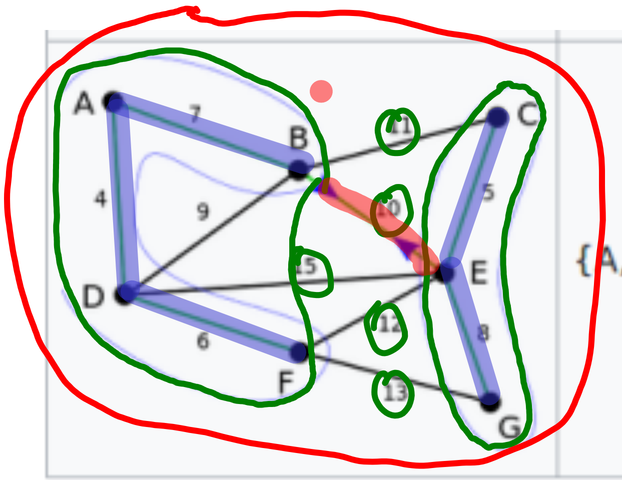
	{A,B,D,F} {C,E,G}	In the first iteration of the outer loop, the minimum weight edge out of every component is added. Some edges are selected twice (AD, CE). Two components remain.



min{e1 c2 e3 e4}

https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm

Borůvka's algorithm examples (3)

	{A,B,C,D,E,F,G}	In the second and final iteration, the minimum weight edge out of each of the two remaining components is added. These happen to be the same edge. One component remains and we are done. The edge BD is not considered because both endpoints are in the same component.

tree .... no cycle

spanning ... A, B, C, D, E, F, G

minimum ... 6 + 4 + 7 + 10 + 5 + 8 =

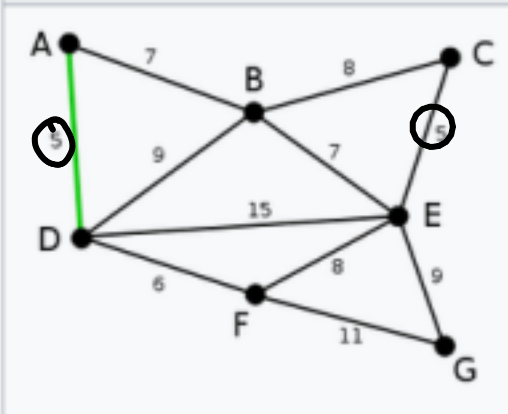
https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm

Kruskal's algorithm

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A ∪ {(u, v)}
      UNION(u, v)
return A
```

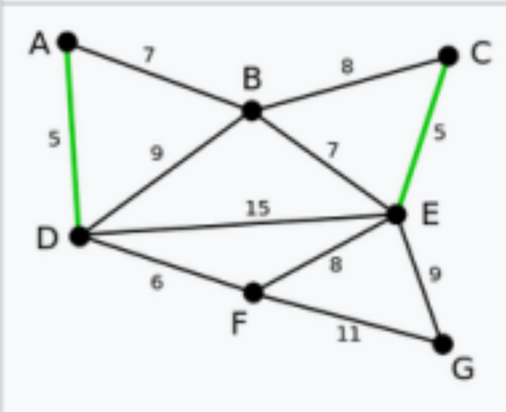
https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

Kruskal's algorithm examples (1)



5, 5, 6, 7, 7, 8, 8, 9, 9, 11, 15

AD and **CE** are the shortest edges, with length 5, and **AD** has been arbitrarily chosen, so it is highlighted.

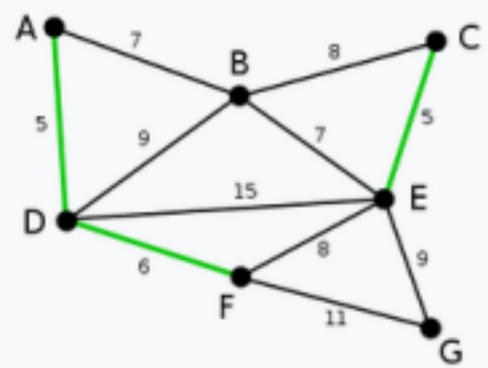


5, 5, 6, 7, 7, 8, 8, 9, 9, 11, 15

CE is now the shortest edge that does not form a cycle, with length 5, so it is highlighted as the second edge.

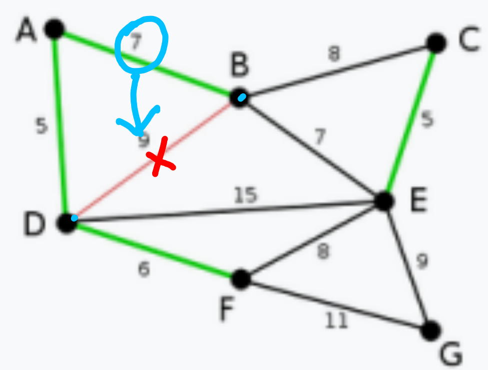
https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

Kruskal's algorithm examples (2)



5, 5, 6, 7, 7, 8, 8, 9, 9, 11, 15

The next edge, **DF** with length 6, is highlighted using much the same method.

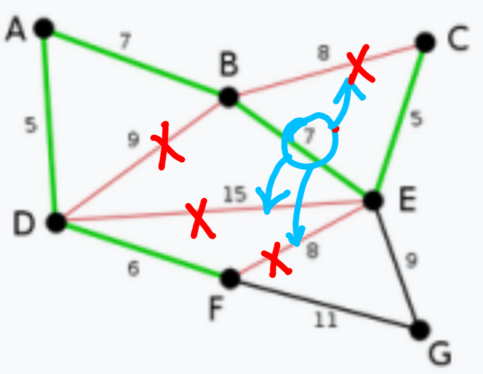


5, 5, 6, 7, 7, ~~8~~, 8, 9, 9, 11, 15

The next-shortest edges are **AB** and **BE**, both with length 7. **AB** is chosen arbitrarily, and is highlighted. The edge **BD** has been highlighted in red, because there already exists a path (in green) between **B** and **D**, so it would form a cycle (**ABD**) if it were chosen.

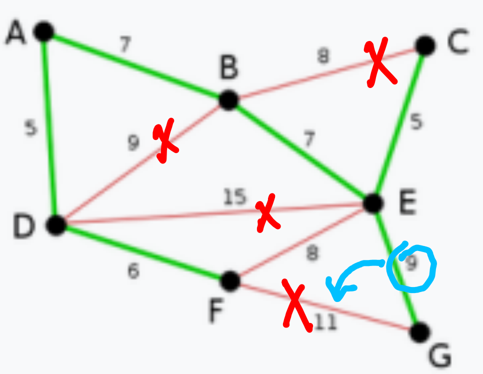
https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

Kruskal's algorithm examples (3)



5, 5, 6, 7, 7, ~~8~~, ~~8~~, ~~8~~, 9, 11, ~~15~~

The process continues to highlight the next-smallest edge, **BE** with length 7. Many more edges are highlighted in red at this stage: **BC** because it would form the loop **BCE**, **DE** because it would form the loop **DEBA**, and **FE** because it would form **FEBAD**.



5, 5, 6, 7, 7, ~~8~~, ~~8~~, ~~8~~, 9, 11, ~~15~~

Finally, the process finishes with the edge **EG** of length 9, and the minimum spanning tree is found.

https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

Prim's algorithm

a greedy algorithm that finds a minimum spanning tree
for a weighted undirected graph.

operates by building this tree one vertex at a time,
from an arbitrary starting vertex,
at each step adding the cheapest possible connection
from the tree to another vertex.

1. Initialize a tree with a single vertex,
   chosen arbitrarily from the graph.
2. Grow the tree by one edge:
   of the edges that connect the tree to vertices
   not yet in the tree, find the minimum-weight edge,
   and transfer it to the tree.
3. Repeat step 2 (until all vertices are in the tree).

https://en.wikipedia.org/wiki/Prim%27s_algorithm

1. Associate with each vertex **v** of the graph
   a number **C[v]** (the cheapest cost of a connection to v)
   and an edge **E[v]** (the cheapest edge).
   Initial values: C[v] = +∞, E[v] = flag for no connection

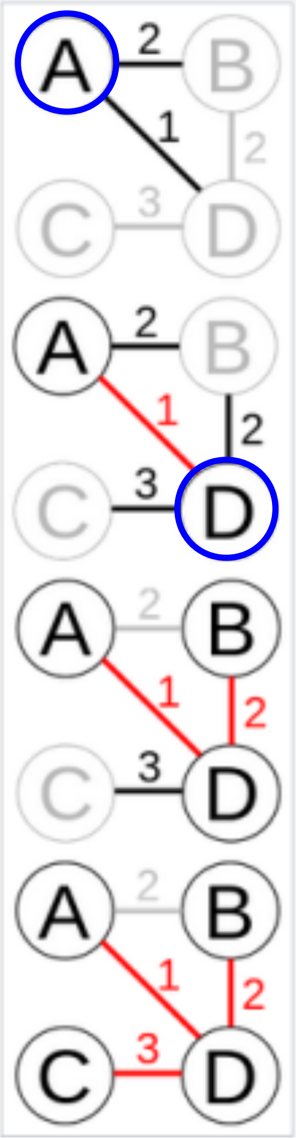
2. Initialize an empty **forest F** and a **set Q** of **vertices**
   that have not yet been included in **F**

3. Repeat the following steps until **Q** is empty:
   a. Find and remove a vertex **v** from **Q**
      having the minimum possible value of **C[v]**
   b. Add **v** to **F** and, if **E[v]** is not the special flag value,
      also add E[v] to F
   c. Loop over the edges **vw** connecting **v** to other
      vertices **w**. For each such edge, if w still belongs to Q
      and **vw** has smaller weight than **C[w]**,
      perform the following steps:
      I) Set **C[w]** to the cost of edge **vw**
      II) Set **E[w]** to point to edge **vw**.

Return F

https://en.wikipedia.org/wiki/Prim%27s_algorithm

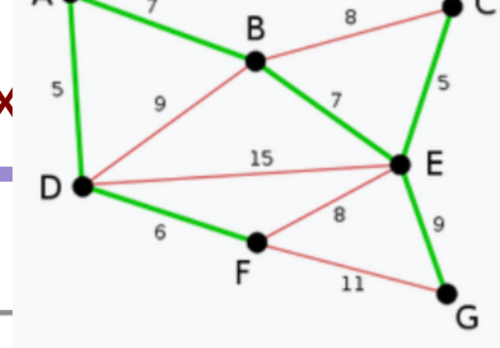
Prim's algorithm

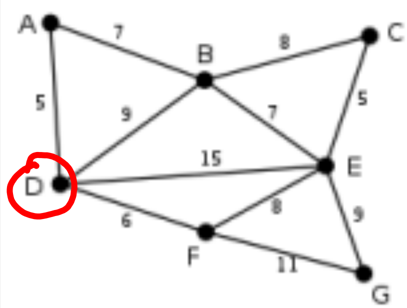
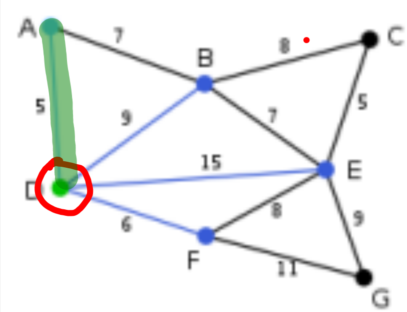


Prim's algorithm starting at vertex A.
In the third step, edges BD and AB both have weight 2, so BD is chosen arbitrarily.
After that step, AB is no longer a candidate for addition to the tree because it links two nodes that are already in the tree.

https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

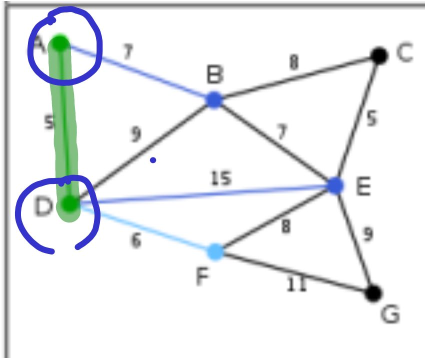
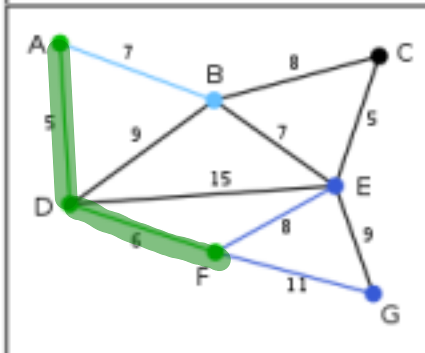
Prim's algorithm ex



	This is the initial weighted graph. It is not a tree, since to be a tree it is required that there are no cycles, and in this case there is. The numbers near the edges indicate the weight. None of the edges is marked, and vertex **D** has been chosen arbitrarily as the starting point.	C, G	A, B, E, F	D
	The second vertex is closest to **D** : **A** is 5 away, **B is** 9, **E is** 15, and **F is** 6. Of these, 5 is the smallest value, so we mark the **DA** edge. 5 9, 15, 6	C, G	B, E, F	A, D

https://es.wikipedia.org/wiki/Algoritmo_de_Prim

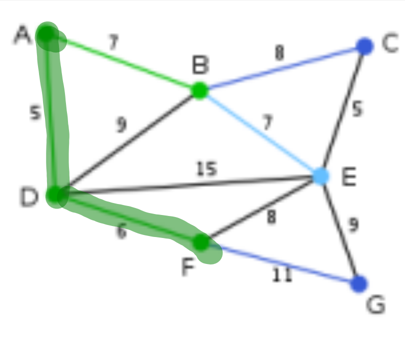
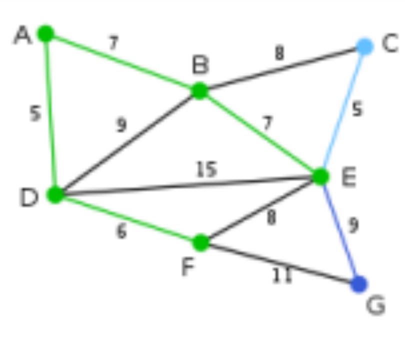
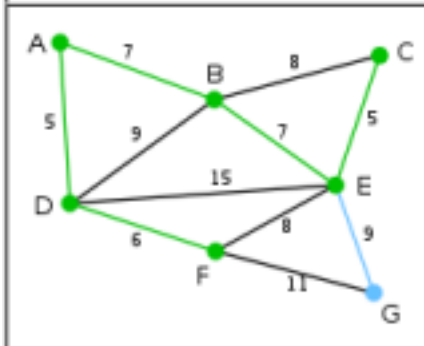
Prim's algorithm examples (2)

	5 9, 15, 6 The next vertex to choose is the closest to **D** or **A.** **B** is 9 away from **D** and 7 away from **A** , **E** is at 15, and **F** is at 6. 6 is the smallest value, so we mark the vertex **F** and the edge **DF** .	C	B, E, G	A, D, F
	The algorithm continues. The vertex **B** , which is at a distance of 7 from **A** , is the next one marked. At this point the edge **DB** is marked in red because its two ends are already in the tree and therefore can not be used.	null	C, E, G	A, D, F, B

https://es.wikipedia.org/wiki/Algoritmo_de_Prim

Prim's algorithm examples (3)

5 9, 15, 6

	Here you have to choose between **C** , **E** and **G. C** is 8 away from **B** , **E** is 7 away from **B** , and **G** is 11 away from **F. E** is closer, so we mark the vertex **E** and the edge **EB** . Two other edges were marked in red because both vertices that join were added to the tree.	null	C, G	A, D, F, B, E
	Only **C** and **G** are available. **C** is 5 away from **E** , and **G is** 9 away from **E.** Choose **C** , and mark with the arc **EC** . The **BC** arc is also marked with red.	null	G	A, D, F, B, E, C
	**G** is the only outstanding vertex, and it is closer to **E** than to **F** , so **EG** is added to the tree. All vertices are already marked, the minimum expansion tree is shown in green. In this case with a weight of 39.	null	null	A, D, F, B, E, C, G

https://es.wikipedia.org/wiki/Algoritmo_de_Prim

References

[1] http://en.wikipedia.org/
[2]