Eulerian Cycle (2A)

Young Won Lim
5/25/18

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

This document was produced by using LibreOffice and Octave.

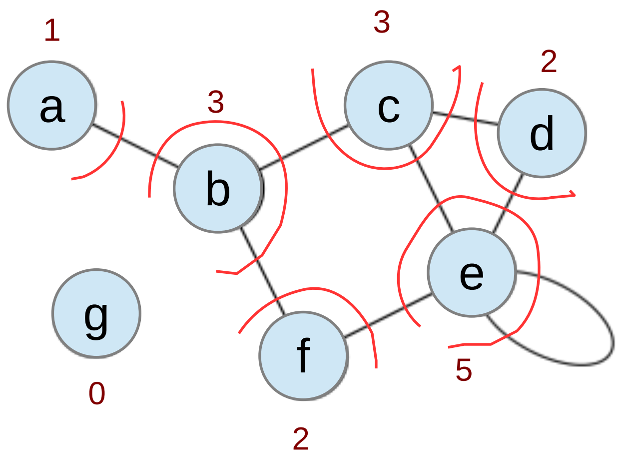
Degree of a vertex

the **degree** (or **valency**) of a vertex is
the number of edges incident to the vertex,
with loops counted twice.

The degree of a vertex v is denoted deg(v)
the maximum degree of a graph G, denoted by Δ(G)
the minimum degree of a graph, denoted by δ(G)

$\Delta(G) = 5$
$\delta(G) = 0$

In a **regular** graph, all degrees are the same

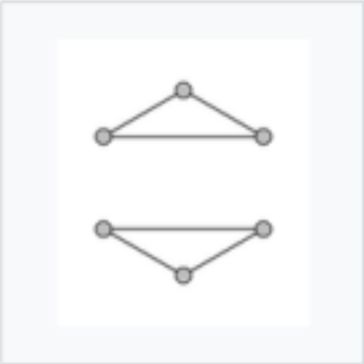


https://en.wikipedia.org/wiki/Degree_(graph_theory)

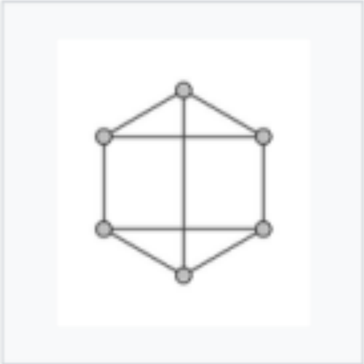
Regular Graphs

a **regular graph** is a graph where each vertex has the same number of neighbors; i.e. every vertex has the same degree or valency.

0-regular graph



1-regular graph



2-regular graph

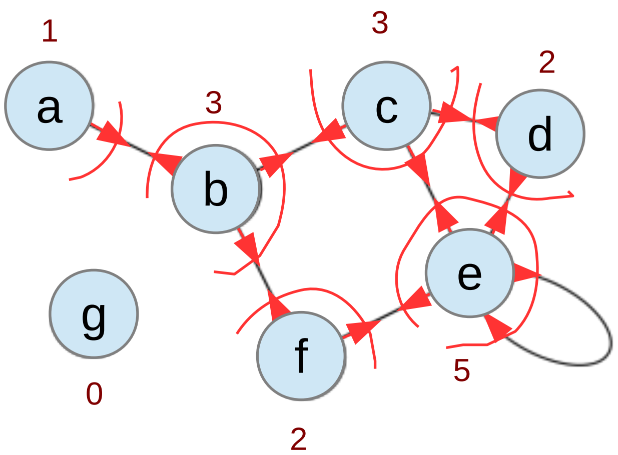
3-regular graph

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

Handshake Lemma

The degree sum formula states that,
given a graph G = (V , E)

$$\sum_{v \in V} \deg(v) = 2|E| \; .$$



This statement (as well as the degree sum formula) is known as the **handshaking lemma**.

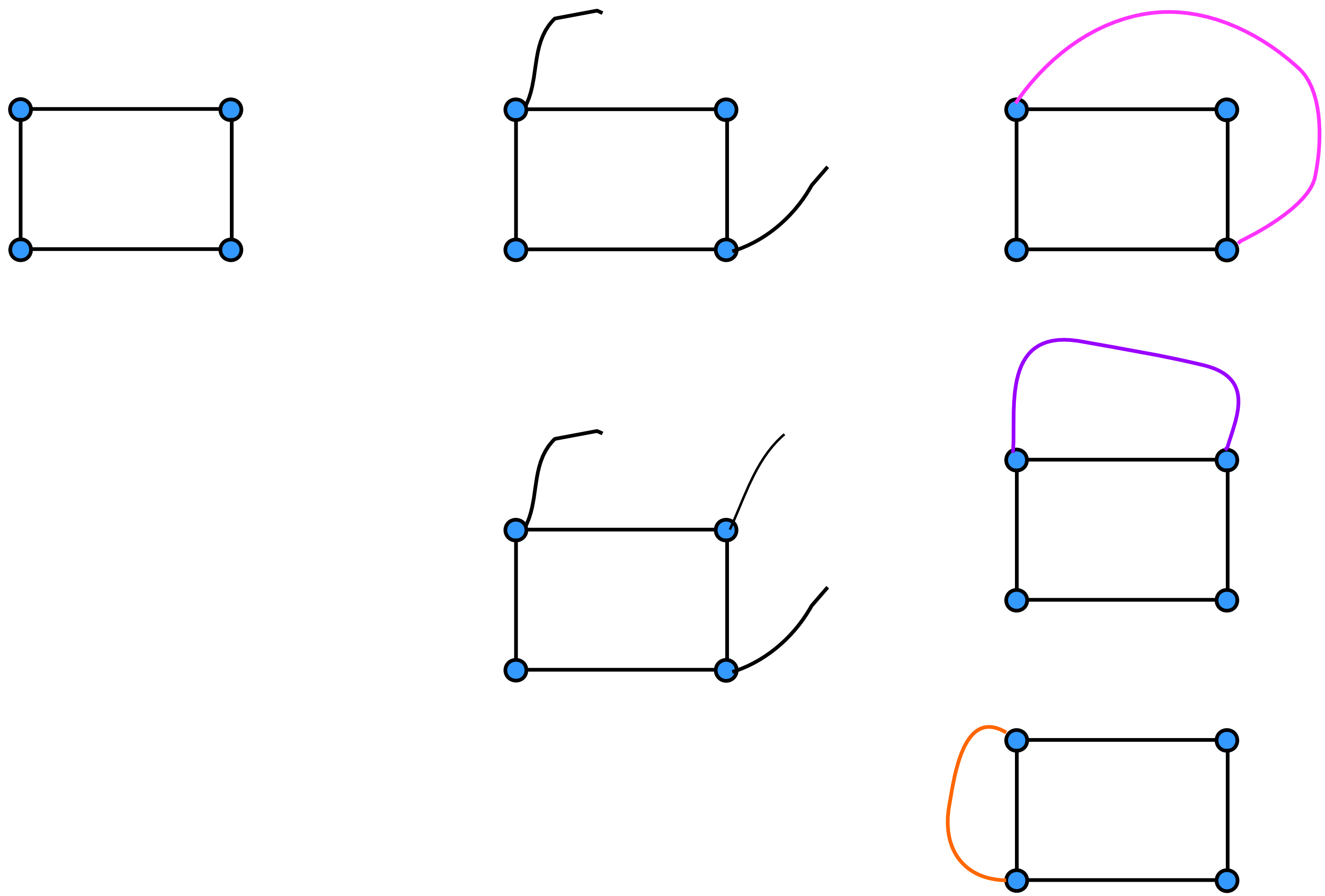
$$\begin{aligned} deg(a) &= 1 \\ deg(b) &= 3 \\ deg(c) &= 3 \\ deg(d) &= 2 \\ deg(e) &= 5 \\ deg(f) &= 2 \\ deg(g) &= 0 \\ \hline & \quad 16 \end{aligned}$$

$$|E| = 8$$

$$2|E| = 16$$

https://en.wikipedia.org/wiki/Degree_(graph_theory)

Adding odd vertex



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

The number of odd vertices

Even vertices : $\{x_{1,} x_{2,} \cdots , x_m\}$

$$S = \underline{deg(x_1)} + \underline{deg(x_2)} + \cdots + \underline{deg(x_m)}$$

$$deg(x_i) : even$$

$$S = \underline{even} + \underline{even} + \cdots + \underline{even}$$

Odd vertices : $\{y_{1,} y_{2,} \cdots , y_n\}$

$$T = \underline{deg(y_1)} + \underline{deg(y_2)} + \cdots + \underline{deg(y_n)}$$

$$deg(y_i) : odd$$

$$T = \underline{odd} + \underline{odd} + \cdots + \underline{odd}$$

$$\left.\begin{array}{l} S : even \\ S+T : even \end{array}\right\} \longrightarrow T : even = \sum n \; odd \; numbers$$

$$\Rightarrow n : even$$

in any graph, the number of vertices with odd degree is even.

# 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

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 – Properties (3)

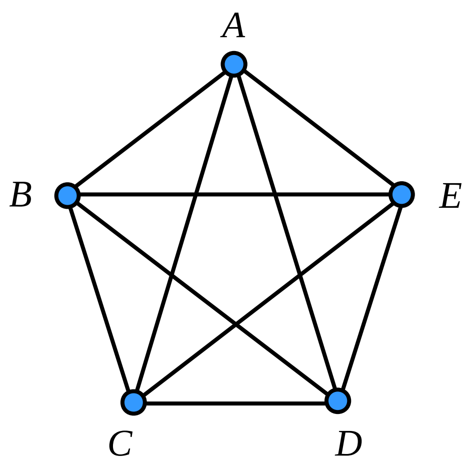
A tournament (with more than two vertices) is Hamiltonian if and only if it is **strongly connected**.

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

Number of Hamiltonian Cycles (1)

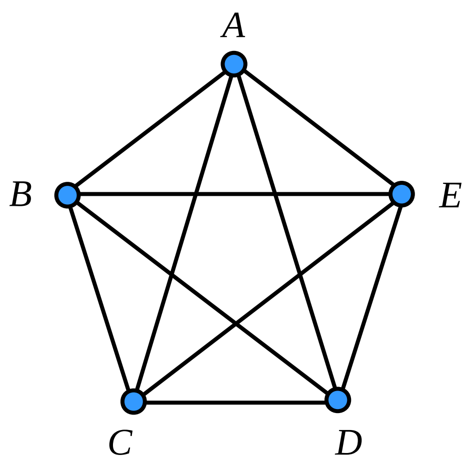


$$(5-1)! = 24$$

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

A	BCDE	AB	CDE	ABC	DE	ABCD	E	ABCDE
				ABD	CE	ABCE	D	ABCED
		AC	BDE	ABE	CD	ABDC	E	ABDCE
						ABDE	C	ABDEC
		AD	BCE	ACB	DE	ABEC	D	ABECD
				ACD	BE	ABED	C	ABEDC
		AE	BCD	ACE	BD			
						ACBD	E	ACBDE
				ADB	CE	ACBE	D	ACBED
				ADC	BE	ACDB	E	ACDBE
				ADE	BC	ACDE	B	ACDEB
						ACEB	D	ACEBD
				AEB	CD	ACED	B	ACEDB
				AEC	BD			
				AED	BC	ADBC	E	ADBCE
						ADBE	C	ADBEC
						ADCB	E	ADCBE
						ADCE	B	ADCEB
						ADEB	C	ADEBC
						ADEC	B	ADECB
						AEBC	D	AEBCD
						AEBD	C	AEBDC
						AECB	D	AECBD
						AECD	B	AECDB
						AEDB	C	AEDBC
						AEDC	B	AEDCB

Number of Hamiltonian Cycles (2)

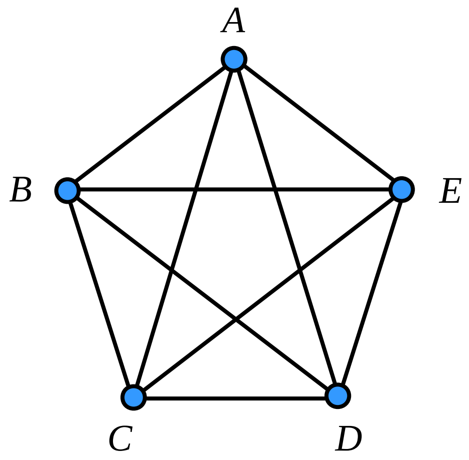


$$(5-1)!=24$$

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

ABCDE	BACDE	CABDE	DABCE	EABCD
ABCED	BACED	CABED	DABEC	EABDC
ABDCE	BADCE	CADBE	DACBE	EACBD
ABDEC	BADEC	CADEB	DACEB	EACDB
ABECD	BAECD	CAEBD	DADBC	EADBC
ABEDC	BAEDC	CAEDB	DADCB	EADCB
ACBDE	BCADE	CBADE	DBACE	EBACD
ACBED	BCAED	CBAED	DBAEC	EBADC
ACDBE	BCDAE	CBDAE	DBCAE	EBCAD
ACDEB	BCDEA	CBDEA	DBCEA	EBCDA
ACEBD	BCEAD	CBEAD	DBEAC	EBDAC
ACEDB	BCEDA	CBEDA	DBECA	EBDCA
ADBCE	BDACE	CDABE	DCABE	ECABD
ADBEC	BDAEC	CDAEB	DCAEB	ECADB
ADCBE	BDCAE	CDBAE	DCBAE	ECBAD
ADCEB	BDCEA	CDBEA	DCBEA	ECBDA
ADEBC	BDEAC	CDEAB	DCEAB	ECDAB
ADECB	BDECA	CDEBA	DCEBA	ECDBA
AEBCD	BEACD	CEABD	DEABC	EDABC
AEBDC	BEADC	CEADB	DEACB	EDACB
AECBD	BECAD	CEBAD	DEBAC	EDBAC
AECDB	BECDA	CEBDA	DEBCA	EDBCA
AEDBC	BEDAC	CEDAB	DECAB	EDCAB
AEDCB	BEDCA	CEDBA	DECBA	EDCBA

Number of Hamiltonian Cycles (3)



$$\frac{(5-1)!}{2} = \frac{24}{2} = 12$$

$\boxed{ABCDE}$
$\boxed{ABCED}$
$\boxed{ABDCE}$
$ABDEC$
$ABECD$
$ABEDC$
$ACBDE$
$ACBED$
$ACDBE$
$ACDEB$
$ACEBD$
$ACEDB$
$ADBCE$
$ADBEC$
$ADCBE$
$ADCEB$
$ADEBC$
$\boxed{ADECB}$
$AEBCD$
$AEBDC$
$AECBD$
$\boxed{AECDB}$
$AEDBC$
$\boxed{AEDCB}$

$A-B-C-D-E-A$

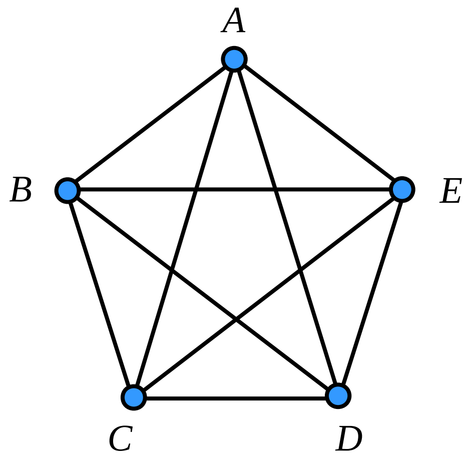
$A-B-C-E-D-A$

$A-B-D-C-E-A$

(n − 1)! / 2

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

Number of Hamiltonian Cycles (4)



$$\frac{(5-1)!}{2} = \frac{24}{2} = 12$$

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

ABCDE	ABCDE	ABCDE
ABCED	ABCED	ABCED
ABDCE	ABDCE	ABDCE
ABDEC	ABDEC	ABDEC
ABECD	ABECD	ABECD
ABEDC	ABEDC	ABEDC
ACBDE	ACBDE	ACBDE
ACBED	ACBED	ACBED
ACDBE	ACDBE	ACDBE
ACDEB	ACDEB	ACEBD
ACEBD	ACEBD	ADBCE
ACEDB	ACEDB	ADCBE
ADBCE	ADBCE	
ADBEC	ADBEC	
ADCBE	ADCBE	
ADCEB	ADCEB	
ADEBC	ADEBC	
ADECB	ADECB	
AEBCD	AEBCD	
AEBDC	AEBDC	
AECBD	AECBD	
AECDB	AECDB	
AEDBC	AEDBC	
AEDCB	AEDCB	

(n − 1)! / 2

References

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

Planar Graph (7A)

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

This document was produced by using LibreOffice and Octave.

Planar Graph

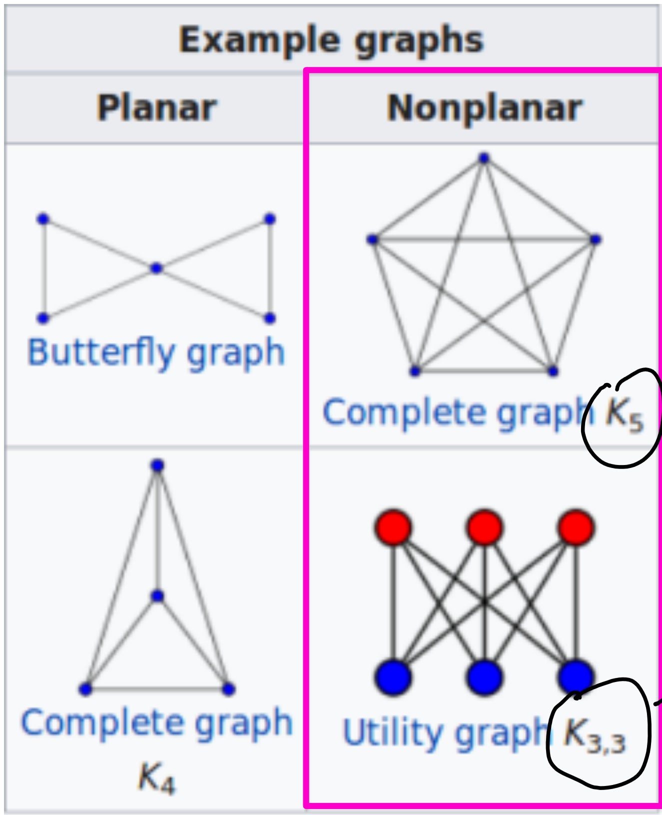
a planar graph is a graph that can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints.

it can be drawn in such a way that no edges cross each other. Such a drawing is called a **plane graph** or **planar embedding** of the graph. (**planar representation**)

A **plane graph** can be defined as a planar graph with a mapping from every node to a point on a plane, and from every edge to a plane curve on that plane,
such that the extreme points of each curve are the points mapped from its end nodes, and all curves are disjoint except on their extreme points.

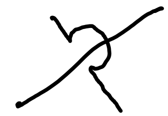
https://en.wikipedia.org/wiki/Planar_graph

Planar Graph Examples

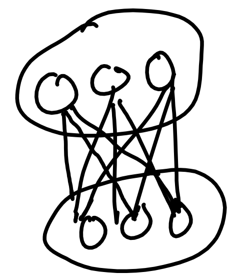


complete bipartite Graph

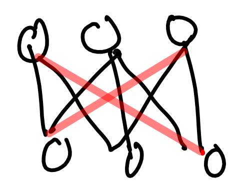
$K5$



$K_{3,3}$



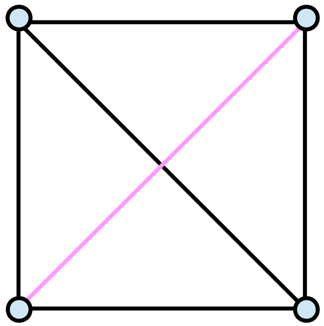
bipartite graph

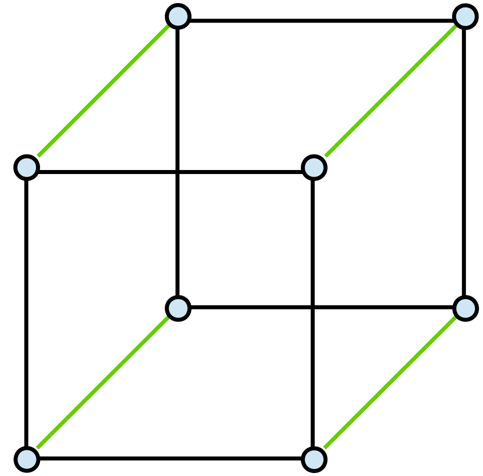


https://en.wikipedia.org/wiki/Planar_graph

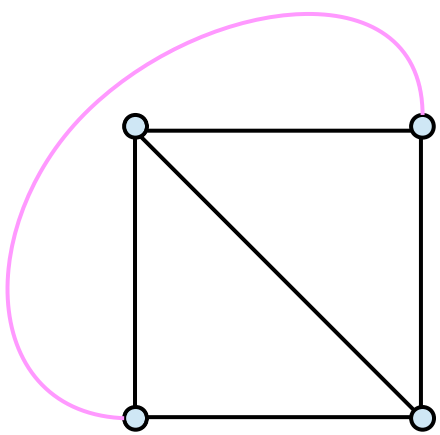
Planar Representation

K_4



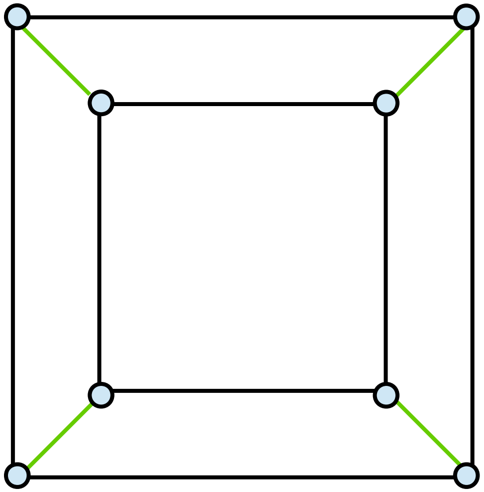


Q_3



No crossing

K_4 Planar



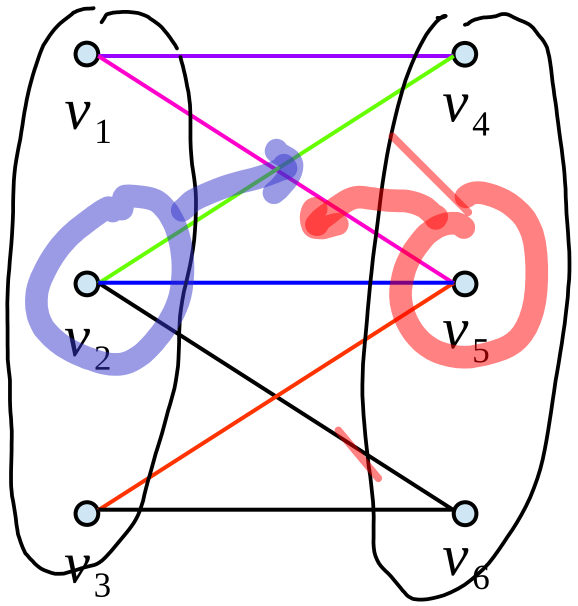
No crossing

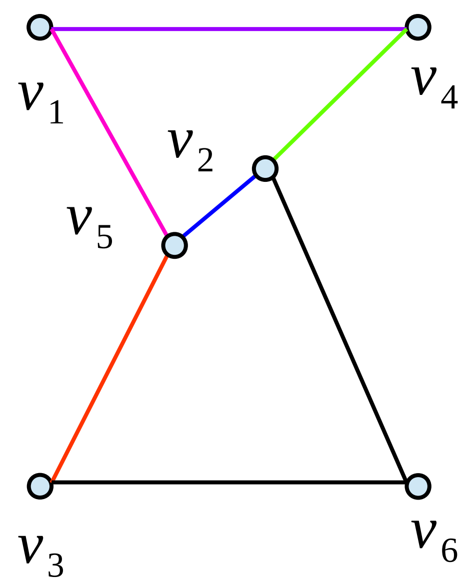
Q_3 Planar

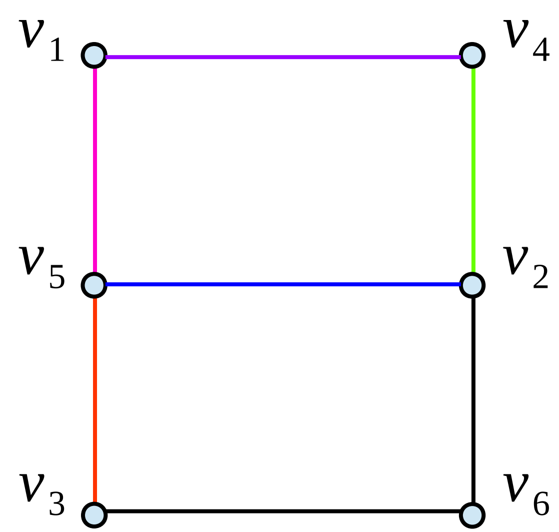
Discrete Mathematics, Rosen

A planar bipartite graph

$\cancel{K_{3,3}}$



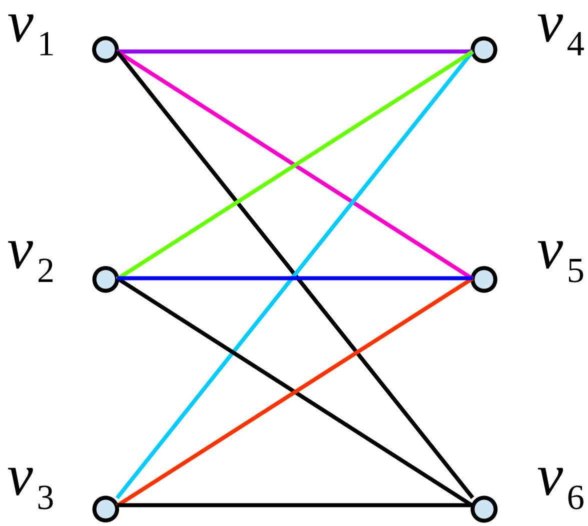


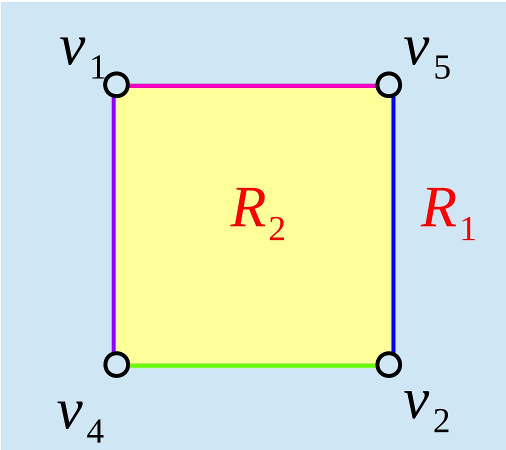


Bipartite graph
but not complete
bipartite graph
$K_{3,3}$

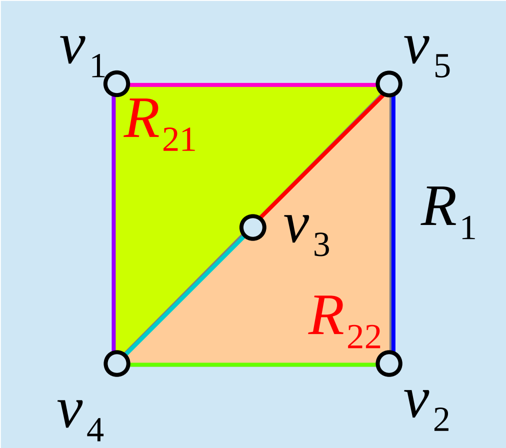
Planar Graph

Non-planar Graph $K_{3,3}$





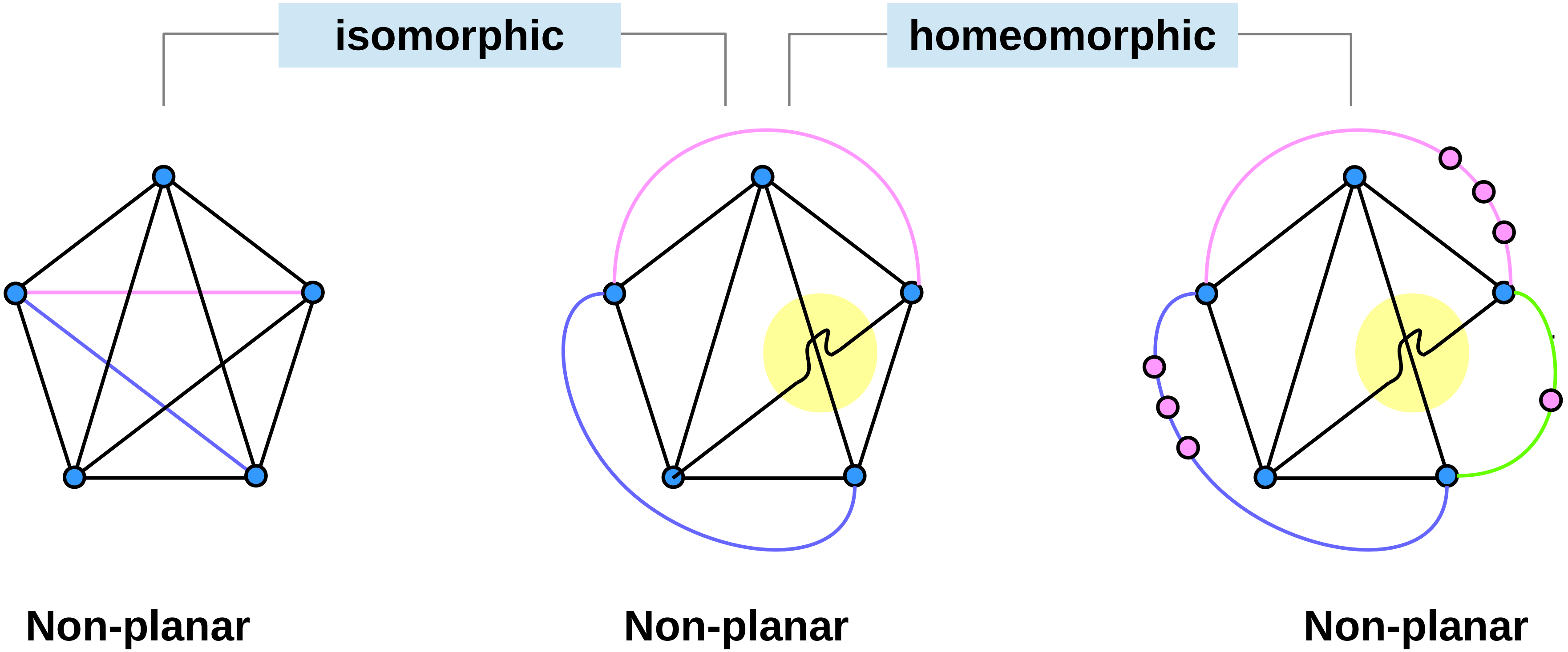
no where v_6



Non-planar

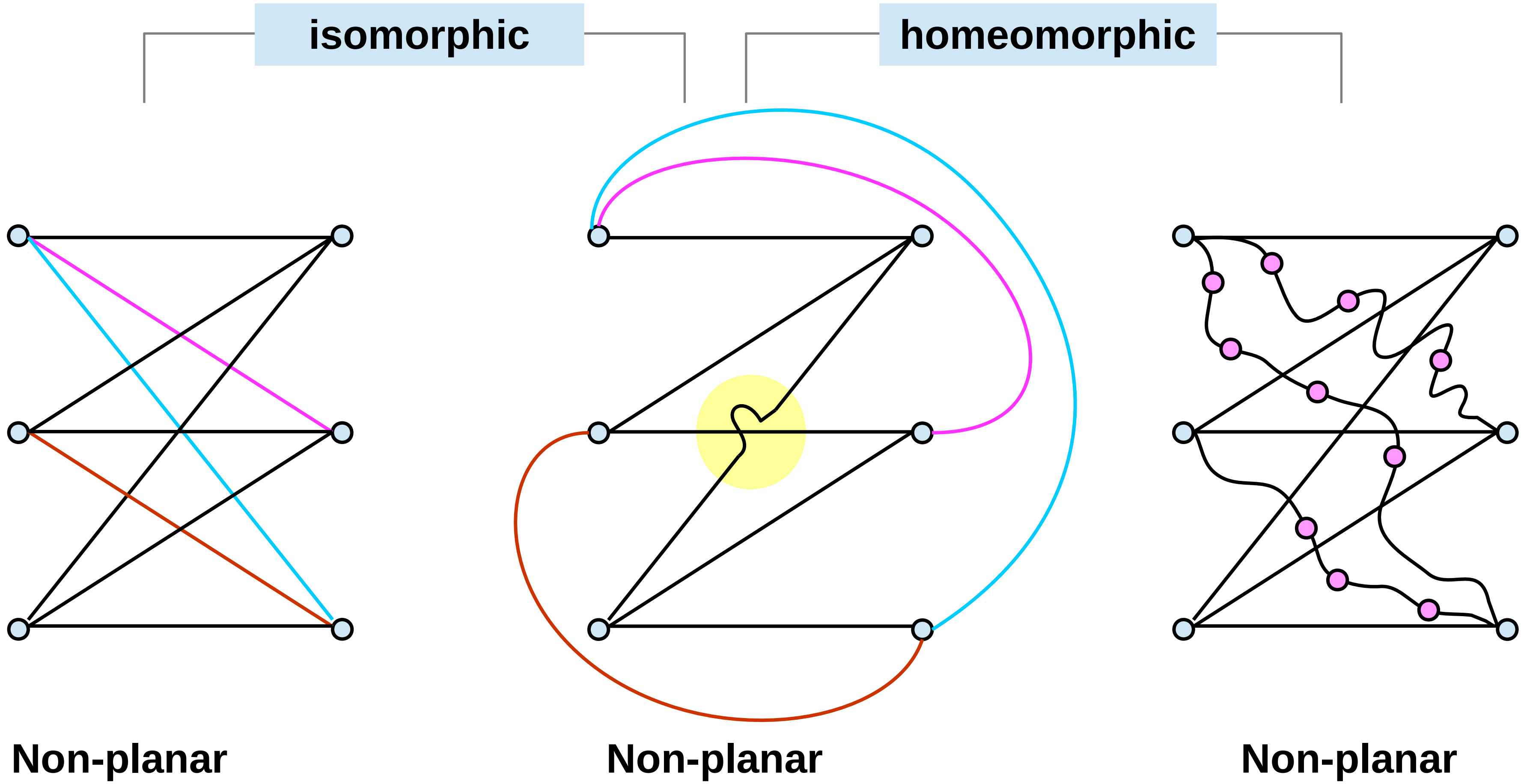
Discrete Mathematics, Rosen

Non-planar graph examples - K_5



All these graphs are similar
in determining whether
they are planar or not

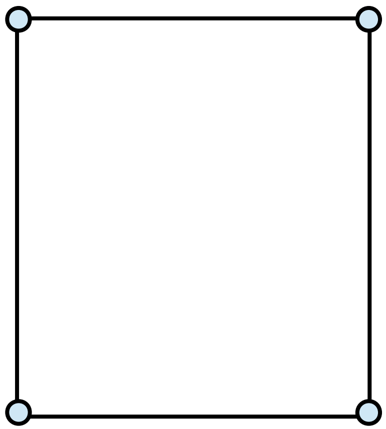
Non-planar graph examples – $K_{3,3}$



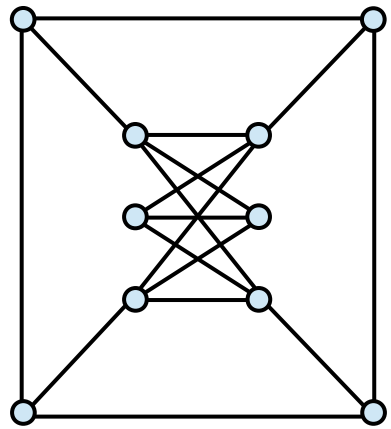
All these graphs are similar
in determining whether
they are planar or not

Non-planar graph examples – embedding $K_{3,3}$

Planar



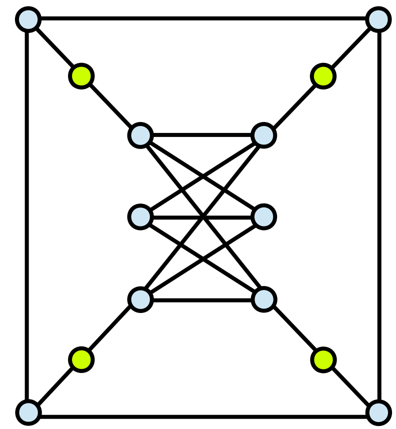
Non-planar



contains $K_{3,3}$

non-planar subgraph

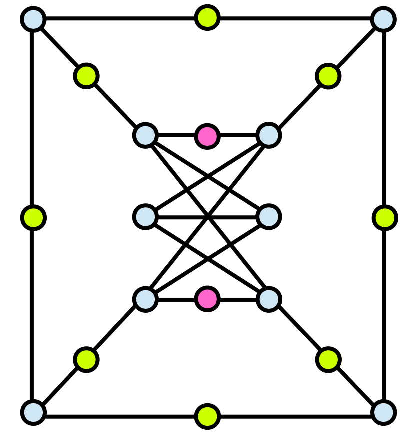
Non-planar



contains $K_{3,3}$

non-planar subgraph

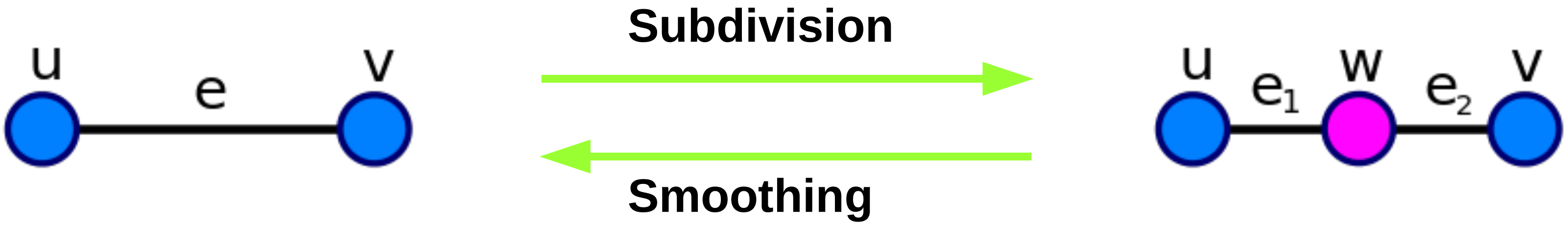
Non-planar



contains a <u>subdivision</u> of $K_{3,3}$

non-planar subgraph

Subdivision and Smoothing



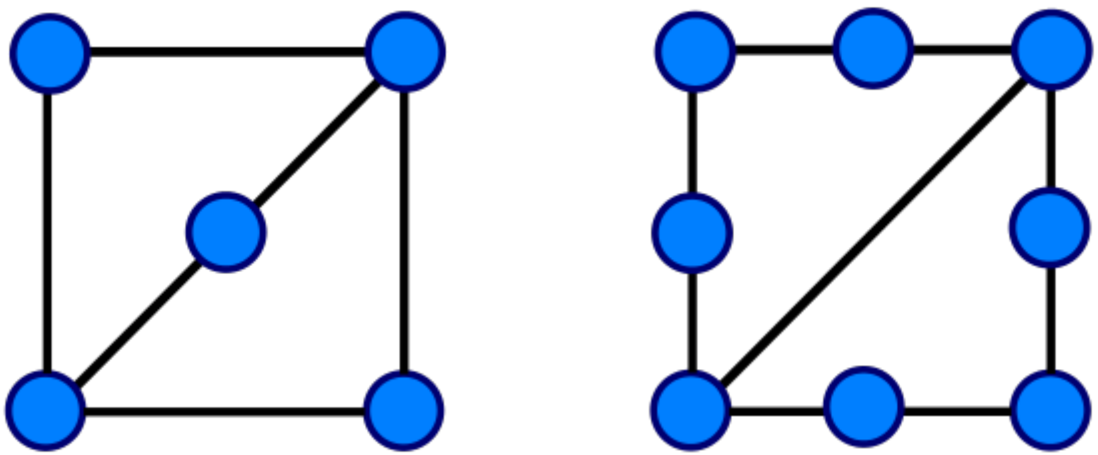
https://en.wikipedia.org/wiki/Planar_graph

Homeomorphism

two graphs G_1 and G_2 are **homeomorphic**
if there is a graph **isomorphism**
from some **subdivision** of G_1
to some **subdivision** of G_2

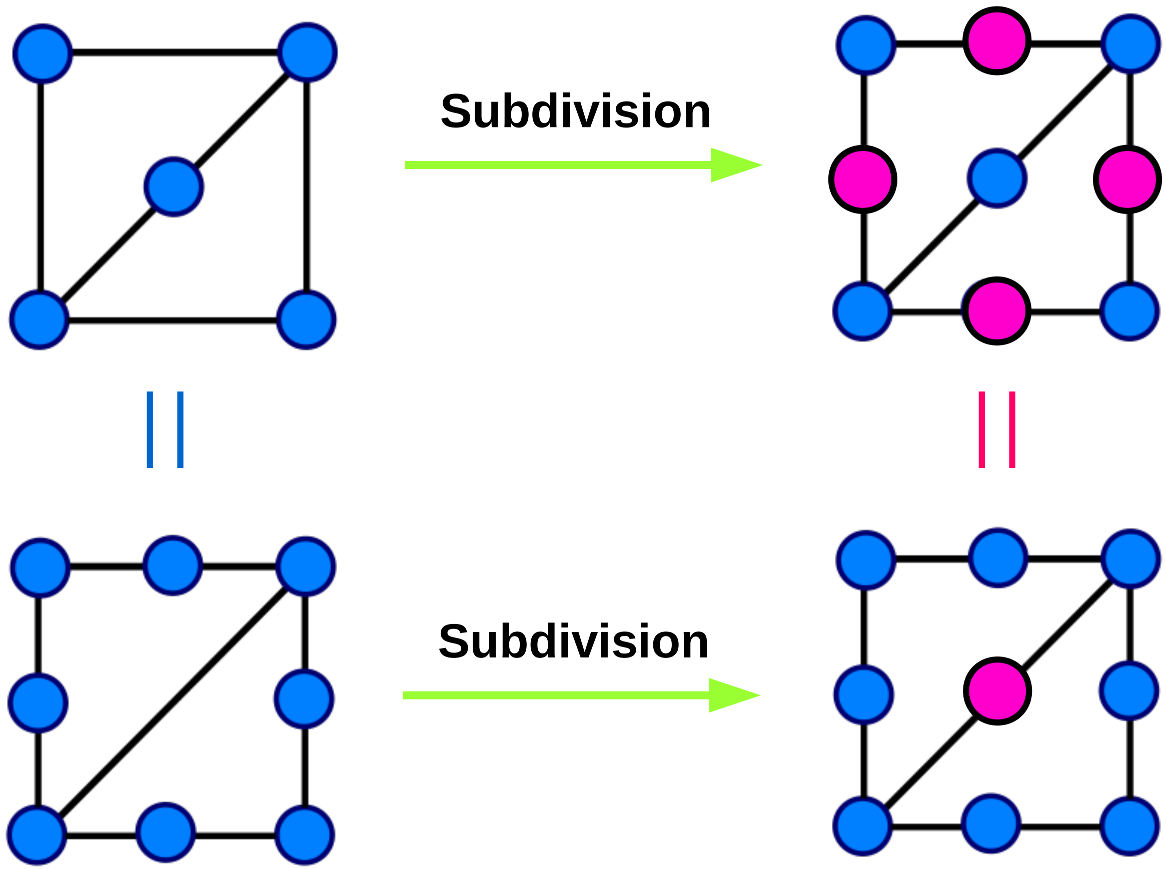
homeo (identity, sameness)

iso (equal)



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

Homeomorphism Examples



homeomorphic → **Subdivision** → **isomorphic**

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

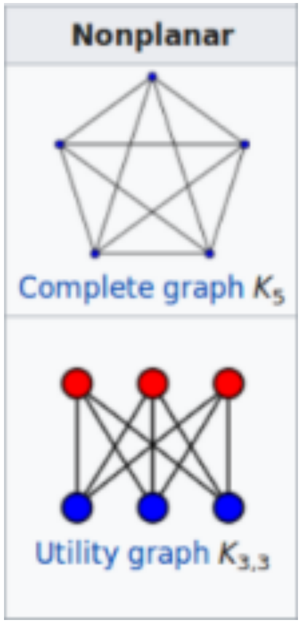
Embedding on a surface

subdividing a graph preserves **planarity**.

Kuratowski's theorem states that

> a finite graph is **planar** if and only if
> it contains **no** subgraph **homeomorphic**
> to $\mathbf{K_5}$ (complete graph on five vertices) or
> $\mathbf{K_{3,3}}$ (complete bipartite graph on six vertices,
> three of which connect to each of the other three).

In fact, a graph **homeomorphic** to $\mathbf{K_5}$ or $\mathbf{K_{3,3}}$
is called a **Kuratowski subgraph**.

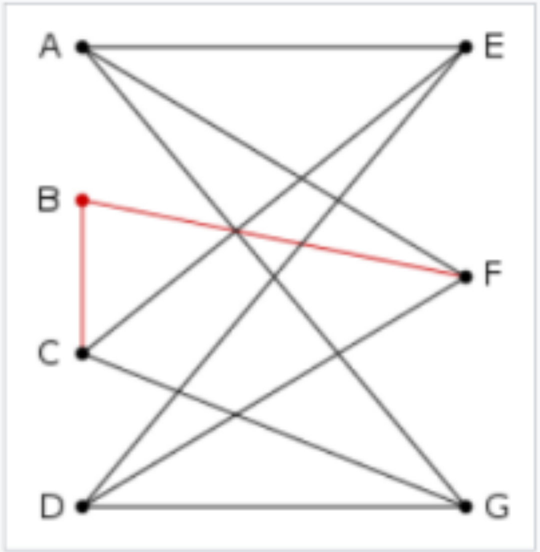


https://en.wikipedia.org/wiki/Planar_graph

Kuratowski's Theorem

A finite graph is planar if and only if
it does not contain a **subgraph**
that is a **subdivision** of the complete graph $\mathbf{K_5}$ or
the complete bipartite graph $\mathbf{K_{3,3}}$ (utility graph).

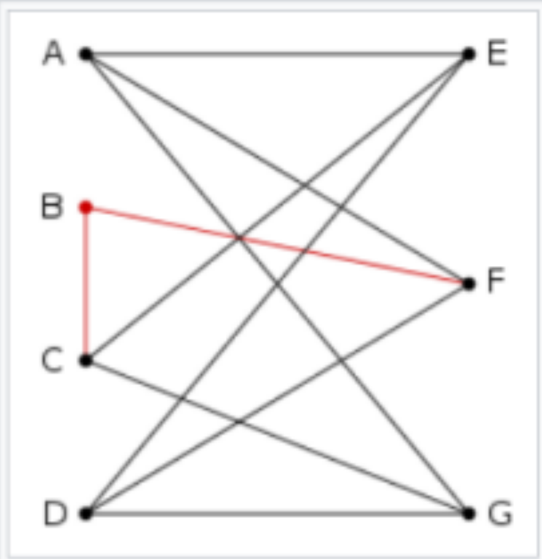
A subdivision of a graph results
from inserting vertices into edges
(changing an edge •——• to •—•—•)
zero or more times.



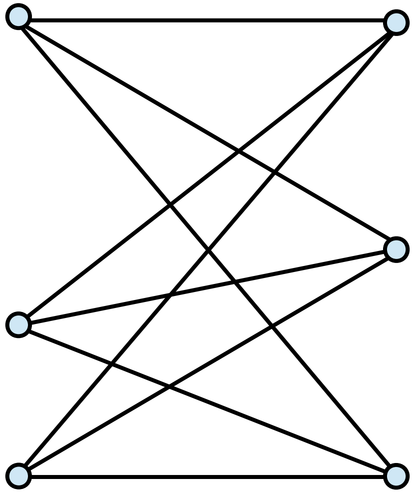
An example of a graph with no K_5 or $K_{3,3}$ subgraph. However, it contains a subdivision of $K_{3,3}$ and is therefore non-planar.

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

Kuratowski's Theorem

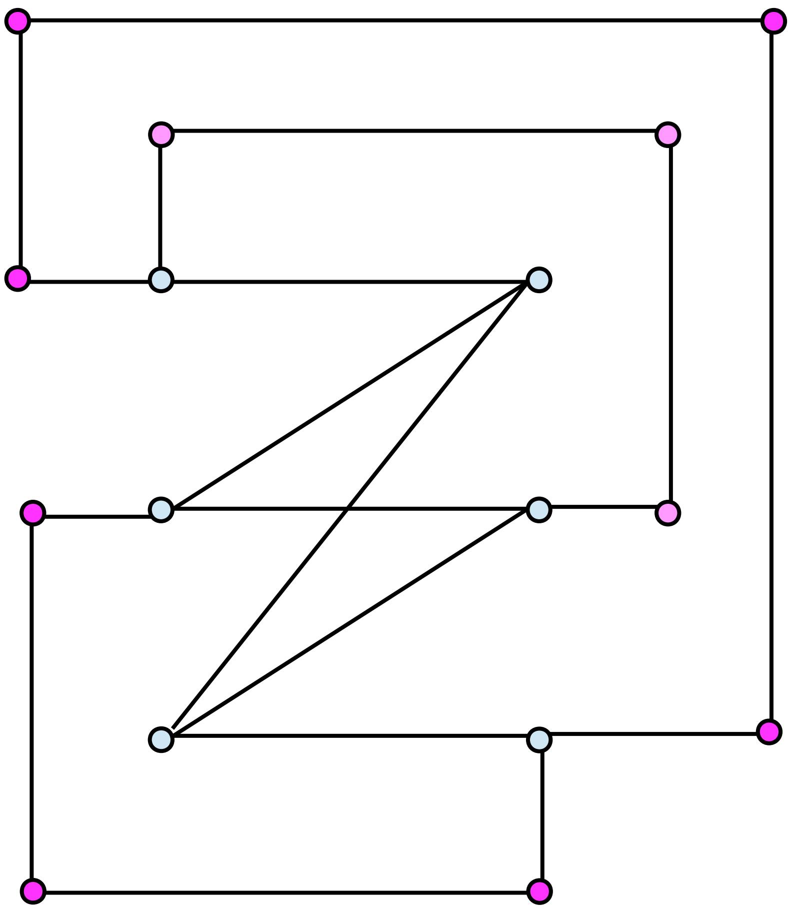


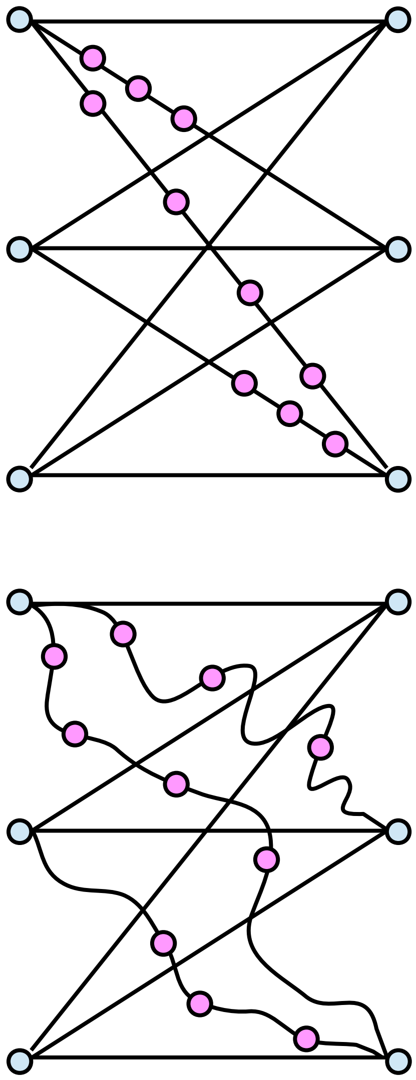
An example of a graph with no K_5 or $K_{3,3}$ subgraph. However, it contains a subdivision of $K_{3,3}$ and is therefore non-planar.



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

A subdivision of $K_{3,3}$





Euler’s Formula

Euler's formula states that if a **finite**, **connected**, **planar graph** is drawn in the plane without any edge intersections, and **v** is the number of **vertices**, **e** is the number of **edges** and **f** is the number of **faces** (regions bounded by edges, including the outer, infinitely large region), then

$$\mathbf{v - e + f = 2}$$

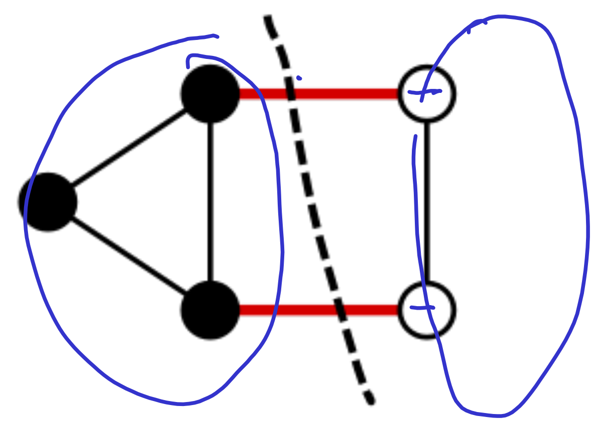
https://en.wikipedia.org/wiki/Planar_graph

Minimum Cut

A cut is minimum if the size or weight of the cut
is not larger than the size of any other cut.

the size of this cut is 2,
and there is no cut of size 1
because the graph is bridgeless.

Cut



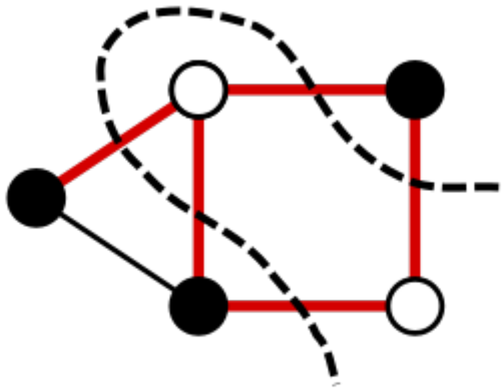
Cutset

cross

https://en.wikipedia.org/wiki/Cut_(graph_theory)

Maximum Cut

A cut is maximum if the size of the cut
is not smaller than the size of any other cut.



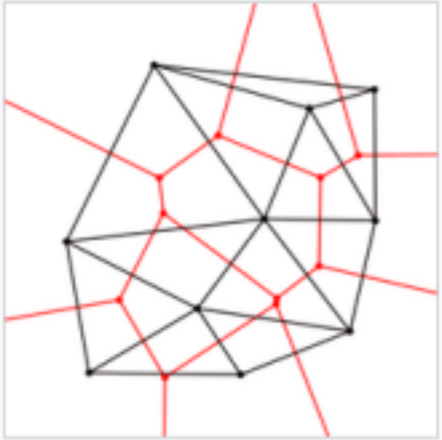
the size of the cut is equal to 5,
and there is no cut of size 6,
or |E| (the number of edges),
because the graph is not bipartite
(there is an odd cycle).

https://en.wikipedia.org/wiki/Cut_(graph_theory)

Infinite Graphs and Tessellations

The concept of duality applies as well
to **infinite graphs** embedded in the plane
as it does to **finite graphs**.

When all faces are bounded regions
surrounded by a cycle of the graph,
an **infinite planar** graph embedding
can also be viewed as a **tessellation** of the plane,
a covering of the plane by closed disks
(the **tiles** of the **tessellation**) whose interiors
(the **faces** of the **embedding**) are disjoint open disks.



A Voronoi diagram (red) and Delaunay triangulation (black) of a finite point set (the black points)

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

References

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

Tree Overview (1A)

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

This document was produced by using LibreOffice and Octave.

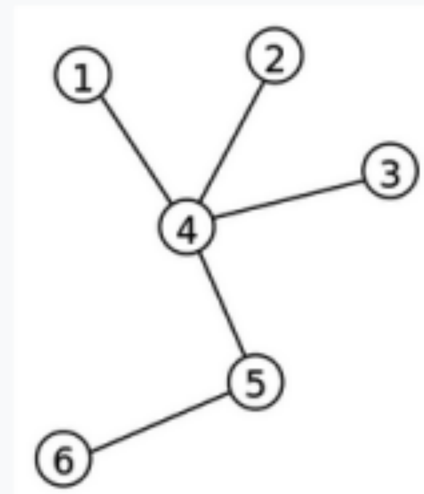
Tree

a tree is an **undirected** graph in which
any two **vertices** are **connected**
by exactly **one path**.

any **acyclic connected** graph is a **tree**.

A **forest** is a disjoint union of trees.

Trees



A labeled tree with 6 vertices and 5 edges.

Vertices	v
Edges	$v - 1$
Chromatic number	2 if $v > 1$

https://en.wikipedia.org/wiki/Tree_(graph_theory)

Tree Condition (1)

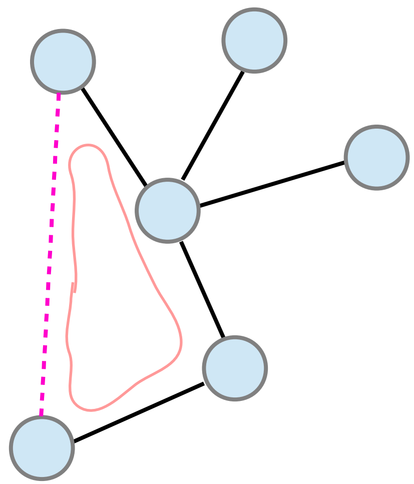
A **tree** is an **undirected** graph G
that satisfies any of the following equivalent conditions:

G is **connected** and has no **cycles**.
G is **acyclic**, and a **simple cycle** is formed if any **edge** is added to G.
G is **connected**, but is not **connected** if any single **edge** is removed from G.
G is **connected** and the 3-vertex complete graph $\mathbf{K_3}$ is not a **minor** of G.
Any **two vertices** in G can be **connected** by a unique **simple path**.

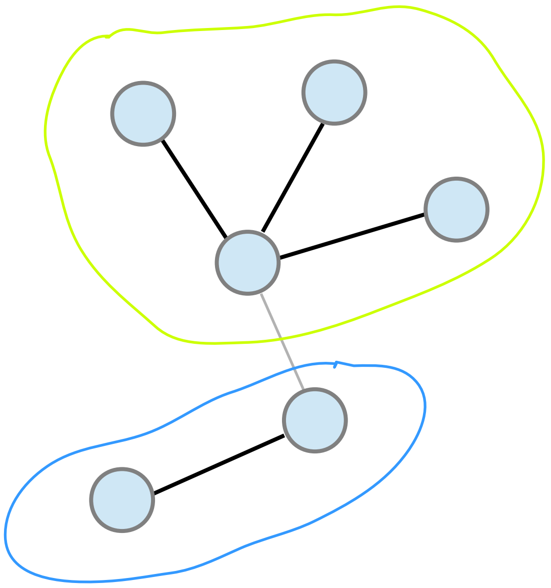
https://en.wikipedia.org/wiki/Tree_(graph_theory)

Tree Condition (2)

G is **acyclic**, and a **simple cycle** is formed if any **edge** is added to G.

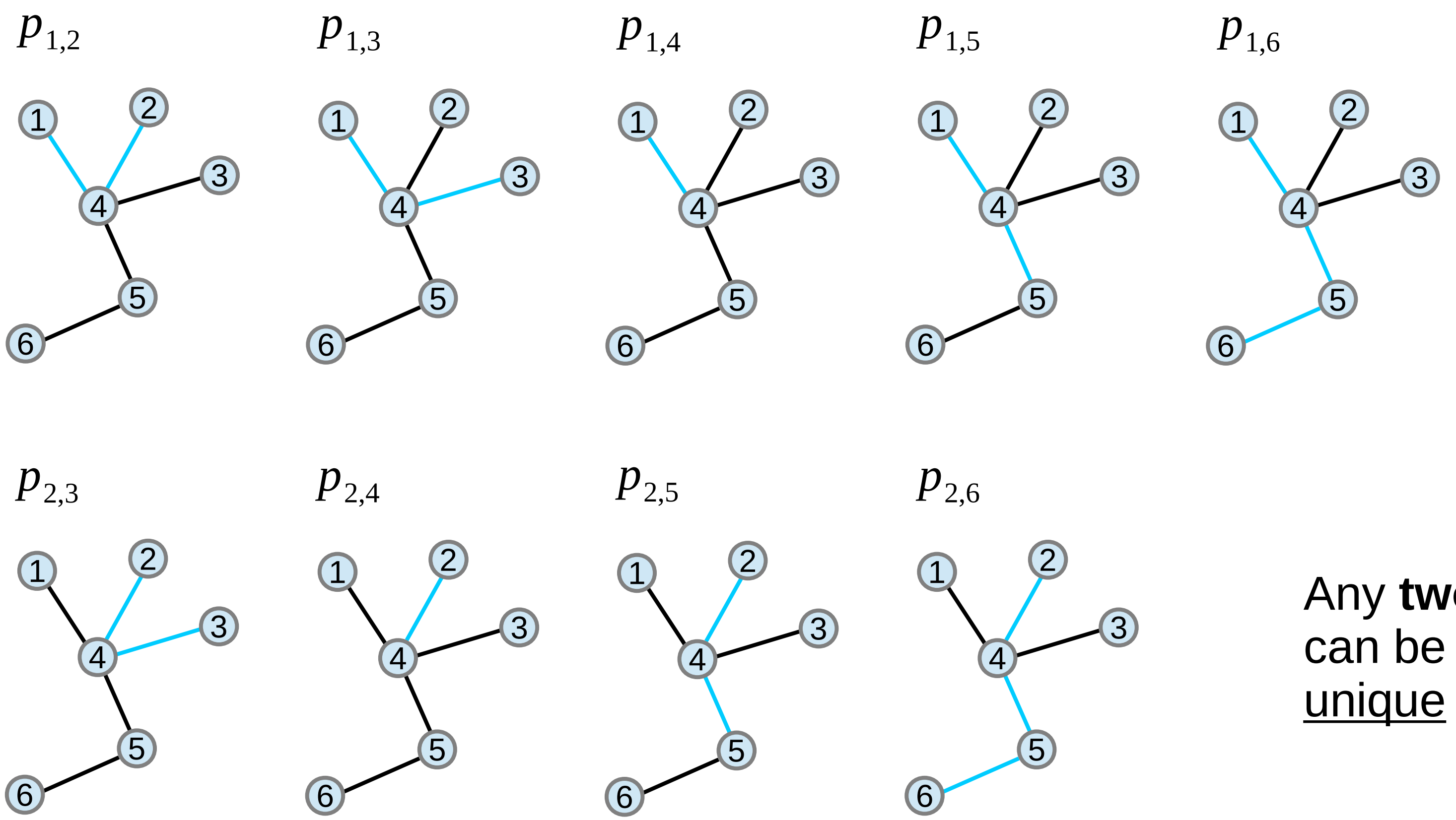


G is **connected**, but is not **connected** if any single **edge** is removed from G.



https://en.wikipedia.org/wiki/Tree_(graph_theory)

Tree Condition (3)

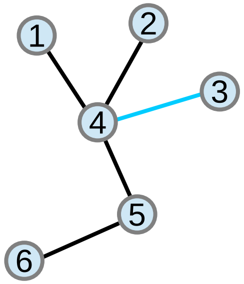


Any **two vertices** in G can be **connected** by a <u>unique</u> **simple path**.

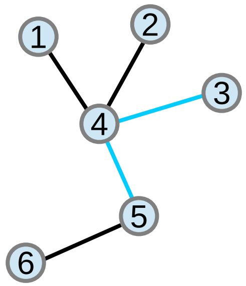
https://en.wikipedia.org/wiki/Tree_(graph_theory)

Tree Condition (4)

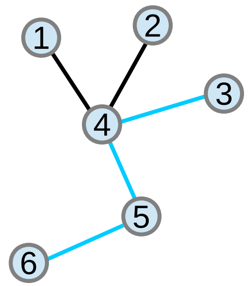
$p_{3,4}$



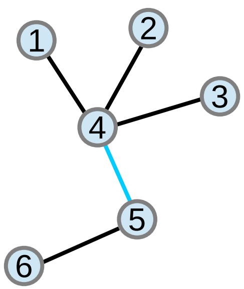
$p_{3,5}$



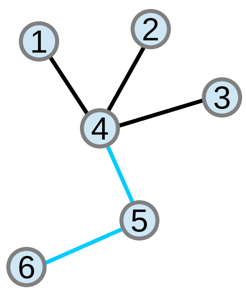
$p_{3,6}$



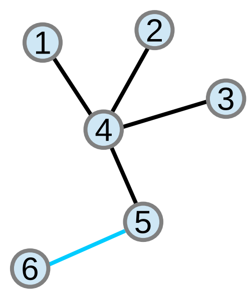
$p_{4,5}$



$p_{4,6}$



$p_{5,6}$

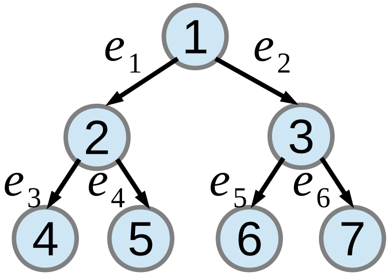


Any **two vertices** in G can be **connected** by a unique **simple path**.

https://en.wikipedia.org/wiki/Tree_(graph_theory)

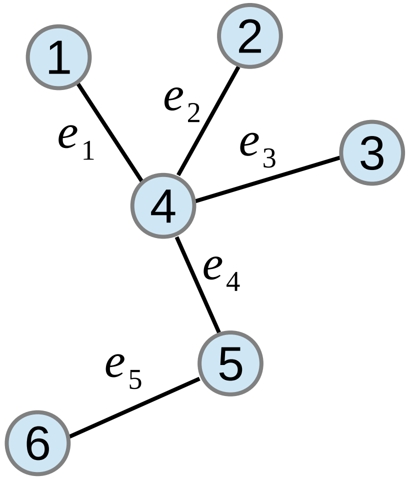
Tree Condition (5)

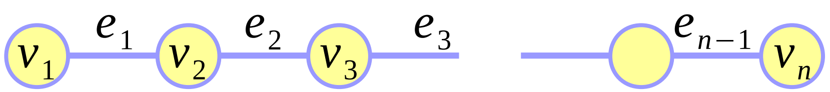
If G has finitely many **vertices**,
say **n vertices**, then the above statements
are also equivalent to any of the following conditions:



G is **connected** and has **n − 1 edges**.

G has **no simple cycles** and has **n − 1 edges**.

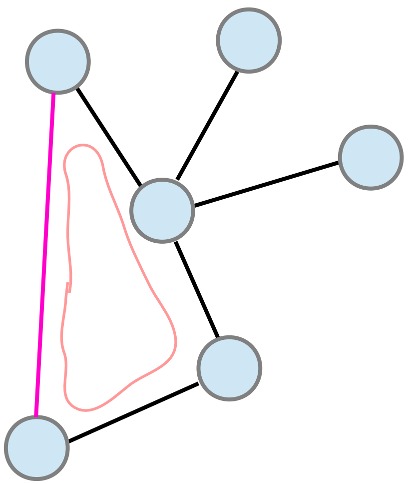


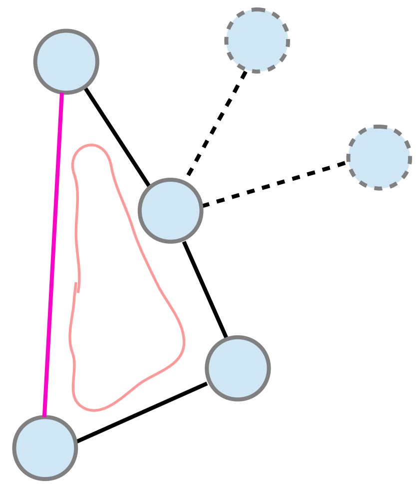


https://en.wikipedia.org/wiki/Tree_(graph_theory)

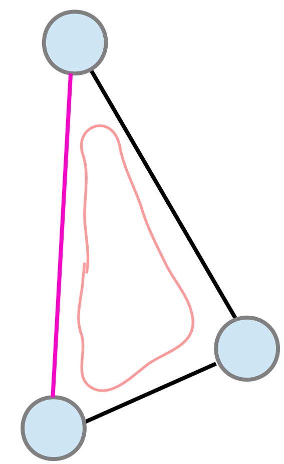
Tree Condition (6)

G is **connected** and the 3-vertex complete graph $\mathbf{K_3}$ is not a **minor** of G.



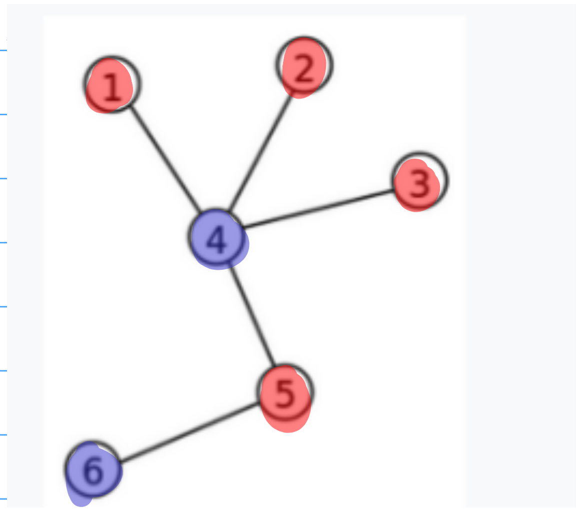


deleting edges
deleting vertices



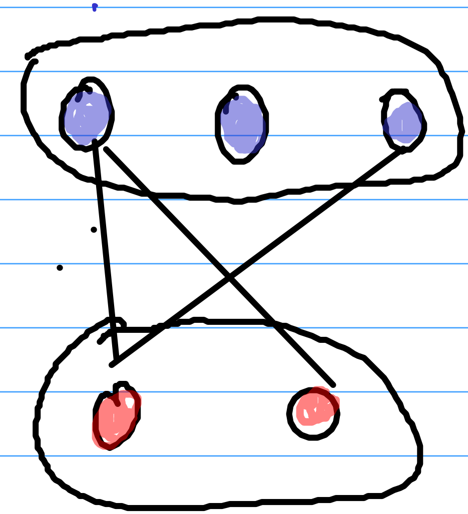
contracting edges

https://en.wikipedia.org/wiki/Tree_(graph_theory)



all tree
chromatic number $= 2$

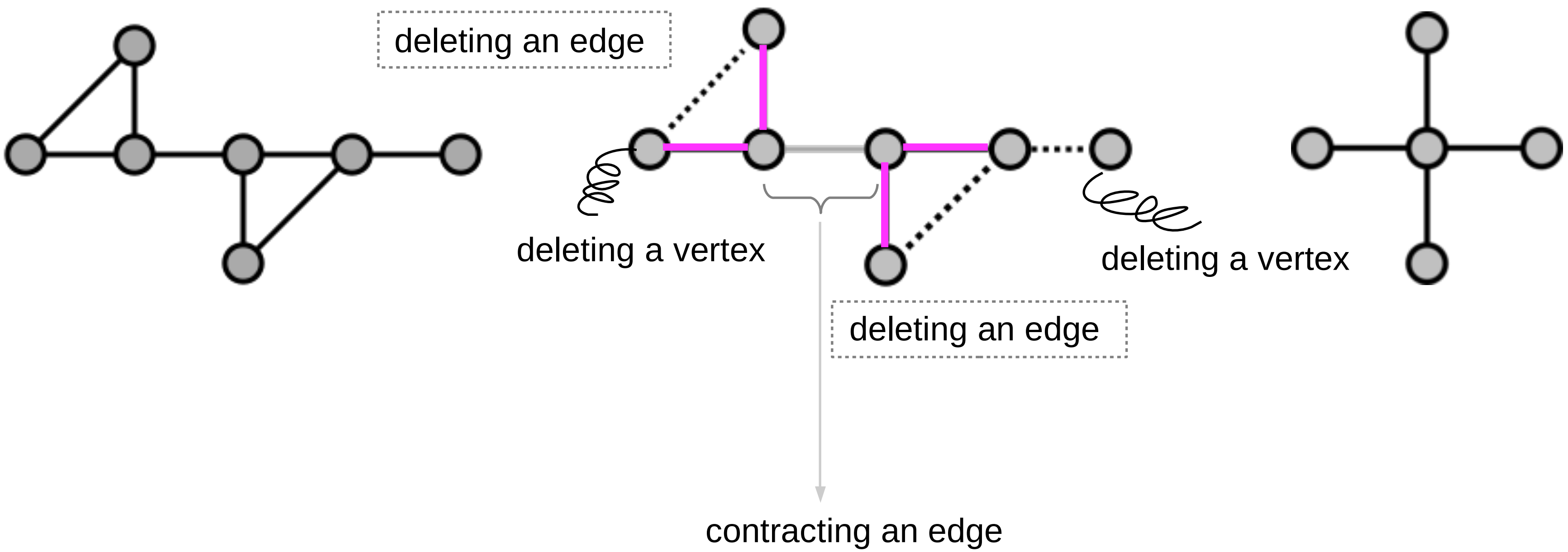
bipartite graph



Chromatic number $\chi = 2$

Graph Minor

In graph theory, an undirected graph H
is called a minor of the graph G
if H can be formed from G
by **deleting edges** and **vertices** and
by **contracting edges**.



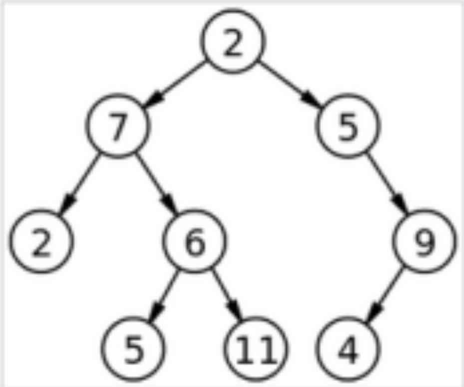
https://en.wikipedia.org/wiki/Graph_minor

Binary Tree

a **binary tree** is a tree data structure in which each **node** has at most two **children**, (the **left child**, the **right child**)

A **recursive definition** using just set theory notions is that a (non-empty) binary tree is a tuple **(L, S, R)**, where **L** and **R** are **binary trees** or the **empty set** and **S** is a **singleton set**.

Some authors allow the binary tree to be the empty set as well.



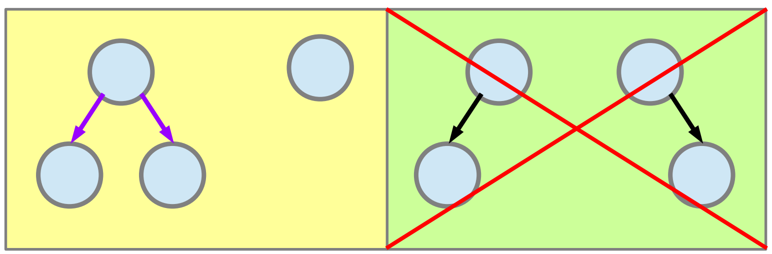
A labeled binary tree of size 9 and height 3, with a root node whose value is 2. The above tree is unbalanced and not sorted.

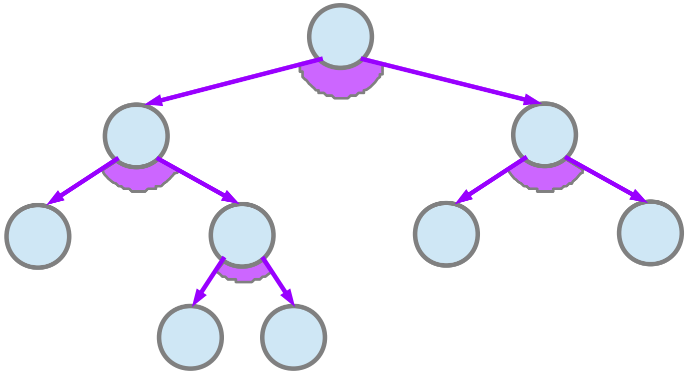
https://en.wikipedia.org/wiki/Binary_tree

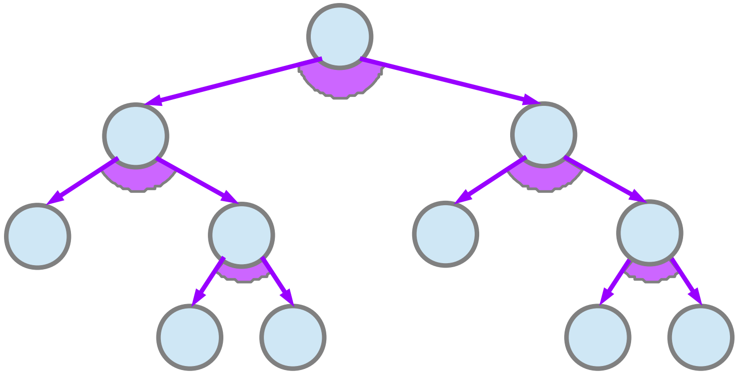
Full Binary Tree

A **rooted binary tree** has a **root node** and every **node** has at most two **children**.

A **full binary tree** is (**proper, plane binary tree**) a **tree** in which every **node** has either **0** or **2 children**.







https://en.wikipedia.org/wiki/Tree_(graph_theory)

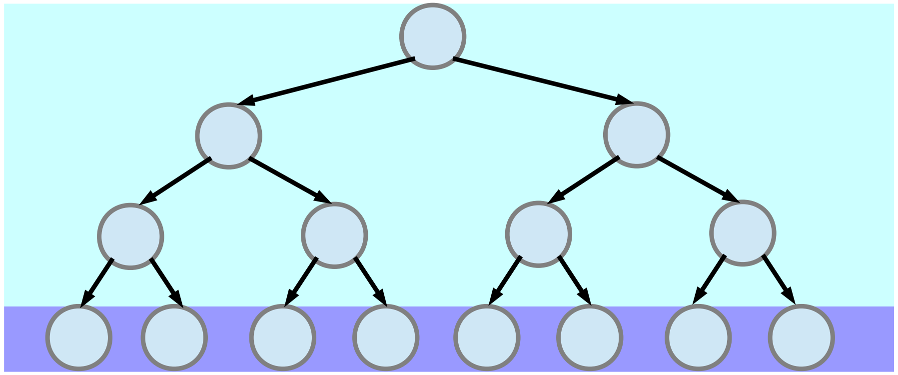
Perfect Binary Trees

A **perfect binary tree** is a binary tree in which
all **interior nodes** have two **children** and
all **leaves** have the same **depth** or same **level**.

also called a **complete binary tree**

two **children**

the same **depth** (**level**).

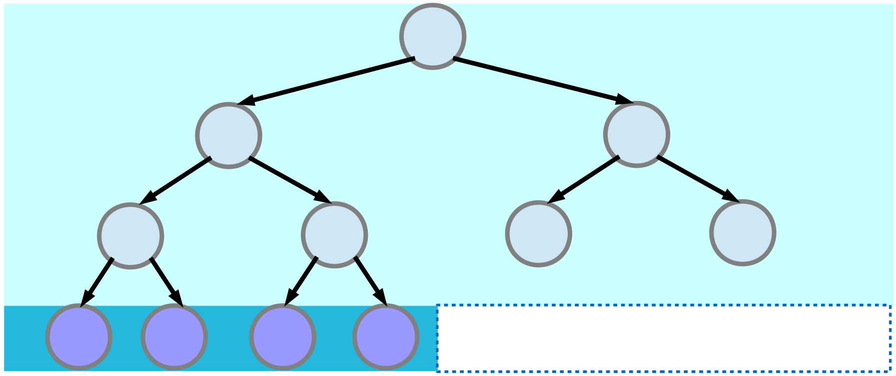


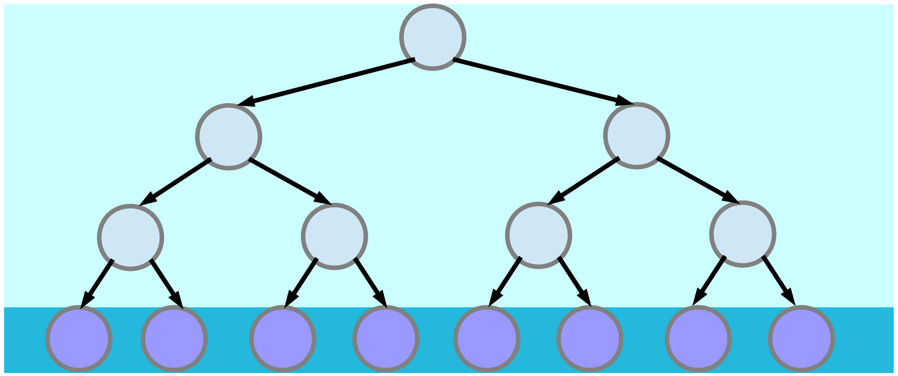
https://en.wikipedia.org/wiki/Tree_(graph_theory)

Complete Binary Trees

In a **complete binary tree**
every level, except possibly the last,
is completely filled,
and all nodes in the last level are
as far left as possible.

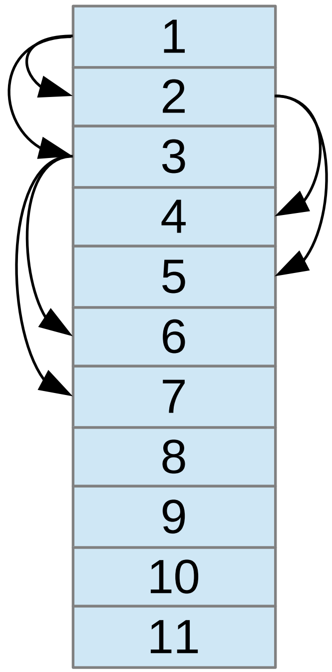
An alternative definition is a **perfect tree**
whose rightmost leaves (perhaps all)
have been removed.





https://en.wikipedia.org/wiki/Tree_(graph_theory)

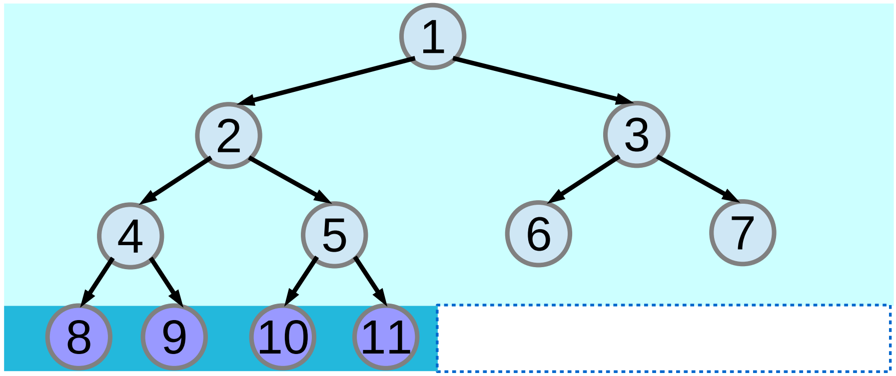
Complete Binary Trees and Linear Arrays

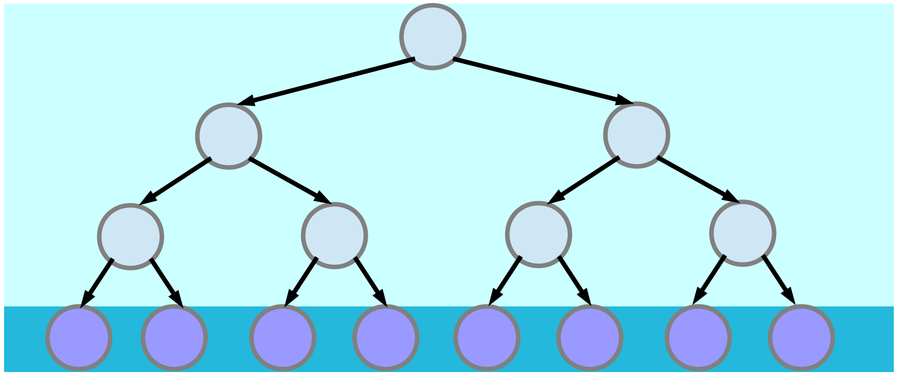


$2 \cdot i$ Left child

$2 \cdot i + 1$ Right child

A complete binary tree can be efficiently represented using an array.





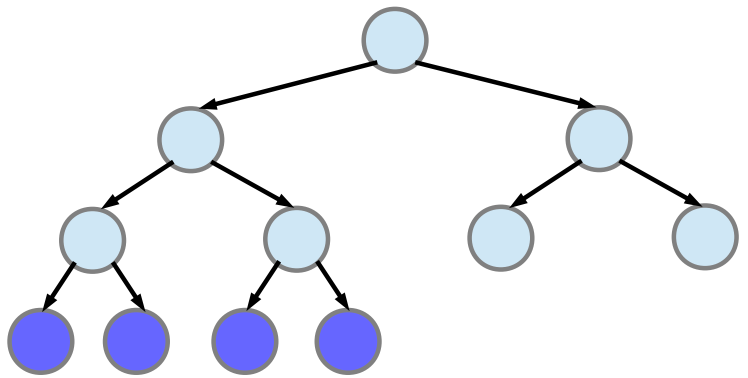
contiguous
no blanks
→ complete

https://en.wikipedia.org/wiki/Tree_(graph_theory)

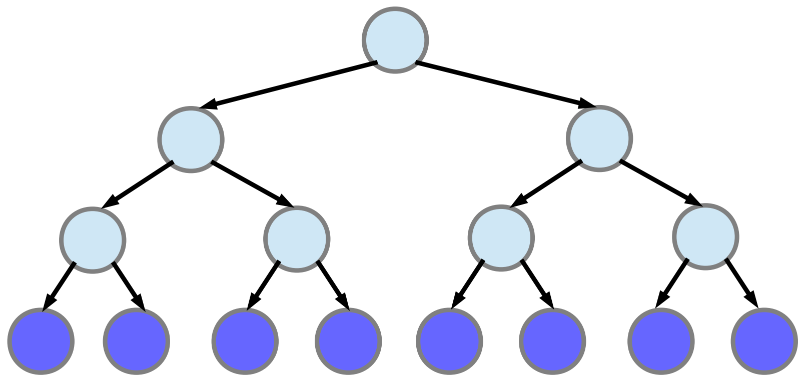
Different use of compute binary trees

Some authors use the term **complete** to refer instead to a **perfect** binary tree as defined above,
in which case they call this type of tree an **almost complete binary tree** or **nearly complete binary tree**.

complete	**nearly complete**



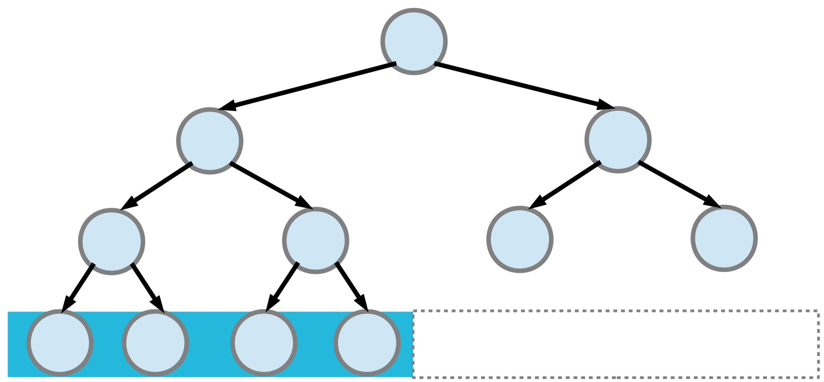
perfect	**complete**



https://en.wikipedia.org/wiki/Tree_(graph_theory)

Properties of Binary Trees (1)

A **complete binary tree**
can have between **1** and $\mathbf{2^{m-1}}$ nodes
at the last level **m**.

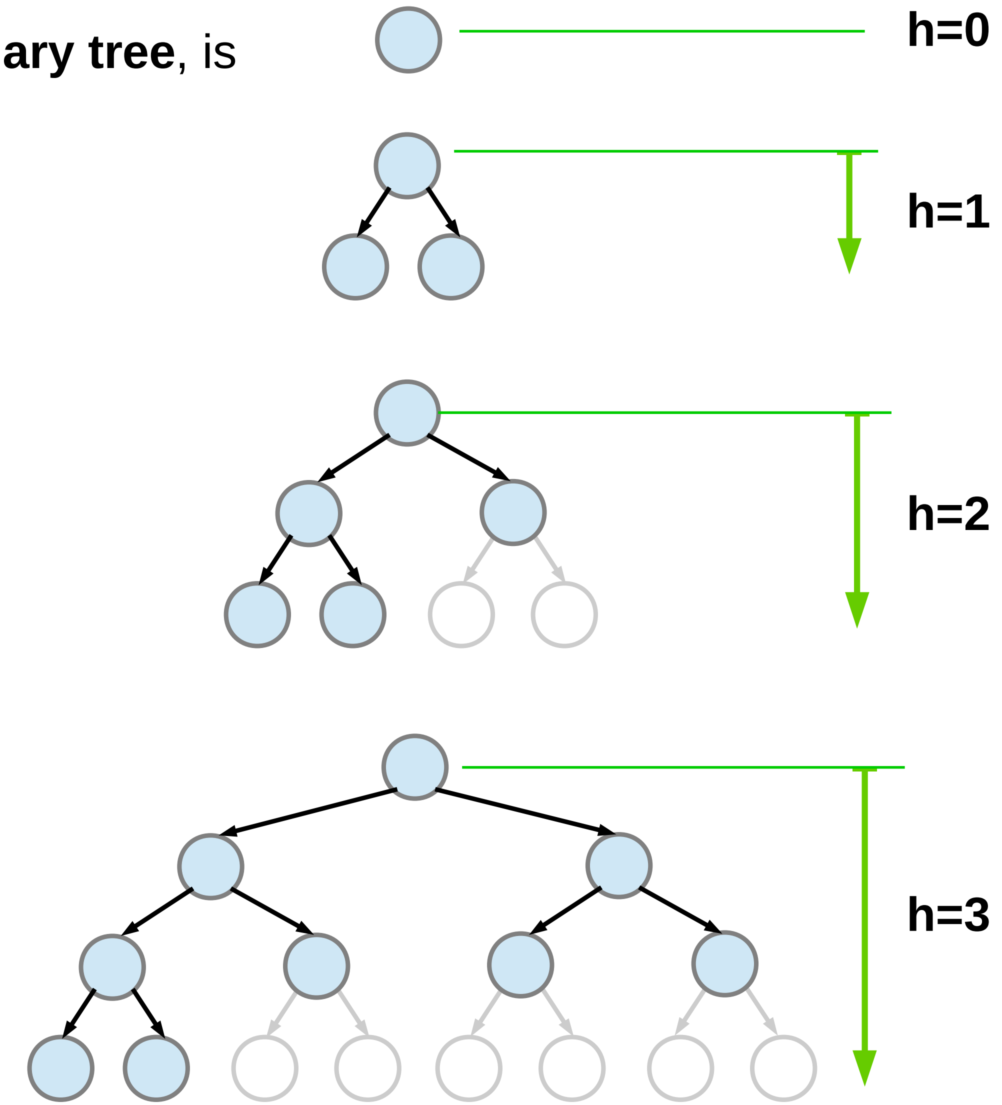


https://en.wikipedia.org/wiki/Tree_(graph_theory)

Properties of Binary Trees (2)

The **number** of **nodes n** in a **full binary tree**, is at least $\mathbf{n = 2^h + 1}$ and at most $\mathbf{n = 2^{h+1} - 1}$, where **h** is the **height** of the tree.

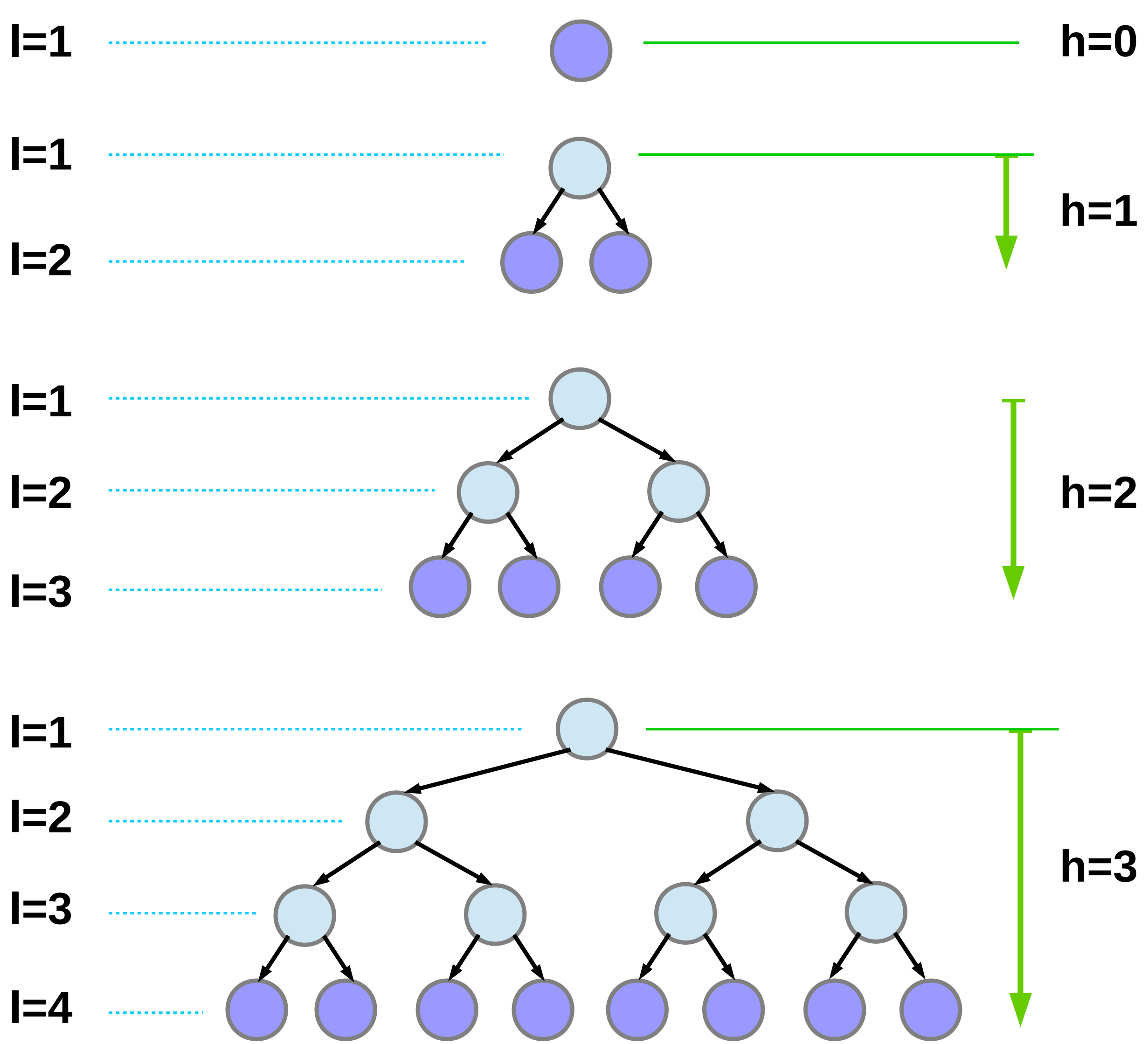
A tree consisting of only a **root node** has a **height** of **0**.



https://en.wikipedia.org/wiki/Tree_(graph_theory)

Properties of Binary Trees (3)

$2^{h+1}-1$	2^h	2^0
$2^{h+1}-1$	2^h-1	2^1-1
	2^h	2^1
$2^{h+1}-1$	2^h-1	2^2-1
	2^h	2^2
$2^{h+1}-1$	2^h-1	2^3-1
	2^h	2^3



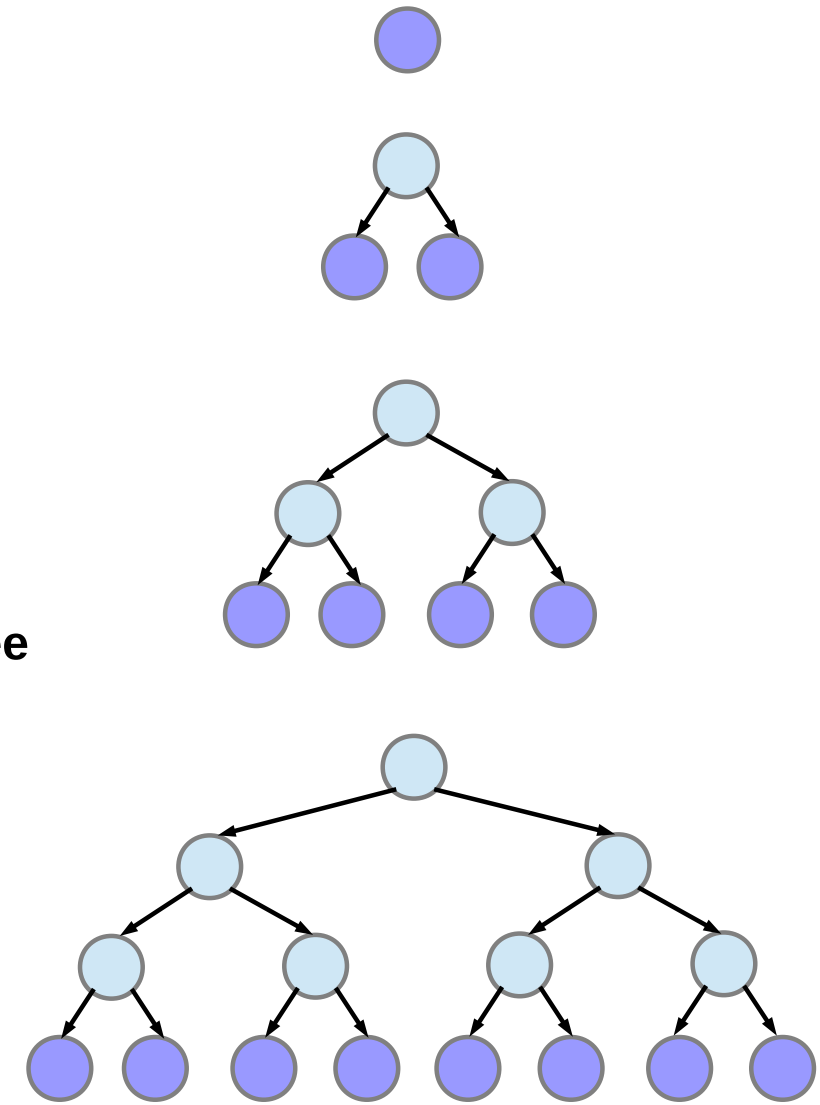
https://en.wikipedia.org/wiki/Tree_(graph_theory)

Properties of Binary Trees (4)

The number of **leaf nodes** is **m**
in a **perfect binary tree**,
is **m=(n+1)/2**

because the number of **non-leaf** (**internal**) **nodes** is **m–1**

This means that a **perfect binary tree** with **m leaves** has
n = 2m–1 nodes.



m	$\mathbf{2^0}$

m-1	$\mathbf{2^1\text{-}1}$
m	$\mathbf{2^1}$

m-1	$\mathbf{2^2\text{-}1}$
m	$\mathbf{2^2}$

m-1	$\mathbf{2^3\text{-}1}$
m	$\mathbf{2^3}$

https://en.wikipedia.org/wiki/Tree_(graph_theory)

References

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

Binary Search Tree (3A)

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

This document was produced by using LibreOffice and Octave.

Binary Search Tree (1)

Binary search trees (BST),
ordered binary trees
sorted binary trees

are a particular type of **container**:
data structures that store "items"
(such as numbers, names etc.) in memory.

They allow fast **lookup**, **addition** and **removal** of items
can be used to implement either dynamic sets of items
lookup tables that allow finding an item by its **key**
(e.g., finding the phone number of a person by name).

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

Binary Search Tree (2)

keep their **keys** in <u>sorted</u> <u>order</u>
lookup operations can use
the principle of **binary search**

allowing to <u>skip</u> searching <u>half</u> of the tree
each operation (**lookup**, **insertion** or **deletion**)
takes time proportional to **log n**

much better than the **linear time**
but slower than the corresponding operations
on **hash tables**.

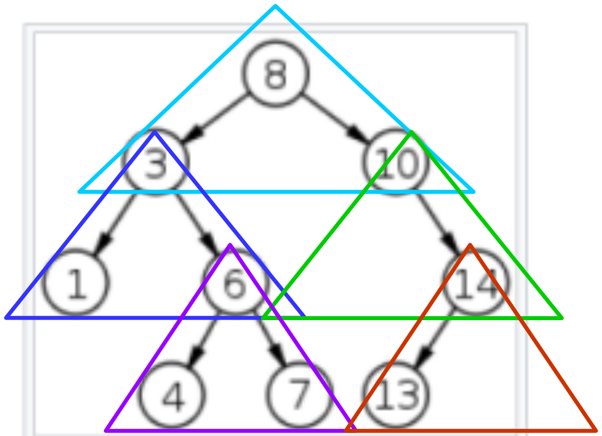
https://en.wikipedia.org/wiki/Binary_search_tree

Binary Search Tree (3)

when **looking** for a **key** in a tree
or **looking** for a **place** to insert a new key,
they traverse the tree from root to leaf,
making comparisons to keys stored in the nodes
deciding to continue in the **left** or **right subtrees**,
on the basis of the comparison.

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

Node, Left Child, Right Child



A binary search tree of size 9 and depth 3, with 8 at the root. The leaves are not drawn.

$3 < 8 < 10$

$1 < 3 < 6$

$10 < 14$

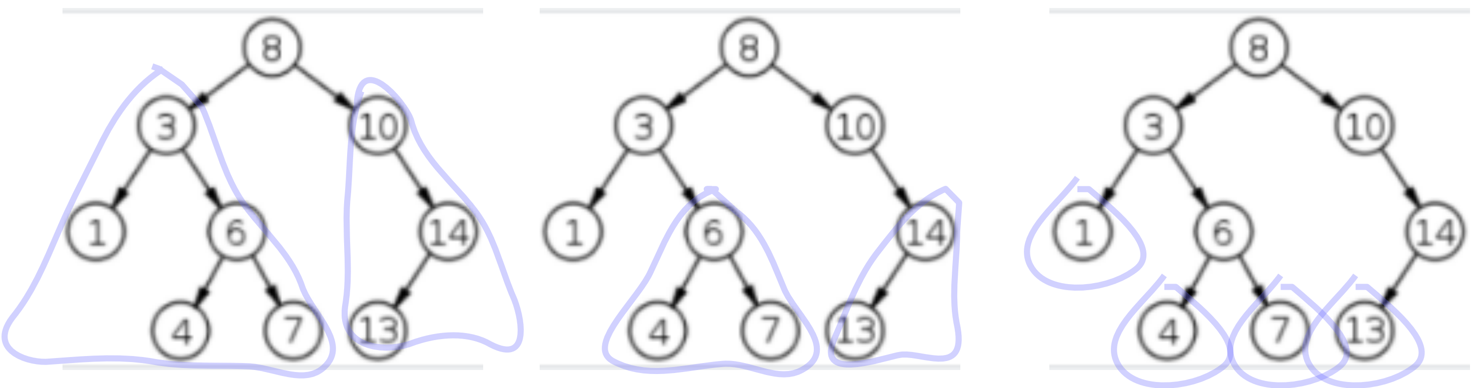
$4 < 6 < 7$

$13 < 14$

1, 3, 4, 6, 7, 8, 10, 13, 14

https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

Subtrees

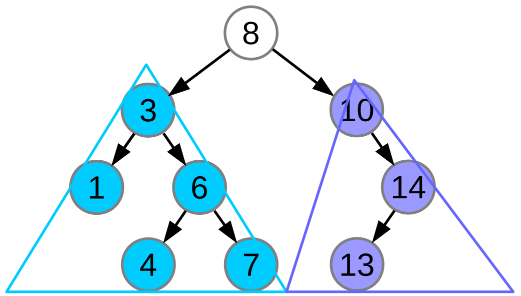


1, 3, 4, 6, 7, 8, 10, 13, 14

https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

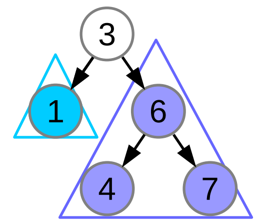
Node, Left Subtree, Right Subtree

$\boxed{1, 3, 4, 6, 7} < 8 < \boxed{10, 13, 14}$

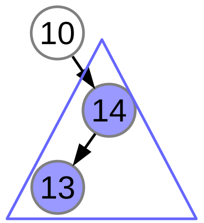


1, 3, 4, 6, 7, 8, 10, 13, 14

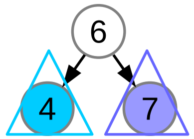
$\boxed{1} < 3 < \boxed{4, 6, 7}$



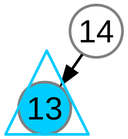
$10 < \boxed{13, 14}$



$\boxed{4} < 6 < \boxed{7}$

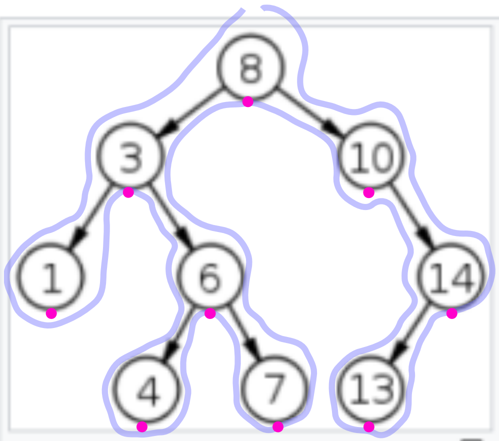


$\boxed{13} < 14$



https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

In-Order Traversal

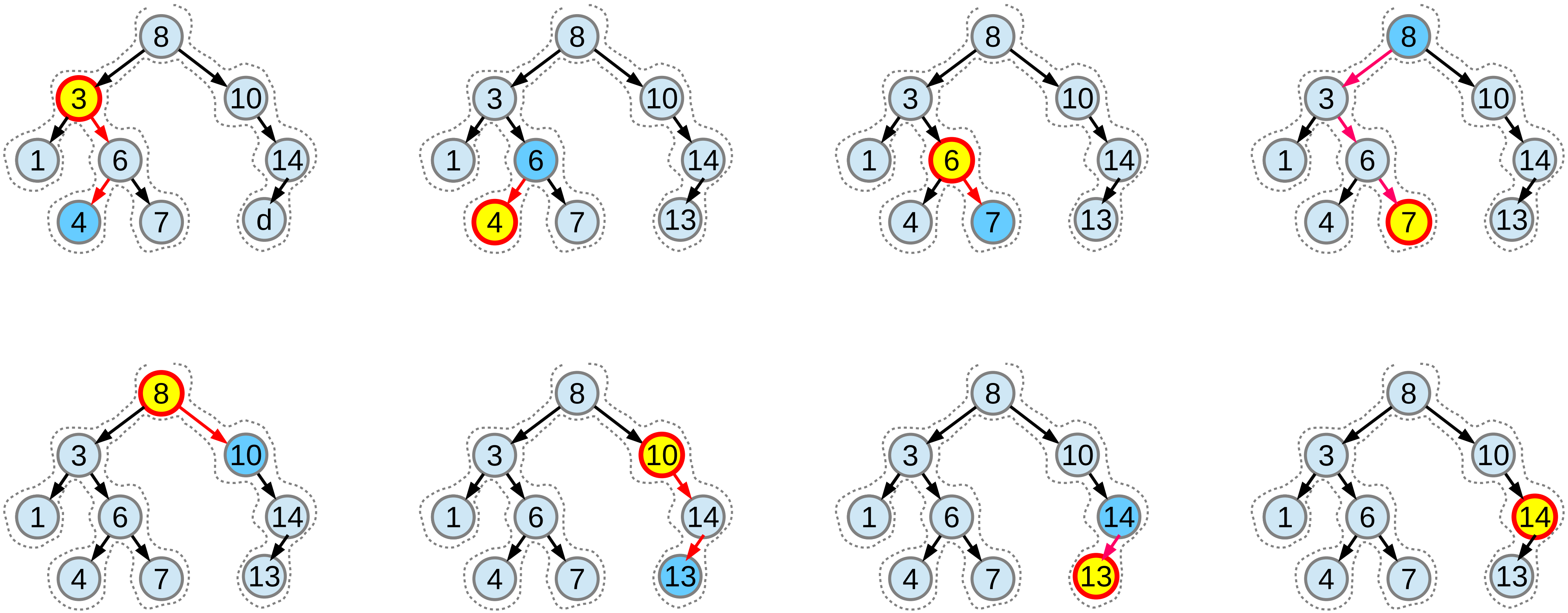


A binary search tree of size 9 and depth 3, with 8 at the root. The leaves are not drawn.

1, 3, 4, 6, 7, 8, 10, 13, 14

https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

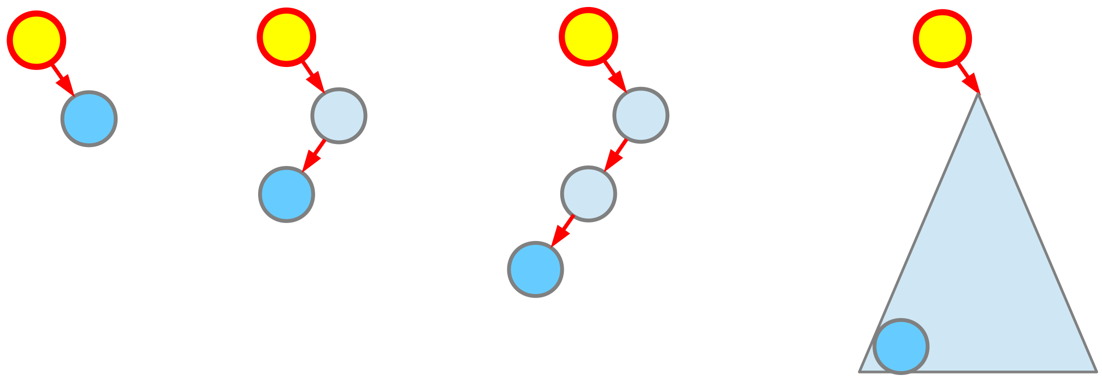
Successor



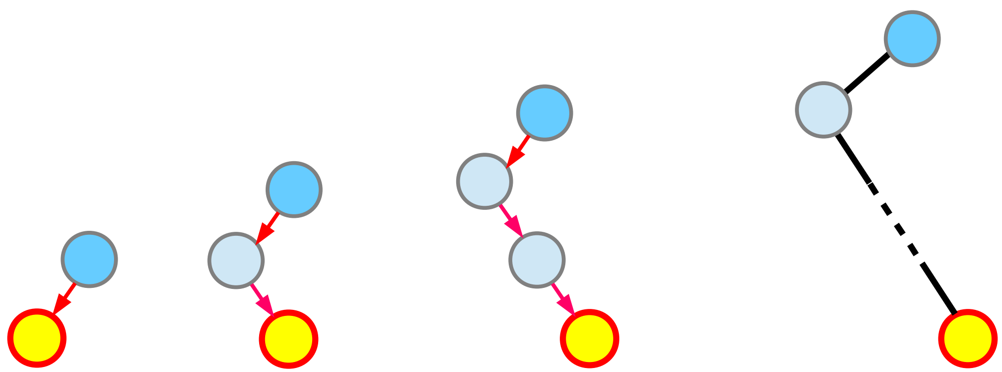
1, 3, 4, 6, 7, 8, 10, 13, 14

https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

Successor



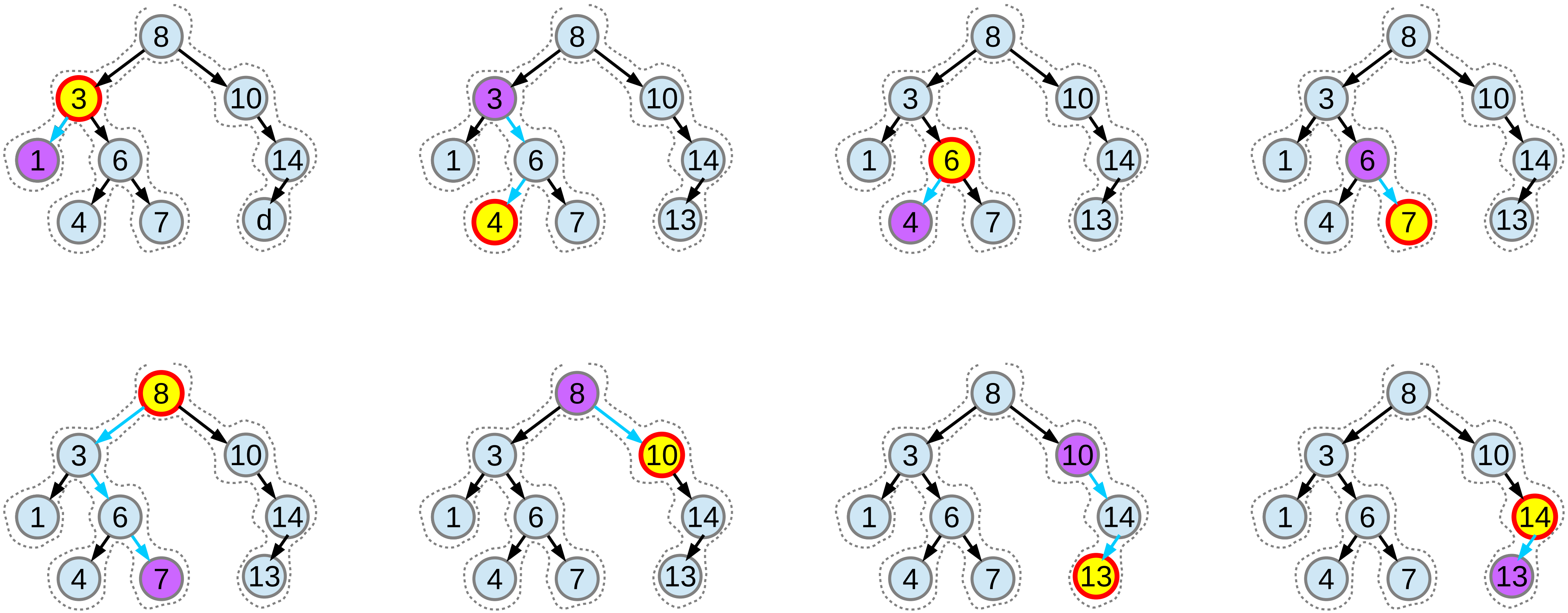
If the right child exists,
then the minimum
in the right subtree
– the rightmost node



the parent of the farthest
node that can be reached
by following only right
edges backward.

https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

Predecessor

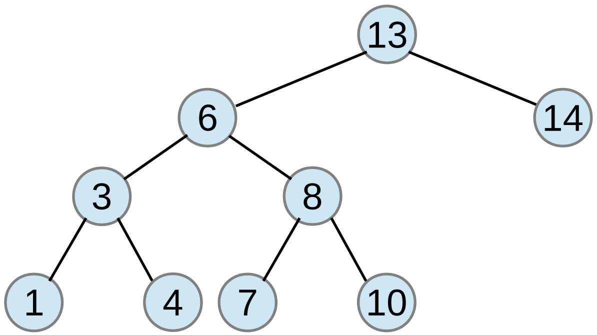


1, 3, 4, 6, 7, 8, 10, 13, 14

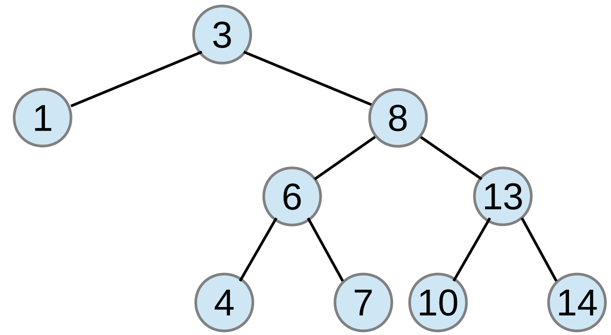
https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

Different BST's with the same data

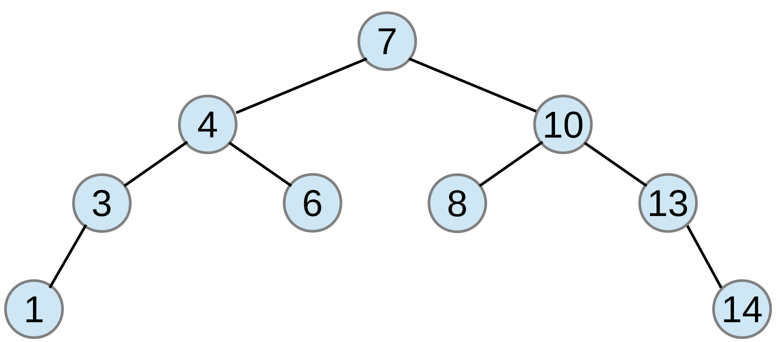
1, 3, 4, 6, 7, 8, 10, 13, 14



1, 3, 4, 6, 7, 8, 10, 13, 14

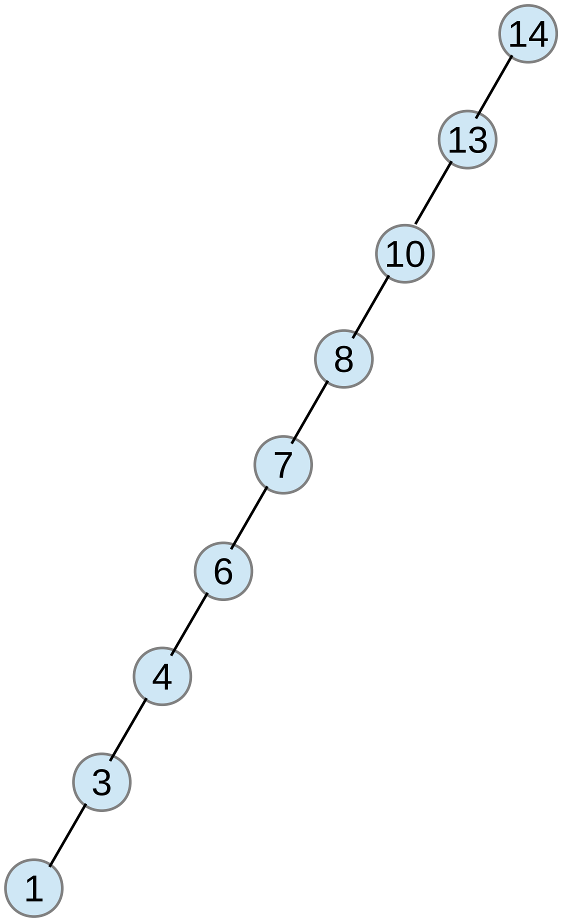


1, 3, 4, 6, 7, 8, 10, 13, 14

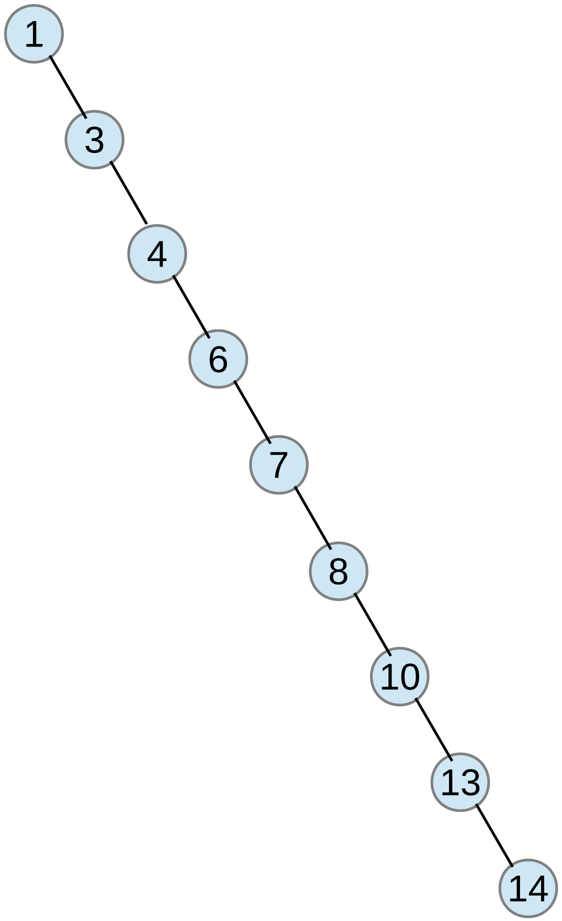


Unbalanced BSTs

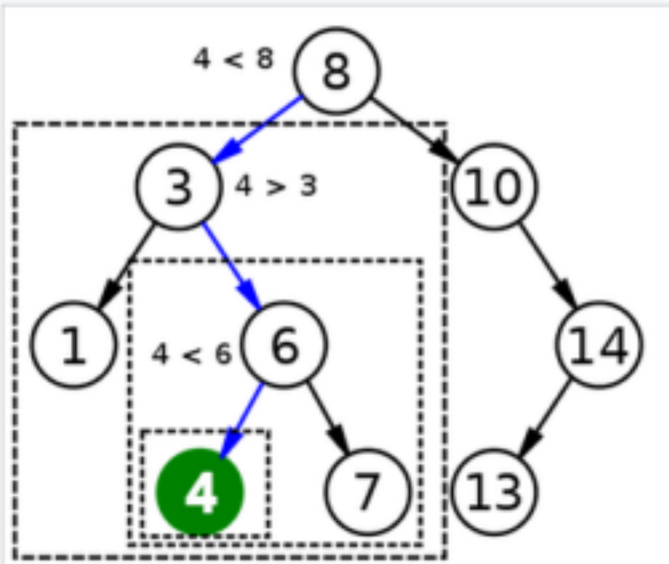
1, 3, 4, 6, 7, 8, 10, 13, 14



1, 3, 4, 6, 7, 8, 10, 13, 14



Binary Search on a Binary Search Tree



Binary search trees are searched using an algorithm similar to binary search.

1, 3, 4, 6, 7, 8, 10, 13, 14

1, 3, 4, 6, 7, 8, 10, 13, 14

1, 3, 4, 6, 7, 8, 10, 13, 14

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

Insertion

Insertion begins as a search would begin;
if the key is not equal to that of the root,
we search the left or right subtrees as before.
Eventually, we will reach an external node
and add the new key-value pair
(here encoded as a record 'newNode')
as its right or left child,
depending on the node's key.

In other words, we examine the root
and recursively insert the new node
to the left subtree if its key is less than that of the root,
or the right subtree if its key is greater than or equal to the root.

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

Deletion

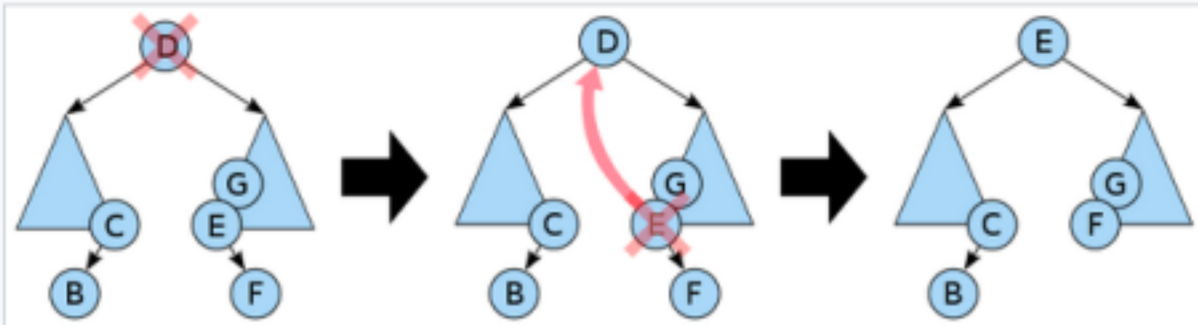
1. Deleting a node with no children:
   simply remove the node from the tree.

2. Deleting a node with one child:
   remove the node and replace it with its child.

3. Deleting a node with two children:
   call the node to be deleted D.
   Do not delete D.
   Instead, choose either its in-order predecessor node
   or its in-order successor node as replacement node E.
   Copy the user values of E to D
   If E does not have a child
      simply remove E from its previous parent G.
   If E has a child, say F, it is a right child.
      Replace E with F at E's parent.

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

Deletion



Deleting a node with two children from a binary search tree. First the leftmost node in the right subtree, the in-order successor *E*, is identified. Its value is copied into the node *D* being deleted. The in-order successor can then be easily deleted because it has at most one child. The same method works symmetrically using the in-order predecessor *C*.

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

References

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

Finite State Machine (1A)

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

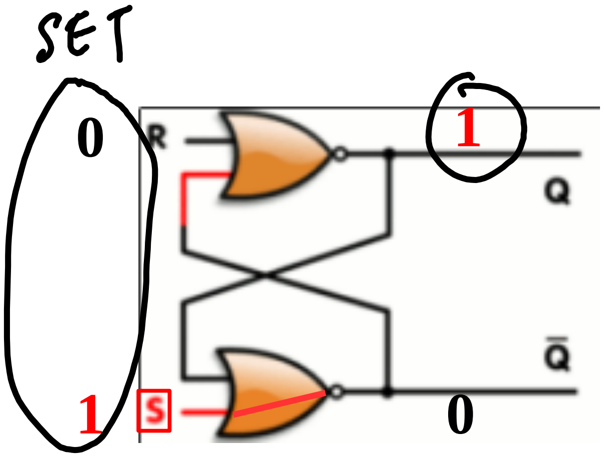
This document was produced by using LibreOffice and Octave.

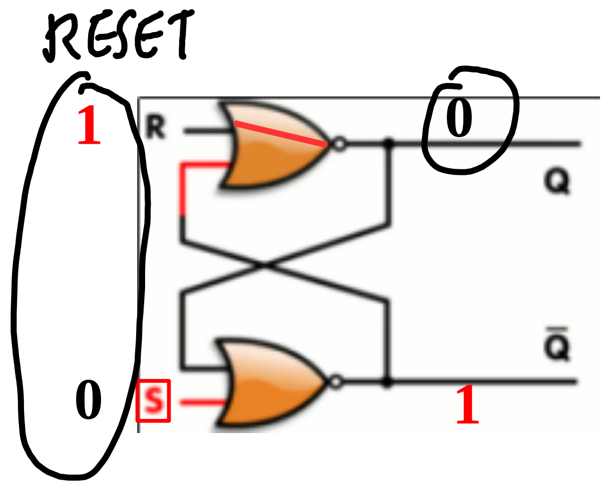
FSM and Digital Logic Circuits

- Latch
- D FlipFlop
- Registers
- Timing
- Mealy machine
- Moore machine
- Traffic Lights Examples

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

NOR-based SR Latch – SET / RESET





SET	S=**1** R=0

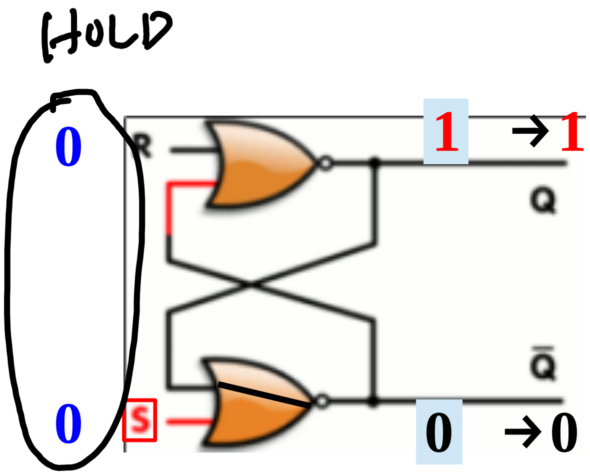
Q=1
$\overline{Q}$=0

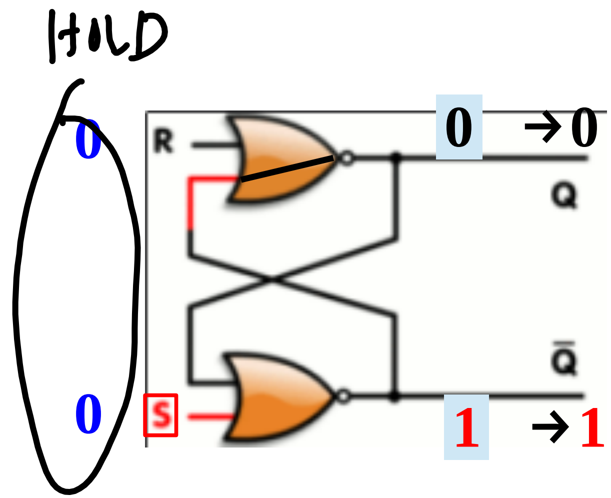
RESET	S=0 R=**1**

Q=0
$\overline{Q}$=1

https://en.wikipedia.org/wiki/Flip-flop_(electronics)

NOR-based SR Latch – HOLD





HOLD	S=0 R=0

Q=old Q

$\overline{Q}$=old $\overline{Q}$

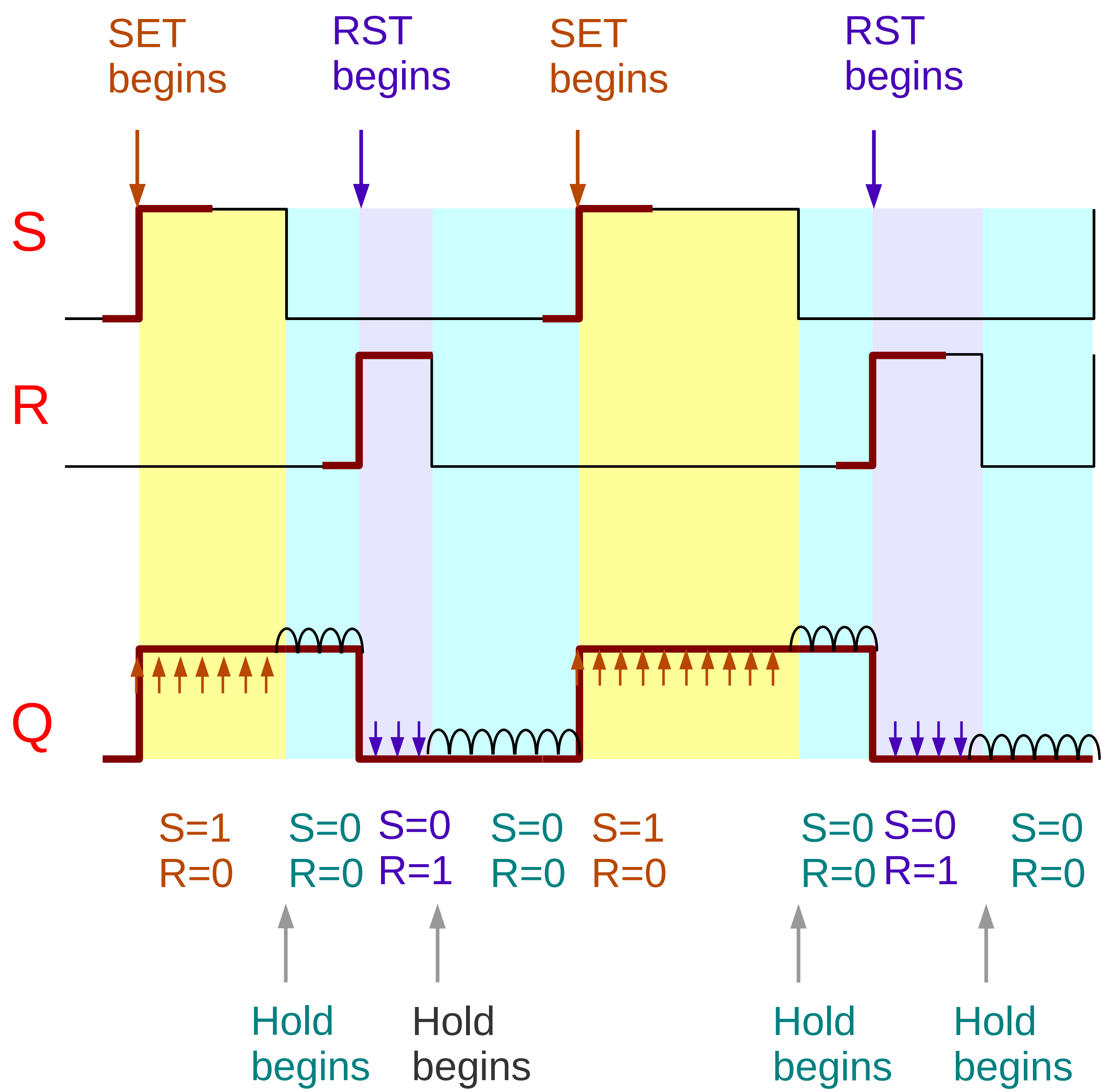
HOLD	S=0 R=0

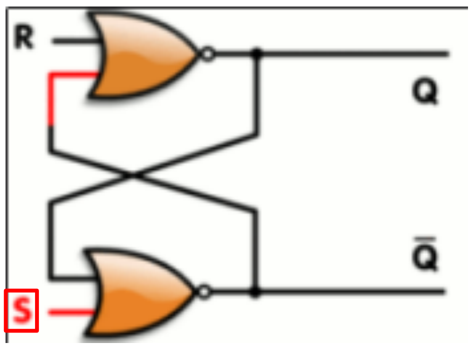
Q=old Q

$\overline{Q}$=old $\overline{Q}$

https://en.wikipedia.org/wiki/Flip-flop_(electronics)

NOR-based SR Latch

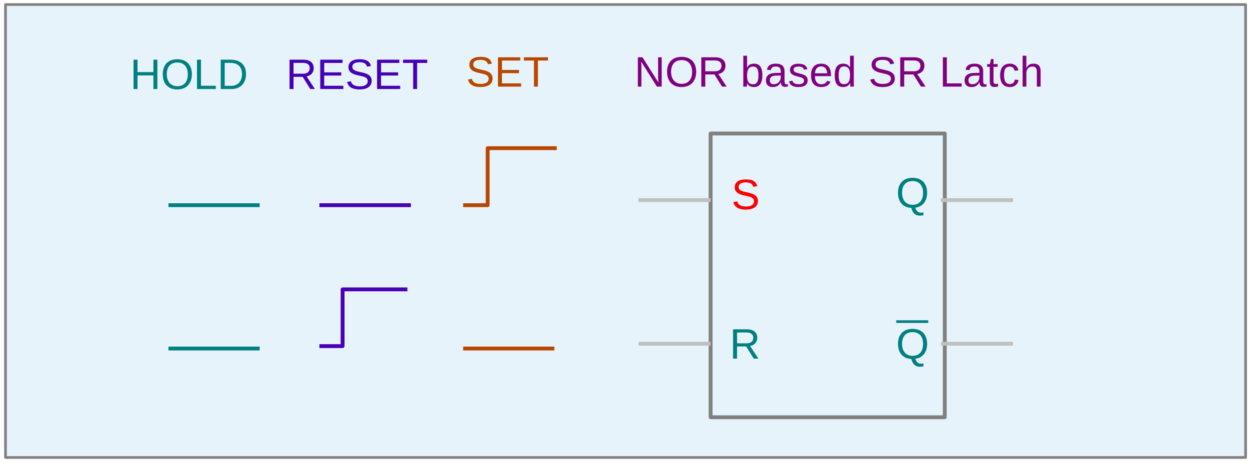


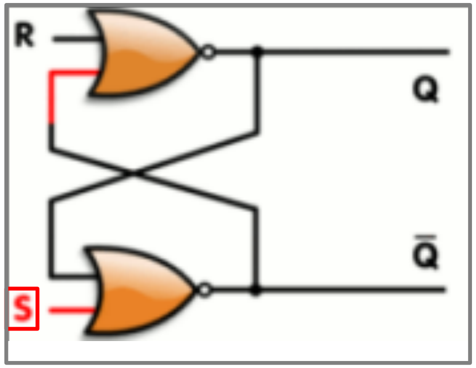


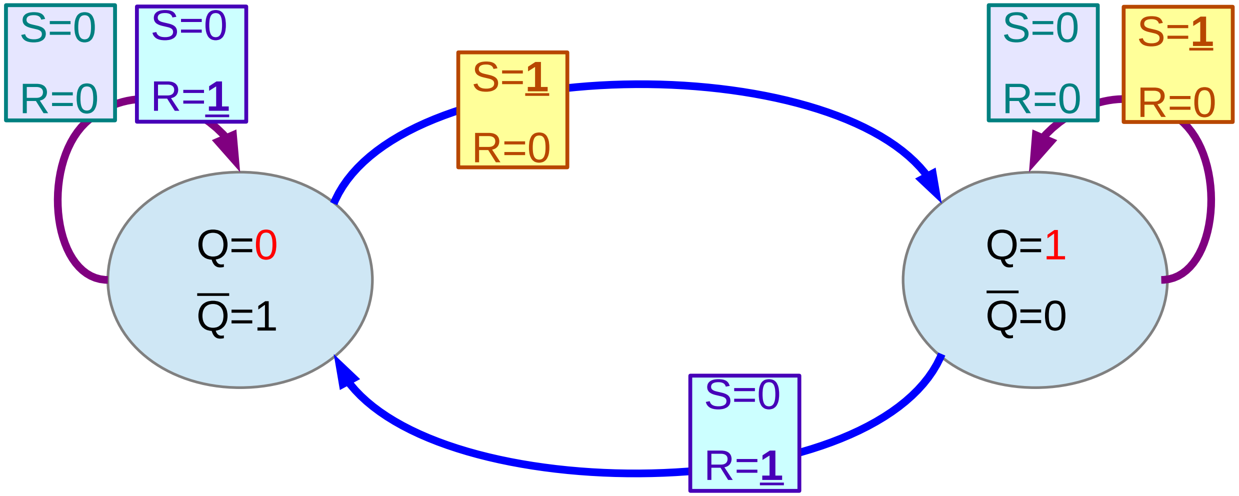
SET	S=**1** R=0	Q=1 $\overline{Q}$=0
RESET	S=0 R=**1**	Q=0 $\overline{Q}$=1
HOLD	S=0 R=0	Q=old Q $\overline{Q}$=old $\overline{Q}$

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

NOR-based SR Latch States



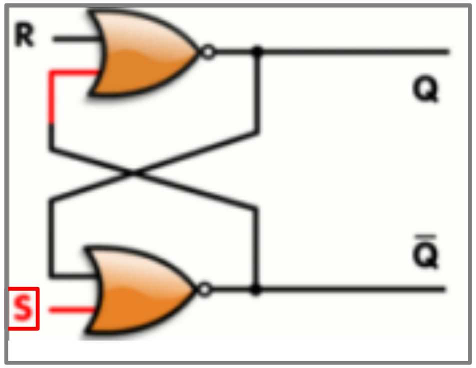


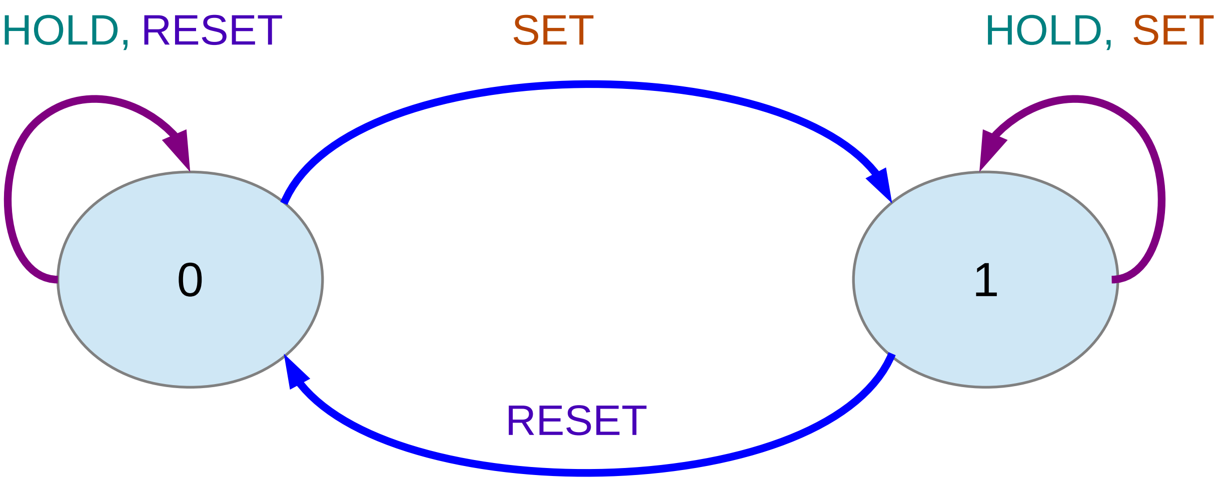


SET	S=1 R=0	Q=1 $\overline{Q}$=0
RESET	S=0 R=1	Q=0 $\overline{Q}$=1
HOLD	S=0 R=0	Q=old Q $\overline{Q}$=old $\overline{Q}$

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

SR Latch States





SET: S=**1**, R=0 — Q=1, $\overline{Q}$=0

RESET: S=0, R=**1** — Q=0, $\overline{Q}$=1

HOLD: S=0, R=0 — Q=old Q, $\overline{Q}$=old $\overline{Q}$

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

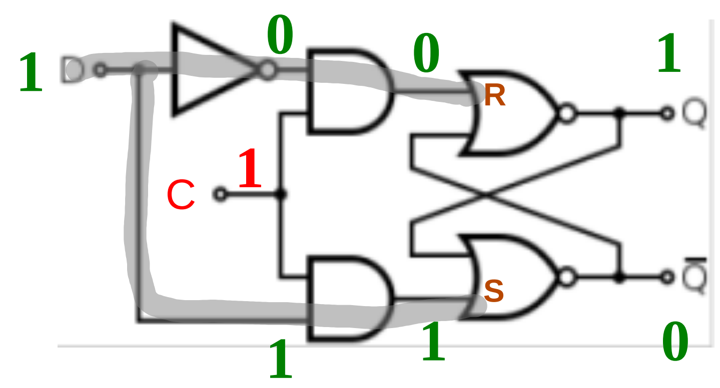
NOR-based D Latch – SET / RESET

https://en.wikipedia.org/wiki/Flip-flop_(electronics)

D=**1**
C=1

SET S=**1** R=0

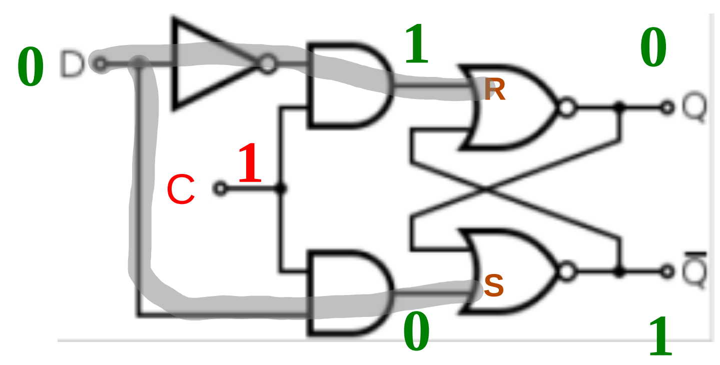
Q=1
$\overline{Q}$=0



D=**0**
C=1

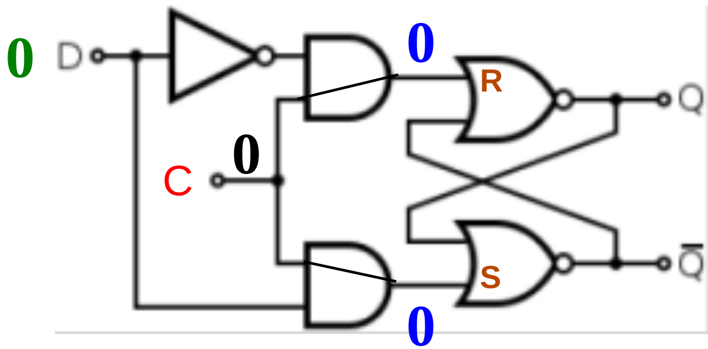
RESET S=0 R=**1**

Q=0
$\overline{Q}$=1



NOR-based D Latch – HOLD

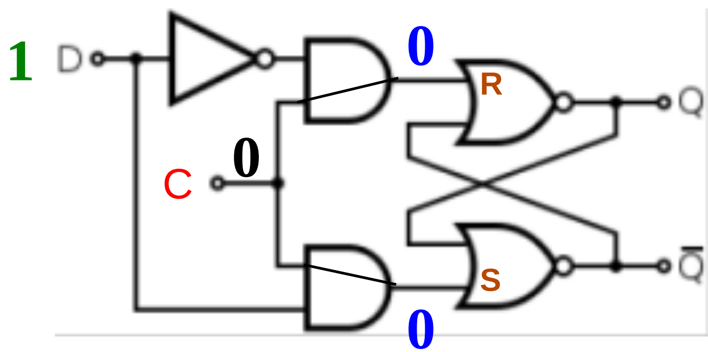
https://en.wikipedia.org/wiki/Flip-flop_(electronics)



D=**X**
C=0

HOLD
S=0
R=0

Q=old Q
$\overline{Q}$=old $\overline{Q}$

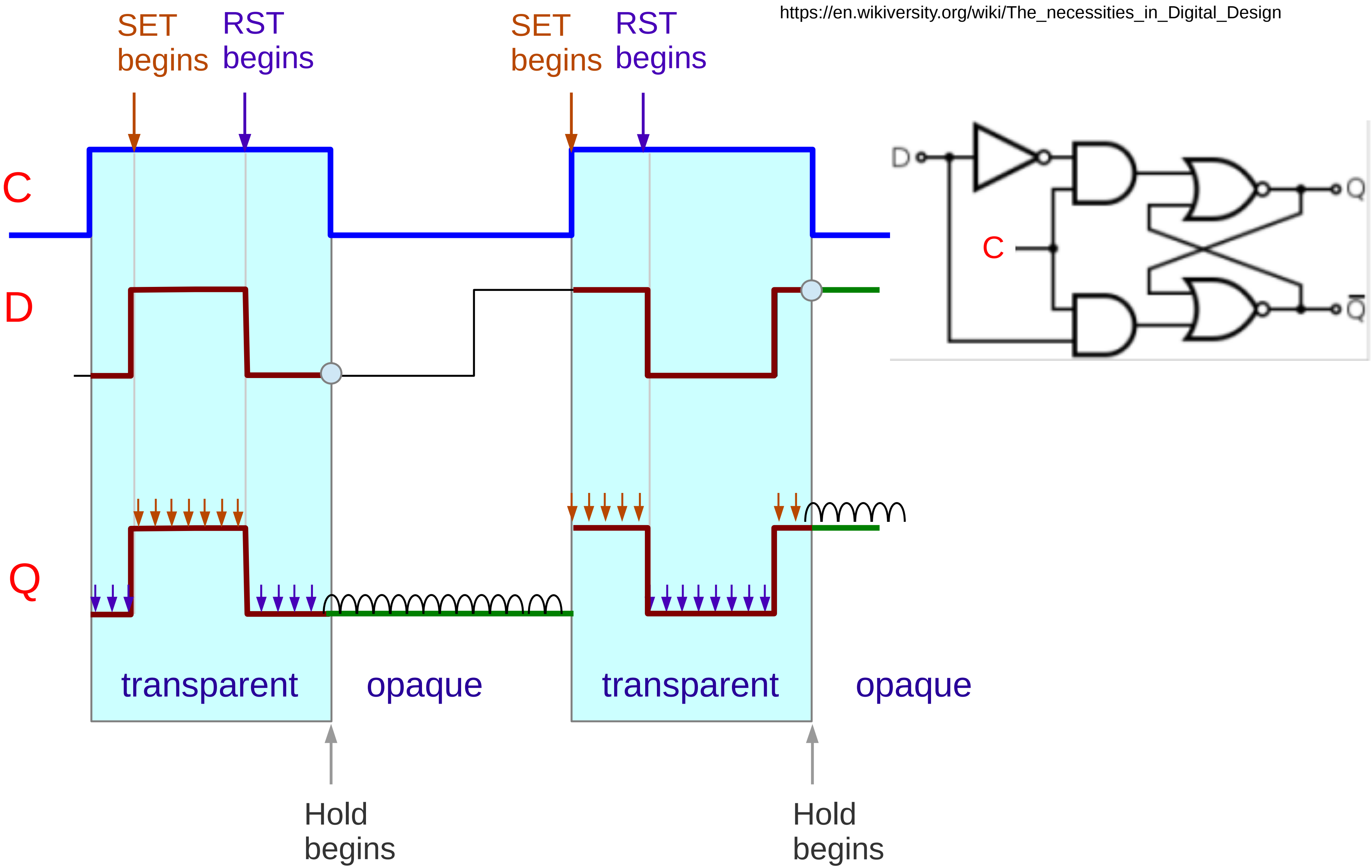


D=**X**
C=0

HOLD
S=0
R=0

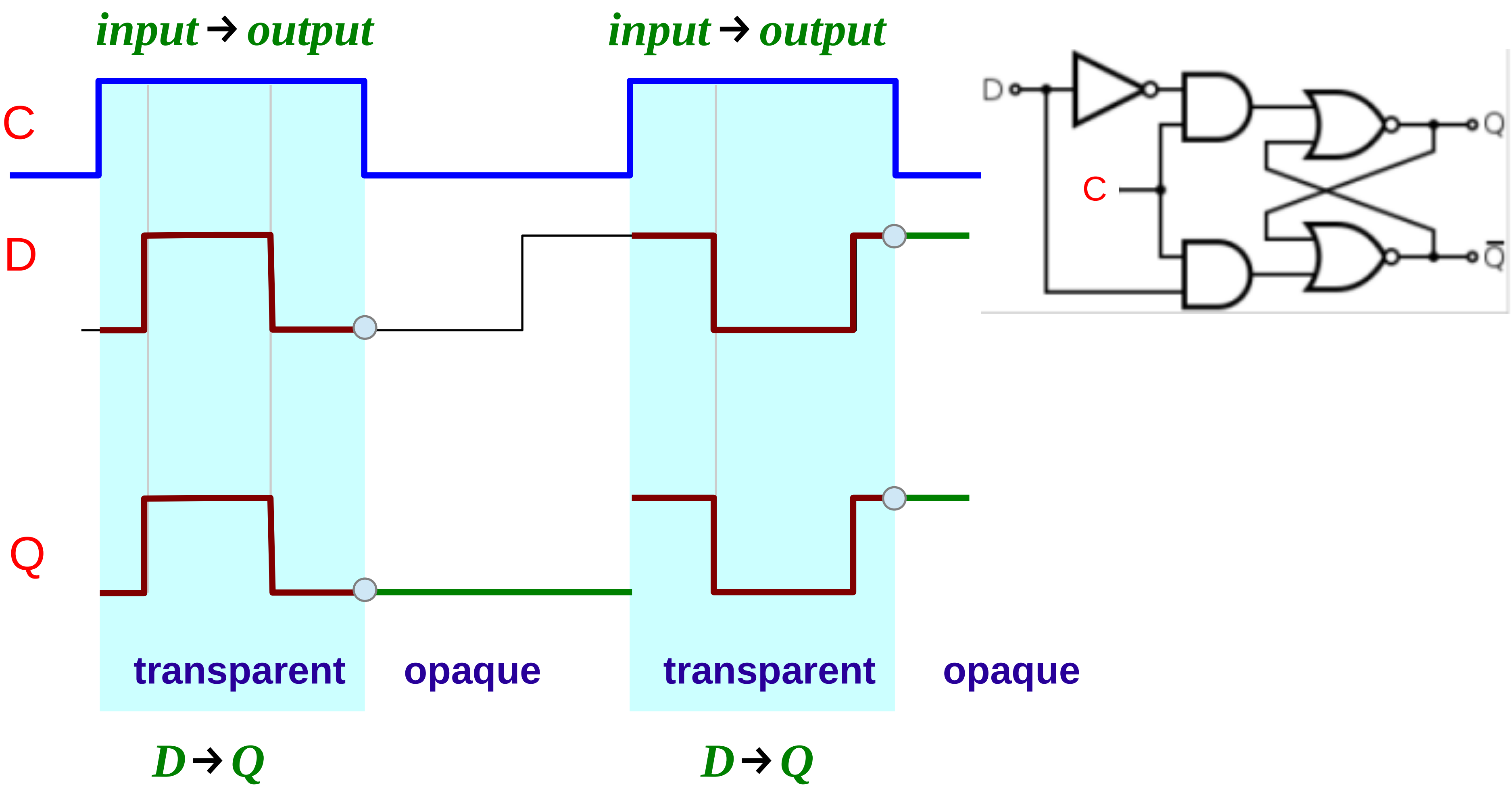
Q=old Q
$\overline{Q}$=old $\overline{Q}$

NOR-based D Latch – Set / Reset / Hold

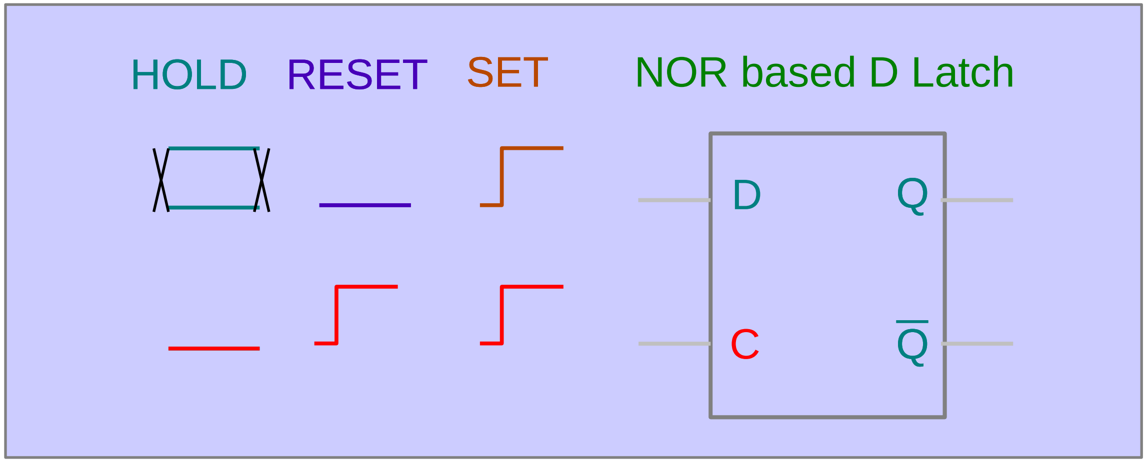


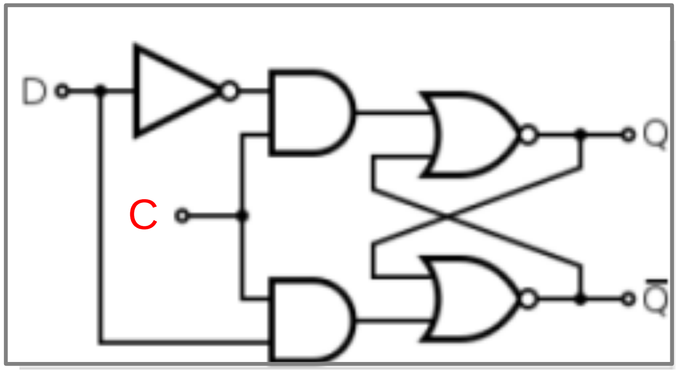
NOR-based D Latch – transparent / opaque

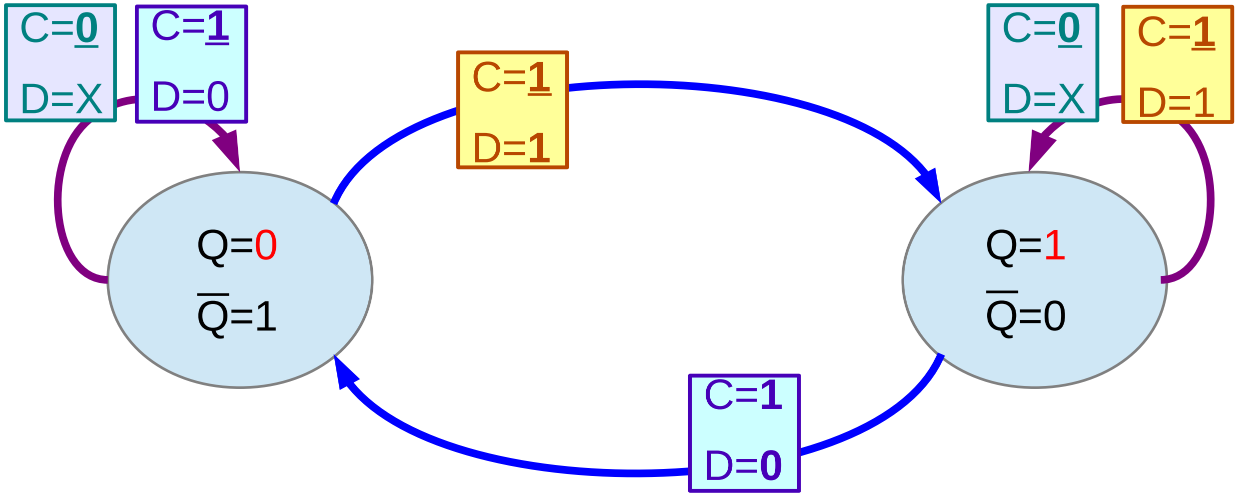
https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design



NOR-based D Latch States



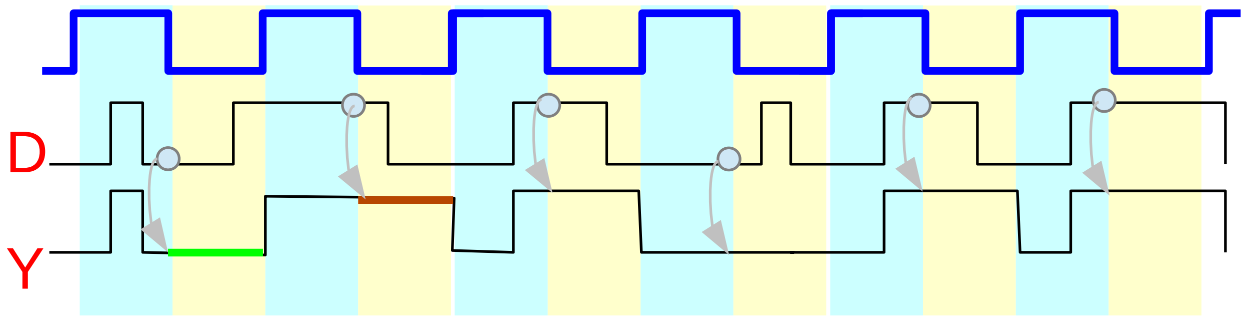


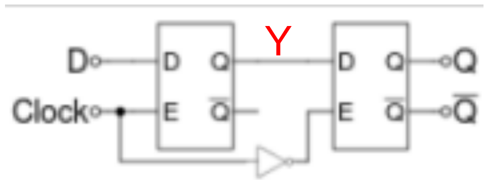


https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

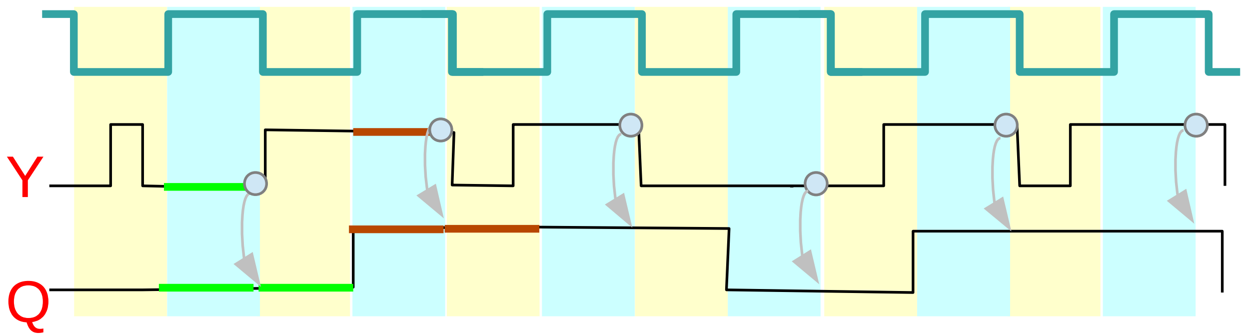
Master-Slave D FlipFlop

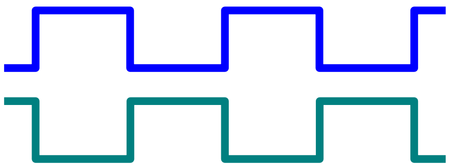
Master D Latch

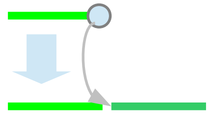




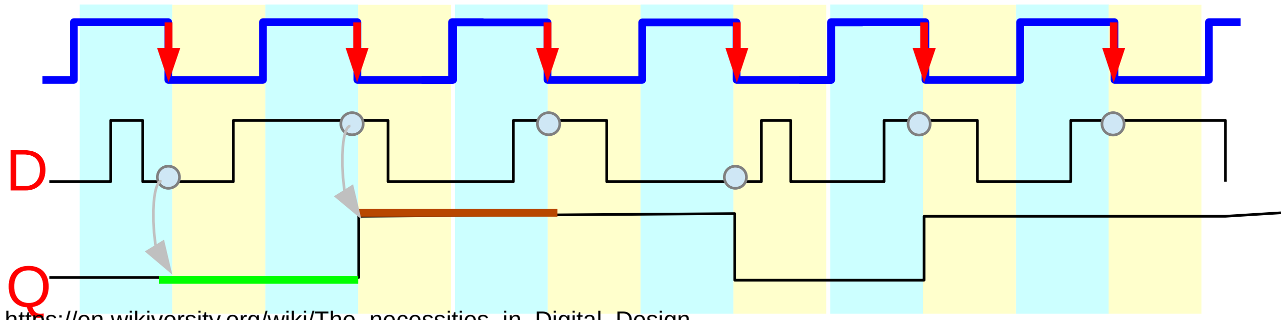
Slave D Latch







Master-Slave D F/F



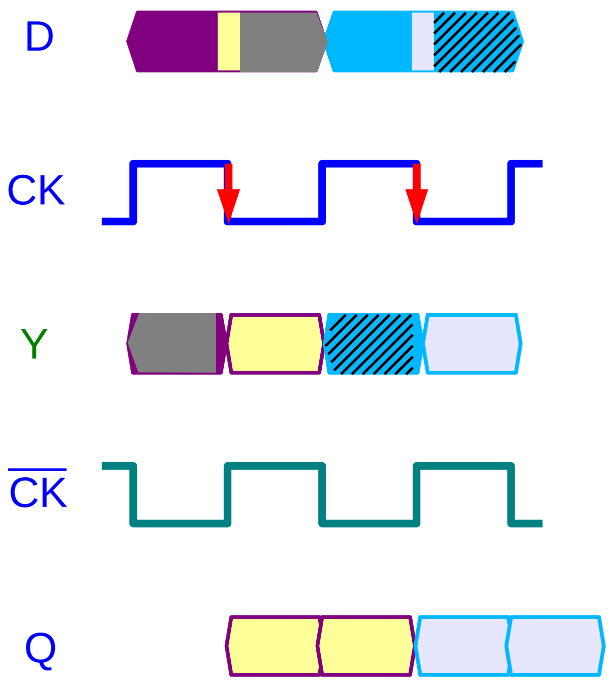
the hold output of the master is transparently reaches the output of the slave

this value is held for another half period

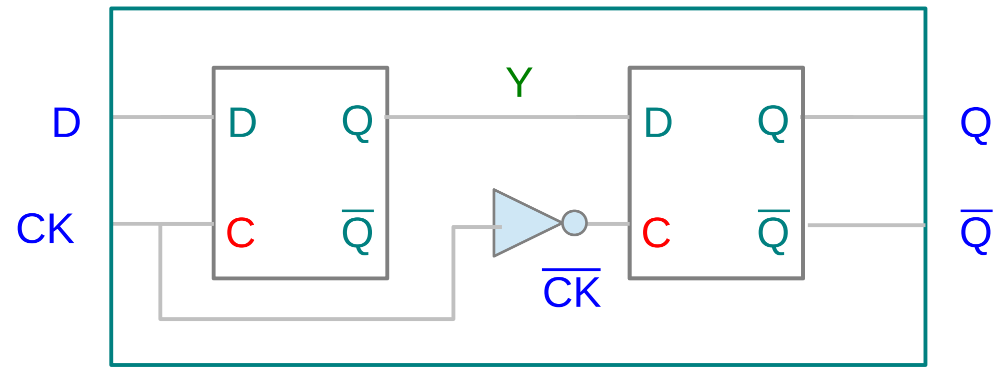
https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

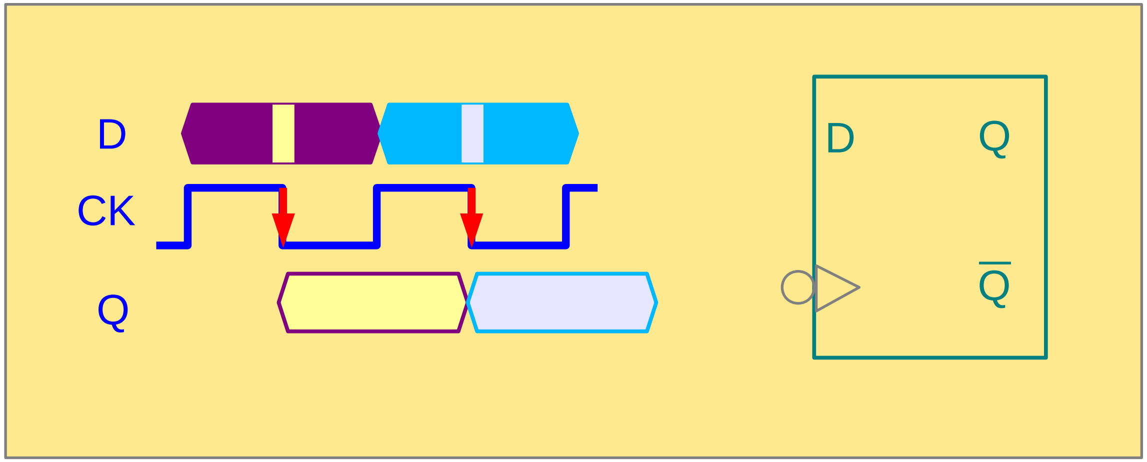
Master-Slave D FlipFlop – Falling Edge

Master D Latch



Slave D Latch



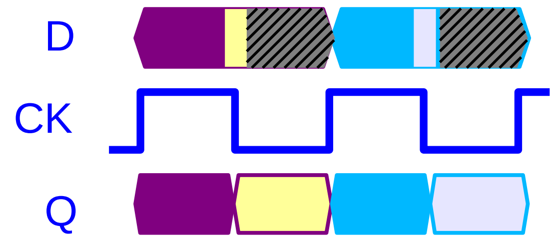


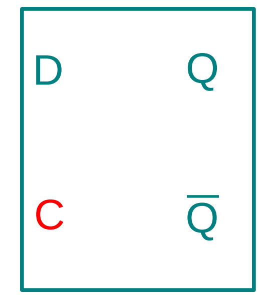
https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

D Latch & D FlipFlop

Level Sensitive D Latch

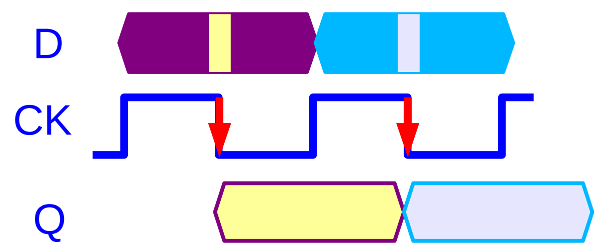
CK=1 transparent
CK=0 opaque

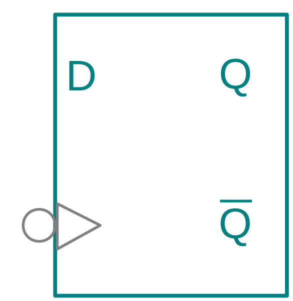




Edge Sensitive D FlipFlop

CK=1→0 transparent
else opaque

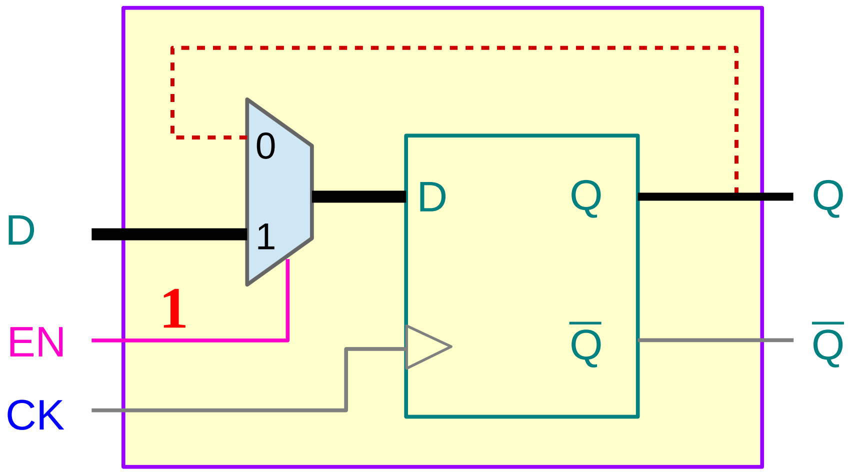




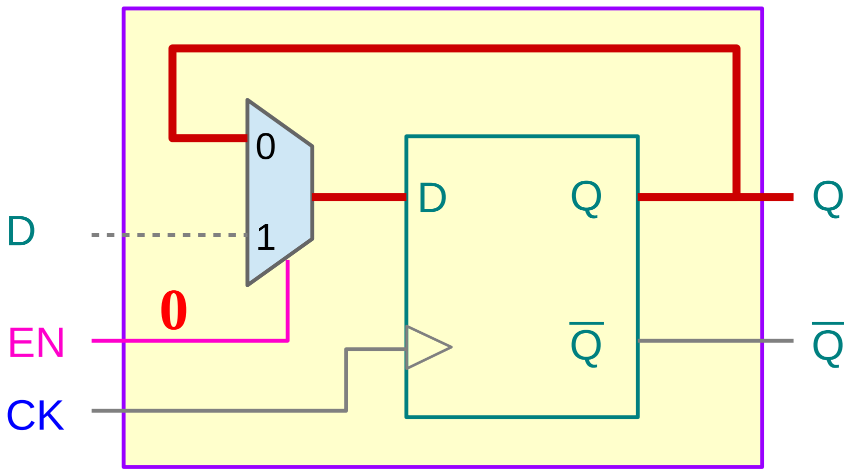
https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

D FlipFlop with Enable (1)

EN=1 Regular D Flip Flop
Sampling **D** input @ **posedge** of CK

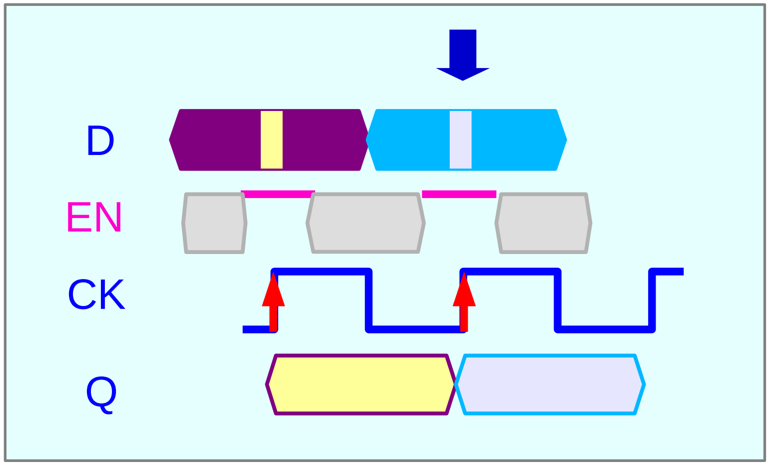


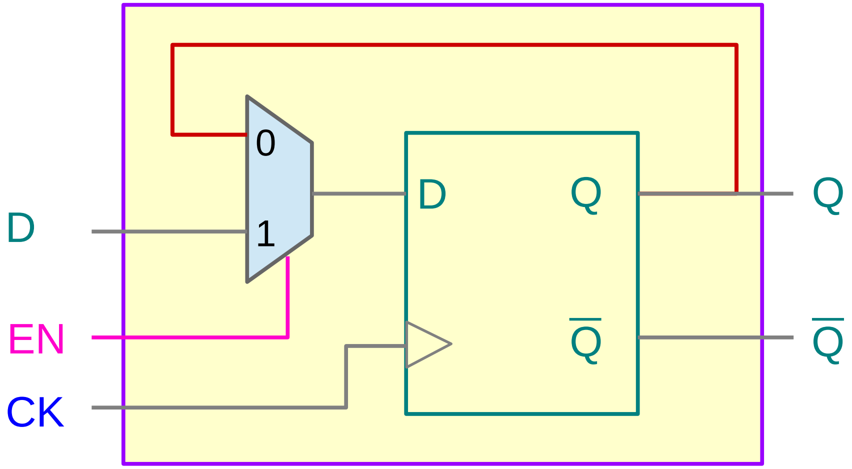
EN=0 Holding D Flip Flop
Sampling **Q** output @ **posedge** of CK

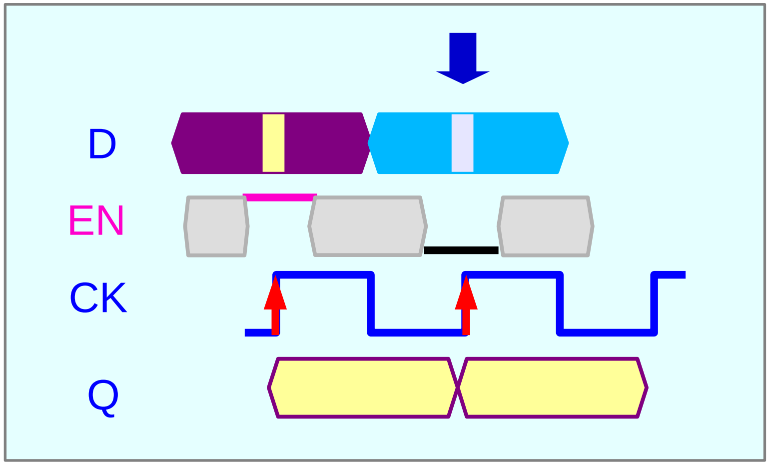


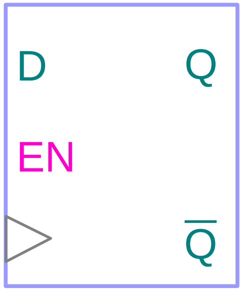
https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

D FlipFlop with Enable (2)



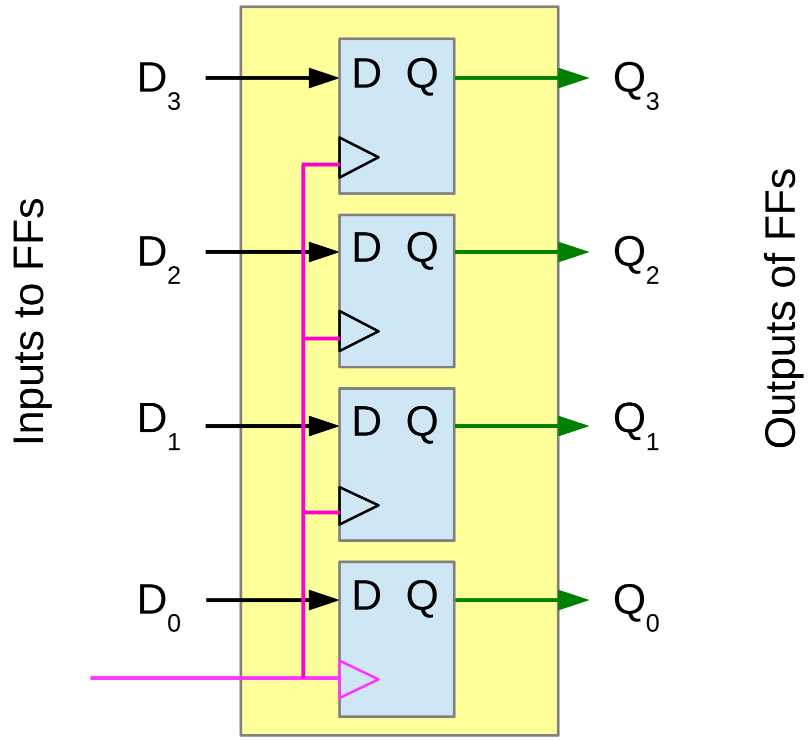


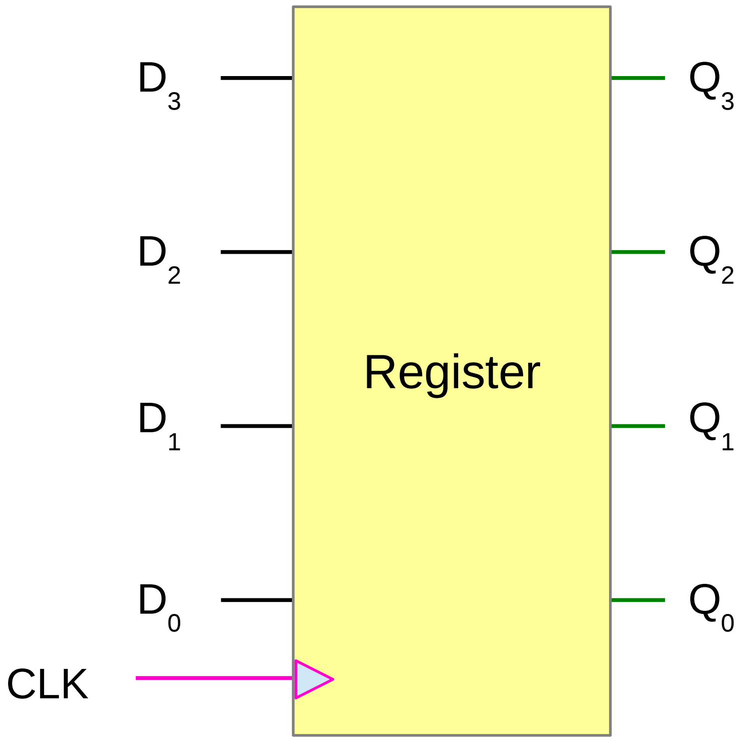




https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

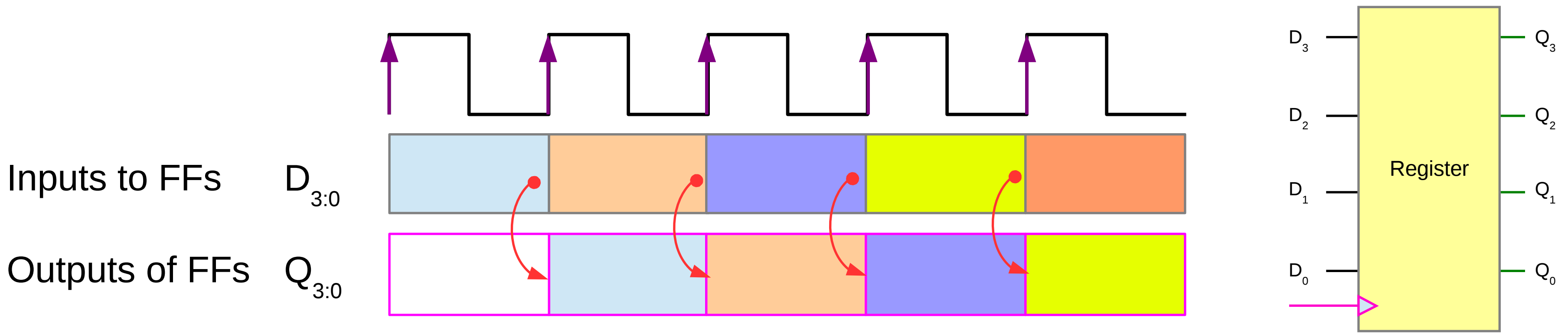
Registers





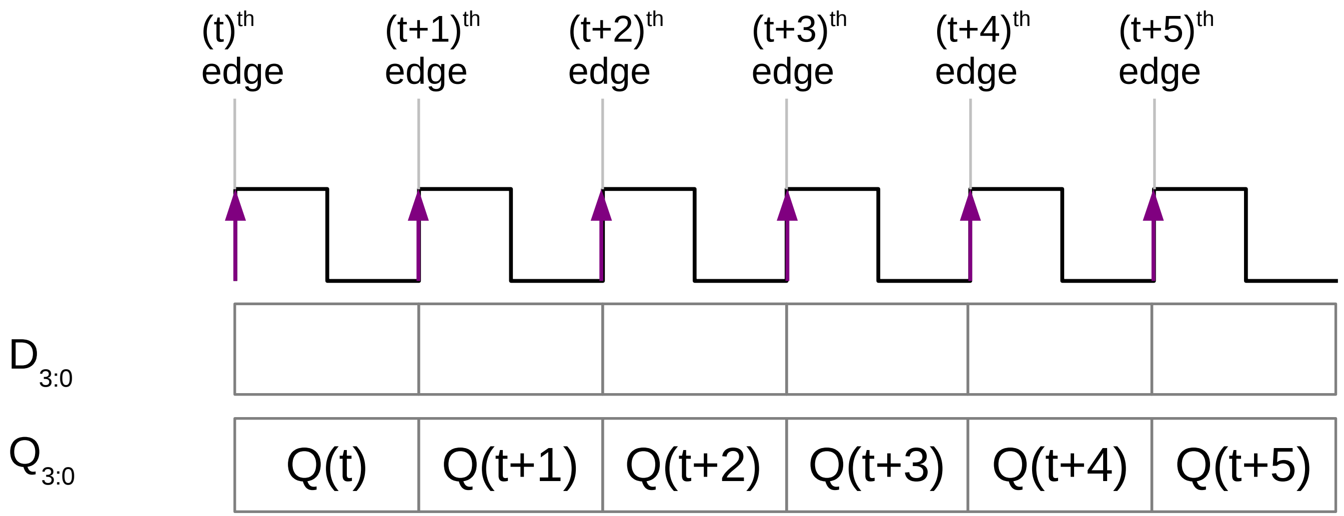
https://en.wikiversity.org/wiki/The_necessities_in_Computer_Design

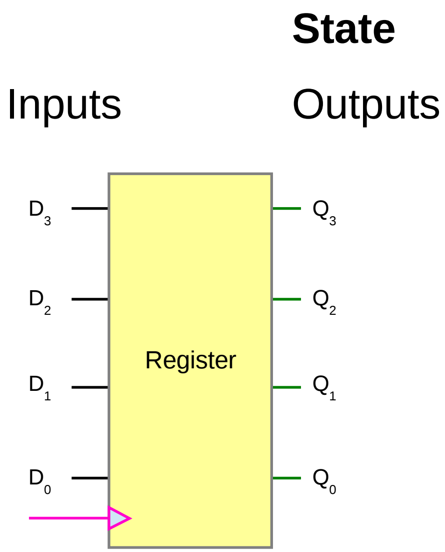
FF Timing (Ideal)



https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

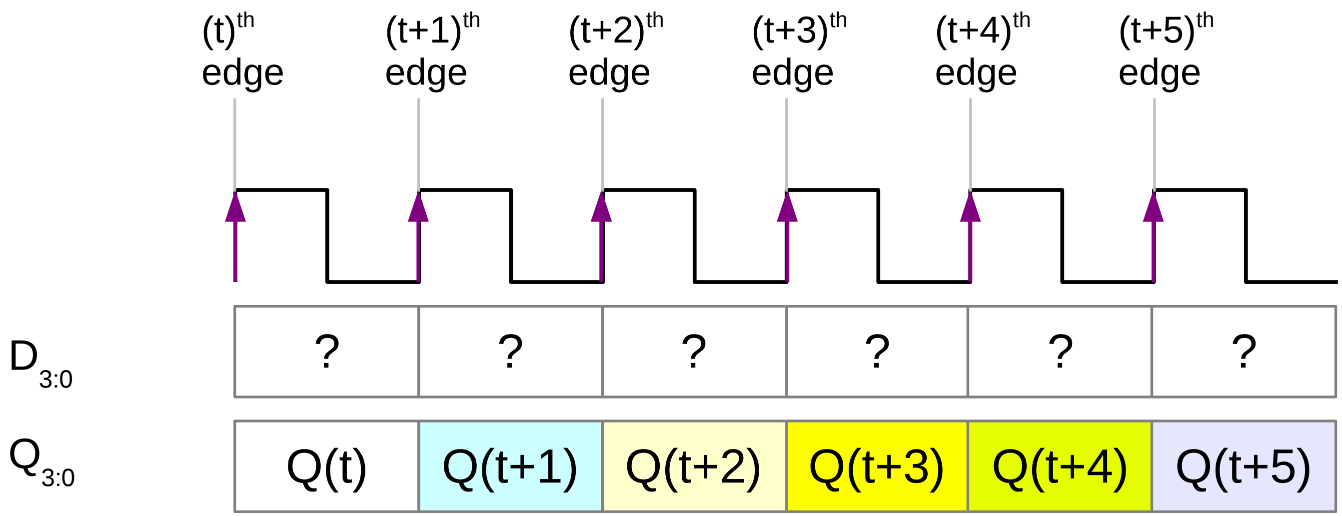
States

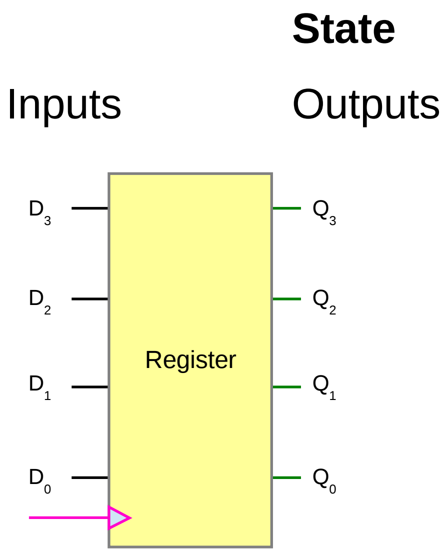




https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

Sequence of States



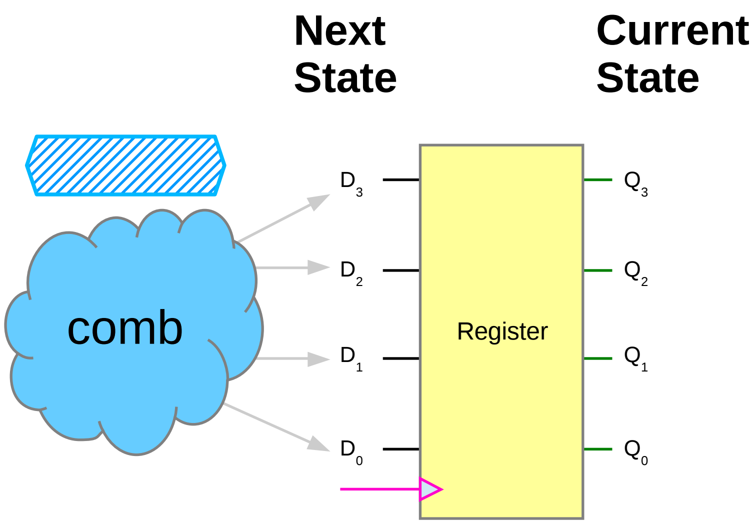


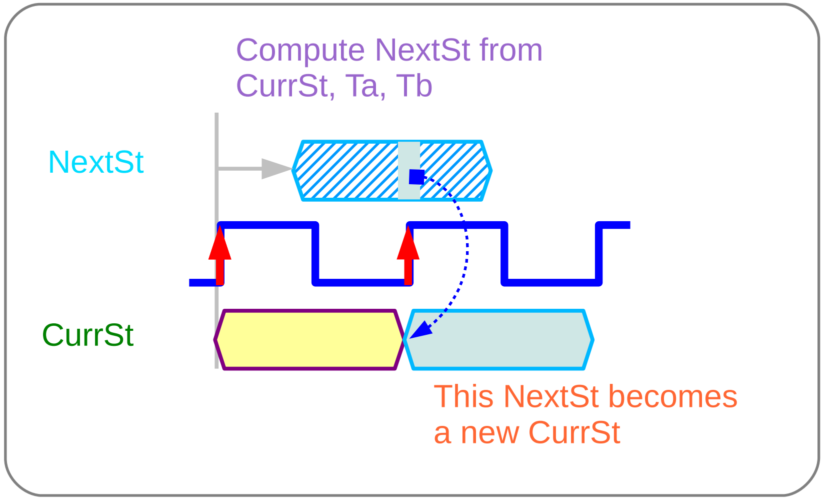
Find inputs to FFs

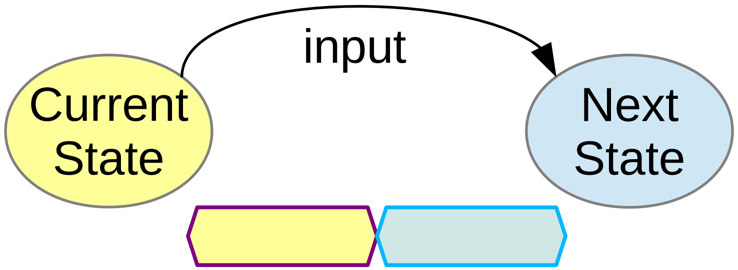
which will make outputs in this sequence

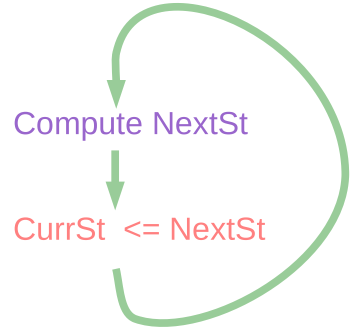
https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

How to change current state



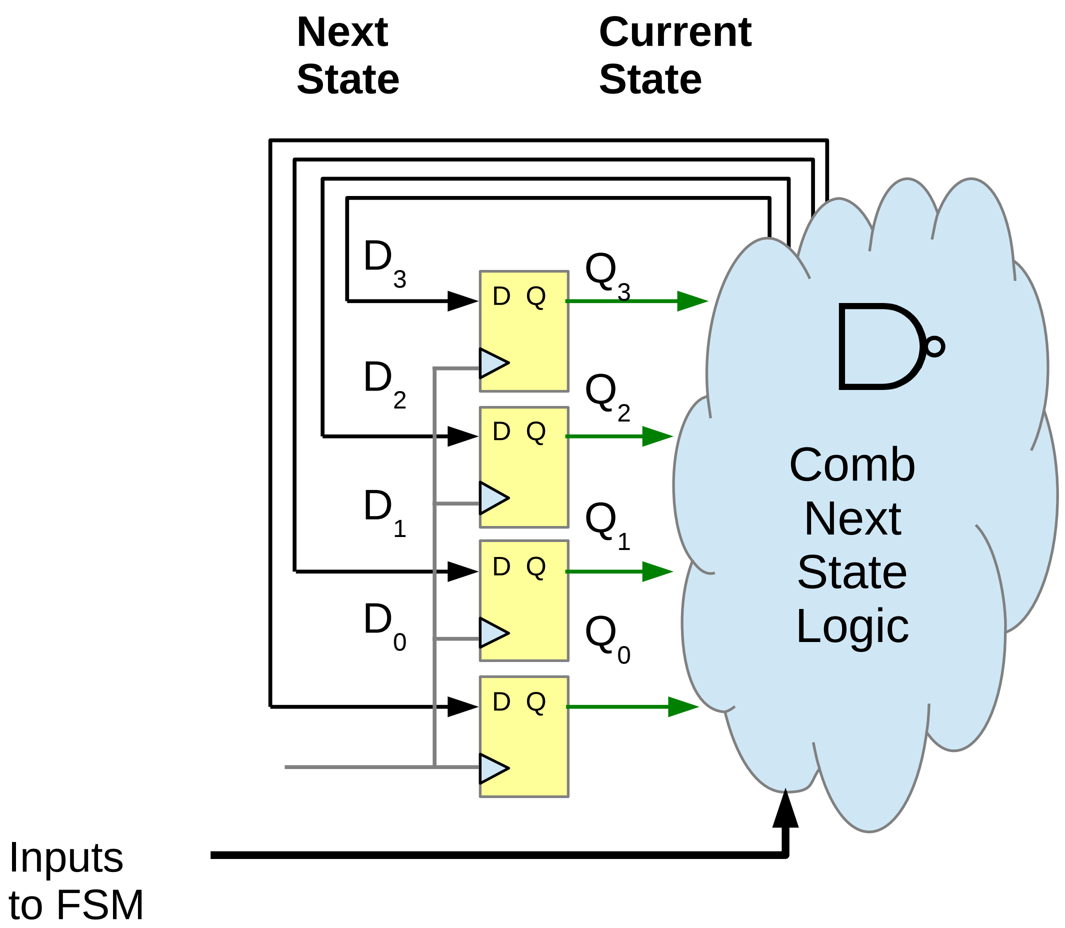






https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

Finding FF Inputs



During the t^{th} clock edge period,

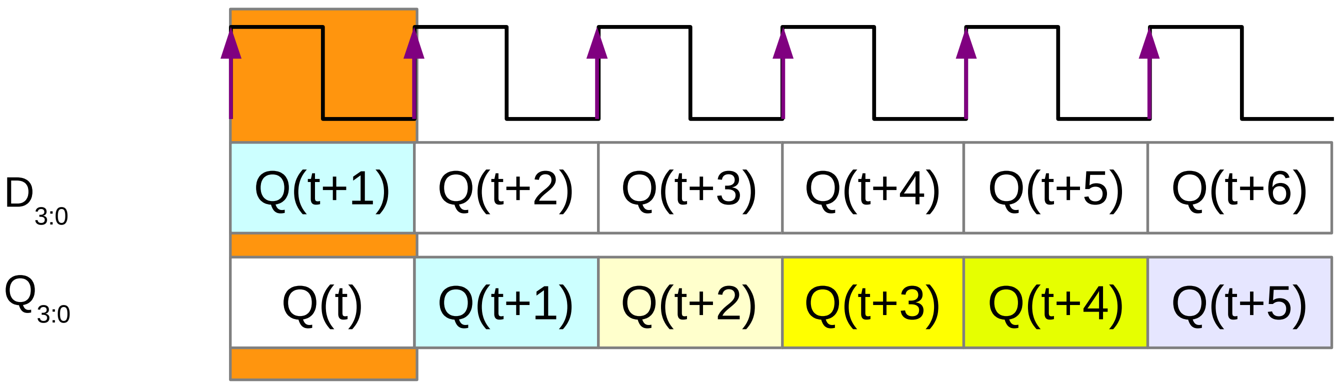
Compute the next state Q(t+1) using the current state Q(t) and other external inputs

Place it to FF inputs

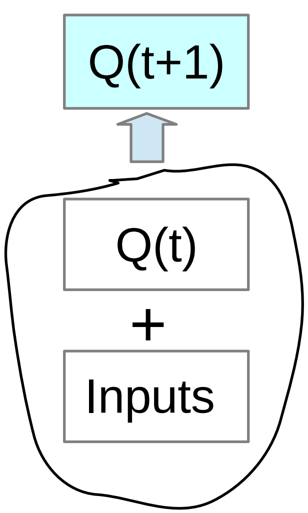
After the next clock edge, $(t+1)^{th}$, the computed next state Q(t+1) becomes the current state

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

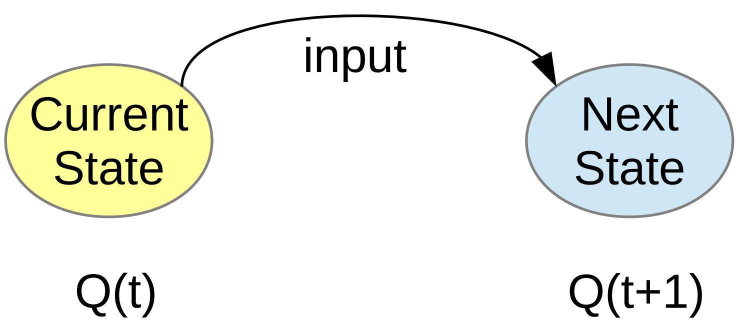
Method of Finding FF Inputs



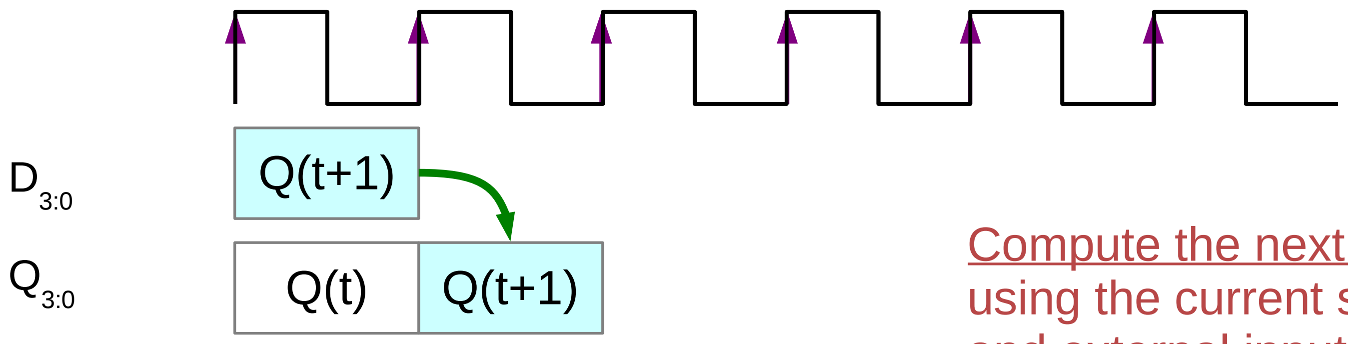
Find the boolean functions
D3, D2, D1, D0
in terms of Q3, Q2, Q1, Q0,
and external inputs
for all possible cases.



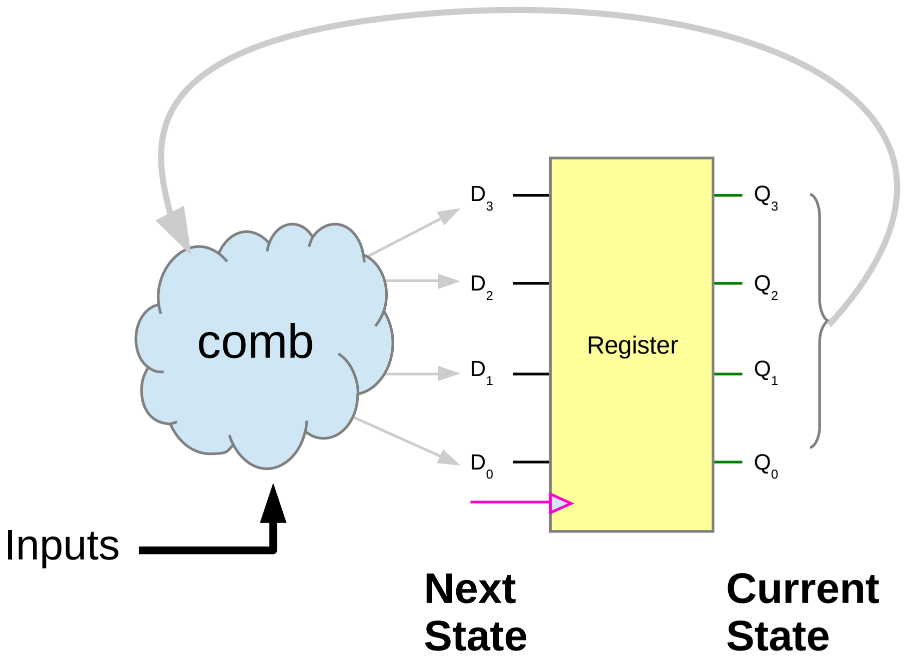
Inputs

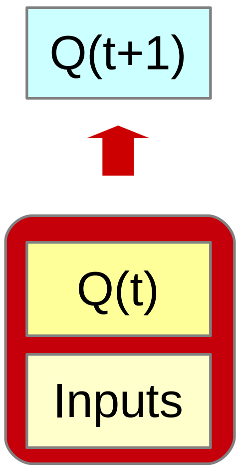


State Transition



Compute the next state
using the current state
and external inputs
in the current clock cycle

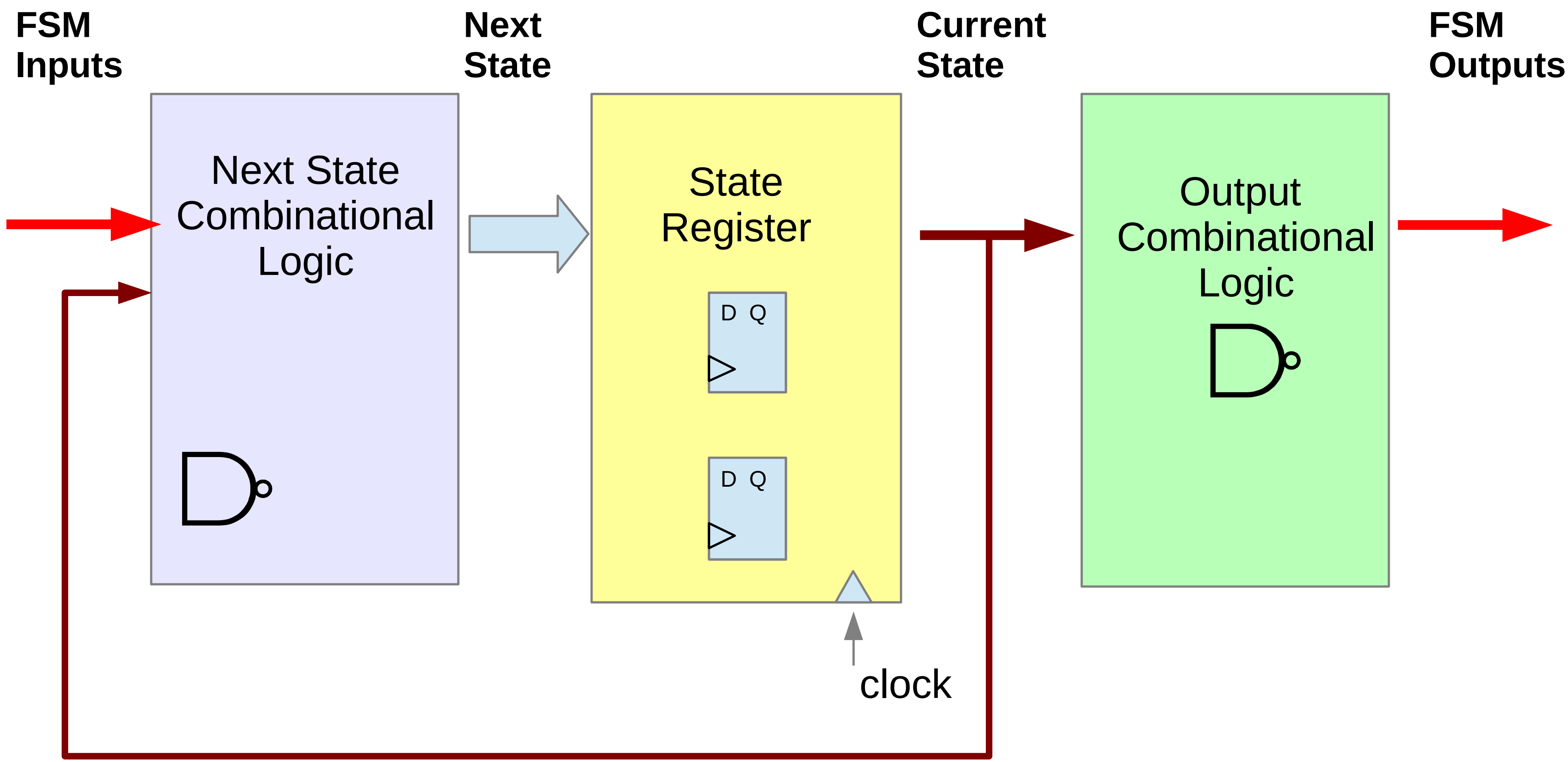




https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

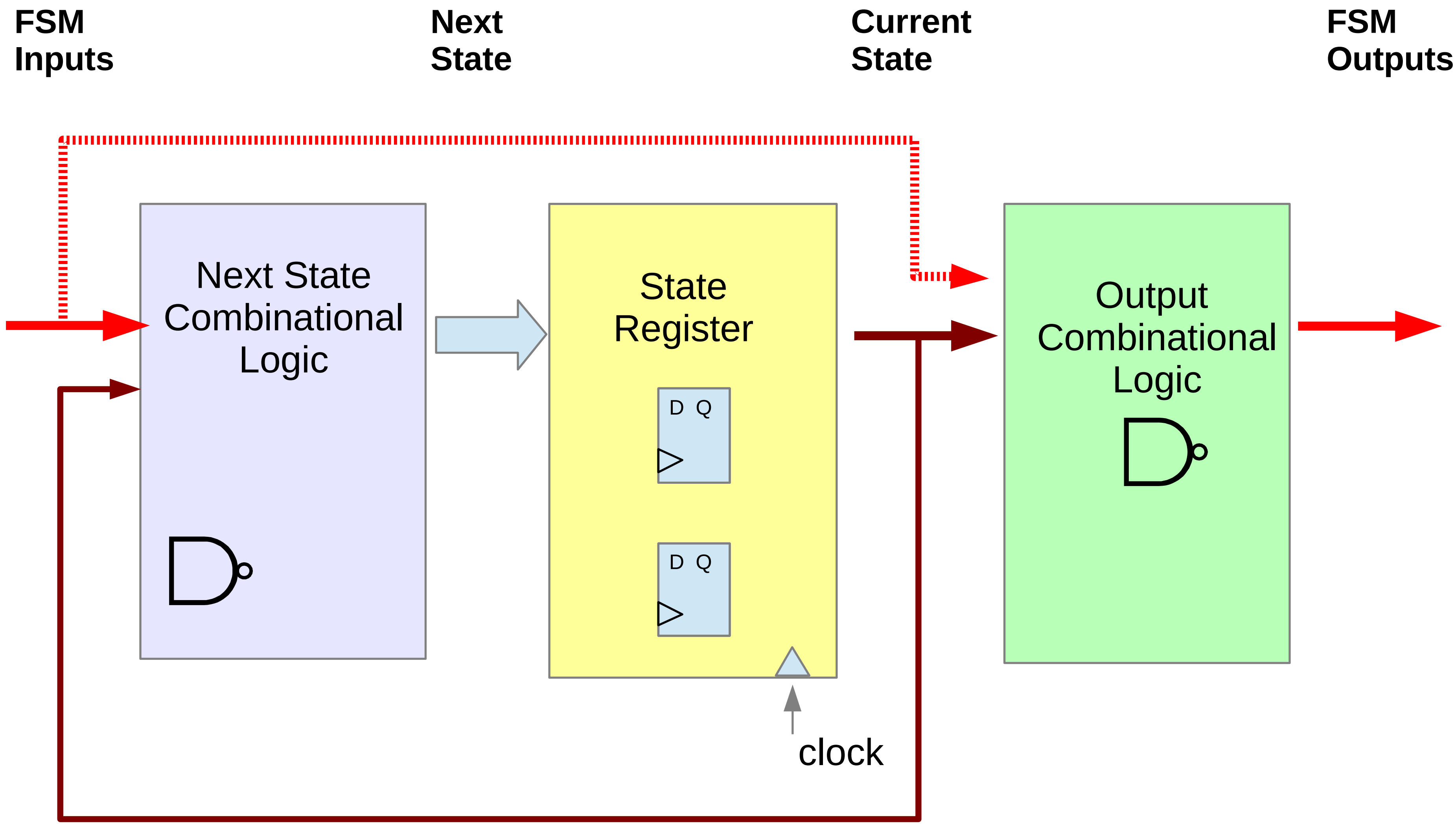
After the next clock edge,
the computed next state (FF Inputs)
becomes the current state (FF Outputs)

Moore FSM



https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

Mealy FSM

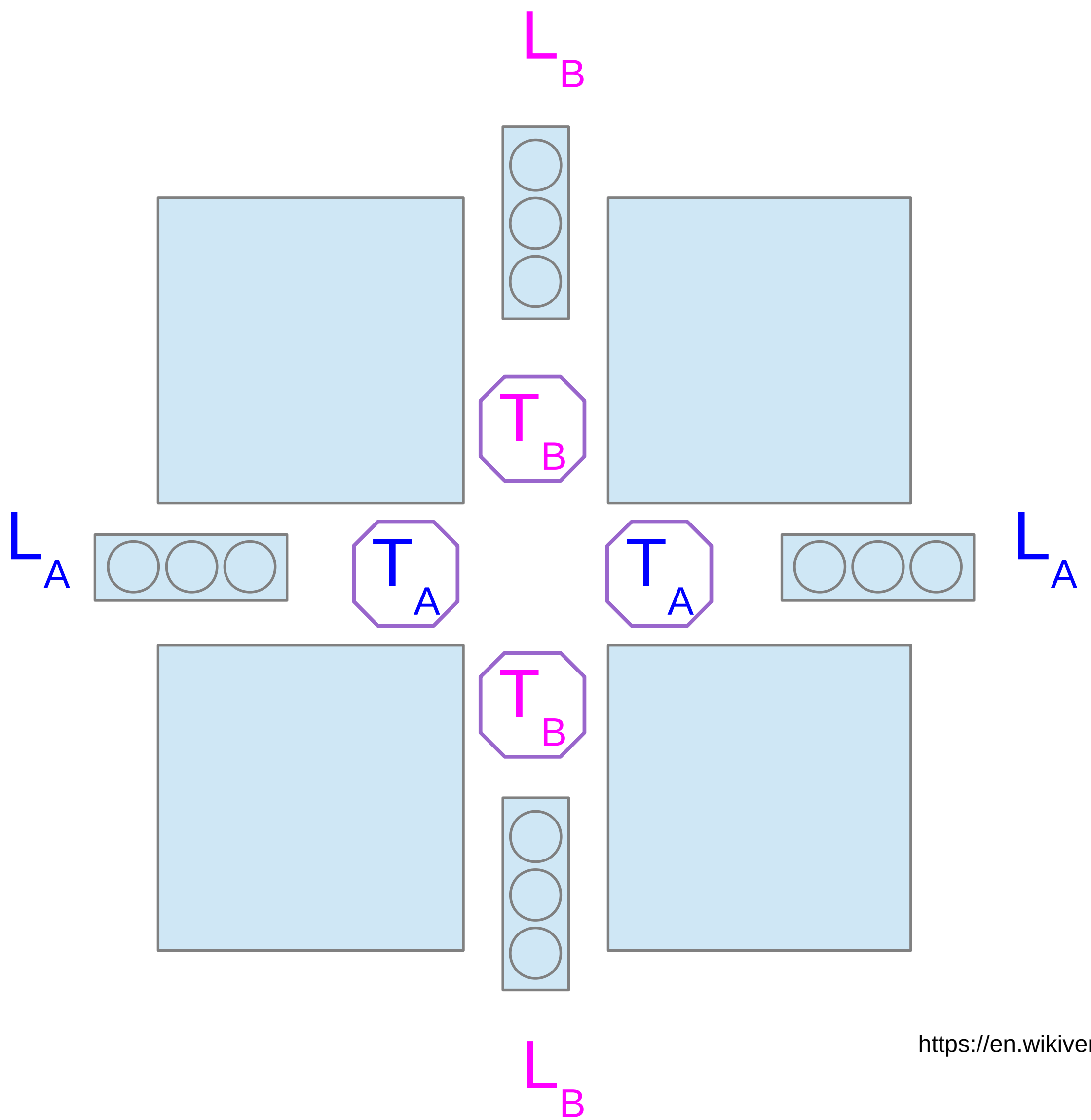


https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

Traffic Lights Example

https://en.wikiversity.org/wiki/The_necessities_in_Computer_Design

FSM Inputs and Outputs



Traffic Lights - Outputs

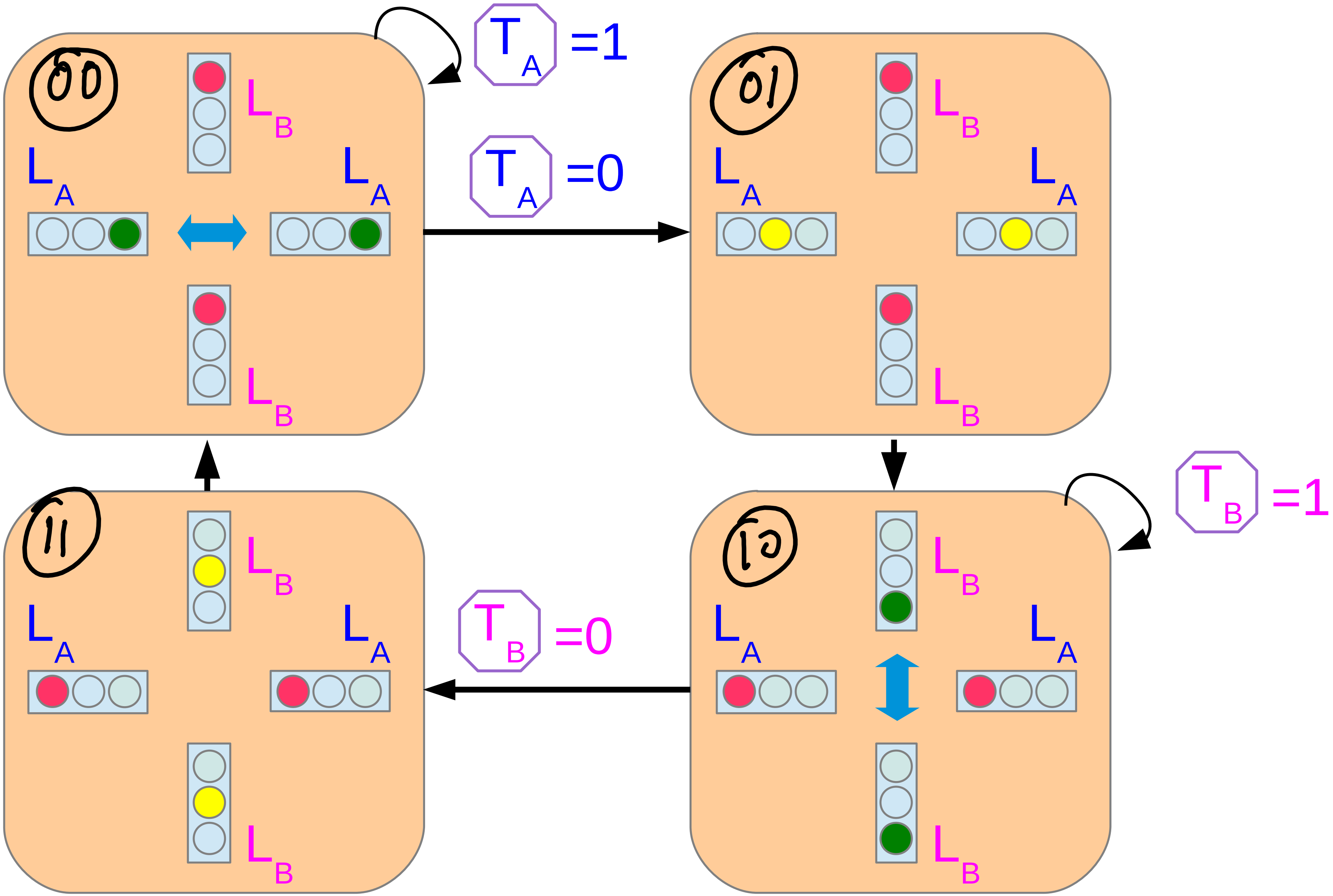
L_A L_B

Sensor - Inputs

T_A T_B

https://en.wikiversity.org/wiki/The_necessities_in_Computer_Design

States



https://en.wikiversity.org/wiki/The_necessities_in_Computer_Design

Moore FSM State Transition Table

S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

	S_1	S_0	T_A	T_B	S'_1
	0	0	0	X	0
	0	0	1	X	0
$\overline{S_1} S_0$ ⇨	0	1	X	X	1
$S_1 \overline{S_0} \overline{T_B}$ ⇨	1	0	X	0	1
$S_1 \overline{S_0} T_B$ ⇨	1	0	X	1	1
	1	1	X	X	0

$$S'_1 = \overline{S_1} S_0 + S_1 \overline{S_0}$$
$$= S_1 \oplus S_0$$

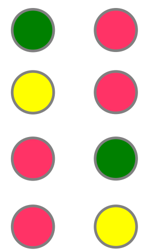
	S_1	S_0	T_A	T_B	S'_0
$\overline{S_1} \overline{S_0} \overline{T_A}$ ⇨	0	0	0	X	1
	0	0	1	X	0
	0	1	X	X	0
$S_1 \overline{S_0} \overline{T_B}$ ⇨	1	0	X	0	1
	1	0	X	1	0
	1	1	X	X	0

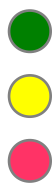
$$S'_0 = \overline{S_1} \overline{S_0} \overline{T_A} + S_1 \overline{S_0} \overline{T_B}$$

https://en.wikiversity.org/wiki/The_necessities_in_Computer_Design

States

S_1	S_2	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1





00
01
10

S_1	S_2	L_{A1}
0	0	0
0	1	0
1	0	1
1	1	1

$L_{A1}=S_1$

S_1	S_2	L_{A0}
0	0	0
0	1	1
1	0	0
1	1	0

$L_{A0}=\overline{S}_1 S_0$

S_1	S_2	L_{B1}
0	0	1
0	1	1
1	0	0
1	1	0

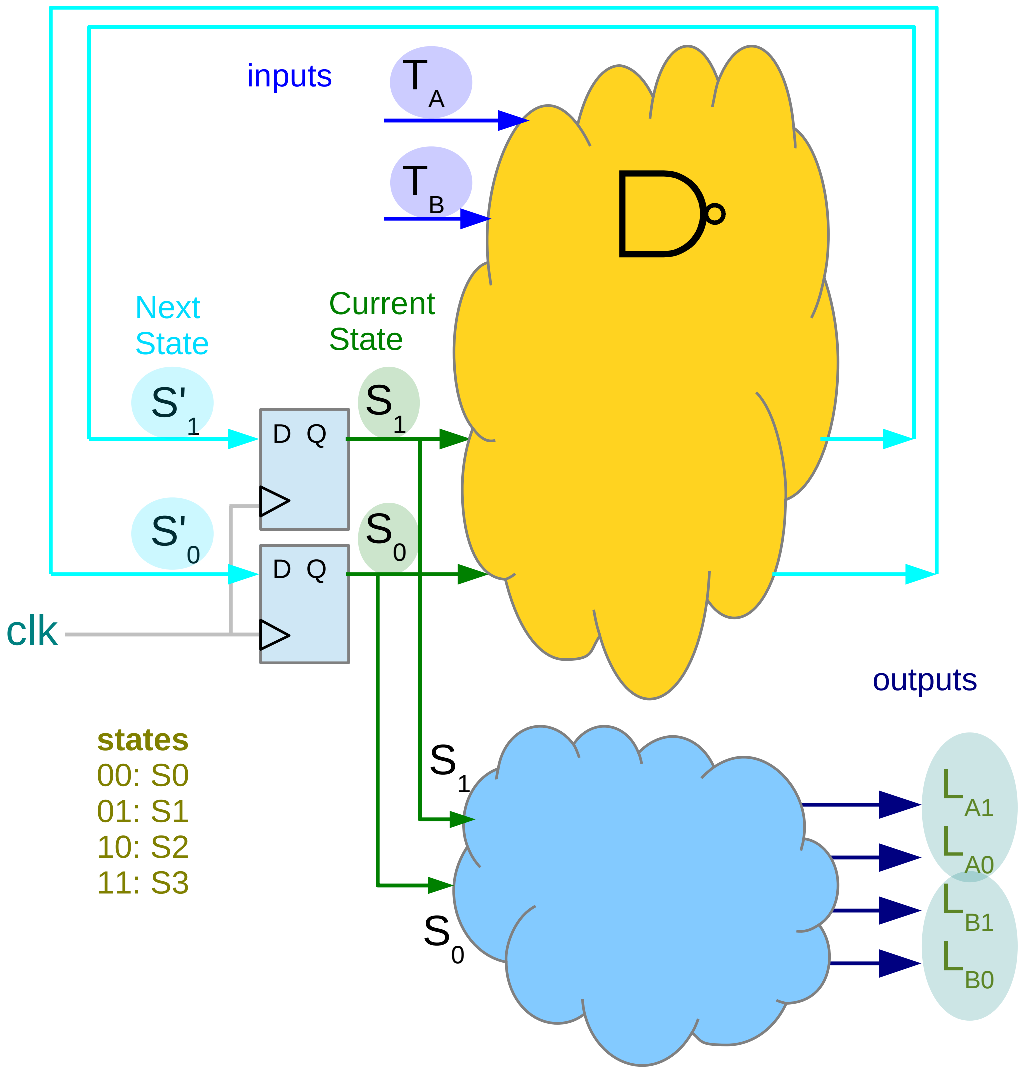
$L_{B1}=\overline{S}_1$

S_1	S_2	L_{B0}
0	0	0
0	1	0
1	0	0
1	1	1

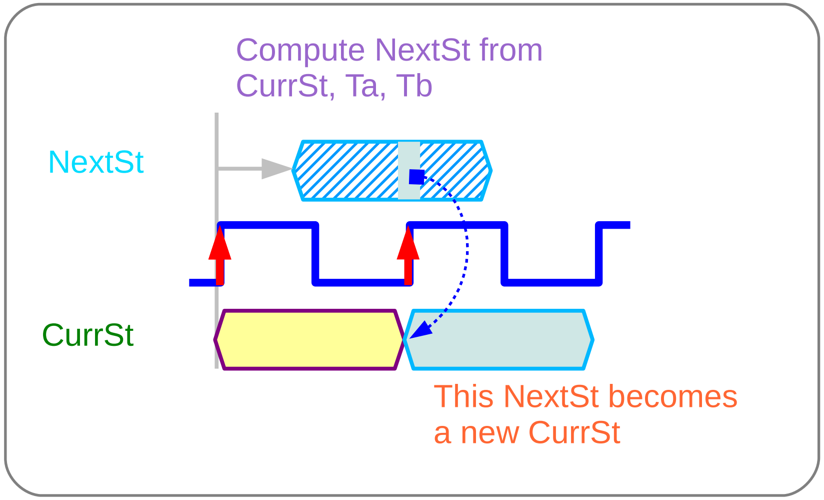
$L_{A0}=S_1 S_0$

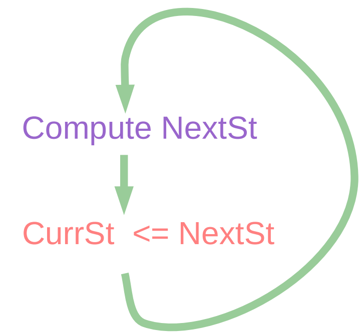
https://en.wikiversity.org/wiki/The_necessities_in_Computer_Design

Moore FSM (1)



states
00: S0
01: S1
10: S2
11: S3

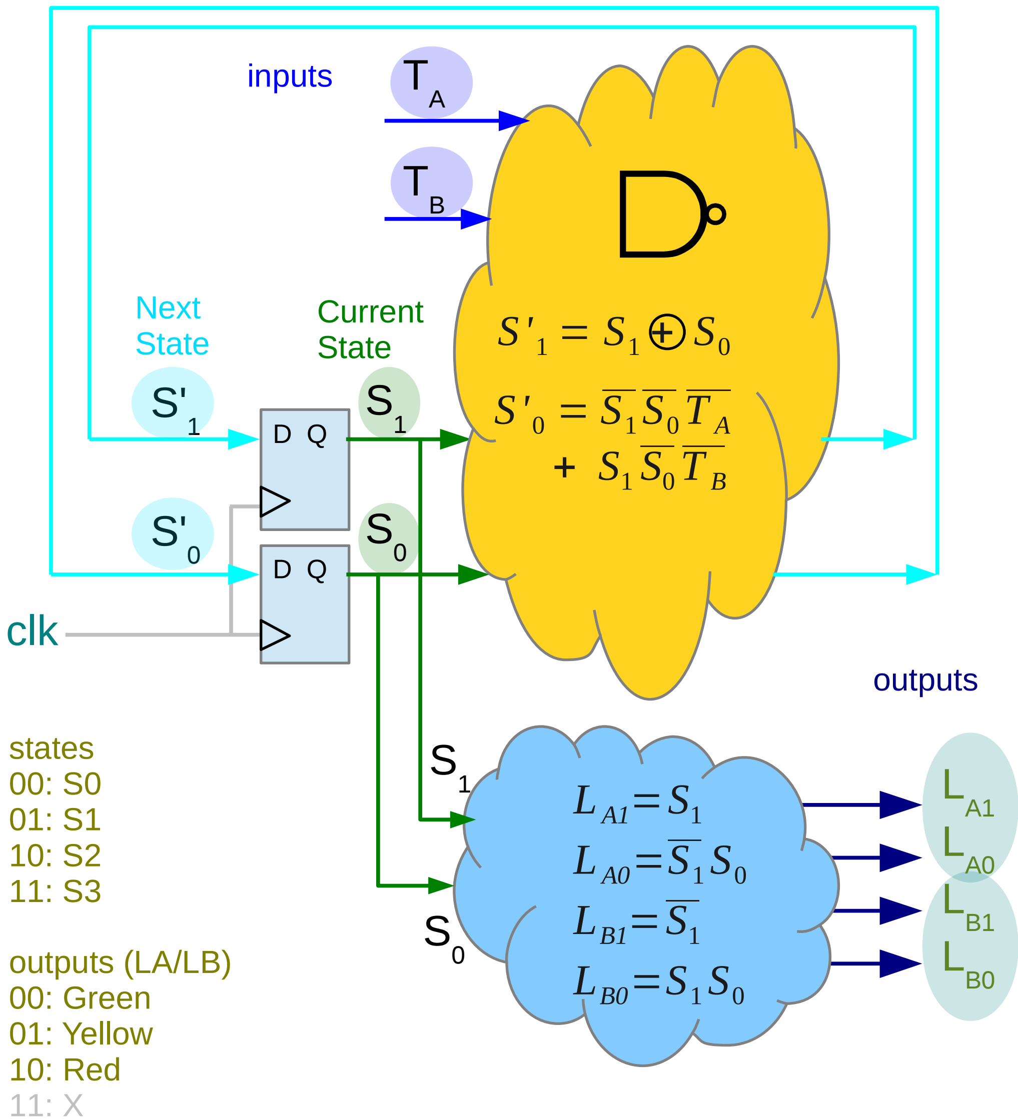




outputs (LA/LB)
00: Green
01: Yellow
10: Red
11: X

https://en.wikiversity.org/wiki/The_necessities_in_Computer_Design

Moore FSM



Inputs T_A T_B

Current State S_1 S_0

Next States

$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\overline{S_0}\,\overline{T_B}$$

Current State S_1 S_0

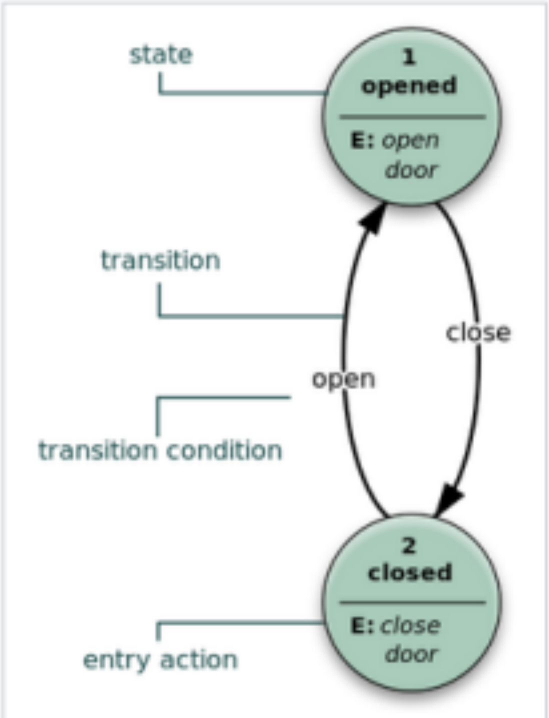
Outputs

$L_{A1} = S_1$ $\quad L_{B1} = \overline{S_1}$

$L_{A0} = \overline{S_1} S_0$ $\quad L_{B0} = S_1 S_0$

https://en.wikiversity.org/wiki/The_necessities_in_Computer_Design

State Diagram



A state diagram for a door that can only be opened and closed

https://en.wikipedia.org/wiki/Finite-state_machine

Acceptors and Recognizers

Fig. 4 Acceptor FSM: parsing the string "nice"

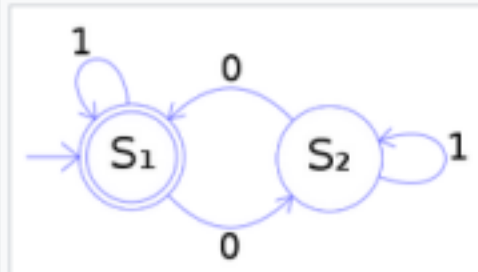


Fig. 5: Representation of a finite-state machine; this example shows one that determines whether a binary number has an even number of 0s, where S_1 is an **accepting state**.

https://en.wikipedia.org/wiki/Finite-state_machine

Classifiers and Transducers

A **classifier** is a generalization of a finite state machine that, similar to an acceptor, produces a single output on termination but has more than two **terminal states**

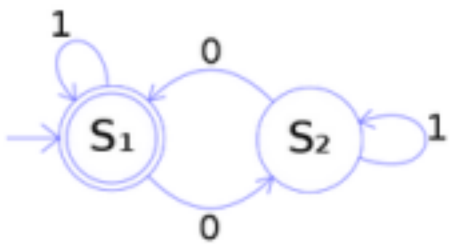
Transducers generate output based on a given **input** and/or a **state** using actions. They are used for control applications and in the field of computational linguistics.

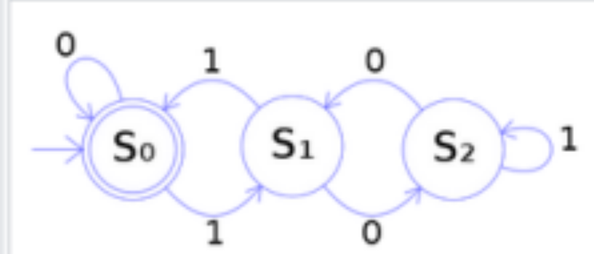
https://en.wikipedia.org/wiki/Finite-state_machine

Moore machine

Example: DFA, NFA, GNFA, or Moore machine [edit]

S_1 and S_2 are states and S_1 is an **accepting state** or a **final state**. Each edge is labeled with the input. This example shows an acceptor for strings over {0,1} that contain an even number of zeros.





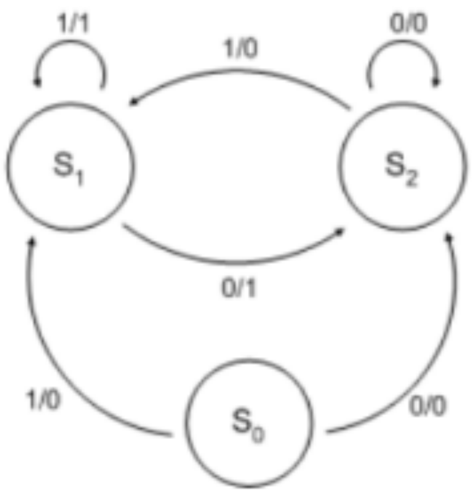
An example of a deterministic finite automaton that accepts only binary numbers that are multiples of 3. The state S_0 is both the start state and an accept state.

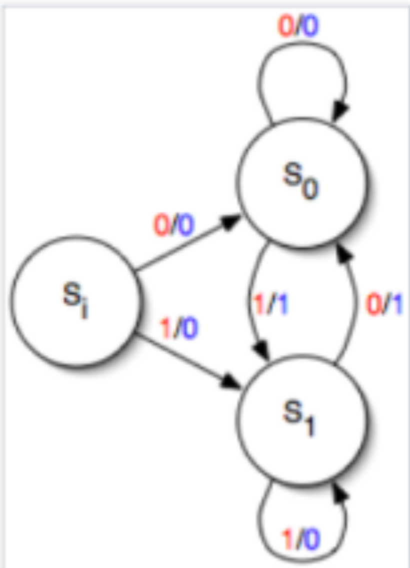
https://en.wikipedia.org/wiki/State_diagram
https://en.wikipedia.org/wiki/Finite-state_transducer

Mealy machine

Example: Mealy machine [edit]

S_0, S_1, and S_2 are states. Each edge is labeled with "*j* / *k*" where *j* is the input and *k* is the output.





State diagram for a simple Mealy machine with one input and one output.

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

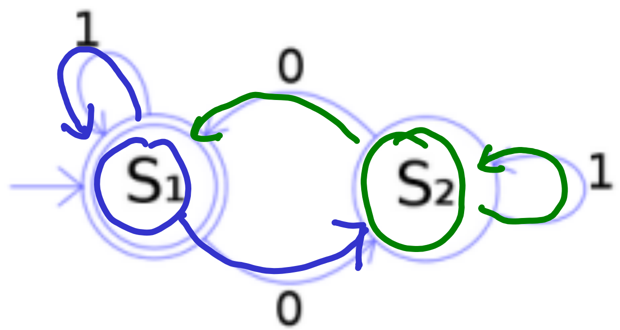
https://en.wikipedia.org/wiki/Mealy_machine

State Transition Table

State Transition Table

Input / State	1	0
S_1	S_1	S_2
S_2	S_2	S_1

State Diagram



states $\{S_1, S_2\}$

inputs $\{0, 1\}$

init state $S1$

final state S_1

$\{S1, S_2\} \times \{1, 0\} \longrightarrow \{S1, S_2\}$

$(S1, 1) \longrightarrow S1$

$(S1, 0) \longrightarrow S_2$

$(S2, 1) \longrightarrow S_2$

$(S_2, 0) \longrightarrow S1$

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

Mathematical Models for acceptors

A **deterministic finite state machine** or **acceptor** deterministic finite state machine is a quintuple (Σ, S, s_0, δ, F), where:

$\{1, 0\}$

- Σ is the input alphabet (a finite, non-empty set of symbols).
- S is a finite, non-empty set of states. $\{S1, S2\}$
- s_0 is an initial state, an element of S. $S1$
- δ is the state-transition function: $\delta : S \times \Sigma \rightarrow S$
- F is the set of final states, a (possibly empty) subset of S.

Σ input alphabet
S state set
→ S_0 init state
δ transition func
→ F final state

Σ

S, Q

$\{S1, S2\} \times \{1, 0\} \rightarrow \{S1, S2\}$

$S \times \Sigma \rightarrow S$

$\delta : S \times \Sigma$

$\{S1, S2\} \times \{0, 1\} \rightarrow \{S1, S2\}$

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

Deterministic Finite Automaton Example (1)

The following example is of a DFA M, with a binary alphabet, which requires that the input contains an even number of 0s.

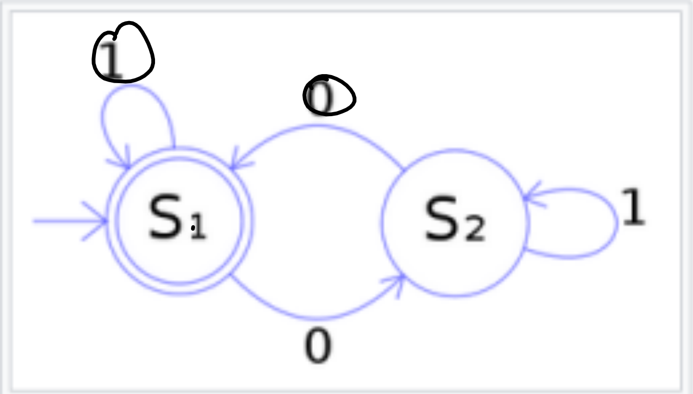
0, 1

$S \; \Sigma \; \delta \; S_0 \; F$

M = (Q, Σ, δ, q0, F) where

- Q = {S1, S2},
- Σ = {0, 1},
- q0 = S1,
- F = {S1}, and
- δ is defined by the following state transition table:

Σ input alphabet
S state set Q
→ S_0 init state q_0
δ transition func
→ F final state



The state diagram for *M*

	0	1
S_1	S_2	S_1
S_2	S_1	S_2

$\{S_1, S_2\} \times \{0, 1\}$

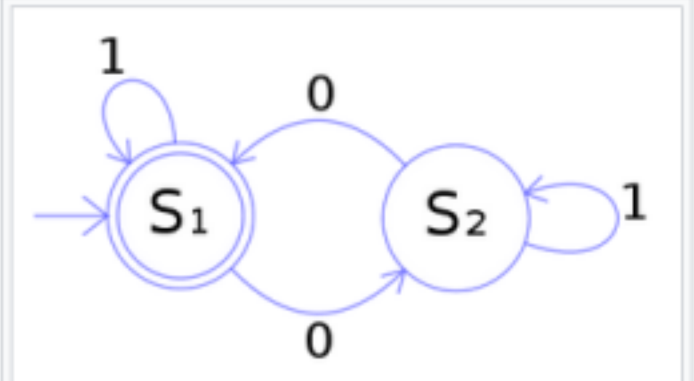
https://en.wikipedia.org/wiki/Deterministic_finite_automaton

Deterministic Finite Automaton Example (2)

The **state S1** represents that there has been an even number of 0s in the input so far, while **S2** signifies an odd number.

A **1** in the input does not change the state of the automaton.
When the input ends, the state will show whether the input contained an even number of **0**s or not.
If the input did contain an even number of **0**s, M will finish in **state S1**, an accepting state, so the input string will be accepted.

The language recognized by M is the regular language given by the regular expression **((1*) 0 (1*) 0 (1*))***, where "*" is the Kleene star, e.g., **1*** denotes any number (possibly zero) of consecutive **ones**.



The state diagram for *M*

https://en.wikipedia.org/wiki/Deterministic_finite_automaton vi

%s/7[0-9]*/777/g

regular expression

zero or more

7

7 □

7 □ □

7 □ □ □

7 □ □ □ □

⋮

*

Kleene

automata ⇒

* 0 +

*

The language recognized by M is the regular language given by the regular expression **((1*) 0 (1*) 0 (1*))***, where "*" is the Kleene star, e.g., **1*** denotes any number (possibly zero) of consecutive **ones**.

$\boxed{\text{Kleene}}$ *

1* = 1

11

111

1111

((1*) 0 (1*) 0 (1*))*

0 0

1 0 1 0 1

1 0 0 1

111 0 1111 0 1

Mathematical Model for transducers (1)

A **finite-state transducer** is a sextuple (Σ, Γ, S, s0, δ, ω), where:
Σ is the input alphabet (a finite non-empty set of symbols).
Γ is the output alphabet (a finite, non-empty set of symbols).
S is a finite, non-empty set of states.
s0 is the initial state, an element of S.
ω is the output function.

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

Mathematical Model for transducers (2)

If the **output** function is a function of a **state** and **input** alphabet ($\omega : S \times \Sigma \rightarrow \Gamma$) that definition corresponds to the **Mealy model**, and can be modelled as a Mealy **machine**.

If the **output** function depends only on a **state** ($\omega : S \rightarrow \Gamma$) that definition corresponds to the **Moore model**, and can be modelled as a **Moore machine**.

A finite-state machine with no output function at all is known as a **semiautomaton** or **transition** system.

$\Gamma = \{ \text{outputs} \}$

$\Sigma = \{ \text{inputs} \}$

$S = \{ \text{states} \}$

$\Sigma = \{ 0, 1 \}$

$S = \{ S_1, S_2 \}$

Moore $\omega : S \rightarrow \Gamma$

Merly $\omega : S \times \Sigma \rightarrow \Gamma$

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

References

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