Graphes (1)

INF 431
Algorithmes, Réseaux, Langages

X 2009

Amphi 2 - 2 février 2011

Benjamin Werner

Prochaines séances

2/2	Graphes 1	B.Werner	TD
9/2	Graphes 2	B.Werner	PC
16/2	Graphes 3	B.Werner	TD
2/3	Graphes 4	B.Werner	TD
9/3	Programmation Dynamique	B.Werner	PC
16/3	Branch and Bound	Leo Liberti	PC
23/3	Réparti 1	Thomas Clausen	TD

Prochaines séances

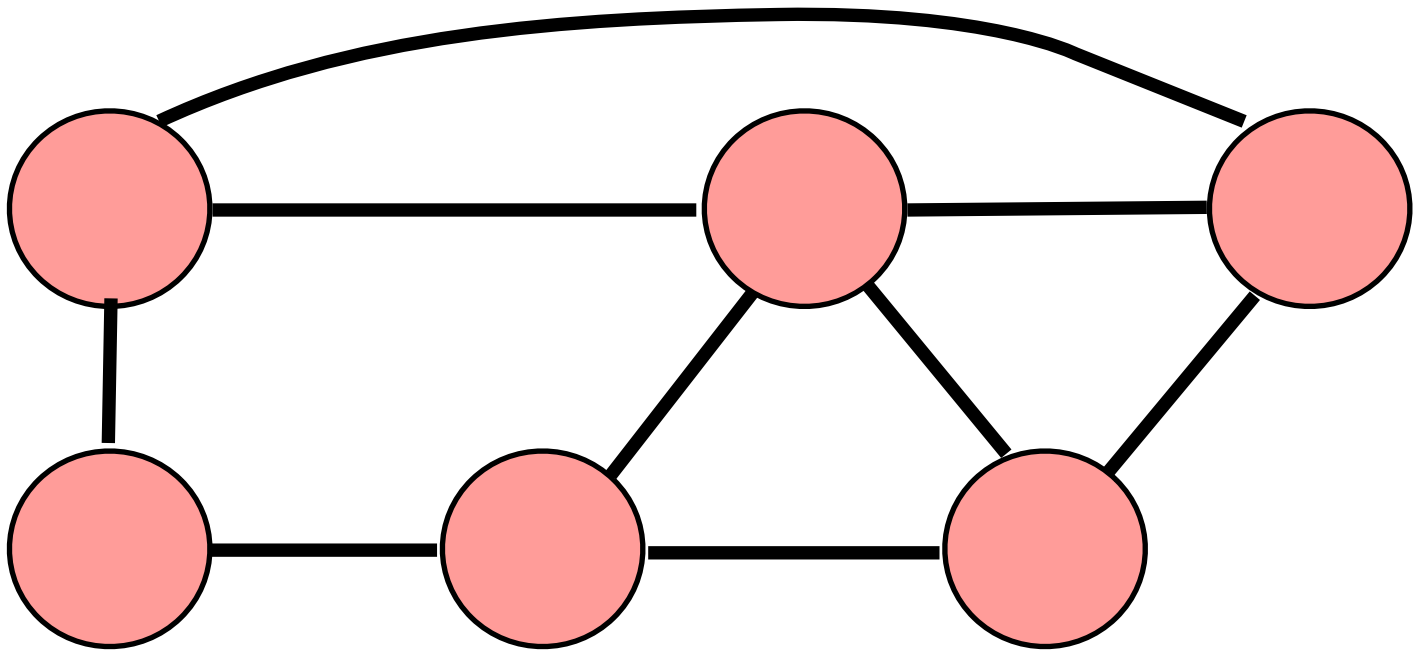
2/2	Graphes 1	B.Werner	TD
9/2	Graphes 2	B.Werner	PC
16/2			TD
2/3			TD
9/3	Programmation Dynamique	B.Werner	PC
16/3	Branch and Bound	Leo Liberti	PC
23/3	Réparti 1	Thomas Clausen	TD

Délégués ?

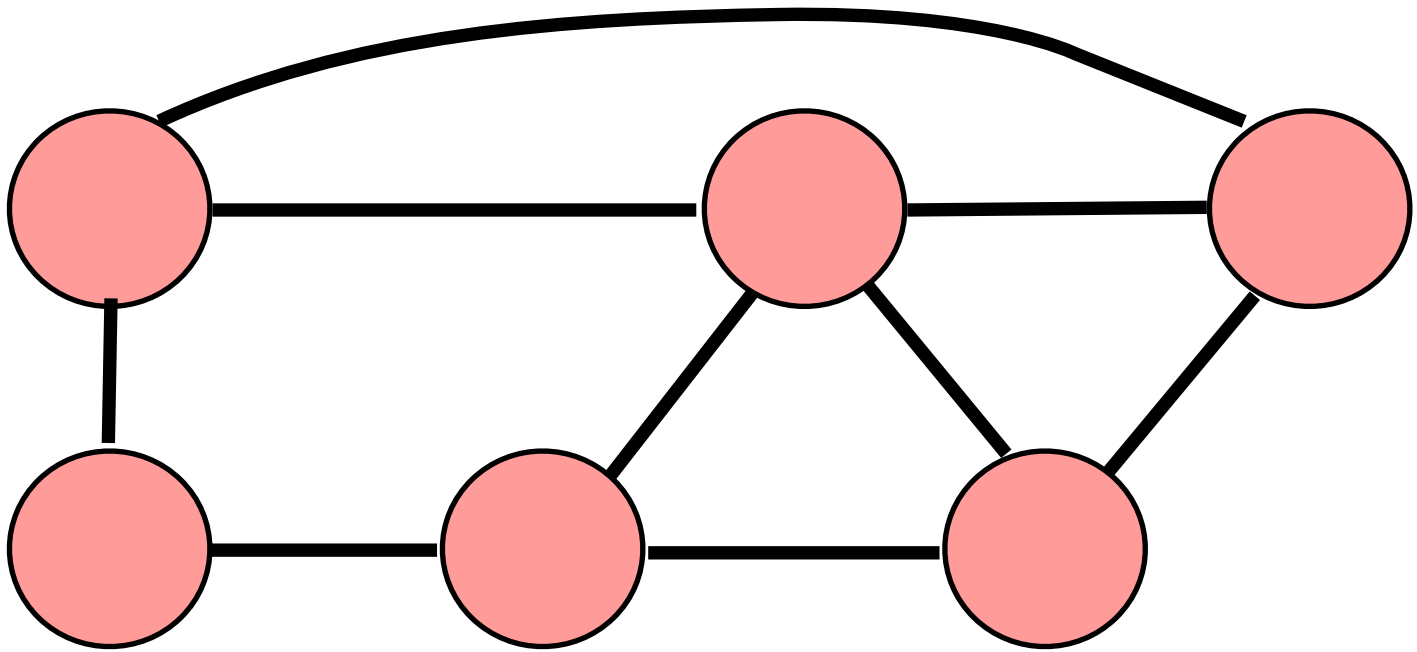
plan

1. Graphes: définitions, généralités
2. Composantes connexes, Union-find
3. Parcours simples: DFS

Pourquoi des Graphes ?



Pourquoi des Graphes ?

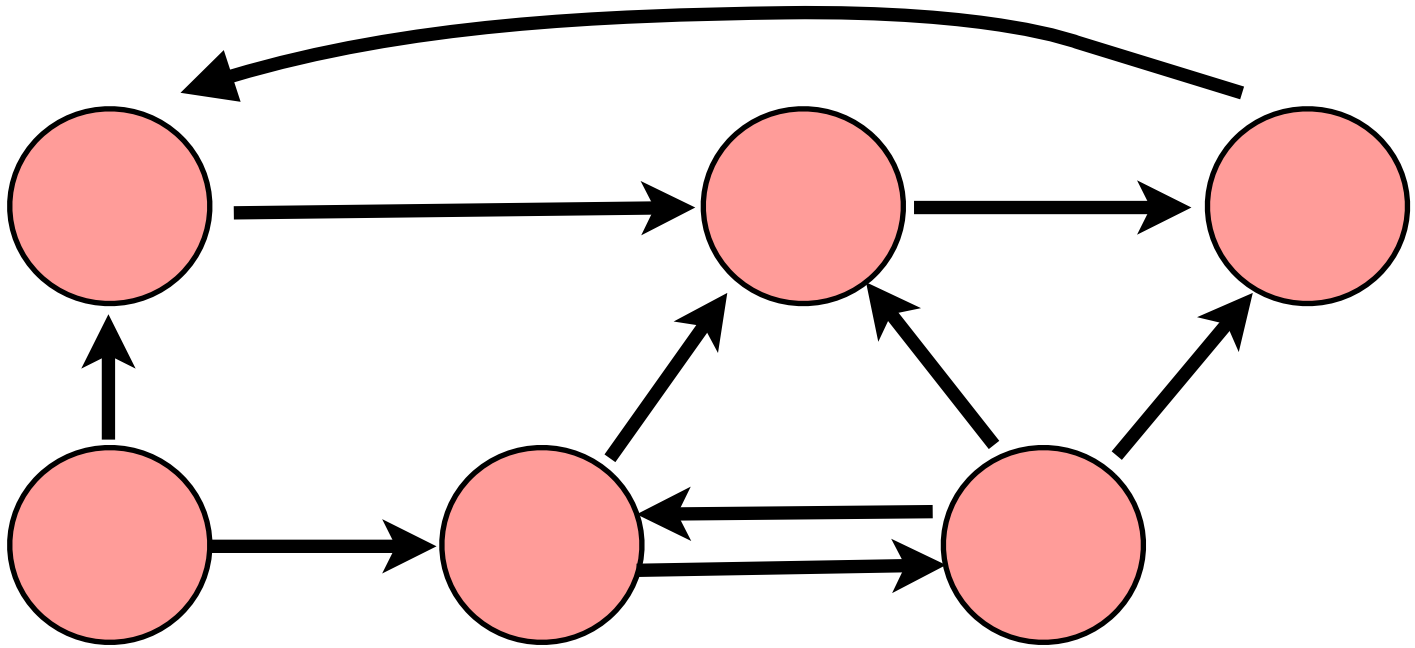


Facebook, google, les GPS, les embouteillages...

de vrais problèmes théoriques et pratiques, de vraies solutions

des exercices

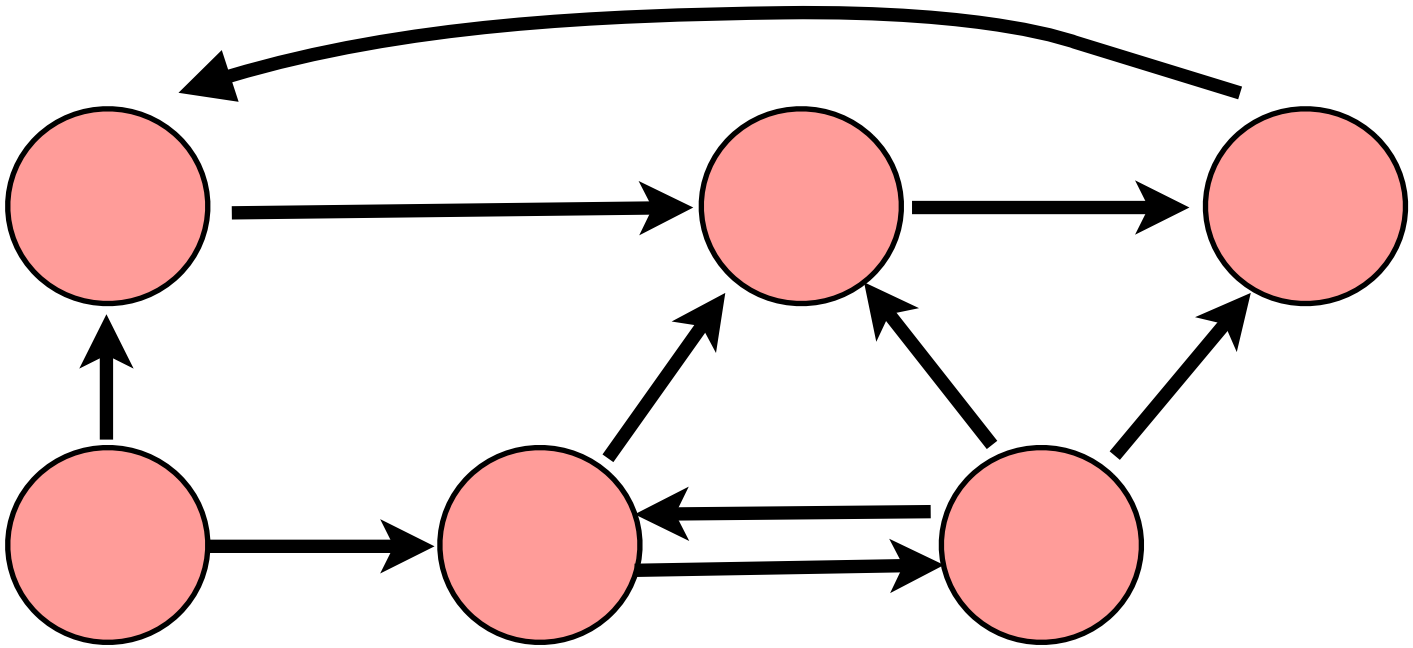
Les Graphes



Ensemble $\mathcal{S}$ de sommets

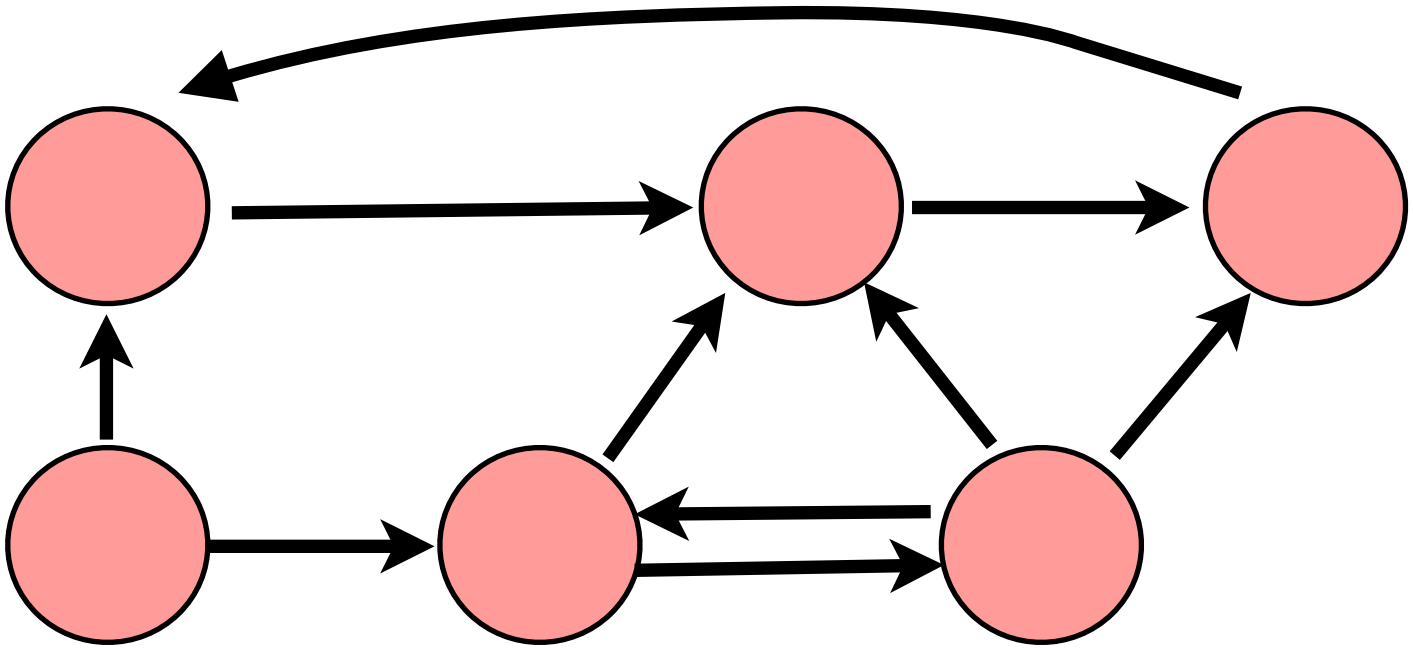
$\mathcal{A} \subset \mathcal{S} \times \mathcal{S}$ les arcs

Les Graphes



Fini / infini

Les Graphes



Fini / infini

En général dans le cours

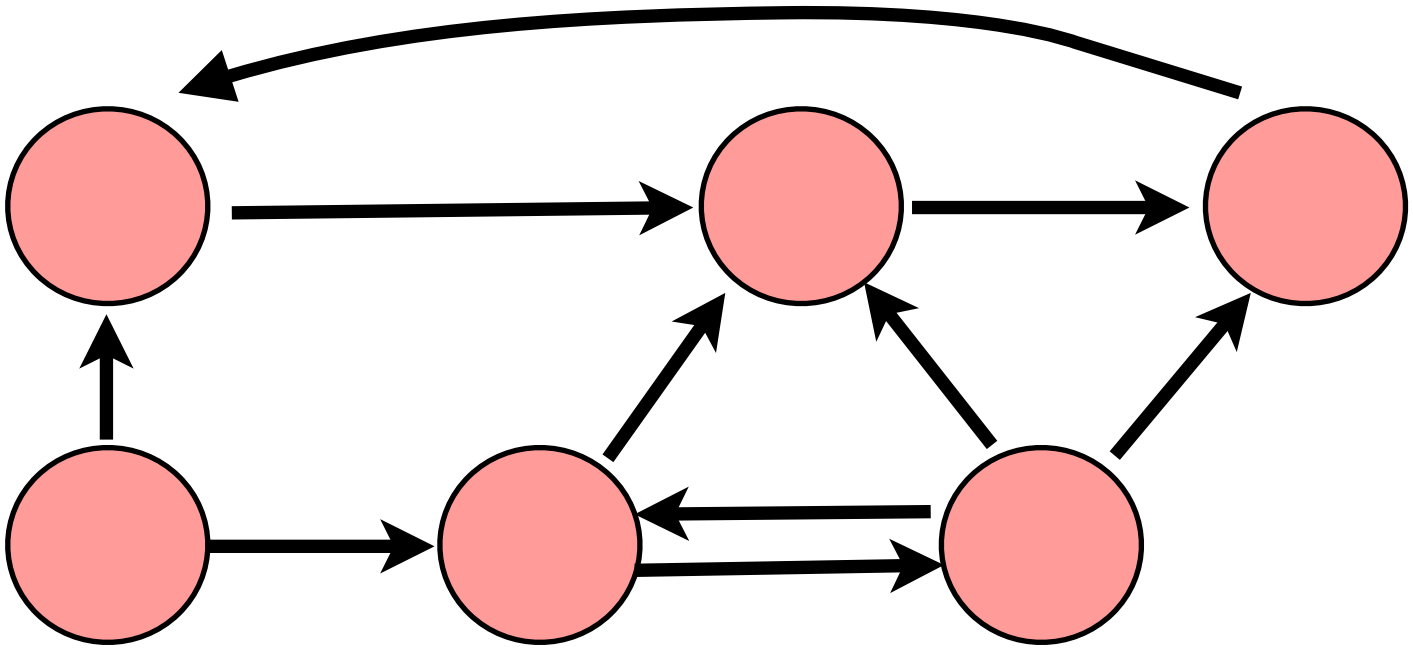
Dimensions

- Nombre de noeuds: n
- Nombre d'arcs / arêtes : a

On a $a \leq n^2$

La complexité des algorithmes sera typiquement fonction de a et n

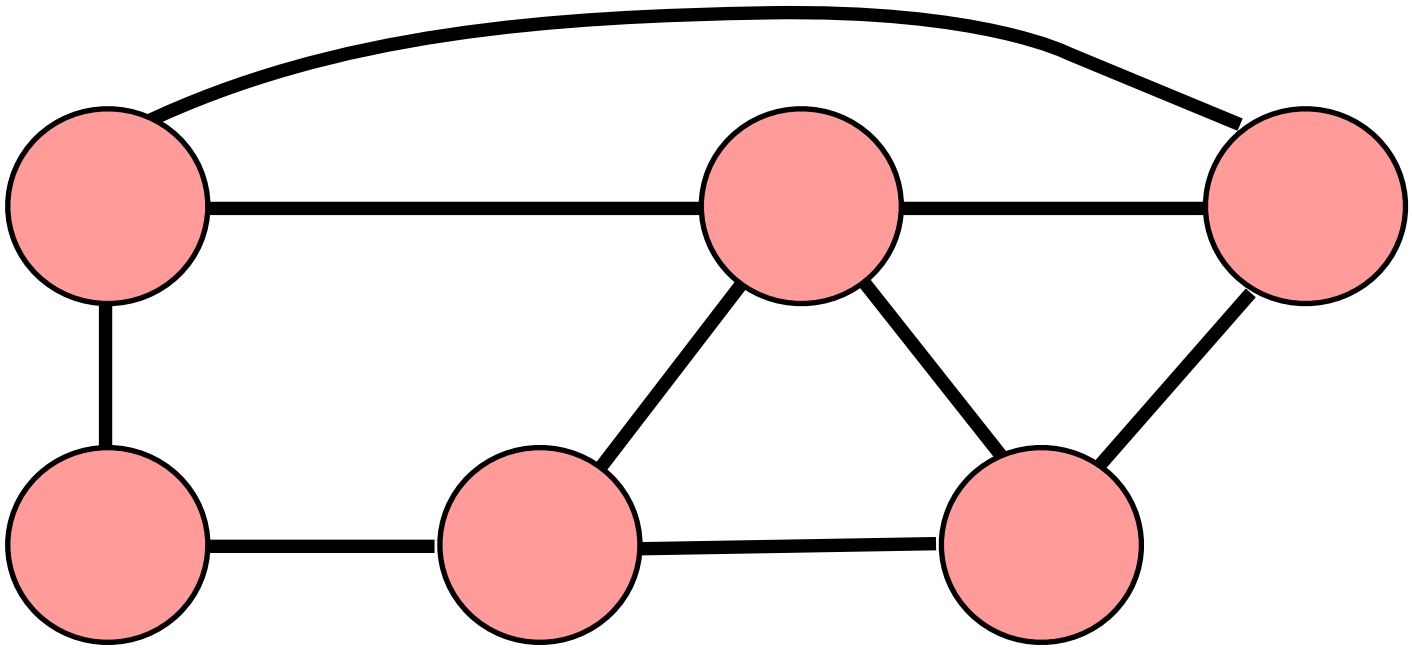
Les Graphes



Fini / infini

Orienté / non-orienté

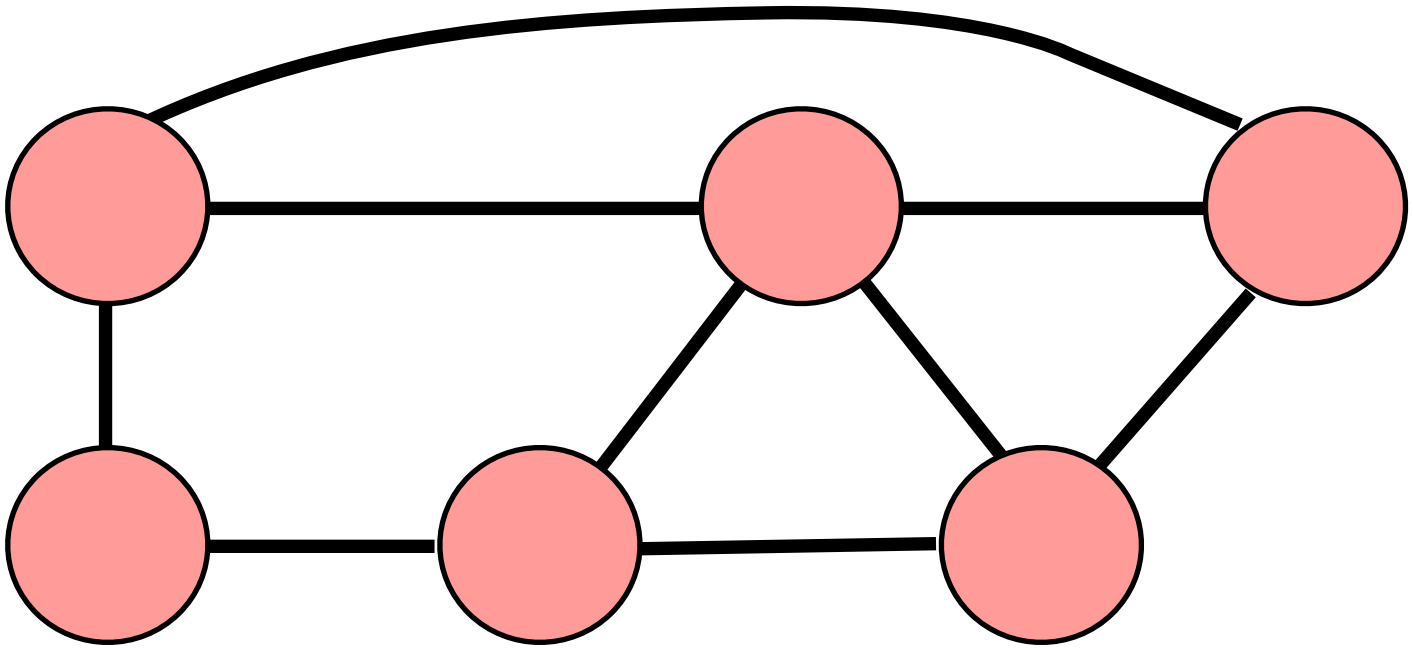
Les Graphes



Fini / infini

Orienté / non-orienté

Les Graphes

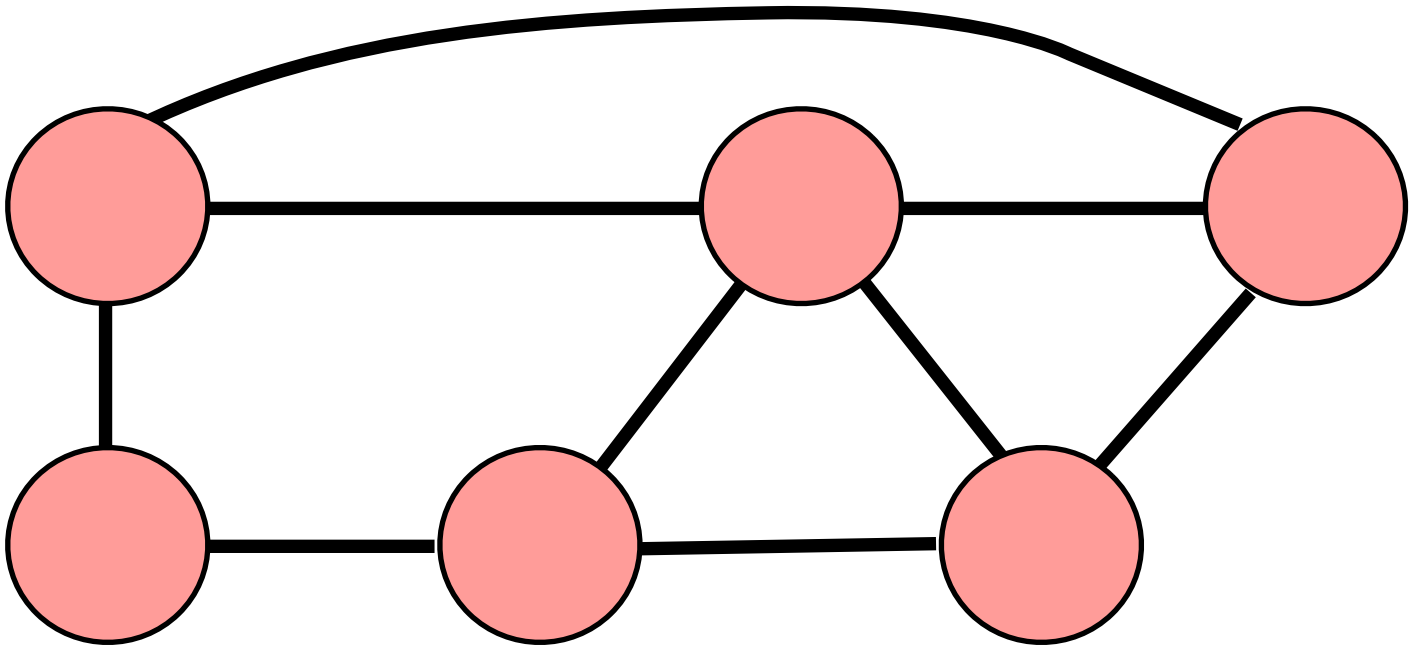


Fini / infini

~~Arcs~~ - Arêtes

Orienté / non-orienté

Les Graphes

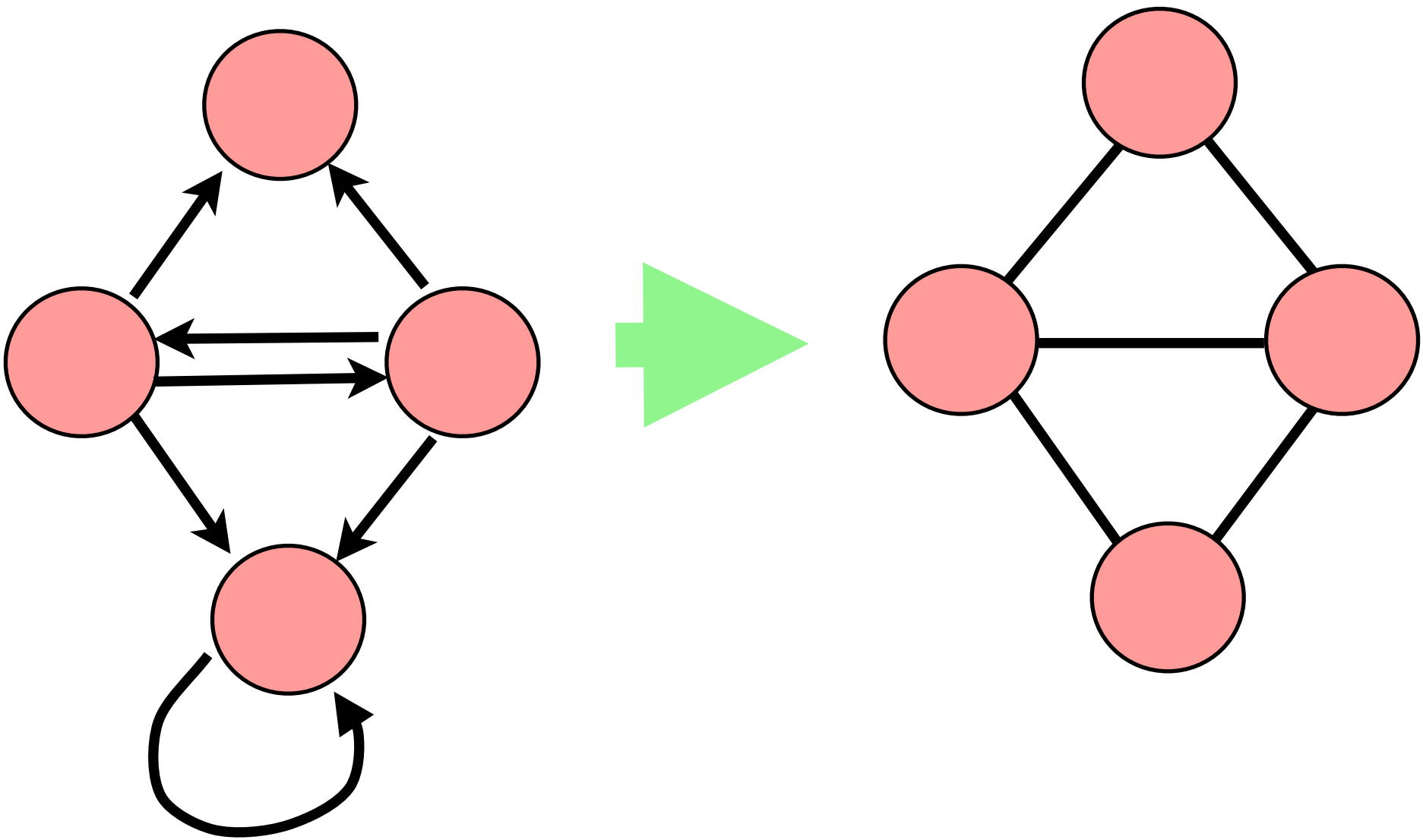


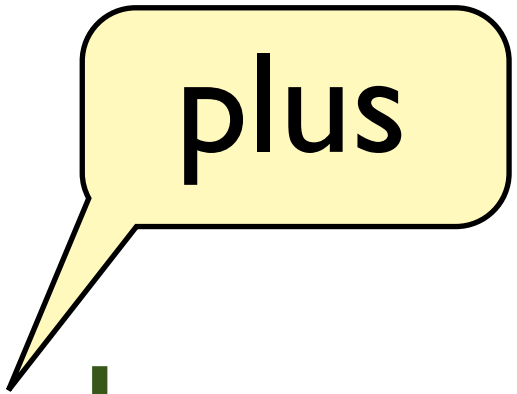
Non-orienté, cas particulier:

(a,b) $\in \mathcal{A} \Leftrightarrow$ **(b,a)** $\in \mathcal{A}$

Graphe sous-jacent

Graphe non-orienté obtenu en enlevant les flèches et les boucles :





Un peu de vocabulaire

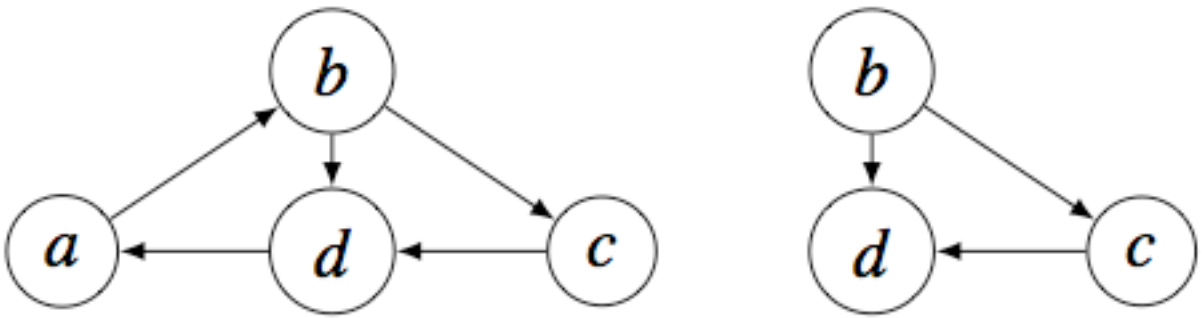
Terminologie

$\alpha=(a,b)$ est orienté de a vers b;
α a pour origine a et destination b;
α est incident à a ainsi qu'à b;
a est un prédécesseur de b, et b un successeur de a;
a et b sont adjacents.

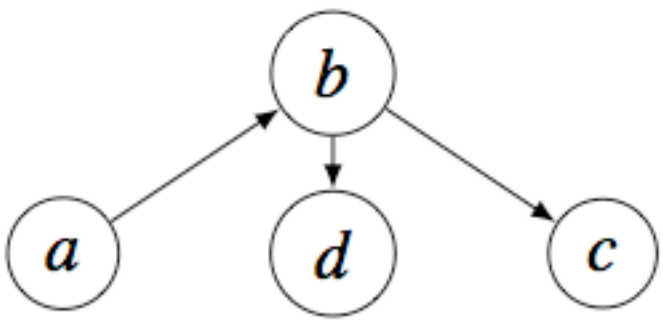
Graphe simple s'il ne comporte pas de boucles ou d'arcs multiples.
multigraphe s'il comporte des arcs multiples (mais pas de boucles).
L'arité (ou degré) de $x \in S$ est le nombre d'arcs partant ou arrivant en x.

Sous-graphe

Si $\mathcal{G} = (\mathcal{S}, \mathcal{A})$, $\mathcal{S}' \subset \mathcal{S}$, on note $\mathcal{A}|\mathcal{S}'$ l'ensemble des arêtes de $\mathcal{A}$ qui ont leurs deux extrémités dans $\mathcal{S}'$.
Le graphe $\mathcal{G}' = \mathcal{G}|\mathcal{S}' = (\mathcal{S}', \mathcal{A}|\mathcal{S}')$ est appelé le graphe induit de $\mathcal{G}$ par $\mathcal{S}'$.



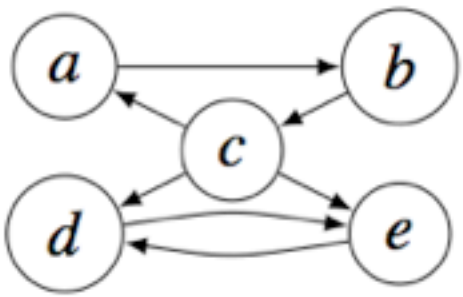
Tout graphe $(\mathcal{S}', \mathcal{A}')$ avec $\mathcal{S}' \subset \mathcal{S}$ et $\mathcal{A}' \subset \mathcal{A}$ est appelé sous-graphe de $\mathcal{G}$; si $\mathcal{S}' = \mathcal{S}$, on parle de graphe couvrant.



Chemins

Un chemin $(s_0, s_1, \ldots, s_p)$ dans $\mathcal{G}$ est une suite finie non vide de sommets telle que $(s_k, s_{k+1}) \in \mathcal{A}$. La longueur du chemin est p (== nombre d'arcs empruntés).

Ex. (c, a, b, c, e) est un chemin dans :



Un chemin est

- simple si les arcs (s_{i-1}, s_i) pour $i = 1, \ldots, p$ sont deux à deux $\neq$.
- élémentaire si les sommets sont deux à deux distincts.
- un circuit si $p \geqslant 1$ et $s_p = s_0$; il est élémentaire si $(s_0, \ldots, s_{p-1})$ est élémentaire.

Rem. Graphe non orienté : chemin $\rightarrow$ chaîne (arêtes distinctes 2 à 2) ; circuit $\rightarrow$ cycle.

Connexité

Graphe non-orienté connexe :

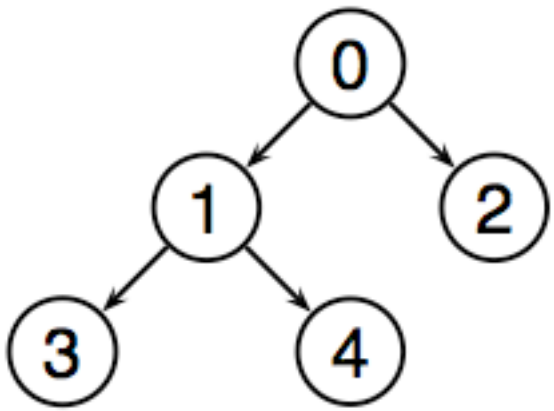
pour tous sommets a et b, chemin de a à b

Graphe orienté fortement connexe :

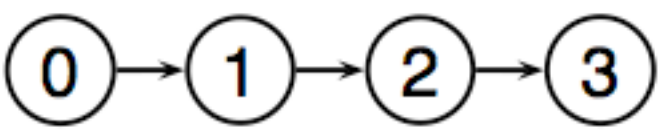
pour tous sommets a et b, chemin de a à b et de b à a.

Graphe/arbre/liste

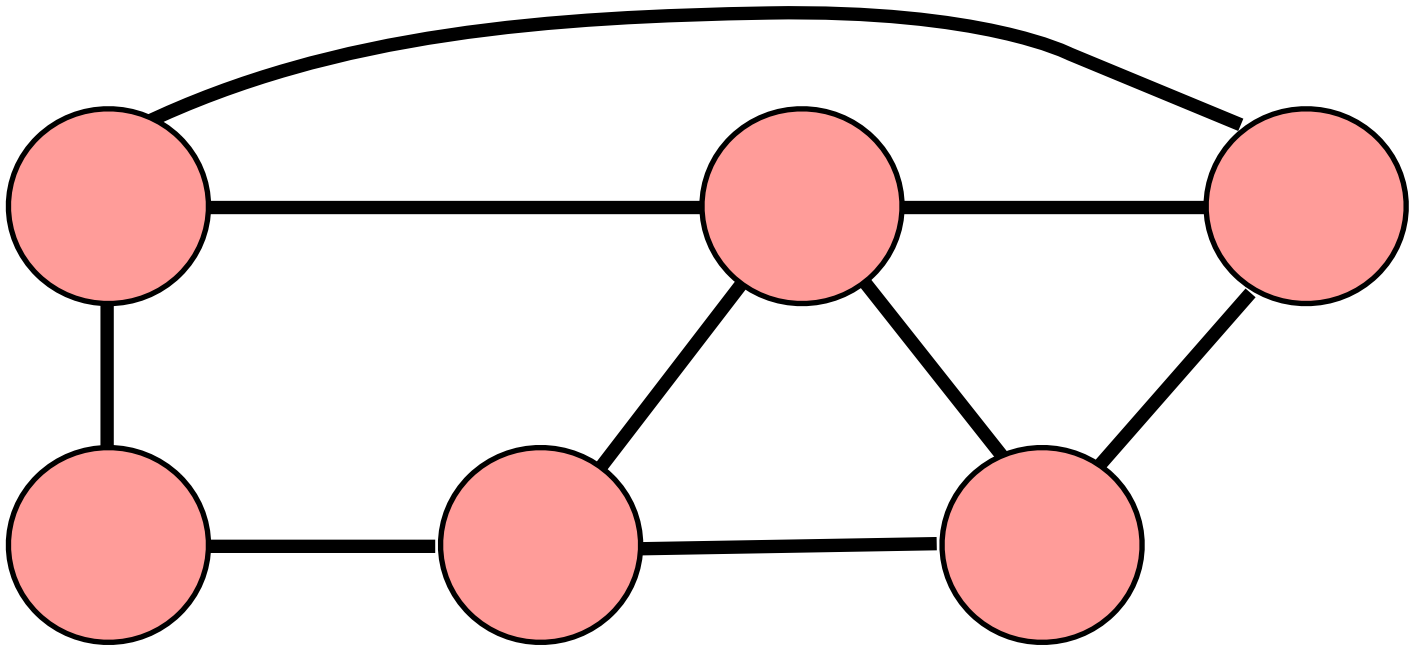
Un arbre est un graphe particulier :



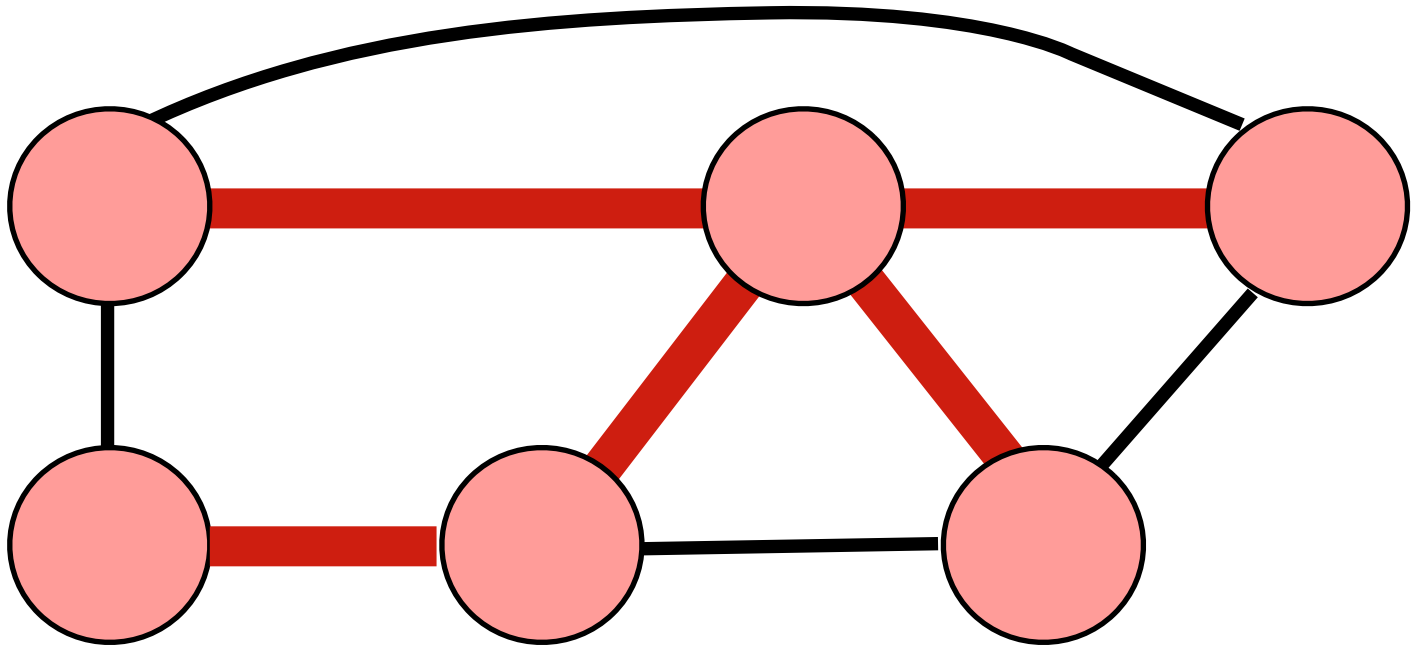
Une liste peut être vue comme un arbre filiforme, et donc un graphe :



Arbre couvrant



Arbre couvrant



Exemple: Un réseau de routes atteignant toutes les villes...

Caractérisation des arbres

Prop. soit $\mathcal{G}$ un graphe simple à n sommets. Les propriétés suivantes sont équivalentes et caractérisent le fait que $\mathcal{G}$ est un arbre.

(i) $\mathcal{G}$ est un graphe connexe sans cycle ;

(ii) $\mathcal{G}$ est un graphe connexe ayant $n-1$ arêtes ;

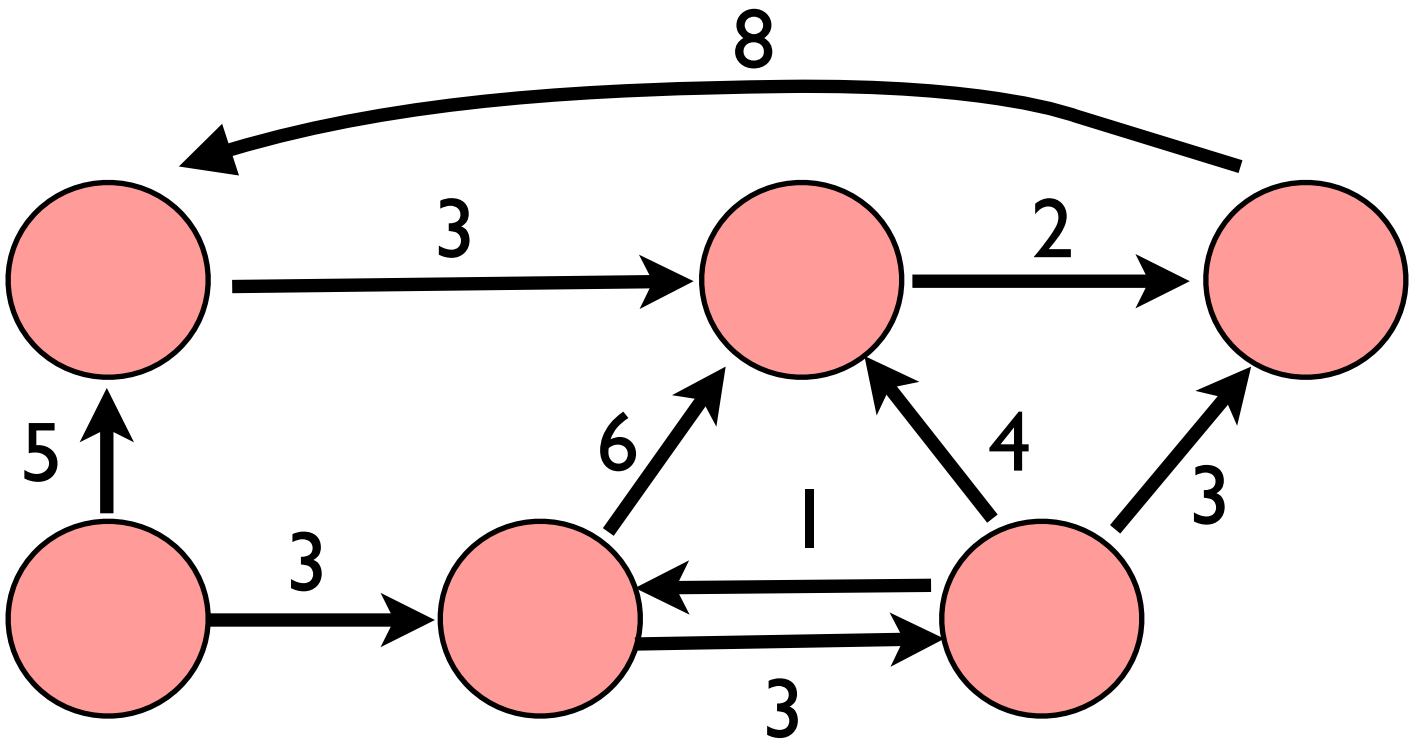
(iii) $\mathcal{G}$ est un graphe sans cycle possédant $n-1$ arêtes ;

(iv) $\mathcal{G}$ est un graphe dans lequel deux sommets quelconques sont liés par une seule chaîne ;

(v) $\mathcal{G}$ est un graphe connexe dont la connexité disparaît dès qu'on enlève une arête quelconque ;

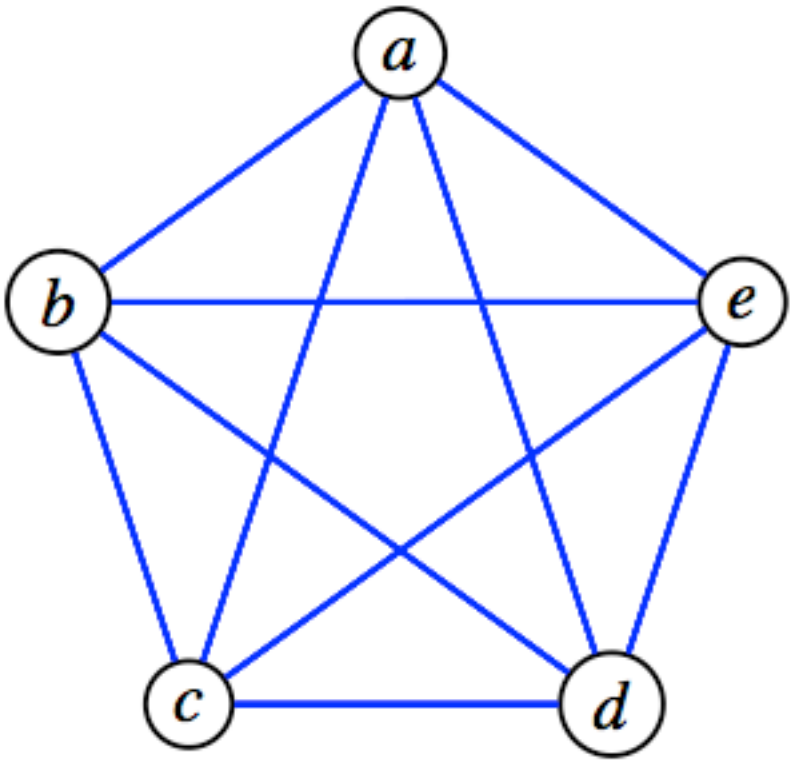
(vi) $\mathcal{G}$ est un graphe sans cycle tel que l'ajout d'une arête quelconque crée un cycle et un seul.

Graphes valués



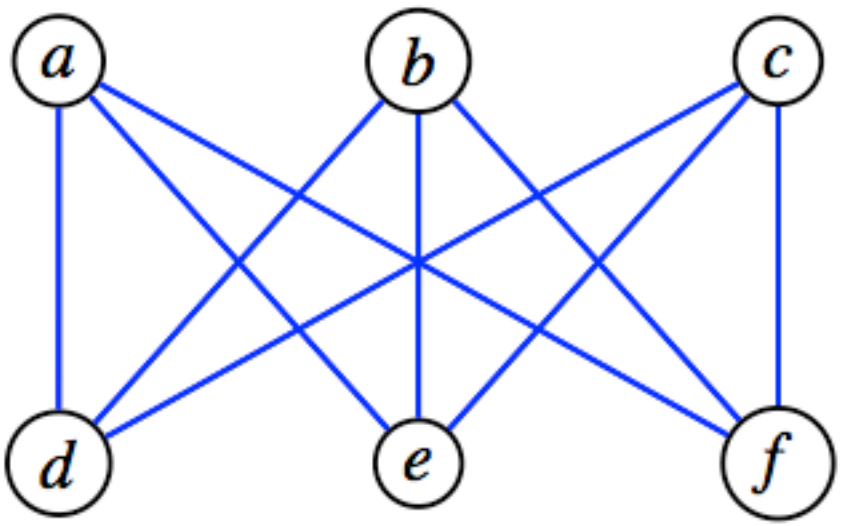
Un poids sur chaque arc : distance, prix, temps de parcours...

Quelques graphes classiques



K_5

graphe complet

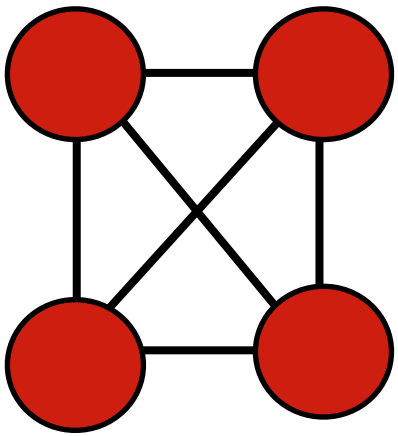


$K_{3,3}$

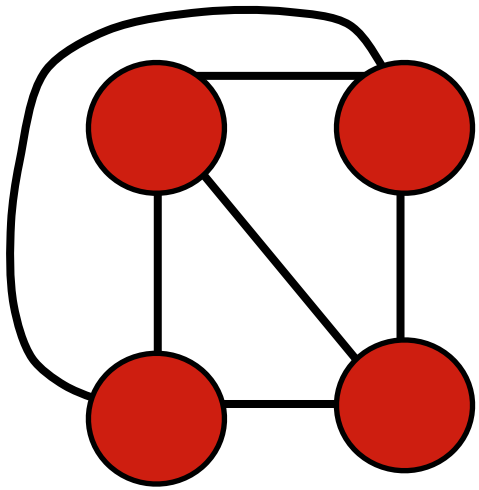
graphe complet biparti

Graphes planaires

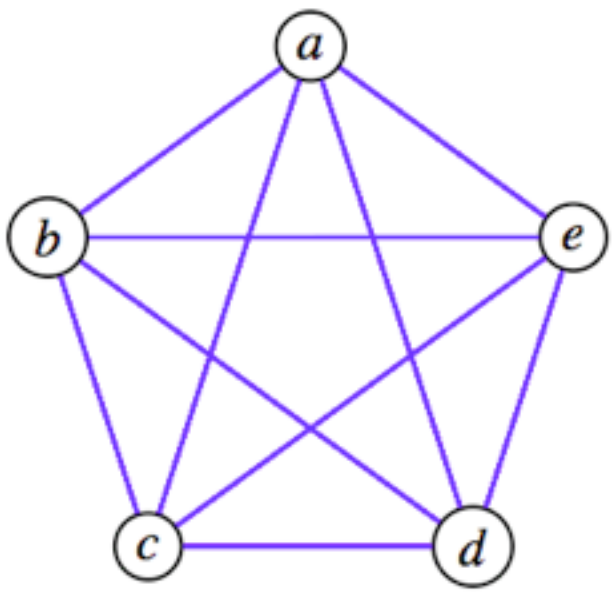
Un graphe est planaire s'il peut être (re)dessiné sans que deux arêtes ne se coupent.



est planaire:



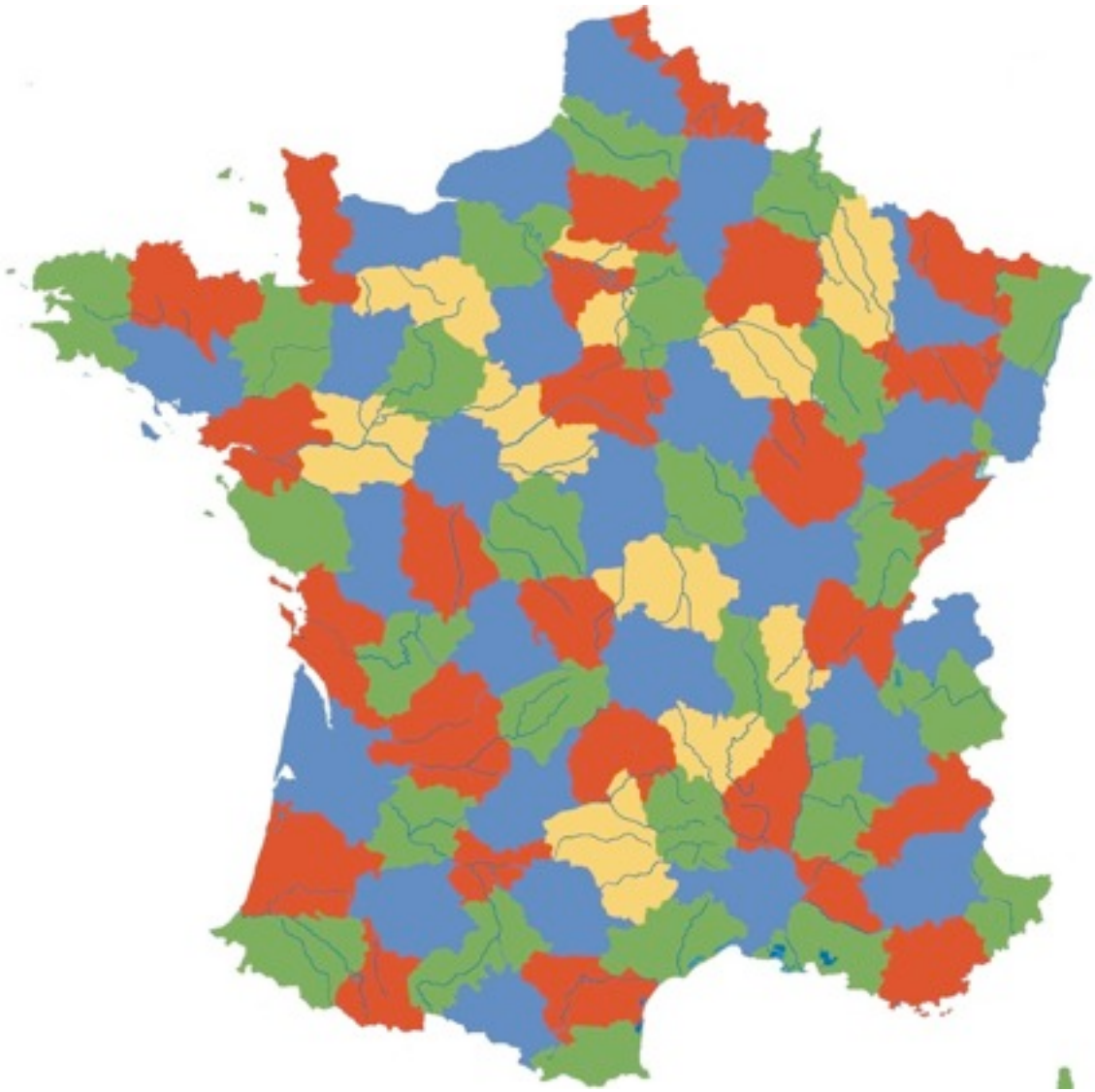
mais



n'est pas planaire

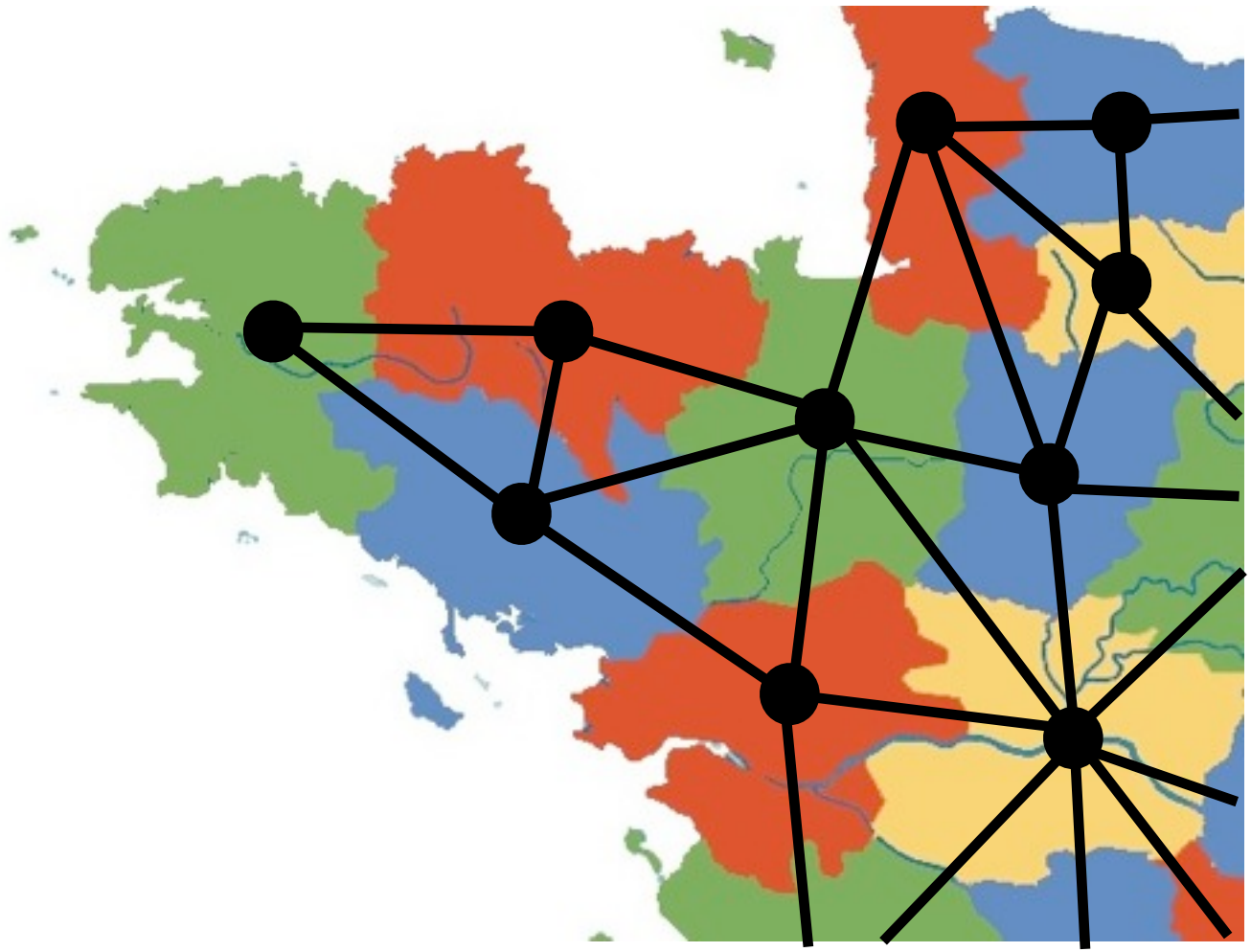
Coloriage: cartes et graphes

4 couleurs suffisent



En fait un problème sur les graphes planaires

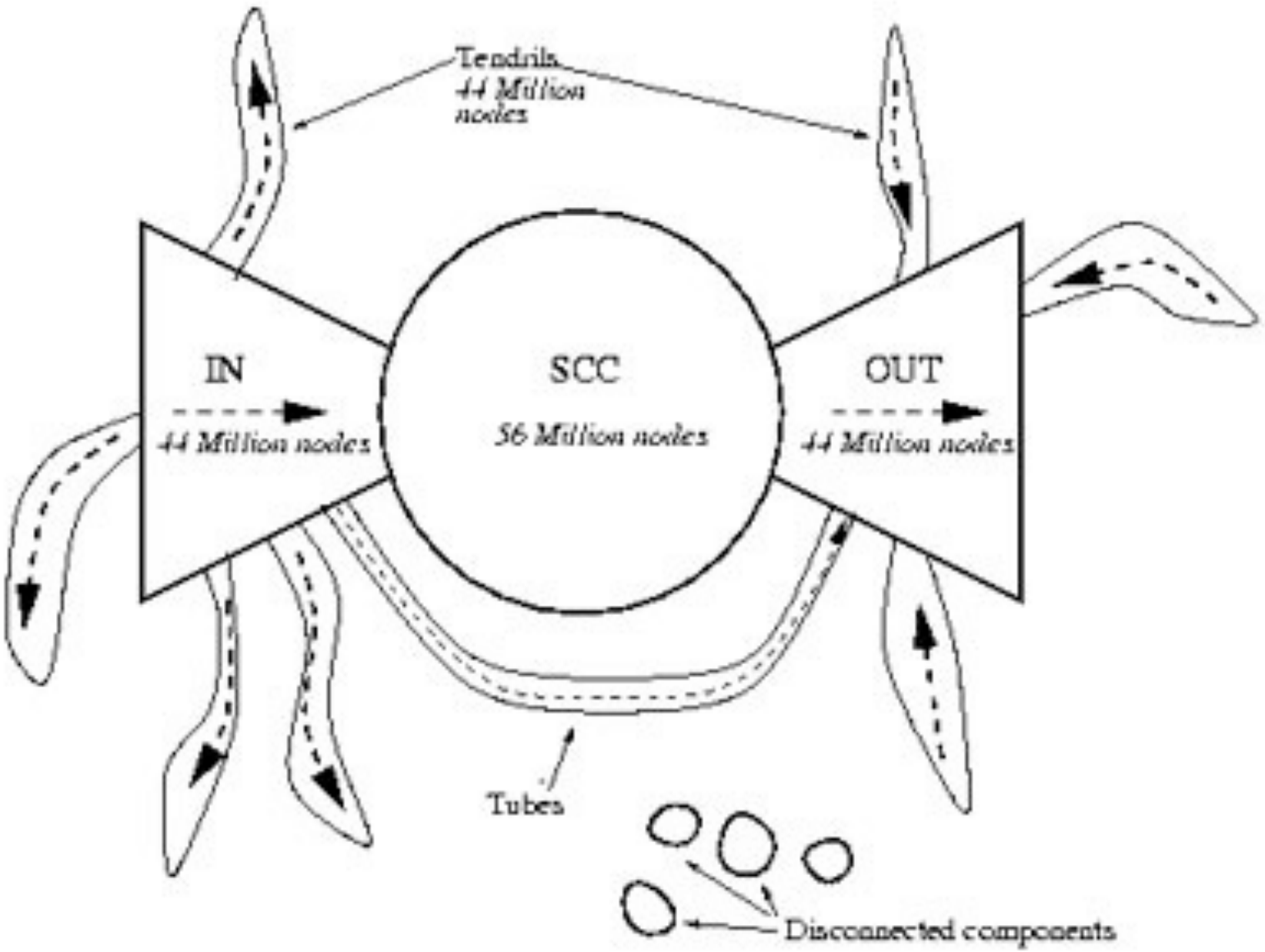
Coloriage de graphe



fonction de coloriage $c:\mathcal{S}\rightarrow\mathcal{C}$ avec $(a,b)\in\mathcal{A}\Rightarrow c(a)\neq c(b)$

on peut prendre $|\mathcal{C}| = 4$

Le graphe du Web



Problèmes fondamentaux

- Y a-t-il un chemin ? trouver un chemin, trouver le plus court chemin
- Connexité, composantes connexes, fortement connexes
- Tri topologique
- Coloriage (théorème des quatre couleurs, attribution de fréquences radio...)
- Arbres couvrants
- Dessins : comment dessiner un graphe plan (circuit électronique), un graphe de 10_6 sommets ?
- Optimisation combinatoire (voyageur de commerce...)

Représentation

- Matrices d'adjacence
- Listes d'adjacences
- Collection des arêtes
- ...

Quelles structures ?

Abstraitement : $\mathcal{G} = (\mathcal{S}, \mathcal{A})$.

Codage de $\mathcal{S}$: ensemble de sommets, par exemple $[0..n-1]$; en Java, n'importe quel objet de `Collection`.

Codage de $\mathcal{A}$:

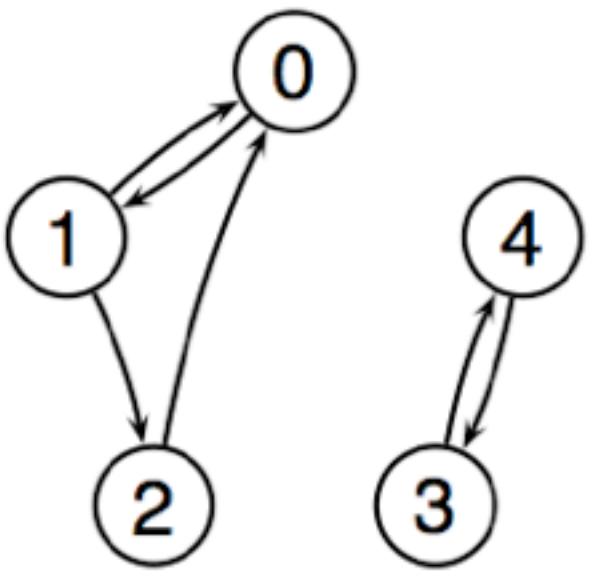
- graphe simple (pas de boucle ni arête multiple) : matrice, tableau de listes (ou autre structure garantissant l'unicité).
- multigraphe : tableau de listes (ou multi-ensembles).

Matrice d'adjacence

Pour $\mathcal{G}$ simple, matrice $n \times n$ telle que :

$$M_{i,j} = \begin{cases} 1 & \text{si } (i,j) \in \mathcal{A}, \\ 0 & \text{sinon.} \end{cases}$$

Ex.



$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Graphe valué : $M_{i,j} = v(i,j)$ (distance)

Matrices d'adjacences

Avantage:

- réponse immédiate (a,b) $\in \mathcal{A}$

Désavantage:

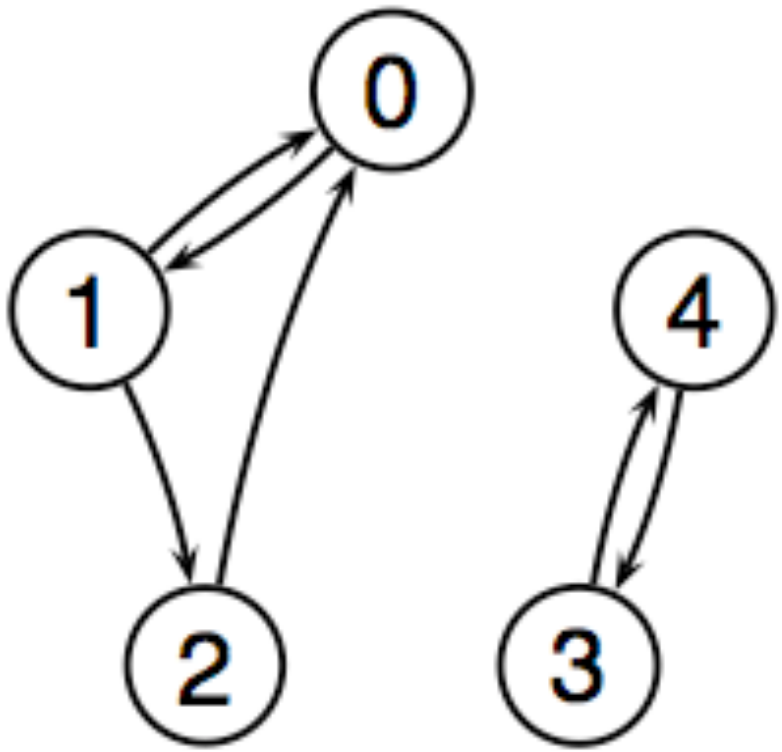
- gourmand en mémoire (surtout si le graphe n'est pas dense en arcs/arêtes)

Listes d'adjacence

On associe à chaque sommet i la liste des sommets j tels que $(i,j) \in \mathcal{A}$.

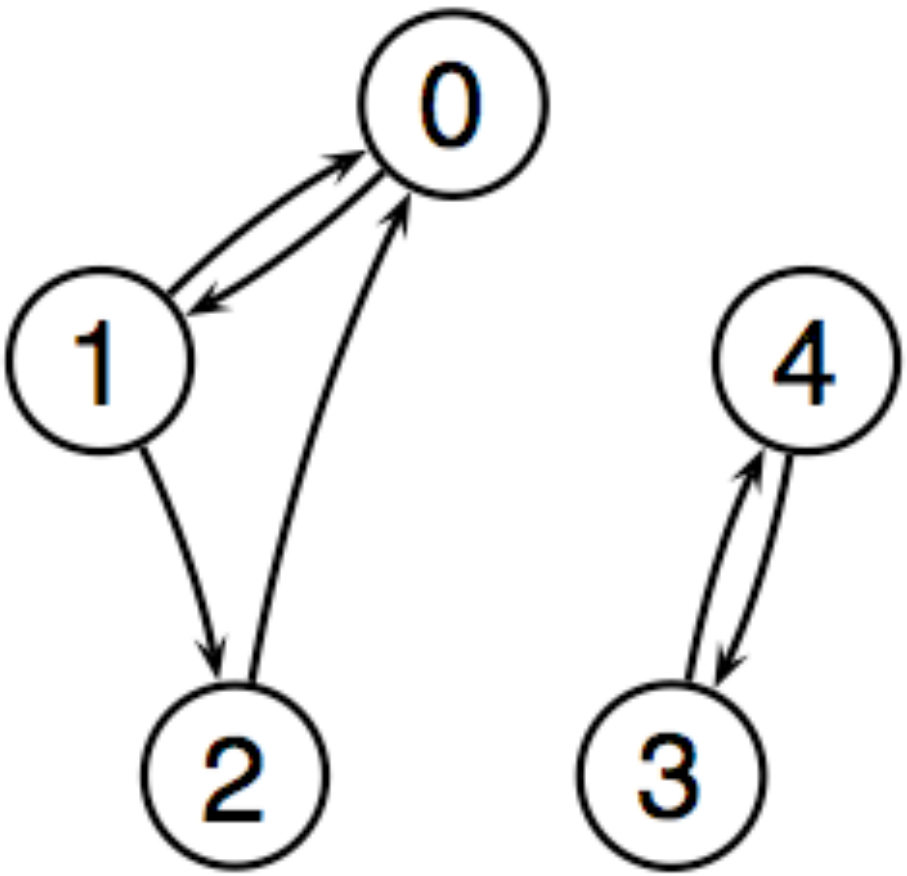
Ex.

```
L[0] = (1)
L[1] = (0, 2)
L[2] = (0)
L[3] = (4)
L[4] = (3)
```



collection d'arcs

```
a[0] = (0,1)
a[1] = (1,0)
a[2] = (1,2)
a[3] = (2,0)
a[4] = (3,4)
a[5] = (4,3)
```

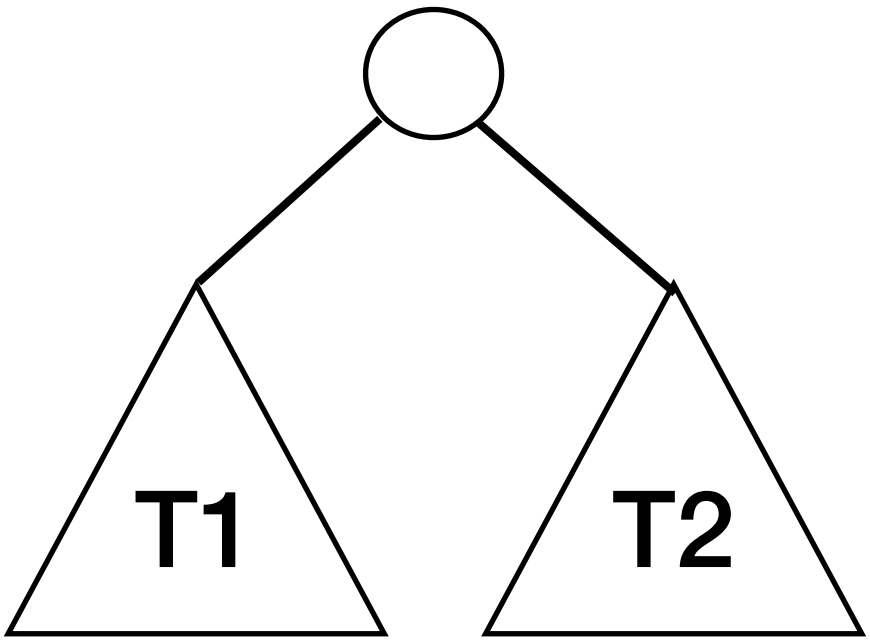


(cet après-midi en TD)

Premiers algorithmes

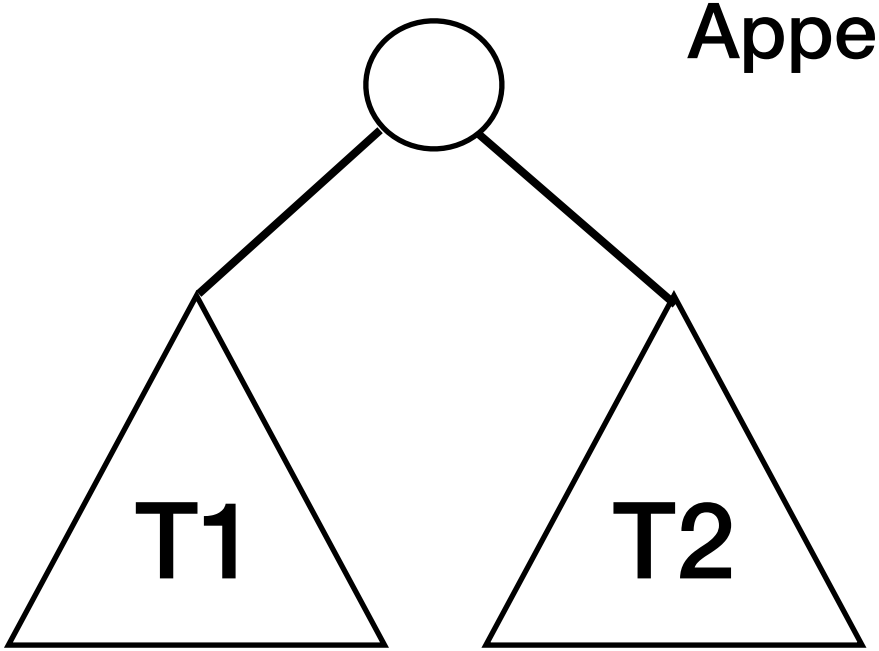
Une structure moins rigide

algo récursif sur un arbre:



Une structure moins rigide

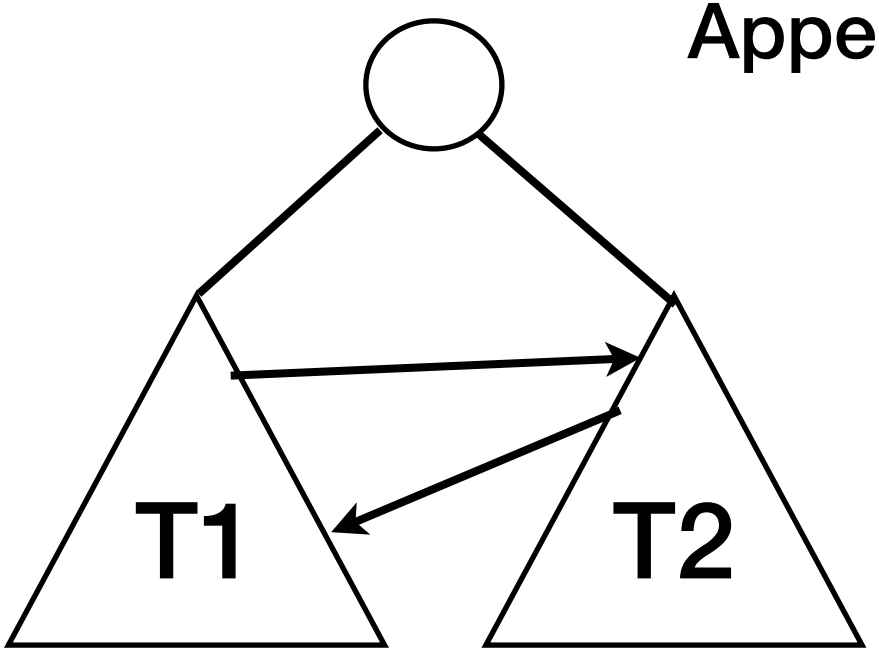
algo récursif sur un arbre:



Appels récursifs sur T1 et T2: ok

Une structure moins rigide

algo récursif sur un arbre:

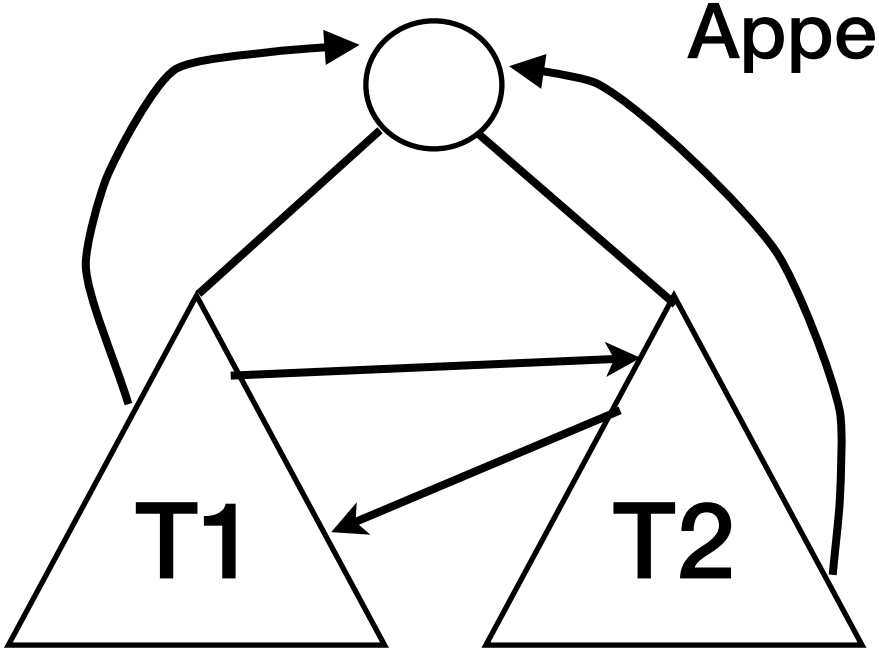


Appels récursifs sur T1 et T2: ok

Graphe: c'est plus compliqué

Une structure moins rigide

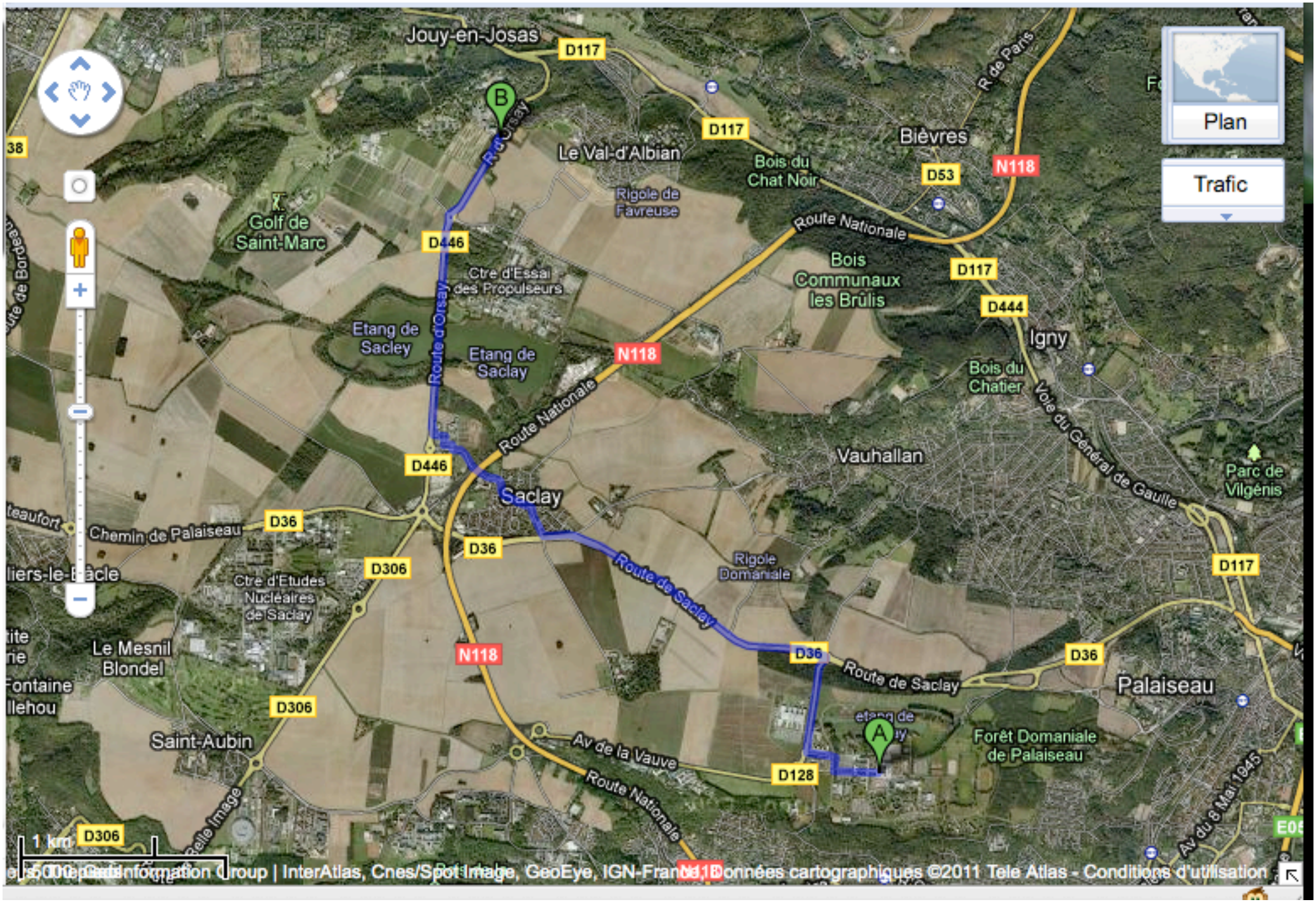
algo récursif sur un arbre:



Appels récursifs sur T1 et T2: ok

Graphe: c'est plus compliqué

Trouver la bonne décomposition suivant le problème



lundi 7 février 2011


Stéphane peut-il rejoindre Chloé ?
Jouy-en-Josas
D117
Le Val-d'Albian
Bièvres
Bois du Chat Noir
D53
N118
Rigole de Favreuse
Golf de Saint-Marc
D446
Route Nationale
Ctre d'Essai des Propulseurs
Bois Communaux les Brûlis
D444
Etang de
Vauhallan
Saclay
Parc de Vilgénis
Chemin de Palaiseau
D36
D306
Rigole Domaniale
Route de Saclay
Ctre d'Etudes Nucléaires de Saclay
Le Mesnil Blondel
Palaiseau
Saint-Aubin
Av de la Vauve
D128
Forêt Domaniale de Palaiseau
Av du 8 Mai 1945
1 km
Plan
Trafic
Données cartographiques ©2011 Tele Atlas - Conditions d'utilisation

Connecter

Question: Peut-on aller de a à b ?

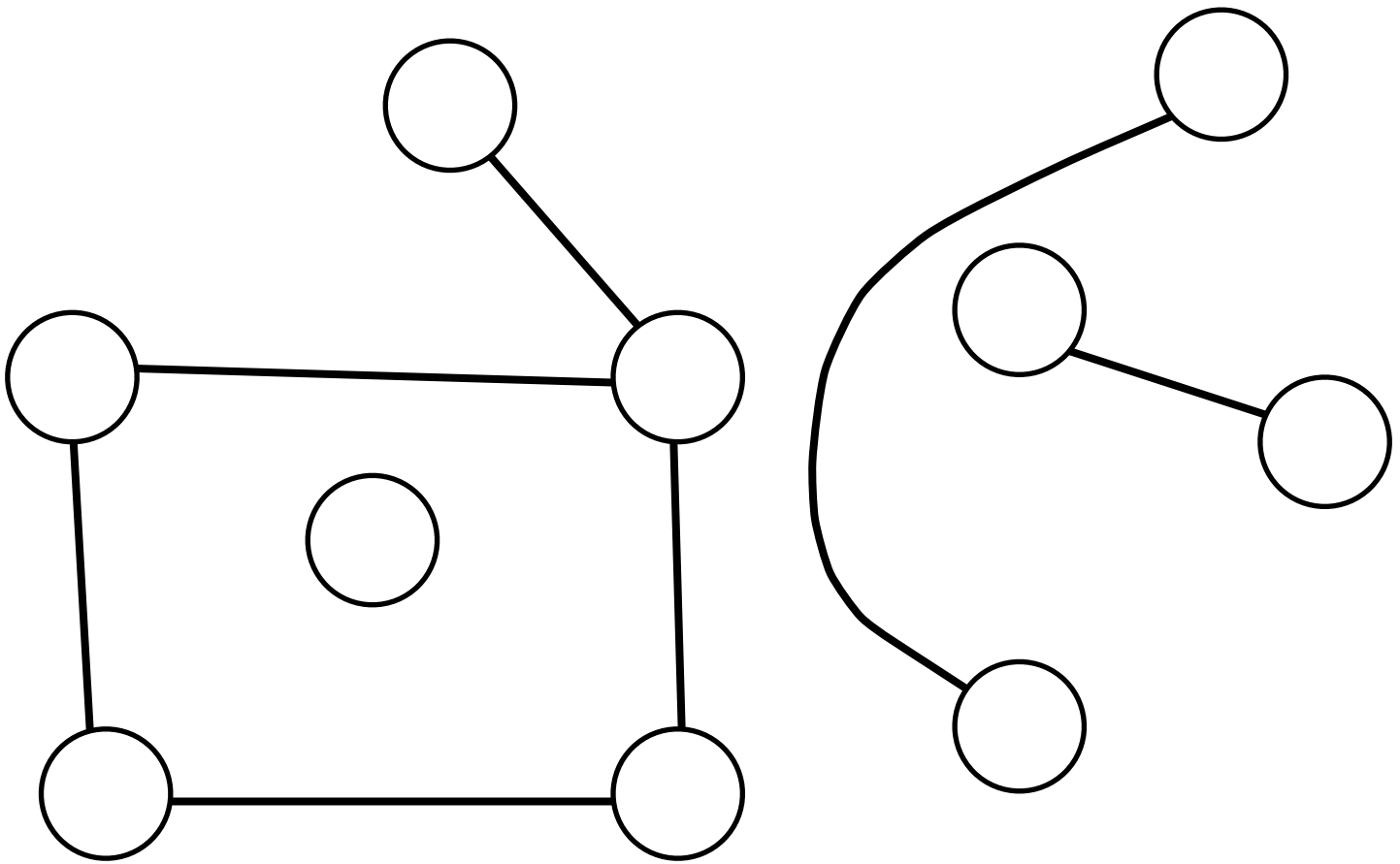
- Calculer tous les chemins (x,y) existants

  - Composantes connexes
  - clôture transitive du graphe

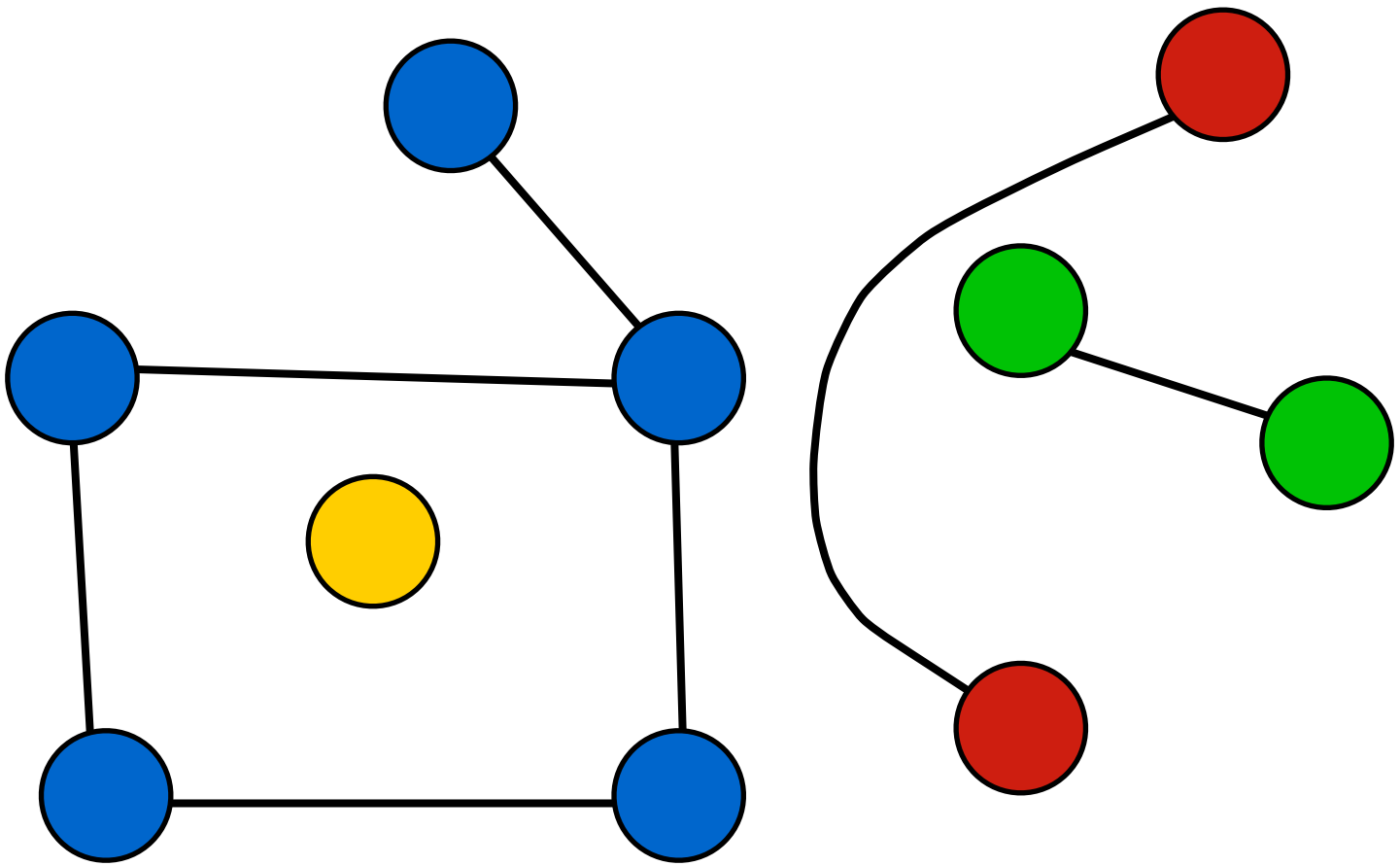
- Partir de a et explorer

  Parcours du graphe

Composantes Connexes



Composantes Connexes



Clôture transitive

$\mathcal{G}$=($\mathcal{S}$,$\mathcal{A}$) un graphe

Sa clôture transitive ($\mathcal{S}$,$\mathcal{A}$*) est définie par:

(a,b)$\in$ $\mathcal{A}$* $\Leftrightarrow$ $\exists$ chemin de a à b dans $\mathcal{G}$

Clôture transitive

$\mathcal{G}$=($\mathcal{S}$,$\mathcal{A}$) un graphe

Sa clôture transitive ($\mathcal{S}$,$\mathcal{A}$*) est définie par:

(a,b)$\in \mathcal{A}^* \Leftrightarrow$ $\exists$ chemin de a à b dans $\mathcal{G}$

Idée:

- Commencer par calculer $\mathcal{A}$*
- tester si (a,b)$\in \mathcal{A}^*$

Clôture transitive

$\mathcal{G}$=($\mathcal{S}$,$\mathcal{A}$) un graphe

Sa clôture transitive ($\mathcal{S}$,$\mathcal{A}$*) est définie par:

(a,b)$\in$ $\mathcal{A}$* $\Leftrightarrow$ $\exists$ chemin de a à b dans $\mathcal{G}$

Idée:

- Commencer par calculer $\mathcal{A}$*
- tester si (a,b)$\in$ $\mathcal{A}$*

Ca marche bien avec les matrices d'adjacence

Première idée : on construit une suite de graphes $(\mathcal{G}^{(r)})_{1 \leqslant r \leqslant n}$,

- $\mathcal{G}^{(1)}$ est le graphe initial ;
- pour $r \geqslant 2$, $\mathcal{G}^{(r)} = (\mathcal{S}, \mathcal{A}^{(r)}, \mathcal{M}^{(r)})$ est défini par $(i,j) \in \mathcal{A}^{(r)}$ ssi il existe un chemin de i à j de longueur exactement r.

Un chemin de longueur $r > 1$ entre i et j s'écrit $(i, k, \ldots, j)$ avec k voisin de i et $(k, \ldots, j)$ un chemin de longueur $r - 1$.
En booléens :

$$\mathcal{M}^{(r)}_{i,j} = \bigvee_{k=0}^{n-1} \mathcal{M}^{(1)}_{i,k} \wedge \mathcal{M}^{(r-1)}_{k,j}.$$

$\leftrightarrow$ produit des matrices $\mathcal{M}^{(1)}$ et $\mathcal{M}^{(r-1)}$ où on a remplacé l'addition par le OU logique, et la multiplication par le ET logique.

	0	1
0	0	1
1	1	1

OU logique $\vee$

	0	1
0	0	0
1	0	1

ET logique $\wedge$

Clôture transitive par matrices: complexité

$A \times A \times A \times A \times A \times A \ldots\ldots \times A$

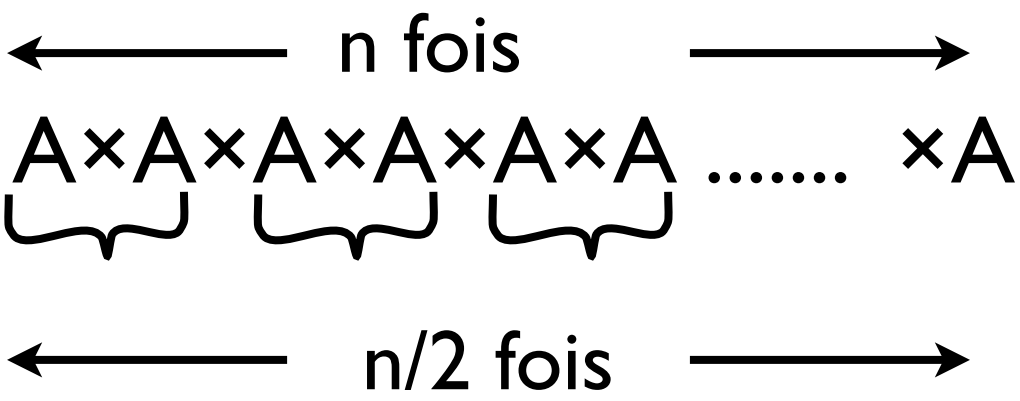
Clôture transitive par matrices: complexité

$A \times A \times A \times A \times A \times A \ldots\ldots \times A$

A en log(n) multiplications

complexité de la multiplication: $O(n^{\omega})$ $2 < \omega < 3$

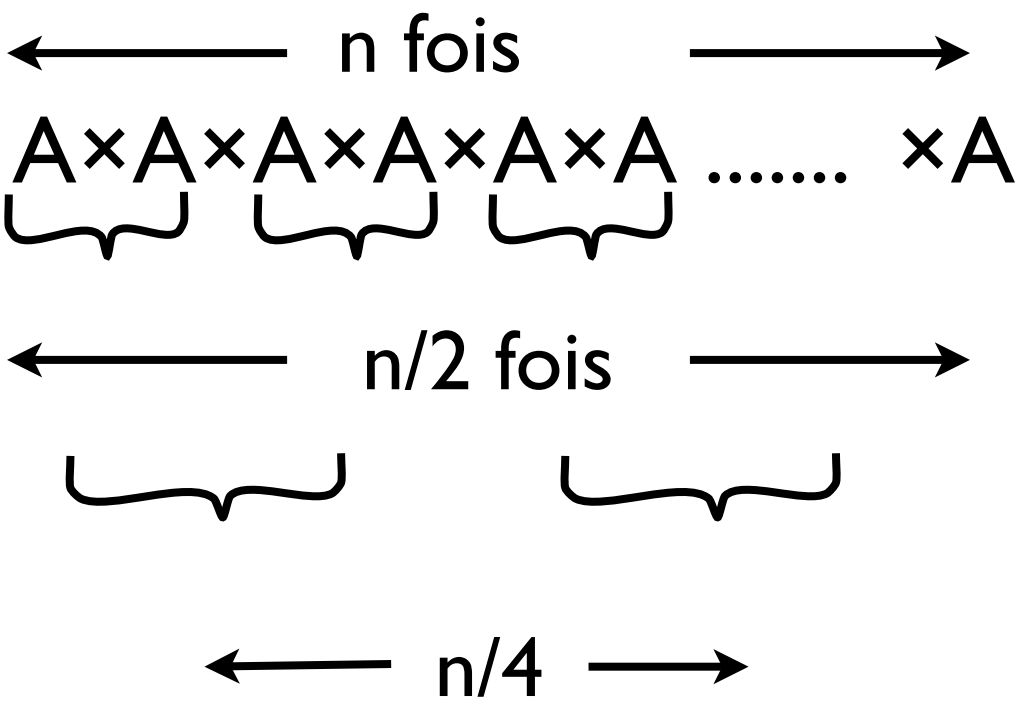
Clôture transitive par matrices: complexité



A en log(n) multiplications

complexité de la multiplication: $O(n^{\omega})$ $2< \omega<3$

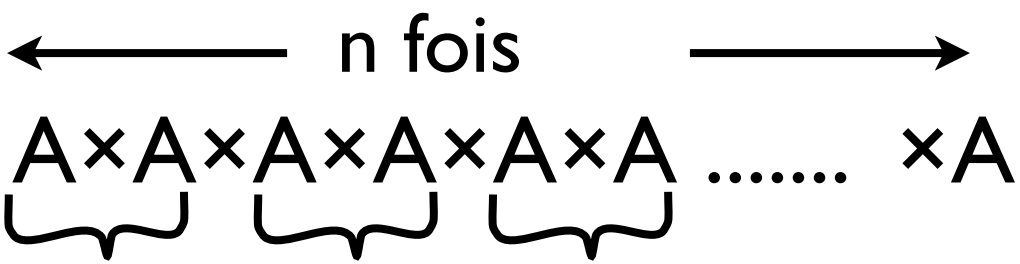
Clôture transitive par matrices: complexité

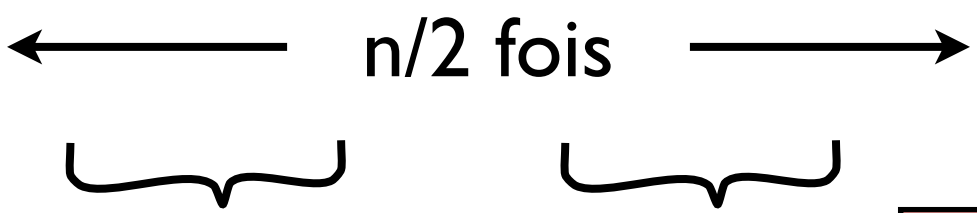


A en log(n) multiplications

complexité de la multiplication: $O(n^{\omega})$ $2< \omega<3$

Clôture transitive par matrices: complexité





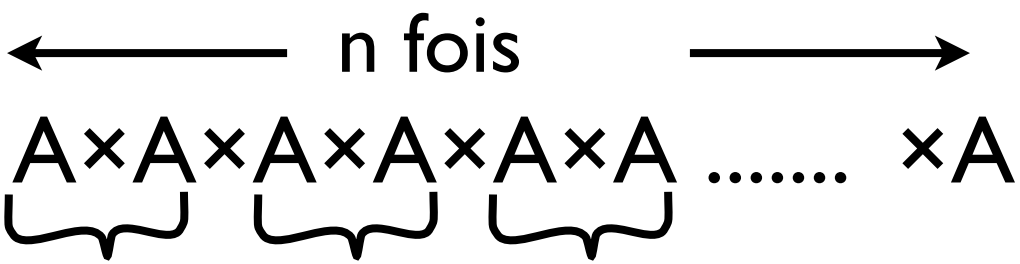
etc

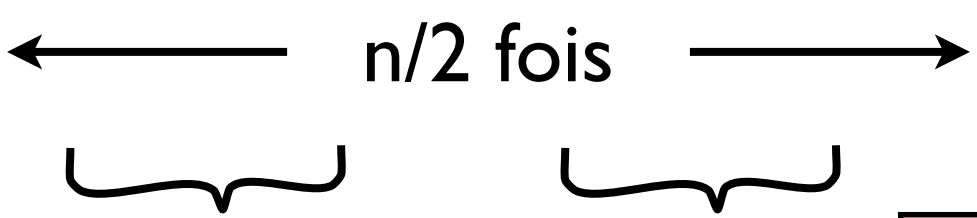
$A^{(2^i)}$ en i multiplications

A en log(n) multiplications

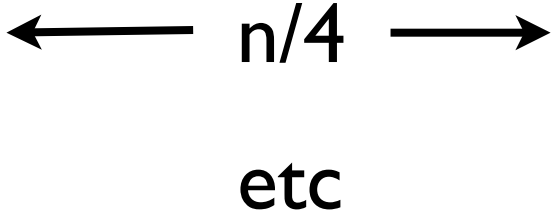
complexité de la multiplication: $O(n^{\omega})$ $2< \omega<3$

Clôture transitive par matrices: complexité





n=101001 en binaire

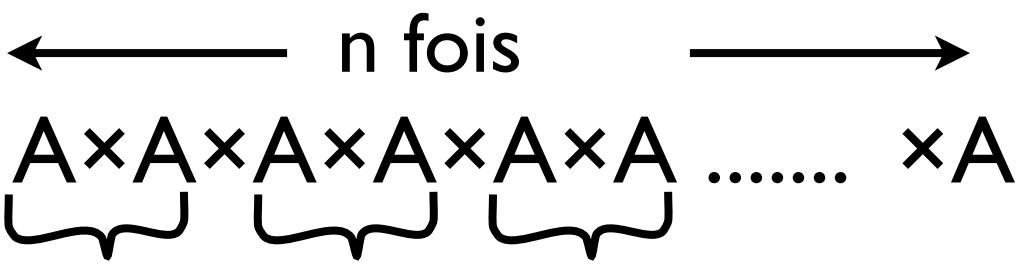


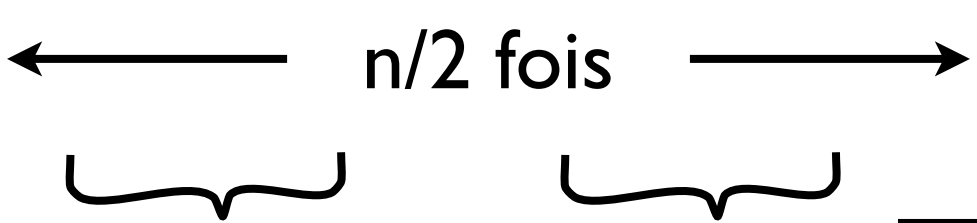
$A^{(2^i)}$ en i multiplications

A en log(n) multiplications

complexité de la multiplication: $O(n^{\omega})$ $2< \omega<3$

Clôture transitive par matrices: complexité





n=101001 en binaire

etc

$A^{(2^i)}$ en i multiplications

$A^{(n)}$ en log(n) multiplications

complexité de la multiplication: $O(n^{\omega})$ $2< \omega<3$

L'algorithme de Roy et Warshall

Idée : construire de proche en proche des chemins (élémentaires) de plus en plus longs.
On construit $\mathcal{G}^{(r)} = (\mathcal{S}, \mathcal{A}^{(r)})$ pour $r \geqslant -1$:

- $(i,j) \in \mathcal{A}^{(-1)}$ ssi $\exists$ chemin $i \to j$ ne passant par aucun sommet intermédiaire.
- $(i,j) \in \mathcal{A}^{(0)}$ ssi $\exists$ chemin $i \to j$ passant au plus par le sommet 0.

Ex.

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

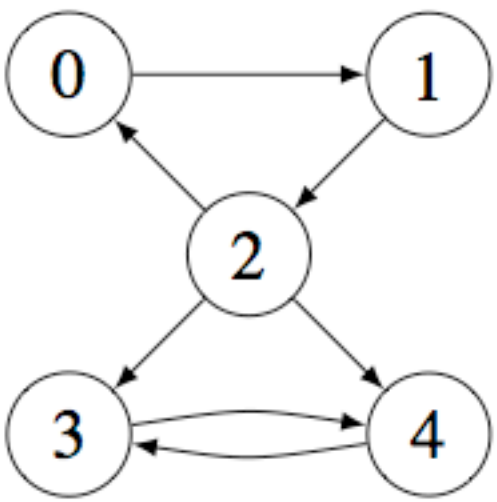
L'algorithme de Roy et Warshall

Idée : construire de proche en proche des chemins (élémentaires) de plus en plus longs.

On construit $\mathcal{G}^{(r)} = (\mathcal{S}, \mathcal{A}^{(r)})$ pour $r \geqslant -1$:

- $(i,j) \in \mathcal{A}^{(-1)}$ ssi $\exists$ chemin $i \to j$ ne passant par aucun sommet intermédiaire.
- $(i,j) \in \mathcal{A}^{(0)}$ ssi $\exists$ chemin $i \to j$ passant au plus par le sommet 0.

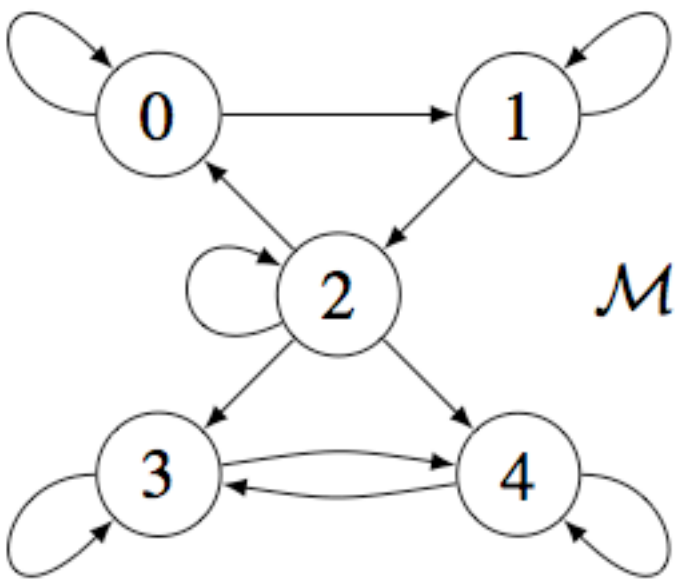
Ex.



$$\mathcal{M} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

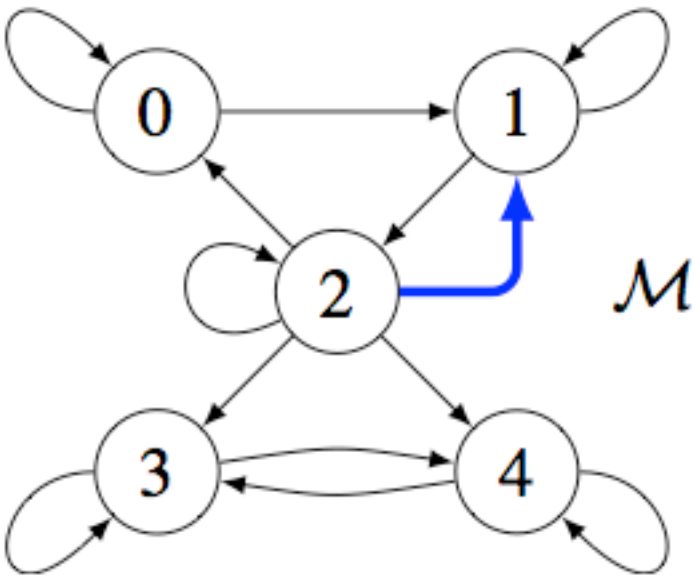
Calcul de $\mathcal{G}^{(-1)}$

On doit rajouter les boucles.



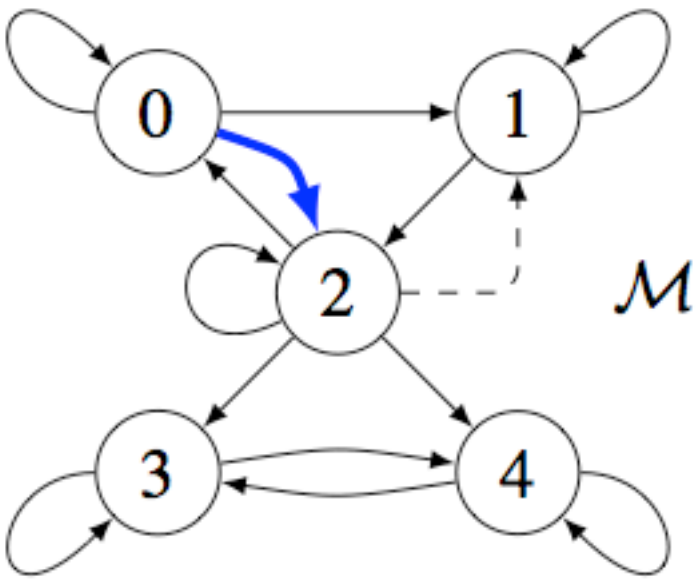
$$\mathcal{M}^{(-1)} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Calcul de $\mathcal{G}^{(0)}$: on rajoute les chemins $(i,0,j)$, ici $\{(2,0,1)\}$.



$$\mathcal{M}^{(0)} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

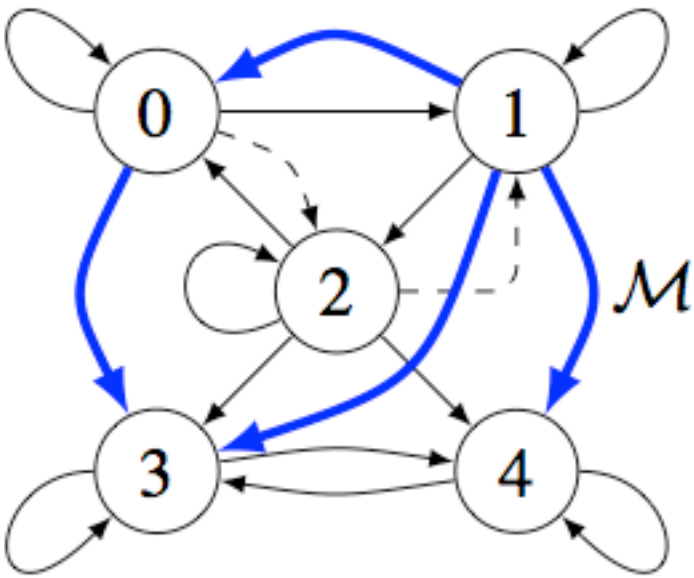
Calcul de $\mathcal{G}^{(1)}$: tous les chemins passant dans $\{0,1\}$, i.e., $\{(0,1,2),(2,0,1,2)\}$.



$$\mathcal{M}^{(1)} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

Calcul de $\mathcal{G}^{(2)}$: on rajoute les chemins

$$\{(0,1,2,0),(0,1,2,3),(0,1,2,4),(1,2,0),(1,2,0,1),(1,2,3),(1,2,4)\}$$



$$\mathcal{M}^{(2)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

Et c'est tout.

Algorithme

$(i,j) \in \mathcal{A}^{(r)}$ ssi $\exists$ chemin $i \to j$ ne passant par aucun sommet intermédiaire d'indice $> r$.

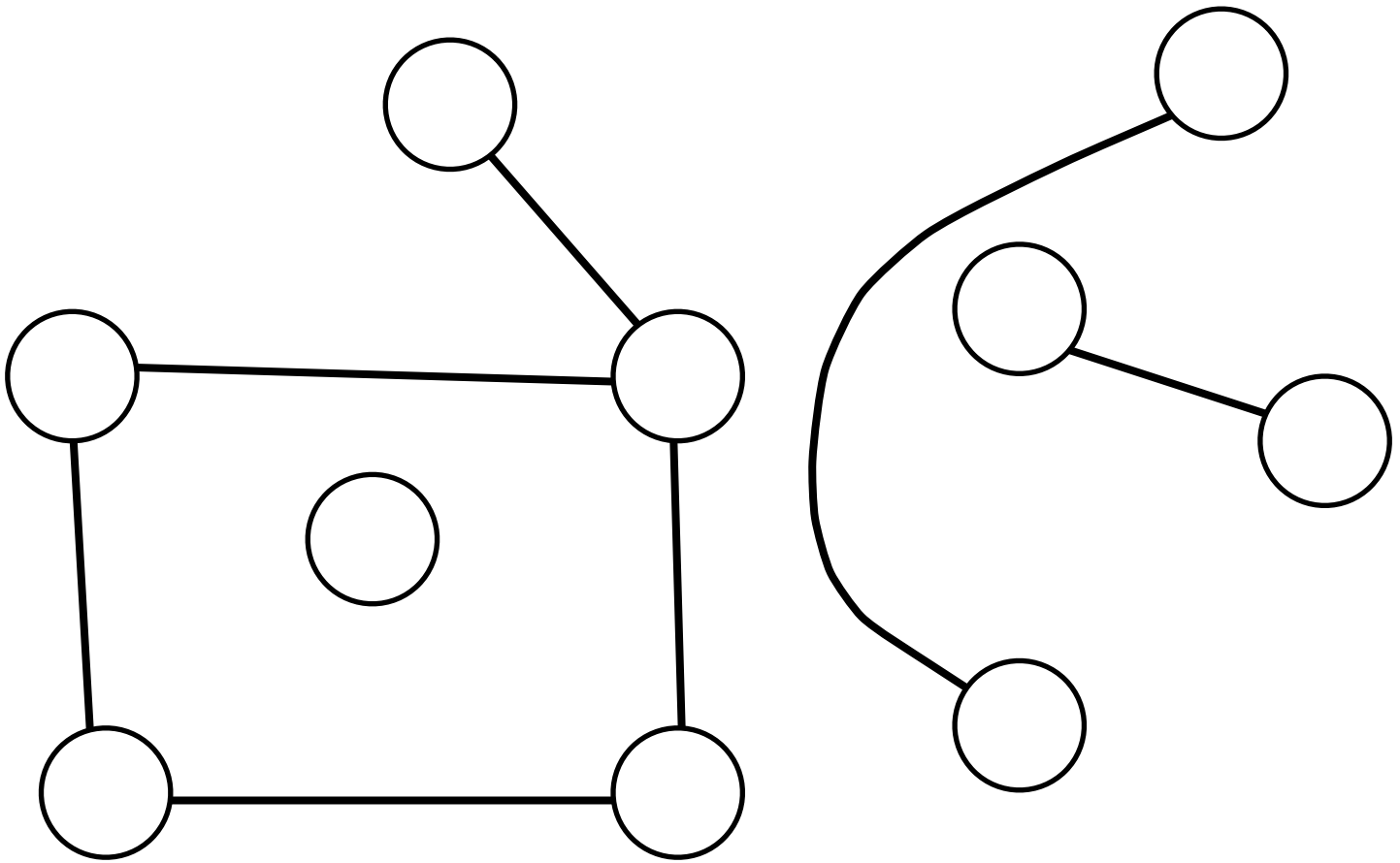
$\iff (i,j) \in \mathcal{A}^{(r-1)}$ ou ($(i,r) \in \mathcal{A}^{(r-1)}$ et $(r,j) \in \mathcal{A}^{(r-1)}$). En booléens :

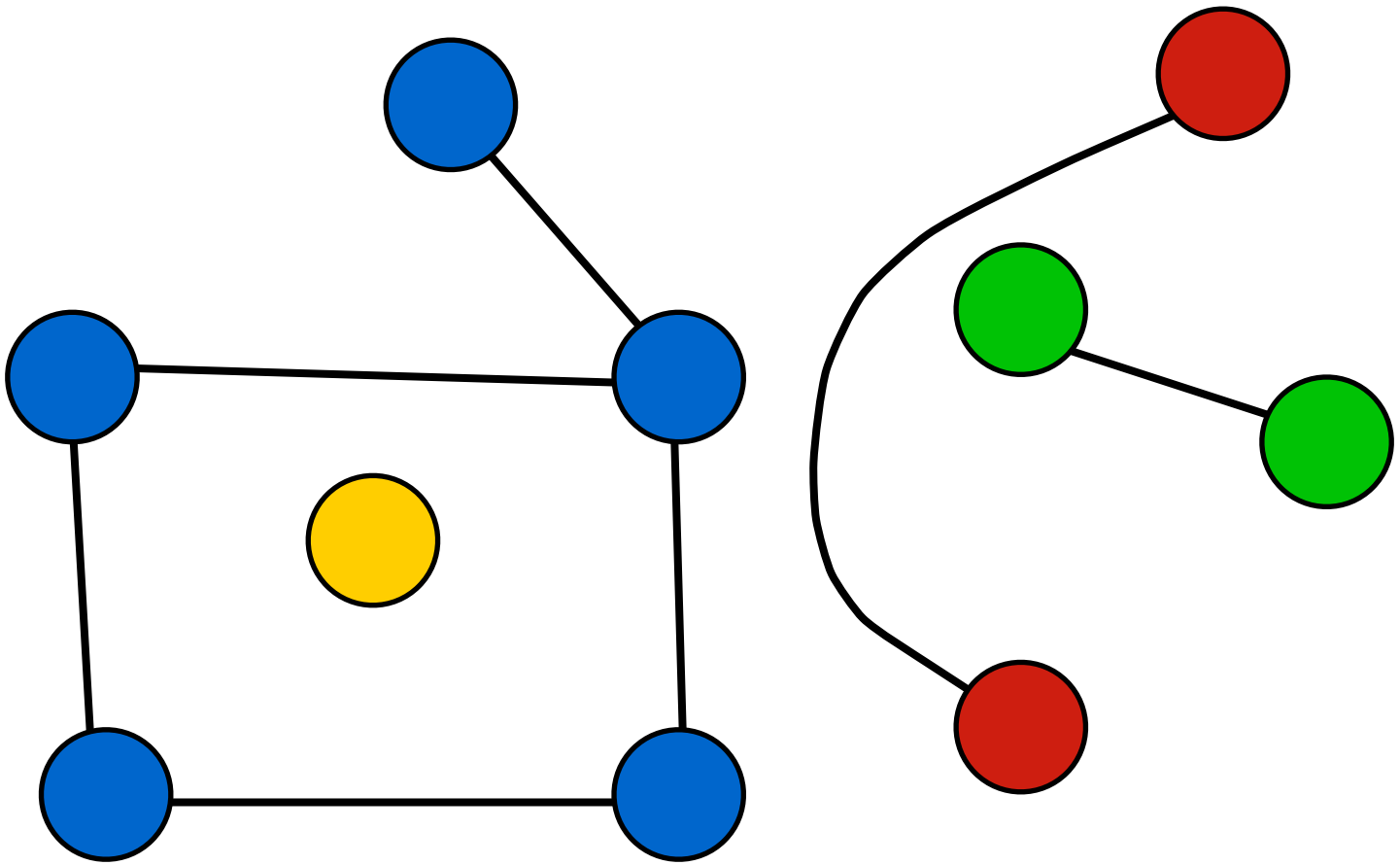
$$\mathcal{M}_{i,j}^{(r)} = \mathcal{M}_{i,j}^{(r-1)} + \mathcal{M}_{i,r}^{(r-1)} \mathcal{M}_{r,j}^{(r-1)}.$$

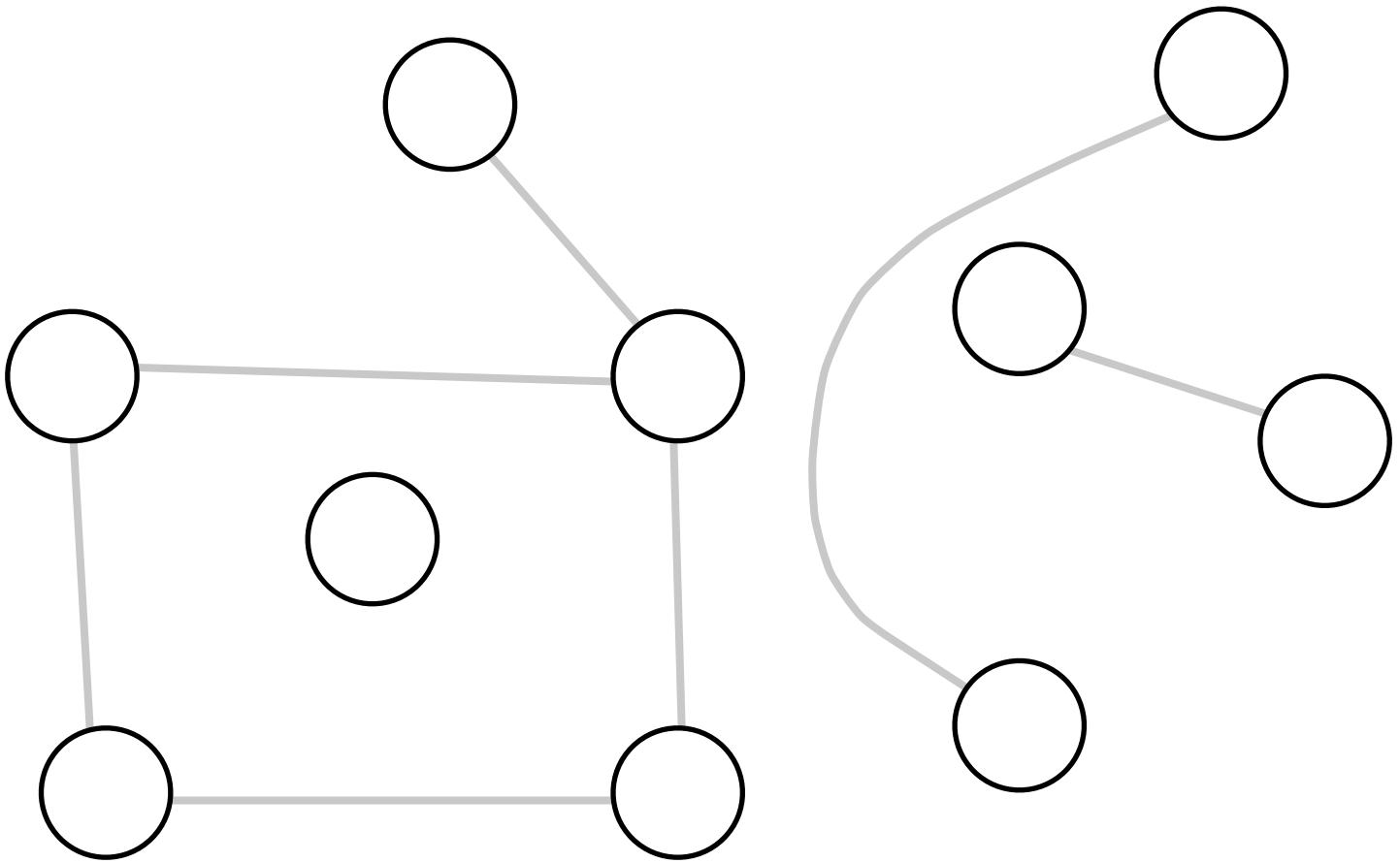
```
RoyWarshall(M)
// M est la matrice d'adjacence booléenne n x n
// d'un graphe G
1. N <- copie(M); P <- matrice_identite(n, n);
2. pour r <- 0 à n-1 faire
     pour i <- 0 à n-1 faire
       pour j <-0 à n-1 faire
         P[i][j] <- N[i][j] OU (N[i][r] ET N[r][j]);
     N := P; // recopie de P dans N
3. retourner N; // matrice d'adjacence de G *
```

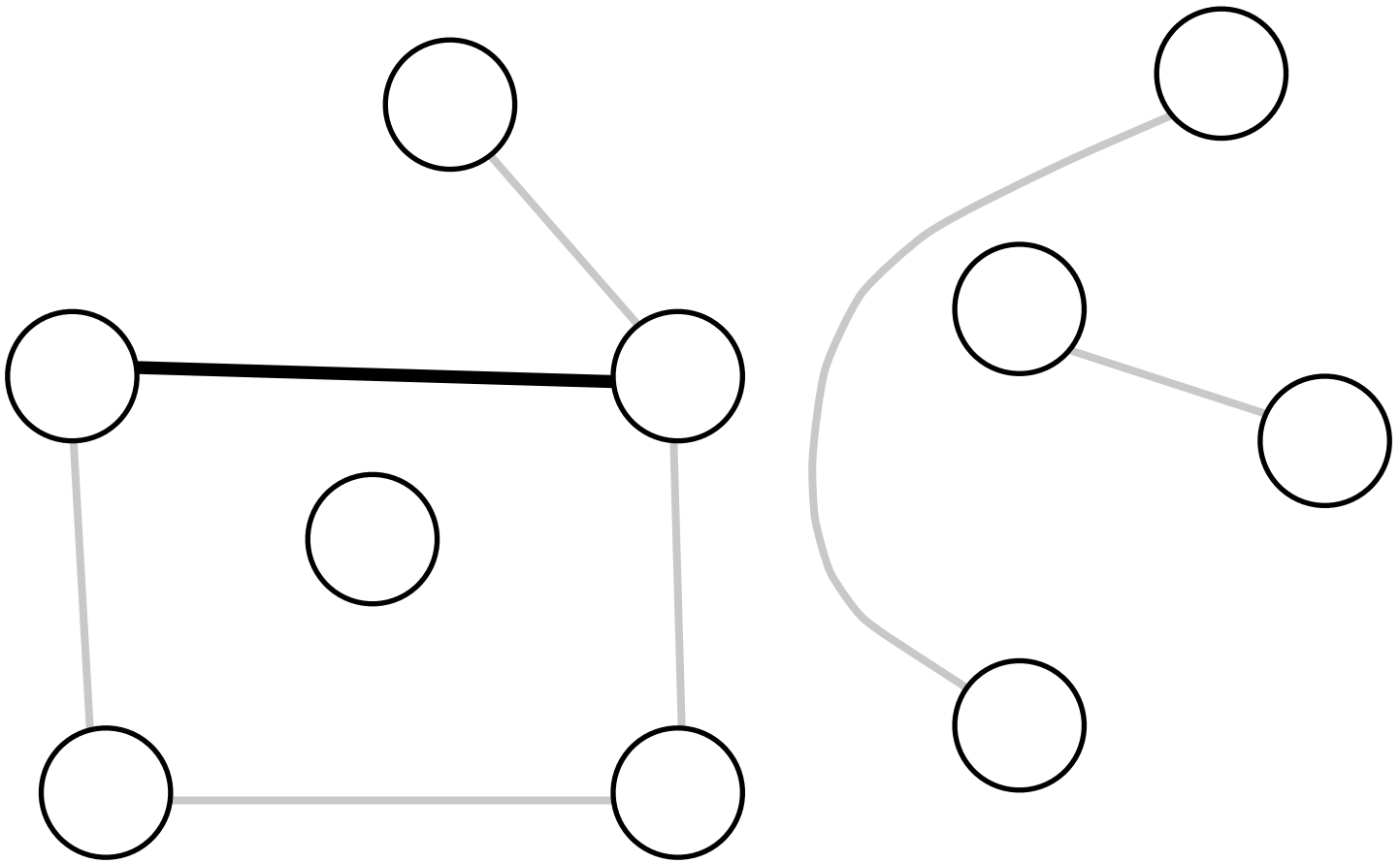
Prop. La complexité de cet algorithme est en $O(n^3)$.

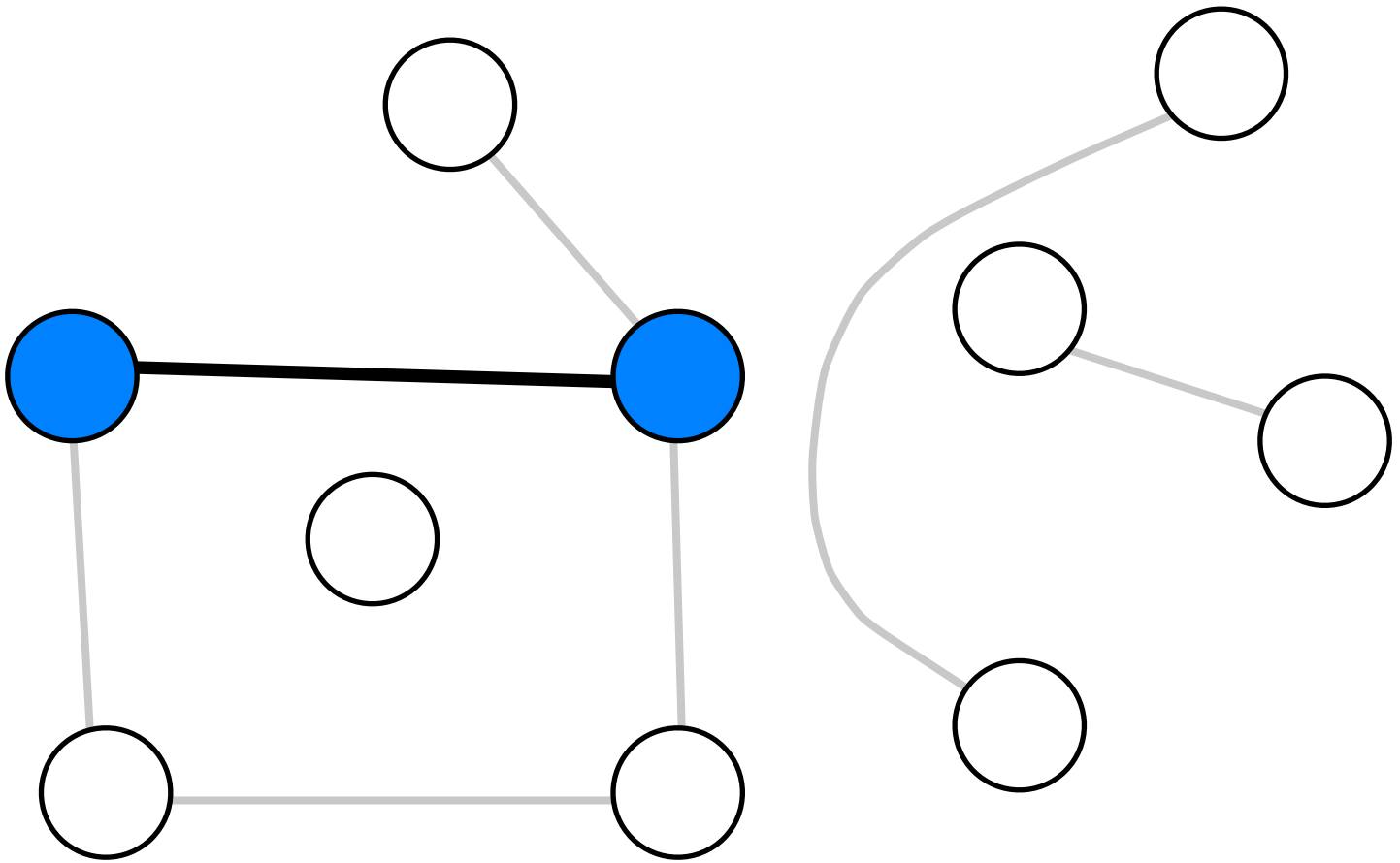
Construction des composantes connexes

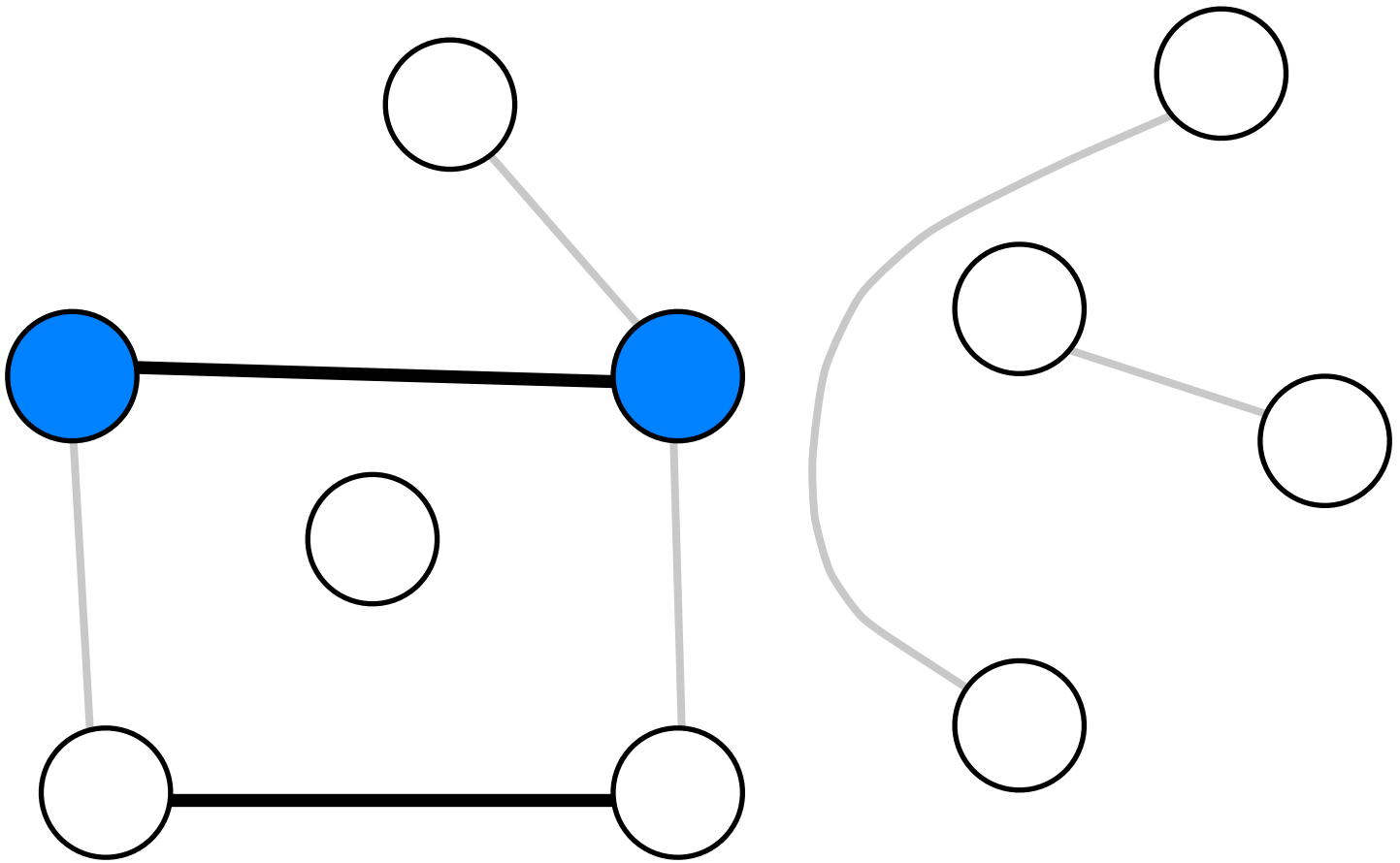


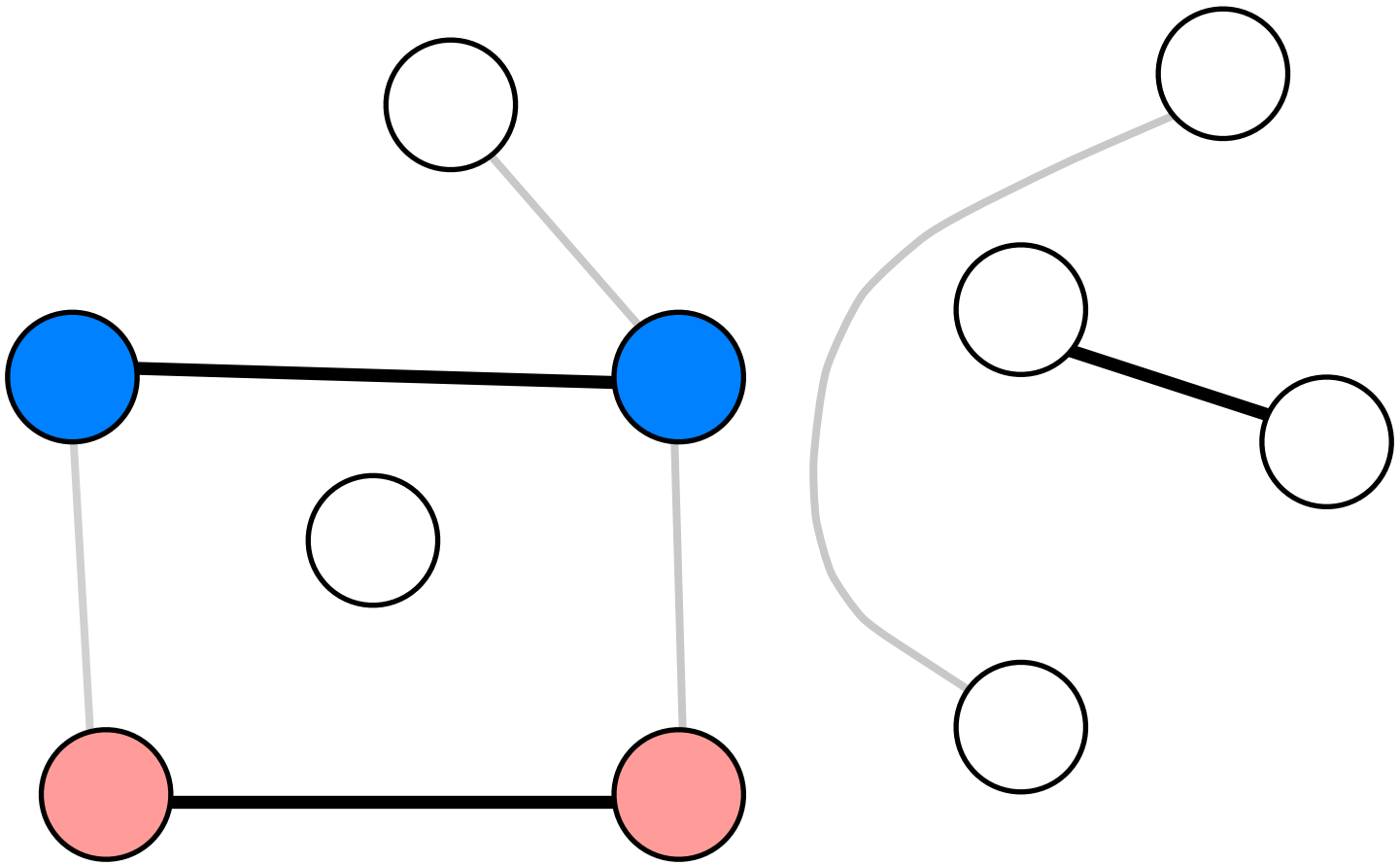


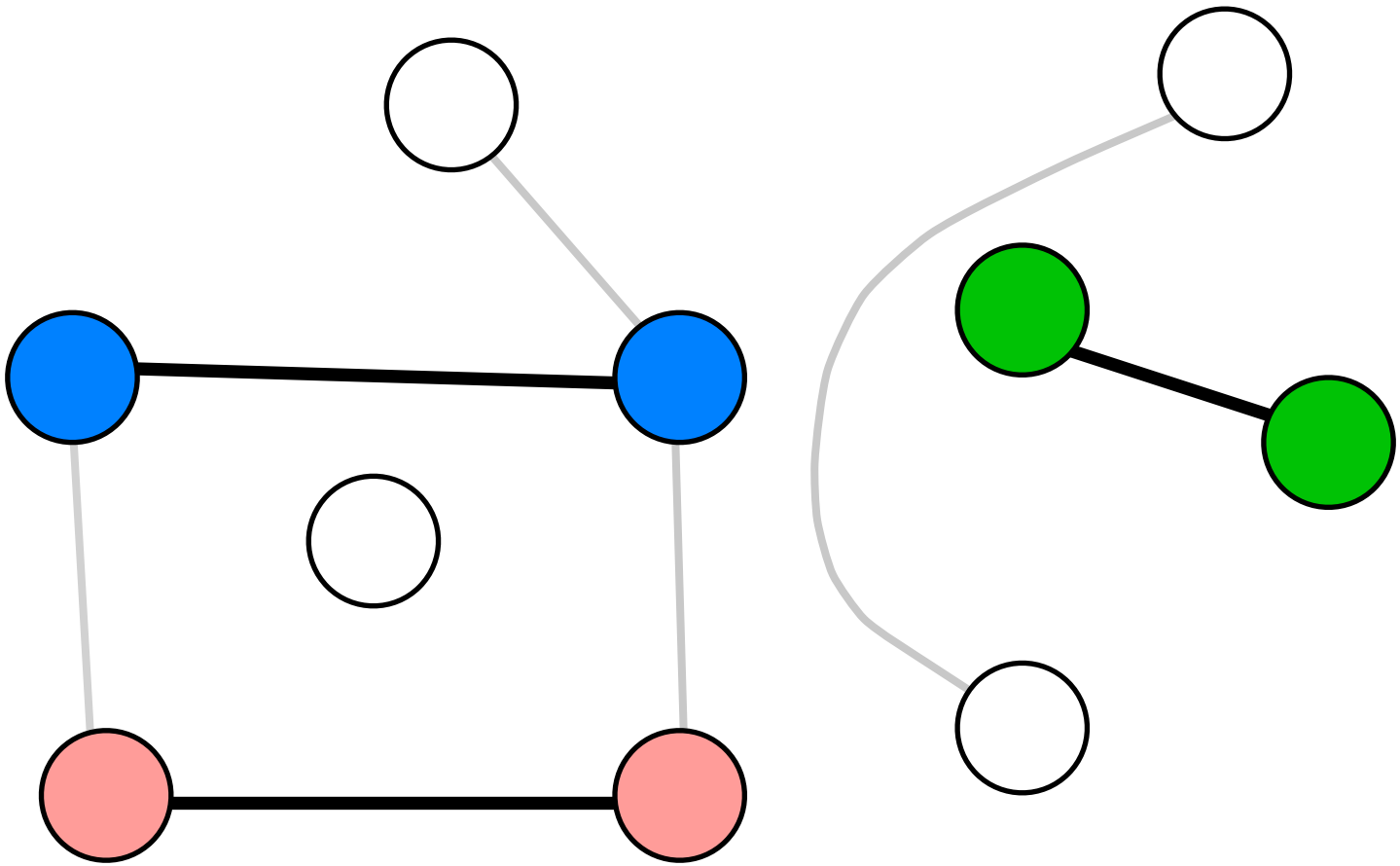




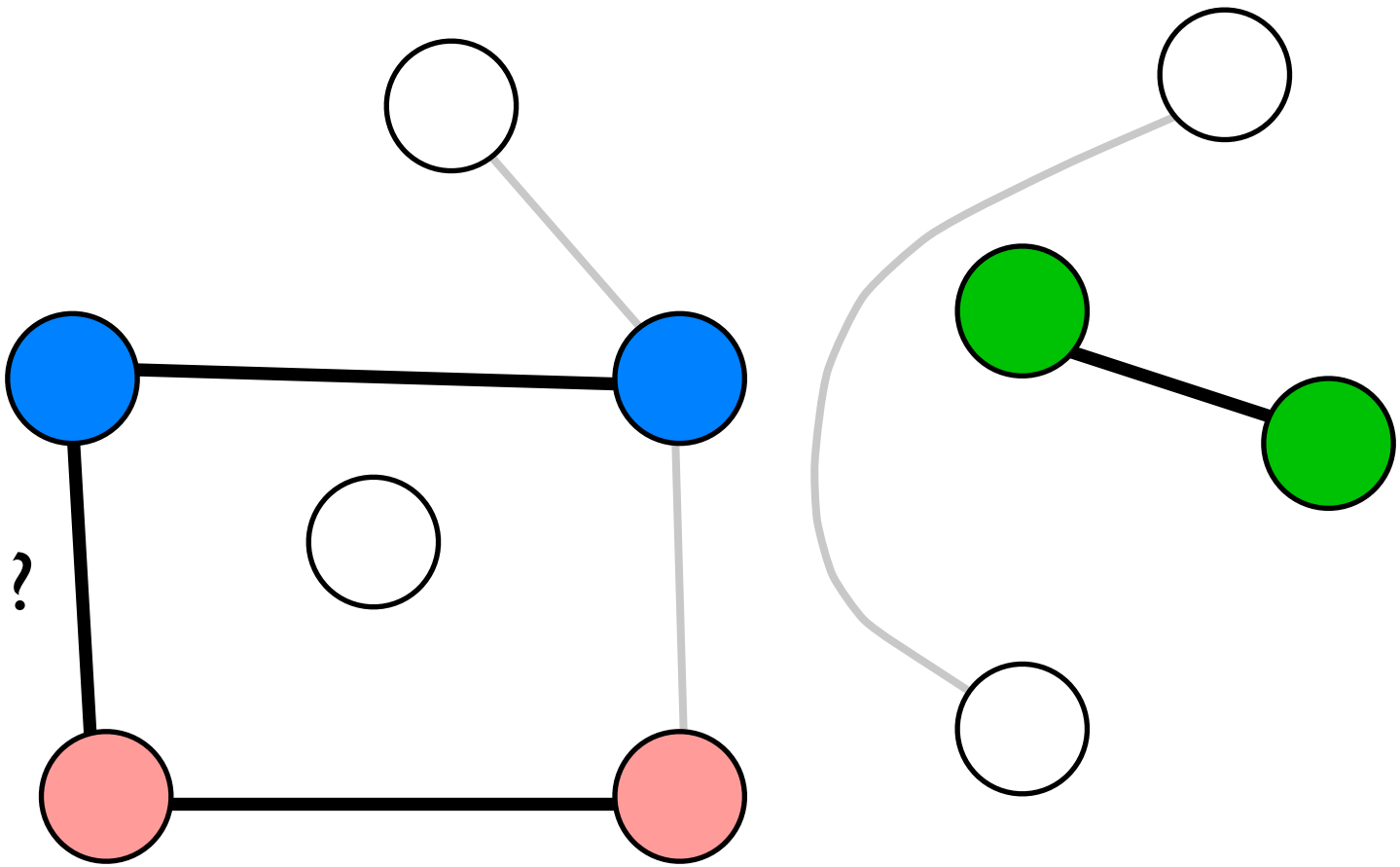


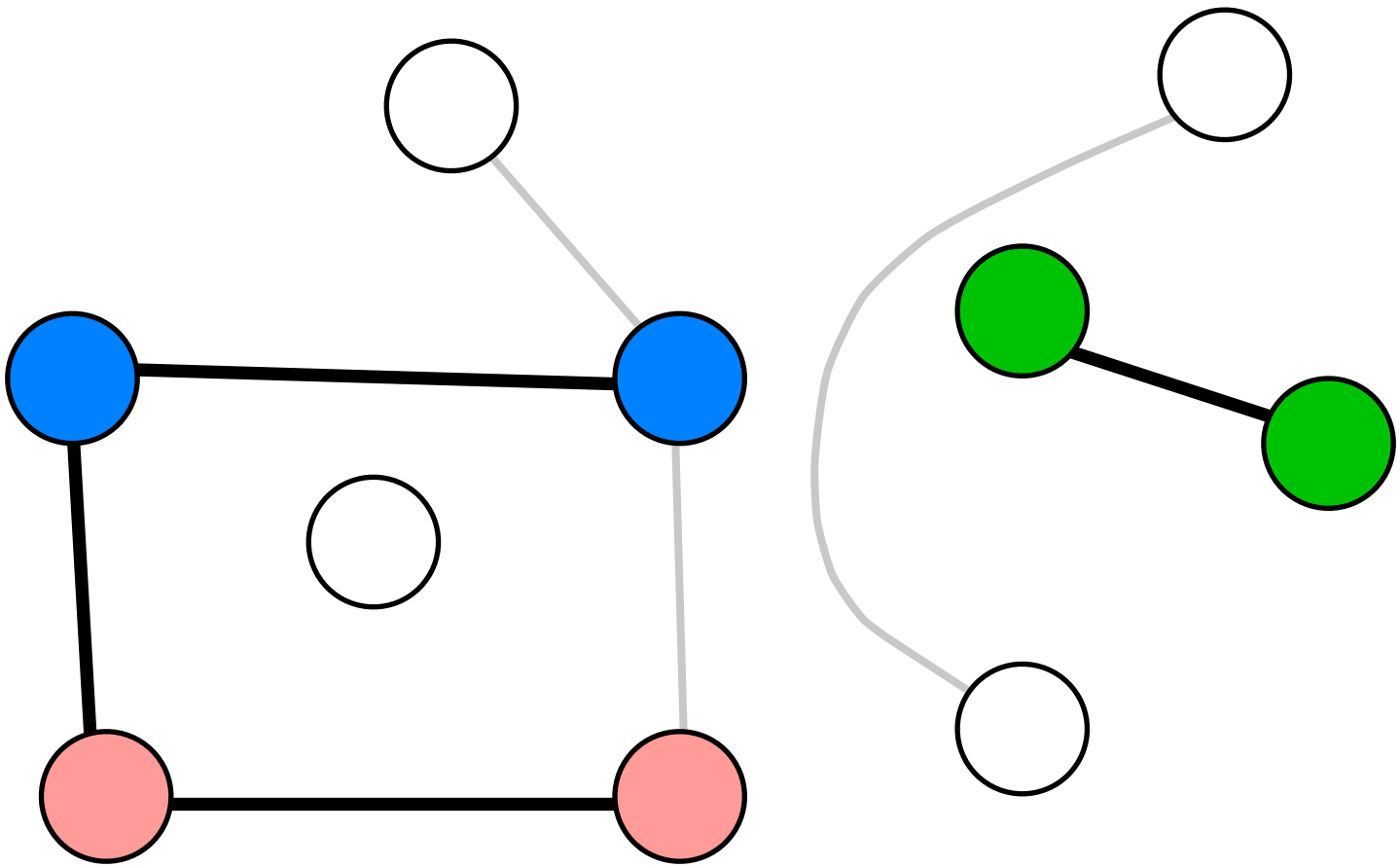






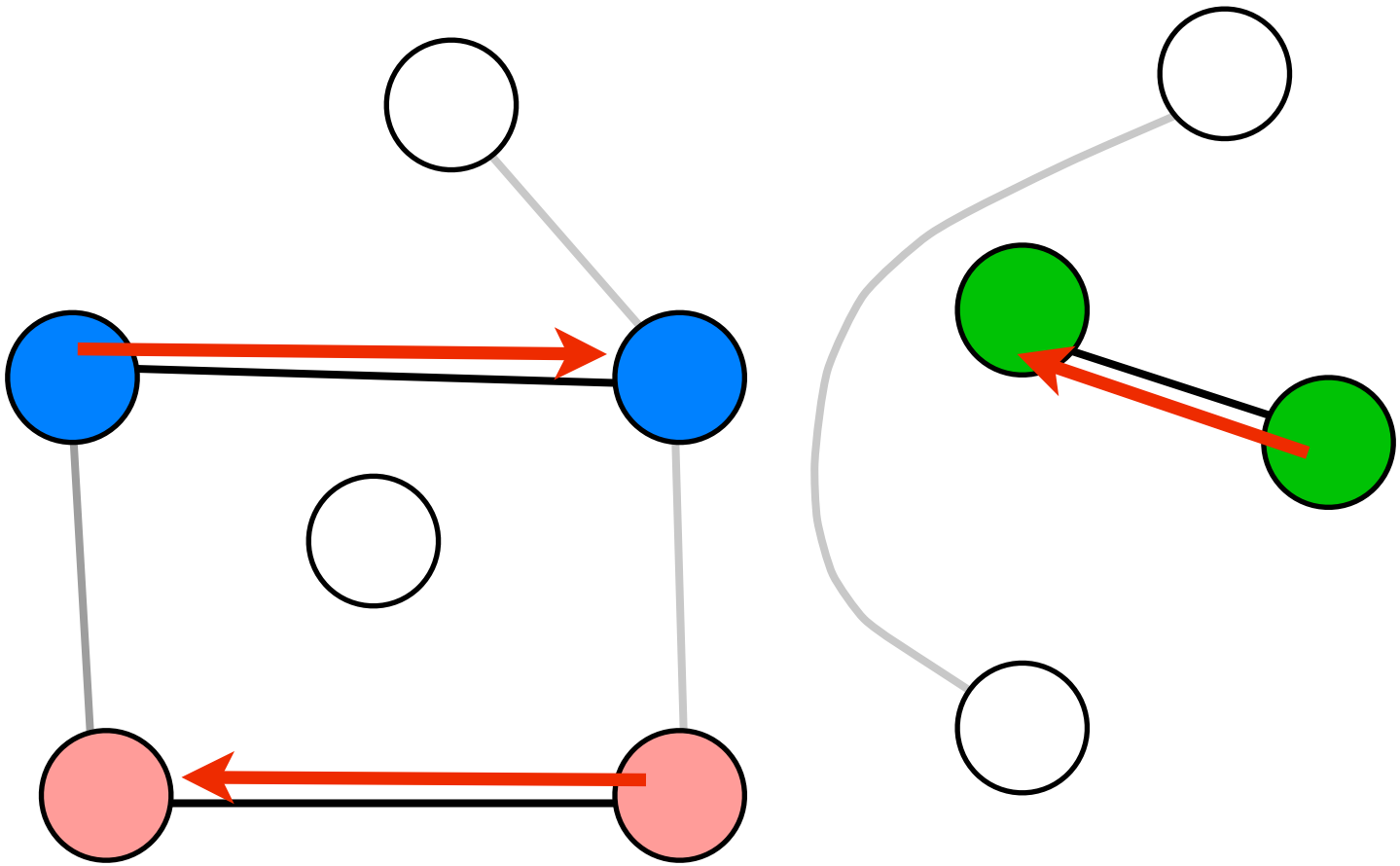
Que faire ?





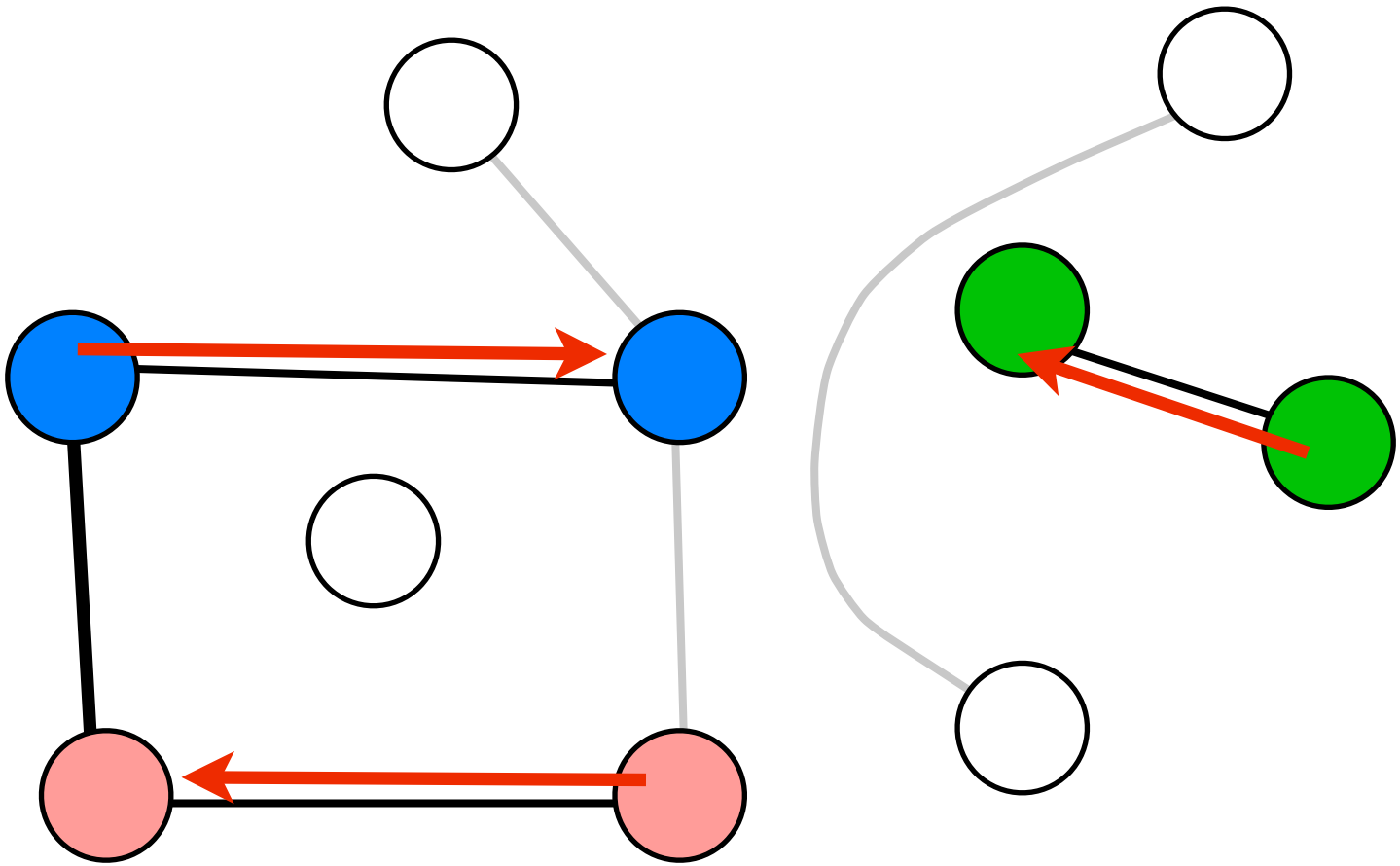
Quelle représentation pour les classes d'équivalence ?

→ Représentant canonique !



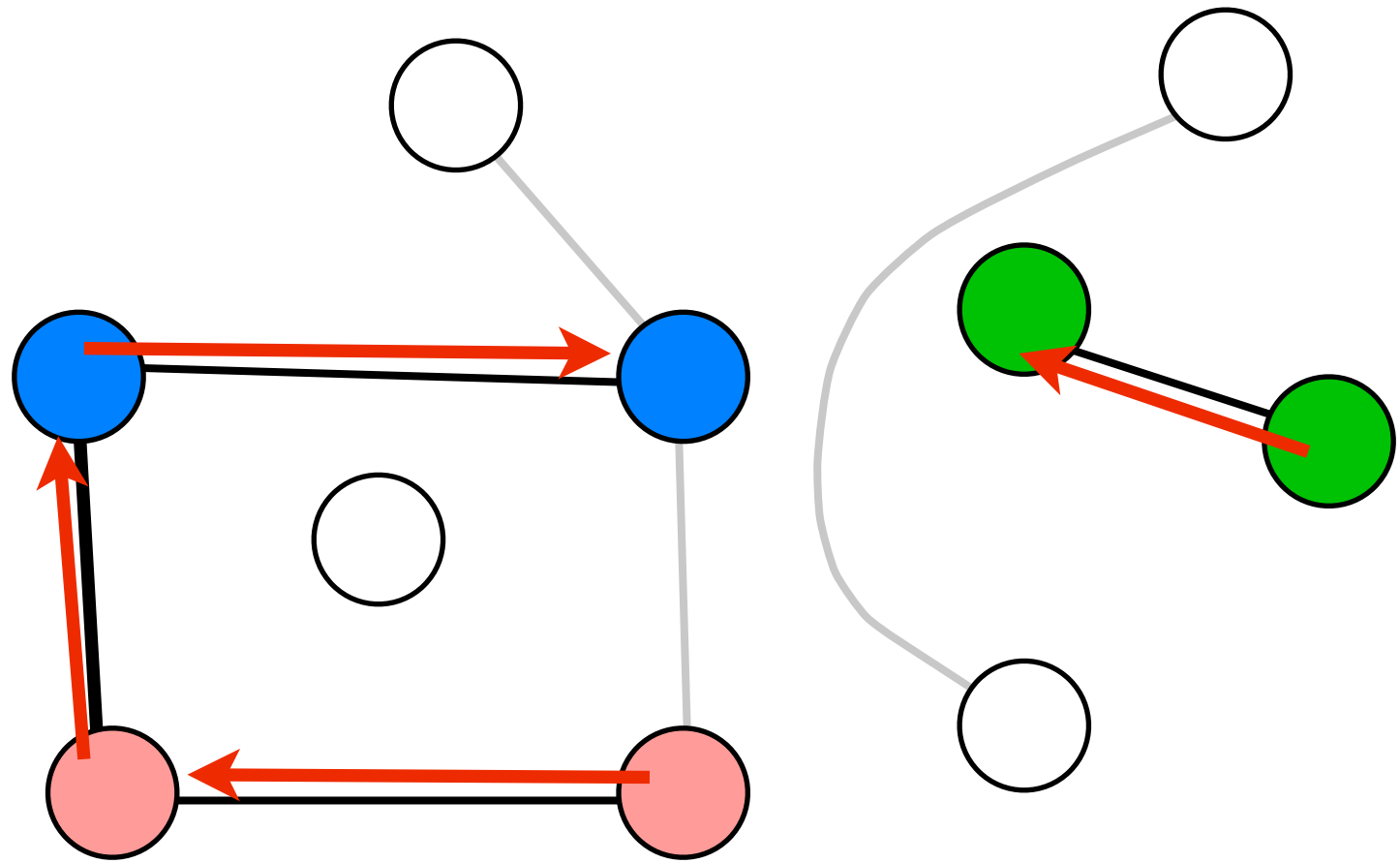
Quelle représentation pour les classes d'équivalence ?

→ Représentant canonique !



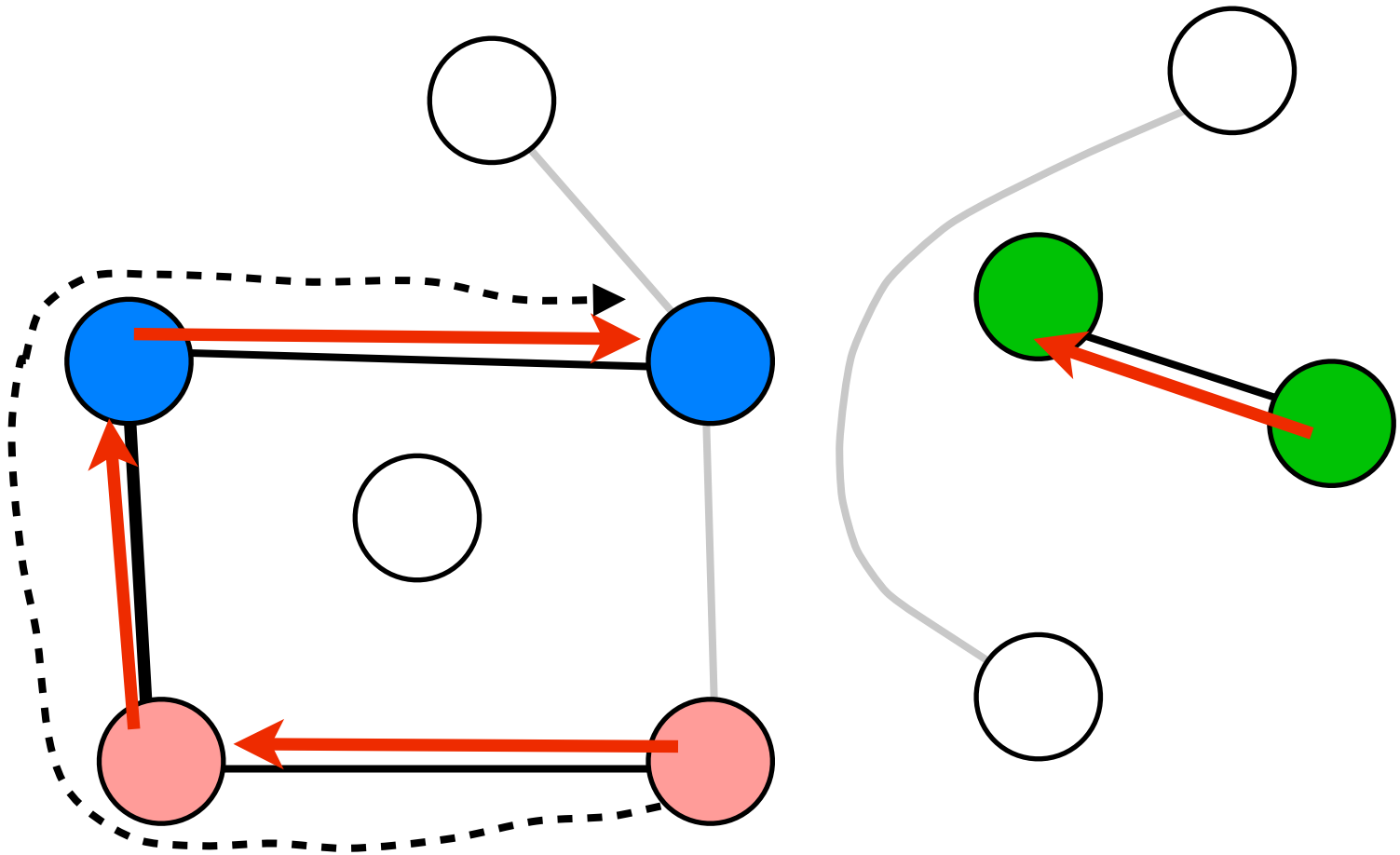
Quelle représentation pour les classes d'équivalence ?

→ Représentant canonique !



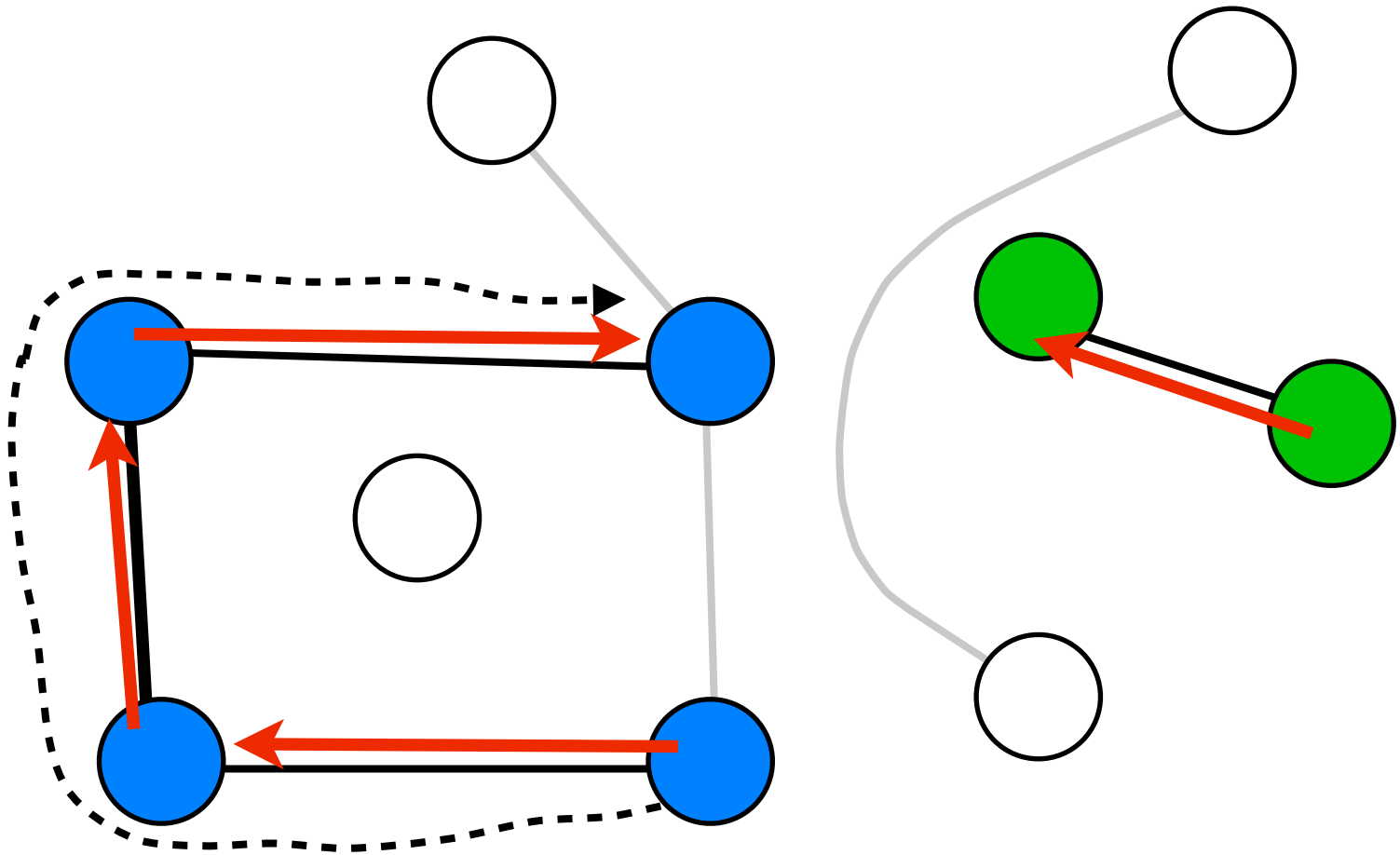
Quelle représentation pour les classes d'équivalence ?

→ Représentant canonique !



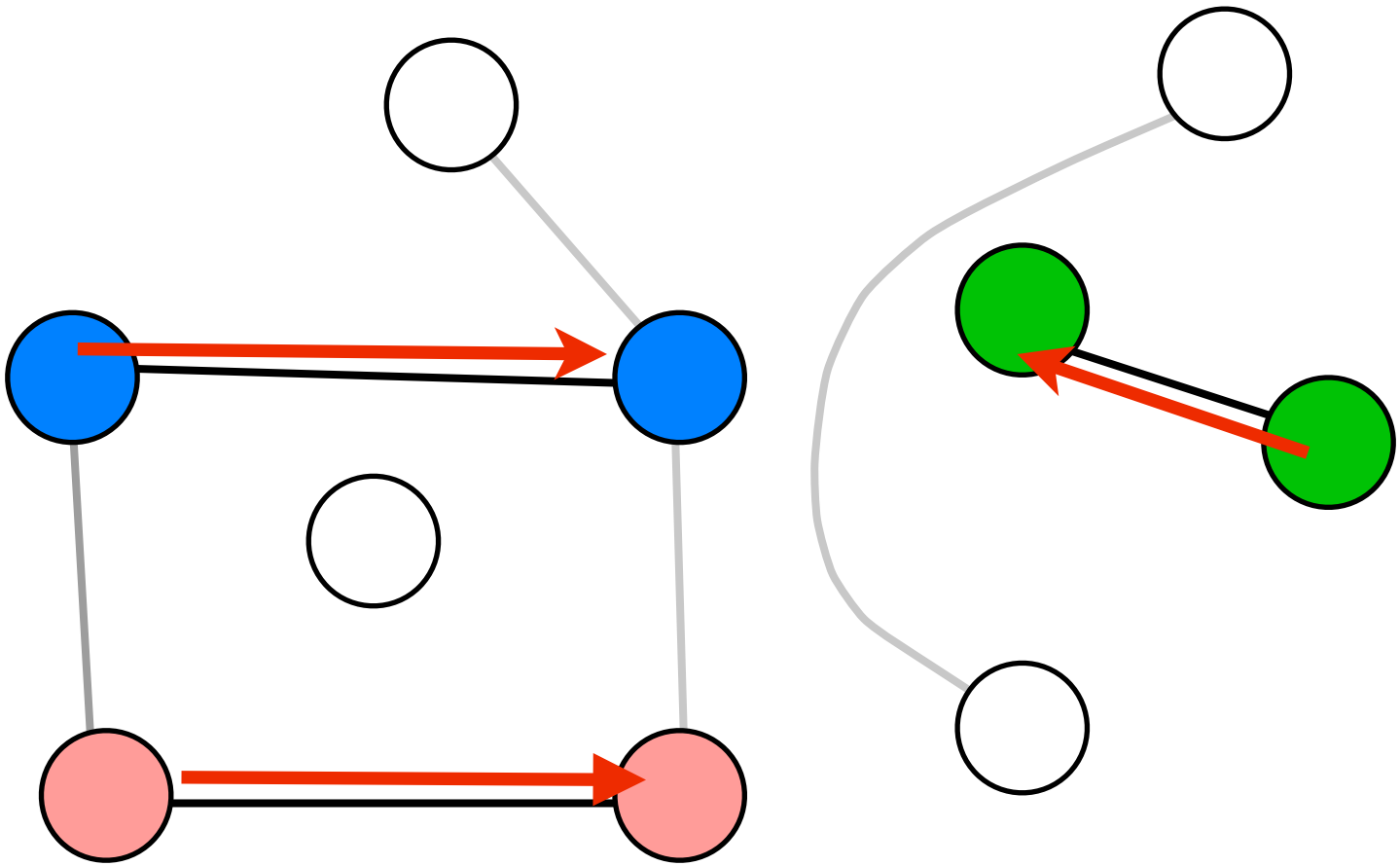
Quelle représentation pour les classes d'équivalence ?

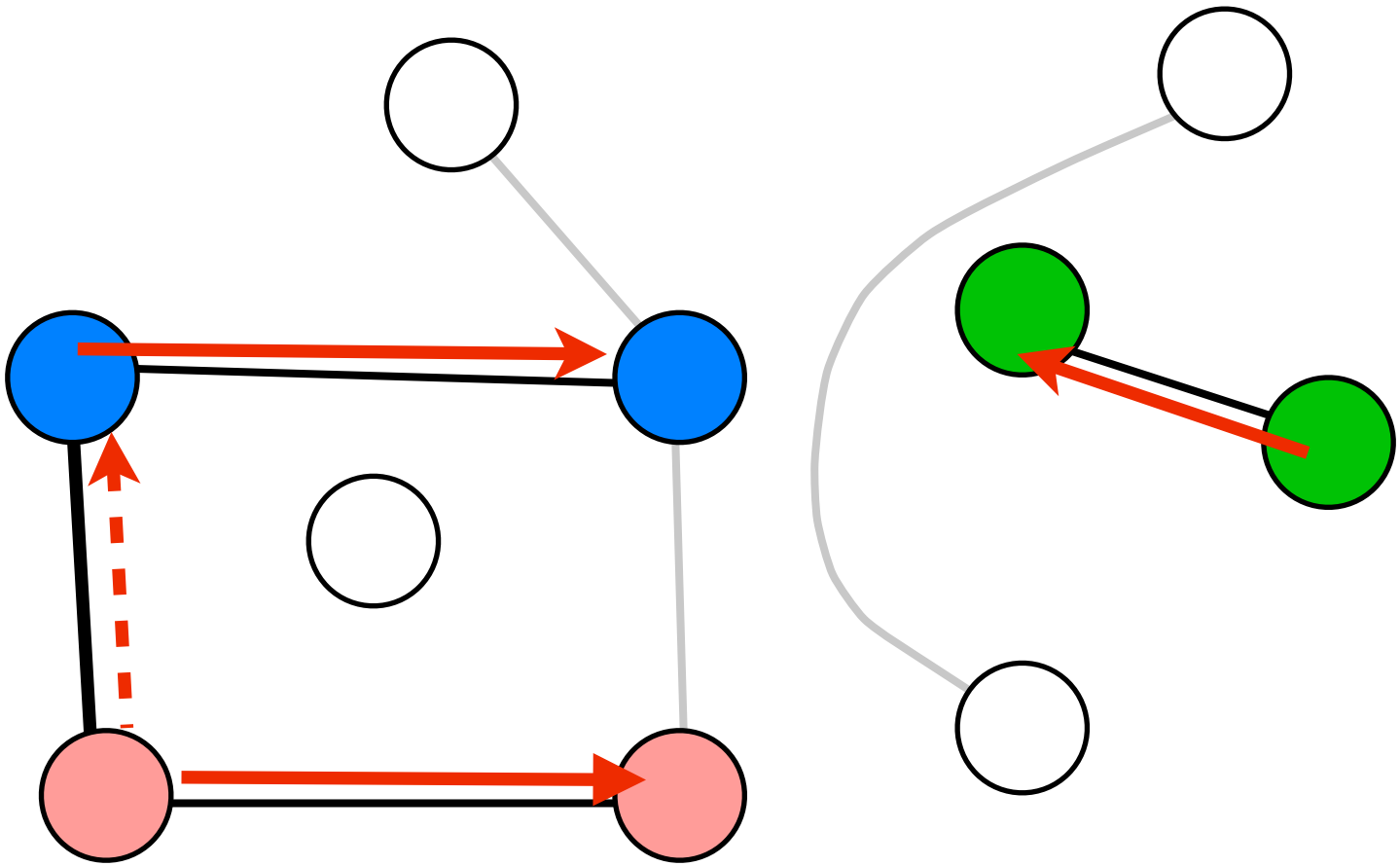
→ Représentant canonique !

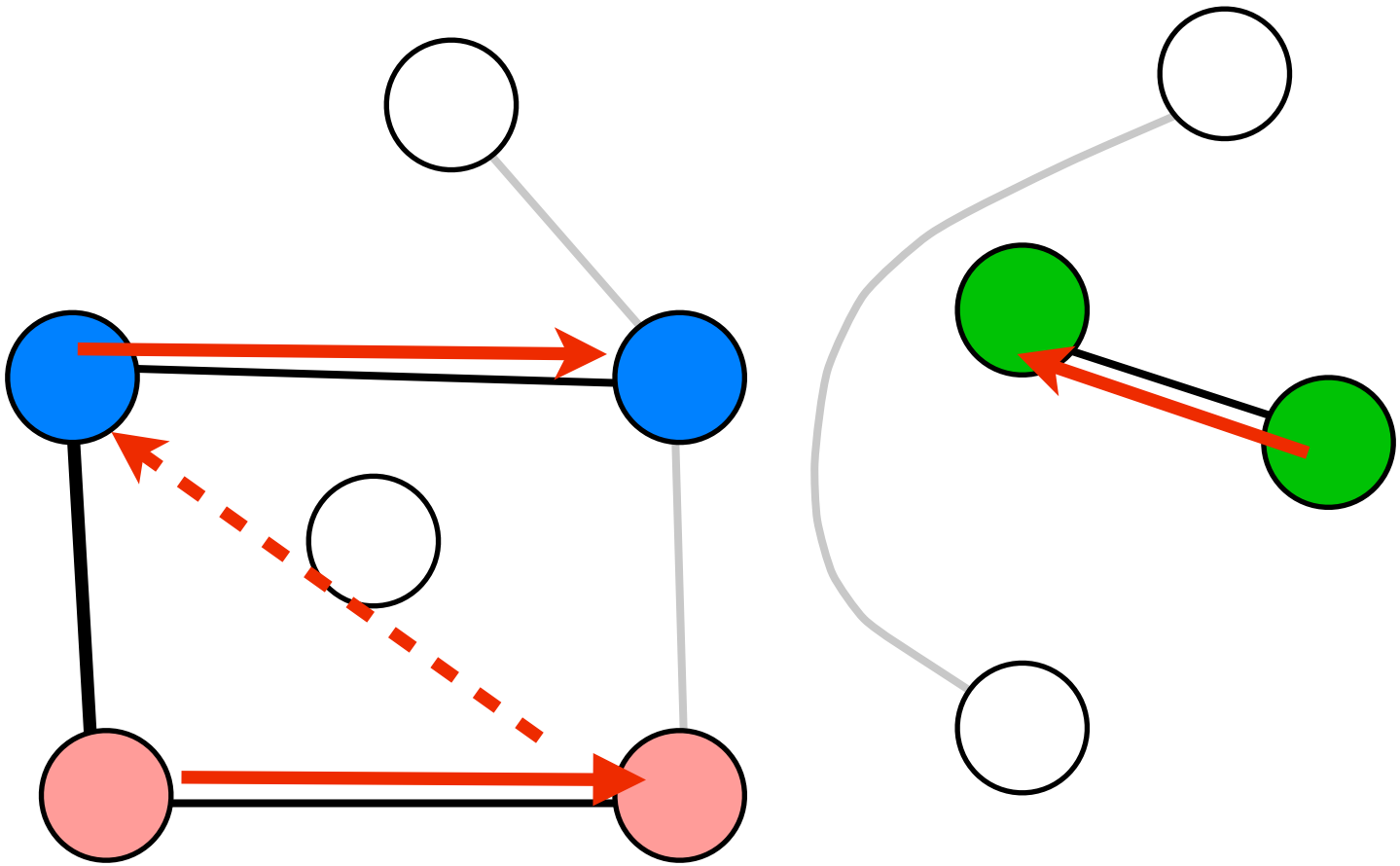


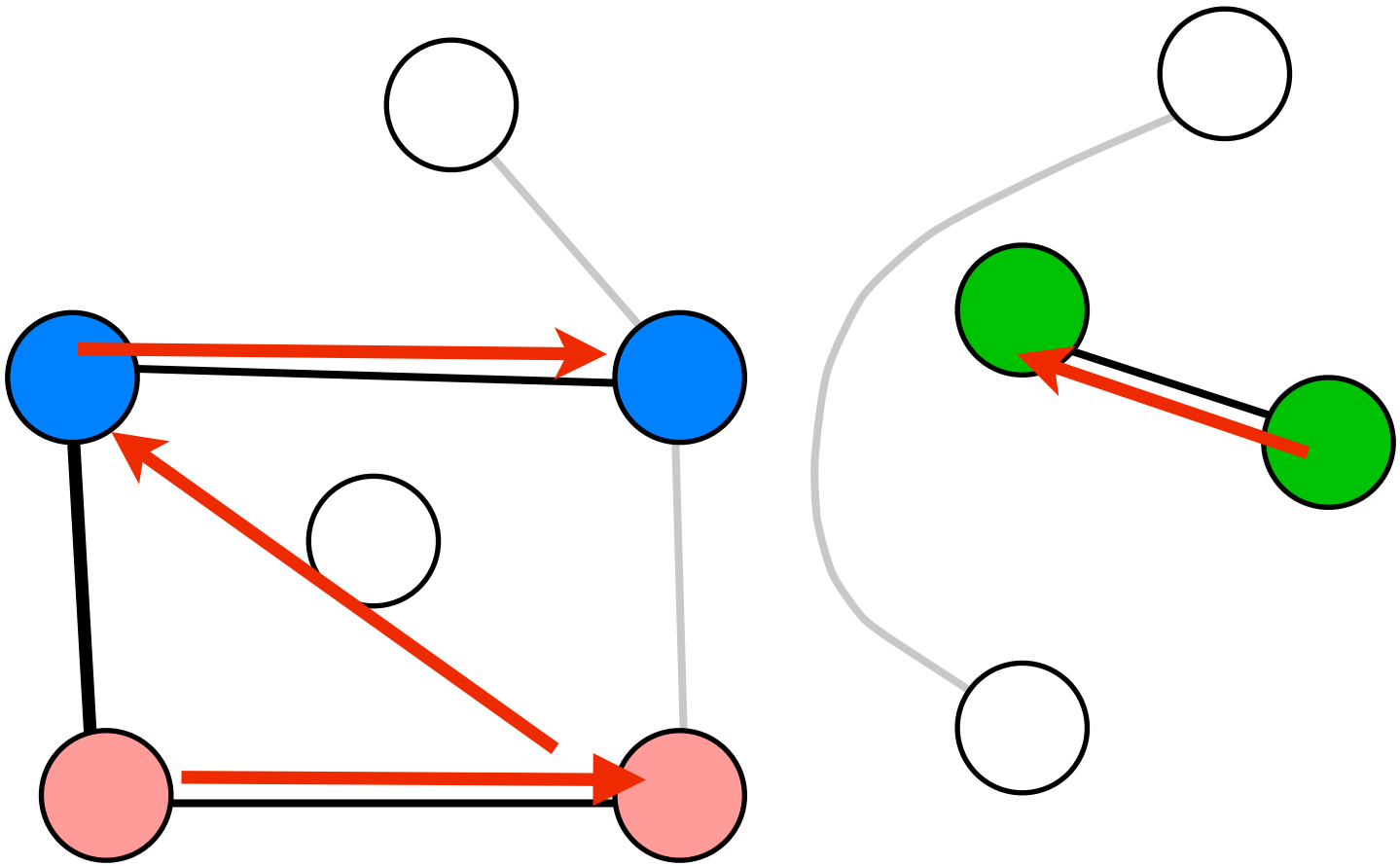
Quelle représentation pour les classes d'équivalence ?

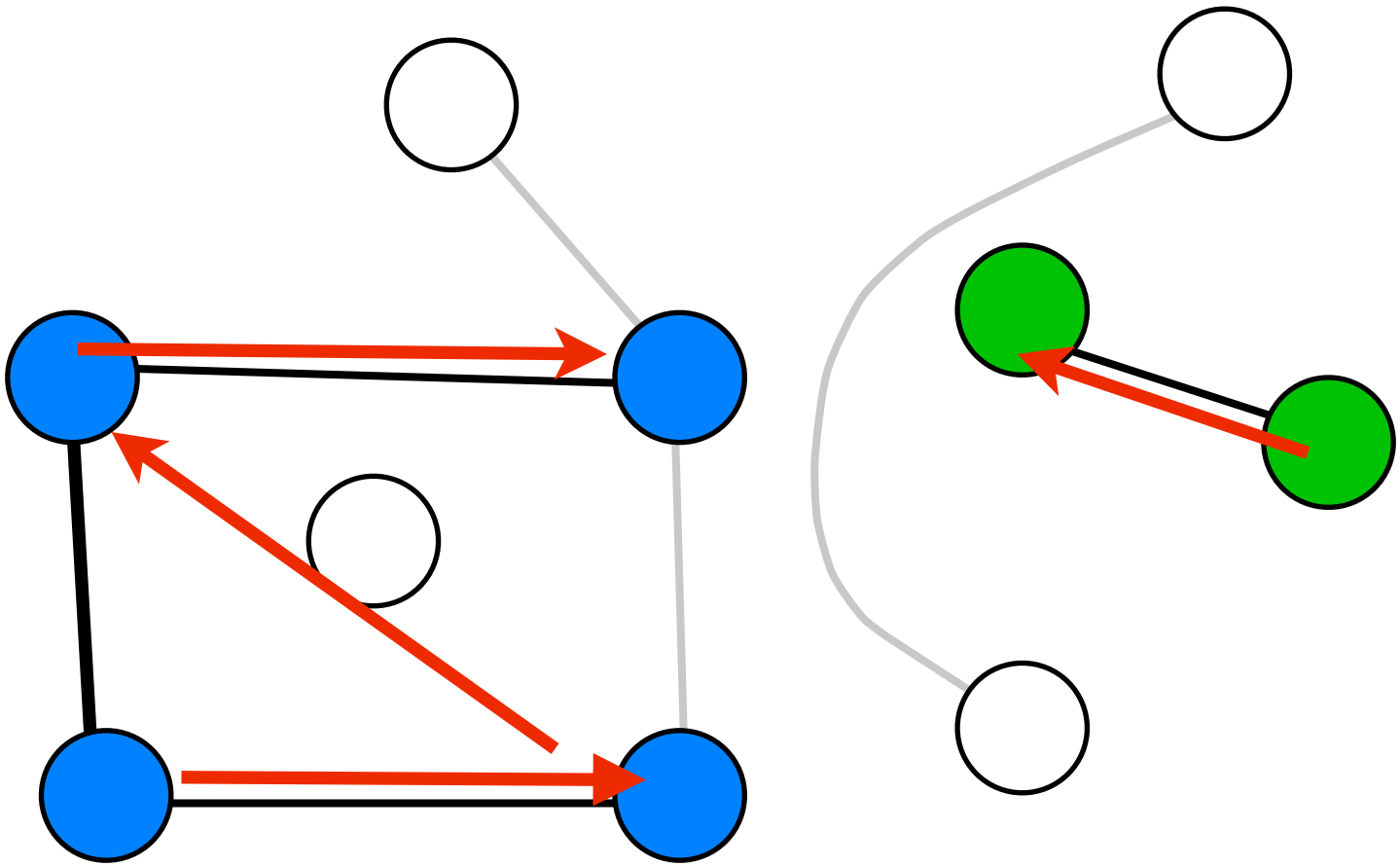
→ Représentant canonique !

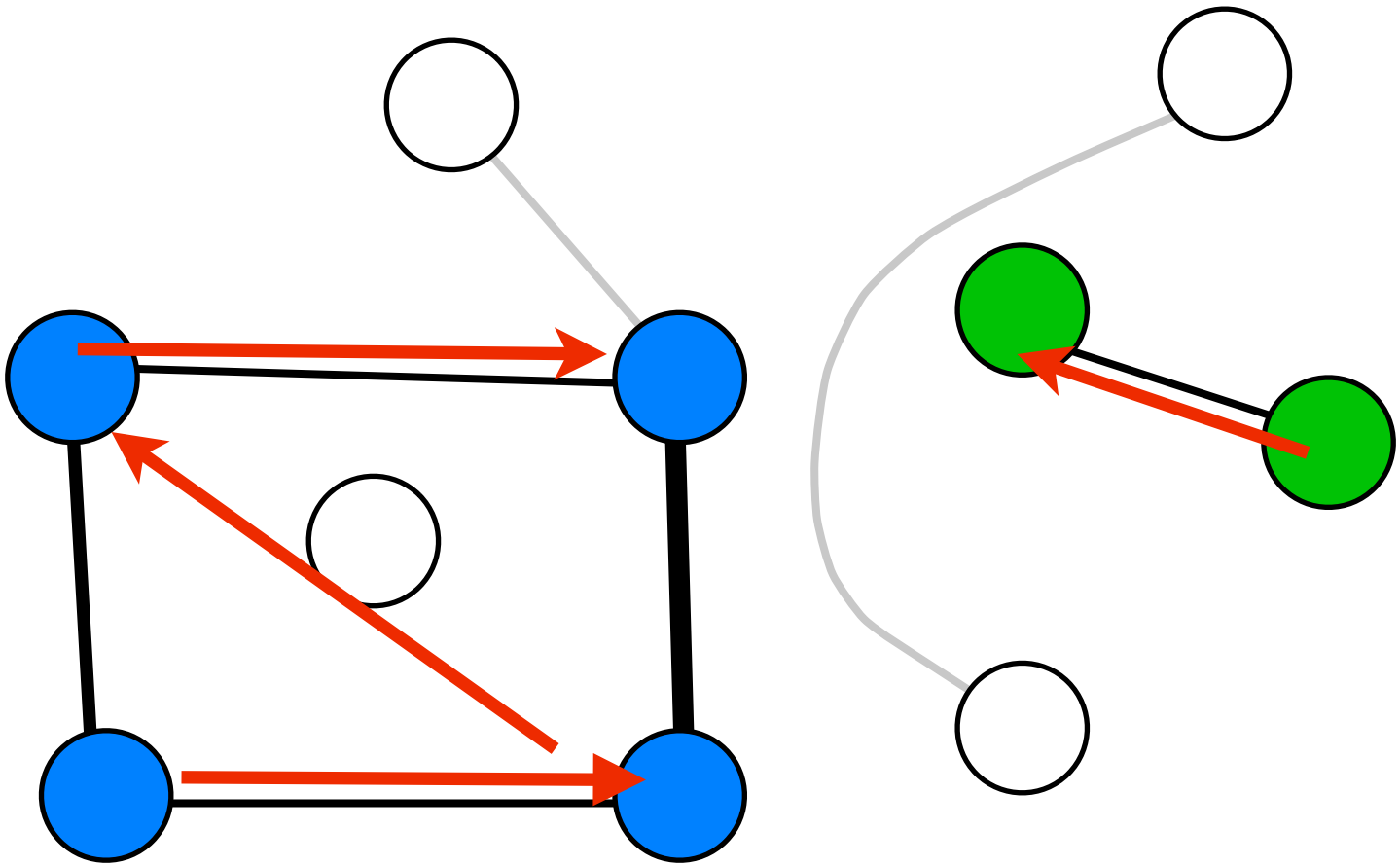


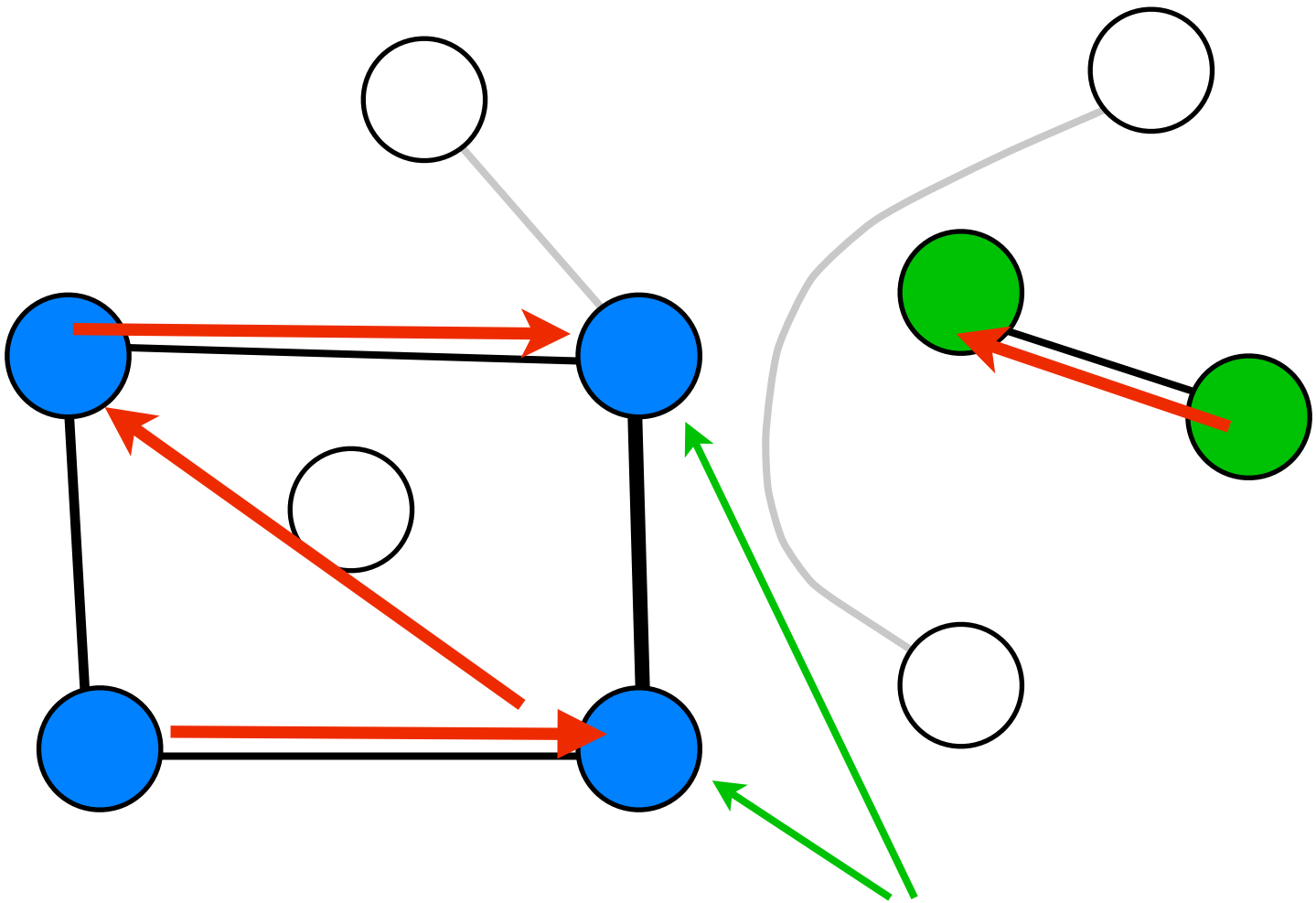




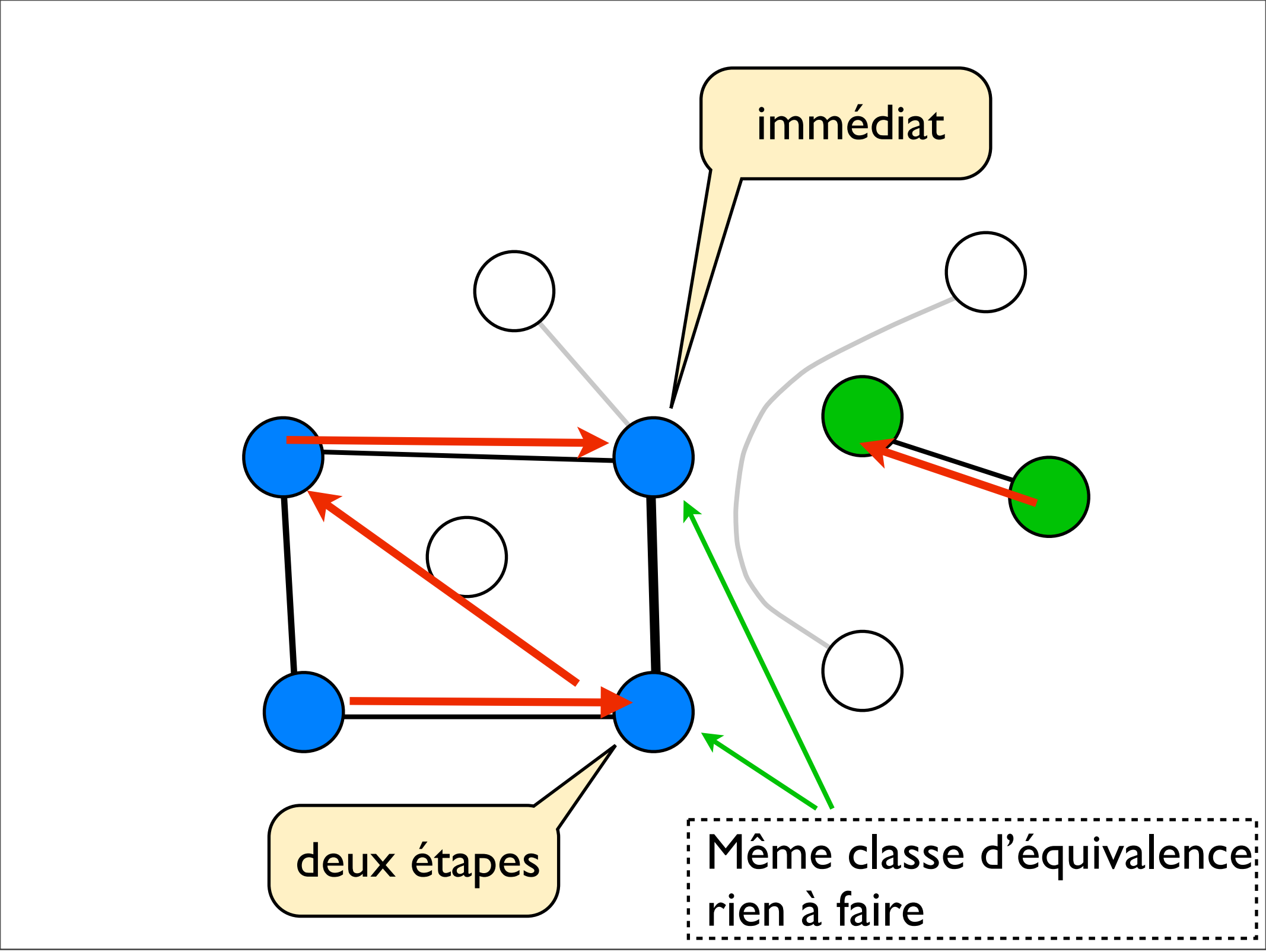


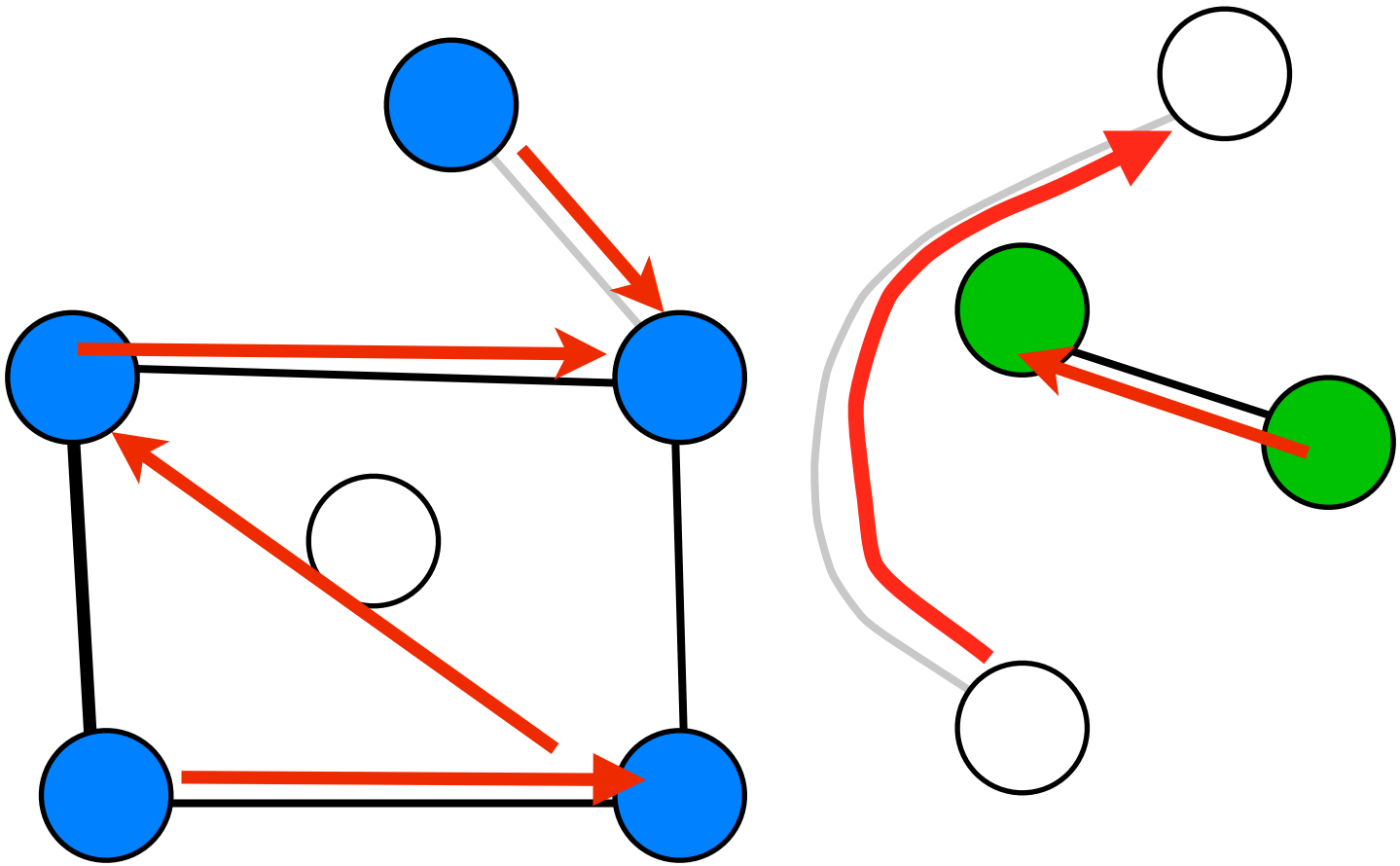


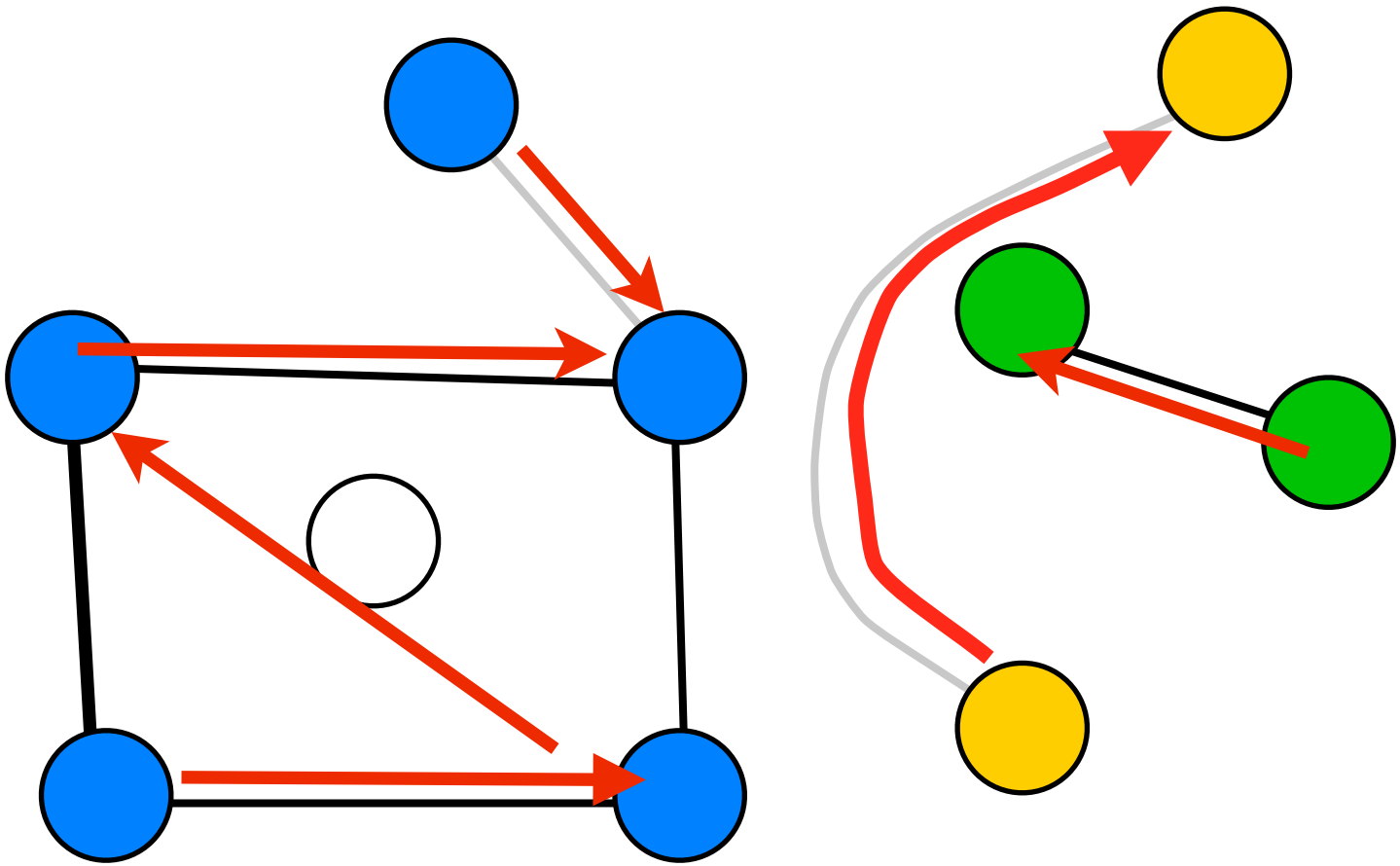


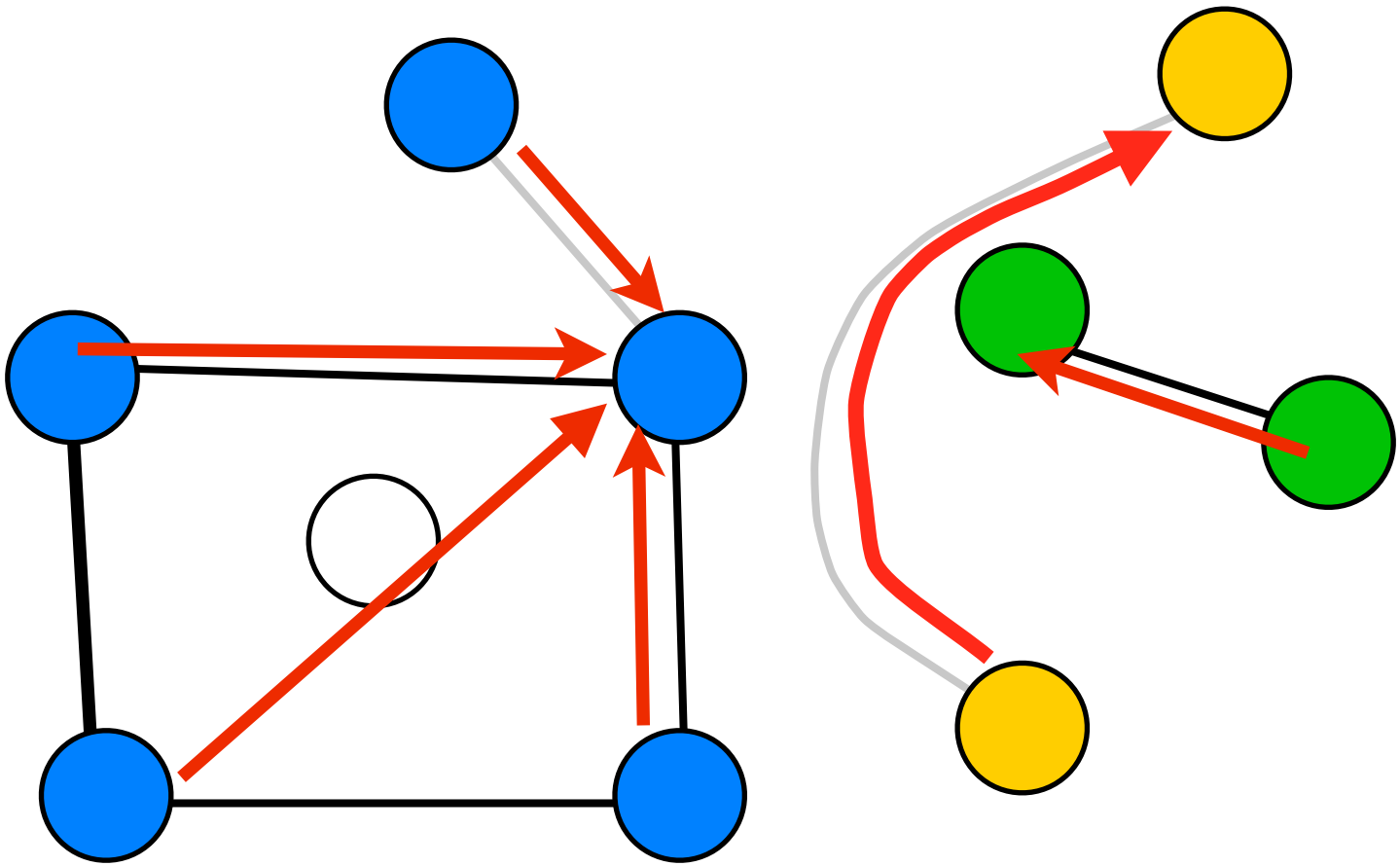


Même classe d'équivalence
rien à faire


immédiat
deux étapes
Même classe d'équivalence
rien à faire

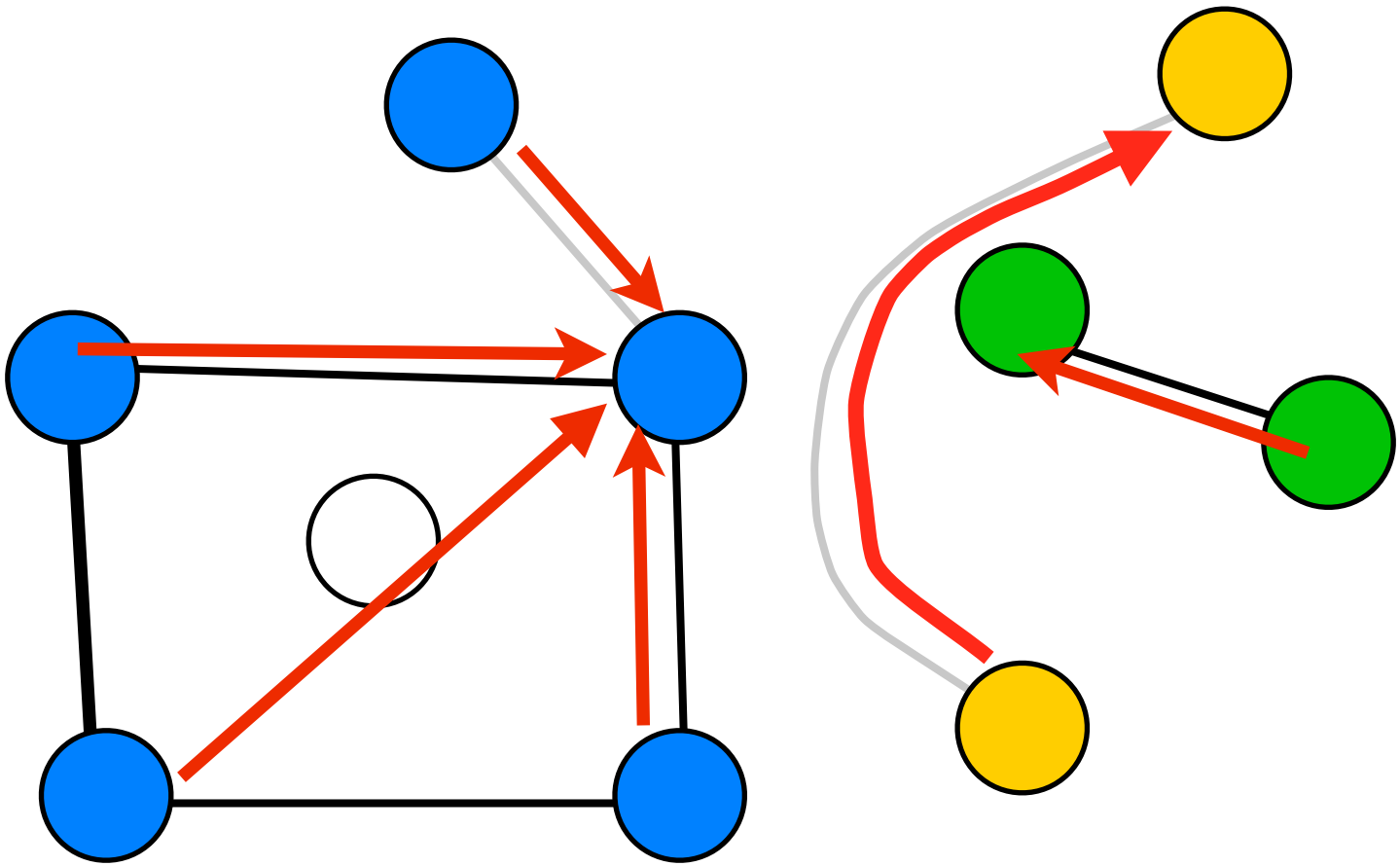






Optimisation: aplatir les composantes au fur et à mesure

Union-Find



Optimisation: aplatir les composantes au fur et à mesure

Parcours

DFS
depth-first search
profondeur d'abord

1895: Tarry

LE PROBLÈME DES LABYRINTHES;

PAR M. G. TARRY.

Tout labyrinthe peut être parcouru en une seule course, en passant deux fois en sens contraire par chacune des allées, sans qu'il soit nécessaire d'en connaître le plan.

Pour résoudre ce problème, il suffit d'observer cette règle unique :

Ne reprendre l'allée initiale qui a conduit à un carrefour pour la première fois que lorsqu'on ne peut pas faire autrement.

Nous ferons d'abord quelques remarques.

A un moment quelconque, avant d'arriver à un car-

1936: Kőnig

Drittes Kapitel.

Das Labyrinthenproblem.

§ 1. Formulierung des Problems. Die Lösung von Wiener.

Der Satz II 5 steht in engem Zusammenhang mit dem *Problem der Labyrinthe*[1]), das in diesem Kapitel behandelt werden soll. Da die Breite der Gänge nicht in Betracht kommt, darf ein Labyrinth mit einem Graphen und zwar mit einem *endlichen* und *zusammenhängenden* Graphen *identifiziert* werden; den Knotenpunkten dieses Graphen entsprechen im Labyrinth die Verzweigungsstellen und die Endpunkte der Sackgassen. So entspricht z. B. dem Labyrinth der Fig. 18 der Graph der Fig. 19 (oder auch der

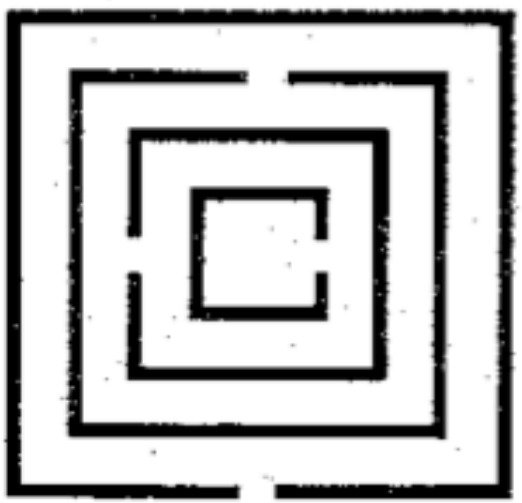


Fig. 18.

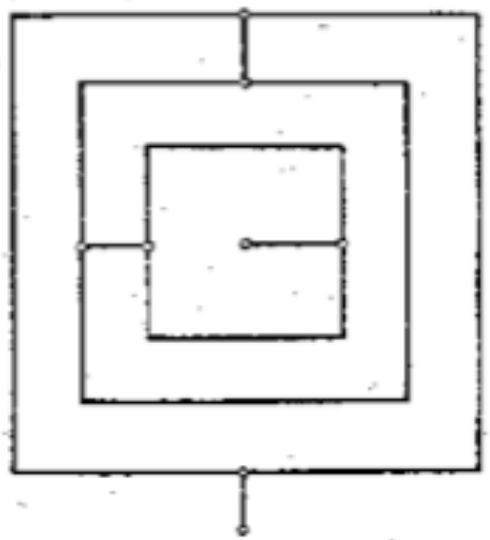


Fig. 19.

1958: Berge

Autre algorithme pour le problème 1 (G. TARRY [5]). — Pour le voyageur perdu dans le labyrinthe, il suffit d'observer la règle suivante :

Ne jamais parcourir deux fois la même arête dans le même sens ; si on est en x, ne prendre l'arête qui nous a conduit la première fois au carrefour x que lorsqu'on ne peut pas faire autrement.

1° Montrons que le voyageur, lorsqu'il est arrêté parce qu'il ne peut plus observer la règle, se trouve en a et que toute arête incidente à a a été parcourue dans les deux directions. En effet, lorsqu'on se trouve en x (avec $x \neq a$), soit k le nombre de fois où ce carrefour a été rencontré antérieurement ; les arêtes incidentes à x ont été parcourues $(k + 1)$ fois vers x et k fois en s'éloignant de x, donc on peut toujours repartir par une arête qui n'a pas encore été utilisée pour s'éloigner de x.

2° Soit $(a_0 = a, a_1, a_2, \ldots)$ la suite des différents carrefours rencontrés, indexés dans l'ordre où ils ont été rencontrés pour la première fois ; montrons que toute arête incidente à a_k a été parcourue deux fois. En effet, ceci est vrai pour $k = 0$ (d'après le 1°) ; supposons que cette proposition soit vraie pour $k \leqslant p - 1$, et montrons qu'elle est alors vraie pour $k = p$. Soit (a_i, a_p) l'arête qui a conduit le voyageur à a_p pour la première fois ; comme $i < p$, cette arête est parcourue dans les deux directions, donc, d'après la règle de Tarry, ceci n'est possible que si toutes les arêtes incidentes à a_p ont été parcourues dans les deux directions. C. Q. F. D.

3° S'il existe un chemin $(a, x_1, x_2, \ldots, x_m, b)$ pour aller de a à b, il est évident que le carrefour b sera tôt au tard rencontré ; car, d'après le 2°, tout sommet adjacent à un sommet rencontré est lui-même un sommet rencontré.

1980: Eco, “earlier than 1327” (?)

« Pour trouver la sortie d'un labyrinthe, récita en effet Guillaume, il n'y a qu'un moyen. A chaque nœud nouveau, autrement dit jamais visité avant, le parcours d'arrivée sera marqué de trois signes. Si, à cause de signes précédents sur l'un des chemins du nœud, on voit que ce nœud a déjà été visité, on placera un seul signe sur le parcours d'arrivée. Si tous les passages ont été déjà marqués, alors il faudra reprendre la même voie, en revenant en arrière. Mais si un ou deux passages du nœud sont encore sans signes, on en choisira un quelconque, pour y apposer deux signes. Quand on s'achemine par un passage qui porte un seul signe, on en appposera deux autres, de façon que ce passage en porte trois dorénavant. Toutes les parties du labyrinthe devraient avoir été parcourues si, en arrivant à un nœud, on ne prend jamais le passage avec trois signes, sauf si d'autres passages sont encore sans signes.

– Comment le savez-vous? Vous êtes expert en labyrinthes?

– Non, je récite un extrait d'un texte antique que j'ai lu autrefois.

– Et selon cette règle, on sort?

– Presque jamais, que je sache. Mais nous tenterons quand même. Et puis dans les prochains jours j'aurai des verres et j'aurai le temps de mieux me pencher sur les livres. Il se peut que là où le parcours des cartouches nous embrouille, celui des livres nous donne une règle.

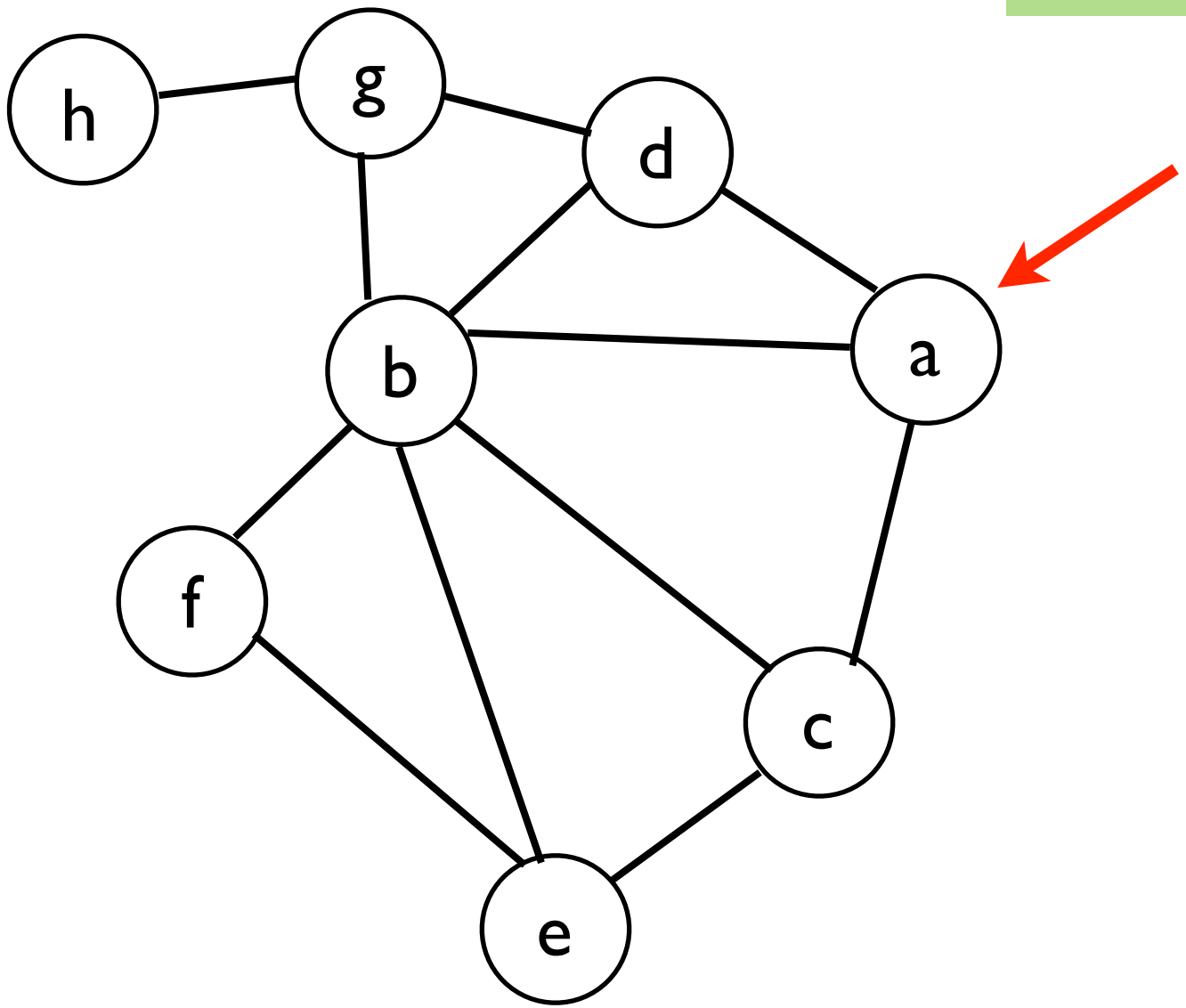
DFS

Applications multiples:

- Sortir d'un labyrinthe
- Trouver des composantes (fortement) connexes
- Détecter des cycles
- Tri topologique (en l'absence de cycles)
- GC

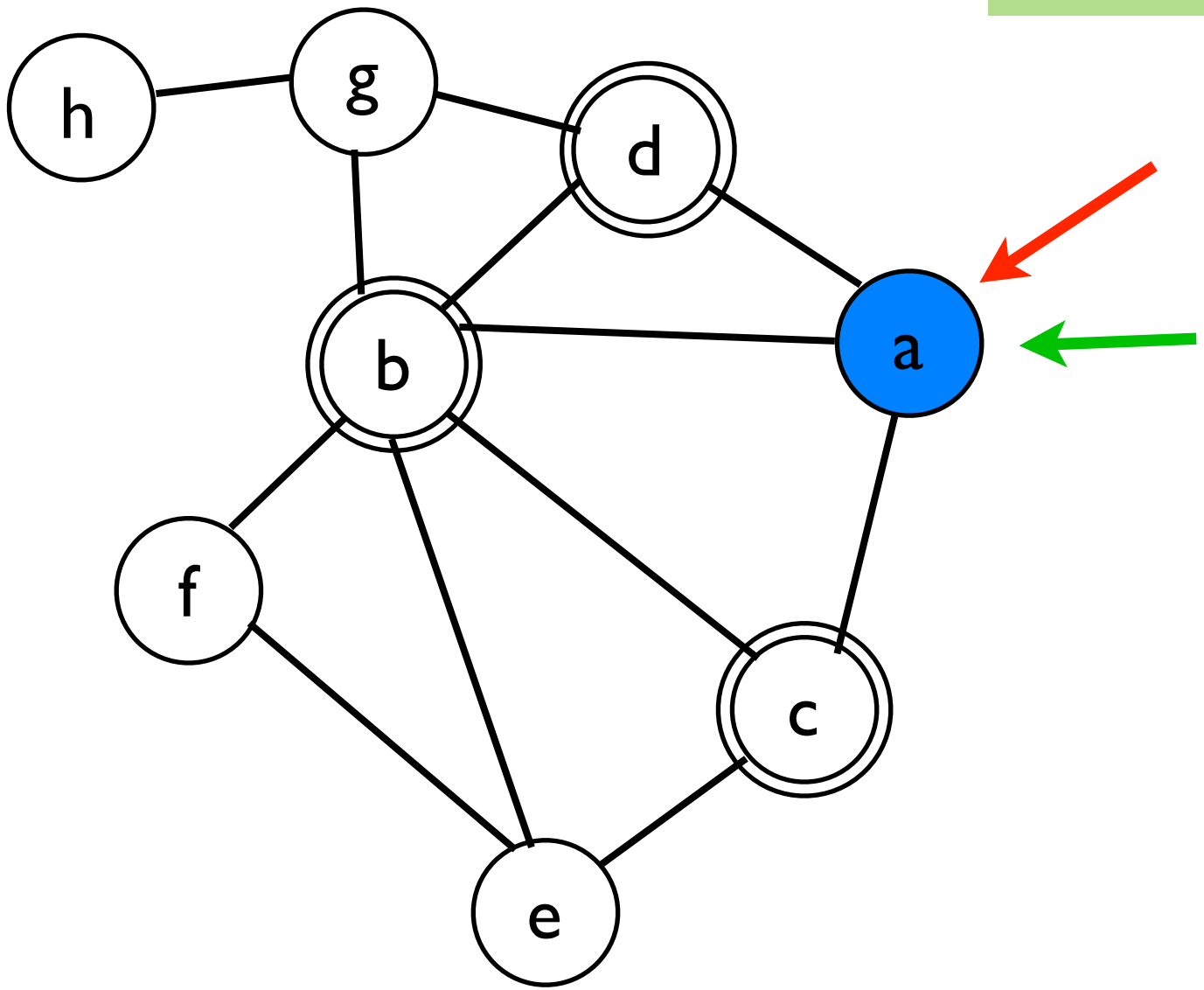
Cette semaine: seulement pour les graphes non-orientés

DFS



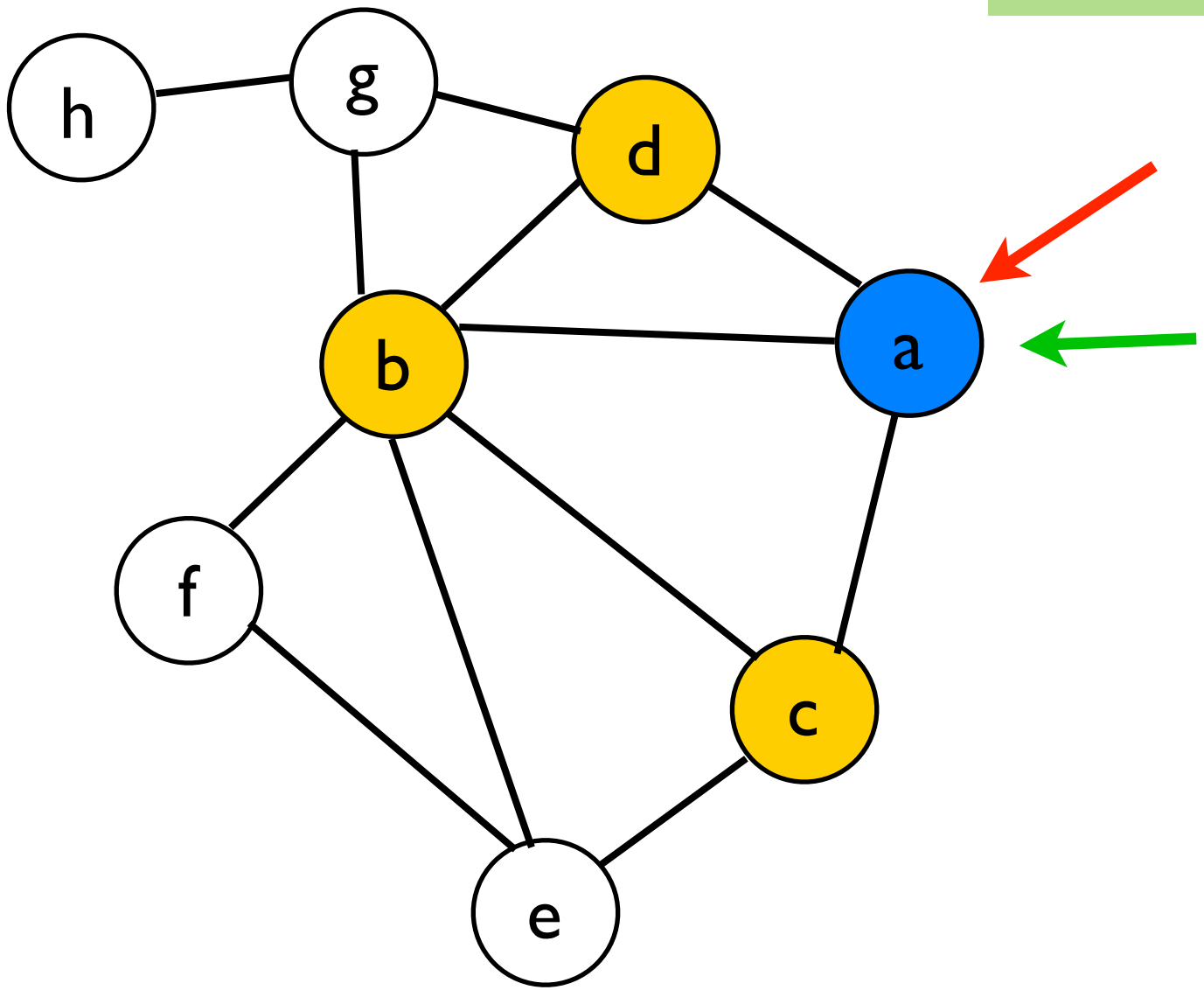
ordre: a

DFS



ordre: a

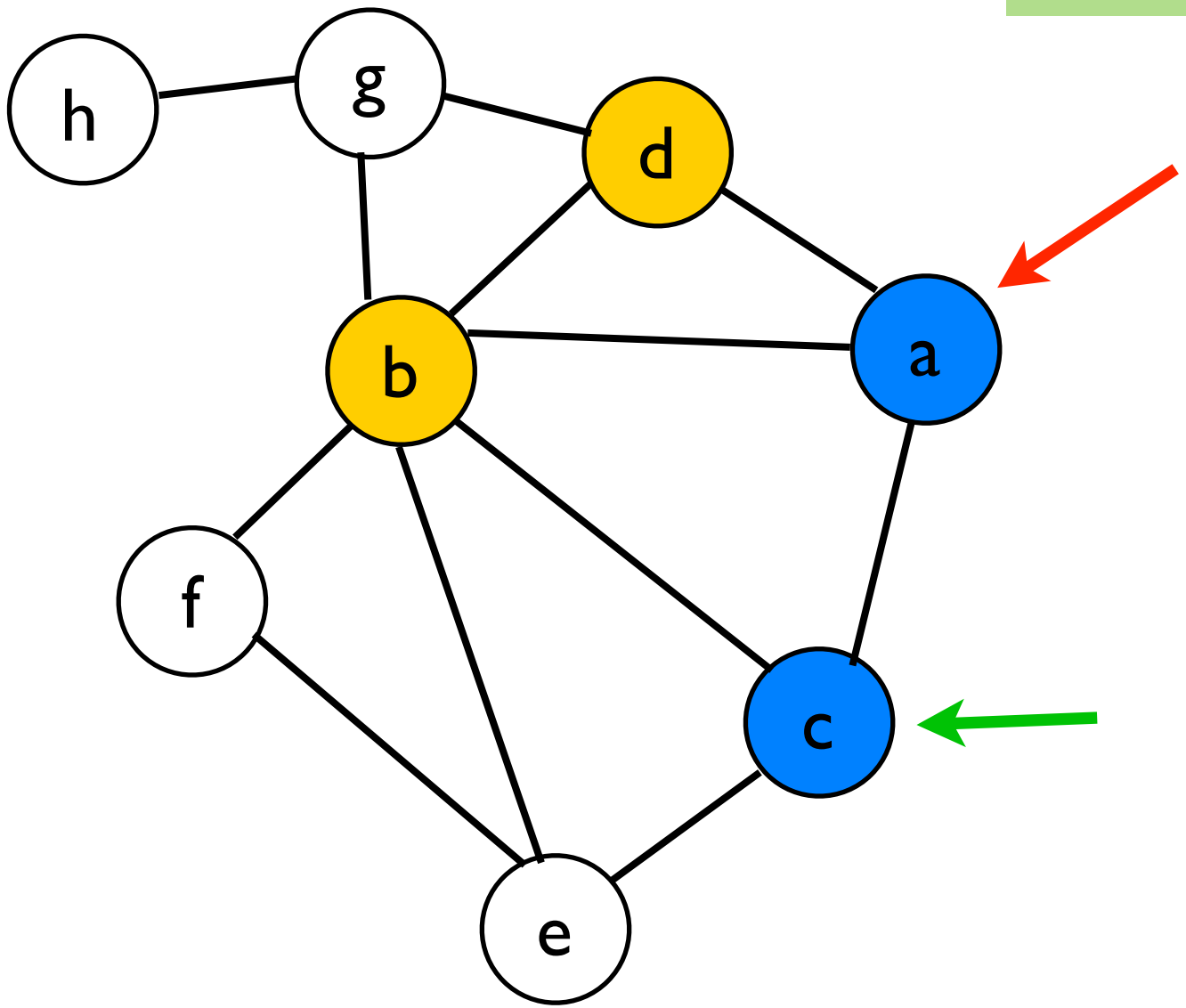
DFS



ordre: a

Pile: c b d

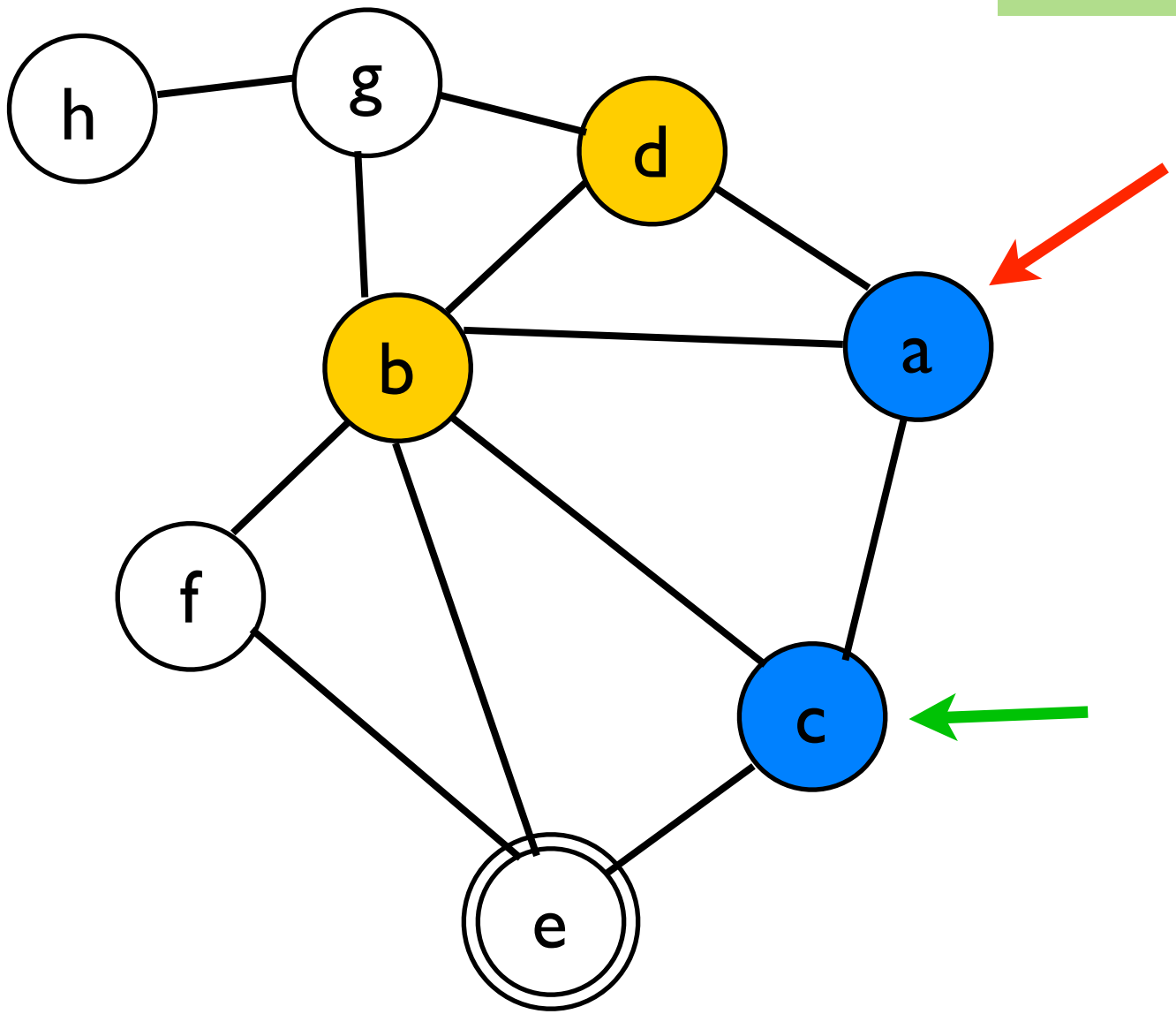
DFS



ordre: a c

Pile: b d

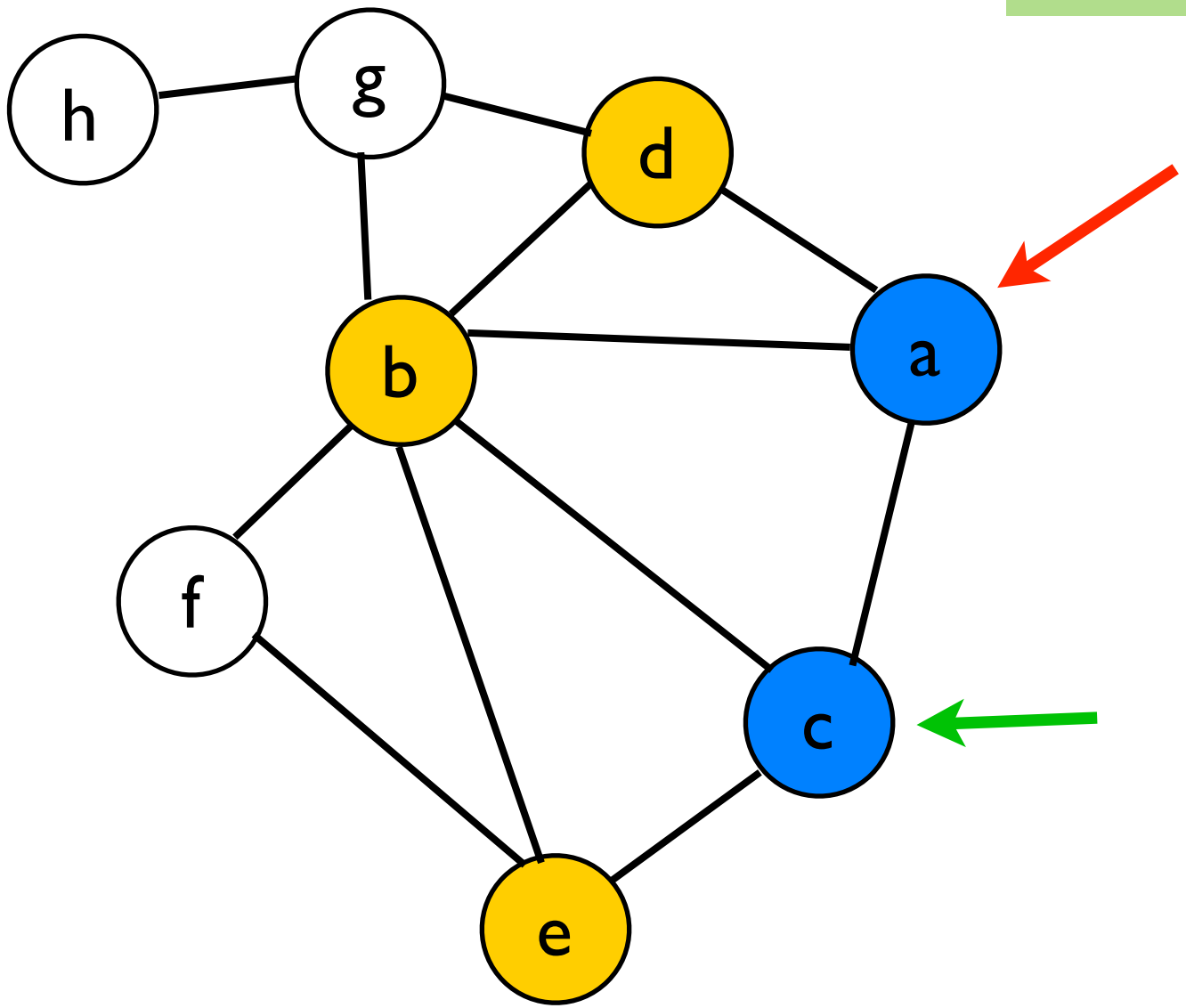
DFS



ordre: a c

Pile: b d

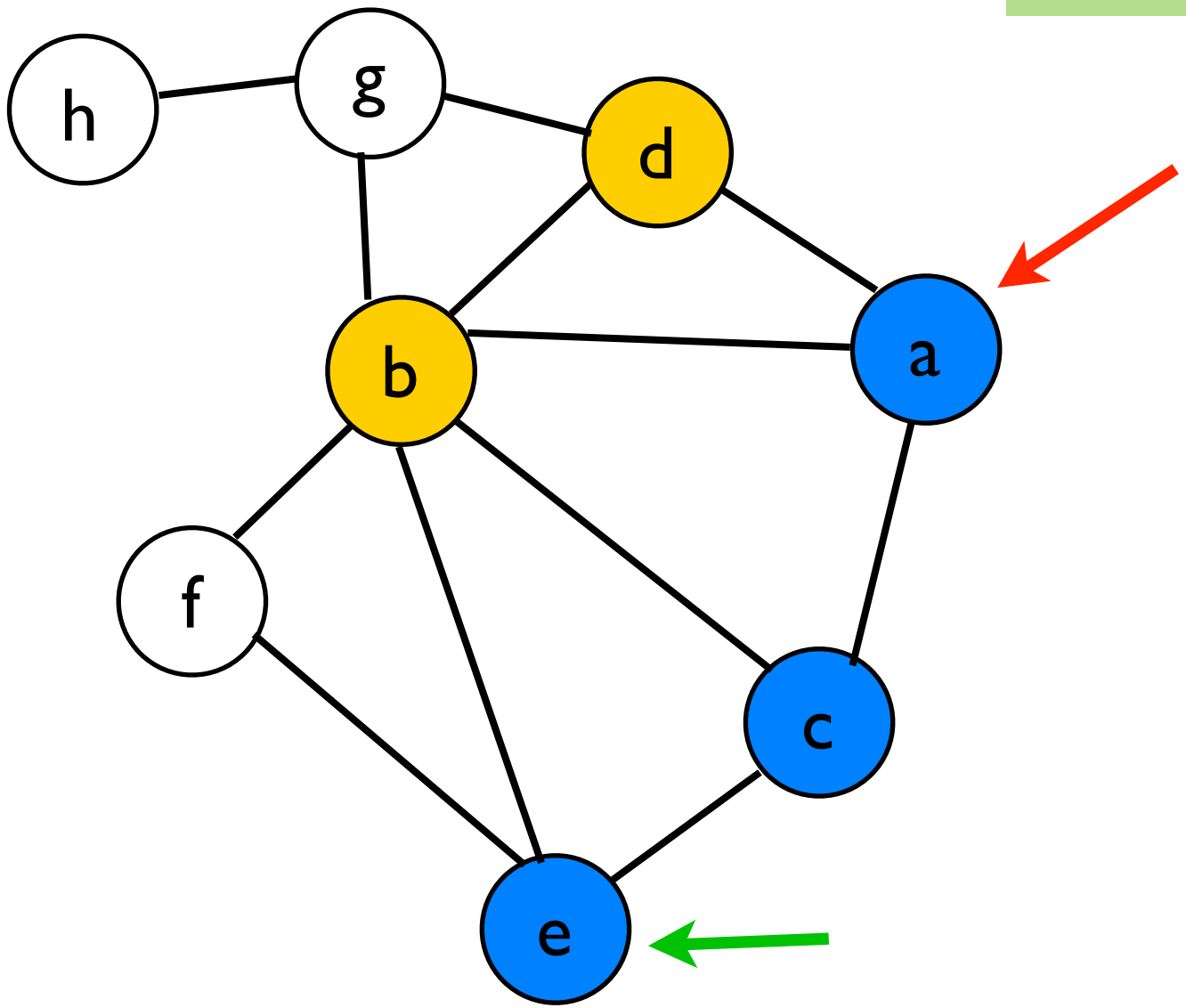
DFS



ordre: a c

Pile: e b d

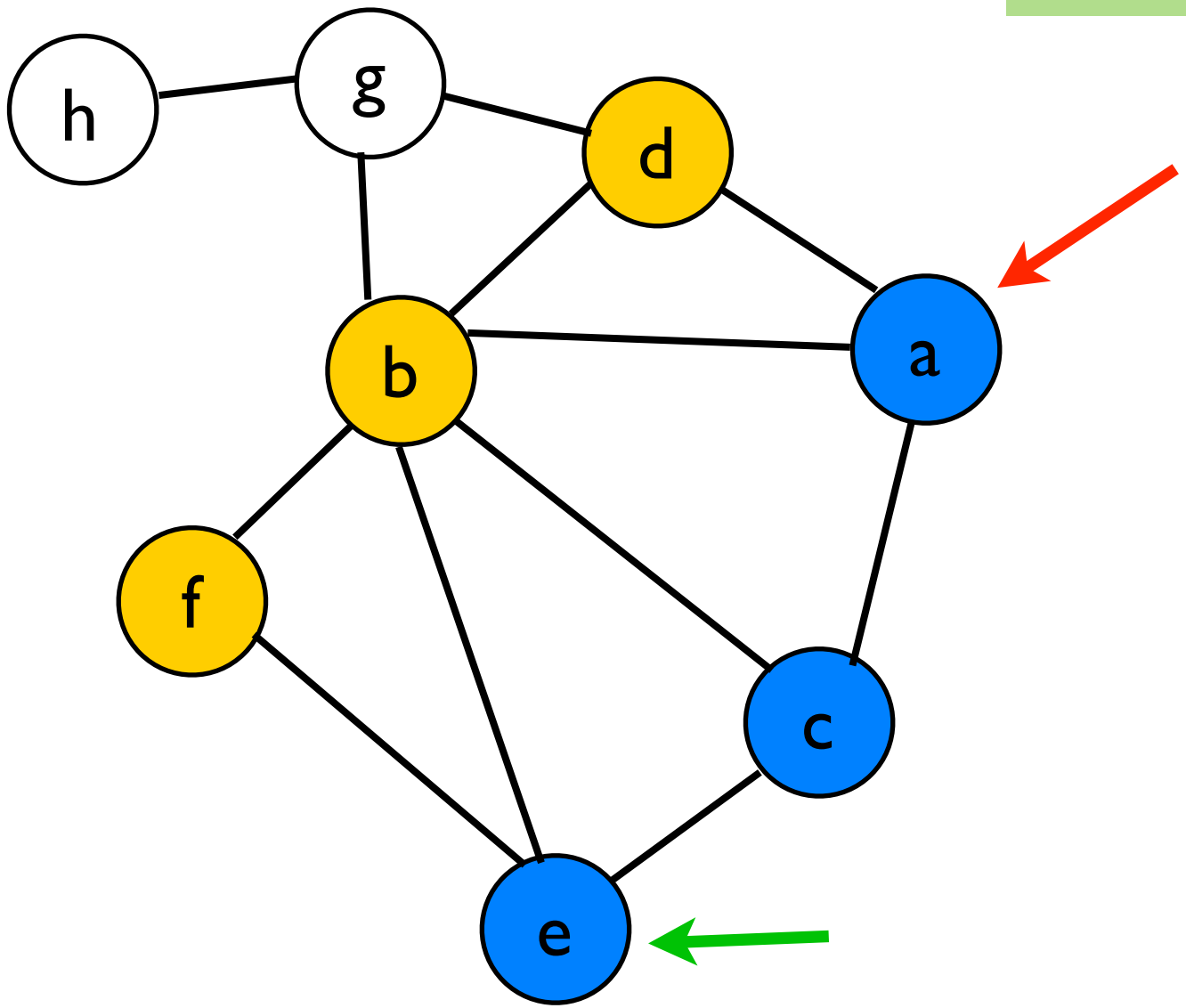
DFS



ordre: a c e

Pile: b d

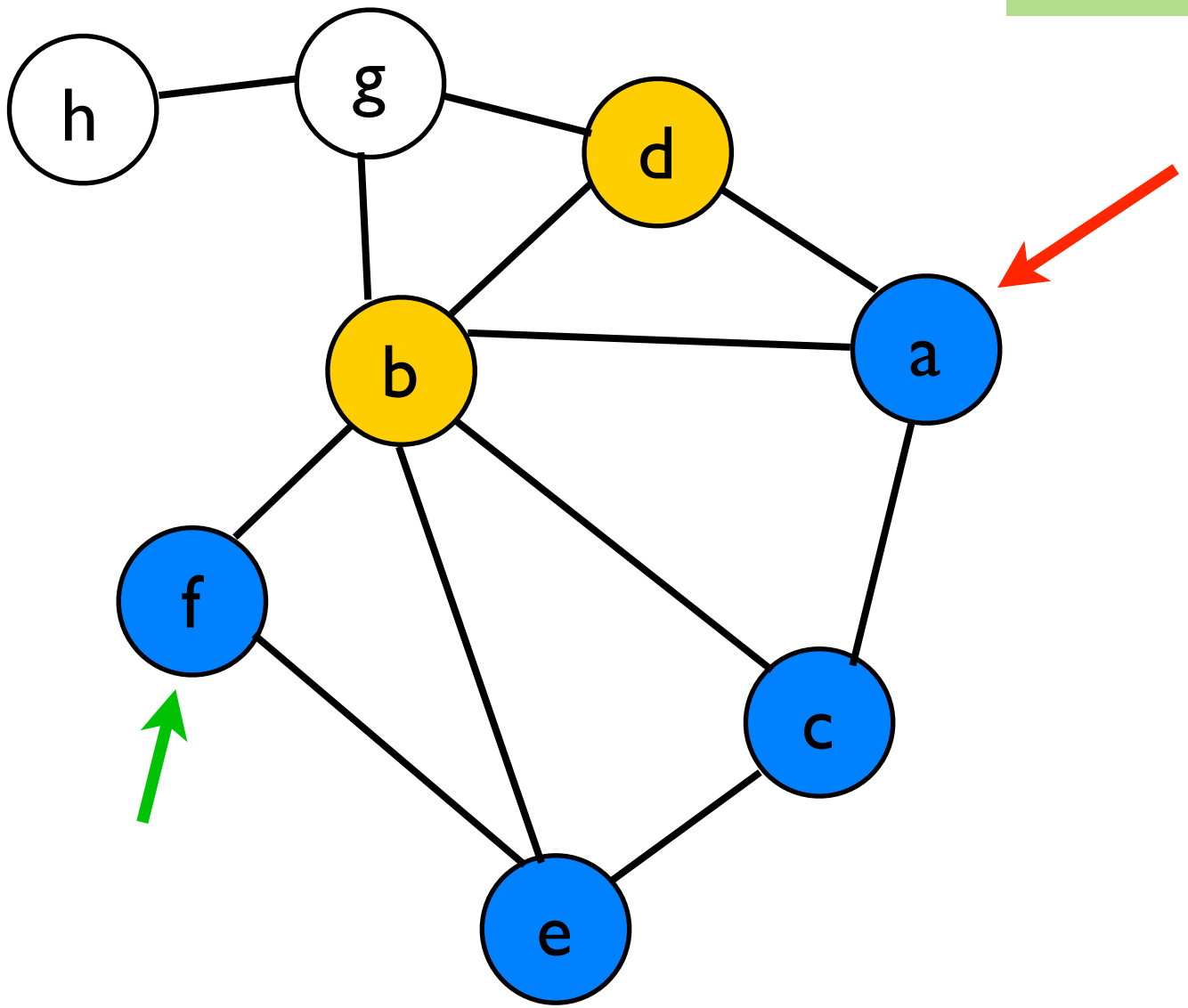
DFS



ordre: a c e

Pile: f b d

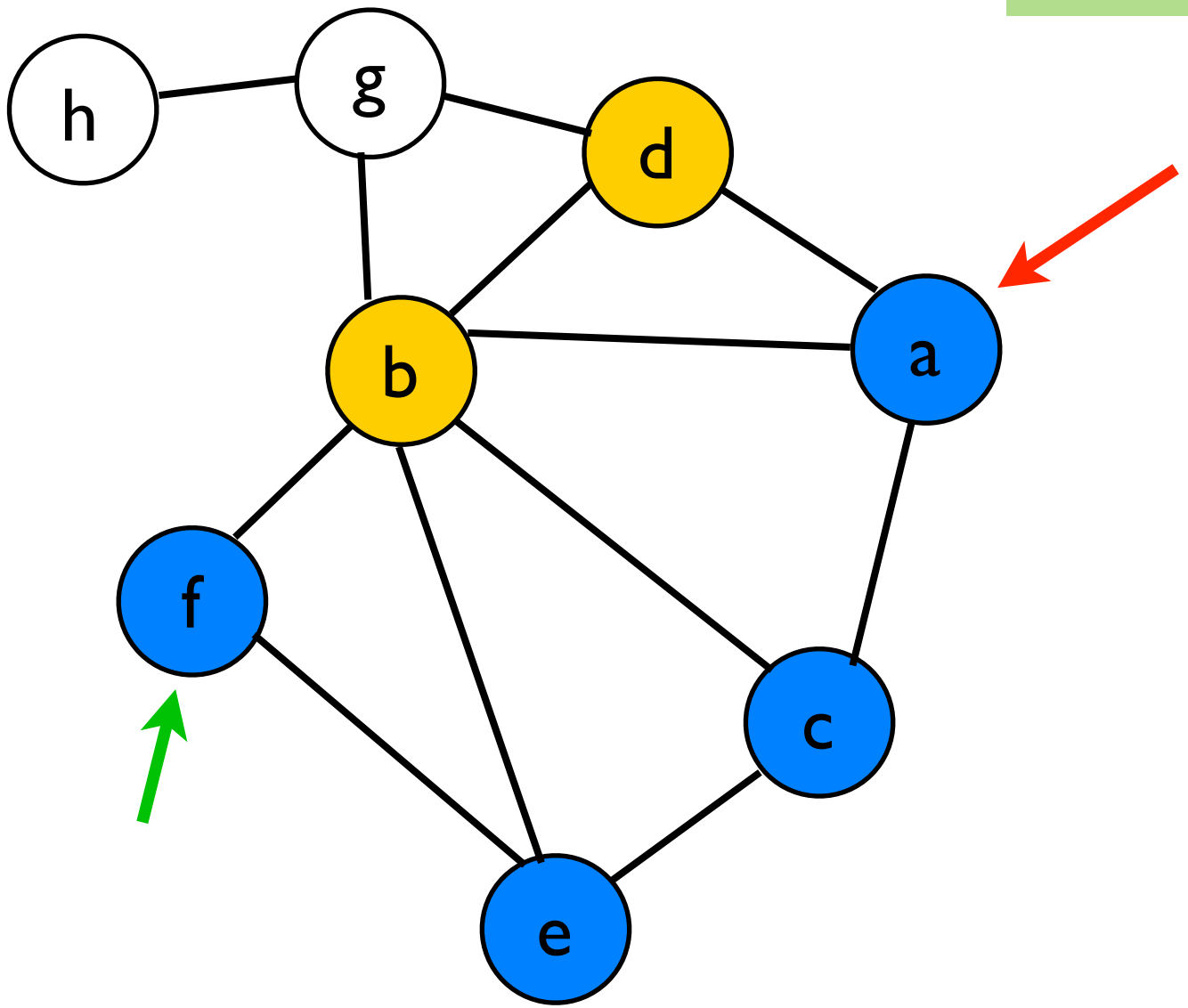
DFS



ordre: a c e f

Pile: b d

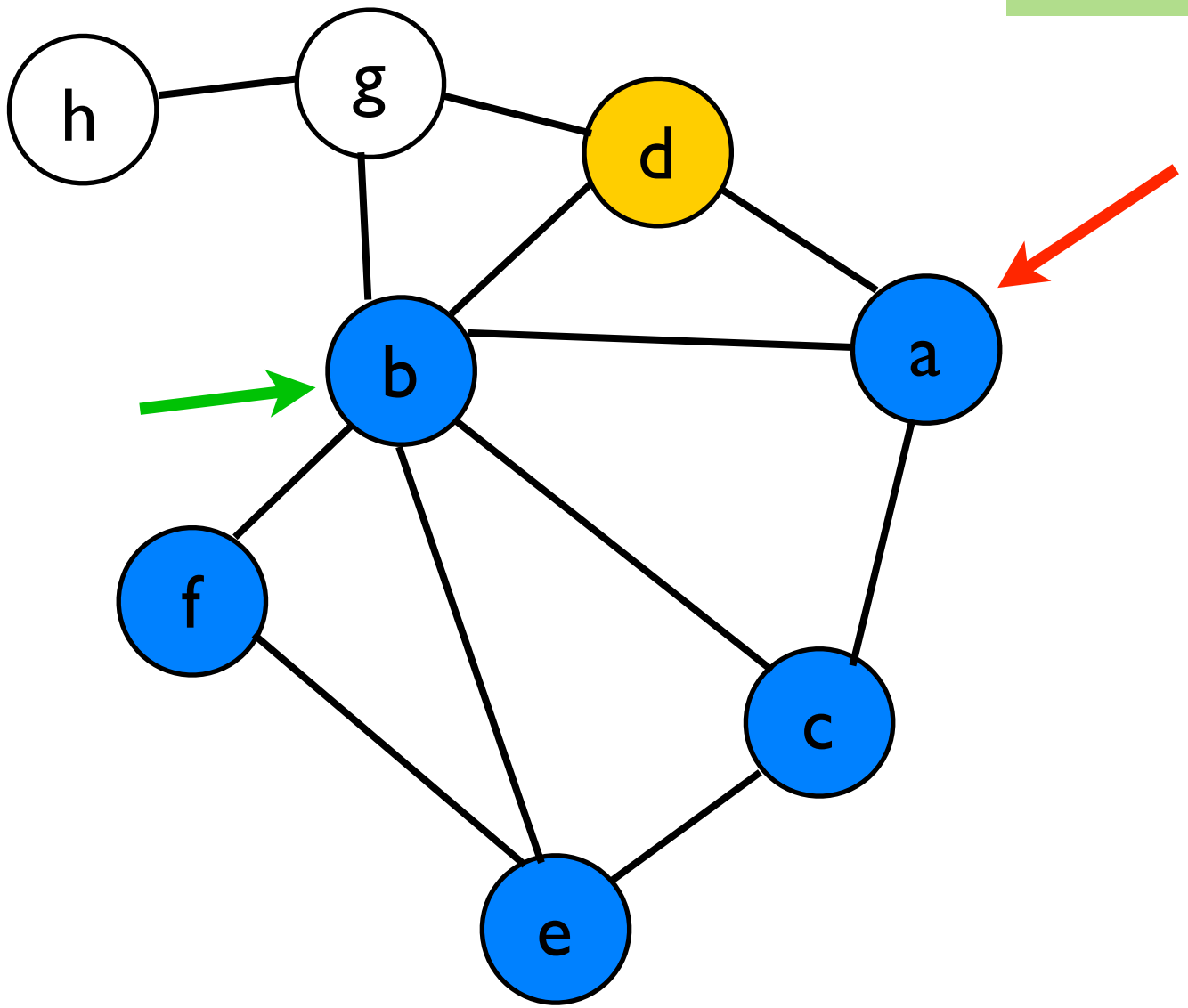
DFS



ordre: a c e f

Pile: b d

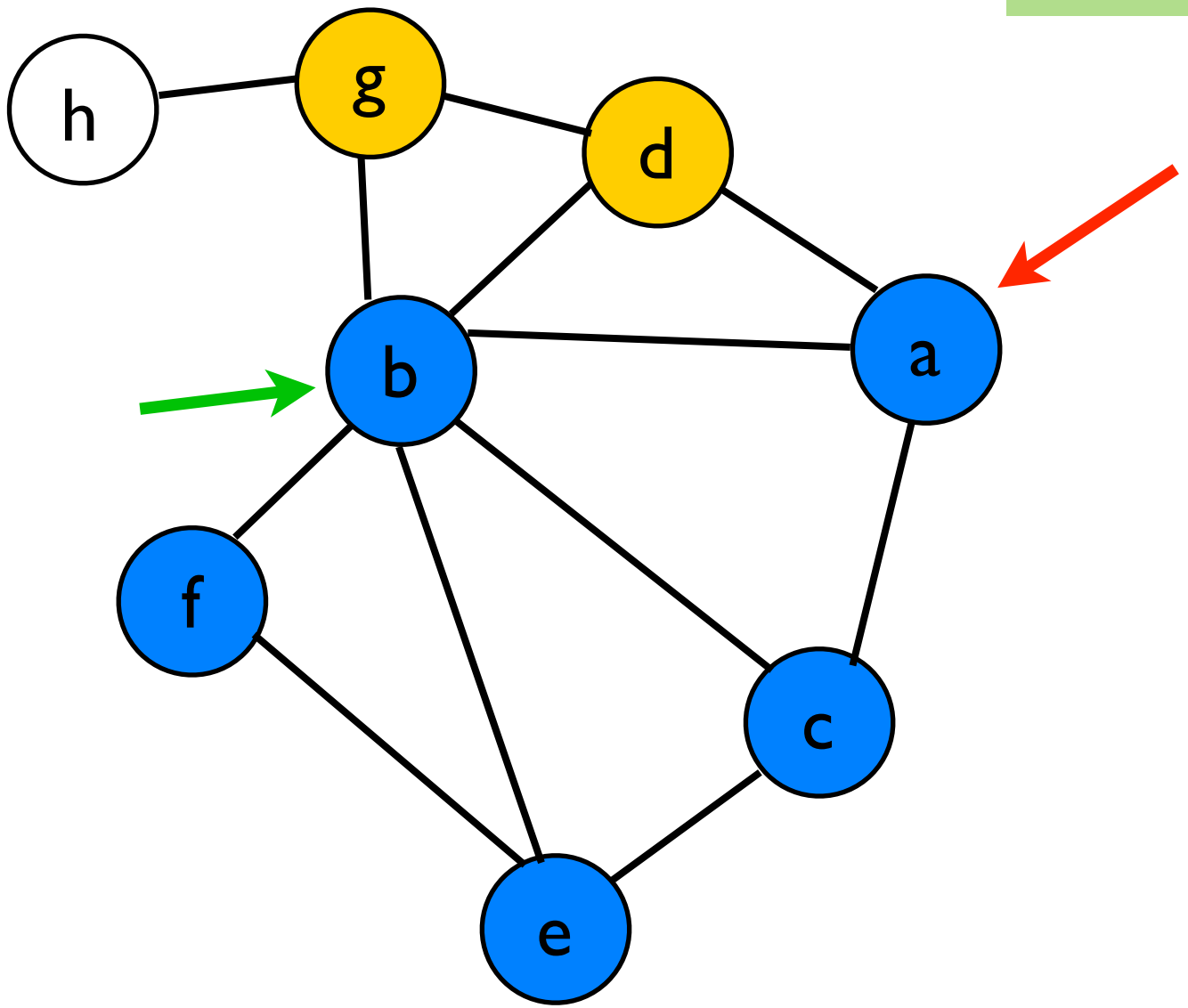
DFS



ordre: a c e f b

Pile: d

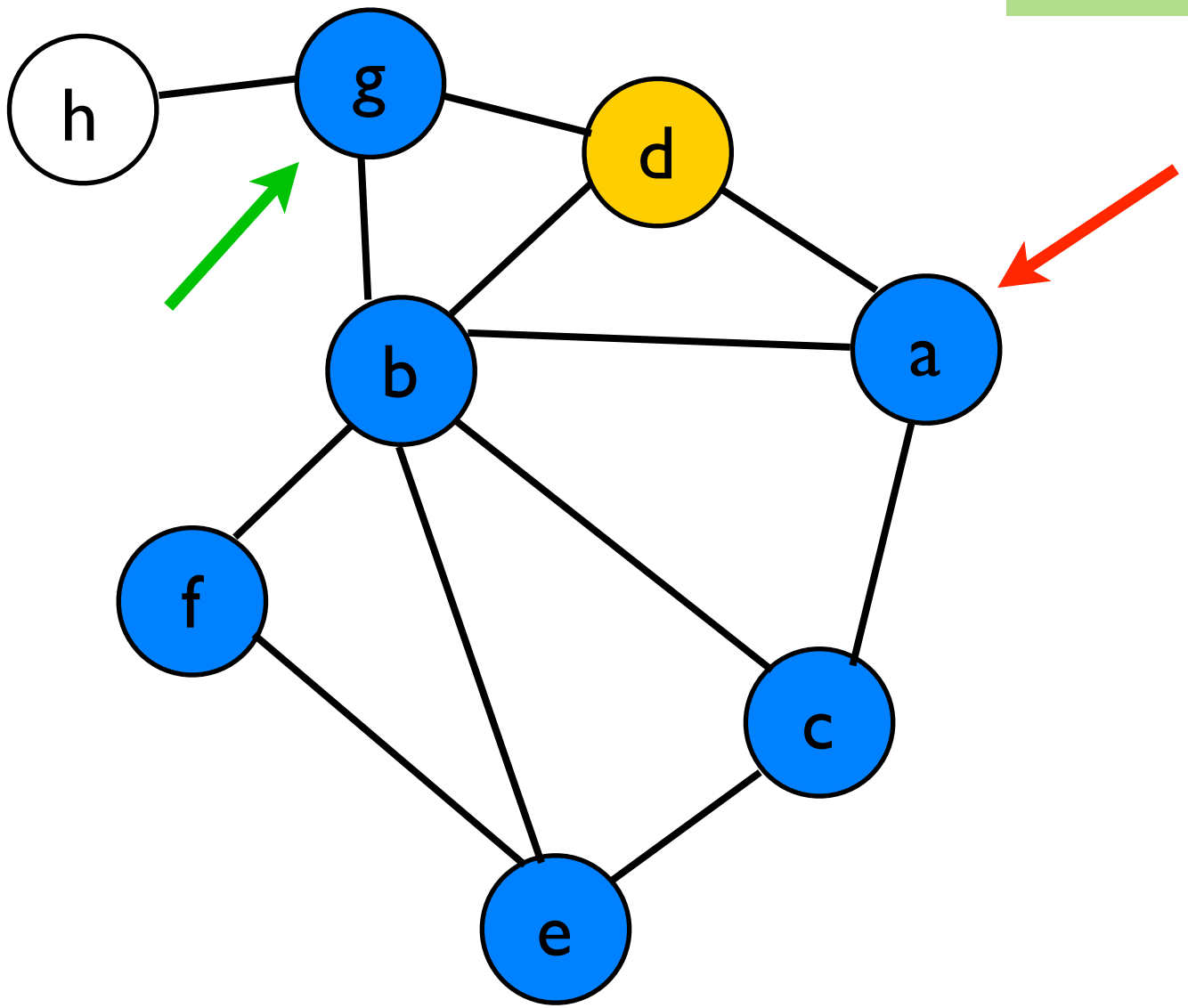
DFS



ordre: a c e f b

Pile: g d

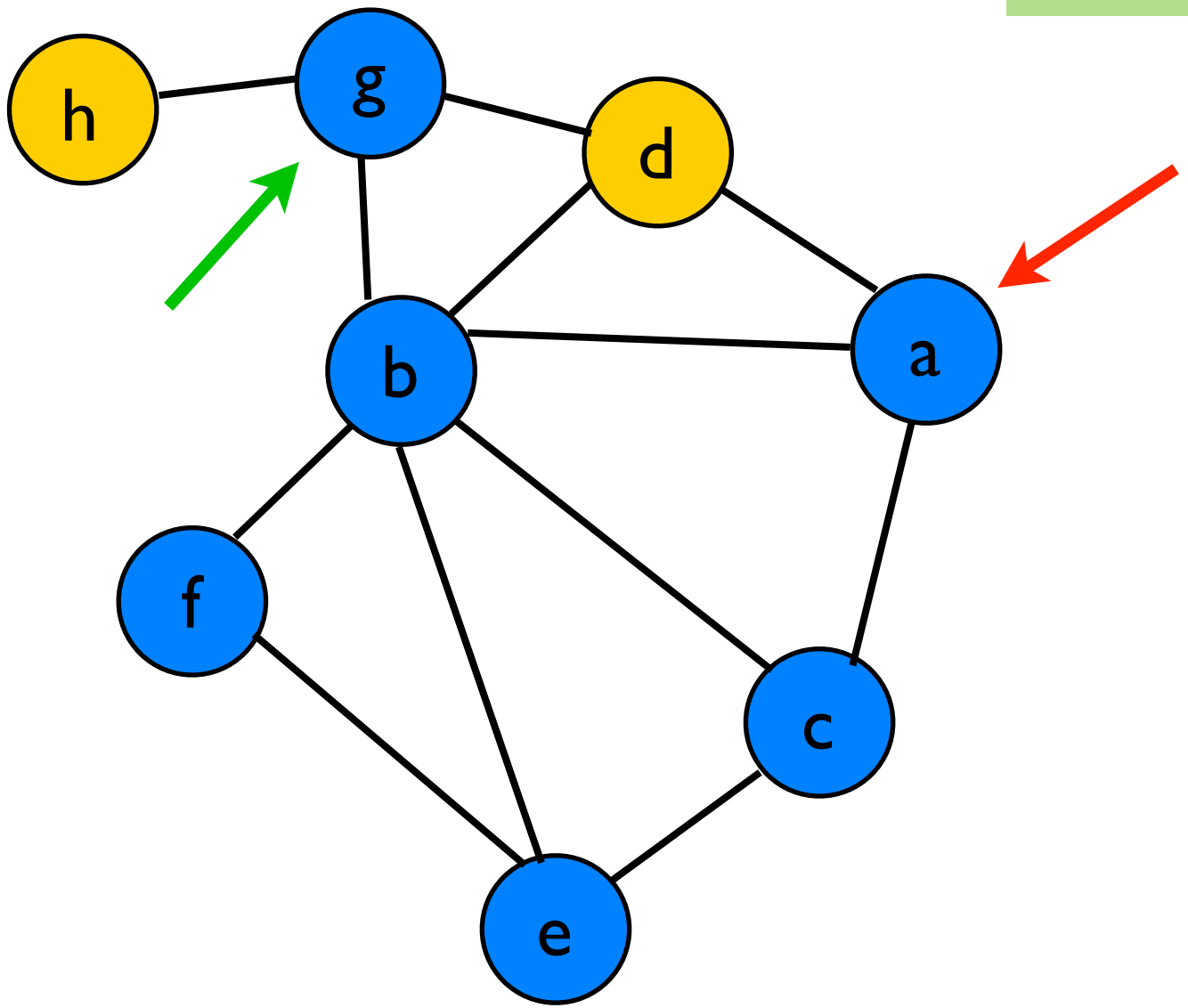
DFS



ordre: a c e f b g

Pile: d

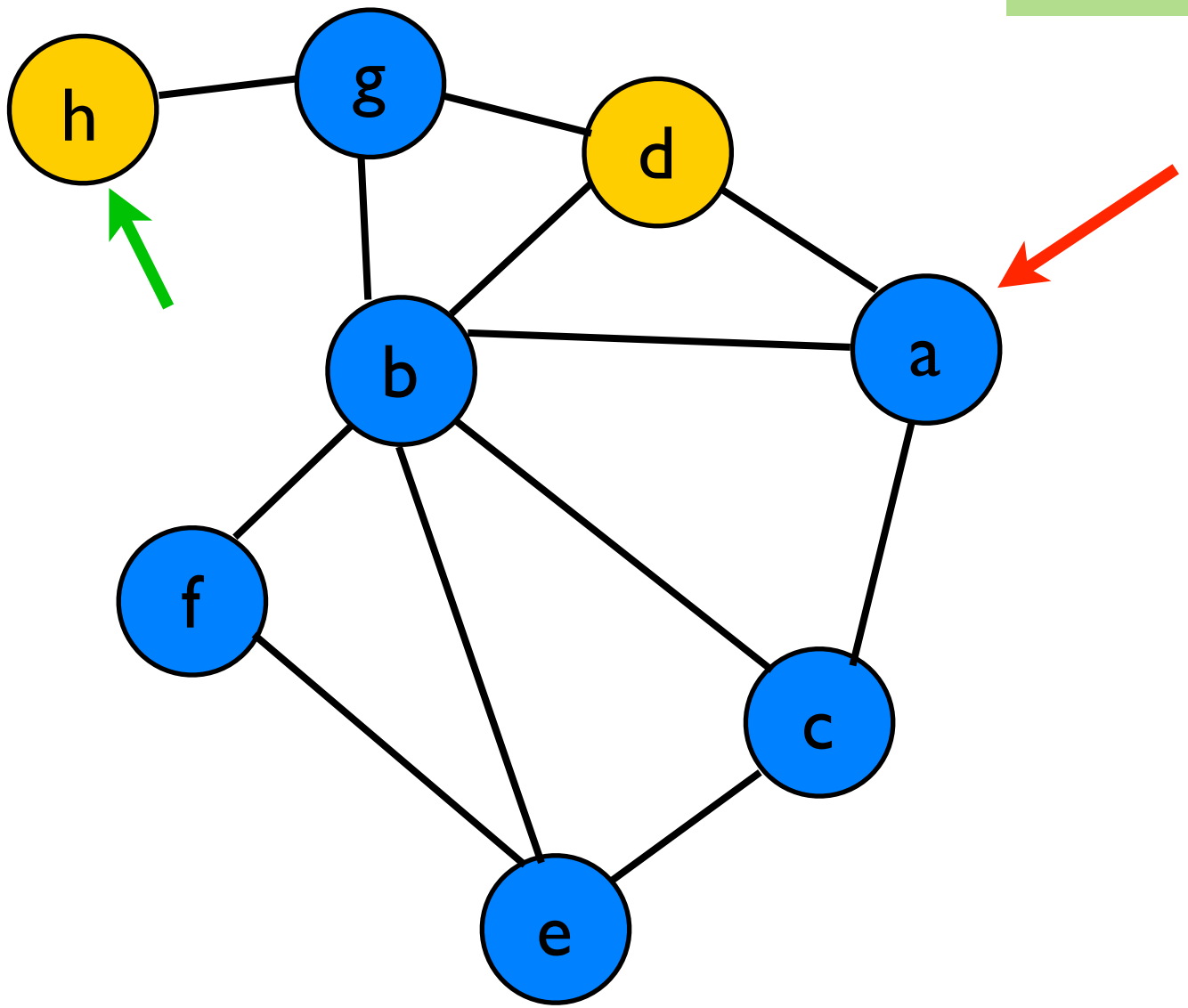
DFS



ordre: a c e f b g

Pile: h d

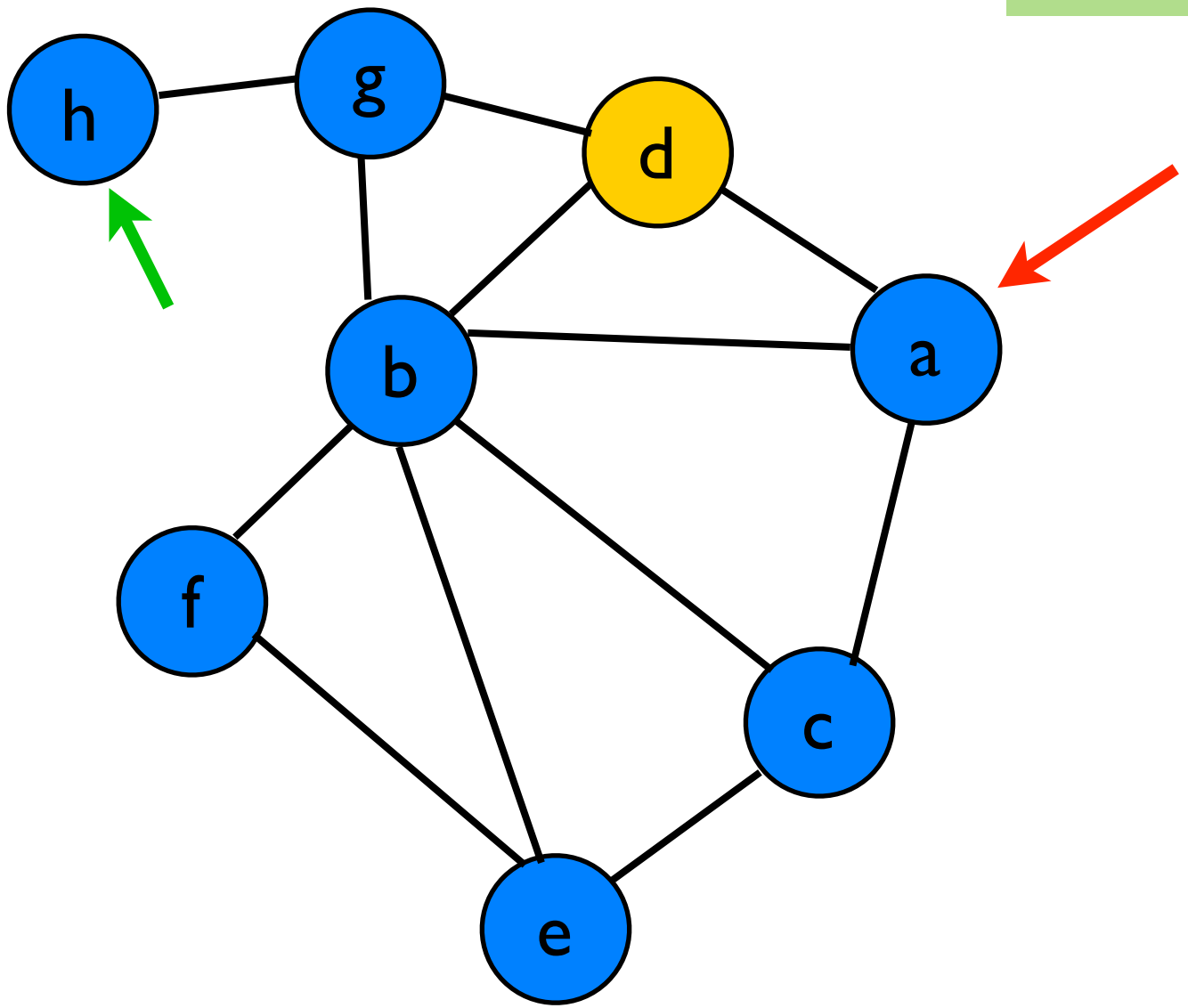
DFS



ordre: a c e f b g

Pile: h d

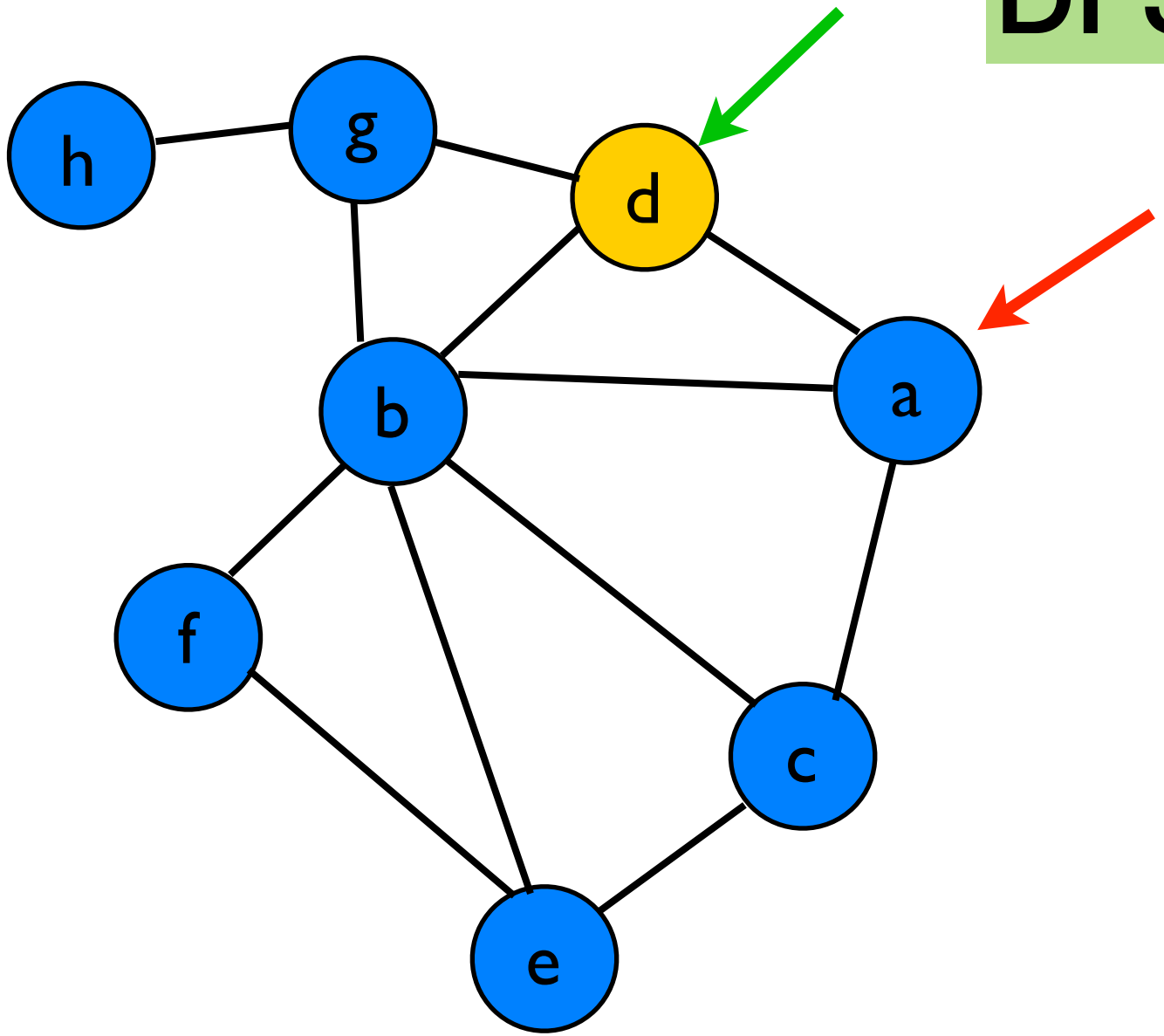
DFS



ordre: a c e f b g h

Pile: d

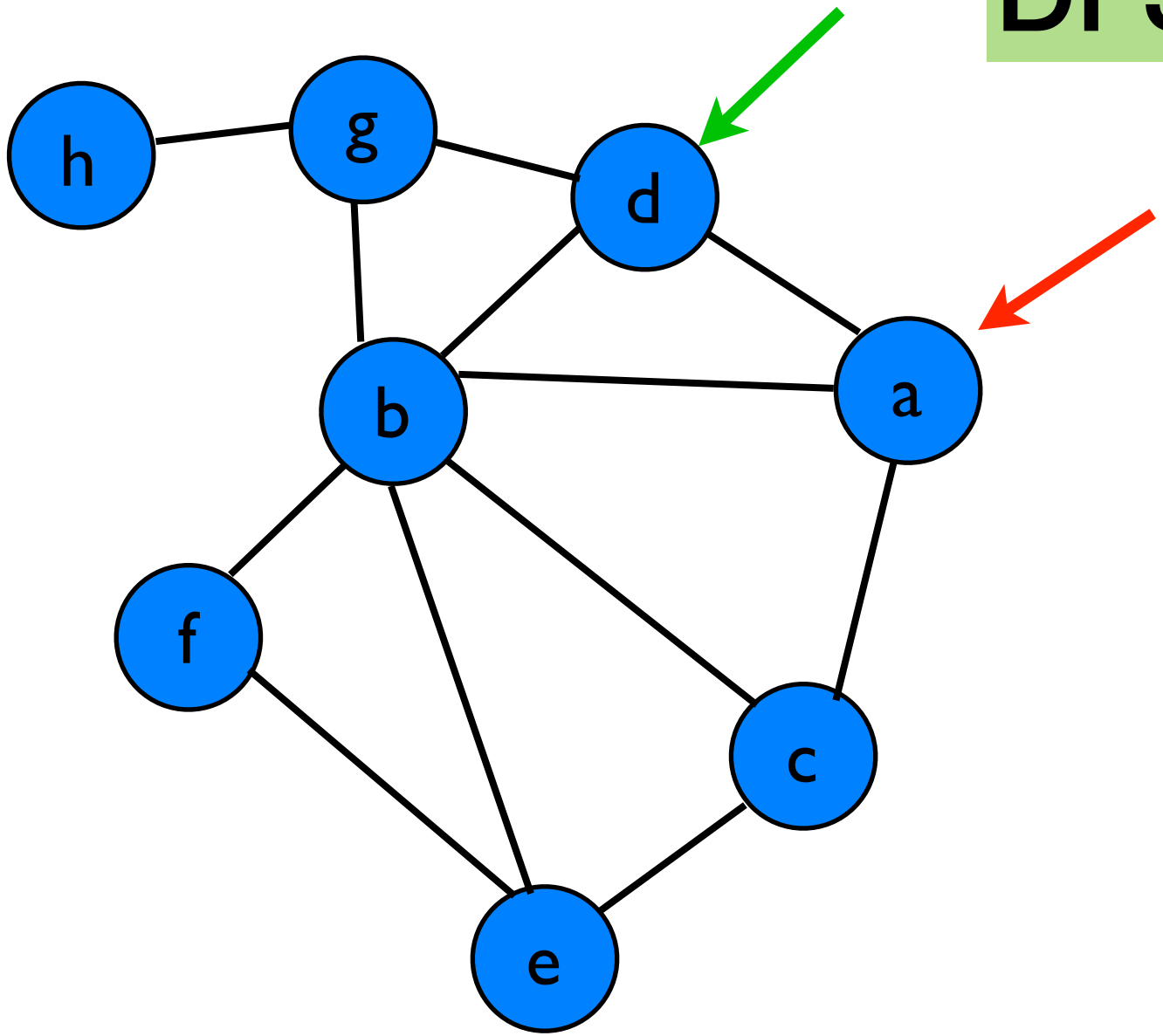
DFS



ordre: a c e f b g h

Pile: d

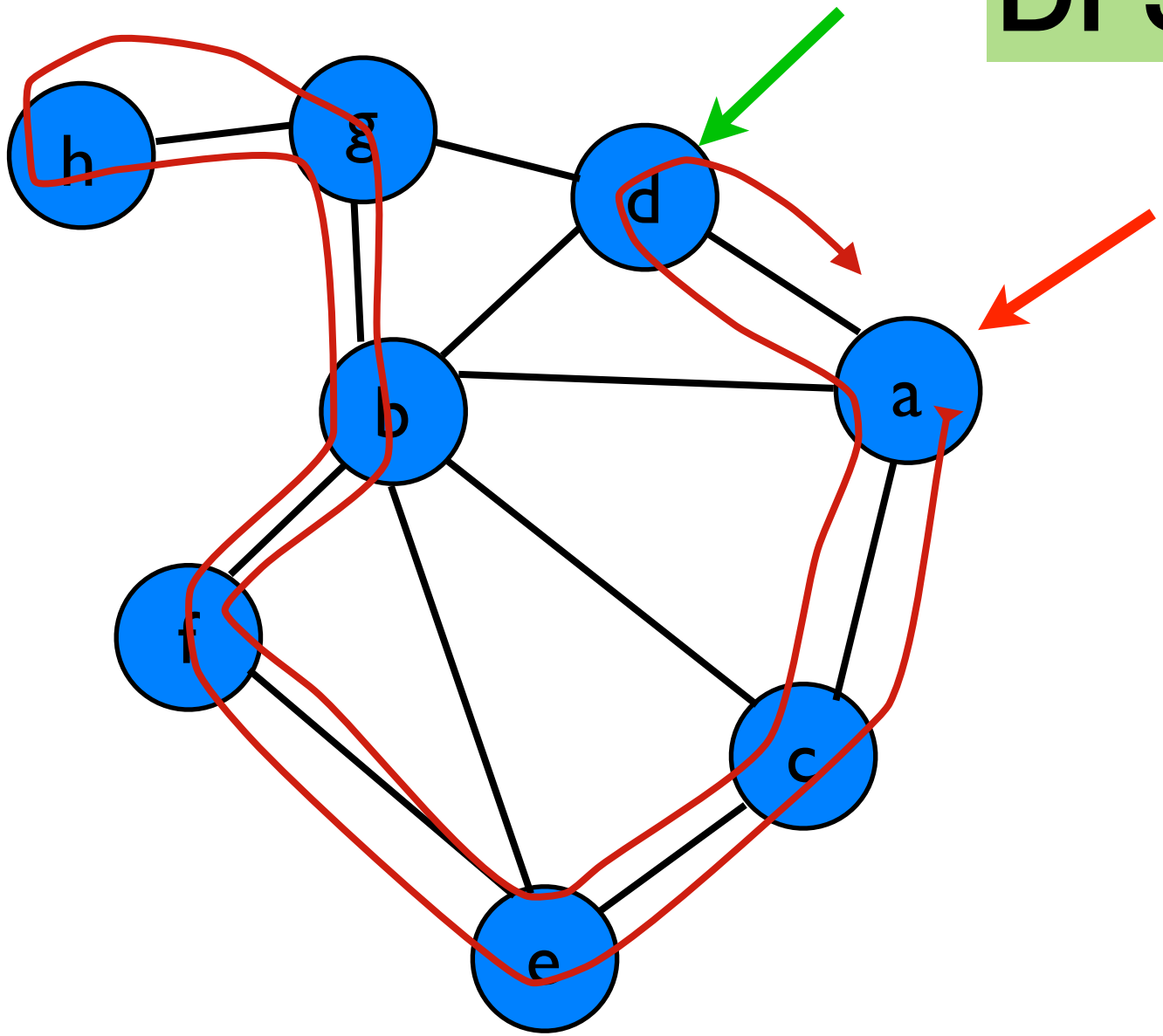
DFS



ordre: a c e f b g h d

Pile:

DFS



ordre: a c e f b g h d

Pile: FINI

```
fonction dfs(G, s)
0. pourtout sommet t faire
        etat[t] <- inexploré;
1. dfsRec(G, s, etat);

fonction dfsRec(G, s, etat)
si etat[s] == inexploré alors
    etat[s] <- encours;
    pourtout t voisin de s
        dfsRec(G, t, etat);
    etat[s] <- exploré;
```

Parcours :

- Permet de trouver les sommets accessibles à partir de a
- Permet de les ordonner, les marquer
- Permet de construire la composante connexe de a

La suite

Cet après-midi: utiliser Union-Find

La semaine prochaine: parcours plus sophistiqués,
trouver les bons chemins