École Nationale Supérieure d'Informatique
pour l'Industrie et l'Entreprise

ENSIIE - Évry

Report of internship M2QF

Graph Theory

CHHEANG Vinha

Advisor: PULIDO NINO Sergio - ENSIIE

Supervisor: Lin Mongkolsery - ITC

Supervisor: PHAUK Sokkhey - ITC

Internship

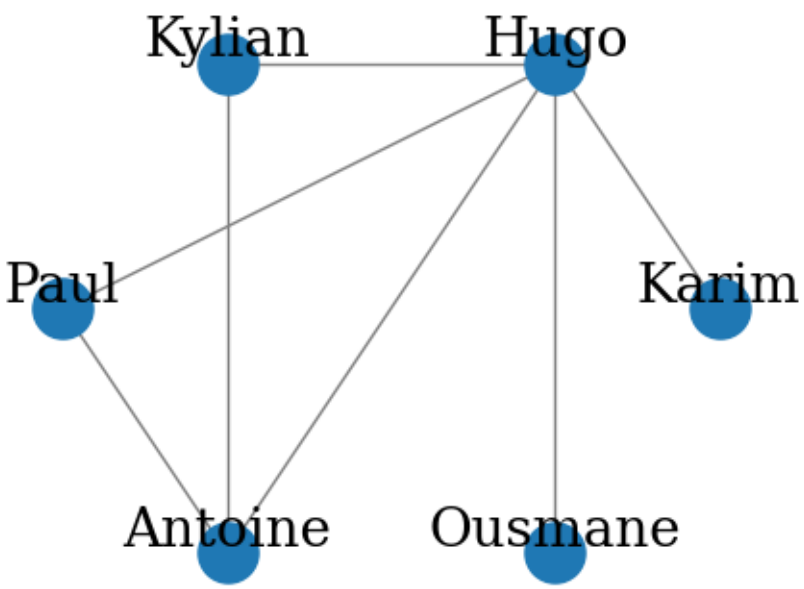
01 May to 30 October 2023

Contents

1 Introduction to Graph Theory

1.1 Undirect Graph

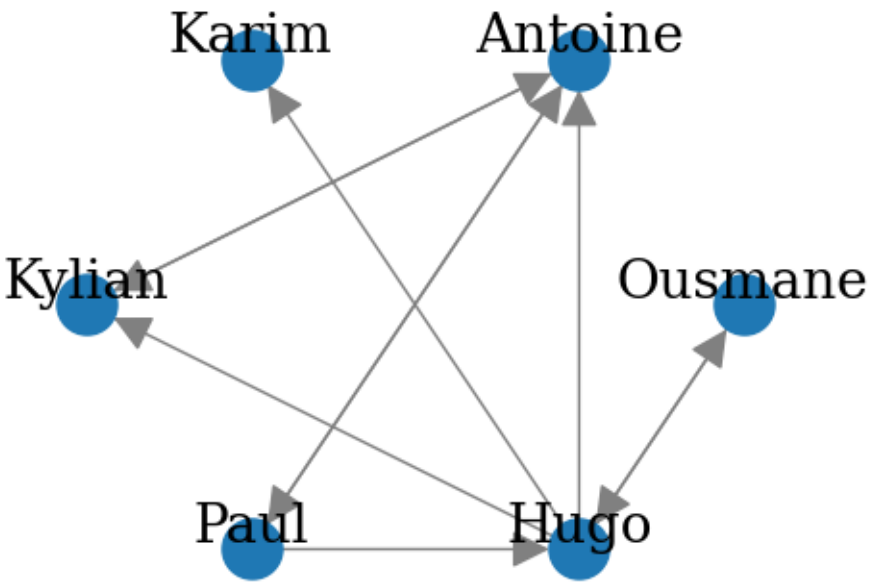
- Graph $G = (V, E)$ is undirect graph.



- V= {vertices } = { Kylian, Antoine, Paul, Karim, Hugo, Ousmane }
- E= { edges } = { [Kylian, Antoine], [Antoine, Paul], [Hugo, Kylian], [Hugo, Antoine], [Paul, Hugo], [Hugo, Karim], [Hugo, Ousmane] }
- $\Gamma(v)$ = { neighbors of v in graph G }
    * Γ (Antoine)= { Kylian, Paul, Hugo }
    * Γ (Kylian)= { Antoine, Hugo }
- Degree $d(v)$ = number of edges which across the vertex v of Graph G.
    * d(Antoine)= 3
    * d(Hugo) = 5
- Path of Graph G.
    * Path = [Kylian, Antoine ,Paul, Hugo, Karim]
- Cycle of graph G.
    * Cycle = [Kylian, Antoine ,Paul, Hugo, Kylian]

1.2 Direct Graph

- Graph $G = (V, E)$ is direct graph.



- V= {vertices } = { Kylian, Antoine, Paul, Karim, Hugo, Ousmane }
- E= { edges } = { (Kylian, Antoine), (Antoine, Kylian), (Antoine, Paul), (Paul,Antoine), (Hugo, Kylian), (Hugo, Antoine), (Paul, Hugo), (Hugo, Karim), (Hugo, Ousmane), (Ousmane, Hugo) }
- $\Gamma(v)$ = { neighbors of v in graph G }
- $\Gamma^+(v)$ = { successors of v in Graph G}
- $\Gamma^-(v)$ = { predecessors of v in Graph G}
    * Γ (Hugo)= { Kylian, Antoine, Paul, Karim, Ousmane }
    * Γ^+ (Hugo)= { Kylian, Antoine, Karim, Ousmane }
    * Γ^- (Hugo)= { Paul, Ousmane}
- Degree $d(v)$ = number of edges which across the vertex v of Graph G.
- Degree in $d^-(v)$ = number of edges which arrive the vertex v.
- Degree out $d^+(v)$ = number of edges which exit the vertex v.
    * d(Hugo) = 6
    * d^-(Hugo) = 2
    * d^+(Hugo) = 4

- Path of Graph G.
    * Path = [Ousmane, Hugo ,Kylian, Antoine, Paul]
- Cycle of graph G.
    * Cycle = [Hugo ,Kylian, Antoine, Paul, Hugo]

2 Graph Connectivity

2.1 Connected Graph

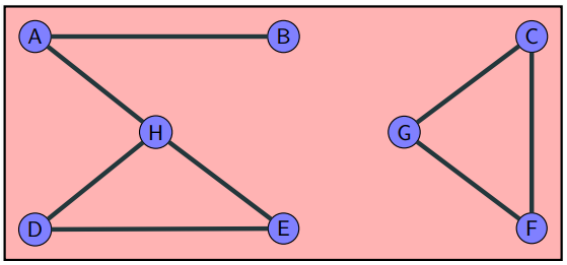


Figure 1: Unconnected Graph

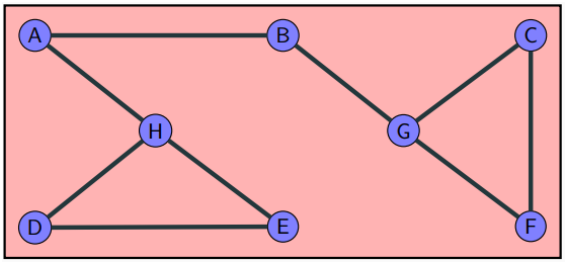


Figure 2: Connected Graph

2.2 Sub-Graph

Let $G' = (V, E)$ is a sub-graph of $G = (V, E)$ if and only if $V' \subseteq V$ and $E' \subseteq E$ and $\forall$ v, u $\in V'$, $[v, u] \in E \Leftrightarrow [v, u] \in E'$.

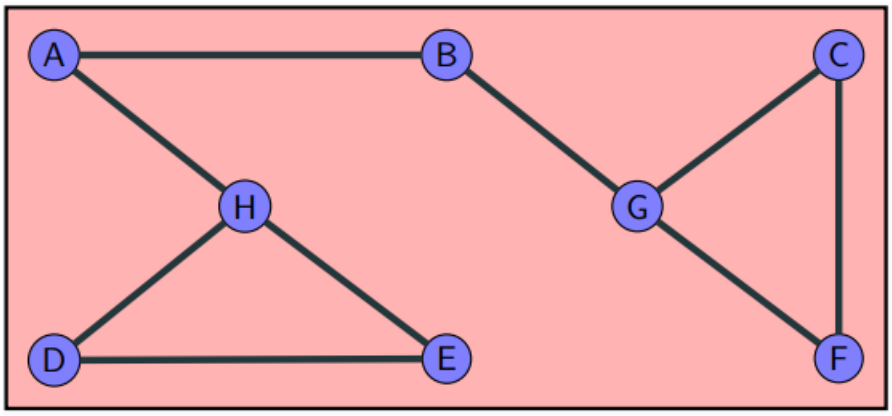


Figure 3: Graph G

Example we have sub graph below:

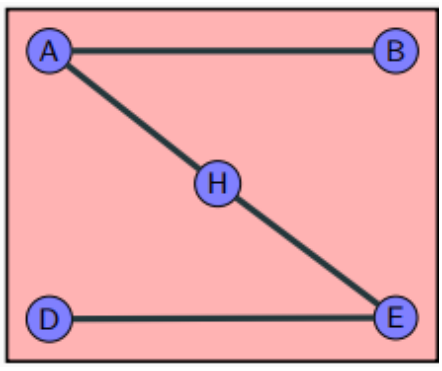


Figure 4: Graph G'_0

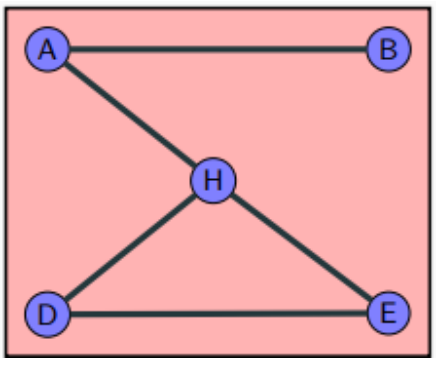


Figure 5: Graph G'_1

2.3 Connected Components

Algorithm 1 Connected Component

* **Input** : Graph $G = (V, E)$
* **Output**: List of components connected

1. $J \longleftarrow 1$:
2. **While** not all nodes have been classified **Do**
   i. Choose an unclassified starting node:
   ii. $S \longleftarrow$ starting node:
   iii. Classify S in component number J
   iv. **While** S is not finished **Do**
      * **While** possible **Do**
        $S \longleftarrow$ an unclassified neighbor of S
        Classify S in J.
      * **End While**
      * Mark S as finished
      * $S \longleftarrow$ node from which S was classified if S is not the starting vertex
   v. **End While**
   vi. $J \longleftarrow J + 1$
3. **End While**

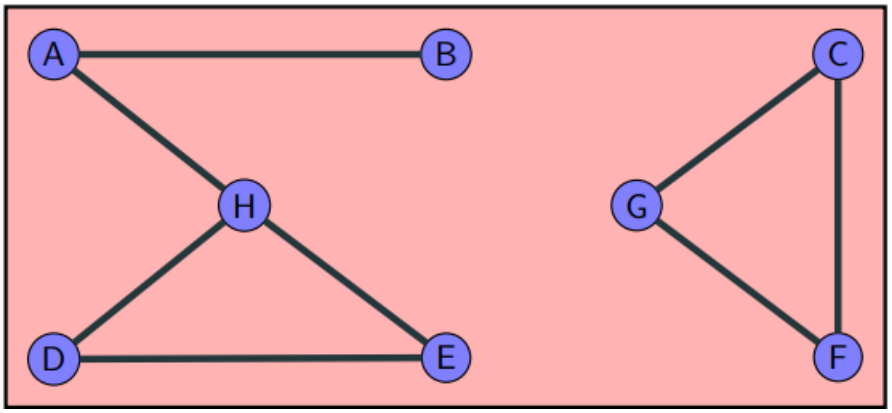


Figure 6: Graph G

We have G as figure above which has 2 components connected.

2.4 Strongly Connected and Strongly Connected Components

2.4.1 Strongly Connected

Note: A directed graph is strongly connected if, for every pair of vertices (u, v), there is a path from u to v and another path from v to u.

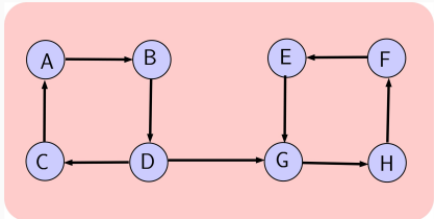


Figure 7: Connected but not strongly connected

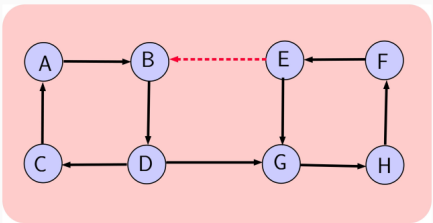


Figure 8: strongly connected

2.4.2 Strongly Connected Components

Algorithm 2 Strongly Connected Components

* **Input** : Graph $G = (V, E)$
* **Output**: List of strongly components connected

1. $X \longleftarrow V$:
2. **While** X is not empty **Do**
   i. $C \longleftarrow \emptyset$: (C is components)
   ii. Mark a vertex x from X with $+$ and $-$:
   iii. **While** possible **Do**
      * Mark with $+$ any successor (not already marked with $+$) of a vertex already marked with $+$;
      * Mark with $-$ any predecessor (not already marked with $-$) of a vertex already marked with $-$;
   iv. **End While**
   v. Write C, the set of vertices marked with $+$ and $-$;
   vi. The sub-graph of G with nodes from C is an strongly connected component of G.
   vii. $X \longleftarrow X - C$
3. **End While**

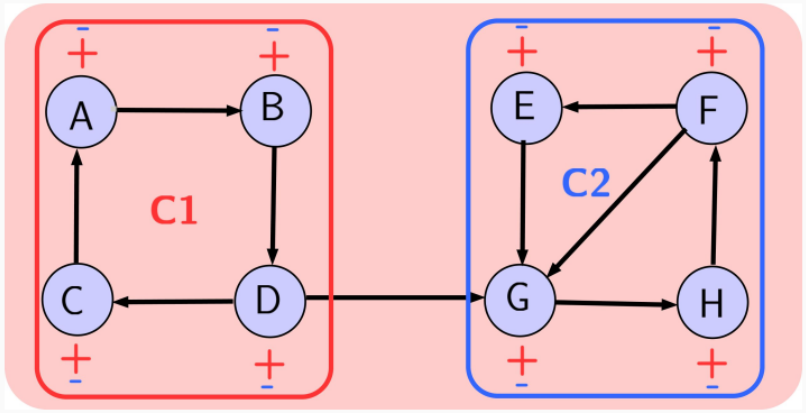
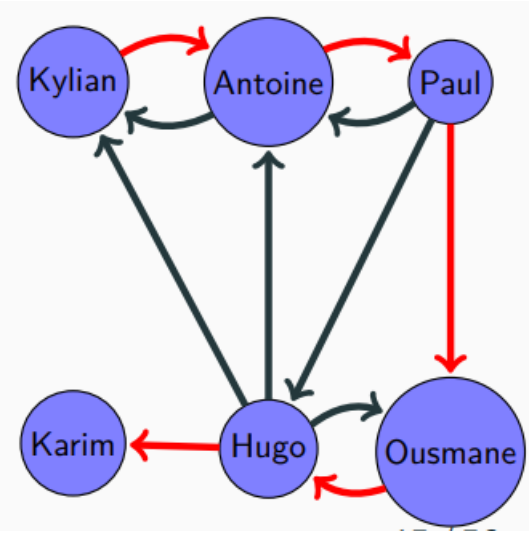


Figure 9: strongly connected component

2.5 Eulerian and Hamiltonian path

- **Hamilton Path** : is a path that contains each vertex of a graph exactly once.



- **Hamilton Theorem**: If G be a a simple undirect graph with n vertices and $d(v) + d(w) \geq n - 1$ whenever v and w are two vertices that not adjacent, then G has a Hamilton path.
- **Euler path**: is a path in a graph that visits every edge exactly once.
- **Euler Theorem**: A connected graph G is Eulerian if and only if number of node (0 or 2 node) with the odd degree.

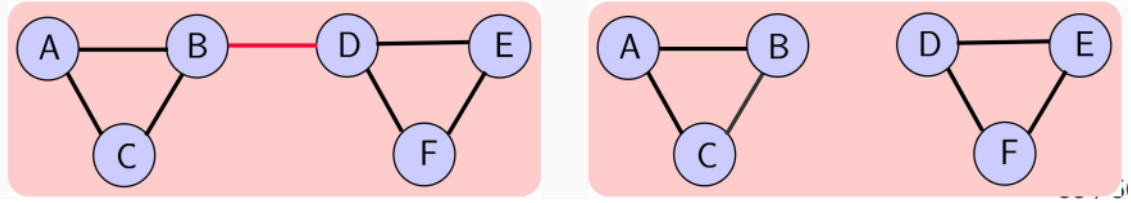


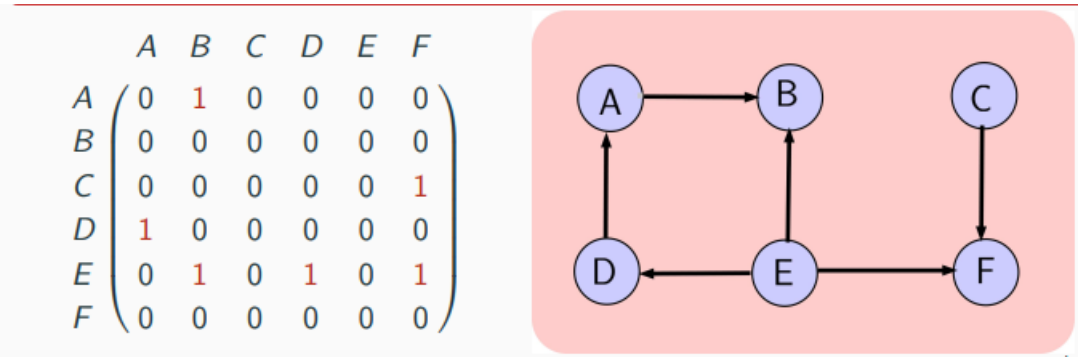
Figure 10: bridge G - Not bridge G

Algorithm 3 Euler walk

* **Input** : Graph $G = (V, E)$ has 2 nodes with odd degree
* **Output**: Euler path G'

1. $G' \longleftarrow G$:
2. Start from a vertex of odd degree.
3. **While** G' is not empty **Do**
   i. Traverse an edge $\{e\}$ that is not a bridge in G' except if we are on a vertex of degree 1 in G'
   ii. $G' \longleftarrow G' \setminus \{e\}$
4. **End While**

2.6 Adjacency matrix



2.7 Transitive Closure

Let $G = (V, E)$ be graph. $\tau(G) = (V, \tau(E))$ was called transitive closure of G if for all $(x, y) \in V$.

$$(x, y) \in \tau(E) \Longleftrightarrow \exists \quad path \quad x \to y \quad in \quad G.$$

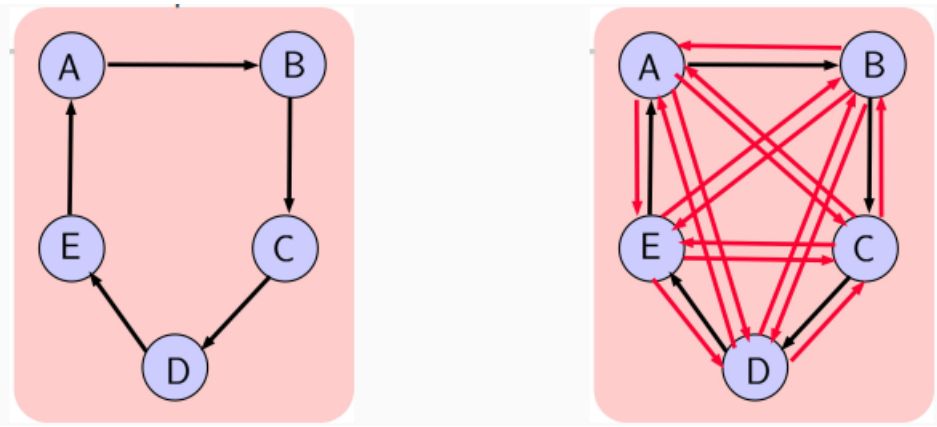


Figure 11: Graph$G$ - Transitive Closure of G

Algorithm 4 Roy Wharshall algorithm

* **Input** : A directed graph $G = (V, E)$
* **Output**: The transitive closure of G

For $w \in V$ **Do**

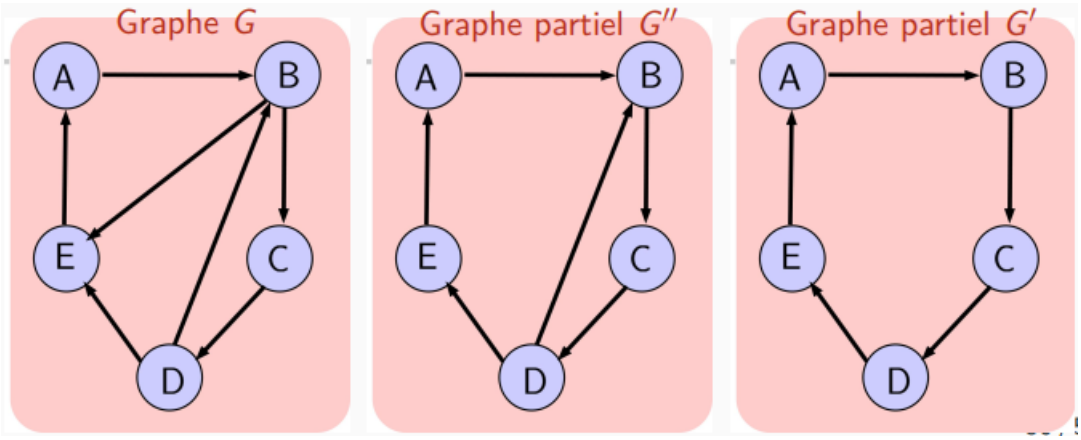
For $v \in V$ **Do**

For $u \in V$ **Do**

If $(u, w) \in E$ and $(v, w) \in E$ **then** Input $(u, v) \in E$

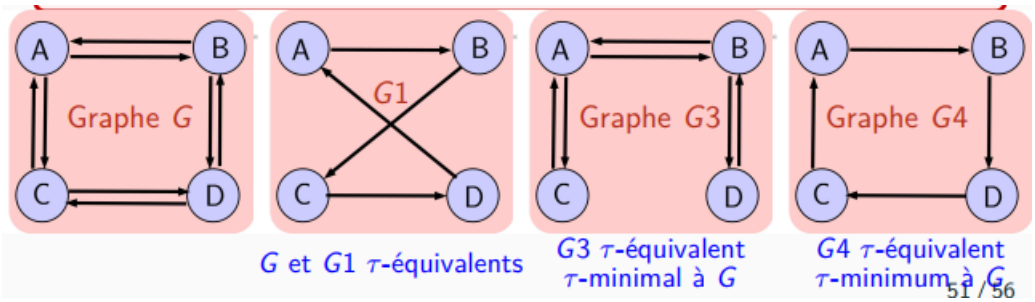
2.8 Partial Graph

Let graph $G = (V, E)$. we have $G' = (V, E')$ is a partial graph of G if E' is include in E.



2.9 τ-equivalent

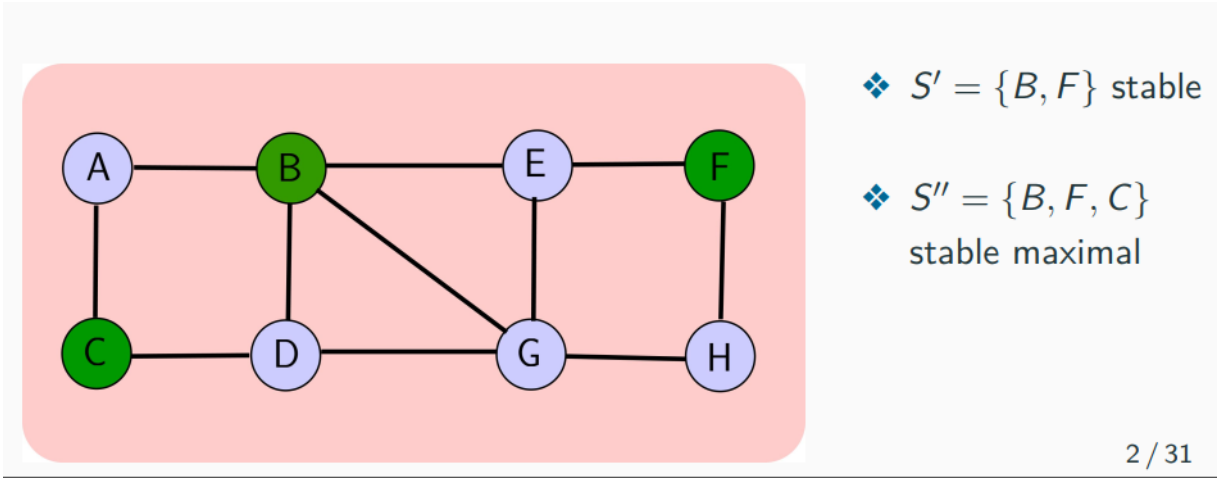
- G and G' was called τ-equivalent if $\tau(G) = \tau(G')$.
- Graph G' was called τ- minimal τ-equivalent with G if G' is a partial graph of G and τ-equivalent to G , if we remove an edge of G' we obtain a graph that is not τ-equivalent to G.
- Graph G' was called τ- minimum τ-equivalent to G if it's τ- minimal τ-equivalent with the number minimum of edges.



3 Graph and Sub-Graph properties

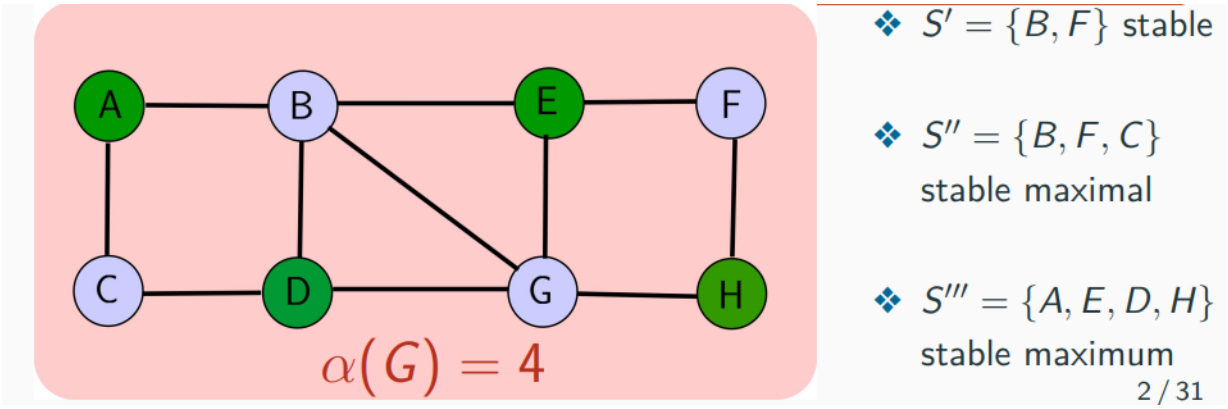
3.1 Stable: stable sets (independent sets)

Let $G = (V, E)$ be a directed or undirected graph. A stable set of G is a subset S of vertices in V such that no two vertices in S are adjacent.



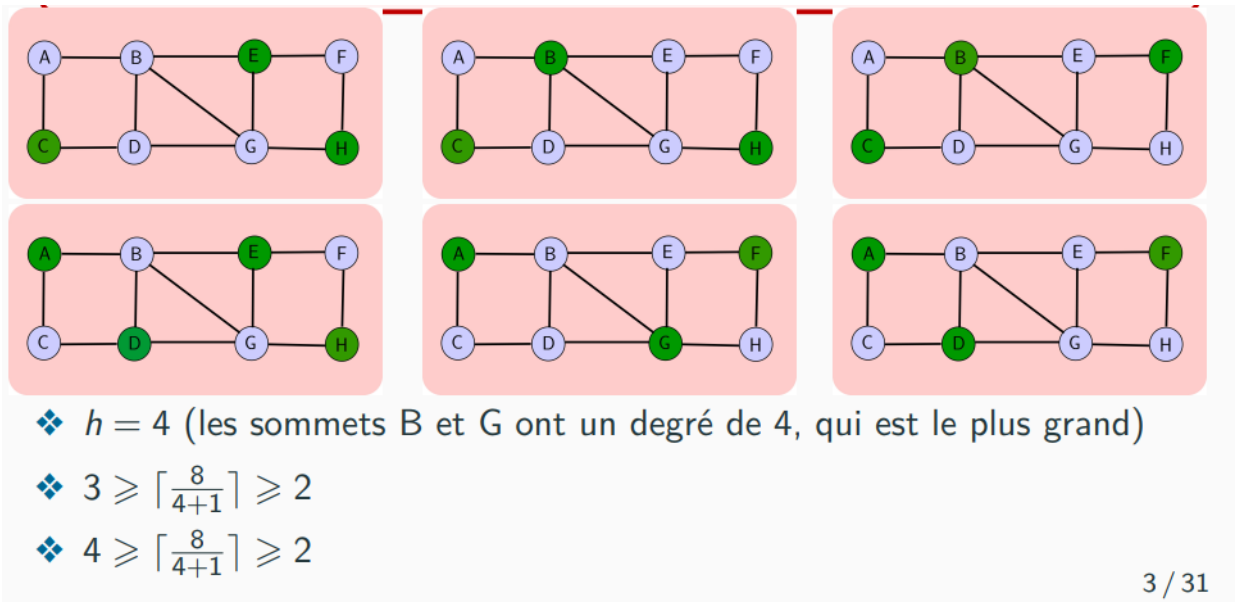
3.2 Number of stability

let $\alpha(G)$ is cardinal of the largest stable set.



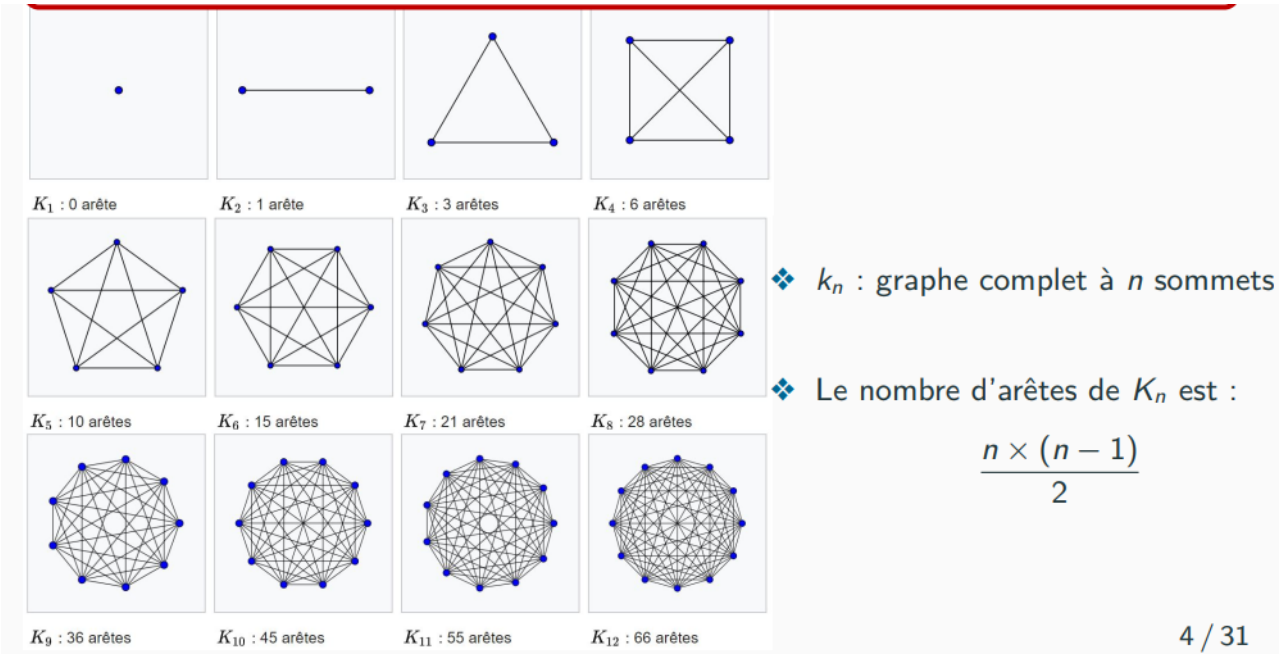
3.3 Stability

Theorem: In a graph with n vertices and maximum degree h, the cardinal of any maximal stable set is greater than or equal to $\lceil \frac{n}{h+1} \rceil$



3.4 Complete Graph

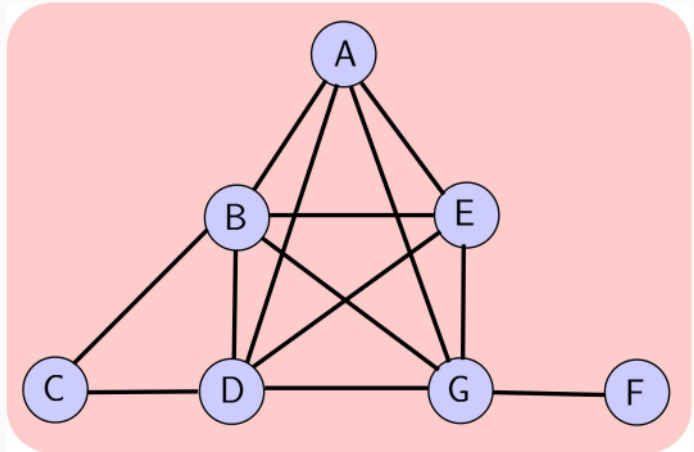
A complete graph is a graph in which all vertices are adjacent to each other, meaning that every pair of vertices is connected by an edge.



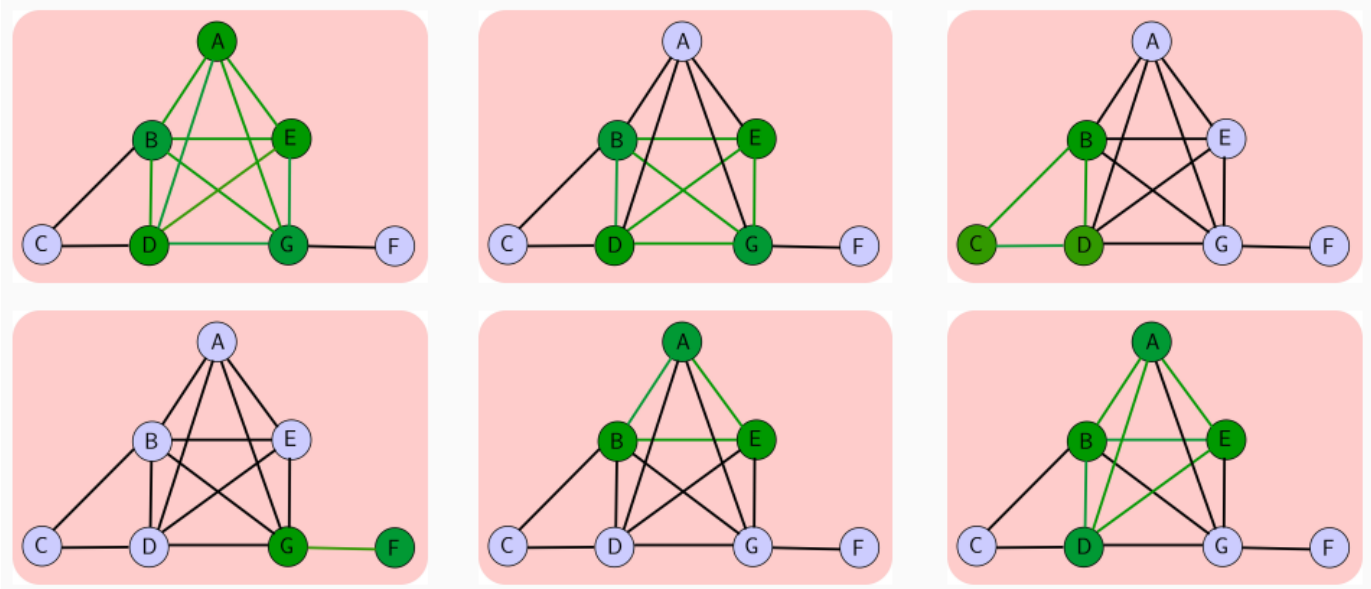
3.5 Cliques

Let an undirected graph G, a clique of G is a complete sub_graph of G.

Now let check the graph $G = (V, E)$ was given below:



Cliques of graph G:



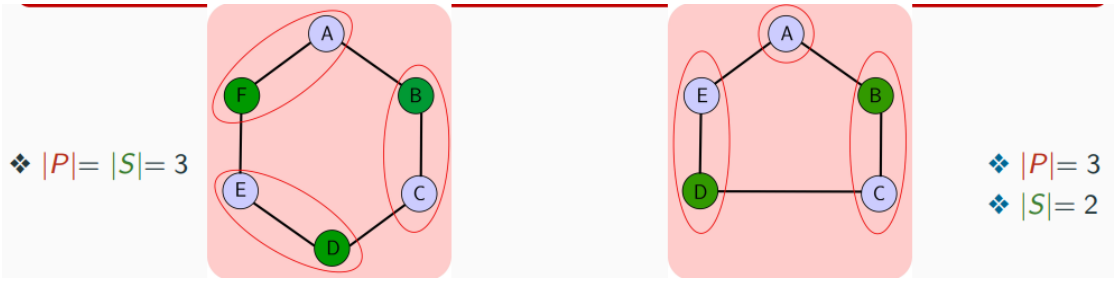
3.6 Partition of vertices into cliques

Let $G = (V, E)$ be a graph. $P = \{C1, C2, ..., Ck\}$ is a partition of the vertices of G into cliques if:

- $C_1, C_2, ..., C_k$ are the cliques of G
- $\forall i, j \in \{1, 2, 3, ..., k\} : V(C_i) \cap V(C_J) = \emptyset$
- $V(C_1) \cup V(C_2) \cup V(C_3) \cup .... \cup V(C_k) = V(G)$

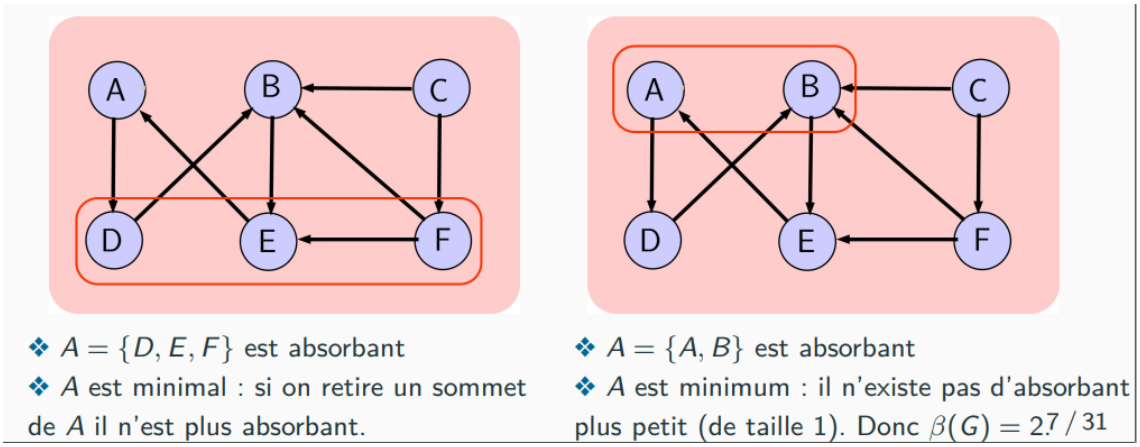
Theorem: Let S be a stable set of G and P be a partition into cliques of the vertices of G. Then, $|S| \leq |P|$. If $|S| = |P|$, then S is a maximum stable set and P is a minimum partition.

Example



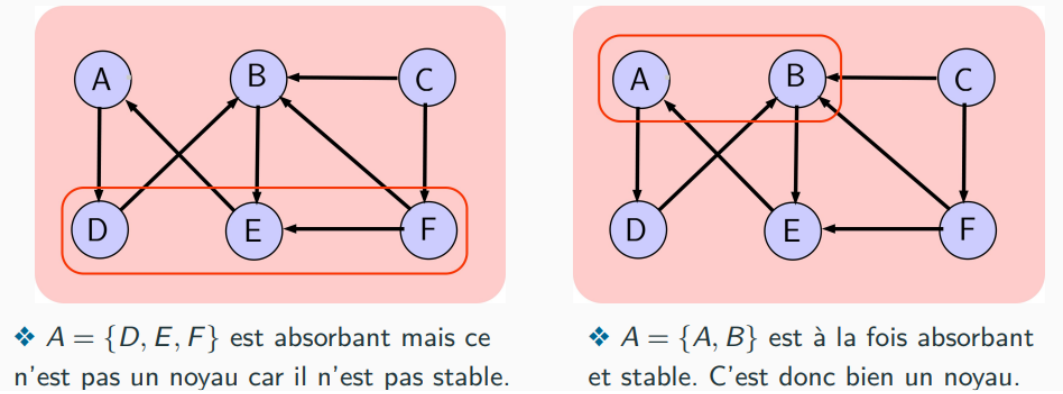
3.7 Absorbing Set

Consider a directed graph $G = (V, E)$. An absorbing set of G is a subset A of the vertices of G such that every vertex not in A has a successor in A. $\forall a \notin A$ there is a vertex $b \in A$ such that $(a, b) \in E$.

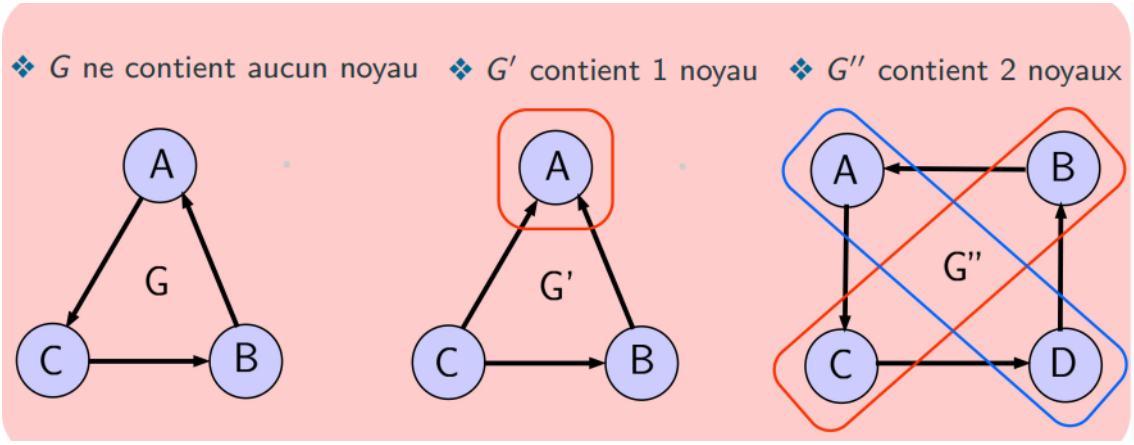


3.8 Kernel

Let $G = (V, E)$ be a directed graph. A subset of vertices K is a kernel of G when K is *stable* and *absorbing*.



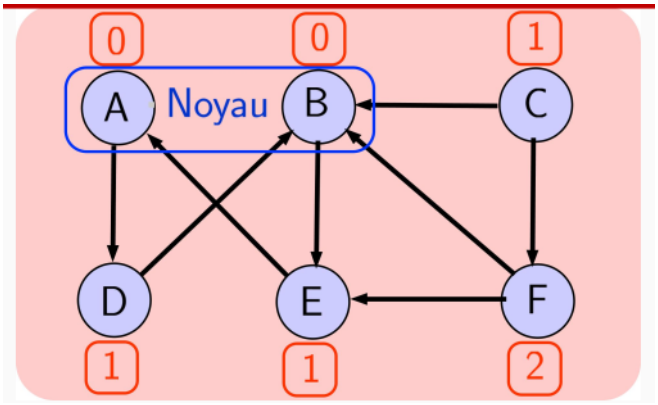
Remark: a graph can contain zero, one, or several kernels. Let see in example below:



3.9 Grundy function

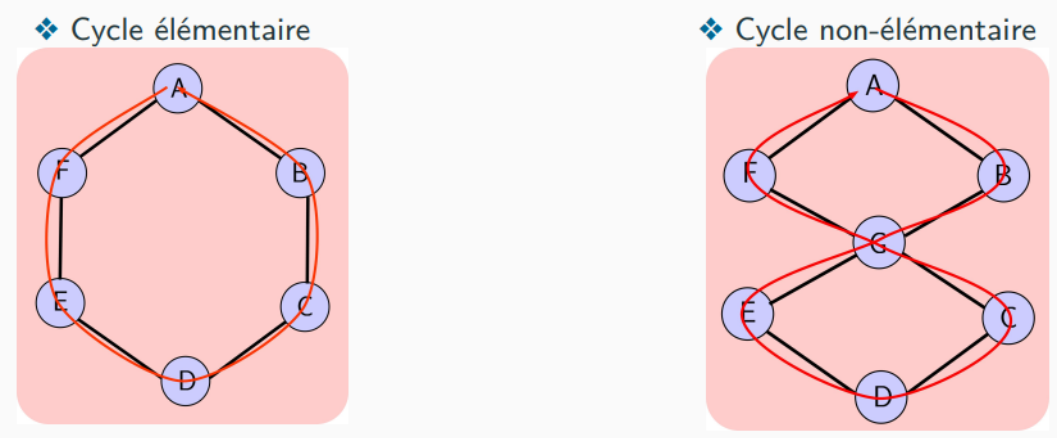
Let $G = (V, E)$ be a directed graph. g is a Grundy function of G if g is a mapping from V to $\mathbf{N}$ (the set of non-negative integers) such that $g(v)$ is the smallest integer not assigned to the successors of v.

Note: If a graph G has a Grundy function, the set of vertices in G such that $g(v) = 0$ forms a kernel of G.



3.10 Cycles basis

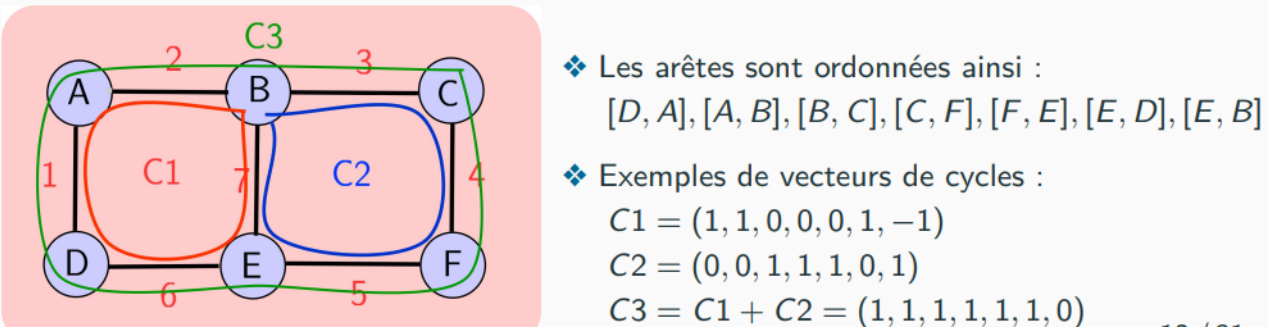
- **Cycles**: is a path that does not contain the same arc twice whose initial and terminal vertices coincide.
- **Elementary cycle** is Cycle that passes at most once through each vertex.



Vector Notation

The arcs of a graph being numbered from 1 to m, one can correspond to any cycle an m-tuple (a "vector") composed of $-1, 1$, and 0 in the following way:

- If the arc does not belong to the cycle, we put "0".
- If the arc belongs to the cycle and is traversed in the correct direction (referred to as "direct"), we put $+1$.
- If the arc belongs to the cycle but is traversed in the wrong direction (referred to as "inverse"), we put -1.



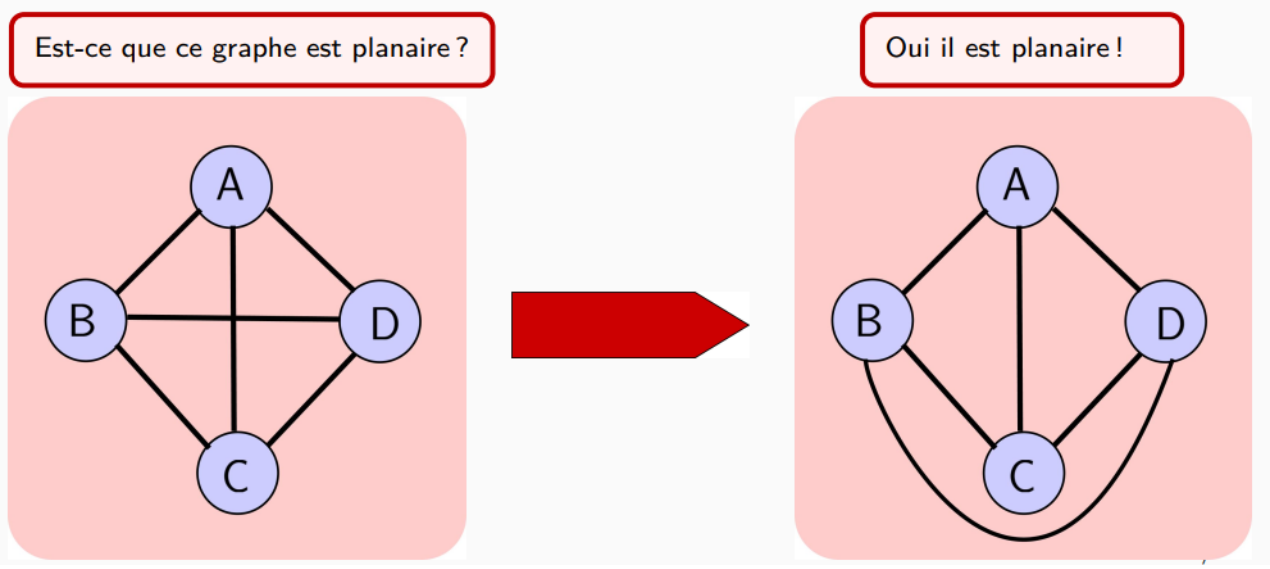
Cyclomatic number $\mu(G)$

Let $\mu(G)$ is number of elements in such a basis of cycles. we consider graph G with n nodes , m arcs and p is number of connect components so :

$$\mu(G) = m - n + p$$

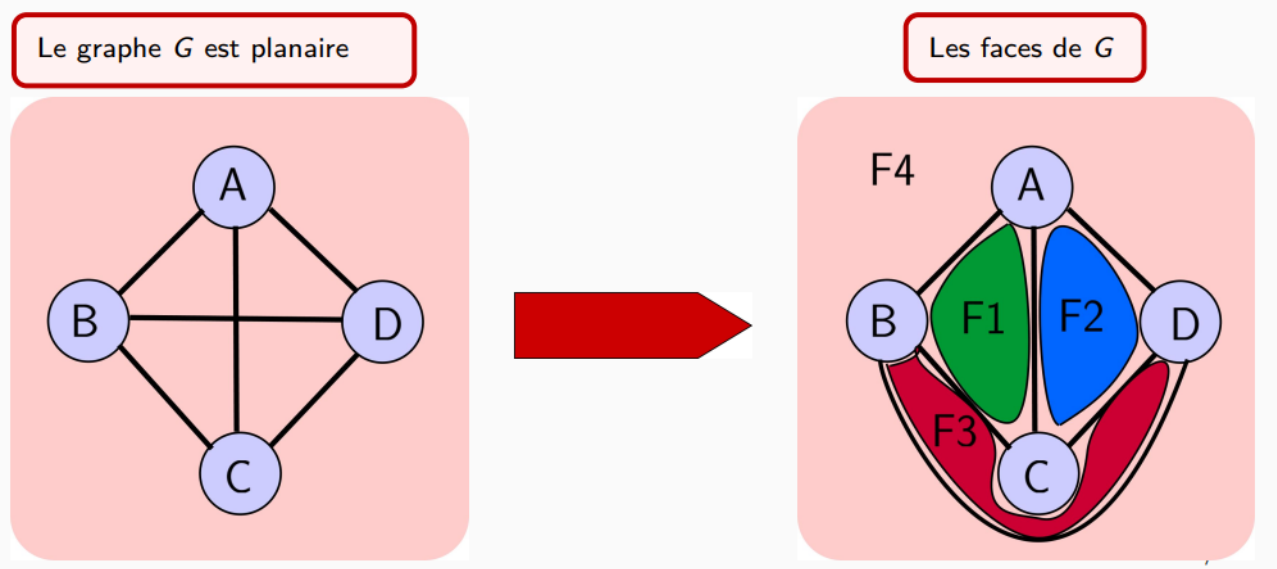
3.11 Planar graph

A planar graph is a graph that has the characteristic of being able to be represented on a plane without any edges (or arcs for a directed graph) crossing each other.



3.12 Face

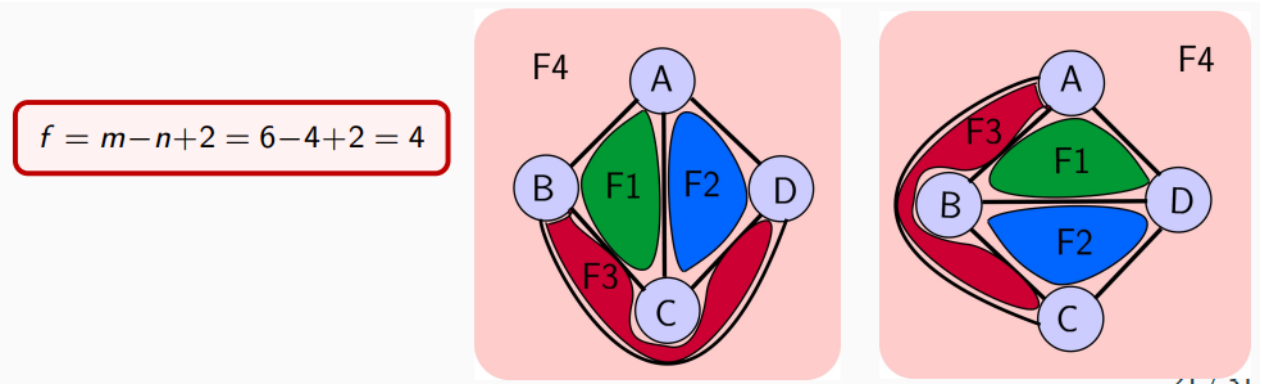
Region on the plane that is enclosed by edges of the graph + face infinity.



3.13 Euler's Formula for Planar Graphs

For any connected planar graph with vertices n, edges m and faces f, we have

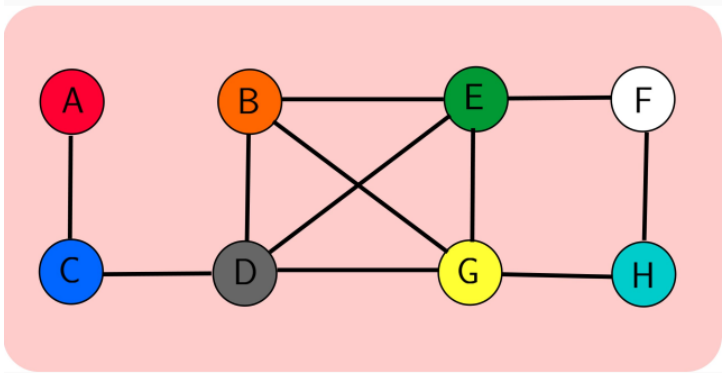
$$n - m + f = 2$$

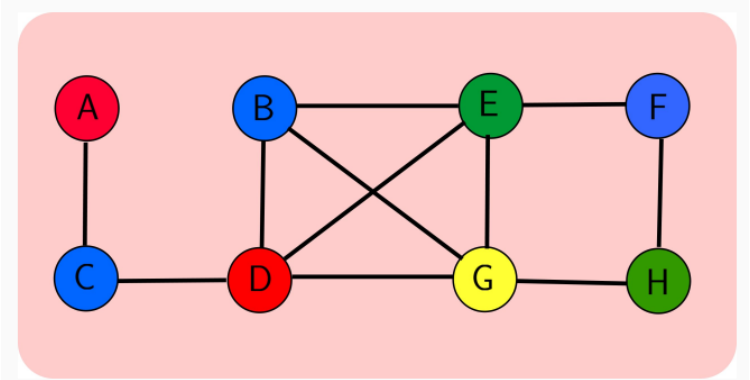


4 Coloring Graph

4.1 Definition

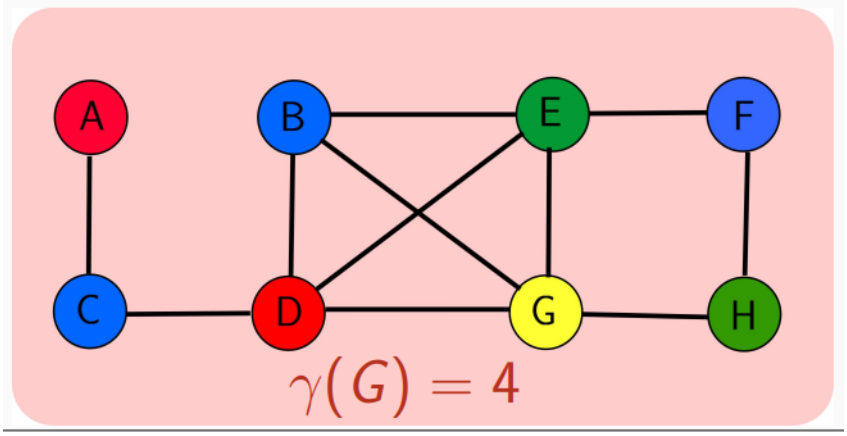
A coloring of vertices of G if no two adjacent vertices of G have the same color.





4.2 Chromatic number $\gamma(G)$

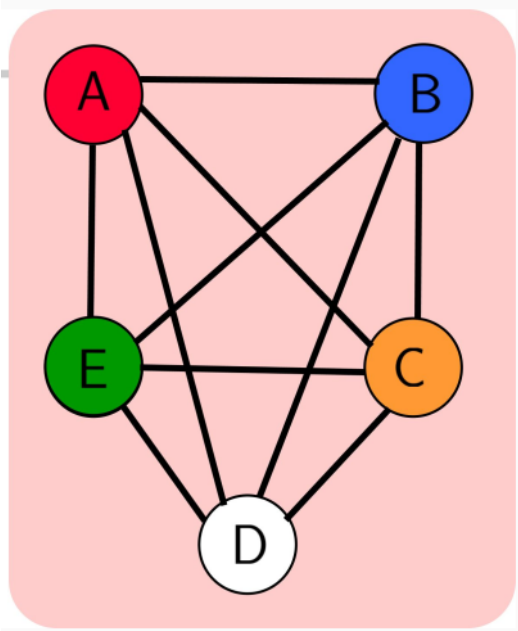
$$\gamma(G) = minimum \quad number \quad of \quad colors \quad of \quad graph(G)$$



Example: Let consider with K_5

The graph K_5 is not planar

The graph K_5 can be colored with exactly 5 colors

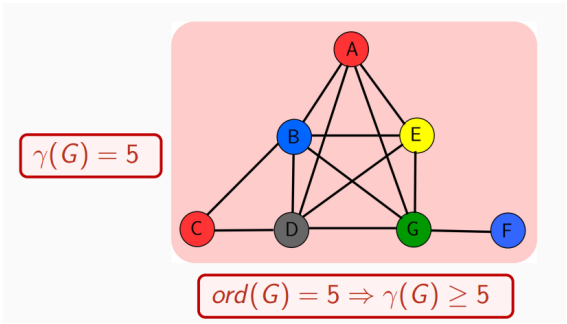


Remark: If $G=(V,E)$ is a planar graph, then $\gamma(G) \leq 4$.

Definition: The order $Ord(G)$ of a graph G is the size of a maximum clique in G.

4.3 Chromatic number properties

- Let $Ord(G)$ = size of cliques maximum of G then $\gamma(G) \geq Ord(G)$

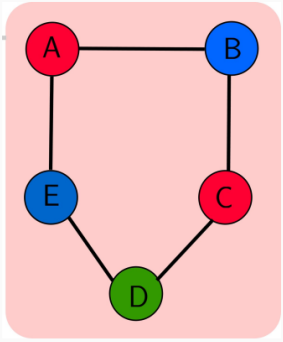


$\gamma(G) = 5$

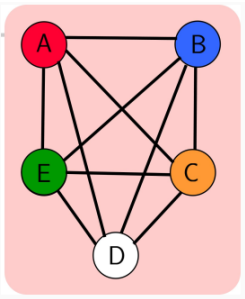
$ord(G) = 5 \Rightarrow \gamma(G) \geq 5$

H.W

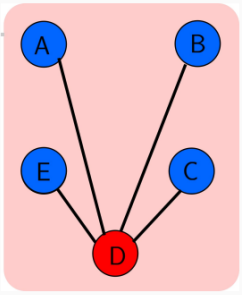
$Ord(G) = 2$
$\gamma(G) = 3$



- **Brooks theorems:** Let $d_{max}(G)$ is a maximum degree in graph G then $\gamma(G) \leq d_{max}(G) + 1$



$d_{max}(G)+1 = 5$ $\quad \gamma(G) = 5$



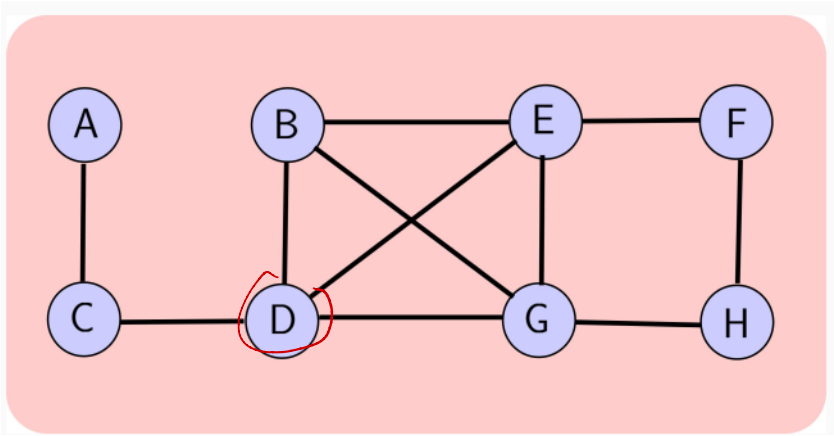
$d_{max}(G)+1 = 5$ $\quad \gamma(G) = 2$

4.4 Welsh Powell's Algorithms

Algorithm 5 Welsh Powell's Algorithms $G = (V, E)$, Output: upper bound color number

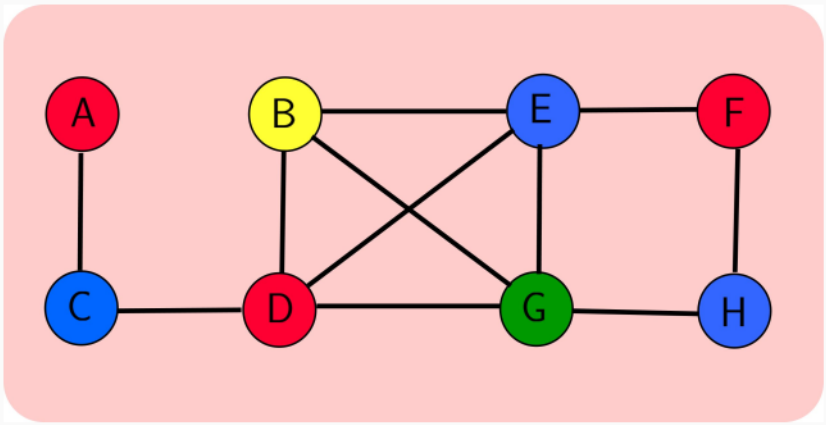
1. $G' \longleftarrow G$:
2. Sort the vertices in descending order of their degrees
3. **While** not all vertices of G' are colored **Do**
   i. Choose a color k different from the colors already used
   ii. Consider the first vertex not yet colored in the descending order of degrees and assign it the color k
   iii. Consider the other vertices not yet colored in the descending order of degrees
   - **If** it is adjacent to a vertex already colored with k **Then**: Do not assign any color to it
   - **Else** assign it the color k
4. **End While**

Example1: Let $G = (V, E)$ in figure below :



1. Applied Welsh Powell's Algorithms to determine color of graph G:
2. Is the solution is optimize the color number?

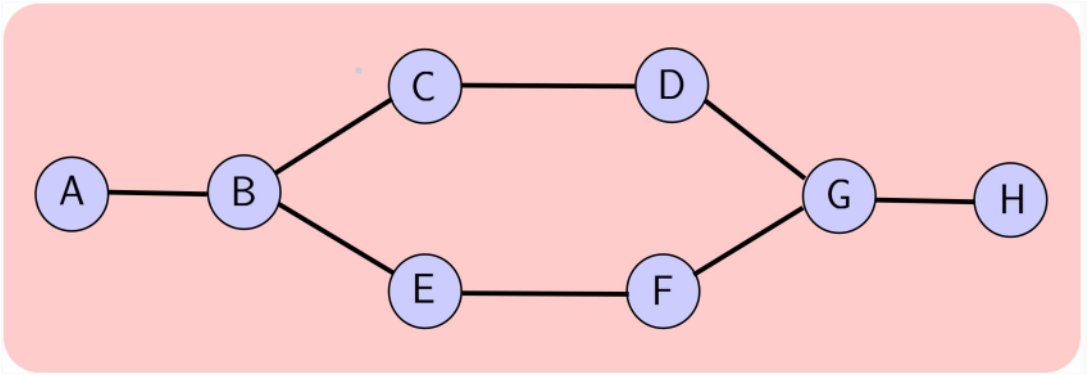
Solution



Sommets	D	E	G	B	F	H	C	A
degré	4	4	4	3	2	2	2	1
Rouge	✔				✔			✔
Bleu	✘	✔			✘	✔	✔	✘
Vert	✘	✘	✔		✘	✘	✘	✘
Jaune	✘	✘	✘	✔	✘	✘	✘	✘

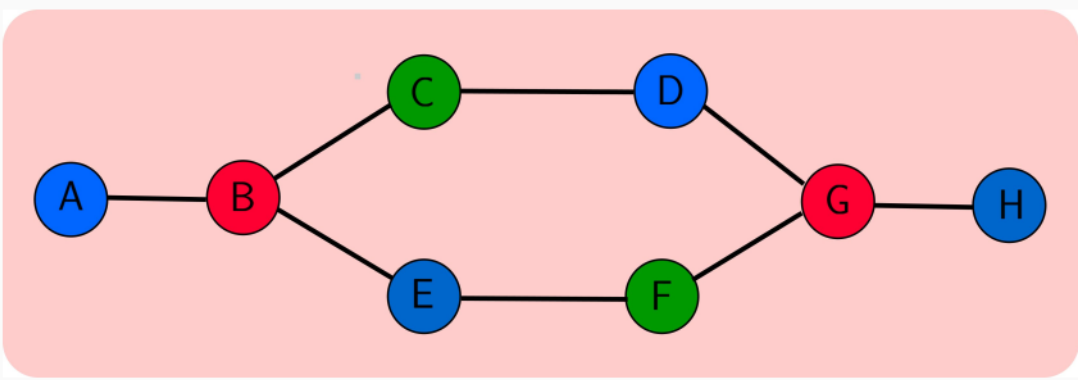
Red
Blue
Green
Yellow

Example2: Let $G = (V, E)$ in figure below :



1. Applied Welsh Powell's Algorithms to determine color of graph G:
2. Is the solution is optimize the color number?

Solution



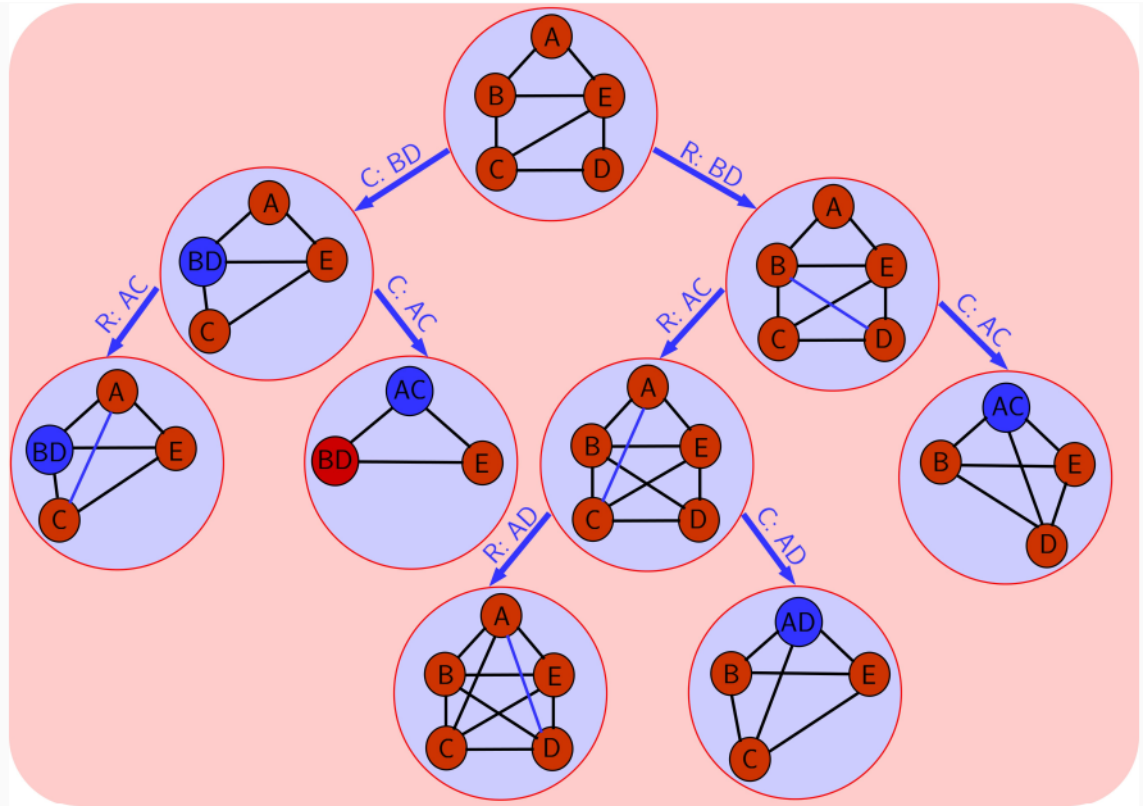
Sommets	G	B	E	F	D	C	H	A
degré	4	4	4	3	2	2	2	1
Rouge	✔	✔						
Bleu	✘	✘	✔		✔		✔	✔
Vert	✘	✘	✘	✔	✘	✔	✘	✘
Jaune								

4.5 Contraction

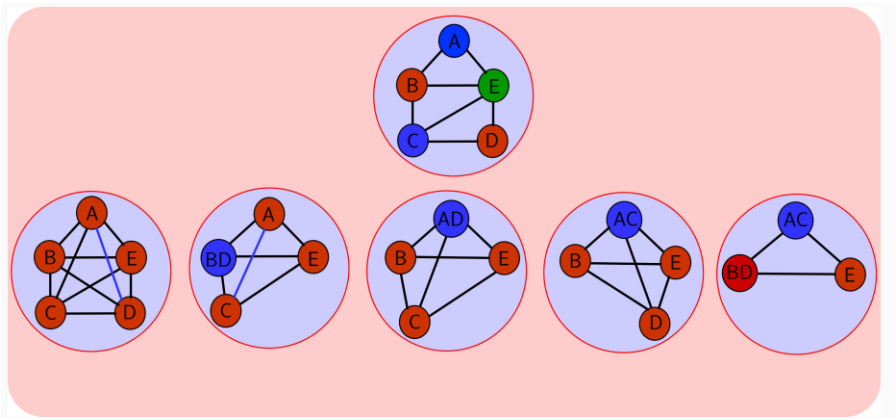
The contraction of a graph G is the graph obtained by identifying the vertices u and v, and removing any edges between them.

Link - contract

- To color a complete graph (clique) containing n vertices, you need n colors.
- Take two unconnected vertices, u and v.
  - If they have the same color, contract them
  - If they have different colors, add an edge between them.
- Continue until the graph becomes a complete graph (clique)



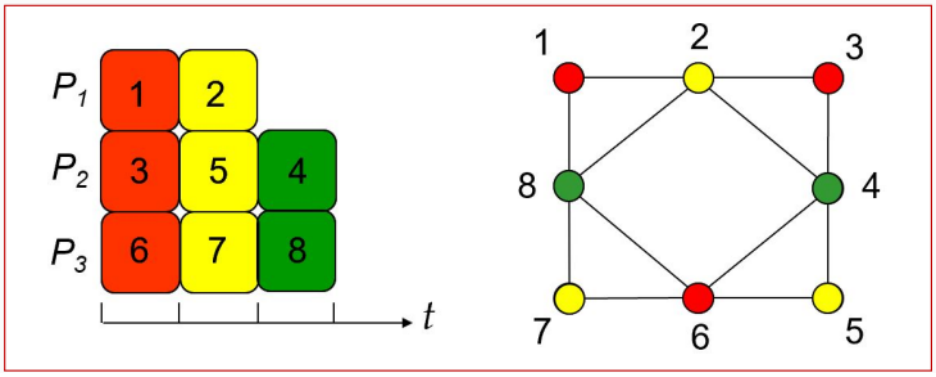
Result:



4.6 Fundamental problem in product planning

Planning n tasks on k process in minimum time, with some tasks not being able to be executed in parallel.

When all tasks have the same execution time $\Longrightarrow$ planning = graph coloring such that each color appears no more than k times



4.7 Edge coloring

- **Definition:** The color the edges of G if no two adjacent edges have the same color.
- **Chromatic index** $\gamma'(G)$ is smallest number of colors needed to color the edges of G without having two neighboring edges with the same color.

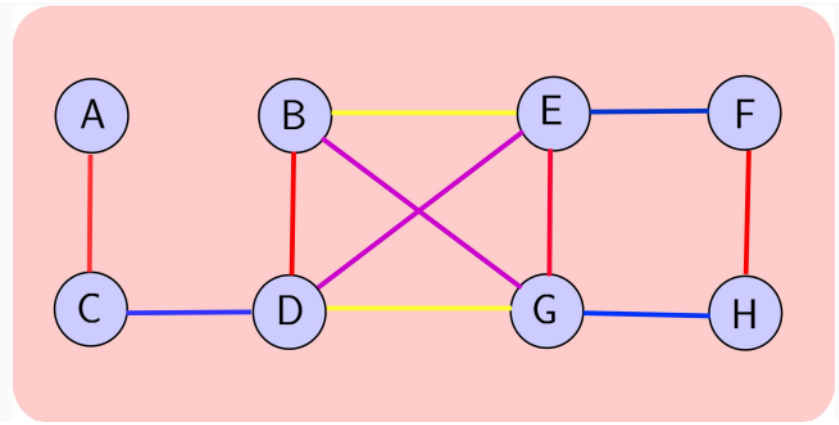
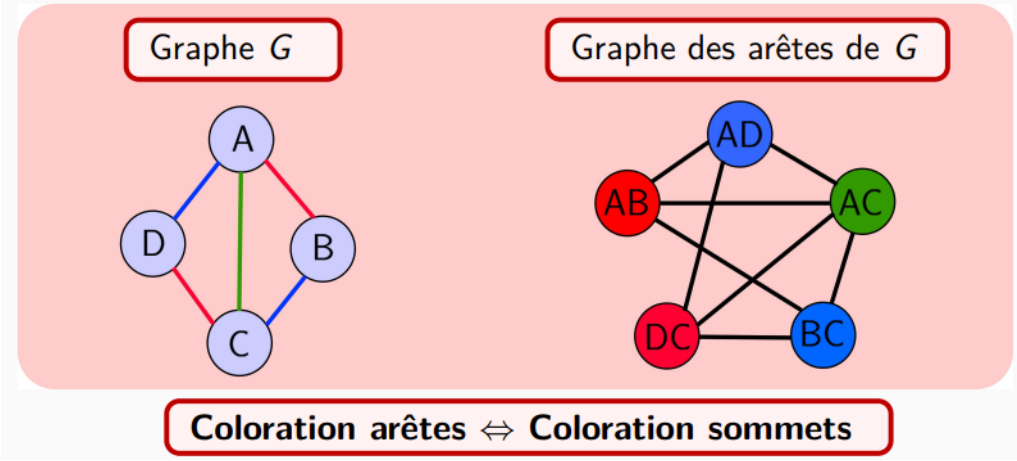


Figure 12: $\gamma'(G) = 4$

4.8 Line Graph

Let graph $L(G) = (Y, B)$ associated with $G = (X, A)$

- $Y = A$ (a vertex per edge of G)
- $[a, b]$ is an edge of L if and only if a and b have a common endpoint in G.

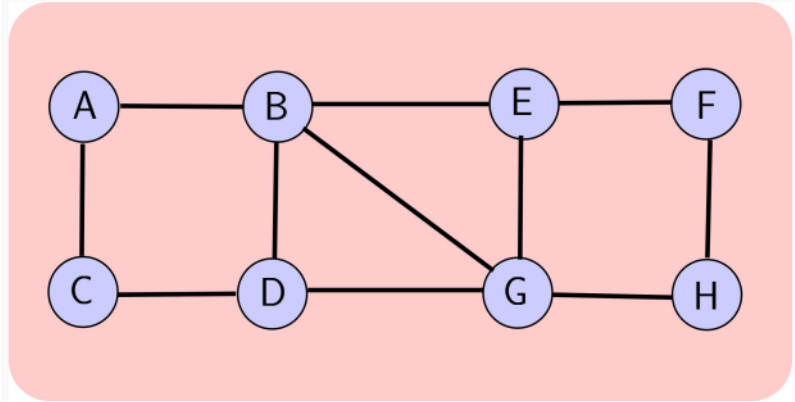


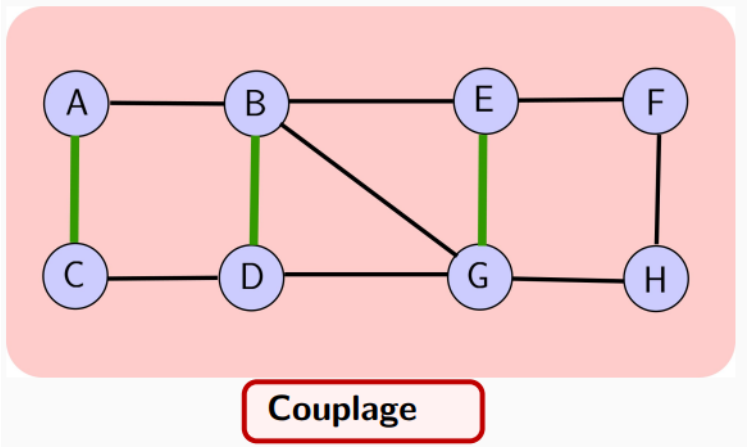
Note: Chromatic index of G is equal to Chromatic number of $L(G)$

5 Matching or Coupling

5.1 Definition

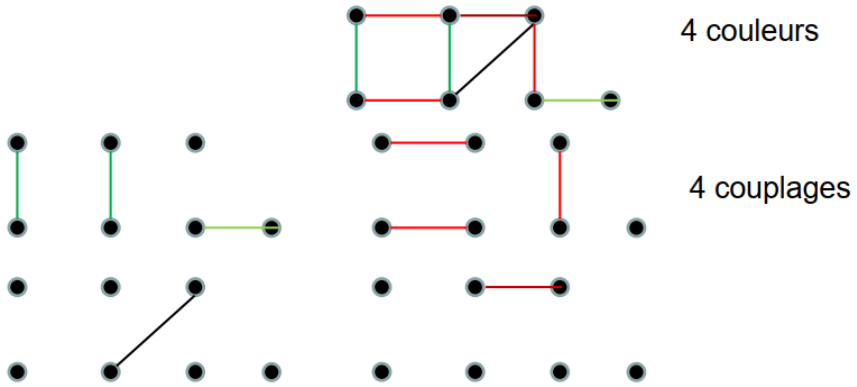
Matching of a graph is a set of edges of this graph which have no vertices in common (the edge not adjacent).





5.2 Edge coloring and Coupling

A coloring of the edges amounts to partition the edges into couplings.



5.3 Maximal coupling

Let K be maximal coupling of graph $G = (V, E)$ if all possible edges of G can be able to create a coupling K and if we add any edge of the graph G to this coupling K it will be lose the quality of coupling K.

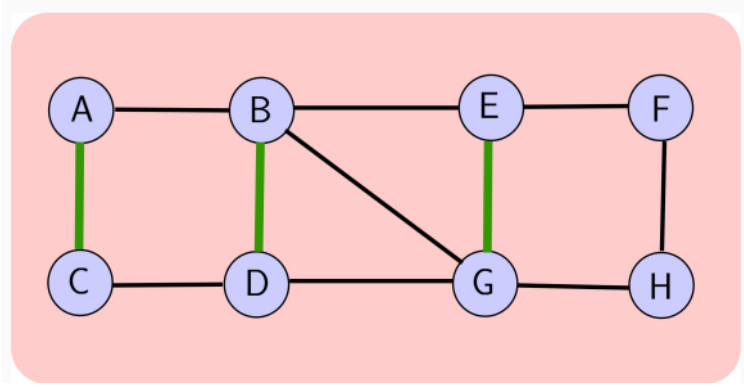


Figure 13: Not maximal coupling

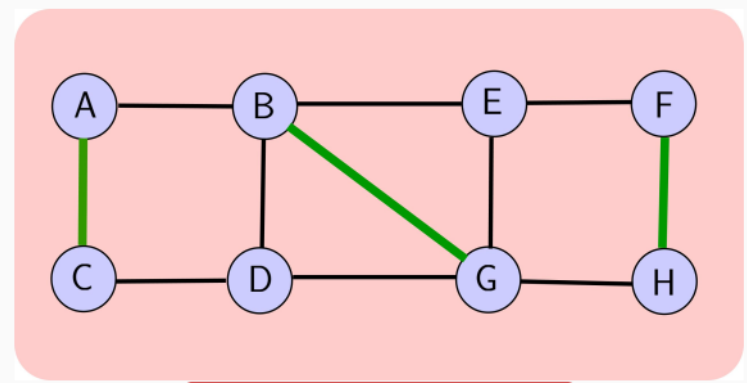


Figure 14: maximal coupling

5.4 Maximum coupling

Coupling that containing the greatest possible number of edges. A graph can have several maximum couplings.

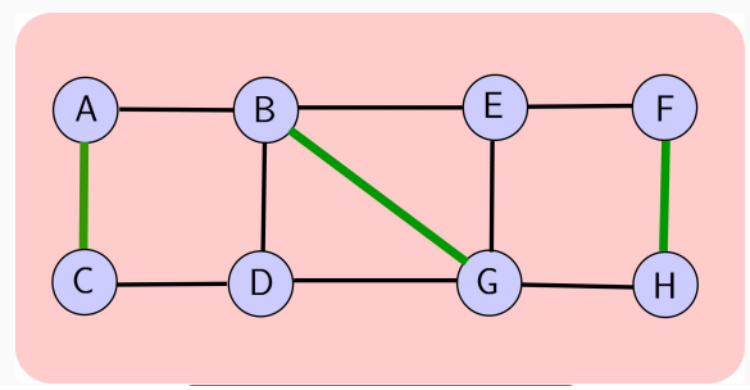


Figure 15: maximal coupling

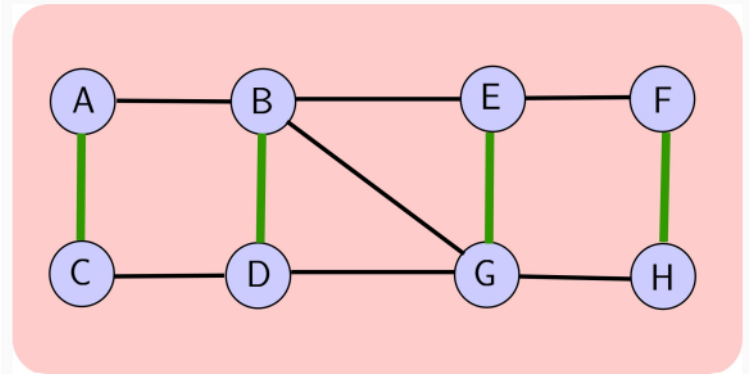
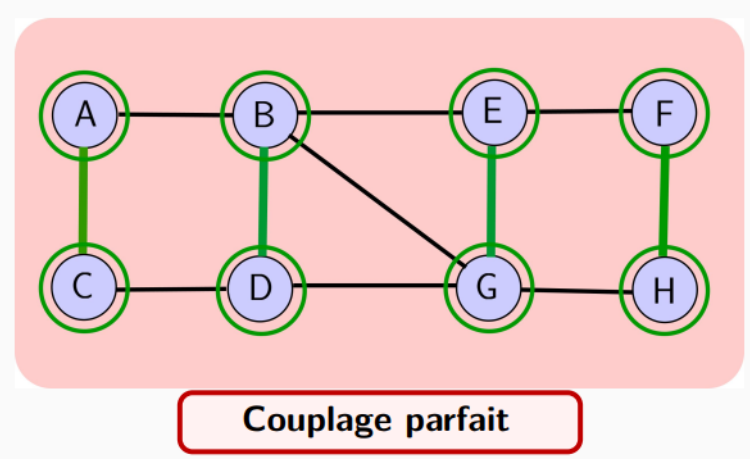


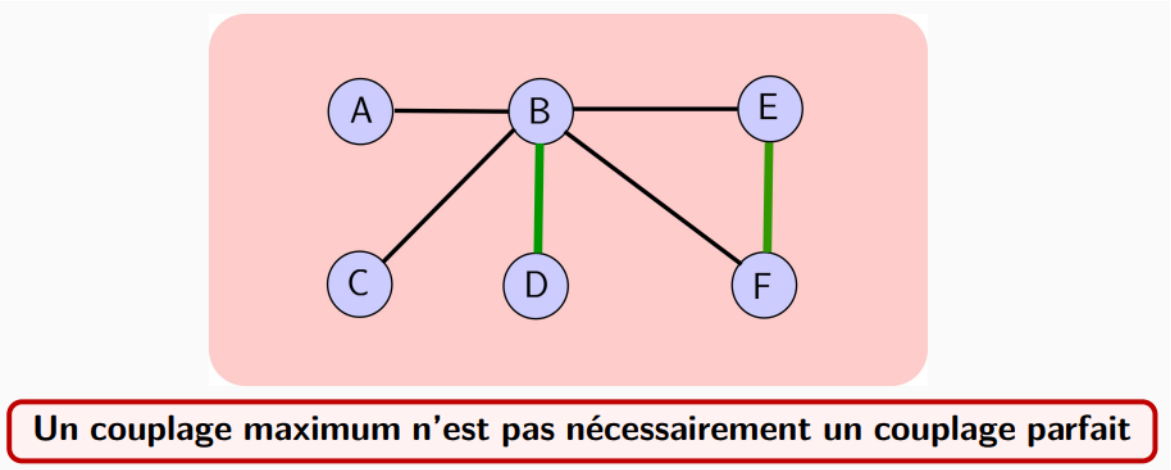
Figure 16: maximum coupling

5.5 Perfect coupling

A coupling such that each vertex of the graph is a vertex of the coupling.



Note: A maximum coupling is not necessary to be a perfect coupling.



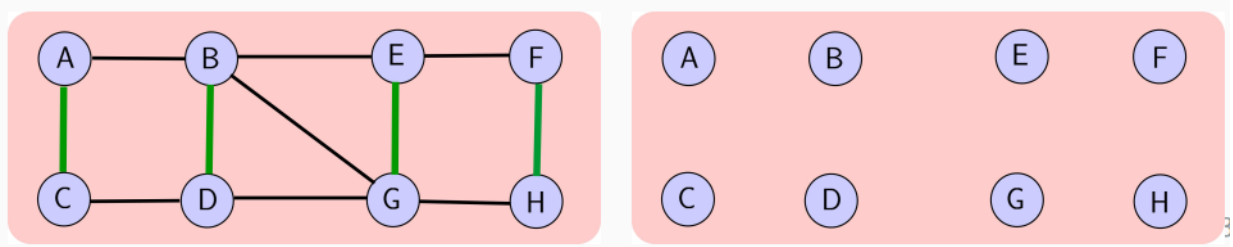
5.6 Maximal coupling algorithm

Algorithm 6 Maximal coupling algorithm

Input: A graph $G = (V, E)$

Output: A maximal coupling C_{max}

1. $G' \longleftarrow G$
2. $C_{max} \longleftarrow \emptyset$
3. **While** there are edges in G' **Do**
   i. Choose an edge e in G'
   ii. $C_{max} \longleftarrow C_{max} \cup \{e\}$
   iii. Remove from G' the edge $\{e\}$ and the neighboring edges of $\{e\}$
4. **End While**



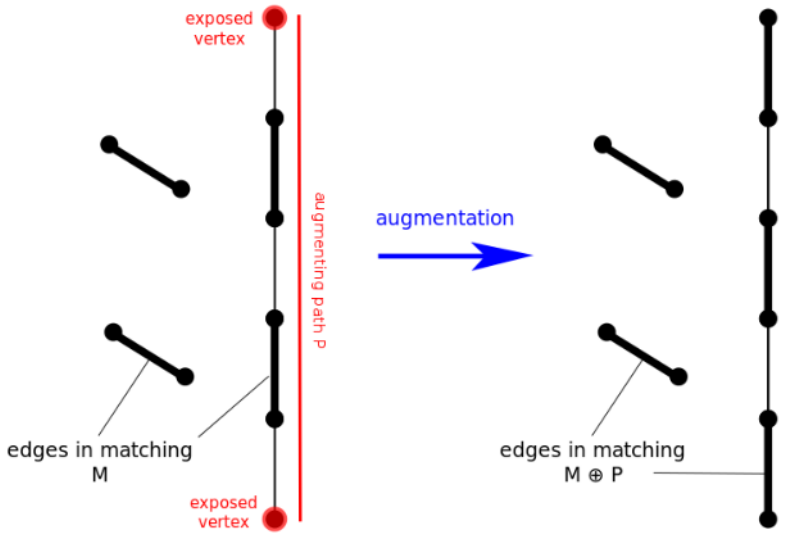
5.7 Alternating path and Increasing path (Augmenting path)

Let graph $G = (V, E)$ be a simple indirect graph and M be a coupling of G.

- **Alternating path** if it passes alternately through edges of M and edges of $E \setminus M$.

- **A path increasing** P is an alternating path that begins and ends with two distinct vertices not covered by M.

- **Properties**



5.8 Theorem [Berge 1957]

Let M be a matching in a graph G. M is maximum if and only if there is no augmenting path.

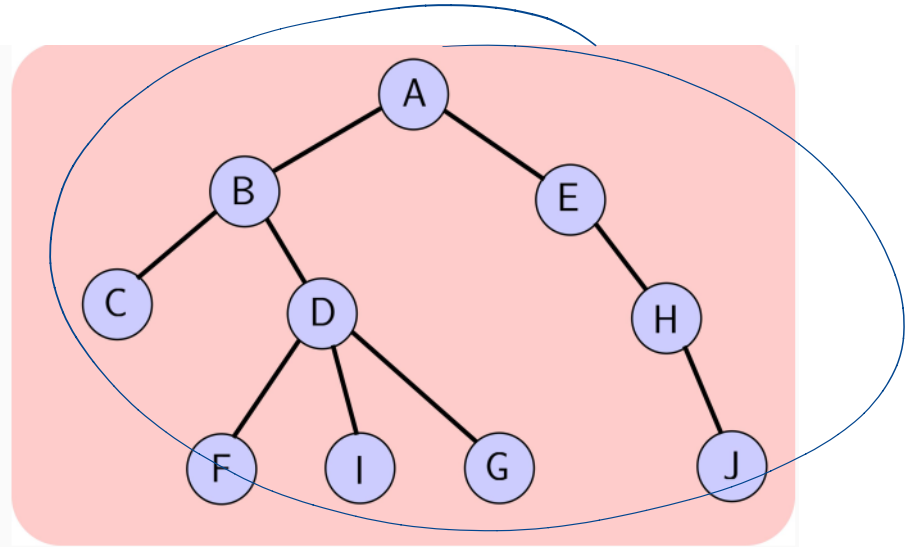
6 Tree

Tree: Connected graph G is acyclic (contains no cycles).

6.1 Definition

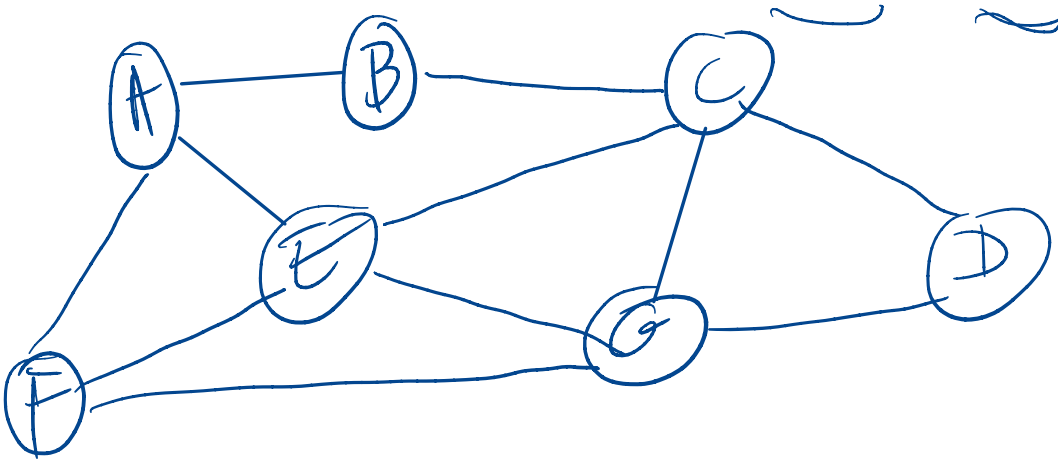
A tree is an undirected graph G that satisfies any of the following equivalent conditions:

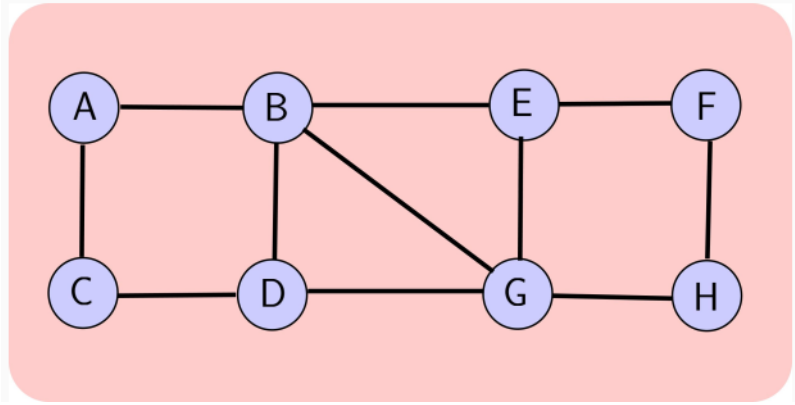
1. G is connected and acyclic (contains no cycles).
2. G is acyclic and exist $n-1$ edges.
3. G is connected and has $n-1$ edges.
4. G is acyclic and adding an edge then we creates only one cycle.
5. G is connected and if we remove an edge then we obtain a graph that not connected
6. G is containing a path and only One between any pair of vertices (simple path)

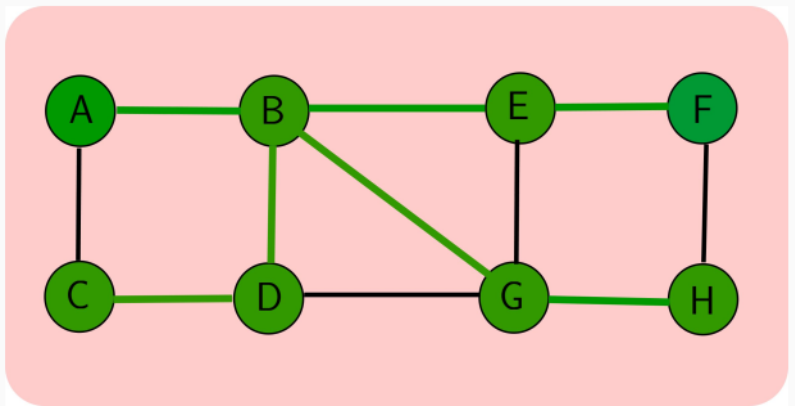


6.2 Spanning Trees

Spanning Trees T of graph G is a partial graph of G connected and acyclic.



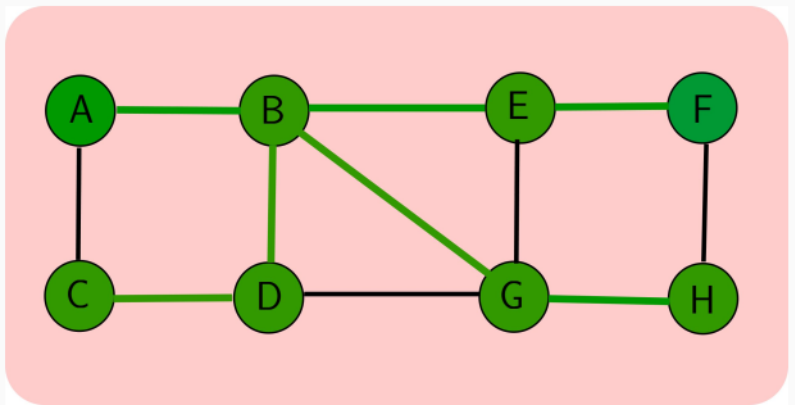


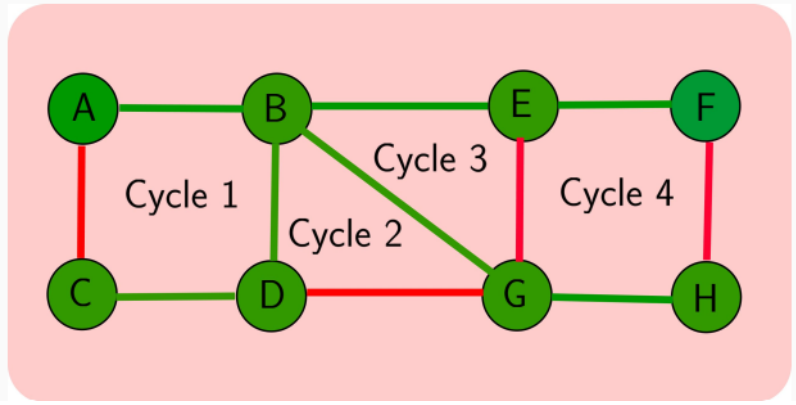


Theorem: A graph G has a partial graph that is a tree if and only if it is connected.

6.3 Construction of a cycle basis of a graph

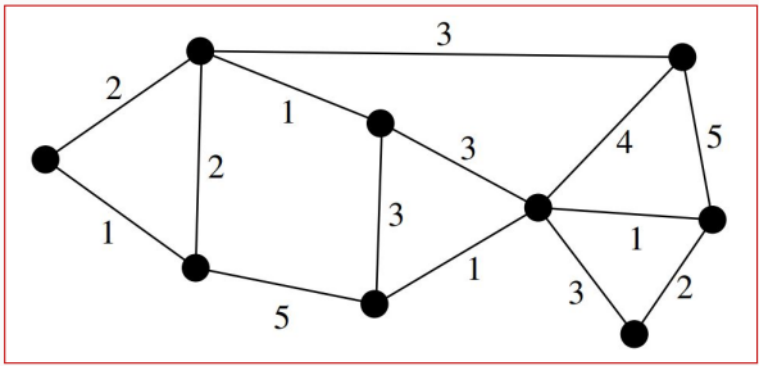
Let G be a connected graph, and T be a spanning tree of G. The cycles constructed by adding the edges of G that do not belong to T (one by one) then we obtain a cycle basis.

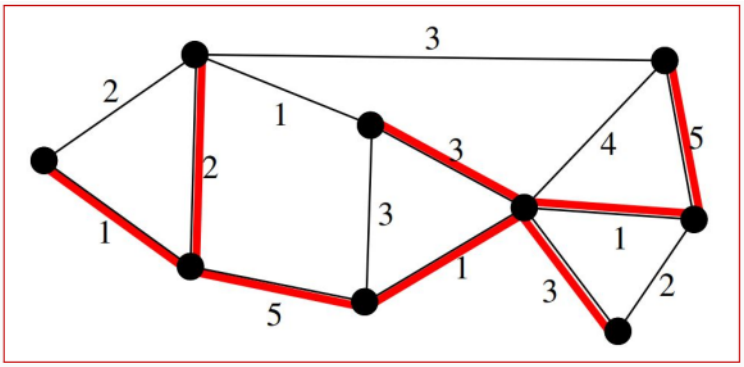




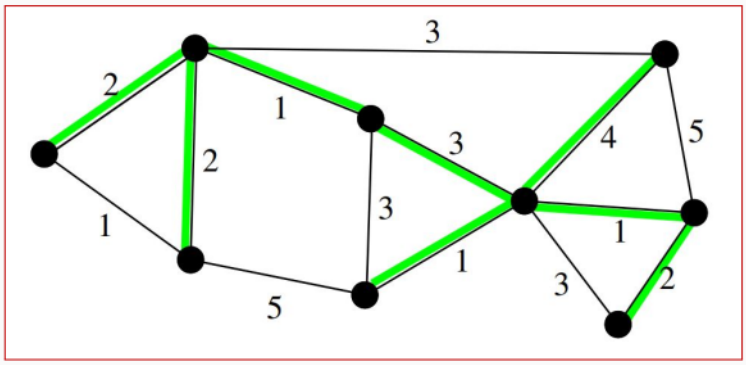
6.4 Weight minimum of spanning tree

Given a connected undirected graph with weighted edges, a Minimum Spanning Tree (MST) of this graph is a spanning tree whose sum of edge weights is minimal (i.e., less than or equal to that of all other spanning trees of the graph).

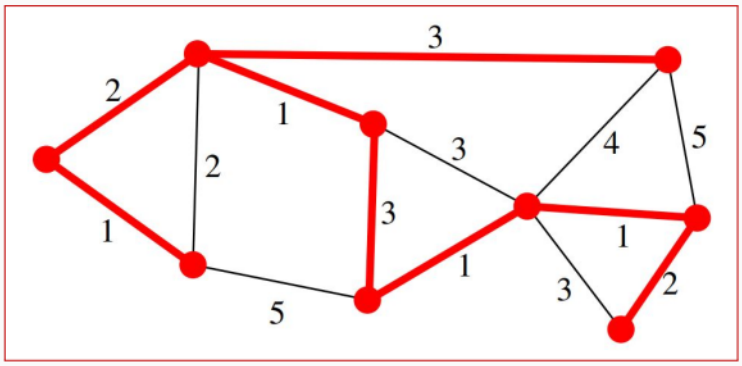




Le poids de cet arbre est : $1+2+5+1+3+5+1+3=21$. Peut-on faire mieux ?



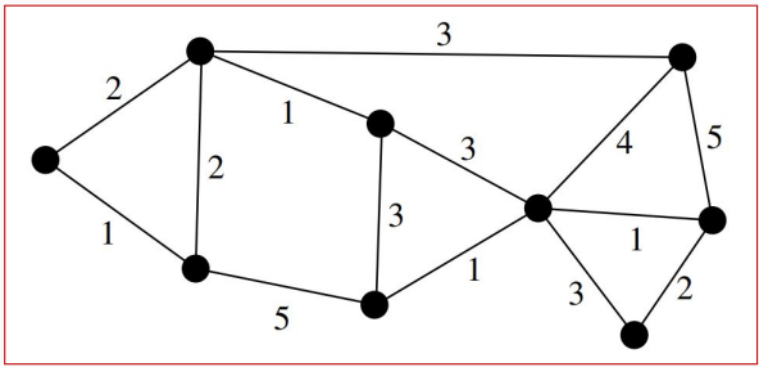
Le poids de cet arbre est : $2+2+1+3+1+4+1+2=16$. Peut-on faire mieux ?



Le poids de cet arbre est : $1+2+1+3+1+3+1+2=14$. C'est le ACM

6.5 Kruskal's Algorithm

:



Algorithm 7 Kruskal's Algorithm

Input: A graph $G = (V, E)$

Output: A minimum-weight spanning tree T

1. $T = (V', E') \longleftarrow (V, \emptyset)$
2. Set integer $k = 0$
3. Sort the edges of G in ascending order of weight
4. **While** $k \leq |V| - 1$ **Do**
   i. Traverse the sorted list
   ii. Select the first edge $\{e\}$ that does not create a cycle
   iii. $k \longleftarrow k + 1$
   iv. $E' \longleftarrow E' \cup \{e\}$
5. **End While**

6.6 Prim's Algorithm

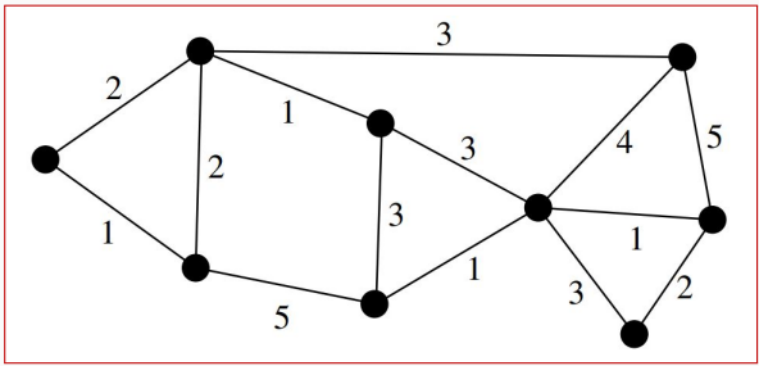
:

Algorithm 8 Prim's Algorithm

Input: A graph $G = (V, E)$

Output: A minimum-weight spanning tree T

1. $T = (V', E') \longleftarrow (\emptyset, \emptyset)$
2. $S \longleftarrow$ a vertex of G
3. Sort the edges of G in ascending order of weight
4. **For** i ranging from 1 to $|V| - 1$ **Do**
   i. Find an edge $[a, b]$ with $a \in S$ and $b \notin S$ of minimum weight
   ii. $T \longleftarrow T \cup [a, b]$
   iii. $S \longleftarrow S \cup \{b\}$
5. **End While**



7 Shortest Path Algorithm in Graph

7.1 Dijkstra's algorithm

The Dijkstra's algorithm finds the shortest path from a source vertex to all other vertices in a graph. It can only be used on graphs without circuits of strictly negative weights.

Algorithm 9 Dijkstra's algorithm

Input: A weighted graph $G = (V, E)$ without circuits of strictly negative weights, a source vertex s.

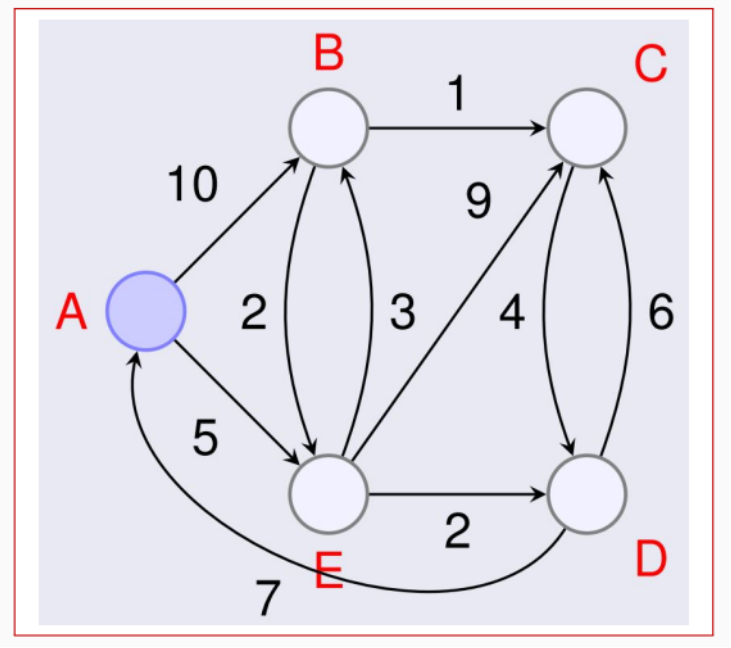
Output: The shortest path from the source to any vertex

1. **For** $v \in V$ **Do**

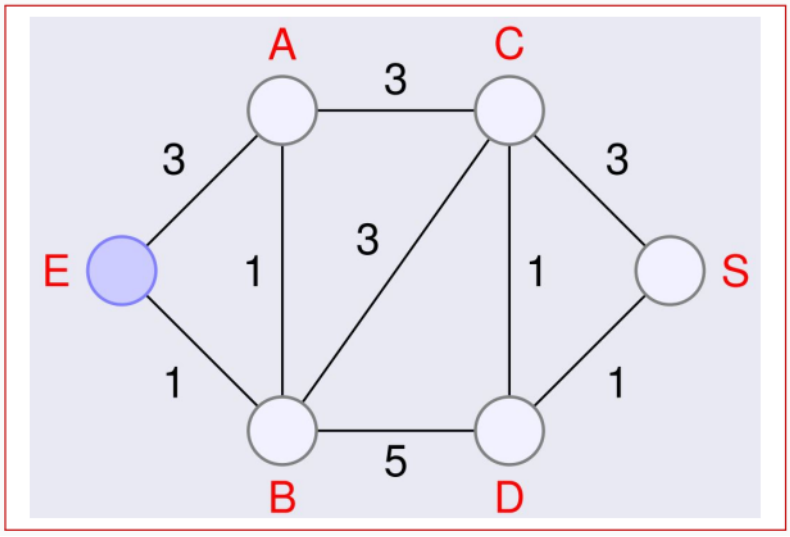
   $d[v] \longleftarrow +\infty \quad \pi[v] \longleftarrow NULL$

2. **End For**
3. $d[s] \longleftarrow 0$
4. $E \longleftrightarrow \emptyset$
5. $F \longleftarrow V(G)$ // Set of vertices of G
6. **While** $F \neq \emptyset$ **Do**
   i. $u \longleftarrow v, d[v] = \min_{x \in F} d[x]$
   ii. $F \longleftarrow F \setminus \{u\}$
   iii. $E \longleftarrow E \cup \{u\}$
   iv. **For** v is neighbour of u **Do**
   If $d[v] > d[v] + w(u, v)$ **Then**
   $d[v] \longleftarrow d[u] + w(u, v)$
   $\pi[v] \longleftarrow u$
   End If
   v. **End For**
7. **End While**

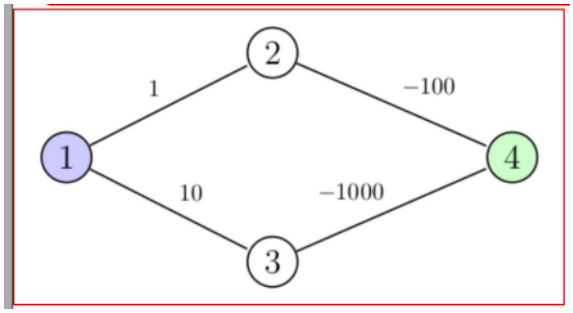
Example 1



Example2



Example 3



7.2 Bellman Ford's algorithm

The Bellman-Ford algorithm finds the shortest path from a source vertex to all other vertices in a graph. It can be use for any types of graph G.

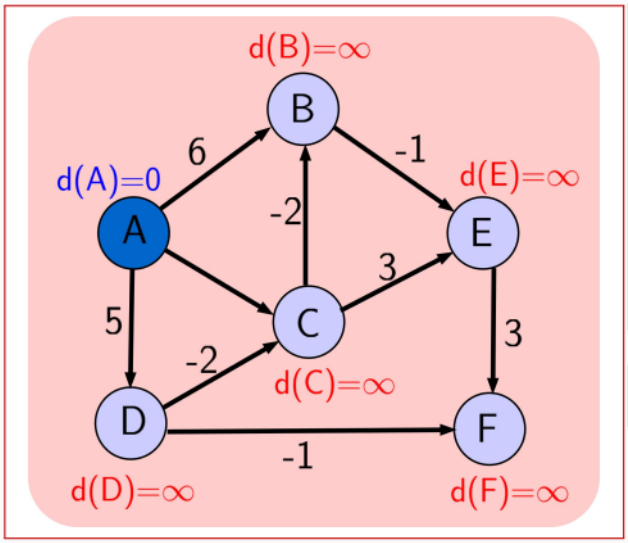
Algorithm 10 Bellman Ford's algorithm

Input: A weighted graph $G = (V, E)$, a source vertex s, and weights on the edges.

Output: A boolean indicating whether there exists a circuit with strictly negative weight, and if so, the shortest path from the source to any vertex.

1. **For** $v \in V$ **Do**
   $d[v] \longleftarrow +\infty \quad \pi[v] \longleftarrow NULL$
2. **End For**
3. $d[s] \longleftarrow 0$
4. **For** i ranging from 1 to $|V| - 1$ **Do**
   i. **For** $(u, v) \in E$ **Do**
   If $d[v] > d[v] + w(u, v)$ **Then**
   $d[v] \longleftarrow d[u] + w(u, v)$
   $\pi[v] \longleftarrow u$
   End If
   ii. **End For**
5. **End For**
   // Detection of negative cycles
6. **For** $(u, v) \in E$ **Do**
   If $d[v] \longleftarrow d[u] + w(u, v)$ **Then**
   Return False
   End If
7. **End For**
8. **Return True**

Example



Sommets	A	B	C	D	E	F
Itération 0	0	∞	∞	∞	∞	∞
Itération 1						
Itération 2						
Itération 3						
Itération 4						
Itération 5						

	$\pi[A]$	$\pi[B]$	$\pi[C]$	$\pi[D]$	$\pi[E]$	$\pi[F]$
valeur	NUL	NUL	NUL	NUL	NUL	NUL

pour i de 1 à $|V|-1$ **faire**
  pour tout arc $(u,v) \in E$ **faire** // 1
    si $d[v] > d[u] + w(u,v)$ **alors**
      $d[v] \leftarrow d[u] + w(u,v)$, $\pi[v] \leftarrow u$

7.3 Floyd Warshall algorithm

The Floyd-Warshall algorithm finds the shortest path between every pair of vertices in the graph. It is applicable to any type of graph.

Algorithm 11 Floyd Warshall's algorithm

Input: A weighted graph $G = (V, E)$ with a weighting function w .

Output: A matrix V^2 shortest path

1. **For** $v \in V$ **Do**

   $d[v][v] = 0$

2. **End For**
3. **For** $(u, v) \in V$ **Do**

   $d[v][u] \longleftarrow w(u, v)$

4. **End For**
5. **For** k ranging from 1 to $|V|$ **Do**

   For i ranging from 1 to $|V|$ **Do**

   For j ranging from 1 to $|V|$ **Do**

   If $d[i][j] < d[i][k] + d[k][j]$

   $d[i][j] \longleftarrow d[i][k] + d[k][j]$

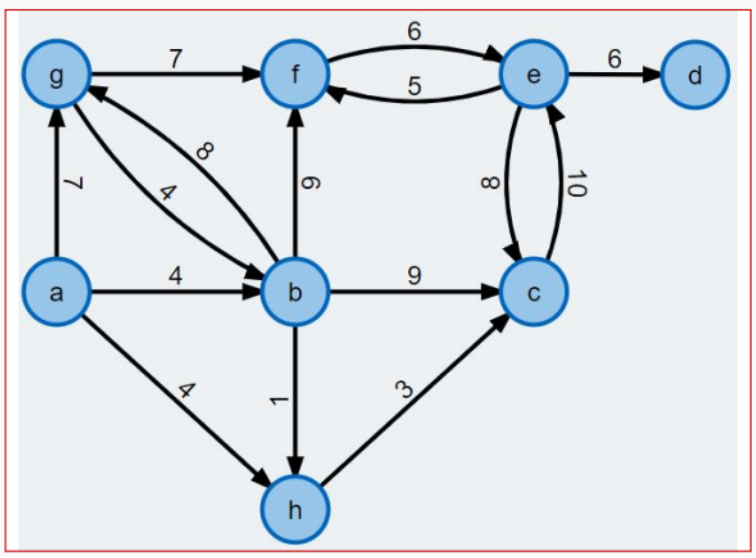
   End IF

   End For

   End For

6. **End For**

Example



	a	b	c	d	e	f	g	h
a	0	4	∞	∞	∞	∞	7	4
b	∞	0	9	∞	∞	6	8	1
c	∞	∞	0	∞	10	∞	∞	∞
d	∞	∞	∞	0	∞	∞	∞	∞
e	∞	∞	8	6	0	5	∞	∞
f	∞	∞	∞	∞	6	0	∞	∞
g	∞	4	∞	∞	∞	7	0	∞
h	∞	∞	3	∞	∞	∞	∞	0