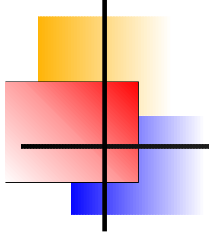
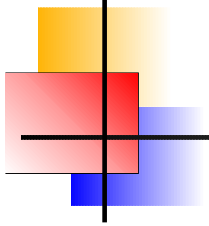
Initiation à l'informatique (MIS102)

Université Bordeaux 1

Année 2006-2007, Licence semestre 1



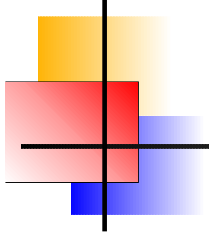
Plan du cours



Dans cette partie...

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.

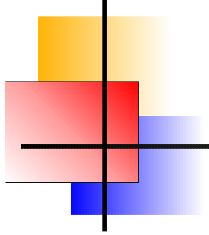
Thème : Étude d'un objet appelé *graphe*.

Contenu : Théorie des graphes (cours).

Algorithmique des graphes (TD).

Programmation des algorithmes de graphes (TP).

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



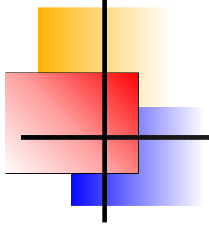
Faut-il des connaissances préalables ?

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.

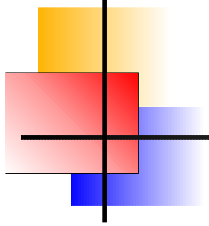


Organisation et site web

Responsables et cours : Cyril Gavoille
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.



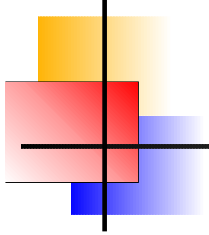
Support de cours

- Livre ($\sim$ 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 individuel noté	1h20	0,15
Examen	1h30	0,60



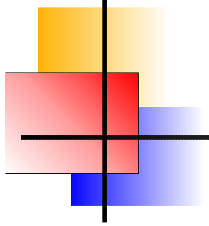
Comptes sur machines - Tutorat

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)



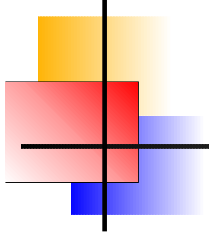
Dans cette partie...

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.*



L'informatique pour non informaticiens

- 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,
- ...

Exemples de domaines en informatique (3)

Les logiciels

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

Exemples de domaines en informatique (4)

Les langages de programmation

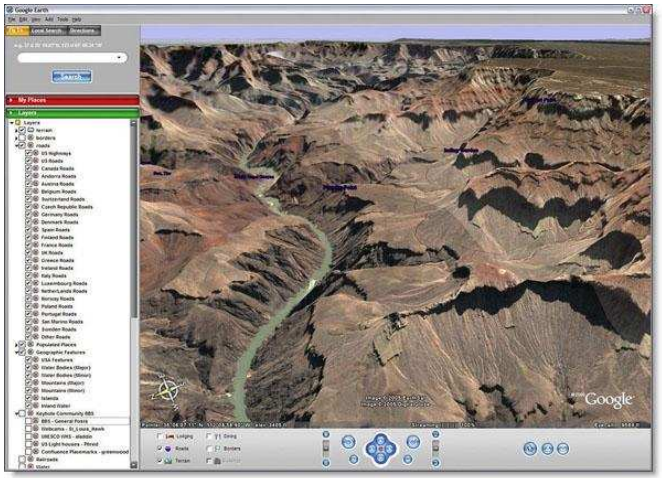
Les langages de programmation sont souvent utilisés dans des domaines spécifiques.

- 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 Google™ France !

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...

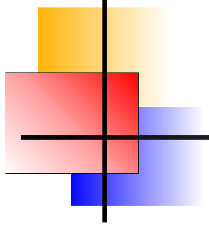


Enseignements à Bordeaux 1

- 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.*



Dans cette partie...

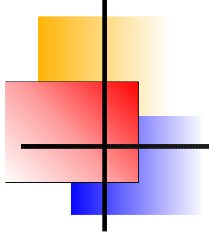
3– Algorithmes et programmes

- 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
```



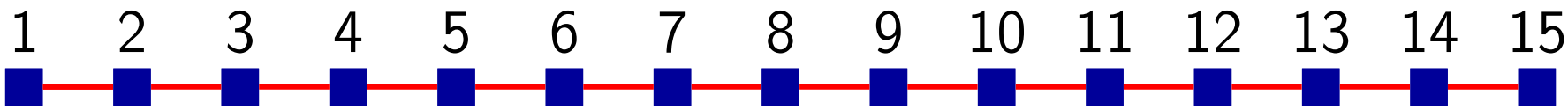
Importance de l'algorithmique

- 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.

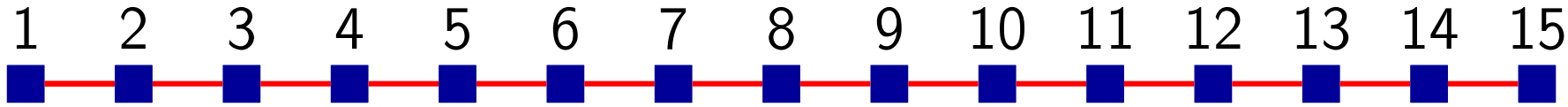
Organisation 1 : linéaire. Numéros croissants. Pizzeria au numéro 1.



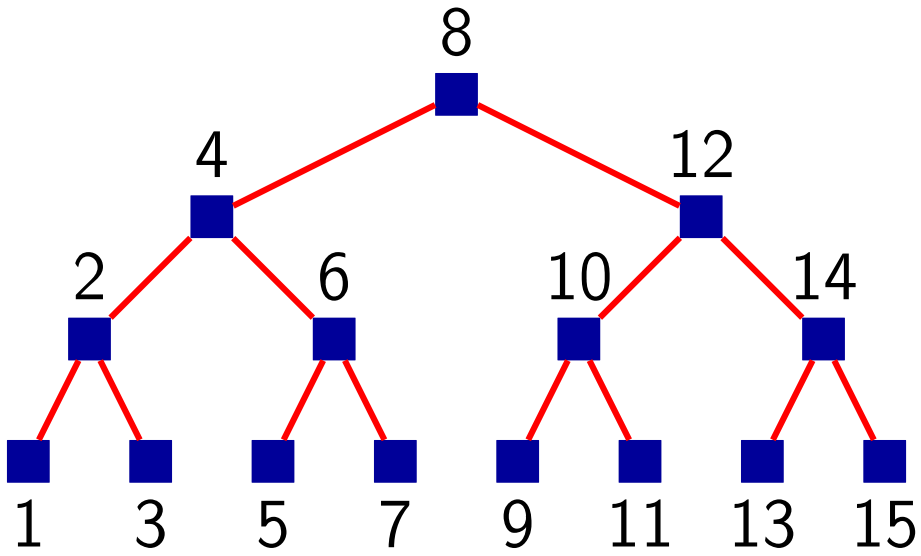
Structures de données

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

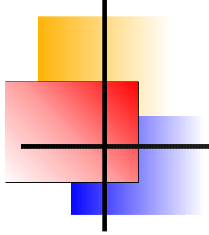
- Dans les deux organisations, le livreur a une méthode simple pour trouver une maison en partant de la pizzeria.
- Note une organisation en étoile avec la pizzeria au milieu permet des trajets très courts, mais choisir la bonne rue prend du temps.
- On suppose qu'il faut une unité de temps pour passer 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

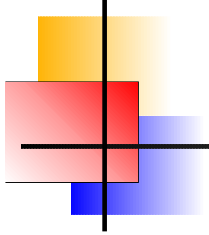
- Dans les deux organisations, le livreur a une méthode simple pour trouver une maison en partant de la pizzeria.
- Note une organisation en étoile avec la pizzeria au milieu permet des trajets très courts, mais choisir la bonne rue prend du temps.
- On suppose qu'il faut une unité de temps pour passer 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	$n-1$	$\sim \log_2(n)$



Complexité

- 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.

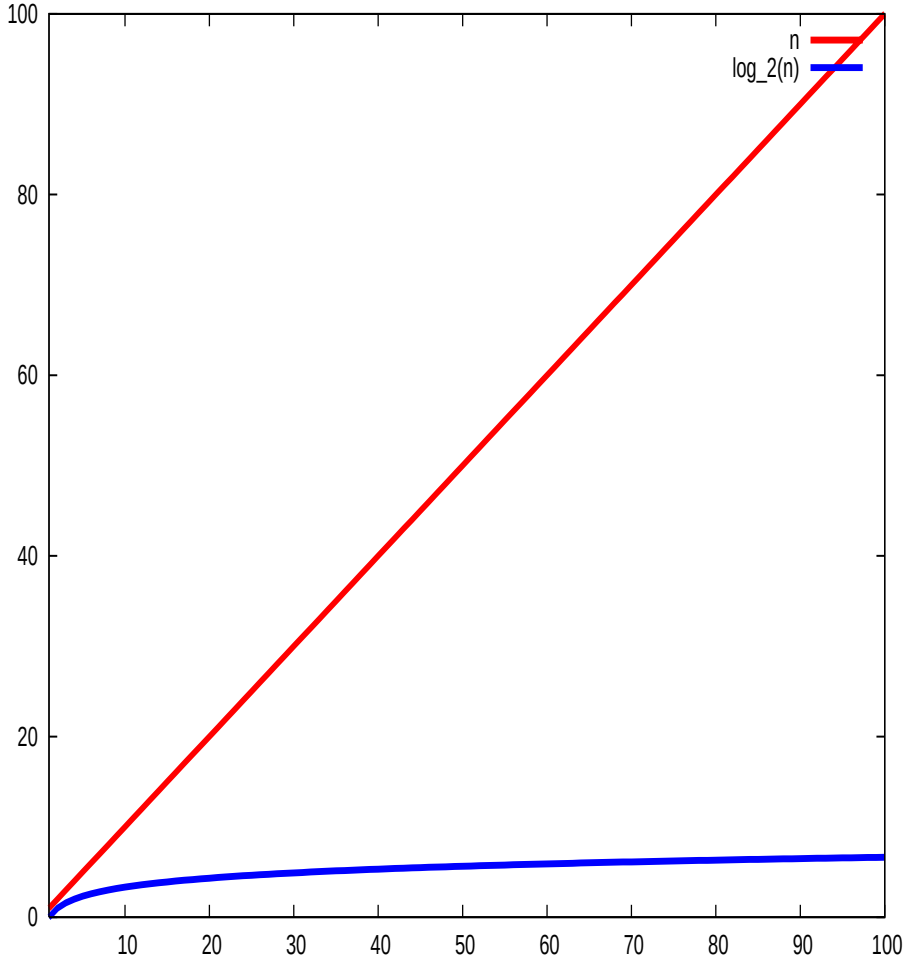


Complexité

- 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

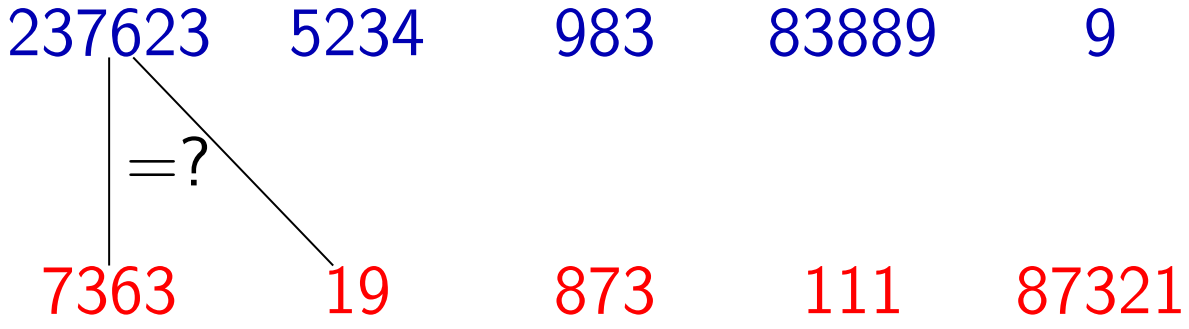
Problème : déterminer si 2 ensembles E_1, E_2 de n entiers ont une valeur commune.

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

Complexité : second exemple

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

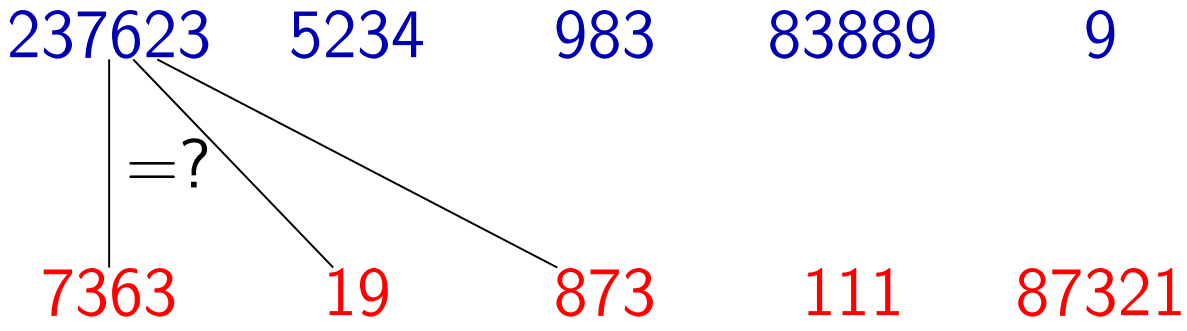
Algorithme 1 : comparer successivement chaque élément de E_1 avec chaque élément de $E_2 \rightsquigarrow n^2$ comparaisons.



Complexité : second exemple

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

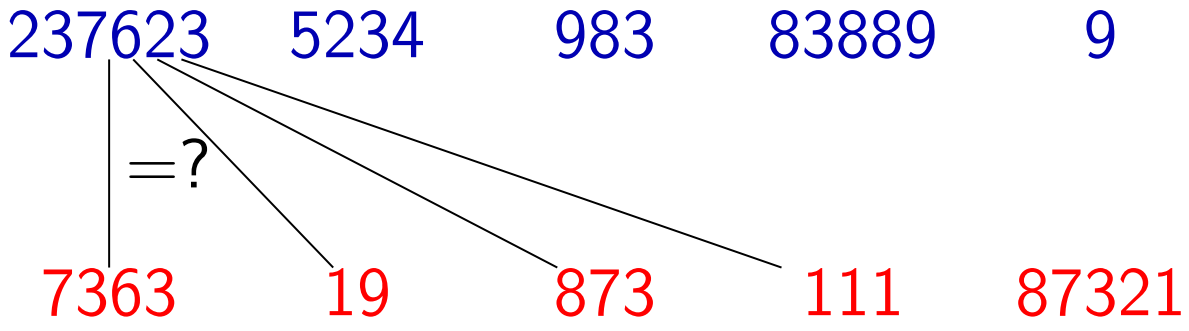
Algorithme 1 : comparer successivement chaque élément de E_1 avec chaque élément de $E_2 \rightsquigarrow n^2$ comparaisons.



Complexité : second exemple

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

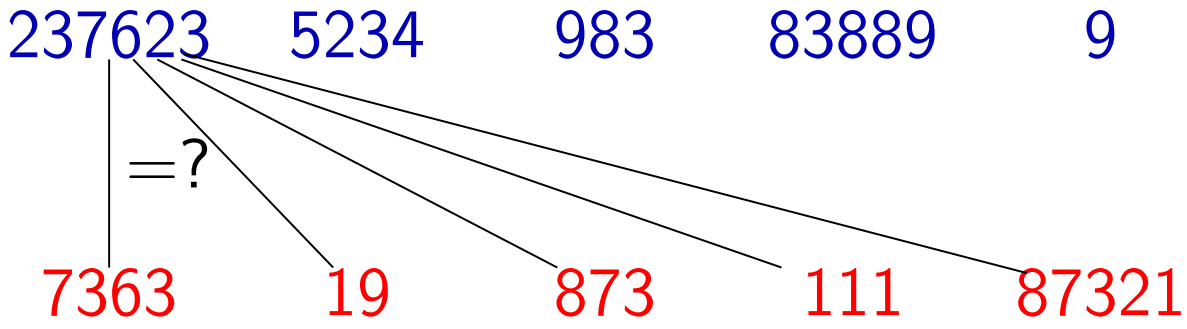
Algorithme 1 : comparer successivement chaque élément de E_1 avec chaque élément de $E_2 \rightsquigarrow n^2$ comparaisons.



Complexité : second exemple

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

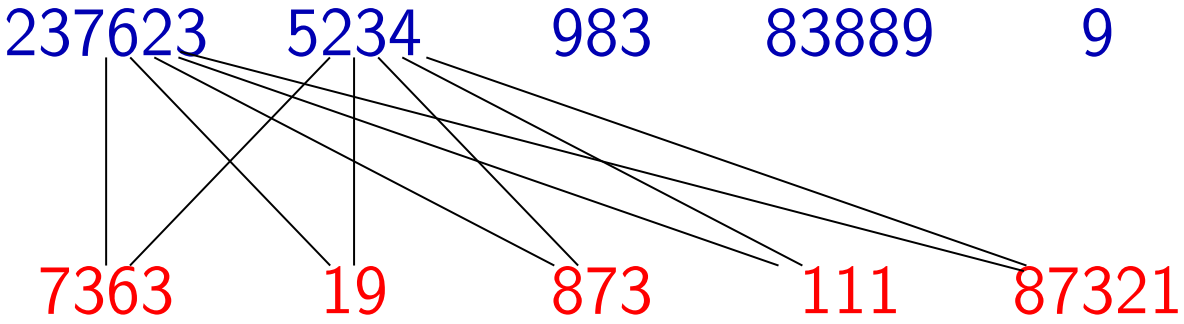
Algorithme 1 : comparer successivement chaque élément de E_1 avec chaque élément de $E_2 \rightsquigarrow n^2$ comparaisons.



Complexité : second exemple

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

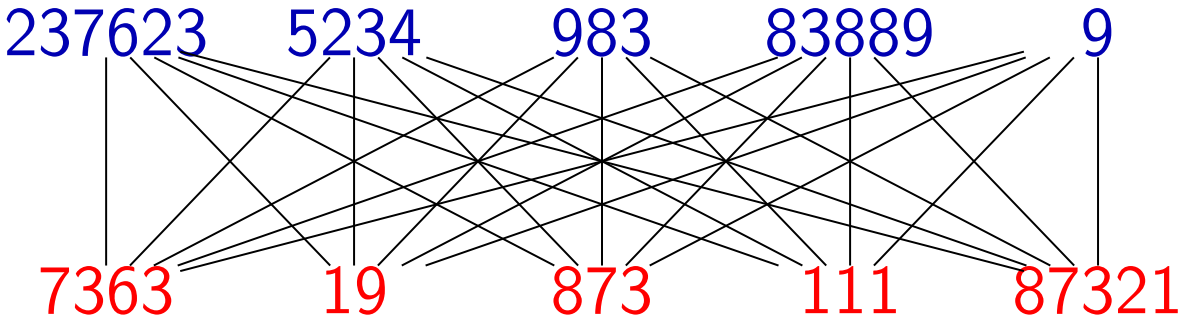
Algorithme 1 : comparer successivement chaque élément de E_1 avec chaque élément de $E_2 \rightsquigarrow n^2$ comparaisons.



Complexité : second exemple

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

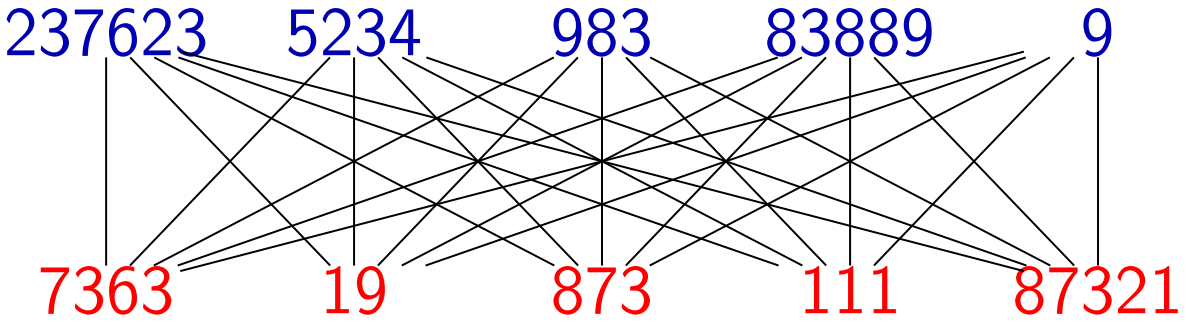
Algorithme 1 : comparer successivement chaque élément de E_1 avec chaque élément de $E_2 \rightsquigarrow n^2$ comparaisons.



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.

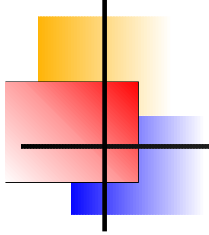


On peut résoudre le problème avec environ $n \log_{10}(n)$ comparaisons !

$\lvert E_1 \rvert = \lvert E_2 \rvert$	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 $O(n \log n)$

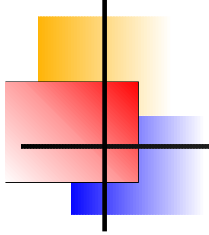
- Sur un ordinateur exécutant une instruction élémentaire en $10^{-8} s$
- Si les ensembles E_1 et E_2 ont $n = 1\,000\,000 = 10^6$ éléments :
  - Exécuter $n \log_{10} n$ instructions élémentaires nécessite $0,06$s.
  - Exécuter n^2 instructions élémentaires nécessite 10^4s soit $\sim$ 2h45.
- Si les ensembles E_1 et E_2 ont $n = 10\,000\,000 = 10^7$ éléments :
  - Exécuter $n \log_{10} n$ instructions élémentaires nécessite $0,7$s.
  - 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.

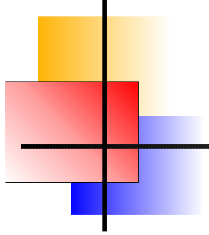
  Chaque style a ses idiomes de programmation.

Quelques instructions Python

1. Affectation : ranger une valeur dans une variable

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

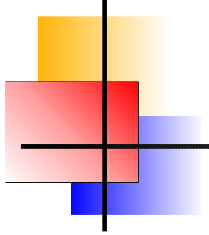
- L'ordinateur effectue les instructions dans l'ordre.
- L'ordre des instructions est donc très important.
- Une variable désigne un emplacement dans lequel on peut mémoriser une valeur. Une variable a un nom.
- En python, le symbole = n'a pas la même signification qu'en mathématique. Il signifie calculer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



Quelques instructions Python (suite)

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,

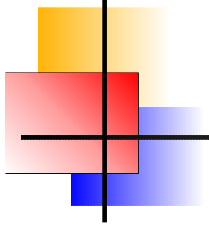
```
# affiche les entiers de 0 à 9
for i in range(10) :
    print i
```

Quelques instructions Python (fin)

4. Définition d'algorithme (fonction) :
En maths : soit la fonction $f : x \mapsto 2x^2 + 1$
En Python :

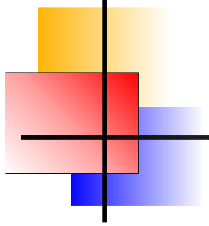
```
def f(x) :
    return 2 * x * x + 1

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/



En résumé...

- 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.



Dans cette partie...

4– Introduction aux graphes

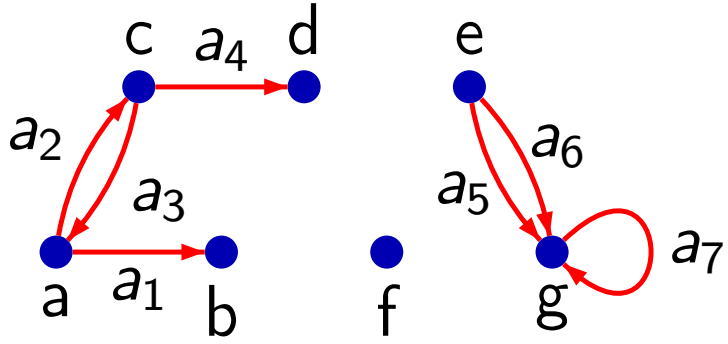
- $1^{\text{ère}}$ sorte de graphes : les graphes orientés
- Quelques exemples de graphes orientés
- $2^{\text{ème}}$ sorte de graphes : les graphes non orientés
- Quelques exemples de graphes non orientés
- Les graphes dans des problèmes courants

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 

- Exemple (graphe simple)
  - Sommets : {a,b,c,d,e,f,g}.
  - Arcs : $\{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$.

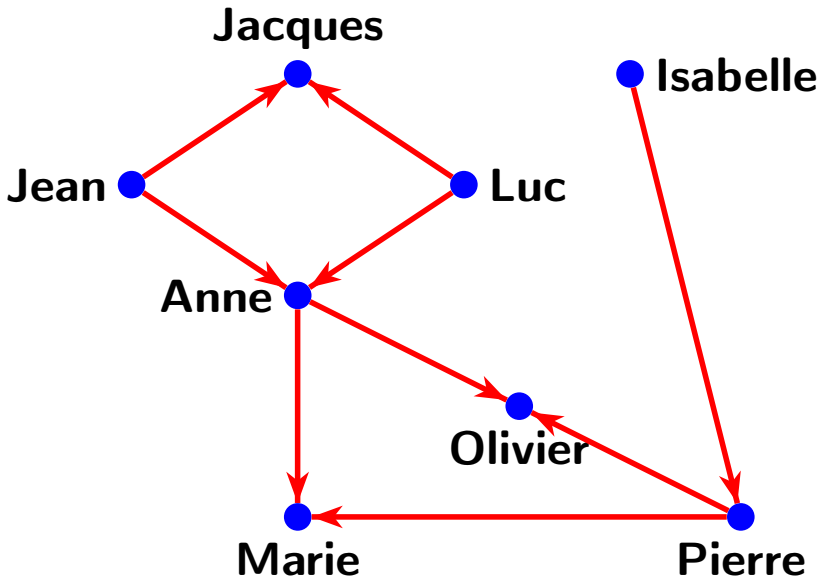


Exemple de graphe orienté : parents

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) :

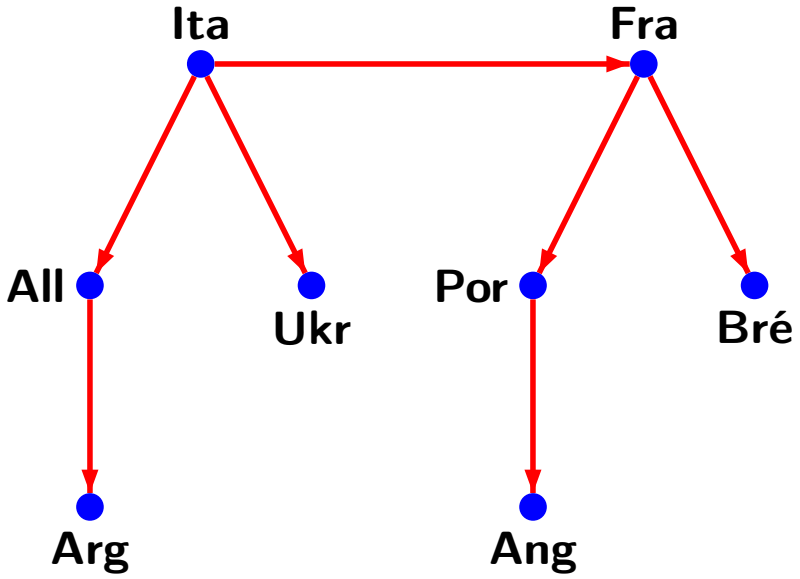


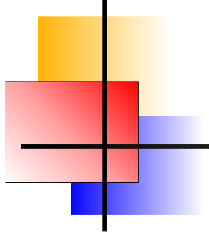
Exemple de graphe orienté : matchs de foot

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).





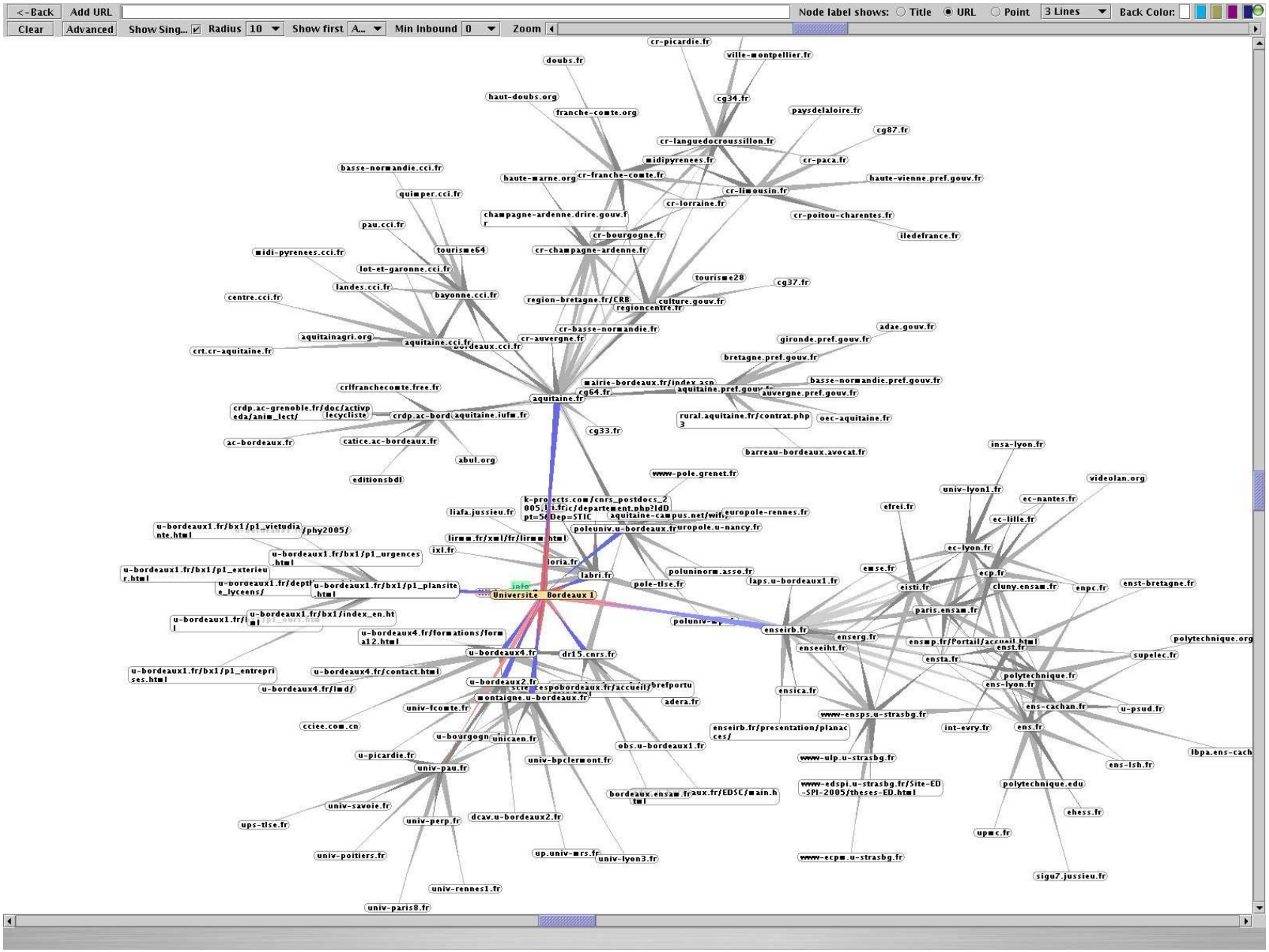
Exemple de graphe orienté : internet

Ensemble de sommets : les pages web.

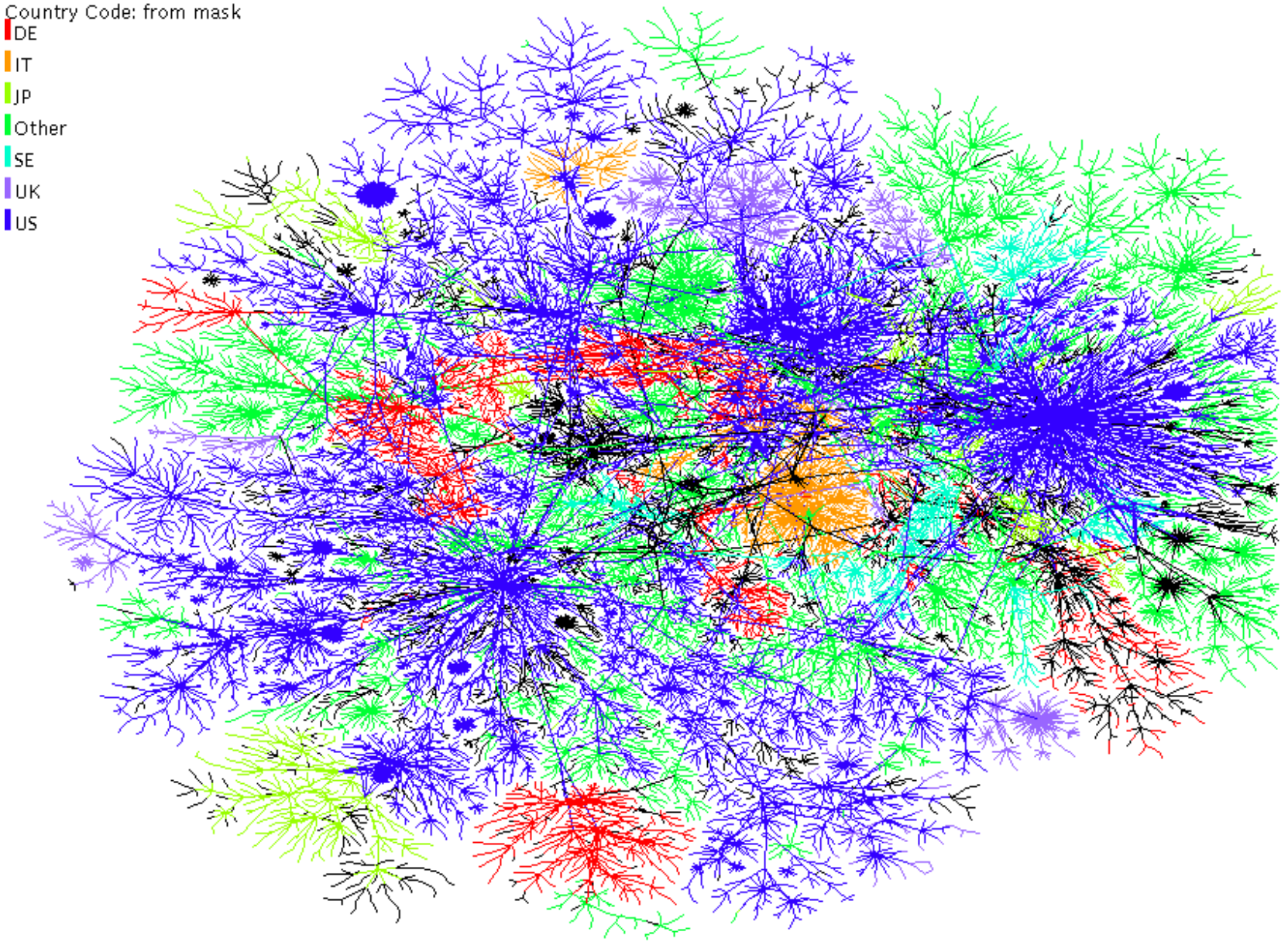
Relation : l'ensemble des couples (w_1, w_2) tels qu'il existe un lien direct sur la page web w_1 qui amène sur la page web w_2.

Relation non symétrique, graphe orienté.

Représentation graphique (internet)



Représentation graphique : internet



$2^{\text{ème}}$ sorte de graphes : graphes non orientés

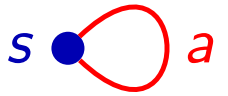
- Un graphe orienté peut représenter une relation entre sommets.
- Une arête associée à (s, t) exprime que s et t sont en relation.
- Un graphe non orienté peut représenter une relation symétrique

$2^{\text{ème}}$ sorte de graphes : graphes non orientés

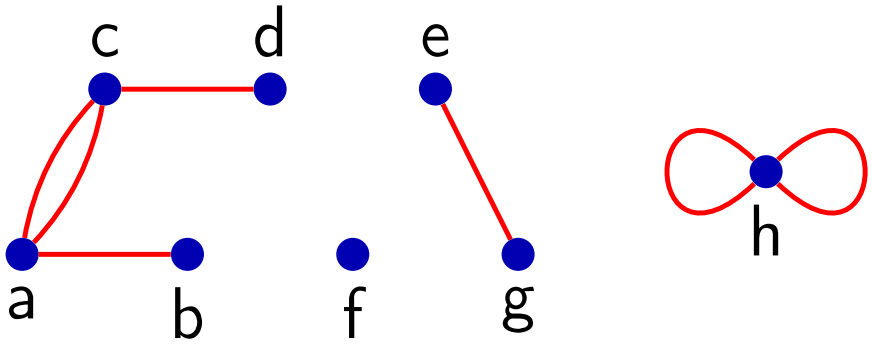
- Un graphe orienté peut représenter une relation entre sommets.
- Une arête associée à (s, t) exprime que s et t sont en relation.
- Un graphe non orienté peut représenter une relation symétrique
- Dans un graphe non orienté, une arête est associée soit à une paire de sommets $\{s, t\}$, soit à un singleton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

$2^{\text{ème}}$ 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 

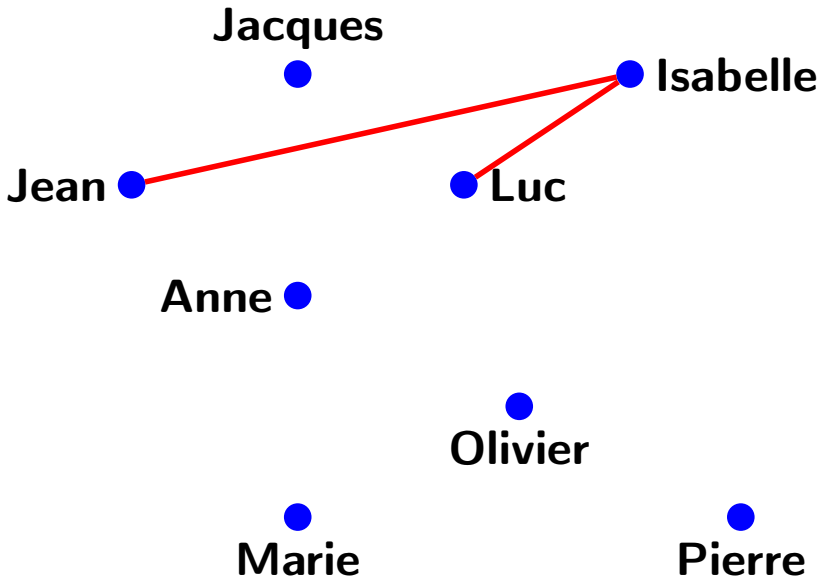
- Exemple



Exemple de graphe non orienté : cousins

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

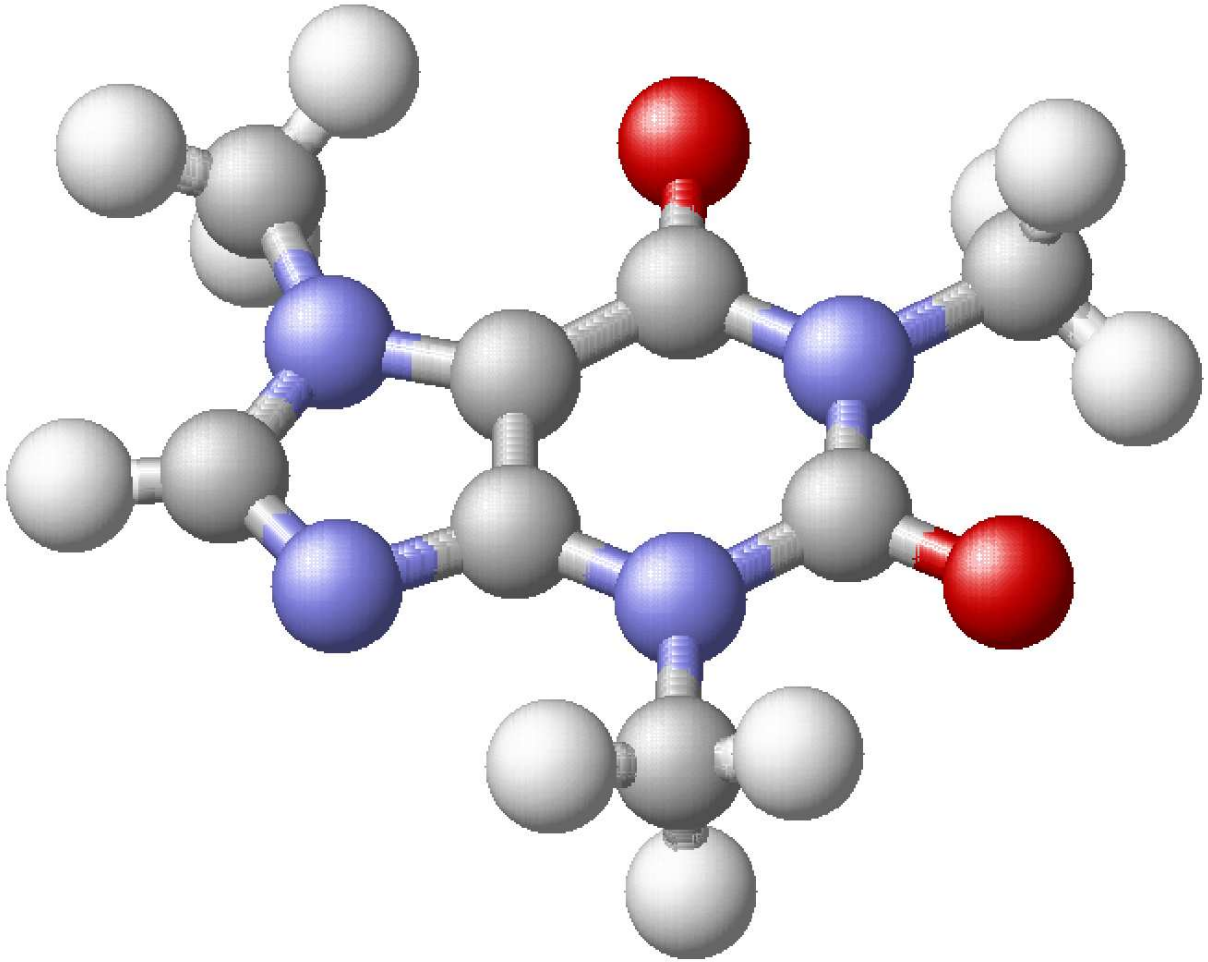
Relation : l'ensemble des couples de personnes (p_1, p_2) tels que p_1 est un cousin de p_2 (relation symétrique).
Représentation graphique (relation symétrique, graphe non orienté) :



Exemple de graphe non orienté : molécule

Ensemble : les atomes de la molécule de caféine.

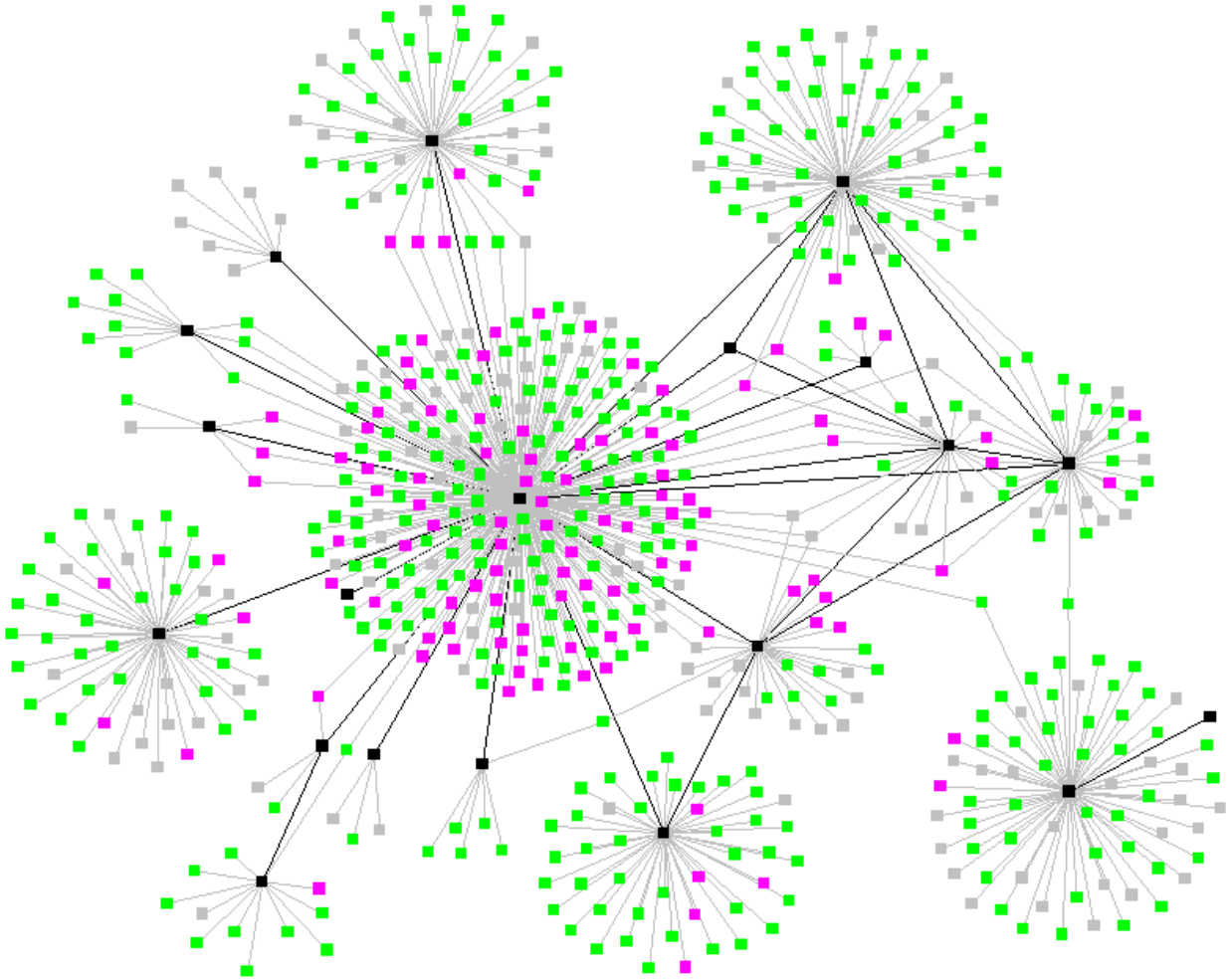
Relation : l'ensemble des couples d'atomes (a_1, a_2) tels que a_1 et a_2 partagent au moins un électron (liaison covalente, relation symétrique)

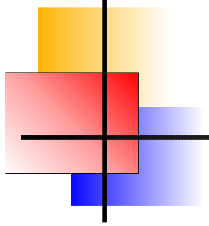


Exemple - graphe non orienté : connaissance

Ensemble : un ensemble de personnes.

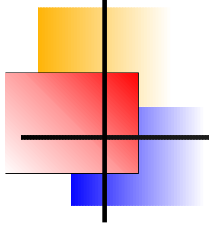
Relation : ensemble des paires de personnes qui se connaissent.





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.

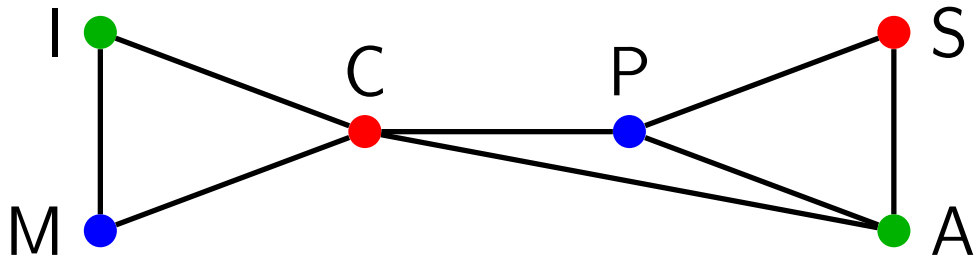


Organisation d'examens

- 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 $\Longrightarrow$ on peut organiser tout en $3 \times 2h$.

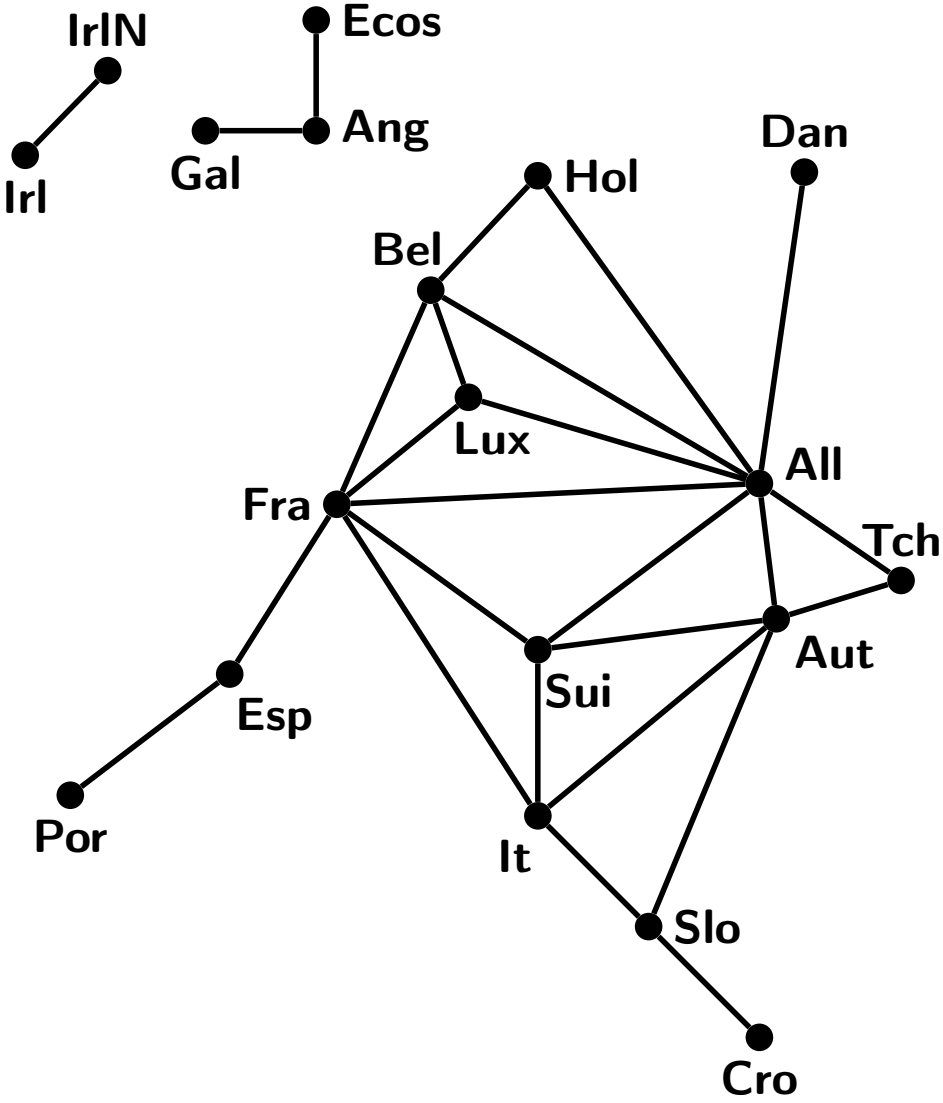


Coloration de carte géographique

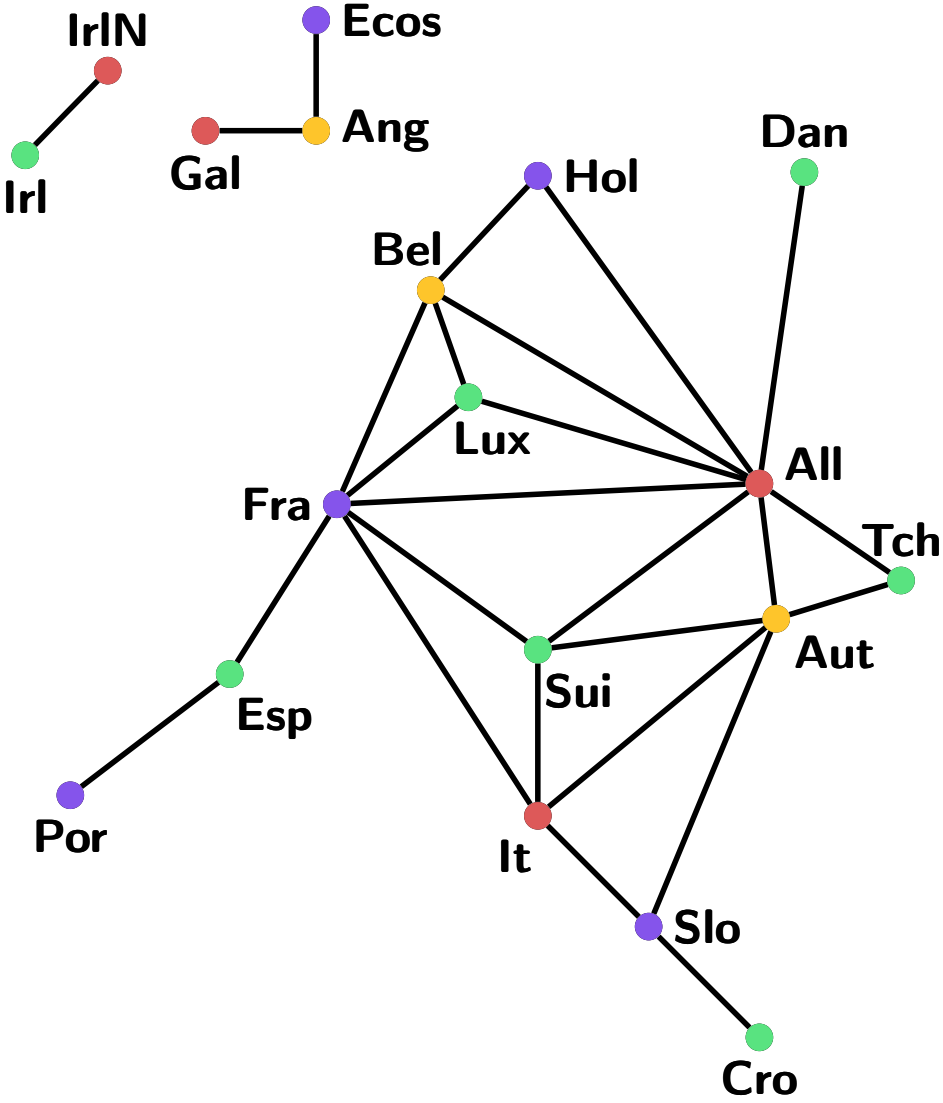
- 1852 Guthrie colorie la carte des cantons anglais avec 4 couleurs sans que 2 cantons adjacents n'aient la même couleur.
- 4 couleurs suffisent-elles pour colorier n'importe quelle carte ?
- 1976 : Oui (Appel & Haken). Résultat obtenu en partie par ordinateur.
  - Carte géograhique $\rightsquigarrow$ 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 colorier le graphe avec 4 couleurs, sans que 2 sommets adjacents n'aient la même couleur.
- On peut deriver de la preuve un algorithme permettant de trouver la coloration.

Coloration de carte géographique

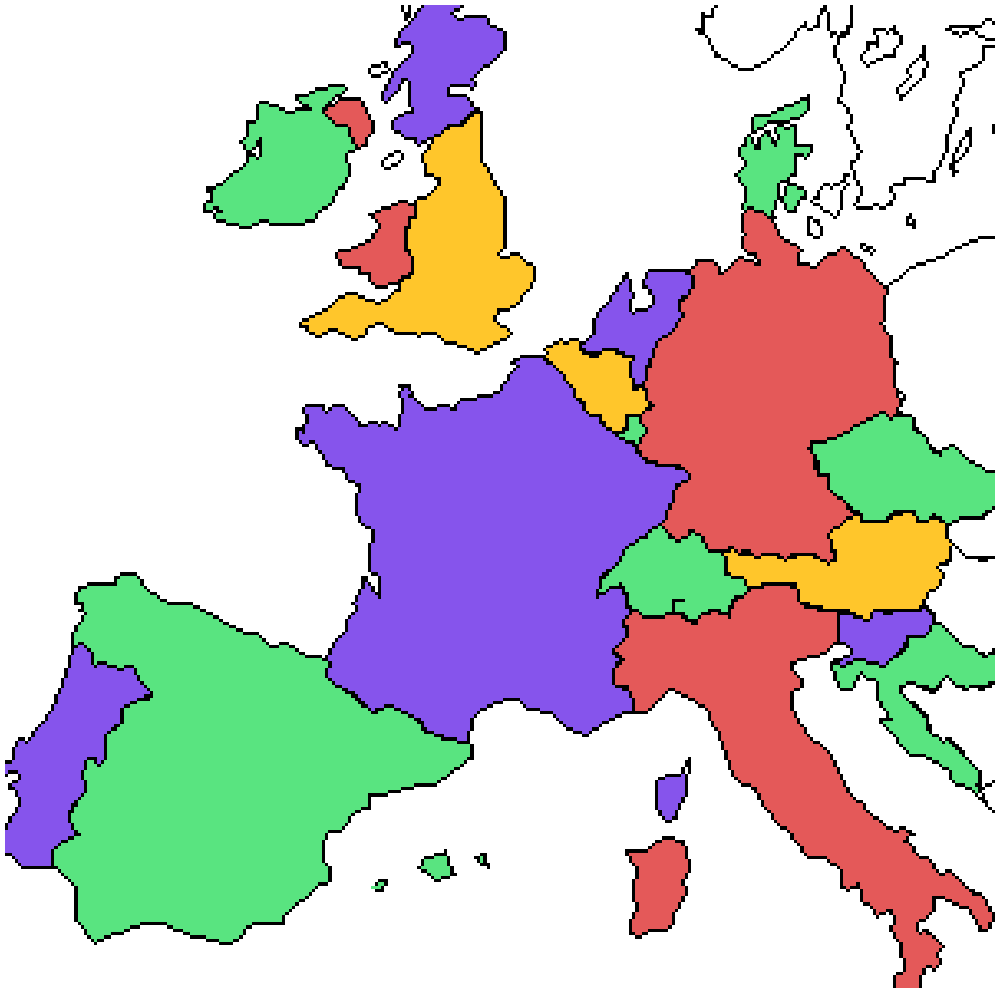
Coloration de carte géographique

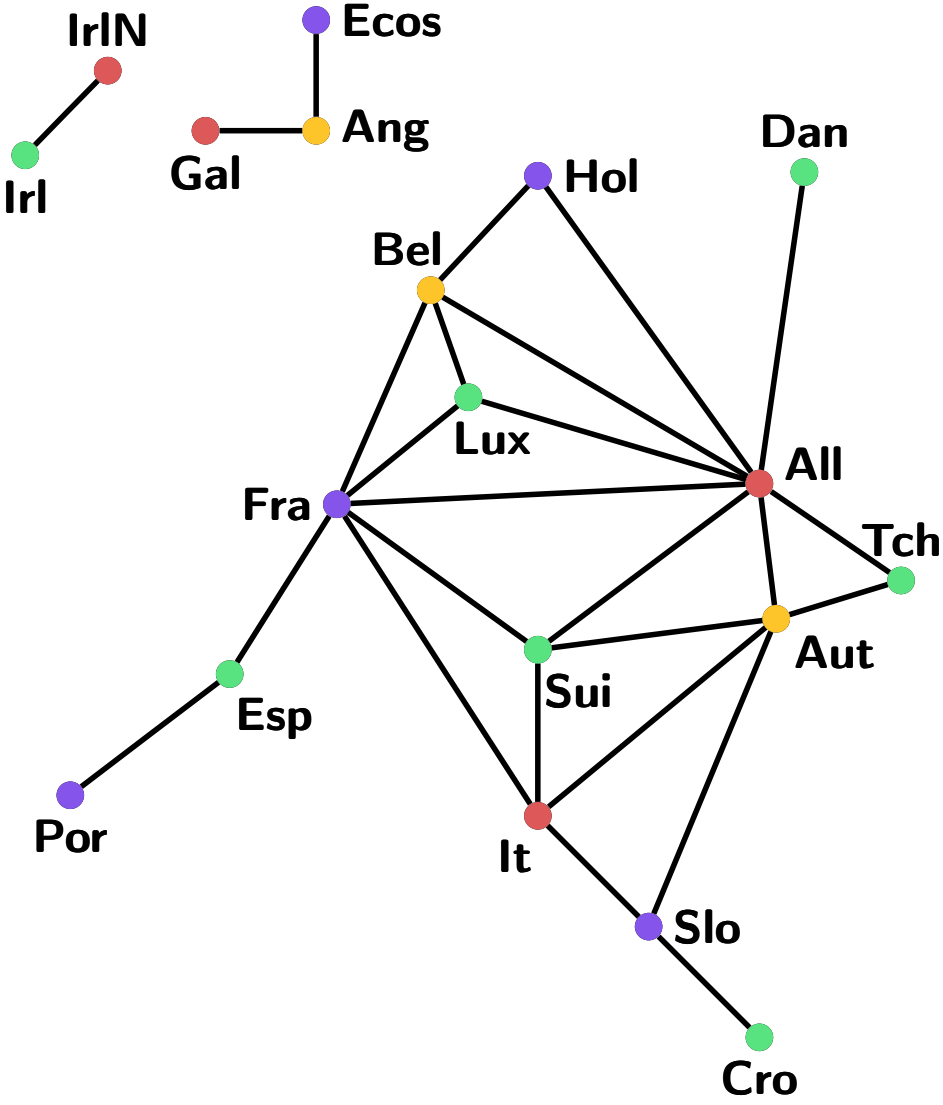


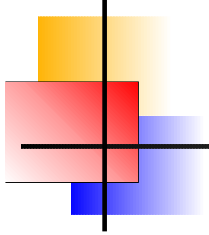
Coloration de carte géographique



Coloration de carte géographique





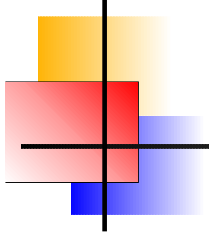


Coloration de carte géographique

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 pas

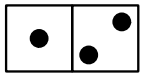
  est équivalent à l'un des problèmes ouverts les plus difficiles, sélectionné par le Clay Mathematical Institute (1 000 000 $).

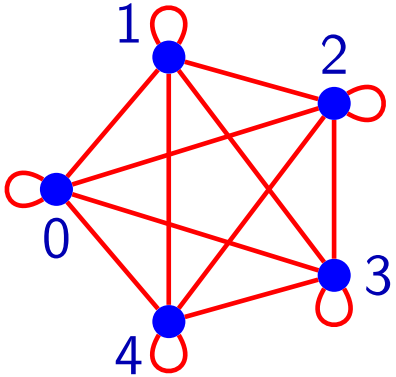


Dominos

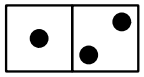
- On dispose d'un jeu de dominos, le plus grand numéro étant n.
- Qu : Peut-on placer tous les dominos (en suivant les règles) ?

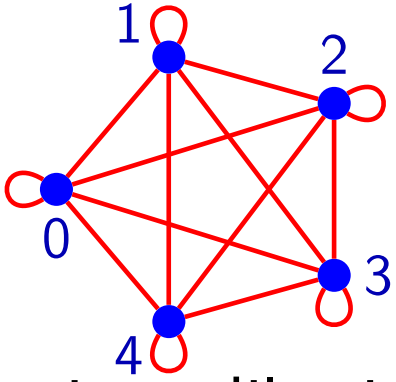
Dominos

- 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 : $\{0, \cdots, n\}$. 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.

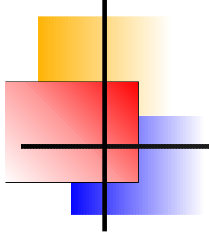


Dominos

- 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 : $\{0, \cdots, n\}$. 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

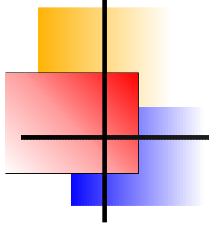
- Les graphes permettent de modéliser des voies de communication
  - les sommets représentent des villes,
  - les arêtes représentent des routes.
- On peut associer à chaque arête d'un tel graphe (orienté s'il y a des sens uniques) un nombre représentant la distance entre les deux sommets reliés.

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.

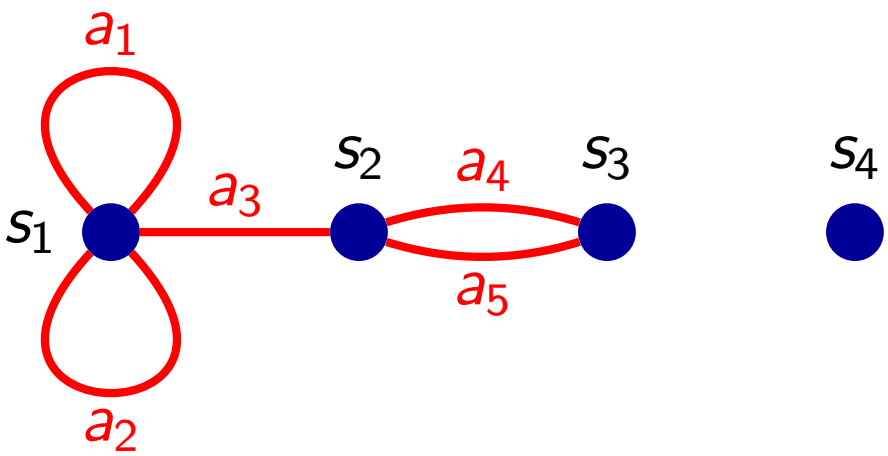
Définition de graphe, version centrée arêtes

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").

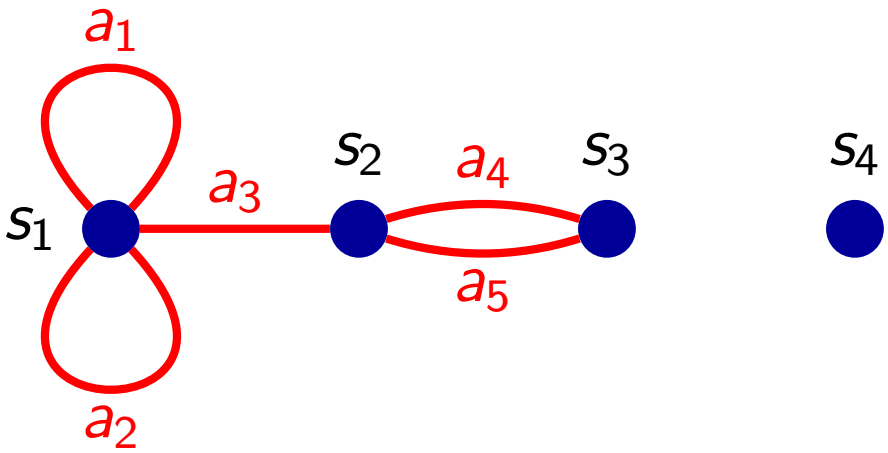
Définition de graphe, centrée sommets

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").



$$inc(s_1) = \{a_1, a_2, a_3\}$$
$$inc(s_2) = \{a_3, a_4, a_5\}$$
$$inc(s_3) = \{a_4, a_5\}$$
$$inc(s_4) = \varnothing$$

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 $inc : S \to \mathcal{P}(A)$ 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

`NB_NODES(g)` donne le nombre de sommets du graphe `g`. Les sommets sont numérotés 0, 1, ..., `NB_NODES(g)`-1.

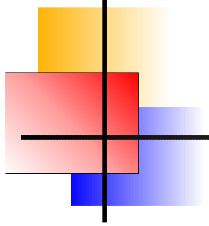
`NODE(g,i)` donne le sommet numéro `i` de `g`.

Def. centrée arêtes

- `NB_EDGES(g)` donne le nombre d'arêtes de `g`. Les arêtes sont numérotées 0, 1, ..., `NB_EDGES(g)`-1.
- `EDGE(g,i)` donne la ième arête de `g`.
- `EDGE_END(e,k)`c pour k = 0 ou 1, donne l'une ou l'autre des extrémités de l'arête `e`.

Def. centrée sommets

- `NB_DARTS(s)` donne le nombre d'arêtes incidentes au sommet `s`. Les arêtes incidentes à `s` sont numérotées 0, 1, ..., `NB_DARTS(s)`-1.
- `INC_EDGE(s,i)` donne la ième arête incidente au sommet `s`.
- `FOLLOW_EDGE(s,e)` donne le sommet obtenu partant de `s` en suivant `e`.

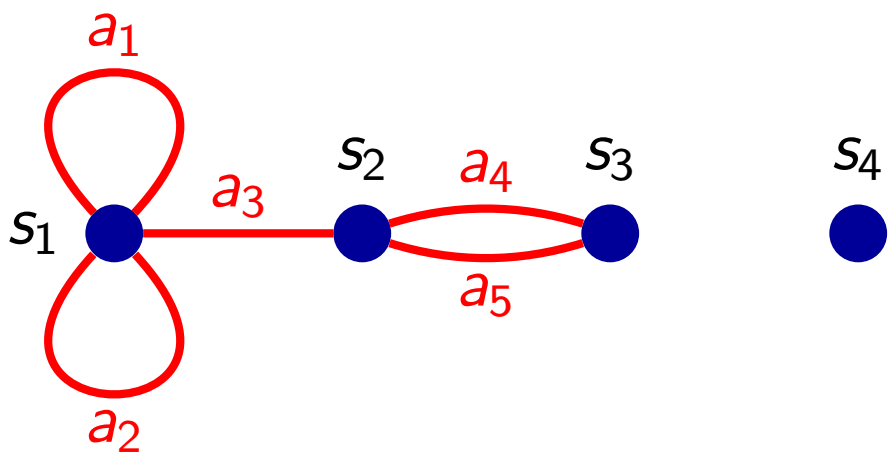


Dans cette partie...

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.



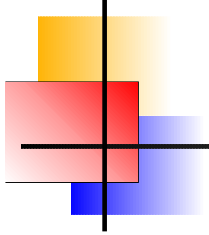
- Ici $d(s_1) = 5$, $d(s_2) = 3$, $d(s_3) = 2$, $d(s_4) = 0$.

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

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

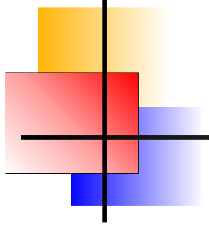
Le degré de s_1 et s_2 dans H est 1 de plus que leur degré dans G.

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

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

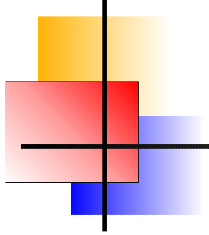
$$\sum_{s \in S(H)} d(s) = 2|A(H)|$$

Nous avons donc prouvé la propriété par induction.



Dans cette partie...

7– Chaînes



La notion de chaîne

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

Définition de chaîne

Une *chaîne* dans un graphe est une suite

$$C = s_1, e_1, s_2, e_2, \ldots, s_k, e_k, s_{k+1}$$

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

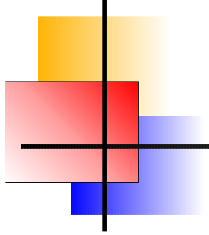
Remarques :

- si $k = 0$, on obtient une chaîne sans arête

$$C = s_1.$$

- pour définir une chaîne, se donner
  - seulement la suite des arêtes, ou
  - seulement la suite des sommets

  ne suffit pas (pourquoi) ?



Existence d'une chaîne

- Vérifier s'il existe une chaîne entre un sommet s et un sommet t n'est pas forcément simple.

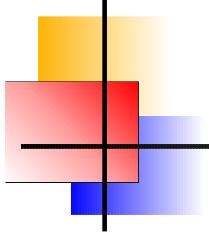
- Pour y arriver, nous allons utiliser une technique pour marquer et démarquer les sommets.

Chaîne simple

- Une chaîne $C = s_1, e_1, s_2, e_2, \ldots, s_k, e_k, s_{k+1}$ est *simple* si et seulement si

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

- 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.

Preuve

- Soit $C = s_1, e_1, s_2, e_2, \ldots, s_k, e_k, s_{k+1}$ 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 :

$$C = s_1, e_1, \ldots, s_i, e_i, \ldots, s_j, e_j, \ldots, s_k, e_k, s_{k+1}.$$

Preuve

- Soit $C = s_1, e_1, s_2, e_2, \ldots, s_k, e_k, s_{k+1}$ 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 :

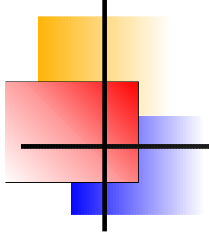
$$C = s_1, e_1, \ldots, s_i, e_i, \ldots, s_j, e_j, \ldots, s_k, e_k, s_{k+1}.$$

Preuve

- Soit $C = s_1, e_1, s_2, e_2, \ldots, s_k, e_k, s_{k+1}$ 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 :

$$C = s_1, e_1, \ldots, s_i, e_i, \ldots, s_j, e_j, \ldots, s_k, e_k, s_{k+1}.$$

- On construit C' plus courte en supprimant de C la partie $e_i, \ldots, s_j$.
- On obtient alors : $C' = s_1, e_1, \ldots s_i, e_j, \ldots, s_k, e_k, s_{k+1}$.



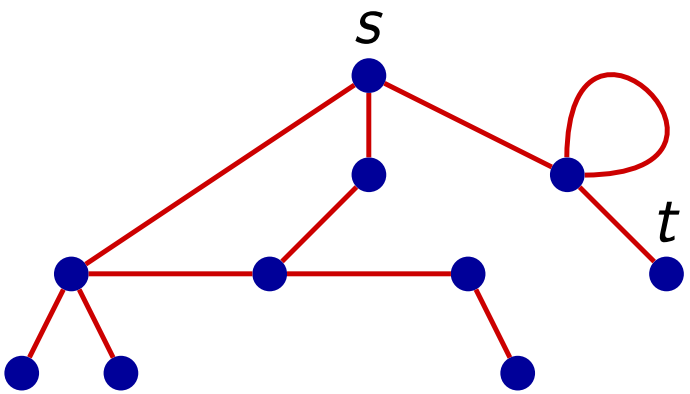
Preuve (suite)

- 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 :

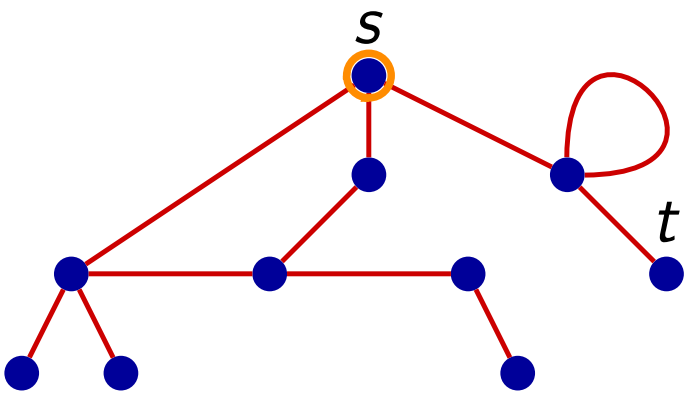
1. démarquer tous les sommets
2. marquer s
3. tant que t n'est pas marqué,
   - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
   - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
   - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer la valeur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



Existence d'une chaîne entre s et t

Algorithme :

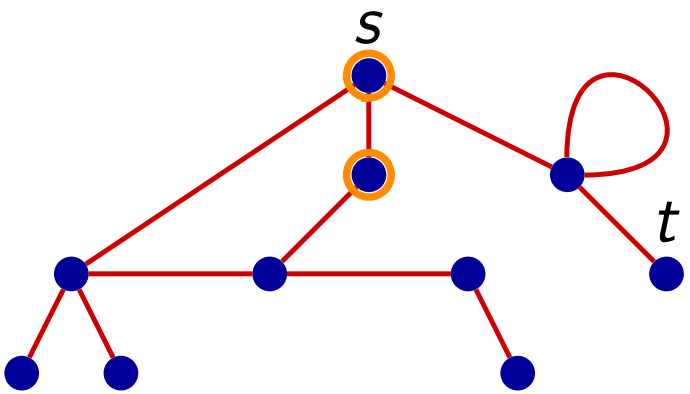
1. démarquer tous les sommets
2. marquer s
3. tant que t n'est pas marqué,
   - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
   - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
   - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer la valeur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



Existence d'une chaîne entre s et t

Algorithme :

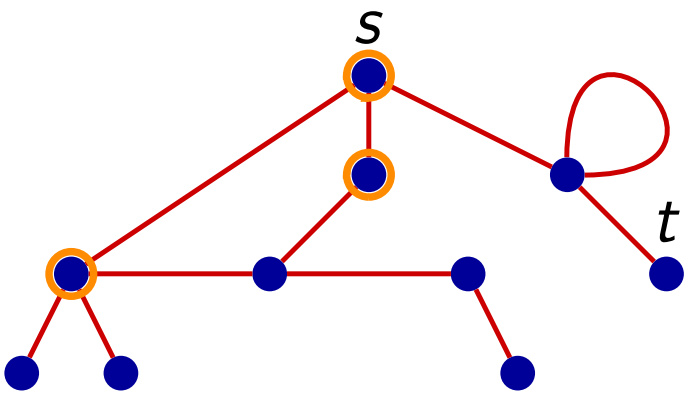
1. démarquer tous les sommets
2. marquer s
3. tant que t n'est pas marqué,
   - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
   - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
   - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer la valeur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



Existence d'une chaîne entre s et t

Algorithme :

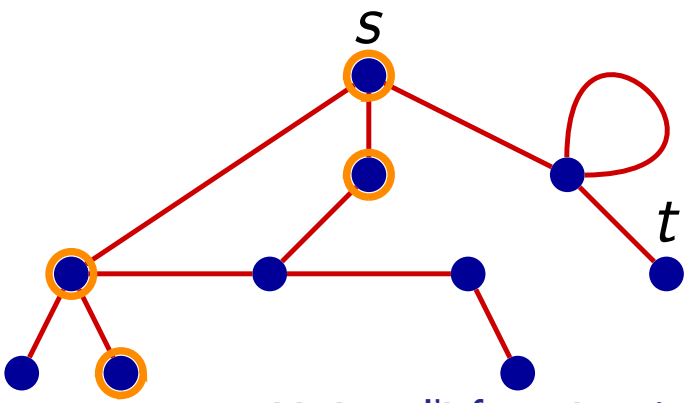
1. démarquer tous les sommets
2. marquer s
3. tant que t n'est pas marqué,
   - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
   - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
   - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer la valeur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



Existence d'une chaîne entre s et t

Algorithme :

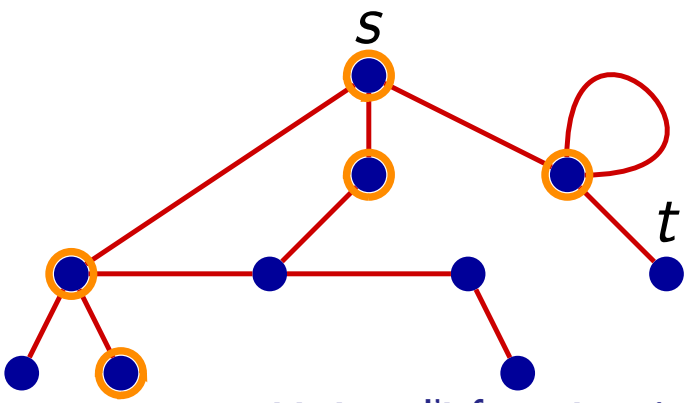
1. démarquer tous les sommets
2. marquer s
3. tant que t n'est pas marqué,
   - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
   - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
   - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer la valeur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



Existence d'une chaîne entre s et t

Algorithme :

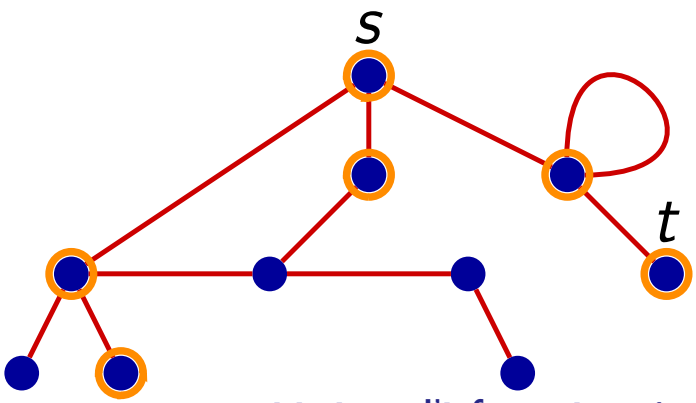
1. démarquer tous les sommets
2. marquer s
3. tant que t n'est pas marqué,
   - 3.1 chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
   - 3.2 si une telle arête n'existe pas, renvoyer la valeur "faux"
   - 3.3 sinon marquer l'extrémité non encore marquée
4. renvoyer la valeur "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, 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 $\Sigma_{s\in S}(d(s)+1) = 2|A| + |S| = O(|A| + |S|)$ 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 $\Sigma_{s\in S}(d(s)+1) = 2|A| + |S| = O(|A| + |S|)$ 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 $\Sigma_{s\in S}(d(s)+1) = 2|A| + |S| = O(|A| + |S|)$ 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 $\Sigma_{s\in S}(d(s)+1) = 2|A| + |S| = O(|A| + |S|)$ 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 $\Sigma_{s\in S}(d(s)+1) = 2|A| + |S| = O(|A| + |S|)$ 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 $\Sigma_{s\in S}(d(s)+1) = 2|A| + |S| = O(|A| + |S|)$ 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 $\Sigma_{s\in S}(d(s)+1) = 2|A| + |S| = O(|A| + |S|)$ 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 $O(|A| + |S|)$.