TD1 Graphes

1) Exercices élémentaires

1.1) Degré, connexité

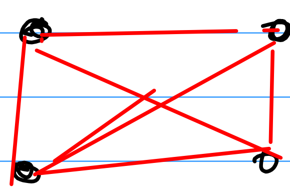
EXERCICE #1 ▸ **Une preuve par construction**

Un graphe est dit k-régulier si tous ses sommets sont de degré k. Prouver la propriété suivante :

Pour tout entier n pair, $n > 2$, il existe un graphe 3-régulier composé de n sommets.

On fait une preuve par récurrence

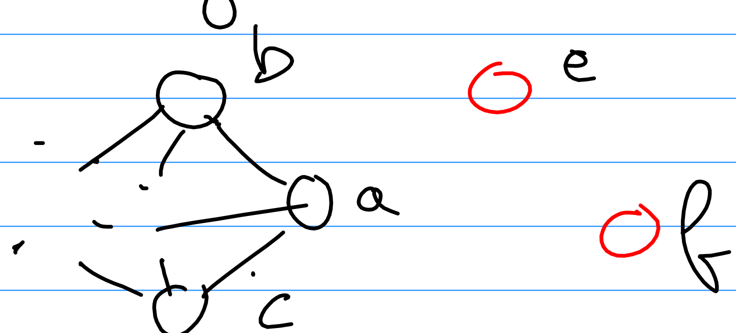
Initialisation : pour $n = 4$



On considère le graphe complet

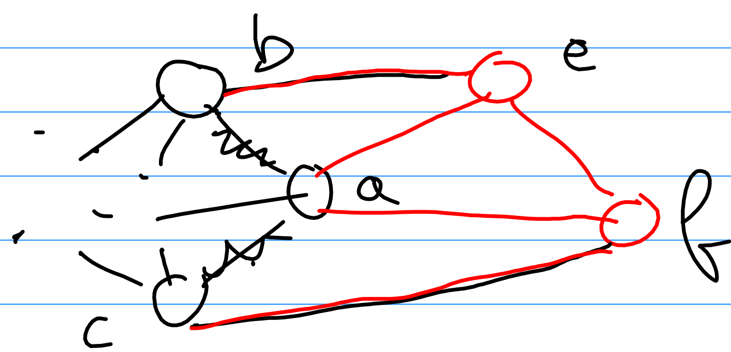
Hérédité : Supposons qu'il existe un graphe 3-régulier pour n sommets avec n pair

On rajoute deux sommets e et f



On considère un sommet s du graphe à n sommets et deux de ses voisins b et c.

On remplace l'arête (b, a) par (b, e) et l'arête (c, a) par (c, f).



On rajoute aussi les arêtes (a, e), (e, f) et (a, f).

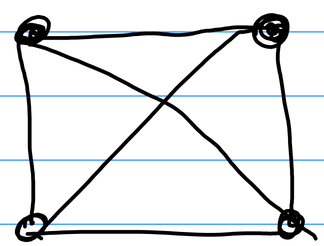
Ainsi les sommets a, b, c sont toujours de degré 3 et e et f le sont aussi.
On a donc construit un graphe avec n+2 sommets qui est 3-régulier.

<u>Conclusion:</u> application de l'axiome de récurrence.

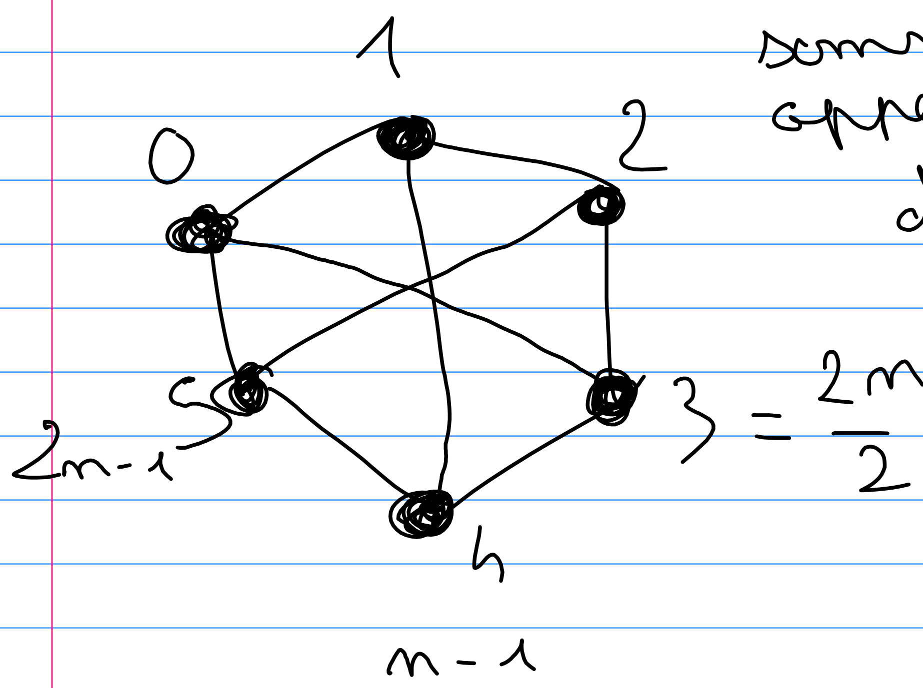
Correction :

Exercice 1 : Preuve par construction

- Pour $n = 4$; on prend le graphe complet



- On dispose les sommets sur un cercle et on associe chaque sommet à son opposé (décalage de $\frac{2n}{2} = n$) pour que chaque sommet soit de degré 3



EXERCICE #2 ▸ **Degré**

Si m est le nombre d'arêtes d'un graphe G, montrer que

$$\sum_{v \in V(G)} d_G(v) = 2m.$$

Notons A l'ensemble des arêtes

$$\sum_{u \in V(G)} d_G(v) = \sum_{u \in V(G)} \sum_{\substack{v \in V(G) \\ \text{tel que} \\ (u,v) \in A}} 1 = \sum_{\substack{u \in V(G), \\ v \in V(G) \\ \text{tel que} \\ (u,v) \in A}} 1 = 2m$$

nombre d'arêtes

Chaque arête (u,v) est comptée dans $d_G(v)$ et dans $d_G(u)$ donc dans $\sum_{u \in V(G)} d_G(v)$ chaque arête est comptée deux fois,

donc $\sum_{v \in V(G)} d_G(v) = 2m$

EXERCICE #3 ▸ **Connexité**

ligne $n-1$

Montrez que tout graphe connexe à n sommets a au moins n arêtes. $n \geq 2$

P_n : "Un graphe connexe à n sommets a au moins n arêtes"

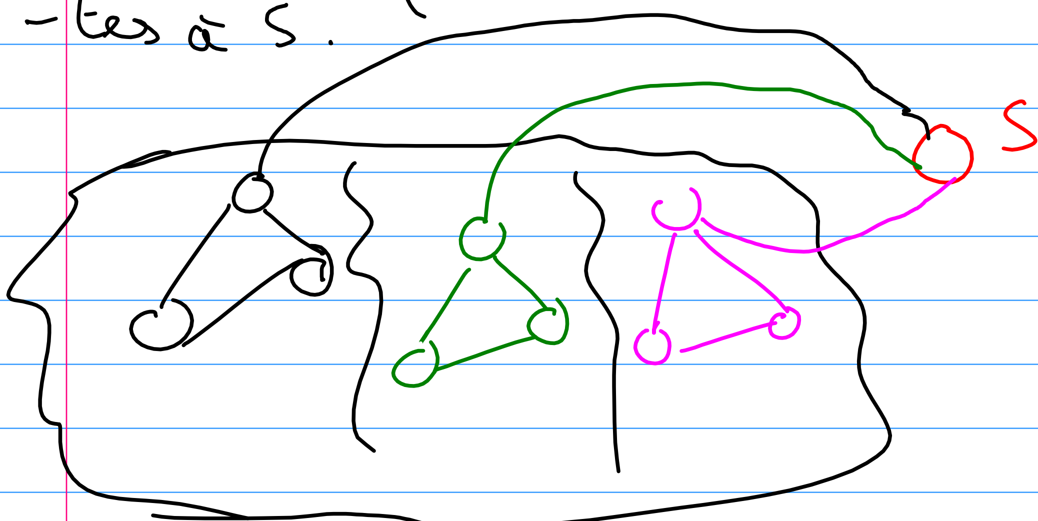
On fait une preuve par récurrence forte.

Initialisation : Un graphe connexe à 2 sommets a au moins une arête sinon il n'est pas connexe

Hérédité : On formule une hypothèse de récurrence forte : pour tout entier $2 \leq m \leq n$, J_m est vraie.

Soit un graphe G connexe à $n+1$ sommets.

Soit un sommet s quelconque de ce graphe ; on enlève s du graphe G ainsi que toutes les arêtes incidentes à s.



Soit $G' = G \setminus \{s\}$ sous graphe de G

et soit ses composantes connexes numérotées de 1 à k et dont les nombres de sommets sont $n_1, \dots, n_k$

avec $n_1 + \dots + n_k = n - 1$

Puisque G est connexe on a au moins une arête dans G reliant s à

Chacune des composantes connexes de G' :

De plus par hypothèse de récurrence forte, chaque composante connexe a au moins :

$n_1 - 1$ arêtes
$n_2 - 1$ arêtes
$\vdots$
$n_k - 1$ arêtes

soit au total $n_1 + \dots + n_k - k$ arêtes

En ajoutant les k arêtes reliant G aux composantes connexes de $G' = G \setminus \{s\}$ on a dans G au moins :

$$n_1 + n_2 + \dots + n_k - k + k = n_1 + \dots + n_k$$
$$= n - 1$$

arêtes

L'hérédité est prouvée

<u>Conclusion</u> : Application de l'axiome

de récurrence.

Preuve de Brigitte :

EXERCICE 3 Connexité

Montrez que tout graphe connexe à n sommets a au moins n-1 arêtes.

Par récurrence sur le nombre n de sommets, où n≥1.

. C'est évident si n=1, pas d'arête !

. Supposons qu'on a démontré qu'un graphe connexe à n-1 sommets possède au moins n-2 arêtes.

Si on a un graphe G connexe à n de sommets,

et que celui-ci possède un sommet de degré 1, alors si on supprime ce sommet (ainsi que l'unique arête qui le rattache), on obtient un graphe, toujours connexe, à n-1 sommets donc par hypothèse de récurrence, il possède au moins n-2 arêtes, et finalement, le graphe initial G a n sommets et possède au moins n-1 arêtes ;

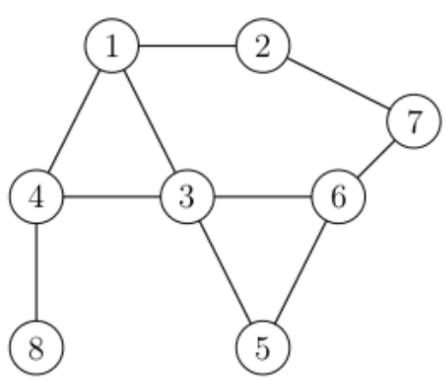
```
et que celui-ci ne possède pas de sommet de degré 1, tous les sommets de G sont au moins de degré2.
```

D'après l'exercice 2 : la somme des degrés des ~~arêtes~~ sommets est 2 fois le nombre d'arêtes, et ici chaque sommet est de degré ≥ 2, donc la somme des degrés est supérieure à 2n : 2a≥2n et a≥n≥n-1.

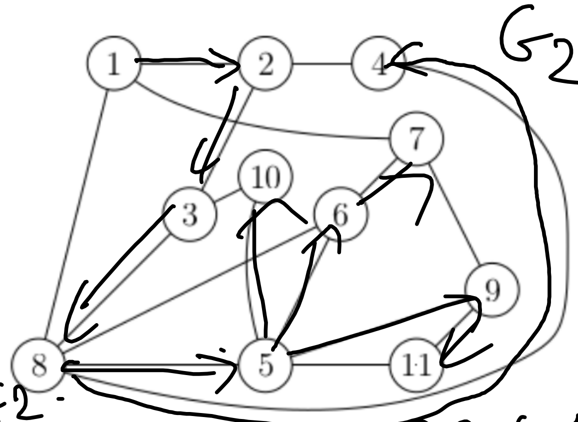
1.1.2 Parcours en largeur

EXERCICE #4 ▸ **Parcours en largeur**
Pour chacun des graphes suivants, donner l'ordre des noeuds rencontrés lors d'un parcours en largeur, en partant du sommet 1. Donner l'arbre résultant de ce parcours.

G_1



G_2



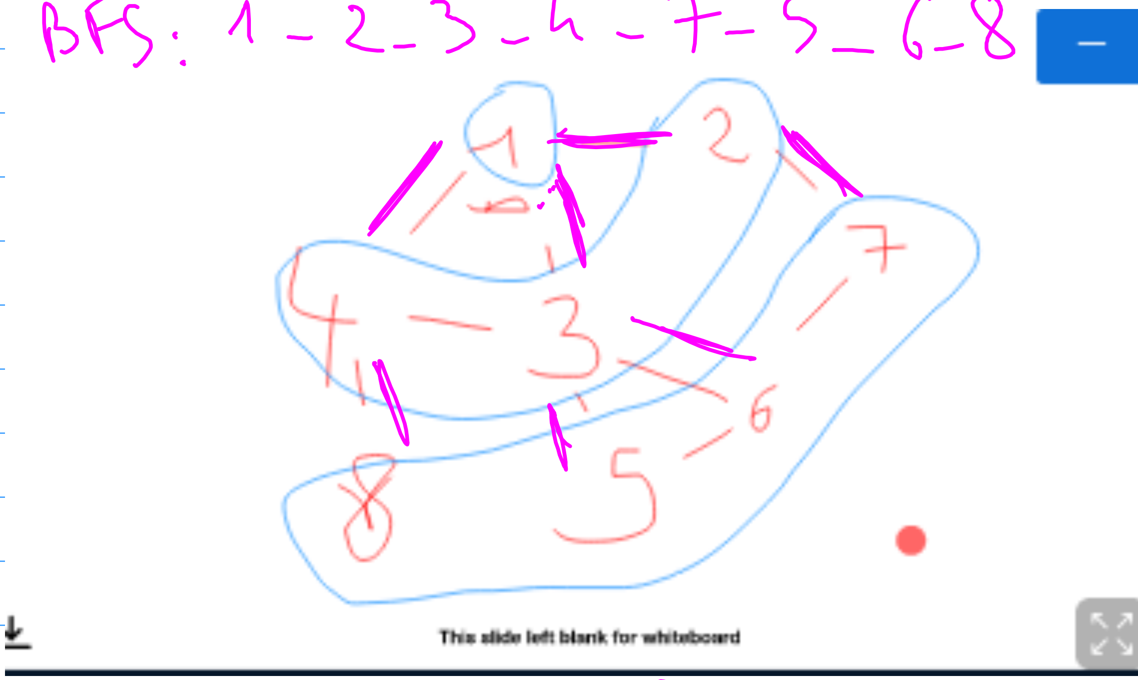
DFS G2 :
1, 2, 3, 8, 4, 5, 6, 7, 9, 11, 10

Quelle est la complexité du parcours en largeur avec les listes d'adjacence ?

→ Sinon on a des parcours en largeur ≠

Convention supplémentaire pour le parcours en largeur sur papier : parcourir les sommets d'un même niveau

BFS : 1 – 2 – 3 – 4 – 7 – 5 – 6 – 8



en violet l'arbre couvrant obtenu par BFS ; les arêtes étaient déjà présentes dans le graphe.

Exercice 4 :

Parcours en largeur de G_1 :

$1 - \underbrace{2 - 3 - 4}_{d=1} - \underbrace{7 - 6 - 5 - 8}_{d=2}$

Parcours en largeur de G_2 :

$1 - \underbrace{2 - 7 - 8}_{d=1} - \underbrace{4 - 3 - 9 - 6 - 5}_{d=2} - \underbrace{10 - 11}_{d=3} -$

La complexité du parcours en largeur avec les listes d'adjacence est :

$O(|V| + |E|)$

nombre de sommets

nombre d'arêtes

EXERCICE #5 ▸ **Profondeur**

Donner l'ordre des noeuds visités dans le parcours en profondeur des deux graphes précédents à partir du noeud 1.

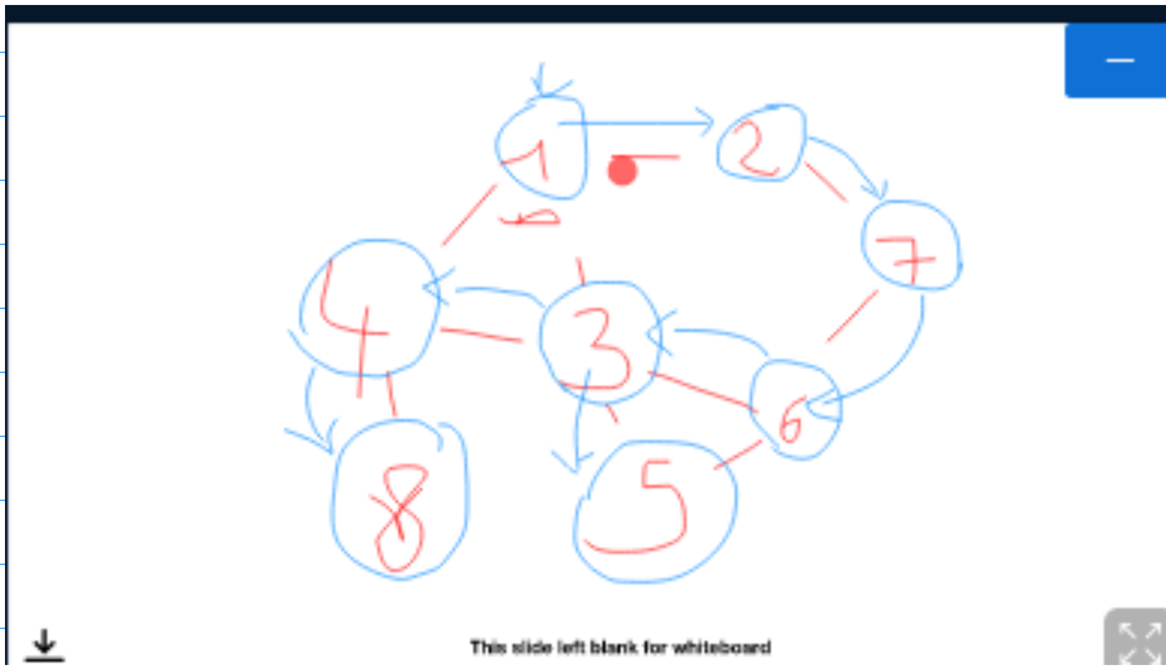
- Parcours en profondeur de G_1 à partir du sommet 1 :

$1 - 2 - 7 - 6 - 5 - 3 - 4 - 8$

- Parcours en profondeur de G_2 à partir du sommet 1

1 – 2 – 4 – 8 – 3 – 10 – 5 – 6 – 7 – 9 – 11.

Correction:



1 – 2 – 7 – 6 – 3 – 4 – 8 – 5

Propriétés du parcours en profondeur:

. Il produit un arbre couvrant comme le parcours en largeur.

Si on appelle T l'arbre couvrant produit (celui qui contient "l'histoire du parcours": on décrète que v a u comme père si l'arête (u, v) a permis de découvrir v), on a:

- Pour chaque appel récursif DFS(u), tous les sommets qui sont marqués "explorés" entre l'invocation de l'appel et son retour sont des descendants de u dans l'arbre T produit.
- Soit T un arbre DFS, et x et y deux sommets dans T, avec $(x, y) \in E$ qui n'est pas une arête de T. Alors soit x est un ancêtre de y, soit le contraire.

↳ permet de prouver qu'un sommet est l'ancêtre d'un autre.

- Pour le graphe G2 :

[1, 2, 3, 8, 4, 5, 6, 7, 9, 11, 10]

EXERCICE #6 ► **Applications des parcours**
Proposer un algorithme qui permet de déterminer si un graphe contient un cycle.

On verra que les parcours de graphes peuvent répondre à des buts très divers ; dans tous les cas, le principe d'un parcours est de *visiter* l'ensemble de la composante connexe. A chaque étape on peut donc distinguer états pour les sommets :

1. les sommets pas encore visités,

2. la frontière, c'est-à-dire les sommets déjà visités, dont certains voisins n'ont pas été visités,

3. les sommets déjà traités, c'est-à-dire qu'ils ont été visités, ainsi que tous leurs voisins, visités.

Initialement, tous les sommets sont à l'état (1), sauf le sommet de départ qui est à l'état (2). Une étape du parcours consiste alors à :

- choisir un sommet s de la frontière (2),

- si tous les voisins de s ont été visités, c'est-à-dire qu'aucun n'est à l'état (1), alors s passe à l'état (3),

- sinon on choisi un de des voisins de s non visités, qui passe de l'état (1) à l'état (2).

Le parcours se termine lorsque tous les voisins (de la composante connexe) sont à l'état (3). On a alors visité tous les sommets, dans un ordre qui dépend comment sont choisis les sommets visités à chaque étape.

Dans tous les cas, au cours d'un parcours, il faut se souvenir des sommets déjà parcourus. Très souvent, on munira les sommets d'une marque permettant d'indiquer leur état.

```
# 0 pour non vu
# 1 pour frontière
# 2 pour vu

# on suppose les sommets numérotés
# de 0 à n-1
```

Application 1: numérotation

ParcoursNumerotation(G,s):

- $num := 0$
- $numero[s] := num;\ num := num + 1.$
- $L := \{s\}$
- $B := Voisins(s)$
- Tant que $B \neq \emptyset$
  - choisir u dans B
  - $numero[s] := num;\ num := num + 1.$
  - $L := L \cup \{u\}.$
  - $B := B - \{u\}.$
  - $B := B \cup (Voisins(u) - L)$

Essentiel

Application 2: composantes connexes

UneComposante(G,s):

- $L := \{s\}$
- $B := Voisins(s)$
- Tant que $B \neq \emptyset$
  - choisir u dans B
  - $L := L \cup \{u\}$.
  - $B := B - \{u\}$.
  - $B := B \cup (Voisins(u) - L)$
- Retourner L

ComposantesConnexes(G)

- Tant que $V \neq \emptyset$
  - Choisir s dans V
  - $L := UneComposante(G, s)$
  - Afficher L
  - $V := V - L$

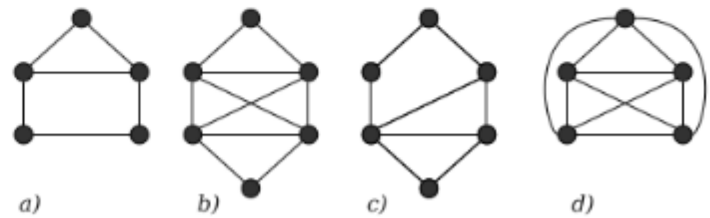
EXERCICE #6 ► **Applications des parcours**

Proposer un algorithme qui permet de déterminer si un graphe contient un cycle.

```
def detect_cycle(self):
        """Détection de cycle pour graphe orienté ou non orienté
         Au départ, tous les noeuds sont marqués comme n'ayant pas été visités.
         Lors du parcours en profondeur, on marque comme étant "en cours de visite"
         les noeuds pour lesquels un appel récursif est en cours
         et comme "déjà visités"
         les noeuds pour lesquels l'appel récursif s'est terminé.
         Si un appel récursif est fait sur un noeud qui est dans l'état "en cours de visite",
         cela signifie que l'ensemble des noeuds en cours de visite forment un cycle,
         donc qu'il y a un cycle dans le graphe. Si cela ne se produit jamais,
         c'est que le graphe ne contient pas de cycle.

        Lorsque le graphe contient un cycle, on est certain de retomber
        sur un noeud dans l'état "en cours de visite"
        au cours du parcours. En effet, lorsque l'on appelle la fonction récursive
        pour la première fois sur un noeud faisant partie d'un cycle, le parcours en profondeur
        va énumérer tous les noeuds de ce cycle, et retomber sur le premier noeud,
        avant que l'appel ne se termine.
        """

        gdict = self.__graph_dict
        TRAITE = 2
        EN_COURS = 1
        PAS_VU = 0
        marque = {vertex : PAS_VU for vertext in self.vertices()}

        def search_dfs(vertex):
            etat[vertex] = True
            rep = False
            for neighbour in gdict[vertex]:
                if marque[neighbour] == PAS_VU:
                    marque[neighbour] = EN_COURS
                    rep = rep or search_dfs(neighbour)
                elif marque[neighbour] == EN_COURS:
                    return True
            marque[neighbour] = TRAITE
            return rep

        for vertex.self.vertices():
            if marque[vertex] == PAS_TRAITE and search_dfs(vertex):
                return True
        return False
```

1.1.3 Cycles : graphes eulériens / hamiltoniens

EXERCICE #7 ▸ **Cycles Eulériens**
Un chemin est dit eulérien si il passe une et une seule fois par chacune des *arêtes* du graphe. Les graphes suivants possèdent-ils un cycle eulérien ?



Un graphe possède un cycle eulérien ssi tous les sommets sont de degré pair : et si le graphe est connexe.

a) → non

b) → oui

c) → non

d) → oui

EXERCICE #8 ▸ **CNS Chemin Eulérien**
Montrer qu'un graphe admet un chemin eulérien ssi il est connexe et au plus 2 de ses sommets sont de degré impair.

□ Soit un chemin eulérien d'origine un sommet A et d'extrémité un sommet B.

Si un sommet d'un graphe eulérien est de degré impair alors c'est forcément un sommet avec un passage qui comporte 1 entrée et pas de sortie ou (exclusif) une sortie et pas d'entrée.

Sinon, comme toutes les arêtes sont parcourues exactement une fois, on peut regrouper les arêtes incidentes à un sommet par paire (entrée, sortie) et on a forcément un sommet ~~de~~ degré pair.

Dans un chemin on a toujours une origine et une extrémité, tous les autres sommets sont des sommets de passage de degré pair d'après ce qui précède si le chemin est eulérien.

On en déduit que si un graphe est eulérien alors il est connexe (évident) et il a plus deux sommets de degré impair.

Réciproquement,

- si un graphe est connexe et a tous ses sommets de degré pair
  ↳ voir preuve de Wikipedia

Rappelons d'abord quelques définitions :

- le *degré* d'un sommet est le nombre d'arêtes incidentes au sommet ;
- un *parcours* est une suite d'arêtes telle que (i) pour chaque arête de la suite on peut distinguer une extrémité initiale et une terminale, (ii) l'extrémité terminale d'une arête est aussi l'extrémité initiale de l'arête qui lui succède dans la suite (et la première arête succède à la dernière) ;
- un *circuit* est un parcours non vide tel qu'aucun sommet n'est l'extrémité initiale (terminale) de plus d'une arête.

La condition suffisante du théorème d'Euler-Hierholzer s'appuie principalement sur les trois faits suivants :

1. Si tous les degrés sont pairs et non tous nuls, alors il existe un circuit ;
2. Un parcours est une union de circuits disjoints – au niveau des arêtes, et non des sommets ;
3. Si l'on retire les arêtes d'un parcours, alors les degrés pairs restent pairs.

Supposons maintenant que chaque sommet a un degré pair et qu'il n'existe pas de parcours contenant toutes les arêtes. Si l'on considère un parcours avec un nombre maximum d'arêtes et que l'on retire ensuite les arêtes du parcours du graphe, par (3), les degrés restent pairs. D'où, par (1), l'existence d'un circuit disjoint de notre parcours maximum. Mais, par (2), l'union de notre parcours et du circuit forme un autre parcours avec plus d'arêtes, ce qui contredit l'hypothèse de maximalité du parcours initial. Cette contradiction implique donc le théorème.

- <u>2ème cas</u> : On a exactement deux sommets A et B de degré impair.

Puisque le graphe est connexe, il existe un chemin simple reliant A à B.
On enlève toutes les arêtes de ce chemin du graphe.
Dans le sous-graphe obtenu, tous les sommets sont de degré pair et d'après le 1er cas, il existe un circuit eulérien disjoint de notre chemin

En faisant l'union de ce circuit eulérien et de notre chemin reliant A à B on obtient un chemin eulérien d'origine A et d'extrémité B (ou vice-versa).

<u>Rem</u> : Quand on parle d'un graphe avec au plus deux sommets de degré impair c'est soit 0 soit 2 car la somme des degrés de tous les sommets est forcément paire (2 fois la somme des arêtes).

EXERCICE #9 ► **Graphes Hamiltoniens**

Un cycle est dit Hamiltonien ssi il passe une et une seule fois par chaque sommet du graphe.

Les graphes suivants possèdent-ils un cycle hamiltonien ?

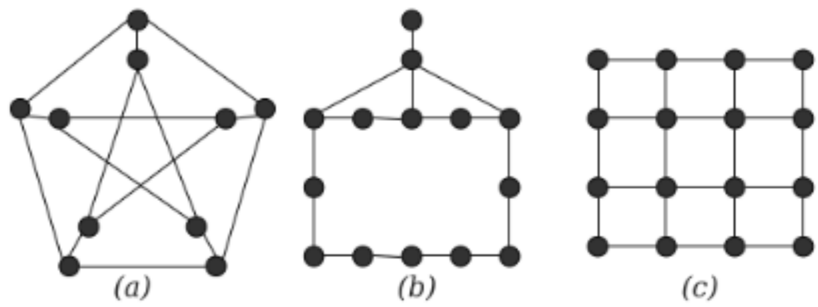
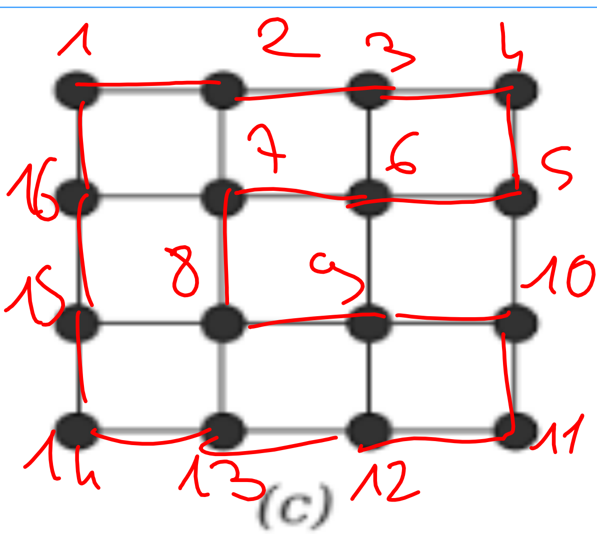
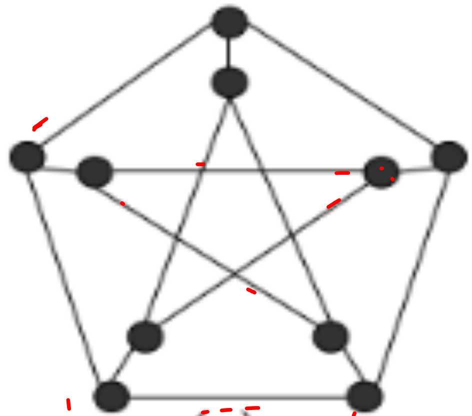


FIGURE 3 – (a) Graphe de Petersen 3-régulier, (b) maison agrandie, (c) grille

Donner un algorithme pour déterminer si un tel cycle existe.



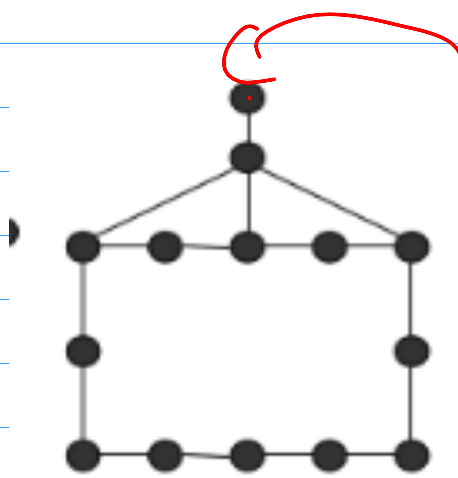
oui cycle hamiltonien



(a)

non il existe des chemins hamiltoniens pour le graphe de Petersen mais pas de cycle hamil-tonien
↳ voir wikipedia

graphe de petersen



(b)

non car on a un sommet de degré 1 il ne peut être ni un sommet intermédiaire du cycle hamiltonien ni l'origine/extrémité car sinon son voisin serait parcouru 2 fois

Dans un cycle hamiltonien, un sommet est de degré ≥ 2.

Pour déterminer si un cycle hamiltonien on peut :

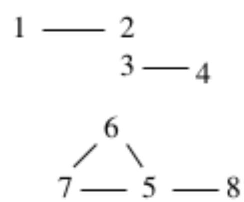
- force brute 1 : énumérer les 2^m ensembles d'arêtes et déterminer pour chacun s'il s'agit d'un cycle et s'il est hamiltonien (passage par les n sommets)

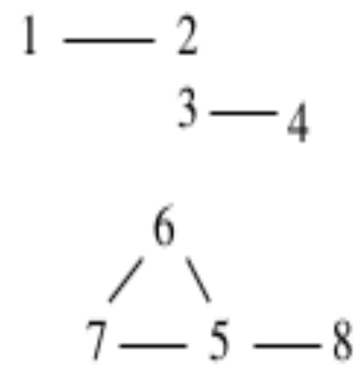
- force brute 2 : exploration à partir d'un sommet de tous les chemins couvrant tous les sommets du graphe en suivant pour chaque sommet toutes les arêtes partant vers des sommets pas encore explorés $\Rightarrow$ complexité en $O(n!)$.

1.1.4 Coloriages

EXERCICE #10 ▸ **Un exemple simple**

Avec l'algorithme polynomial vu en cours (simplification de Kempe), colorier le graphe suivant :





Tant qu'il y a des sommets,
on sélectionne le sommet de degré minimal
on élague

On obtient :

Code de couleur

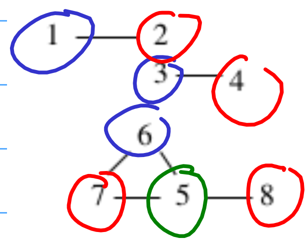
1	B
2	A
3	B
4	A
8	A
5	C
6	B
7	A

	A
	B
	C

On assigne les couleurs dans l'ordre inverse en choisissant de façon gloutonne la première couleur libre. Ici on peut colorier avec trois couleurs.

Let's color !

- We assign colors to the nodes greedily, in the reverse order in which nodes are removed from the graph.
- The color of the next node is the first color that is available, *i.e.* not used by any neighbour.



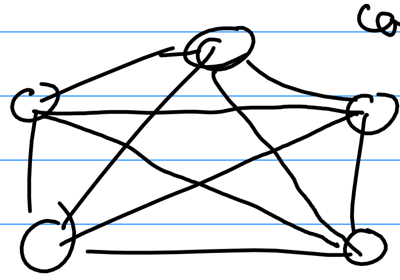
EXERCICE #11 ▸ **Coloriage et Bipartisme**

On dit qu'un graphe est biparti si on peut partitionner ses sommets en deux ensembles V_1 et V_2 de sorte qu'il n'y ait aucune arête entre deux sommets de V_1 (resp. de V_2). Les seules arêtes joignent donc un sommet de V_1 à un sommet de V_2.

1. Montrer qu'un graphe à n sommets est n-coloriable. Donner un graphe à 5 sommets qui n'est pas 4 coloriable.
2. Montrer qu'un graphe est deux coloriable ssi il est biparti.

1) Il est clair qu'un graphe à n sommets est n coloriable

2) Un graphe complet à 5 sommets n'est pas 4 coloriable car la couleur de chaque sommet doit être distincte des couleurs de ses 4 voisins.



2)

$\Rightarrow$ Supposons qu'un graphe est 2-coloriable

Soit V_1 l'ensemble des sommets blancs et soit V_2 l'ensemble des sommets noirs.

Si on considère une paire de sommets de V_1, ils ne peuvent être adjacents car ils sont de même couleur. De même pour une paire de sommets de V_2

Le graphe est donc biparti

$\Leftarrow$ On suppose que le graphe biparti.

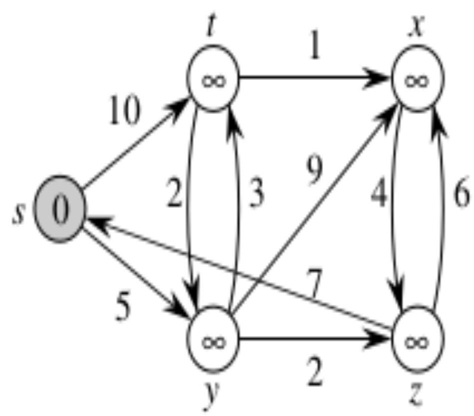
Si on colorie en blanc les sommets de V_1 et en noir ceux de V_2, alors on n'aura pas de sommets adjacents de la même couleur car les sommets de V_1 ne sont adjacents qu'à des sommets de V_2 et réciproquement.

Le graphe est donc biparti

1.1.5 Distances

<u>EXERCICE #12</u> ▸ **Dijkstra**

Appliquer l'algorithme au graphe suivant (source = s) :



s	t	y	x	z
$\boxed{(0,s)}$	(∞, None)	(∞, None)	(∞, None)	(∞, None)
$\boxed{(0,s)}$	(10, s)	$\boxed{(5,s)}$	(∞, None)	(∞, None)
	(8, y)	$\boxed{(5,s)}$	(14, y)	$\boxed{(7,y)}$
	$\boxed{(8,y)}$		(13, z)	$\boxed{(7,y)}$
	$\boxed{(8,y)}$		$\boxed{(9,t)}$	
			$\boxed{9,t}$	

EXERCICE #13 ▸ **Application : fiabilité des réseaux - notion de coupure - facultatif**

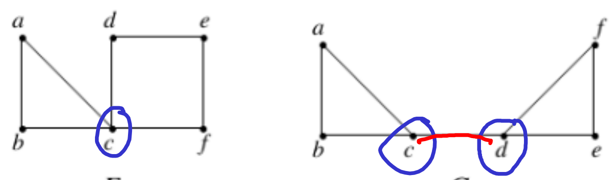
Si on considère un réseau informatique où tout le monde doit communiquer avec tout le monde, il est important que le graphe associé soit connexe. Maintenant, il faut aussi que le retrait d'une machine (ou d'un lien) soit sans douleur, d'où les définitions suivantes :

> Le retrait d'un sommet et de toutes les arêtes incidentes à ce sommet conduit à former un sous-graphe avec plus de composantes connexes que dans le graphe initial. Ces sommets sont appelés **points de coupure**. Le retrait d'un point coupure à partir d'un graphe connexe produit un sous-graphe qui n'est pas connexe. De façon similaire, une arête dont le retrait produit un graphe avec plus de composantes connexes que dans le graphe initial est appelée un **séparateur**.

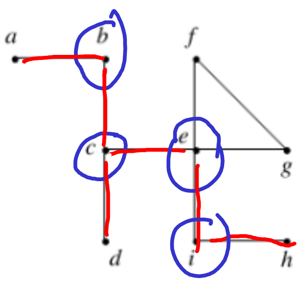
Dans les graphes suivants trouver les points de coupure et les séparateurs :

En bleu les points de coupure

En rouge les séparateurs



F G



H