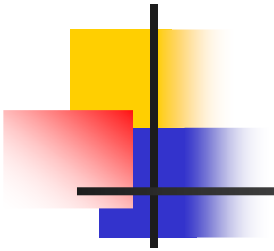


Dans cette partie...

3- Programmation

- Notion de programme
- Quelques instructions du langage Python

Qu'est-ce qu'un programme?

- C'est une suite d'instructions écrites dans un langage de programmation compréhensible par l'ordinateur.
- Cela permet à l'ordinateur d'appliquer un algorithme.
- Exemple : afficher les diviseurs de *n*

```
fonction affiche_diviseurs(n)
  si n > 0 alors
    pour tout entier i entre 1 et n faire
      si n est divisible par i alors
        afficher i
      finsi
    finpour
  finsi
```

```
def affiche_diviseurs(n):
  if n > 0:
    for i in range(1, n+1):
      if n % i == 0:
        print i
```

Remarque : `n % i` donne le reste de la division de *n* par *i*

L'affectation : ranger une valeur dans une variable

- Exemples :

```
i = 1
x = 2*i+1
i = i+1
x = x+i
```

- L'ordinateur effectue les instructions dans l'ordre. L'ordre des instructions est donc très important.
- Une variable désigne un emplacement dans lequel on peut mémoriser une valeur. Une variable a un nom.
- En python, le symbole = n'a pas la même signification qu'en mathématique. Il signifie calculer la valeur à droite du symbole =, et la ranger dans la variable dont le nom se trouve à gauche.

Instruction conditionnelle `if…: else :…`

Instruction conditionnelle si… alors… sinon…

```
i=10
x=6
if i > x :
   print "test VRAI"
   print i, "est supérieur à", x
else :
   print "test FAUX"
   print i, "n'est pas supérieur à", x
```

Attention à l'indentation !

Répétition : **while… :**

Répétition : tant que … faire…

```
i = 5
while i > 0 :
   print i
   i = i - 1
```

Ou :

```
i = 5
while i > 0 :
   i = i - 1
   print i
```

Attention à l'ordre des instructions !

Répétition : **for … in … range(….):**

Répétition : pour… parcourant la liste ….

```
# affiche les entiers de 0 à 9
for i in range(10) :
  print i
```

range(a,b,c): « donne la liste » des entiers appartenant à l'intervalle **[a,b[** par pas de **c** .

Exemple : **range(1,9,1) =>[1,2,3,4,5,6,7,8]**
range(1, 9, 2) => [1,3,5,7]

Remarque: **range(a,b) ⇔ range(a,b,1)**
range(b) ⇔ range(0,b,1)

Notion de fonction

En maths : soit la fonction $f : x \rightarrow 2x^2 + 1$

décrit la façon de calculer **f(x)** à partir de la donnée **x**

En Python :

```
def f(x) :
  return 2 * x * x + 1
# exemples d'appel de la fonction
y = 2 * f(2)
print f(5)
for k in range(10) :
  print f(k)
```

- Une fonction Python peut utiliser d'autres fonctions.
- Un programme est en général composé de plusieurs fonctions.

Quelques pièges à éviter !

Affichage ou sortie (**`print`** ou **`return`**)

Position du **`return`** (dans la boucle ou après la boucle)

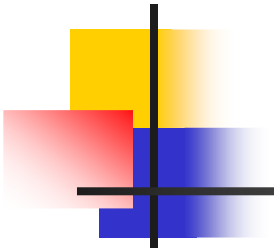
```
def f(x):
  for i in range(x):
    print i

def g(x):
  for i in range(x):
    return i

def h(x):
  for i in range(x):
  return i
```

Quel sera le résultat de :
`f`, **`g`** et **`h`** pour **`x=3`** ?

Il y a d'autres instructions dans le langage... **http://www.python.org/**



Dans cette partie...

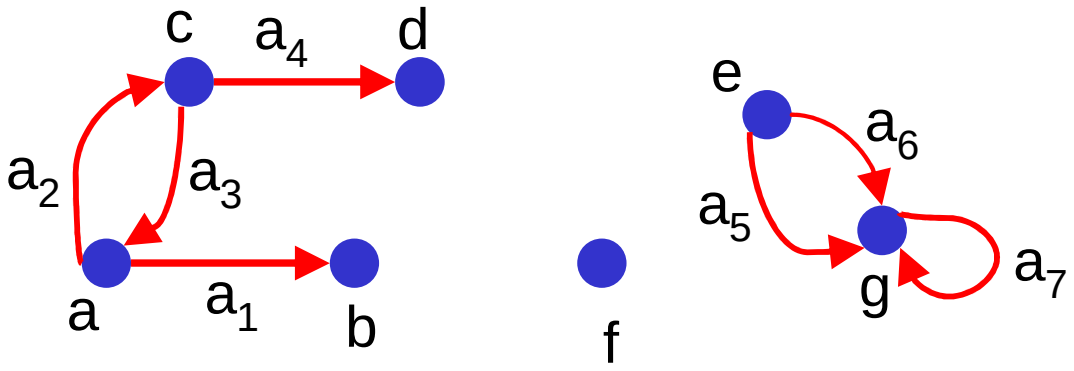
4- Introduction aux graphes

- 1ère sorte de graphes : les graphes orientés
- Quelques exemples de graphes orientés
- 2ème sorte de graphes : les graphes non orientés
- Quelques exemples de graphes non orientés
- Les graphes dans des problèmes courants

1ère sorte de graphes : les graphes orientés

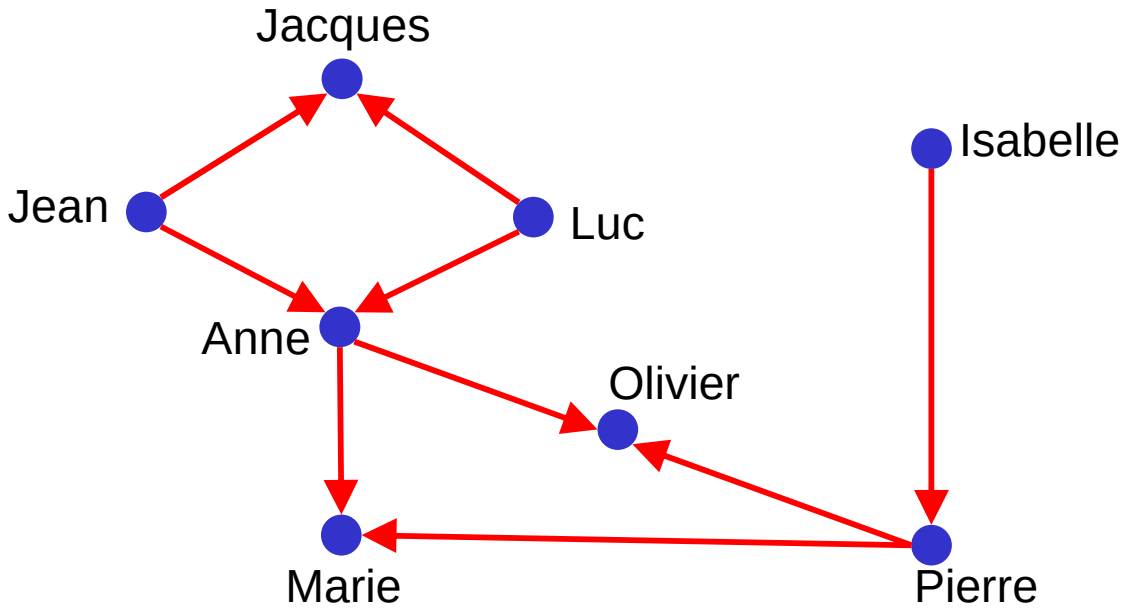
Un graphe orienté représente une relation non symétrique

- Il est défini par :
  - un ensemble fini de *sommets* représenté par un point
  - un ensemble fini d'*arcs* orientés
  - à chaque arc est associé est un *couple de sommets*.
  - un arc *a* associé à (s,t) est représenté par s ⟶ t
  - Exemple (graphe simple)
    - Sommets : {a,b,c,d,e,f,g}. Arcs : {a_1, a_2, a_3, a_4, a_5, a_6, a_7}.



Exemple de graphe orienté : parents

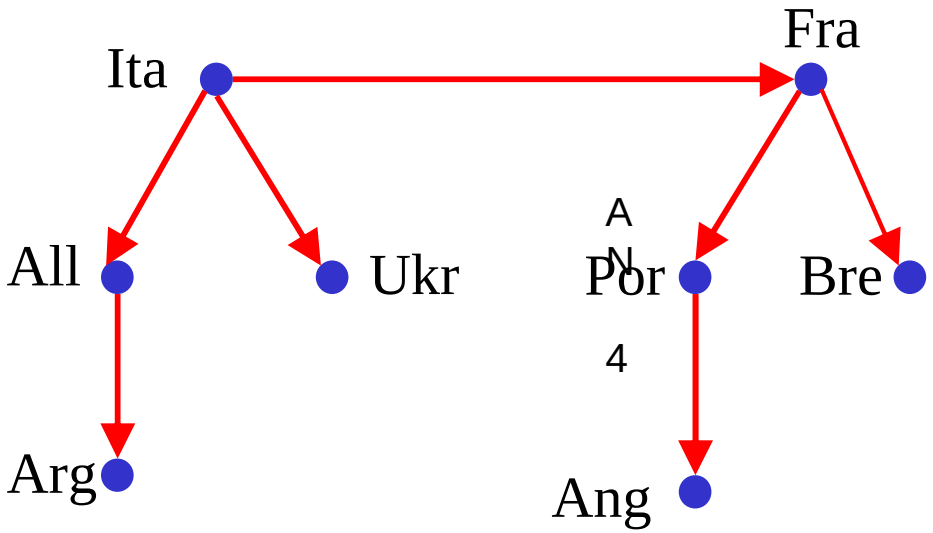
- Ensemble : toutes les personnes assistant à un repas de Noël.
- Relation : l'ensemble des couples de personnes (p1, p2) tels que p1 a pour parent p2.
- Représentation graphique (relation non symétrique, graphe orienté) :



- Qui sont les parents de Anne ?
- Même question pour Jacques.
- Ajouter une fille à Pierre

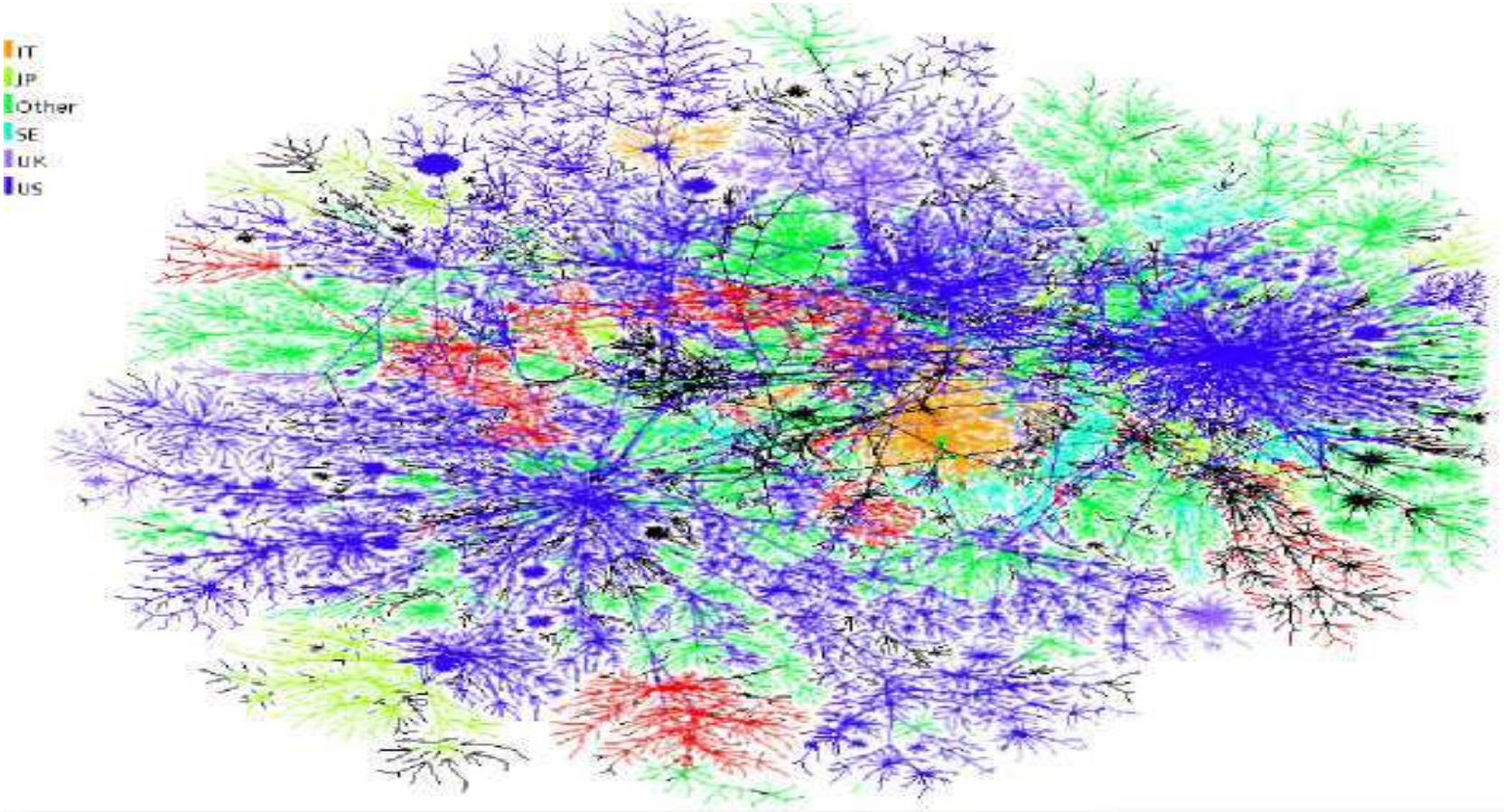
Exemple de graphe orienté : matchs de foot

- Ensemble : équipes de foot.
- Relation : l'ensemble des couples d'équipes *(eq1,eq2)* telles que *eq1* a battu *eq2* à partir des 1/4 finale de la coupe du monde 2006.
- Représentation graphique (pas de match nul, relation non symétrique, graphe orienté).



Changement d'échelle : Internet

- Dans la réalité les graphes sont de grande taille.
- On conserve une vue locale.

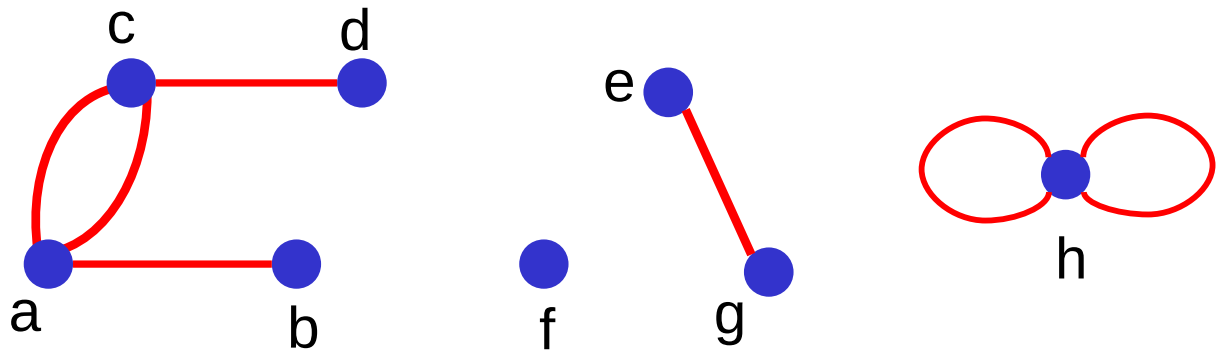


2ème sorte de graphes : graphes non orientés

- Un graphe non orienté représente une relation symétrique
- Dans un graphe non orienté, une arête est associée soit à une paire de sommets {s, t}, soit à un singleton {s}.
- On représente un graphe non orienté par un dessin, dans lequel les arêtes n'ont pas de flèche.
  - Un sommet est dessiné comme un point
  - Une arête a associée à {s, t} est représentée par 

  - Une arête a associée à {s} est représentée par 

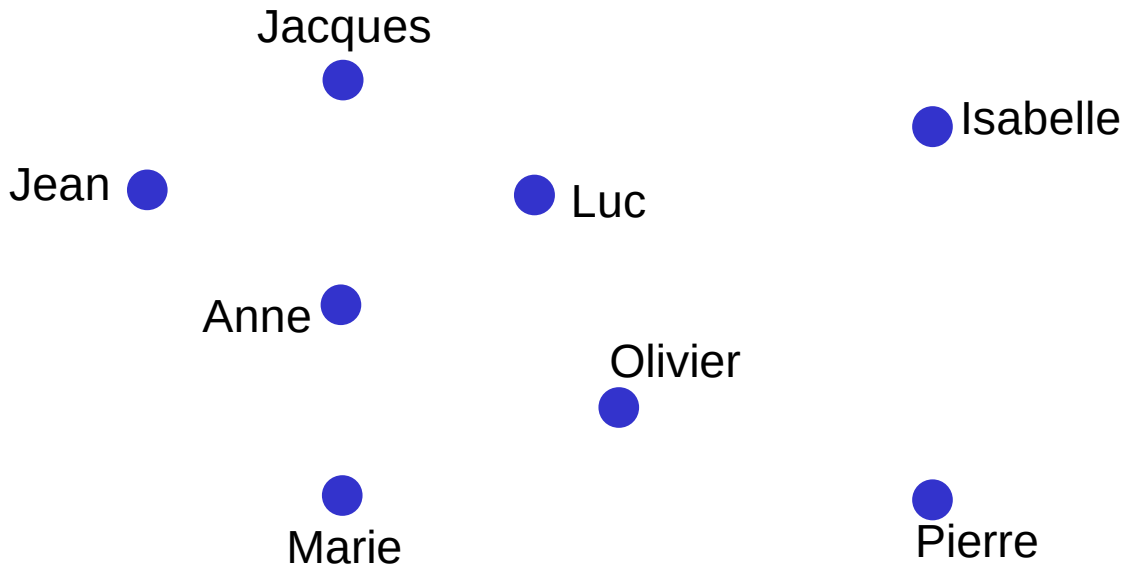
- Exemples



Exemple de graphe non orienté : cousins

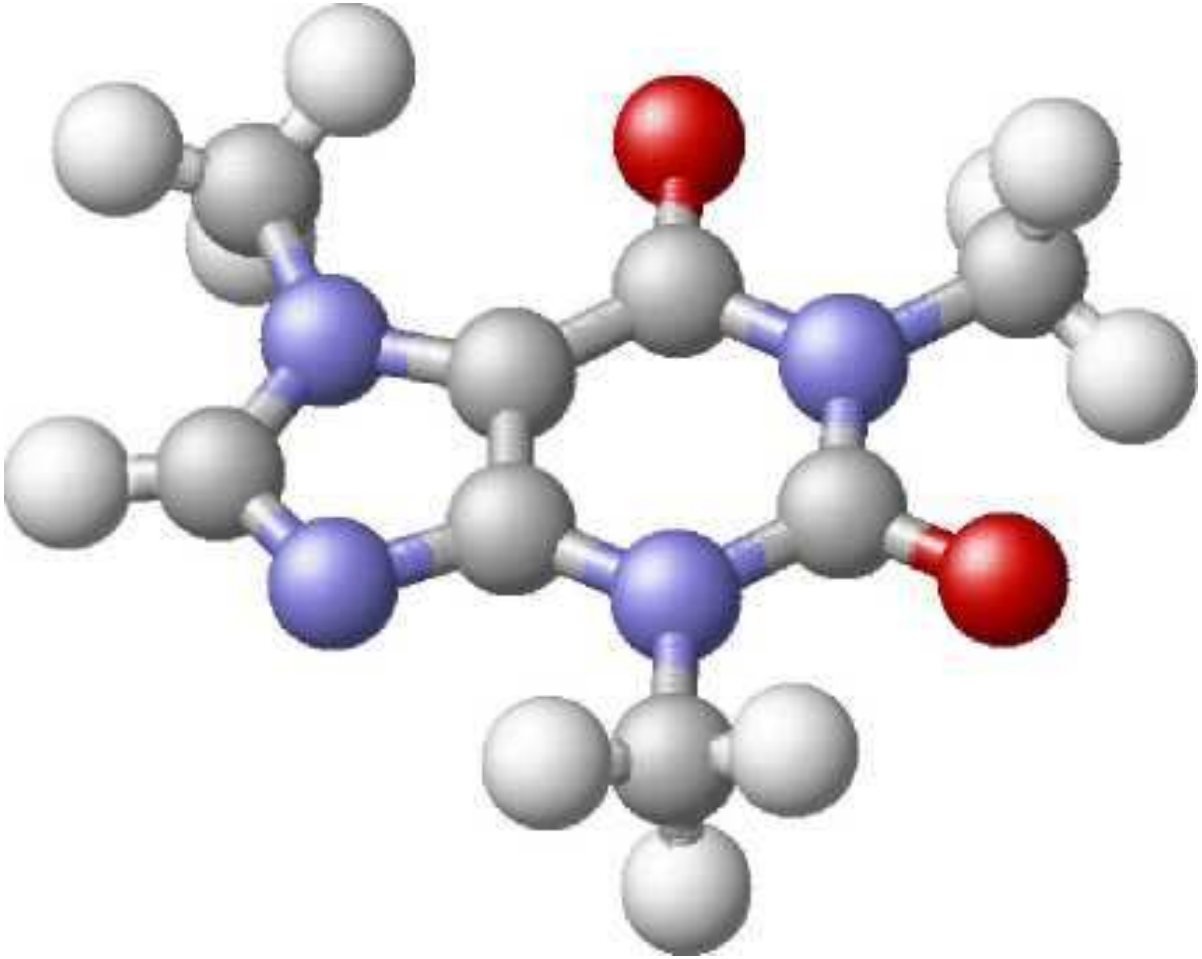
- Ensemble : toutes les personnes assistant à un repas de Noël.
- Relation : l'ensemble des couples de personnes (p1,p2) tels que p1 est un cousin de p2 (relation symétrique).
- Représentation graphique (relation symétrique, graphe non orienté)

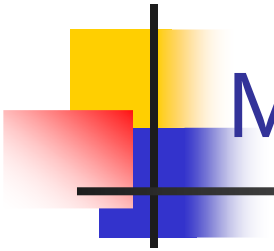
En utilisant le graphe des parents compléter le graphe des cousins.



Exemple de graphe non orienté : molécule

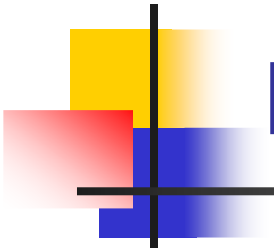
- Ensemble : les atomes de la molécule de caféine.
- Relation : l'ensemble des couples d'atomes partageant au moins un électron (liaison covalente, relation symétrique)





Modélisation de problèmes par des graphes

- Les graphes sont des objets permettant de représenter plusieurs situations de la vie courante.
  - Des problèmes de la vie courante se traduisent en questions à résoudre sur les graphes.
  - Le développement d'algorithmes sur les graphes permet donc de répondre à des problèmes concrets et pratiques.
  - Cela permet aussi de reléguer la résolution de ces problèmes à des systèmes informatisés.



Exemple : Organisation d'examens

- Des étudiants passent des examens de Maths (M), Informatique (I), Chimie (C), Physique (P), Anglais (A) et Sport (S) de 2h.
- Des étudiants passent M, I, C, d'autres C, P, A et d'autres A, S, P

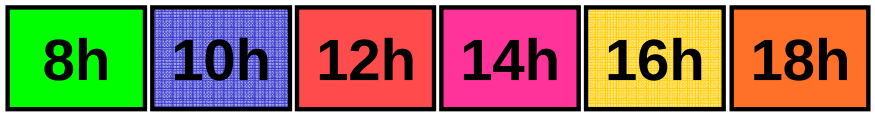
Question : Quel est le temps le plus court pour organiser les examens?

Exemple : Organisation d'examens

- On fait un graphe avec comme sommets M,I,C,P,S,A.
  - Si un étudiant doit passer deux des examens, les sommets correspondants sont incompatibles: les épreuves ne peuvent avoir lieu en même temps.
  - On forme le graphe en reliant 2 sommets s'ils sont incompatibles.

1. Dessiner le graphe.

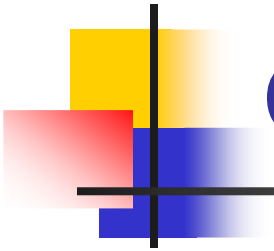
- On cherche à attribuer à chaque examen un créneau horaire.



  - On colorie des sommets adjacents avec des couleurs différentes.
  - On souhaite utiliser un nombre minimum de couleurs (de créneaux).

2.1 Colorier le graphe solution

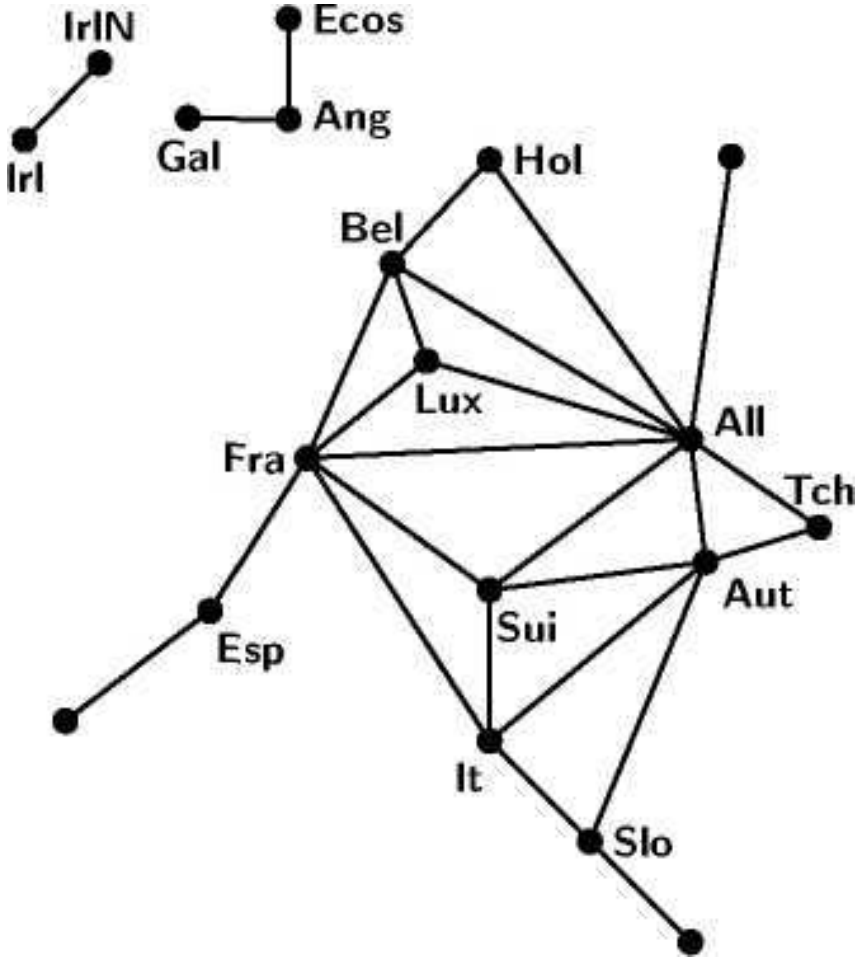
2.2 Quel est le temps le plus court pour organiser les examens ?

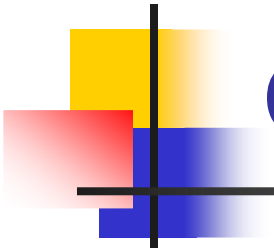


Coloration de carte géographique

- 1852 Guthrie colorie la carte des cantons anglais avec 4 couleurs sans que 2 cantons adjacents n'aient la même couleur.
- 4 couleurs suffisent-elles pour colorier n'importe quelle carte?
- 1976 : Oui (Appel & Haken). Résultat obtenu en partie par ordinateur.
  - Carte géograhique ~> graphe.
    Les sommets représentent les pays, et 2 sommets sont reliés par une arête si les pays correspondants partagent une frontière.
  - But : montrer qu'on peut colorier le graphe avec 4 couleurs, sans que 2 sommets adjacents n'aient la même couleur.
- On peut extraire de la preuve un algorithme permettant de trouver la coloration.

Coloration de carte géographique

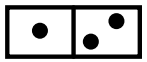


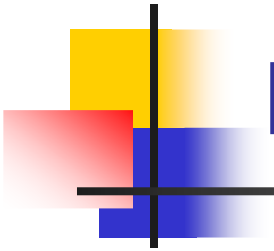


Coloration de graphes

- Partant d'un graphe quelconque (ne représentant pas une carte), on peut avoir besoin de plus de 4 couleurs.
- Tester si on peut colorier avec seulement 3 couleurs est difficile.
- Pour ce dernier problème, trouver un algorithme avec une complexité polynomiale, ou
- prouver qu'un tel algorithme n'existe pas est équivalent à l'un des problèmes ouverts les plus difficiles, sélectionné par le Clay Mathematical Institute (1 000 000$).

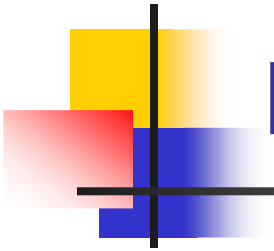
Dominos

- On dispose d'un jeu de dominos, le plus grand numéro étant *n*.
- Question : Peut-on placer tous les dominos (en suivant les règles)?
- Graphe. Sommets : {0, • • • , n}. Arêtes : toutes les paires {*s*, *t*}.
- L'arête {*s*, *t*} représente le domino dont les numéros sont *s* et *t*.
  - Exemple : l'arête {1,2} représente 
  - Dessiner le graphe pour n =4
- On peut placer tous les dominos s'il existe un chemin dans le graphe passant une seule fois par chaque arête : chemin Eulérien.
- Un résultat vu plus loin permet de résoudre le problème



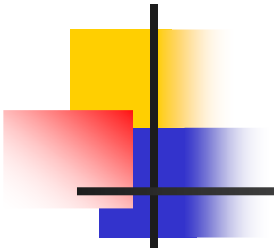
Problèmes de parcours

- Les graphes permettent de modéliser des voies de communication :
  - les sommets représentent des villes,
  - les arêtes représentent des routes.
- On peut associer à chaque arête d'un tel graphe (orienté s'il y a des sens uniques) un nombre représentant la distance entre les deux sommets reliés.



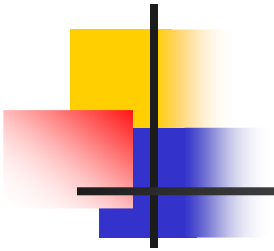
Problèmes de parcours

- Deux questions naturelles
  - Peut-on trouver un algorithme efficace pour calculer un trajet le plus court possible entre deux villes données?
  - Peut-on trouver un algorithme efficace pour calculer un trajet le plus court possible reliant toutes les villes?
- Réponses
  - Oui pour le 1er problème, (il y a des algorithmes efficaces).
  - ? ? pour le 2ème, sélectionné par le Clay Math Institute (1 M $).



Dans cette partie...

5- Graphes : définition



Une définition de graphe

Un *graphe* est un triplet (*S, A, inc*), où

- *S* est un ensemble d'objets appelés les *sommets* du graphe,
- *A* est un ensemble d'objets appelés les *arêtes* du graphe,
- *inc* est une fonction $inc : S \rightarrow P(A)$ telle que

$$\forall a \in A \ |\{s \in S \ , a \in inc(s)\}| \in \{1, 2\}$$

inc(s) = ensemble des arêtes incidentes au sommet *s*.

(incidentes = qui "touchent").

Interprétation de la définition

Exemple : pour décrire les vols d'une compagnie aérienne,

- on décrit chaque ville (= sommet) desservie,
- on décrit les lignes (= arêtes) vers ou depuis chaque ville.

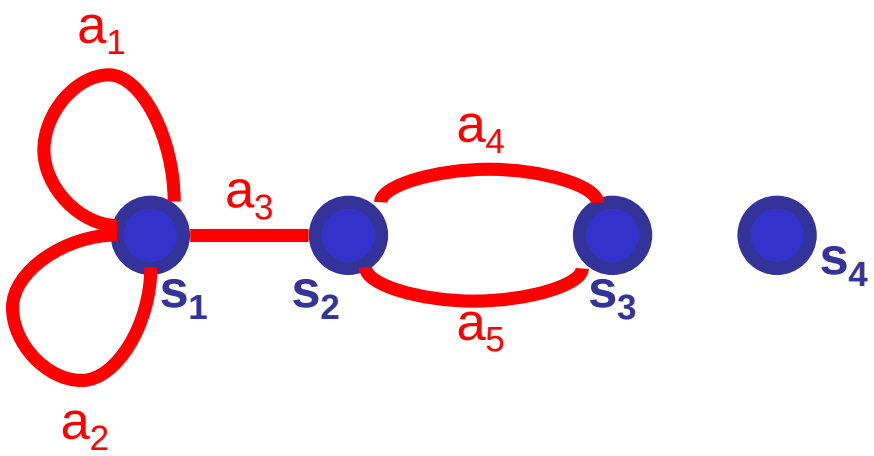
Par exemple : Ville Bordeaux, liaison AF6257 (vers Paris).

La fonction *inc* est appliquée à un sommet et renvoie l'ensemble des arêtes qui "touchent" le sommet.

La propriété de $inc : S \rightarrow P(A)$ pour forcer une arête à avoir une ou deux extrémités est :

$$\forall a \in A \ |\{s \in S \ , a \in inc(s)\}| \in \{1, 2\}$$

Exemple



$S = \{s_1, s_2, s_3, s_4\}$

$A = \{a_1, a_2, a_3, a_4, a_5\}$

$inc(s_1) = \{a_1, a_2, a_3\}$

$inc(s_2) = \{a_3, a_4, a_5\}$

$inc(s_3) = \{a_4, a_5\}$

$inc(s_4) = \varnothing$

$a_3 \in inc(s_1)$ et $a_3 \in inc(s_2)$ donc s_1 et s_2 sont les extrémités de l'arête a_3

Exercices

Dessiner le graphe défini par :

$S = \{s_0, s_1, s_2, s_3, s_4, s_5\}$

$A = \{a_0, a_1, a_2, a_3, a_4, a_5, a_6\}$

$inc(s_0) = \{a_0, a_1, a_5\}$

$inc(s_1) = \{a_0, a_1, a_2\}$

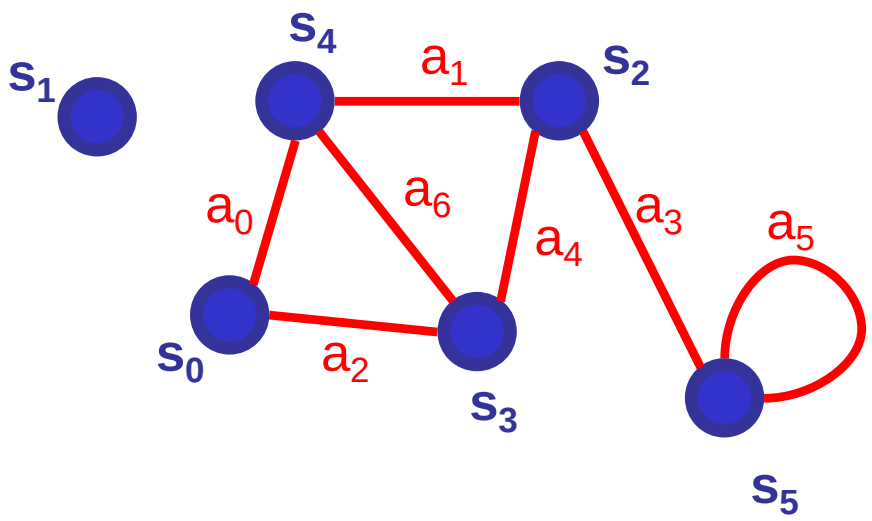
$inc(s_2) = \emptyset$

$inc(s_3) = \{a_3, a_6\}$

$inc(s_4) = \{a_2, a_3, a_4\}$

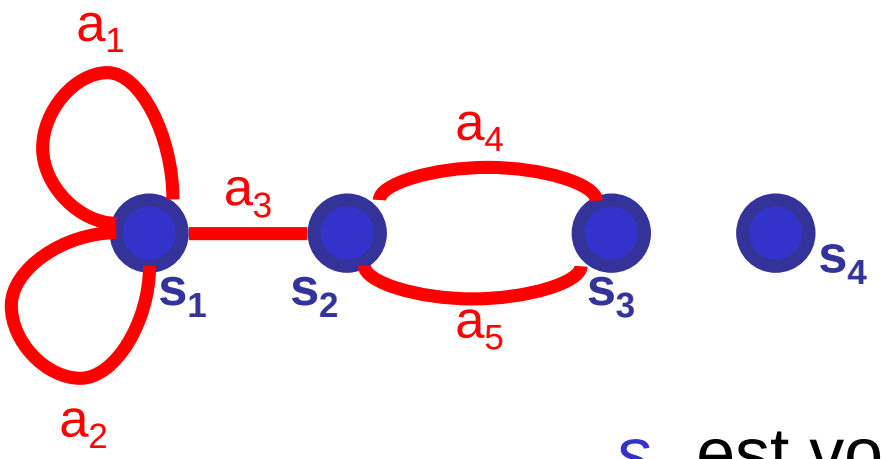
$inc(s_5) = \{a_4, a_5\}$

Déterminer les ensembles *A* et *S* ainsi que la fonction *inc* pour le graphe ci-dessous :



Voisins d'un sommet

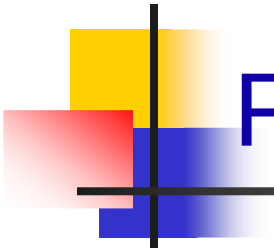
Deux sommets sont voisins si ce sont les extrémités d'une même arête.



s_3 est voisin de s_2

L'ensemble des voisins de s_1 est $\{s_1, s_2\}$

s_4 n'a pas de voisins (il est isolé)



Fonctions de manipulation de graphes

Plusieurs fonctions prédéfinies permettent de manipuler des graphes :

nomGraphe(G) : retourne le nom du graphe G.

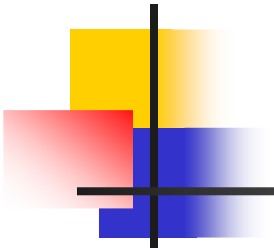
nbSommets(G) : retourne le nombre de sommets du graphe G.

listeSommets(G) : retourne la liste des sommets du graphe G.

degre(s) : retourne le degré du sommet s.

listeVoisins(s) : retourne la liste des voisins du sommet s.

dessiner(G) : dessine le graphe G.

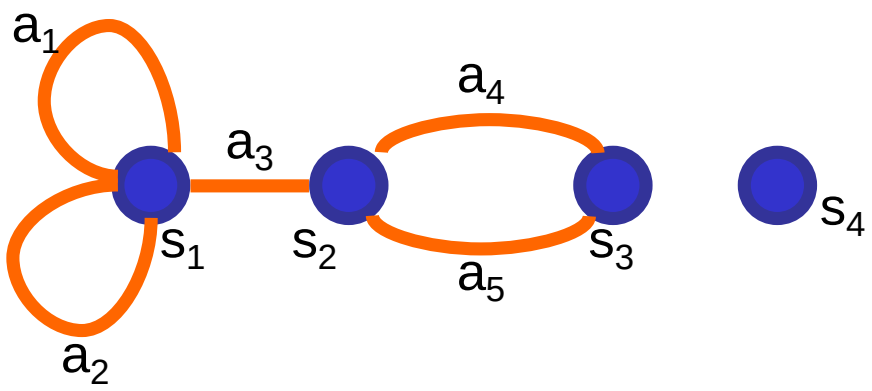


Dans cette partie...

6- Degré d’un sommet

Degré d'un sommet

- Le degré d'un sommet s, noté $d(s)$, est le nombre de brins d'arêtes ayant s comme extrémité.
- Une boucle compte deux fois.



Ici $d(s_1) = 5$, $d(s_2) = 3$, $d(s_3) = 2$, $d(s_4) = 0$.

Attention ! Le degré de s n'est pas le nombre de voisins de s.

Un théorème important

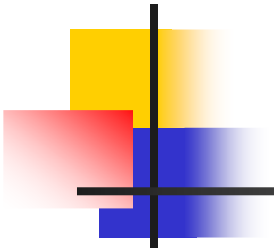
- Si G est un graphe (non orienté), on note
  - $S(G)$ l'ensemble de ses sommets.
  - $A(G)$ l'ensemble de ses arêtes.
- Théorème.

  Pour un graphe G ayant au moins un sommet, la somme des degrés des sommets est égale à deux fois le nombre d'arêtes. D'où la formule :

$$\sum_{s \in S(G)} d(s) = 2\left|A(G)\right|$$

- Conséquence.

  Dans n'importe quel graphe, le nombre de sommets qui ont un degré impair est pair.



Preuves

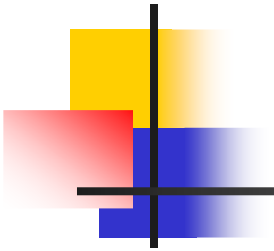
- Preuve directe

Une idée de preuve : faire la somme des degrés compte chaque arête 2 fois, car chaque arête a 2 extrémités qui contribuent chacune à une unité de cette somme.

- Preuve par induction (récurrence)

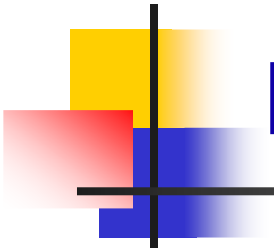
Vérifier que $\Sigma_{s \in S(G)} d(s) = 2|A(G)|$ pour un graphe sans arête, puis prouver que :

si $\Sigma_{s \in S(G)} d(s) = 2|A(G)|$ est vrai pour tout graphe ayant au plus k arêtes, alors c'est aussi vrai pour tout graphe ayant $k + 1$ arêtes.



Dans cette partie...

7- Chaînes dans un graphe



La notion de chaîne

- Il est souvent nécessaire de savoir si l'on peut aller d'un sommet à un autre en suivant des arêtes.
- Exemple d'utilité : Est-ce possible de prendre le train pour aller
  - De Bordeaux à Rome?
  - De Bordeaux à Oslo?
  - De Bordeaux à Reykjavik?
- La notion de chaîne exprime cette idée.

Définition de chaîne

- Une *chaîne* dans un graphe est une suite

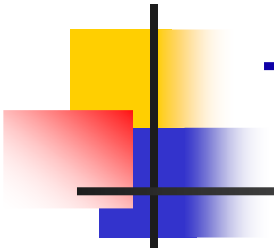
$$C = s_1, a_1, s_2, \ldots, s_k, a_k, s_{k+1}$$

de $k + 1$ sommets et k arêtes en alternance, telle que les extrémités de a_i sont s_i et s_{i+1}.

- On dit alors que C est une *chaîne entre* s_1 et s_{k+1}
- Remarques
  - Si $k = 0$, on obtient une chaîne sans arête $C = s_1$.
  - Pour définir une chaîne, se donner seulement la suite des arêtes, ou seulement la suite des sommets ne suffit pas (pourquoi ?).

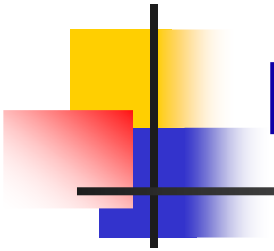
Chaîne simple

- Une chaîne $C = s_1, a_1, s_2, \ldots, s_k, a_k, s_{k+1}$ est *simple* si et seulement si:
  - Quels que soient i et j, alors ($i < j$ et $s_i = s_j$) *implique* ($i = 1$ et $j = k+1$) [où i et j varient entre 1 et $k+1$].
- Autrement dit "un sommet figure au plus une fois dans la chaîne" [sauf pour le sommet de début et de fin. S'ils sont les mêmes, il s'agit d'un *cycle*].



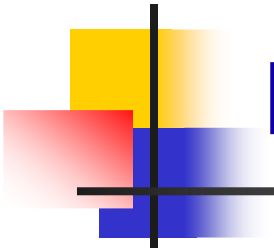
Théorème

- Dans un graphe *G*, s'il existe une chaîne entre deux sommets *s* et *t*, alors il existe une chaîne *simple* entre *s* et *t*.
- La preuve est *constructive*. On prend une chaîne non simple et on en supprime les cycles pour *construire* une chaîne simple.



Existence d'une chaîne

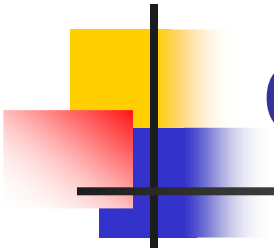
- Vérifier s'il existe une chaîne entre un sommet s et un sommet *t* n'est pas forcément simple.
- Pour y arriver, nous allons utiliser une technique pour marquer et démarquer les sommets.
  - Les marques laissées sur les sommets agissent en quelque sorte comme une mémoire d’un parcours effectué dans le graphe.



Existence d'une chaîne entre *s* et *t*

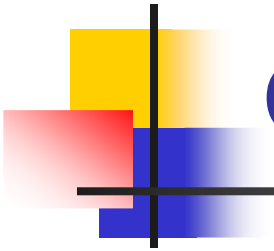
Algorithme :

- 1. démarquer tous les sommets
- 2. marquer s
- 3. tant que *t* n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "**faux**"
  - 3.3 sinon marquer l'extrémité non encore marquée
- 4. renvoyer la valeur "vrai"



Quelques remarques sur l'algorithme

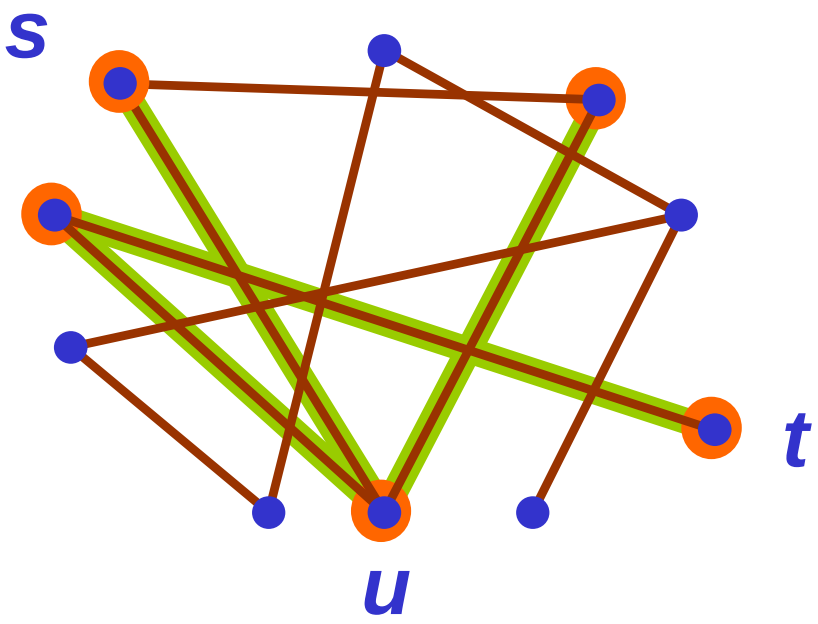
- La première phase consiste à se mettre dans un état de départ adéquat afin d'assurer le bon fonctionnement de l'algorithme.
- L'action « renvoyer » correspond à la réponse attendu et donné par l'algorithme, l'action « renvoyer » implique que l'algorithme est arrivé à sa conclusion et s'arête.
- On suppose ici que la recherche d'une arête qui satisfait la condition se fait en suivant une procédure connue.
- On suppose aussi que l'on a une façon facile de vérifier si un sommet est marqué et de le marquer ou de le démarquer le cas échéant.



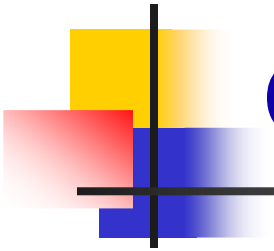
Quelques remarques sur l'algorithme

- Il reste encore à prouver que l'algorithme fait bien le travail que l'on attend de lui. C'est-à-dire:
  - Que l'on obtient éventuellement et à chaque fois une réponse de sa part,
  - Et que cette réponse est bien la bonne (!).

Existence d'une chaîne entre s et *t*

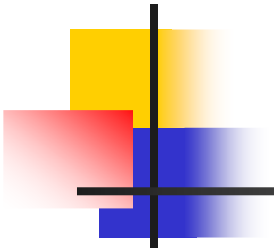


L'algorithme visite d'abord le voisin *u* de *s* qui se trouve au bas du graphe puis remonte vers le haut à droite. Il sélectionne ensuite voisin de *u* à gauche et en bas de *s* avant de « trouver » *t*.



Complexité de l'algorithme

- Étape 1 :
- Étape 2 :
- Étape 3.1 :
- Étapes 3.2 et 3.3 :

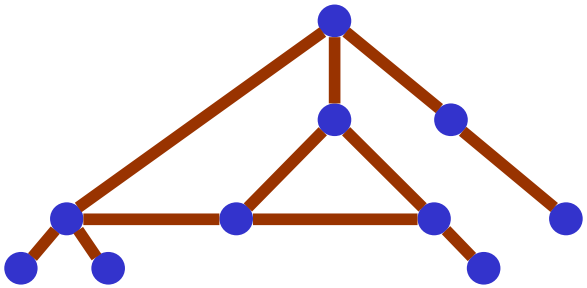


Dans cette partie...

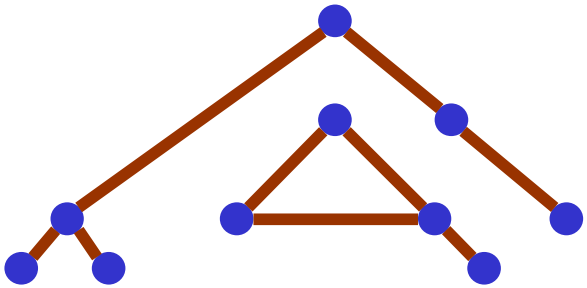
8 - Connexité

Connexité

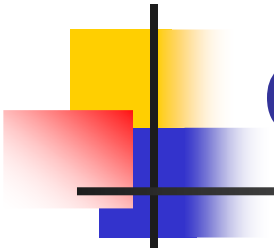
- La connexité exprime la possibilité d'aller de n'importe quel sommet du graphe à n'importe quel autre sommet du graphe.
- Informellement, un graphe est connexe s'il est en un seul morceau.



Connexe



Non connexe



Connexité

- La connexité exprime la possibilité d'aller de n'importe quel sommet du graphe à n'importe quel autre sommet du graphe.
  - Formellement, un graphe G est connexe si et seulement si

    $\forall s,t \in S(G)$, il existe une chaîne entre s et t.
- On met au point des algorithmes pour déterminer si un graphe est connexe.

Déterminer si un graphe est connexe

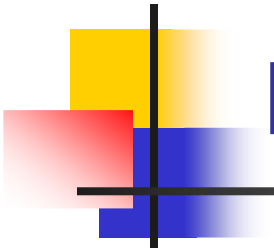
- Approche simple: la première idée est d'appliquer la définition.
  - Donc, pour déterminer si un graphe est connexe, on vérifie si pour chaque couple *(s, t)* de sommets, il existe une chaîne entre *s* et *t*.
  - Le nombre de couple *(s,t)* est $|S|^2$.
  - Au total, cette méthode a une complexité de l'ordre de

$$|S|^2 \cdot |S|(|A|+|S|) = |S|^3(|A|+|S|)$$

- Nous allons montrer qu'on peut résoudre le problème en remplaçant le facteur $|S|^3$ par $|S|^2$

Différence entre $O(n^3)$, $O(n^2)$ et $O(n)$

- Pour apprécier la différence entre $O(n^3)$ et $O(n)$, imaginons un ordinateur capable d'exécuter une instruction élémentaire en 10ns = 10^{-8} sec (ce qui est raisonnable).
  - Imaginons aussi un graphe de 1 000 000 sommets (un réseau routier par exemple).
- Exécuter n instructions élémentaires nécessite 10ms, soit 0,01sec.
- Exécuter n^2 instructions élémentaires nécessite 10^4 sec soit ~ 2h45.
- Exécuter n^3 instructions élémentaires nécessite 10^{10} sec soit ~ 321 ans.



Déterminer si un graphe est connexe

Theorème. Soit *G* un graphe. On considère les propriétés suivantes :

A. *G* est connexe.

B. Pour tout sommet *s* de *G*, il existe une chaîne entre *s* et chacun des sommets de *G*.

C. Il existe un sommet *s* de *G* tel qu'il existe une chaîne entre *s* et chacun des sommets de *G*.

Les propriétés *A*, *B* et C sont équivalentes, c'est-à-dire :

$$A \Leftrightarrow B \Leftrightarrow C$$

Preuve (parties *A => B et B => C)*

- $A \Rightarrow B$. On suppose que A est vraie, c'est-à-dire que G est connexe.

- $B \Rightarrow C$ (évident ?)

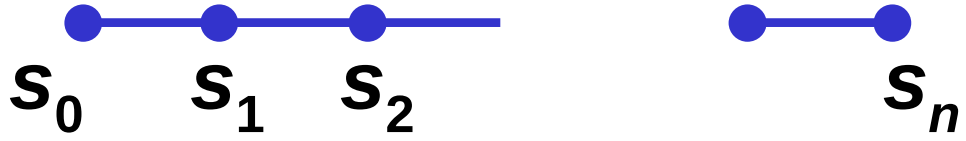
- Reste à montrer que $C \Rightarrow A$

Conséquence: un algorithme plus efficace

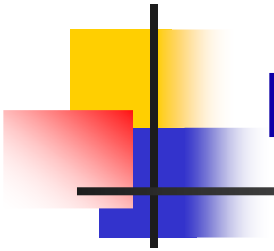
- Pour déterminer si un graphe est connexe, on peut donc :
  - Choisir un sommet *s* arbitraire dans le graphe.
  - Vérifier si pour chaque sommet *t* du graphe, il existe une chaîne entre *s* et *t*
- Temps de calcul
  1. Il faut donc faire $|S|$ vérifications.
  2. Le coût de chaque vérification est $|S| \cdot (|A| + |S|)$
  3. Au total, la complexité est donc $|S|^2 \cdot (|A| + |S|)$

Encore une amélioration

- On observe que l'algorithme parcourt la même chaîne plusieurs fois. Dans le graphe suivant :



  - (On choisit s_0 comme sommet de départ)
  - Il va parcourir la chaîne entre s_0 et s_1,
  - puis entre s_0 et s_2, refaisant alors le parcours entre s_0 et s_1,
  - puis entre s_0 et s_3, refaisant alors le parcours entre s_0 et s_2, etc.



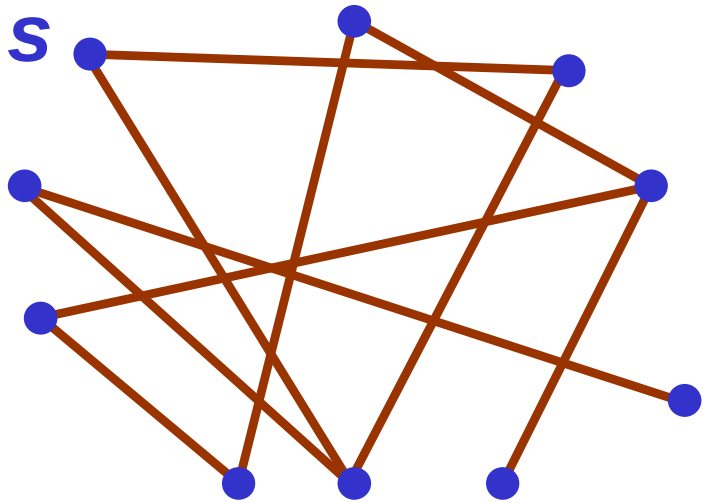
Méthode finale pour tester la connexité

- Voici un algorithme encore plus efficace.
  1. démarquer tous les sommets
  2. marquer un sommet arbitraire *s*
  3. tant qu'il existe une arête « bicolore » :
     3.1 marquer l'extrémité non encore marquée
  4. si tous les sommets sont marqués, renvoyer "vrai"
  5. sinon renvoyer "faux"

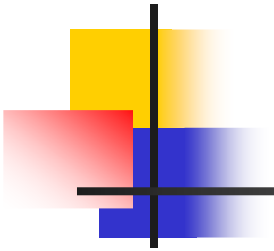
- Complexité $|S| \cdot (|S| + |A|)$ car on applique une seule fois l'algorithme de « parcours » du graphe depuis *s*.

Méthode finale pour tester la connexité

- L'algorithme ressemble à celui de recherche d'une chaîne
  - Recherche d'arêtes « bicolores »
  - Marquage des sommets



Utiliser l'algorithme précédent pour vérifier si le graphe ci-contre est connexe ?

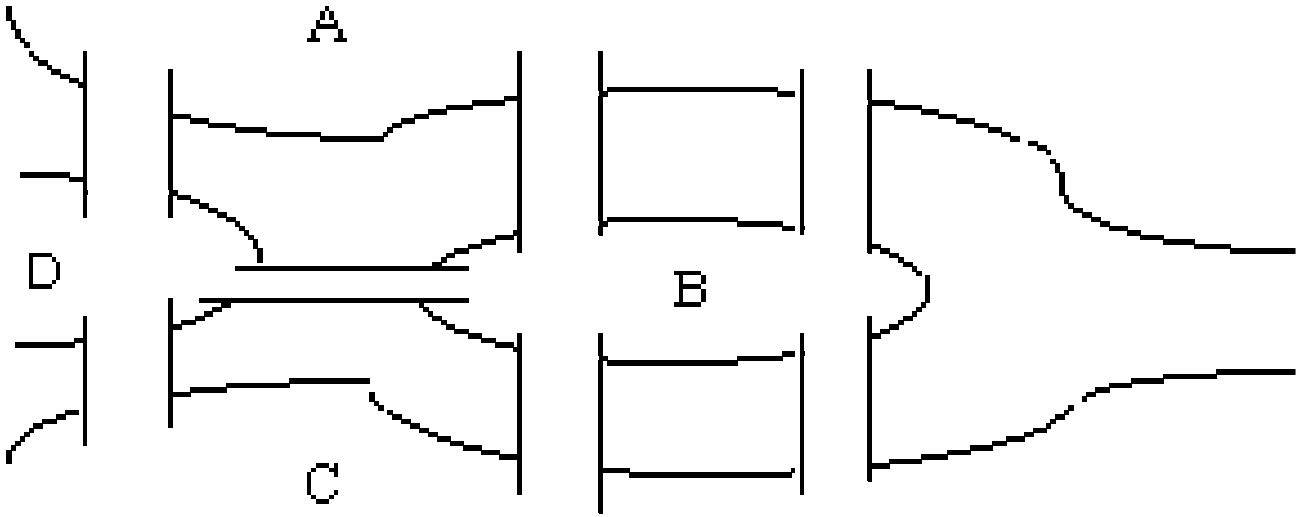


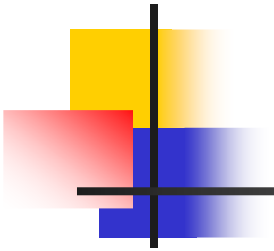
Dans cette partie...

9- Graphes Eulériens

Les ponts de Königsberg

La ville de Königsberg (maintenant Kaliningrad en Russie) est traversée par la rivière Pregolya, et comporte deux îles. Ces îles sont reliées entre elles et aux berges par des ponts, comme sur la figure ci-dessous.





Promenade

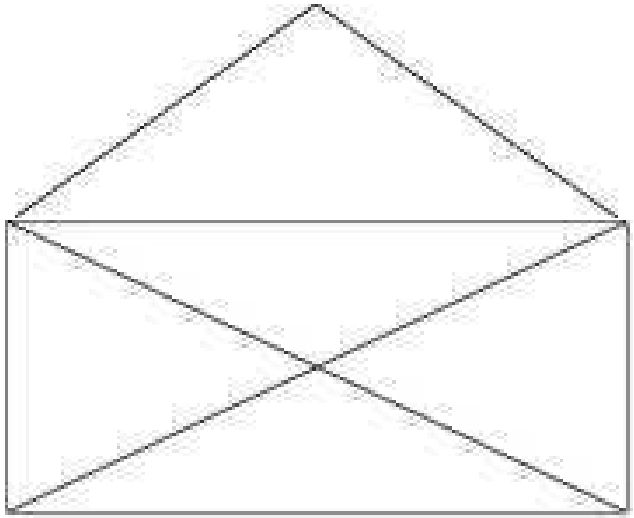
Peut-on

- commencer une promenade sur une île ou une rive,
- terminer la promenade sur n'importe quelle autre (ou la même) île ou rive
- en passant exactement une fois sur chacun des ponts ?

C'est le mathématicien Euler qui en 1735 a trouvé la réponse (d'où le nom "eulérien").

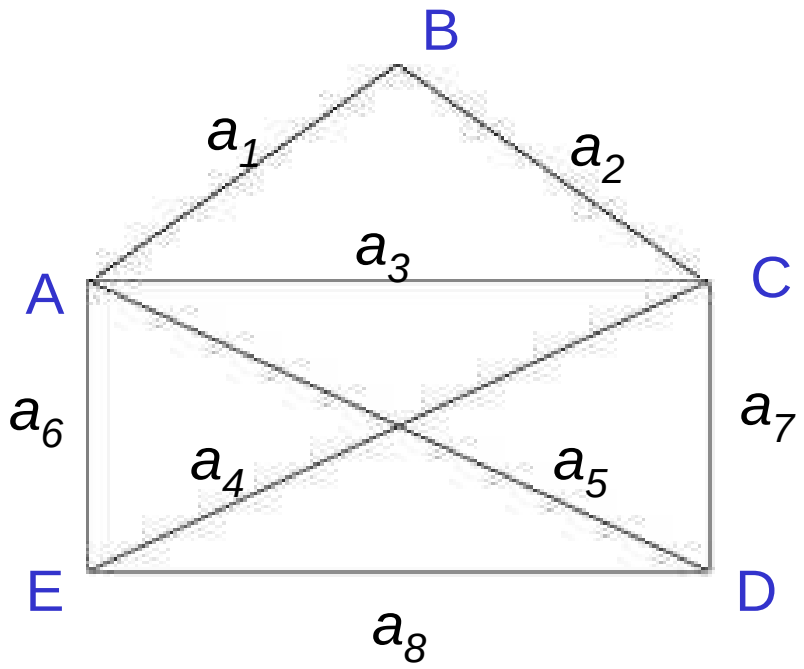
Dessiner une enveloppe

Est-ce possible de dessiner cette enveloppe sans lever le crayon **ET** sans passer deux fois sur le même trait ?



C'est un problème de graphe

Dans un graphe *G*, est-il possible de trouver une chaîne *C* qui contient tous les sommets de *G* et une et une seule fois toutes les arêtes de *G* ?



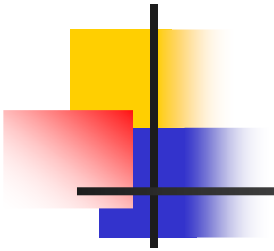
Donner une solution pour le graphe ci-contre

Définition de graphe eulérien

Un graphe G est *eulérien* si et seulement si il est possible de trouver une chaîne $C = s_1, a_1, s_2, a_2, \ldots, s_m, a_m, s_{m+1}$
telle que :
$S(G) = \{s_1, s_2, \ldots, s_{m+1}\}$, $A(G) = \{a_1, a_2, \ldots, a_m\}$ et $|A(G)| = m$.
Une telle chaîne est appelée *chaîne eulérienne*.
Si $s_1 = s_{m+1}$ c'est un *cycle eulérien*.

Remarque 1 : Dans la suite de sommets $s_1, s_2, \ldots, s_{m+1}$ certains sommets peuvent apparaître plusieurs fois.

Remarque 2 : Par contre, dans la suite d'arêtes $a_1, a_2, \ldots, a_m$, chaque arête apparaît exactement une fois.



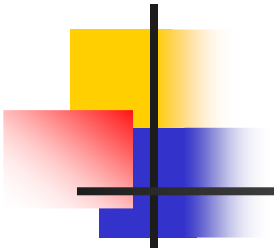
Conditions nécessaires

Théorème :

Un graphe eulérien est nécessairement connexe.

D'autres façons de dire la même chose :

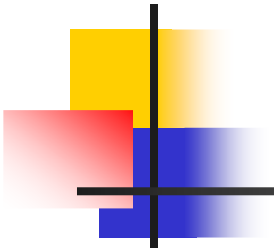
- Tout graphe eulérien est connexe,
- Si un graphe est eulérien, alors il est connexe,
- G eulérien $\Rightarrow$ G connexe,
- Si un graphe n'est pas connexe, alors il n'est pas eulérien,
- G est connexe est une condition nécessaire pour que G soit eulérien.



Parité des sommets

Définition

Un sommet est *pair* si son degré est pair.
Un sommet est *impair* si son degré est impair.

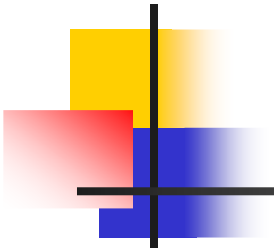


Encore une condition nécessaire

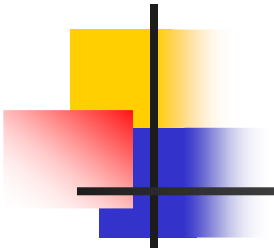
Théorème :

Dans un graphe eulérien, le nombre de sommets impairs est nécessairement 0 *ou* 2*.*

- Le graphe de la ville de Königsberg n'est donc pas eulérien.
- Pour ce qui concerne l'enveloppe, le théorème ne nous donne pas le droit de prédire que son graphe est eulérien.



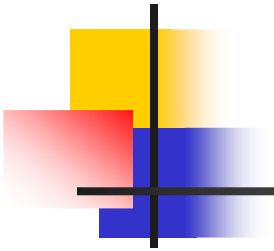
Preuve



La preuve nous donne plus

Cette preuve nous indique comment trouver une chaîne eulérienne dans un graphe ayant deux sommets impairs :

Il faut commencer par un sommet impair, et terminer par l'autre sommet impair.



Conditions suffisantes

Théorème.

Un graphe G est eulérien si et seulement si G est connexe et le nombre de sommets impairs de G est 0 ou 2.

- Il nous reste à établir que si *G* est connexe et possède 0 ou 2 sommets impairs, alors *G* possède une chaîne passant une seule fois par chacune des arêtes de *G.*
- Nous allons donner une preuve "constructive" de ce théorème en construisant cette chaîne, dite *eulérienne.*

Une remarque utile sur les cycles

Un **cycle** est une chaîne

$$C = s_1, a_1, s_2, \ldots, s_k, a_k, s_{k+1}$$

telle que $|\{a_1, \ldots, a_k\}| = k$ et $s_1 = s_{k+1}$.

Remarque :

Si $s_1, a_1, s_2, \ldots, s_k, a_k, s_{k+1}$ est un cycle, alors $s_2, a_2, s_3, \ldots, a_k, s_1, a_1, s_2$ est aussi un cycle.

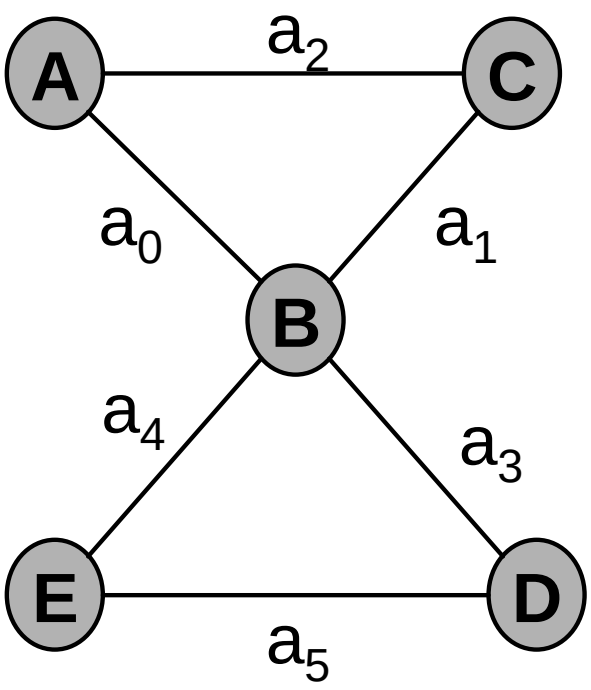
Cas 1 : tous les degrés sont pairs

Algorithme 1 : *construit un cycle C contenant une seule fois chaque arête de G, G étant un graphe connexe où tous les sommets sont pairs.*

1. Choisir un sommet s_1 arbitraire, et former $C = s_1$
2. Tant que le dernier sommet de *C* a une arête incidente *a* qui n'appartient pas à *C*, ajouter à C l'arête *a* et son sommet extrémité. *(À prouver : C est un cycle après l'étape 2)*
3. Si toutes les arêtes de *G* sont dans *C*, alors retourner *C*.
4. Sinon, soit s_i un sommet de *C* ayant une arête *a* qui n'appartient pas à *C*. Former $C' = s_i, a_i, s_{i+1},..., a_m, s_1, a_1,..., a_{i-1}, s_i$.
5. Poser C = C', puis continuer à l'étape 2.

Exemple

Faire tourner l'algorithme précédent sur le graphe ci-dessous :



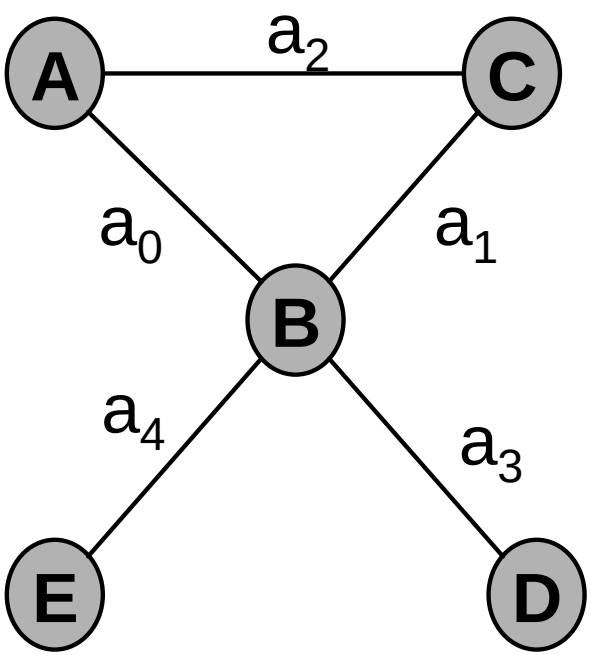
Cas 2 : *G* a deux sommets impairs

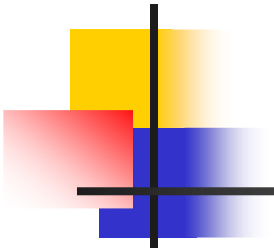
Algorithme 2 : *construit une chaîne C contenant une seule fois chaque arête de G, G étant un graphe connexe avec 2 sommets impairs, disons s et t.*

1. Former l'arête $a = \{s, t\}$, et un nouveau graphe G' tel que $S(G') = S(G)$, et $A(G') = A(G) \cup \{a\}$. Tous les sommets de G' sont pairs.
2. Déterminer un cycle eulérien C pour G' par l'algorithme 1. On suppose que C contient la sous-chaîne s, a, t dans ce sens (sinon inverser le rôle de s et t).
3. Former $C'' = s, a, t, a_i, s_{i+1}, \ldots, s$.
4. Retourner $C = t, a_i, s_{i+1}, \ldots, s$.

Exemple

Faire tourner l'algorithme précédent sur le graphe ci-dessous :





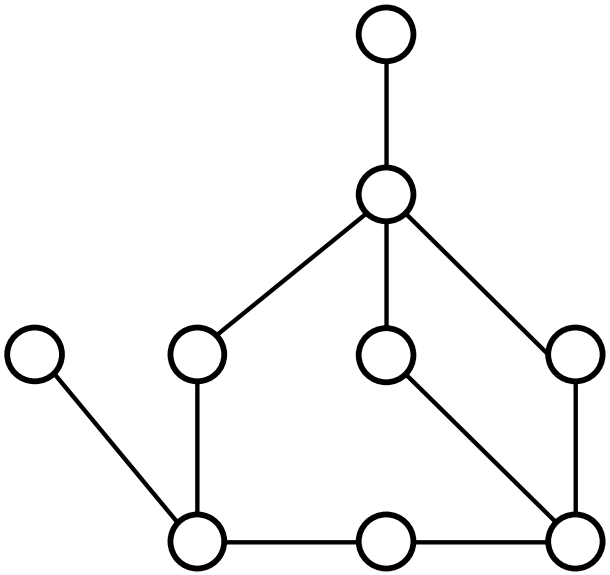
Dans cette partie...

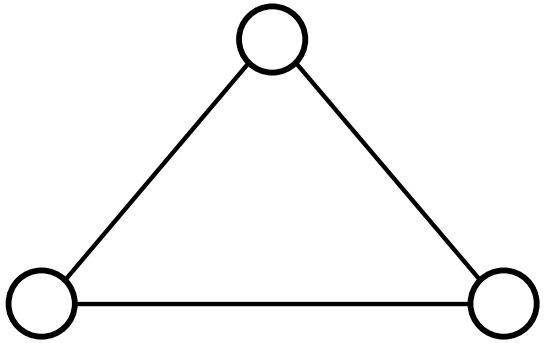
10- Coloration

Coloration

Définition. *Une coloration d'un graphe G en k couleurs est une fonction c* : *S(G)* $\rightarrow$ {1,..., *k*} *telle que, pour tous sommets voisins x et y, c*(*x*) ≠ *c*(*y*).

Un graphe qui a une coloration en *k* couleurs est dit k-coloriable.





Graphes 2-coloriables et longueurs des cycles

Soit C = s_1, a_1,..., a_m, s_{m+1} une chaîne. La longueur de C est son nombre *m* d'arêtes.

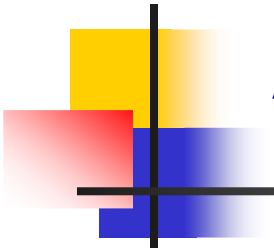
Rappel : C est un cycle si les arêtes a_1,..., a_m sont toutes différentes, et si $s_1 = s_{m+1}$. (La chaîne C = s_1 est donc un cycle de longueur 0).

Théorème. *G est 2-coloriable si et seulement si G n'a pas de cycle de longueur impaire.*

Étape 1 : si *G* contient un cycle de longueur impaire, alors on ne peut pas le colorier avec 2 couleurs.
Étape 2 : si *G* ne contient pas de tel cycle, alors il est 2-coloriable. Pour le montrer, on va construire un algorithme de 2-coloration de *G.*

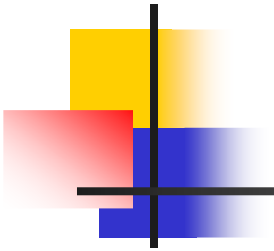
Algorithme de 2-coloration

- On essaie de colorier en deux couleurs, ou bien de trouver un cycle de longueur impaire.
- Quitte à travailler indépendamment sur chaque partie connexe du graphe, on peut supposer G connexe.



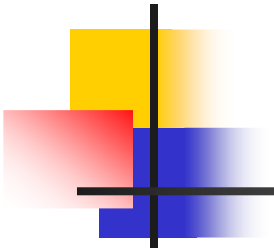
Algorithme de 2-coloration

1. Colorier en couleur **1** un sommet arbitraire s_0
2. Tant qu'il existe un sommet s non colorié ayant au moins un voisin colorié :
   - 2.1 si la couleur de tous les voisins coloriés de s est **1**, colorier s en **2**.
   - 2.2 si la couleur de tous les voisins coloriés de s est **2**, colorier s en **1**.
   - 2.3 sinon, deux voisins de s ont des couleurs différentes, et on a un cycle de longueur impaire.



Pourquoi l'algorithme est-il correct ?

À chaque fois qu'on colorie un sommet, on s'assure qu'il sera de couleur différente de celle de ses voisins déjà coloriés.
Donc, si l'algorithme colorie tous les sommets, il produit une 2-coloration.



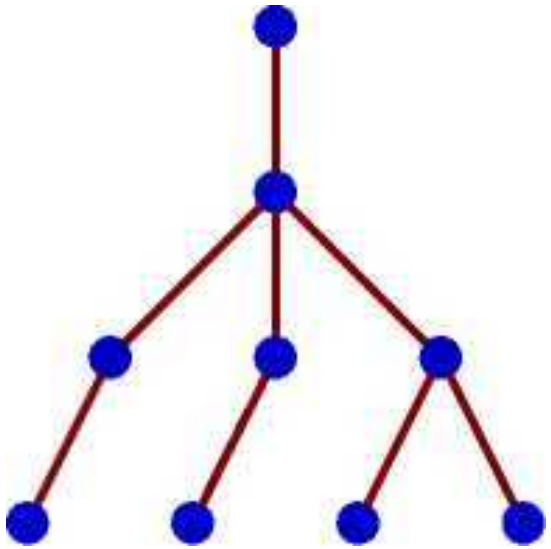
Pourquoi l'algorithme est-il correct ?

Pourquoi y a-t-il un cycle de longueur impaire si l'algorithme détecte que 2 voisins x et y du sommet à colorier s sont déjà coloriés avec 2 couleurs différentes ?

Dans le cas d'un arbre

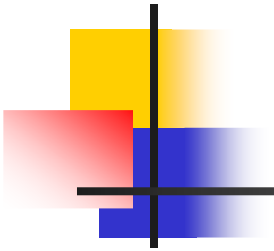
Définition. *Un arbre est un graphe connexe sans cycle.*

Exemple :



Corollaire. *Tout arbre a une coloration en deux couleurs.*

Preuve ?

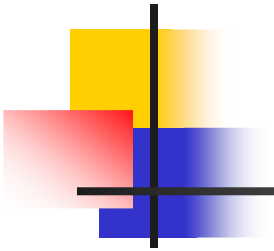


Colorier un graphe en 3 couleurs?

Question : Existe-t-il un algorithme "efficace" permettant de savoir si un graphe peut être colorié en trois couleurs? Existe-t-il un critère simple, facilement vérifiable ?

Réponse : on ne sait pas faire beaucoup mieux que de tester toutes les façons possibles de colorier en trois couleurs chacun des sommets et de vérifier que c'est une coloration. Malheureusement, ce n'est pas efficace, il y a 3^n colorations possibles pour un graphe à n sommets!

Challenge : Il est offert une prime de 1 million de $ à qui trouvera un algorithme de complexité n^c (c constante), ou bien démontrera qu'il n'existe pas d'algorithme de complexité polynomiale.



Comment faire ?

Comment faire alors pour colorier rapidement un graphe avec peu de couleurs?

On utilise une heuristique, c'est-à-dire un algorithme qui ne donne pas à coup sûr le nombre minimum de couleurs, mais qui essaie de s'en rapprocher. Cet algorithme doit être efficace.

Heuristique DEGMIN

Étant donné un graphe G, donne une coloration pour G

Si G a au moins un sommet, faire :

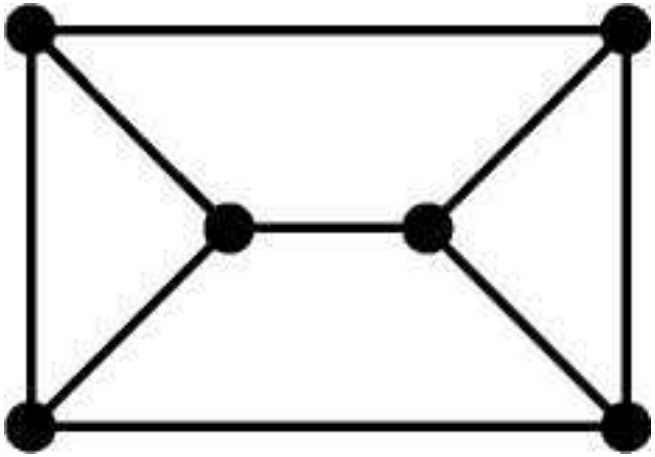
- Choisir un sommet s de degré minimum *d.*
- Soit *G'* le graphe obtenu en supprimant de *G* le sommet s et toutes ses arêtes incidentes.
- Colorier le graphe *G'* en réutilisant DEGMIN.
- Rajouter *s à G'* pour obtenir *G.*
- Colorier s avec la plus petite couleur différente de ses *d* voisins.

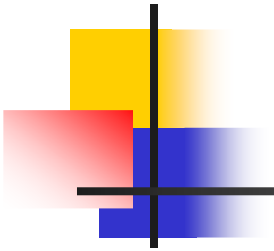
Remarque 1 Pour tout graphe, cet algorithme donne une coloration.

Remarque 2 La complexité est $O(|S(G)| + |A(G)|)$.

Heuristique DEGMIN : un exemple

Couleurs : **bleu = 1, rouge = 2, vert = 3**



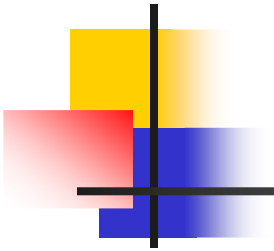


Combien de couleurs utilise DEGMIN ?

Est-ce que DEGMIN marche bien en pratique? Est-ce qu'il colorie avec peu de couleurs?

On va montrer que :

- Si G est un cycle, alors trois couleurs au plus sont utilisées.
- Si G est un arbre, alors deux couleurs au plus sont utilisées.
- Si G est 2-coloriable, alors deux couleurs au plus sont utilisées.
- Si G peut être dessiné sur le plan sans que deux arêtes se croisent, alors six couleurs au plus sont utilisées.



DEGMIN sur arbres et cycles

Cycles

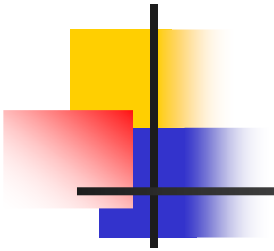
Dans un cycle, chaque sommet a un degré 2. Donc l'heuristique utilise au plus 3 couleurs.
C'est ce qu'on peut faire de mieux si la longueur est impaire.

Arbres (nouvelle preuve qu'ils sont 2-coloriables)

Dans un arbre, il y a un sommet de degré 1 (pourquoi?).
Si on enlève ce sommet, on obtient à nouveau un arbre (?).
Donc chaque sommet enlevé par l'heuristique a un degré 1.
Donc l'heuristique utilise 2 couleurs au plus.

Graphes 2-coloriables.

Par induction, on utilise 2 couleurs sur *G'.*
G est 2-coloriable, on réutilise une de ces couleurs pour s



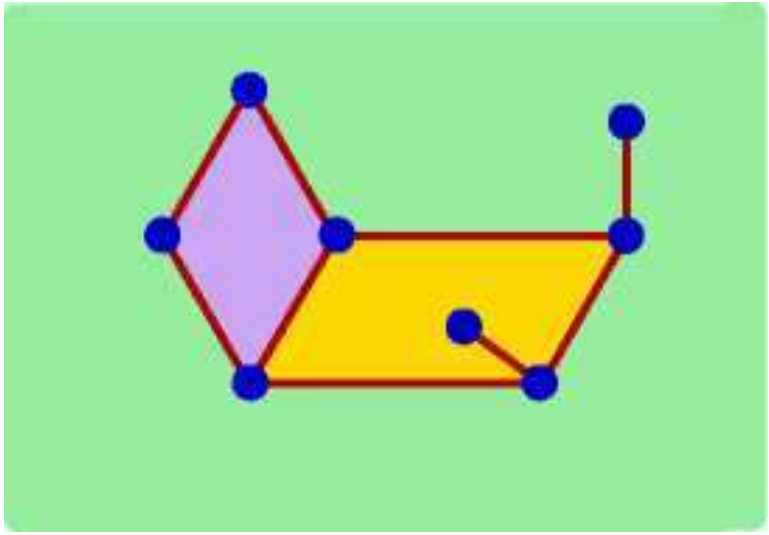
DEGMIN : une propriété importante

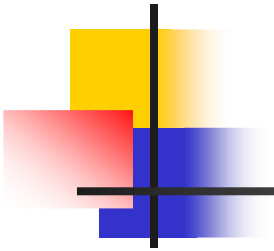
Proposition. *Si à chaque étape le sommet s choisi a un degré d, alors DEGMIN utilisera au plus d* + 1 *couleurs pour colorier le graphe.*

Preuve. Par induction sur le nombre de sommets :

DEGMIN : Graphes planaires

- Un graphe est **planaire** s'il peut être dessiné sur le plan sans que deux arêtes se croisent.
- Une **face** de graphe planaire est une zone du plan délimitée par les arêtes.
- Un graphe fini a une face d'aire infinie. Les autres faces sont d'aire finie.





DEGMIN : Graphes planaires

Euler frappe encore !

Théorème. *Soit G un graphe planaire connexe, n son nombre de sommets, e son nombre d'arêtes et f son nombre de faces. Alors*

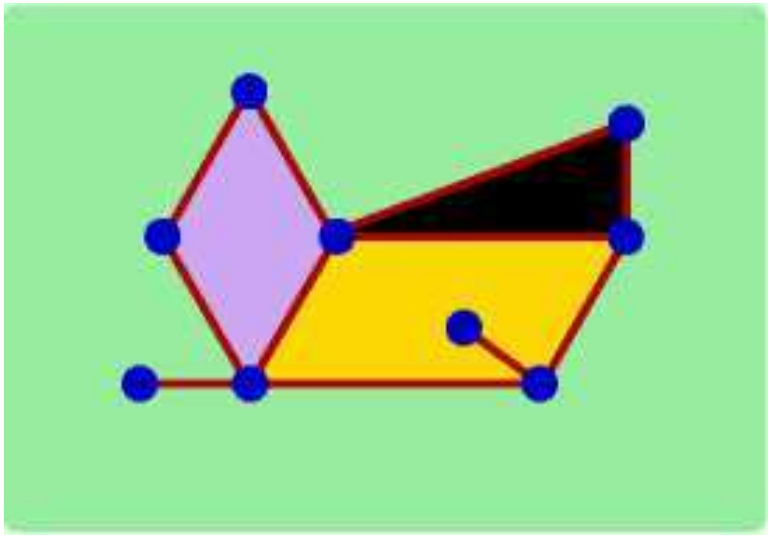
$$n - e + f = 2.$$

Preuve : Par induction sur le nombre de sommets.

Graphes planaires : longueur des faces

La longueur d'une face est la longueur de la chaîne qui la délimite.

Exemple.



La longueur de la face noire est 3 celle de la face violette est 4, celle de la face jaune est 6, celle de la face externe est 9. (Noter que l'arête interne de la face jaune, prise "aller-retour" dans la chaîne, contribue pour 2 unités).

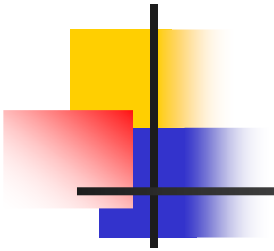
Graphes planaires : densité d'arêtes

Soit $\{F_1,\ldots, F_f\}$ l'ensemble des f faces du graphe planaire connexe G et $L(F_i)$ la longueur de la face F_i.

Proposition. Si G est planaire et connexe, alors

$$L(F_1) + \ldots + L(F_f) = 2\,|A(G)| \qquad (1)$$

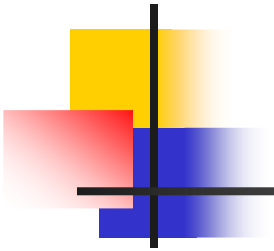
Preuve. Analogue au 1er théorème sur les degrés.



DEGMIN : Graphes planaires

Théorème. *Soit G un graphe planaire simple à au moins 3 sommets. Alors $e < 3n - 6$, et il existe un sommet de degré 5.*

Preuve.



DEGMIN : Graphes planaires

Théorème de 6 couleurs :

Tout graphe planaire est coloriable avec 6 couleurs, sans que deux voisins n'aient la même couleur.

Preuve.

On sait que tout graphe planaire a un sommet de degré ≤ 5.

On sait que si l'algorithme DEGMIN choisit toujours un sommet de degré d, il colorie le graphe avec au plus $d + 1$ couleurs.