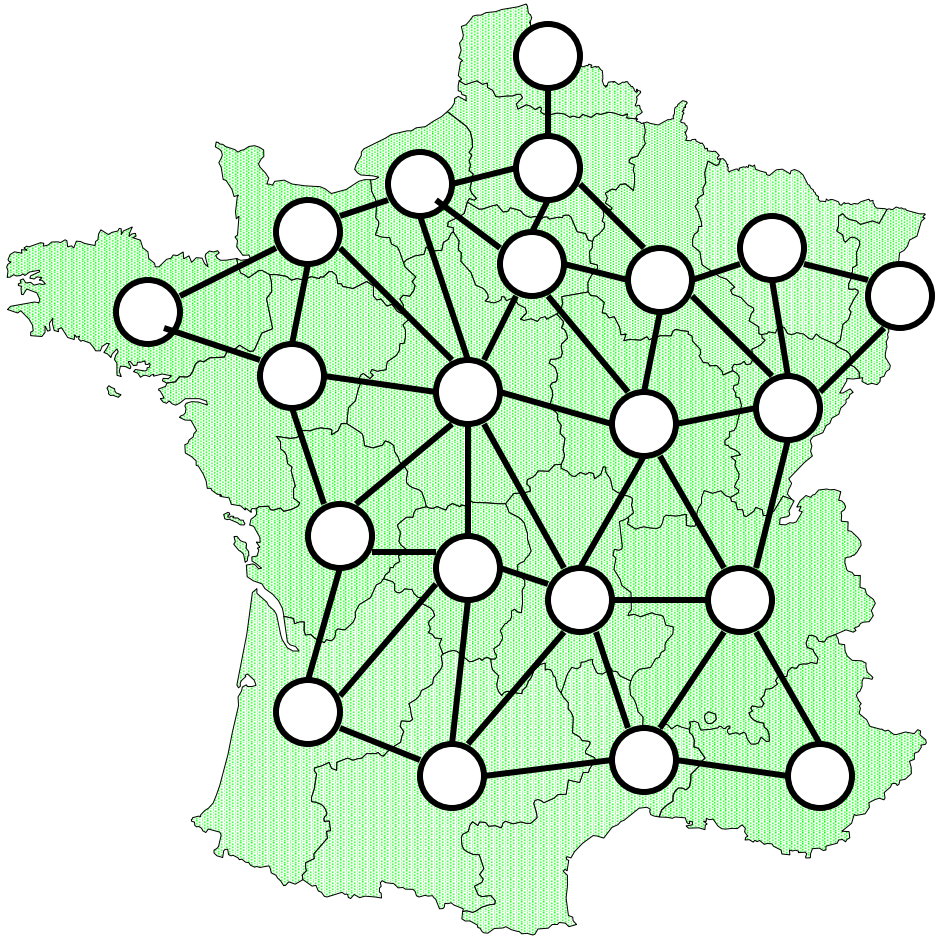
Algo Design

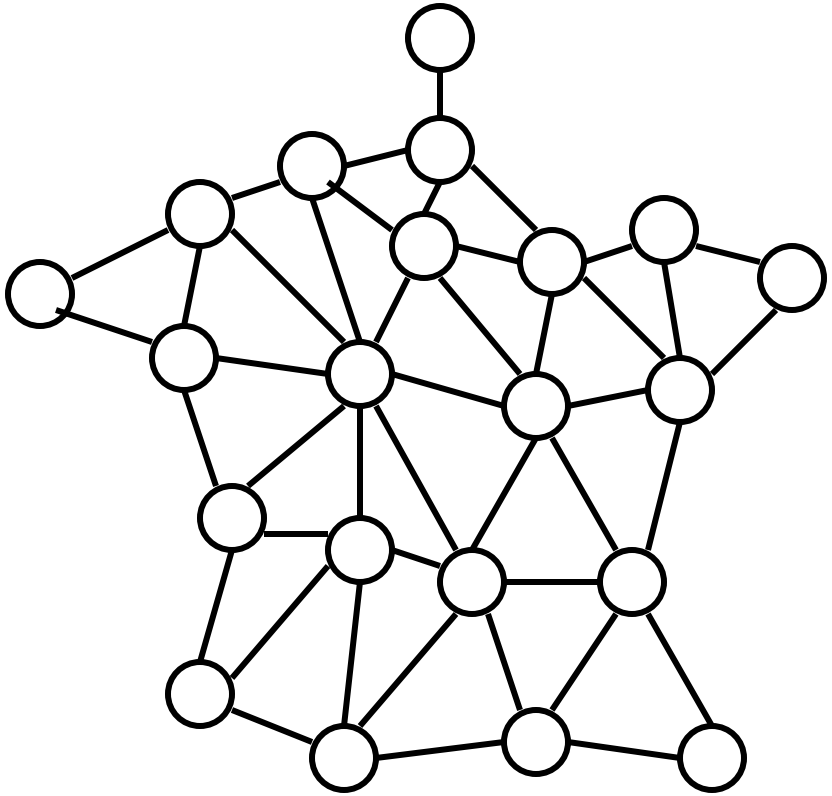
Cours 6

Graphs

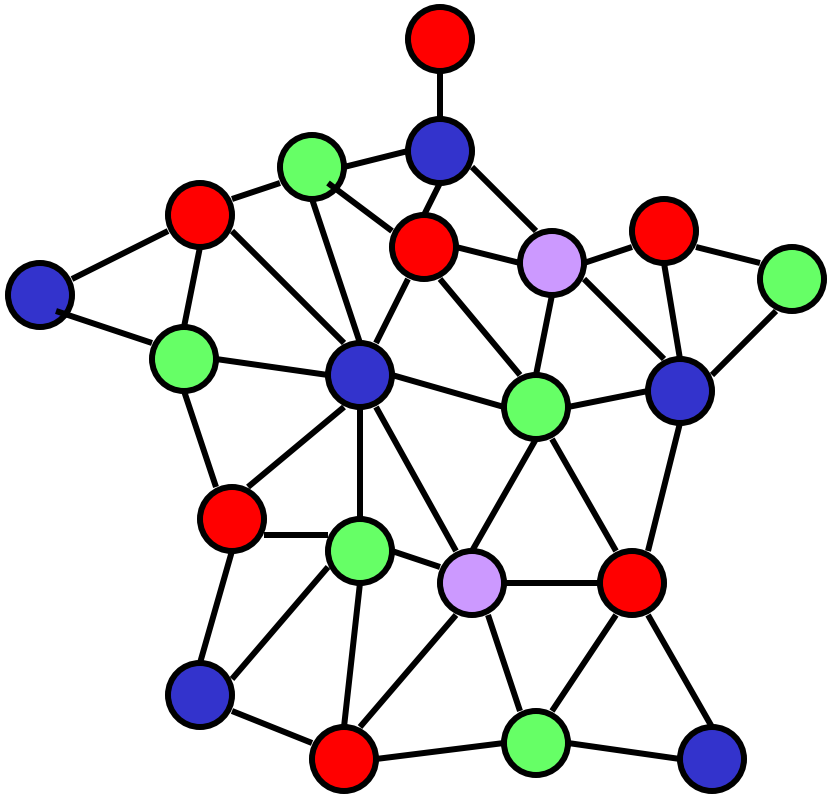
Graphes : motivation



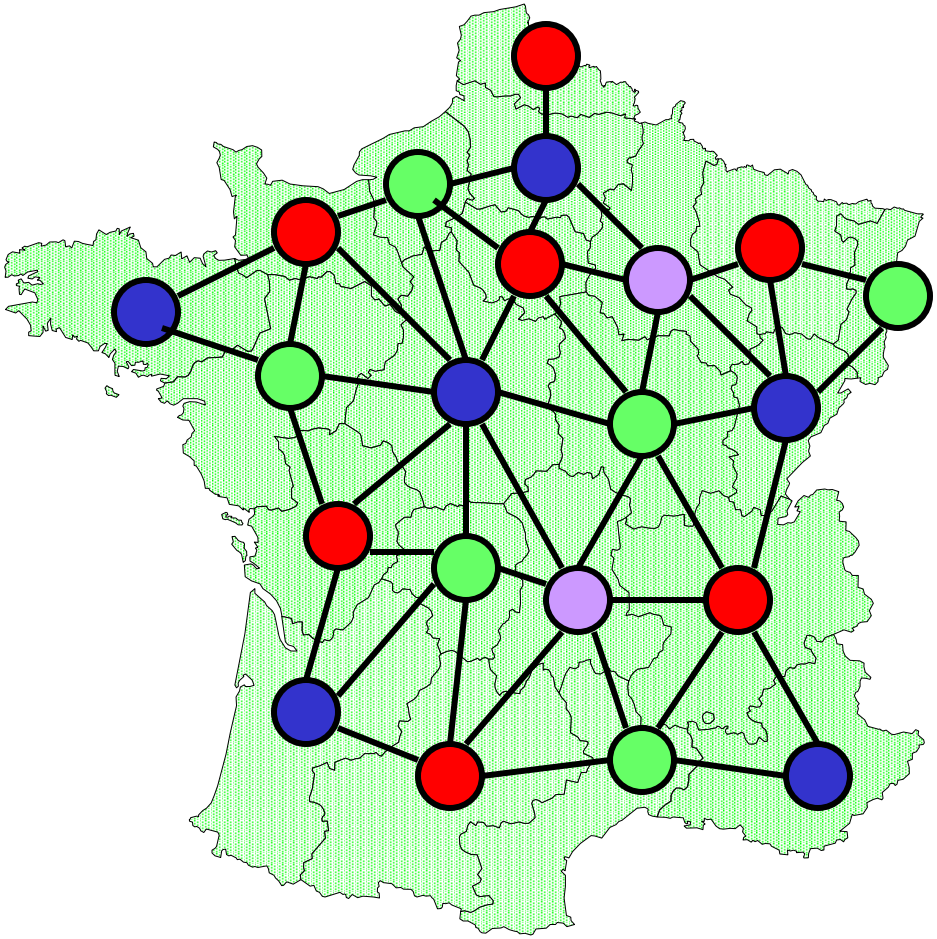
Graphes : motivation



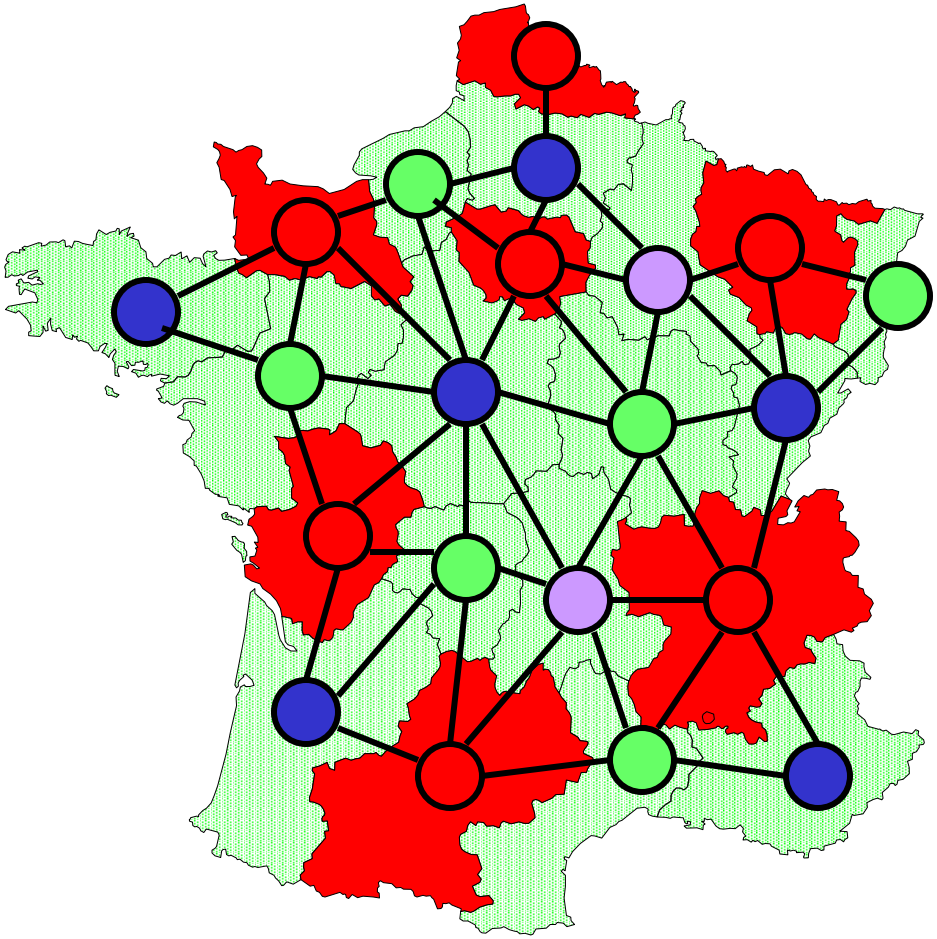
Graphes : motivation



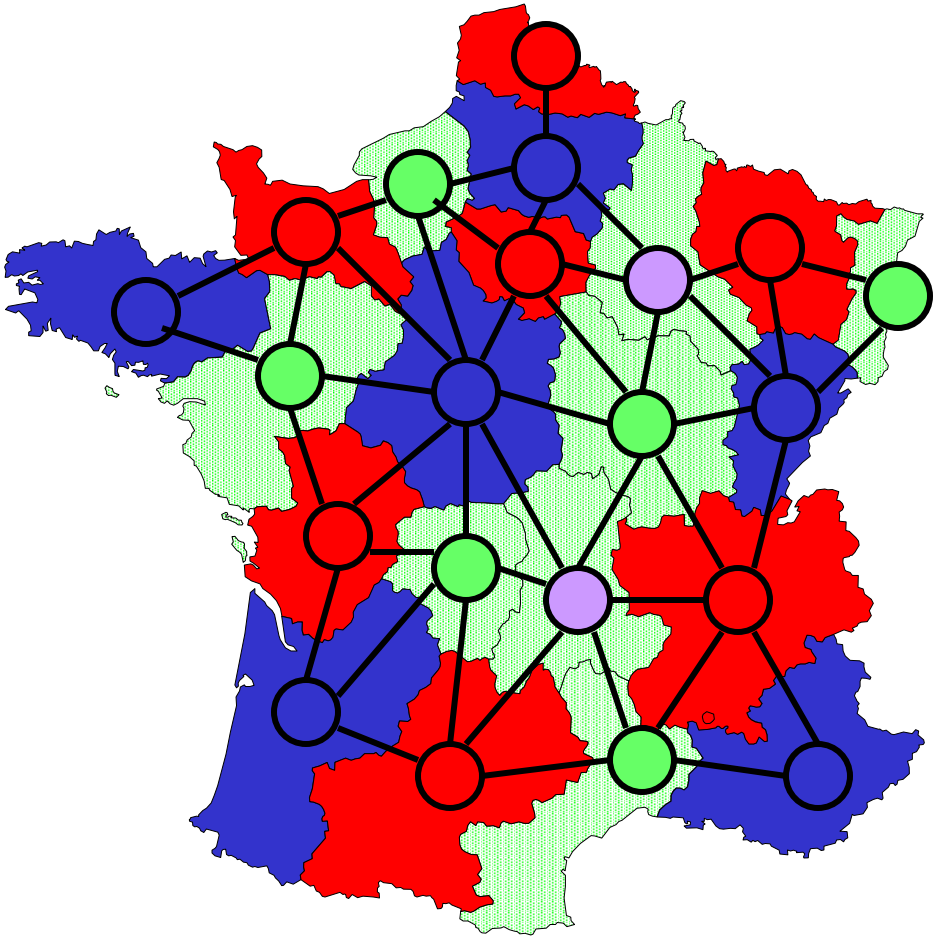
Graphes : motivation



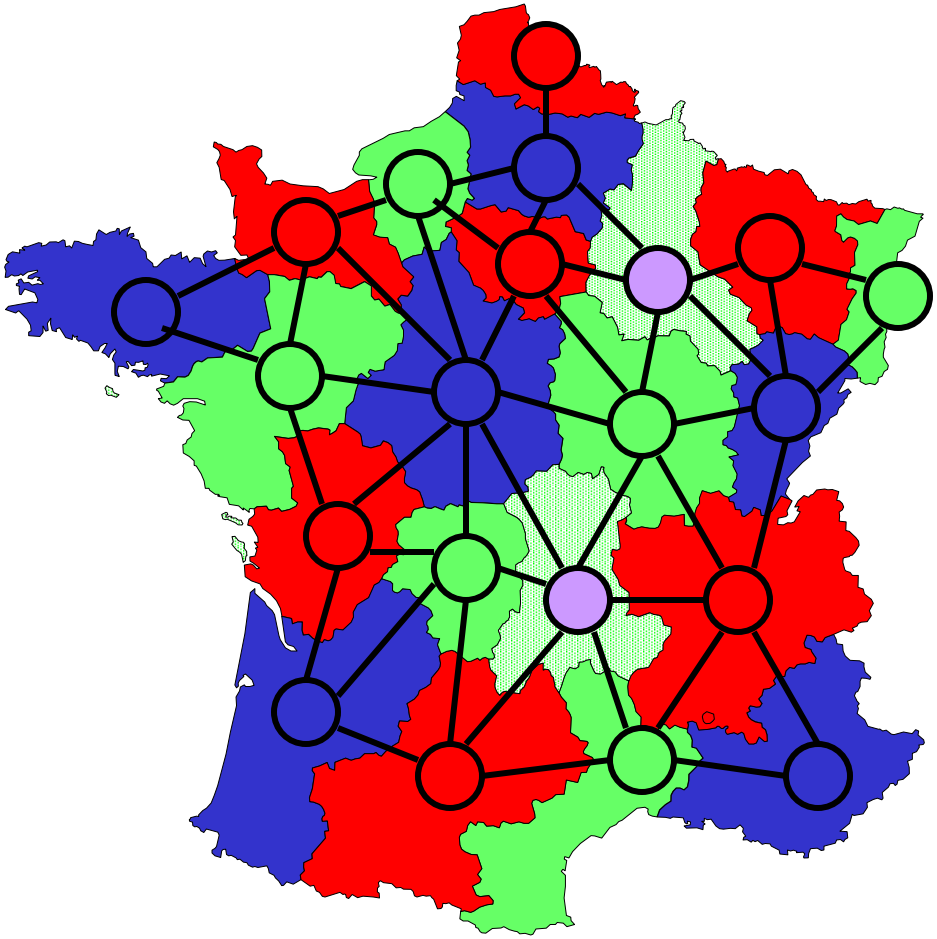
Graphes : motivation



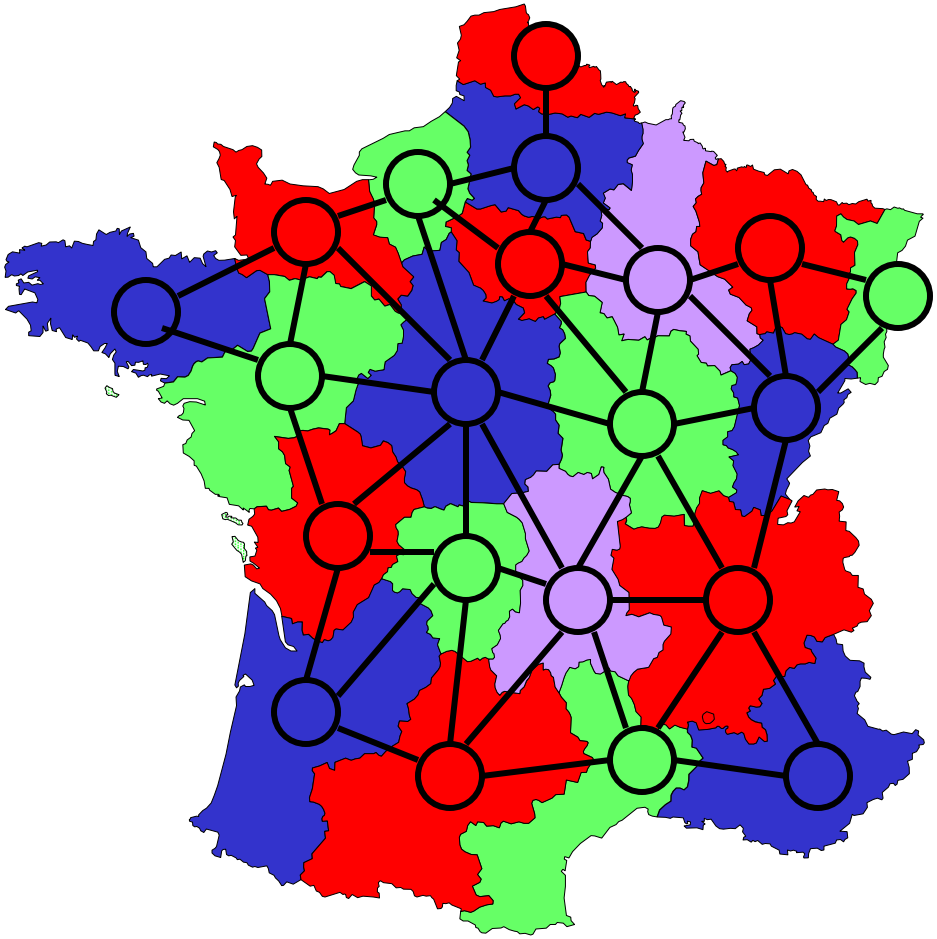
Graphes : motivation



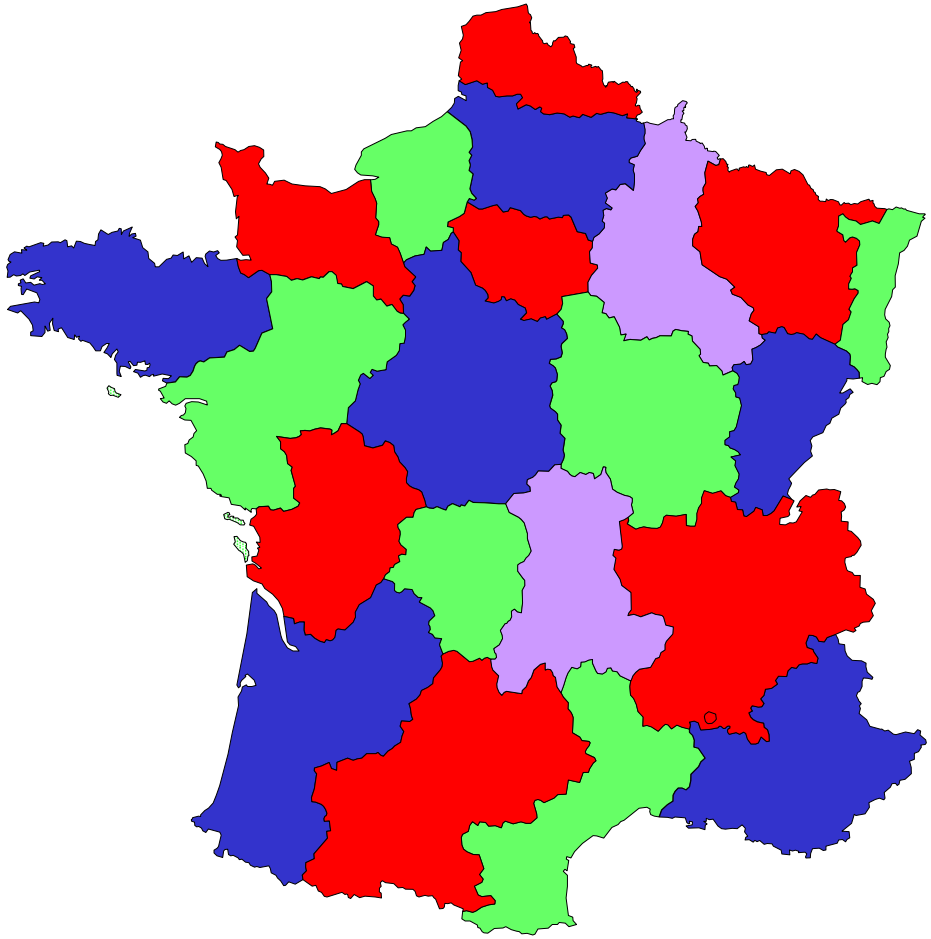
Graphes : motivation



Graphes : motivation



Graphes : motivation



Graphes : Définition

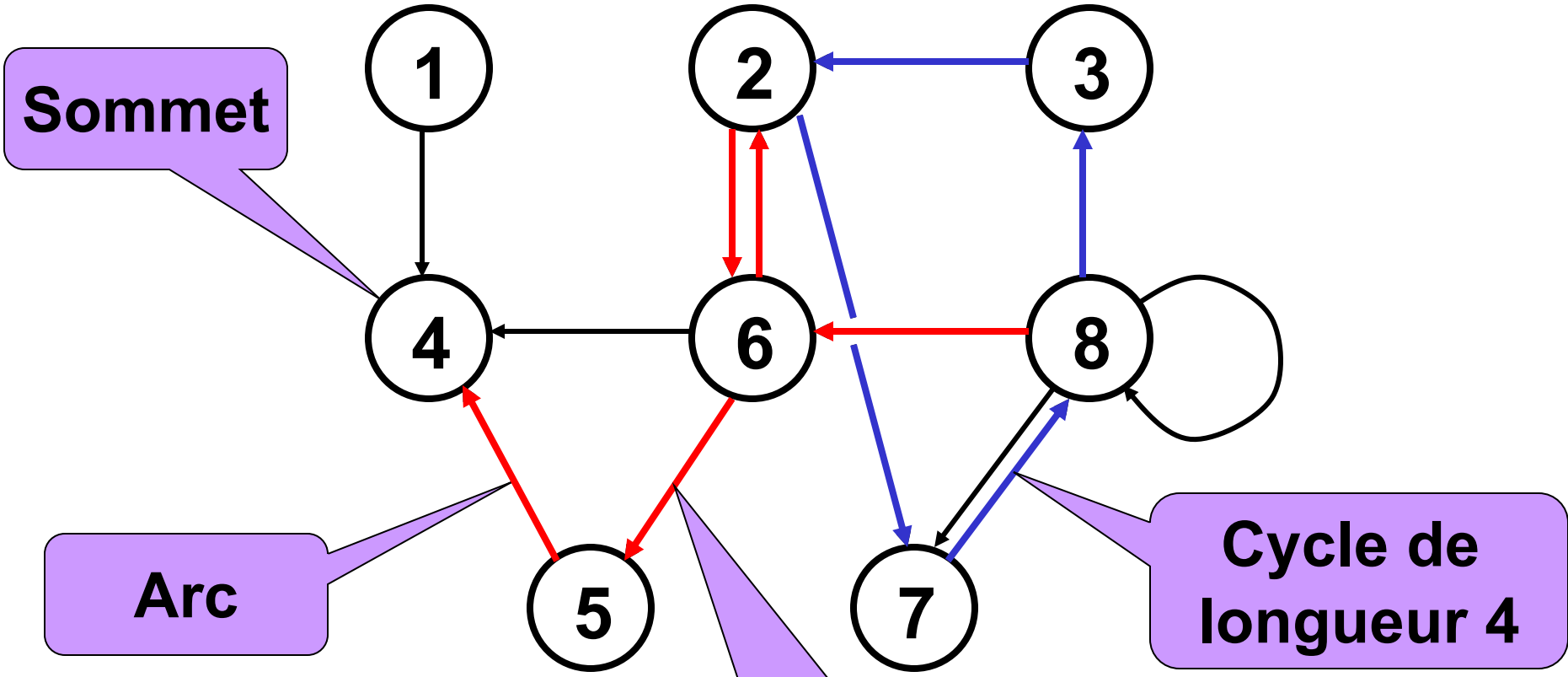
- Définition formelle :

  Un *graphe orienté* G est un couple *(S,A)*

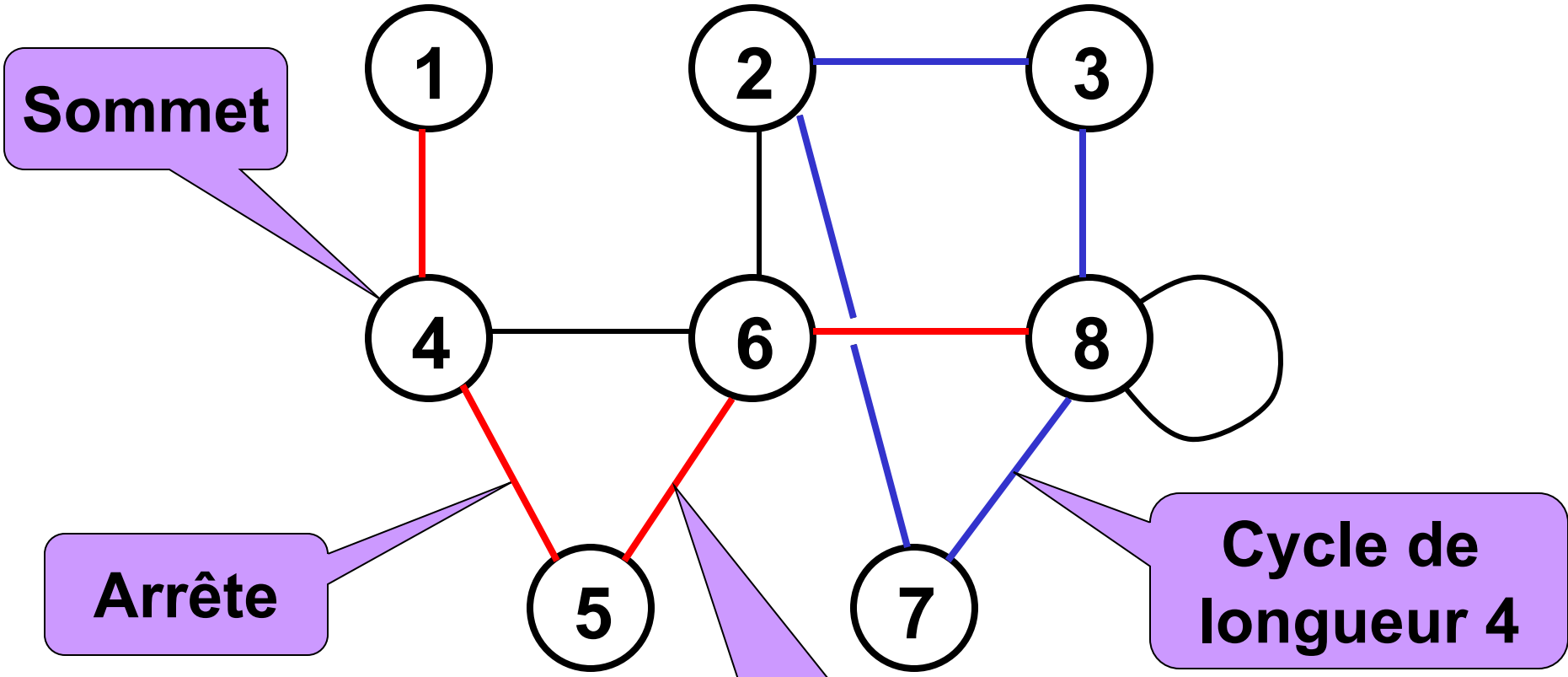
  où ***S*** est un ensemble fini de **sommets**

  et ***A*** un sous-ensemble de *SxS* **(arcs)**

Graphes orienté



Graphes non-orienté



Représentation

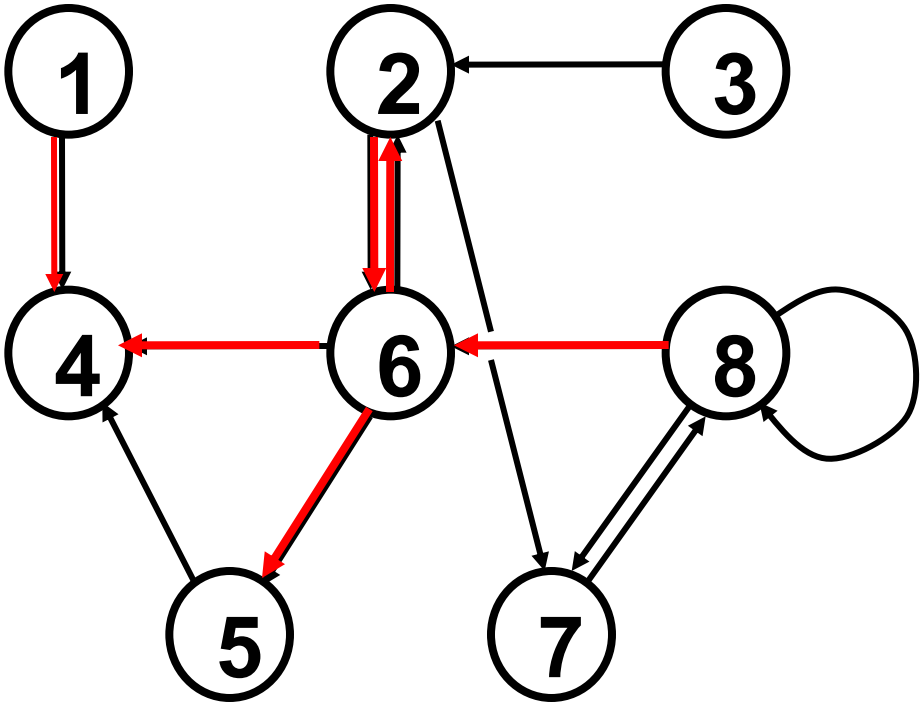
- Définition théorique $G=(S,A)$ *(pour les matheux)*
- Représentation graphique *(pour les humains)*
- **Matrice d'adjacence** *(pour les ordinateurs)*
  - si $G=(\{1,\ldots n\},A)$

    M matrice carrée d'ordre n telle que

    $M_{ij}=1$ si (i,j) est un arc

    $M_{ij}=0$ sinon
  - Complexité spatiale en $\theta(n^2)$

Matrice d'adjacence

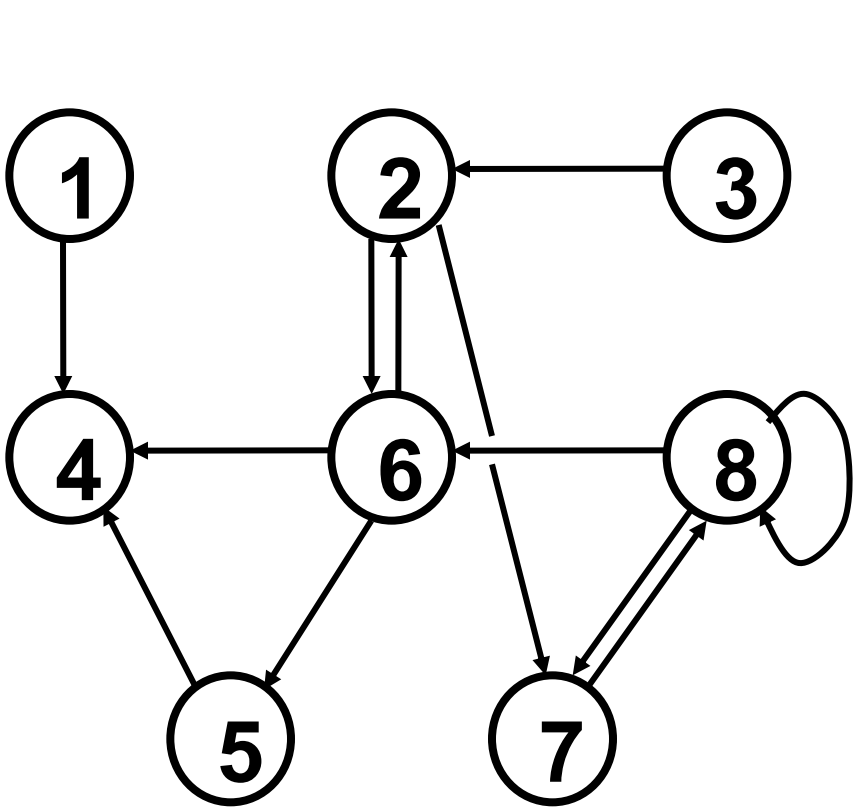


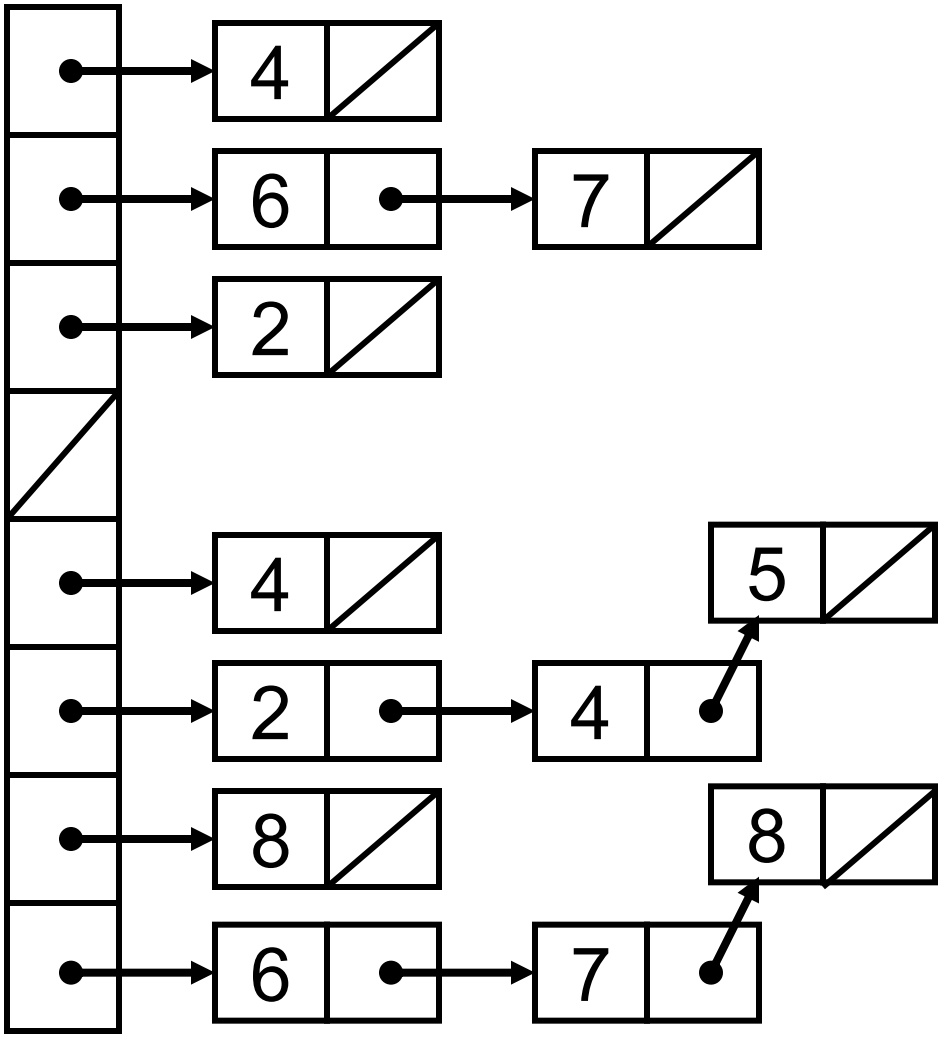
$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Représentation

- **Listes de successeurs**
  - tableau **T** de *n* listes chaînées
  - **T**[i] contient la liste des successeurs de *i*
  - Complexité spatiale $\theta(n+nb\ arcs)$

Listes de successeurs





Parcours de graphe

- Problème :

  comment parcourir un graphe, i.e.

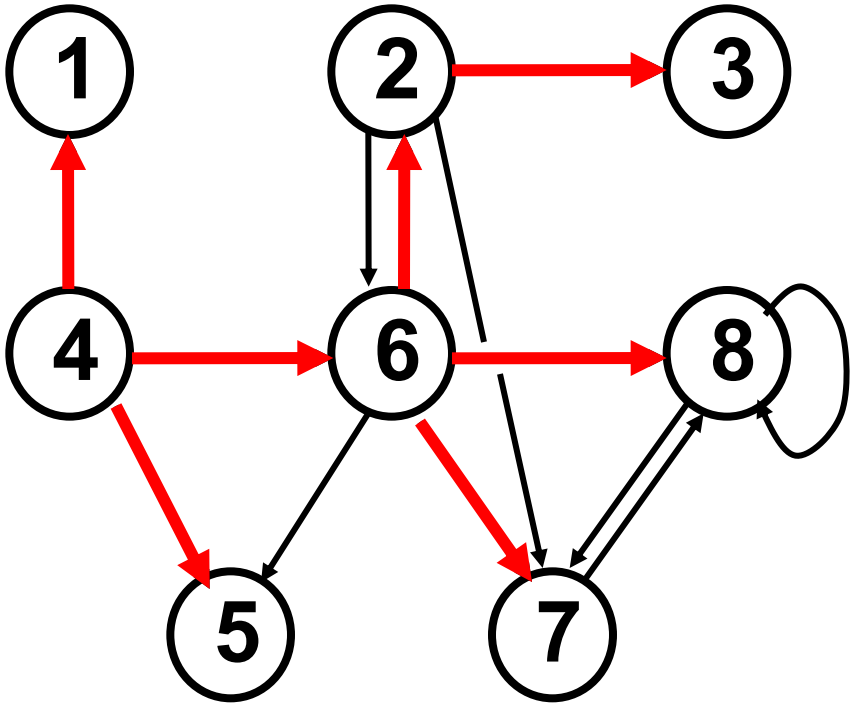
  énumérer ses sommets ?

  (sans se perdre…!)

- Applications :
  - Labyrinthes
  - Ordonnancement de tâches
  - Recherche des plus courts chemins

Parcours de graphe

- Problème :comment parcourir un graphe, i.e. **énumérer ses sommets** ?
- **But : Produire une « *forêt couvrante* »**

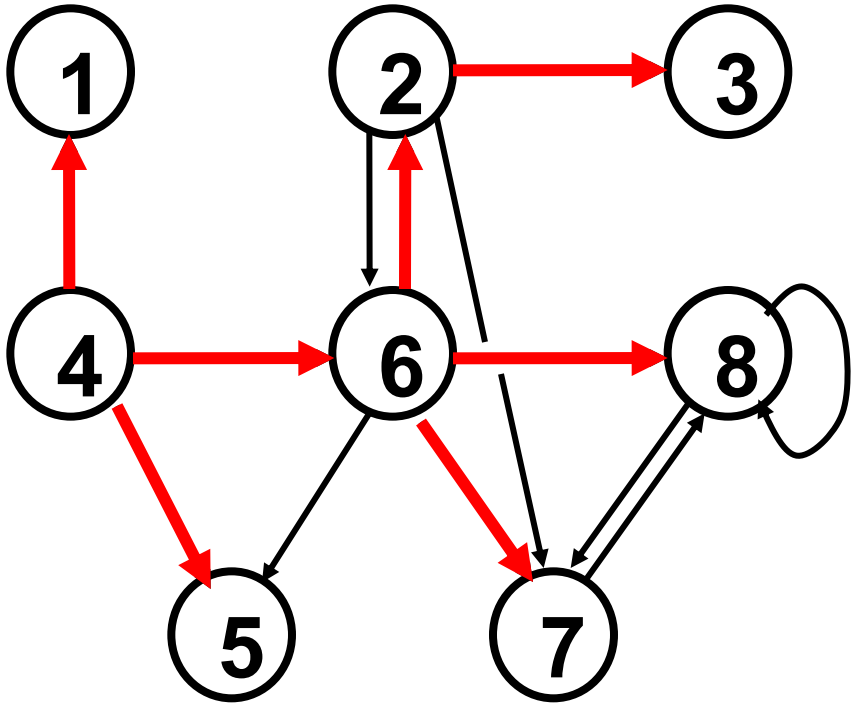


Forêts : définition

- Donnée : un graphe G=({1,…n},A)
- Une **forêt** est un graphe dans lequel chaque sommet a **au plus** un prédécesseur (son père)
- Une forêt peut être vue comme un ensemble d'arbres

Forêts : codage

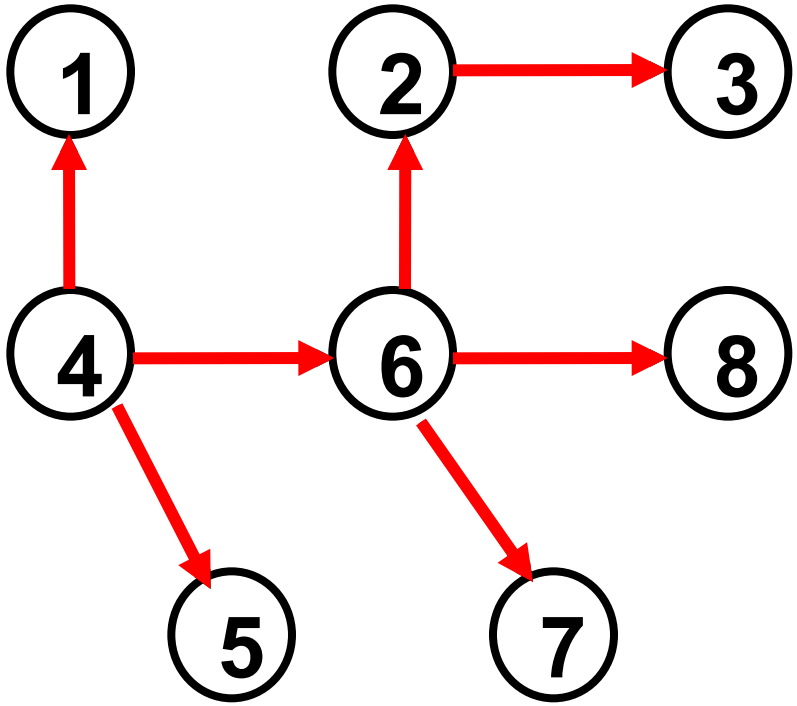
- un **tableau** T[1…n] contient le père de chaque sommet et ***vide*** pour les racines



1	2	3	4	5	6	7	8
4	6	2	V	4	4	6	6

Forêts : codage

- un **tableau** T[1…n] contient le père de chaque sommet et ***vide*** pour les racines



1	2	3	4	5	6	7	8
4	6	2	V	4	4	6	6

Parcours de graphe

- Problème :

  comment parcourir un graphe, i.e.

  énumérer ses sommets ?

- **But : Produire une « *forêt couvrante* »**

- Parcours *en* ***largeur*** *d'abord*
- Parcours *en* ***profondeur*** *d'abord*

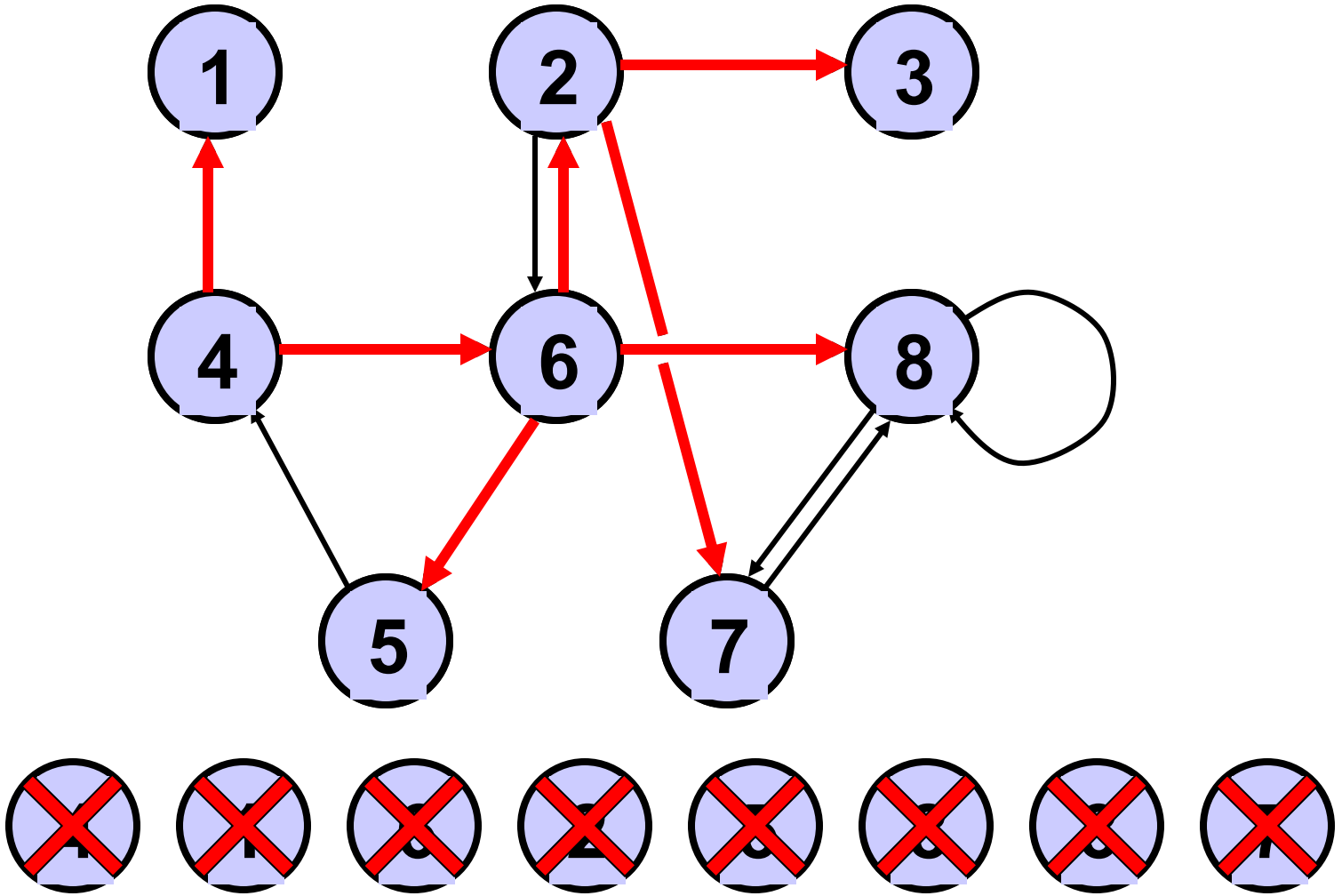
Parcours en Largeur

- Parcours à partir d'un sommet ***s***
- Règle : « *on cherche à atteindre tous les voisins avant de s'écarter* »
- **Initialisation** :
  - tous les sommets sont **blancs**
  - à chaque sommet i est associé sa distance d[i] au sommet *s*; (initialement d[i] est infini)
  - le sommet ***s*** est **bleu**, d[***s***]=0, T[***s***]=vide

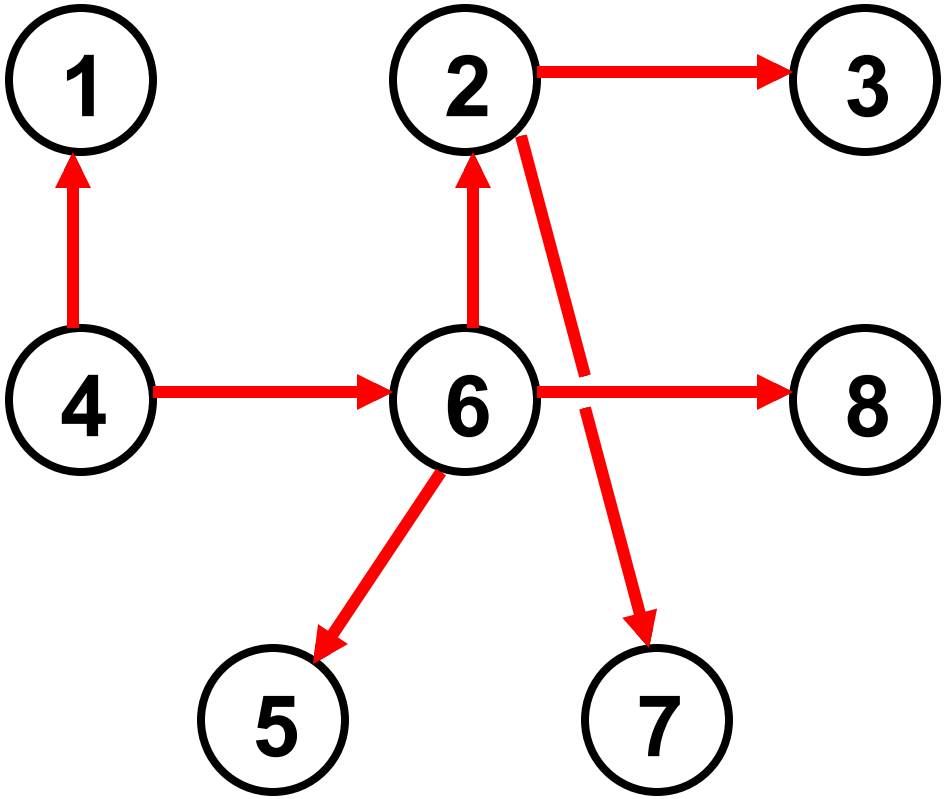
Parcours en Largeur

- F est une **file** qui contient initialement le sommet de départ ***s***
- tant que F n'est pas vide
  - soit ***u*** le premier sommet de F
  - pour tout ***v*** voisin blanc de u
    - colorier ***v*** en **bleu**
    - d[***v***]=d[***u***]+1
    - T[***v***]=***u***
    - ajouter ***v*** dans F

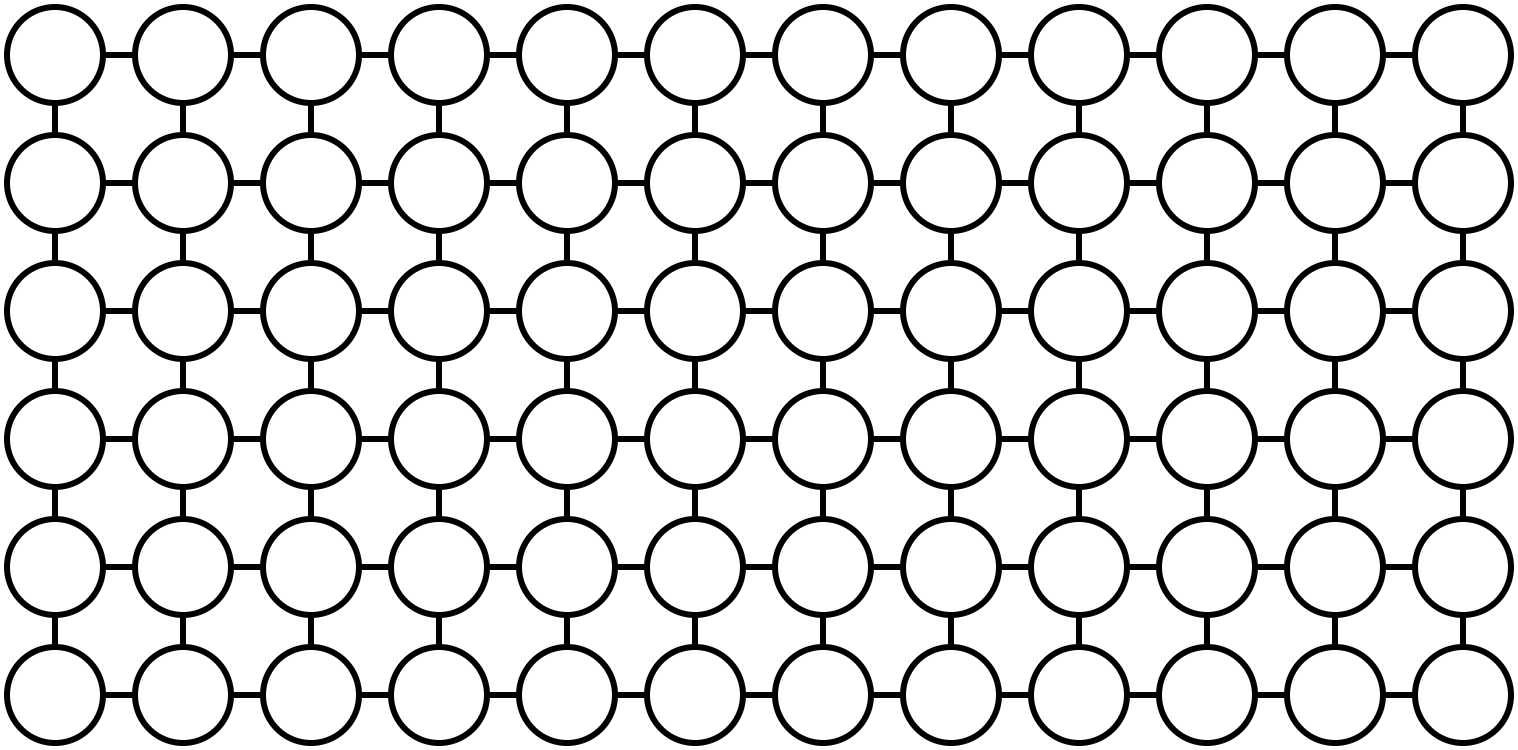
Parcours en Largeur



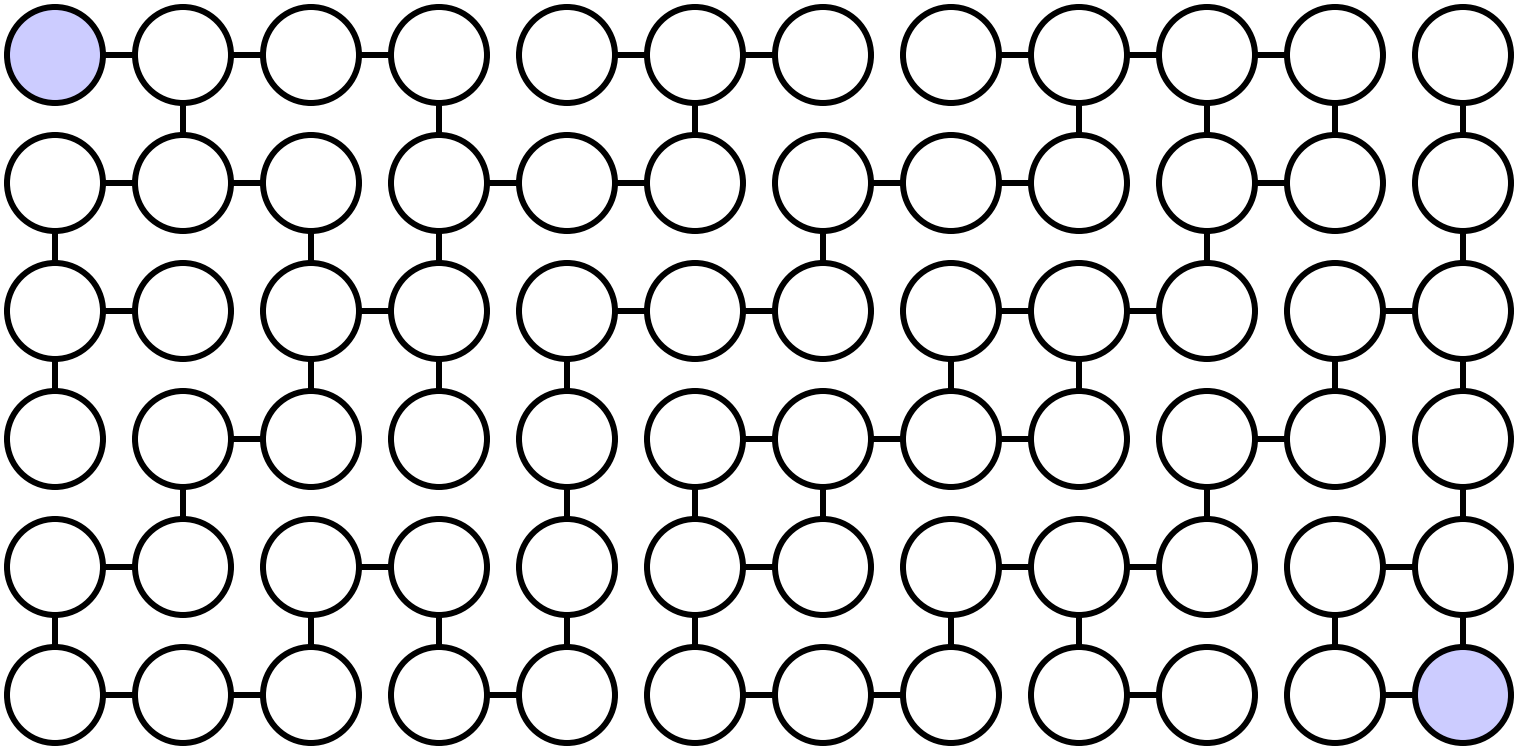
Parcours en Largeur



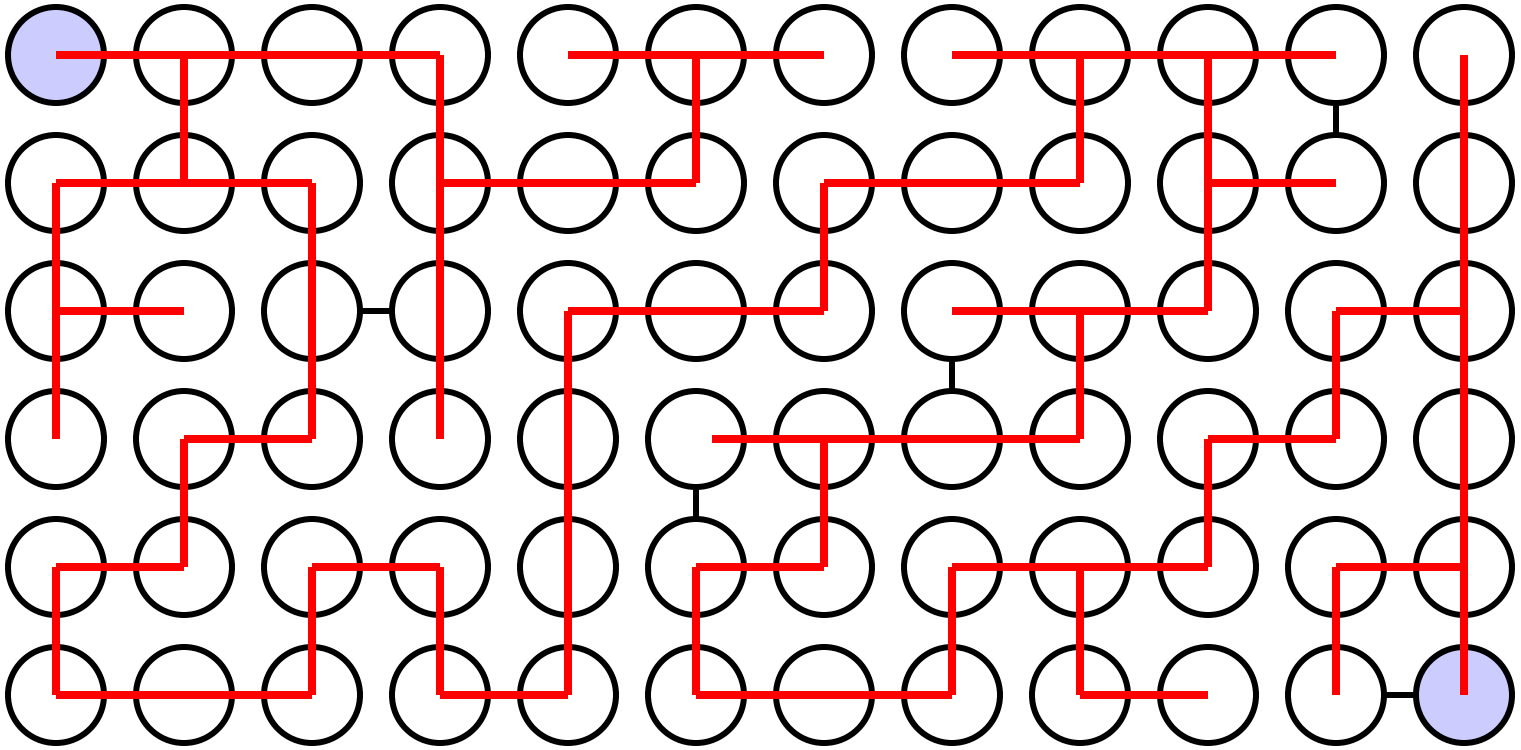
Application : labyrinthe



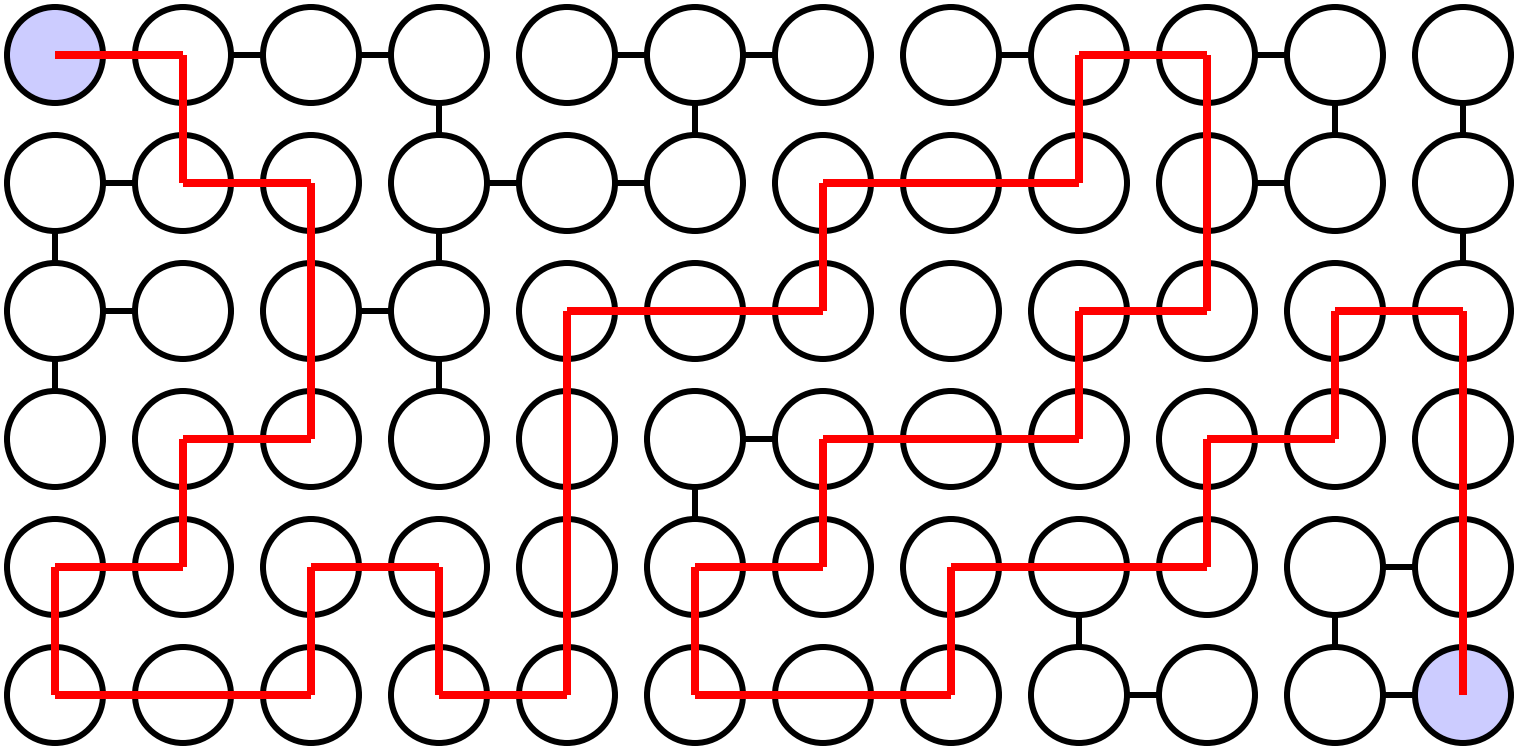
Application : labyrinthe



Application : labyrinthe



Application : labyrinthe



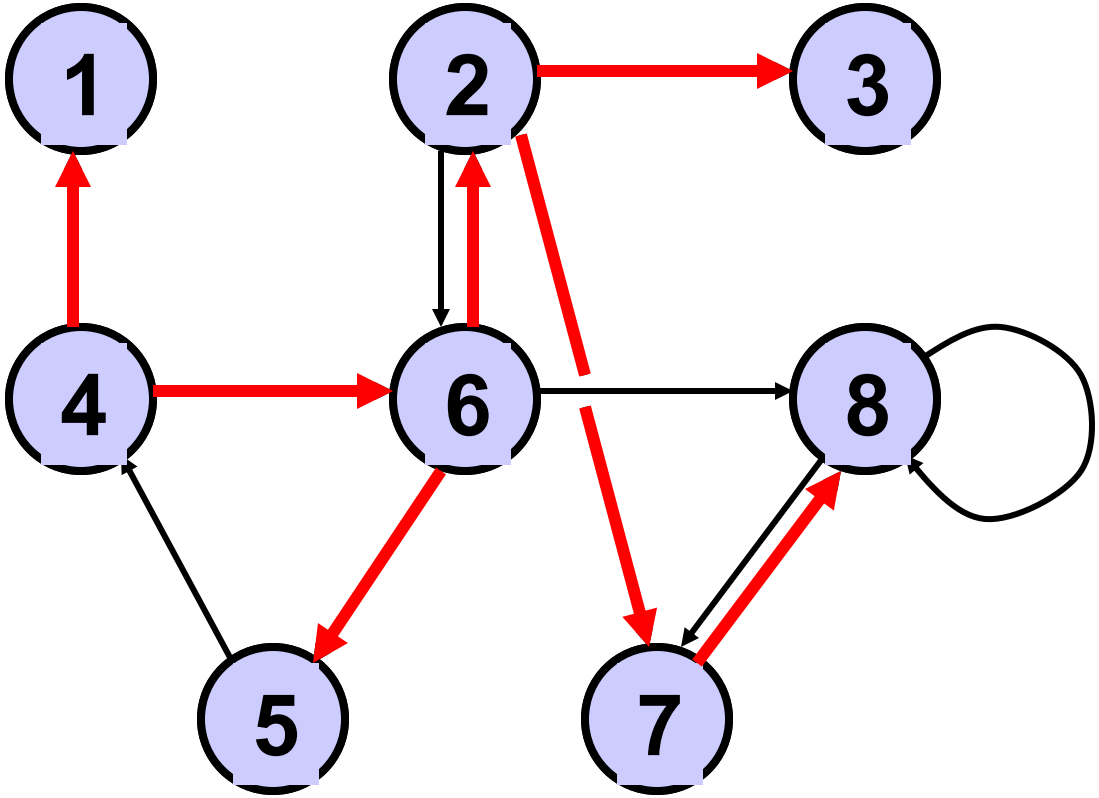
Parcours en Profondeur

- Règle : « *on cherche à aller le plus en profondeur puis on remonte* »
- Initialisation :
  - tous les sommets sont **blancs**
  - à chaque sommet **i** est associé sa date de début de traitement **d**[i] ainsi que sa date de fin de traitement **f**[i]
  - tous les T[i] sont vides
  - un chronomètre **D** est mis à zéro

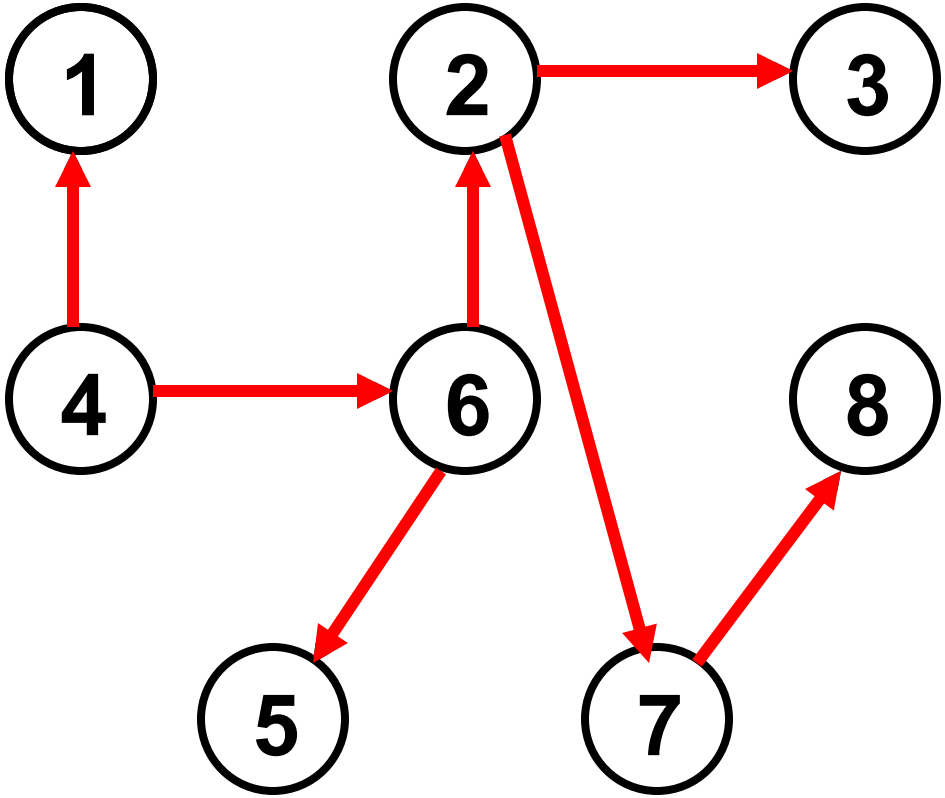
Parcours en Profondeur

- Parcours à partir d'un sommet ***s*** :
  - colorier ***s*** en **bleu**
  - **D=D+1**; **d[*s*]=D**
  - pour tout ***v*** voisin **blanc** de ***s***
    - T[***v***]=***s***
    - Parcourir le graphe en profondeur à partir de ***v***
  - **D=D+1**; **f[*s*]=D**

Parcours en Profondeur



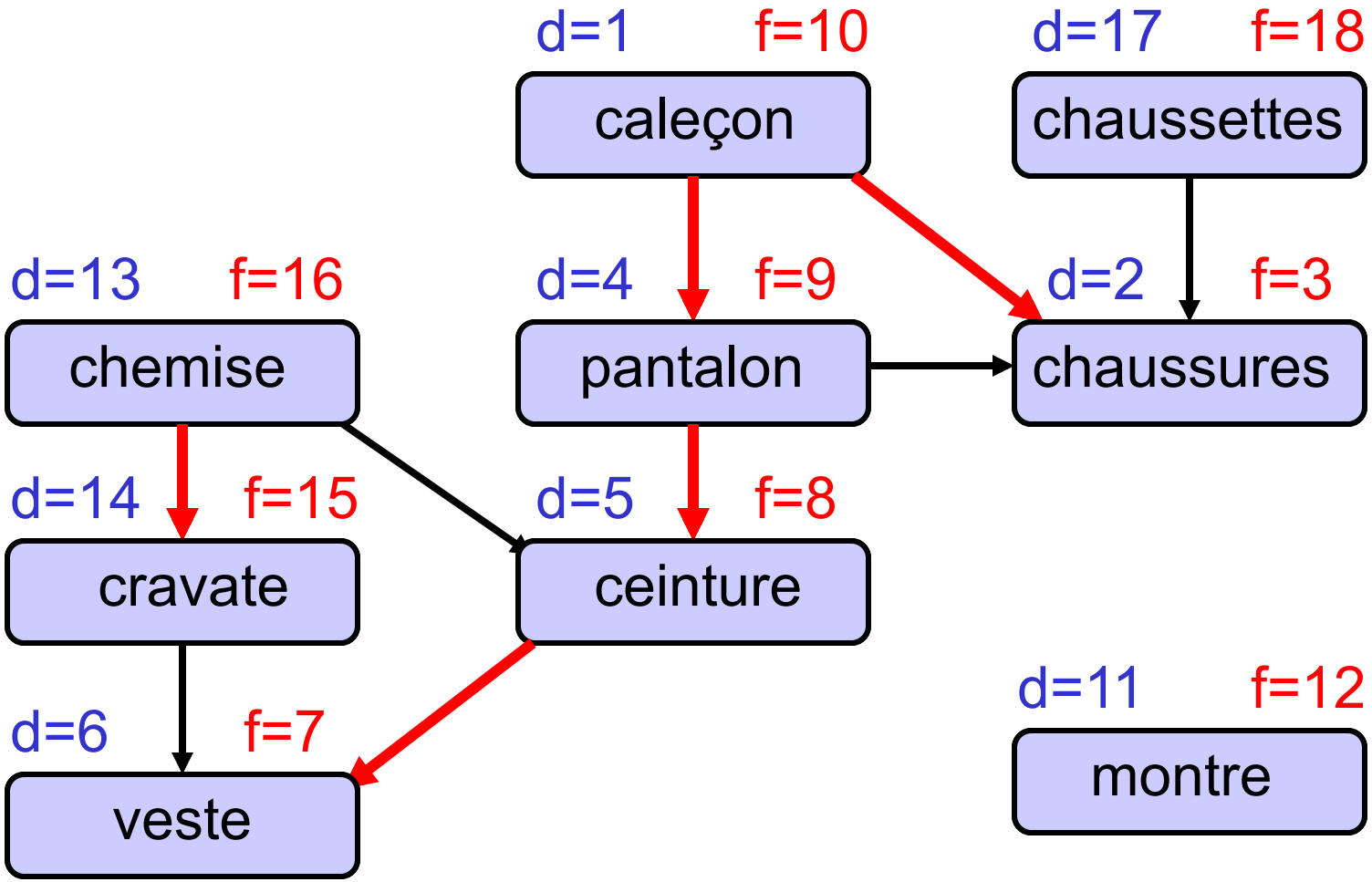
Parcours en Profondeur



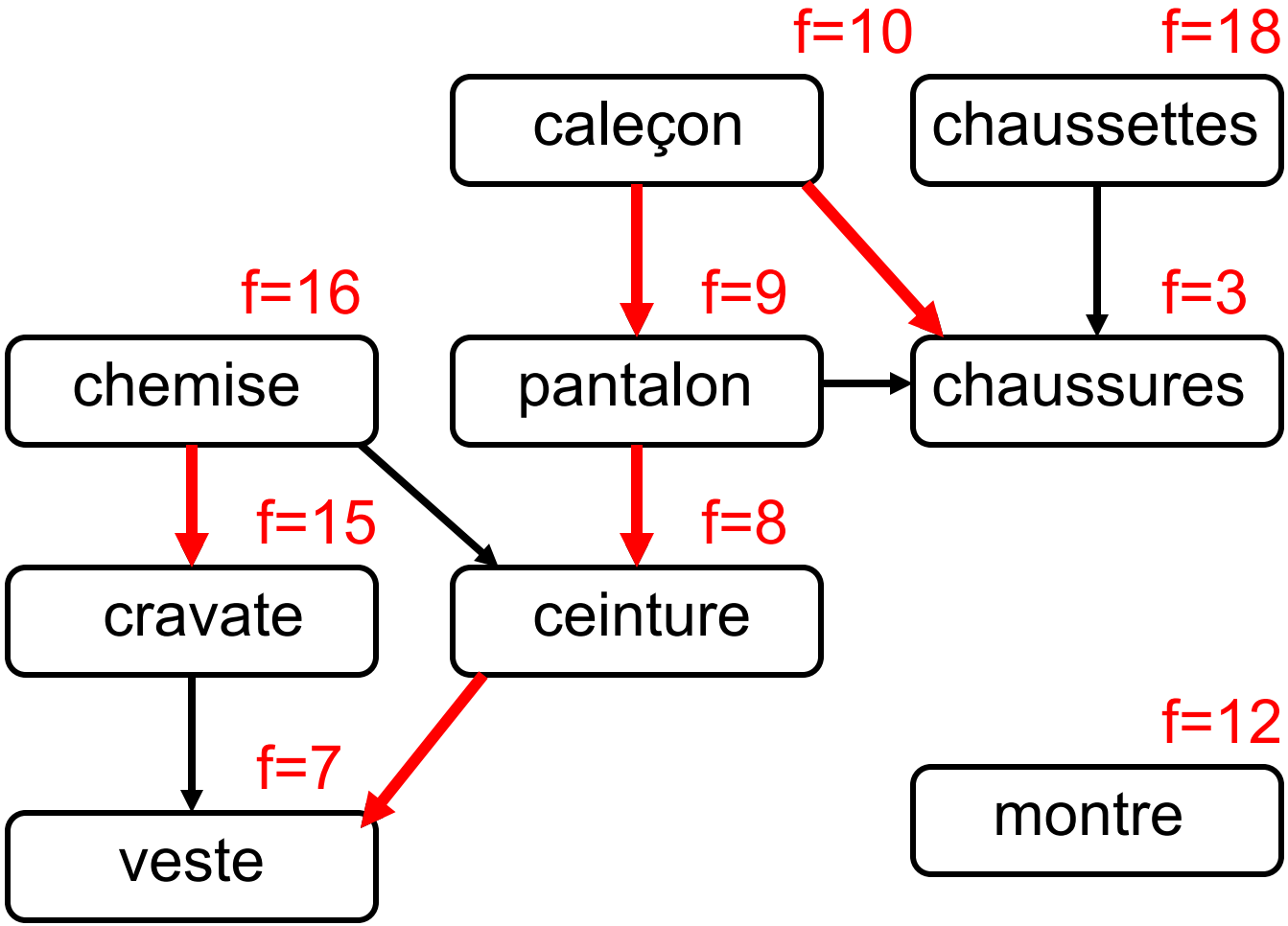
Tri topologique

- Des tâches doivent être réalisées mais certaines doivent l'être avant d'autres
- Exemple : ordonnancement de tâches
- Solution en temps linéaire
  - Effectuer un **parcours en profondeur d'abord**
  - Retourner la liste des sommets dans l'ordre décroissant des dates de fin de traitement

Tri topologique



Tri topologique



chaussettes
chemise
cravate
montre
caleçon
pantalon
ceinture
veste
chaussures

Chemins les plus courts

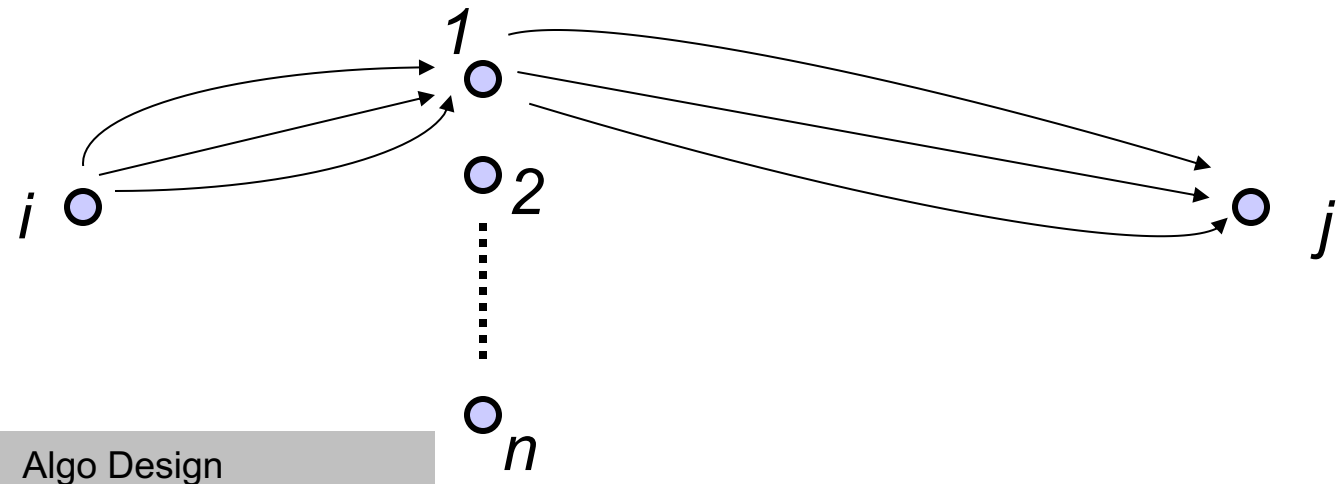
- Théorème :
  Soit $G=(S,A)$ un graphe de matrice d'adjacence **M**;
  le nombre de chemins de longueur k reliant i à j est exactement $(M^k)_{ij}$

- Preuve : par récurrence

Chemins les plus courts

- Théorème :
  Soit *G=(S,A)* un graphe de matrice d'adjacence **M**;
  le nombre de chemins de longueur k reliant *i* à *j* est exactement $(M^k)_{ij}$

- Preuve : $(M^k)_{ij} = (M^x)_{i1} \times (M^{k-x})_{1j} + (M^x)_{i2} \times (M^{k-x})_{2j} + \ldots$



Chemins les plus courts

- Corrolaire :

  Soit $N=M+M^2+\ldots+M^n$

  Il existe un chemin reliant *i* à *j* **ssi**

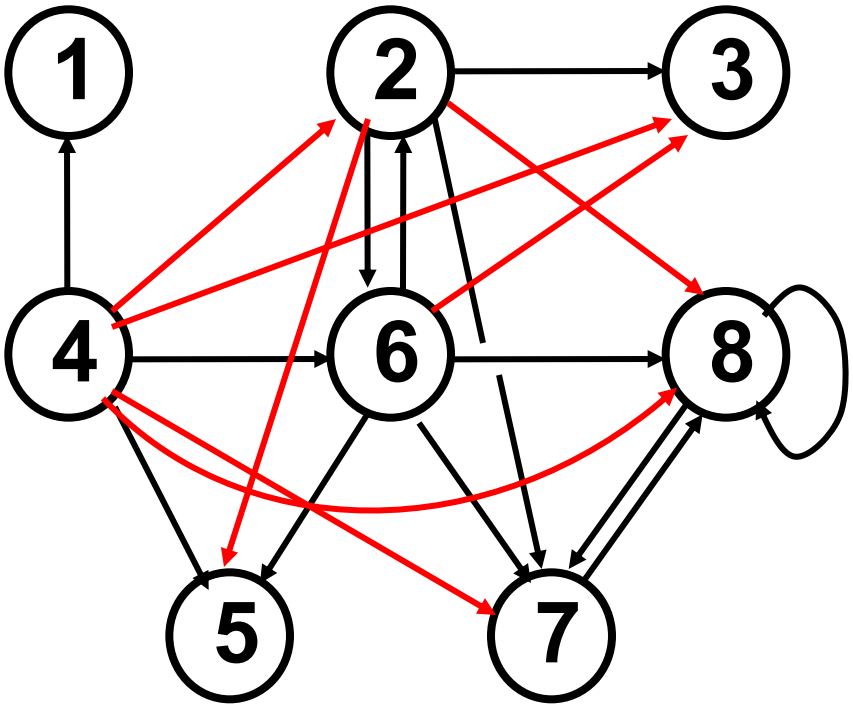
  N_{ij} est non-nul

Fermeture transitive

- La matrice d'adjacence de la **fermeture transitive** du graphe de matrice M est déduite de

  $N = Id + M + M^2 + \ldots + M^n$

- Complexité : $\Theta(n^4)$



Fermeture efficace

- Soit **N** la matrice identique à **M** mais avec des 1 sur la diagonale
- **pour *k* variant de *1* à *n* faire**
  - **pour *i* variant de *1* à *n* faire**
    - **pour *j* variant de *1* à *n* faire**
      - $N_{ij}=N_{ij}$ **ou** (N_{ik} **et** N_{kj})
- Complexité : $\Theta(n^3)$

Aho, Hopcroft et Ullman

- On généralise la notion de matrice d'adjacence
- Soit **N** la matrice identique à **M** mais avec des 0 sur la diagonale
- **pour *k* variant de *1* à *n* faire**
  - **pour *i* variant de *1* à *n* faire**
    - **pour *j* variant de *1* à *n* faire**
      - N_{ij}=**min**(N_{ij}, N_{ik} + N_{kj})
- Complexité : $\Theta(n^3)$

Travaux Dirigés

Implémentation de

l'algorithme de

Aho, *Hopcroft* et *Ullman*

(et application à la spéculation monétaire…)