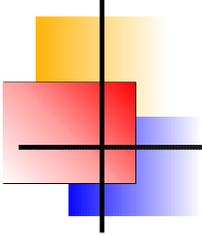


# Initiation à l'informatique (MIS102)

Université Bordeaux 1

Année 2006-2007, Licence semestre 1



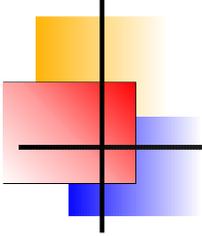
# Plan du cours

---

- |   |                                |    |                   |
|---|--------------------------------|----|-------------------|
| 1 | Présentation et organisation   | 6  | Degré             |
| 2 | Qu'est-ce que l'informatique ? | 7  | Chaînes           |
| 3 | Algorithmes et programmes      | 8  | Connexité         |
| 4 | Introduction aux graphes       | 9  | Graphes Eulériens |
| 5 | Graphes : définition           | 10 | Coloration        |

## 1– Présentation et organisation

- ▶ Objectif et contenu
- ▶ Faut-il des connaissances préalables ?
- ▶ Organisation et site web
- ▶ Support de cours
- ▶ Modalités de contrôle
- ▶ Comptes et tutorat



# Objectifs et contenu

---

**Objectif** : Donner une idée fidèle du contenu des études supérieures en informatique.

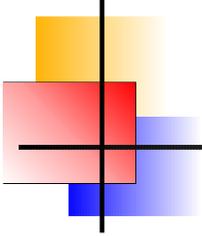
**Thème** : Étude d'un objet appelé *graphe*.

**Contenu** : Théorie des graphes (cours).

Algorithmique des graphes (TD).

Programmation des algorithmes de graphes (TP).

**4 notions à acquérir** : Graphe, algorithme, programme, complexité.



# Faut-il des connaissances préalables ?

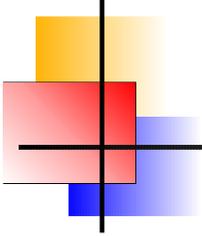
---

## Non prérequis

- ▶ Connaissance d'un langage, d'un système d'exploitation,
- ▶ connaissance de la programmation,
- ▶ connaissance de logiciels destinés au grand public.

## Prérequis

- ▶ Il sera nécessaire de pouvoir comprendre un raisonnement mathématique pour les preuves des théorèmes.



# Organisation et site web

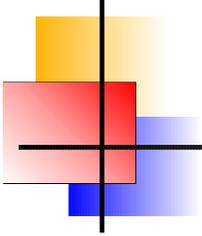
---

**Responsables et cours :** Cyril Gavaille  
Catherine Pannier  
Matthias Robine  
Marc Zeitoun

**Planning :** 6 séances de cours  
5 séances de TD (2h40)  
4 séances de TP (2h40)  
+ travail individuel

**Site Web :** <http://dept-info.labri.fr/initinfo/>  
Supports de cours.  
Textes des TD, TP.  
Annales d'examens.

- Livre (~ 10 euros) : *Initiation à l'informatique*  
par Robert Strandh et Irène Durand  
**Non autorisé à l'examen**  
En vente à la librairie Georges  
(cours de la Libération, Talence)  
[Version html en ligne sur le site du cours](#)
- Transparents



# Modalités de contrôle

---

Épreuve	Durée	Coef.
DS	1h20	0,25
TP <b>individuel</b> noté	1h20	0,15
Examen	1h30	0,60

Vous avez reçu un compte pour les ordinateurs de l'université.

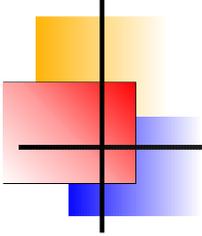
**Il faut suivre les instructions pour activer le compte au moins 24 heures (de préférence plus) avant le premier TP.**

## Tutorat pour

- ▶ activation de comptes,
- ▶ prise en main de l'environnement informatique,
- ▶ soutien pour les cours d'informatique,
  - ▶ du mercredi 20/9 au vendredi 22/9 : 10h-16h (bât. A22/ATI)
  - ▶ à partir du lundi 25/9, entre 12h et 14h (bât. A22/ATI)

## 2– Qu'est-ce que l'informatique ?

- ▶ Qu'est-ce que l'informatique ?
- ▶ L'informatique même pour non informaticiens
- ▶ Quelques domaines de l'informatique
- ▶ Enseignements à Bordeaux 1



# Qu'est-ce que l'informatique ?

---

- ▶ Dans la vie quotidienne : ordinateur avec logiciels.
- ▶ En entreprise : un outil de communication et de production.
- ▶ À l'université : une discipline scientifique.
  - ▶ Une partie pratique (par exemple, autour de la programmation).
  - ▶ Une partie théorique similaire aux maths (objets abstraits).
  - ▶ Les objets en mathématiques : nombres, relations, fonction, transformations, *etc.*
  - ▶ Les objets en informatique : algorithmes, programmes, preuves, systèmes de réécriture, images numériques, **graphes**, *etc.*

- ▶ Le travail d'un scientifique ou d'un ingénieur nécessite de plus en plus la manipulation de logiciels.
- ▶ Ces logiciels sont de plus en plus sophistiqués.
- ▶ Souvent, ces logiciels nécessitent de la programmation.
- ▶ Il faut des connaissances informatiques (algorithmique et programmation) pour
  - ▶ programmer efficacement, et
  - ▶ maintenir les programmes.

# Exemples de domaines en informatique (1)

## Les bases de données



- ▶ 1.070.000.000 internautes en 2005,
- ▶ 42 298 371 sites web en 2003,
- ▶ Trouver **rapidement** un billet d'avion, un trajet, une page web,...



# Exemples de domaines en informatique (2)

## La sécurité

- ▶ Transports,
- ▶ Médecine,
- ▶ Finance,
- ▶ Communications,
- ▶ Énergie,
- ▶ Systèmes embarqués,
- ▶ ....



## Les logiciels

- ▶ Navigateurs internet,
- ▶ Anti-virus,
- ▶ Pare-feu ou passerelle,
- ▶ Clients de messagerie (mail),
- ▶ Jeux,
- ▶ ...



## Les langages de programmation

Les langages de programmation sont souvent utilisés dans des domaines spécifiques.

- ▶ HTML, php, javascript pour la création de pages web,
- ▶ SQL pour les bases de données,
- ▶ Java pour les applications embarquées, les serveurs, + ...
- ▶ C pour les systèmes d'exploitation (Windows, Unix), + ...

- ▶ **Python** pour... demandez à  !

# Exemples de domaines en informatique (5)

## Image et son

- ▶ MP3, JPEG, MPEG : codage et compression
- ▶ Voix par IP, numérisation et transformation.
- ▶ Image 3D, jeux vidéos...



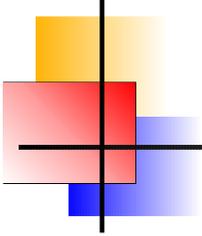
- ▶ Algorithmique (des graphes en particulier)
- ▶ Bio-informatique
- ▶ Cryptologie
- ▶ Génie logiciel
- ▶ Image et son
- ▶ Réseaux
- ▶ Systèmes d'exploitation
- ▶ Vérification du logiciel

et aussi...

- ▶ Architecture
- ▶ Bases de données
- ▶ Programmation, *etc.*

## 3– Algorithmes et programmes

- ▶ Qu'est-ce qu'un algorithme ?
- ▶ Structures de données
- ▶ Complexité
- ▶ Programmation et langage Python
- ▶ En résumé...



# Qu'est-ce qu'un algorithme ?

---

- ▶ Un algorithme est une **méthode systématique** (comme une recette) pour résoudre un problème donné.
- ▶ Il se compose d'une suite d'**opérations simples** à effectuer pour résoudre un problème.
- ▶ Cette méthode peut donc être appliquée par un ordinateur.
- ▶ Exemple : afficher tous les diviseurs d'un entier positif  $n$ .

**si**  $n > 0$  **alors**

**pour tout** entier  $i$  entre 1 et  $n$  **faire**

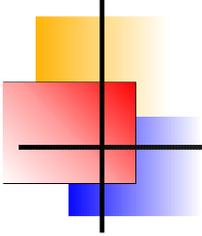
**si** le reste de la division de  $n$  par  $i$  est nul **alors**

            afficher  $i$

**fin si**

**fin pour**

**fin si**



# Importance de l'algorithmique

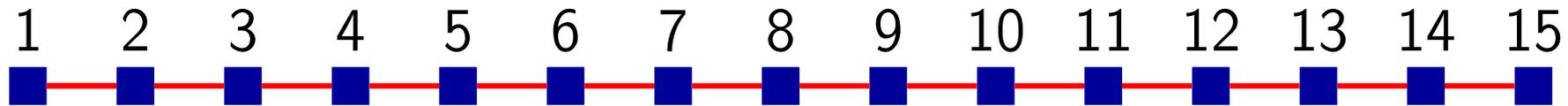
---

- ▶ Il est facile d'écrire des algorithmes faux ou inefficaces.
- ▶ Une erreur peut faire la différence entre plusieurs années et quelques minutes de calculs sur une même machine.
- ▶ C'est souvent une question d'utilisation de structures de données ou d'algorithmes connus dans la littérature.
- ▶ Une structure de données est une façon particulière d'organiser les données.

# Structures de données

**Exemple** Construire une ville de 15 maisons ■ en évitant aux livreurs de pizzas qui suivent les rues — un trajet trop long depuis la pizzeria.

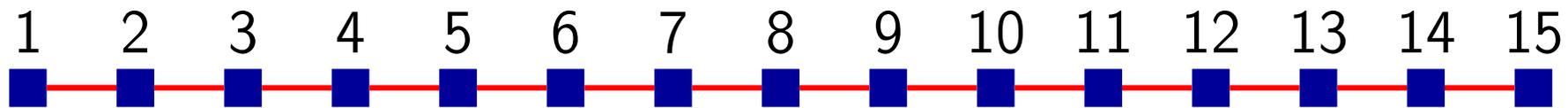
**Organisation 1** : linéaire. Numéros croissants. Pizzeria au numéro 1.



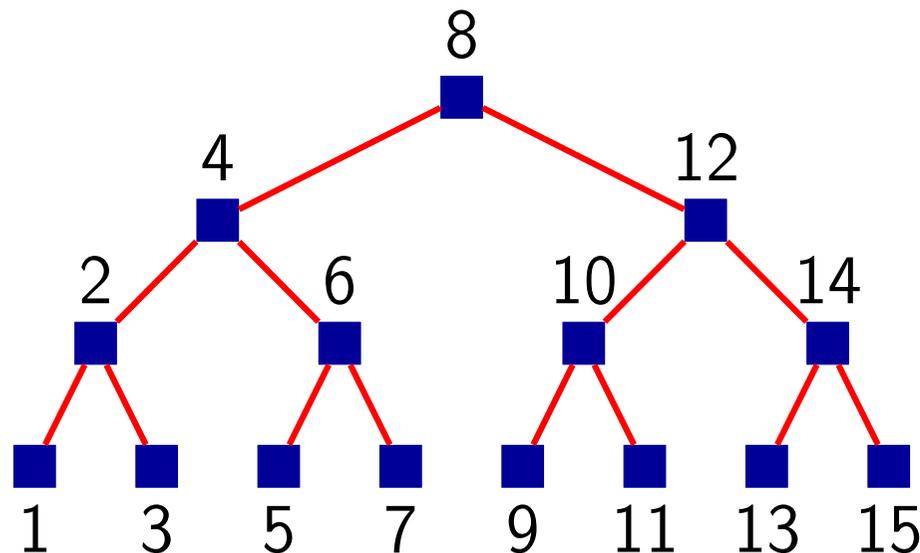
# Structures de données

**Exemple** Construire une ville de 15 maisons ■ en évitant aux livreurs de pizzas qui suivent les rues — un trajet trop long depuis la pizzeria.

**Organisation 1** : linéaire. Numéros croissants. Pizzeria au numéro 1.



**Organisation 2** : Embranchements. À l'ouest de la maison  $k$ , numéros  $< k$ , et à l'est, numéros  $> k$ . La pizzeria est au numéro 8.



# Complexité

- ▶ Dans les deux organisations, le livreur a une méthode simple pour trouver une maison en partant de la pizzeria.
- ▶ **Note** une organisation en étoile avec la pizzeria au milieu permet des trajets très courts, mais **choisir** la bonne rue prend du temps.
- ▶ On suppose qu'il faut une unité de temps pour passer d'une maison à une autre (en suivant une rue).
- ▶ **Quel est, dans le cas le pire, le temps mis par un livreur pour aller jusqu'à une maison depuis la pizzeria ?**

Nombre de maisons	Temps organisation 1	Temps organisation 2
15	14	3

# Complexité

- ▶ Dans les deux organisations, le livreur a une méthode simple pour trouver une maison en partant de la pizzeria.
- ▶ **Note** une organisation en étoile avec la pizzeria au milieu permet des trajets très courts, mais **choisir** la bonne rue prend du temps.
- ▶ On suppose qu'il faut une unité de temps pour passer d'une maison à une autre (en suivant une rue).
- ▶ **Quel est, dans le cas le pire, le temps mis par un livreur pour aller jusqu'à une maison depuis la pizzeria ?**

Nombre de maisons	Temps organisation 1	Temps organisation 2
15	14	3
1023	1022	9

# Complexité

- ▶ Dans les deux organisations, le livreur a une méthode simple pour trouver une maison en partant de la pizzeria.
- ▶ **Note** une organisation en étoile avec la pizzeria au milieu permet des trajets très courts, mais **choisir** la bonne rue prend du temps.
- ▶ On suppose qu'il faut une unité de temps pour passer d'une maison à une autre (en suivant une rue).
- ▶ **Quel est, dans le cas le pire, le temps mis par un livreur pour aller jusqu'à une maison depuis la pizzeria ?**

Nombre de maisons	Temps organisation 1	Temps organisation 2
15	14	3
1023	1022	9
1073741823	1073741822	29

# Complexité

- ▶ Dans les deux organisations, le livreur a une méthode simple pour trouver une maison en partant de la pizzeria.
- ▶ **Note** une organisation en étoile avec la pizzeria au milieu permet des trajets très courts, mais **choisir** la bonne rue prend du temps.
- ▶ On suppose qu'il faut une unité de temps pour passer d'une maison à une autre (en suivant une rue).
- ▶ **Quel est, dans le cas le pire, le temps mis par un livreur pour aller jusqu'à une maison depuis la pizzeria ?**

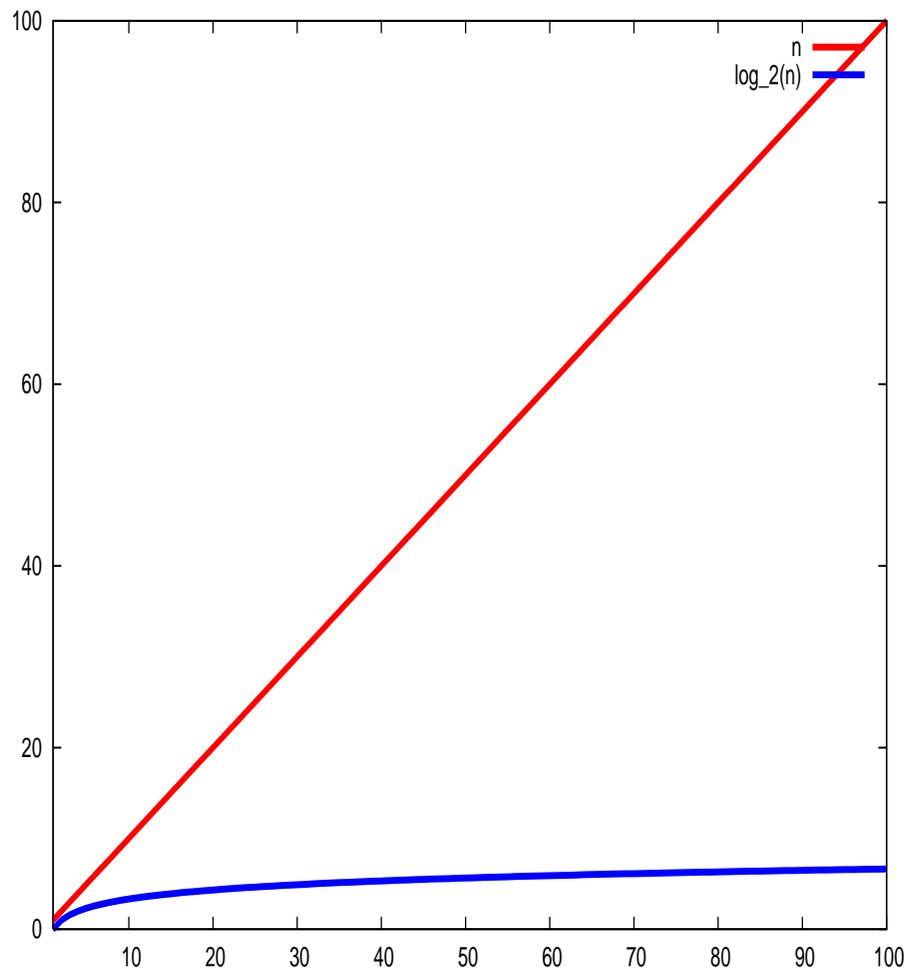
Nombre de maisons	Temps organisation 1	Temps organisation 2
15	14	3
1023	1022	9
1073741823	1073741822	29
$n$	$n - 1$	$\sim \log_2(n)$

- ▶ La **complexité** d'un algorithme est la fonction qui à un entier  $n$  associe le **nombre maximal d'instructions élémentaires** que l'algorithme effectue, lorsqu'il travaille **sur des objets de taille  $n$** .
- ▶ En pratique, on se contente d'un ordre de grandeur.
- ▶ Exemples d'opérations élémentaires :
  - ▶ **additionner, soustraire, multiplier** ou **diviser** deux nombres,
  - ▶ **tester** si une valeur est égale à une autre valeur,
  - ▶ **affecter** une valeur à une variable.

- ▶ Un algorithme a une complexité  $O(f(n))$  si, sur une entrée de taille  $n$  suffisamment grande, il effectue moins de  $K.f(n)$  opérations élémentaires, pour une constante  $K$ .
- ▶ Pour déterminer si un algorithme est efficace, on compte le **nombre d'opérations** nécessaire à effectuer **dans le pire des cas** et *en fonction de la taille de la donnée*.
- ▶ **Exemple**
  - ▶ avec l'organisation 1 de la ville, de taille  $n$  **maisons**, l'algorithme naturel pour trouver une maison a une complexité  $O(n)$ .
  - ▶ avec l'organisation 2 d'une ville de taille  $n$  **maisons**, l'algorithme naturel pour trouver une maison a une complexité  $O(\log_2(n))$ .

# Différence entre $n$ et $\log n$

- ▶ Si  $n = 10^6$ , alors  $\log_2 n \approx 20$ , soit 50 000 fois moins.
- ▶ Si  $n = 10^9$ , alors  $\log_2 n \approx 30$ , soit 30 000 000 fois moins.



# Complexité : second exemple

**Problème** : déterminer si 2 ensembles  $E_1, E_2$  de  $n$  entiers ont une valeur commune.

**Algorithme 1** : comparer successivement chaque élément de  $E_1$  avec chaque élément de  $E_2 \rightsquigarrow n^2$  comparaisons.

237623    5234    983    83889    9

7363    19    873    111    87321

# Complexité : second exemple

**Problème** : déterminer si 2 ensembles  $E_1, E_2$  de  $n$  entiers ont une valeur commune.

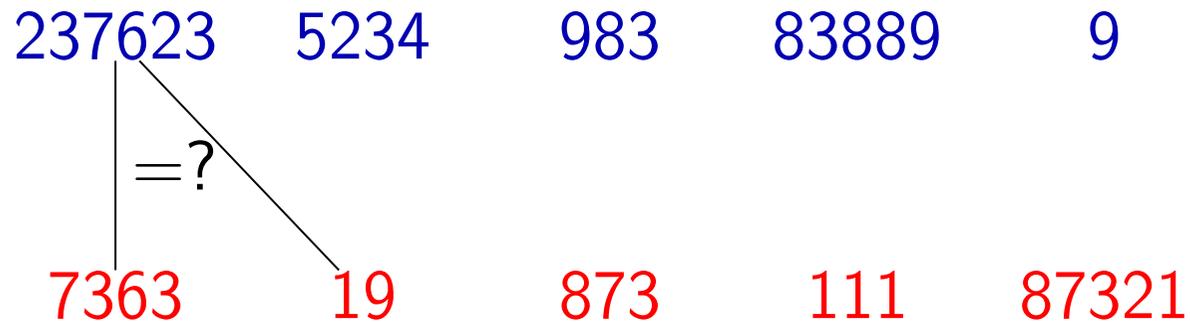
**Algorithme 1** : comparer successivement chaque élément de  $E_1$  avec chaque élément de  $E_2 \rightsquigarrow n^2$  comparaisons.

237623	5234	983	83889	9
=?				
7363	19	873	111	87321

# Complexité : second exemple

**Problème** : déterminer si 2 ensembles  $E_1, E_2$  de  $n$  entiers ont une valeur commune.

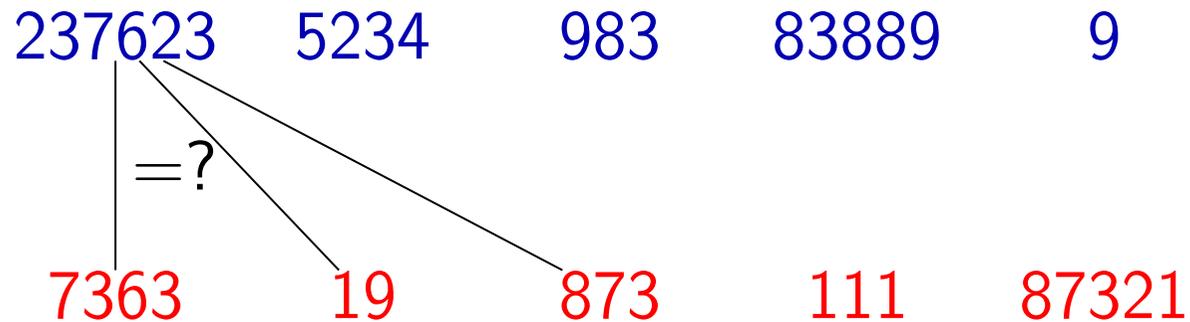
**Algorithme 1** : comparer successivement chaque élément de  $E_1$  avec chaque élément de  $E_2 \rightsquigarrow n^2$  comparaisons.



# Complexité : second exemple

**Problème** : déterminer si 2 ensembles  $E_1, E_2$  de  $n$  entiers ont une valeur commune.

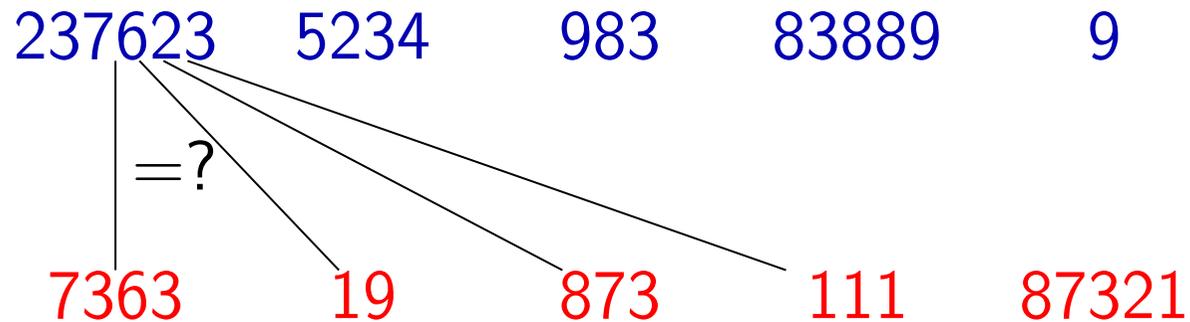
**Algorithme 1** : comparer successivement chaque élément de  $E_1$  avec chaque élément de  $E_2 \rightsquigarrow n^2$  comparaisons.



# Complexité : second exemple

**Problème** : déterminer si 2 ensembles  $E_1, E_2$  de  $n$  entiers ont une valeur commune.

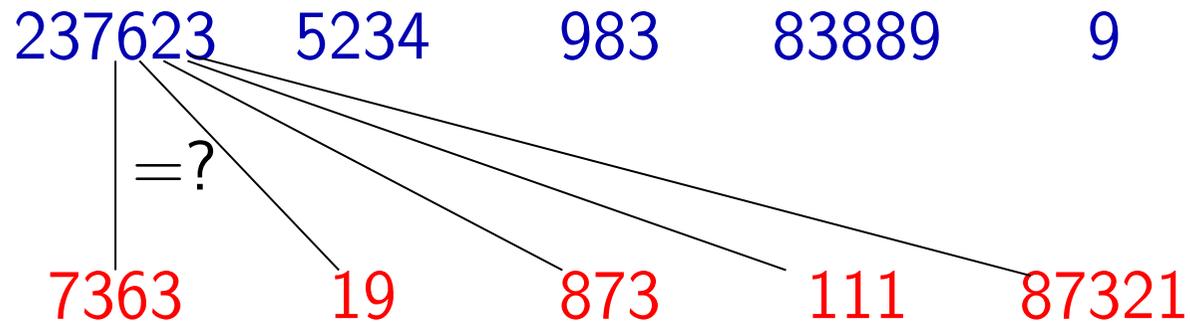
**Algorithme 1** : comparer successivement chaque élément de  $E_1$  avec chaque élément de  $E_2 \rightsquigarrow n^2$  comparaisons.



# Complexité : second exemple

**Problème** : déterminer si 2 ensembles  $E_1, E_2$  de  $n$  entiers ont une valeur commune.

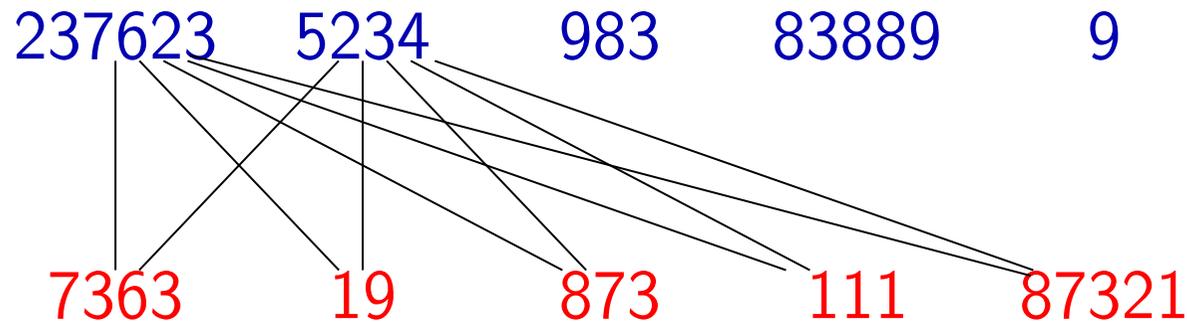
**Algorithme 1** : comparer successivement chaque élément de  $E_1$  avec chaque élément de  $E_2 \rightsquigarrow n^2$  comparaisons.



# Complexité : second exemple

**Problème** : déterminer si 2 ensembles  $E_1, E_2$  de  $n$  entiers ont une valeur commune.

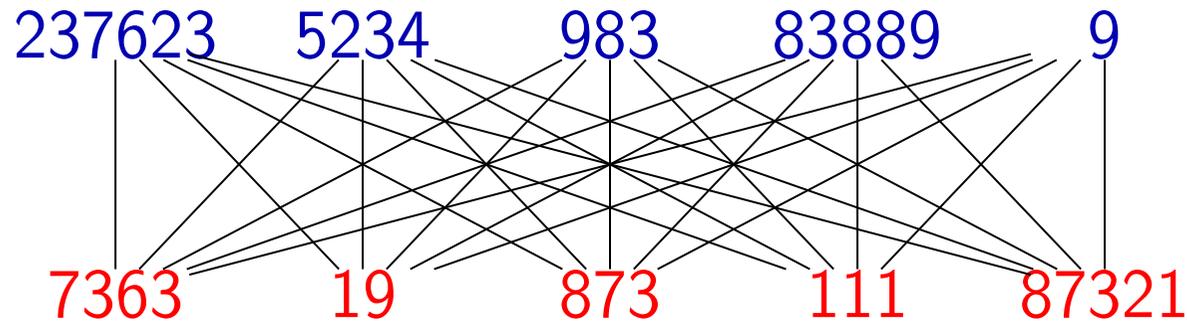
**Algorithme 1** : comparer successivement chaque élément de  $E_1$  avec chaque élément de  $E_2 \rightsquigarrow n^2$  comparaisons.



# Complexité : second exemple

**Problème** : déterminer si 2 ensembles  $E_1, E_2$  de  $n$  entiers ont une valeur commune.

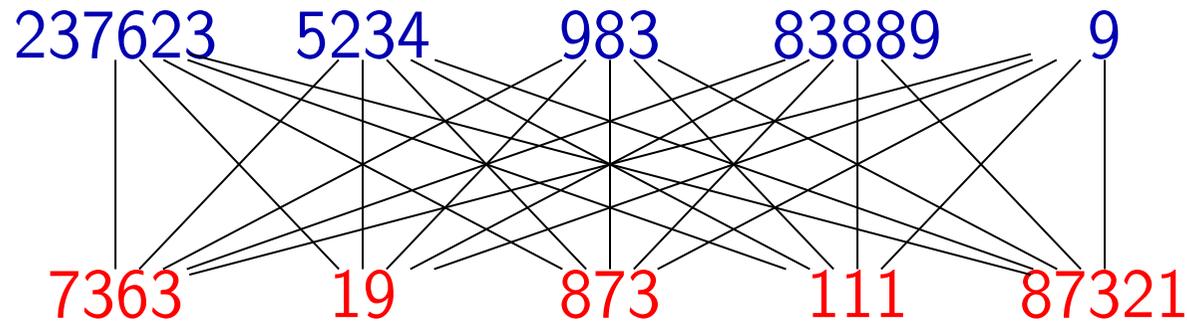
**Algorithme 1** : comparer successivement chaque élément de  $E_1$  avec chaque élément de  $E_2 \rightsquigarrow n^2$  comparaisons.



# Complexité : second exemple

**Problème** : déterminer si 2 ensembles  $E_1, E_2$  de  $n$  entiers ont une valeur commune.

**Algorithme 1** : comparer successivement chaque élément de  $E_1$  avec chaque élément de  $E_2 \rightsquigarrow n^2$  comparaisons.

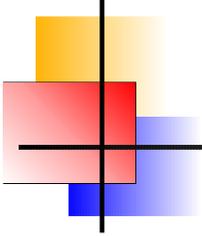


On peut résoudre le problème avec environ  $n \log_{10}(n)$  comparaisons !

$ E_1  =  E_2 $	Algorithme 1	Algorithme 2
$n$	$n^2$	$n \log_{10}(n)$
10	100	10
1000	1000000	3000
100000	10000000000	500000

# Différence entre $O(n^2)$ et $O(n \log n)$

- ▶ Sur un ordinateur exécutant une instruction élémentaire en  $10^{-8}s$
- ▶ Si les ensembles  $E_1$  et  $E_2$  ont  $n = 1\,000\,000 = 10^6$  éléments :
  - ▶ Exécuter  $n \log_{10} n$  instructions élémentaires nécessite  $0,06s$ .
  - ▶ Exécuter  $n^2$  instructions élémentaires nécessite  $10^4s$  soit  $\sim 2h45$ .
- ▶ Si les ensembles  $E_1$  et  $E_2$  ont  $n = 10\,000\,000 = 10^7$  éléments :
  - ▶ Exécuter  $n \log_{10} n$  instructions élémentaires nécessite  $0,7s$ .
  - ▶ Exécuter  $n^2$  instructions élémentaires nécessite  $10^6s$  soit  $\sim 11$  jours
- ▶ En informatique, on manipule parfois des ensembles énormes.
  - ▶ Google indexe plusieurs milliards de pages web,
  - ▶ Google reçoit près de 200 millions de requêtes/jour.

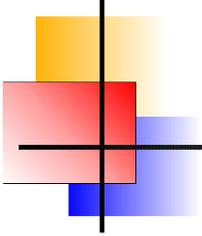


# Choix d'un langage de programmation

---

Paramètres importants pour le choix d'un langage :

- ▶ facilité d'apprentissage, facilité d'utilisation,
- ▶ rapidité d'exécution, rapidité de compilation,
- ▶ absence de défauts dans le compilateur,
- ▶ pérennité (fabricant, langage, implémentation),
- ▶ disponibilité de programmeurs,
- ▶ expressivité du langage (structuration, styles),
- ▶ normalisation, conformité des implémentations.



# Choix d'un langage pour l'enseignement

---

- ▶ Facilité d'apprentissage (moins important dans l'industrie),
- ▶ Utilité plus tard,
- ▶ Facilité de programmer de façon propre et modulaire.

Nous avons choisi le langage Python.

# Caractéristiques de Python

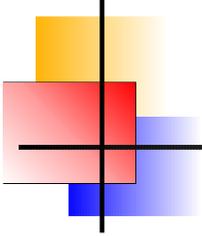
- 😊 implémentation libre et gratuite existe,
- 😊 facilité de manipulation de listes,
- 😊 grand nombre de bibliothèques,
- 😊 largement répandu, beaucoup d'applications,
- ☹️ efficacité moyenne du code,
  - ▶ structure de **bloc** donnée par l'indentation (particulier à Python),
  - ▶ orienté objet...

# Qu'est 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.
- ▶ Par exemple en langage Python :

```
def liste_des_diviseurs(n) :  
    if n > 0 :  
        # pour chaque entier i entre 1 et n...  
        for i in range (1,n+1) :  
            # ...si le reste de la division de n par i est nul...  
            if n % i == 0 :  
                # ...on affiche i  
                print i,  
        print "."
```

```
liste_des_diviseurs(120)
```

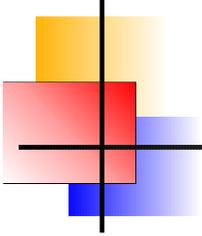


# Un mot sur la programmation

---

- ▶ Il ne suffit pas de construire un programme qui marche.
- ▶ L'essence de la programmation est l'**organisation** pour faciliter la maintenance (représentant environ 80% du coût d'un logiciel).
- ▶ Cela nécessite la construction d'*abstractions* (sous-programmes, modules, classes, extensions syntaxiques, *etc.*).
- ▶ Plusieurs styles de programmation adaptés aux différents types de problèmes :
  - ▶ programmation impérative,
  - ▶ programmation fonctionnelle,
  - ▶ programmation orientée objets,
  - ▶ programmation logique.

Chaque style a ses idiomes de programmation.



# Quelques instructions Python

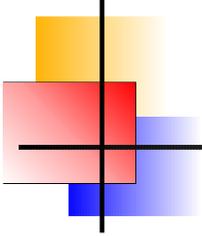
## 1. Affectation : ranger une valeur dans une variable

```
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**.

## 2. Instruction conditionnelle `if` :

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



# Quelques instructions Python (suite)

---

## 3. Répétition :

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

Mais aussi,

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

# Quelques instructions Python (fin)

4. Définition d'algorithme (fonction) :

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

En Python :

```
def f(x) :  
    return 2 * x * x + 1
```

```
y = 2 * f(2)
```

- ▶ Une fonction Python peut utiliser d'autres fonctions.
- ▶ Un programme est en général composé de plusieurs fonctions.
- ▶ Il y a d'autres instructions dans le langage...

<http://www.python.org/>

- ▶ **Algorithme** : méthode donnant une suite d'opérations simples pour traiter un problème donné.
- ▶ **Structure de données** : organisation particulière d'un ensemble de données dans le but d'y accéder rapidement.
- ▶ **Complexité** : façon d'exprimer l'efficacité d'un algorithme, indépendamment d'un ordinateur ou d'un langage de programmation particulier.
- ▶ **Programme** : Réalisation d'un algorithme dans un langage de programmation particulier, comme Python par exemple.

## 4– Introduction aux graphes

- ▶ 1<sup>ère</sup> sorte de graphes : les graphes orientés
- ▶ Quelques exemples de graphes orientés
- ▶ 2<sup>ème</sup> 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 est défini par

- ▶ un ensemble de *sommets* (souvent fini),
- ▶ un ensemble d'*arcs* (souvent fini),
- ▶ à chaque arc est associé est un *couple de sommets*.
- ▶ On représente un graphe par un dessin.

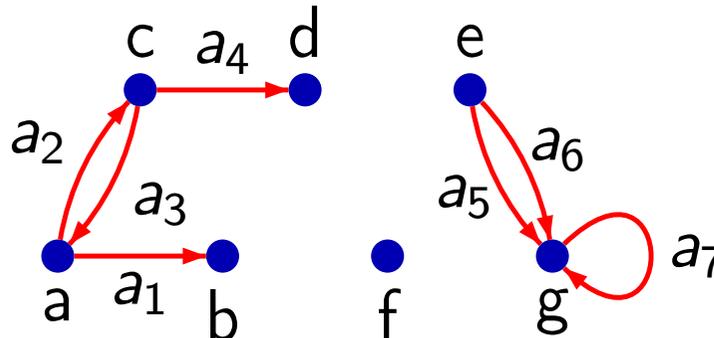
- ▶ Un sommet est dessiné comme un point ●

- ▶ L'arc *a* associé à  $(s, t)$  est représenté par  $s \xrightarrow{a} t$

- ▶ Exemple (graphe simple)

- ▶ Sommets :  $\{a, b, c, d, e, f, g\}$ .

- ▶ Arcs :  $\{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$ .

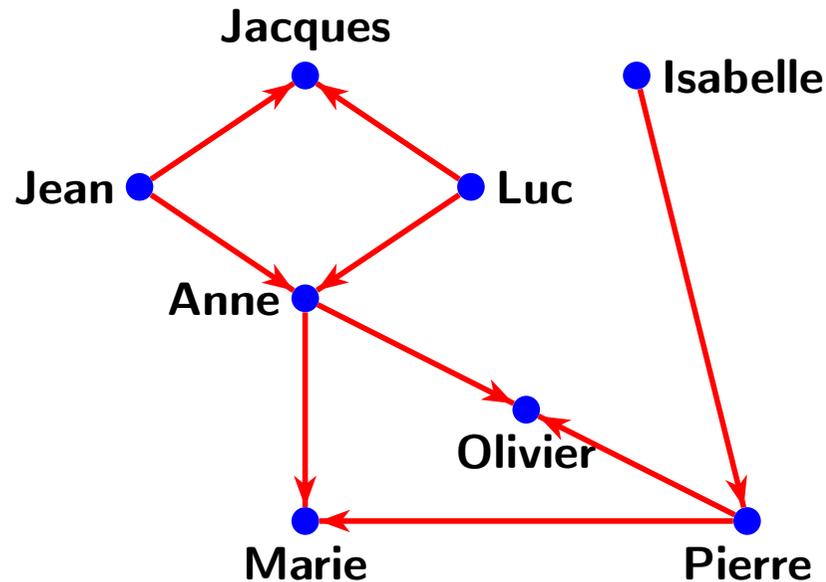


# Exemple de graphe orienté : parents

**Ensemble** : toutes les personnes assistant à un repas de Noël.

**Relation** : l'ensemble des couples de personnes  $(p_1, p_2)$  tels que  $p_1$  a pour parent  $p_2$  (relation **non symétrique**).

**Représentation graphique** (relation **non symétrique**, graphe **orienté**) :

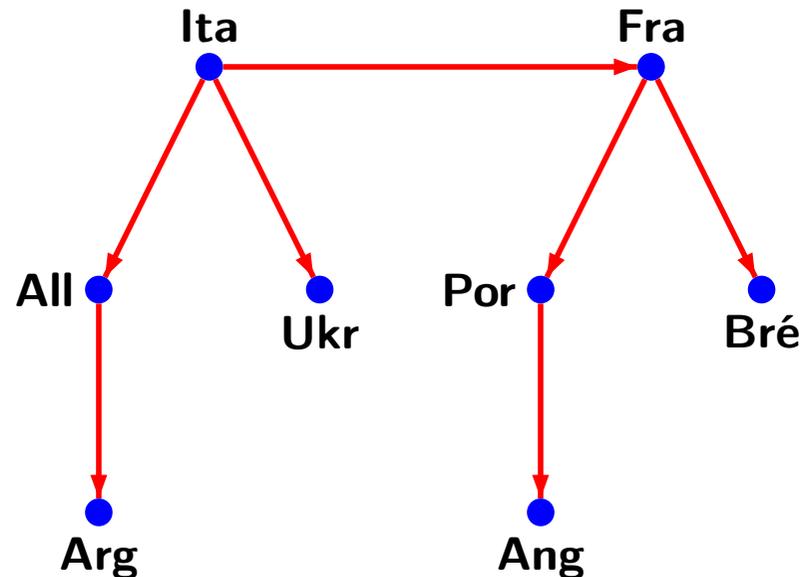


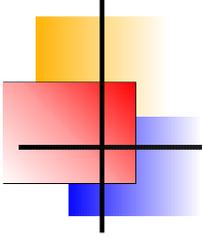
# Exemple de graphe orienté : matchs de foot

**Ensemble** : équipes de foot.

**Relation** : l'ensemble des couples d'équipes  $(eq_1, eq_2)$  telles que  $eq_1$  a battu  $eq_2$  à partir des 1/4 finale de la coupe du monde 2006.

**Représentation graphique** (pas de match nul, relation **non symétrique**, graphe **orienté**).





# Exemple de graphe orienté : internet

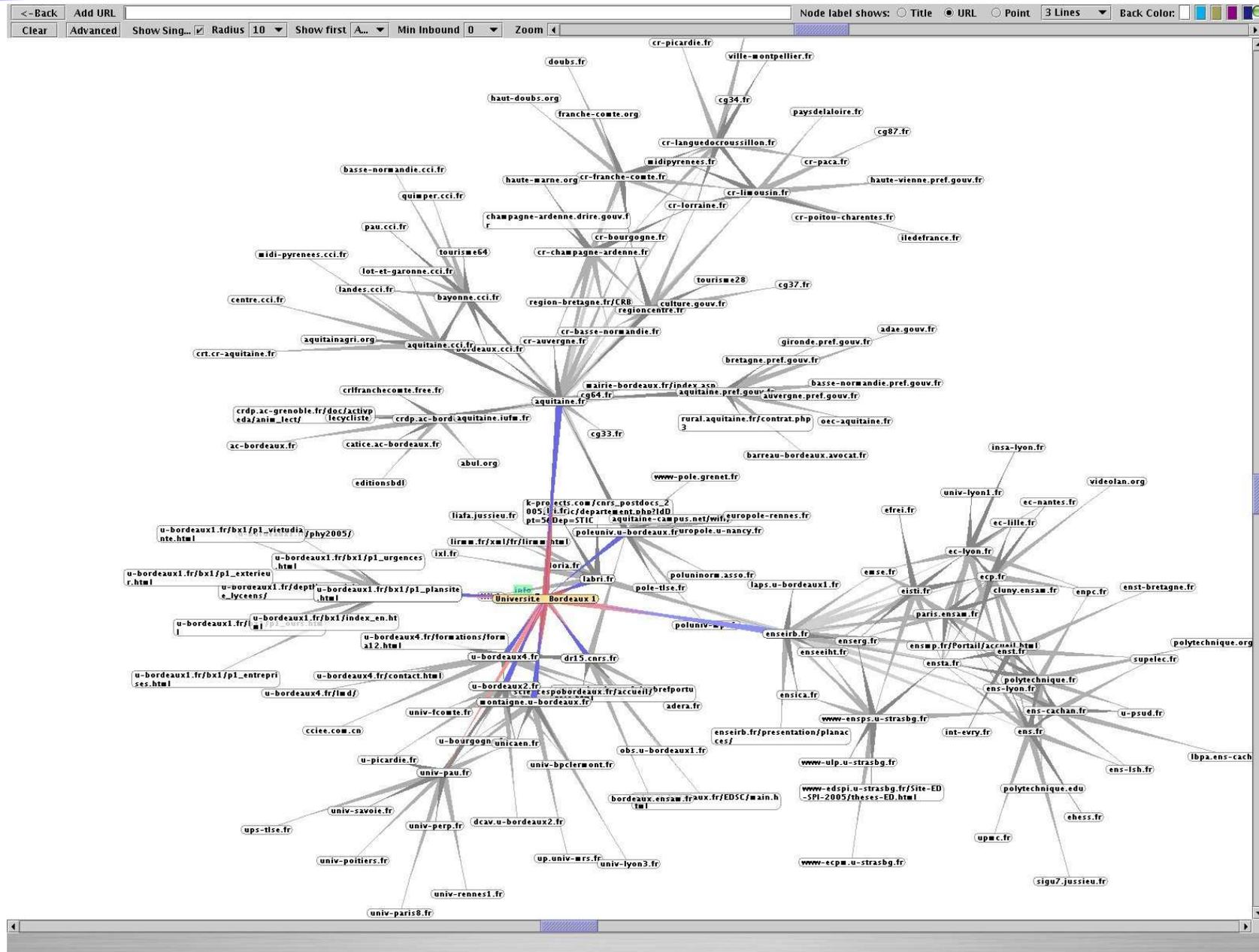
---

Ensemble de sommets : les pages web.

**Relation** : l'ensemble des couples  $(w_1, w_2)$  tels qu'il existe un lien direct sur la page web  $w_1$  qui amène sur la page web  $w_2$ .

Relation **non symétrique**, graphe **orienté**.

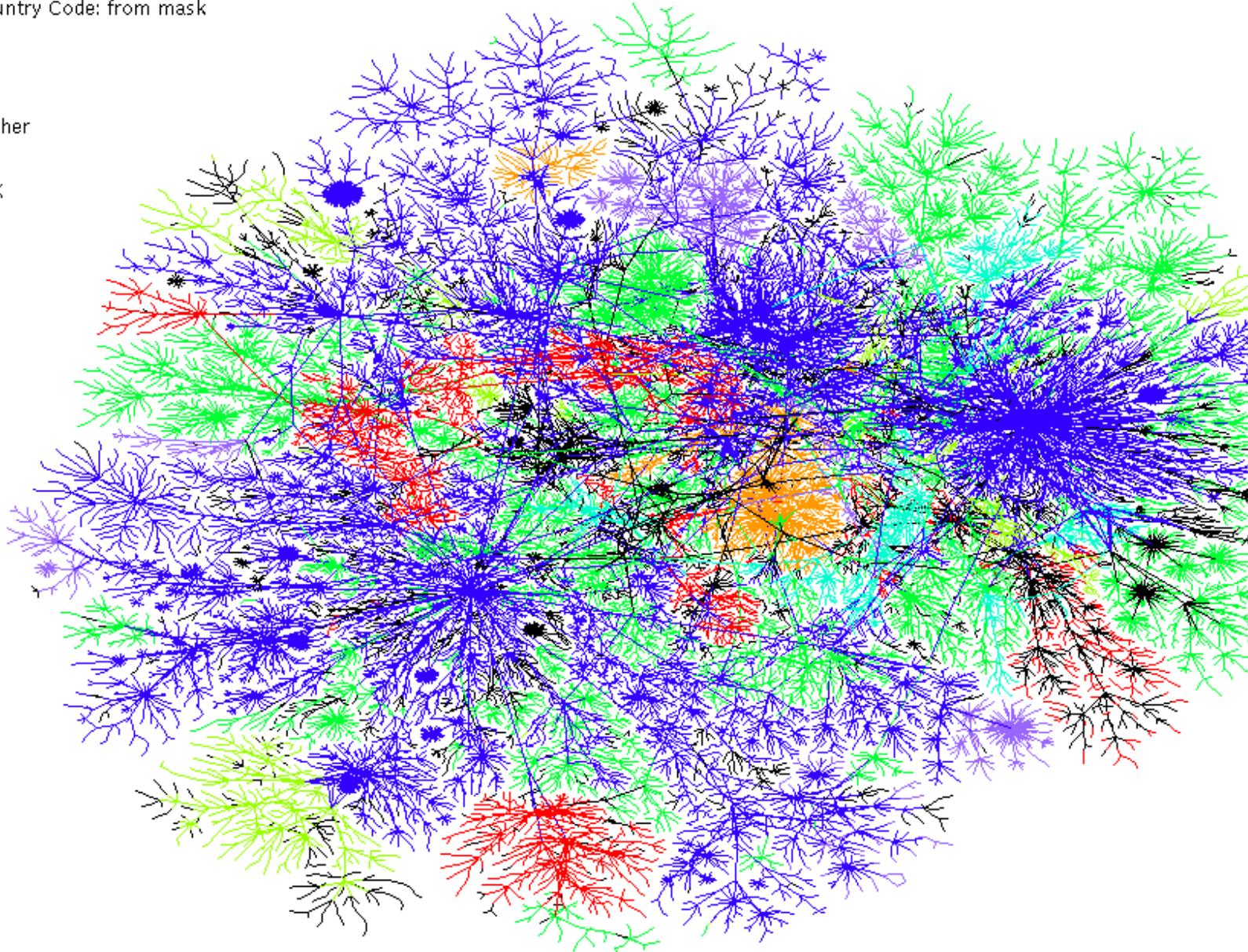
# Représentation graphique (internet)

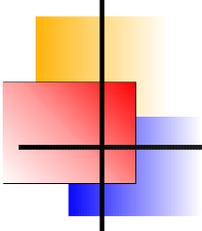


# Représentation graphique : internet

Country Code: from mask

- DE
- IT
- JP
- Other
- SE
- UK
- US

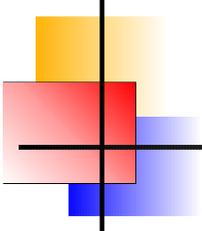




## 2<sup>ème</sup> sorte de graphes : graphes non orientés

---

- ▶ Un graphe orienté peut représenter une relation entre sommets.
- ▶ Une arête associée à  $(s, t)$  exprime que  $s$  et  $t$  sont en relation.
- ▶ Un graphe **non orienté** peut représenter une relation **symétrique**



## 2<sup>ème</sup> sorte de graphes : graphes non orientés

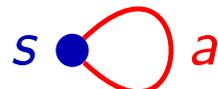
- ▶ Un graphe orienté peut représenter une relation entre sommets.
- ▶ Une arête associée à  $(s, t)$  exprime que  $s$  et  $t$  sont en relation.
- ▶ Un graphe **non orienté** peut représenter une relation **symétrique**
- ▶ Dans un graphe non orienté, une **arête** est associée soit à une **paire de sommets**  $\{s, t\}$ , soit à un **singleton**  $\{s\}$ .

## 2<sup>ème</sup> sorte de graphes : graphes non orientés

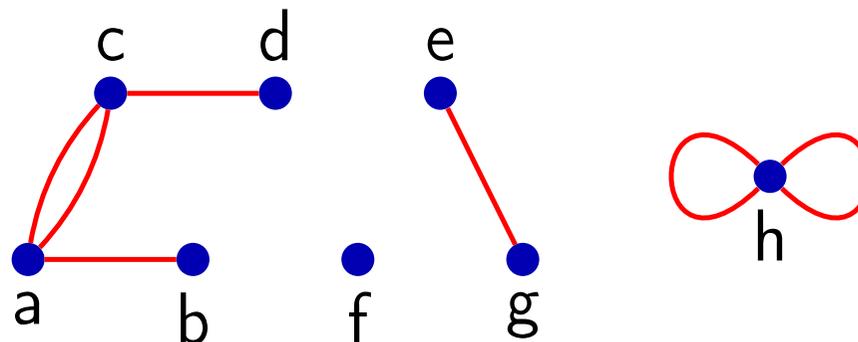
- ▶ Un graphe orienté peut représenter une relation entre sommets.
- ▶ Une arête associée à  $(s, t)$  exprime que  $s$  et  $t$  sont en relation.
- ▶ Un graphe **non orienté** peut représenter une relation **symétrique**
- ▶ Dans un graphe non orienté, une **arête** est associée soit à une **paire 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 

### ▶ Exemple

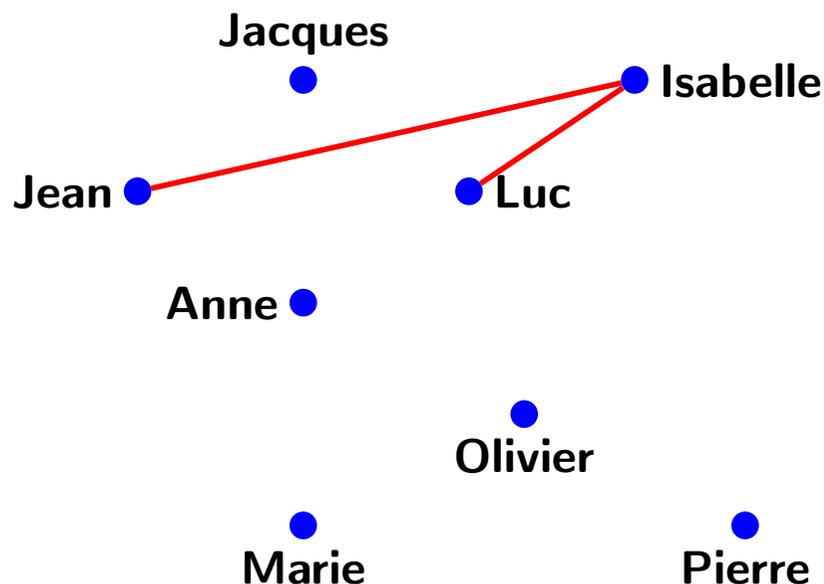


# Exemple de graphe non orienté : cousins

**Ensemble** : toutes les personnes assistant à un repas de Noël.

**Relation** : l'ensemble des couples de personnes  $(p_1, p_2)$  tels que  $p_1$  est un cousin de  $p_2$  (relation **symétrique**).

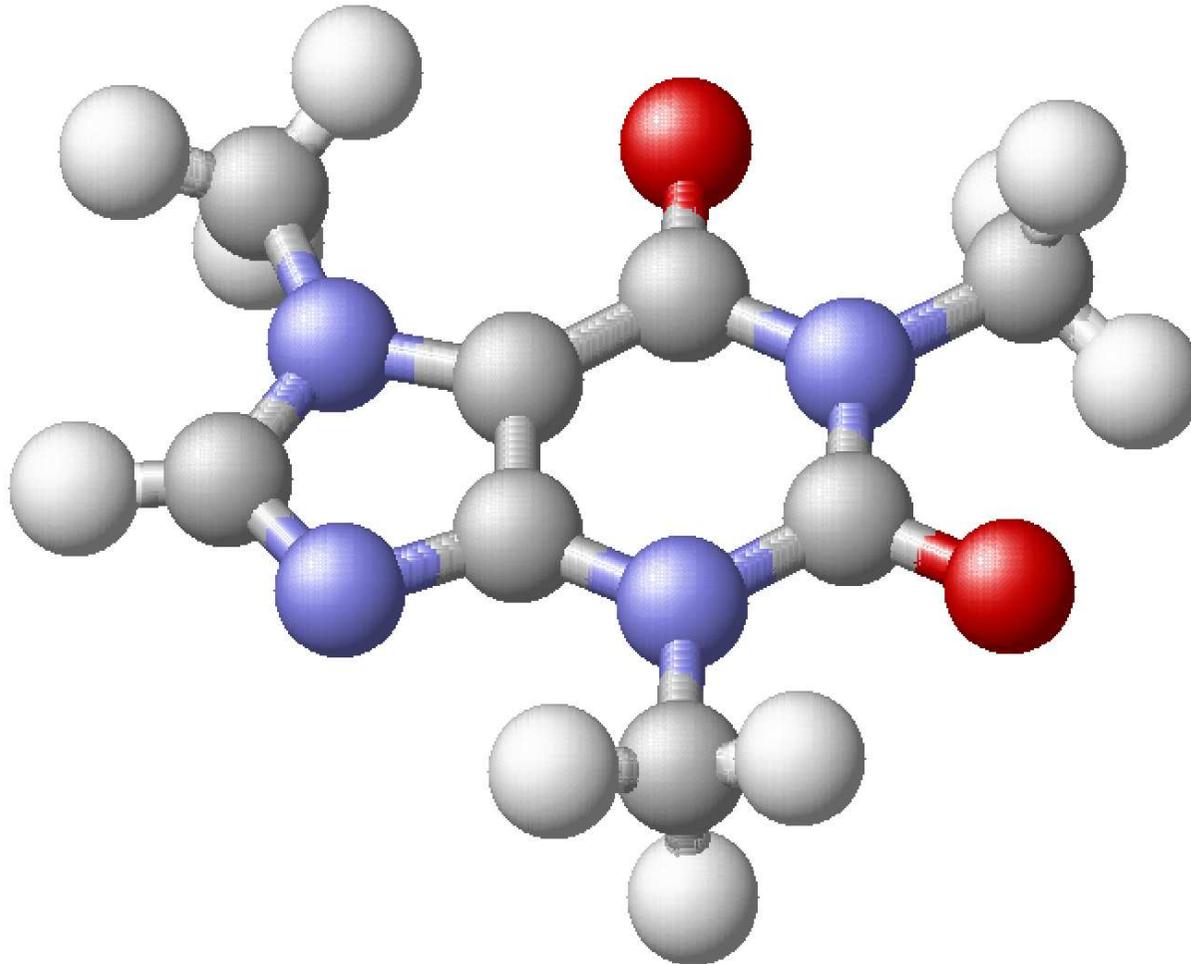
Représentation graphique (relation **symétrique**, graphe **non orienté**) :



# Exemple de graphe non orienté : molécule

**Ensemble** : les atomes de la molécule de caféine.

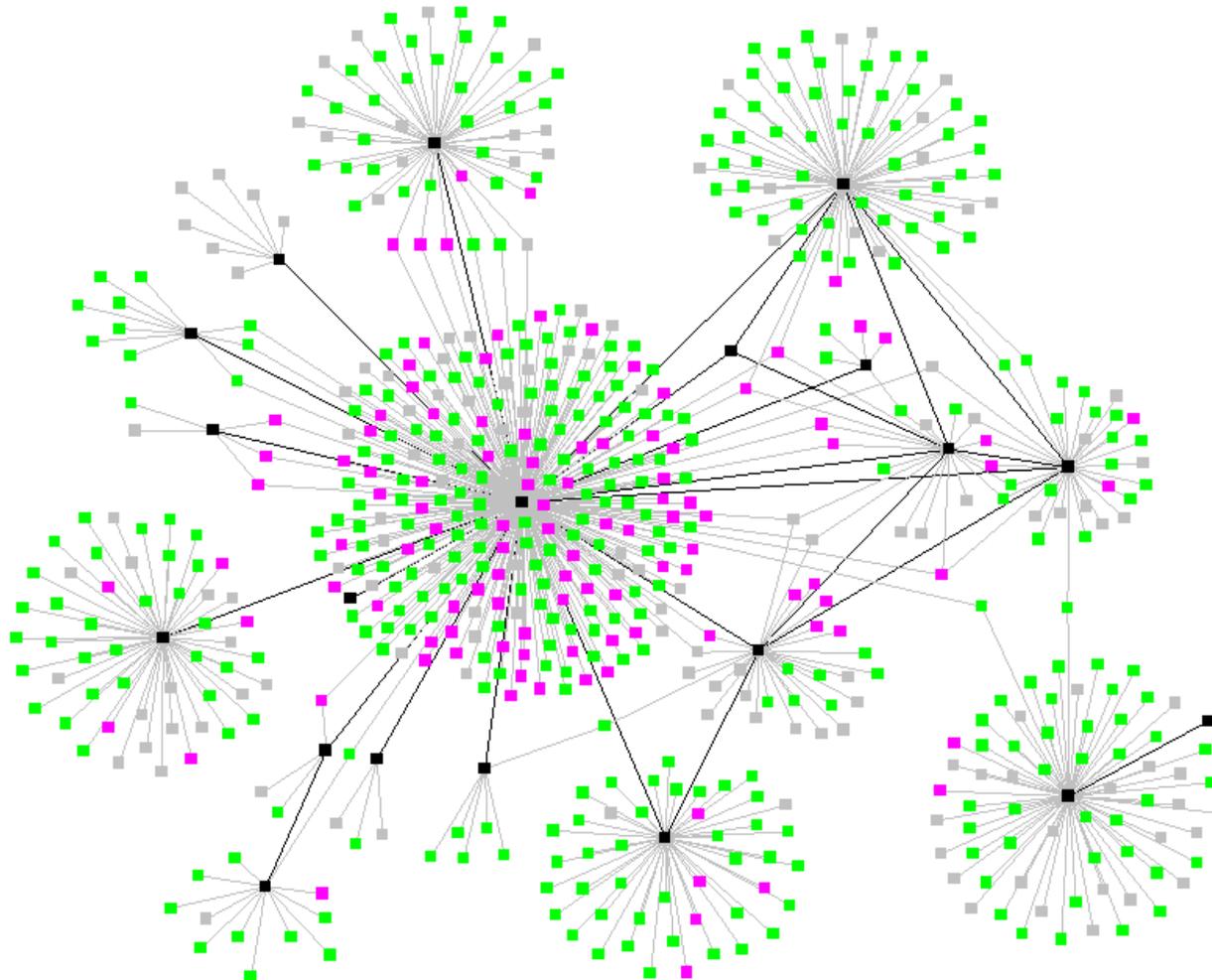
**Relation** : l'ensemble des couples d'atomes  $(a_1, a_2)$  tels que  $a_1$  et  $a_2$  partagent au moins un électron (liaison covalente, relation **symétrique**)

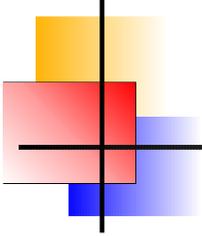


# Exemple - graphe non orienté : connaissance

**Ensemble** : un ensemble de personnes.

**Relation** : ensemble des paires de personnes qui se connaissent.

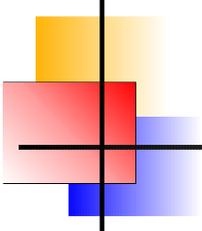




# Modélisation de problèmes par des graphes

---

- ▶ Les graphes sont des objets permettant de représenter plusieurs situations de la vie courante.
- ▶ Des problèmes de la vie courante se **traduisent** en problèmes sur les graphes.
- ▶ Le développement d'algorithmes sur les graphes permet donc de répondre à des problèmes concrets et pratiques.



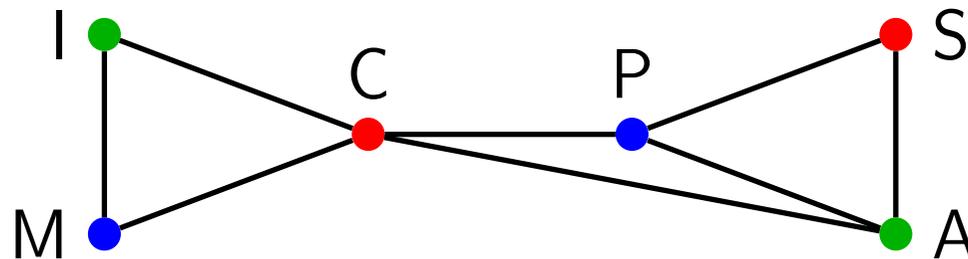
# Organisation d'examens

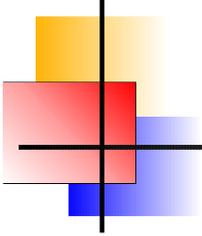
---

- ▶ 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.
- ▶ **Qu** : Quel est le temps le plus court pour organiser les examens ?

# 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.
- ▶ **Qu** : Quel est le temps le plus court pour organiser les examens ?
- ▶ On fait un graphe avec comme sommets M,I,C,P,S,A.
- ▶ 2 sommets sont reliés si un étudiant passe les deux examens.
  - ▶ on relie 2 sommets s'ils sont incompatibles pour le même créneau
- ▶ On colorie les sommets avec le nombre minimal de couleurs, pour que 2 sommets reliés aient des couleurs différentes.
  - ▶ chaque couleur représente un créneau différent.
- ▶ Ici, 3 couleurs  $\implies$  on peut organiser tout en  $3 \times 2h$ .





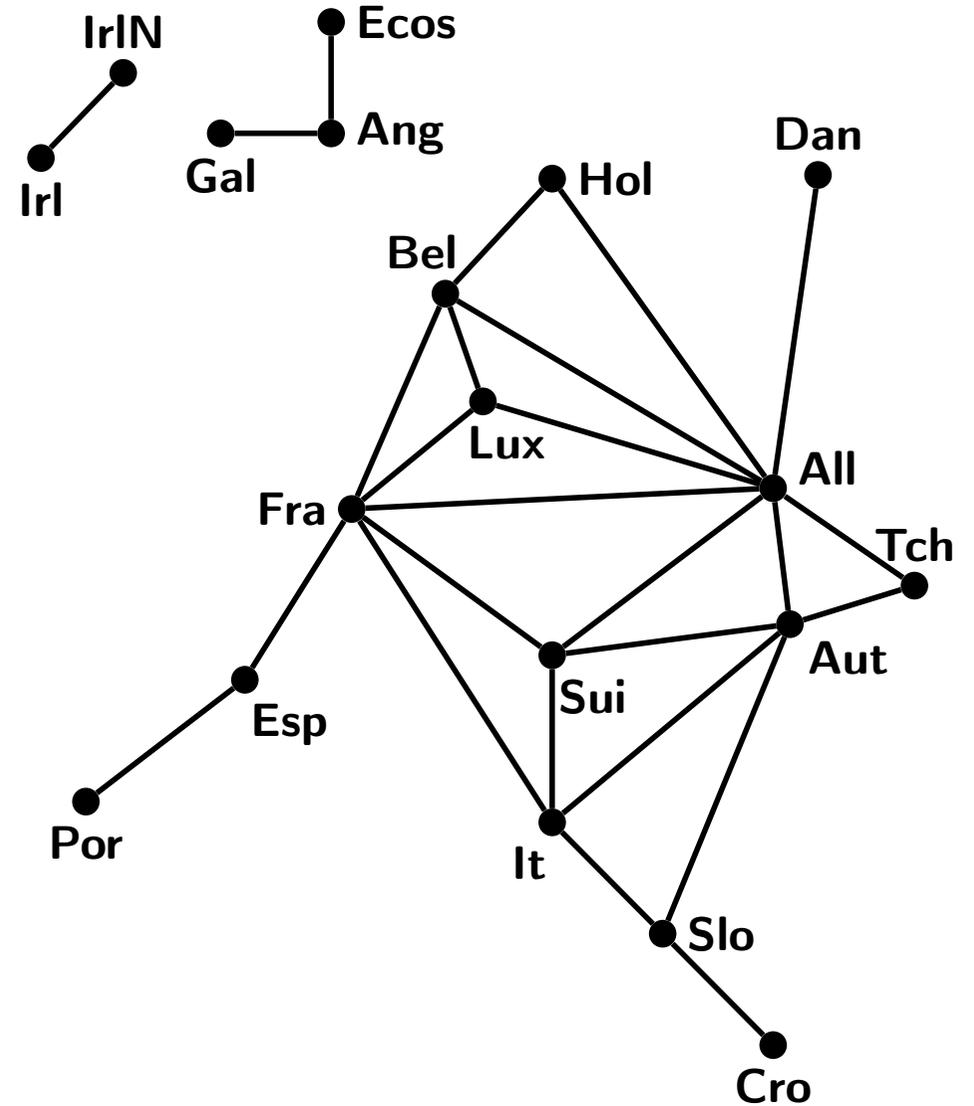
# Coloration de carte géographique

- ▶ 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 géographique  $\rightsquigarrow$  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 colorie le graphe avec 4 couleurs, sans que 2 sommets adjacents n'aient la même couleur.
- ▶ On peut dériver de la preuve un algorithme permettant de trouver la coloration.

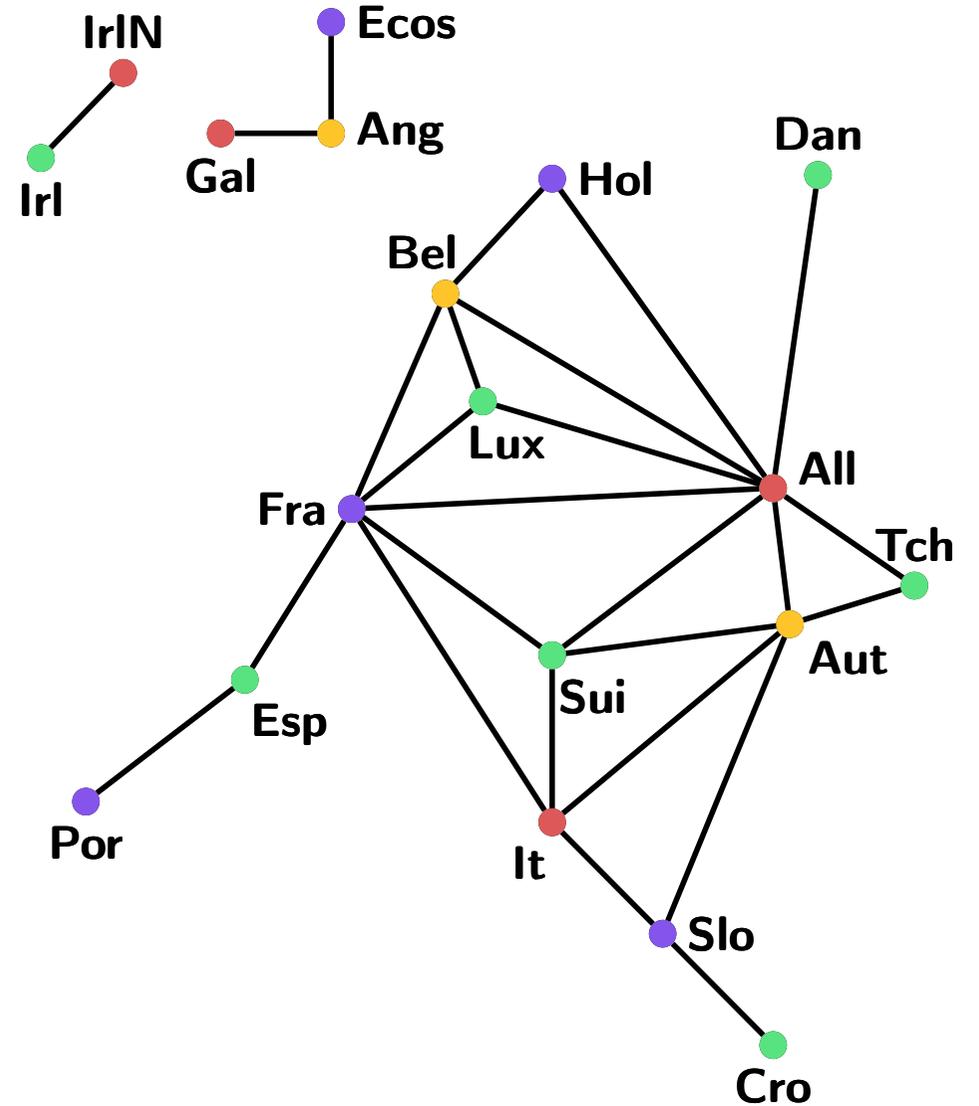
# Coloration de carte géographique



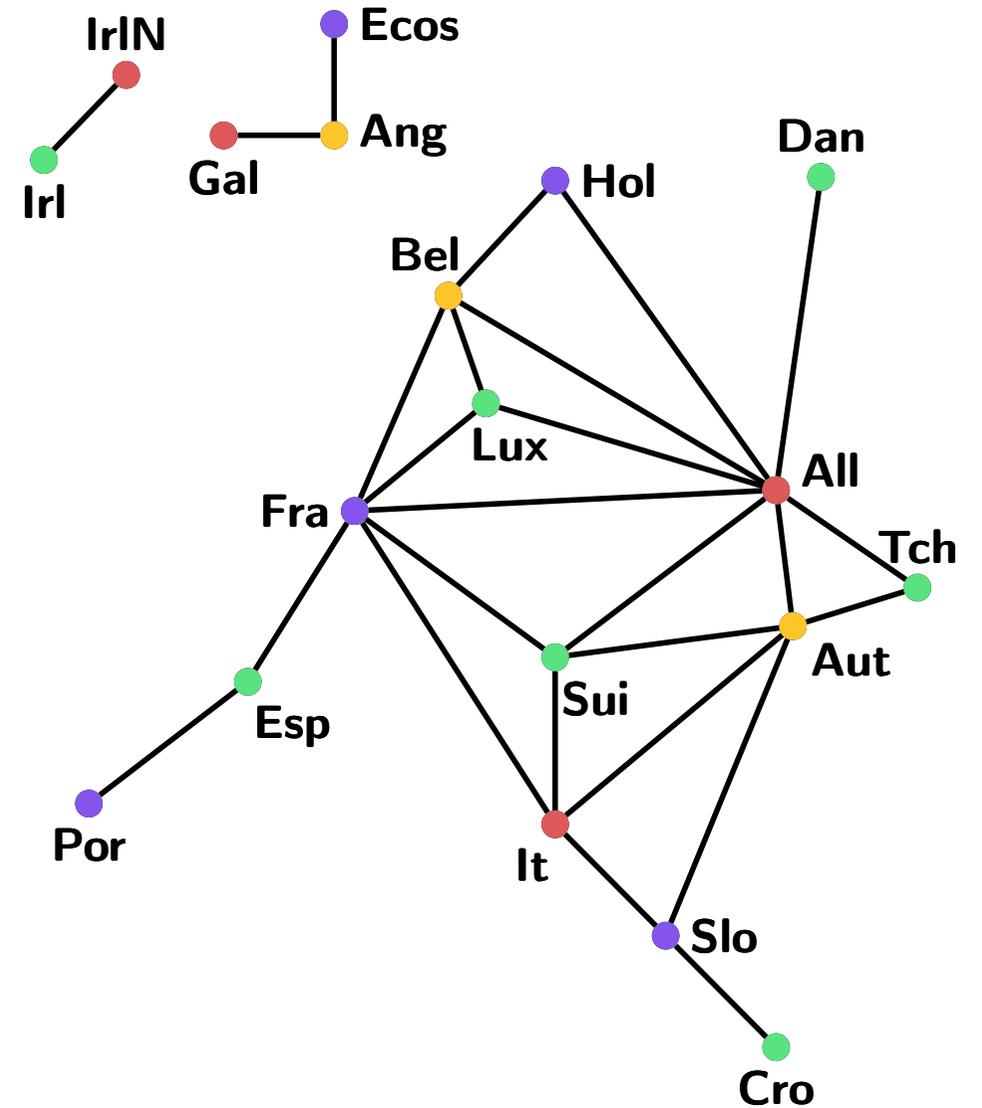
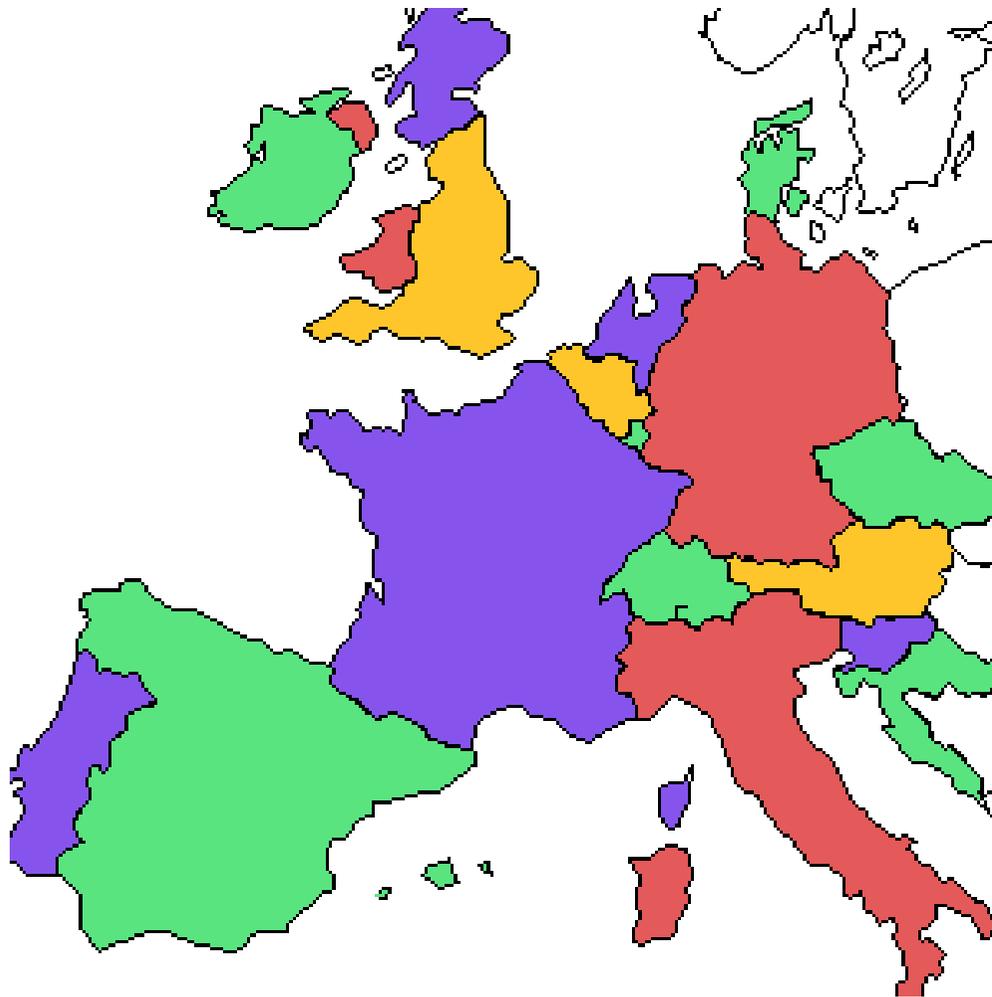
# Coloration de carte géographique



# Coloration de carte géographique

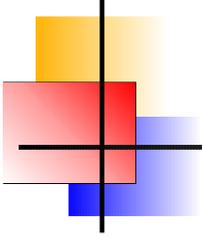


# Coloration de carte géographique



## Remarques

- ▶ 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 pasest équivalent à l'un des problèmes ouverts les plus difficiles, sélectionné par le Clay Mathematical Institute (1 000 000 \$).

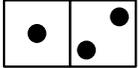


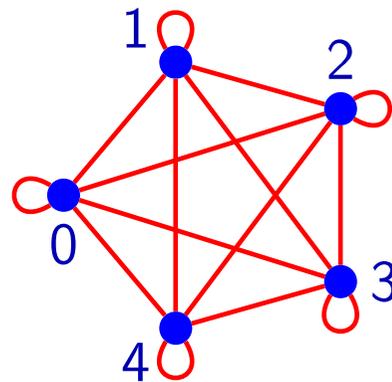
# Dominos

---

- ▶ On dispose d'un jeu de dominos, le plus grand numéro étant  $n$ .
- ▶ **Qu** : Peut-on placer tous les dominos (en suivant les règles) ?

# Dominos

- ▶ On dispose d'un jeu de dominos, le plus grand numéro étant  $n$ .
- ▶ **Qu** : Peut-on placer tous les dominos (en suivant les règles) ?
- ▶ Graphe. **Sommets** :  $\{0, \dots, n\}$ . **Arêtes** : toutes les paires  $\{s, t\}$ .
- ▶ L'arête  $\{s, t\}$  représente le domino dont les numéros sont  $s$  et  $t$ .
  - ▶ Ex. l'arête  $\{1, 2\}$  représente .
  - ▶ Pour  $n = 4$  par exemple, on réalise le graphe suivant.

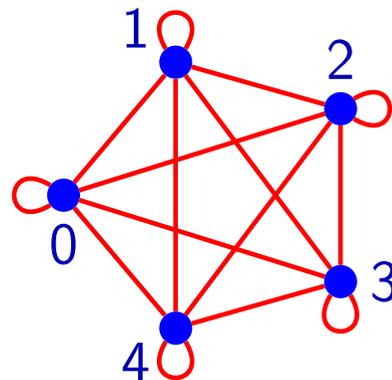


# Dominos

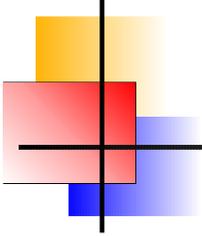
- ▶ On dispose d'un jeu de dominos, le plus grand numéro étant  $n$ .
- ▶ **Qu** : Peut-on placer tous les dominos (en suivant les règles) ?
- ▶ Graphe. **Sommets** :  $\{0, \dots, n\}$ . **Arêtes** : toutes les paires  $\{s, t\}$ .
- ▶ L'arête  $\{s, t\}$  représente le domino dont les numéros sont  $s$  et  $t$ .
  - ▶ Ex. l'arête  $\{1, 2\}$  représente 

•	••
---	----

.
  - ▶ Pour  $n = 4$  par exemple, on réalise le graphe suivant.



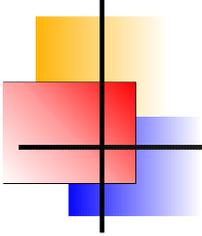
- ▶ On peut placer tous les dominos s'il existe un **chemin** dans le graphe passant une seule fois par chaque arête : **chemin Eulérien**.
- ▶ Un résultat vu plus loin permet de résoudre le problème.



# Problèmes de parcours

---

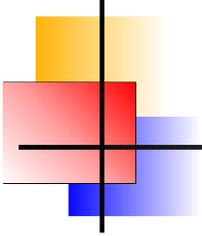
- ▶ Les graphes permettent de modéliser des voies de communication
  - ▶ les sommets représentent des villes,
  - ▶ les arêtes représentent des routes.
- ▶ On peut associer à chaque arête d'un tel graphe (orienté s'il y a des sens uniques) un nombre représentant la distance entre les deux sommets reliés.



# Problèmes de parcours

---

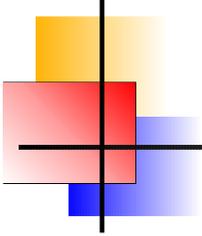
- ▶ Les graphes permettent de modéliser des voies de communication
  - ▶ les sommets représentent des villes,
  - ▶ les arêtes représentent des routes.
- ▶ On peut associer à chaque arête d'un tel graphe (orienté s'il y a des sens uniques) un nombre représentant la distance entre les deux sommets reliés.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



# 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.
- ▶ **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

# Définition de graphe, version centrée arêtes

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

- ▶  $S$  est un ensemble d'objets appelés les *sommets* du graphe,
- ▶  $A$  est un ensemble d'objets appelés les *arêtes* du graphe,
- ▶  $ext$  est une fonction  $ext : A \longrightarrow \mathcal{P}(S)$  telle que

$$\forall a \in A, \quad |ext(a)| \in \{1, 2\}$$

- ▶ Intuition :  $ext(a) =$  le(s) sommet(s) extrémité(s) de l'arête  $a$ .

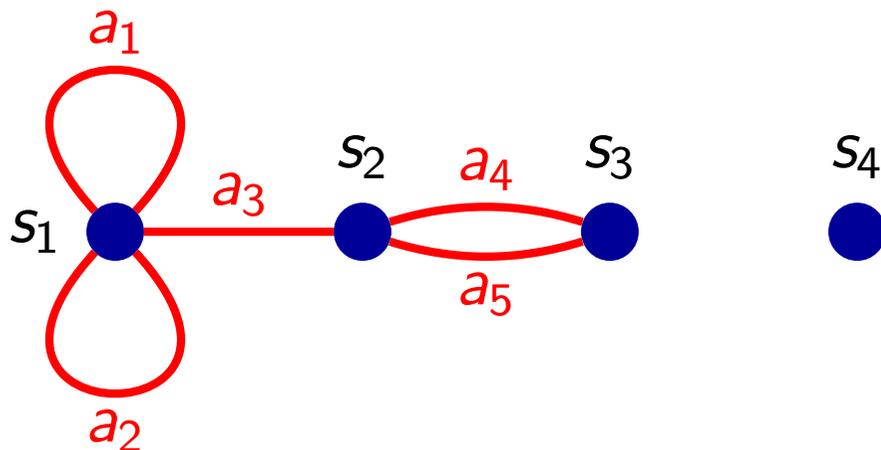
# Définition de graphe, version centrée arêtes

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

- ▶  $S$  est un ensemble d'objets appelés les *sommets* du graphe,
- ▶  $A$  est un ensemble d'objets appelés les *arêtes* du graphe,
- ▶  $ext$  est une fonction  $ext : A \longrightarrow \mathcal{P}(S)$  telle que

$$\forall a \in A, \quad |ext(a)| \in \{1, 2\}$$

- ▶ Intuition :  $ext(a) =$  le(s) sommet(s) extrémité(s) de l'arête  $a$ .



$$ext(a_1) = \{s_1\}$$

$$ext(a_2) = \{s_1\}$$

$$ext(a_3) = \{s_1, s_2\}$$

$$ext(a_4) = \{s_2, s_3\}$$

$$ext(a_5) = \{s_2, s_3\}$$

# Interprétation de la définition

- ▶ Intuition : pour décrire les vols d'une compagnie aérienne,
  - ▶ on décrit chaque **ligne** (= **arête**) par un numéro de vol,
  - ▶ pour chaque ligne, on décrit les **villes** (= **sommets**) qu'elle relie.
  - ▶ Ex. vol **AF6257**, relie {**Bordeaux**, **Paris**}.
- ▶ La fonction **ext** prend comme argument une arête et renvoie un ensemble de sommets (les extrêmités de l'arête).
- ▶ Pour forcer une arête à avoir **une ou deux** extrémités, il faut une restriction sur la taille de l'ensemble renvoyé.
- ▶ La façon d'exprimer cela est :  $ext : A \longrightarrow \mathcal{P}(S)$  telle que

$$\forall a \in A, \quad |ext(a)| \in \{1, 2\}.$$

- ▶  $\mathcal{P}(S)$  désigne l'ensemble des ensembles de sommets.
- ▶  $|ext(a)|$  est le nombre d'éléments de l'ensemble  $ext(a) \subseteq S$ .

# Définition de graphe, centrée sommets

Une autre définition : 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  $inc : S \longrightarrow \mathcal{P}(A)$  telle que

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

- ▶ Intuition :  $inc(s)$  = ensemble des arêtes incidentes au sommet  $s$ .  
(incidentes = qui "touchent").

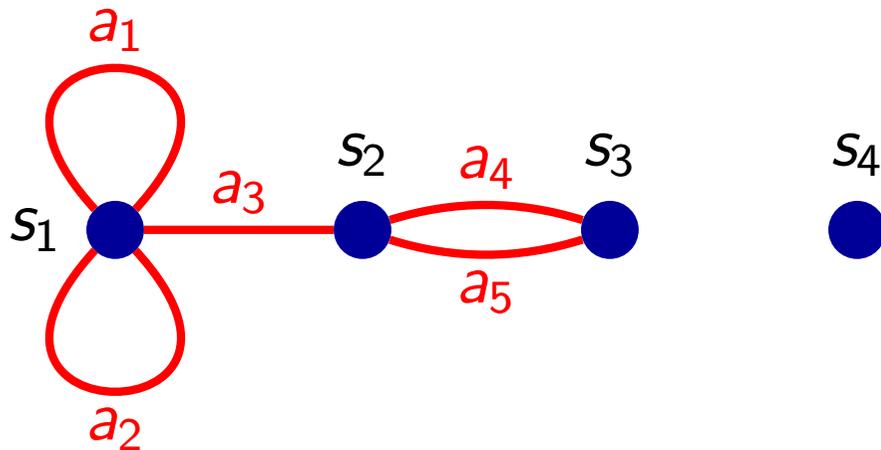
# Définition de graphe, centrée sommets

Une autre définition : 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  $inc : S \longrightarrow \mathcal{P}(A)$  telle que

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

- ▶ Intuition :  $inc(s)$  = ensemble des arêtes incidentes au sommet  $s$ .  
(incidentes = qui "touchent").



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

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

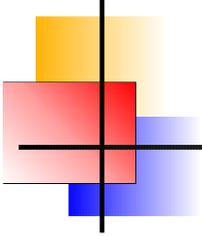
$$inc(s_3) = \{a_4, a_5\}$$

$$inc(s_4) = \emptyset$$

# Interprétation de la définition

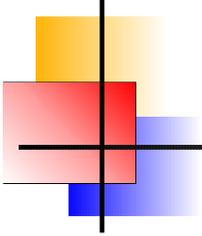
- ▶ C'est une autre façon de décrire les **mêmes objets (les graphes)**.
- ▶ Intuition : pour décrire les vols d'une compagnie aérienne,
  - ▶ on décrit chaque **ville** (= sommet) desservie,
  - ▶ on décrit les **lignes** (= **arêtes**) vers ou depuis chaque ville.
  - ▶ Ex. Ville **Bordeaux**, liaison **AF6257** (vers **Paris**).
- ▶ Le rôle de la fonction **inc** est très différent de celui de **ext**.
- ▶ **inc** est appliquée à un **sommet** et renvoie l'**ensemble des arêtes** qui "touchent" le sommet.
- ▶ La propriété de **inc** :  $S \rightarrow \mathcal{P}(A)$  pour forcer une arête à avoir **une ou deux** extrémités est :

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



# Le graphe en tant que type abstrait

- ▶ Chaque définition a des conséquences sur la programmation.
- ▶ À partir d'un objet de type *graphe*, on peut :
  - ▶ récupérer l'ensemble des sommets du graphe,
  - ▶ récupérer l'ensemble des arêtes du graphe,
  - ▶ **avec la définition centrée arêtes** : à partir d'une arête du graphe, récupérer le(s) sommet(s) extrémités de l'arête.
  - ▶ **avec la définition centrée sommets** : à partir d'un sommet du graphe, récupérer le(s) arêtes(s) dont le sommet est extrémité.
- ▶ On appelle une telle collection d'opérations un ***type abstrait***.
  - ▶ Pour programmer une application, on se pose la question :  
“Quels sont les objets manipulés par le programme, et quelles sont les opérations sur ces objets ?”
- ▶ Le choix du type abstrait dépend souvent de l'application.



# Les deux types abstraits en Python

$\text{NB\_NODES}(g)$  donne le nombre de sommets du graphe  $g$ . Les sommets sont numérotés  $0, 1, \dots, \text{NB\_NODES}(g)-1$ .

$\text{NODE}(g, i)$  donne le sommet numéro  $i$  de  $g$ .

## Def. centrée arêtes

▶  $\text{NB\_EDGES}(g)$  donne le nombre d'arêtes de  $g$ . Les arêtes sont numérotées  $0, 1, \dots, \text{NB\_EDGES}(g)-1$ .

▶  $\text{EDGE}(g, i)$  donne la  $i$ ème arête de  $g$ .

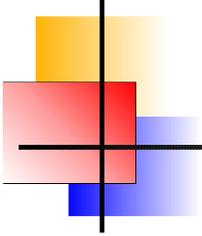
▶  $\text{EDGE\_END}(e, k)$  pour  $k = 0$  ou  $1$ , donne l'une ou l'autre des extrémités de l'arête  $e$ .

## Def. centrée sommets

▶  $\text{NB\_DARTS}(s)$  donne le nombre d'arêtes incidentes au sommet  $s$ . Les arêtes incidentes à  $s$  sont numérotées  $0, 1, \dots, \text{NB\_DARTS}(s)-1$ .

▶  $\text{INC\_EDGE}(s, i)$  donne la  $i$ ème arête incidente au sommet  $s$ .

▶  $\text{FOLLOW\_EDGE}(s, e)$  donne le sommet obtenu partant de  $s$  en suivant  $e$ .



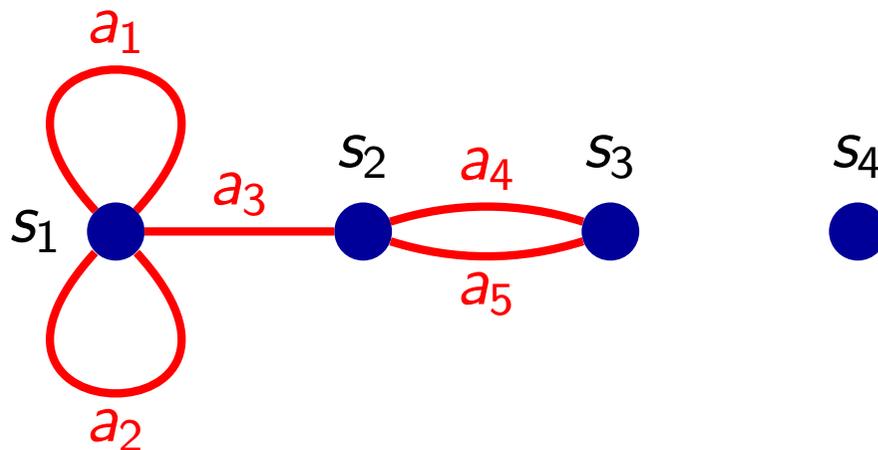
Dans cette partie...

---

## 6– Degré

# 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.



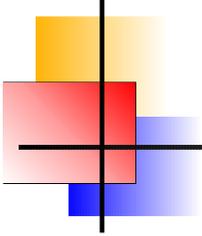
- ▶ Ici  $d(s_1) = 5$ ,  $d(s_2) = 3$ ,  $d(s_3) = 2$ ,  $d(s_4) = 0$ .

# Quelques propriétés des degrés

- ▶ Si  $G$  est un graphe (non orienté), on note
  - ▶  $S(G)$  l'ensemble de ses sommets, et
  - ▶  $A(G)$  l'ensemble de ses arêtes.
- ▶ **Théorème.** Pour un graphe  $G$  ayant au moins un sommet,

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

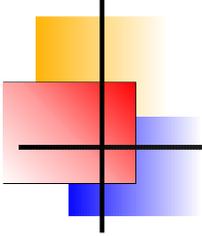
- ▶ **Une idée de preuve :** faire la somme des degrés compte chaque arête 2 fois, car chaque arête a 2 extrémités qui contribuent chacune à une unité de cette somme.
- ▶ **Conséquence.** Dans n'importe quel graphe, le nombre de sommets qui ont un degré impair est pair.



# Preuve formelle : technique d'induction

---

1. Vérifier que  $\sum_{s \in S} d(s) = 2|A|$  pour un graphe sans arête,
2. Prouver que  
si  $\sum_{s \in S} d(s) = 2|A|$  est vrai pour tout graphe à au plus  $k$  arêtes,  
alors c'est aussi vrai pour tout graphe à  $k + 1$  arêtes.



# Preuve formelle du théorème (1/2)

---

Par induction sur le nombre d'arêtes dans le graphe.

1. **Cas de base** La propriété est trivialement vraie pour un graphe avec  $|A| = 0$ , car le degré de chaque sommet du graphe est 0.
2. **Hypothèse d'induction** On suppose que pour un graphe  $G$  avec au moins un sommet et au plus  $k$  arêtes,

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

## Preuve formelle du théorème (2/2)

3. **Induction** Soit  $H$  un graphe à  $k + 1$  arêtes. En supprimant l'une des arêtes, disons  $a$  entre  $s_1$  et  $s_2$ , on obtient un graphe  $G$  ayant au plus  $k$  arêtes. D'après l'hypothèse d'induction, on a

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

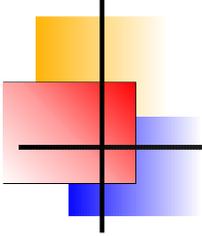
Le degré de  $s_1$  et  $s_2$  dans  $H$  est 1 de plus que leur degré dans  $G$ .

$$\text{Donc : } \sum_{s \in S(H)} d(s) = \sum_{s \in S(G)} d(s) + 2 = 2(|A(G)| + 1)$$

Comme  $G$  est obtenu de  $H$  en supprimant une arête, on a  $|A(G)| + 1 = |A(H)|$ , et donc finalement

$$\sum_{s \in S(H)} d(s) = 2|A(H)|$$

Nous avons donc prouvé la propriété par induction.



Dans cette partie...

---

## 7– Chaînes

- ▶ 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

$$C = s_1, a_1, s_2, a_2, \dots, s_k, a_k, s_{k+1}$$

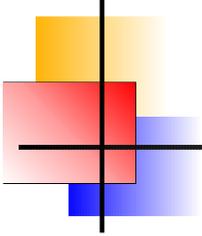
de  $k + 1$  sommets et  $k$  arêtes en alternance, telle que  $\forall i, 1 \leq i \leq k$ , les extrémités de  $a_i$  sont  $s_i$  et  $s_{i+1}$ . On dit alors que  $C$  *est une chaîne entre  $s_1$  et  $s_{k+1}$* .

## 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 sommetsne suffit pas (pourquoi) ?



# Existence d'une chaîne

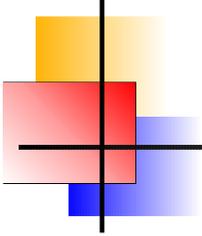
---

- ▶ 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.

- ▶ Une chaîne  $C = s_1, a_1, s_2, a_2, \dots, s_k, a_k, s_{k+1}$  est *simple* si et seulement si

$$\forall i, j \in [1, k + 1], \quad (i < j \text{ et } s_i = s_j) \implies (i = 1 \text{ et } j = k + 1)$$

- ▶ 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  $s \in S(G)$  et  $t \in S(G)$ , alors il existe une chaîne *simple* entre  $s$  et  $t$ .

La preuve est *constructive*. On prend une chaîne non simple et on supprime les cycles.

- ▶ Soit  $C = s_1, a_1, s_2, a_2, \dots, s_k, a_k, s_{k+1}$  la chaîne.
- ▶ Si  $C$  est simple, le travail est terminé.
- ▶ Sinon, il y a deux sommets  $s_i$  et  $s_j$  tels que  $i, j \in [1, k + 1]$ , pour lesquels  $i < j$ ,  $\{i, j\} \neq \{1, k + 1\}$ , et  $s_i = s_j$ .
- ▶ On écrit donc :

$$C = s_1, a_1, \dots, s_i, a_i, \dots, s_j, a_j, \dots, s_k, a_k, s_{k+1}.$$

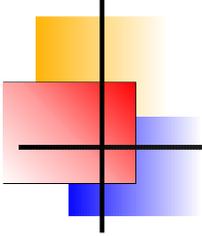
- ▶ Soit  $C = s_1, a_1, s_2, a_2, \dots, s_k, a_k, s_{k+1}$  la chaîne.
- ▶ Si  $C$  est simple, le travail est terminé.
- ▶ Sinon, il y a deux sommets  $s_i$  et  $s_j$  tels que  $i, j \in [1, k + 1]$ , pour lesquels  $i < j$ ,  $\{i, j\} \neq \{1, k + 1\}$ , et  $s_i = s_j$ .
- ▶ On écrit donc :

$$C = s_1, a_1, \dots, s_i, a_i, \dots, s_j, a_j, \dots, s_k, a_k, s_{k+1}.$$

- ▶ Soit  $C = s_1, a_1, s_2, a_2, \dots, s_k, a_k, s_{k+1}$  la chaîne.
- ▶ Si  $C$  est simple, le travail est terminé.
- ▶ Sinon, il y a deux sommets  $s_i$  et  $s_j$  tels que  $i, j \in [1, k + 1]$ , pour lesquels  $i < j$ ,  $\{i, j\} \neq \{1, k + 1\}$ , et  $s_i = s_j$ .
- ▶ On écrit donc :

$$C = s_1, a_1, \dots, s_i, a_i, \dots, s_j, a_j, \dots, s_k, a_k, s_{k+1}.$$

- ▶ On construit  $C'$  plus courte en supprimant de  $C$  la partie  $a_i, \dots, s_j$ .
- ▶ On obtient alors :  $C' = s_1, a_1, \dots, s_i, a_j, \dots, s_k, a_k, s_{k+1}$ .



# Preuve (suite)

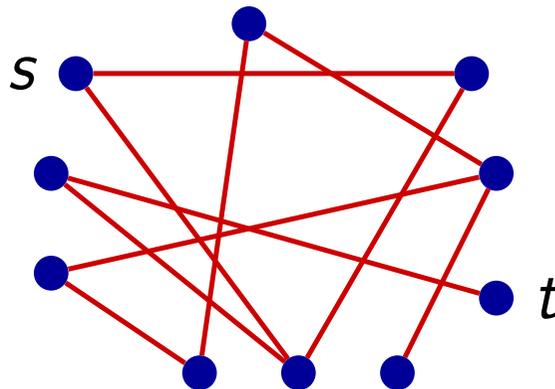
---

- ▶ Il reste à vérifier que  $C'$  est une chaîne.
- ▶ Chaque arête de la chaîne doit être entourée de ses deux extrémités.
- ▶ C'est le cas dans  $C'$ .
- ▶ On répète le procédé tant que  $C'$  n'est pas simple.
- ▶ [Ou : on fait une preuve par induction sur la longueur de la chaîne.]

# Existence d'une chaîne entre $s$ et $t$

Algorithme :

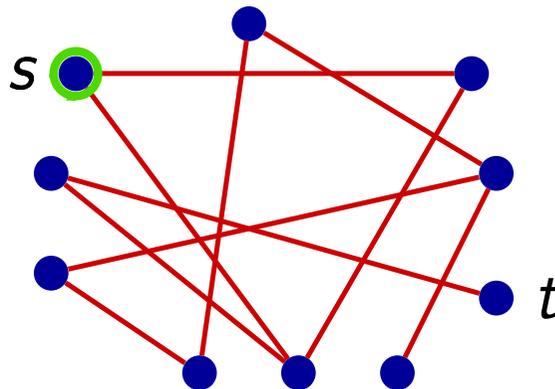
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

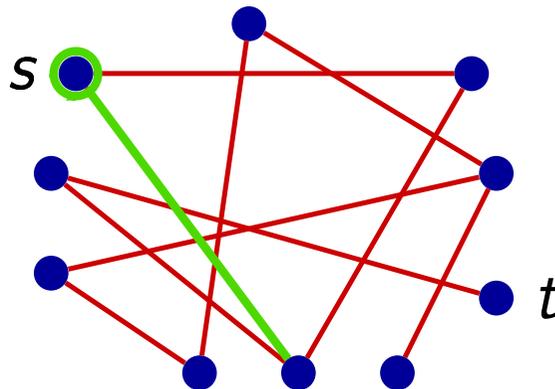
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

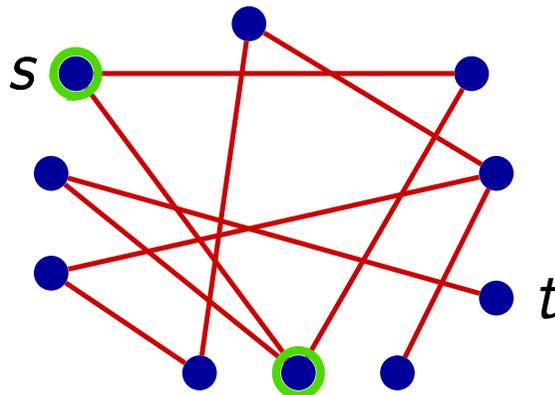
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

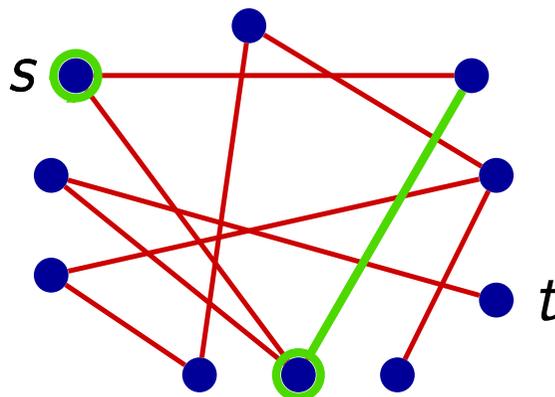
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

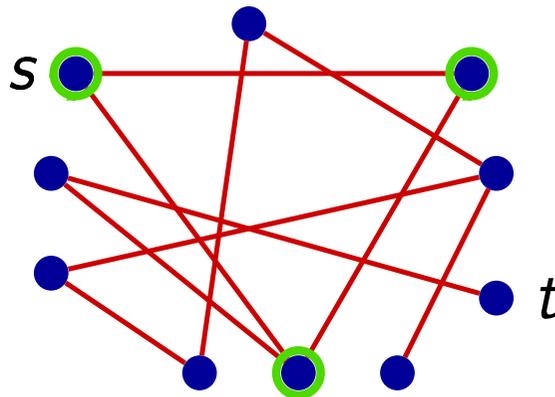
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

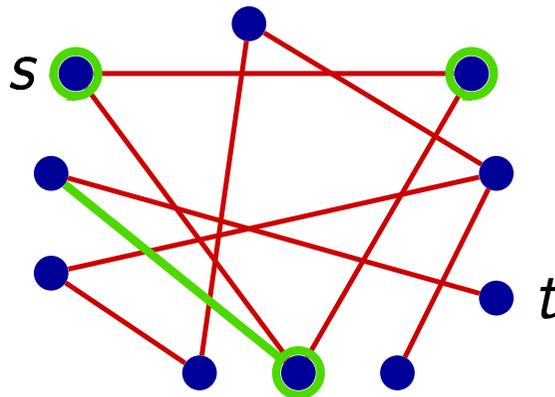
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

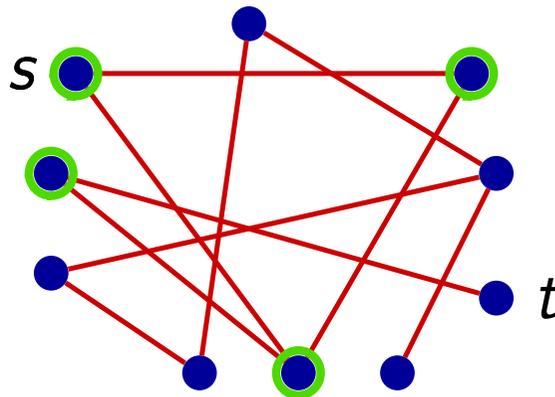
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

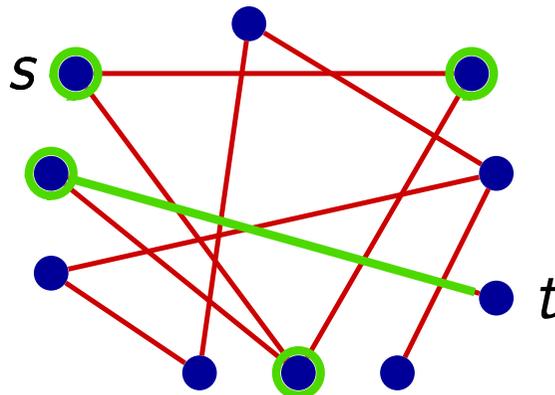
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

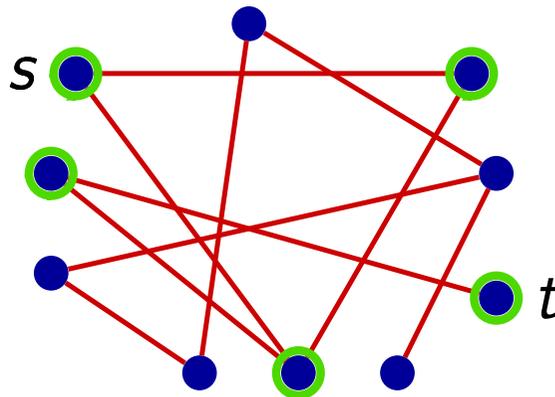
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

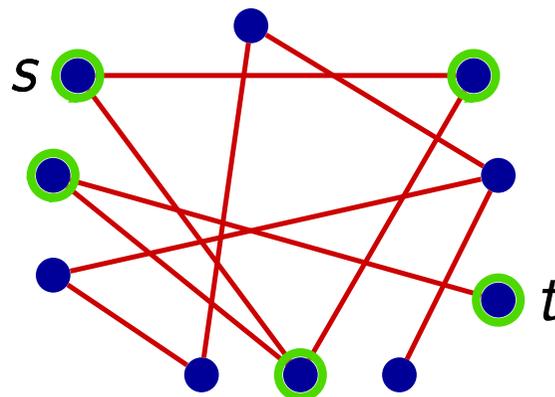
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# 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"

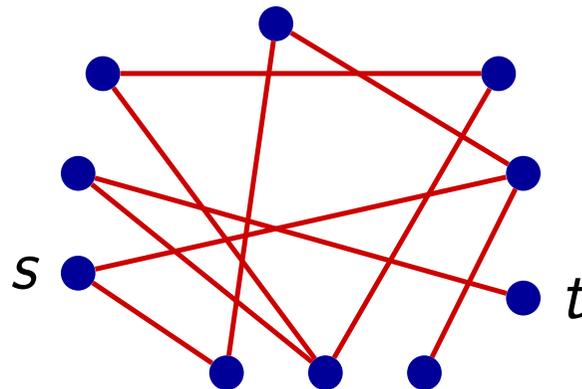


Vrai !

# Existence d'une chaîne entre $s$ et $t$

Algorithme :

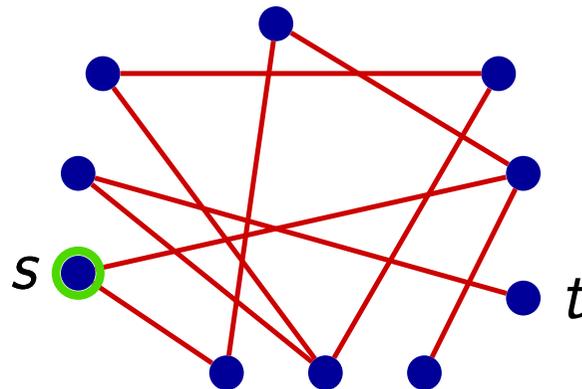
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

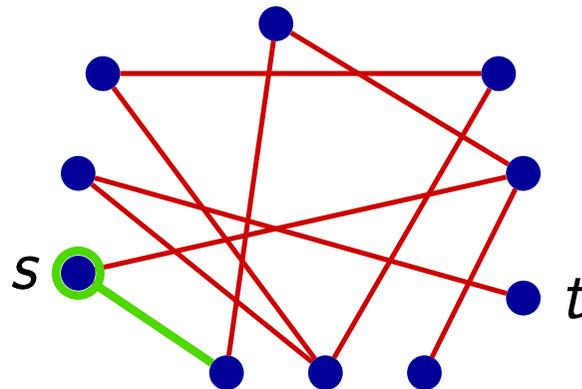
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

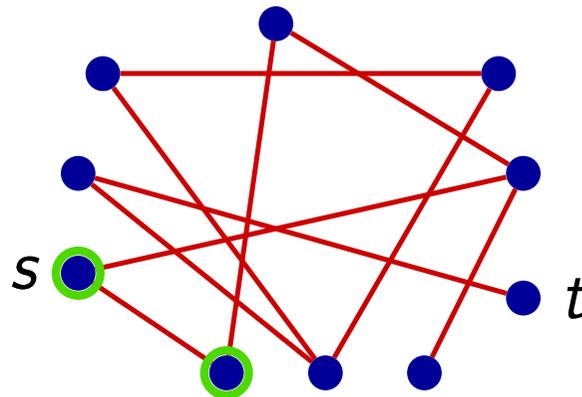
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

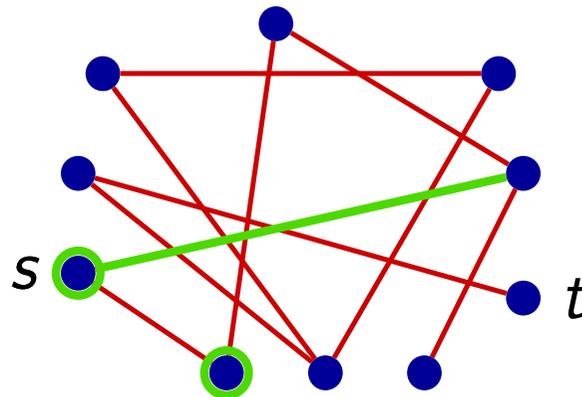
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

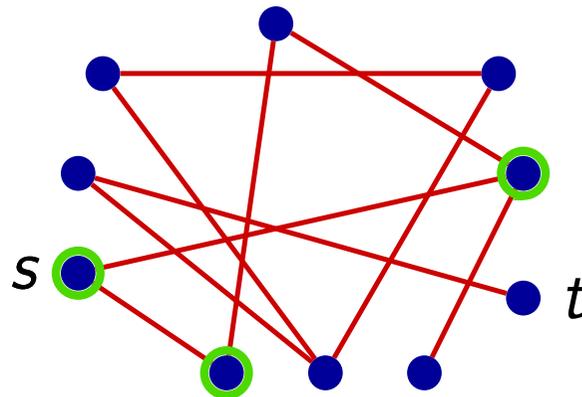
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

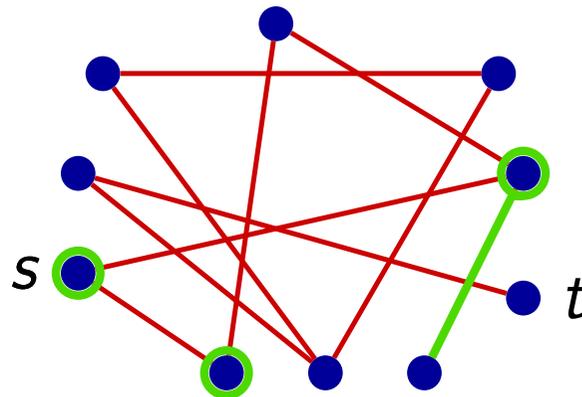
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

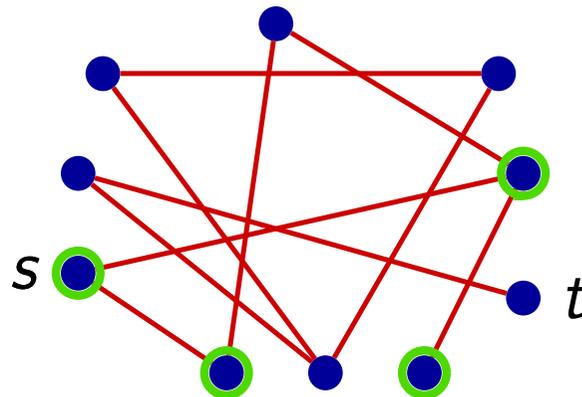
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

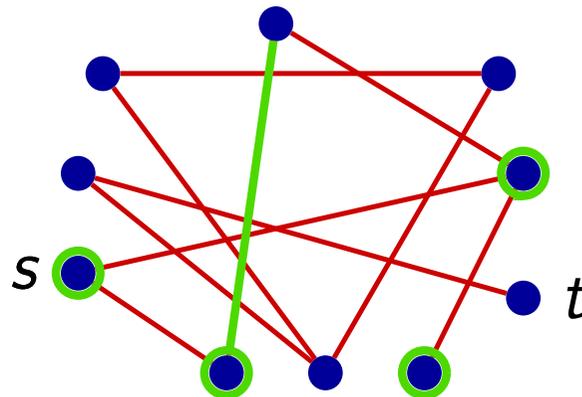
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

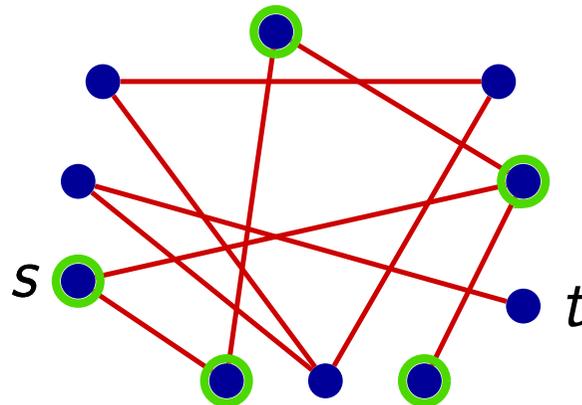
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# Existence d'une chaîne entre $s$ et $t$

Algorithme :

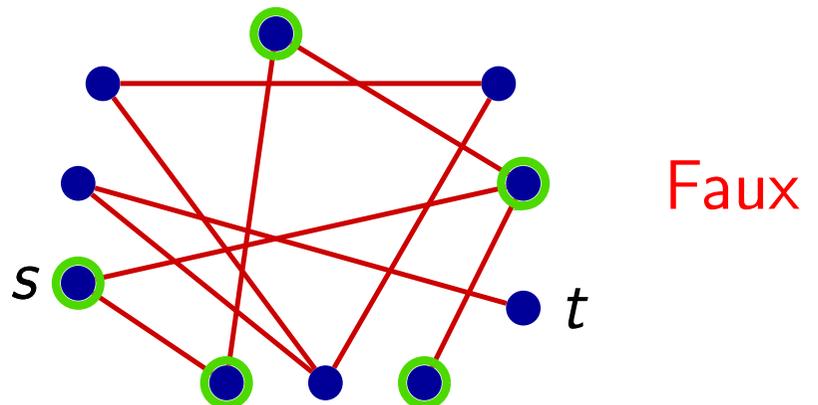
1. démarquer tous les sommets
2. marquer  $s$
3. tant que  $t$  n'est pas marqué,
  - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
  - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
  - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer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



# 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"



# Existence d'une chaîne, complexité

Avec la définition centrée sommets :

- ▶ **Étape 1** :  $|S|$  unités de temps.
- ▶ **Étape 2** : 1 unité de temps.
- ▶ **Étape 3.1** : au pire, pour chercher une arête « bicolore », on passe  $d(s) + 1$  unité de temps par sommet  $s$ , ce qui coûte  $\sum_{s \in S} (d(s) + 1) = 2|A| + |S| = O(|A| + |S|)$  unités de temps.
- ▶ **Étapes 3.2 et 3.3** : 1 unité de temps.
- ▶ Au pire, on recommence l'étape 3 pour chaque sommet :  $|S|$  fois.
- ▶ La complexité de l'algorithme est donc au pire de l'ordre de :

$$|S|(|A| + |S|).$$

(en négligeant les étapes 1, 2, 3.2, 3.3 et un facteur 2).

- ▶ **Remarque** Il existe un algorithme de complexité  $O(|A| + |S|)$ .

# Existence d'une chaîne, complexité

Avec la définition centrée sommets :

- ▶ Étape 1 :  $|S|$  unités de temps.
- ▶ Étape 2 : 1 unité de temps.
- ▶ Étape 3.1 : au pire, pour chercher une arête « bicolore », on passe  $d(s) + 1$  unité de temps par sommet  $s$ , ce qui coûte  $\sum_{s \in S} (d(s) + 1) = 2|A| + |S| = O(|A| + |S|)$  unités de temps.
- ▶ Étapes 3.2 et 3.3 : 1 unité de temps.
- ▶ Au pire, on recommence l'étape 3 pour chaque sommet :  $|S|$  fois.
- ▶ La complexité de l'algorithme est donc au pire de l'ordre de :

$$|S|(|A| + |S|).$$

(en négligeant les étapes 1, 2, 3.2, 3.3 et un facteur 2).

- ▶ **Remarque** Il existe un algorithme de complexité  $O(|A| + |S|)$ .

# Existence d'une chaîne, complexité

Avec la définition centrée sommets :

- ▶ Étape 1 :  $|S|$  unités de temps.
- ▶ Étape 2 : 1 unité de temps.
- ▶ Étape 3.1 : au pire, pour chercher une arête « bicolore », on passe  $d(s) + 1$  unité de temps par sommet  $s$ , ce qui coûte  $\sum_{s \in S} (d(s) + 1) = 2|A| + |S| = O(|A| + |S|)$  unités de temps.
- ▶ Étapes 3.2 et 3.3 : 1 unité de temps.
- ▶ Au pire, on recommence l'étape 3 pour chaque sommet :  $|S|$  fois.
- ▶ La complexité de l'algorithme est donc au pire de l'ordre de :

$$|S|(|A| + |S|).$$

(en négligeant les étapes 1, 2, 3.2, 3.3 et un facteur 2).

- ▶ **Remarque** Il existe un algorithme de complexité  $O(|A| + |S|)$ .

# Existence d'une chaîne, complexité

Avec la définition centrée sommets :

- ▶ Étape 1 :  $|S|$  unités de temps.
- ▶ Étape 2 : 1 unité de temps.
- ▶ Étape 3.1 : au pire, pour chercher une arête « bicolore », on passe  $d(s) + 1$  unité de temps par sommet  $s$ , ce qui coûte  $\sum_{s \in S} (d(s) + 1) = 2|A| + |S| = O(|A| + |S|)$  unités de temps.
- ▶ Étapes 3.2 et 3.3 : 1 unité de temps.
- ▶ Au pire, on recommence l'étape 3 pour chaque sommet :  $|S|$  fois.
- ▶ La complexité de l'algorithme est donc au pire de l'ordre de :

$$|S|(|A| + |S|).$$

(en négligeant les étapes 1, 2, 3.2, 3.3 et un facteur 2).

- ▶ **Remarque** Il existe un algorithme de complexité  $O(|A| + |S|)$ .

# Existence d'une chaîne, complexité

Avec la définition centrée sommets :

- ▶ Étape 1 :  $|S|$  unités de temps.
- ▶ Étape 2 : 1 unité de temps.
- ▶ Étape 3.1 : au pire, pour chercher une arête « bicolore », on passe  $d(s) + 1$  unité de temps par sommet  $s$ , ce qui coûte  $\sum_{s \in S} (d(s) + 1) = 2|A| + |S| = O(|A| + |S|)$  unités de temps.
- ▶ Étapes 3.2 et 3.3 : 1 unité de temps.
- ▶ Au pire, on recommence l'étape 3 pour chaque sommet :  $|S|$  fois.
- ▶ La complexité de l'algorithme est donc au pire de l'ordre de :

$$|S|(|A| + |S|).$$

(en négligeant les étapes 1, 2, 3.2, 3.3 et un facteur 2).

- ▶ **Remarque** Il existe un algorithme de complexité  $O(|A| + |S|)$ .

# Existence d'une chaîne, complexité

Avec la définition centrée sommets :

- ▶ Étape 1 :  $|S|$  unités de temps.
- ▶ Étape 2 : 1 unité de temps.
- ▶ Étape 3.1 : au pire, pour chercher une arête « bicolore », on passe  $d(s) + 1$  unité de temps par sommet  $s$ , ce qui coûte  $\sum_{s \in S} (d(s) + 1) = 2|A| + |S| = O(|A| + |S|)$  unités de temps.
- ▶ Étapes 3.2 et 3.3 : 1 unité de temps.
- ▶ Au pire, on recommence l'étape 3 pour chaque sommet :  $|S|$  fois.
- ▶ La complexité de l'algorithme est donc au pire de l'ordre de :

$$|S|(|A| + |S|).$$

(en négligeant les étapes 1, 2, 3.2, 3.3 et un facteur 2).

- ▶ Remarque Il existe un algorithme de complexité  $O(|A| + |S|)$ .

# Existence d'une chaîne, complexité

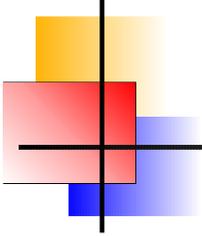
Avec la définition centrée sommets :

- ▶ Étape 1 :  $|S|$  unités de temps.
- ▶ Étape 2 : 1 unité de temps.
- ▶ Étape 3.1 : au pire, pour chercher une arête « bicolore », on passe  $d(s) + 1$  unité de temps par sommet  $s$ , ce qui coûte  $\sum_{s \in S} (d(s) + 1) = 2|A| + |S| = O(|A| + |S|)$  unités de temps.
- ▶ Étapes 3.2 et 3.3 : 1 unité de temps.
- ▶ Au pire, on recommence l'étape 3 pour chaque sommet :  $|S|$  fois.
- ▶ La complexité de l'algorithme est donc au pire de l'ordre de :

$$|S|(|A| + |S|).$$

(en négligeant les étapes 1, 2, 3.2, 3.3 et un facteur 2).

- ▶ Remarque Il existe un algorithme de complexité  $O(|A| + |S|)$ .



Dans cette partie...

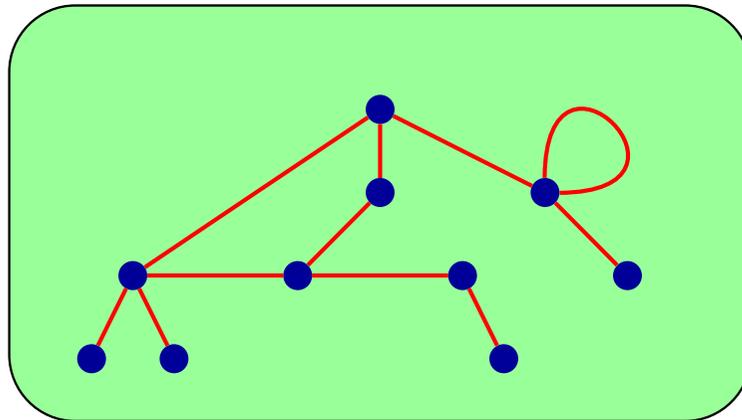
---

## 8– Connexité

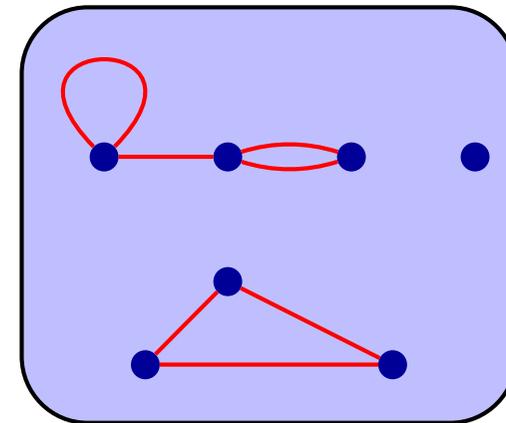
# Connexité

- ▶ 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



- ▶ 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 cherche des algorithmes pour déterminer si un graphe est connexe.

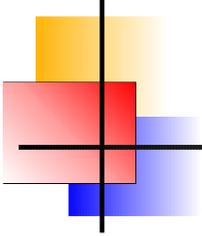
# Déterminer si un graphe est connexe

## Approche simple

- ▶ La première idée est toujours d'appliquer la définition.
- ▶ Donc, pour déterminer si un graphe est connexe, vérifier si pour chaque couple  $(s, t)$  de sommets, il existe une chaîne entre  $s$  et  $t$ .
- ▶ Il y a  $|S|^2$  tels couples.
- ▶ Au total, la méthode simple a une complexité

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

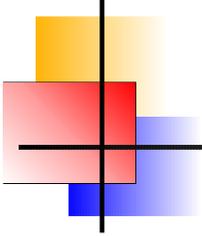
- ▶ Nous allons montrer qu'on peut résoudre le problème en remplaçant le facteur  $|S|^3$  par  $|S|^2$ .



# Différence entre $O(n^3)$ et $O(n)$

---

- ▶ Pour apprécier la différence entre  $O(n^3)$  et  $O(n)$ , imaginons un ordinateur capable d'exécuter une instruction élémentaire en  $10\text{ns} = 10^{-8}\text{s}$  (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 0,01s.
- ▶ Exécuter  $n^3$  instructions élémentaires nécessite  $10^{10}\text{s}$  soit  $\sim 321$  ans.



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

---

- ▶ Imaginons encore un ordinateur capable d'exécuter une instruction élémentaire en 10ns et un graphe de 1 000 000 sommets.
- ▶ Exécuter  $n$  instructions élémentaires nécessite 10ms, soit 0,01s.
- ▶ Exécuter  $n^2$  instructions élémentaires nécessite  $10^4$ s soit  $\sim 2\text{h}45$ .
- ▶ Exécuter  $n^3$  instructions élémentaires nécessite  $\sim 321$  ans.

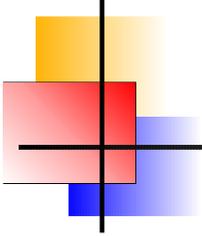
## Theorème

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 :

$$A \iff B \iff C.$$

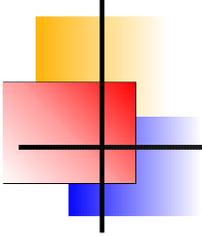


# Preuve (parties $A \implies B$ et $B \implies C$ )

---

- ▶ On suppose que  $A$  est vraie, c'est-à-dire que  $G$  est connexe.
- ▶ On choisit un sommet  $s$  de  $G$ .
- ▶ Comme  $G$  est connexe, pour tout sommet  $t$  de  $G$ , il existe une chaîne entre  $s$  et  $t$ . Donc  $B$  est vraie.
- ▶ On a donc montré que  $A \implies B$ .
- ▶ Si  $B$  est vraie, alors  $C$  est aussi vraie (évident) donc  $B \implies C$ .

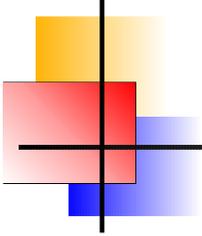
Il reste à montrer que  $C \implies A$ .



## Preuve (partie $C \implies A$ )

---

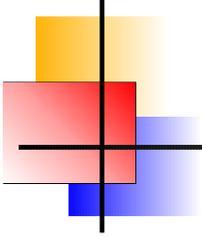
- ▶ On suppose que  $C$  est vraie : il existe donc un sommet  $s$  de  $G$  tel qu'il existe une chaîne entre  $s$  et chacun des sommets de  $G$ .
- ▶ Il faut prouver que  $A$  est vraie, c'est-à-dire que  $G$  est connexe.
- ▶ Soient donc  $x$  et  $y$  deux sommets de  $G$ . On veut montrer qu'il existe une chaîne entre  $x$  et  $y$ .



## Preuve (partie $C \implies A$ )

---

- ▶ On suppose que  $C$  est vraie : il existe donc un sommet  $s$  de  $G$  tel qu'il existe une chaîne entre  $s$  et chacun des sommets de  $G$ .
- ▶ Il faut prouver que  $A$  est vraie, c'est-à-dire que  $G$  est connexe.
- ▶ Soient donc  $x$  et  $y$  deux sommets de  $G$ . On veut montrer qu'il existe une chaîne entre  $x$  et  $y$ .



## Preuve (partie $C \implies A$ )

---

- ▶ On suppose que  $C$  est vraie : il existe donc un sommet  $s$  de  $G$  tel qu'il existe une chaîne entre  $s$  et chacun des sommets de  $G$ .
- ▶ Il faut prouver que  $A$  est vraie, c'est-à-dire que  $G$  est connexe.
- ▶ Soient donc  $x$  et  $y$  deux sommets de  $G$ . On veut montrer qu'il existe une chaîne entre  $x$  et  $y$ .

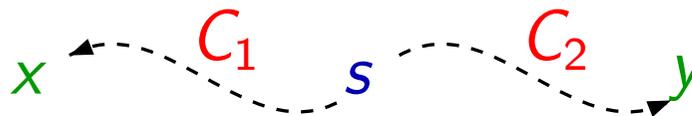
# Preuve (partie $C \implies A$ )

- ▶ On suppose que  $C$  est vraie : il existe donc un sommet  $s$  de  $G$  tel qu'il existe une chaîne entre  $s$  et chacun des sommets de  $G$ .
- ▶ Il faut prouver que  $A$  est vraie, c'est-à-dire que  $G$  est connexe.
- ▶ Soient donc  $x$  et  $y$  deux sommets de  $G$ . On veut montrer qu'il existe une chaîne entre  $x$  et  $y$ .

$x$                        $s$                        $y$

# Preuve (partie $C \implies A$ )

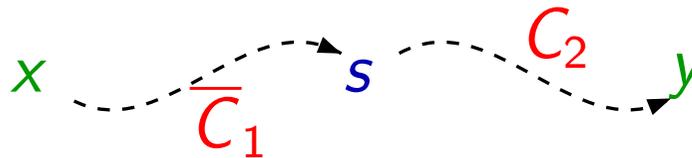
- ▶ On suppose que  $C$  est vraie : il existe donc un sommet  $s$  de  $G$  tel qu'il existe une chaîne entre  $s$  et chacun des sommets de  $G$ .
- ▶ Il faut prouver que  $A$  est vraie, c'est-à-dire que  $G$  est connexe.
- ▶ Soient donc  $x$  et  $y$  deux sommets de  $G$ . On veut montrer qu'il existe une chaîne entre  $x$  et  $y$ .



- ▶ On sait qu'il existe des chaînes  $C_1$  entre  $s$  et  $x$ , et  $C_2$  entre  $s$  et  $y$ .

# Preuve (partie $C \implies A$ )

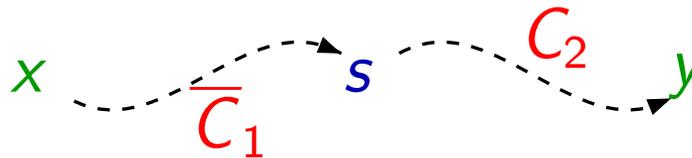
- ▶ On suppose que  $C$  est vraie : il existe donc un sommet  $s$  de  $G$  tel qu'il existe une chaîne entre  $s$  et chacun des sommets de  $G$ .
- ▶ Il faut prouver que  $A$  est vraie, c'est-à-dire que  $G$  est connexe.
- ▶ Soient donc  $x$  et  $y$  deux sommets de  $G$ . On veut montrer qu'il existe une chaîne entre  $x$  et  $y$ .



- ▶ On sait qu'il existe des chaînes  $C_1$  entre  $s$  et  $x$ , et  $C_2$  entre  $s$  et  $y$ .
- ▶ En lisant  $C_1$  à l'envers, on obtient une chaîne  $\overline{C_1}$  entre  $x$  et  $s$ .

# Preuve (partie $C \implies A$ )

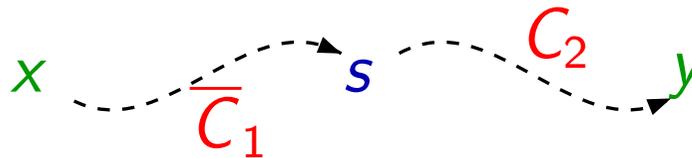
- ▶ On suppose que  $C$  est vraie : il existe donc un sommet  $s$  de  $G$  tel qu'il existe une chaîne entre  $s$  et chacun des sommets de  $G$ .
- ▶ Il faut prouver que  $A$  est vraie, c'est-à-dire que  $G$  est connexe.
- ▶ Soient donc  $x$  et  $y$  deux sommets de  $G$ . On veut montrer qu'il existe une chaîne entre  $x$  et  $y$ .



- ▶ On sait qu'il existe des chaînes  $C_1$  entre  $s$  et  $x$ , et  $C_2$  entre  $s$  et  $y$ .
- ▶ En lisant  $C_1$  à l'envers, on obtient une chaîne  $\bar{C}_1$  entre  $x$  et  $s$ .
- ▶ En recollant la chaîne  $\bar{C}_1$  à  $C_2$ , on obtient une chaîne  $x$  et  $y$ .

# Preuve (partie $C \implies A$ )

- ▶ On suppose que  $C$  est vraie : il existe donc un sommet  $s$  de  $G$  tel qu'il existe une chaîne entre  $s$  et chacun des sommets de  $G$ .
- ▶ Il faut prouver que  $A$  est vraie, c'est-à-dire que  $G$  est connexe.
- ▶ Soient donc  $x$  et  $y$  deux sommets de  $G$ . On veut montrer qu'il existe une chaîne entre  $x$  et  $y$ .



- ▶ On sait qu'il existe des chaînes  $C_1$  entre  $s$  et  $x$ , et  $C_2$  entre  $s$  et  $y$ .
- ▶ En lisant  $C_1$  à l'envers, on obtient une chaîne  $\bar{C}_1$  entre  $x$  et  $s$ .
- ▶ En recollant la chaîne  $\bar{C}_1$  à  $C_2$ , on obtient une chaîne  $x$  et  $y$ .
- ▶ Donc  $A$  est vraie et  $C \implies A$ .

Pour déterminer si un graphe est connexe, on peut donc :

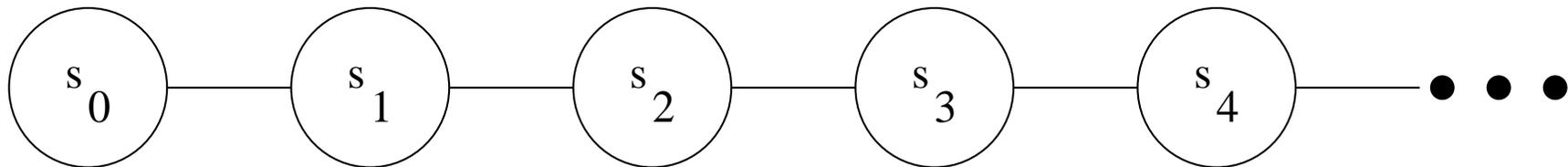
- ▶ Choisir un sommet  $s$  **arbitraire** dans le graphe.
- ▶ Vérifier si pour **chaque sommet**  $t$  du graphe, il existe une chaîne entre  $s$  et  $t$ .

## Complexité

1. Il faut donc faire  $|S|$  vérifications.
2. Le coût de chaque vérification est  $|S|. (|A| + |S|)$ .
3. Au total, la complexité est donc  $|S|^2. (|A| + |S|)$ .

# Encore une amélioration

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



il va parcourir 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  $s_2$ , refaisant alors le parcours entre  $s_0$  et  $s_1$ , *etc.*

Voici l'algorithme plus efficace.

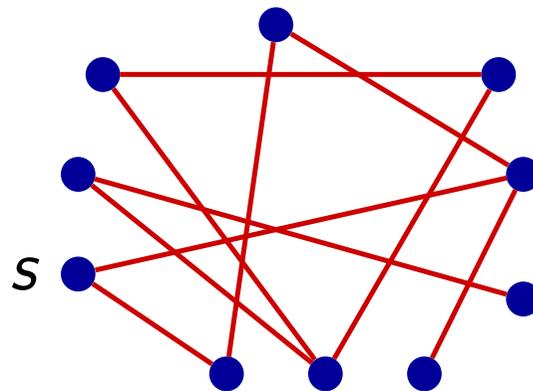
1. démarquer tous les sommets
2. marquer un sommet arbitraire  $s$
3. tant qu'il existe une arête « bicolore » :
  - 3.1 marquer l'extrémité non encore marquée
4. si tout les sommets sont marqués, renvoyer "vrai"
5. sinon renvoyer "faux"

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

**Remarque** On peut faire encore mieux :  $O(|S| + |A|)$ .

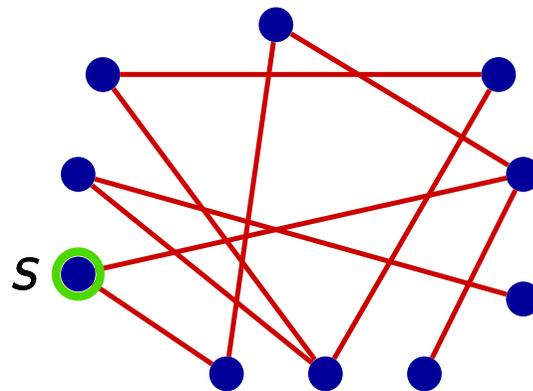
# Méthode finale pour la connexité - Exemple

L'algorithme ressemble à celui de recherche d'une chaîne.  
Déroulement possible de l'algorithme sur le graphe suivant :



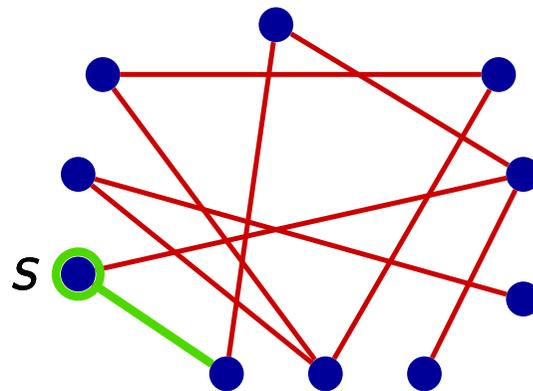
# Méthode finale pour la connexité - Exemple

L'algorithme ressemble à celui de recherche d'une chaîne.  
Déroulement possible de l'algorithme sur le graphe suivant :



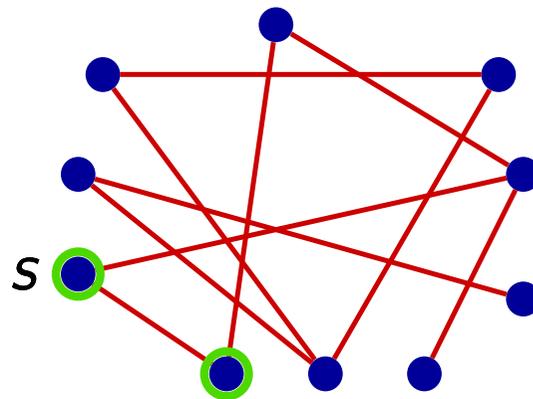
# Méthode finale pour la connexité - Exemple

L'algorithme ressemble à celui de recherche d'une chaîne.  
Déroulement possible de l'algorithme sur le graphe suivant :



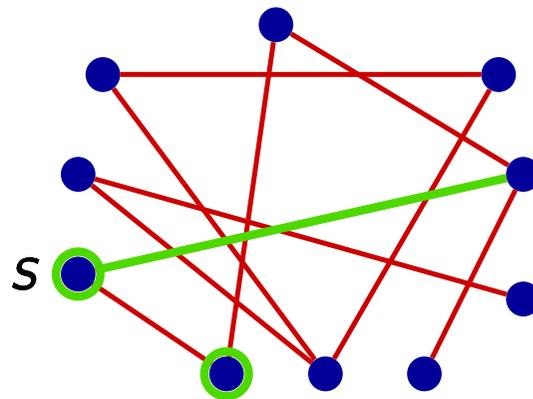
# Méthode finale pour la connexité - Exemple

L'algorithme ressemble à celui de recherche d'une chaîne.  
Déroulement possible de l'algorithme sur le graphe suivant :



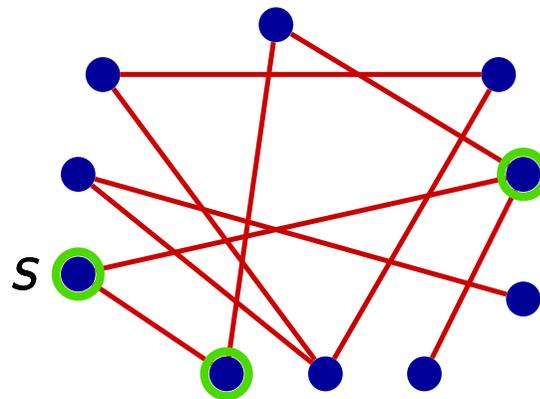
# Méthode finale pour la connexité - Exemple

L'algorithme ressemble à celui de recherche d'une chaîne.  
Déroulement possible de l'algorithme sur le graphe suivant :



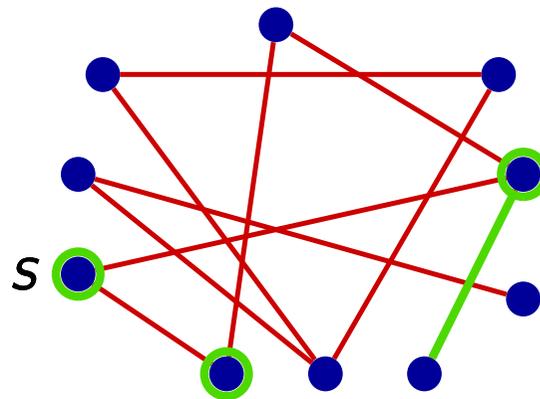
# Méthode finale pour la connexité - Exemple

L'algorithme ressemble à celui de recherche d'une chaîne.  
Déroulement possible de l'algorithme sur le graphe suivant :



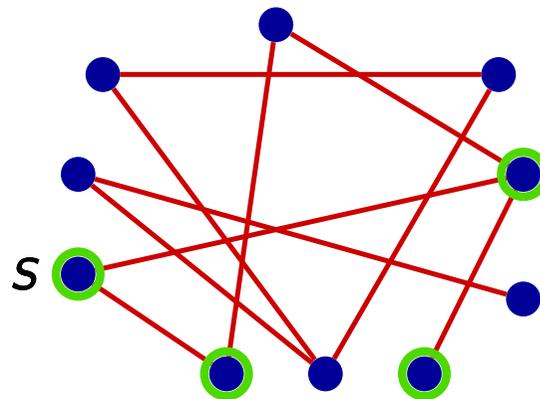
# Méthode finale pour la connexité - Exemple

L'algorithme ressemble à celui de recherche d'une chaîne.  
Déroulement possible de l'algorithme sur le graphe suivant :



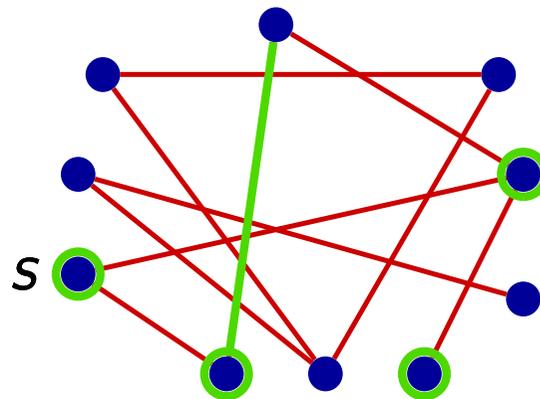
# Méthode finale pour la connexité - Exemple

L'algorithme ressemble à celui de recherche d'une chaîne.  
Déroulement possible de l'algorithme sur le graphe suivant :



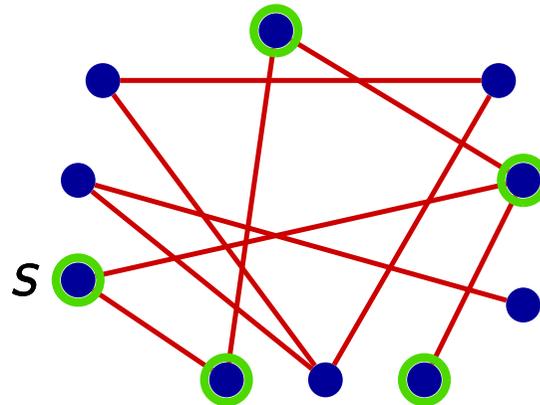
# Méthode finale pour la connexité - Exemple

L'algorithme ressemble à celui de recherche d'une chaîne.  
Déroulement possible de l'algorithme sur le graphe suivant :



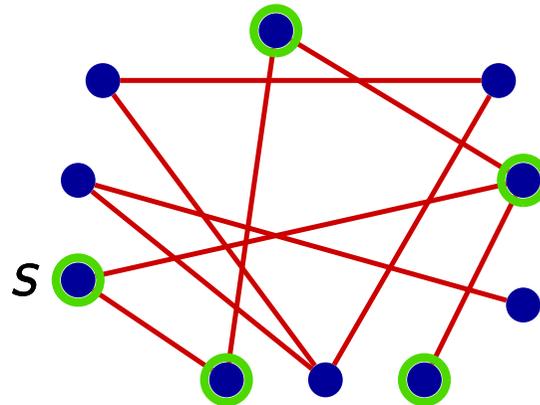
# Méthode finale pour la connexité - Exemple

L'algorithme ressemble à celui de recherche d'une chaîne.  
Déroulement possible de l'algorithme sur le graphe suivant :

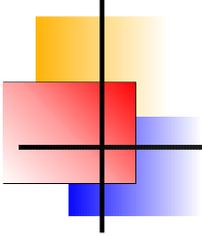


# Méthode finale pour la connexité - Exemple

L'algorithme ressemble à celui de recherche d'une chaîne.  
Déroulement possible de l'algorithme sur le graphe suivant :



Non connexe (sommets pas tous marqués, plus d'arêtes « bicolores »).



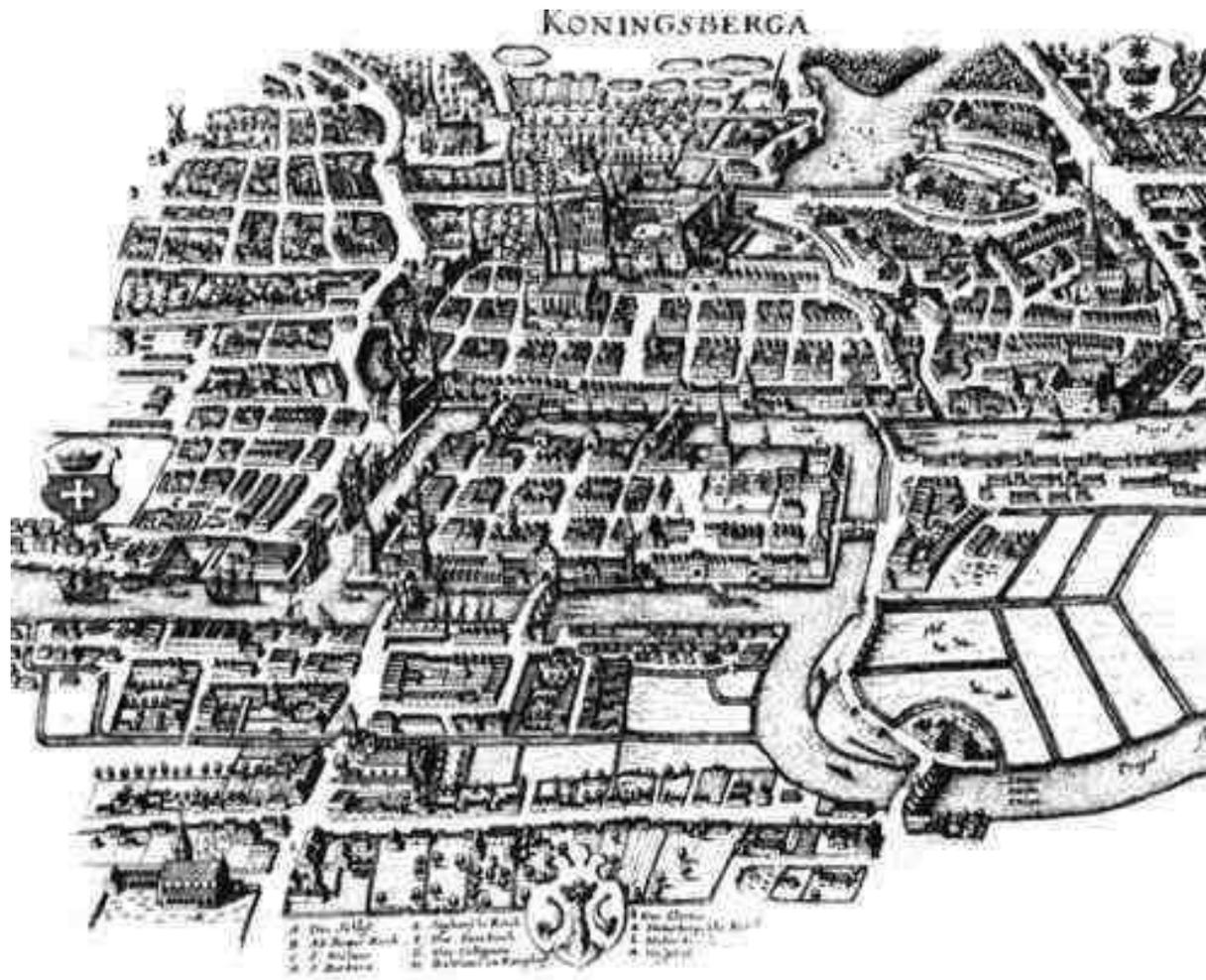
Dans cette partie...

---

## 9– Graphes Eulériens

# Graphes Eulériens

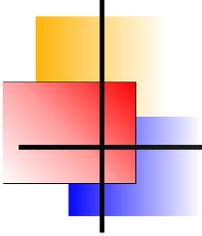
Les ponts de la ville de Königsberg au 18<sup>e</sup> siècle  
(maintenant Kaliningrad en Russie) :



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").



# C'est un problème de graphe

---

Dans un graphe  $G$ , est-il possible de trouver une chaîne

$$C = s_1, a_1, s_2, a_2, \dots, s_m, a_m, s_{m+1}$$

telle que

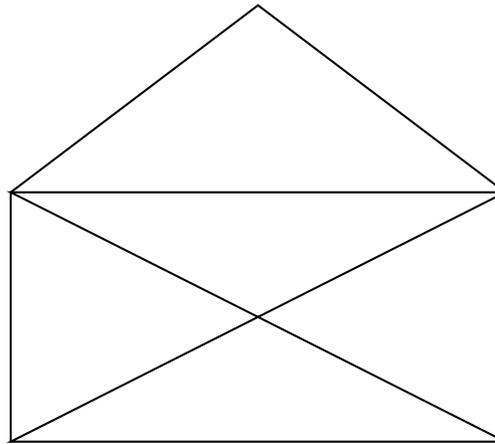
$$A(G) = \{a_1, a_2, \dots, a_m\}$$

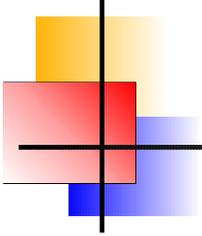
et

$$|A(G)| = m.$$

# Dessiner une enveloppe

Est-ce possible de dessiner cette enveloppe sans lever le crayon :





# Définition de graphe eulérien

Un graphe  $G$  est *eulérien* si et seulement si il est possible de trouver une chaîne

$$C = s_1, a_1, s_2, a_2, \dots, s_m, a_m, s_{m+1}$$

telle que

$$S(G) = \{s_1, s_2, \dots, s_m, s_{m+1}\},$$

$$A(G) = \{a_1, a_2, \dots, a_m\}$$

$$\text{et } |A(G)| = m.$$

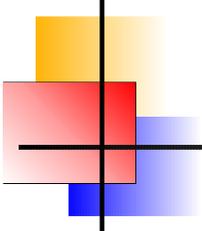
**Remarque 1** Dans la suite de sommets  $s_1, s_2, \dots, s_{m+1}$ , certains sommets peuvent apparaître plusieurs fois.

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

**Théorème.** *Un graphe eulérien est forcément 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.



# Preuve du théorème

Soit  $G$  un graphe eulérien. Donc, il existe une chaîne

$$C = s_1, a_1, s_2, a_2, \dots, s_m, a_m, s_{m+1}$$

telle que

$$S(G) = \{s_1, s_2, \dots, s_m, s_{m+1}\},$$

$$A(G) = \{a_1, a_2, \dots, a_m\}$$

$$\text{et } |A(G)| = m.$$

- ▶ En arrêtant  $C$  au sommet  $s_i$ , on obtient une chaîne entre  $s_1$  et  $s_i$ .
- ▶ Comme  $S(G) = \{s_1, s_2, \dots, s_m, s_{m+1}\}$ , il existe une chaîne entre  $s$  et chaque sommet de  $G$ .

Donc  $G$  est connexe, par le point  $C$  du théorème de connexité.

## Définition

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

**Théorème :** *Dans un graphe eulérien, le nombre de sommets impairs est forcément 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 constater que son graphe est eulérien.

Soit  $G$  un graphe eulérien. Il existe donc une chaîne

$$C = s_1, a_1, s_2, a_2, \dots, s_m, a_m, s_{m+1}$$

telle que

$$S(G) = \{s_1, s_2, \dots, s_m, s_{m+1}\},$$

$$A(G) = \{a_1, a_2, \dots, a_m\}$$

$$\text{et } |A(G)| = m.$$

$$C = s_1, a_1, s_2, a_2, \dots, s_m, a_m, s_{m+1}$$

- ▶ Examinons un sommet autre que  $s_1$  et  $s_{m+1}$  dans cette chaîne.
- ▶ Un tel sommet  $s_i$  est entouré dans  $C$  de  $a_{i-1}$  et  $a_i$  ce qui donne une contribution de 2 au degré de  $s_i$ .
- ▶ Comme  $C$  contient chaque arête exactement une fois,  $s_i$  est pair.
- ▶ Pour  $s_1$  et  $s_{m+1}$  il existe deux cas :
  - ▶ **Premier cas** :  $s_1 \neq s_{m+1}$   
 $s_1$  et  $s_{m+1}$  sont impairs. Il y a donc 2 sommets impairs.
  - ▶ **Deuxième cas** :  $s_1 = s_{m+1}$   
 $s_1 = s_{m+1}$  est pair. Il y a donc 0 sommets impairs.

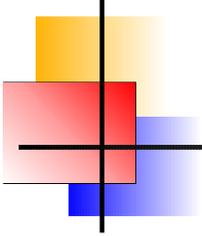
Cette preuve nous indique comment trouver une chaîne eulérienne dans un graphe ayant deux sommets impairs : il faut

- ▶ commencer dans un sommet impair, et
- ▶ terminer dans l'autre sommet impair.

**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*.

- ▶ Un *cycle* est une chaîne  $C = s_1, a_1, s_2, \dots, s_m, a_m, s_{m+1}$  telle que  $|\{a_1, \dots, a_m\}| = m$  et  $s_1 = s_{m+1}$ .
- ▶ **Remarque.** Si  $s_1, a_1, s_2, \dots, s_m, a_m, s_{m+1}$  est un cycle, alors  $s_2, a_2, s_3, \dots, a_m, s_1, a_2, s_2$  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  $C$  possède une arête  $a$  qui n'appartient pas à  $C$ , ajouter  $a$  et son sommet extrémité à  $C$ .

*(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  $C$ .

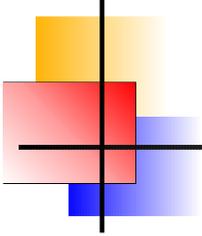
Former  $C' = s_i, a_i, s_{i+1}, \dots, a_m, s_1, a_1, \dots, a_{i-1}, s_i$ .

5. Poser  $C = C'$ , puis continuer à l'étape 2.

## Cas 2 : $G$ est connexe avec deux sommets imp

**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  $a = \{s, t\}$ , 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. Calculer le cycle  $C'$  pour  $G'$  par l'Algorithme 1.  
( $C'$  est un cycle eulérien pour  $G'$ .)  
On suppose que  $C'$  contient la sous-chaîne  $s, a, t$  dans ce sens (sinon inverser le rôle de  $s$  et  $t$ ).
3. Former  $C'' = s, a, t, a_i, s_{i+1}, \dots, s$ .
4. Retourner  $C = t, a_i, s_{i+1}, \dots, s$ .



Dans cette partie...

---

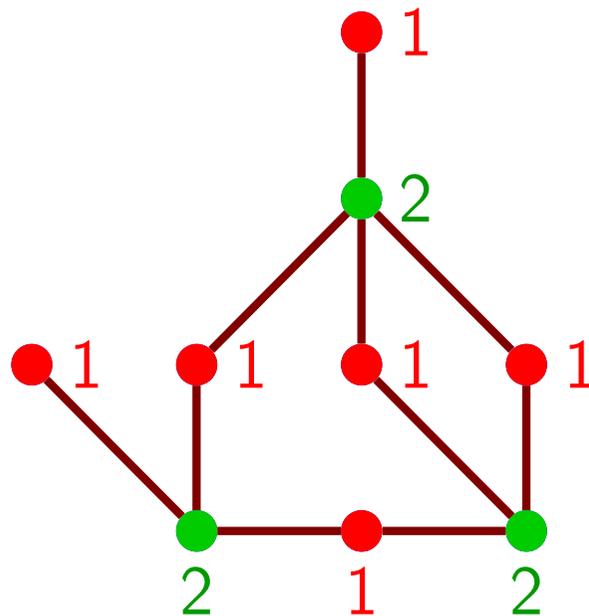
## 10– Coloration



# Coloration

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

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

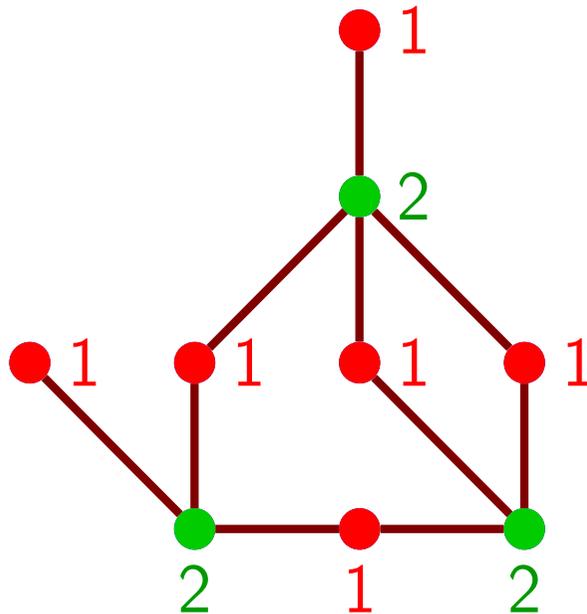


2-coloriable

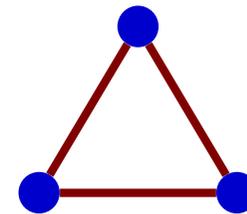
# Coloration

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

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



2-coloriable



Non 2-coloriable

Soit  $C = s_1, a_1, \dots, a_m, s_{m+1}$  une chaîne.

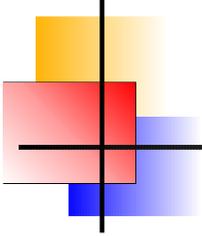
- ▶ La **longueur** de  $C$  est son nombre  $m$  d'arêtes.
- ▶  $C$  est un **cycle** si les arêtes  $a_1, \dots, a_m$  sont toutes différentes, et si  $s_1 = s_{m+1}$ .  
(la chaîne  $C = s_1$  est donc un cycle de longueur 0).

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

**Etape 1 :** si  $G$  contient un cycle de longueur impaire, alors on ne peut pas le colorier avec 2 couleurs.

**Etape 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  $G$ .

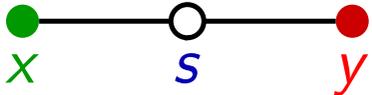


# Algorithme de 2-coloration

---

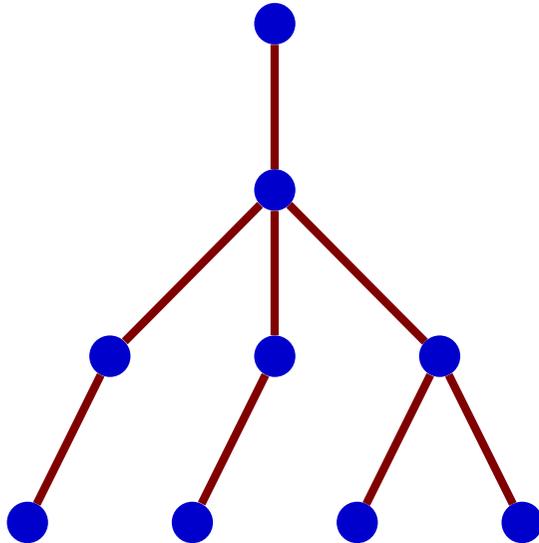
- ▶ On essaie de colorier en deux couleurs, ou bien de trouver un cycle de longueur impaire.
  - ▶ Quitte à travailler indépendamment sur chaque partie connexe du graphe, on peut supposer  $G$  connexe.
1. Colorier en couleur **1** un sommet arbitraire  $s_0$
  2. tant qu'il existe un sommet  $s$  non 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 une couleur différente, 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 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 ? On a détecté :
  - (a) une chaîne  $C_1$  de  $s_0$  à  $x$  de sommets coloriés,
  - (b) une chaîne  $C_2$  de  $s_0$  à  $y$  de sommets coloriés,
  - (c) des arêtes 
- ▶ Par (a) et (b), on a une chaîne simple entre  $x$  et  $y$ , de longueur impaire car  $x$  et  $y$  ont des couleurs différentes. En fermant cette chaîne par les arêtes (c), on obtient un cycle de longueur impaire.

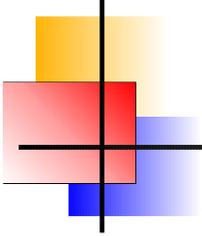
**Définition.** *Un arbre est un graphe connexe sans cycle.*

Exemple :



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

**Preuve.** Il n'y a aucun cycle, donc pas de cycle de longueur impaire !



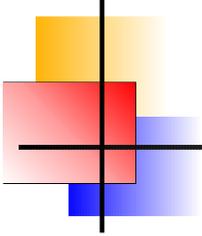
# Colorier un graphe en 3 couleurs ?

---

**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*.

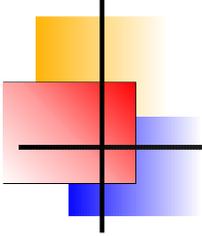
*Étant donné un graphe  $G$ , donne une coloration pour  $G$*

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  $O(|S(G)| + |A(G)|)$ .



# Combien de couleurs sont utilisées ?

---

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.

# Heuristique sur arbres et cycles

## ▶ Cycles

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

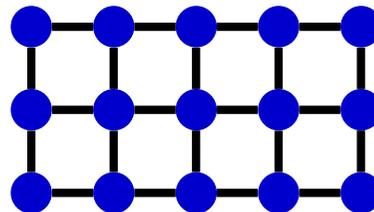
## ▶ Arbres (nouvelle preuve qu'ils sont 2-coloriables)

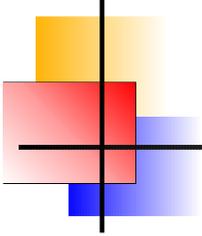
- ▶ Dans un arbre, il y a un sommet de degré 1 (pourquoi?).
- ▶ Si on enlève ce sommet, on obtient à nouveau un arbre.
- ▶ Donc chaque sommet enlevé par l'heuristique a un degré 1.
- ▶ Donc l'heuristique utilise 2 couleurs au plus.

## ▶ Graphes 2-coloriables.

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

## ▶ **Exercice.** Si $G$ est une grille, DEGMIN utilise 1 ou 2 couleurs.





# Une propriété importante de DEGMIN

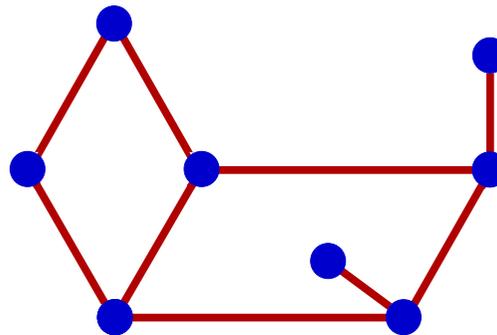
**Proposition.** *Si à chaque étape le sommet  $s$  choisi a un degré  $\leq d$ , alors DEGMIN utilisera au plus  $d + 1$  couleurs pour colorier le graphe.*

**Preuve.** Par induction sur le nombre de sommets : s'il y a 1 sommet, alors c'est vrai,  $d = 0$ . Supposons que c'est vrai pour tout graphe à  $n$  sommets, et montrons le résultat pour un graphe  $G$  à  $n + 1$  sommets.

- ▶ Dans  $G$  il existe un sommet  $s$  de degré  $\leq d$ .
- ▶ Le graphe  $G'$  obtenu en supprimant  $s$  de  $G$  a  $n$  sommets.
- ▶ Par induction, DEGMIN l'a colorié avec au plus  $d + 1$  couleurs.
- ▶ Une couleur de  $\{1, \dots, d + 1\}$  est inutilisée sur les  $d$  voisins de  $s$ .
- ▶ C'est une telle couleur que choisit DEGMIN pour  $s$ , donc tous les sommets sont coloriés avec  $\leq d + 1$  couleurs. **CQFD**

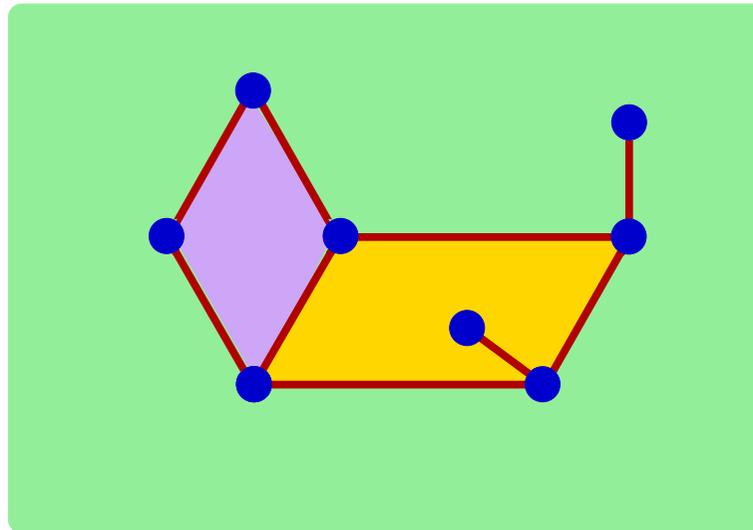
# Graphes planaires : théorème des 6 couleurs

- ▶ 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.



# Graphes planaires : théorème des 6 couleurs

- ▶ 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.



# Euler frappe encore

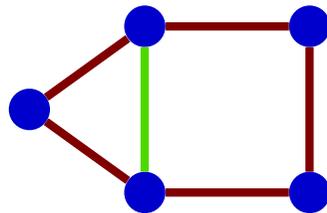
**Théorème.** Soit  $G$  un graphe planaire connexe,  $n$  son nombre de sommets,  $e$  son nombre d'arêtes et  $f$  son nombre de faces. Alors

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

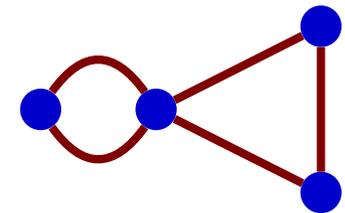
**Preuve.** Induction sur le nombre de sommets. Si  $n = 1$ , toutes les arêtes sont des boucles, chacune détermine une face en plus de la face infinie. Donc  $f = e + 1$ , et  $n - e + f = 2$ .

Si  $n \geq 2$ , comme  $G$  est connexe, il y a une arête entre deux sommets différents. On la « contracte » et on obtient un graphe  $G'$  avec  $n' = n - 1$  sommets,  $e' = e - 1$  arêtes et  $f' = f$  faces. Par induction,  $n' - e' + f' = 2$ , donc  $(n - 1) - (e - 1) + f = 2$ , soit  $n - e + f = 2$ .

Avant contraction

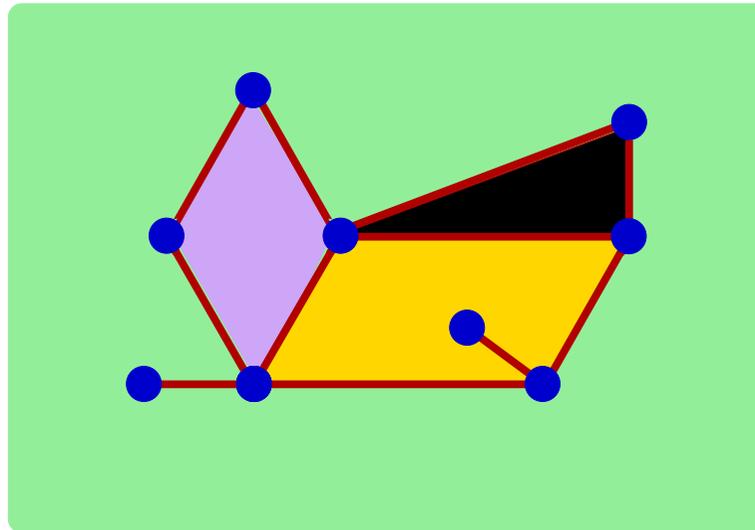


Après contraction

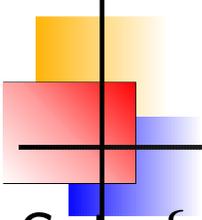


# Longueur des faces

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).



# Densité d'arêtes des graphes planaires

Soit  $\{f_1, \dots, f_k\}$  l'ensemble des faces du graphe planaire connexe  $G$  et  $\ell(f_i)$  la longueur de la face  $f_i$ . Soit  $f = k$  le nombre de faces de  $G$ .

**Théorème.** Soit  $G$  un graphe planaire connexe. On a

$$\ell(f_1) + \dots + \ell(f_k) = 2|A(G)| \quad (1)$$

**Preuve.** Analogue au 1er théorème sur les degrés.

# Densité d'arêtes des graphes planaires

Soit  $\{f_1, \dots, f_k\}$  l'ensemble des faces du graphe planaire connexe  $G$  et  $\ell(f_i)$  la longueur de la face  $f_i$ . Soit  $f = k$  le nombre de faces de  $G$ .

**Théorème.** Soit  $G$  un graphe planaire connexe. On a

$$\ell(f_1) + \dots + \ell(f_k) = 2|A(G)| \quad (1)$$

**Preuve.** Analogue au 1er théorème sur les degrés.

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

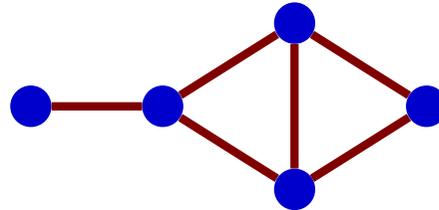
**Preuve.** On peut supposer  $G$  connexe, quitte à ajouter des arêtes. Comme  $n \geq 3$ , et que  $G$  est simple (pas de boucle ni arête multiple) chaque face a une longueur au moins 3. Donc, par (1) :  $2e \geq 3f$ .  
Donc  $3n - e = 3n - 3e + 2e \geq 3n - 3e + 3f = 3(n - e + f) = 6$   
(car  $n - e + f = 2$  par la formule d'Euler), donc  $e \leq 3n - 6$ .

Si  $d(s) \geq 6$  pour tout  $s$ , on aurait  $2e = \sum_{s \in S(G)} d(s) \geq 6n$ , impossible.

# Graphes Eulériens et graphes 2-coloriables

Le graphe **dual** d'un graphe  $G$  planaire connexe est le graphe  $G^*$  défini comme suit :

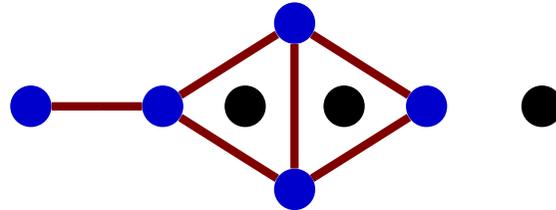
- ▶ On place un sommet de  $G^*$  sur chaque face de  $G$ .
- ▶ Pour chaque arête qui sépare deux faces  $f_i$  et  $f_j$  de  $G$ , on met une arête entre les sommets correspondants dans  $G^*$ .



# Graphes Eulériens et graphes 2-coloriables

Le graphe **dual** d'un graphe  $G$  planaire connexe est le graphe  $G^*$  défini comme suit :

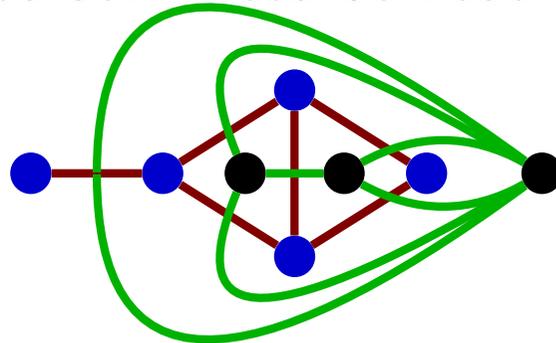
- ▶ On place un sommet de  $G^*$  sur chaque face de  $G$ .
- ▶ Pour chaque arête qui sépare deux faces  $f_i$  et  $f_j$  de  $G$ , on met une arête entre les sommets correspondants dans  $G^*$ .



# Graphes Eulériens et graphes 2-coloriables

Le graphe **dual** d'un graphe  $G$  planaire connexe est le graphe  $G^*$  défini comme suit :

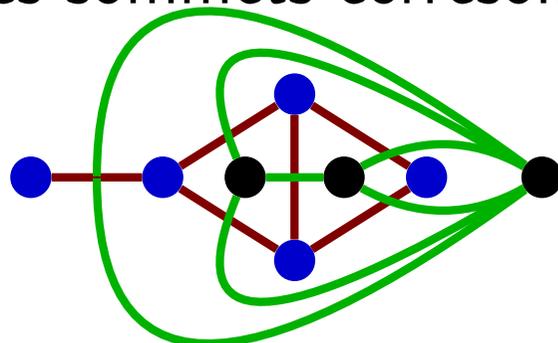
- ▶ On place un sommet de  $G^*$  sur chaque face de  $G$ .
- ▶ Pour chaque arête qui sépare deux faces  $f_i$  et  $f_j$  de  $G$ , on met une arête entre les sommets correspondants dans  $G^*$ .



# Graphes Eulériens et graphes 2-coloriables

Le graphe **dual** d'un graphe  $G$  planaire connexe est le graphe  $G^*$  défini comme suit :

- ▶ On place un sommet de  $G^*$  sur chaque face de  $G$ .
- ▶ Pour chaque arête qui sépare deux faces  $f_i$  et  $f_j$  de  $G$ , on met une arête entre les sommets correspondants dans  $G^*$ .



**Propriété.** Si  $G$  est un graphe planaire connexe :  $(G^*)^* = G$ .

**Théorème.** Les conditions suivantes sont équivalentes pour un graphe planaire connexe  $G$ .

- $G$  est 2-coloriable.
- $G^*$  n'a aucun sommet de degré impair.
- $G^*$  a un **cycle** Eulérien.

- ▶ Cette introduction à l'informatique a eu pour objectif d'en donner une idée fidèle, en insistant sur
  - ▶ les aspects algorithmiques, théoriques ou reliés à la recherche,
  - ▶ les applications pratiques dans le domaine de la programmation.
- ▶ Les notions de ce cours, graphes, algorithmes, programmes, et complexité interviennent partout en informatique et au-delà, mais...
- ▶ Ce n'est qu'un avant-goût de ce que vous rencontrerez si vous choisissez des études en informatique.

Bonne continuation !