Graphes et Algorithmes – Partie V
Connexité

FISA Informatique 1ère année

2020 - 2021

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES HAUTS-DE-FRANCE

Université Polytechnique HAUTS-DE-FRANCE

Connexité – Plan

- Connexité et convexité
- Algorithmes basés sur les parcours
- Accessibilité et multiplication de matrices
- Algorithme de Warshall
- Graphe réduit et … acyclique
- Nombre cyclomatique

Convexité

- Ensemble convexe
  - Un ensemble E est **convexe** si le segment de ligne joignant deux éléments quelconques de E est également dans E

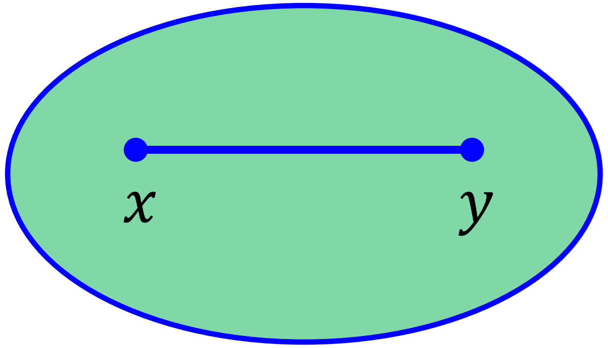
Convexité

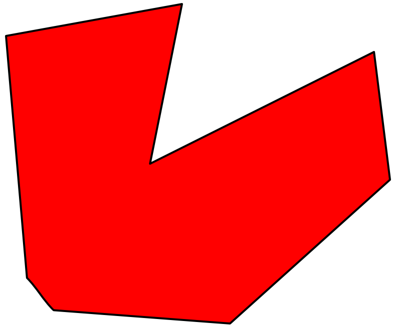
- Ensemble convexe
  - Un ensemble E est **convexe** si le segment de ligne joignant deux éléments quelconques de E est également dans E
  - Pour tout $x, y \in E$, et tout $\alpha \in [0,1]$, on a
$$\alpha x + (1 - \alpha) y \in E$$

Convexité

- Ensemble convexe
  - Un ensemble E est **convexe** si le segment de ligne joignant deux éléments quelconques de E est également dans E
  - Pour tout $x, y \in E$, et tout $\alpha \in [0,1]$, on a

$$\alpha x + (1-\alpha)y \in E$$





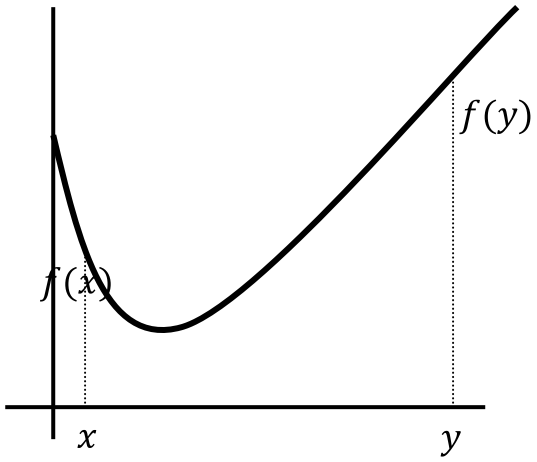
Convexité

- Fonction convexe
  - Une fonction f est convexe sur un ensemble E si pour tout $x, y \in E$, et tout $\alpha \in [0,1]$, on a
$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y)$$

Convexité

- Fonction convexe
  - Une fonction f est convexe sur un ensemble E si pour tout $x, y \in E$, et tout $\alpha \in [0,1]$, on a

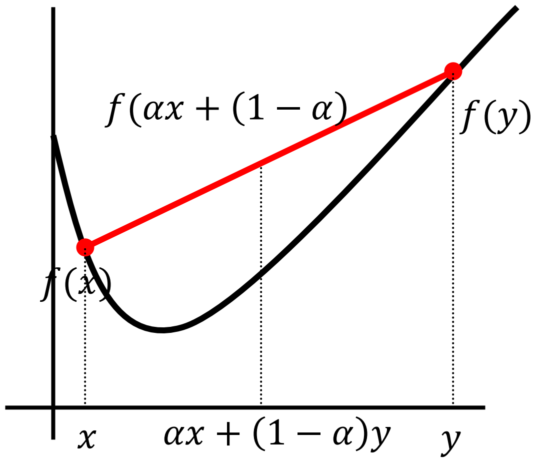
$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y)$$



Convexité

- Fonction convexe
  - Une fonction f est convexe sur un ensemble E si pour tout $x, y \in E$, et tout $\alpha \in [0,1]$, on a

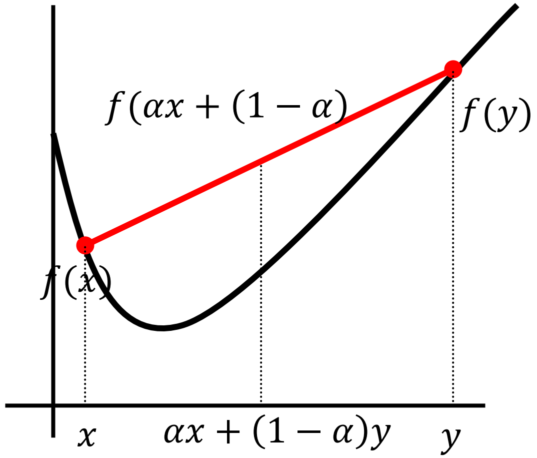
$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y)$$

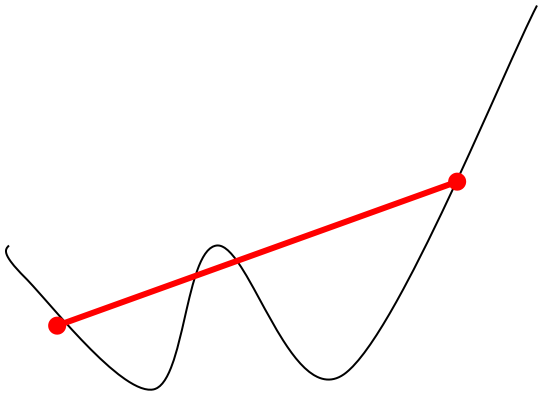


Convexité

- Fonction convexe
  - Une fonction f est convexe sur un ensemble E si pour tout $x, y \in E$, et tout $\alpha \in [0,1]$, on a

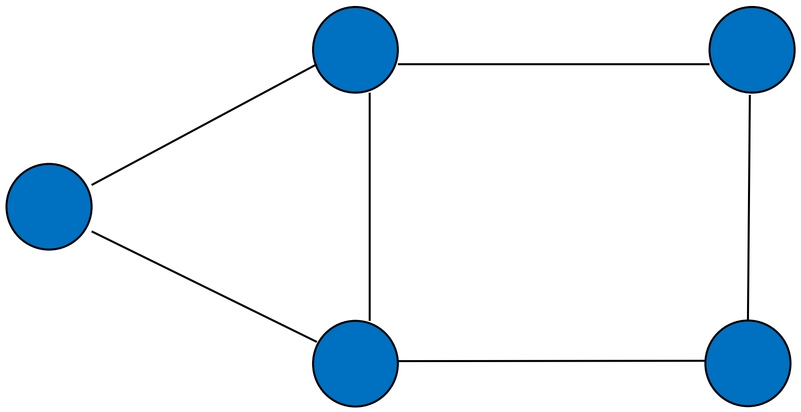
$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y)$$



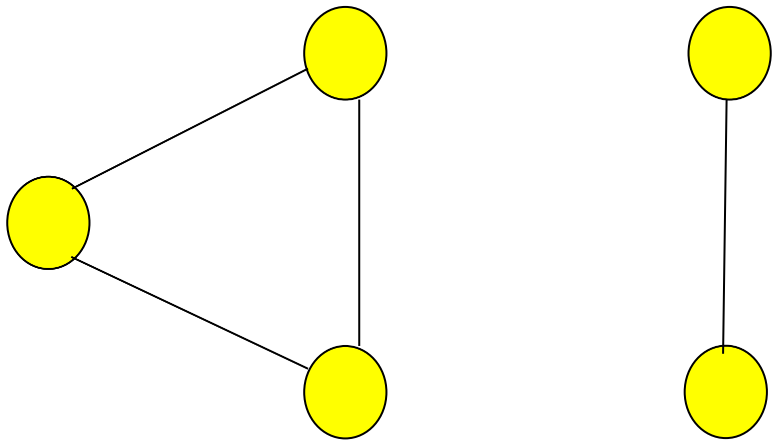


Connexité

- Un graphe **non orienté** est **connexe** si deux sommets quelconques sont connectés par une chaîne



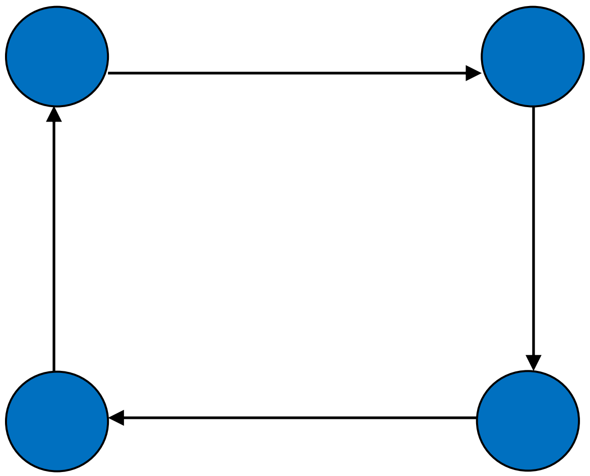
Connexe



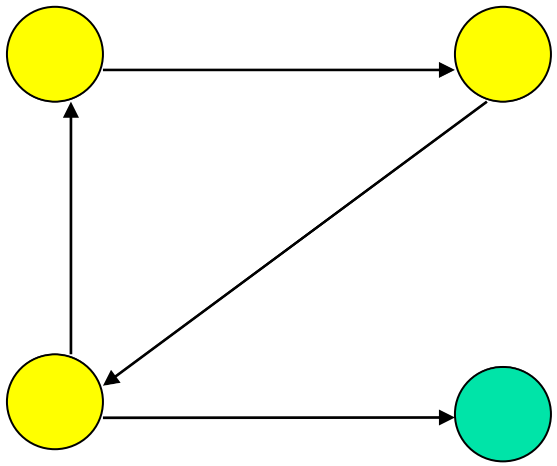
Non connexe

Forte connexité

- Un graphe **orienté** est **fortement connexe** s'il existe un **chemin** de n'importe quel sommet vers n'importe quel autre sommet



Fortement connexe



Non fortement connexe

Connexe $\neq$ convexe

- Ensemble convexe
- Fonction convexe

- Graphe connexe

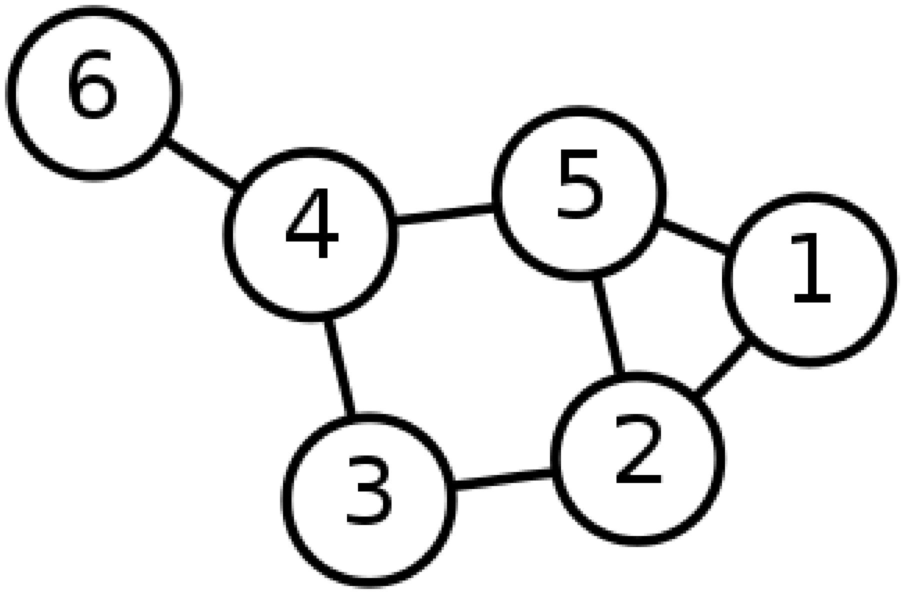
Connexe $\neq$ convexe

- Ensemble convexe
- Fonction convexe

- Graphe connexe
- **Sous-graphe convexe**

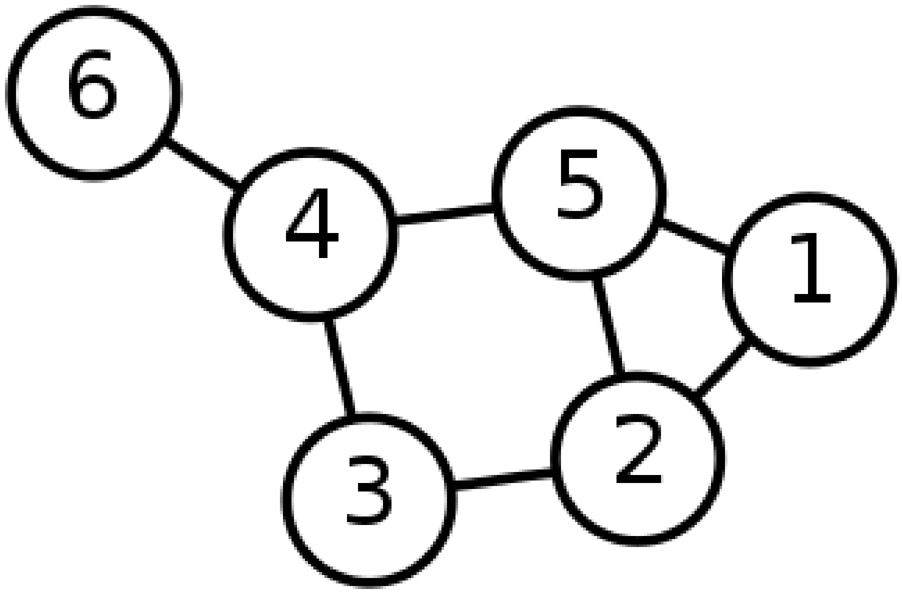
Sous-graphe convexe

- Un sous-graphe **convexe** d'un graphe non orienté G est un sous-graphe qui contient chaque plus court chemin entre deux des sommets de G



Sous-graphe convexe

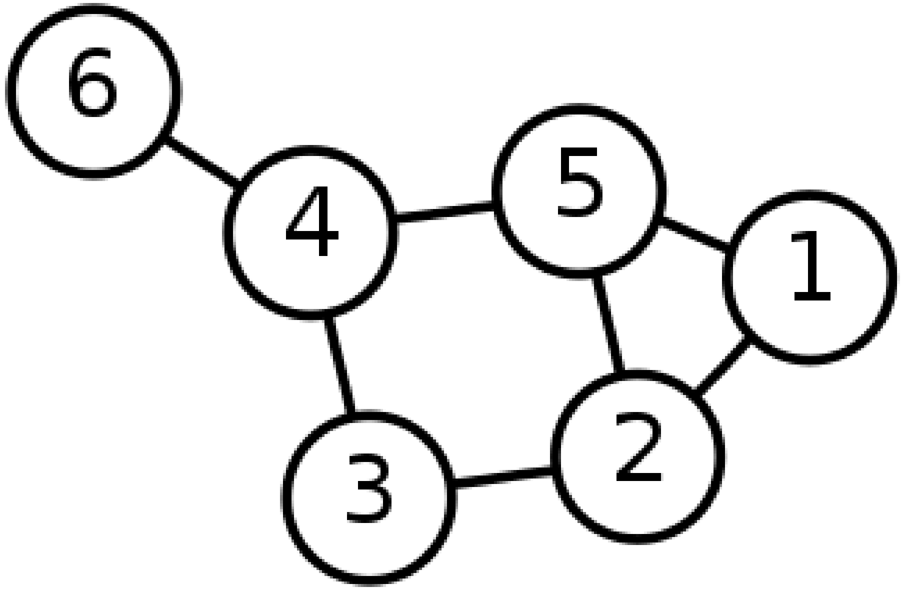
- Un sous-graphe **convexe** d'un graphe non orienté G est un sous-graphe qui contient chaque plus court chemin entre deux des sommets de G



Le triangle formé par les sommets 1-2-5 (**cycle**) **est convexe**

Sous-graphe convexe

- Un sous-graphe **convexe** d'un graphe non orienté G est un sous-graphe qui contient chaque plus court chemin entre deux des sommets de G



Le triangle formé par les sommets 1-2-5 (**cycle**) **est convexe**

Le chemin < 2 – 3 – 4 > **n'est pas convexe**
$\rightarrow$ il n'inclut pas un des deux plus courts chemins de 2 à 4

Accessibilité

- Soit $G = (S, A)$ un graphe orienté
- Rappel des notations
  - Ensemble des successeurs d'un sommet $i \in S$: $V^+(i) = \{j \in S: (i,j) \in A\}$
  - Ensemble des prédécesseurs d'un sommet $i \in S$: $V^-(i) = \{j \in S: (j,i) \in A\}$
  - Ensemble des voisins d'un sommet $i \in S$: $V(i) = V^-(i) \cup V^+(i)$
- Si $I \subseteq S$, on note

  $V(I) = \bigcup_{i \in I} V(i)$,

  $V^+(I) = \bigcup_{i \in I} V^+(i)$

  $V^-(I) = \bigcup_{i \in I} V^-(i)$

Accessibilité

- Soit $G = (S, A)$ un graphe orienté et $I \subseteq S$, on note
  $V(I) = \bigcup_{i \in I} V(i)$, $\quad V^+(I) = \bigcup_{i \in I} V^+(i)$ et $\quad V^-(I) = \bigcup_{i \in I} V^-(i)$
- Soit k un entier, soit $i \in S$, on note $V^{+k}(i) = V^+\left(V^{+(k-1)}(i)\right)$
- **Convention** $V^{+0}(i) = V^{-0}(i) = V^0(i) = \{i\}$

$\rightarrow$ successeurs, prédécesseurs, voisins **indirects** de i

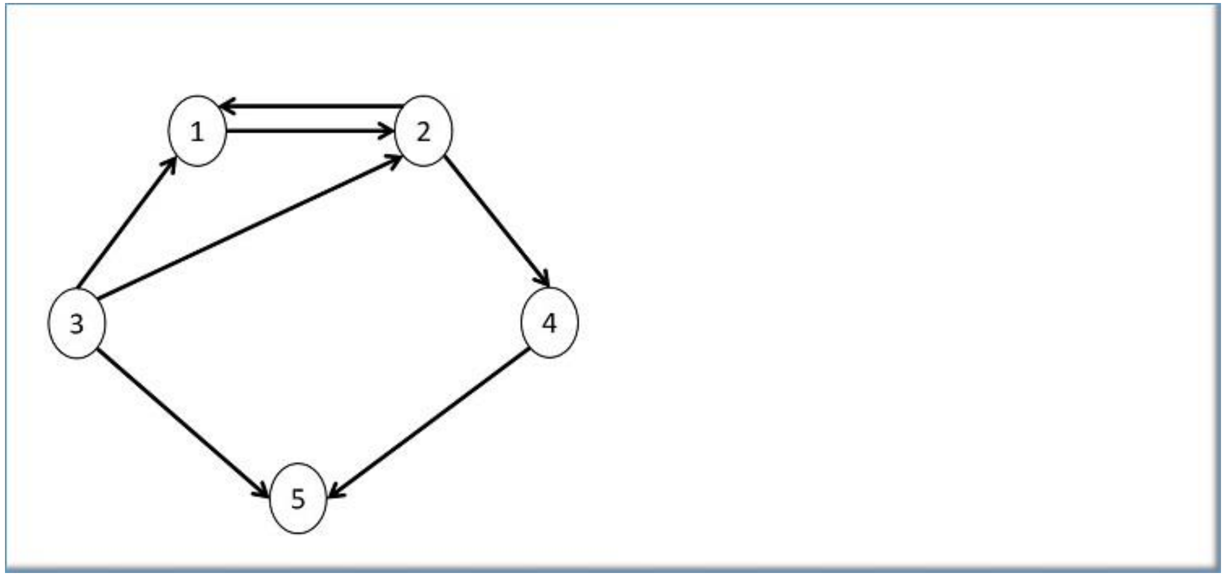
- Sommets **accessibles** depuis i : $V^*(i) = \bigcup_{k \geq 0} V^k(i)$
- $\boldsymbol{G}$ **est connexe ssi** $\boldsymbol{V^*(i) = S}$

Fermeture transitive

- La fermeture transitive d'un graphe orienté (resp. non orienté) $G = (S, A)$ est un graphe $G^* = (S, A^*)$ tel que $\forall i, j \in S$
  - $(i, j) \in A^* \leftrightarrow$ il existe un chemin (resp. une chaîne) de i à j
  - $(i, j) \in A^* \leftrightarrow j \in V^{+*}(i)$ (resp. $j \in V^*(i)$)

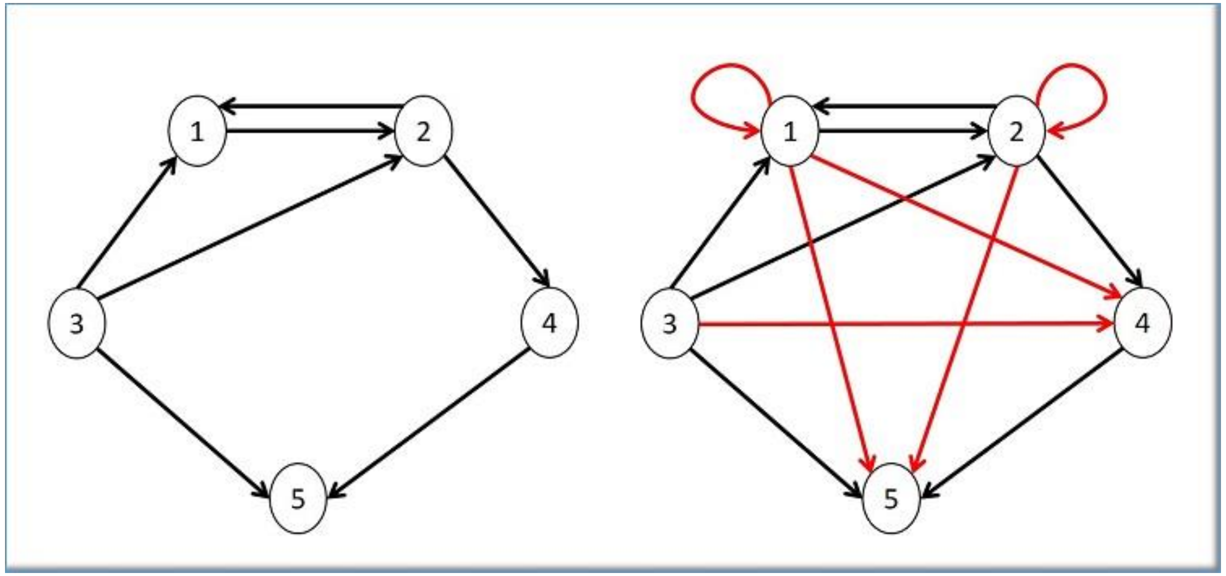
Fermeture transitive

- La fermeture transitive d'un graphe orienté (resp. non orienté) $G = (S, A)$ est un graphe $G^* = (S, A^*)$ tel que $\forall i, j \in S$
  - $(i, j) \in A^* \leftrightarrow$ il existe un chemin (resp. une chaîne) de i à j
  - $(i, j) \in A^* \leftrightarrow j \in V^{+*}(i)$ (resp. $j \in V^*(i)$)



Fermeture transitive

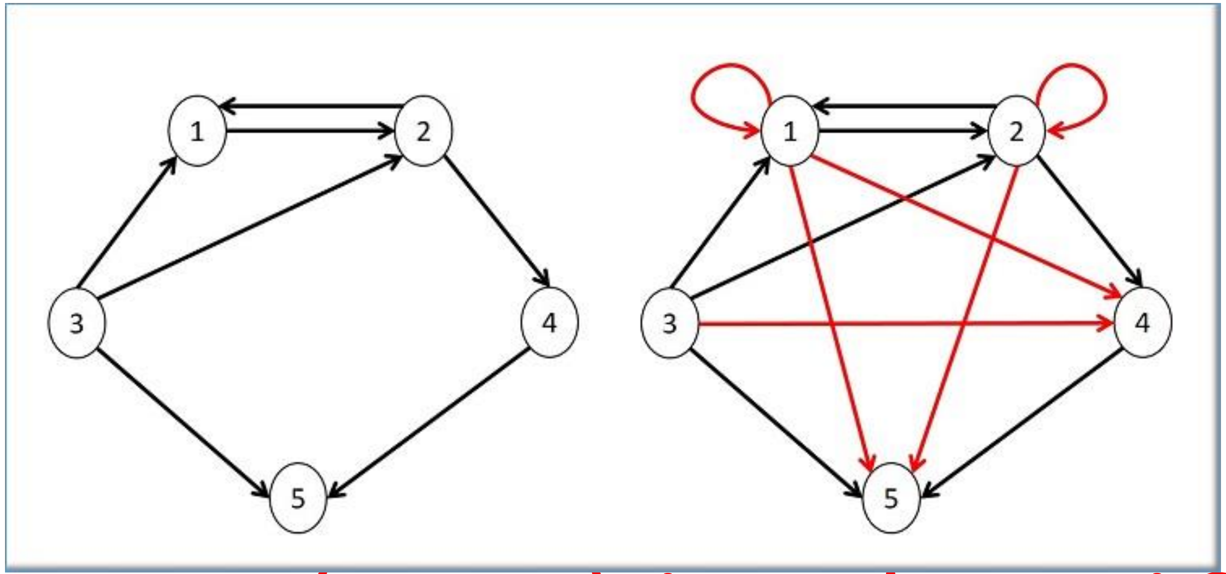
- La fermeture transitive d'un graphe orienté (resp. non orienté) $G = (S, A)$ est un graphe $G^* = (S, A^*)$ tel que $\forall i, j \in S$
  - $(i, j) \in A^* \leftrightarrow$ il existe un chemin (resp. une chaîne) de i à j
  - $(i, j) \in A^* \leftrightarrow j \in V^{+*}(i)$ (resp. $j \in V^*(i)$)



clôture / fermeture transitive

Fermeture transitive

- La fermeture transitive d'un graphe orienté (resp. non orienté) $G = (S, A)$ est un graphe $G^* = (S, A^*)$ tel que $\forall i, j \in S$
  - $(i, j) \in A^* \leftrightarrow$ il existe un chemin (resp. une chaîne) de i à j
  - $(i, j) \in A^* \leftrightarrow j \in V^{+*}(i)$ (resp. $j \in V^*(i)$)



clôture / fermeture transitive

  - **G est fortement connexe (connexe) si et seulement si G^* est complet**

Relation de connexité

- Soit $G = (S, A)$ un graphe **non orienté**, on définit la **relation binaire C sur S** telle que $\forall i, j \in S$

$$i\ C\ j \leftrightarrow \begin{cases} i = j, \text{ ou} \\ \text{il existe une chaîne entre } i \text{ et } j \end{cases}$$

- C est une relation d'équivalence, i.e. $\forall i, j, k \in S$
  - C est réflexive $\leftrightarrow$ $i\ C\ i$
  - C est symétrique $\leftrightarrow$ $i\ C\ j \Rightarrow j\ C\ i$
  - C est transitive $\leftrightarrow$ $i\ C\ j$ et $j\ C\ k \Rightarrow i\ C\ k$

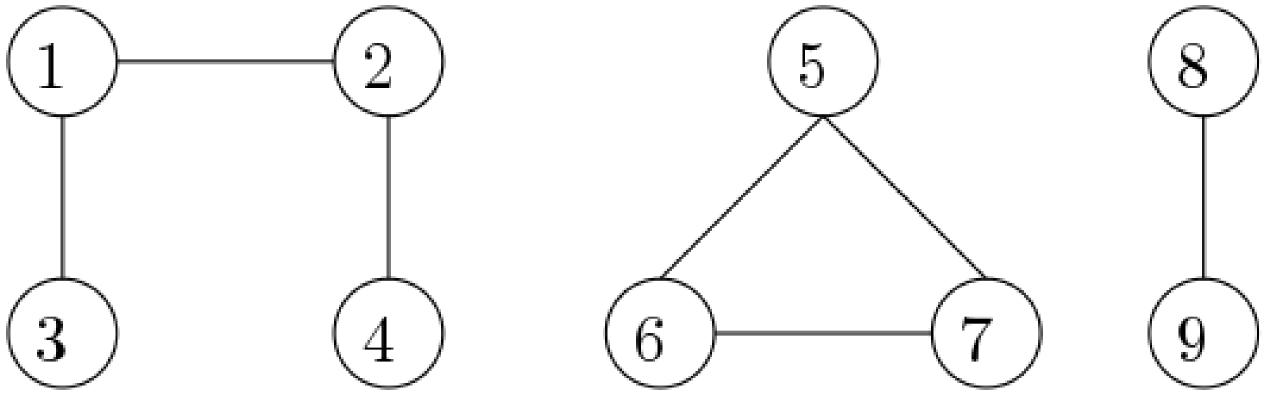
Relation de connexité

- Soit $G = (S, A)$ un graphe **non orienté**, on définit la **relation binaire C sur S** telle que $\forall i, j \in S$

$$i\ C\ j \leftrightarrow \begin{cases} i = j, \text{ ou} \\ \text{il existe une chaîne entre } i \text{ et } j \end{cases}$$

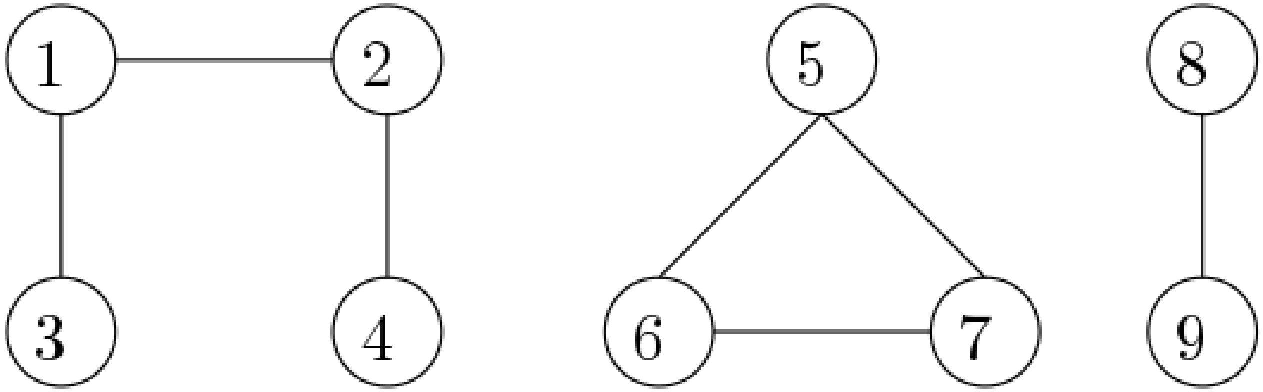
- C est une relation d'équivalence, i.e. $\forall i, j, k \in S$
  - $C(i) = \{j \in S : \ i\ C\ j\}$ classe d'équivalence de i
  - Le sous graphe engendré par $C(i)$ est appelé **composante connexe** de G contenant i, noté $CC(i)$
  - ***G* est connexe ssi *G* possède une seule composante connexe**

Exemple



$\mathcal{C}(1) = \{ x \in S \mid 1\ \mathcal{C}\ x \}$
$= \{ 1, 2, 3, 4 \}$

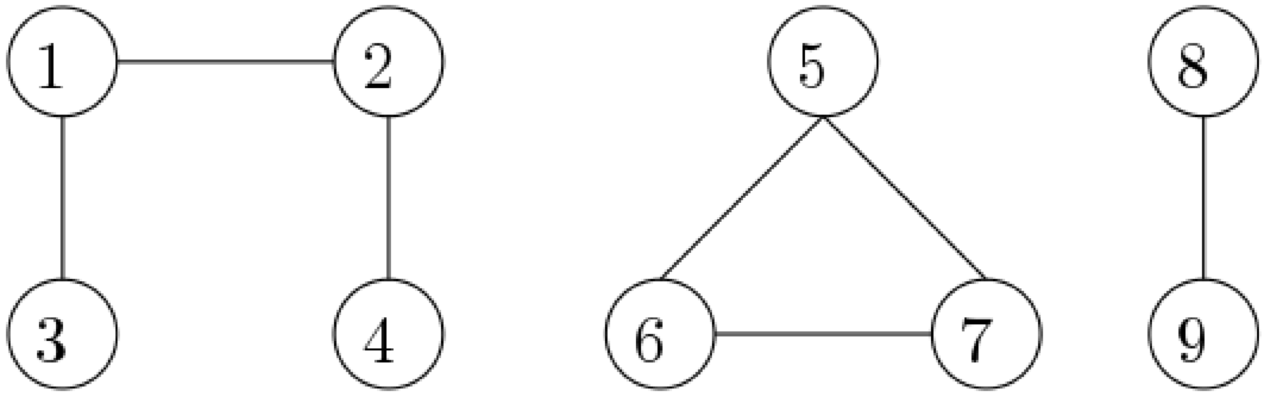
Exemple



$\mathcal{C}(1) = \{ x \in S \mid 1 \, \mathcal{C} \, x \}$
$= \{ 1, 2, 3, 4 \}$

$\mathcal{C}(2) = \mathcal{C}(3) = \mathcal{C}(4) = \mathcal{C}(1)$

Exemple



$C(1) = \{ x \in S \mid 1 \ C \ x \}$
$= \{ 1, 2, 3, 4 \}$

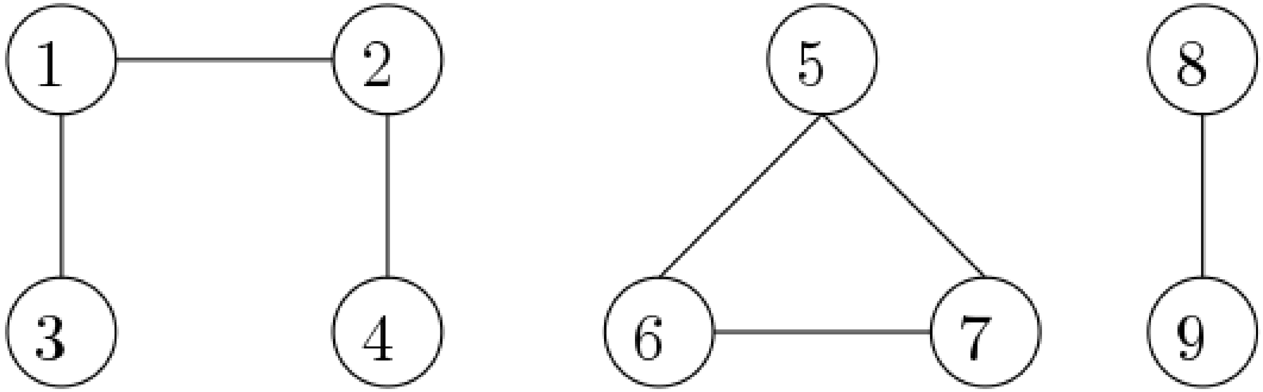
$C(5) = C(6) = C(7)$

$C(8) = C(9)$

$C(2) = C(3) = C(4) = C(1)$

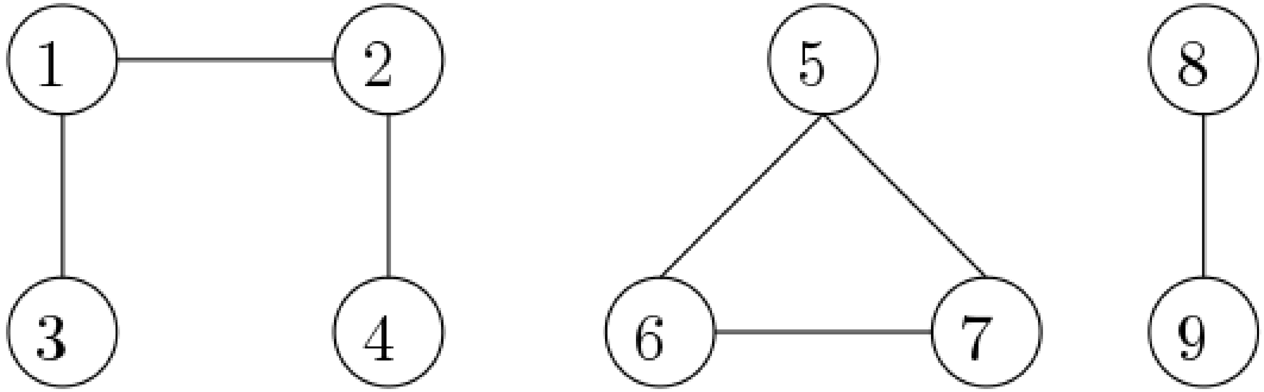
Exemple

Non connexe

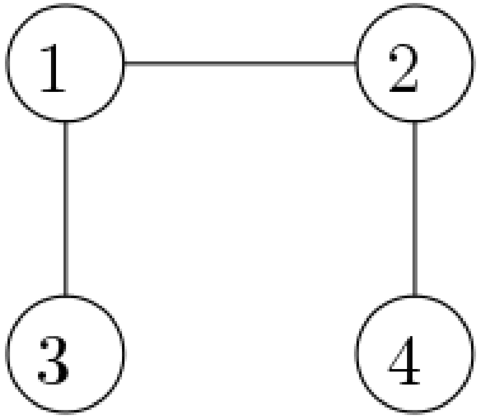


Exemple

Non connexe



Connexe



Propriétés

- Soit $G = (S, A)$ un graphe **non orienté**
  - Si $j \in CC(i)$ alors $CC(i) = CC(j)$
  - G connexe $\Leftrightarrow \forall i \in S, CC(i) = S$
  - $j \in CC(i) \Leftrightarrow (i, j) \in A^*$
- Pour tout graphe non orienté $G = (S, A)$ on a

$$v(G) = |A| - |S| + p \geq 0$$

avec p est le nombre de composantes connexes de G

$v(G)$ est le nombre cyclomatique de G

Entrée : Un graphe $G = (S, A)$ et
Une source $s \in S$

Sortie
$d[i]$: distance de s à $i \in S$
$p[i]$: prédécesseur de i

Algorithmes basés sur les parcours

- Parcours en Largeur (PeL) : **rappel**

```
procédure PEL (Entrée : G, s ∈ S
               Sortie : d, p)
  pour x ∈ S – {s} faire
     c[x] ← Blanc ;
     d[x] ← + ∝
     p[x] ← Nul ;
  fin pour
  c[s] ← Gris;  d[s] ← 0;  p[s] ← Nul ;
  F ← ∅ ; enfiler(F, s) ;
tant que F ≠ ∅ faire
     x ← défiler(F);
     pour y ∈ Adj[x] faire
        si (c[y] = Blanc) alors
           c[y] ← gris;
           d[y] ← d[x] + 1;  p[y] ← x;
           enfiler(F, y);
        fin si
     fin pour
     c[x] ← noir;
fin tant que
```

$Adj[j] = V^{+}(i)\ ou\ V(i)$
si G orienté ou non

$c[i]$: couleur de $i \in S$
Blanc : Non découvert
Gris : Découvert
Noir : Terminé
F : file des sommets Gris

Algorithme CC – PeL

Entrée : Un graphe $G = (S, A)$

Sortie : $cc[i]$: numéro de la CC de i

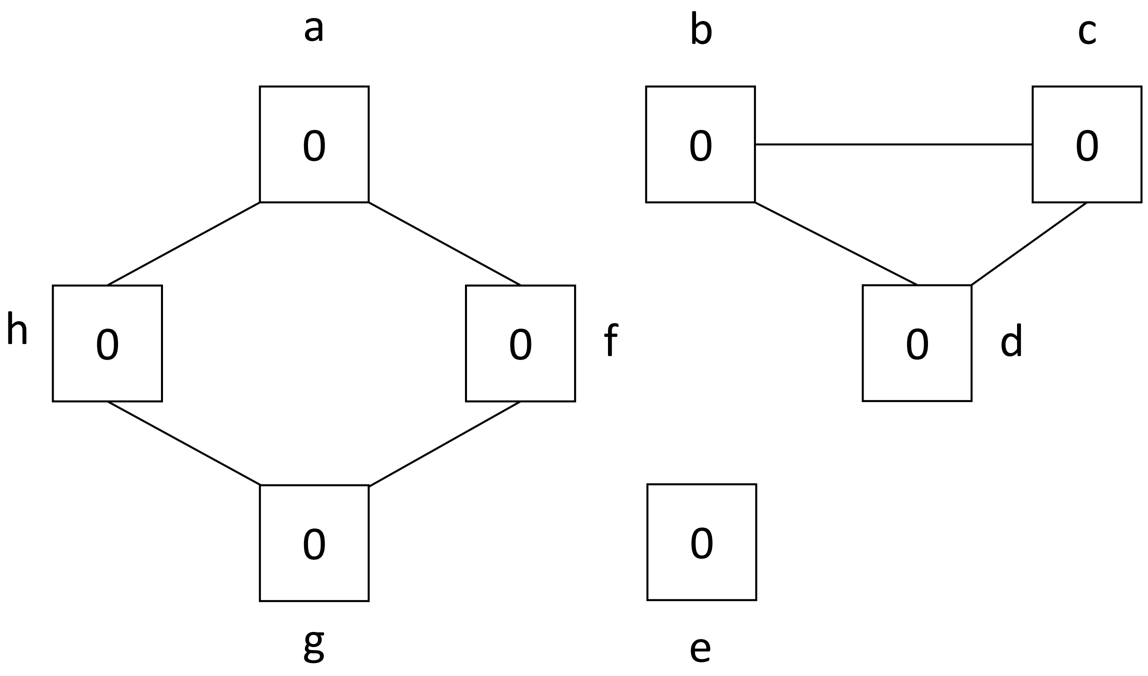
```
procédure CC-PEL (Entrée : G,
                  Sortie : cc)
   pour x ∈ S faire
      cc[x] ← 0 ;
   fin pour
   ncc ← 0 ;
   pour x ∈ S faire
      si(cc[x] = 0) alors
         ncc ← ncc + 1 ;
         PeL2(G, x, ncc, cc) ;
      fin si
   fin pour
```

```
procédure PeL2(Entrée : G, s, ncc,
               Entrée/Sortie : cc)
   cc[s] ← ncc ;     F ← ∅ ;     enfiler(F, s) ;
   tant que F ≠ ∅ faire
      x ← défiler(F);
      pour y ∈ Adj[x] faire
         si (cc[y] = 0) alors
            cc[y] ← ncc;
            enfiler(F, y);
         fin si
      fin pour
fin tant que
```

F : file des sommets visités

$Adj[j] = V(i)$

Algorithme CC – PeL – Exemple

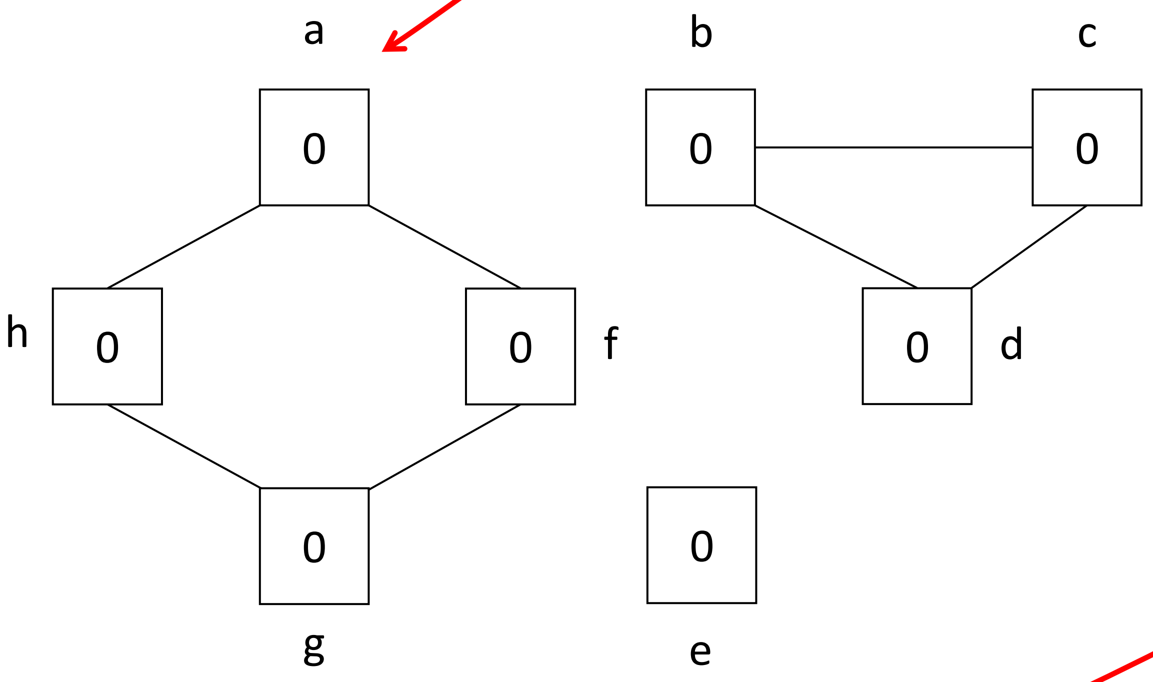


ncc = 0

procédure **CC-PEL** (Entrée : G,
Sortie : cc)

pour x ∈ S faire
  cc[x] ← 0 ;
fin pour
ncc ← 0 ;
pour x ∈ S faire
  si(cc[x] = 0) alors
    ncc ← ncc + 1 ;
    PeL2(G, x, ncc, cc) ;
  fin si
fin pour

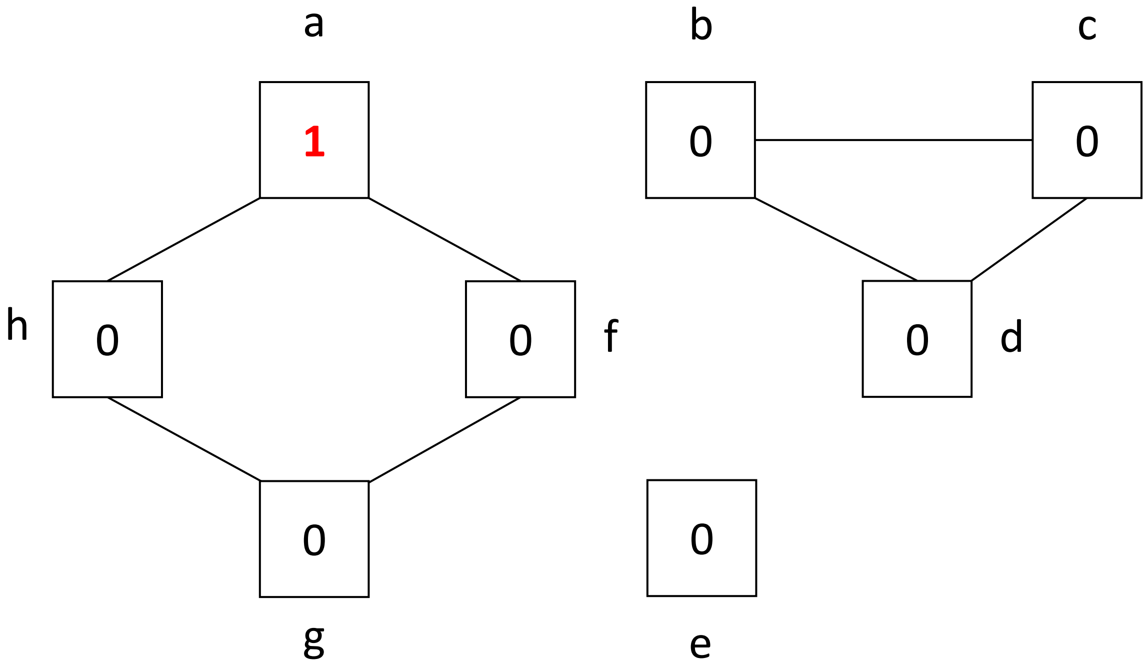
Algorithme CC – PeL – Exemple



ncc = 1

```
procédure CC-PEL (Entrée : G,
                  Sortie : cc)
  pour x ∈ S faire
    cc[x] ← 0 ;
  fin pour
  ncc ← 0 ;
  pour x ∈ S faire
    si(cc[x] = 0) alors
        ncc ← ncc + 1 ;
        PeL2(G, x, ncc, cc) ;
    fin si
  fin pour
```

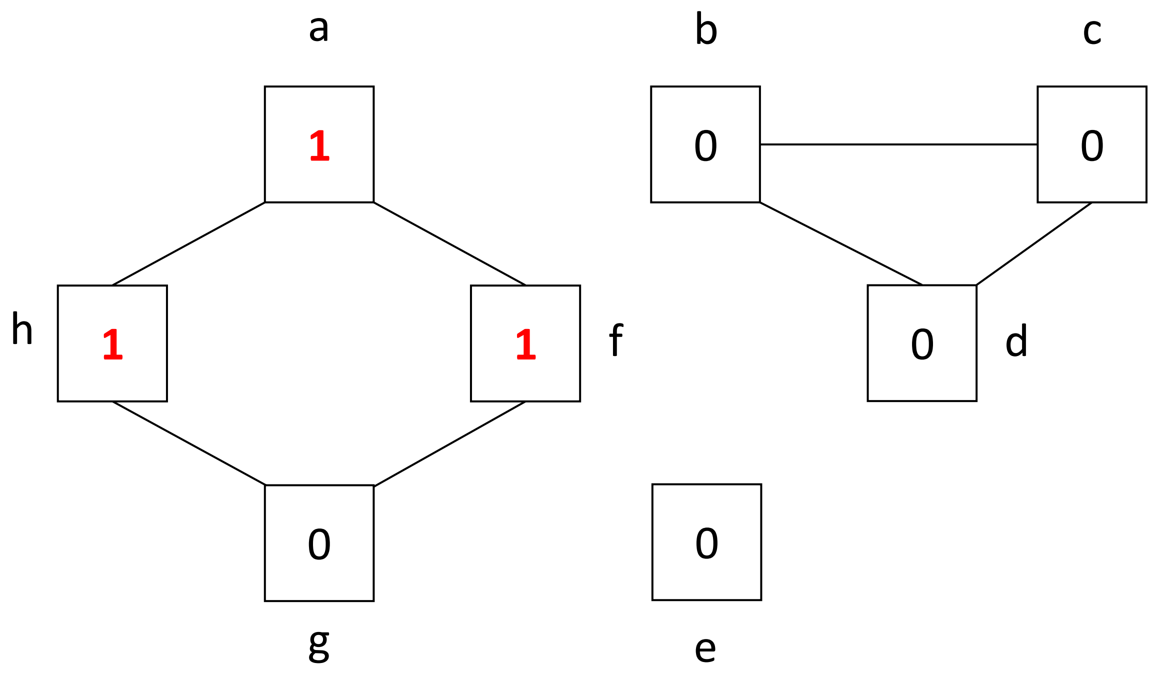
Algorithme CC – PeL – Exemple



ncc = 1

procédure **PeL2**(Entrée : G, s, ncc,
Entrée/Sortie : cc)
cc[s] $\leftarrow$ ncc ; F $\leftarrow \varnothing$; enfiler(F, s) ;
tant que F $\neq \varnothing$ faire
x $\leftarrow$ défiler(F);
pour y $\in Adj[\mathrm{x}]$ faire
si (cc[y] = 0) alors
cc[y] $\leftarrow$ ncc;
enfiler(F, y);
fin si
fin pour
fin tant que

Algorithme CC – PeL – Exemple

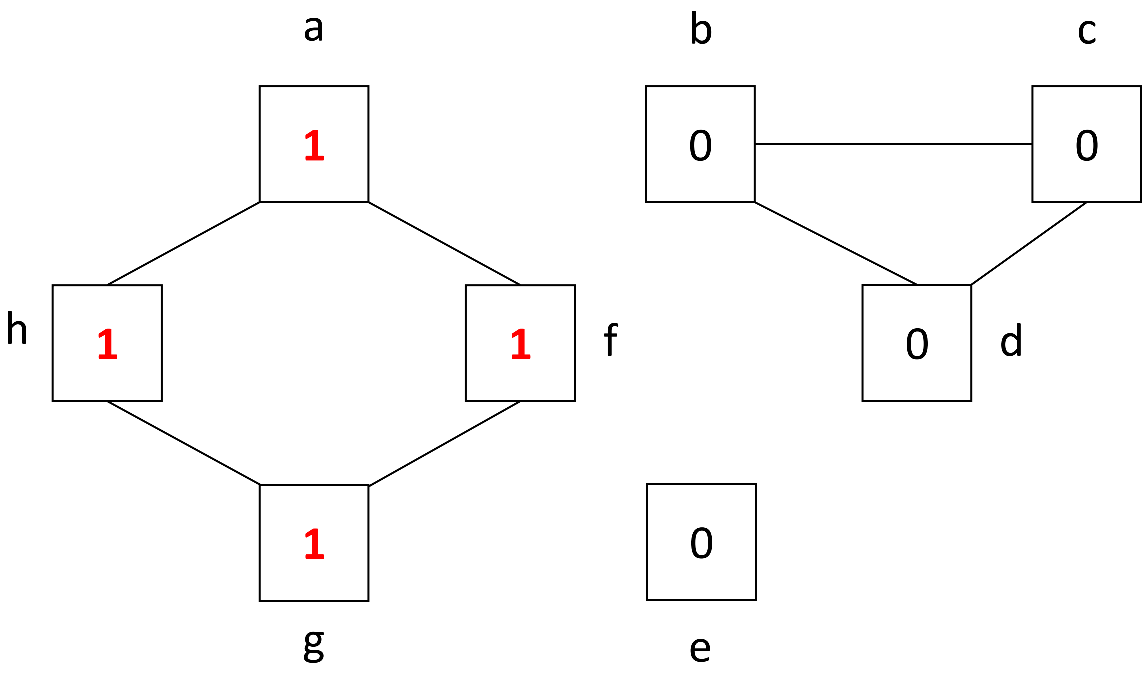


ncc = 1

procédure **PeL2**(Entrée : G, s, ncc,
Entrée/Sortie : cc)

cc[s] ← ncc ; F ← ∅ ; enfiler(F, s) ;
tant que F ≠ ∅ faire
x ← défiler(F);
pour y ∈ $Adj[\mathrm{x}]$ faire
si (cc[y] = 0) alors
cc[y] ← ncc;
enfiler(F, y);
fin si
fin pour
fin tant que

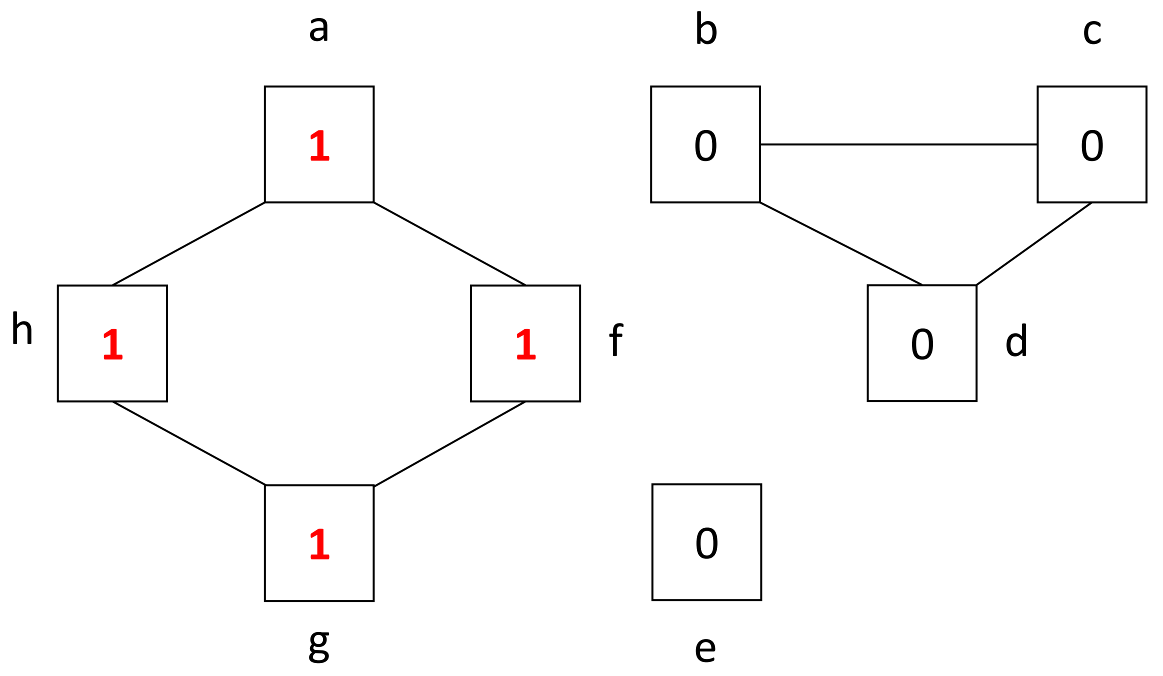
Algorithme CC – PeL – Exemple



ncc = 1

procédure **PeL2**(Entrée : G, s, ncc,
                Entrée/Sortie : cc)
  cc[s] $\leftarrow$ ncc ;   F $\leftarrow \varnothing$;   enfiler(F, s) ;
  tant que F $\neq \varnothing$ faire
    x $\leftarrow$ défiler(F);
    pour y ∈ $Adj[\mathrm{x}]$ faire
      si (cc[y] = 0) alors
        cc[y] $\leftarrow$ ncc;
        enfiler(F, y);
      fin si
    fin pour
fin tant que

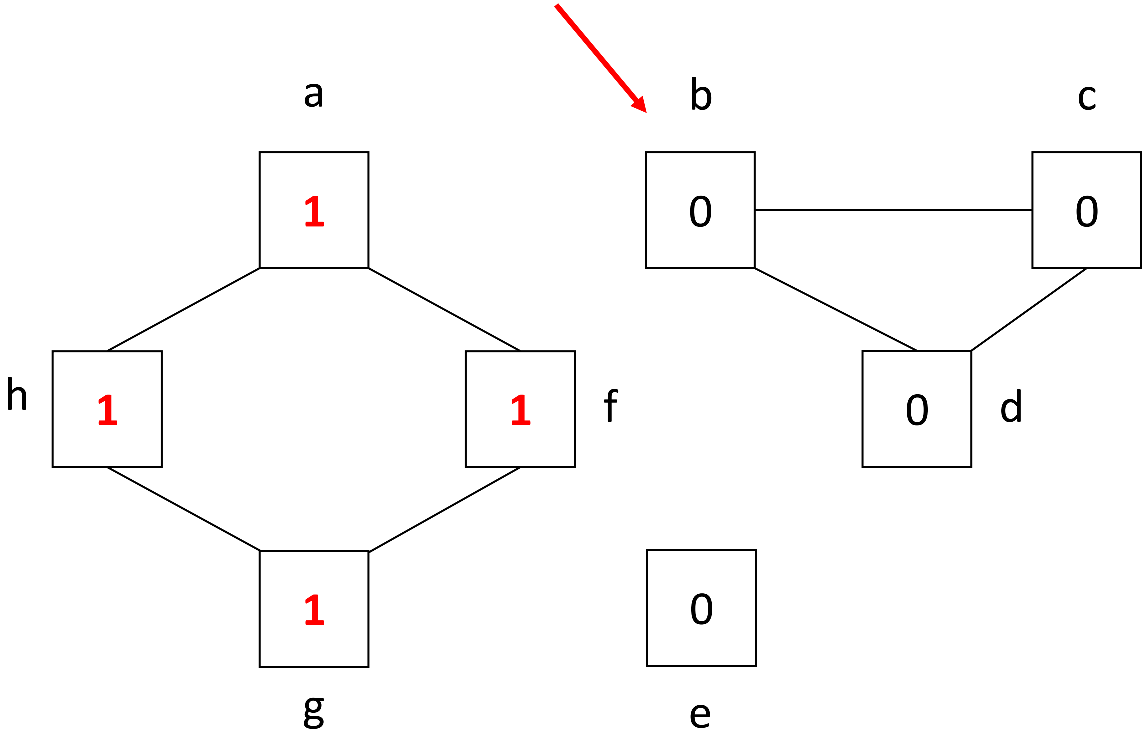
Algorithme CC – PeL – Exemple



ncc = 1

procédure **PeL2**(Entrée : G, s, ncc,
    Entrée/Sortie : cc)
  cc[s] ← ncc ;   F ← ∅ ;   enfiler(F, s) ;
  tant que F ≠ ∅ faire
    x ← défiler(F);
    pour y ∈ $Adj[x]$ faire
      si (cc[y] = 0) alors
        cc[y] ← ncc;
        enfiler(F, y);
      fin si
    fin pour
fin tant que

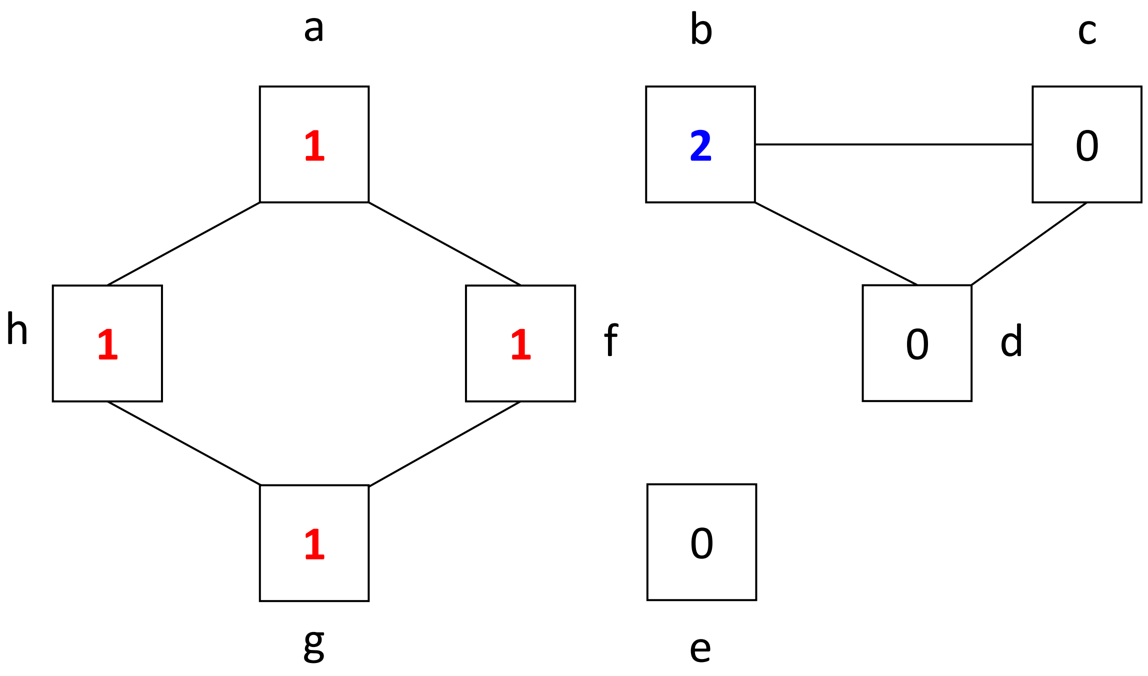
Algorithme CC – PeL – Exemple



ncc = 2

procédure **CC-PEL** (Entrée : G,
Sortie : cc)
pour x ∈ S faire
cc[x] ← 0 ;
fin pour
ncc ← 0 ;
pour x ∈ S faire
si(cc[x] = 0) alors
ncc ← ncc + 1 ;
PeL2(G, x, ncc, cc) ;
fin si
fin pour

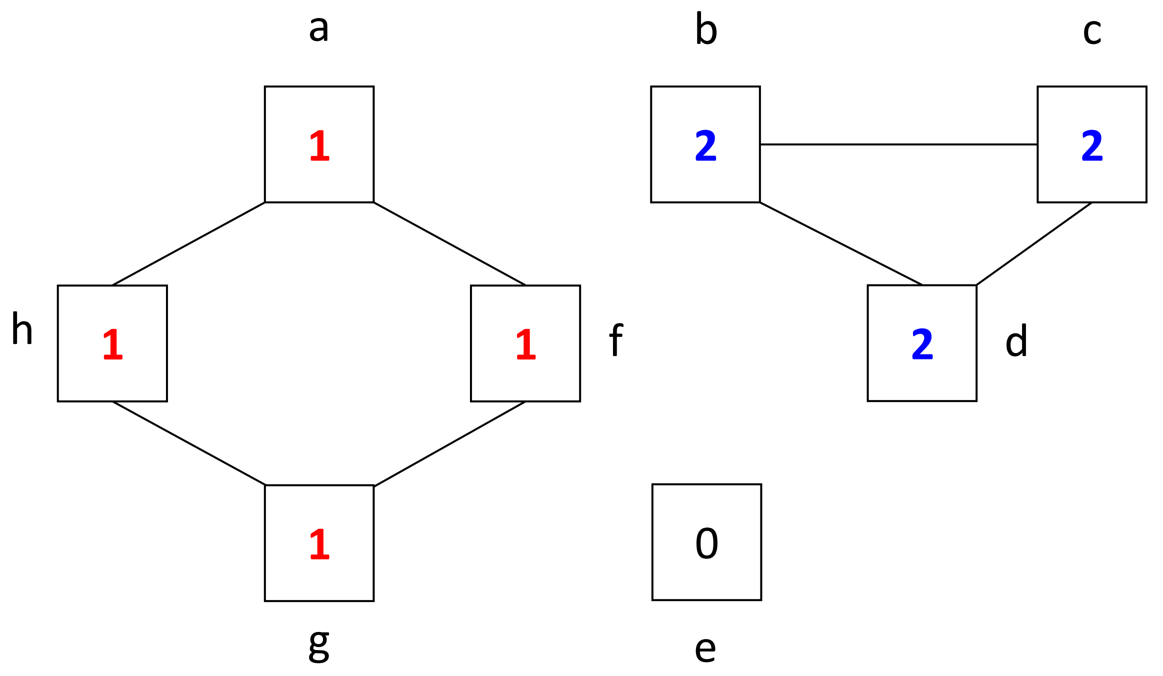
Algorithme CC – PeL – Exemple



ncc = 2

procédure **PeL2**(Entrée : G, s, ncc,
Entrée/Sortie : cc)
cc[s] $\leftarrow$ ncc ; F $\leftarrow \varnothing$; enfiler(F, s) ;
tant que F $\neq \varnothing$ faire
x $\leftarrow$ défiler(F);
pour y ∈ $Adj[\mathrm{x}]$ faire
si (cc[y] = 0) alors
cc[y] $\leftarrow$ ncc;
enfiler(F, y);
fin si
fin pour
fin tant que

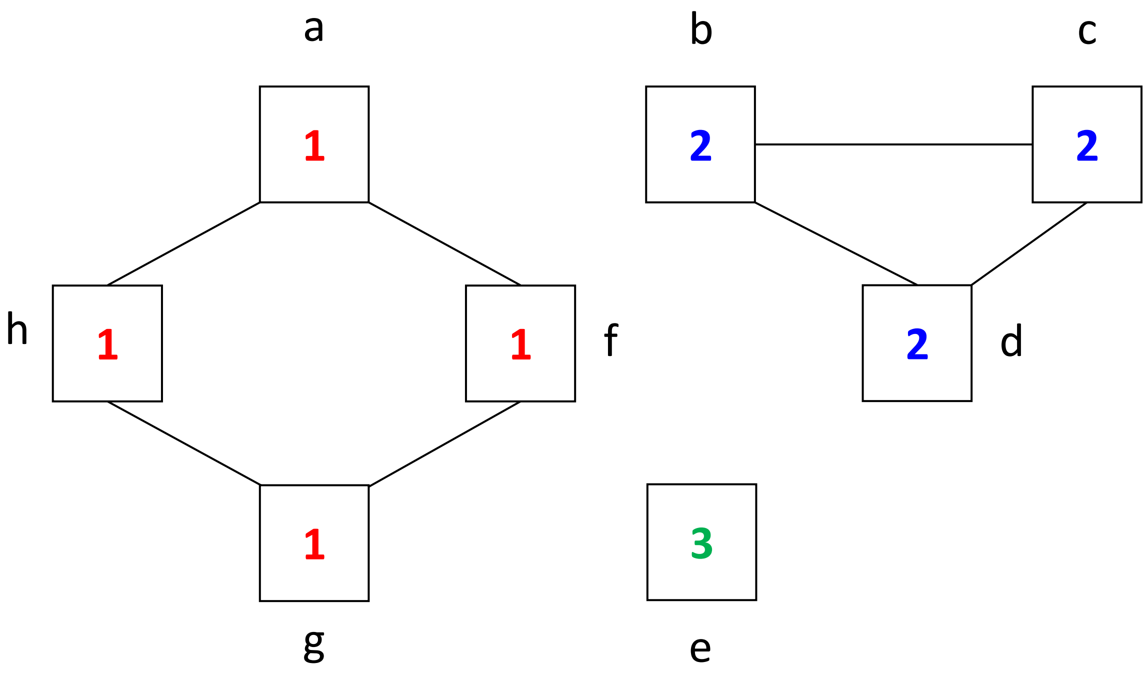
Algorithme CC – PeL – Exemple



ncc = 2

procédure **PeL2**(Entrée : G, s, ncc,
Entrée/Sortie : cc)
cc[s] ← ncc ; F ← ∅ ; enfiler(F, s) ;
tant que F ≠ ∅ faire
x ← défiler(F);
pour y ∈ $Adj[x]$ faire
si (cc[y] = 0) alors
cc[y] ← ncc;
enfiler(F, y);
fin si
fin pour
fin tant que

Algorithme CC – PeL – Exemple



ncc = 3

```
procédure CC-PEL (Entrée : G,
                  Sortie : cc)
   pour x ∈ S faire
      cc[x] ← 0 ;
   fin pour
   ncc ← 0 ;
   pour x ∈ S faire
      si(cc[x] = 0) alors
         ncc ← ncc + 1 ;
         PeL2(G, x, ncc, cc) ;
      fin si
   fin pour
```

Algorithmes basés sur les parcours

- Parcours en Profondeur (PeP) : **rappel**

Entrée : Un graphe $G = (S, A)$

Sortie

$d[i]$: date de découverte de s à $i \in S$
$f[i]$: date de fin de traitement de s à $i \in S$
$p[i]$: prédécesseur de i

$Adj[j] = V^+(i)\ ou\ V(i)$
si G orienté ou non

$c[i]$: couleur de $i \in S$
Blanc : Non découvert
Gris : Découvert
Noir : Terminé

```
procédure PEP (Entrée : G
               Sortie : d, f, p)
  pour x ∈ S faire
     c[x] ← Blanc ;   p[x] ← Nul ;
  fin pour
  temps ← 1;
  pour x ∈ S faire
     si(c[x] = Blanc) alors
         Visiter(x) ;
     fin si
  fin pour
```

```
procédure Visiter(x)
    c[x] ← Gris ;
    d[x] ← temps ;    temps ← temps + 1;
    pour y ∈ Adj[x] faire
        si (c[y] = Blanc) alors
            p[y] ← x;
            Visiter(y);
        fin si
    fin pour
    c[x] ← Noir;
    f[x] ← temps ;    temps ← temps + 1;
```

Algorithme CC – PeP

Entrée : Un graphe $G = (S, A)$

Sortie

$d[i]$: date de découverte de s à $i \in S$
$f[i]$: date de fin de traitement de s à $i \in S$
$p[i]$: prédécesseur de i
$cc[i]$: CC de i

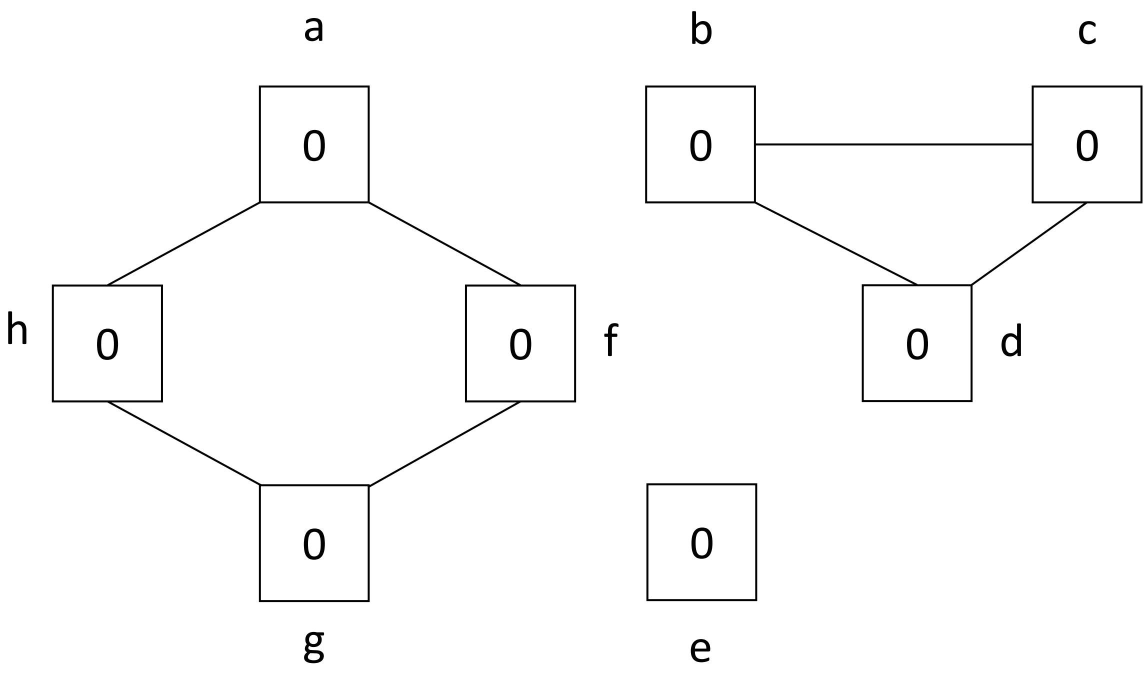
```
procédure CC-PEP (Entrée : G
                  Sortie : d, f, p)
  pour x ∈ S faire
    c[x] ← Blanc ;   p[x] ← Nul ;
    cc[x] ← 0 ;
  fin pour
  temps ← 1;  ncc ← 0 ;
  pour x ∈ S faire
    si(c[x] = Blanc) alors
      ncc ← ncc + 1 ;
      Visiter2(x) ;
    fin si
  fin pour
```

```
procédure Visiter2(x)
  c[x] ← Gris ;      cc[i] ← ncc ;
  d[x] ← temps ;     temps ← temps + 1;
  pour y ∈ Adj[x] faire
    si (c[y] = Blanc) alors
      p[y] ← x;
      Visiter2(y);
    fin si
  fin pour
  c[x] ← Noir;
  f[x] ← temps ;     temps ← temps + 1;
```

$c[i]$: couleur de $i \in S$
Blanc : Non découvert
Gris : Découvert
Noir : Terminé

$Adj[j] = V(i)$

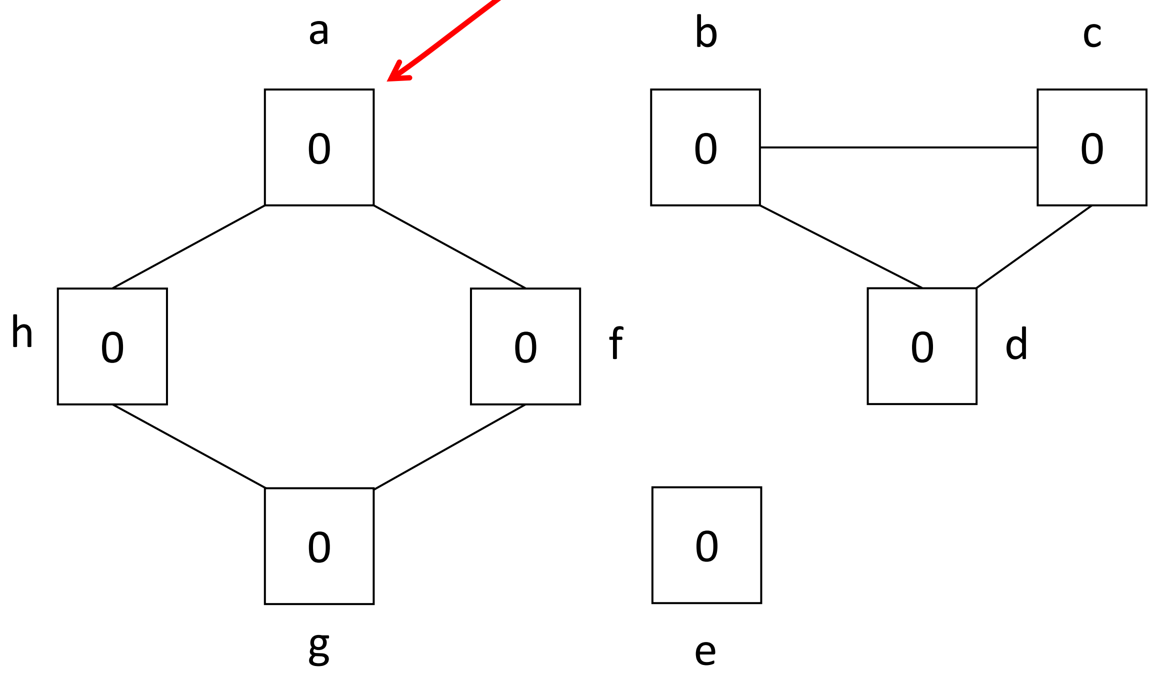
Algorithme CC – PeP – Exemple



ncc = 0

procédure CC-PEP (Entrée : G
                                    Sortie : d, f, p)
    pour x ∈ S faire
        c[x] ← Blanc ;   p[x] ← Nul ;
        cc[x] ← 0 ;
    fin pour
    temps ← 1; ncc ← 0 ;
    pour x ∈ S faire
        si(c[x] = Blanc) alors
            ncc ← ncc + 1 ;
            Visiter2(x) ;
        fin si
    fin pour

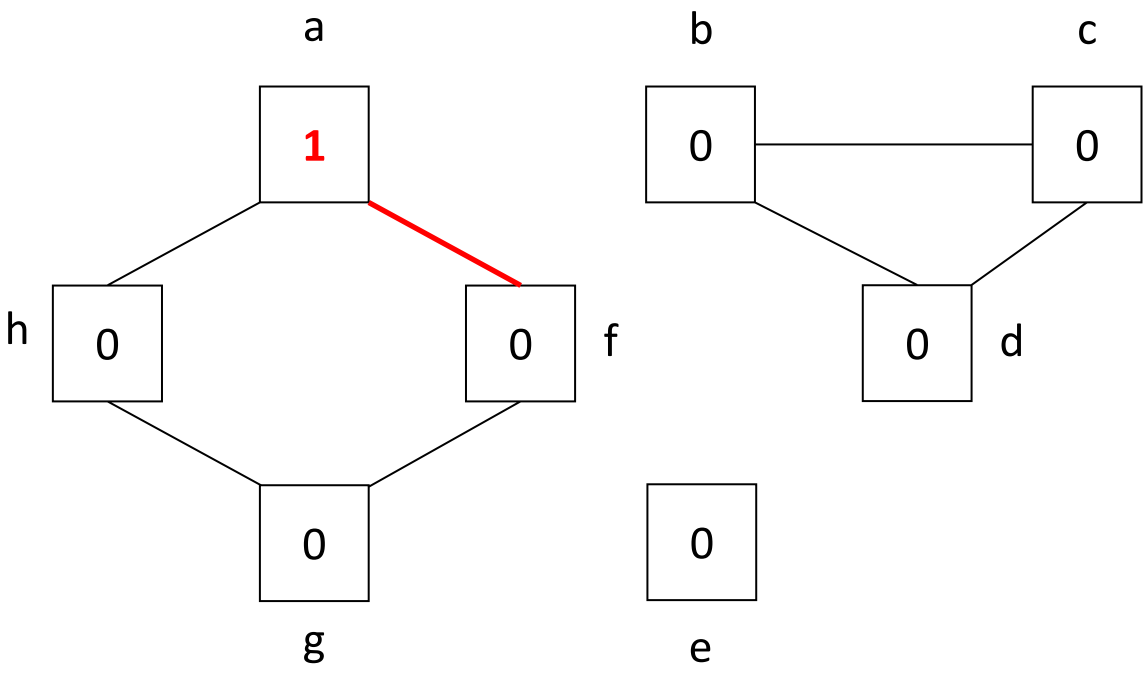
Algorithme CC – PeP – Exemple



ncc = 1

procédure CC-PEP (Entrée : G
Sortie : d, f, p)
pour x ∈ S faire
c[x] ← Blanc ; p[x] ← Nul ;
cc[x] ← 0 ;
fin pour
temps ← 1; ncc ← 0 ;
pour x ∈ S faire
si(c[x] = Blanc) alors
ncc ← ncc + 1 ;
Visiter2(x) ;
fin si
fin pour

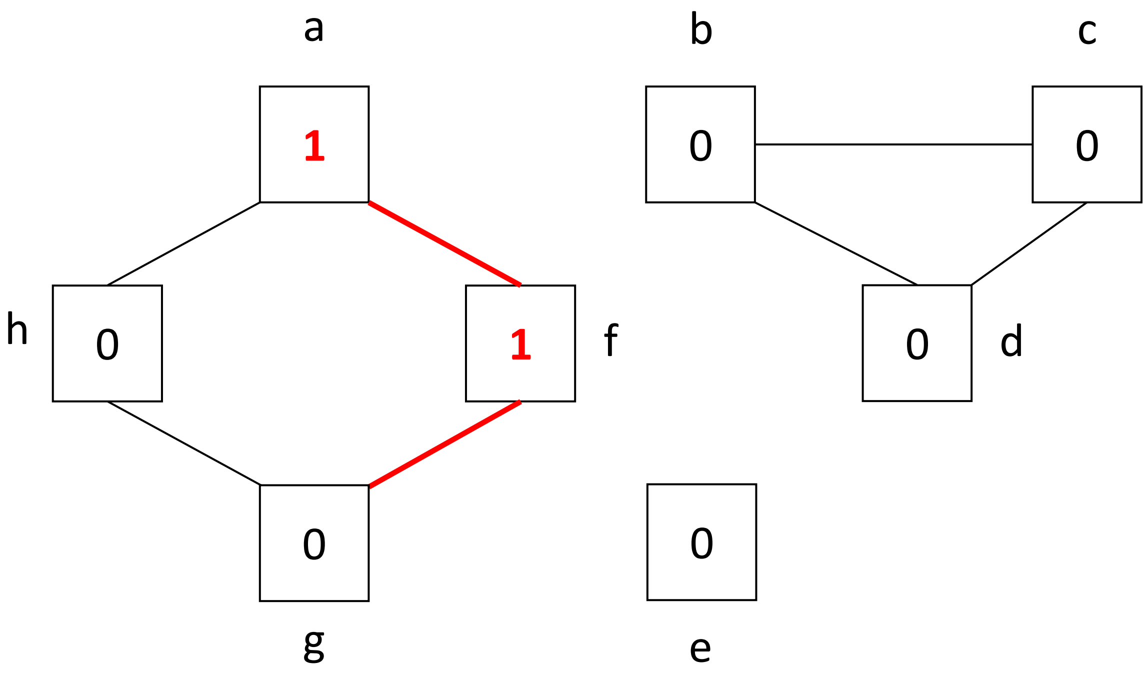
Algorithme CC – PeP – Exemple



ncc = 1

procédure **Visiter2**(x)
    c[x] ← Gris ;   cc[i] ← ncc ;
    d[x] ← temps ;   temps ← temps + 1;
    pour y ∈ $Adj[x]$ faire
        si (c[y] = Blanc) alors
            p[y] ← x;
            Visiter2(y);
        fin si
    fin pour
    c[x] ← Noir;
    f[x] ← temps ;   temps ← temps + 1;

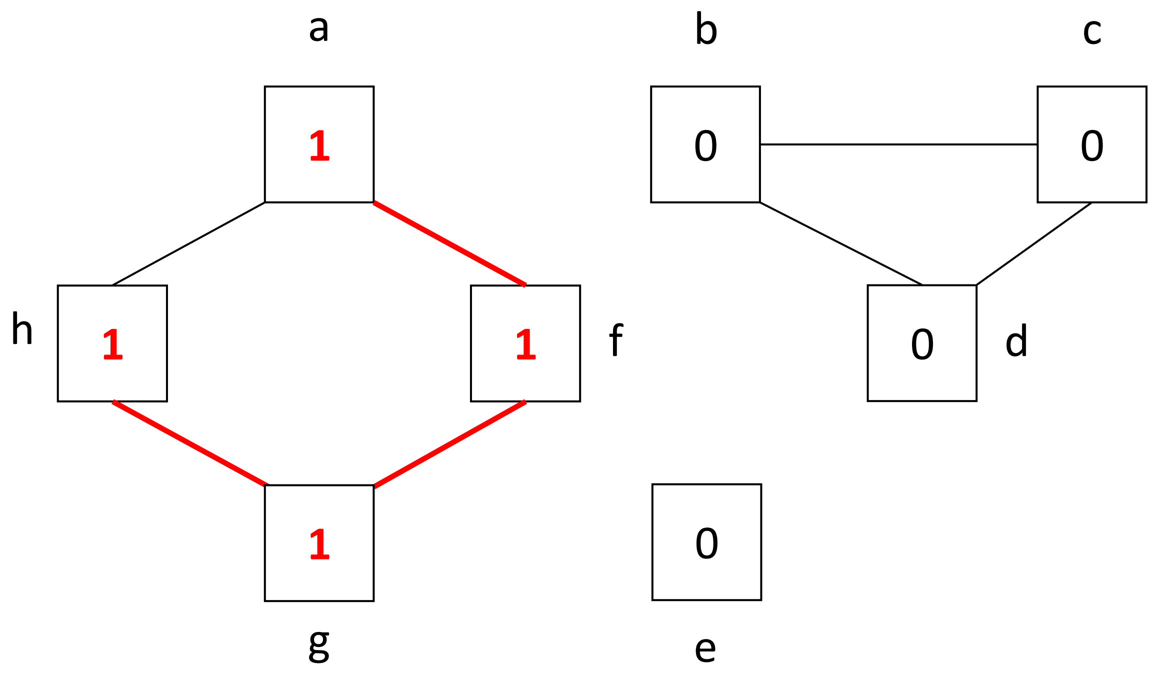
Algorithme CC – PeP – Exemple



ncc = 1

procédure **Visiter2**(x)
    c[x] ← Gris ;   cc[i] ← ncc ;
    d[x] ← temps ;   temps ← temps + 1;
    pour y ∈ $Adj[x]$ faire
        si (c[y] = Blanc) alors
            p[y] ← x;
            Visiter2(y);
        fin si
    fin pour
    c[x] ← Noir;
    f[x] ← temps ;   temps ← temps + 1;

Algorithme CC – PeP – Exemple



ncc = 1

procédure **Visiter2**(x)

  c[x] ← Gris ;   cc[i] ← ncc ;

  d[x] ← temps ;   temps ← temps + 1;

  pour y ∈ Adj[x] faire

    si (c[y] = Blanc) alors

      p[y] ← x;

      Visiter2(y);

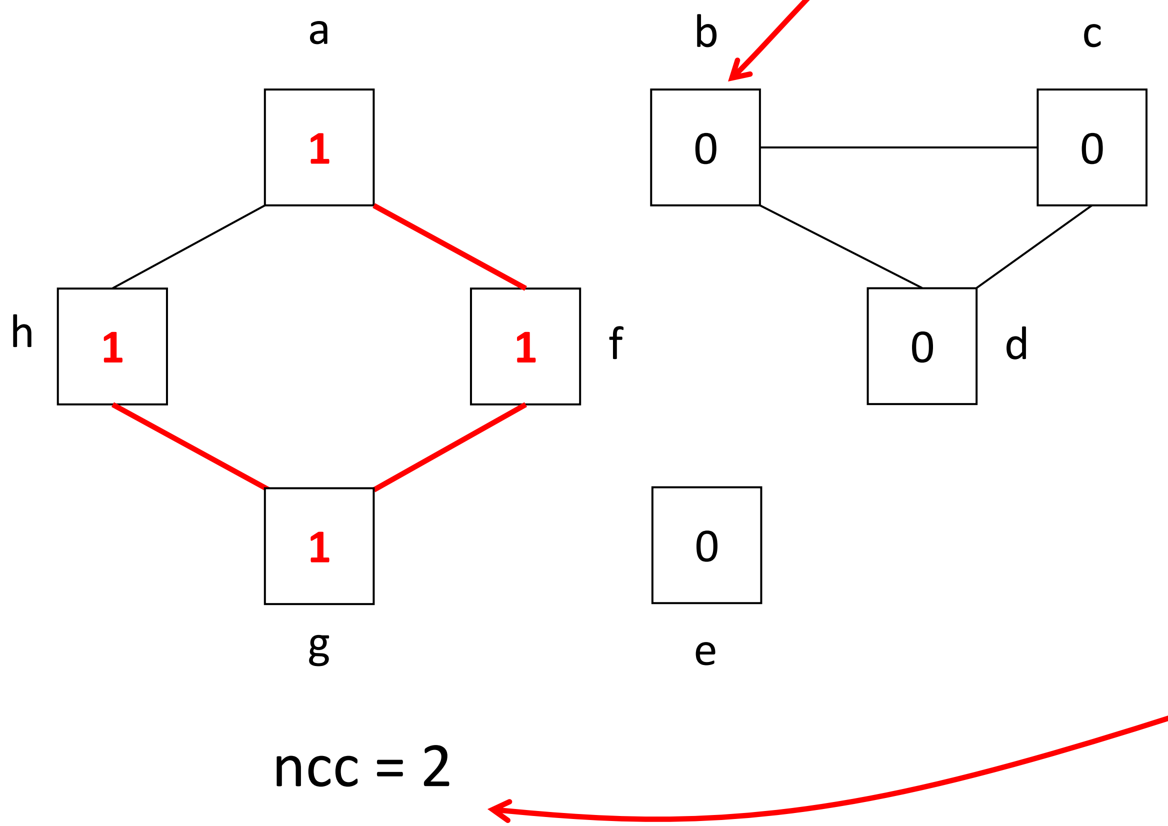
    fin si

  fin pour

  c[x] ← Noir;

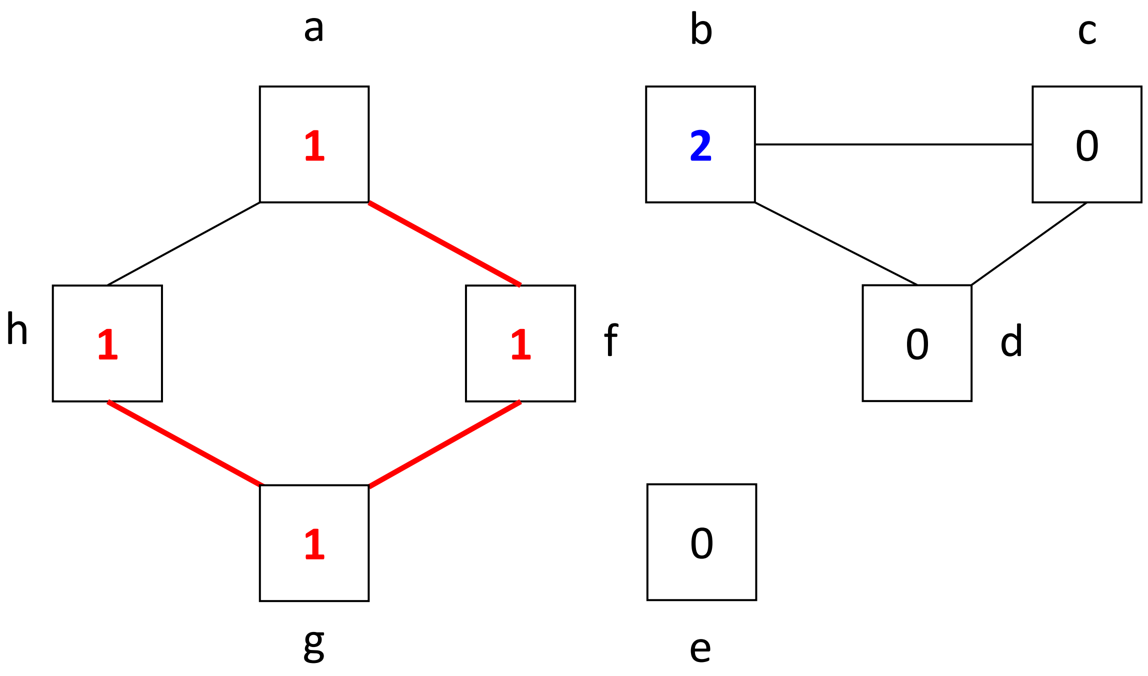
  f[x] ← temps ;   temps ← temps + 1;

Algorithme CC – PeP – Exemple



procédure CC-PEP (Entrée : G
Sortie : d, f, p)
pour x ∈ S faire
c[x] ← Blanc ; p[x] ← Nul ;
cc[x] ← 0 ;
fin pour
temps ← 1; ncc ← 0 ;
pour x ∈ S faire
si(c[x] = Blanc) alors
ncc ← ncc + 1 ;
Visiter2(x) ;
fin si
fin pour

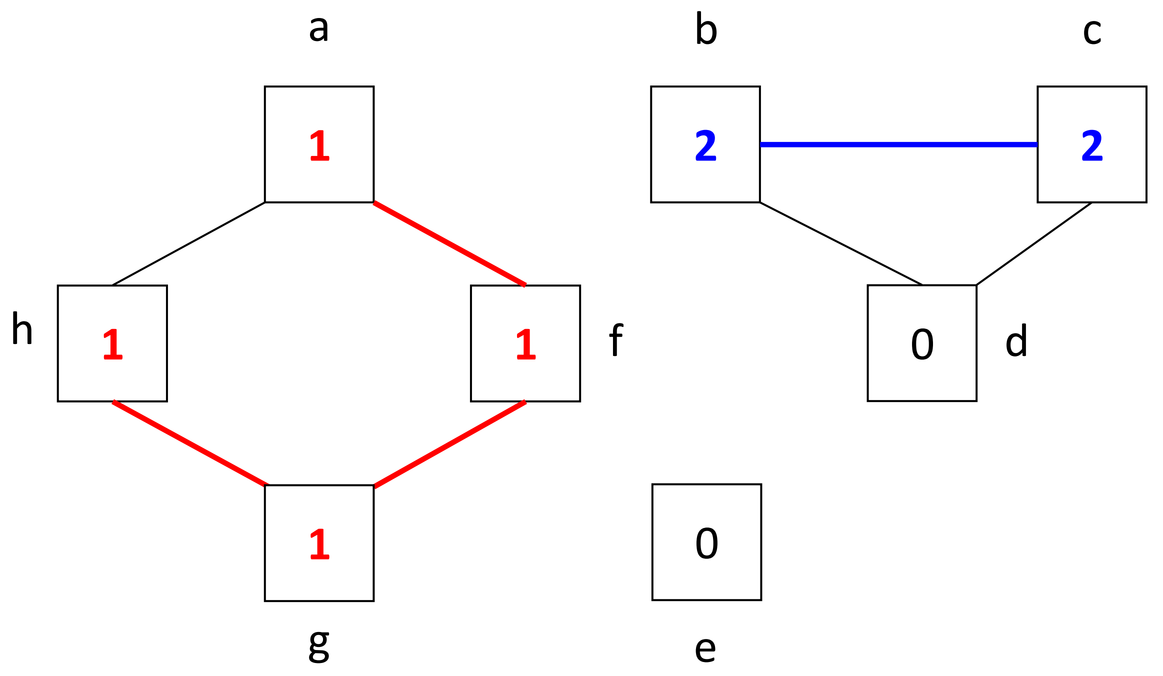
Algorithme CC – PeP – Exemple



ncc = 2

procédure **Visiter2**(x)
    c[x] ← Gris ;   cc[i] ← ncc ;
    d[x] ← temps ;   temps ← temps + 1;
    pour y ∈ $Adj[\mathrm{x}]$ faire
        si (c[y] = Blanc) alors
            p[y] ← x;
            Visiter2(y);
        fin si
    fin pour
    c[x] ← Noir;
    f[x] ← temps ;   temps ← temps + 1;

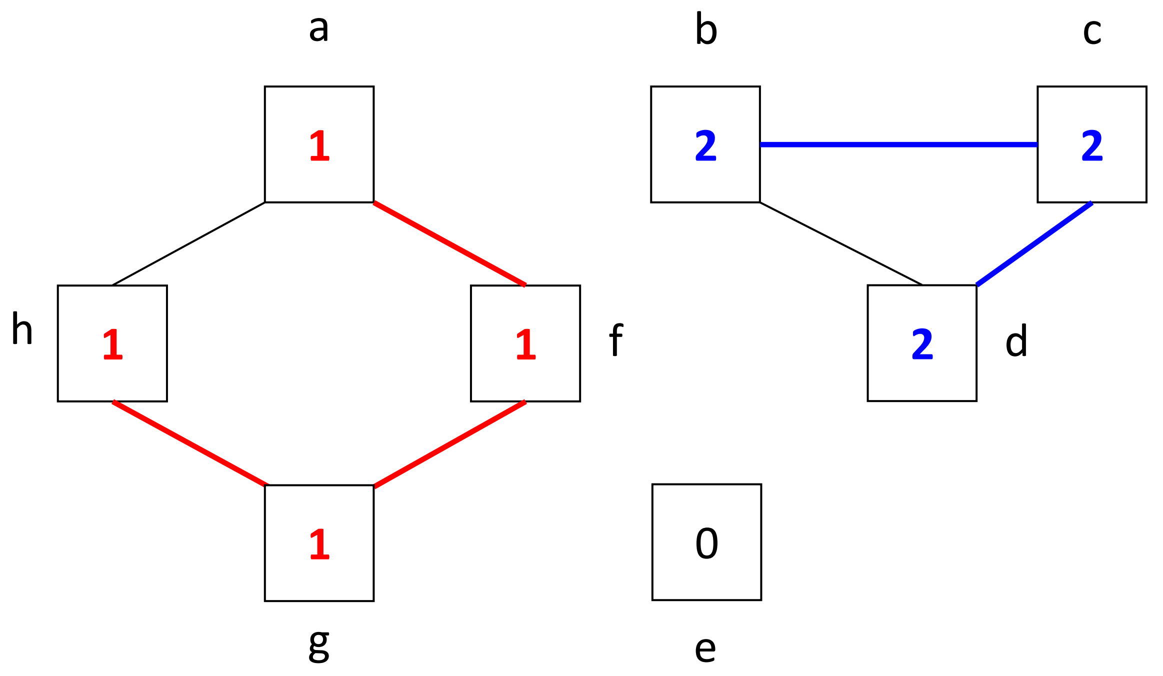
Algorithme CC – PeP – Exemple



ncc = 2

procédure **Visiter2**(x)
    c[x] ← Gris ;   cc[i] ← ncc ;
    d[x] ← temps ;   temps ← temps + 1;
    pour y ∈ $Adj[\mathrm{x}]$ faire
        si (c[y] = Blanc) alors
            p[y] ← x;
            Visiter2(y);
        fin si
    fin pour
    c[x] ← Noir;
    f[x] ← temps ;   temps ← temps + 1;

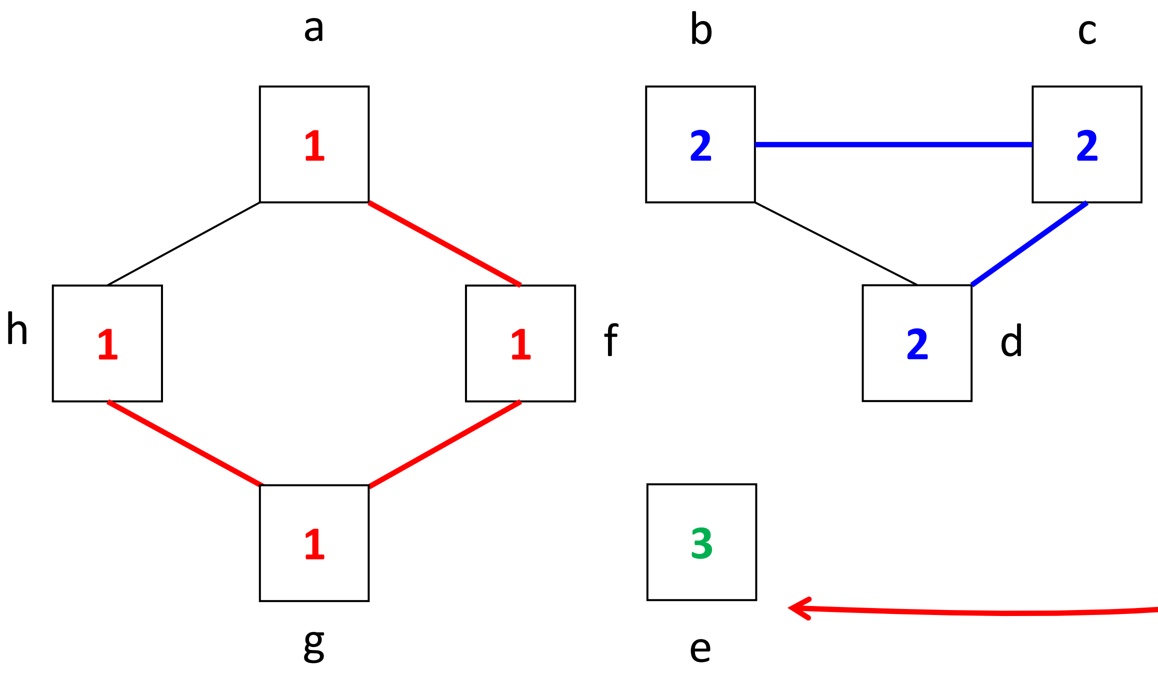
Algorithme CC – PeP – Exemple



ncc = 2

procédure **Visiter2**(x)
    c[x] ← Gris ;   cc[i] ← ncc ;
    d[x] ← temps ;   temps ← temps + 1;
    pour y ∈ Adj[x] faire
        si (c[y] = Blanc) alors
            p[y] ← x;
            Visiter2(y);
        fin si
    fin pour
    c[x] ← Noir;
    f[x] ← temps ;   temps ← temps + 1;

Algorithme CC – PeP – Exemple



ncc = 3

```
procédure CC-PEP (Entrée : G
                  Sortie : d, f, p)
  pour x ∈ S faire
    c[x] ← Blanc ;   p[x] ← Nul ;
    cc[x] ← 0 ;
  fin pour
  temps ← 1;  ncc ← 0 ;
  pour x ∈ S faire
    si(c[x] = Blanc) alors
        ncc ← ncc + 1 ;
        Visiter2(x) ;
    fin si
  fin pour
```

Accessibilité et multiplication de matrices

- Soient $A = (a_{ij})$ et $B = (b_{ij})$ deux matrices carrées d'ordre n, le produit $C = A \times B = (c_{ij})$ est défini par

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = a_{i1} b_{1j} + \cdots + a_{in} b_{nj}$$

$$A = \begin{pmatrix} 2 & -1 & 0 \\ 3 & -5 & 1 \\ 2 & -3 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 2 & -1 \\ 0 & 7 & 4 \\ -1 & 0 & 6 \end{pmatrix}$$

$$A \times B = \begin{pmatrix} 2 & -3 & -6 \\ 2 & -29 & -17 \\ 1 & -17 & -8 \end{pmatrix} \qquad B \times A = \begin{pmatrix} 6 & -8 & 1 \\ 29 & -47 & 11 \\ 10 & -17 & 6 \end{pmatrix}$$

Multiplication de matrices booléennes

- Soient $A = (a_{ij})$ et $B = (b_{ij})$ deux matrices booléennes carrées d'ordre n, le produit $C = A \otimes B = (c_{ij})$ est défini par

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \otimes b_{kj} = a_{i1} \otimes b_{1j} \oplus \cdots \oplus a_{in} \otimes b_{nj}$$

a	b	$a \otimes b$	$a \oplus b$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

$$a \otimes b = \min(a, b)$$
$$a \oplus b = \max(a, b)$$

Accessibilité et matrices booléennes

- Soit M la matrice d'adjacence d'un graphe $G = (S, A)$
  - On a, pour $i, j \in S$

$$M[i,j] = \begin{cases} 1 & \text{si } (i,j) \in A \\ 0 & \text{sinon} \end{cases}$$

- Soit k un entier positif, on note

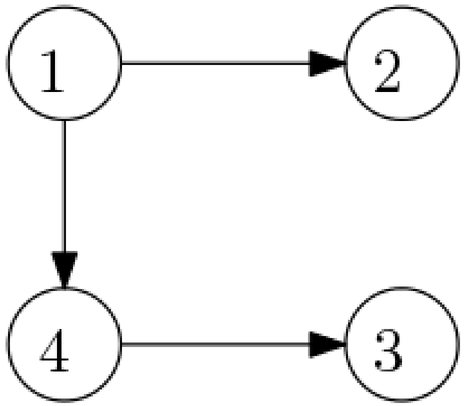
$$M^{(k+1)} = \begin{cases} M & \text{si } k = 0 \\ M \otimes M^{(k)} & \text{si } k > 0 \end{cases}$$

  - Avec la convention $M^{(0)} = I$
  - où I est la matrice identité $I[i,j] = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{sinon} \end{cases}$

Accessibilité et matrices booléennes

- **Proposition :** soient k un entier positif et M la matrice d'adjacence d'un graphe $G = (S, A)$, pour $i, j \in S$

$$M^{(k)}[i,j] = \begin{cases} 1 & \text{s'il existe une} \begin{cases} \text{un chemin} \\ \text{chaine de } i \text{ à } j \text{ de longueur } k \end{cases} \\ 0 & \text{sinon} \end{cases}$$



M	1	2	3	4
1	0	1	0	1
2	0	0	0	0
3	0	0	0	0
4	0	0	1	0

$M^{(2)}$	1	2	3	4
1	0	0	1	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

Accessibilité et matrices booléennes

- **Proposition :** soient k un entier positif et M la matrice d'adjacence d'un graphe $G = (S, A)$, pour $i, j \in S$

$$M^{(k)}[i,j] = \begin{cases} 1 & \text{s'il existe une} \begin{cases} \text{un chemin} \\ \text{chaine de } i \text{ à } j \text{ de longueur } k \end{cases} \\ 0 & \text{sinon} \end{cases}$$

- **Corollaire :** la suite $M^{(k)}$ est stationnaire à partir du rang $n - 1$:

$$M^{(k)} = M^{(n-1)}, \forall k \geq n$$

Accessibilité et matrices booléennes

- Soit $M^{[k]} = \sum_{h=0}^{k} M^{(h)} = M^{(0)} \oplus M^{(1)} \oplus \cdots \oplus M^{(k)}$
- **Proposition 2 :** soient k un entier positif et M la matrice d'adjacence d'un graphe $G = (S, A)$, pour $i, j \in S$

$$M^{[k]}[i,j] = \begin{cases} 1 & \text{s'il existe une} \left\{ \begin{array}{l} \text{un chemin} \\ \text{chaine de } i \text{ à } j \end{array} \right. \text{de longueur } \leq k \\ 0 & \text{sinon} \end{cases}$$

- **Corollaire** : la suite $M^{[k]}$ est stationnaire à partir du rang $n - 1$:

$$M^{[k]} = M^{[n-1]}, \forall k \geq n$$

Conséquences

- Soit $M^{[k]} = \sum_{h=0}^{k} M^{(h)} = M^{(0)} \oplus M^{(1)} \oplus \cdots \oplus M^{(k)}$
- Proposition 2 : Soient k un entier positif et M la matrice d'adjacence d'un graphe $G = (S, A)$, pour $i, j \in S$

$$M^{[k]}[i,j] = \begin{cases} 1 & \text{s'il existe une} \begin{cases} \text{un chemin} \\ \text{chaine de } i \text{ à } j \text{ de longueur } \leq k \end{cases} \\ 0 & \text{sinon} \end{cases}$$

- $\boldsymbol{M^{[k]}}$ est la matrice d'adjacence du graphe $\boldsymbol{G^k = (S, V^k)}$
- $\boldsymbol{M^{[n-1]}}$ est la matrice d'adjacence de la ferm. transitive $\boldsymbol{G^* = (S, V^*)}$
- **Théorème :** G connexe $\Leftrightarrow M^{[n-1]} = \mathbf{1}$

CC : algorithme naïf

- $M^{(n-1)} = \sum_{h=0}^{n-1} M^{(h)} = M^{(0)} \oplus M^{(1)} \oplus \cdots \oplus M^{(n-1)}$
- Calcul de $M^{(0)}$
- Calcul de $M^{(1)}$
- Calcul de $M^{(2)}$
- …
- Calcul de $M^{(k)}$
- …
- Calcul de $M^{(n-1)}$

multiplication ($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$)

pour $i = 1$ à n faire
  pour $j = 1$ à n faire
    $C[i,j] = 0$;
    pour $k = 1$ à n faire
      $C[i,j] = \max(C[i,j], A[i,k] \times B[k,j])$;
    fin pour
  fin pour
fin pour

$O(n^3)$

addition($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$)

pour $i = 1$ à n faire
  pour $j = 1$ à n faire
    $C[i,j] = \min(A[i,j], B[i,j])$
  fin pour
fin pour

$O(n^2)$

CC : algorithme naïf

- $M^{(n-1)} = \sum_{h=0}^{n-1} M^{(h)} = M^{(0)} \oplus M^{(1)} \oplus \cdots \oplus M^{(n-1)}$
- Calcul de $M^{(0)}$ $O(1)$
- Calcul de $M^{(1)}$ $O(n^2)$
- Calcul de $M^{(2)}$ $O(n^3)$
- …
- Calcul de $M^{(k)}$ $O((k-1)n^3)$
- …
- Calcul de $M^{(n-1)}$ $O((n-2)n^3)$

Total : $O(n^4)$

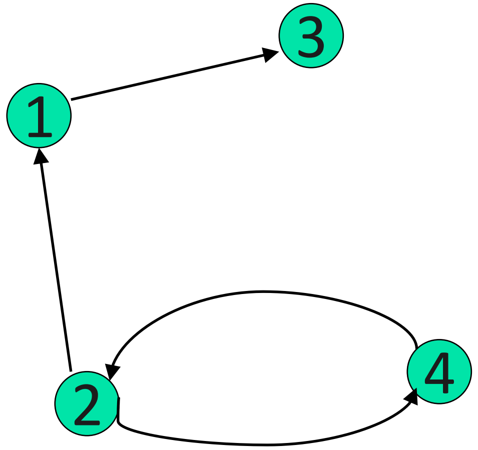
CC : algorithme amélioré

- $M^{(n-1)} = \sum_{h=0}^{n-1} M^{(h)} = M^{(0)} \oplus M^{(1)} \oplus \cdots \oplus M^{(n-1)}$
- Calcul de $X = M^{(k)}$ avec $k = 2^p$
- Exemple : $k = 16 = 2^4$
- $X = M$
- $X = X \otimes X \left(= M^{(2)}\right)$
- $X = X \otimes X \left(= M^{(4)}\right)$
- $X = X \otimes X \left(= M^{(8)}\right)$
- $X = X \otimes X \left(= M^{(16)}\right)$

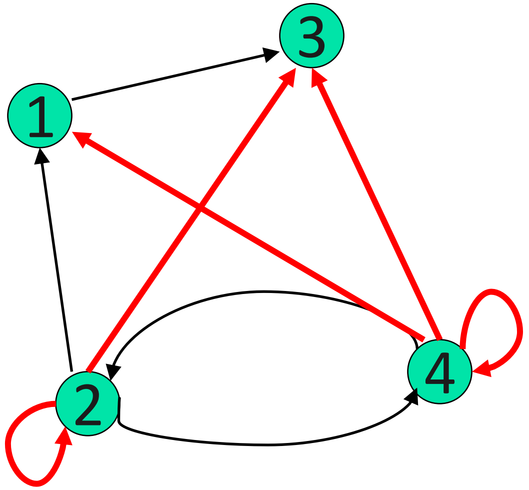
Total : $O(n^3 \log(n))$

Algorithme de Warshall

- But : déterminer la fermeture transitive d'un graphe
- $G = (S, A)$ $\rightarrow$ M sa matrice d'adjacence
- $G^* = (S, A^*)$ $\rightarrow$ M^* sa matrice d'adjacence



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

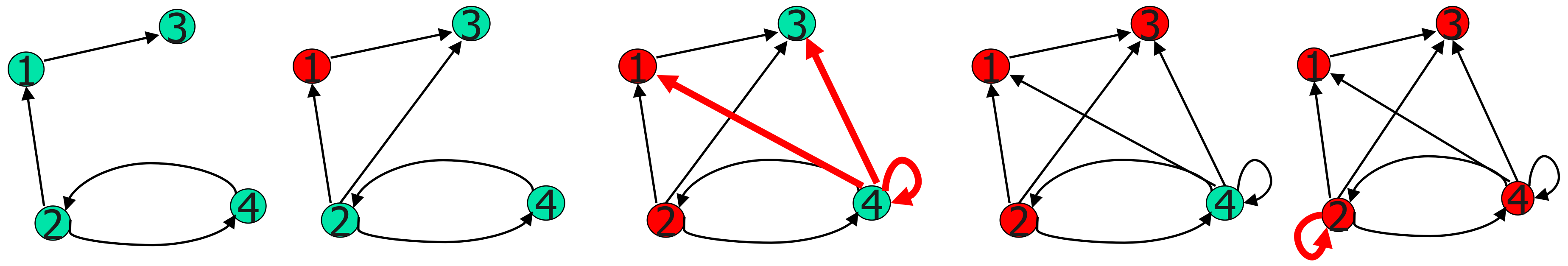


$$\begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{matrix}$$

Calculer $M^* = M^{[n-1]} = M^{(0)} \oplus M^{(1)} \oplus \cdots \oplus M^{(n-1)}$

Algorithme de Warshall

- **Principe :** un chemin existe entre i et j, si et seulement si
  - il y a un arc de i à j; ou
  - il y a un chemin de i à j passant par le sommet 1; ou
  - il y a un chemin de i à j passant par les sommets 1 et / ou 2; ou
  - ...
  - il y a un chemin de i à j passant par l'un des autres sommets



Algorithme de Warshall

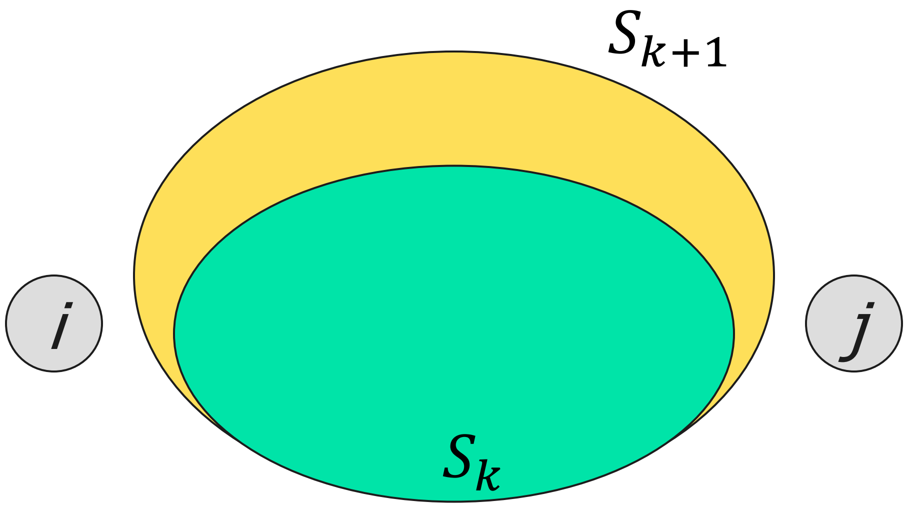
- Soit $\pi_{ij} = < i = s_1, s_2, \dots, s_{p-1}, j = s_p >$ un chemin de i à j, les sommets $\{s_2, \dots, s_{p-1}\}$ sont appelés des **sommets intermédiaires**
- Soit $S = \{1, \dots, n\}$, on définit $S_k = \{1, \dots, k\}$ avec $S_0 = \emptyset$
  $\rightarrow S_{k+1} = S_k + \{k\}$ et $S_n = S$
- Pour tout $i, j \in S$, on définit

$$W^k[i,j] = \begin{cases} 1 & \begin{array}{c} \text{s'il exite un chemin de } i \text{ à } j \text{ dont} \\ \text{les sommets intermédiaires sont dans } S_k \end{array} \\ 0 & \text{sinon} \end{cases}$$

  - Initialisation : $W^0[i,j] = M[i,j]$
  - Terminaison : $W^n[i,j] = M^*[i,j]$
  - Itération : $W^k[i,j] \rightarrow W^{k+1}[i,j]$ **?**

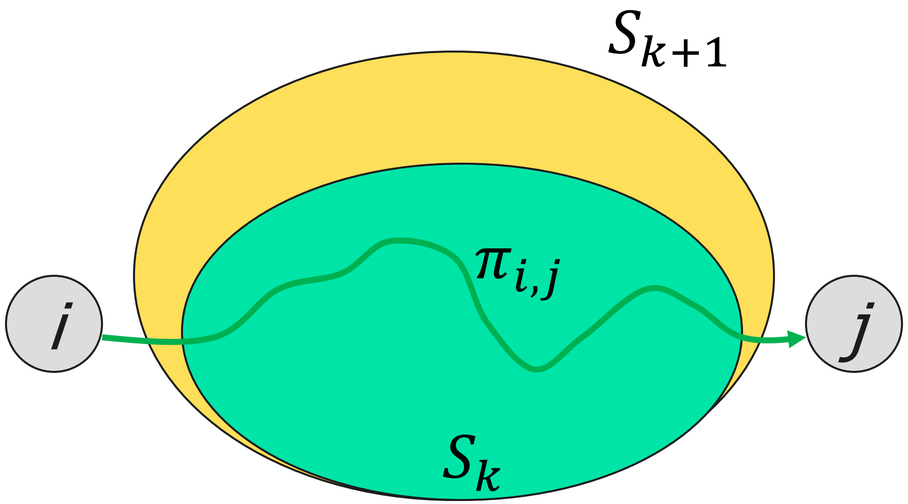
Algorithme de Warshall

- Itération : $W^k[i,j] \rightarrow W^{k+1}[i,j]$ **?**



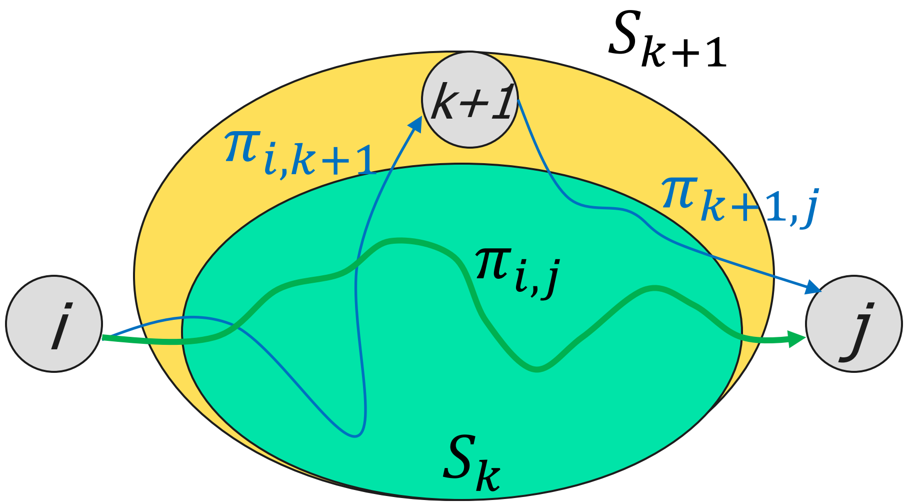
Algorithme de Warshall

- Itération : $W^k[i,j] \rightarrow W^{k+1}[i,j]$ **?**



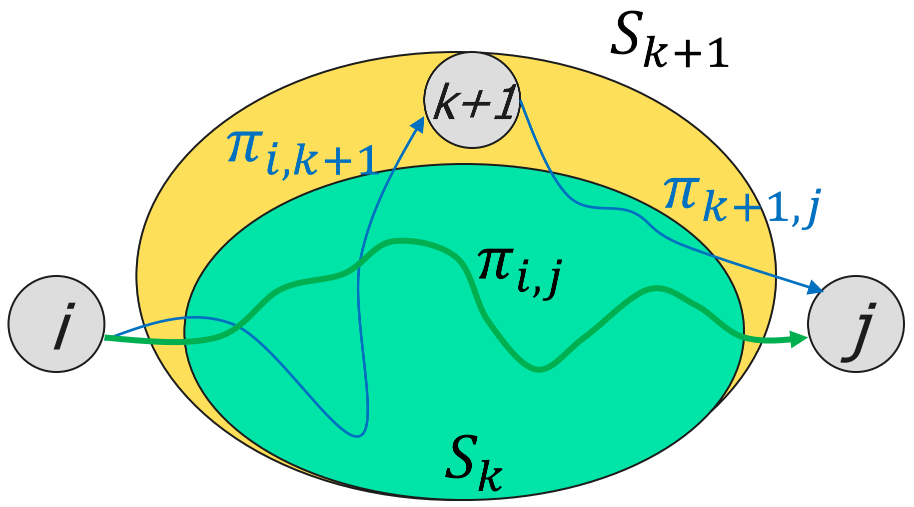
Algorithme de Warshall

- Itération : $W^k[i,j] \rightarrow W^{k+1}[i,j]$ **?**



Algorithme de Warshall

- Itération : $W^k[i,j] \rightarrow W^{k+1}[i,j]$
- $W^{k+1}[i,j] = \begin{cases} W^k[i,j] \textbf{ ou} \\ W^k[i,k+1] \textbf{ et } W^k[k+1,j] \end{cases}$



Algorithme de Warshall

```
procédure Warshall (Entrée M,
                    Sortie M*)
   pour i de 1 à n faire
      pour j de 1 à n faire
         M*[i,j] ← M[i, j]
      fin pour
   fin pour
```

Algorithme de Warshall

```
procédure Warshall (Entrée M,
                    Sortie M*)
  pour i de 1 à n faire
    pour j de 1 à n faire
      M*[i,j] ← M[i, j]
    fin pour
  fin pour
  pour k de 1 à n faire
    pour i de 1 à n faire
      pour j de 1 à n faire
        M*[i, j] ← M*[i, j] ou (M*[i, k] et M*[k, j])
      fin pour
    fin pour
  fin pour
```

Algorithme de Warshall

procédure Warshall (Entrée M,
Sortie M*)

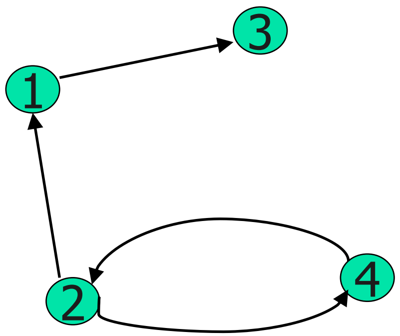
```
pour i de 1 à n faire
    pour j de 1 à n faire
        M*[i,j] ← M[i, j]
    fin pour
fin pour
```

```
pour k de 1 à n faire
    pour i de 1 à n faire
        si (M*[i, k] = 1) alors
            pour j de 1 à n faire
                M*[i, j] ← M*[i, j] ou M*[k, j]
            fin pour
        fin si
    fin pour
fin pour
```

Algorithme valide pour le cas non orienté et pour le cas orienté

Complexité : $O(n^3)$

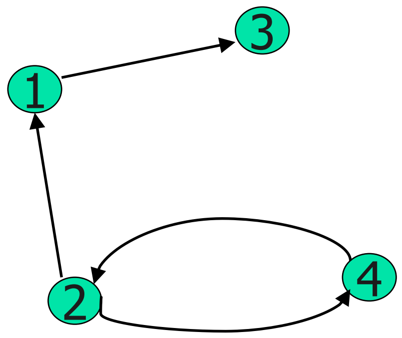
Algorithme de Warshall

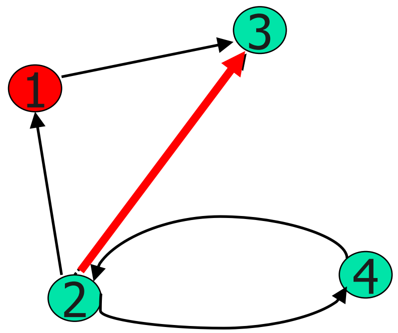


$W^0 = M$

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

Algorithme de Warshall





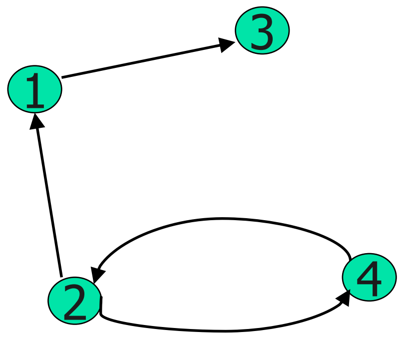
$W^0 = M$

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

W^1

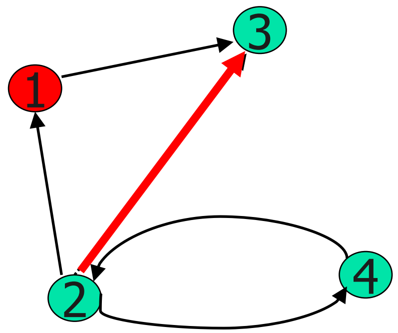
$$\begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & \mathbf{1} & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$$

Algorithme de Warshall



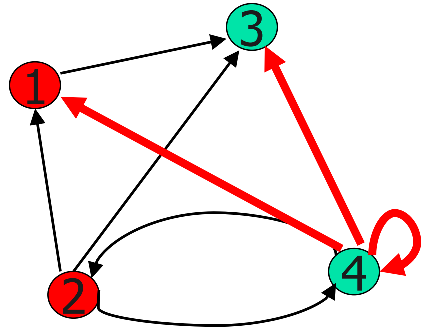
$W^0 = M$

0	0	1	0
1	0	0	1
0	0	0	0
0	1	0	0



W^1

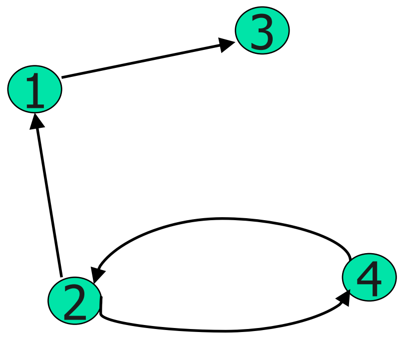
0	0	1	0
1	0	**1**	1
0	0	0	0
0	1	0	0



W^2

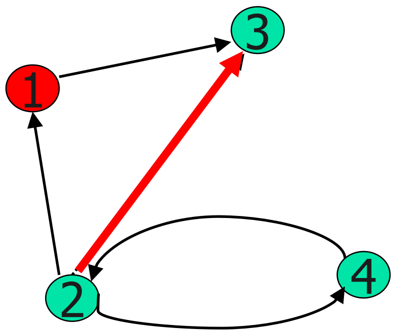
0	0	1	0
1	0	1	1
0	0	0	0
1	1	**1**	**1**

Algorithme de Warshall



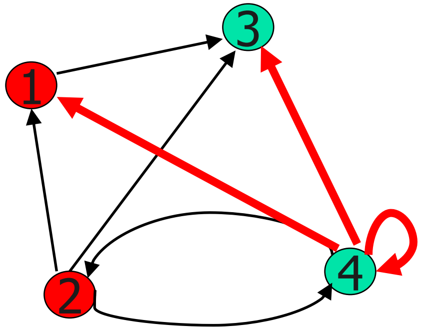
$W^0 = M$

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



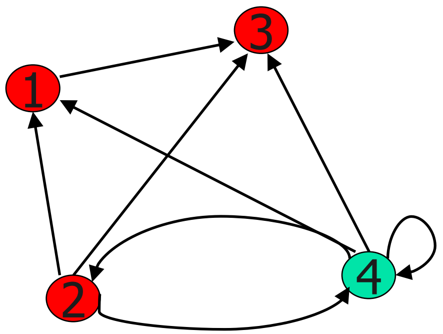
W^1

$$\begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & \mathbf{1} & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$$



W^2

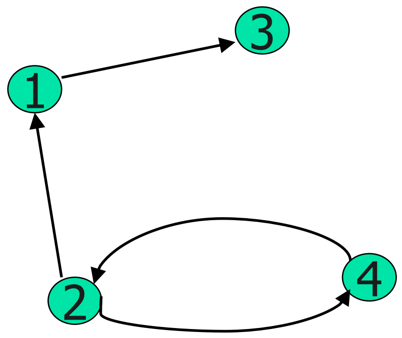
$$\begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ \mathbf{1} & 1 & \mathbf{1} & \mathbf{1} \end{matrix}$$



W^3

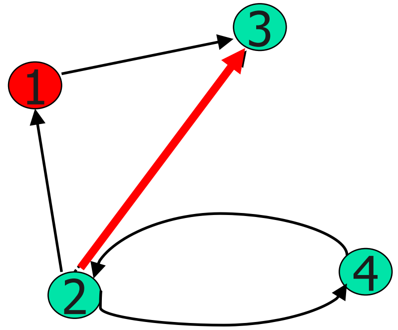
$$\begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{matrix}$$

Algorithme de Warshall



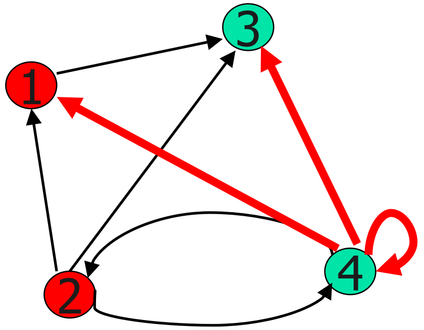
$W^0 = M$

$$\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{array}$$



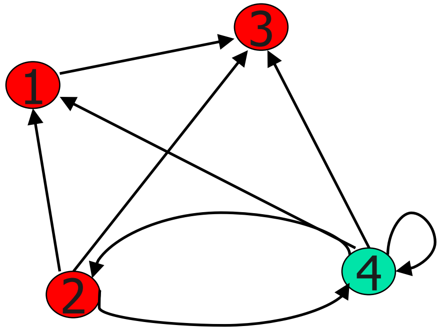
W^1

$$\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{array}$$



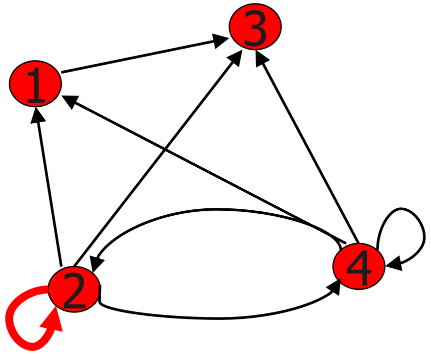
W^2

$$\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{array}$$



W^3

$$\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{array}$$

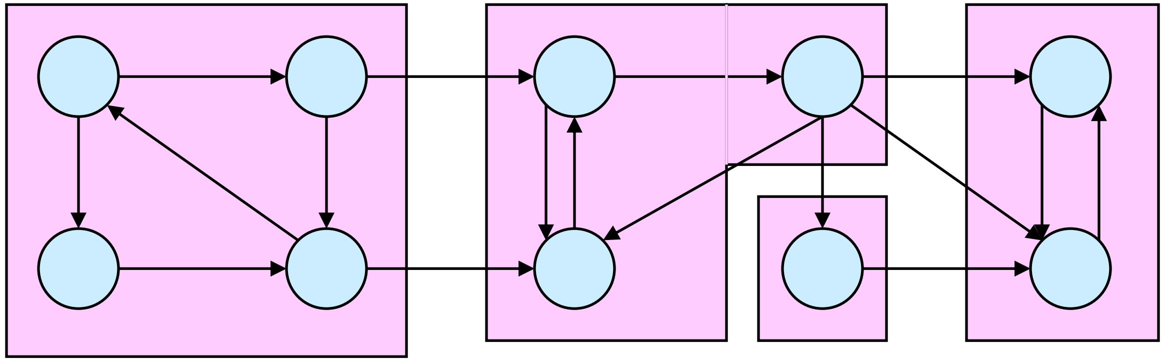


W^4

$$\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{array}$$

Forte connexité

- Un graphe orienté $G = (S, A)$ est fortement connexe s'il existe un chemin **de n'importe quel sommet vers n'importe quel autre** sommet
- Une **composante fortement connexe** (**CFC**) de G est un ensemble maximal de sommets $C \subseteq S$ tel que pour tout $i, j \in C$, il existe un chemin de i à j **et** un chemin de j à i



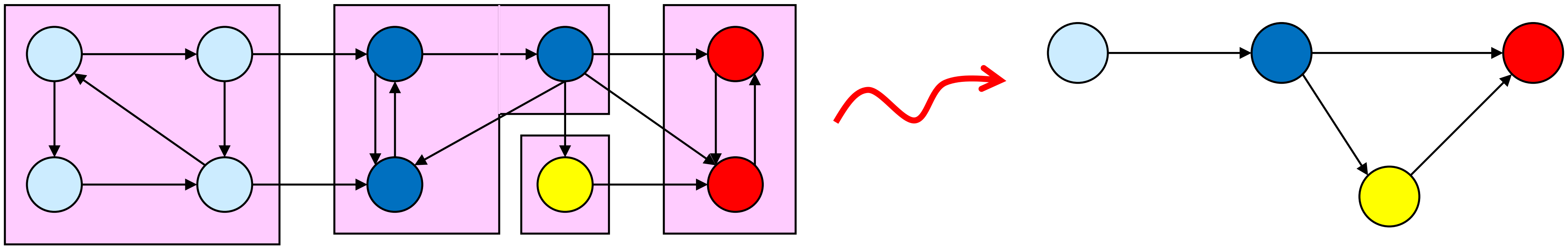
Forte connexité

- Soit $G = (S, A)$ un graphe orienté. On définit C_f la relation binaire de forte connexité sur S : pour tout $i, j \in S$
  - $i\ C_f\ j$ ⇔ (i = y) OU

    (∃ un chemin de i à j dans G **et** ∃ un chemin de j à i dans G)
- C_f est une relation d'équivalence
- Les classes d'équivalence de C_f sont les composantes **fortement** connexes de G
- G orienté est un graphe fortement connexe ⇔ G possède **une seule** composante fortement connexe

Forte connexité – Graphe réduit

- Soit $G = (S, A)$ un graphe orienté. Le **graphe réduit** de G est le graphe **quotient** $G_R = G/C_f = (S_R, A_R)$ avec
  - $S_R = S/C_f$: *classes d'équivalence de* C_f
  - $(\hat{\imath}, \hat{\jmath}) \in A_R \iff \exists i \in \hat{\imath}, \exists j \in \hat{\jmath}, (i, j) \in A$



$\rightarrow$ le graphe réduit G_R est **acyclique**

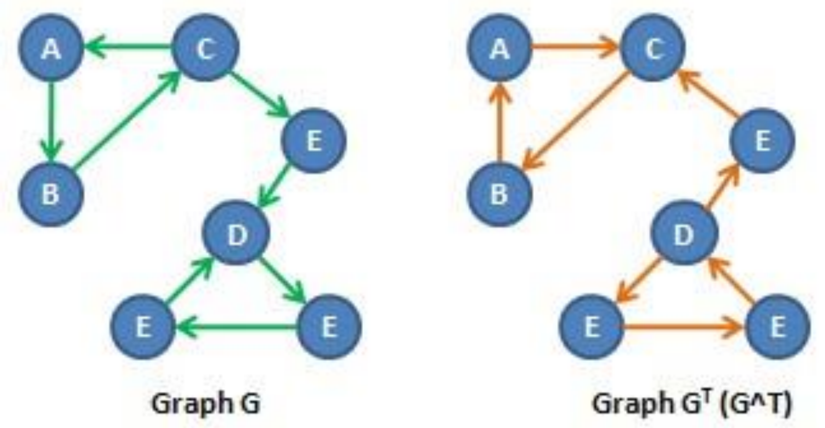
Forte connexité – Graphe réduit

- Le graphe réduit G_R est **acyclique**
- Preuve (par l'absurde)
  - Supposons qu'il existe un circuit $\hat{\pi}$ dans $G_R = (S_R, A_R)$
  - Soit $(\hat{\imath}, \hat{\jmath}) \in \hat{\pi} \Longrightarrow (\hat{\imath}, \hat{\jmath}) \in A_R$ et $\hat{\imath} \neq \hat{\jmath} \in S_R$
  - Par construction de G_R, si $\exists$ un chemin dans G_R alors $\exists$ un chemin dans G
  - $(\hat{\imath}, \hat{\jmath}) \in A_R \Longrightarrow \forall i \in \hat{\imath}$ et $\forall j \in \hat{\jmath}$, $\exists$ un chemin de i à j dans G
  - $\hat{\pi}$ est un circuit $\Longrightarrow \exists$ un chemin de $\hat{\jmath}$ à $\hat{\imath}$ dans G_R
  - $\Longrightarrow \exists$ un chemin de j à i dans G
  - $\Longrightarrow i$ et j sont dans la même CFC, d'où la contradiction

Graphe transposé

- Le graphe **transposé** d'un graphe orienté $G = (S, A)$ est

$$G^T = (S, A^T) \text{ avec } A^T = \{(i,j) : (j,i) \in A\}$$



- Remarques
  - On peut créer G^T en $O(n+m)$ si on utilise les listes d'adjacence
  - G et G^T ont les mêmes CFC

    $\rightarrow$ i et j sont accessibles dans G l'un à partir de l'autre si et seulement si ils le sont aussi dans G^T

Composantes fortement connexes – Algorithme

$CFC - PeP(G = (S, A))$

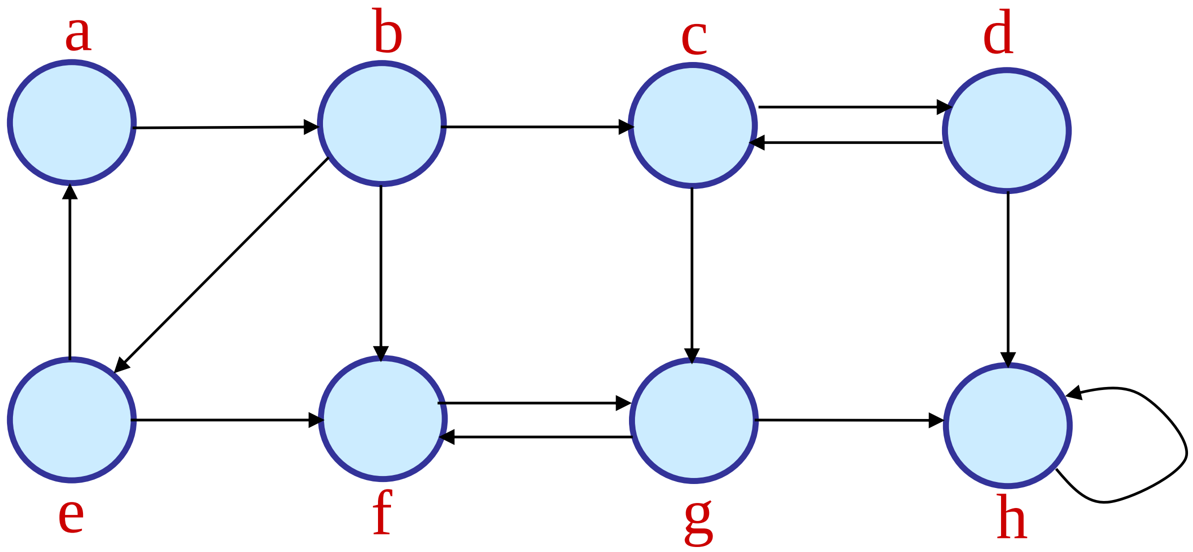
1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$
2. Calculer $G^T = (S, A^T)$
3. Appeler $PeP(G^T)$, en considérant les sommets dans l'ordre décroissant des $f[i]$
4. Les sommets de chaque arbre de la forêt formée dans le second appel de PeP forment une CFC

Compexité temporelle : $O(n + m)$

Composantes fortement connexes – Algorithme

Exemple

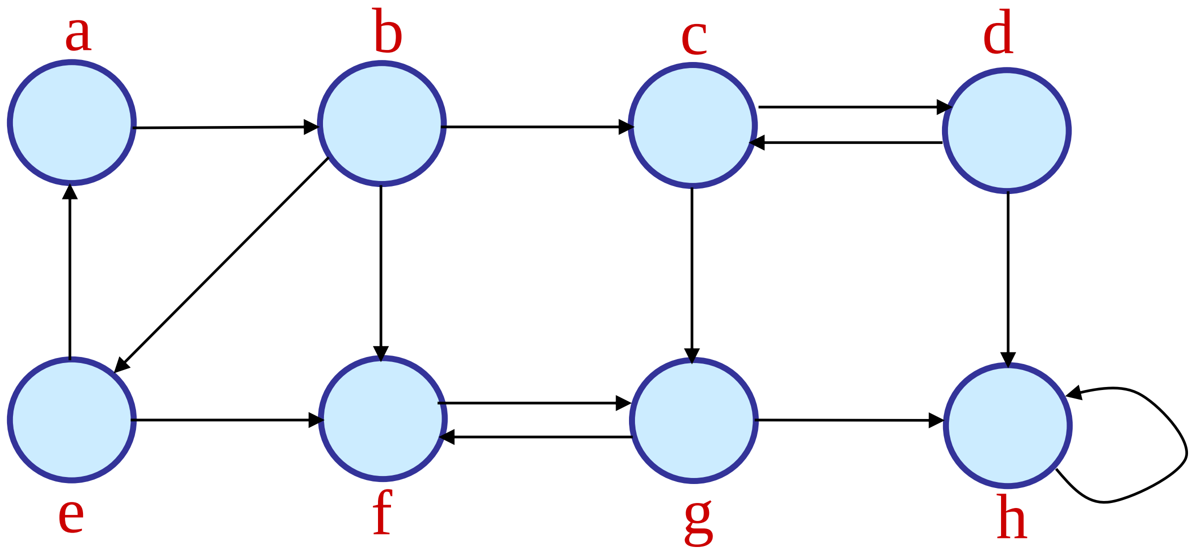
G



Composantes fortement connexes – Algorithme

Exemple

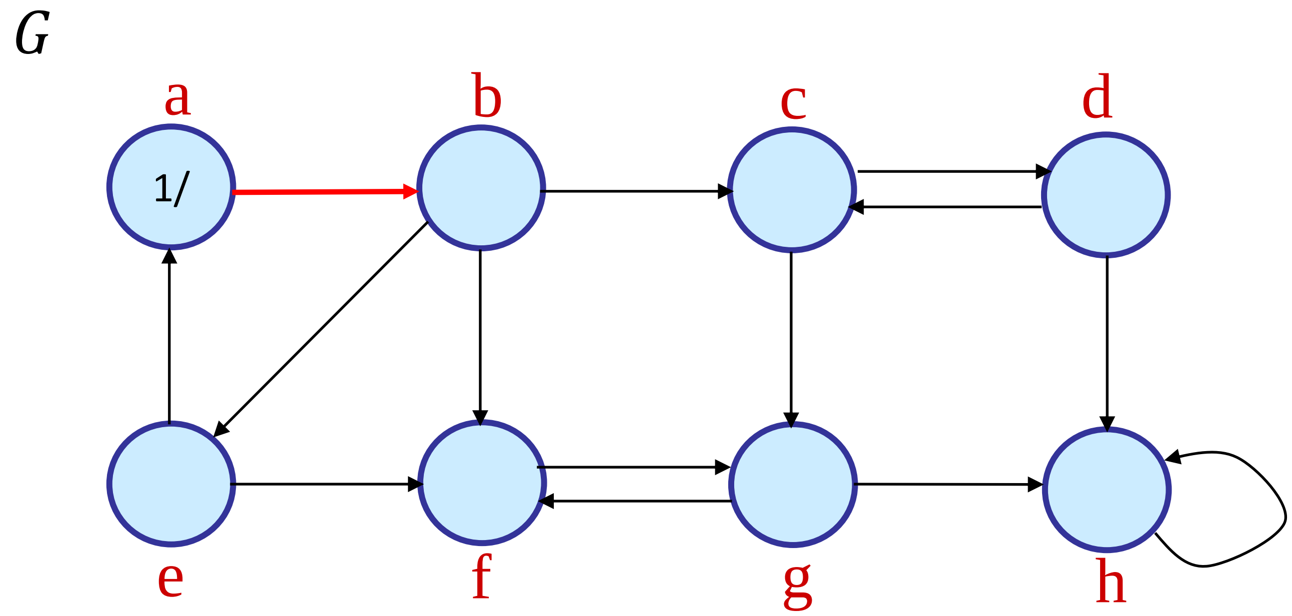
G



1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$

Composantes fortement connexes – Algorithme

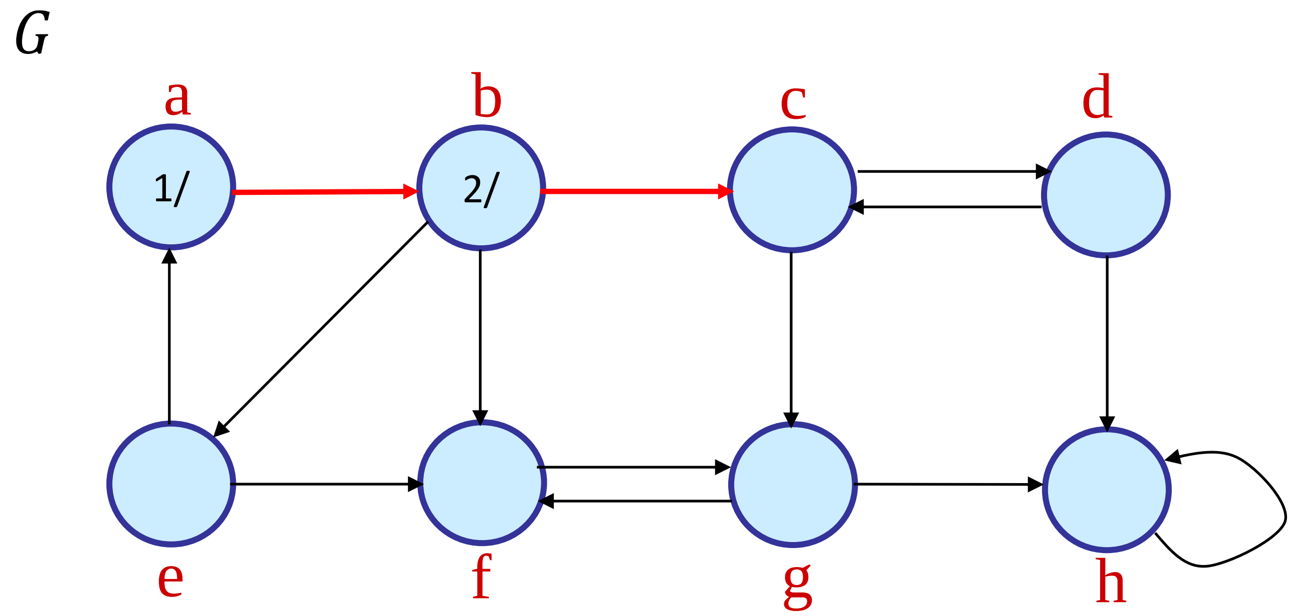
Exemple



1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$

Composantes fortement connexes – Algorithme

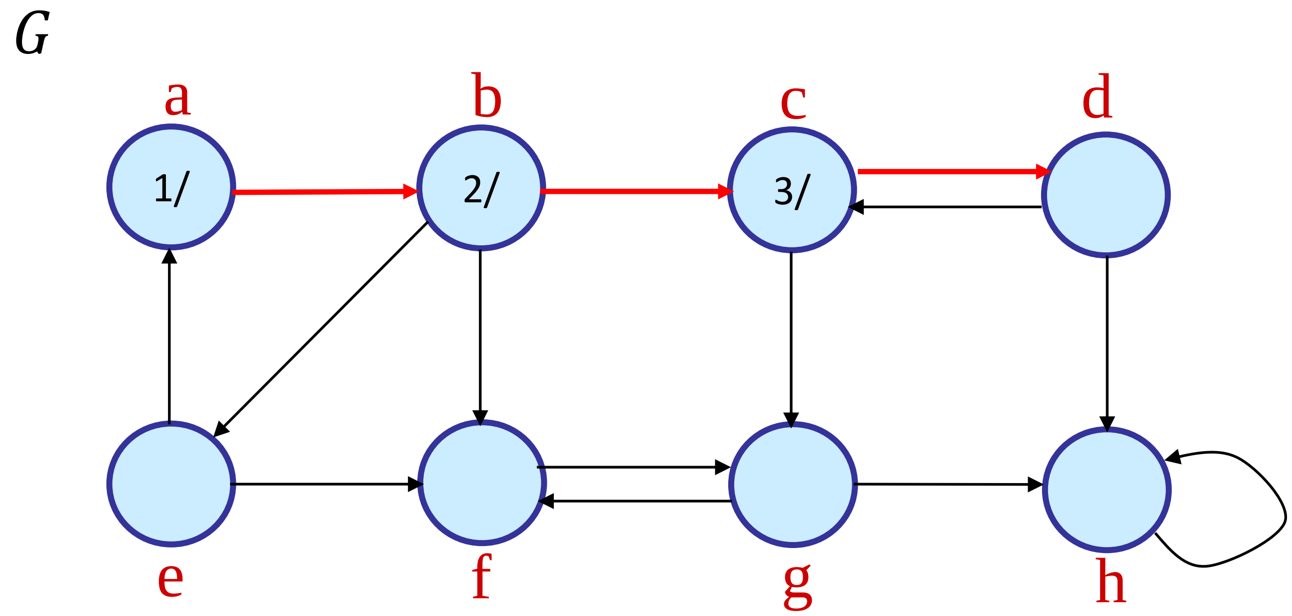
Exemple



1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$

Composantes fortement connexes – Algorithme

Exemple

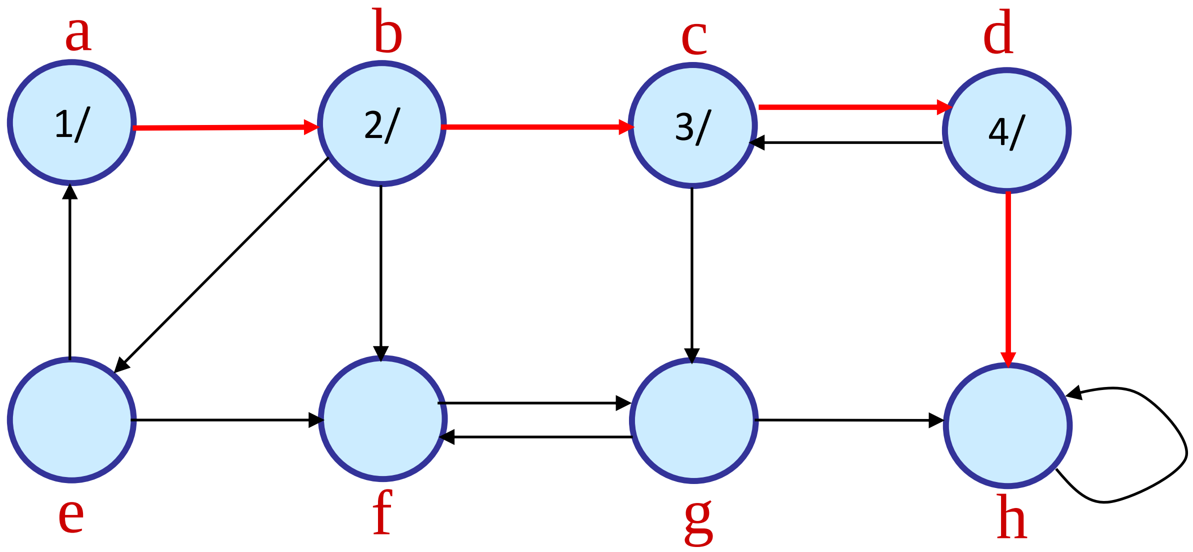


1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$

Composantes fortement connexes – Algorithme

Exemple

G

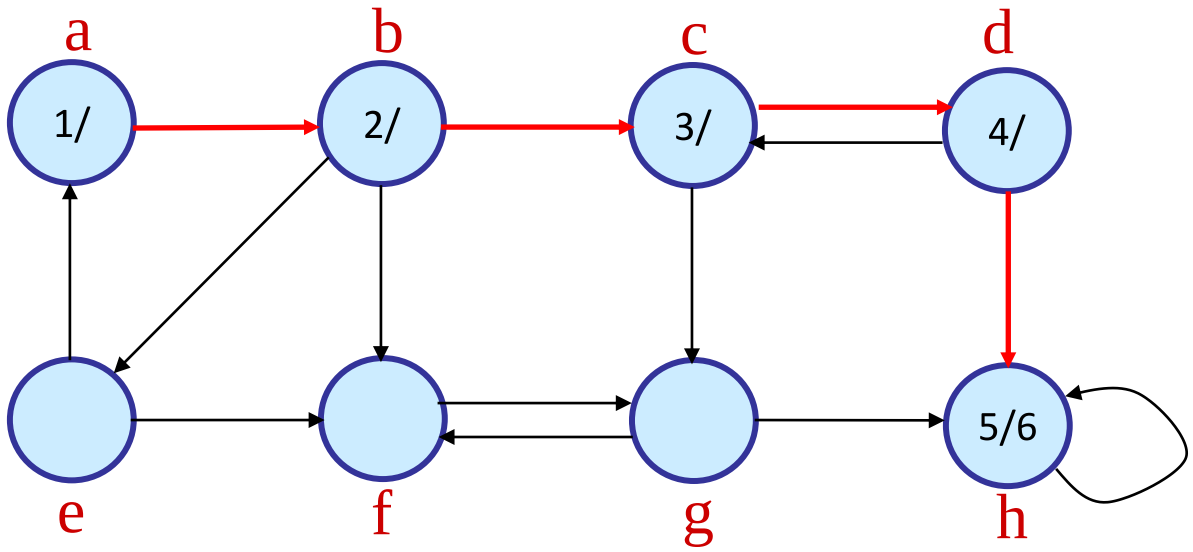


1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$

Composantes fortement connexes – Algorithme

Exemple

G

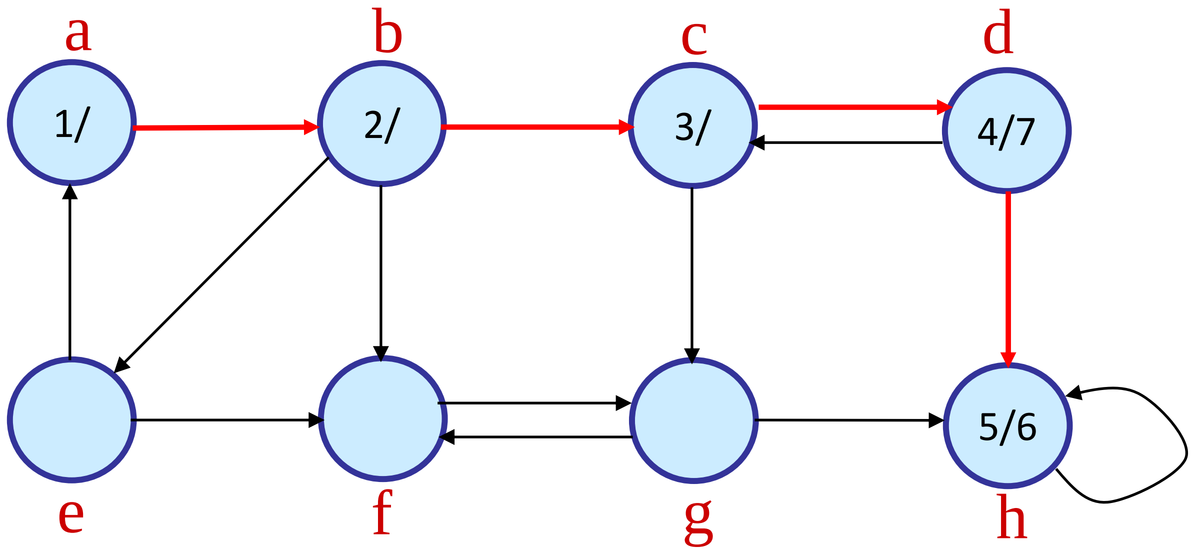


1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$

Composantes fortement connexes – Algorithme

Exemple

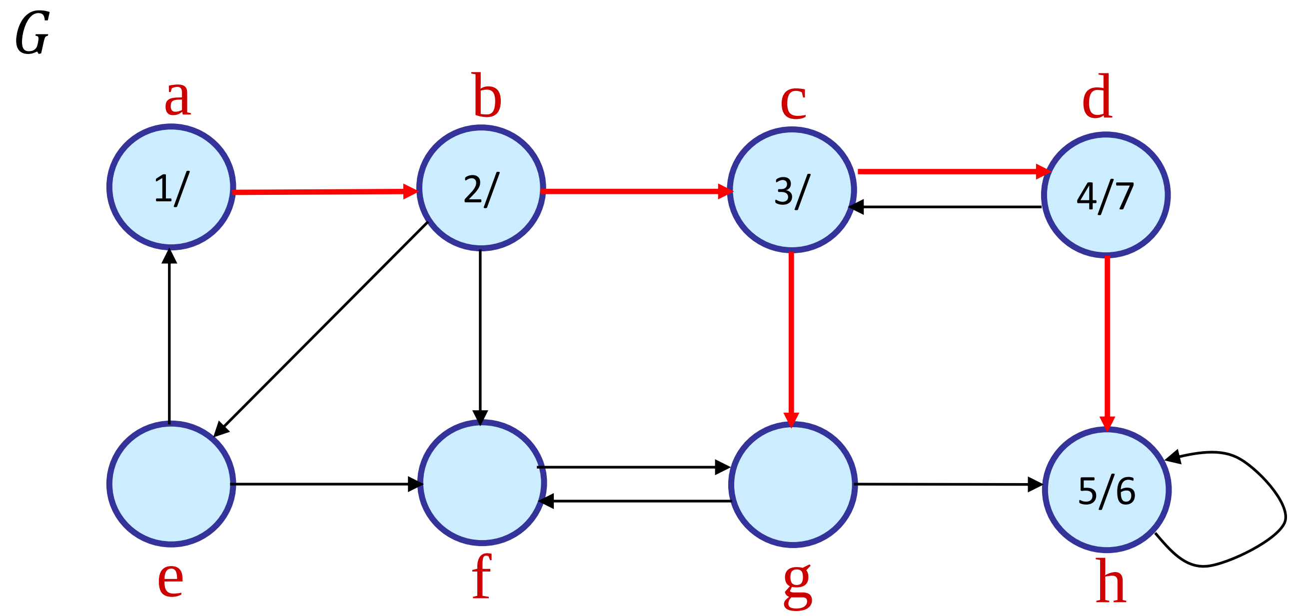
G



1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$

Composantes fortement connexes – Algorithme

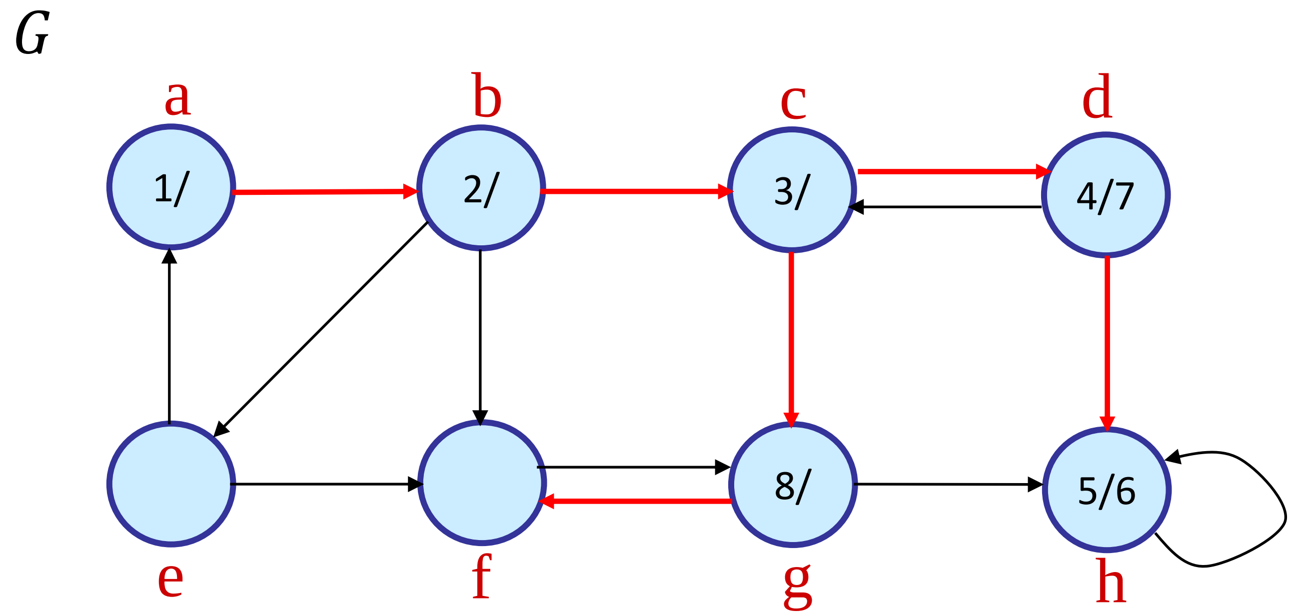
Exemple



1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$

Composantes fortement connexes – Algorithme

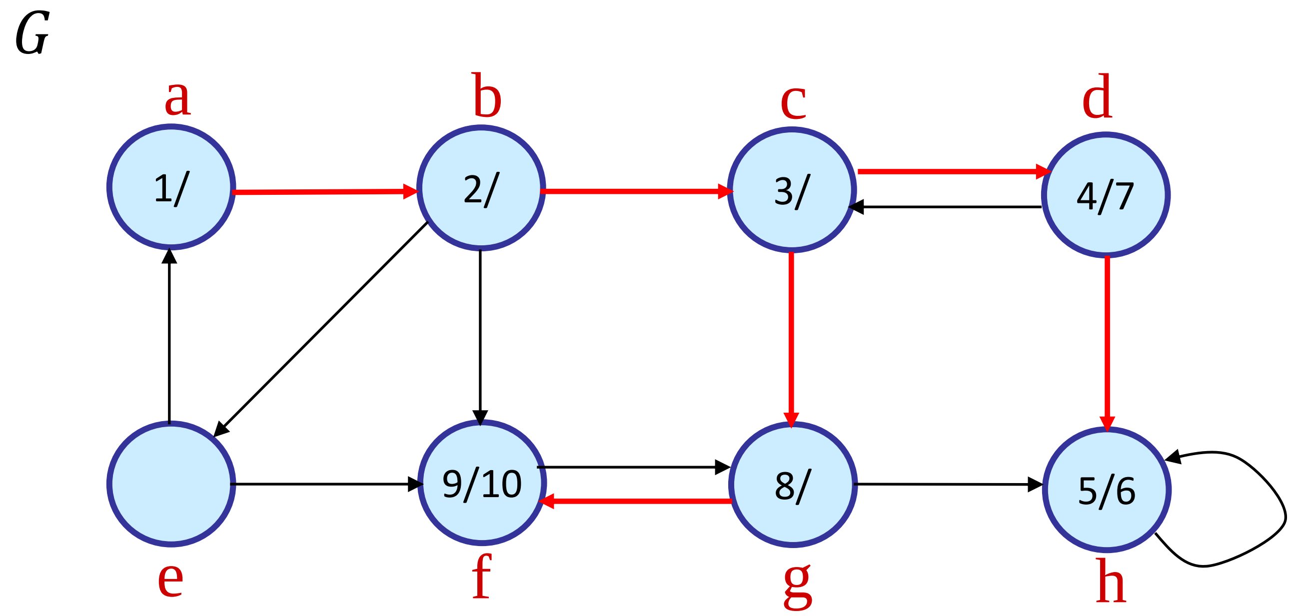
Exemple



1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$

Composantes fortement connexes – Algorithme

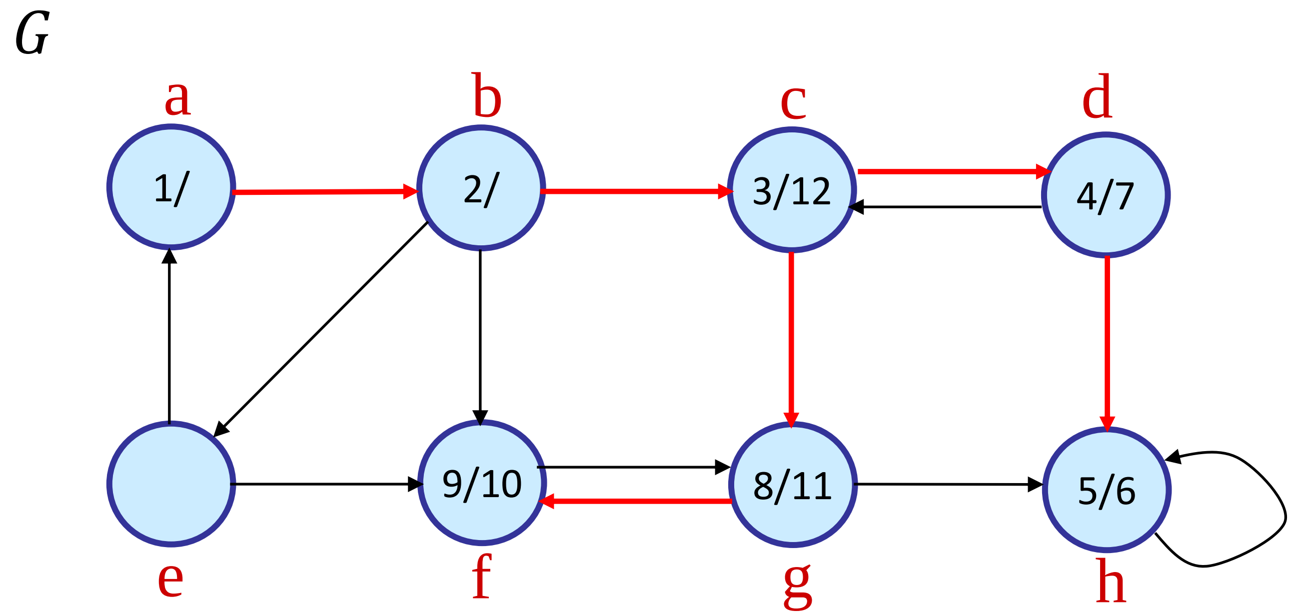
Exemple



1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$

Composantes fortement connexes – Algorithme

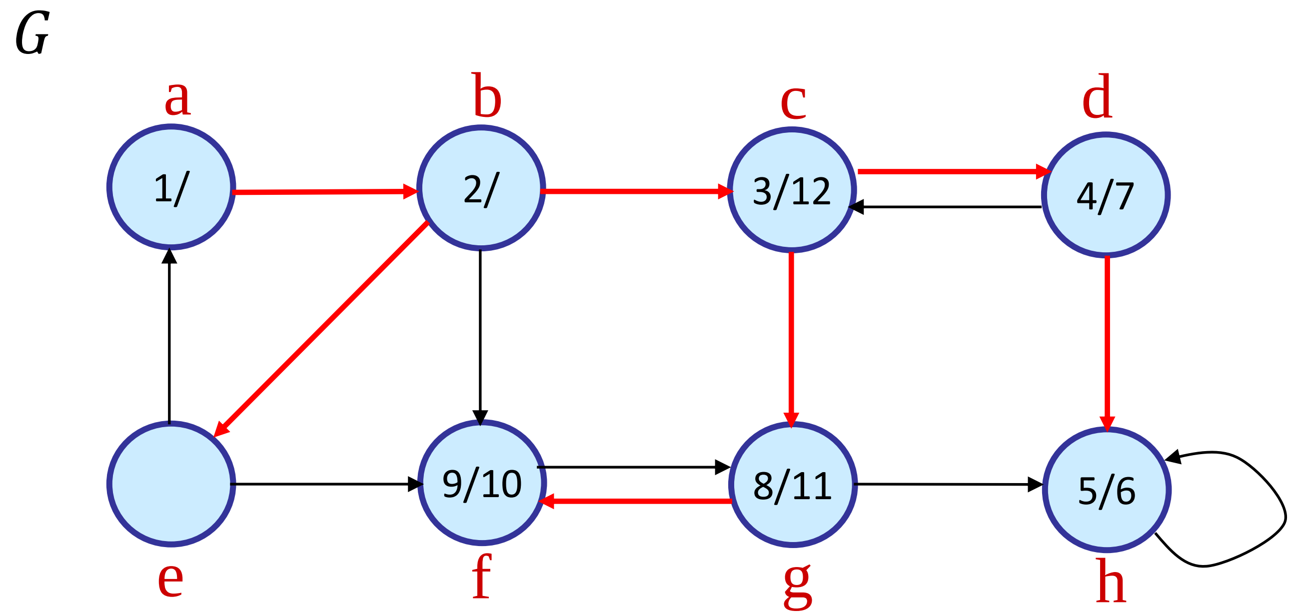
Exemple



1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$

Composantes fortement connexes – Algorithme

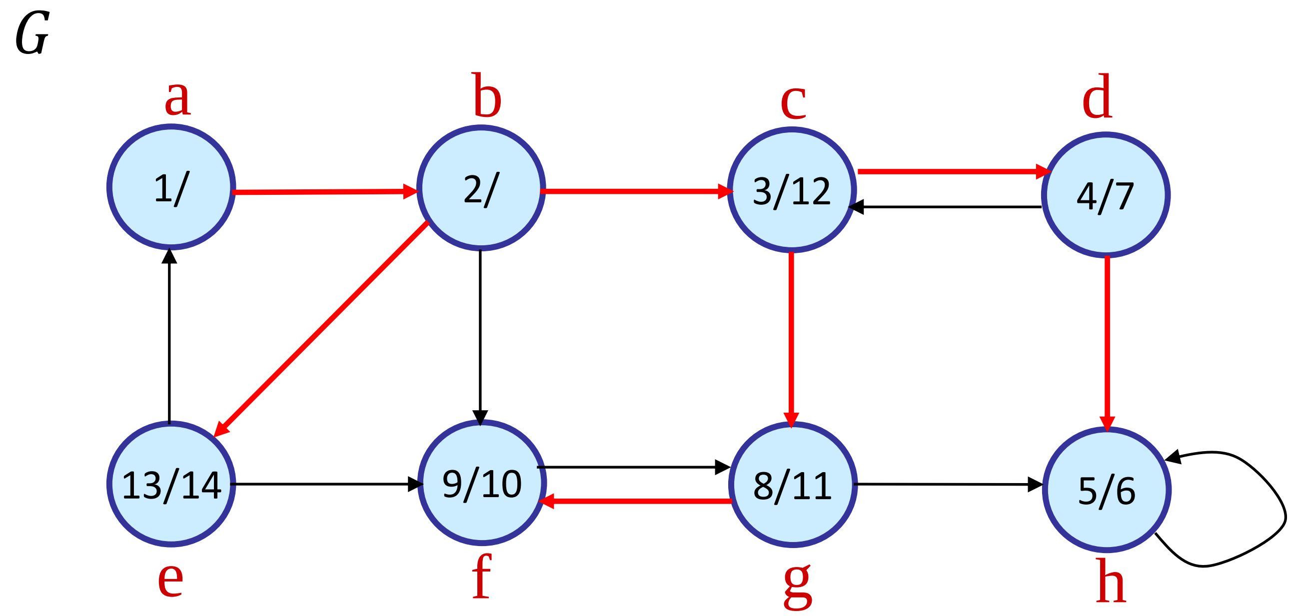
Exemple



1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$

Composantes fortement connexes – Algorithme

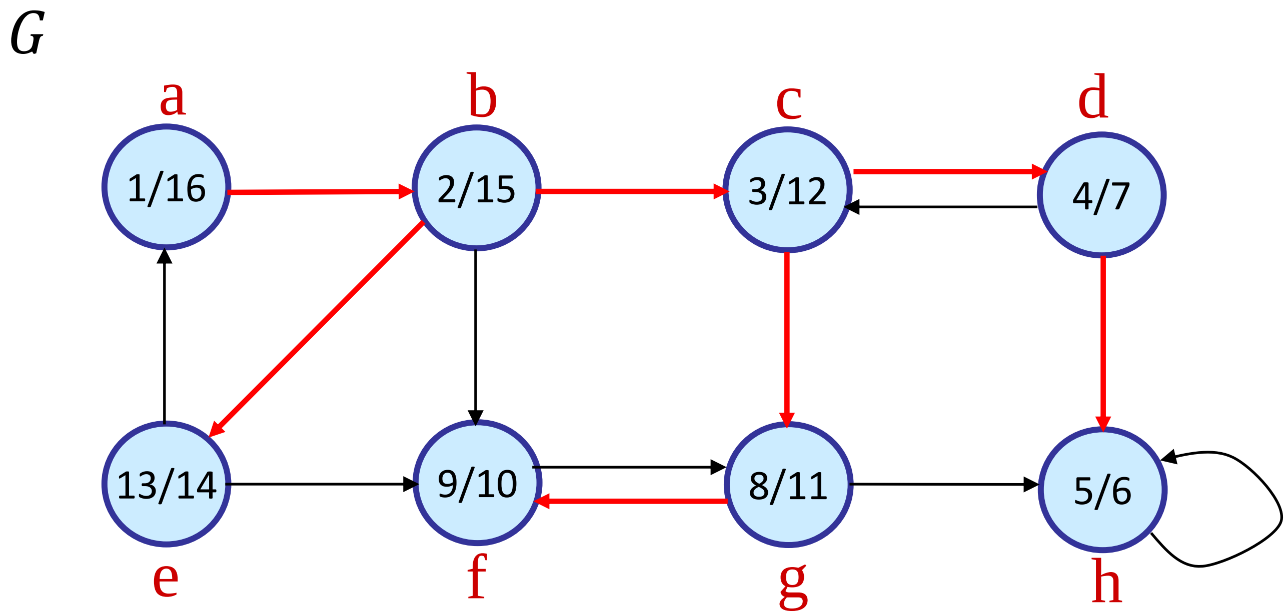
Exemple



1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$

Composantes fortement connexes – Algorithme

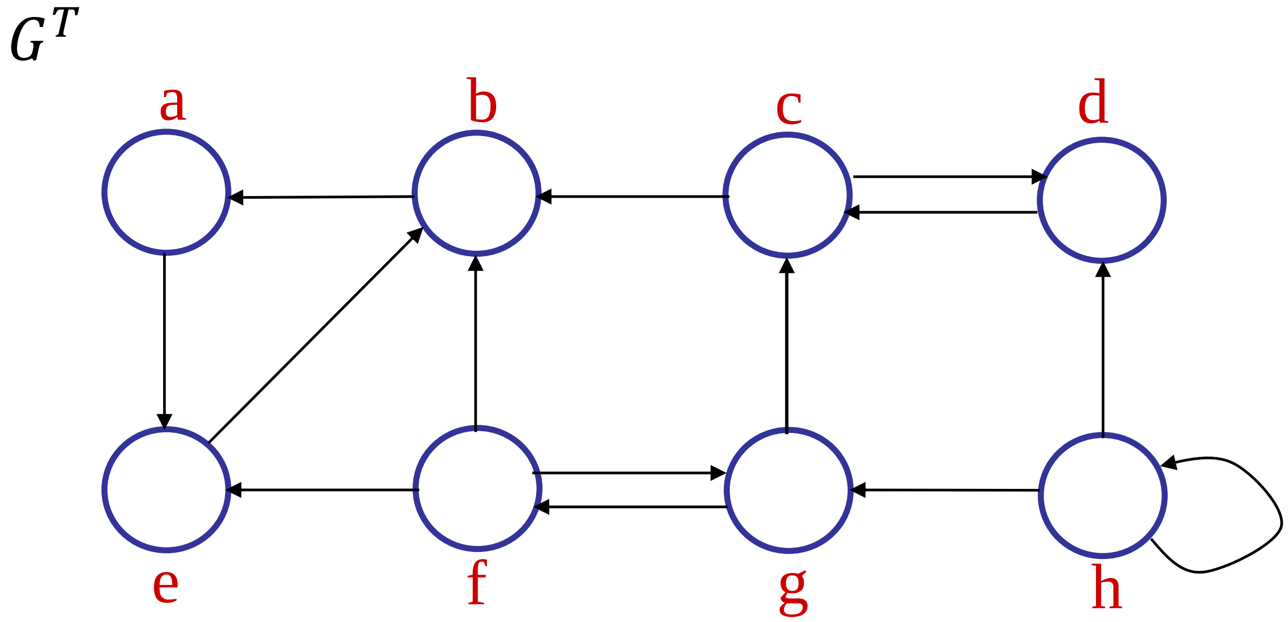
Exemple

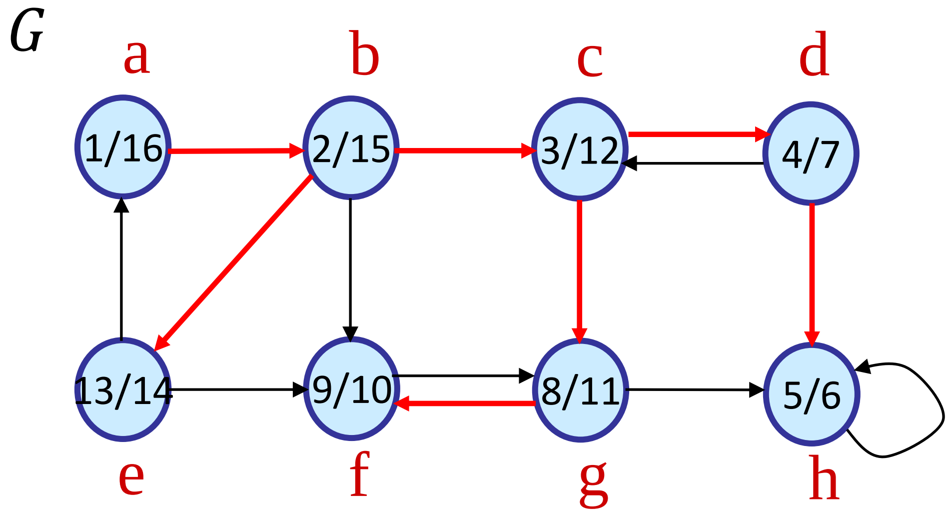


1. Appeler $PeP(G)$ pour calculer $f[i]$ pour tout $i \in S$

Composantes fortement connexes – Algorithme

Exemple

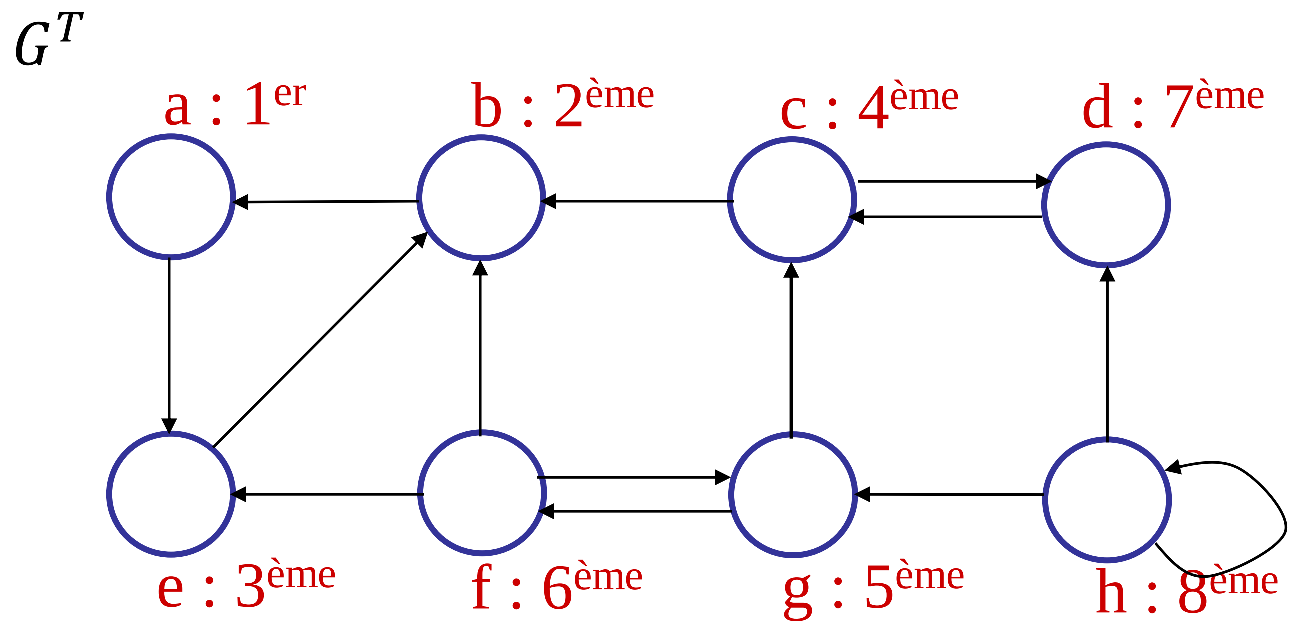


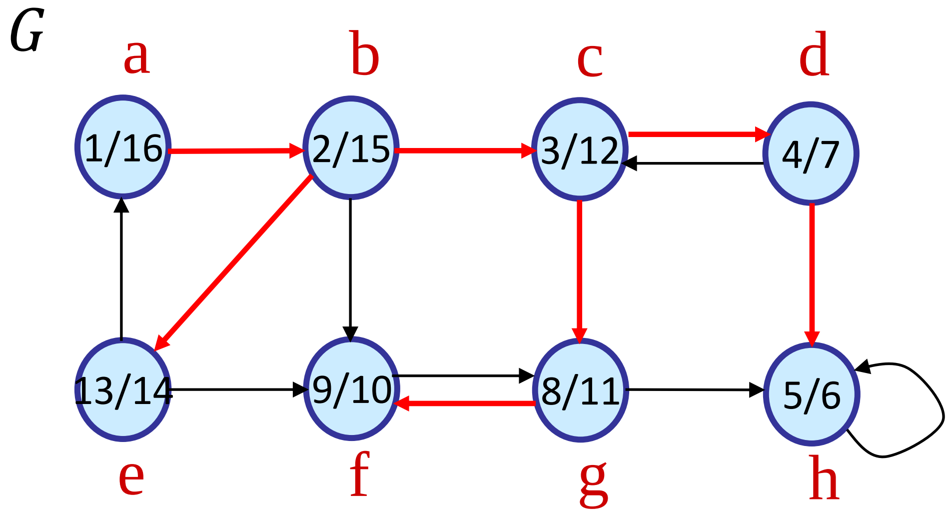


2. Calculer $G^T = (S, A^T)$

Composantes fortement connexes – Algorithme

Exemple

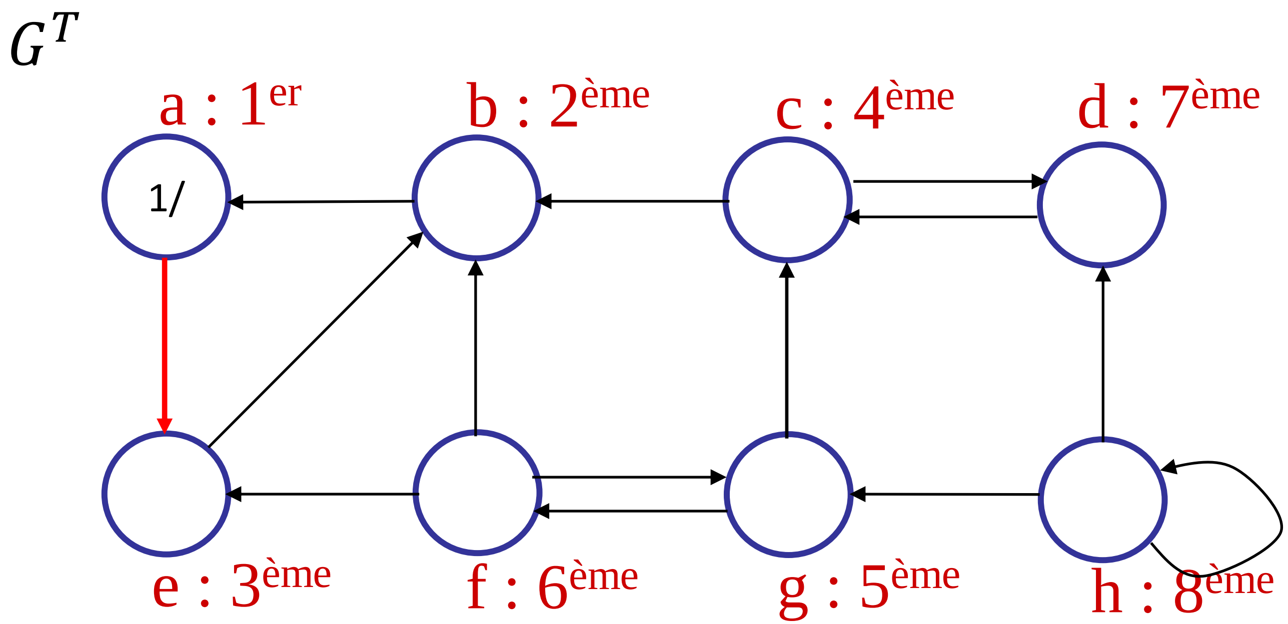


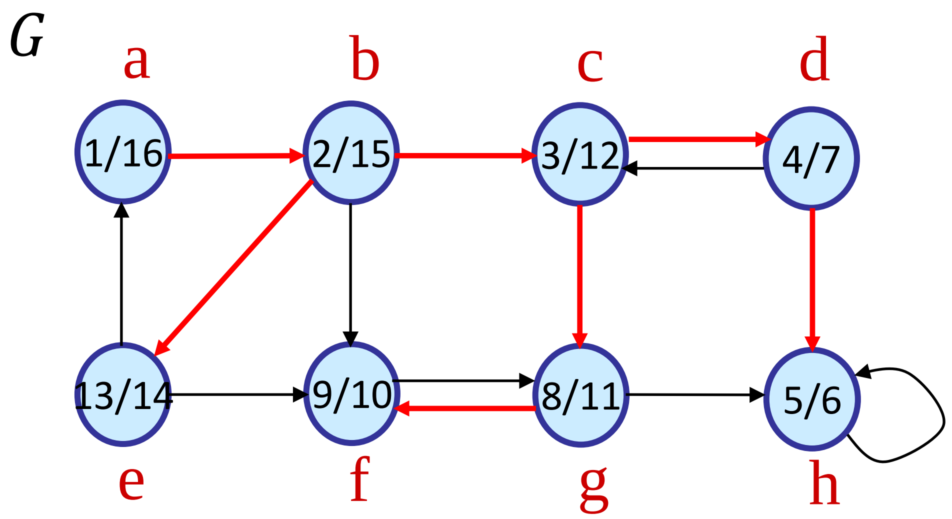


3. Appeler $PeP(G^T)$, en considérant les sommets dans l'ordre décroissant des $f[i]$

Composantes fortement connexes – Algorithme

Exemple

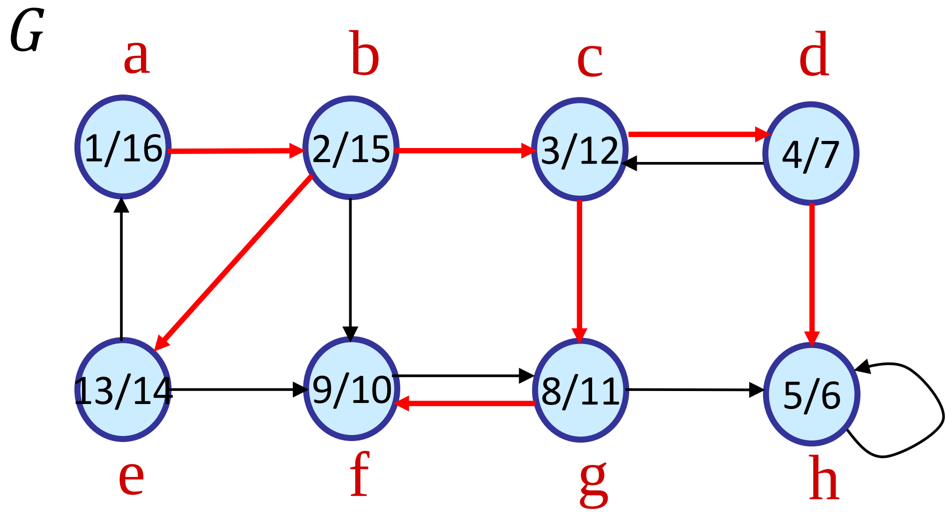


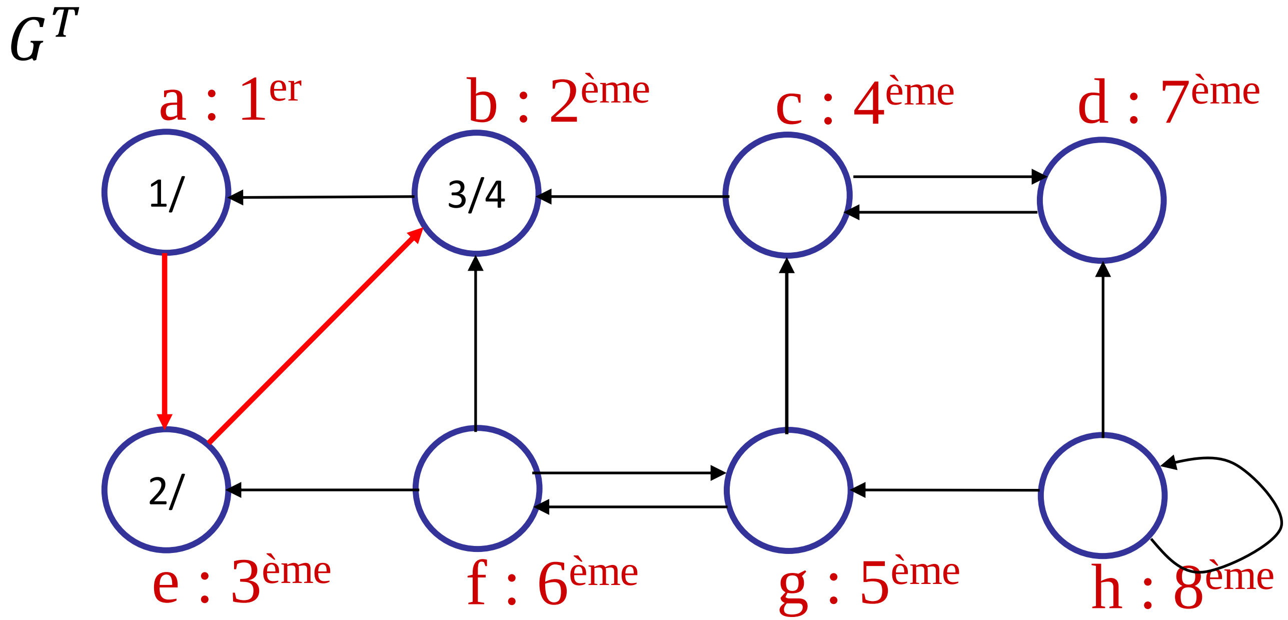


3. Appeler $PeP(G^T)$, en considérant les sommets dans l'ordre décroissant des $f[i]$

Composantes fortement connexes – Algorithme

Exemple

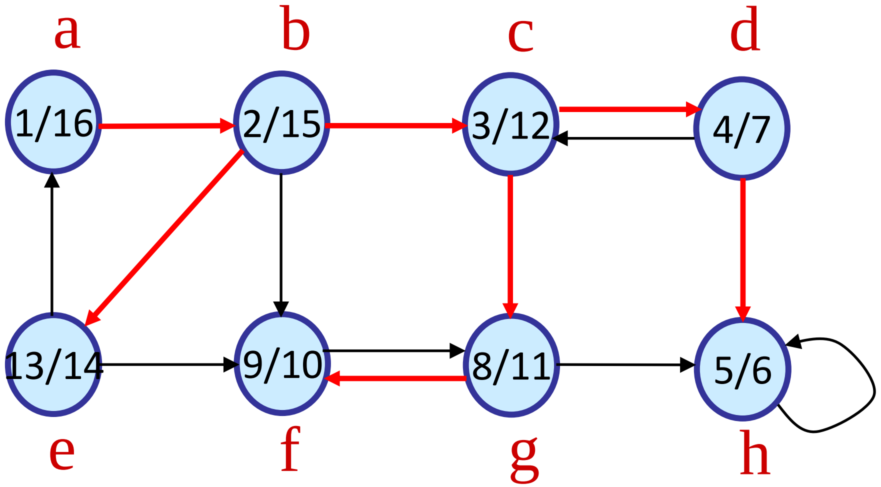




3. Appeler $PeP(G^T)$, en considérant les sommets dans l'ordre décroissant des $f[i]$

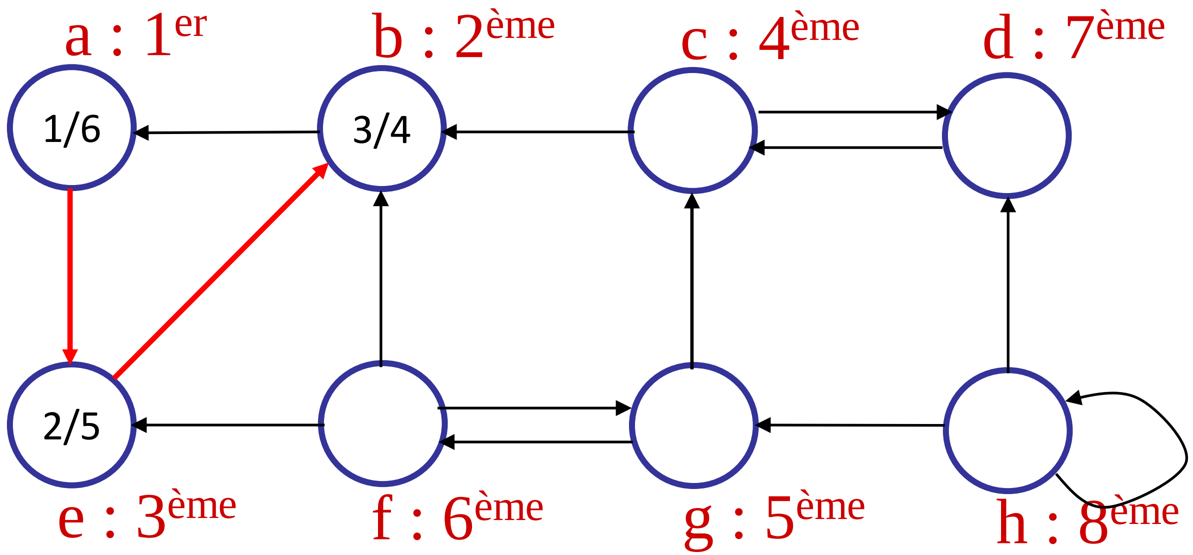
Composantes fortement connexes – Algorithme

G



Exemple

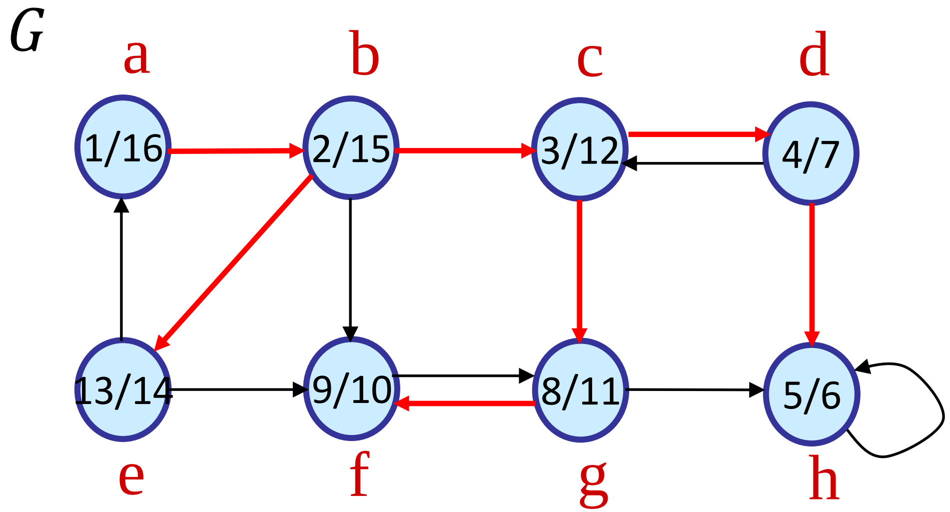
G^T

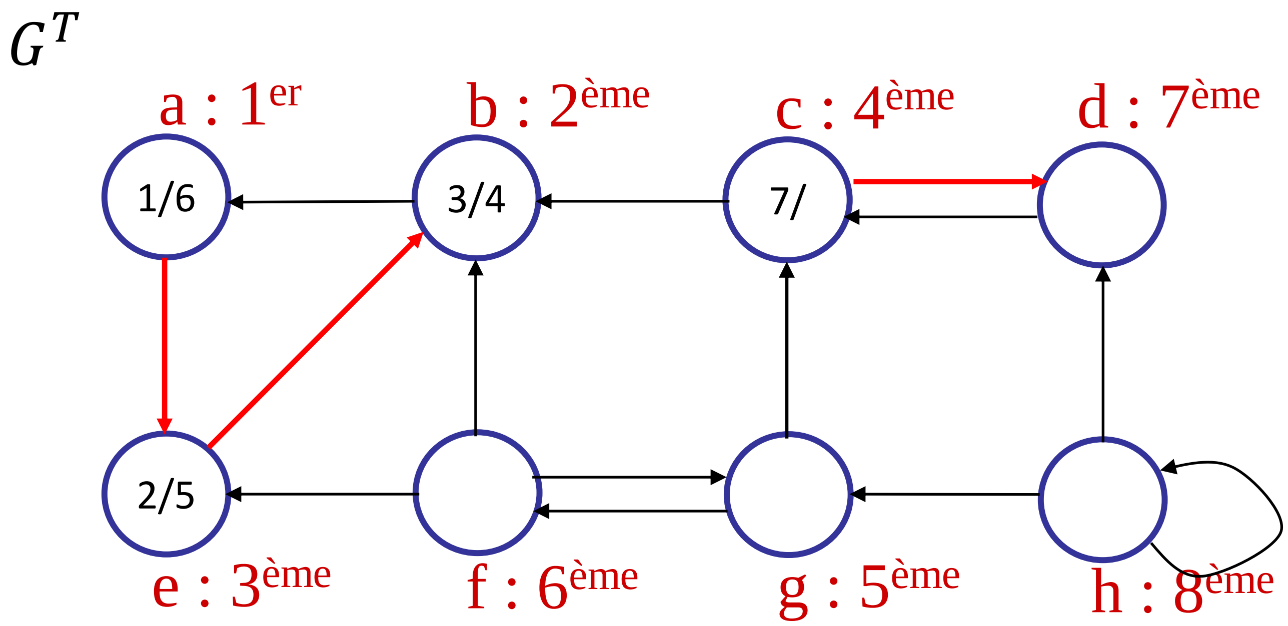


3. Appeler $PeP(G^T)$, en considérant les sommets dans l'ordre décroissant des $f[i]$

Composantes fortement connexes – Algorithme

Exemple

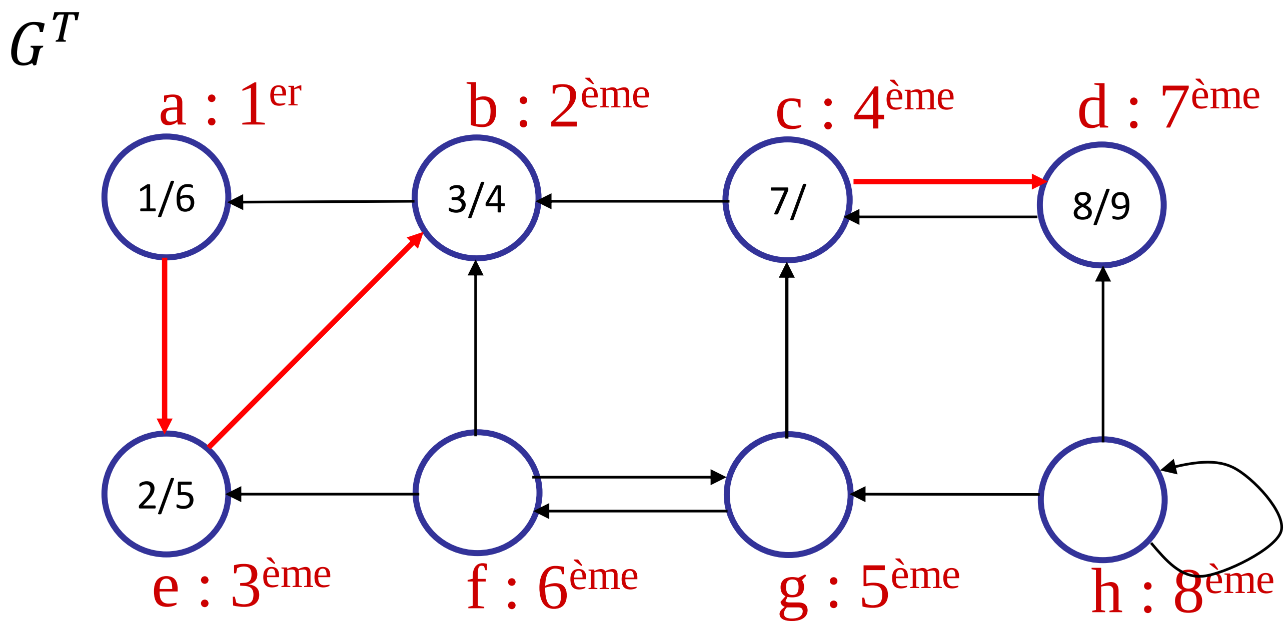


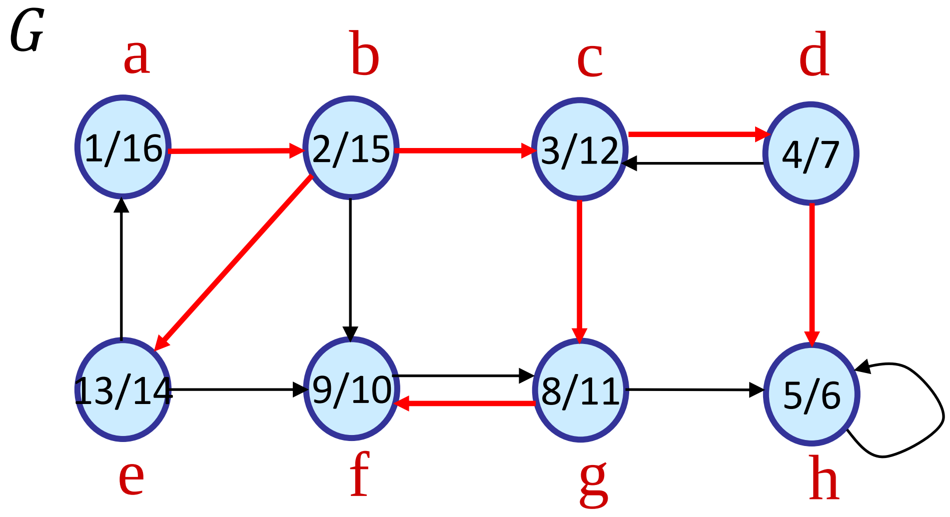


3. Appeler $PeP(G^T)$, en considérant les sommets dans l'ordre décroissant des $f[i]$

Composantes fortement connexes – Algorithme

Exemple

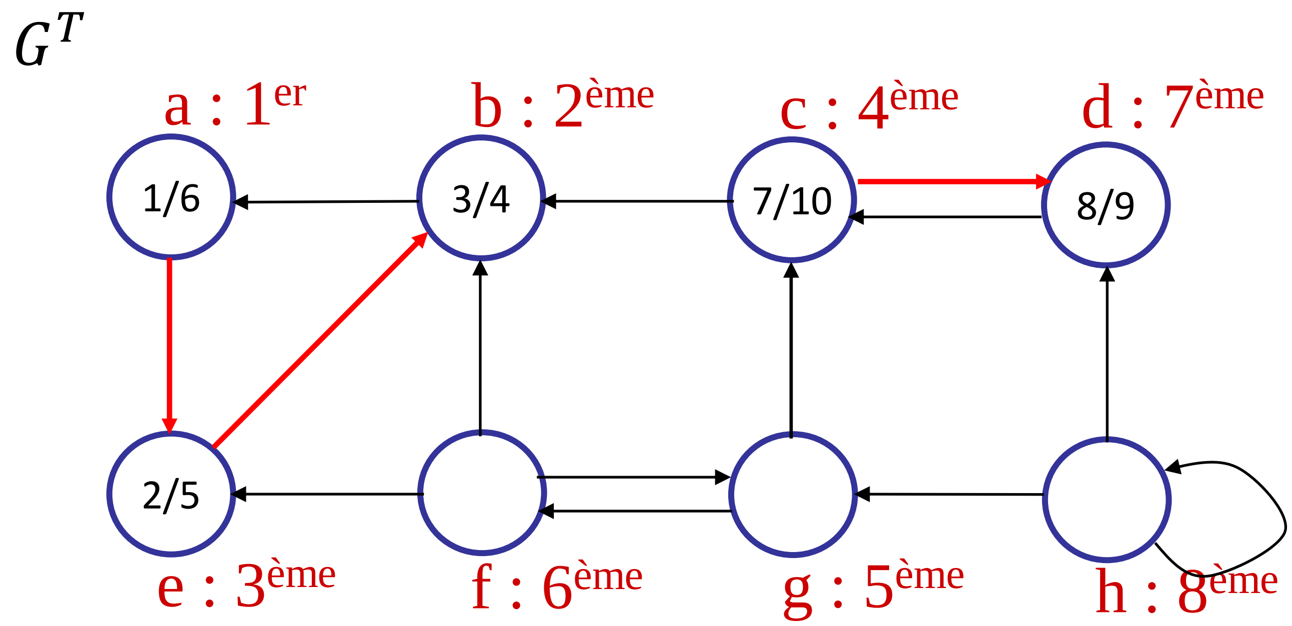


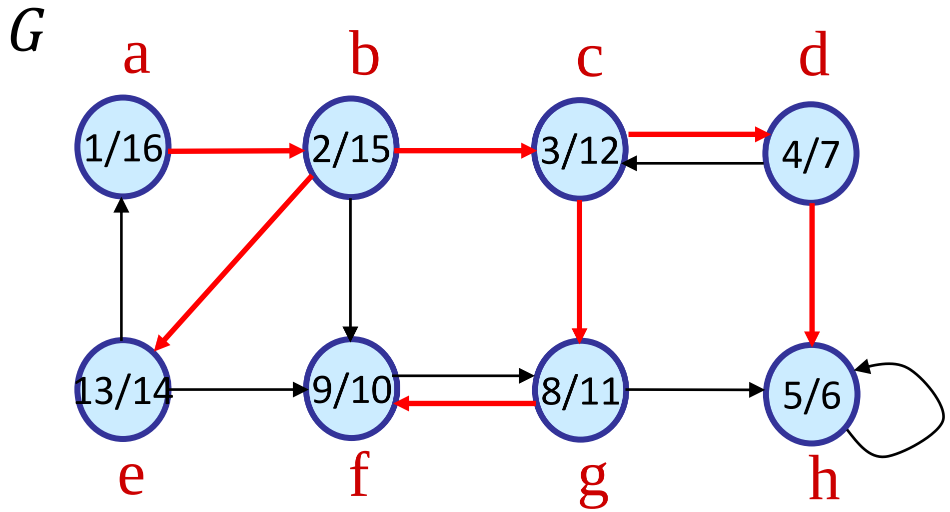


3. Appeler $PeP(G^T)$, en considérant les sommets dans l'ordre décroissant des $f[i]$

Composantes fortement connexes – Algorithme

Exemple

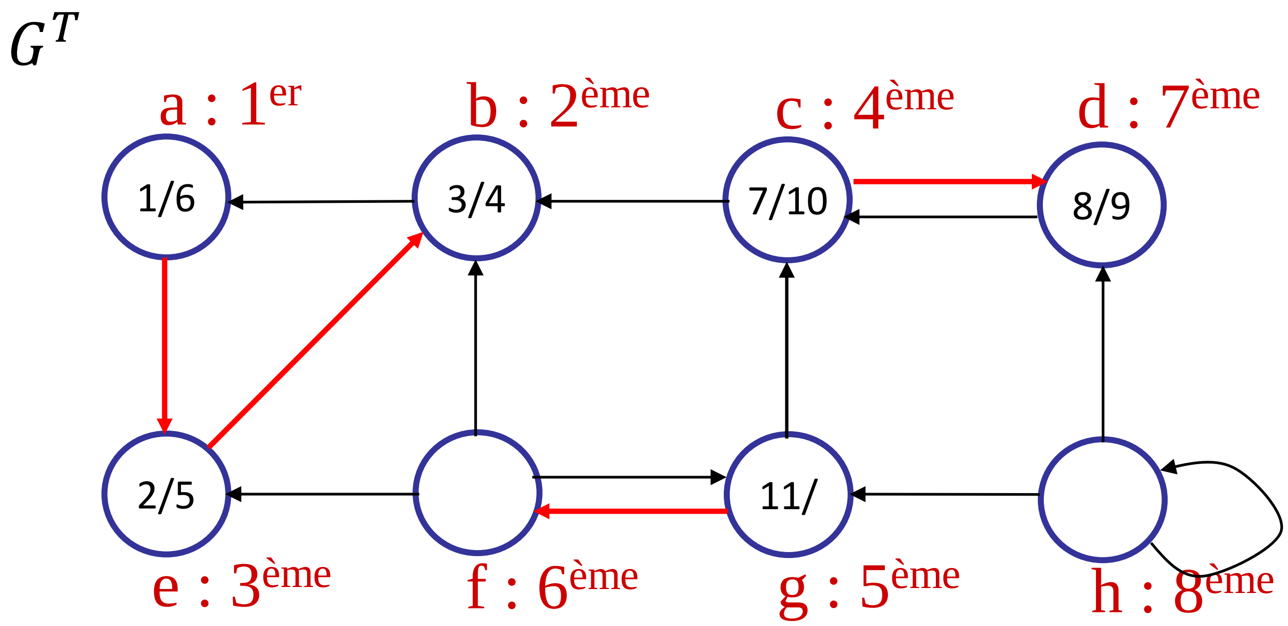


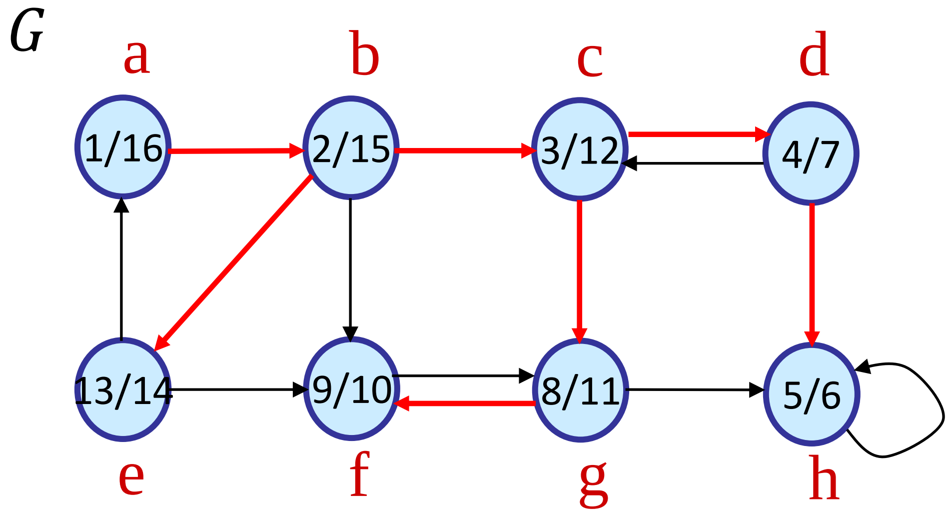


3. Appeler $PeP(G^T)$, en considérant les sommets dans l'ordre décroissant des $f[i]$

Composantes fortement connexes – Algorithme

Exemple

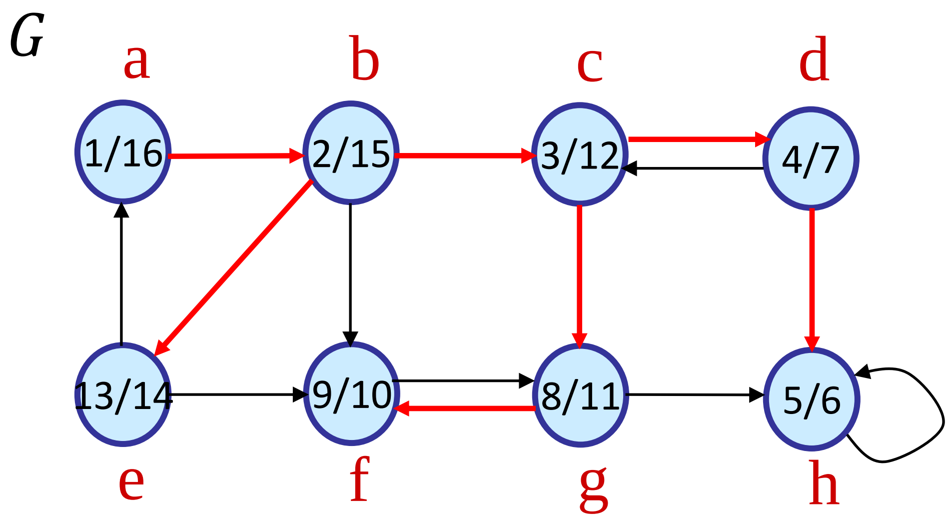


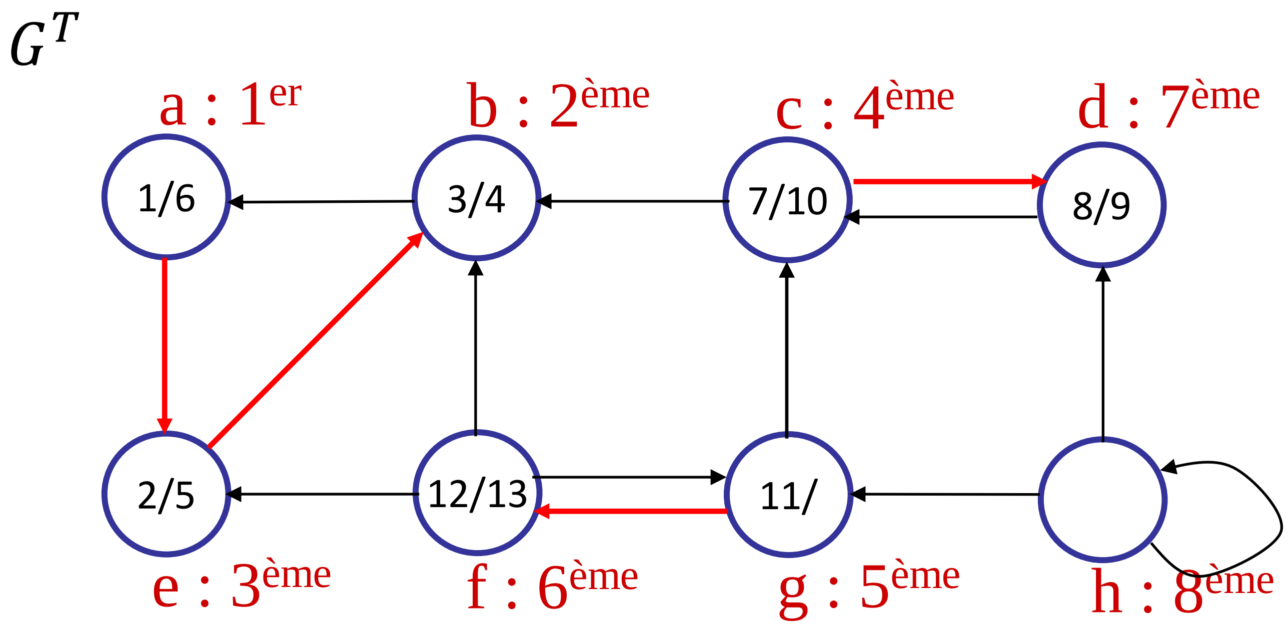


3. Appeler $PeP(G^T)$, en considérant les sommets dans l'ordre décroissant des $f[i]$

Composantes fortement connexes – Algorithme

Exemple

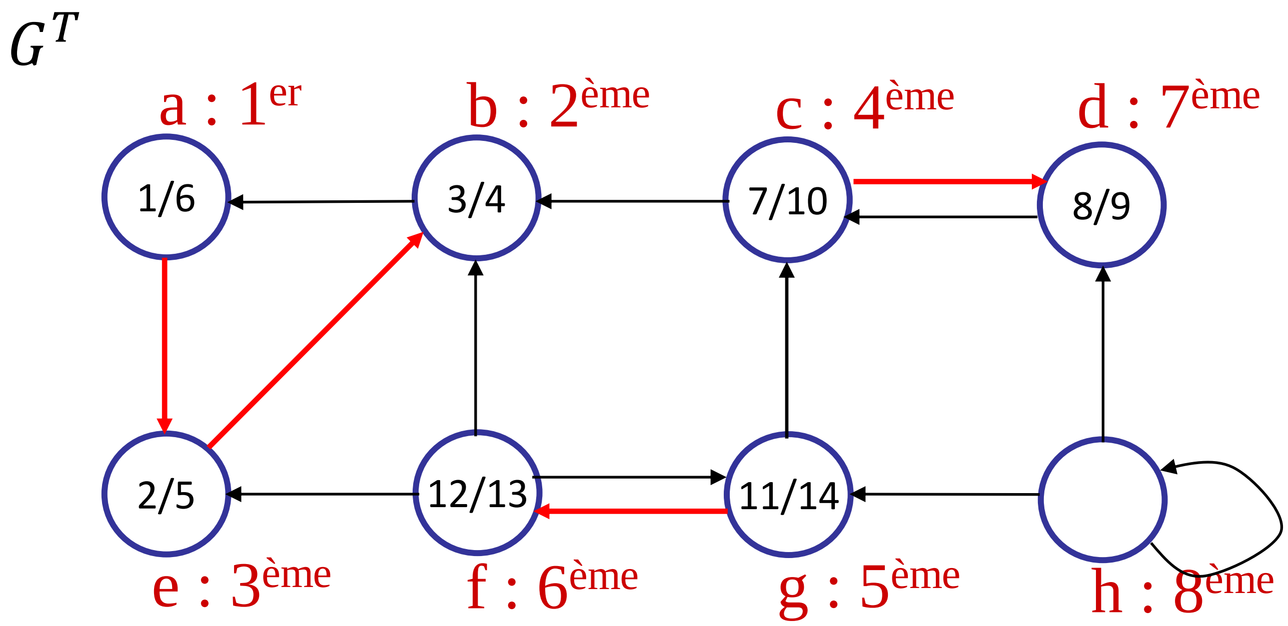


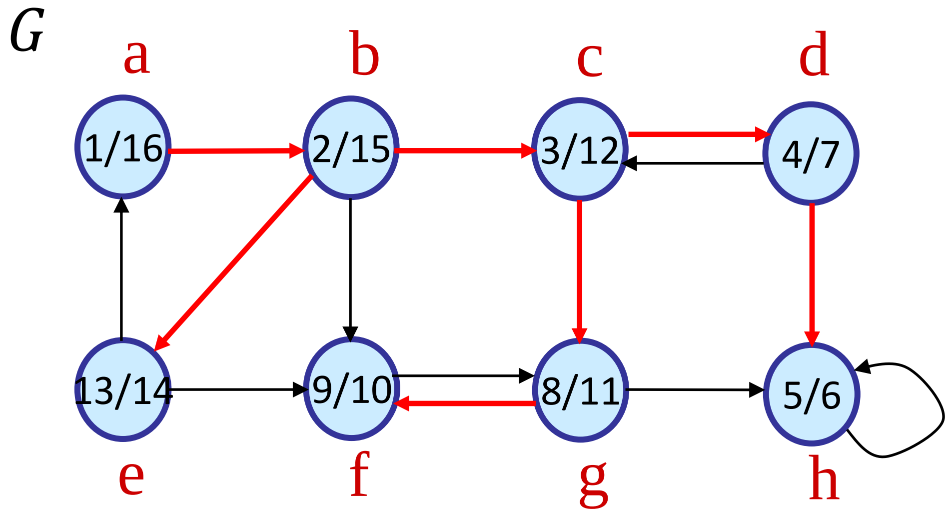


3. Appeler $PeP(G^T)$, en considérant les sommets dans l'ordre décroissant des $f[i]$

Composantes fortement connexes – Algorithme

Exemple

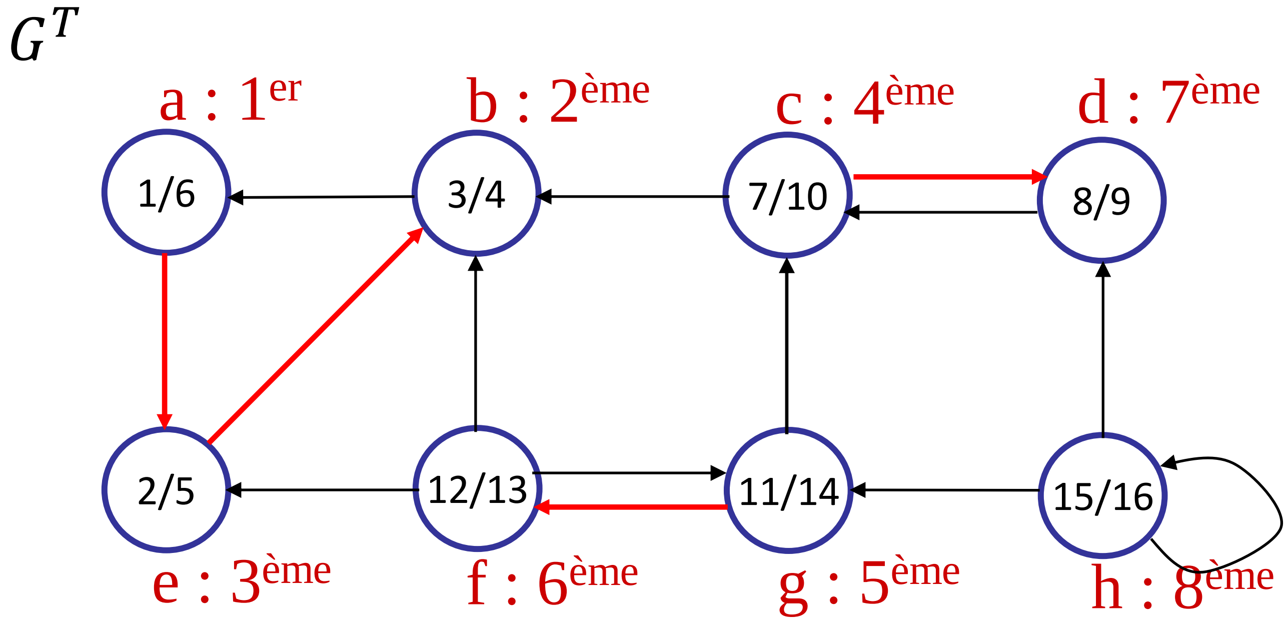


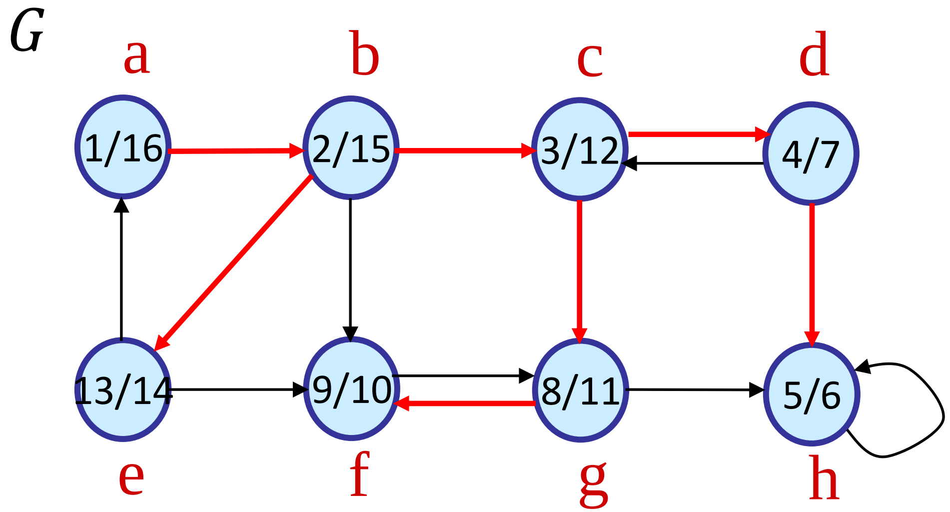


3. Appeler $PeP(G^T)$, en considérant les sommets dans l'ordre décroissant des $f[i]$

Composantes fortement connexes – Algorithme

Exemple

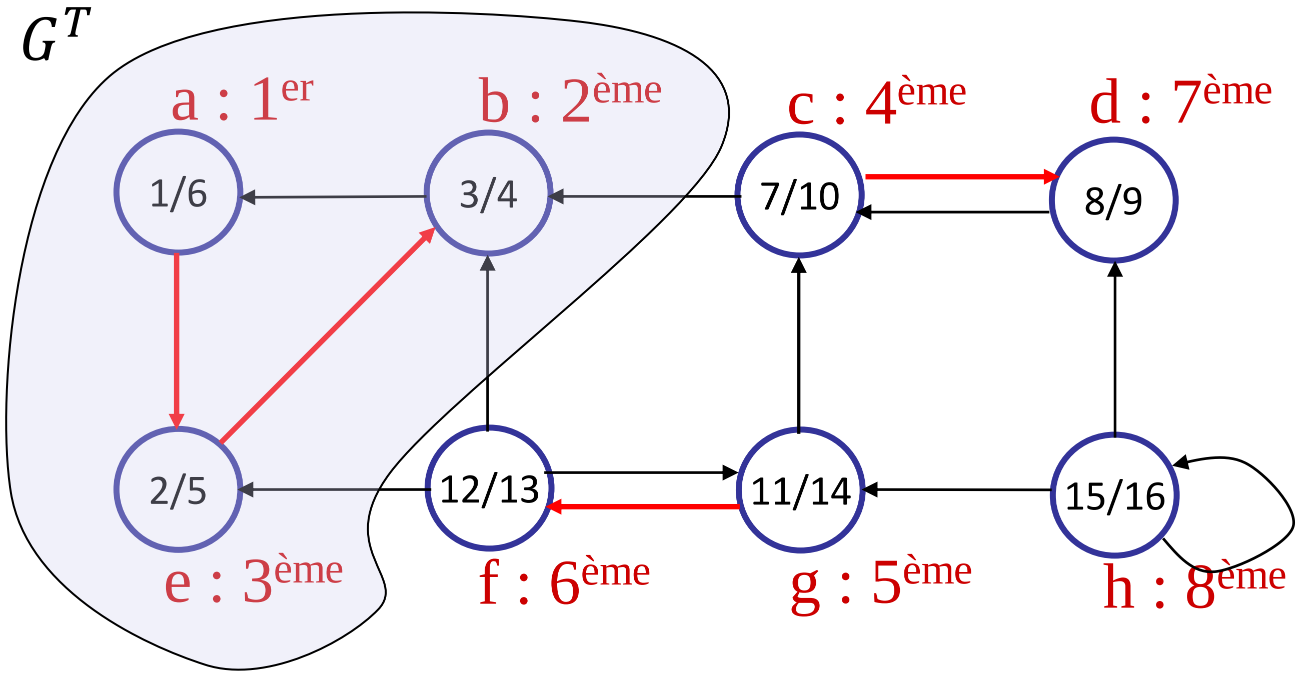


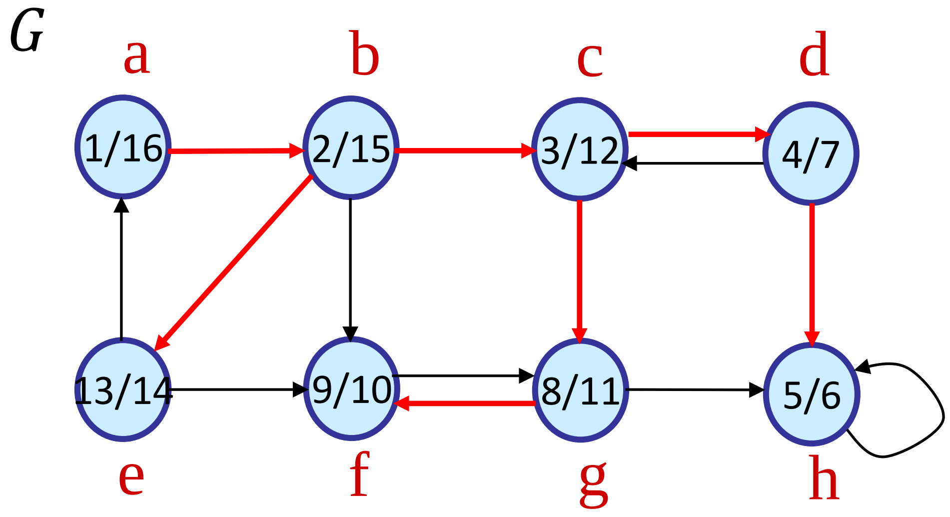


4. Les sommets de chaque arbre de la forêt formée dans le second appel de PeP forment une CFC

Composantes fortement connexes – Algorithme

Exemple

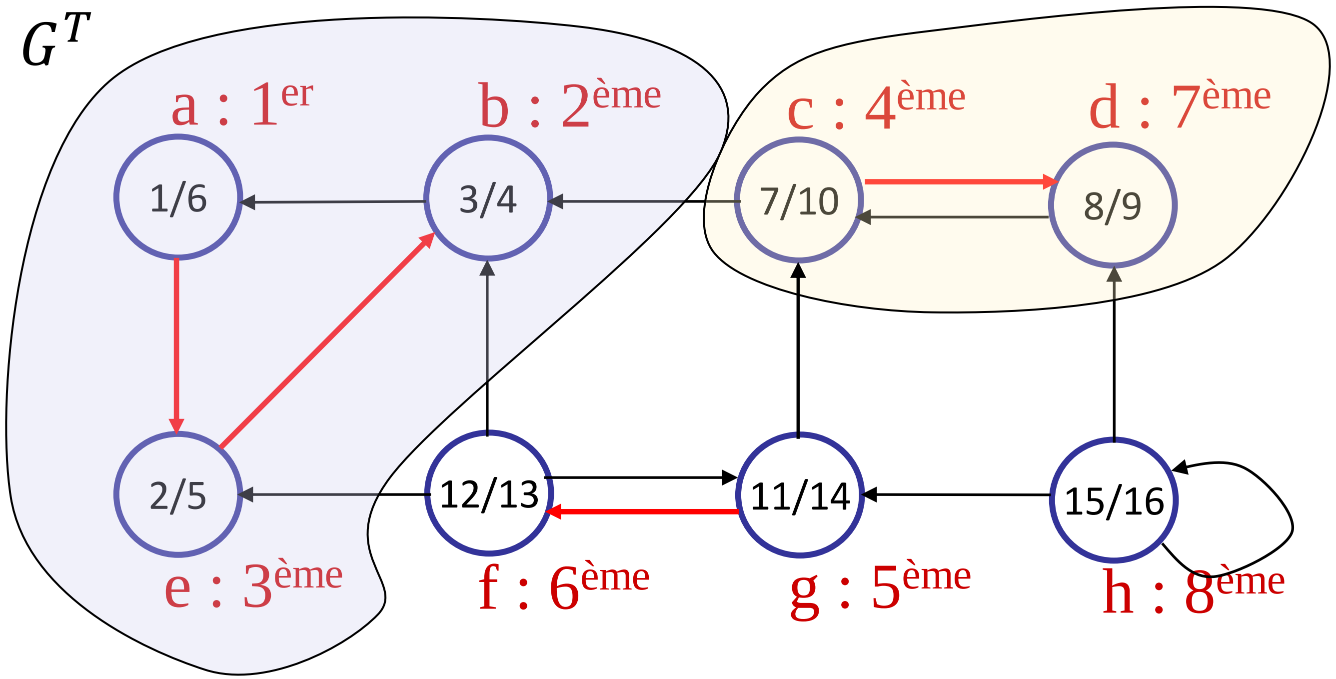


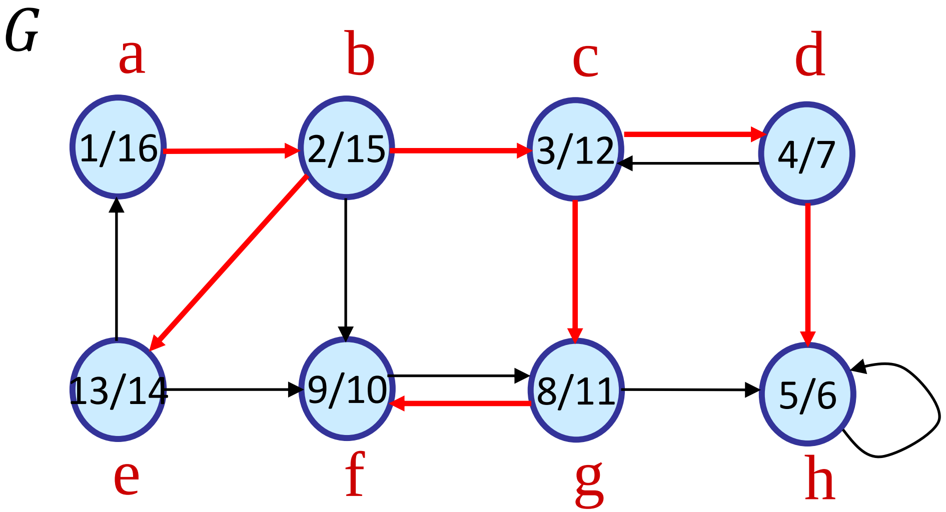


4. Les sommets de chaque arbre de la forêt formée dans le second appel de PeP forment une CFC

Composantes fortement connexes – Algorithme

Exemple

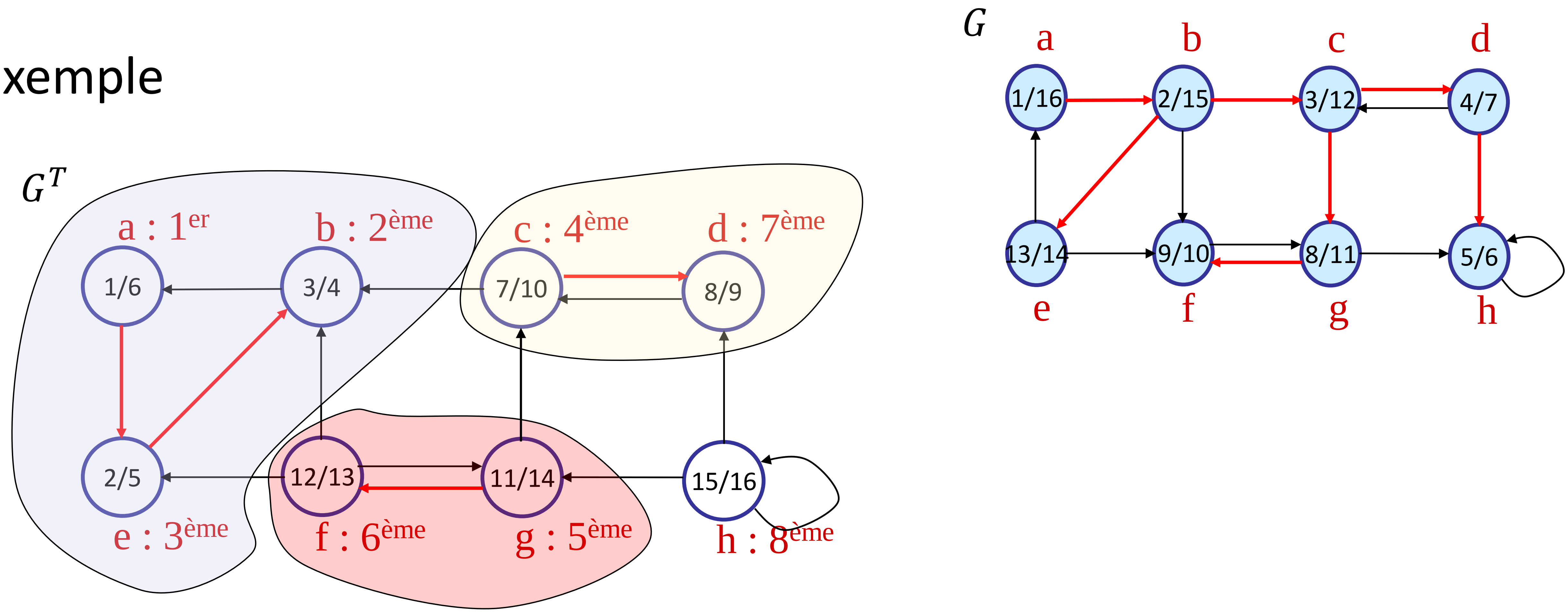




4. Les sommets de chaque arbre de la forêt formée dans le second appel de PeP forment une CFC

Composantes fortement connexes – Algorithme

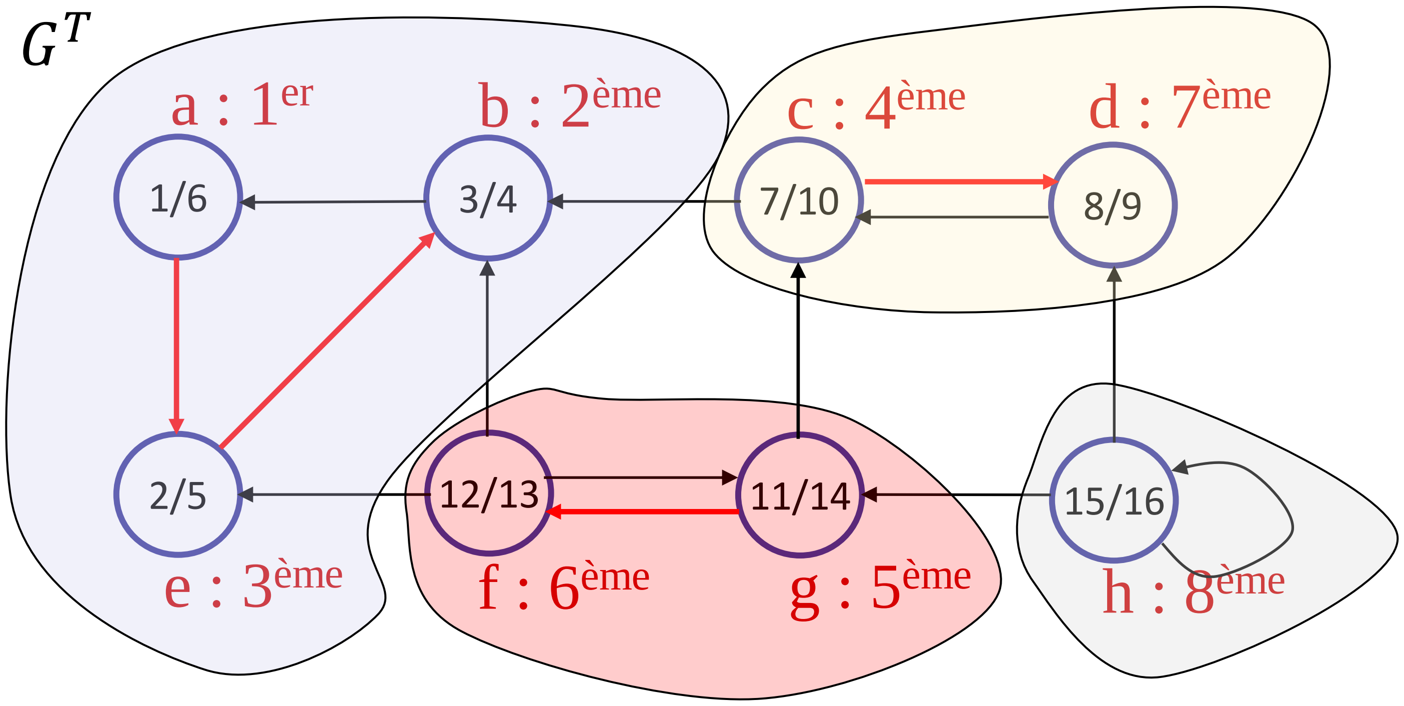
Exemple

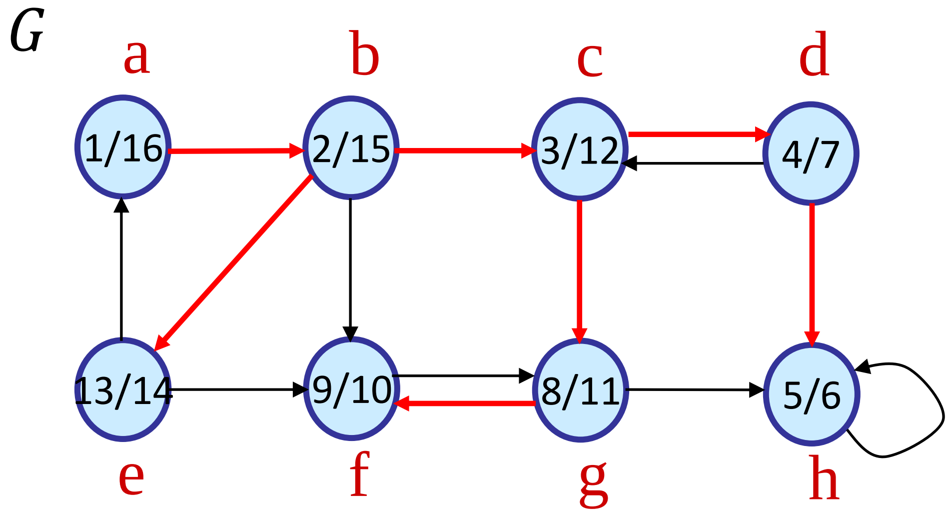


4. Les sommets de chaque arbre de la forêt formée dans le second appel de PeP forment une CFC

Composantes fortement connexes – Algorithme

Exemple

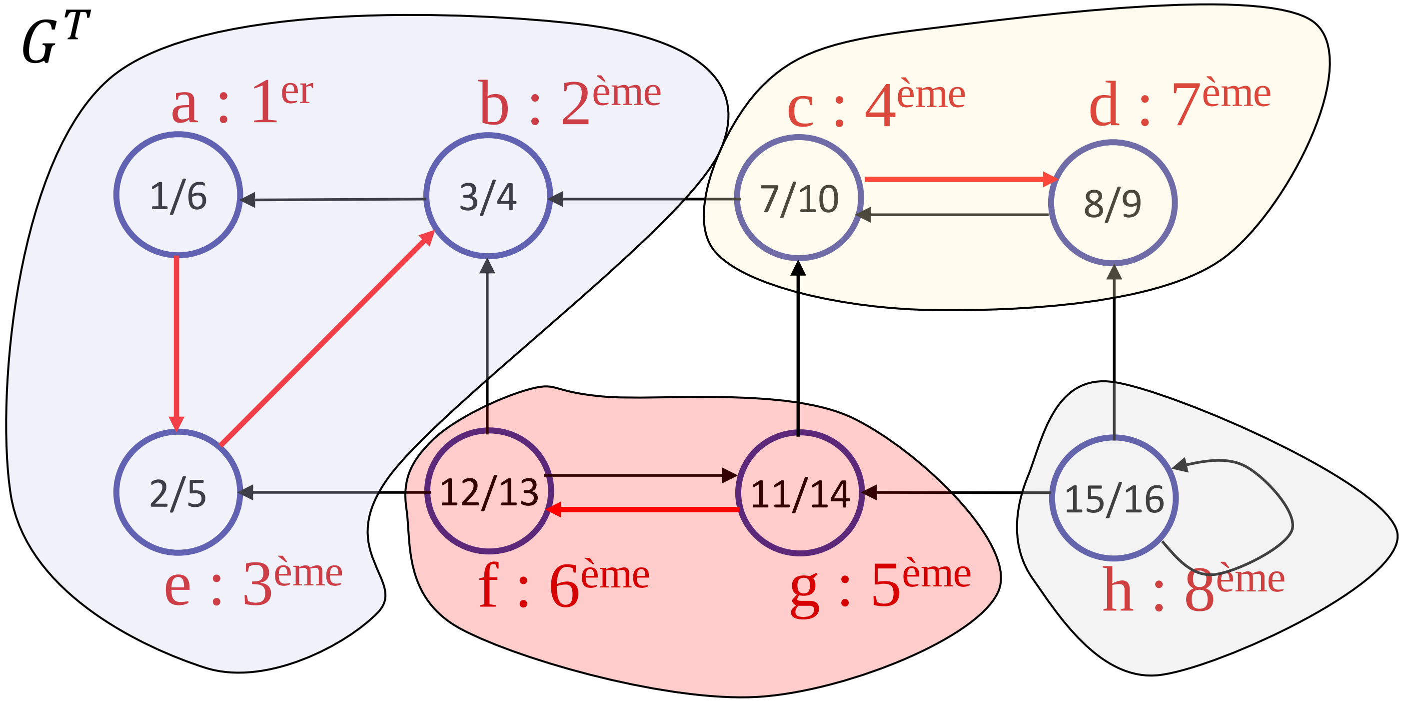


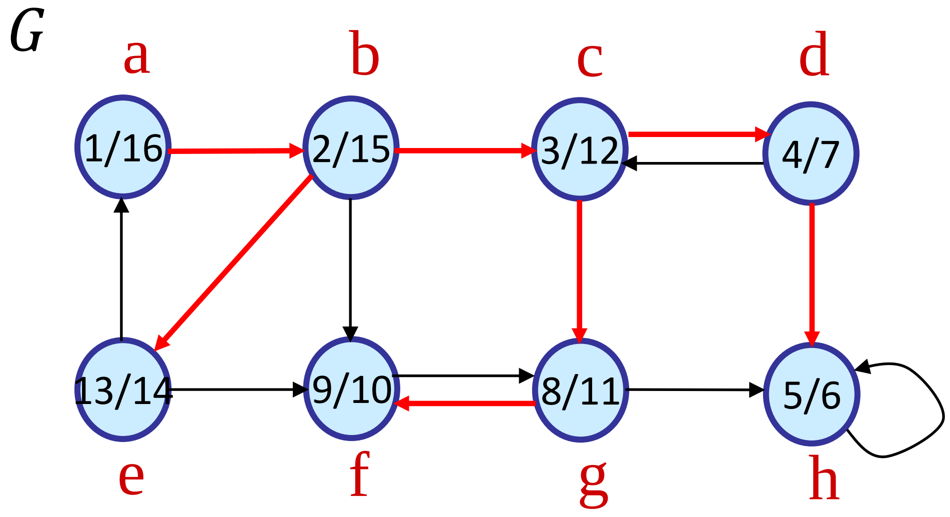


4. Les sommets de chaque arbre de la forêt formée dans le second appel de PeP forment une CFC

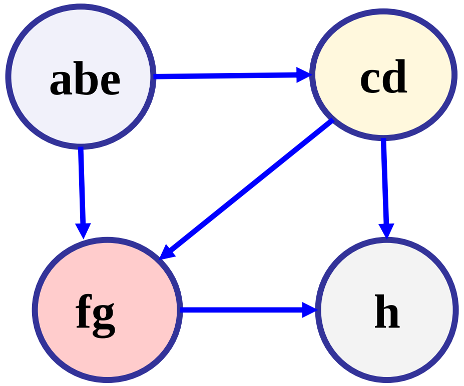
Composantes fortement connexes – Algorithme

Exemple





Le graphe réduit de G



Composantes fortement connexes – Algorithme

- Principe et Justification
  - Second PeP : ordre décroissant des temps de fin du 1er PeP PeP,

    $\rightarrow$ sommets du graphe réduit parcourus dans un **ordre topologique**
  - PeP execute sur $G^T \rightarrow$ impossible de visiter un sommet j à partir d'un sommet i avec i et j dans des CFC différentes
- **Théorème du chemin blanc :** dans une forêt de PeP d'un graphe $G = (S, A)$, j est un descendant de $i \Leftrightarrow$ au moment $d[i]$ où le parcours découvre i, il existe un chemin de i à j composé uniquement de sommets blancs (sauf pour i qui a juste été coloré en Gris)

Composantes fortement connexes – Algorithme

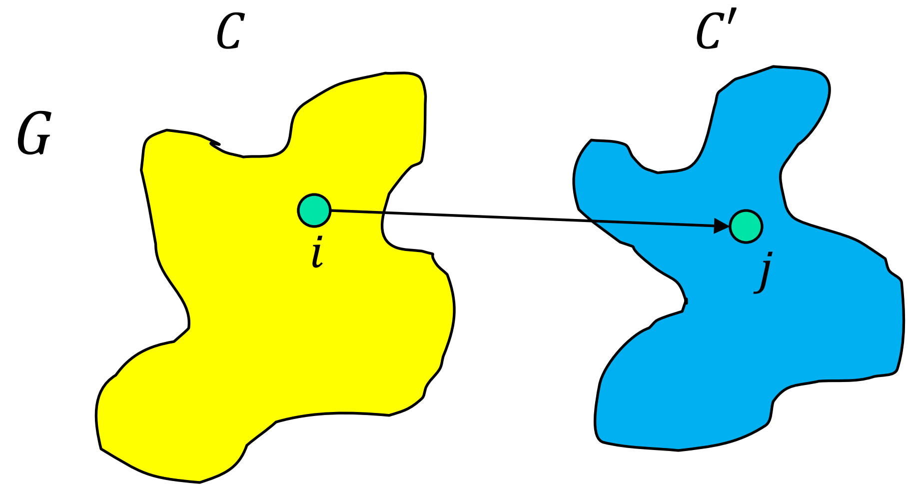
- Notations

Soit $I \subseteq S$, on définit

- $d(I) = \min\{d(i) : i \in I\}$: premier temps de découverte des sommets dans I
- $i_I^- = \text{argmin}\{d(i) : i \in I\}$: premier sommet découvert dans I (i.e., $d(I) = d(i_I^-)$)
- $f(I) = \max\{f(i) : i \in I\}$: dernier temps de fin de traitement des sommets dans I
- $i_I^+ = \text{argmax}\{f(i) : i \in I\}$: dernier sommet traité dans I (i.e., $f(I) = f(i_I^+)$)

CFC et dates de fin dans PeP

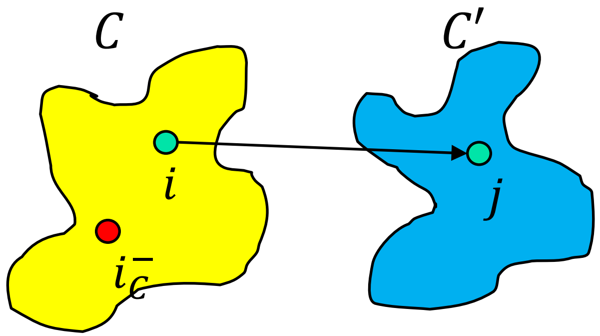
- **Proposition :** soient C et C' des CFC distinctes de $G = (S, A)$, on a
$$(i, j) \in A \cap C \times C' \Longrightarrow f(C) > f(C')$$



Preuve : On considère les deux cas possibles

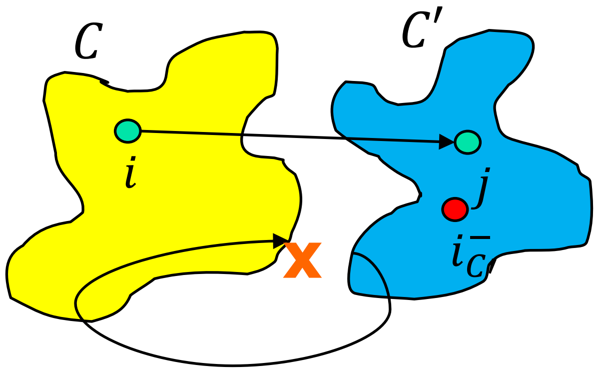
- Cas 1 : $d(C) < d(C')$
- Cas 2 : $d(C) > d(C')$

CFC et dates de fin dans PeP



- Cas 1 : $d(C) < d(C')$
  - Soit i_C^- le premier sommet découvert dans C
  - Au temps $d[i_C^-]$, tous les sommets dans $C \cup C'$ sont blancs
  - Donc il existe des chemins de sommets blancs de i_C^- vers tous les sommets dans $C \cup C'$
  - Par le théorème du chemin blanc, tous les sommets dans $C \cup C'$ sont des descendants de i_C^- dans l'arbre de PeP.
  - Par le théorème des parenthèses, on a $f[i_C^-] = f(C) > f(C')$

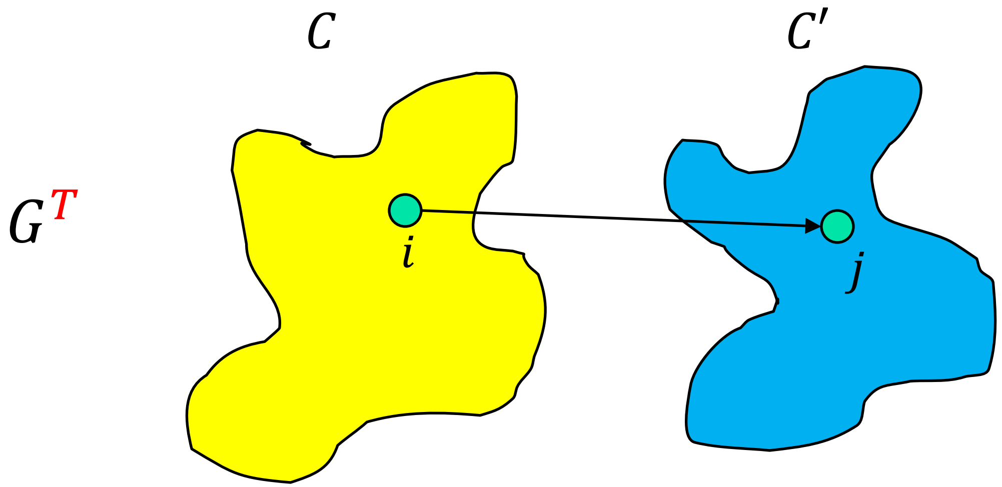
CFC et dates de fin dans PeP



- Cas 2 : $d(C) > d(C')$
  - Soit $i_{C'}^-$ le premier sommet découvert dans C'
  - Au temps $d[i_{C'}^-]$, tous les sommets dans C' sont blancs
  - $\Rightarrow$ il y a un chemin blanc de $i_{C'}^-$ à chaque sommet dans C'
  - $\Rightarrow$ tous les sommets dans C' deviennent des descendants de $i_{C'}^-$
  - $\Rightarrow$ A nouveau, $f[i_{C'}^-] = f(C')$
  - Au moment $d[i_{C'}^-]$, tous les sommets dans C sont aussi blancs
  - $\Rightarrow$ puisqu'il y a un arc (i, j), on ne peut pas avoir de chemin de C' à C
  - $\Rightarrow$ aucun sommet dans C n'est atteignable depuis $i_{C'}^-$
  - $\Rightarrow$ au moment $f[i_{C'}^-]$, tous les sommets dans C sont encore blancs
  - $\Rightarrow \forall j \in C, f[j] > f\left[i_{C'}^-\right] \Rightarrow f(C) > f(C')$

CFC et dates de fin dans PeP

- **Corollaire :** soient C et C' des CFC distinctes de $G^{\color{red}T} = (S, A^{\color{red}T})$, on a
$$(i,j) \in A^{\color{red}T} \cap C \times C' \Longrightarrow f(C) \color{red}{<} \color{black} f(C')$$



Preuve :

- $(i,j) \in A^{\color{red}T} \Longrightarrow (i,j) \in A$
- Les CFCs de G et $G^{\color{red}T}$ sont les mêmes

Déroulement de CFC – PeP(G)

- Quand on effectue le second PeP sur G^T, on commence par la CFC C tel que $f(C)$ soit maximale
  - Le second PeP part d'un certain $i \in C$, et il visite tous les sommets dans C
  - Le Corollaire dit que puisque $f(C) > f(C')$ pour tout $C \neq C'$, il n'y a aucun arc depuis C vers C' dans G^T
  - $\Rightarrow$ PeP visitera *seulement* les sommets dans C
  - $\Rightarrow$ l'arbre PeP de racine en i contient **exactement** les sommets de C

Déroulement de CFC – PeP(G)

- La racine suivante choisie dans le second PeP est une CFC C' telle que $f(C')$ soit maximum parmi toutes les CFC autres que C
  - PeP visitera tous les sommets dans C', mais seuls les arcs hors de C' vont à C, que nous avons déjà visités
  - $\Rightarrow$ les seuls arcs d'arbre seront vers les sommets dans C'

$\rightarrow$ on répète le processus, et chaque fois que nous choisissons une racine dans le second PeP, elle peut atteindre seulement

- Des sommets dans sa CFC $\Rightarrow$ Arcs d'arbre
- Des sommets dans les CFCs *déjà visités* dans le second PeP $\Rightarrow$ Non arcs d'arbre

Validité de CFC – PeP(G)

- **Théorème :** l'algorithme $CFC - PeP(G)$ calcule effectivement les CFCs d'un graphe orienté $G = (S, A)$

- **Preuve** : par récurrence sur le nombre d'arbres de $PeP(G^T)$ dont les sommets forment une CFC

Nombre cyclomatique

- **Théorème :** Pour tout graphe $G = (S, A)$ on a
$$v(G) = m - n + p \geq 0, \text{ avec}$$
  - $n = |S|$
  - $m = |A|$
  - p le nombre de composantes connexes de G

- $v(G)$ est le nombre cyclomatique de G
- Si G non orienté et acyclique alors $v(G) = 0$

Nombre cyclomatique

- **Preuve :**
  - Soit un graphe $G = (S, A)$ avec $A = \{a_1, a_2, \dots, a_m\}$.
  - On définit $A_k = \{a_1, a_2, \dots, a_k\}$ avec $A_0 = \emptyset$
  - Considérer la séquence des graphes partiels pour $k = 0,1, \dots, n$
    $$G_k = (S, A_k)$$
  - G_0 est constitué de n sommets isolés
  - $A_{k+1} = A_k + \{a_{k+1}\}$
  - $G_n = G$
  - $v(G_k)$ est le nombre cyclomatique de G_k

  $\rightarrow$ Preuve par récurrence sur $v(G_k) \geq 0$

Nombre cyclomatique

- **Notations** : Pour $G_k = (S, A_k), k = 0,1, \dots, n$
  - m_k : nombre d'arêtes de G_k
  - p_k : nombre de composantes connexes de G_k
  - $v(G_k)$: nombre cyclomatique de G_k
- **Base d'induction** : G_0 est sans cycle

$$v(G_0) = m_0 - n + n_0 = 0 - n + n = 0$$

- **Hypothèse de récurrence** : Supposons que pour $k < n$ on a

$$v(G_h) = m_h - n + n_h \geq 0, \forall h \leq k$$

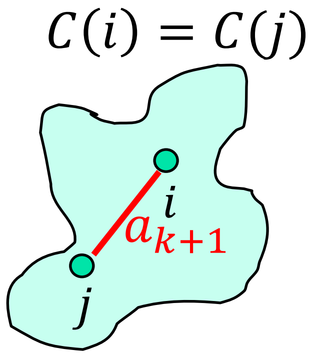
- Montrons que

$$v(G_{k+1}) = m_{k+1} - n + n_{k+1} \geq 0$$

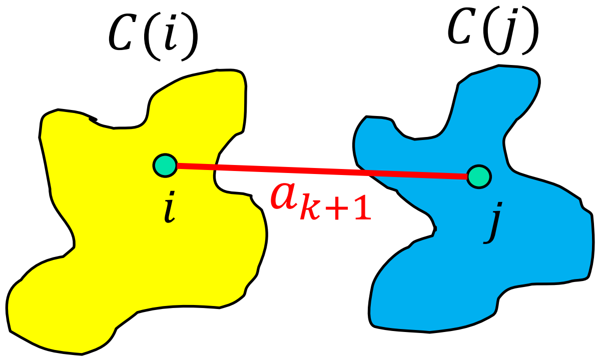
Nombre cyclomatique

- $G_{k+1} = (S, A_{k+1})$ avec $A_{k+1} = A_k + \{a_{k+1}\}$
- En posant $a_{k+1} = (i, j)$, deux cas se présentent

Cas 1: $\boldsymbol{CC(i) = CC(j)}$



Cas 2: $\boldsymbol{CC(i) \neq CC(j)}$

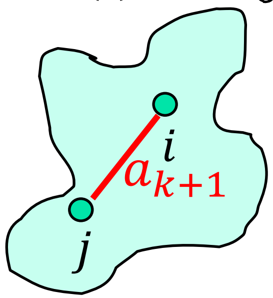


Nombre cyclomatique

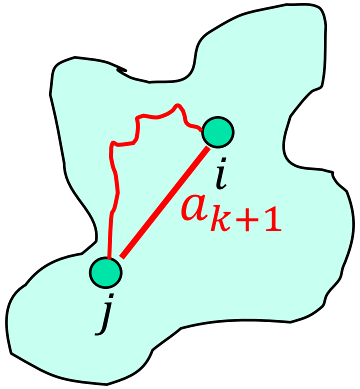
Cas 1: $\boldsymbol{CC(i) = CC(j)}$

- Cas 1 : $CC(i) = CC(j)$
  - $m_{k+1} = m_k + 1$
  - $p_{k+1} = p_k$
  - $v(G_{k+1}) = m_{k+1} - n + p_{k+1} = m_k + 1 - n + p_k$
  - $\Rightarrow$ $v(G_{k+1}) = v(G_k) + 1$
  - L'ajout de l'arête a_{k+1} crée un nouveau cycle

$C(i) = C(j)$

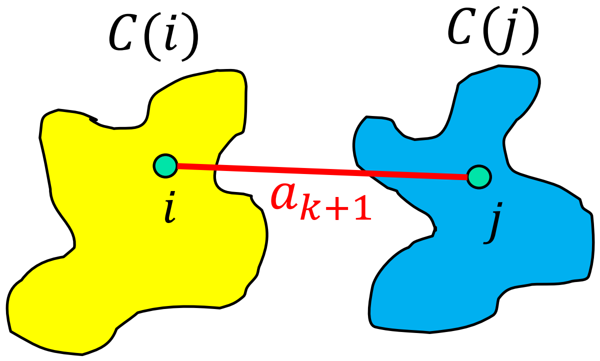


$C(i) = C(j)$

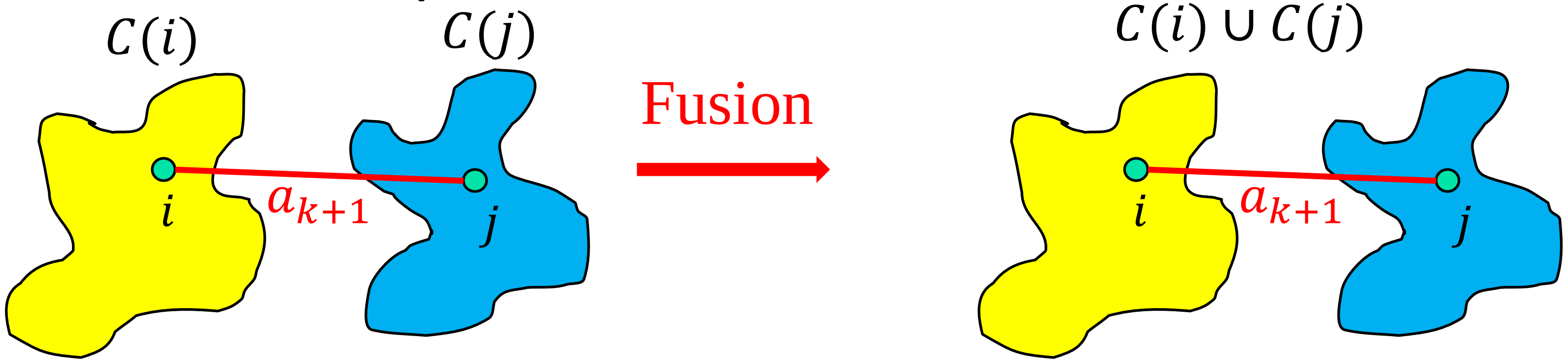


Nombre cyclomatique

Cas 2: $\boldsymbol{CC(i) \neq CC(j)}$



- Cas 2 : $CC(i) \neq CC(j)$
  - $m_{k+1} = m_k + 1$
  - $p_{k+1} = p_k - 1$
  - $v(G_{k+1}) = m_{k+1} - n + p_{k+1} = m_k + 1 - n + p_k - 1$
  - $\Rightarrow v(G_{k+1}) = v(G_k)$
  - L'ajout de l'arête a_{k+1} ne crée pas un nouveau cycle
  - Le nombre de composantes connexes diminue d'une unité



Nombre cyclomatique

- Conséquences
  - La preuve constructive de ce théorème offre
    - Un algorithme (de base) de recherche de cycles dans un graphe
    - Un algorithme pour déterminer les composantes connexes d'un graphe non orienté

- **Corollaire** : soit G un graphe non orienté, on a
  - G connexe $\rightarrow m \geq n - 1$
  - G sans cycle $\rightarrow m \leq n - 1$
  - G arbre $\rightarrow m = n - 1$