

		semaines	
		CI	TD/TP
Cours 1	I. Présentation II. Notions fondamentales – Définitions et propriétés III. Coloration d'un graphe IV. Représentation et modélisation des graphes	04/09	11/09
Cours 2	V. Chemins dans un graphe (chaînes et chemins – cycles et circuits – connexité – forte connexité – graphes eulériens et hamiltoniens) VI. Parcours de graphe (BFS -DFS) VII. Recherche des composantes fortement connexes dans un graphe orienté VIII. k-connexité	11/09	18/09
Cours 3	IX. Recherche des plus courts chemins (PCC) dans un graphe non pondéré X. Graphes étiquetés, valués, pondérés	18/09	25/09
Cours 4	XI. (suite) Algorithme A* XII. Application aux développements d'IA dans les jeux : Algorithme minimax	25/09	02/10

Cours 5	XIII. Arbres couvrants de poids minimum	02/09	09/10
Cours 6	XIV. Étude des flots	09/10	16/10
Cours 7	Révisions et approfondissements	16/10	23/10

Cours 1 :

I. Présentation

- Exemples de graphes
- Domaines d'applications
- Problématiques

II. Notions fondamentales et définitions

*sommets - arêtes/arcs - orientation - adjacence - degrés - ordre - taille
densité – graphes complets – sous-graphes –graphes partiels*

III. Coloration d'un graphe

- Algorithme naïf
- Algorithme de Welsh et Powell
- Problèmes de coloration

IV. Modélisation d'un graphe

- Matrice d'adjacence
- Modélisation objet

Théorie des graphes

I. Présentation

- **Graphe ?**

→ modèle représentant des **liens** (des **relations** ou des **interactions**) entre des **objets**

Les **objets** sont les **sommets** (ou **nœuds**) du graphe.

Les **relations** entre les objets sont représentées par les **arêtes** du graphe. Une arête relie deux sommets (relation binaire).

Graphe **non orienté** : relation symétrique

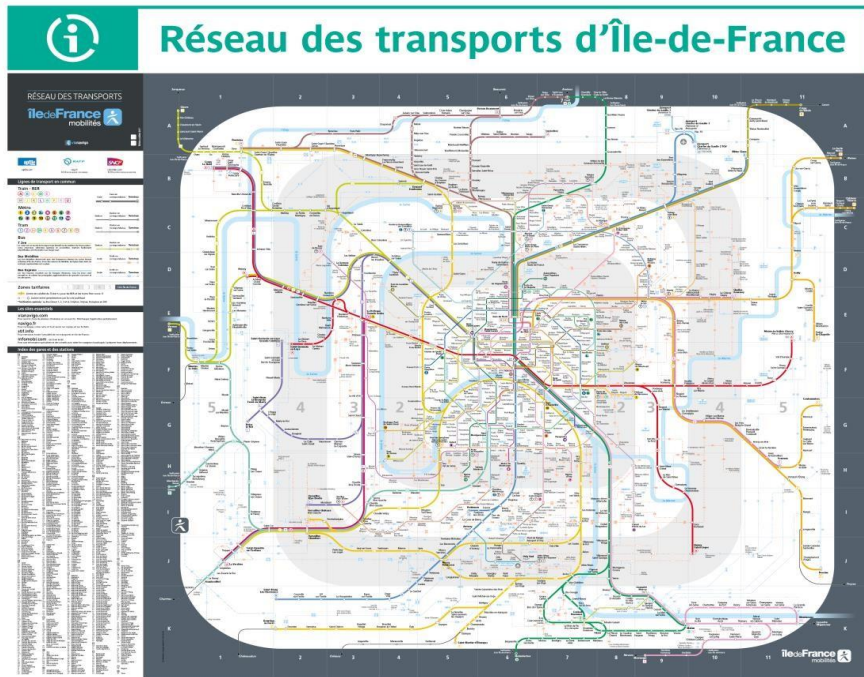
Graphe **orienté** : relation non symétrique

Les arêtes ont un sens, on les appelle des **arcs**

Exemples de graphes

- Réseaux : réseaux de transports

- ✓ réseau ferré : graphe non orienté



Sommets = stations

2 stations sont reliées par une arête si elles se suivent sur une même ligne.

Arêtes = tronçons de ligne entre 2 stations

- ✓ Réseau routier : graphe orienté (sens interdits)

Exemples de graphes

• Réseaux : réseaux sociaux

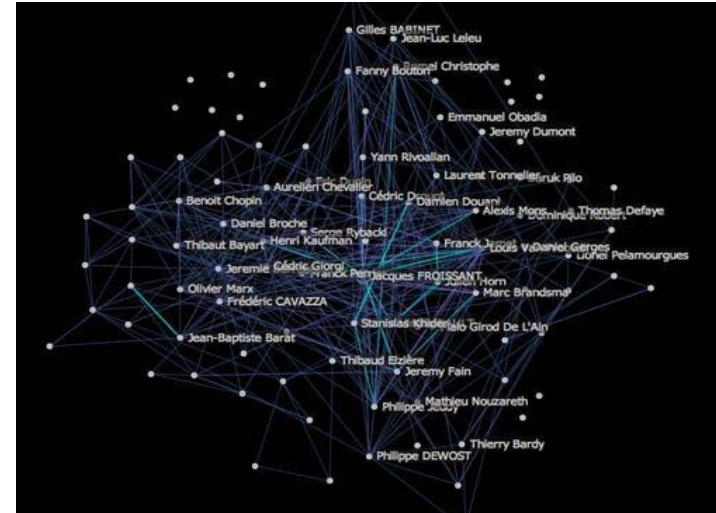
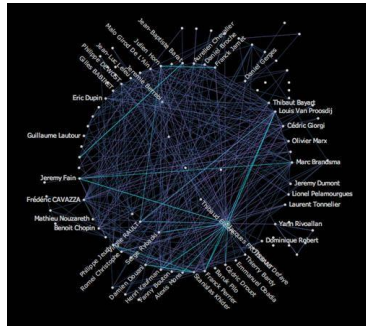


Réseaux Facebook : graphes non orientés

Sommets = membres

2 membres sont reliés par une arête s'ils sont « amis »

Relation entre 2 sommets = lien d'amitié



<http://www.ziserman.com/blog/2008/05/13/voir-mon-reseau-social-sur-facebook/>



Réseaux Twitter : graphes orientés

Sommets = membres

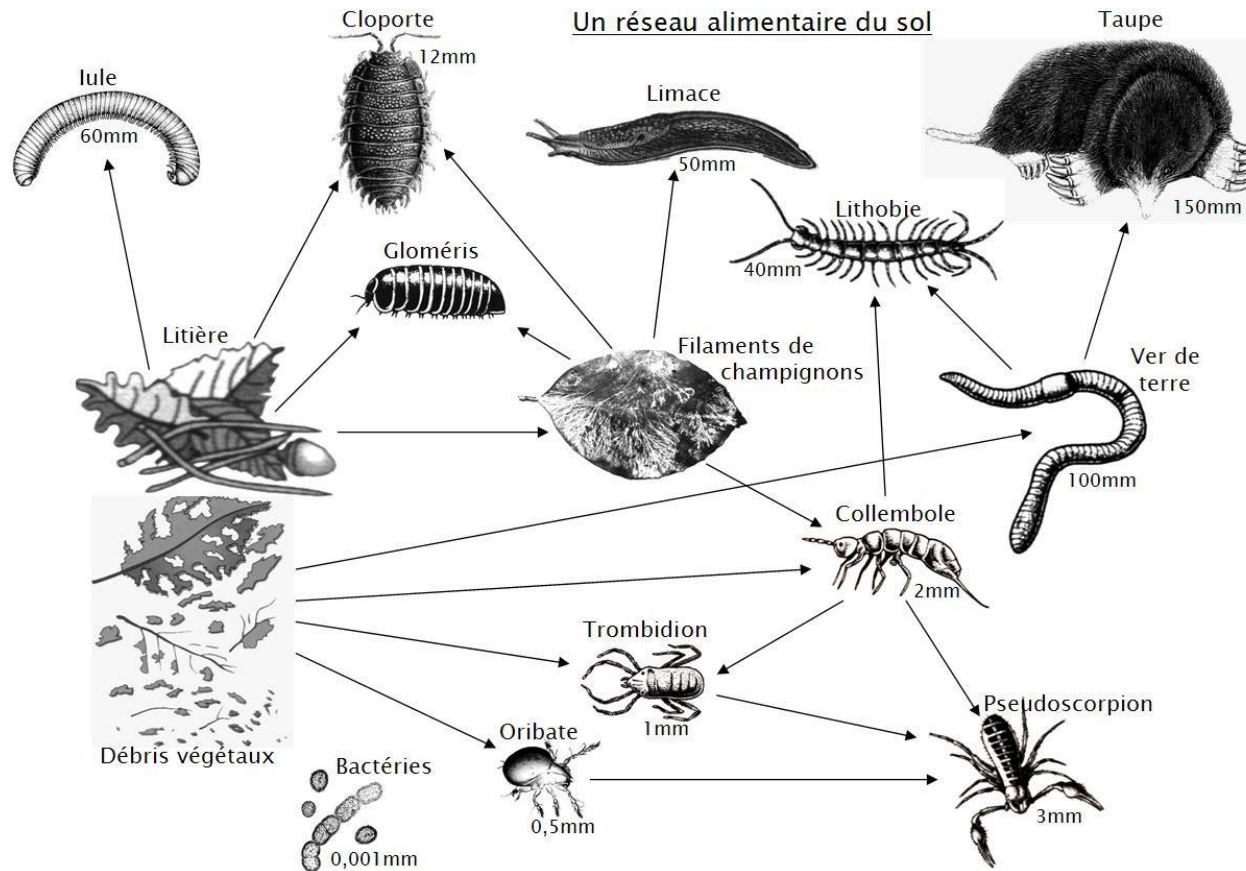
Relation entre 2 sommets = « est suivi par »

Il existe un arc allant d'un membre A à un membre B si A est suivi par B

Exemples de graphes

• Réseaux : réseaux trophiques

Un réseau trophique représente les chaînes alimentaires d'un écosystème.



Sommets = espèces

Relation = « est mangé par »

Exemples de graphes

• Graphes d'états

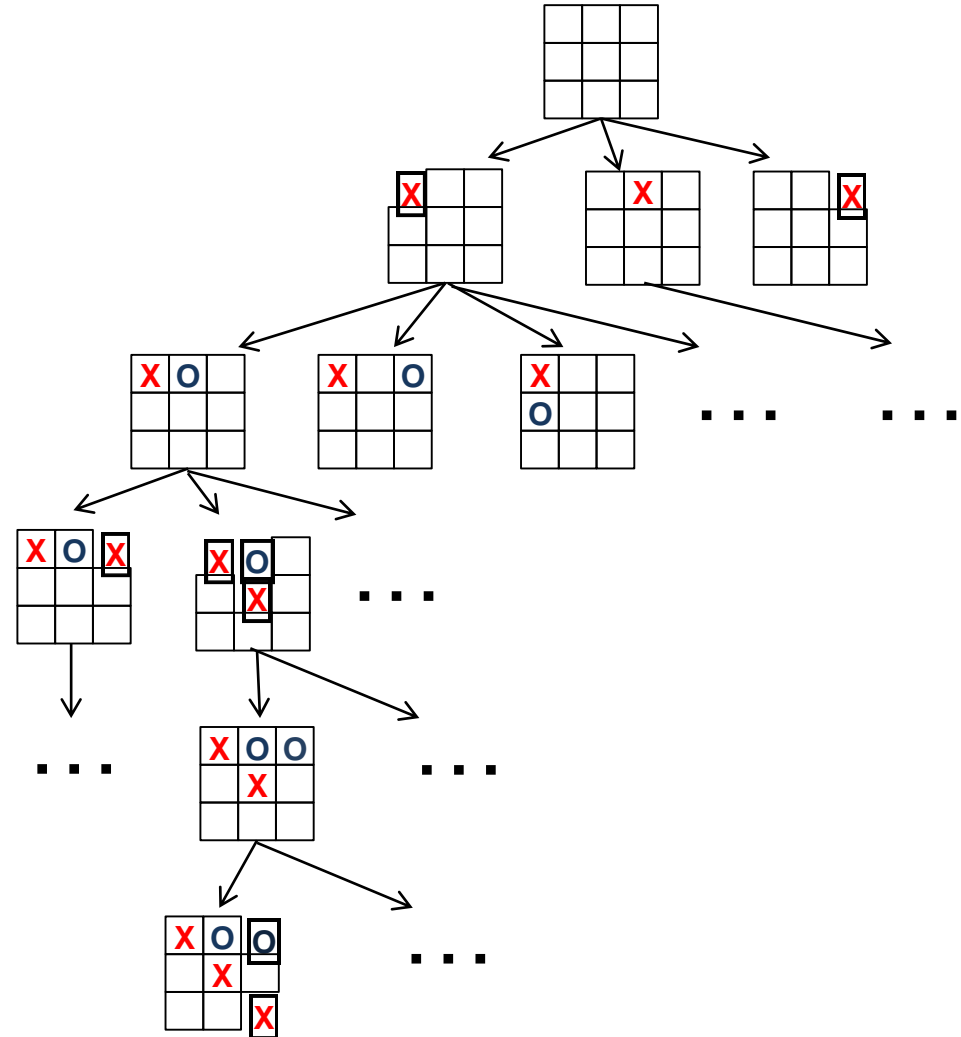


Arbre de jeu

Arbre représentant les déroulements possibles d'un jeu
exemple : le jeu du morpion

Sommets = états du plateau de jeu

Transitions (arcs) = un joueur joue



Exemples de graphes

• Graphes d'états-transitions

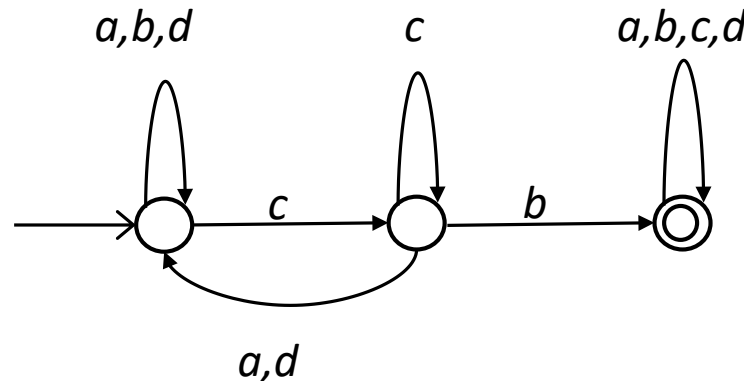
✓ automates

Un automate décrit les **états** successifs que peut prendre un système et les **transitions** pouvant provoquer des changements d'états.

Un automate est représenté par un graphe orienté dont les arcs sont **étiquetés** par des symboles (lettres, mots, nombres ...) qui décrivent les transitions.

Étant donné un « mot » fourni en entrée, l'automate lit les symboles du mot un par un et va d'état en état selon les transitions. Le mot lu est soit reconnu (accepté) par l'automate soit rejeté.

Exemple : automate qui reconnaît tous les mots construits avec un alphabet de 4 lettres {a,b,c,d} qui contiennent la séquence «cb»

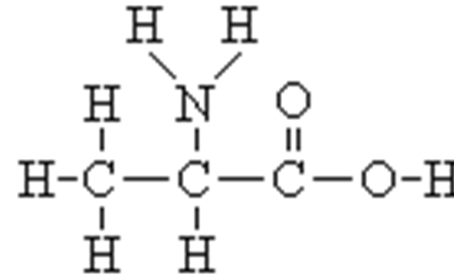


Théorie des graphes

Domaines d'applications

- chimie (isomères)
- biologie
- physique quantique
- sciences sociales
- transports /énergies
- informatique (compilation, codage et décodage, traitement de textes, complexité algorithmique, topologie des réseaux, protocoles de transferts)
- électronique
- gestion de projets
- ...

Molécule d'Aniline



Théorie des graphes

Différentes problématiques

- **Analyse des interactions entre individus/institutions**

- ✓ influences/enjeux de pouvoir
- ✓ recherche de communautés
- ✓ cohésion, stabilité
- ✓ diffusion, contagion

→ **réseaux sociaux, réseaux écologiques**

Théorie des graphes

Différentes problématiques

- **Recherche de chemins**

- ✓ Tous les chemins d'un sommet à un autre

- ✓ Plus court chemin entre deux sommets

→ **réseaux de transports, de télécommunications (routage) ...**

- ✓ Chemin passant une et une seule fois par chaque arête, ou par chaque sommet d'un graphe

→ Problème du facteur chinois, ou du voyageur de commerce

→ **réseaux urbains (nettoyage de rues, collecte des déchets)**

Théorie des graphes

Différentes problématiques

- **Optimisation de réseaux** en fonction du **coût**, du **trafic** ...
 - **Construction / maintenance / évolution des réseaux**
 - **Distribution d'énergies, câblages, circuits électroniques, télécommunications**

Théorie des graphes

Différentes problématiques

- **Problèmes de flots**

→ **Acheminement – Circulation - Gestion des débits**

→ **Transport de marchandises**

→ **Réseaux de canalisations**

→ **Acheminement par paquets sur internet**

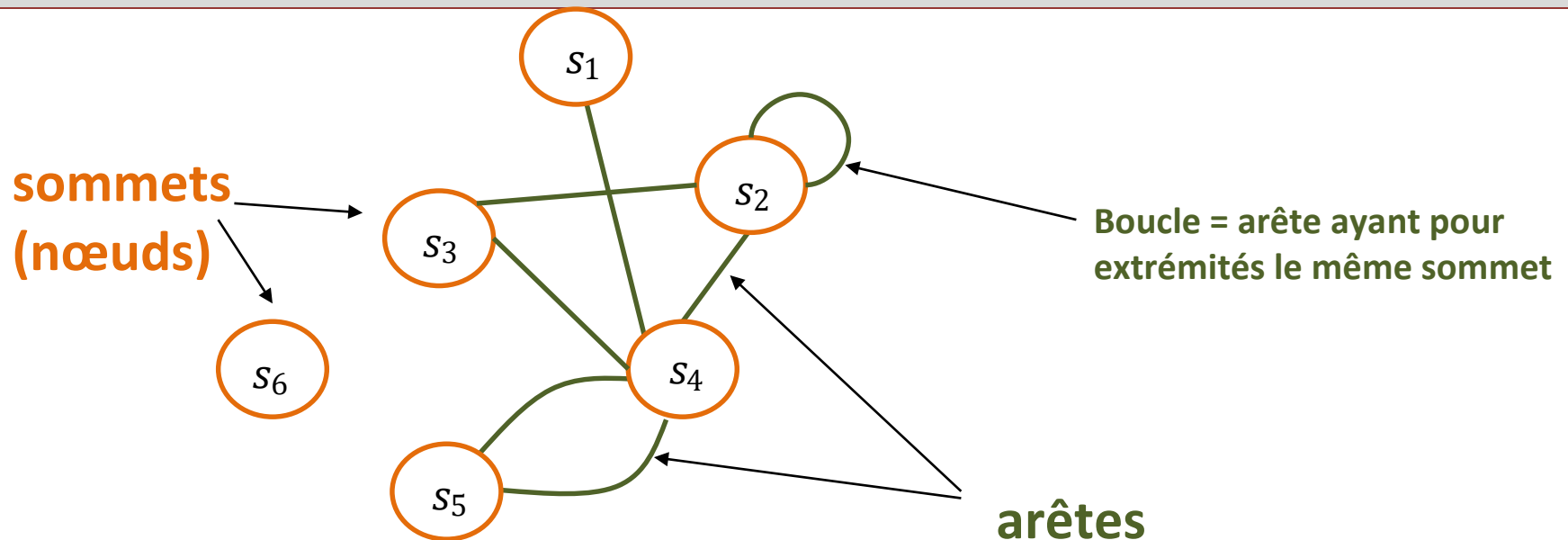
Théorie des graphes

Différentes problématiques

- **Problèmes d'incompatibilité**
- **Problèmes d'affectation**
- **Ordonnancement/planification de projets**

Théorie des graphes

II. Définitions – Propriétés



Représentation sagittale d'un multi-graphe non orienté

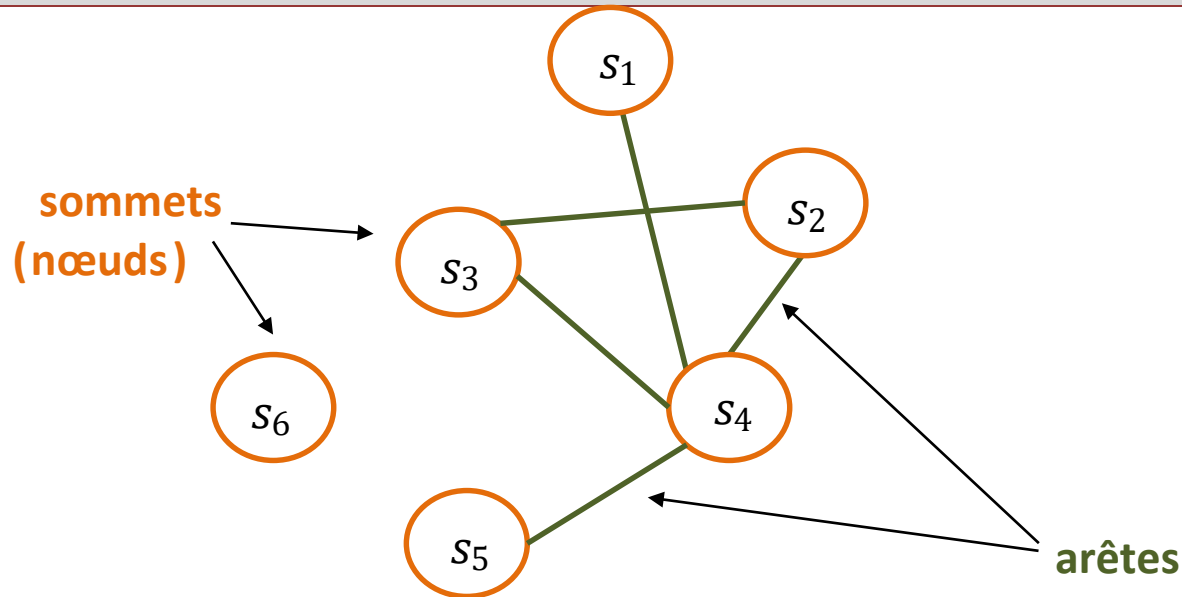
Ce graphe est un **multi-graphe** : plusieurs arêtes relient 2 mêmes sommets (s_4 et s_5) et il possède une boucle. Dans la suite de ce cours, sauf mention contraire, nous ne travaillerons que sur des **graphes simples** (pas de liens multiples entre 2 sommets et pas de boucle)

Théorie des graphes

II. Définitions – Propriétés

- Un graphe simple non orienté est un couple $G=(S,A)$, où S est un ensemble d'objets appelés les **sommets** du graphe et A est un ensemble de couples non ordonnés de sommets (s_i, s_j) appelées **arêtes**.
- Deux sommets sont dits **adjacents** (ou **voisins**) s'ils sont reliés par une arête (s'ils sont les **extrémités** d'une même arête).
L'ensemble des sommets adjacents au sommet s_i est noté **Adj(s_i)**
- L'**ordre** du graphe est le nombre de sommets
- La **taille** du graphe est le nombre d'arêtes

Graphe – Définitions



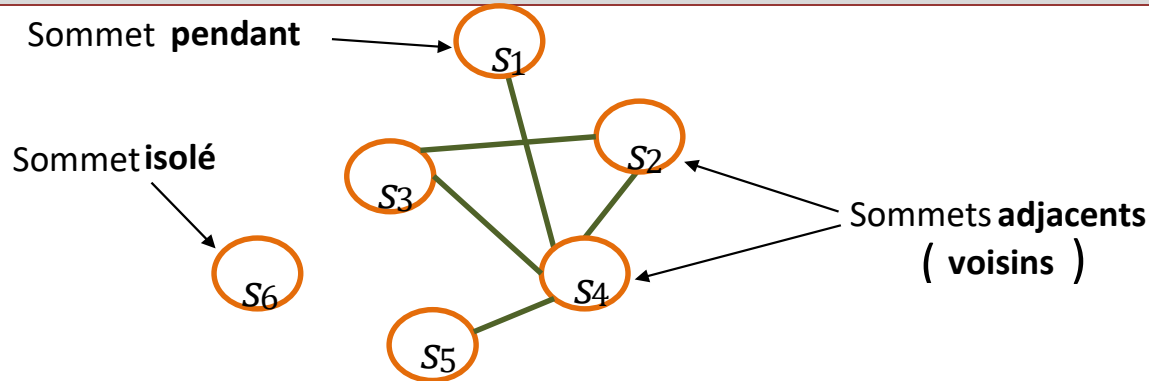
$S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$ est l'ensemble des sommets du graphe

$A = \{(s_1, s_4), (s_2, s_3), (s_2, s_4), (s_3, s_4), (s_4, s_5)\}$ est l'ensemble des arêtes du graphe

ordre du graphe = $\text{card}(S)$ = nombre de sommets = **6** **taille** du

graphe = $\text{card}(A)$ = nombre d'arêtes = **5**

Sommets et arêtes



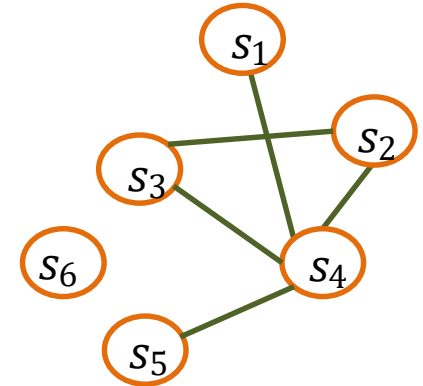
- Deux sommets sont **adjacents** s'il existe au moins une arête entre ces 2 sommets. $\rightarrow s_2$ et s_4 sont adjacents. $\text{Adj}(s_2) = \{s_3, s_4\}$
 - Un sommet est **pendant** s'il n'est adjacent qu'à un seul autre sommet. $\rightarrow s_1$ est pendant.
 - Un sommet est **isolé** s'il n'est adjacent à aucun autre sommet. $\rightarrow s_6$ est isolé.
-
- Deux arêtes sont **adjacentes** si elles ont au moins une extrémité en commun. $\rightarrow (s_2, s_3)$ et (s_3, s_4) sont adjacentes
 - Une arête est **incidente** à un sommet s si s est l'une de ses extrémités.
 \rightarrow L'arête $a = (s_1, s_4)$ est **incidente** à s_1 et à s_4 .

Degré d'un sommet

- Le **degré** d'un sommet est le nombre d'arêtes incidentes à ce sommet.

→ **degré** de $s_3 = \text{deg}(s_3)$ = nombre d'arêtes incidentes à $s_3 = 2$

Dans un graphe simple d'ordre n , le degré d'un sommet est aussi égal au nombre de sommets qui lui sont adjacents et $\text{deg}(s) \leq n-1$



Le degré est une des mesures utilisées pour étudier la **centralité** d'un nœud dans un réseau. Il donne une indication sur l'importance du nœud dans le système : aptitude à capter ce qui se passe à proximité sur le réseau (information, virus ...)

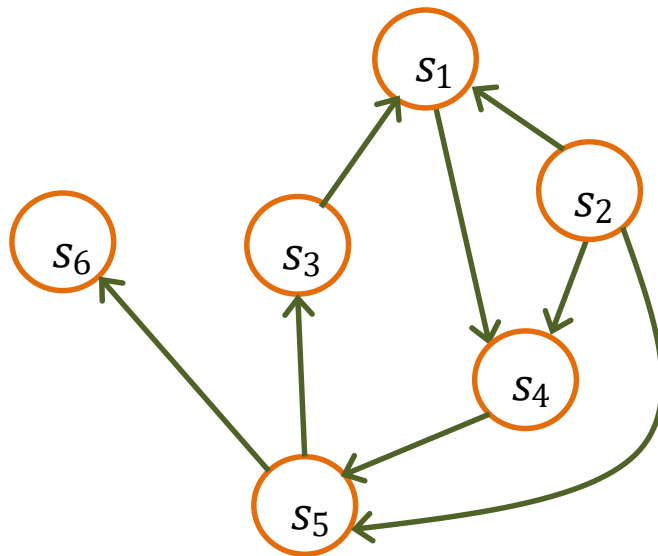
- ✓ influence
- ✓ popularité

Graphe orienté

Dans un graphe **orienté**, les arêtes sont des couples ordonnés de sommets. Elles ont un sens, on les appelle des **arcs**.

Soit $a = (s_i, s_j)$ un arc :

a est **sortant** en s_i , **entrant** en s_j . s_i est le **prédécesseur** de s_j . s_j est le **successeur** de s_i



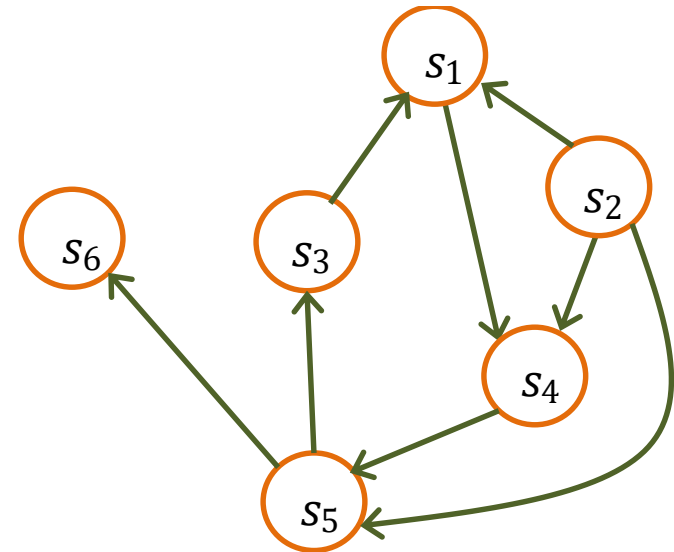
Pred(s_4) = { s_1 , s_2 } **Succ**(s_4) = { s_5 }

Graphe orienté

Demi-degré extérieur (sortant) et demi-degré intérieur (entrant)

- Le **demi-degré extérieur** $\text{deg}^+(s)$ d'un nœud s est le nombre d'arcs sortants de s .
- Le **demi-degré intérieur** $\text{deg}^-(s)$ d'un nœud s est le nombre d'arcs entrants en s .

s	$\text{deg}^+(s)$	$\text{deg}^-(s)$	$\text{deg}(s)$
s_1	1	2	3
s_2	3	0	3
s_3	1	1	2
s_4	1	2	3
s_5	2	2	4
s_6	0	1	1



Formule de la somme des degrés Le lemme des poignées de main

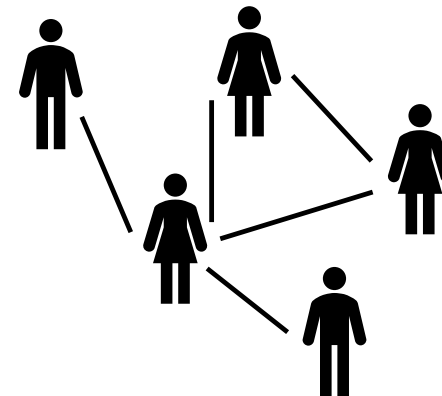
$$\sum_{s \in S} \deg(s) = 2 \times \text{nombre d'arêtes}$$

Démonstration : chaque arête est comptée deux fois dans la somme des degrés (une fois pour chacune de ses extrémités)



Problème :

Le nombre de personnes sur Terre qui ont donné un nombre impair de poignées de main est-il pair ou impair ?



Lemme des poignées de main

Le nombre de personnes sur Terre qui ont donné un nombre impair de poignées de main est-il pair ou impair ?

Représentation du problème :

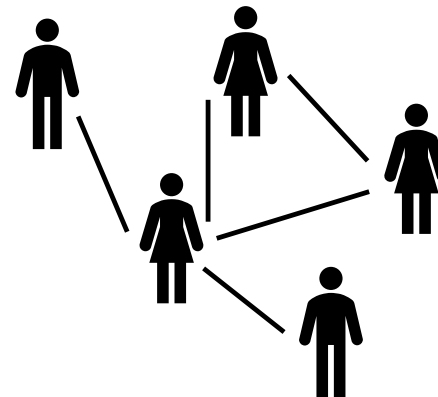
graphe non orienté 1 sommet = 1 personne
1 arête = 1 poignée de mains

Résolution

$\sum_{s \in S} \deg s = 2 \times \text{nombre d'arêtes}$ donc la somme des degrés

La somme des degrés impairs est la somme de tous les degrés moins la somme des degrés pair.

La différence de deux nombres pairs est paire.

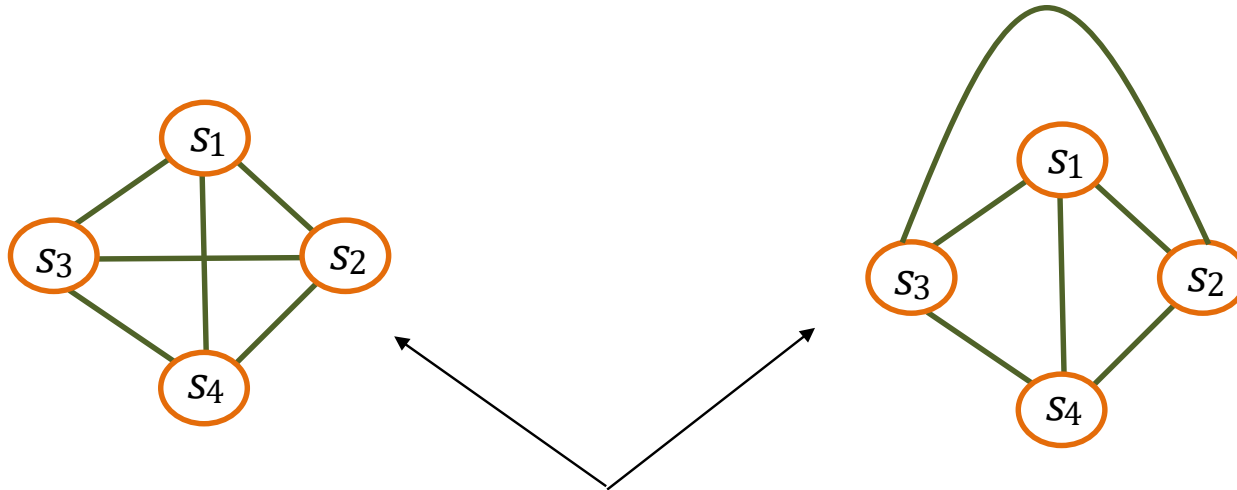


est paire.

→ Le nombre de personnes sur Terre qui ont donné un nombre impair de poignées de main est forcément pair !

Dessiner un graphe

! Ne pas confondre le graphe et sa représentation sagittale (son dessin)



Un seul et même graphe :

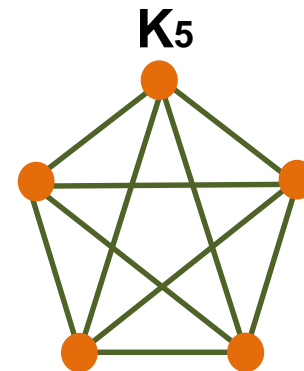
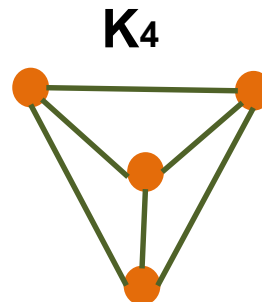
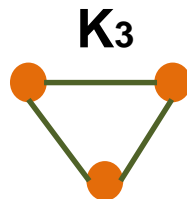
$$S = \{s_1, s_2, s_3, s_4\}$$

$$A = \{(s_1, s_2), (s_1, s_3), (s_1, s_4), (s_2, s_3), (s_2, s_4), (s_3, s_4)\}$$

Graphes complets

- Un graphe simple est **complet** si et seulement si ses sommets sont 2 à 2 adjacents

On note K_n le graphe complet d'ordre n .



K_n possède $n(n-1)/2$ arêtes $((n-1) + (n-2) + \dots + 1)$

Densité d'un graphe

- La **densité** d'un graphe $G=(S,A)$ d'ordre n est le rapport entre sa taille (son nombre d'arêtes) et le nombre maximal d'arêtes qu'il pourrait avoir (s'il était complet).

$$\text{dens } (G) = \frac{\text{taille } (G)}{\text{taille } (K_n)} = \frac{2\text{card}(A)}{n(n-1)}$$

Un graphe **creux** est un graphe de faible densité, contrairement à un graphe **dense**.

La densité est un indicateur pour l'analyse structurelle d'un graphe :

- ✓ Mesure de cohésion et de coopération dans les réseaux sociaux
- ✓ Indicateur de stabilité (résistance aux perturbations) dans les réseaux trophiques

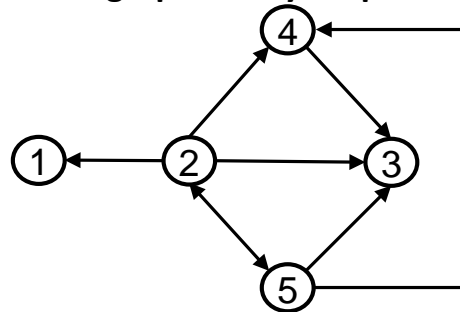
Programmation :

- ✓ Les structures de données utilisées pour stocker des graphes peuvent être adaptées à leur densité.
- ✓ Certains algorithmes sont construits pour être efficaces sur des graphes creux mais sont plutôt mauvais si on les utilise sur des graphes denses, et réciproquement.

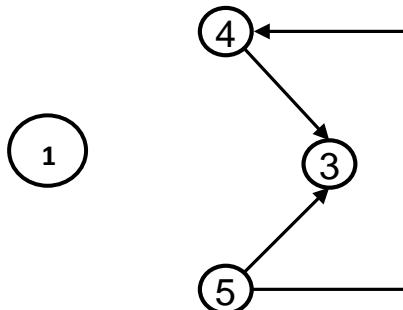
Sous-graphes – Graphes partiels

Soit $G=(S,A)$ un graphe.

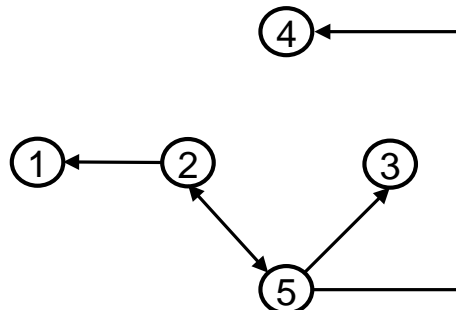
- Un **sous-graphe** de G est un graphe G' ayant pour sommets un sous-ensemble S' des sommets de G
- Un **graphe partiel** de G est un graphe G' ayant pour arêtes un sous-ensemble A' des arêtes de G



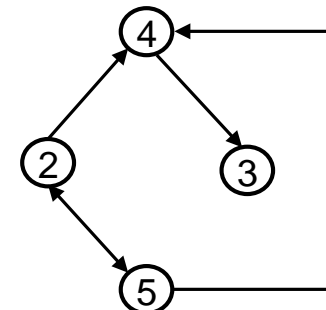
Un graphe G



Sous-graphe de G obtenu en ôtant le sommet 2



Un graphe partiel de G



Un sous-graphe partiel de G

Théorie des graphes

III. Coloration d'un graphe



Problème :

On doit stocker 8 produits chimiques $p_1, p_2 \dots p_8$ mais pour des raisons de sécurité certains produits ne peuvent pas être stockés dans le même hangar :

p_1 ne peut pas être stocké avec p_4 p_2 ne peut pas être stocké avec p_3 p_3 ne peut pas être stocké avec p_2, p_4 ou p_5 p_4 ne peut pas être stocké avec p_1, p_3 ou p_7 p_5 ne peut pas être stocké avec p_3 ou p_6 p_6 ne peut pas être stocké avec p_5 p_7 ne peut pas être stocké avec p_4 ou p_8 p_8 ne peut pas être stocké avec p_3 ou p_7

Combien de hangars faut-il et comment y répartir les produits ?

Résolution :

1. Représenter le problème sous-forme de graphe.

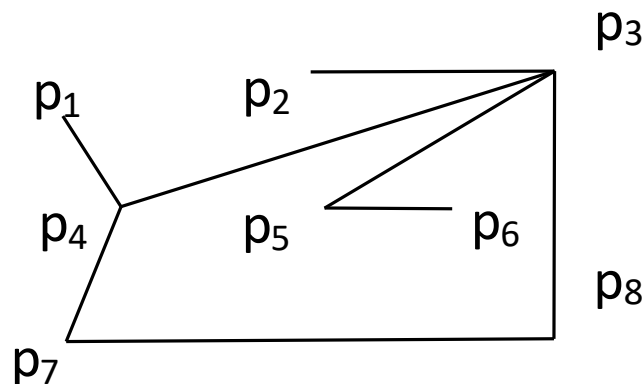
Sommets = produits

Relation = « n'est pas compatible avec »

(Deux produits sont adjacents s'ils ne peuvent pas être stockés dans le même hangar.) La relation est symétrique
: **graphe non orienté**

2. Attribuer un hangar à chaque sommet de telle sorte que 2 sommets reliés par une arête n'aient pas le même hangar.

C'est un problème de **coloration du graphe**.



Coloration d'un graphe

Définitions

- Étant donné un graphe simple $G = (S, A)$, une **coloration propre** de sommets est une fonction f qui attribue une couleur (valeur) à chaque sommet de sorte que deux sommets adjacents aient des couleurs différentes : $\forall (x, y) \in A, f(x) \neq f(y)$
- Un graphe est **k-colorable** s'il existe une coloration propre avec k couleurs
- Le nombre minimum de couleurs nécessaires pour obtenir une coloration propre est appelé le **nombre chromatique** de G et noté $\chi(G)$. C'est le plus petit entier k pour lequel le graphe est k -colorable.

Coloration d'un graphe

Problématiques :

1. Etant donné un graphe simple G d'ordre n , comment trouver une coloration propre de G ? *
2. Etant donné un entier $k < n$, G est-il k -colorable ? **
3. Quel est le nombre chromatique de G ? *
4. Peut-on trouver une coloration propre minimale ? **

Domaines d'applications :

- Cartographie
- Planification (réunions, emploi du temps ...)
- Transports, stockages (problèmes d'incompatibilité)
- Télécommunications : attribution de fréquences
- Informatique : allocation de registres

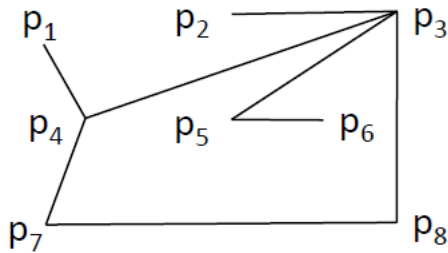
* *Problème de production (trouver une solution)*

** *Problème de décision (déterminer si une solution existe)*

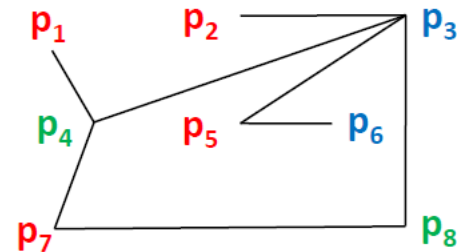
Trouver une coloration propre

A. Algorithme naïf : coloration séquentielle

- Parcourir les sommets les uns après les autres :
- Attribuer à chaque sommet la « plus petite » couleur non déjà utilisée pour un de ses sommets adjacents.



sommets	couleur
p_1	• 1
p_2	• 1
p_3	• 2
p_4	• 3
p_5	• 1
p_6	• 2
p_7	• 1
p_8	• 3



On obtient 3 couleurs. Mais est-ce la meilleure solution ?

A. Algorithme naïf : coloration séquentielle

- Parcourir les sommets les uns après les autres :
- Attribuer à chaque sommet la « plus petite » couleur non déjà utilisée pour un de ses sommets adjacents.

Si n et m sont respectivement l'ordre (nombre de sommets) et la taille (nombre d'arêtes) du graphe, la complexité de l'algorithme séquentielle est en $O(n+m)$

C'est un algorithme glouton*. La solution obtenue n'est pas forcément optimale.


La coloration obtenue dépend de la numérotation des sommets (de l'ordre de parcours)

** algorithme glouton : on fait un choix optimum localement dans l'espoir de trouver une solution globale optimale*


On peut améliorer l'algorithme séquentiel en choisissant un ordre de parcours judicieux.

B. Algorithme de Welsh et Powell

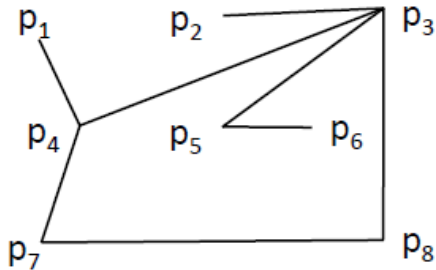
L'idée est que les sommets qui ont beaucoup de voisins sont plus difficiles à colorer et qu'il faut donc les colorer en premier.

- 
1. Trier la liste des sommets par ordre décroissant de degrés
 2. Parcourir la liste triée :
 - Trouver le premier sommet non déjà coloré et lui attribuer la « plus petite » couleur c non déjà utilisée.
 - Parcourir la suite de la liste en attribuant cette couleur c aux sommets non colorés et non adjacents aux sommets déjà colorés avec c
 3. Recommencer en 2 s'il reste des sommets non colorés

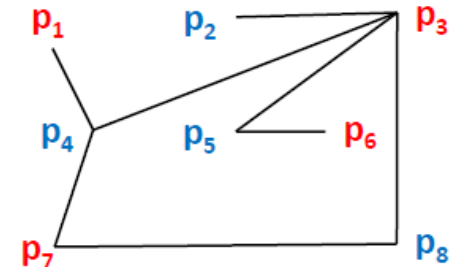
B. Algorithme de Welsh et Powell

1. Trier la liste des sommets par ordre décroissant de degrés
 2. Parcourir la liste triée :
 - Trouver le premier sommet non déjà colorié et lui attribuer la « plus petite » couleur c non déjà utilisée.
 - Parcourir la suite de la liste en attribuant cette couleur c aux sommets non coloriés et non adjacents aux sommets déjà coloriés avec c
 3. Recommencer en 2 s'il reste des sommets non coloriés
- 

B. Algorithme de Welsh et Powell



sommets	degrés	tour 1	tour 2
p_3	4	• 1	
p_4	3		• 2
p_5	2		• 2
p_7	2	• 1	
p_8	2		• 2
p_1	1	• 1	
p_2	1		• 2
p_6	1	• 1	



On obtient 2 couleurs.
C'est la meilleure
solution puisqu'il y a
des sommets adjacents

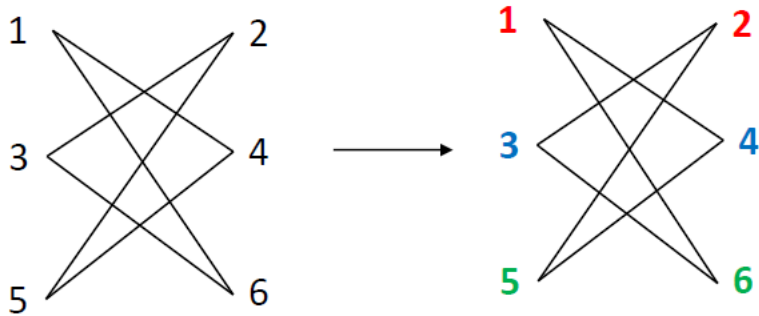
Mais l'algorithme donne-t-il toujours le nombre chromatique ?

B. Algorithme de Welsh et Powell

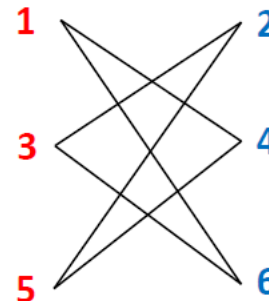
En utilisant un tri par dénombrement, on obtient une complexité de l'ordre de $O(n+m)$.

C'est un algorithme glouton. **La solution obtenue n'est pas forcément optimale : le nombre de couleurs obtenu n'est pas forcément le nombre chromatique.**

Graphe couronne à 6 sommets



L'algorithme donne 3 couleurs alors que le nombre chromatique est 2



Coloration d'un graphe

- ✓ **Algorithme « systématique » pour déterminer si un graphe est k-colorable**

Soit $G=(S,A)$ un graphe simple d'ordre n .

Une k -coloration de G est une application surjective $c : S \rightarrow \{1, \dots, k\}$

Le nombre de combinaisons à tester est donc le nombre de surjections de $\{1, \dots, n\}$ dans $\{1, \dots, k\}$:

$$S_{n,k} = \sum_{i=0}^k (-1)^{k-i} \binom{k}{i} i^n$$

- ✓ **Algorithme « systématique » (Brute Force) pour trouver une coloration propre optimale et déterminer le nombre chromatique**

Tester toutes les colorations possibles avec 1,2,3 ... couleurs jusqu'à trouver une coloration propre.

Théorie des graphes

IV. Représentation et modélisation des graphes

A. Représentation par matrice d'adjacence

Soit $G=(S,A)$ un graphe simple d'ordre n .

La matrice d'adjacence M de G est la matrice carrée d'ordre n définie par :

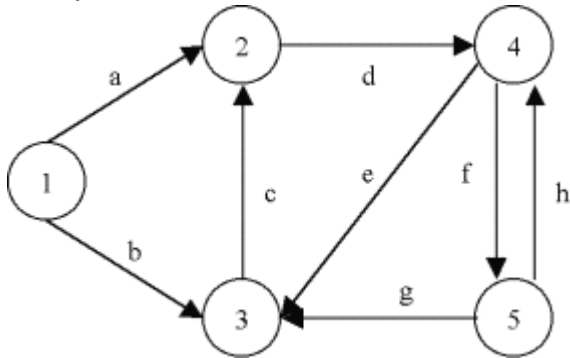
$$m_{i,j} = \begin{cases} 1 & \text{si } (v_i, v_j) \in A \\ 0 & \text{si } (v_i, v_j) \notin A \end{cases}$$

Si le graphe est non orienté, la matrice est symétrique.

IV. Représentation et modélisation

A. Représentation par matrice d'adjacence

Exemple :



Le graphe précédent sera représenté par la matrice d'adjacence (sans pondération) suivante.

	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	1	0
3	0	1	0	0	0
4	0	0	1	0	1
5	0	0	1	1	0

A. Représentation par matrice d'adjacence

Remarques :

- Seulement m cases de la matrice sont non nulles sur n^2 cases.
- Cette représentation est efficace au niveau de l'espace mémoire utilisé lorsque le graphe est suffisamment dense (i.e. lorsqu'il y a suffisamment d'arcs).
- Elle permet d'implémenter assez facilement beaucoup d'algorithmes.
- Deux arcs ayant les mêmes extrémités ne peuvent pas être représentés avec cette matrice
- Une matrice d'adjacence pondérée contient les poids des arêtes ou arcs reliant deux nœuds
- Dans le cas d'un graphe non orienté, la matrice est symétrique.

Structures de stockage en C :

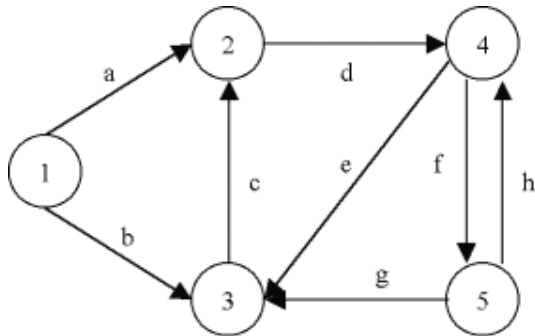
La définition d'un graphe avec une matrice d'adjacence :

```
int **adjacence ; // matrice dynamique d'adjacence nœud-nœud
```

B. Tableau dynamique d'arêtes

Un graphe peut être représenté en mémoire par un tableau dynamique dont la taille allouée est de m cases. Chaque case est associée à une arête composée de 3 valeurs : les 2 nœuds adjacents et le poids de l'arête (1 par défaut). Ainsi, une case indique la relation qu'il existe entre deux nœuds.

Exemple



Ci-dessous, un extrait du tableau des arêtes :

	x	y	poids
1	1	2	5
2	1	3	1
3	2	4	2
4	3	2	4
5	4	3	1
6	4	5	7
7	5	3	9
8	5	4	1

B. Tableau dynamique d'arêtes

Remarques

- m cases de la matrice remplies.
- Cette représentation occupe beaucoup de place en mémoire.
- Son utilisation est utile dans quelques cas rares : algorithmes de Kruskal et de Prim pour la recherche de l'arbre couvrant d'un graphe de poids minimum.

Structures de stockage en C :

La définition d'une arête :

```
typedef struct arete
{
    int x, y ; // les 2 sommets d'une arête
    float poids ; // poids d'une arête par défaut à 1
} t_arete;
```

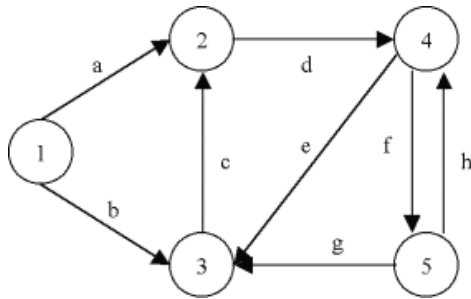
La définition d'un tableau dynamique d'arêtes:

```
t_arete * incidence ; // tableau dynamique d'arêtes
```

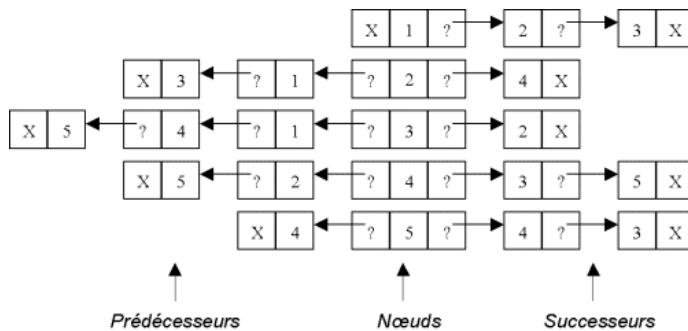
C. Files d'attentes avec des listes chaînées

Un graphe peut être représenté par des files d'attente de nœuds mémorisées avec des listes chaînées. Nous proposons ici une possibilité mais de nombreuses autres peuvent convenir. On définit tout d'abord une liste des nœuds et à chaque nœud, on associe une liste de nœuds successeurs et une liste de nœuds prédécesseurs.

Exemple :



Le graphe précédent sera représenté de la manière suivante.



C. Files d'attentes avec des listes chaînées

Remarques :

- Cette représentation est nettement plus efficace au niveau de la mémoire occupée que la représentation précédente.
- Elle est très bien adaptée au parcours du graphe, aussi bien en sens direct qu'en sens indirect.
- Cette structure est souple. L'ajout et la suppression d'arcs ou de nœuds sont plus aisés qu'avec la représentation matricielle.

Structures de stockage en C :

Pour modéliser un graphe sous forme de files d'attentes codées avec des listes chaînées, il y a trois structures de stockage :

- Un nœud
- Le graphe est un tableau dynamique à 2 entrées composé de files d'attentes de nœuds

Définition d'un nœud :

```
typedef struct noeud
{
    int data; // numéro de sommet
    struct noeud *suivant ; // pointeur sur le nœud suivant
} t_noeud;
```

Définition du graphe :

```
t_noeud **graphe[2] ; // graphe de 2 listes (prédécesseurs et successeurs) de noeuds
```

Annexe 1

Complexité algorithmique

Efficacité (performance) d'un algorithme :

- Quantité d'espace mémoire utilisé
 - Rapidité d'exécution
- **complexité** = nombre d'instructions à exécuter en fonction de la quantité n de données traitées (ordre de grandeur)
- *complexité linéaire $O(n)$, logarithmique $O(\ln(n))$, polynomiale $O(n^k)$, exponentielle $O(k^n)$...*

Estimations (approximatives) du temps d'exécution en fonction de la complexité algorithmique et de la quantité de données n pour des processeurs exécutant respectivement **1 million d'opérations par seconde (1 MIPS)** (années 70) et 10^4 MIPS (années 2010)

Complexité	$\log_2(n)$		n		$n \log_2(n)$		n^2		n^3		$1,5^n$		2^n		$n!$	
n																
10	<1 ms	<1 □s	<1 ms	<1 □s	<1 ms	<1 □s	<1 ms	<1 □s	1 ms	<1 □s	<1 ms	<1 □s	1 ms	<1 ms	3 s	362 □s
15	<1 ms	<1 □s	<1 ms	<1 □s	<1 ms	<1 □s	<1 ms	<1 □s	3 ms	<1 □s	<1 ms	<1 □s	32 ms	3 □s	15 j	2 min
20	<1 ms	<1 □s	<1 ms	<1 □s	<1 ms	<1 □s	<1 ms	<1 □s	8 ms	<1 □s	3 ms	<1 □s	1s	<1 ms	□	7 ans
30	<1 ms	<1 □s	<1 ms	<1 □s	<1 ms	<1 □s	1 ms	<1 □s	27 ms	<1 ms	191 ms	<1 ms	17 min	<1 s	□	□
50	<1 ms	<1 □s	<1 ms	<1 □s	<1 ms	<1 □s	2 ms	<1 □s	125 ms	<1 ms	10 min	63 ms	35 ans	1 jour		
100	<1 ms	<1 □s	<1 ms	<1 □s	<1 ms	<1 □s	10 ms	1 □s	1 s	100 □s	12 000 ans	1 an	□	□		
1000	<1 ms	<1 □s	1 ms	<1 □s	6 ms	<1 □s	1 s	<1 ms	16 min	100 ms	□	□				
10^4	<1 ms	<1 □s	10 ms	1 □s	100 ms	9 □s	1 min	10 ms	11 jours	1 min	□	□				
10^5	<1 ms	<1 □s	100 ms	<1 ms	1 s	<1 ms	2 h	1 s	31 ans	1 jour						
10^6	<1 ms	<1 □s	1 s	<1 ms	13 s	1 ms	11 jours	1 min	31 000 ans	3 ans						
$2 \cdot 10^6$	<1 ms	<1 □s	2 s	<1 ms	29 s	2 ms	46 jours	6 min	□	25 ans						
10^7	<1 ms	<1 □s	10 s	1 ms	2 min	16 ms	3 ans	2h30	□	3000 ans						
10^9	<1 ms	<1 □s	> 16 min	100 ms	> 5h30	2 s	31 000 ans	3 ans	□	□						

∞ : > 10^5 ans. Un algorithme de complexité exponentielle est très vite inutilisable !