Inf 431 – Cours 3

Connexité 2
Plus courts chemins

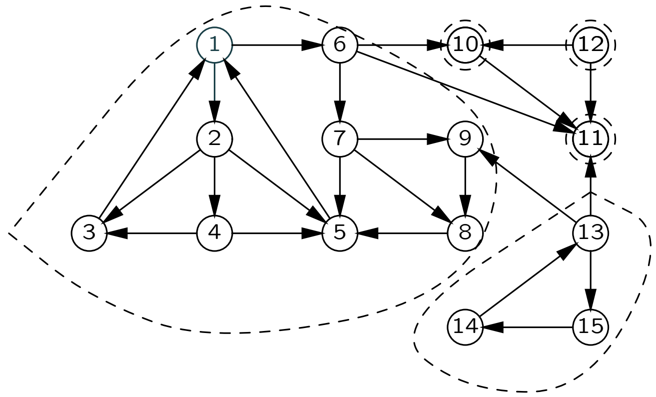
jeanjacqueslevy.net

secrétariat de l'enseignement:
Catherine Bensoussan
cb@lix.polytechnique.fr
Aile 00, LIX,
01 69 33 34 67

www.enseignement.polytechnique.fr/informatique/IF

Connexité forte (1/11)

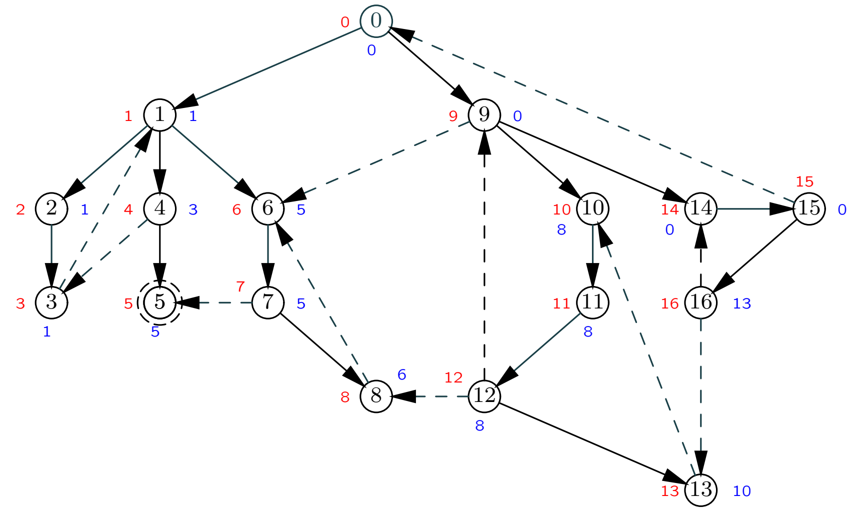
Composantes fortement connexes = parties maximales où toute paire de sommets distincts est reliée par un chemin.



Plan

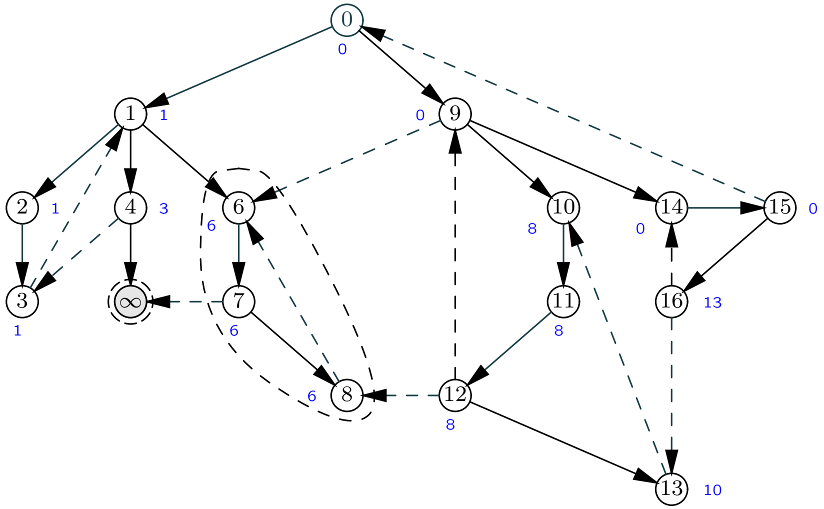
1. Composantes fortement connexes – Tarjan
2. Fermeture transitive – Warshall
3. Plus courts chemins
4. Plus court chemin – Dijkstra

Connexité forte (2/11)



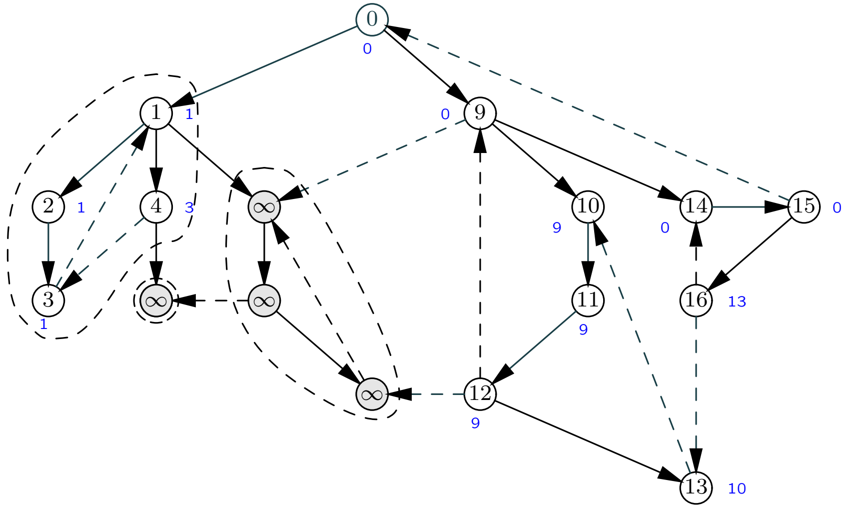
(point d'attaches, cf. biconnexité)

Connexité forte (3/11)



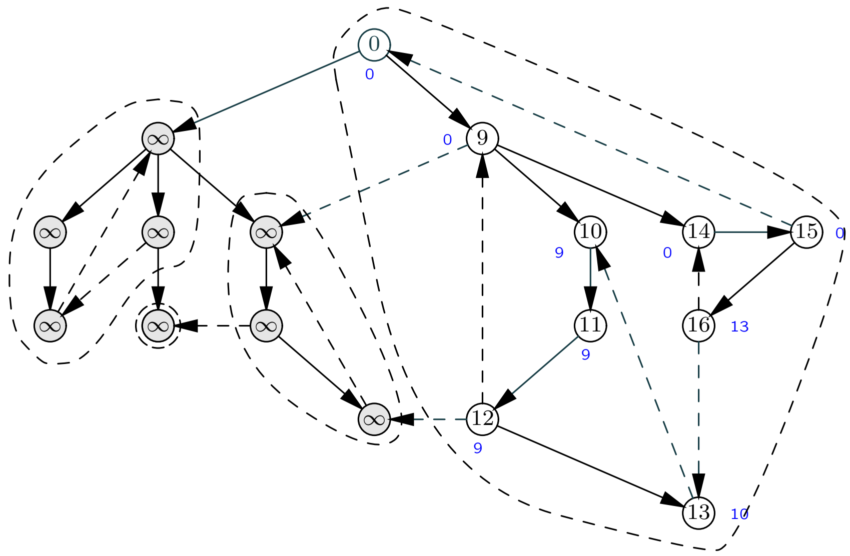
(point d'attaches, cf. biconnexité)

Connexité forte (4/11)



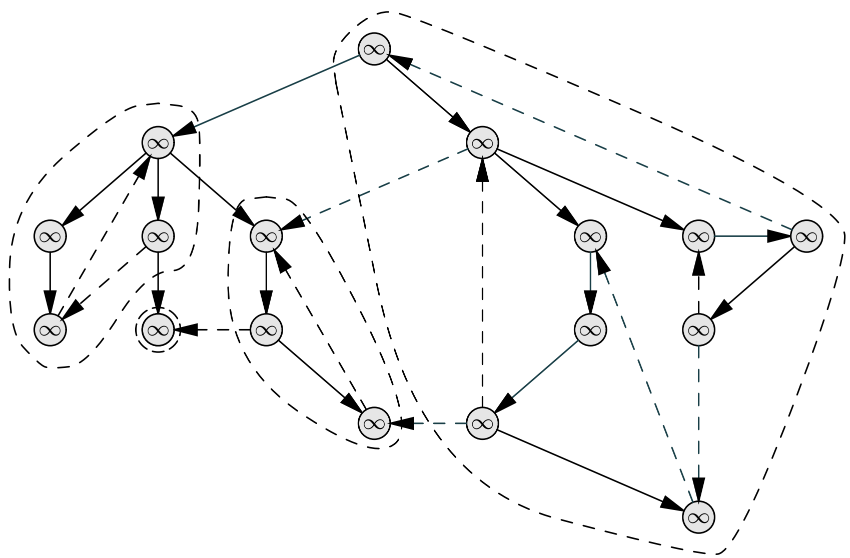
(point d'attaches, cf. biconnexité)

Connexité forte (5/11)



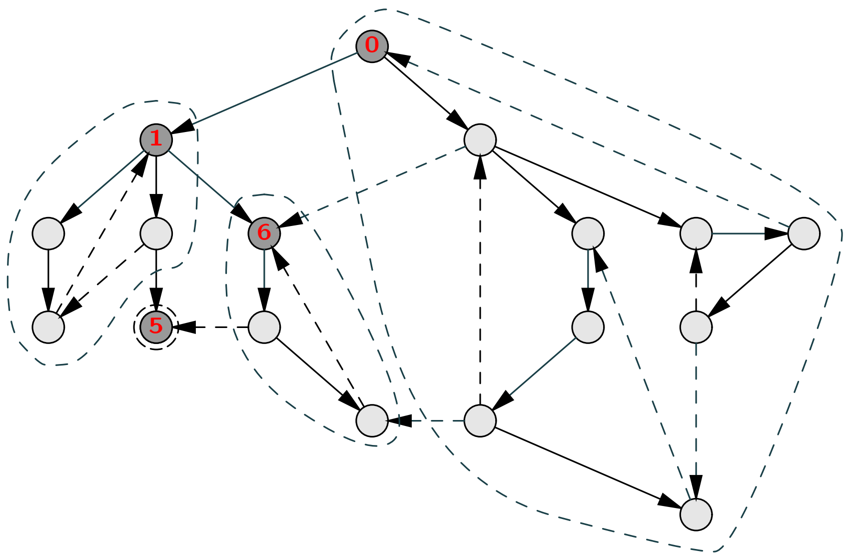
(point d'attaches, cf. biconnexité)

Connexité forte (6/11)



(point d'attaches, cf. biconnexité)

Connexité forte (7/11)

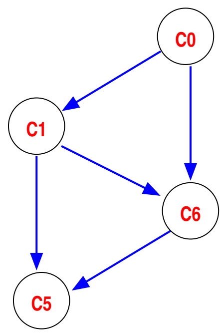


point d'attaches en foncé ≡ premier sommet visité par `dfs`

Connexité forte (8/11)

Le graphe $G' = (V', E')$ des composantes fortement connexes du graphe $G = (V, E)$ est défini par :

- $V' = \{ C \mid C$ composante fortement connexe de $G \}$
- $E' = \{ (C, C') \mid \exists e, \;\; e \in E \;\wedge\; org(e) \in C \;\wedge\; ext(e) \in C' \}$.



Proposition 1 Le graphe $G = (V', E')$ des composantes fortement connexes de $G = (V, E)$ est acyclique.

Connexité forte (9/11)

[Tarjan, 72]

```
static int numOrdre;

static void imprimerCompFortementConnexes (Graphe g) {
  int n = g.succ.length; int num = new int[n]; numOrdre = -1;
  for (int x = 0; x < n; ++x) num[x] = -1;
  Pile p = new Pile(n);
  for (int x = 0; x < n; ++x)
    if (num[x] == -1)
      imprimerComposanteDe(g, x, num, p);
}
```

Connexité forte (10/11)

[Tarjan, 72]

```
static int imprimerComposanteDe (Graphe g, int x, int[ ] num, Pile p) {
  Pile.empiler(x, p);
  int min = num[x] = ++numOrdre;
  for (Liste ls = g.succ[x]; ls != null; ls = ls.suivant) {
    int y = ls.val; int m;
    if (num[y] == -1)
      m = imprimerComposanteDe (g, y, num, p);
    else m = num[y];
    min = Math.min (min, m);
  }
  if (min == num[x]) {
    int y; do {
      y = Pile.depiler(p);
      System.out.print (y + " ");
      num[y] = g.succ.length; // équivalent à num[y] ← ∞
    } while (y != x);
    System.out.println();
    min = g.succ.length;
  }
  return min;
}
```

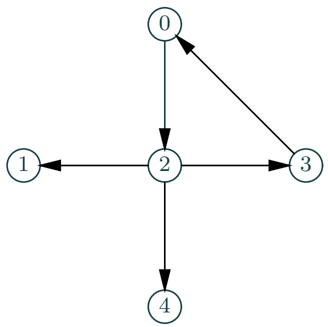
Connexité forte (11/11)

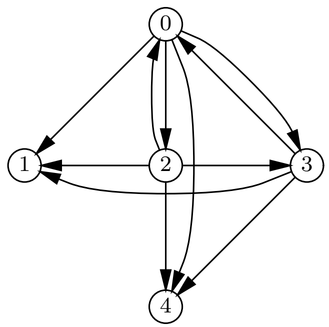
Le point d'attache est le sommet en premier visité par `dfs`

- pour un circuit : test acyclicité
- pour tous les circuits élémentaires contenant un fils dans l'arbre de recouvrement d'un sommet donné : points d'articulation dans un graphe non-orienté.
- pour une composante fortement connexe : points d'attache de cette composante
- Les arcs de retour e vérifient $num[ext(e)] < num[org(e)]$
  $\Rightarrow$ calculs de minimum
  $\Rightarrow$ arithmétique sur la numérotation par `dfs`
- Complexité $O(V+E) \equiv$ linéaire sur le nombre de sommets et d'arcs
  Donc $O(V^2)$ dans le pire cas.

Fin de `dfs`

Fermeture transitive (1/3)





Le graphe $G^+ = (V, E^+)$ de la fermeture transitive de $G^+ = (V, E)$ est tel que E^+ est minimum vérifiant

- $E \subset E^+$
- $(x,y) \in E^+ \ \wedge \ (y,z) \in E^+ \ \Rightarrow \ (x,z) \in E^+$ (transitivité)

Fermeture transitive (2/3)

E matrice $n \times n$ d'adjacence pour un graphe avec n sommets.

On veut calculer

$$\begin{aligned} E^+ &= E + E^2 + E^3 + \cdots \\ &= E + E^2 + E^3 + \cdots + E^{n-1} \end{aligned}$$

puisque les chemins passant par des sommets distincts sont de longueur inférieure à n.

La multiplication de matrices $n \times n$ a une complexité $O(n^3)$.

La fermeture transitive peut donc se faire en $O(n^4)$ opérations.

Faire mieux ?

Fermeture transitive (3/3)

Définition inductive des chemins selon le plus grand des nœuds intermédiaires par lesquels ils passent.

E_k est la fermeture transitive en n'utilisant des chemins passant par des sommets intermédiaires strictement inférieurs à k.

$$\begin{aligned} &E_0 = E \\ &E_{k+1} = E_k \cup \{(x,y) \mid (x,k) \in E_k, (k,y) \in E_k\} \end{aligned}$$

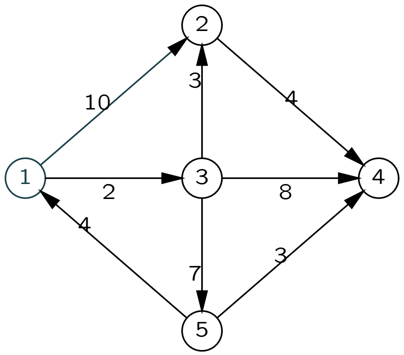
[Warshall]

```
static Graphe fermetureTransitive (Graphe g) {
  int n = g.e.length;
  Graphe gplus = copieGraphe (g);
  for (int k = 0; k < n; ++k)
    for (int x = 0; x < n; ++x)
      for (int y = 0; y < n; ++y)
        gplus.e[x][y] = gplus.e[x][y] || (gplus.e[x][k] && gplus.e[k][y]);
  return gplus;
}
```

Complexité en $O(n^3)$ ou encore $O(V^3)$.

Plus courts chemins (1/2)

$G = (V, E, d)$ est un graphe valué ($d : V \times V \mapsto \mathbf{N}$).
$d(x, y)$ est la distance de x à y.



On représente G par sa matrice d'adjacence d à valeurs entières.
On pose $d(x, y) = +\infty$ si pas d'arc de x à y.

Plus courts chemins (2/2)

Plus courts chemins entre tous les sommets d'un graphe.

$d^k_{x,y}$ est la longueur du plus court chemin entre x et y passant par des sommets intermédiaires strictement inférieurs à k.

$$d^0_{x,y} = d(x, y)$$
$$d^{k+1}_{x,y} = \min\{d^k_{x,y},\ d^k_{x,k} + d^k_{k,y}\}$$

```
[Floyd-Warshall]
static Graphe plusCourtsChemins (Graphe g) {
  int n = g.d.length;
  Graphe gplus = copieGraphe (g);
  for (int k = 0; k < n; ++k)
   for (int x = 0; x < n; ++x)
    for (int y = 0; y < n; ++y)
     gplus.d[x][y] = Math.min (gplus.d[x][y], gplus.d[x][k] + gplus.d[k][y]);
  return gplus;
}
```

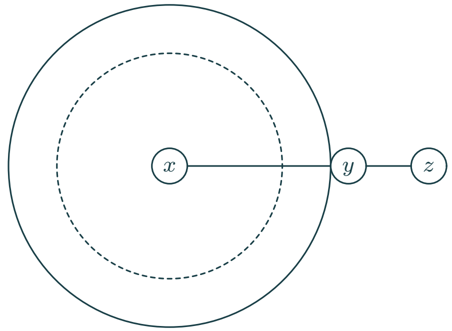
Complexité en $O(n^3)$ ou encore $O(V^3)$. Espace mémoire en $O(n^2)$.

Floyd-Warshall = programmation dynamique (cf. cours 4).

Plus court chemin (1/5)

Plus court chemin entre deux sommets x et y dans un graphe valué.
Soit $d_{x,y}$ sa longueur.

$$dist_{x,x} = 0$$
$$dist_{x,z} = \min\{dist_{x,y} + d(y, z) \mid 0 \leq y < n\}$$



Plus court chemin (2/5)

Plus court chemin entre un sommet x et tous les autres sommets.
(Problème généralisé)

- On envoie une onde de la source x vers les sommets destination.
- Deux sous-ensembles S et Q de V :
  - S sommets `NOIR` dont la distance de x minimale est définitive ;
  - Q sommets `GRIS` accessibles à partir de x dont on ne connait que partiellement la distance minimale depuis x.
- Au début $S = \emptyset$, $Q = \{x\}$
- Soit y le sommet à distance minimale depuis x dans Q. On fait

  `NOIR` $S \leftarrow S \cup \{y\}$

  `GRIS` $Q \leftarrow Q - \{y\} \ \cup \ \{z \mid (y, z) \in E,\ z \notin S\}$
- Pour les nouveaux dans Q, on met à jour la distance minimale connue à partir de x. (relaxation)
- Parcours en largeur selon la distance à x ;
  algorithme glouton (cf. cours 4)

Plus court chemin (3/5)

[Dijkstra, 1959] Exécution

```
static int[ ] plusCourtChemin (GrapheV g, int x, int u) {
  int n = g.succ.length; int[ ] couleur = new int[n];
  int[ ] dMin = new int[n]; int[ ] chemin = new int[n];
  for (int t = 0; t < n; ++t) {
    couleur[t] = BLANC; dMin[t] = Integer.MAX_VALUE;
    chemin[t] = -1;
  }
  dMin[x] = 0; couleur[x] = GRIS; int y;
  while ( (y = iMin(dMin, couleur)) != -1) {
    couleur[y] = NOIR;
    if (y == u) return chemin;
    for (ListeV ls = g.succ[y]; ls != null; ls = ls.suivant) {
      int z = ls.val; int dyz = ls.d;
      if (dMin[y] + dyz < dMin[z]) {
        dMin[z] = dMin[y] + dyz; // relaxation
        chemin[z] = y; couleur[z] = GRIS;
  } } }
  return chemin;
}
```

où `iMin(dMin, couleur)` rend l'indice du minimum `GRIS` dans `dMin`

Plus court chemin (4/5)

[Dijkstra, 1959] Exécution

```
static int[ ] plusCourtChemin (GrapheV g, int x, int u) {
  int n = g.succ.length; int[ ] couleur = new int[n];
  int[ ] dMin = new int[n]; int[ ] chemin = new int[n];
  for (int t = 0; t < n; ++t) {
    couleur[t] = BLANC; dMin[t] = Integer.MAX_VALUE;
    chemin[t] = -1;
  }
  dMin[x] = 0; couleur[x] = GRIS; int y;
  while ( (y = iMin(dMin, couleur)) != -1) {
    couleur[y] = NOIR;

    for (ListeV ls = g.succ[y]; ls != null; ls = ls.suivant) {
      int z = ls.val; int dyz = ls.d;
      if (dMin[y] + dyz < dMin[z]) {
        dMin[z] = dMin[y] + dyz; // relaxation
        chemin[z] = y; couleur[z] = GRIS;
  } } }
  return chemin;
}
```

où `iMin(dMin, couleur)` rend l'indice du minimum `GRIS` dans `dMin`

Plus court chemin (5/5)

Complexité en $O(V^2 + E)$, soit $O(n^2)$. Mémoire en $O(n)$.

En utilisant des files de priorité pour retrouver le sommet à distance minimum et changer la distance par relaxation, on arrive à

Complexité en $O((V + E)\log V)$, soit $O(n^2 \log n)$. Mémoire en $O(n)$.

(On peut ramener à $O(V \log V + E)$, soit $O(n^2)$ avec des tas de Fibonacci). [Fredman, Tarjan]

Exercice 1 Calculer le plus court chemin depuis tous les sommets jusqu'à un même sommet destination.

Exercice 2 Que se passe-t-il s'il y a des distances négatives ?

Autres problèmes sur les graphes

- Planarité (dfs, [Tarjan, 72])
- Circuit hamiltonien
- Voyageur de commerce
- Coloriage de graphe, coloriage de graphes planaires
- Optimisation de flot
- *Matching*
- Couverture
- Isomorphisme de graphes, de sous-graphes,
- etc

Problème ouvert

Beaucoup de problèmes sur les graphes sont NP-complets.