Graphes

Laurent Lemarchand
lemarch@univ-brest.fr
http://www.lisyc.univ-brest.fr/pages_perso/lemarch/Cours

Relations

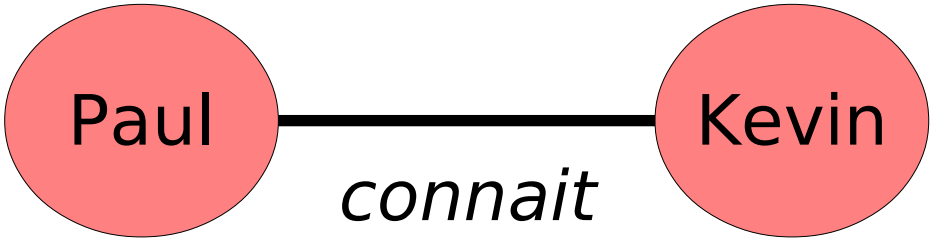
- Relation entre 2 entités
  - *Plus petit que*
  - *Fait avant*
  - *Exécuté par*
  - ....
- Graphiquement :

relations symétriques ou non

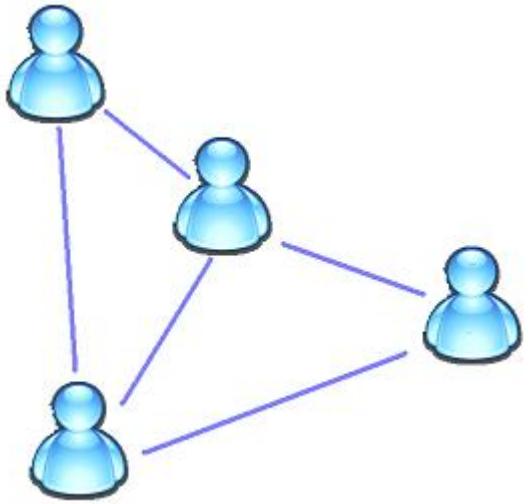
Graphe !

Relations symétriques

- Relation entre 2 entités
  - *Se connaissent*



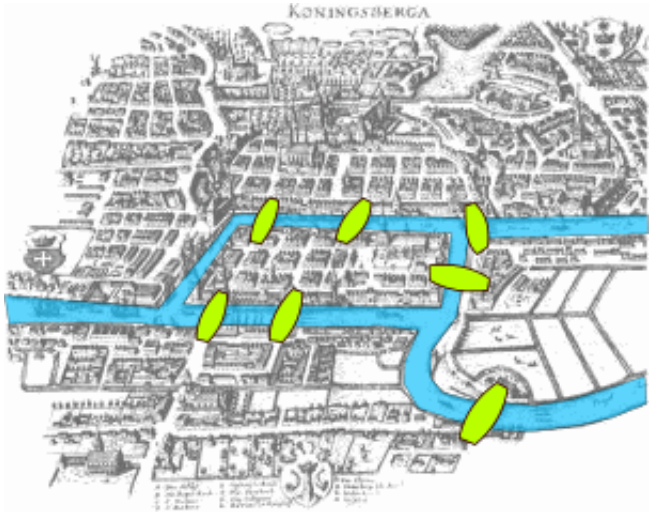
- Généralisation : réseau social

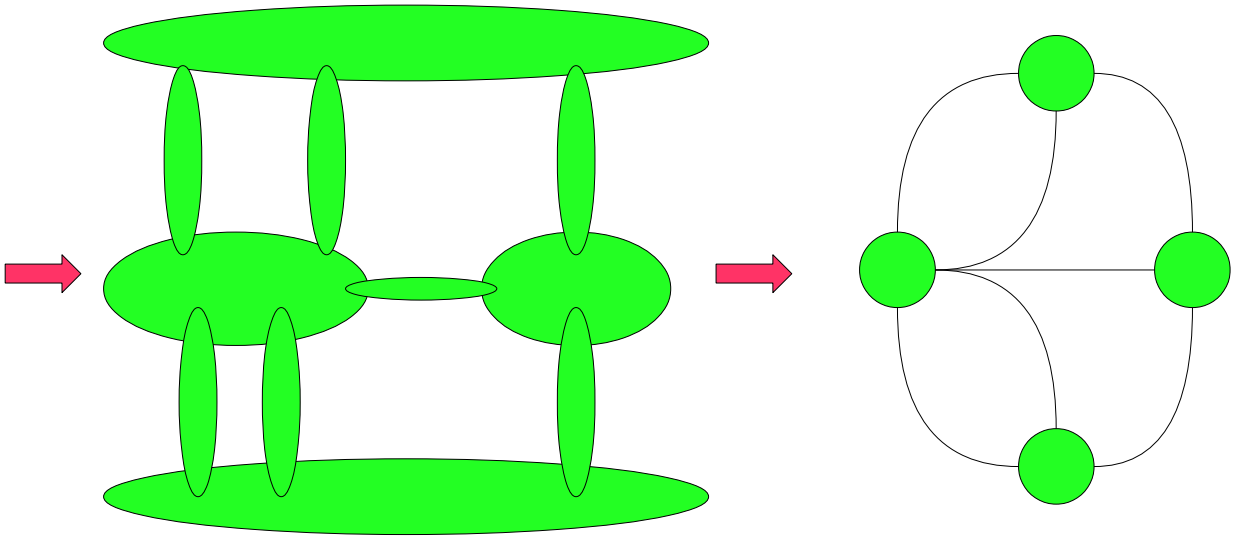


Degré moyen
=
nombre de Dunbar
=
125

Historiquement (1)
Ponts de Konigsberg (Euler, 1736)

- Ville sur 2 iles, reliées par un pont, et reliées aux rives.
- Passer sur tous les ponts 1 et 1 seule fois et revenir à son point de départ

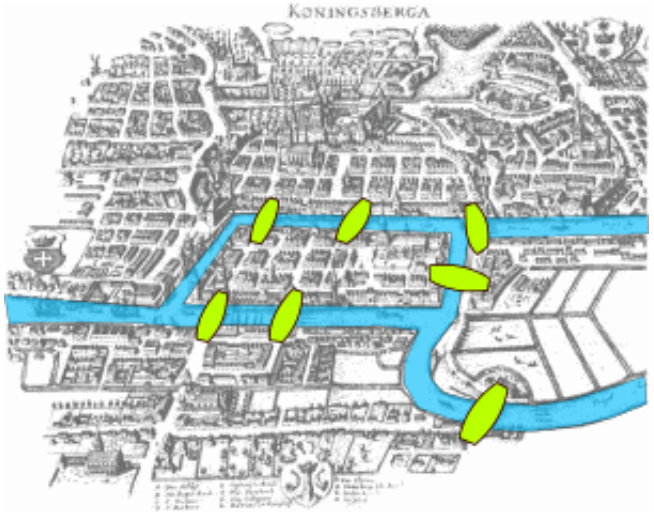


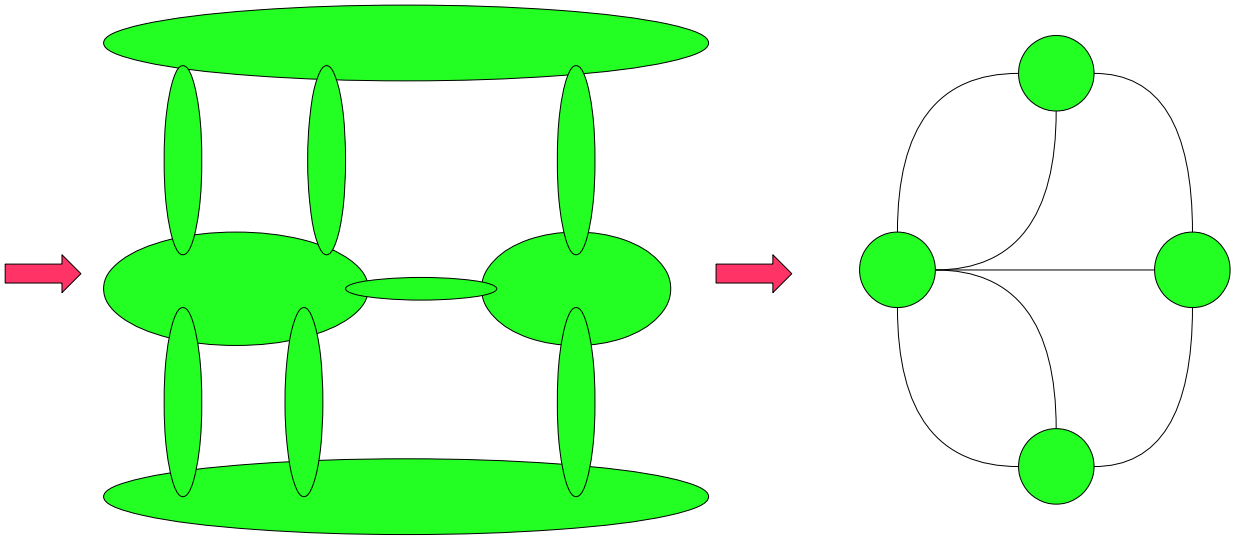


http://www.wikipedia.org

Historiquement (2)
Ponts de Konigsberg (Euler, 1736)

- Revient à trouver un *cycle eulérien* : chaine passant une et une seule fois par toutes les arêtes
- Impossible si sommets de degré impair





http://www.wikipedia.org

Parcours eulériens et hamiltoniens

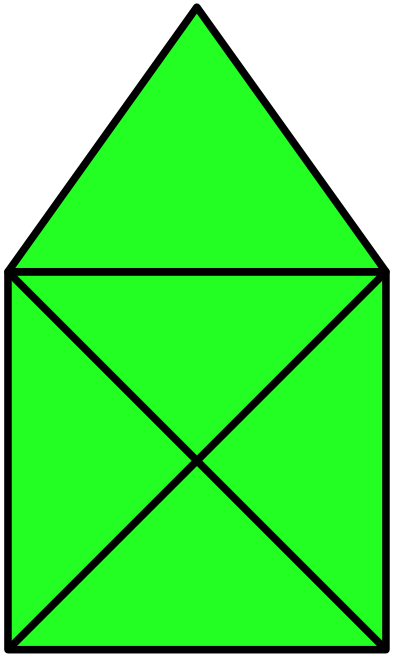
- Recherche de cycles
  - Eulériens : passant une et une seule fois sur chaque arête
  - Hamiltoniens : passant une et une seule fois par chaque sommet
- Problèmes associés
  - Recherche de cycle eulérien *de longueur minimale*
  - Postier Chinois : cycle de longueur minimale passant *au moins une fois sur chaque arête*
  - Voyageur de Commerce : *cycle hamiltonien de longueur minimale*

Parcours eulérien

- Théorème d'Euler

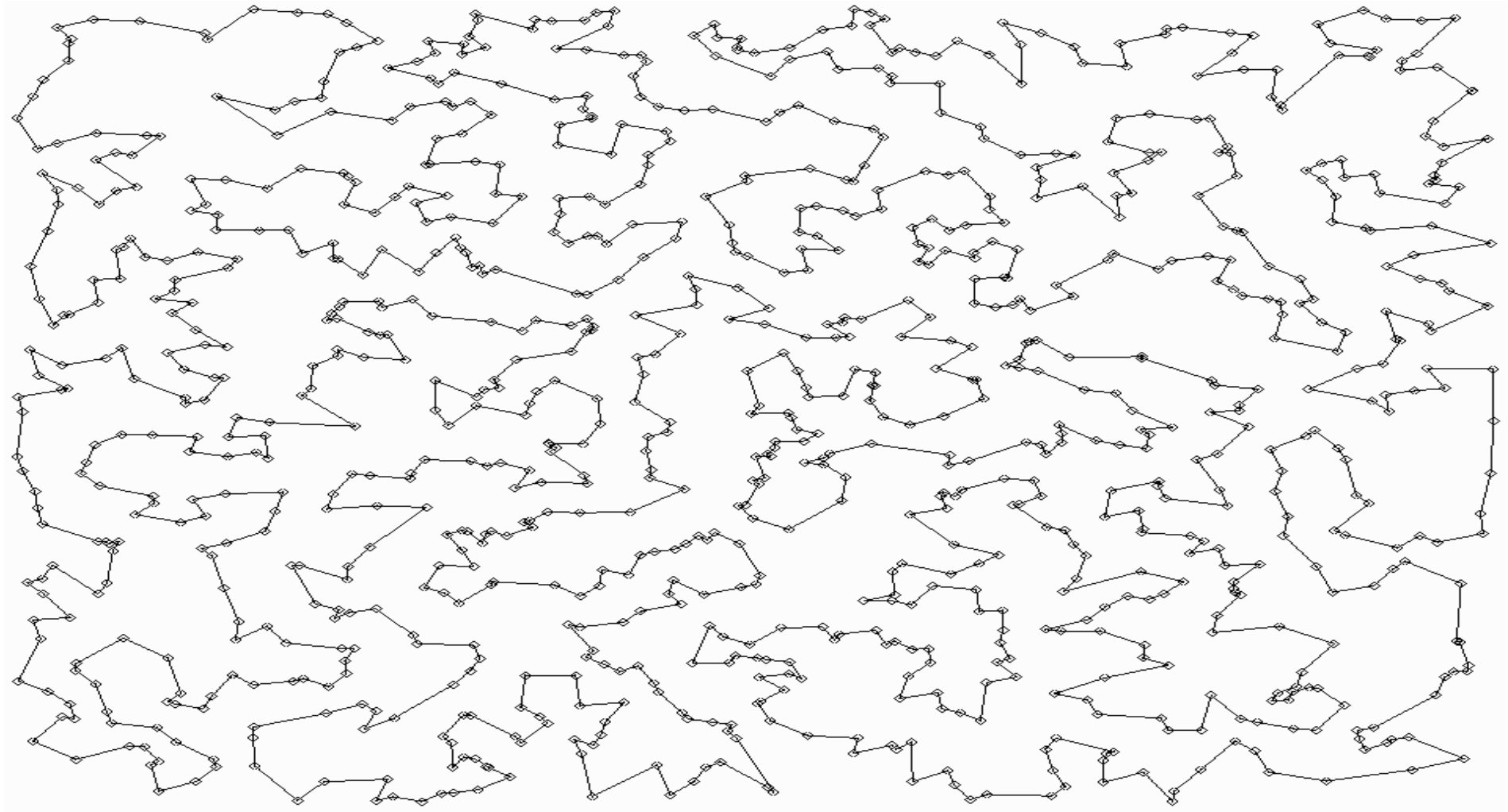
> Un multigraphe admet un parcours eulérien *ssi* il est connexe et possède 0 ou 2 sommets de degré impair

- Exemple
  - Dessiner l'enveloppe sans lever le crayon ni repasser plusieurs fois sur un trait
  - Postier chnois : on peut repasser sur un trait



Parcours hamiltonien

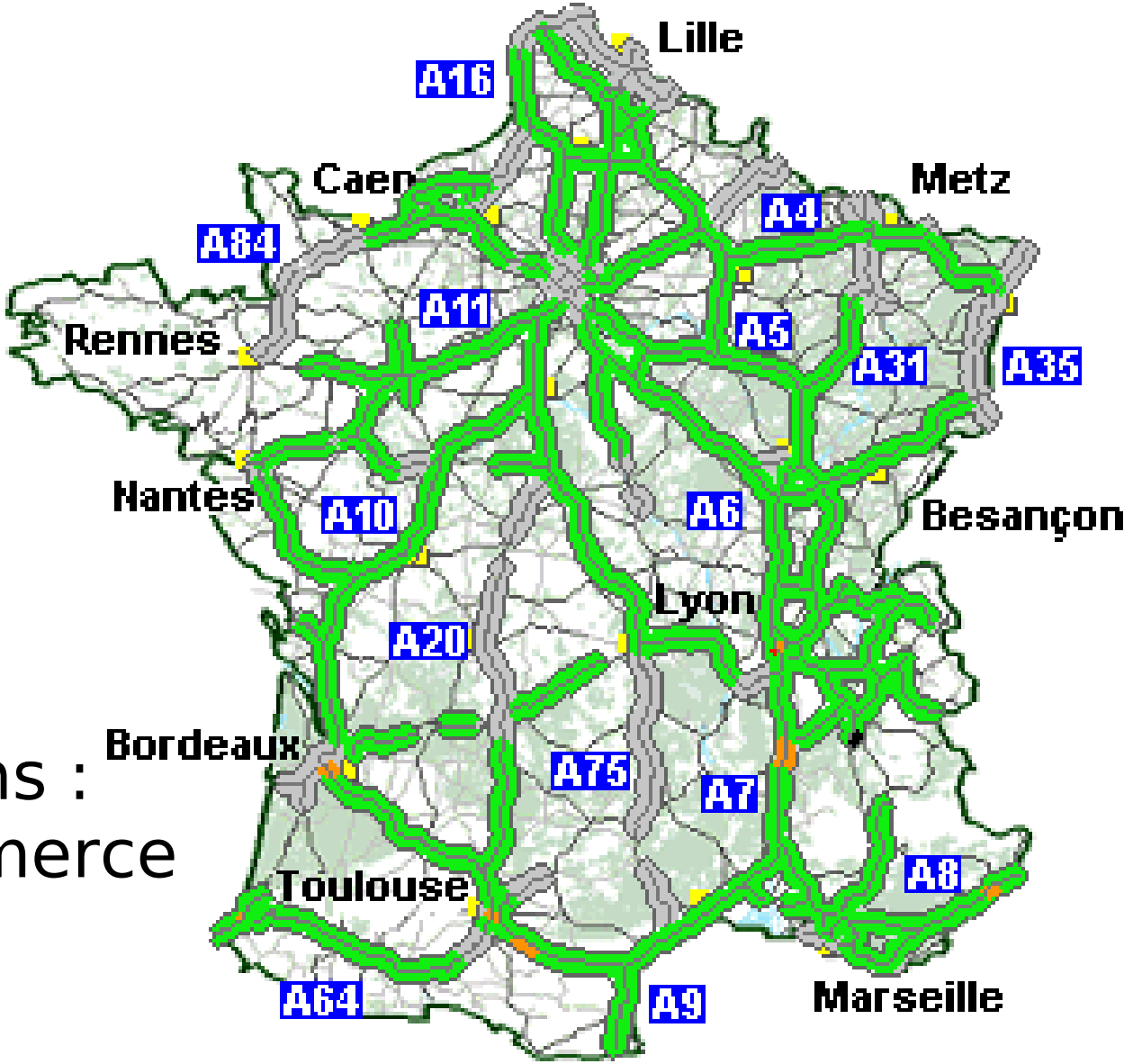
- Recherche de cycle hamiltionien de longueur minimale : le *Problème du Voyageur de Commerce*



http://www.cs.berkeley.edu/~bonachea

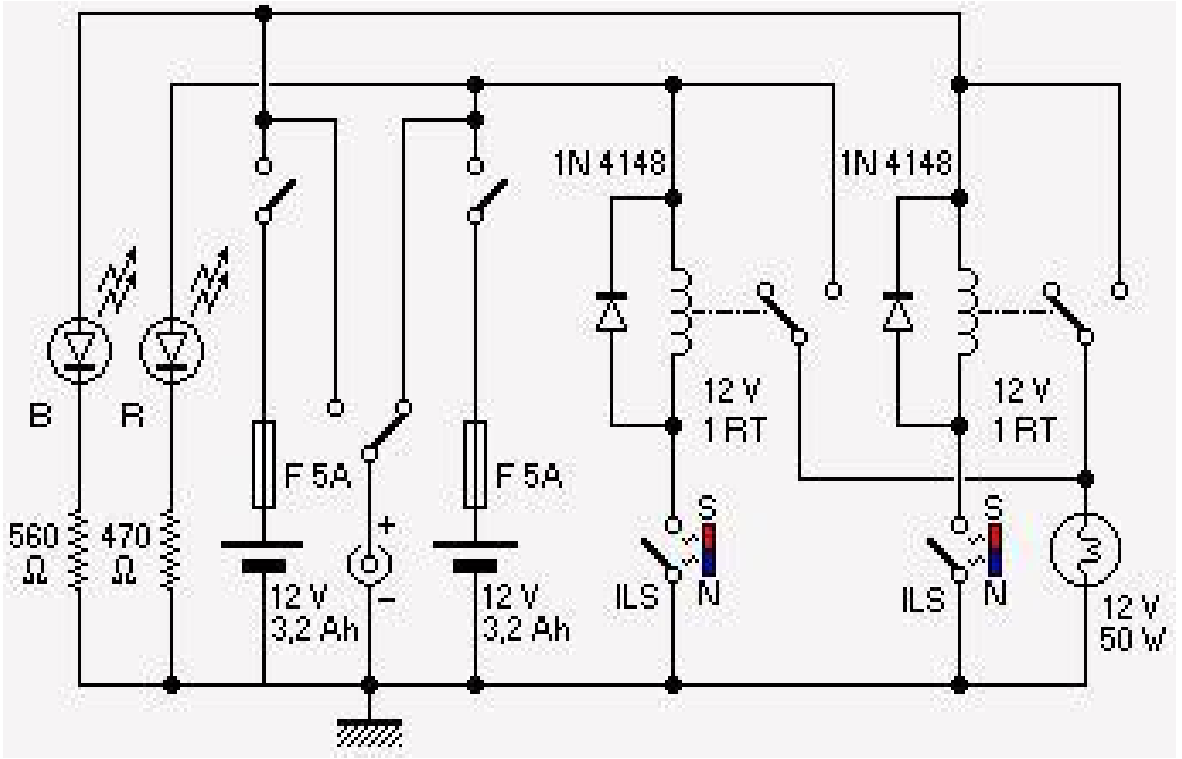
Quelques applications
Cartographie

- Trajets
  - *Par où ?*
  - *Quel coût ?*
- Routages
  - Géographie
  - Internet
- Cycles hamiltoniens : Voyageur de commerce
- Cycles eulériens : Postier chinois

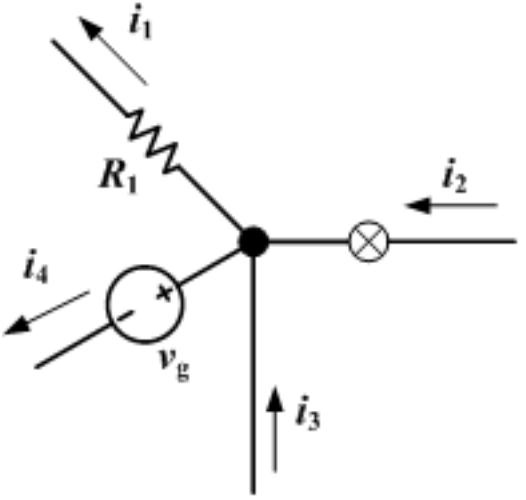


Quelques applications
Circuits électriques

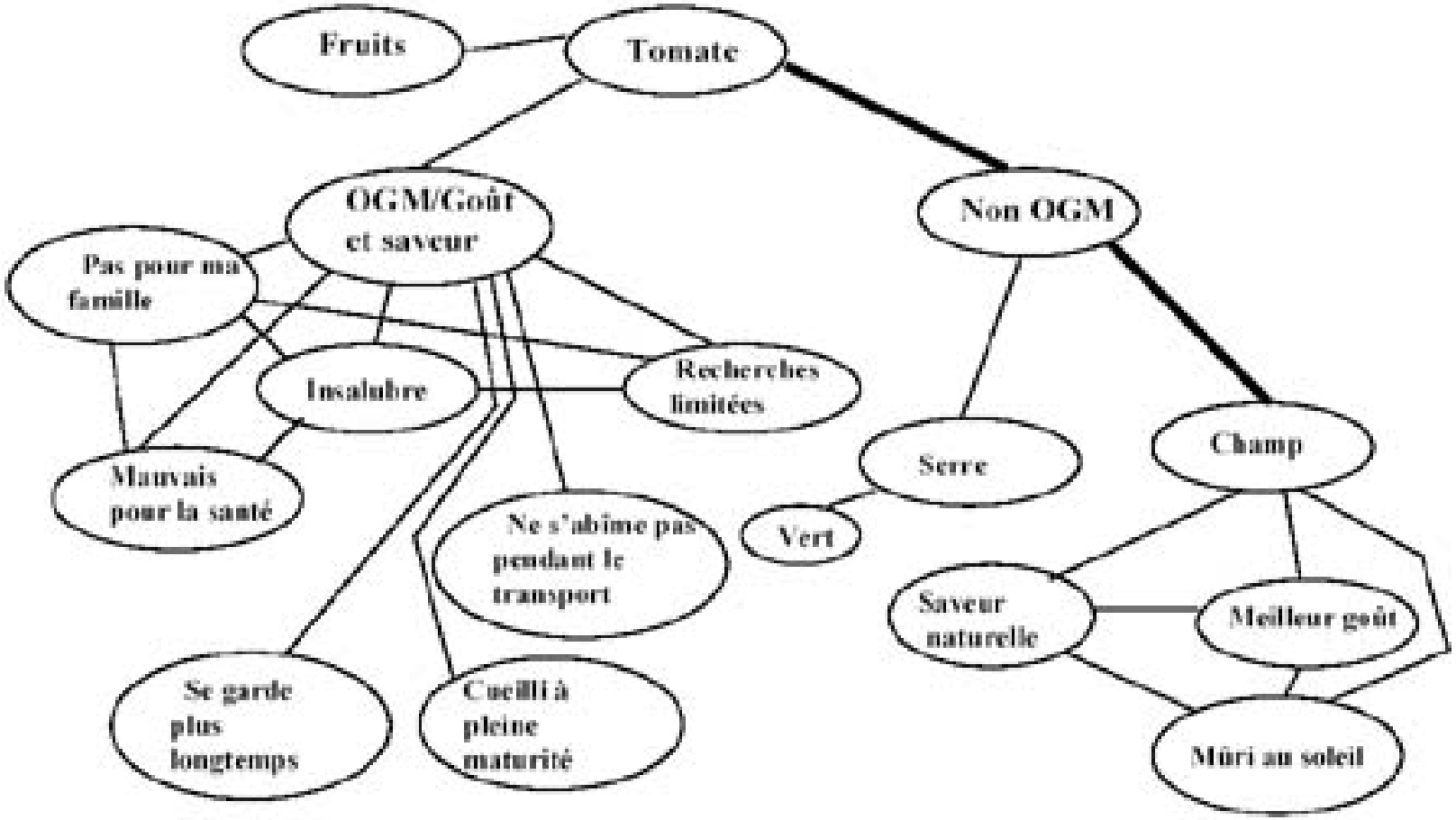
- Noeuds d'un réseau électrique



- Lois de kirchoff
  - *Conservation de l'énergie*
  - *Conservation de la charge*



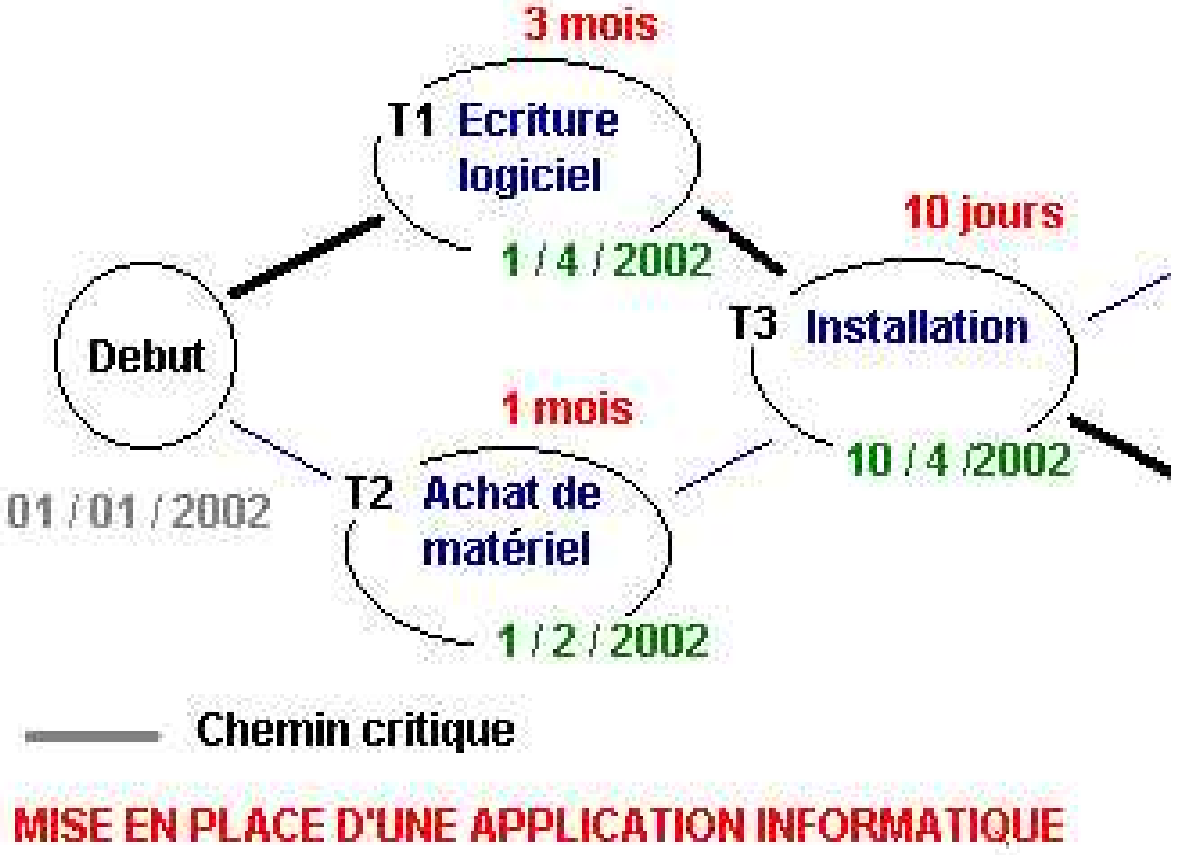
Quelques applications
Réseau sémantique



- La notion de *soleil* est elle reliée à celle de *tomate* ?
- Applications en recherche web

Quelques applications
Gestion de projets

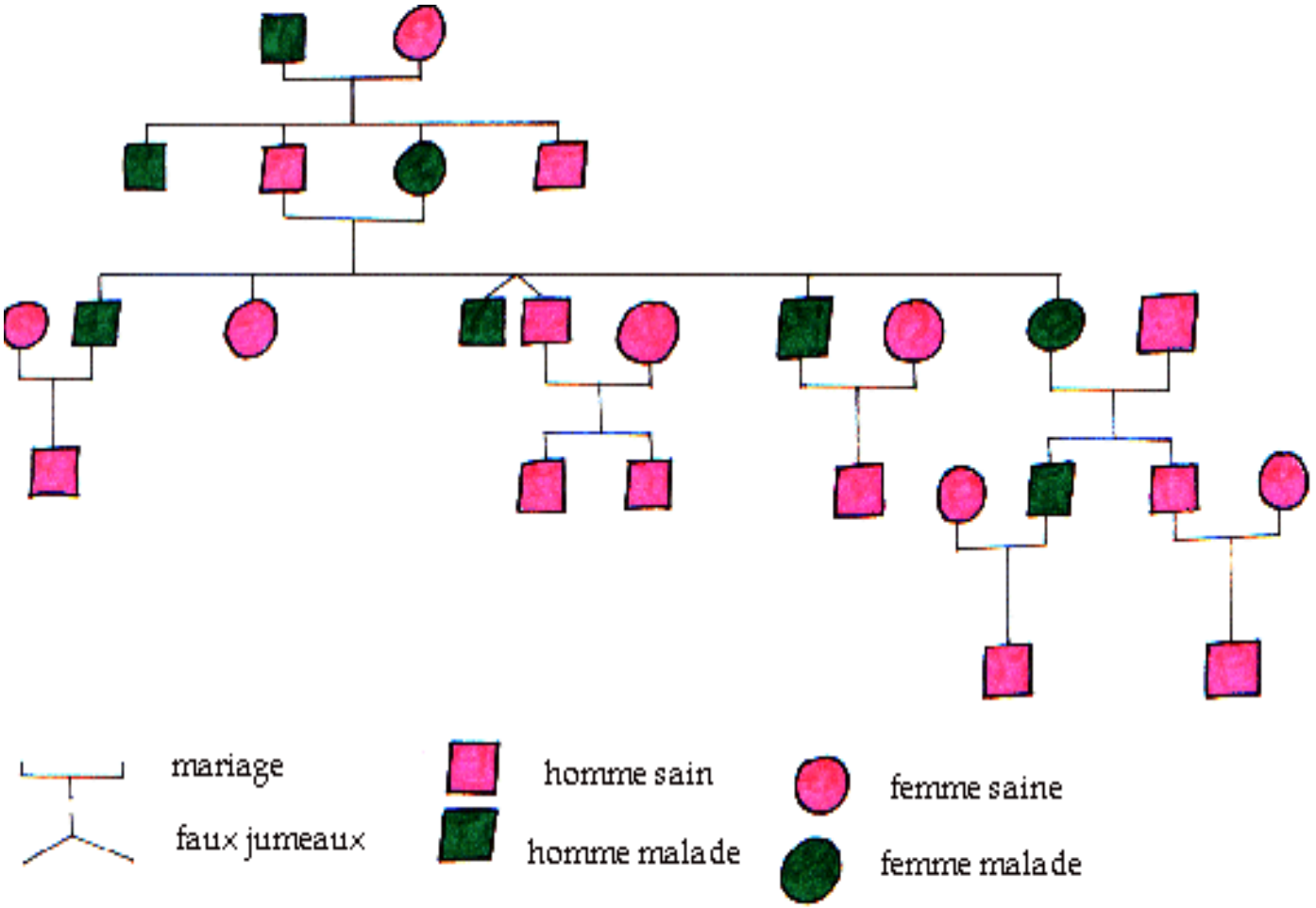
- Réaliser des tâches
  - *Dépendances*
  - *Durées*



- Temps global ?

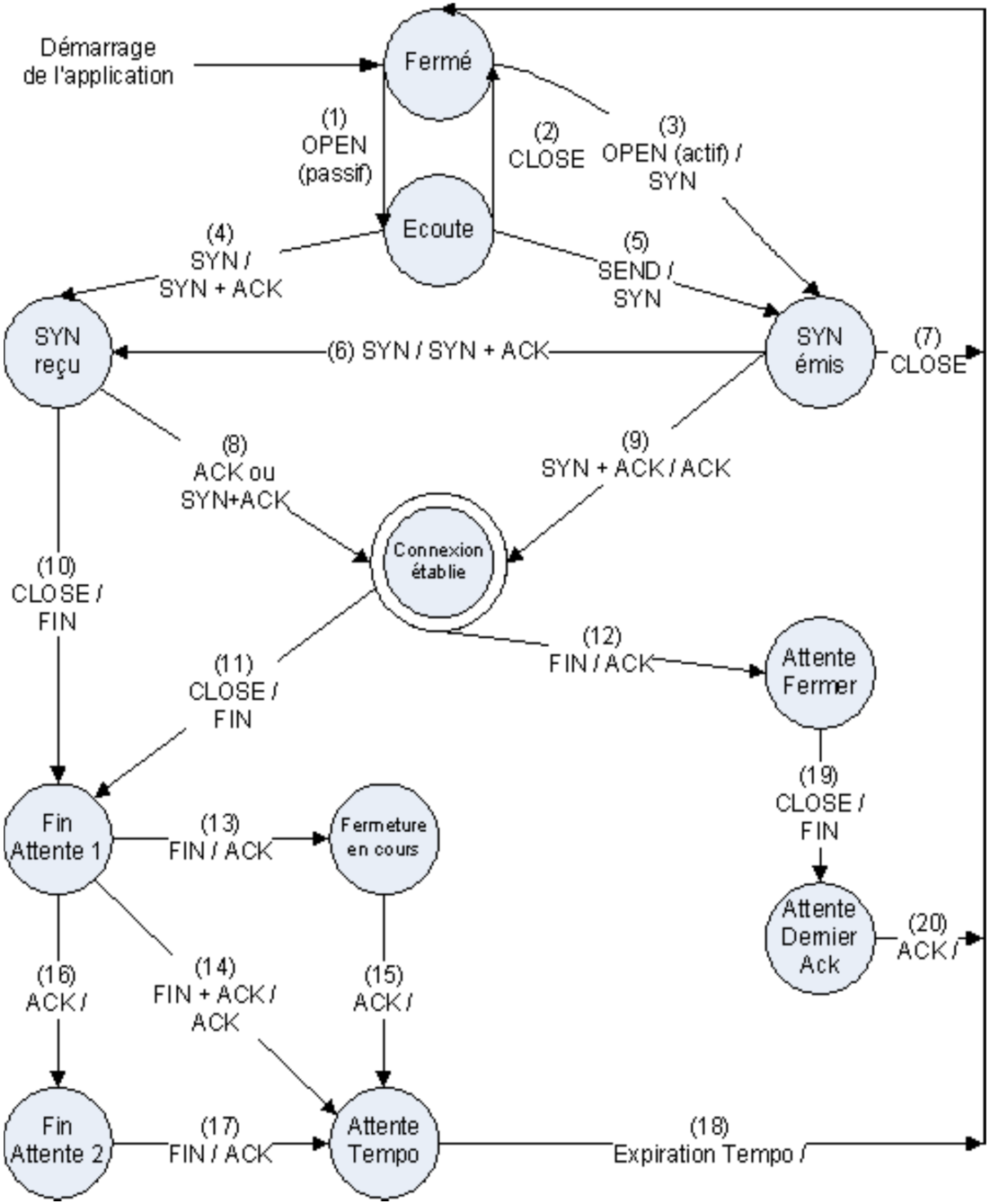
Quelques applications
Généalogie

- Graphes particuliers : arbres
- Affectations génétiques



Quelques applications
Automates

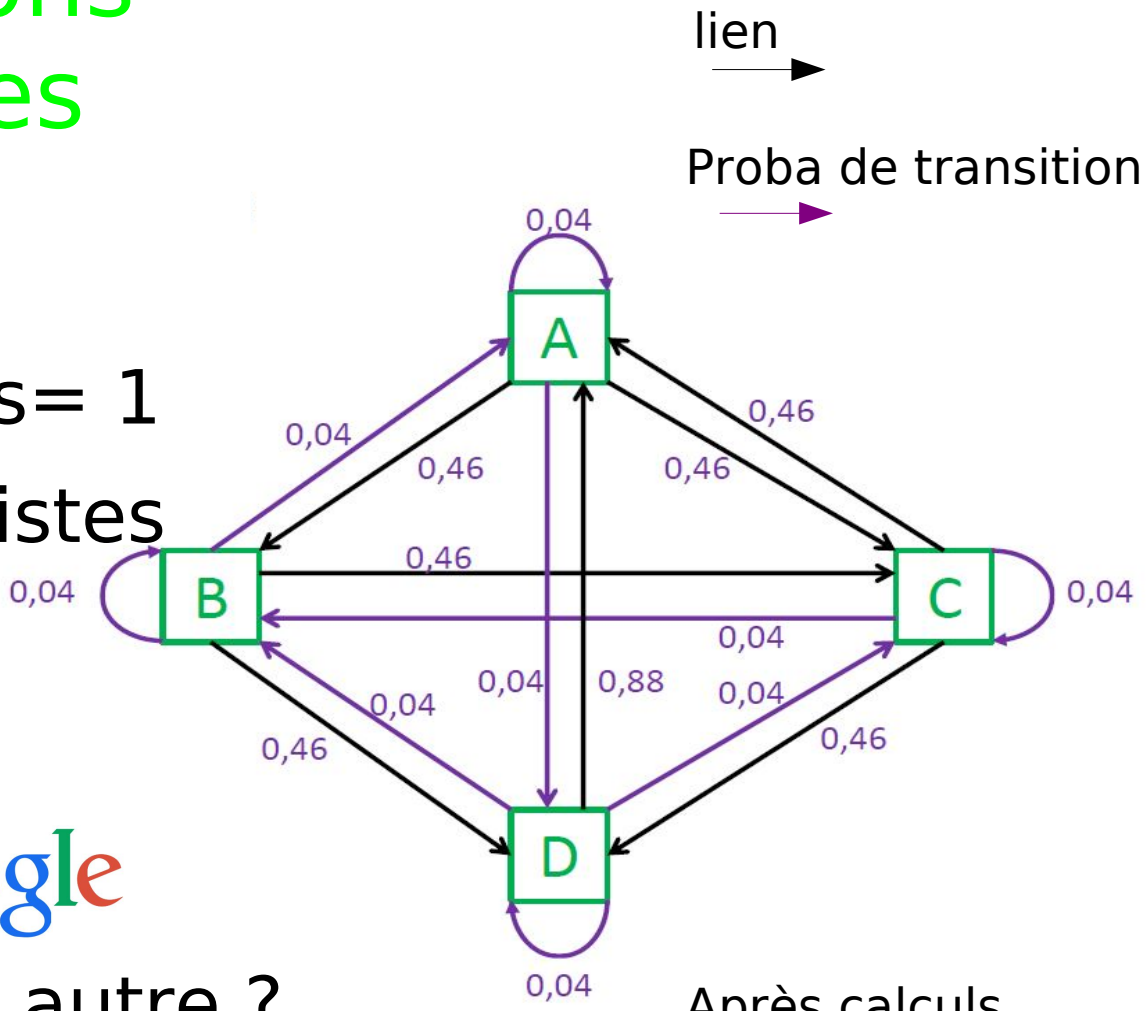
- Décrire un protocole
  - *TCP*
  - *Téléphone*
  - *Feux tricolores*
  - *...*
- Notions
  - États
  - Transitions



Quelques applications
Graphes probalistes

- Chaines de Markov
  - $\sum$ des poids des arcs= 1
  - Transitions probabilistes
  - Proba d'un état ?

- Ex : pageRank Google
  - Surf lien local ou autre ?
  - Ex : 16% choix local
  - Classement des pages

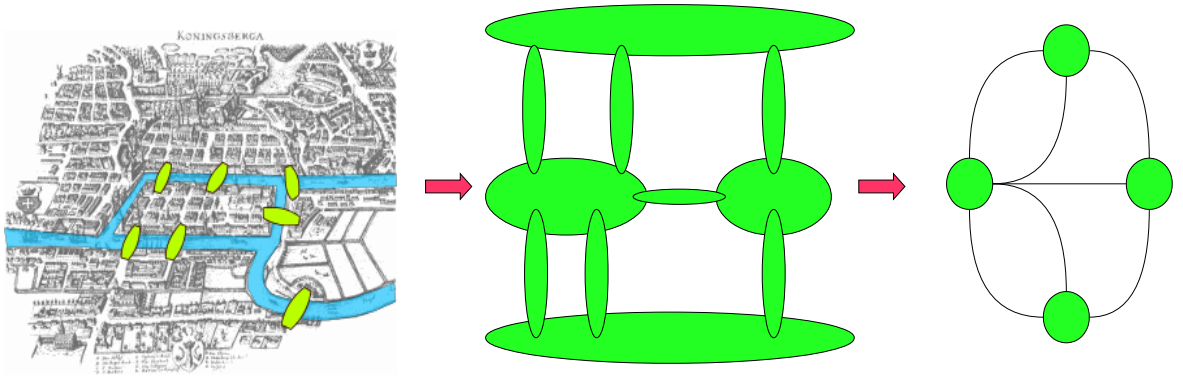


Après calculs ...
P(A) = 34%
P(B) = 18%
P(C) = 26%
P(D) = 22%

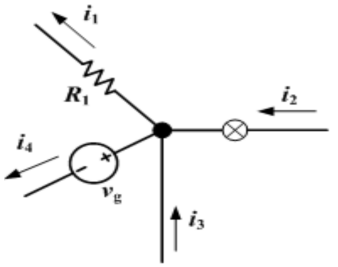
A > C > D > B

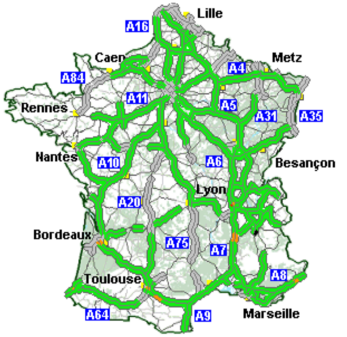
En résumé

- Réalisation de graphes
  - Modélisation de problèmes
  - Représentation graphique (aide à la solution)



- Exploitation du modèle
  - Vérification de propriétés





  - Optimisation de solutions

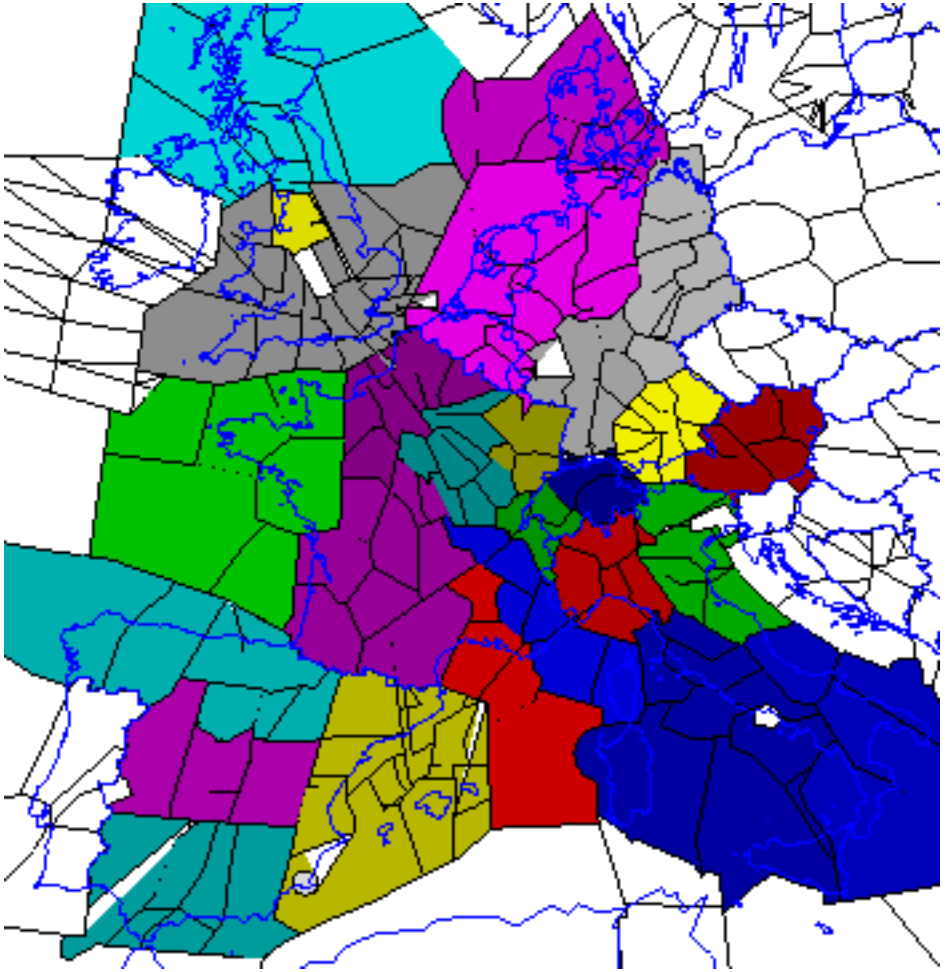
Exemples de modélisations

- Contrôle aérien
  - Minimiser les changements d'écran et répartir le travail entre contrôleurs
- Optimisation de circuits VLSI
  - Découper les circuits trop gros et recalculer leur implantation
- Modèle ?
- Propriétés du modèle à prendre en compte/améliorer ?
- Algorithme d'optimisation des solutions ?

Contrôle aérien : problème

- A un niveau de vol donné

Altitude standard à laquelle vole un avion.



FL260 correspond à Flight Level (niveau de vol) 260

(26 000 pieds, 7 900m environ).

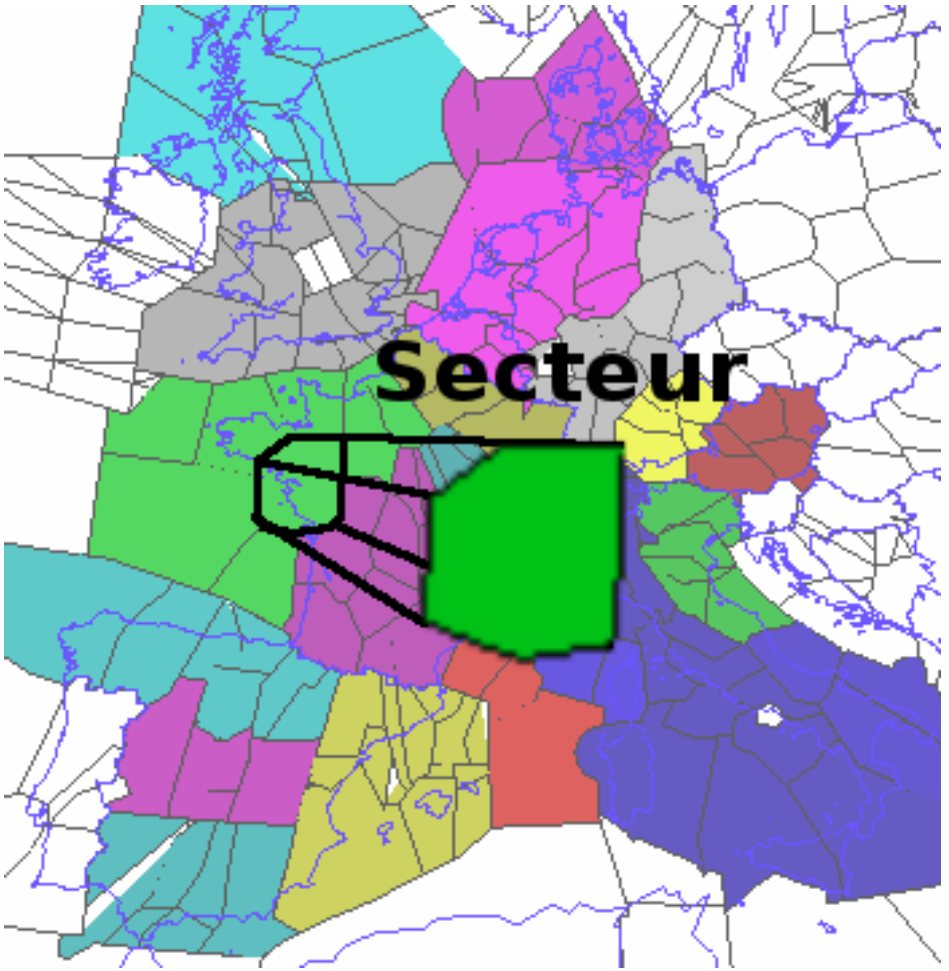
http://www.emse.fr/spip/IMG/pdf/Bichot-11-05-07.pdf

Contrôle aérien : problème

- Secteur

Zone géographique sous

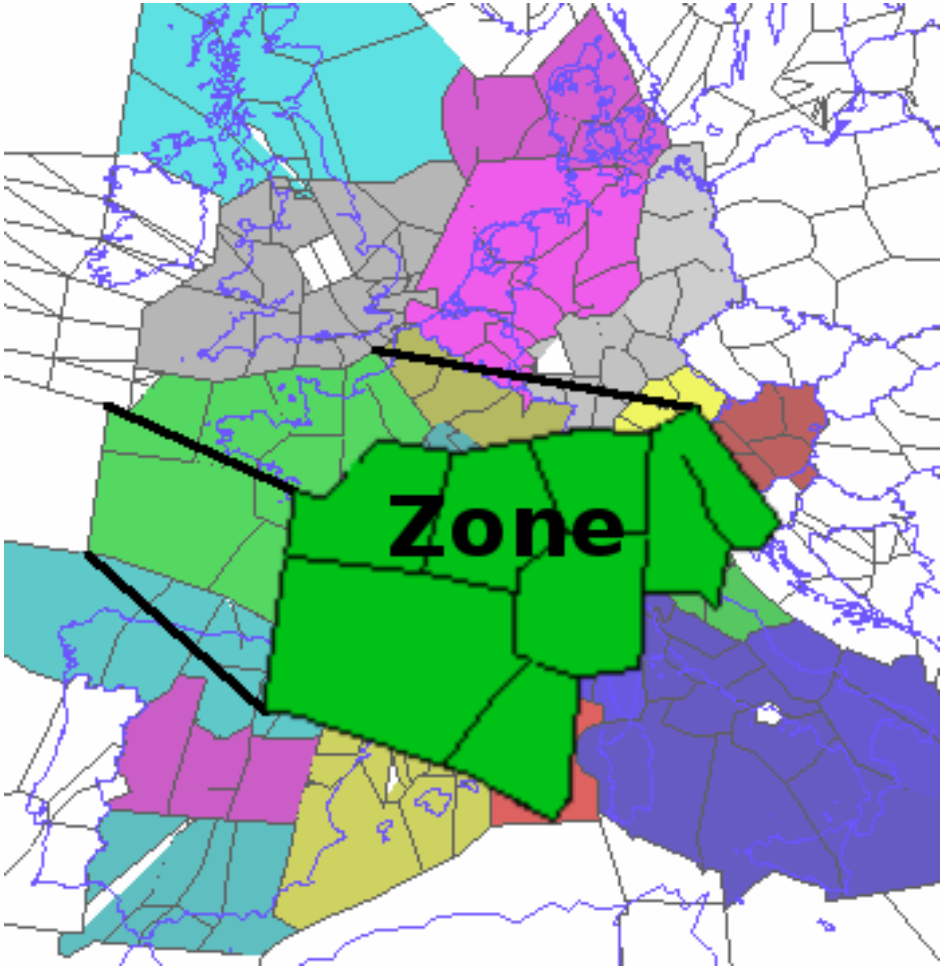
La responsabilité d'un controleur aérien



Contrôle aérien : problème

- Zone de qualification

Ensemble de secteurs sur lesquels un contrôleur aérien est qualifié



Contrôle aérien : objectifs

Augmenter la sûreté, la fluidité et la capacité de l'espace aérien européen.

- Diminuer la charge de travail des contrôleurs aériens :
  - surveillance du trafic
  - résolution de conflits ou conflits potentiels
  - Coordination entre secteurs.

Partition de l'espace

- Trouver une partition $P_k = \{S_1, \ldots, S_k\}$ des sommets d'un graphe G(S, A) :
  - une partie = une zone de qualification
  - un sommet = un secteur
- Les secteurs possèdent des flux d'avions entre eux : G est connexe et pondéré
- k, le nombre de parties = le nombre de zones de qualification

- Espace aérien européen :
  k = 32 parties, 762 sommets, 10 328 arcs.

Plan

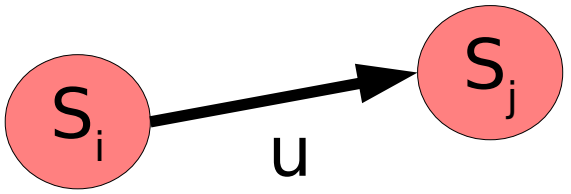
- Relations
- Notions de base sur les graphes
  - Définitions
  - Implantation
  - Parcours
- Algorithmes d'optimisation
  - Calculs de chemins, arbres, flots
- Ordonnancement
  - Définitions
  - PERT et MPM

Bibliographie

- Optimisation combinatoire, T2: programmation discrète
  M. Sakarovitch, Hermann, 1984
- Méthodes d'optimisation combinatoire
  I. Charon, Masson, 1996
- Introduction à l'algorithmique
  T. Cornen, Dunod, 1994
- Wikipédia
  http://fr.wikipedia.org/wiki/Théorie_des_graphes
- Algorithmic graph theory
  J. A. McHugh, Prentice Hall, 1990

Graphes
Terminologie

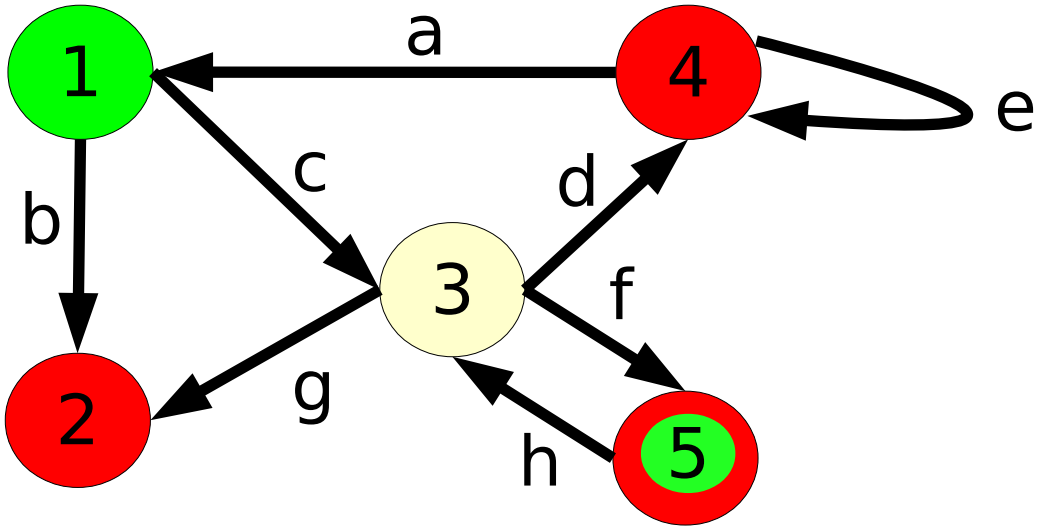
- **Graphe orienté** G : 2 ensembles
  - X : ensemble des **sommets** de G
  - U : ensemble des **arcs** de G
- Un arc $u \in U$ est un couple ordonné (S_i, S_j)
  - $S_i \in X$ est le sommet **initial** de u
  - $S_j \in X$ est le sommet **final/terminal** de u



U dénote une relation entre S_i et S_j

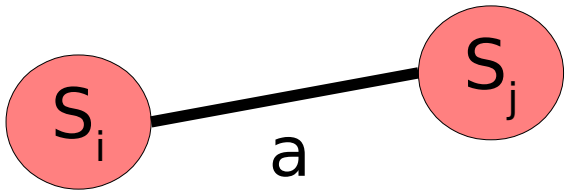
Graphes
Terminologie

- **ordre** de G = (X, U) : nombre de sommets |X|
- **p-graphe** : $\forall S_i, S_j \in X$,l'ensemble des arcs $(S_i \rightarrow S_j)$ a un cardinal $\leq p$
- Si $S_i \rightarrow S_j \in U$, S_j est un **successeur** de S_i et S_i est un **prédécesseur** de S_j



Graphes
Terminologie

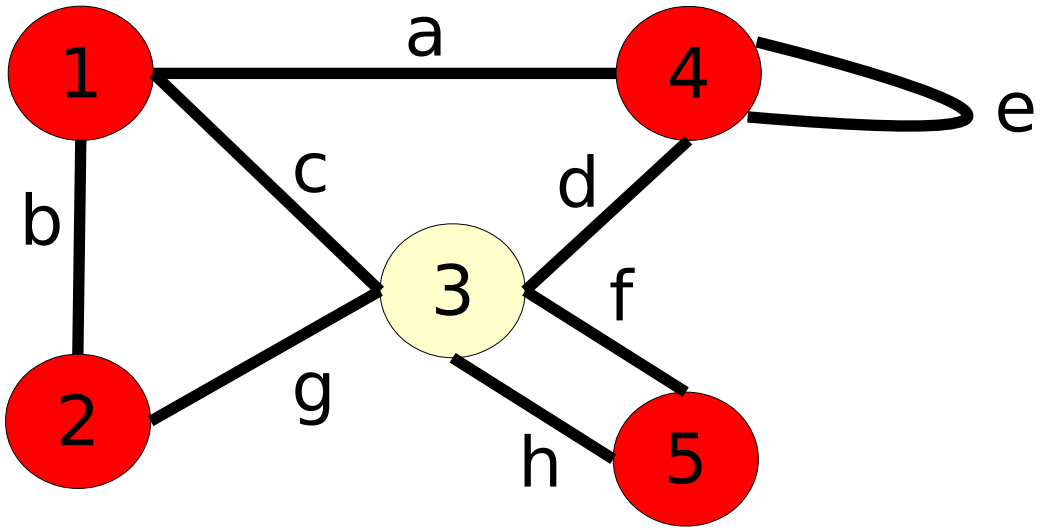
- **Graphe non orienté** G = (X, U)
  - U : ensemble des **arêtes** de G
- Une arête a ∈ U est un couple non ordonné (S_i, S_j)
  - S_i, S_j ∈ X sont **adjacents**



a dénote une relation symétrique entre S_i et S_j

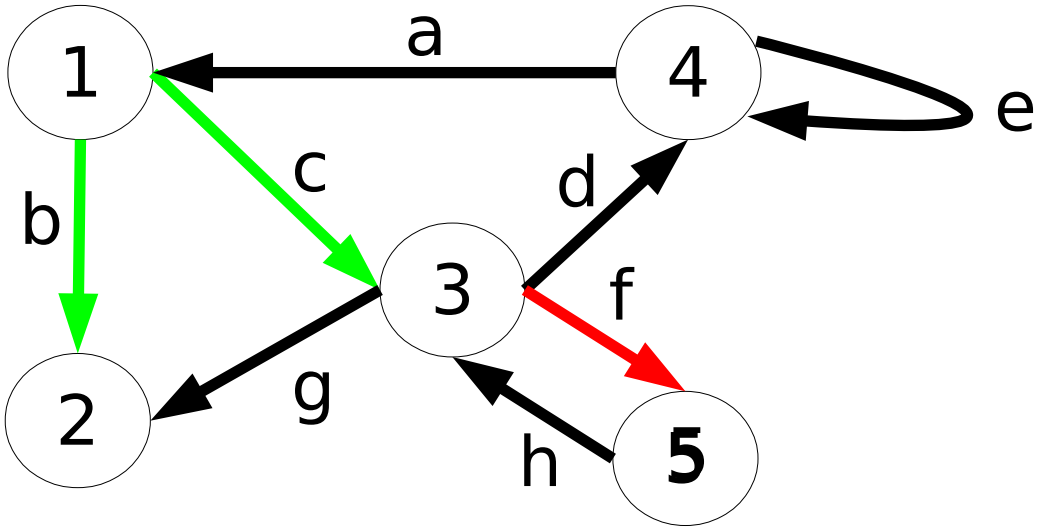
Graphes
Terminologie

- **multigraphe**, si plusieurs arêtes sur au moins un couple S_i, S_j
- graphe **simple** si au plus une arête pour tout couple S_i, S_j et aucune boucle



Graphes
Terminologie

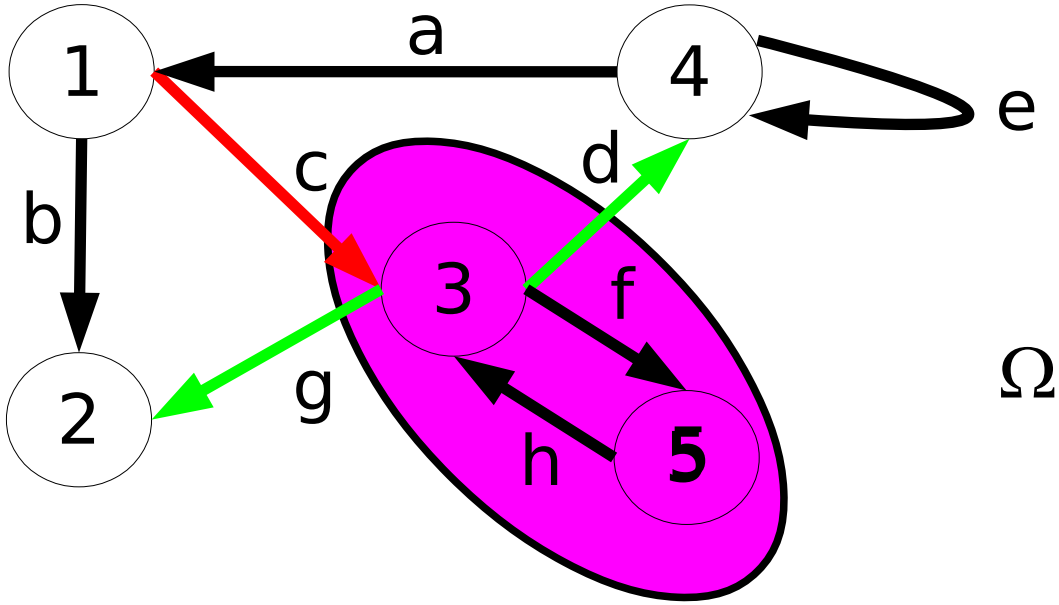
- **degré, demi-degrés**, $d^+(S_i) + d^-(S_i) = d(S_i)$
  nombre d'arcs sortants (resp. entrants)
  demi-degré extérieur (resp. intérieur) de S_i



- $d^+(S_1) = 2$; $d^-(S_5) = 1$

Graphes
Terminologie

- **Cocycles positif, négatif** de $A \subseteq X$
  - $\Omega^+(A) = \{ u \in U \mid initial(u) \in A \wedge final(u) \notin A \}$
  - $\Omega^-(A) = \{ u \in U \mid initial(u) \notin A \wedge final(u) \in A \}$



$$\Omega(A) = \Omega^+(A) \cup \Omega^-(A)$$

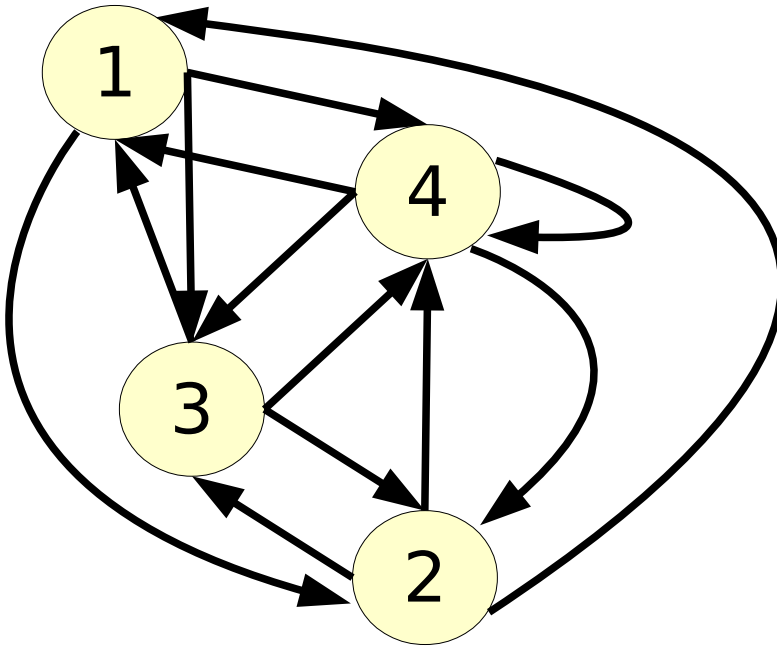
- $\Omega^+(A) = \{d, g\}$; $\Omega^-(A) = \{c\}$

Graphes
Terminologie

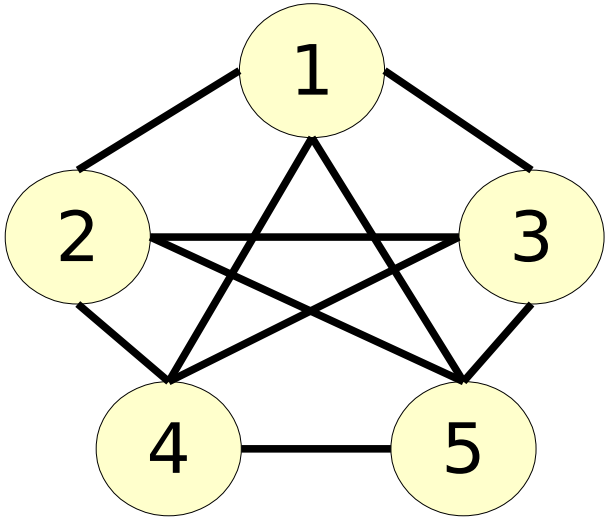
- **Graphe complet, clique**
  - G = (X, U) est complet ssi

    $\forall S_i, S_j \in X^2, i \neq j,$

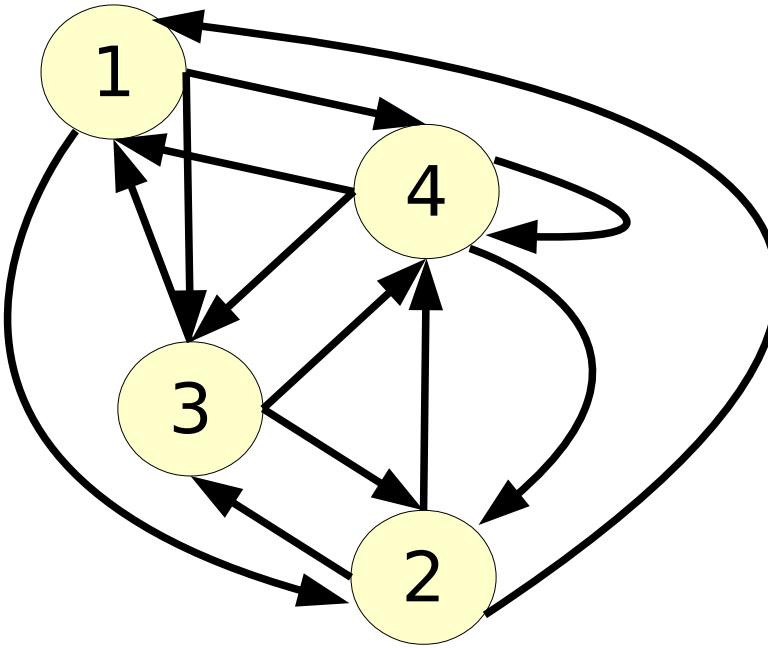
    $\exists$ arc $u=(S_i \rightarrow S_j) \in U$



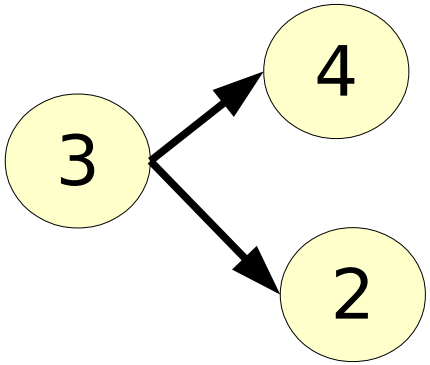
- **Clique** pour les graphes non orientés



Graphes
Terminologie



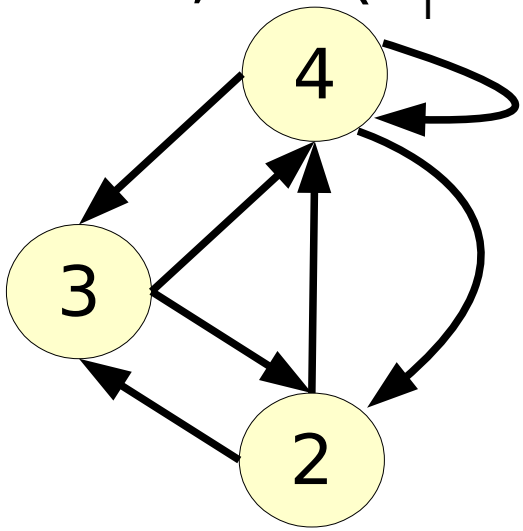
- **Sous-graphe**
  G' = (X', U') est un sous-graphe de G= (X,U) ssi
  $X' \subseteq X \wedge U' \subseteq U$



- Sous-graphe G'=(X', U') **engendré** par $X' \subseteq X$:

  $U' = \{u \in U \mid \exists S_i, S_j \in X', u=(S_i \rightarrow S_j)$ *ou* $u=(S_j \rightarrow S_i)\}$

  $X' = \{S_2, S_3, S_4\}$

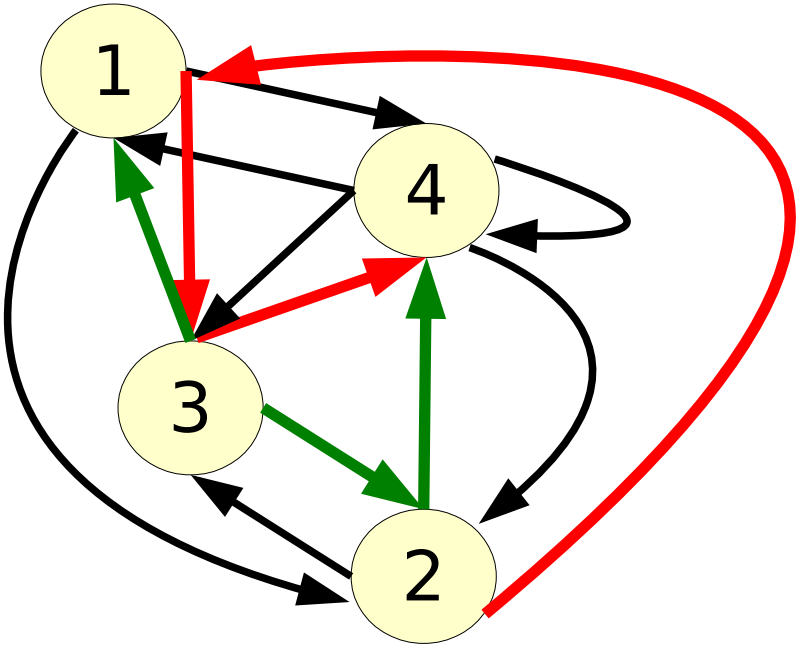


Graphes
Terminologie

- **Chemin dans G = (X, U)**
  C = $u_1, u_2, ..., u_m$ avec

  $u_1, u_2, ..., u_m \in U$
  est un chemin *ssi*

  $\forall a \in]1..m]$,
  $initial(u_a) = final(u_{a-1})$

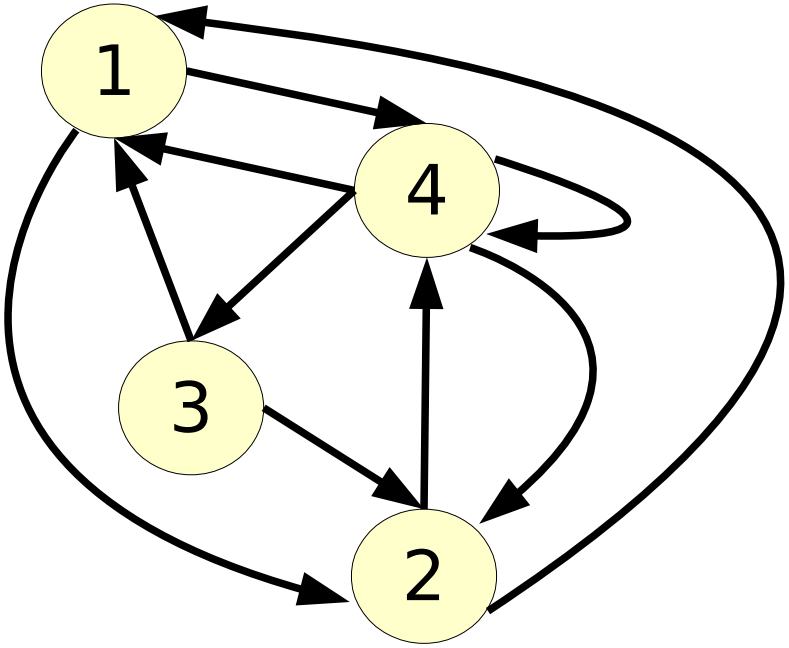


- **Chaine dans G = (X, U)**
  C = $u_1, u_2, ..., u_m$, avec

  $u_1, u_2, ..., u_m \in U$
  est une chaine *ssi*

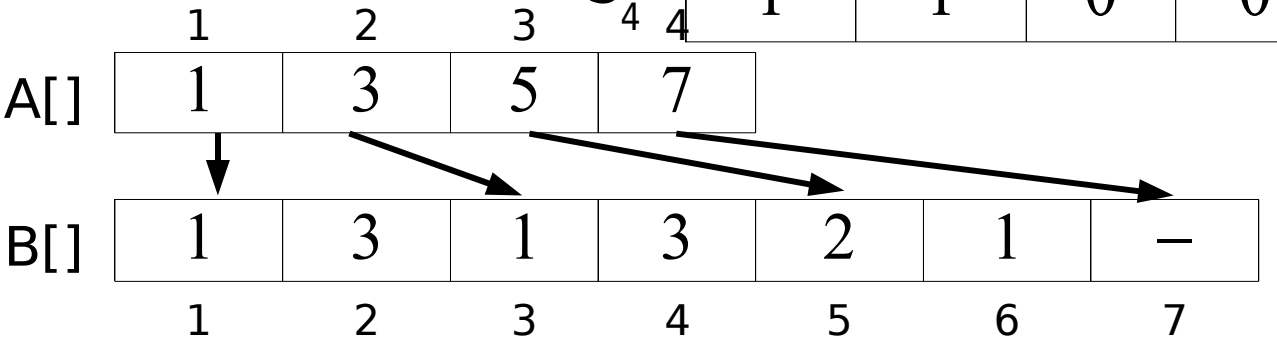
  $\forall a \in]1..m]$, u_a et u_{a-1} sont adjacents

Graphes
Implantation



- G = (X, U)
  représentation en mémoire
- Matrices
  - Incidence sommets arcs
  - Incidence sommets arêtes
  - Adjacence
- Listes
  - Adjacence
  - Incidence
  - Cocycles

	S_1	S_2	S_3	S_4
S_1	1	0	0	1
S_2	0	1	1	0
S_3	0	0	1	1
S_4	1	1	0	0



Implantation
Matrice d'incidence sommets-arcs

- G = (X, U)

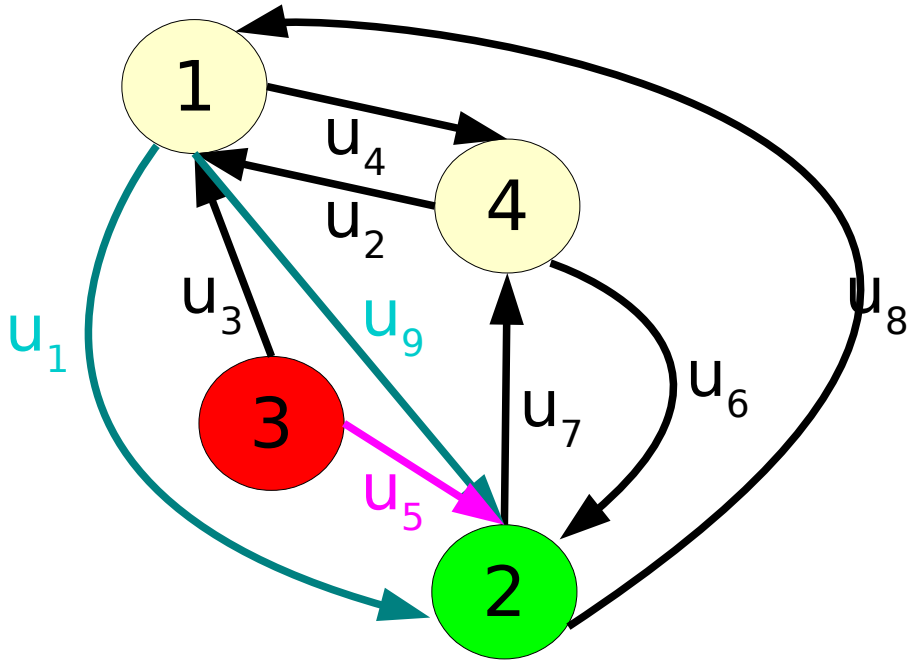
  $\forall\ u = (S_i \rightarrow S_j) \in U$

  $A_{iu} = 1$

  $A_{ju} = -1$

  $\forall\ x \neq i$ et j, $A_{xu} = 0$

- Boucles ?
- Taille mémoire
  |X| x |U|
- Sommets-arêtes pour graphes non orientés
  $A_{iu} = A_{ju} = 1$

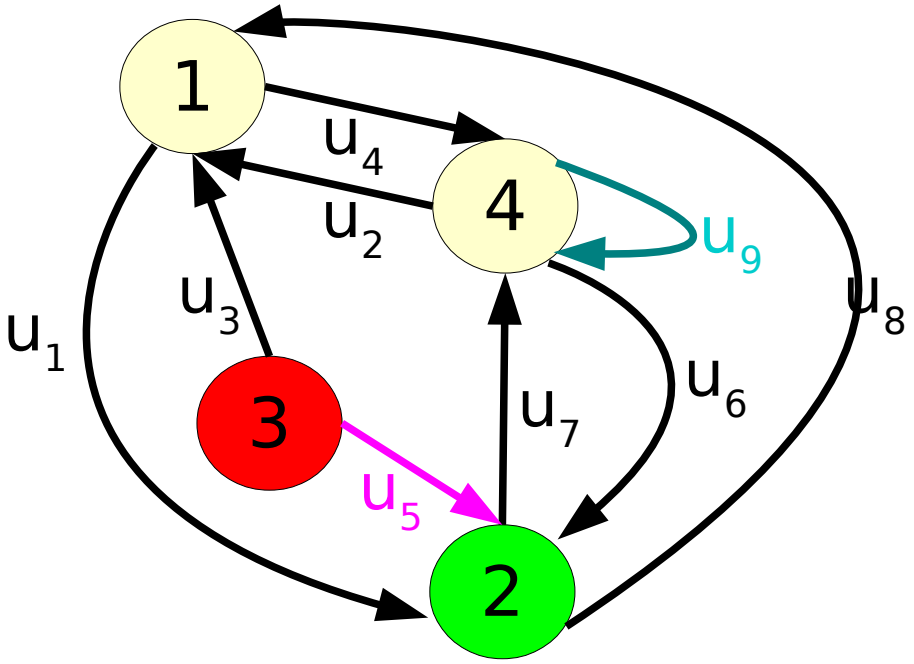


	U_1	U_2	U_3	U_4	U_5	U_6	U_7	U_8	U_9
S_1	1	−1	−1	1	0	0	0	−1	1
S_2	−1	0	0	0	−1	−1	1	1	−1
S_3	0	0	1	0	1	0	0	0	0
S_4	0	1	0	−1	0	1	−1	0	0

Implantation
Matrice d'adjacence sommets-sommets

- $G = (X, U)$
  $\forall\ u = (S_i \rightarrow S_j)$

$$\begin{cases} u \in U \Rightarrow A_{ij} = 1 \\ u \notin U \Rightarrow A_{ij} = 0 \end{cases}$$



- Taille mémoire
  $|X|^2$
- Multi-graphes ?

	S_1	S_2	S_3	S_4
S_1	0	1	0	1
S_2	1	0	0	1
S_3	1	1	0	0
S_4	1	1	0	1

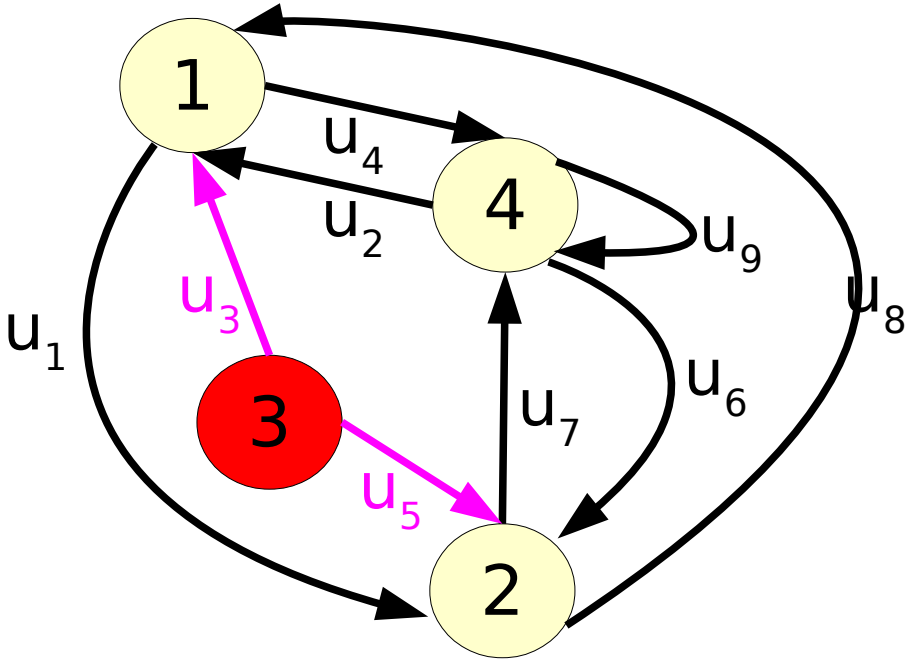
Implantation
Liste d'adjacence

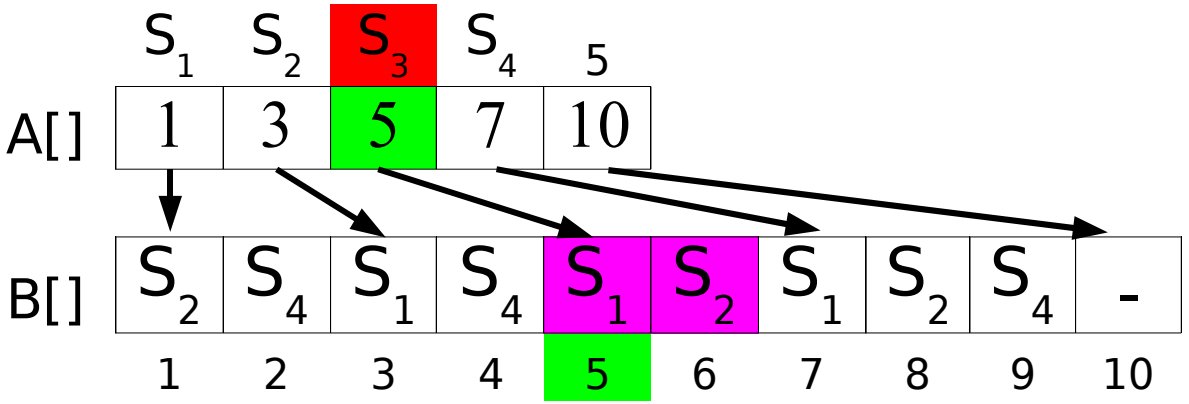
- G = (X, U) (1-graphe)
  - 2 tableaux A et B
- B[] contient, à partir de l'index A[i], la liste des sommets adjacents à S_i
- On a :

$d^+(S_i) = A[i+1]-A[i]$

$A[i] = \sum_{j=1}^{i-1} d^+(S_j) + 1$

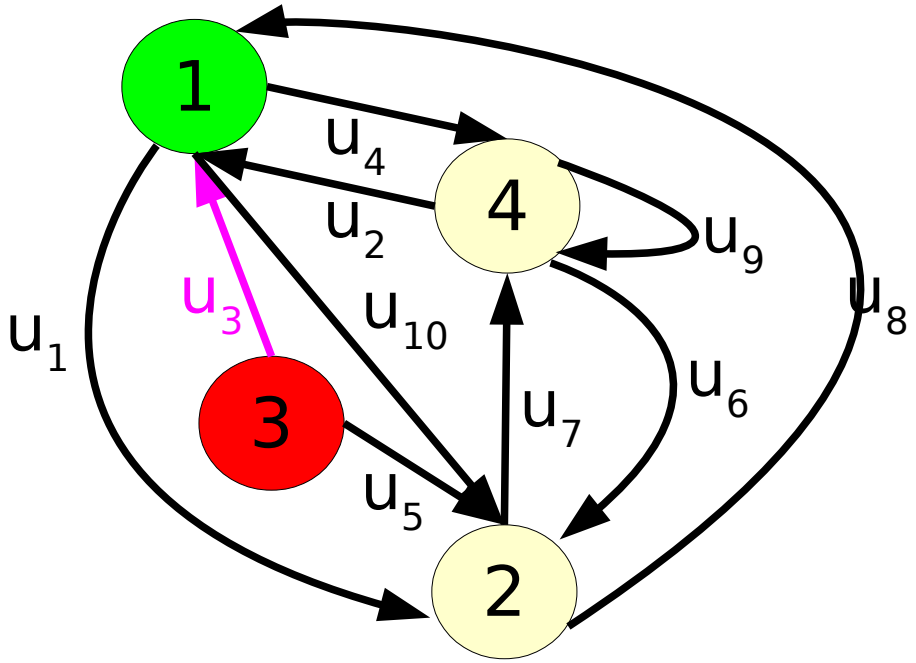
- Taille mémoire |X|+|U|





Implantation
Liste des arcs

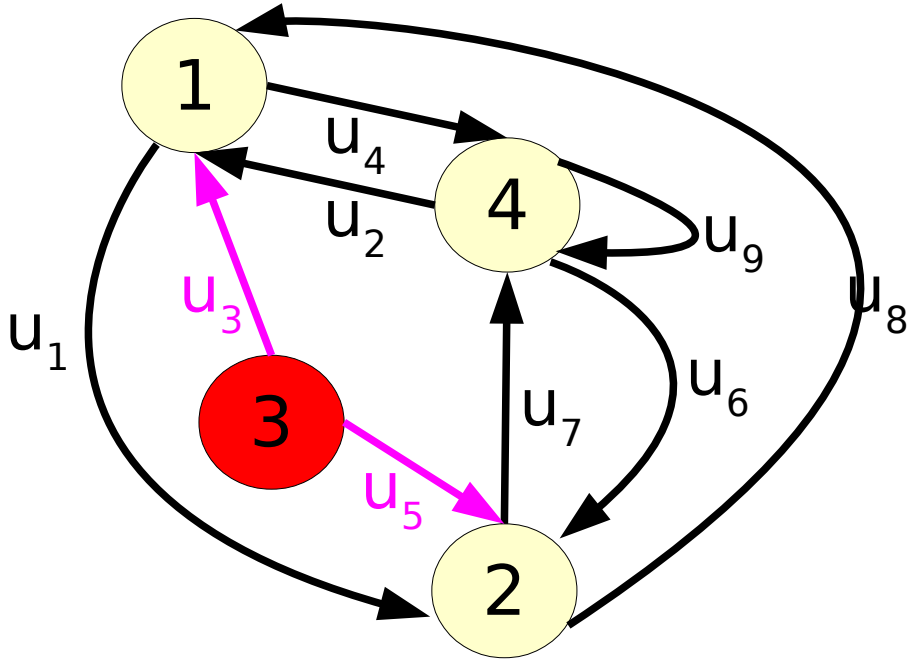
- G = (X, U)
  - 2 tableaux EI[] et EF[]
- EI[u] contient le sommet initial de l'arc u, EF[u] le sommet final de u
- Tout type de graphe
- Taille mémoire 2*|U|

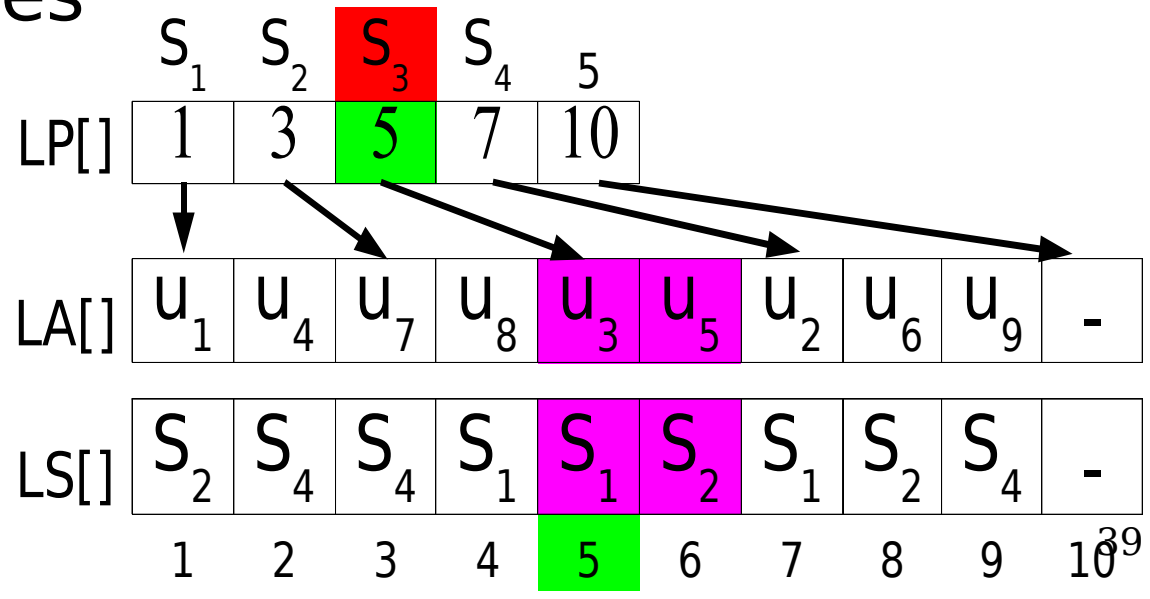


	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
EI[]	1	4	3	1	3	4	2	2	4	2
EF[]	2	1	1	4	2	2	4	1	4	1

Implantation
Liste des cocycles

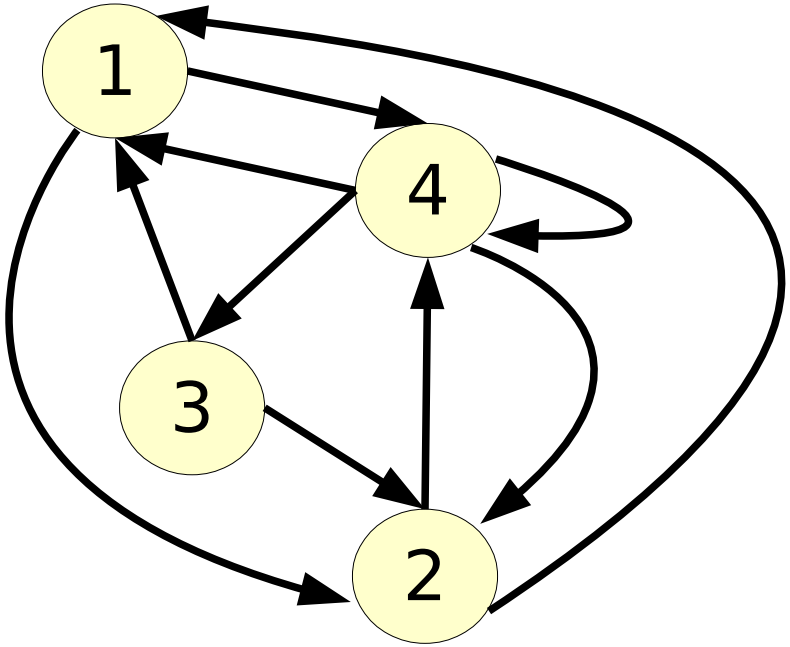
- G = (X, U)
  - 2 tableaux LP[] et LA[]
- LA[] contient, à partir de l'index LP[i], la liste des arcs émanant de S_i (cocycle positif $\Omega^+(i)$)
- LS[] contient la liste des sommets terminaux associés aux arcs
- Taille mémoire |X|+2*|U|





Graphes
Choix de l'implantation

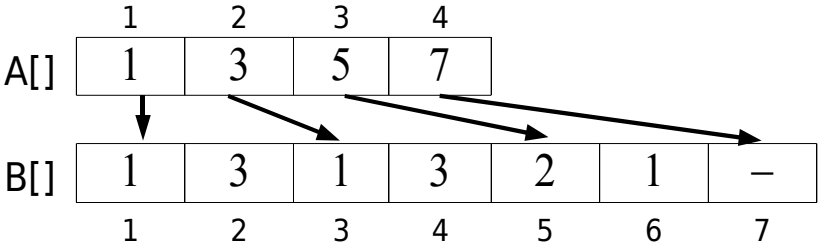
- G = (X, U)
  - Suivant type de graphe
  - Suivant type d'application



- Matrices
  - Gourmandes en mémoire
  - Calculs directs

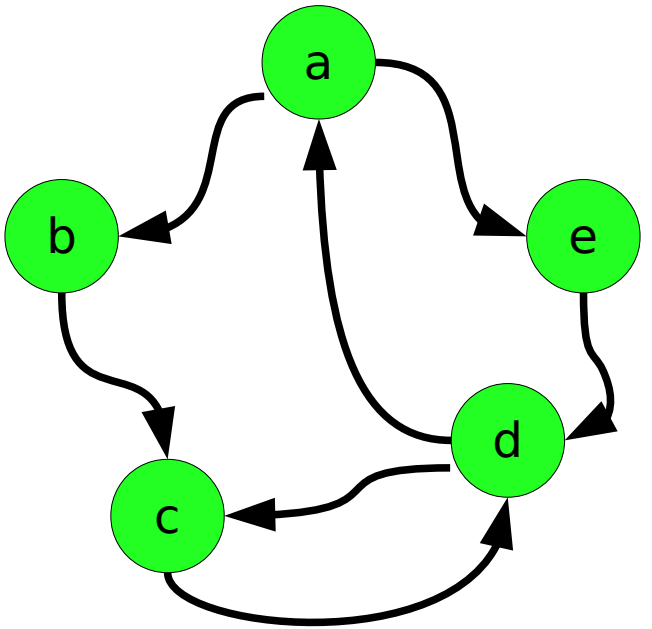
	S_1	S_2	S_3	S_4
S_1	1	0	0	1
S_2	0	1	1	0
S_3	0	0	1	1
S_4	1	1	0	0

- Listes
  - Calculs plus complexes
  - Beaucoup moins d'espace



Recherche dans un graphe

- Données : un graphe G = (X, U) quelconque
- Objectif : Comment visiter tous les sommets du graphe une seule fois ?
- Applications : trouver une donnée, afficher les sommets ...
- Exemple : y a t il un sommet nommé c ?



Algorithme de recherche en profondeur

```
Profondeur(Graphe G;  Sommet V)( )
    Sommet S
    Liste li_adj
    Si  Ouvert( V, G ) =  vrai
        Visiter( V, G )
        li_adj ← ListSadj( V, G )
        Pour chaque S dans li_adj
        Si  Ouvert( S, G ) =  vrai
            Profondeur( G, S )
```

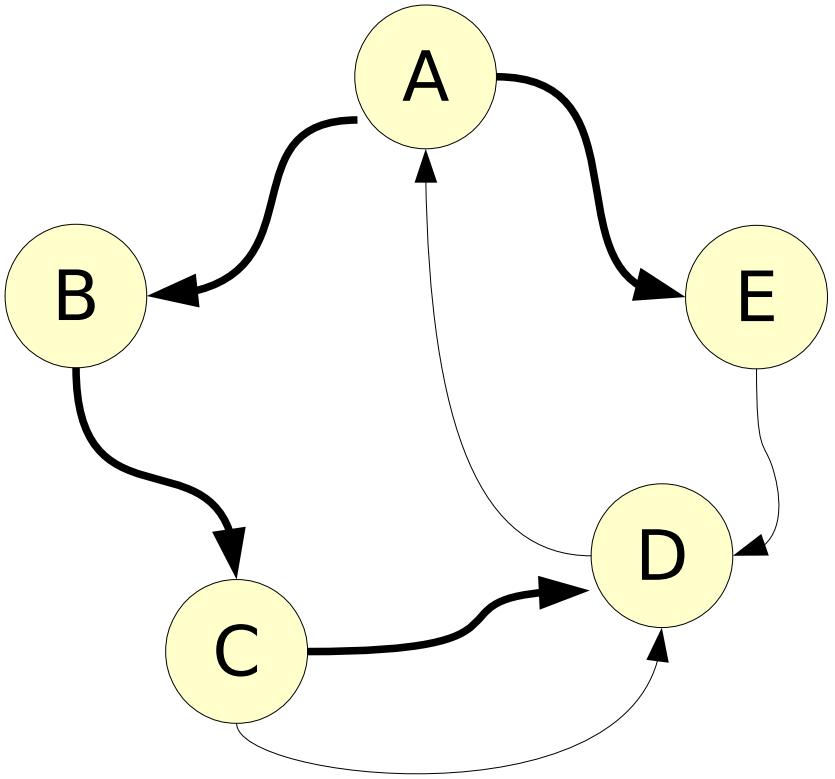
Algorithme de recherche en profondeur

- avec les fonctions :

  - Visiter(Sommet S; Graphe G)
    effectue une opération sur un sommet et le ferme.

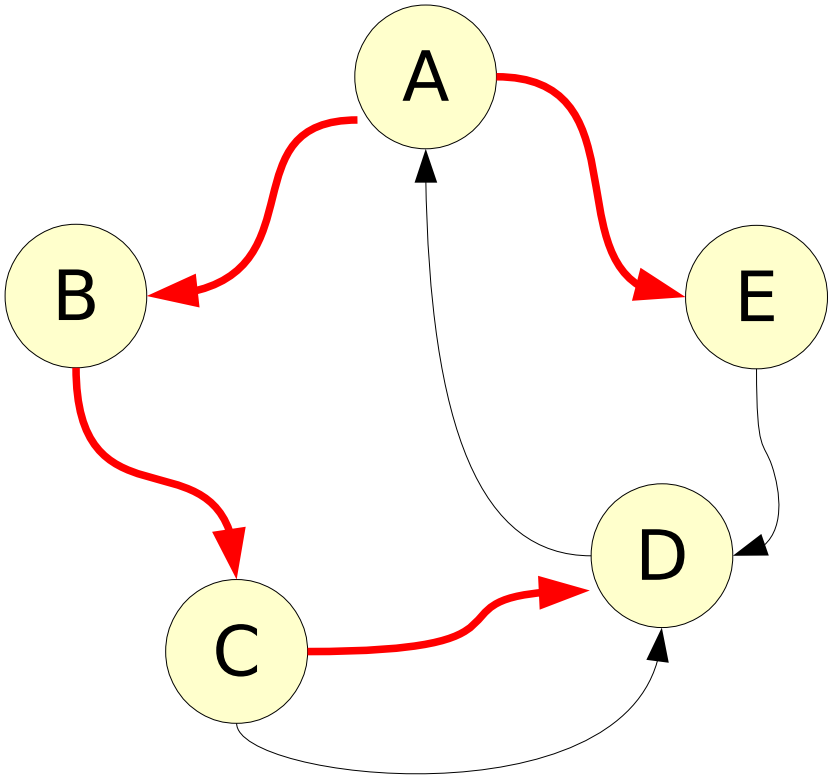
  - Ouvert(Sommet S; Graphe G) (booléen)
    retourne vrai si S est ouvert et faux si S est fermé.

Exemple de parcours en profondeur



#	Ancetres & sommet courant	Sommets adjacents

Exemple de parcours en profondeur

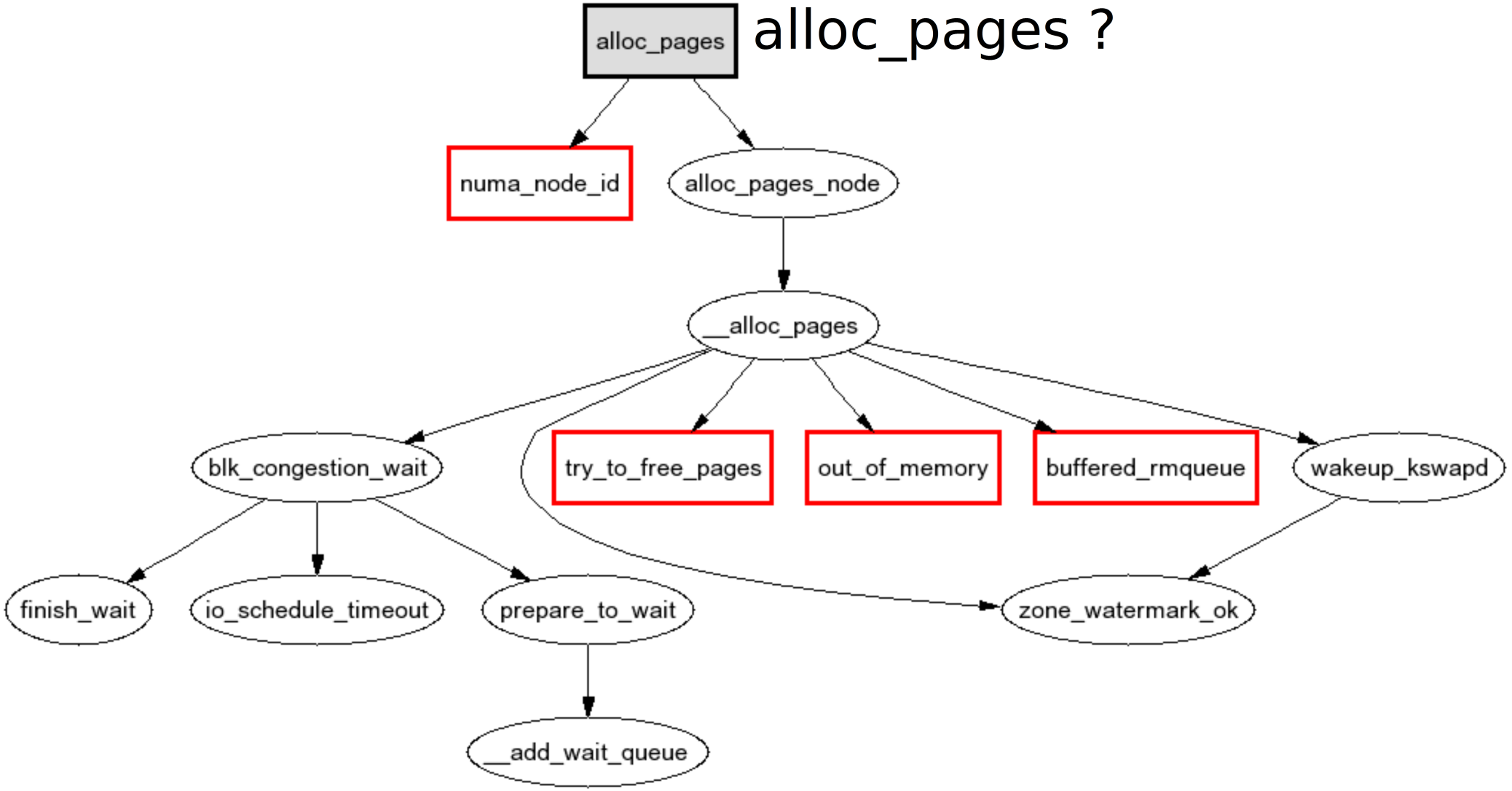


#	Ancetres & sommet courant	Sommets adjacents
1	A	B E
2	A B	C
3	A B C	D
4	A B C D	~~A~~
5	A B C	~~D~~
6	A B	~~C~~
7	A	~~B~~ E
8	AE	~~D~~
9	A	~~BE~~
10	-	-

Arbre de recouvrement

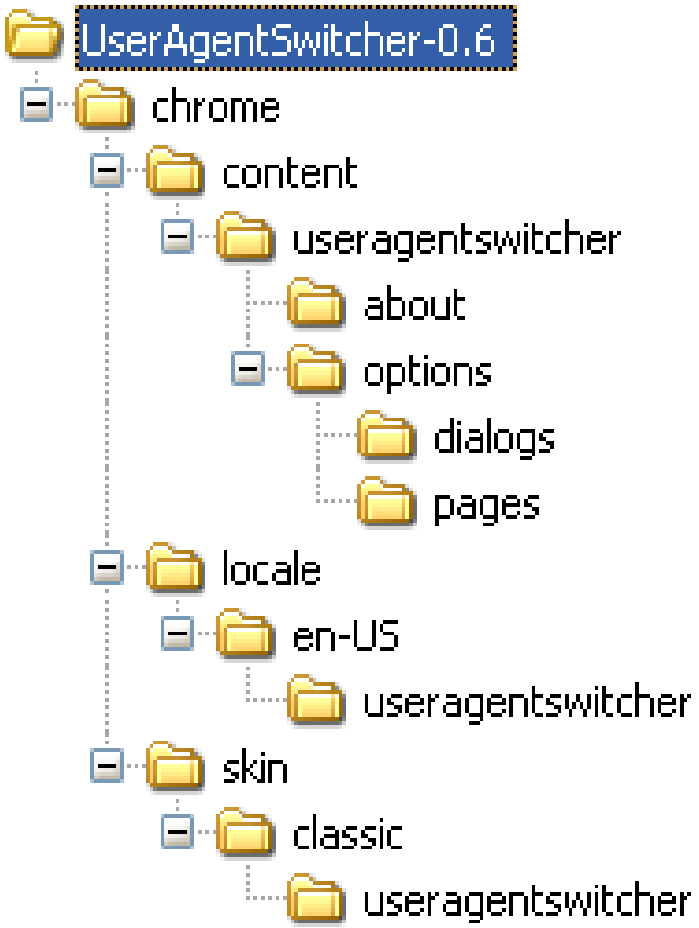
Applications

finish_wait est elle appelée par le programme alloc_pages ?



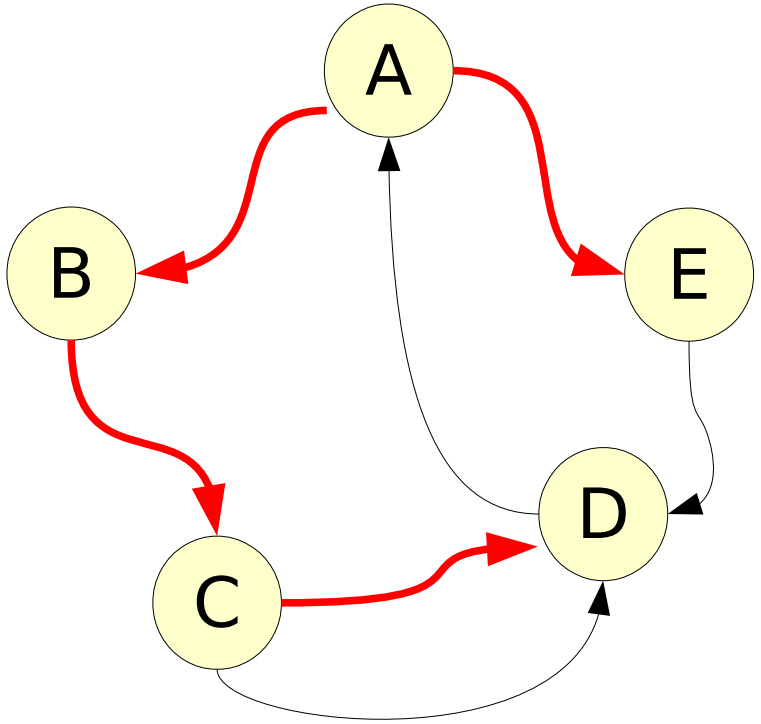
http://www.csn.ul.ie/~mel/projects/codeviz/alloc_pages.png

Applications



Quels sont les répertoires à sauvegarder incrémentalement ?

Remarques



- Le choix parmi les sommets ouverts est arbitraire
  → ordres de visite variable
- Si aucun chemin S0→S, alors S non visité
  → composante connexe
- Arbre de recouvrement 
- Explorer les adjacents avant les successeurs
  → largeur d'abord

Recherche en largeur

- Même principe de marquage que pour une recherche en profondeur
- On visite S_0 puis tous ses sommets adjacents ouverts. Pour ceux-ci, on examine ensuite leurs sommets adjacents ouverts, *etc*.
- Applications : trouver une donnée, afficher les sommets ...

- Pile pour stocker les adjacents non encore explorés

Algorithme de recherche en largeur

Largeur(Graphe G; Sommet S_0)( )

Sommet S

Pile pile

Liste li_adj

mettre(S_0, pile)

visiter(S_0, G)

tant que vide(pile) = faux

retirer(S, pile)

li_adj ← ListSadj(S, G)

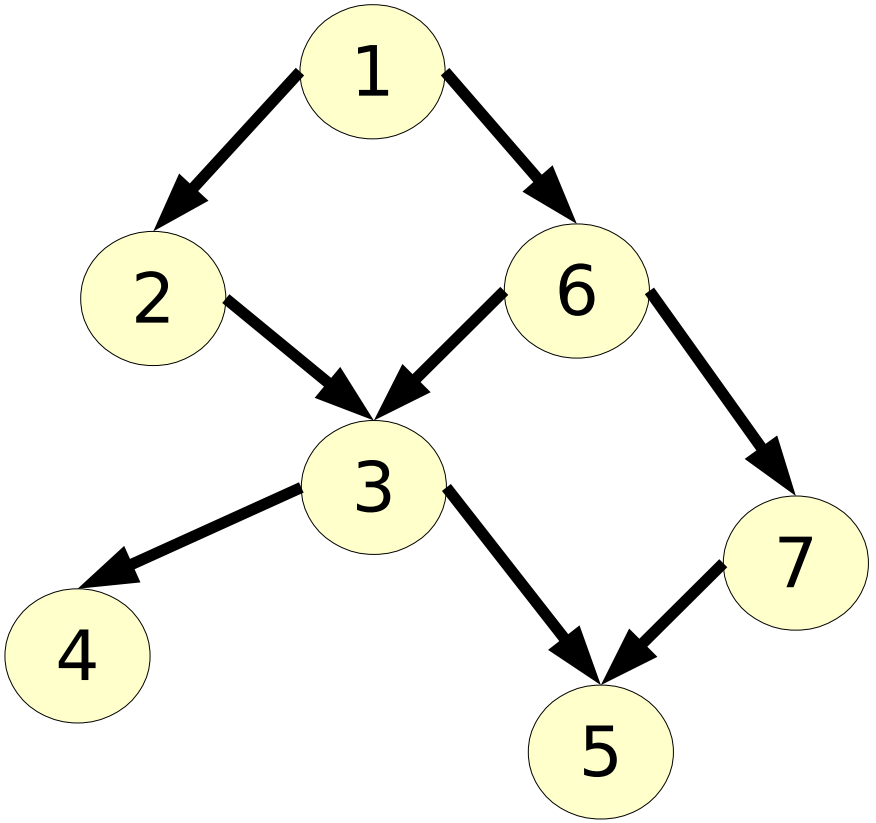
Pour chaque S dans li_adj

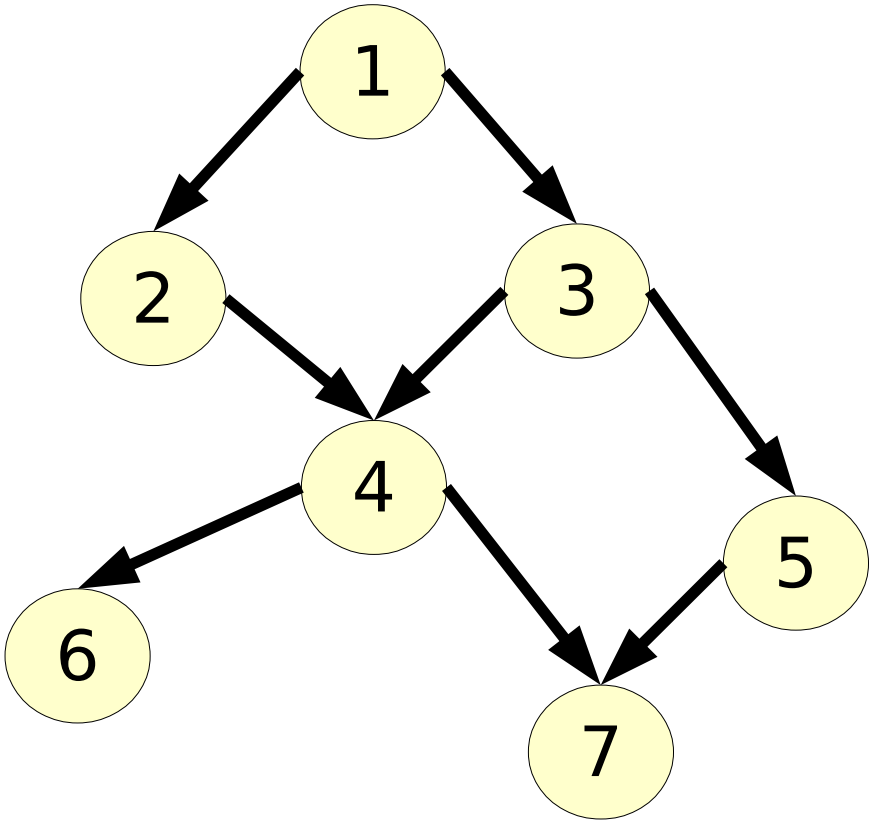
Si Ouvert(S, G) = vrai

visiter(S, G)

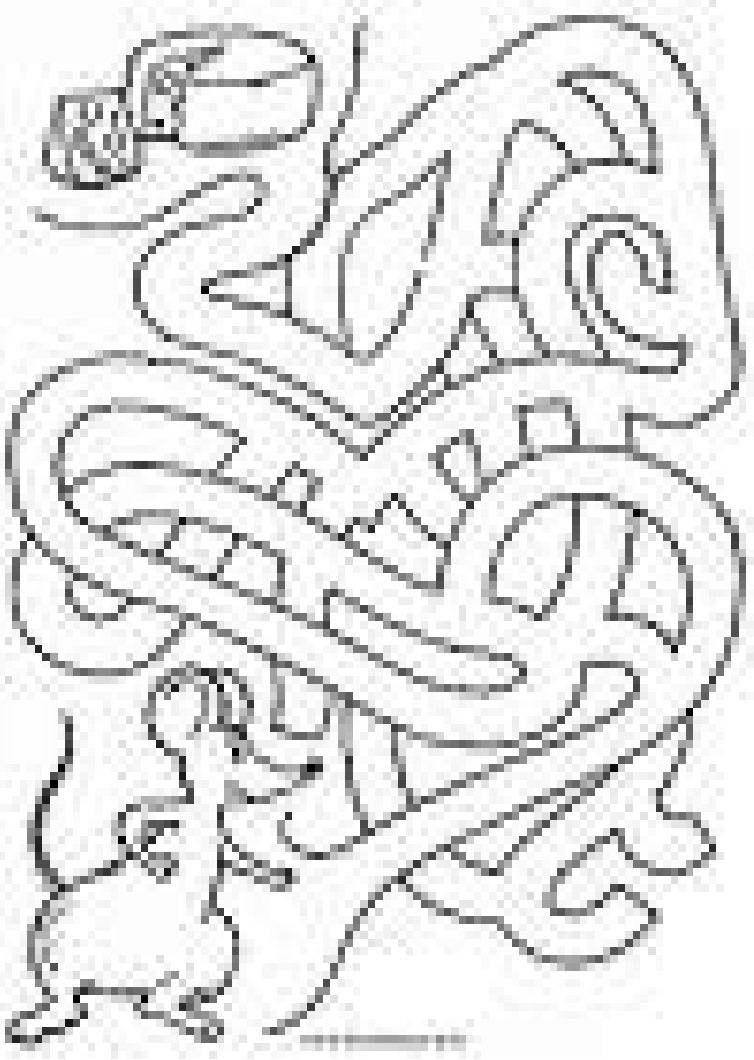
mettre(S, pile)

Largeur d'abord *vs*. Profondeur d'abord





Composantes connexes

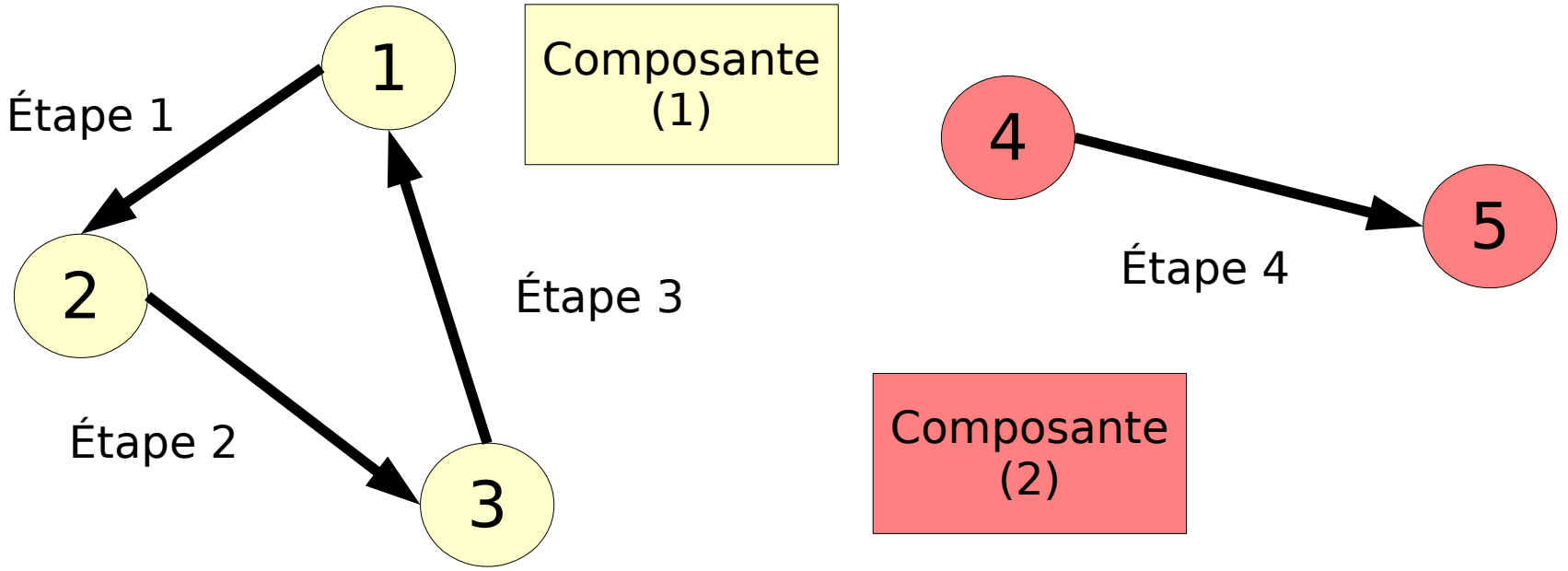


www.atelier-duotang.com

- Existe il un chemin entre 2 sommets i et j
  → i et j sont ils dans la même *composante connexe*
- Tous les sommets sont ils interconnectés
  → composante *fortement* connexe

Composantes connexes
Principe

1. Appliquer la recherche en profondeur à partir de S_0
2. Si tous les sommets ne sont pas atteints, recommencer à partir d'un sommet non atteint

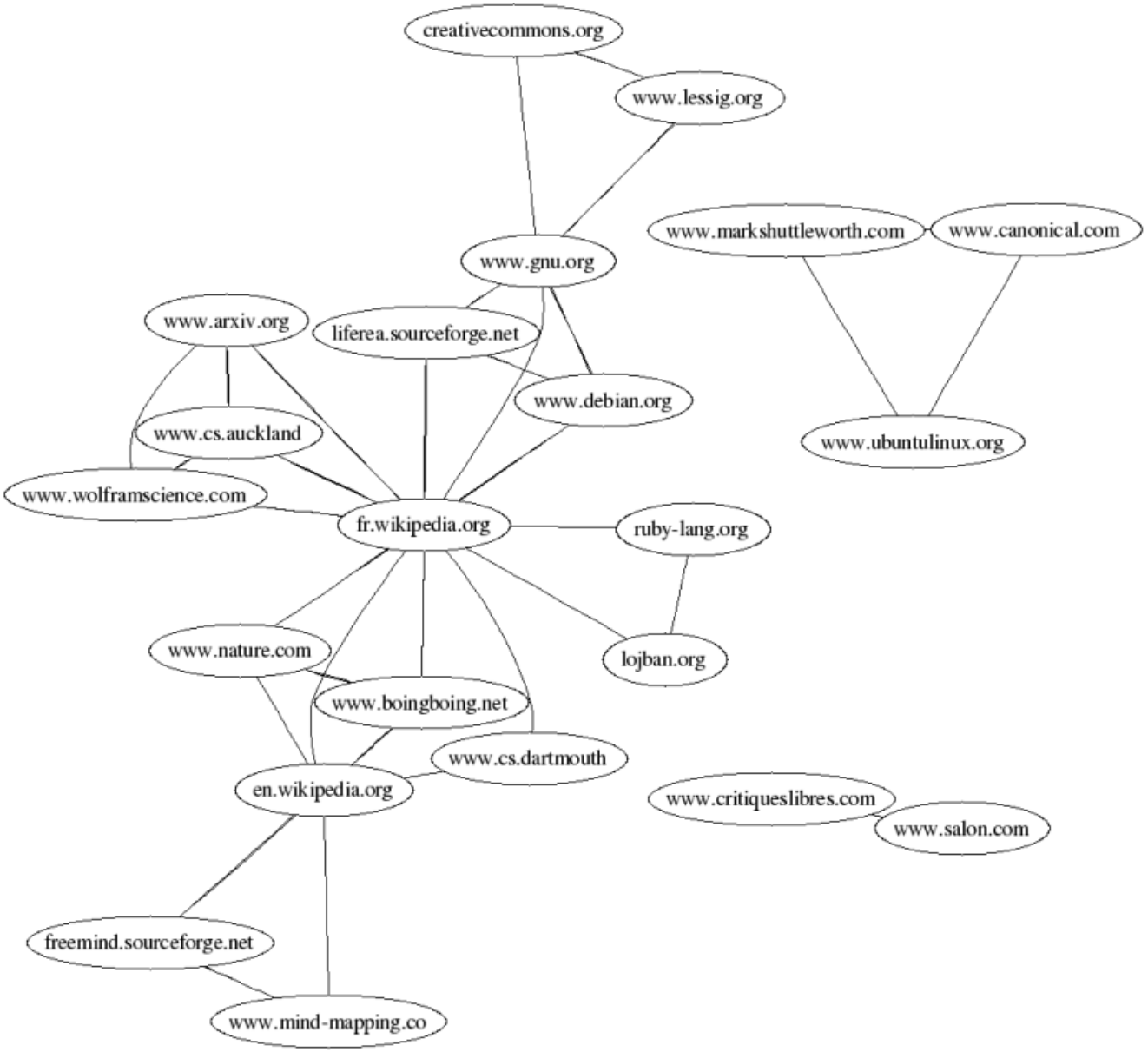


Composantes connexes - Algorithme

```
Connexité(Graphe G)(Liste de Sommets )
    Sommet S
    Liste L ← sommets(G)
    NC ← 0
    Pour chaque S dans L
        Si ouvert(S, G) = vrai
        NC ← NC+1
        Profondeur2(G, S, NC)
    retourner L
```

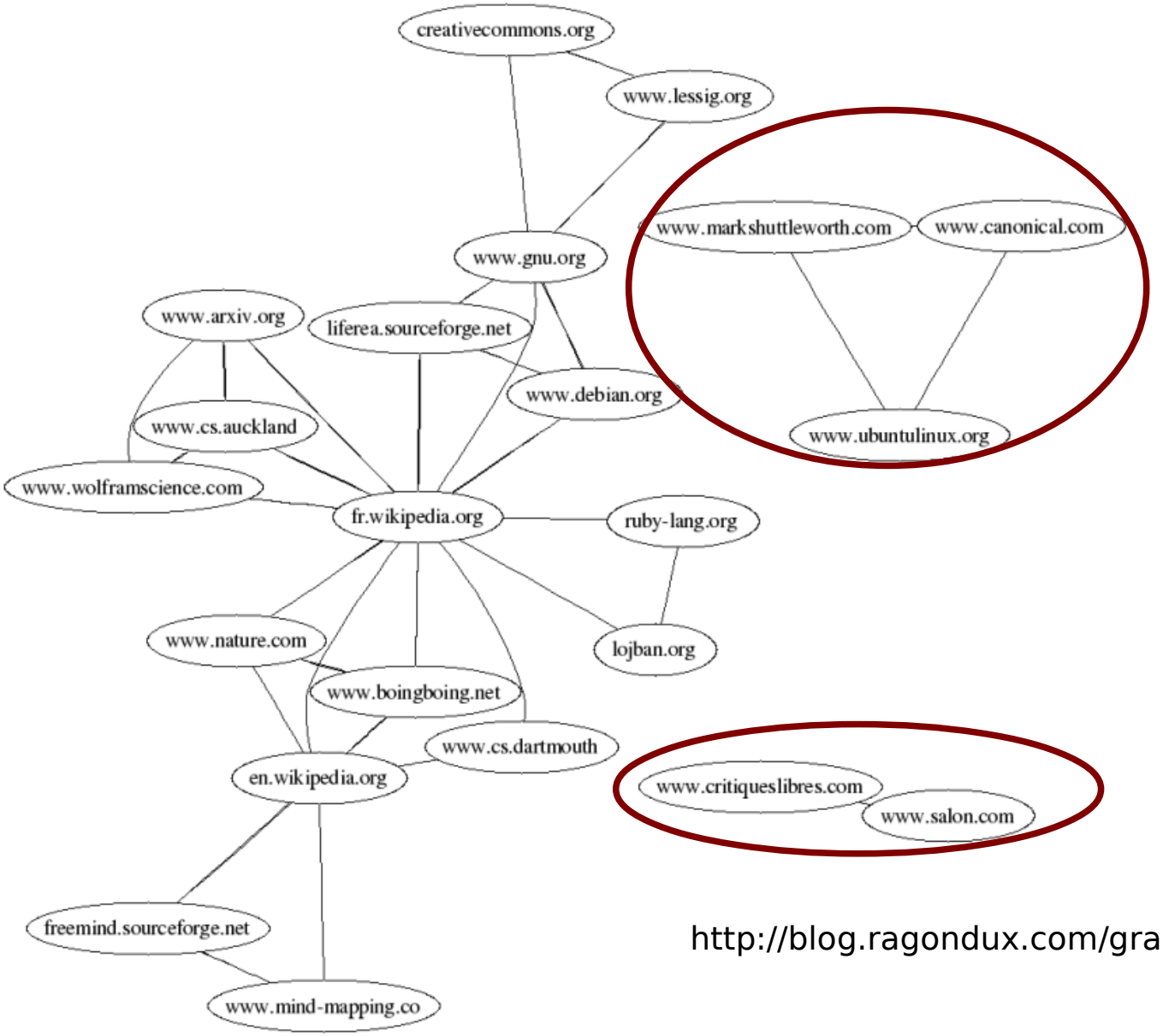
```
Profondeur2(Graphe G;
   Sommet S0 , entier NC)
    ...
    Visiter(V, G, NC)
    ...
        Profondeur2(G, S, NC)
```

Exercice


creativecommons.org
www.lessig.org
www.markshuttleworth.com
www.canonical.com
www.gnu.org
www.arxiv.org
liferea.sourceforge.net
www.debian.org
www.cs.auckland
www.ubuntulinux.org
www.wolframscience.com
fr.wikipedia.org
ruby-lang.org
www.nature.com
lojban.org
www.boingboing.net
www.cs.dartmouth
en.wikipedia.org
www.critiqueslibres.com
www.salon.com
freemind.sourceforge.net
www.mind-mapping.co

Exercice

- Le graphe n'est pas connexe !



http://blog.ragondux.com/graphes

Composantes fortement connexes

- Composante C **fortement** connexe $\Leftrightarrow$
  $\forall i, j \in C, \exists$ chemin $(i \rightarrow j)$ (donc **orienté**)

- $\neq$ composante connexe $\Leftrightarrow$
  $\forall i, j \in C, \exists$ chaine $(i \leftrightarrow j)$ (donc **non-orienté**)

- Composante fortement connexe de S_0 donné : ensemble X t.q
  - (1) $\forall S \in X$, chemin$(S_0 \rightarrow S)$
  - (2) $\forall S \in X$, chemin$(S \rightarrow S_0)$

Forte connexité - Algorithme

- Utilisation d'algorithmes de parcours orientés :
  - (1) $\Rightarrow$ algorithme basé sur les **successeurs**
  - (2) $\Rightarrow$ algorithme basé sur les **prédecesseurs**

(1)/(2)

```
ProfondeurSuccs(Graphe G;  Sommet S0)
    ... adjacents ← Sucesseurs(V, G)  ...
```

- Algorithme :

```
FConnexité(Graphe G, Sommet S0)(Liste)
    X1 ← ProfondeursSuccs(G,  S0)
    X2 ← ProfondeursPreds(G,  S0)
    retourner X1 ∩ X2
```

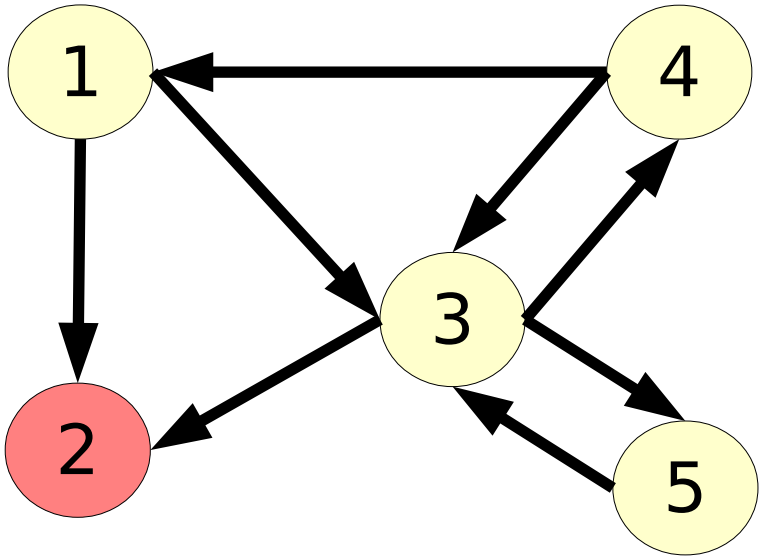
Forte connexité - Exemple

1. ProfondeurSuccs(G, S_1)

   $\rightarrow X_1 = \{S_1, S_2, S_3, S_4, S_5\}$

2. ProfondeurPreds(G, S_1)

   $\rightarrow X_2 = \{S_1, S_3, S_4, S_5\}$

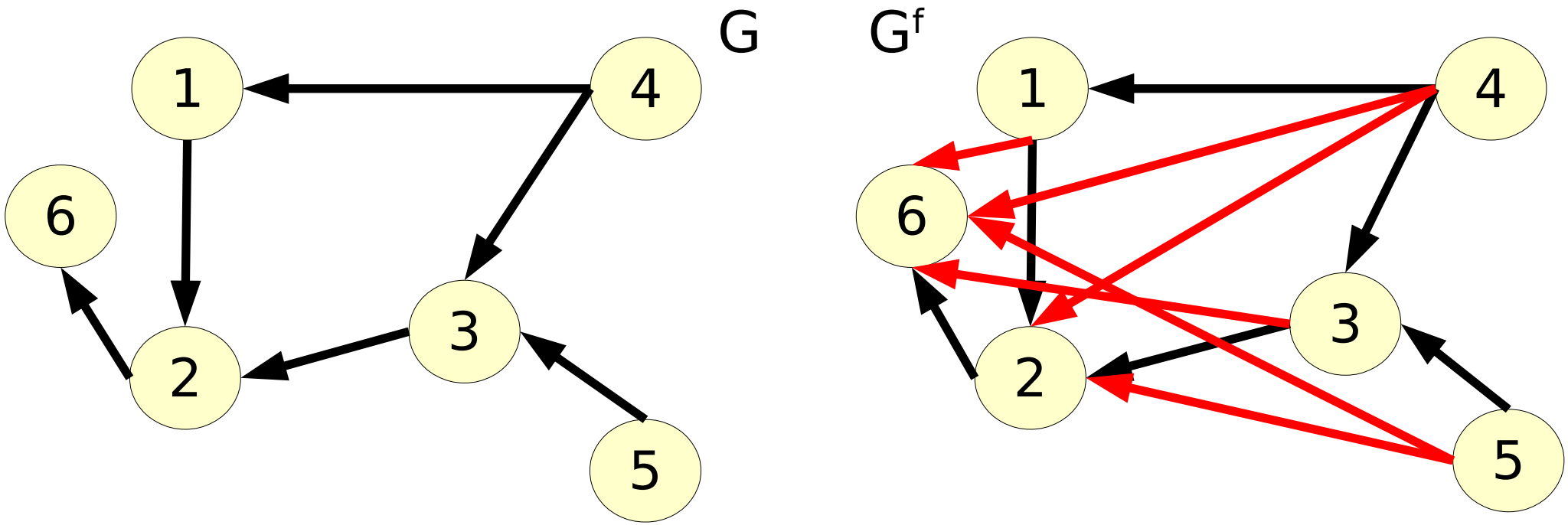


Composante fortement connexe associée à S_1

$$X_1 \cap X_2 = \{S_1, S_3, S_4, S_5\}$$

Fermeture transitive

- Pour un graphe G=(X,U), graphe G^f = (X,U') t.q U' = {u'=(Si→Sj) in U' | ∃ chemin $S_i \rightarrow S_j$ dans G}



- Si le graphe est fortement connexe, sa fermeture transitive est un graphe complet

Fermeture transitive
Algorithme de Roy-Warshall

- A partir de G = {X, U}, ajouter itérativement des arcs $S_i \rightarrow S_j$, si $S_i \rightarrow S_k$ et $S_k \rightarrow S_j$ existent
- A partir de la matrice d'adjacence du graphe

RoyWarshall(Graphe G=(X,U))(graphe G_f)

1-Initialisation

$M[i, j] \leftarrow$ vrai *ssi* e $S_i \rightarrow S_j \in G$

2-Itération courante

$\forall S_i \in X, \forall S_j \in X, \forall S_k \in X$

$M[i, j] \leftarrow M[i, j] \vee (M[i, k] \wedge M[k, j])$

Exercice

- Peut on aller dans toutes les stations malgré les travaux ?