Graphes et Algorithmes – Partie IV
Parcours

FISA Informatique 1ère année

2020 - 2021

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES HAUTS-DE-FRANCE

Université Polytechnique HAUTS-DE-FRANCE

Parcours – Plan

- Exploration d’un graphe
- Algorithmes classiques de parcours
  - Parcours en largeur
  - Parcours en profondeur

Exploration d’un graphe

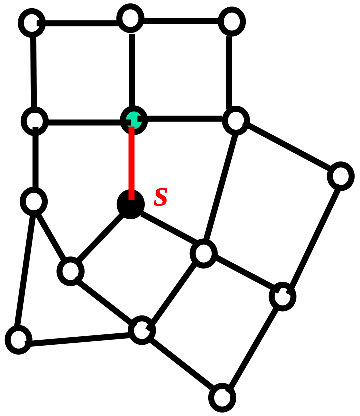
- Il existe de nombreux problèmes nécessitant l’exploration d’un graphe
  - Accessibilité
  - Cheminement
  - Connexité

Exploration d’un graphe

- Il existe de nombreux problèmes nécessitant l’exploration d’un graphe
  - Accessibilité
  - Cheminement
  - Connexité
- Algorithmes de complexité linéaires (en fonction de $|A|$)
  - Travaux notamment de Tarjan dans les années 70
  - Principe : exploration systématique des sommets du graphe

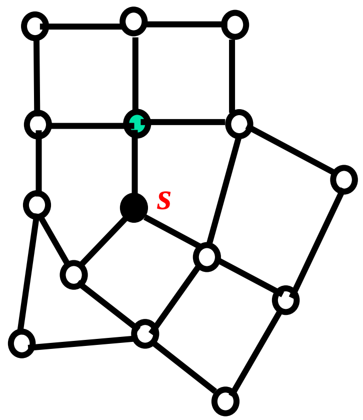
Exploration d'un graphe

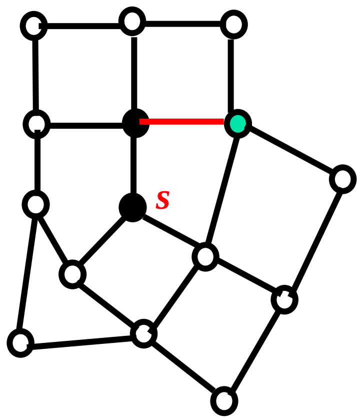
- Exploration systématique
  - Visiter les sommets du graphe en empruntant systématiquement ses arcs / arêtes



Exploration d'un graphe

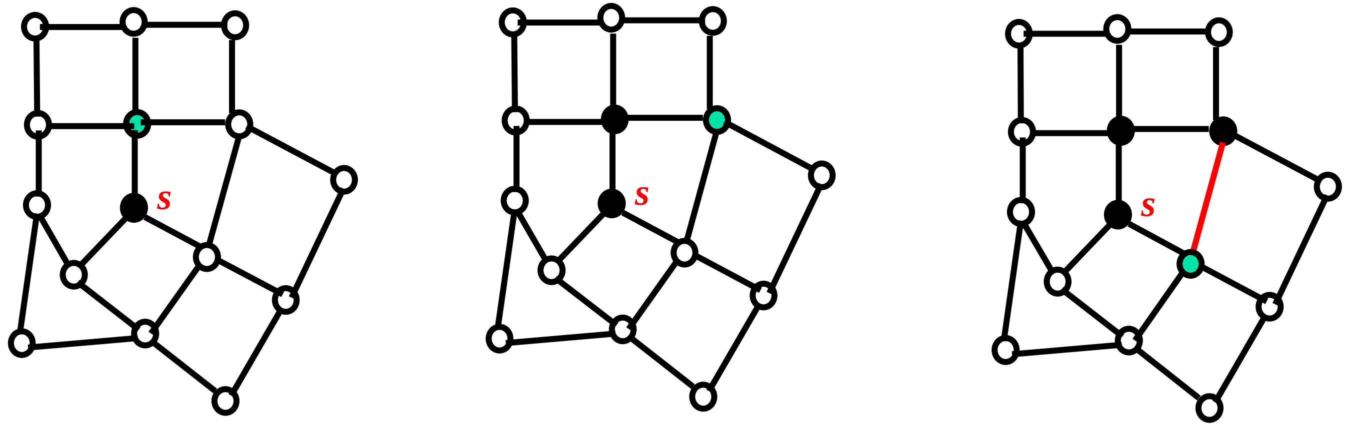
- Exploration systématique
  - Visiter les sommets du graphe en empruntant systématiquement ses arcs / arêtes





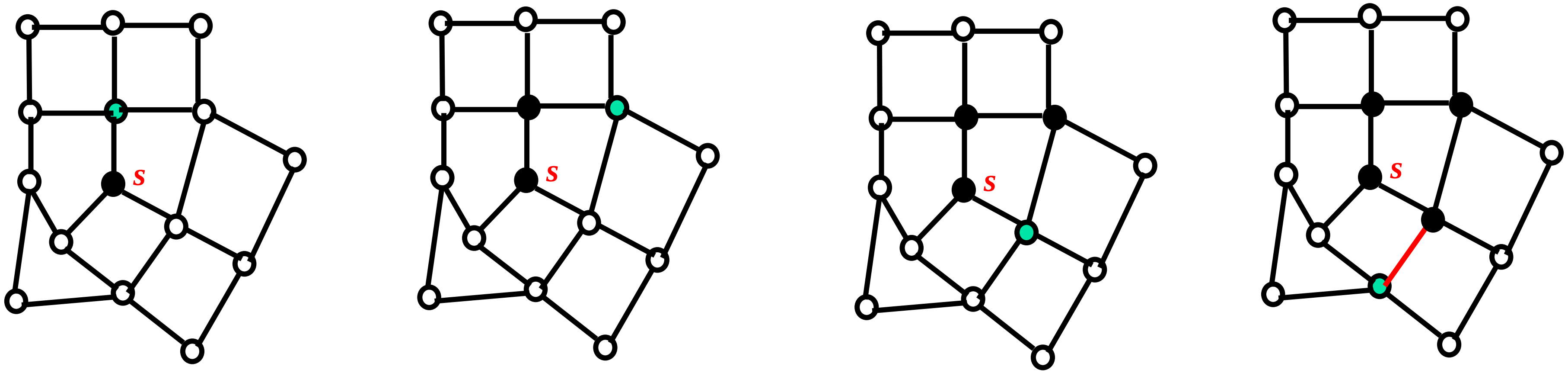
Exploration d’un graphe

- Exploration systématique
  - Visiter les sommets du graphe en empruntant systématiquement ses arcs / arêtes



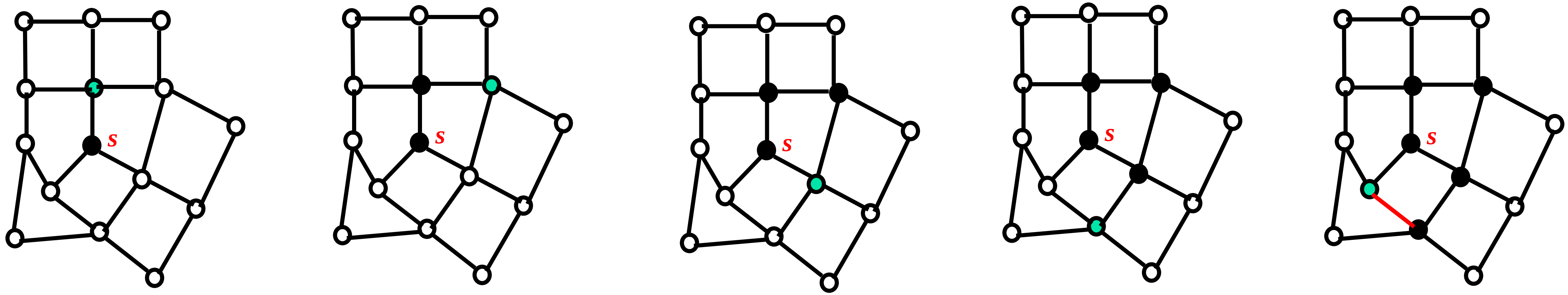
Exploration d’un graphe

- Exploration systématique
  - Visiter les sommets du graphe en empruntant systématiquement ses arcs / arêtes



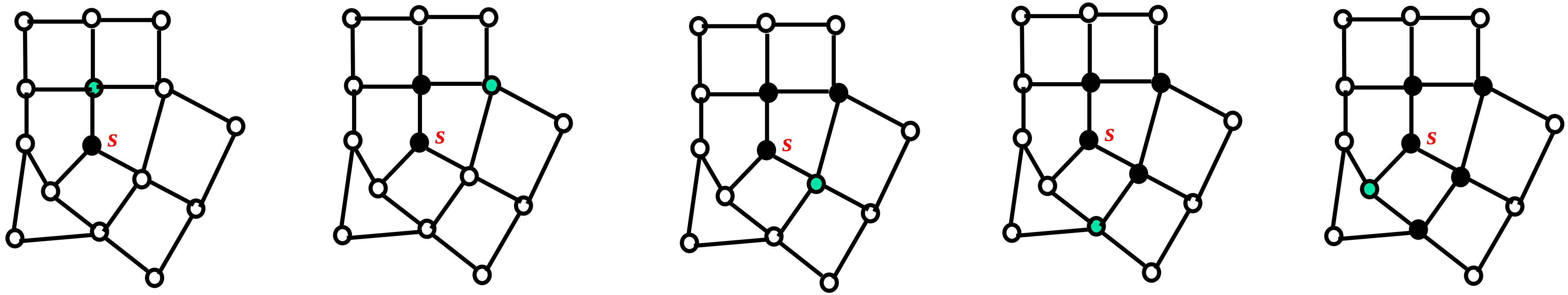
Exploration d'un graphe

- Exploration systématique
  - Visiter les sommets du graphe en empruntant systématiquement ses arcs / arêtes



Exploration d’un graphe

- Exploration systématique
  - Visiter les sommets du graphe en empruntant systématiquement ses arcs / arêtes



… … … …

Exploration d’un graphe

- Extensions des parcours d’arbres
- Méthode générale

Exploration d'un graphe

- Extensions des parcours d'arbres
- Méthode générale

```
procédure exploration (G, s ∈ S)
   VAR L : Liste;
début
   L ← {s} ;
   tant que L ≠ ∅ faire
      sélectionner x dans L ;
      L ← L – {x} ;
      marquer x comme visité
      pour tout sommet y ∈ S adjacent à x faire
         si y est non marqué alors
            L ← L + {y} ;
         fin si
   fin tant que
```

Exploration d’un graphe

- Extensions des parcours d’arbres
- Méthode générale

```
procédure exploration (G, s ∈ S)
   VAR L : Liste;
début
   L ← {s} ;
   tant que L ≠ ∅ faire
      sélectionner x dans L ;
      L ← L – {x} ;
      marquer x comme visité
      pour tout sommet y ∈ S adjacent à x faire
         si y est non marqué alors
            L ← L + {y} ;
         fin si
   fin tant que
```

Deux approches principales selon l’ordre de parcours des sommets

→ Parcours en largeur (**PeL**)
  Breadth-first Search (BFS)

→ Parcours en profondeur (**PeP**)
  Depth-first Search (DFS)

Exploration d’un graphe

- Permet de découvrir la structure d’un graphe …

Exploration d’un graphe

- Permet de découvrir la structure d’un graphe …
- … et résoudre des problèmes
  - Graphe connexe ou non
  - Trouver les composantes connexes
  - Plus court chemin
  - Recherche de cycles
  - Tri topologique
  - Trouver un arbre / une forêt couvrante
  - etc.

Parcours en largeur de $G = (S, A)$

- Notion de distance entre sommets
  - Distance $\delta(i, j)$ entre i et $j \in S$: **nombre minimal** d'arcs (arêtes) d'un chemin (une chaîne) les joignant (si existe), sinon $\delta(i, j) =+\propto$

Parcours en largeur de $G = (S, A)$

- Notion de distance entre sommets
  - Distance $\delta(i, j)$ entre i et $j \in S$: **nombre minimal** d'arcs (arêtes) d'un chemin (une chaîne) les joignant (si existe), sinon $\delta(i, j) =+\propto$
- La distance δ est une **métrique** : Pour tout $i, j, k \in S$
  1. $\delta(i, j) \geq 0, \text{avec } \delta(i, j) = 0 \; ssi \; i = j$
  2. $\delta(i, j) = \delta(j, i)$
  3. $\delta(i, j) + \delta(j, k) \geq \delta(i, k)$

Parcours en largeur de $G = (S, A)$

- Notion de distance entre sommets
  - Distance $\delta(i,j)$ entre i et $j \in S$: **nombre minimal** d'arcs (arêtes) d'un chemin (une chaîne) les joignant (si existe), sinon $\delta(i,j) =+\propto$
- La distance δ est une **métrique** : Pour tout $i, j, k \in S$
  1. $\delta(i,j) \geq 0, \text{avec}\, \delta(i,j) = 0\ ssi\ i = j$
  2. $\delta(i,j) = \delta(j,i)$
  3. $\delta(i,j) + \delta(j,k) \geq \delta(i,k)$
- Pour tout $s \in S$ et $(i,j) \in A$: $\delta(s,j) \leq \delta(s,i) + 1$

Parcours en largeur de $G = (S, A)$

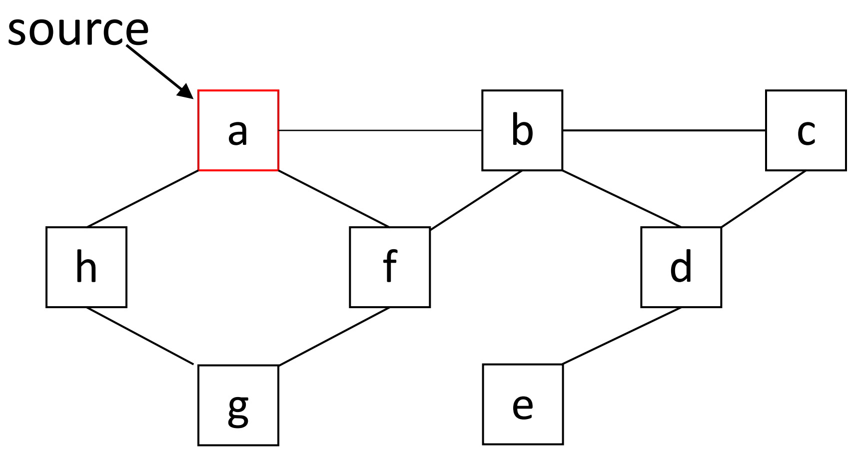
- Notion de distance entre sommets
  - Distance $\delta(i, j)$ entre i et $j \in S$: **nombre minimal** d'arcs (arêtes) d'un chemin (une chaîne) les joignant (si existe), sinon $\delta(i, j) =+\propto$
- La distance δ est une **métrique** : Pour tout $i, j, k \in S$
  1. $\delta(i, j) \geq 0, \text{avec}\, \delta(i, j) = 0 \; ssi \; i = j$
  2. $\delta(i, j) = \delta(j, i)$
  3. $\delta(i, j) + \delta(j, k) \geq \delta(i, k)$
- Pour tout $s \in S$ et $(i, j) \in A$: $\delta(s, j) \leq \delta(s, i) + 1$
- Diamètre d'un graphe connexe : $diam(G) = \max\{\delta(i, j) : i, j \in S\}$

Parcours en largeur de $G = (S, A)$

- En entrée de l'algorithme
  - Le graphe $G = (S, A)$, **orienté ou non**
  - Un sommet origine (sommet source)
- En sortie de l'algorithme
  - pour tout $i \in S$
    - $d[i] = \begin{cases} \delta(s,i) & \text{si } i \text{ est accessible depuis } s \\ +\infty & sinon \end{cases}$
    - $p[i] = j$, avec *j* le prédécesseur de i sur ce chemin (NUL si pas de chemin)

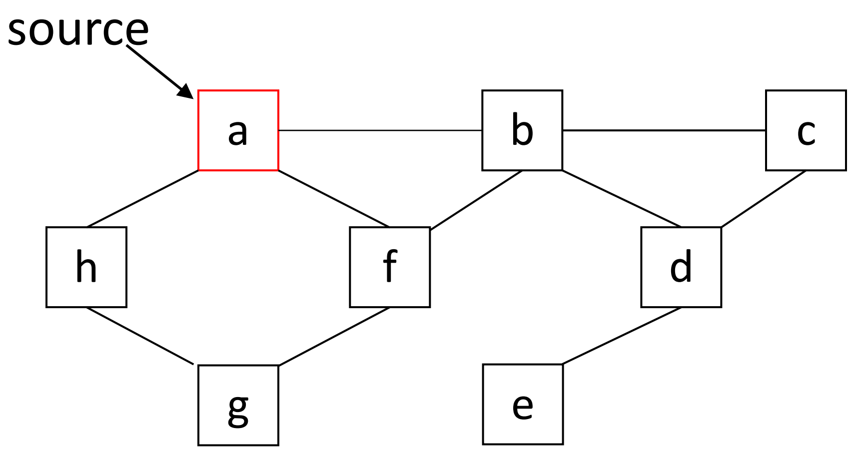
Parcours en largeur de $G = (S, A)$

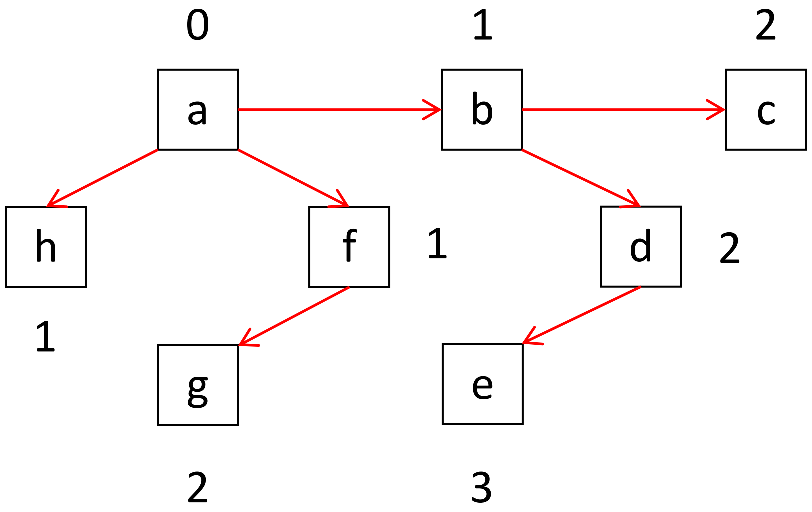
- Notion de distance entre sommets

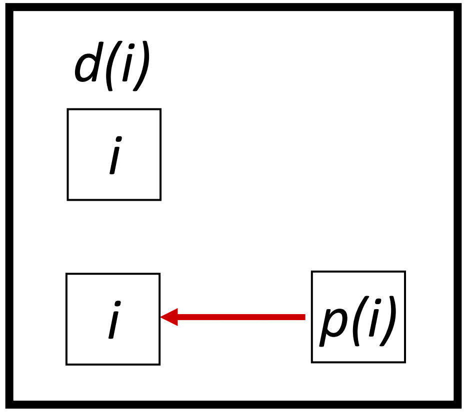


Parcours en largeur de $G = (S, A)$

- Notion de distance entre sommets



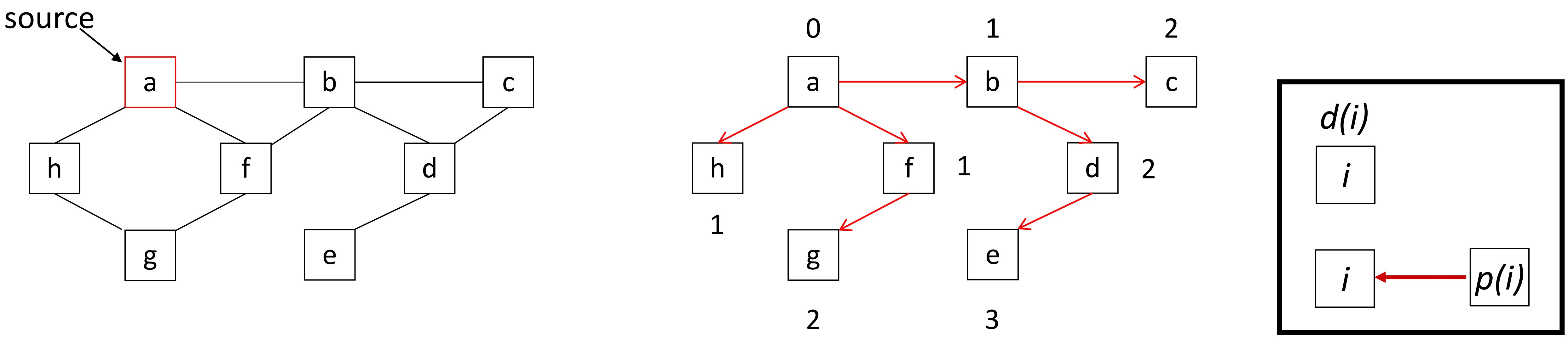




- En traçant le graphe contenant tous les arcs $(p[i], i)$, on obtient l'arborescence des plus courts chemins, avec s comme racine

Parcours en largeur de $G = (S, A)$

- Notion de distance entre sommets



- En traçant le graphe contenant tous les arcs $(p[i], i)$, on obtient l'arborescence des plus courts chemins, avec s comme racine
- L'arbre PeL de racine s contient tous les sommets accessibles depuis s

Parcours en largeur de $G = (S, A)$

- Principe : visite les sommets à une distance k avant ceux à une distance $k + 1$

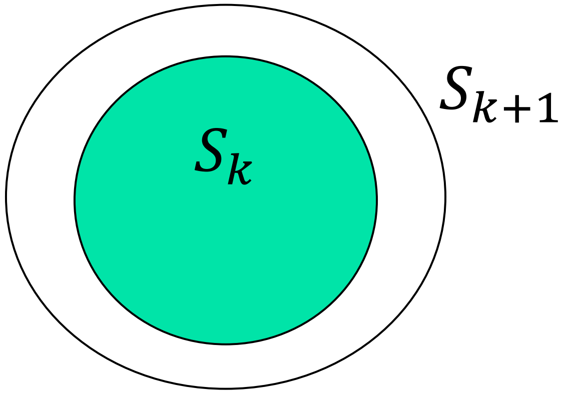
Parcours en largeur de $G = (S, A)$

- Principe : visite les sommets à une distance k avant ceux à une distance $k + 1$
- Élargit la frontière entre les sommets découverts et non découverts uniformément sur toute la largeur de la frontière
  - $s \in S$ est «découvert» la 1ère fois qu'il est rencontré pendant la recherche
  - $s \in S$ est «terminé» si tous les sommets adjacents ont été découverts

Parcours en largeur de $G = (S, A)$

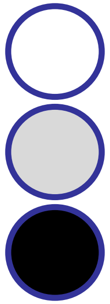
- Principe : visite les sommets à une distance k avant ceux à une distance $k + 1$
- Élargit la frontière entre les sommets découverts et non découverts uniformément sur toute la largeur de la frontière
  - $s \in S$ est «découvert» la 1ère fois qu'il est rencontré pendant la recherche
  - $s \in S$ est «terminé» si tous les sommets adjacents ont été découverts

$$S_k = \{i \in S : \delta(s, i) = k\}$$



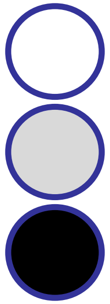
Parcours en largeur de $G = (S, A)$

- On utilise des **couleurs** pour suivre la progression de l'algorithme
  - Blanc : sommet pas encore découvert
  - Gris : sommet découvert mais pas encore terminé
  - Noir : sommet terminé
  - On note $\boldsymbol{c}[\boldsymbol{i}]$ **la couleur du sommet** $\boldsymbol{i} \in \boldsymbol{S}$



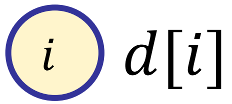
Parcours en largeur de $G = (S, A)$

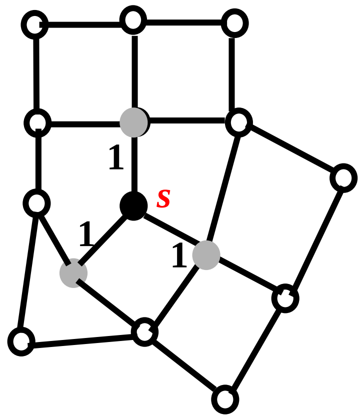
- On utilise des couleurs pour suivre la progression de l'algorithme
  - Blanc : sommet pas encore découvert
  - Gris : sommet découvert mais pas encore terminé
  - Noir : sommet terminé
  - On note $c[i]$ la couleur du sommet $i \in S$



- Remarque
  - Les couleurs ne sont pas obligatoires pour implémenter l'algorithme

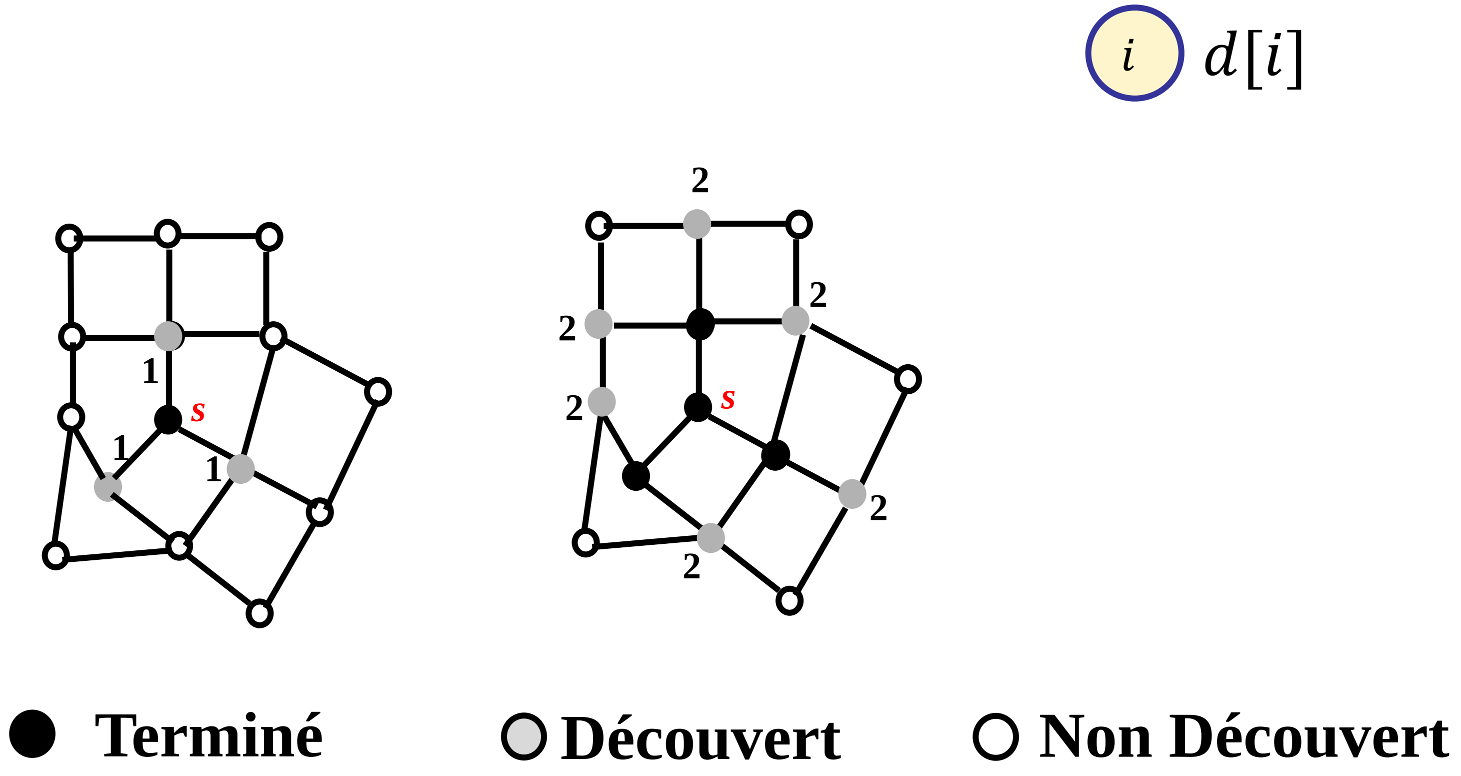
Parcours en largeur de $G = (S, A)$



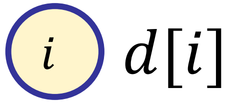


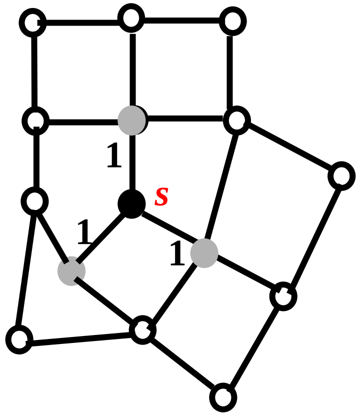
● **Terminé** ◉ **Découvert** ○ **Non Découvert**

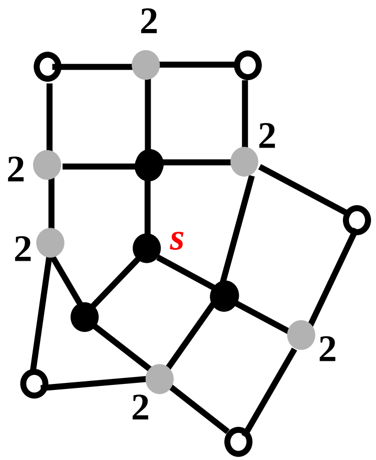
Parcours en largeur de $G = (S, A)$

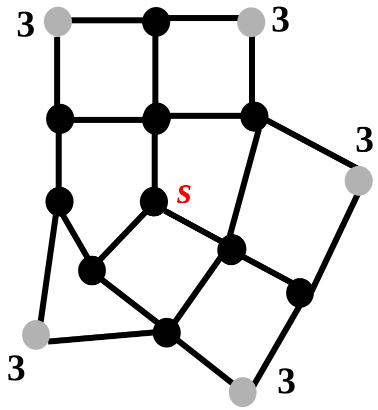


Parcours en largeur de $G = (S, A)$





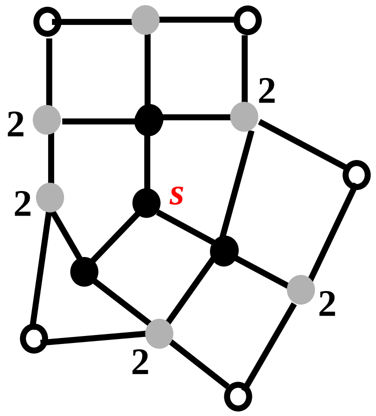




Terminé **Découvert** **Non Découvert**

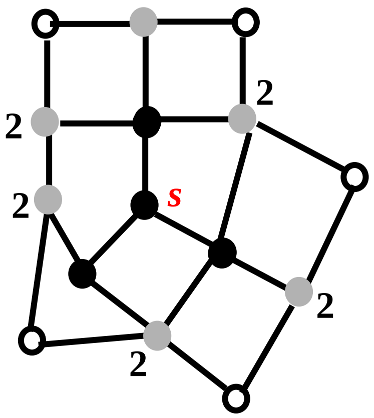
Parcours en largeur de $G = (S, A)$

- Les sommets colorés en gris représentent la frontière entre les sommets découverts et ceux non encore atteints



Parcours en largeur de $G = (S, A)$

- Les sommets colorés en gris représentent la frontière entre les sommets découverts et ceux non encore atteints



- L'algorithme utilise une File (FIFO)
  - « Premier arrivé premier servi » (*First In First Out*)
  - La file contient les sommets gris du graphe

Parcours en largeur de $G = (S, A)$

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

procédure PEL (Entrée : G, s $\in$ S
Sortie : d, p)

$$\left.\begin{array}{l}
\text{pour } x \in S - \{s\} \text{ faire} \\
\quad c[x] \leftarrow \text{Blanc ;} \\
\quad d[x] \leftarrow + \propto \\
\quad p[x] \leftarrow \text{Nul ;} \\
\text{fin pour} \\
c[s] \leftarrow \text{Gris; } d[s] \leftarrow 0;\ p[s] \leftarrow \text{Nul ;} \\
F \leftarrow \varnothing \text{ ; enfiler(F, s) ;}
\end{array}\right\} \textbf{Initialisation}$$

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

Parcours en largeur de $G = (S, A)$

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

procédure PEL (Entrée : G, s $\in$ 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) ;
```

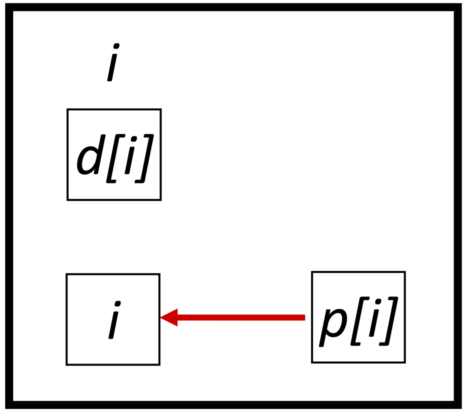
Traitement

```
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
```

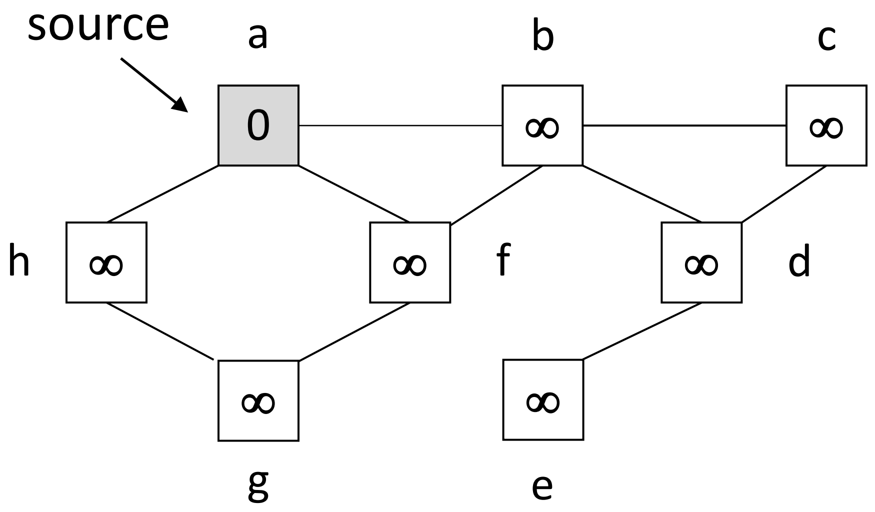
$c[i]$: couleur de $i \in S$
Blanc : Non découvert
Gris : Découvert
Noir : Terminé
F : file des sommets Gris

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

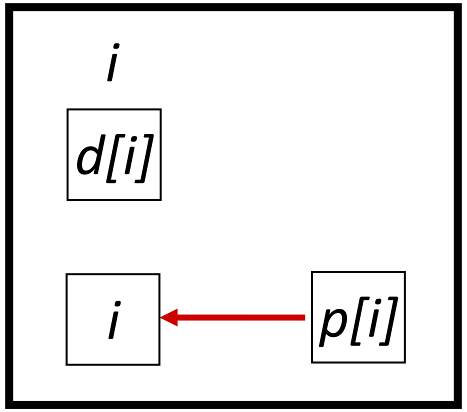
Parcours en largeur de $G = (S, A)$



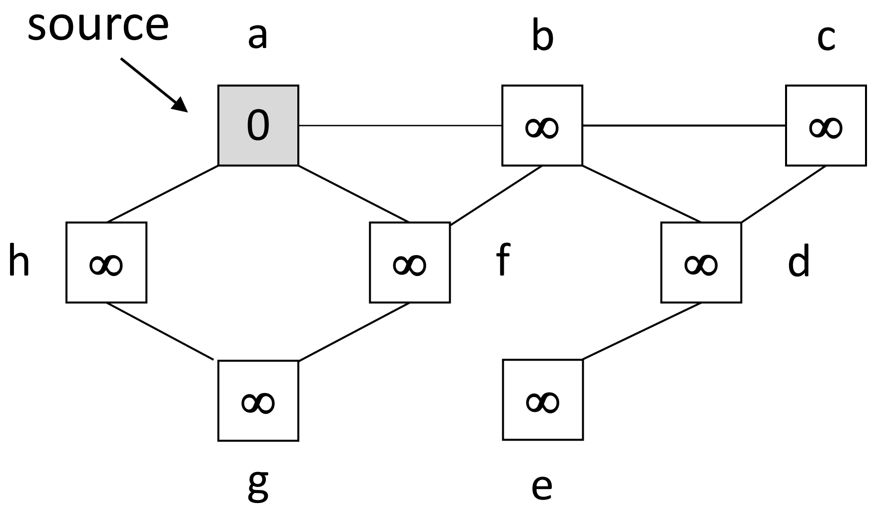
- Exemple



Parcours en largeur de $G = (S, A)$



- Exemple

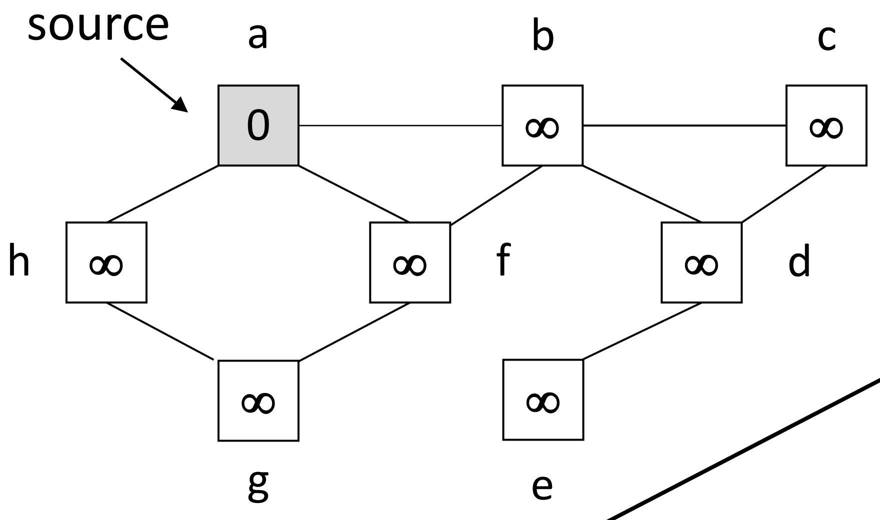


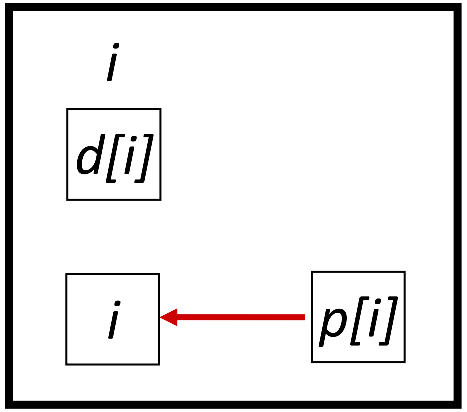
F = **a**
$d[i]$ = 0
$p[i]$ = NUL

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) ;

Parcours en largeur de $G = (S, A)$

- Exemple



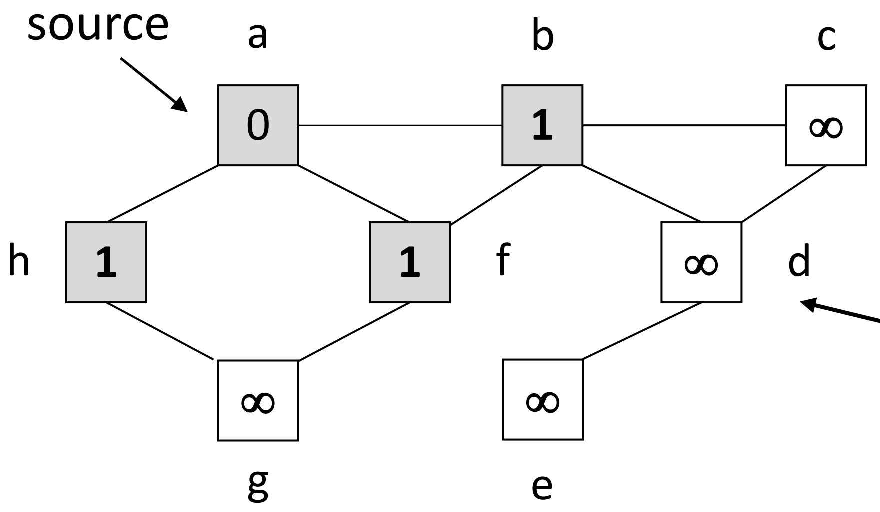


tant que F $\neq \emptyset$ faire
    $\boxed{\text{x} \leftarrow \text{défiler(F);}}$ → X = a
    pour y $\in Adj$[x] faire
        si (c[y] = Blanc) alors
            c[y] $\leftarrow$ gris;
            d[y] $\leftarrow$ d[x] + 1; p[y] $\leftarrow$ x;
            enfiler(F, y);
        fin si
    fin pour
    c[x] $\leftarrow$ noir;
fin tant que

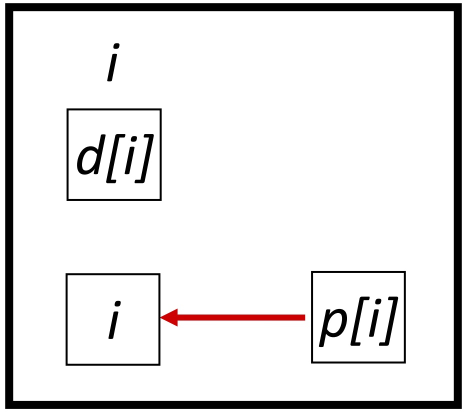
F = ~~a~~ $\emptyset$
$d[i]$ = 0
$p[i]$ = NUL

Parcours en largeur de $G = (S, A)$

- Exemple



F =	**b f h**
$d[i]$ =	1 1 1
$p[i]$ =	a a a

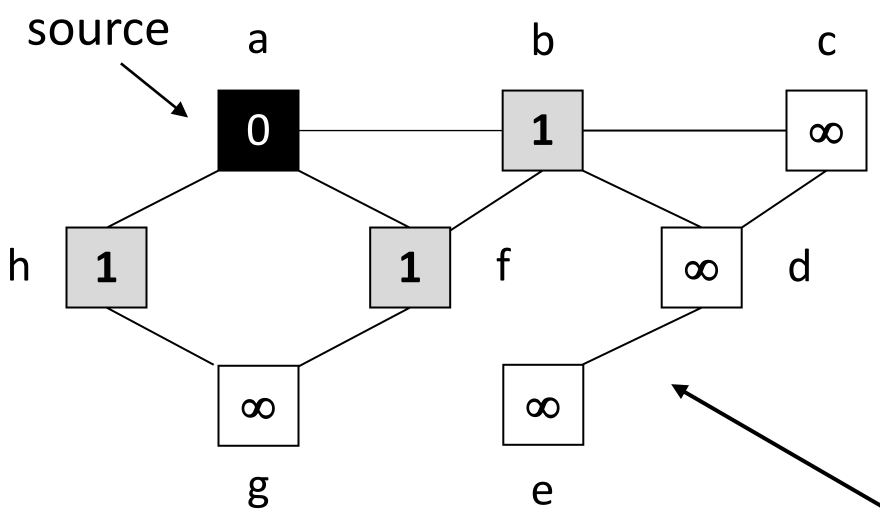


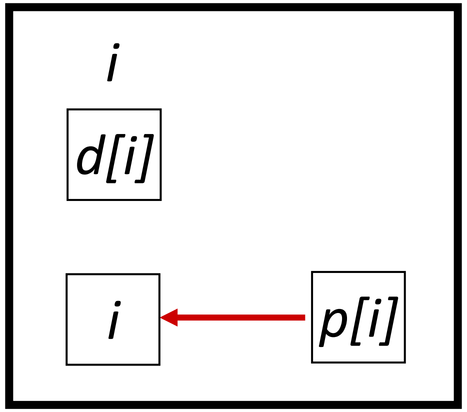
tant que F $\neq \emptyset$ faire
    x $\leftarrow$ défiler(F);
    pour y $\in Adj[\mathrm{x}]$ faire
        si (c[y] = Blanc) alors
            c[y] $\leftarrow$ gris;
            d[y] $\leftarrow$ d[x] + 1; p[y] $\leftarrow$ x;
            enfiler(F, y);
        fin si
    fin pour
    c[x] $\leftarrow$ noir;
fin tant que

X = a

Parcours en largeur de $G = (S, A)$

- Exemple





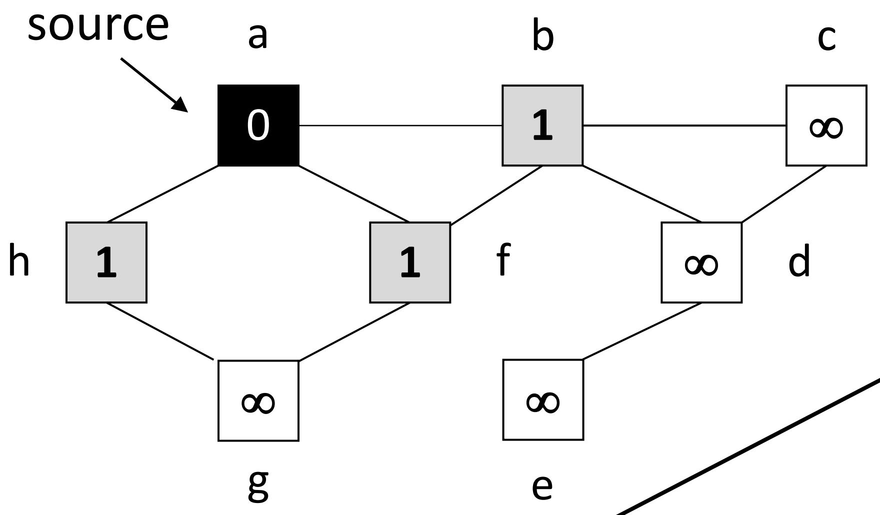
$x = a$

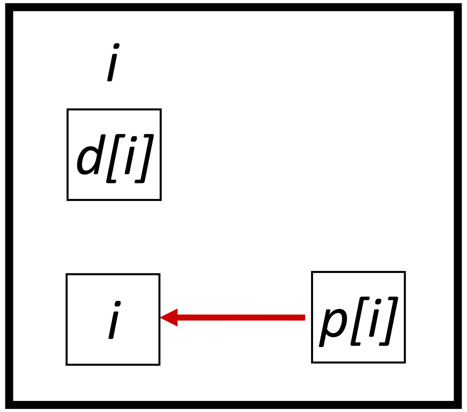
F =	**b f h**
$d[i]$ =	1 1 1
$p[i]$ =	a a a

tant que F $\neq \emptyset$ faire
    x $\leftarrow$ défiler(F);
    pour y $\in Adj[\mathrm{x}]$ faire
        si (c[y] = Blanc) alors
            c[y] $\leftarrow$ gris;
            d[y] $\leftarrow$ d[x] + 1; p[y] $\leftarrow$ x;
            enfiler(F, y);
        fin si
    fin pour
    $\boxed{\text{c[x]} \leftarrow \text{noir;}}$
fin tant que

Parcours en largeur de $G = (S, A)$

- Exemple



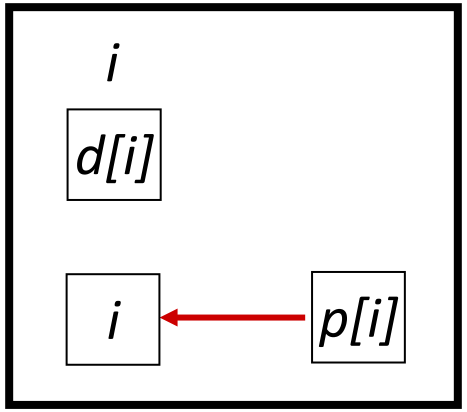



```
tant que F ≠ ∅ faire
    x ← défiler(F);
    pour y ∈ Adj[x] faire
        si (c[y] = Blanc) alors
            c[y] ← gris;
            d[y] ← d[x] + 1;  p[y] ← x;
            enfiler(F, y);
        fin si
    fin pour
    c[x] ← noir;
fin tant que
```

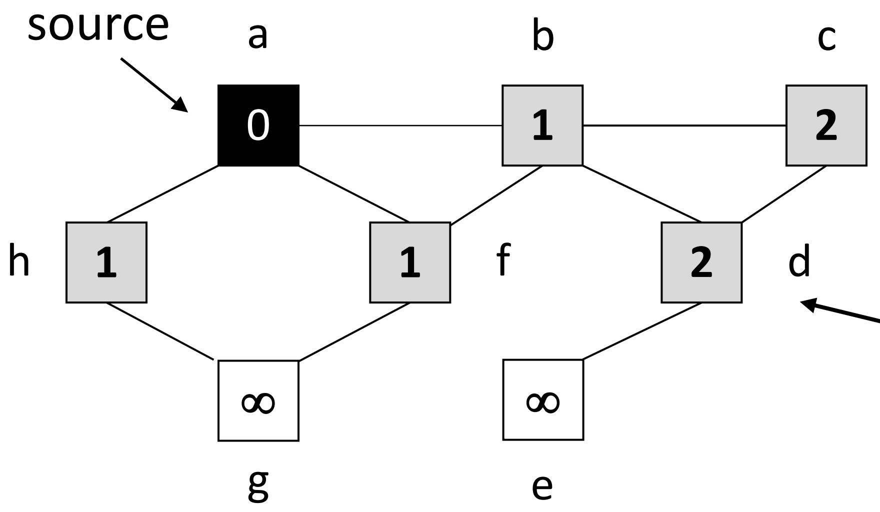
x = b

F = ~~**b**~~ **f h**
$d[i]$ = ~~1~~ 1 1
$p[i]$ = ~~a~~ a a

Parcours en largeur de $G = (S, A)$



- Exemple



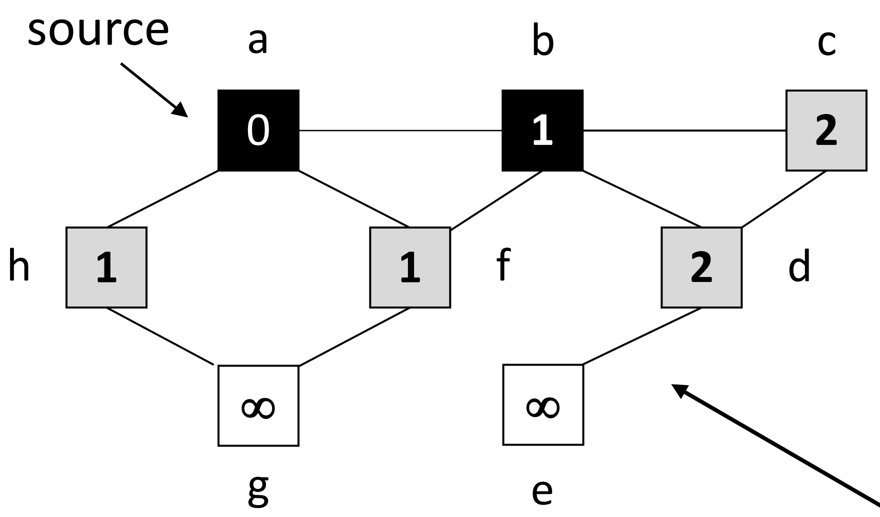
F =	**f h c d**
$d[i]$ =	1 1 2 2
$p[i]$ =	a a b b

tant que F $\neq \emptyset$ faire
  x $\leftarrow$ défiler(F);
  pour y ∈ $Adj[\mathrm{x}]$ faire
    si (c[y] = Blanc) alors
      c[y] $\leftarrow$ gris;
      d[y] $\leftarrow$ d[x] + 1; p[y] $\leftarrow$ x;
      enfiler(F, y);
    fin si
  fin pour
  c[x] $\leftarrow$ noir;
fin tant que

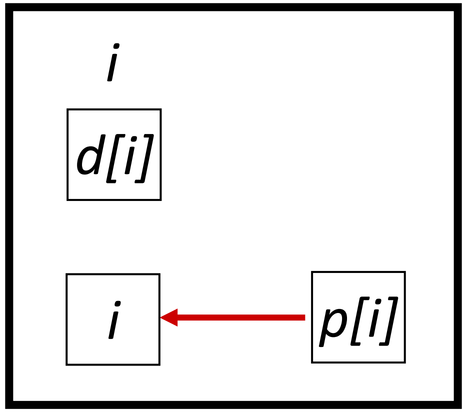
x = b

Parcours en largeur de $G = (S, A)$

- Exemple



F =	**f h c d**
$d[i]$ =	1 1 2 2
$p[i]$ =	a a b b

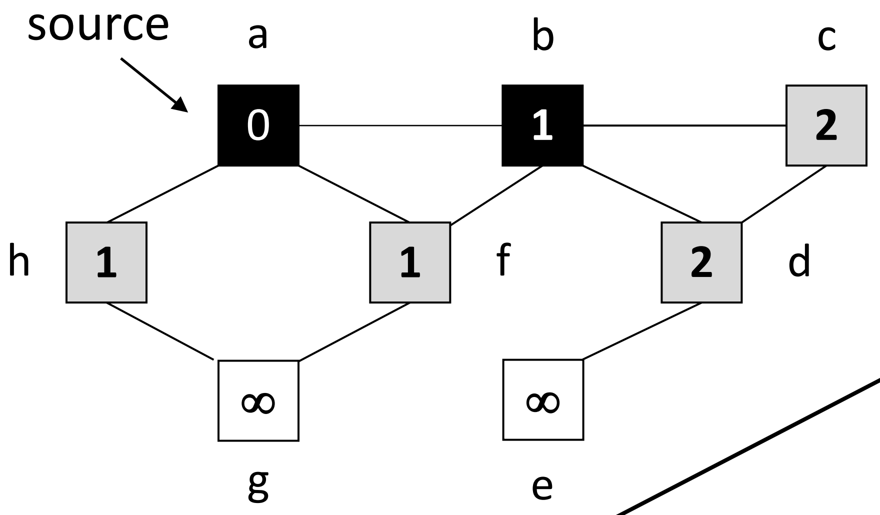


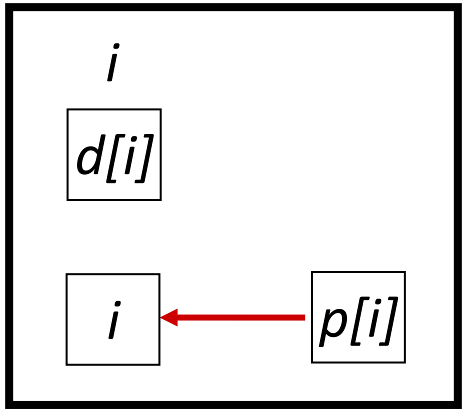
x = b

tant que F $\neq \emptyset$ faire
    x $\leftarrow$ défiler(F);
    pour y ∈ $Adj[\mathrm{x}]$ faire
        si (c[y] = Blanc) alors
            c[y] $\leftarrow$ gris;
            d[y] $\leftarrow$ d[x] + 1; p[y] $\leftarrow$ x;
            enfiler(F, y);
        fin si
    fin pour
    $\boxed{\text{c[x]} \leftarrow \text{noir;}}$
fin tant que

Parcours en largeur de $G = (S, A)$

- Exemple



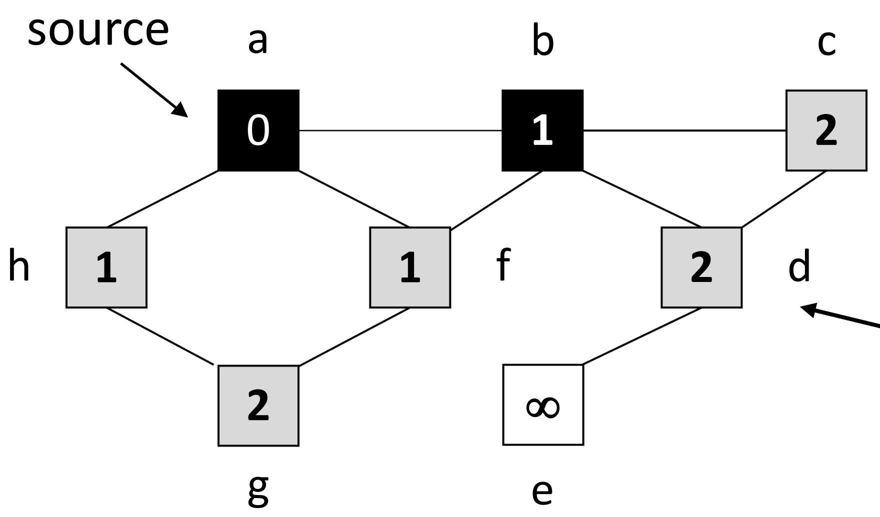


F =	**~~f~~ h c d**
$d[i]$ =	~~1~~ 1 2 2
$p[i]$ =	~~a~~ a b b

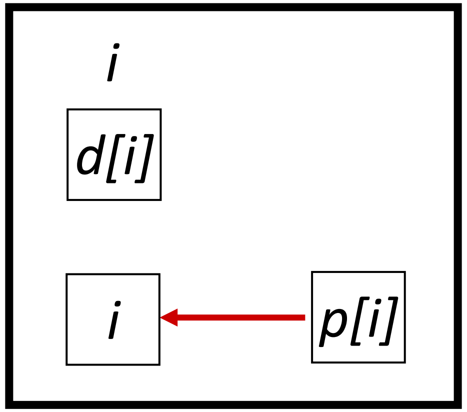
tant que F ≠ ∅ faire
    $\boxed{\text{x} \leftarrow \text{défiler(F);}}$ → x = f
    pour y ∈ $Adj[\mathrm{x}]$ faire
        si (c[y] = Blanc) alors
            c[y] ← gris;
            d[y] ← d[x] + 1; p[y] ← x;
            enfiler(F, y);
        fin si
    fin pour
    c[x] ← noir;
fin tant que

Parcours en largeur de $G = (S, A)$

- Exemple



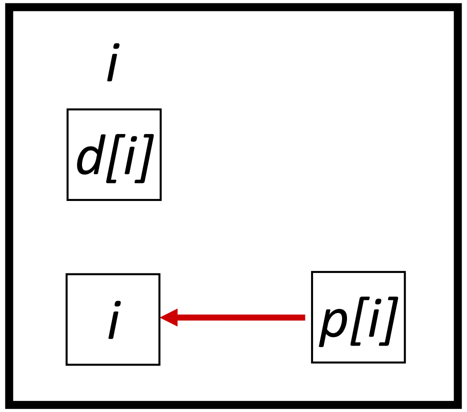
F =	**h c d g**
$d[i]$ =	1 2 2 2
$p[i]$ =	a b b f



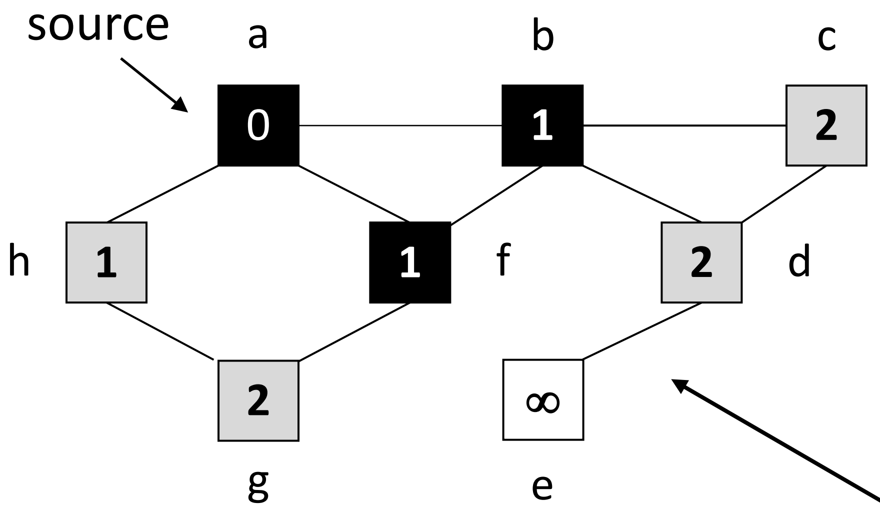

```
tant que F ≠ ∅ faire
    x ← défiler(F);
    pour y ∈ Adj[x] faire
        si (c[y] = Blanc) alors
            c[y] ← gris;
            d[y] ← d[x] + 1;  p[y] ← x;
            enfiler(F, y);
        fin si
    fin pour
    c[x] ← noir;
fin tant que
```

x = f

Parcours en largeur de $G = (S, A)$



- Exemple

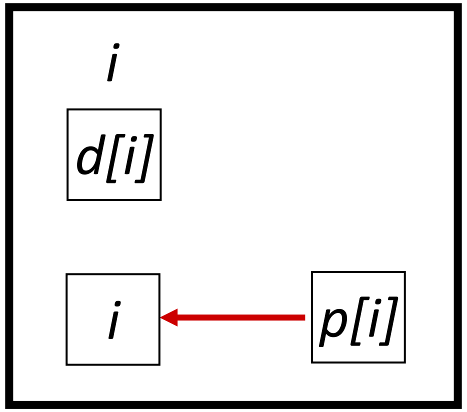


F =	**h c d g**
$d[i]$ =	1 2 2 2
$p[i]$ =	a b b f

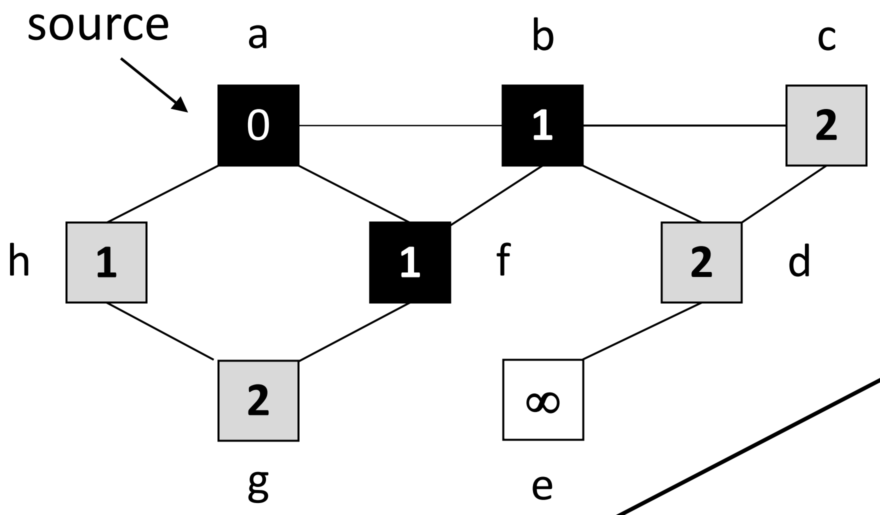
```
tant que F ≠ ∅ faire
    x ← défiler(F);
    pour y ∈ Adj[x] faire
        si (c[y] = Blanc) alors
            c[y] ← gris;
            d[y] ← d[x] + 1;  p[y] ← x;
            enfiler(F, y);
        fin si
    fin pour
    c[x] ← noir;
fin tant que
```

x = f

Parcours en largeur de $G = (S, A)$



- Exemple



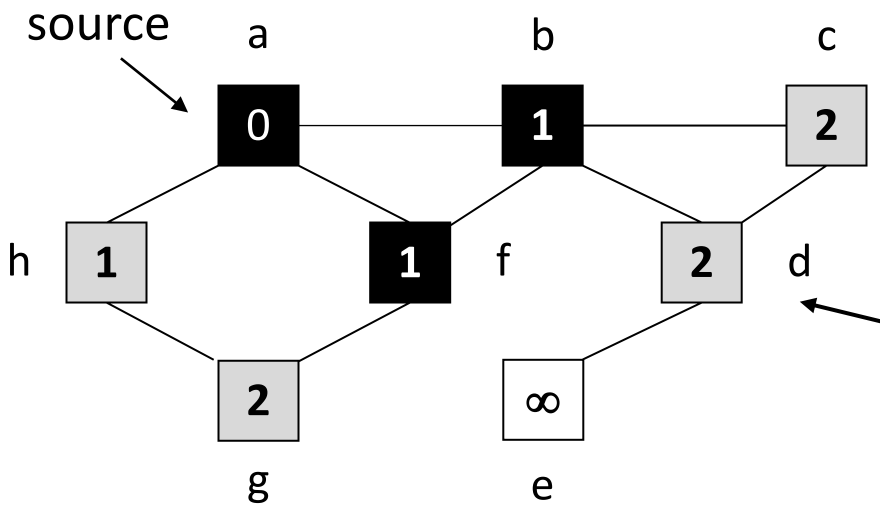

```
tant que F ≠ ∅ faire
    x ← défiler(F);
    pour y ∈ Adj[x] faire
        si (c[y] = Blanc) alors
            c[y] ← gris;
            d[y] ← d[x] + 1;  p[y] ← x;
            enfiler(F, y);
        fin si
    fin pour
    c[x] ← noir;
fin tant que
```

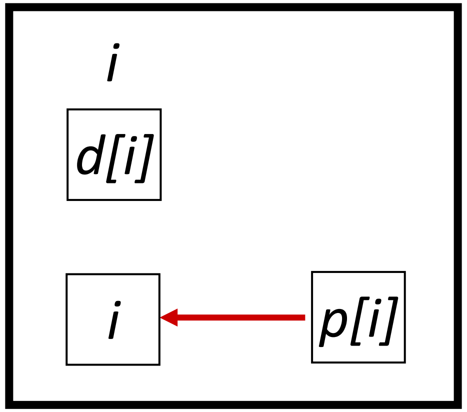
x = h

F =	**~~h~~ c d g**
$d[i]$ =	~~1~~ 2 2 2
$p[i]$ =	~~a~~ b b f

Parcours en largeur de $G = (S, A)$

- Exemple





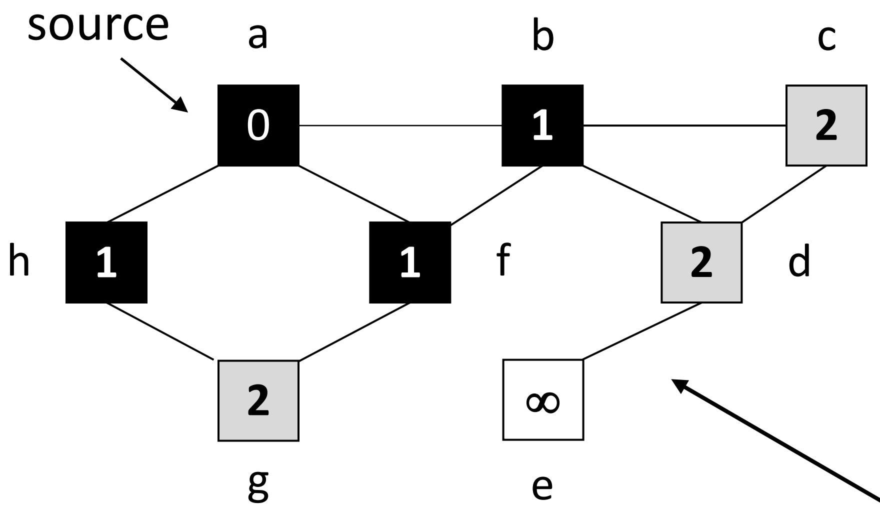
F =	**c d g**
$d[i]$ =	2 2 2
$p[i]$ =	b b f

```
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
```

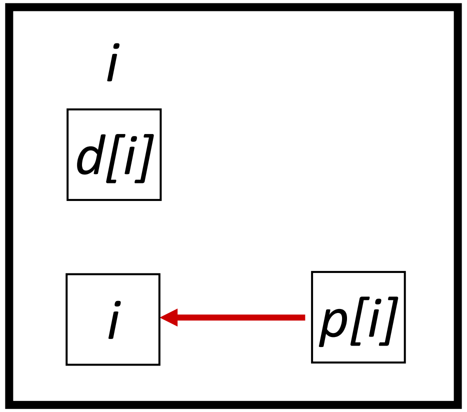
x = h

Parcours en largeur de $G = (S, A)$

- Exemple



F =	c d g
$d[i]$ =	2 2 2
$p[i]$ =	b b f

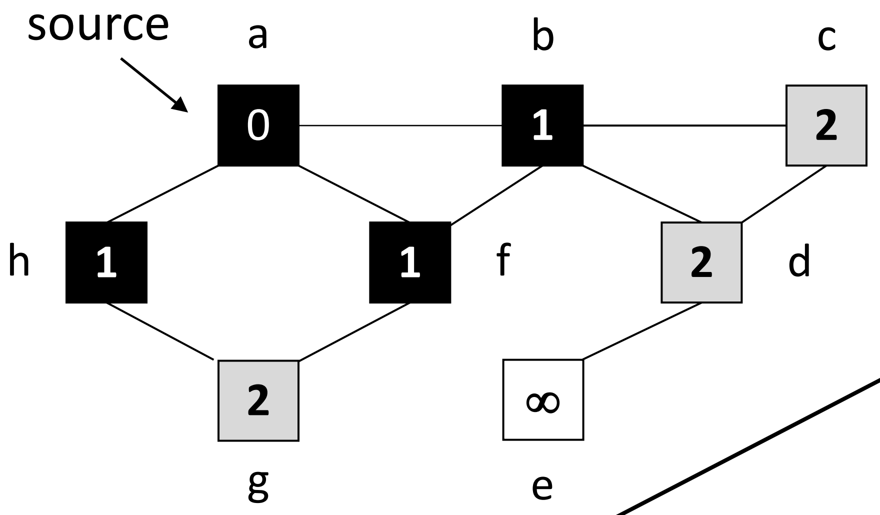


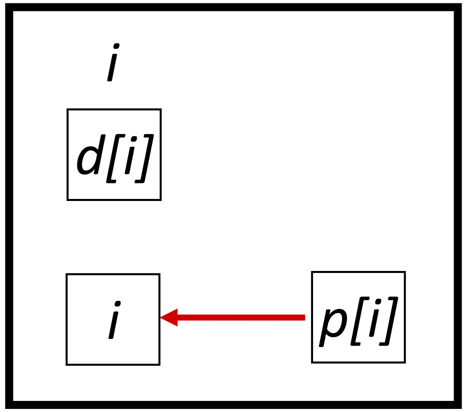
x = h

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
    $\boxed{\text{c[x] ← noir;}}$
fin tant que

Parcours en largeur de $G = (S, A)$

- Exemple





tant que F ≠ ∅ faire

  $\boxed{\text{x} \leftarrow \text{défiler(F);}}$ → x = c

  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

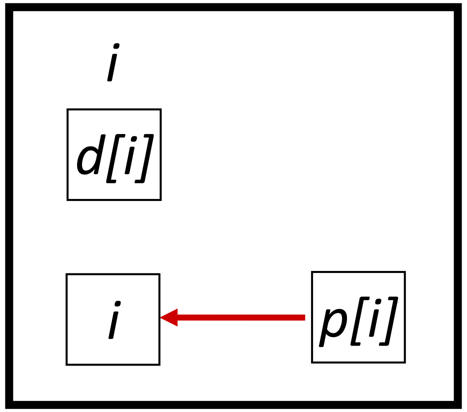
  fin pour

  c[x] ← noir;

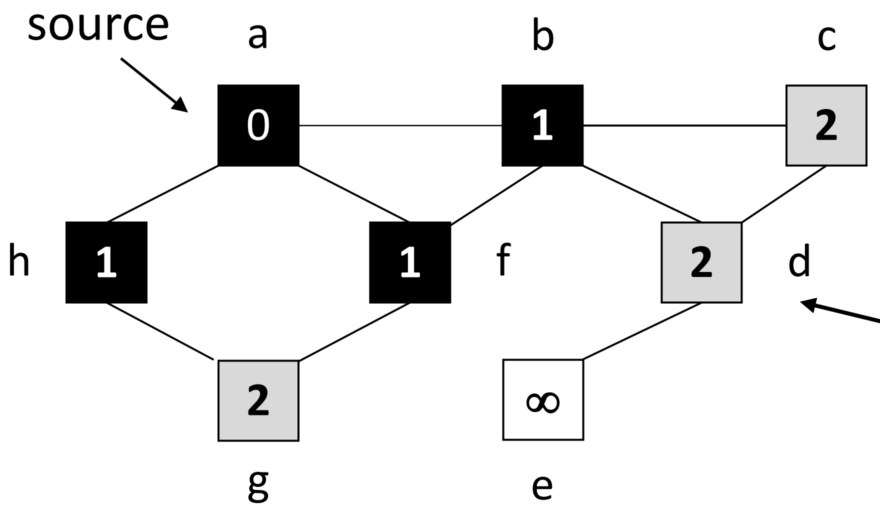
fin tant que

F =	**~~c~~ d g**
$d[i]$ =	~~2~~ 2 2
p[*i*] =	~~b~~ b f

Parcours en largeur de $G = (S, A)$



- Exemple



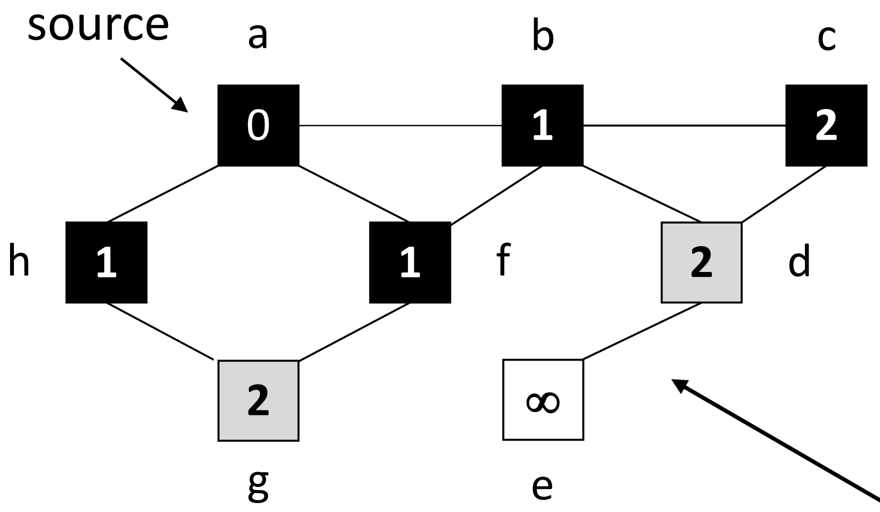
F =	**d g**
$d[i]$ =	2 2
$p[i]$ =	b f

```
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
```

x = c

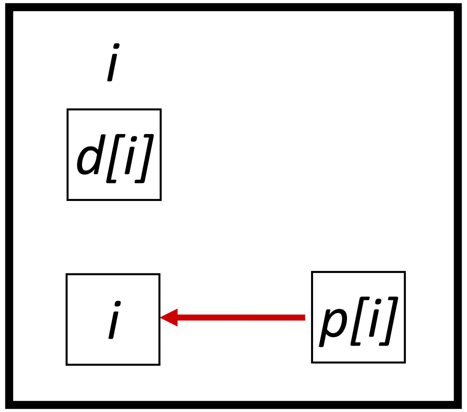
Parcours en largeur de $G = (S, A)$

- Exemple



F =	**d g**
$d[i]$ =	2 2
$p[i]$ =	b f

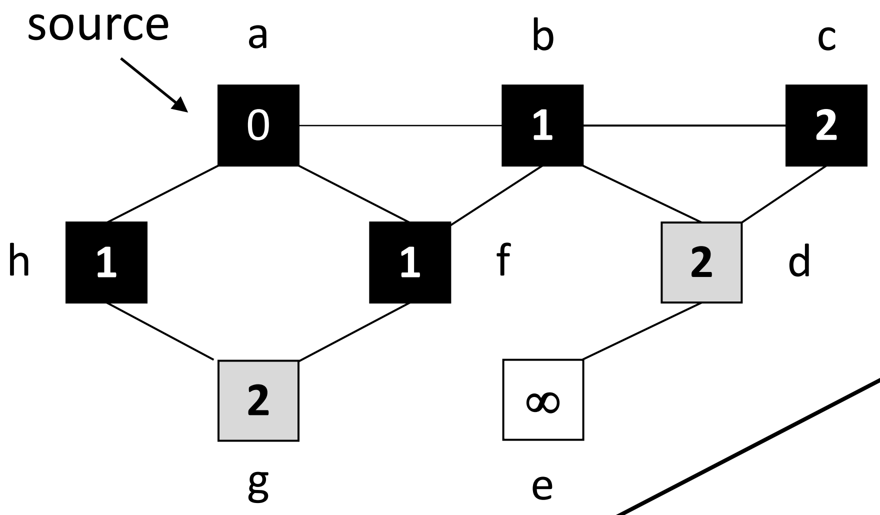
tant que F $\neq \emptyset$ faire
    x $\leftarrow$ défiler(F);
    pour y $\in Adj[\mathrm{x}]$ faire
        si (c[y] = Blanc) alors
            c[y] $\leftarrow$ gris;
            d[y] $\leftarrow$ d[x] + 1; p[y] $\leftarrow$ x;
            enfiler(F, y);
        fin si
    fin pour
    $\boxed{\text{c[x]} \leftarrow \text{noir;}}$
fin tant que

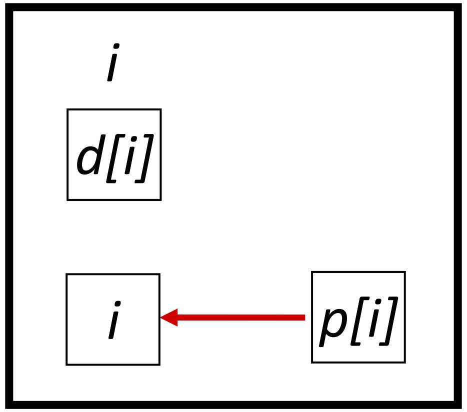


X = c

Parcours en largeur de $G = (S, A)$

- Exemple





tant que F ≠ ∅ faire

  $\boxed{\text{x} \leftarrow \text{défiler(F);}}$ → x = d

  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

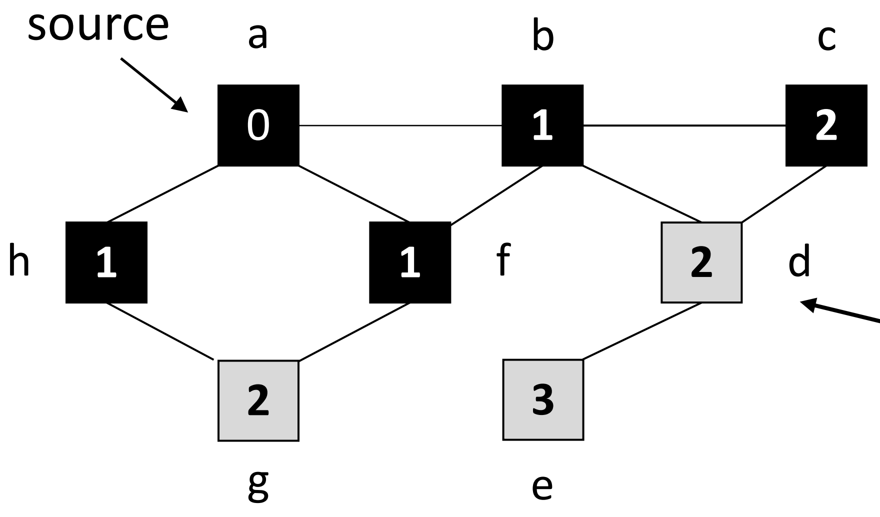
  c[x] ← noir;

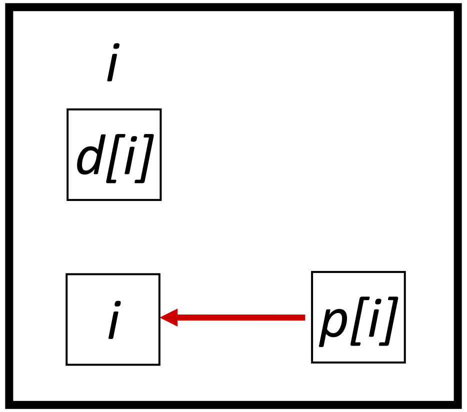
fin tant que

F =	**~~d~~ g**
$d[i]$ =	~~2~~ 2
p[*i*] =	~~b~~ f

Parcours en largeur de $G = (S, A)$

- Exemple





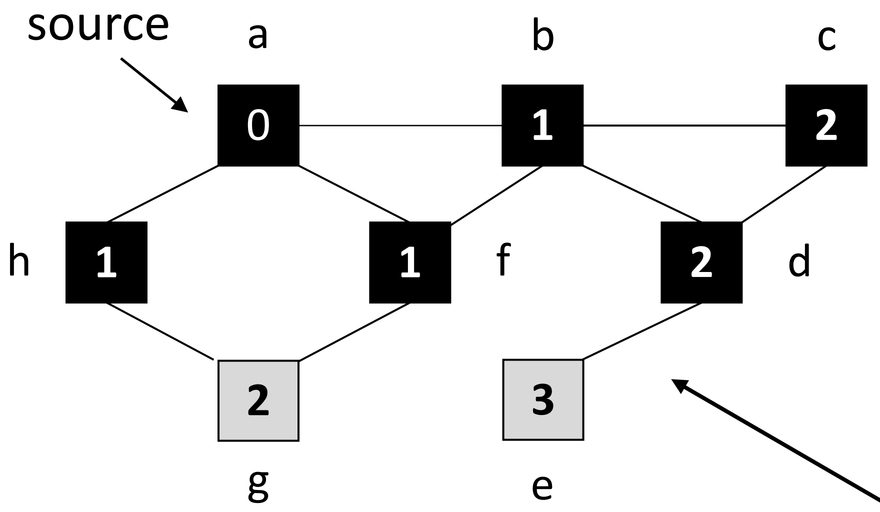
F =	**g e**
$d[i]$ =	2 3
$p[i]$ =	f d

x = d

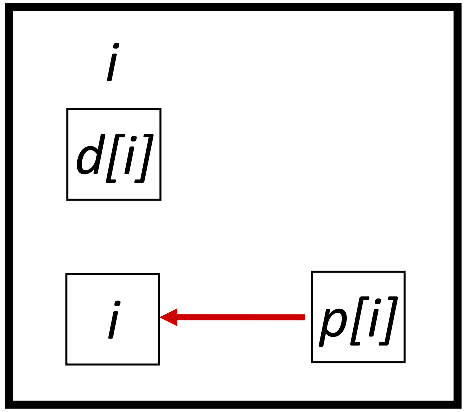
tant que F ≠ ∅ faire
    x ← défiler(F);
    pour y ∈ $Adj[\mathrm{x}]$ faire
        si (c[y] = Blanc) alors
            c[y] ← gris;
            d[y] ← d[x] + 1; p[y] ← x;
            enfiler(F, y);
        fin si
    fin pour
    c[x] ← noir;
fin tant que

Parcours en largeur de $G = (S, A)$

- Exemple



F = **g e**
$d[i]$ = 2 3
$p[i]$ = f d

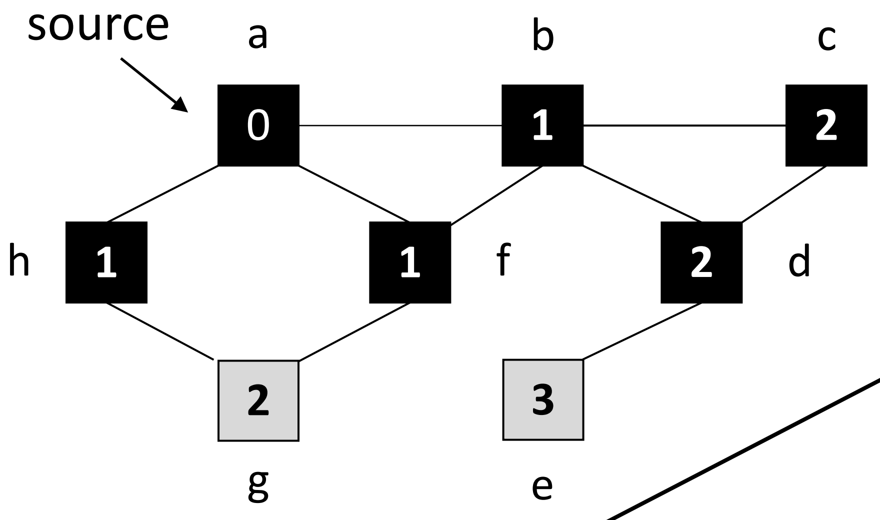


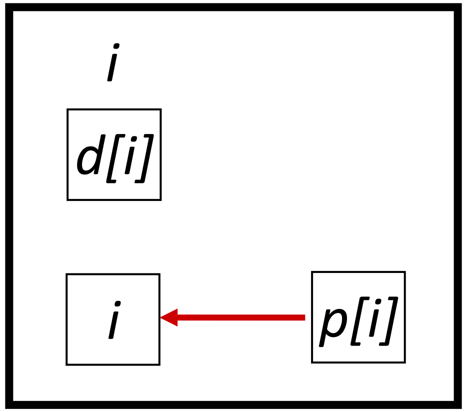
x = d

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
 $\boxed{\text{c[x] ← noir;}}$
fin tant que

Parcours en largeur de $G = (S, A)$

- Exemple





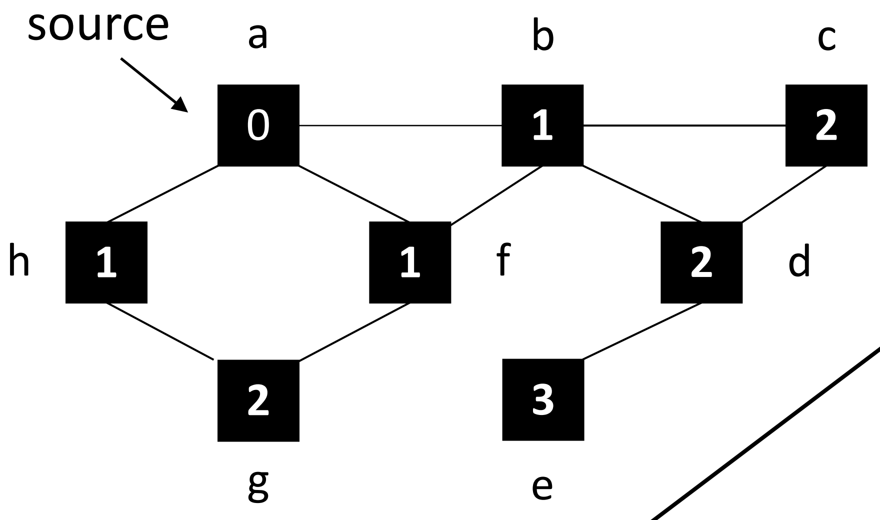

```
tant que F ≠ ∅ faire
    x ← défiler(F);
    pour y ∈ Adj[x] faire
        si (c[y] = Blanc) alors
            c[y] ← gris;
            d[y] ← d[x] + 1;  p[y] ← x;
            enfiler(F, y);
        fin si
    fin pour
    c[x] ← noir;
fin tant que
```

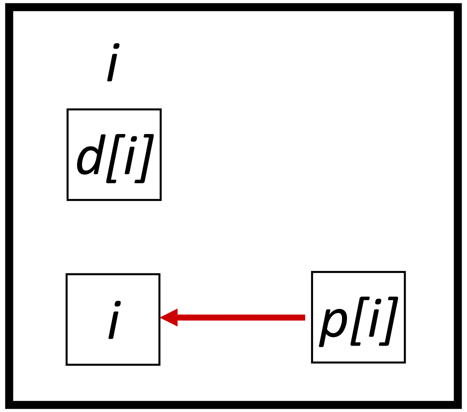
x = g puis e

F = ~~g e~~ ∅
$d[i]$ = ~~2 3~~
$p[i]$ = ~~f d~~

Parcours en largeur de $G = (S, A)$

- Exemple





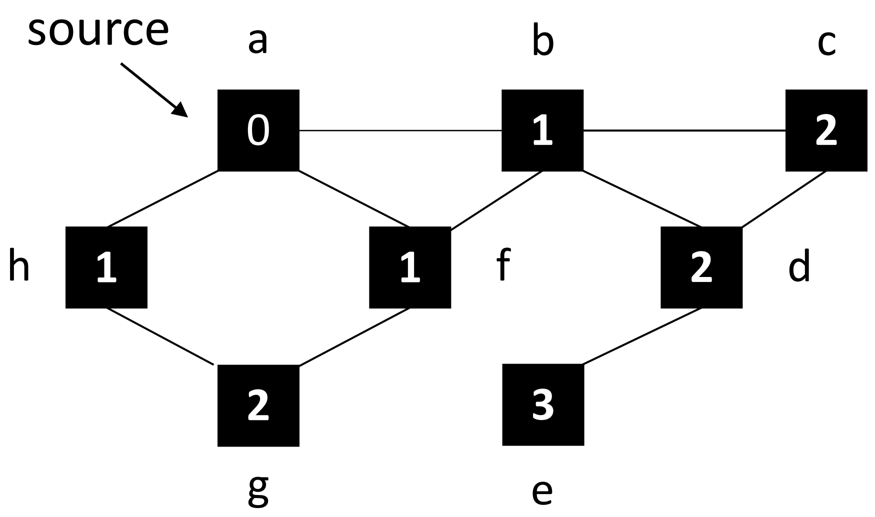
F = ~~g e~~ $\varnothing$
$d[i]$ = ~~2 3~~
$p[i]$ = ~~f d~~

tant que F $\neq \varnothing$ faire → stop

    x ← défiler(F);
    pour y ∈ $Adj[\mathrm{x}]$ 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

Parcours en largeur de $G = (S, A)$

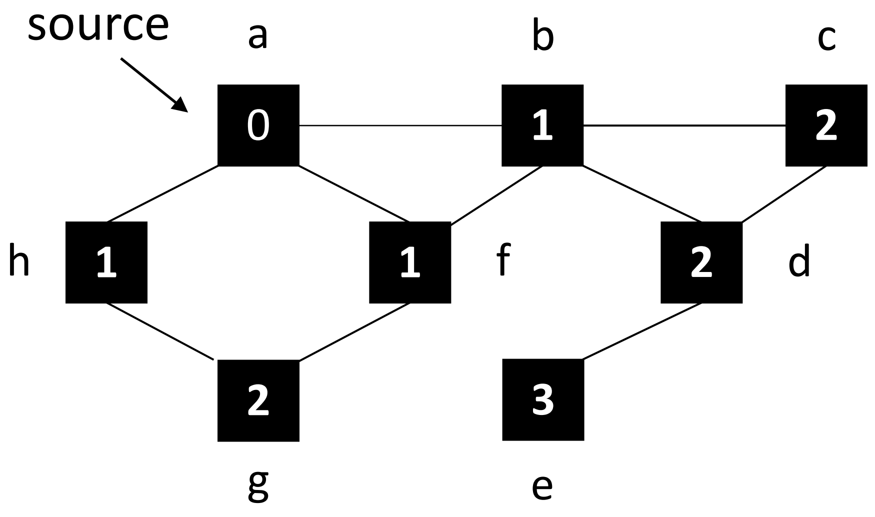
- Exemple



i	a	b	c	d	e	f	g	h
$d[i]$	0	1	2	2	3	1	2	1
$p[i]$	$\emptyset$	a	b	b	d	a	f	a

Parcours en largeur de $G = (S, A)$

- Exemple



i	a	b	c	d	e	f	g	h
$d[i]$	0	1	2	2	3	1	2	1
$p[i]$	$\varnothing$	a	b	b	d	a	f	a

Arbre du parcours en largeur

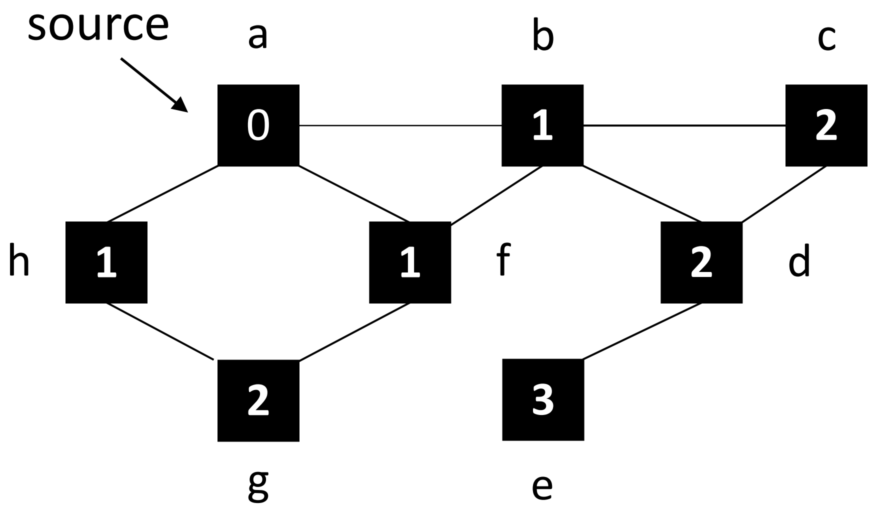
Le sous-graphe des prédécesseurs p de $G = (S, A)$ est $G_p = (S_p, A_p)$ où

$\rightarrow S_p = \{i \in S: p[i] \neq Nul\} \cup \{s\}$

$\rightarrow A_p = \left\{(p[i], i) \in A: i \in S_p - \{s\}\right\}$

Parcours en largeur de $G = (S, A)$

- Exemple



i	a	b	c	d	e	f	g	h
$d[i]$	0	1	2	2	3	1	2	1
$p[i]$	$\varnothing$	a	b	b	d	a	f	a

Arbre du parcours en largeur

Le sous-graphe des prédécesseurs p de $G = (S, A)$ est $G_p = (S_p, A_p)$ où

$\rightarrow S_p = \{i \in S : p[i] \neq Nul\} \cup \{s\}$

$\rightarrow A_p = \left\{(p[i], i) \in A : i \in S_p - \{s\}\right\}$

Le sous-graphe G_p est **un arbre en largeur d'abord** si:

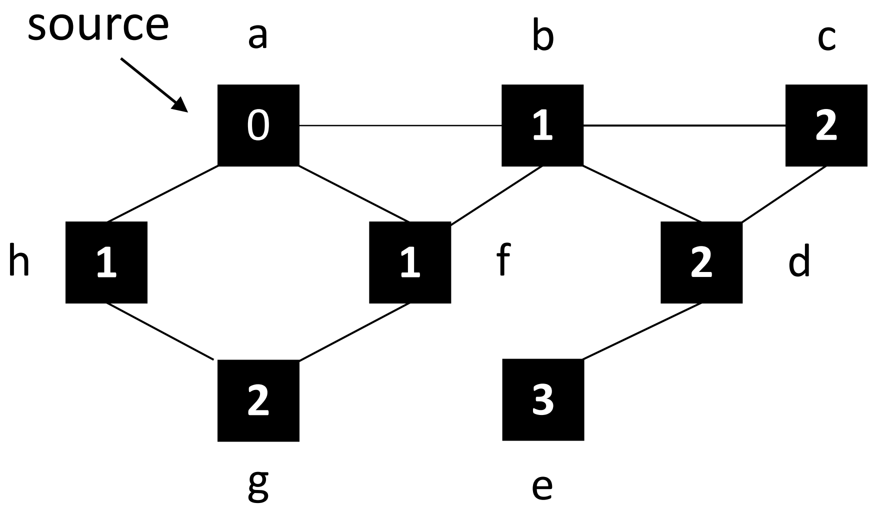
S_p est constitué des sommets accessibles à partir de s et pour tout $i \in S_p$, il existe un chemin simple unique de s vers i dans G_p qui est aussi un chemin le plus court de s vers i dans G

Les arêtes dans A_p sont appelées arêtes d'arbre

On a : $|A_p| = |S_p| - 1$

Parcours en largeur de $G = (S, A)$

- Exemple



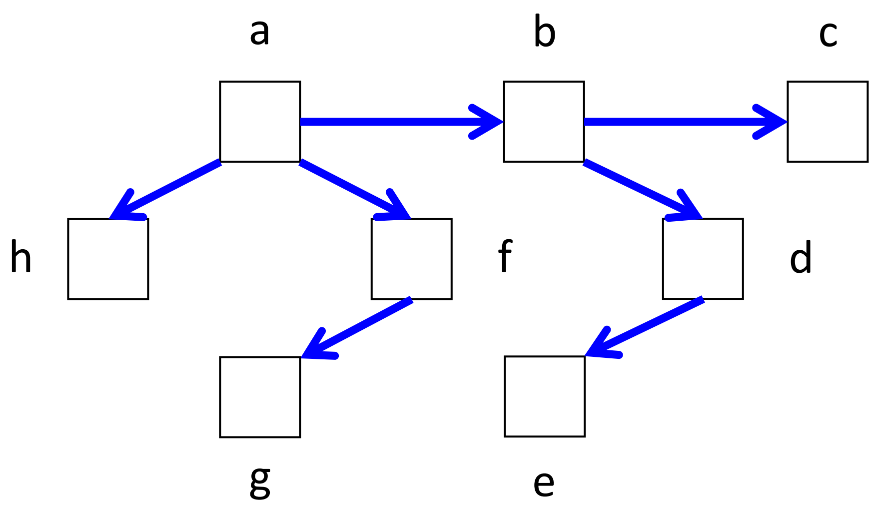
i	a	b	c	d	e	f	g	h
$d[i]$	0	1	2	2	3	1	2	1
$p[i]$	$\varnothing$	a	b	b	d	a	f	a

Arbre du parcours en largeur

Le sous-graphe des prédécesseurs p de $G = (S, A)$ est $G_p = (S_p, A_p)$ où

$\rightarrow S_p = \{i \in S : p[i] \neq Nul\} \cup \{s\}$

$\rightarrow A_p = \left\{(p[i], i) \in A : i \in S_p - \{s\}\right\}$



Complexité du Parcours en largeur

- L'initialisation prend $O(n)$

pour x $\in$ S – {s} faire
  c[x] $\leftarrow$ Blanc ;
  d[x] $\leftarrow + \propto$
  p[x] $\leftarrow$ Nul ;
fin pour
c[s] $\leftarrow$ Gris; d[s] $\leftarrow$ 0; p[s] $\leftarrow$ Nul ;
F $\leftarrow \varnothing$; enfiler(F, s) ;

Complexité du Parcours en largeur

- L'initialisation prend $O(n)$
- Boucle de traversée
  - Chaque sommet est enfilé et défilé au plus une fois
  - Chaque opération prend $O(1)$

  $\rightarrow$ temps total de mise en file d'attente : $O(n)$

```
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
```

Complexité du Parcours en largeur

```
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
```

- L'initialisation prend $O(n)$
- Boucle de traversée
  - Chaque sommet est enfilé et défilé au plus une fois
  - Chaque opération prend $O(1)$

  $\rightarrow$ temps total de mise en file d'attente : $O(n)$
  - La liste d'adjacence de chaque sommet est explorée au plus une fois

  $\rightarrow$ Somme des longueurs de toutes les listes d'adjacences : $O(m)$

Complexité du Parcours en largeur

```
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
```

- L'initialisation prend $O(n)$
- Boucle de traversée
  - Chaque sommet est enfilé et défilé au plus une fois
  - Chaque opération prend $O(1)$

  $\rightarrow$ temps total de mise en file d'attente : $O(n)$
  - La liste d'adjacence de chaque sommet est explorée au plus une fois

  $\rightarrow$ Somme des longueurs de toutes les listes d'adjacences : $O(m)$
- Total : $O(n+m)$ pour un graphe avec n sommets et m arêtes représenté par sa liste d'adjacence

Validité du Parcours en largeur

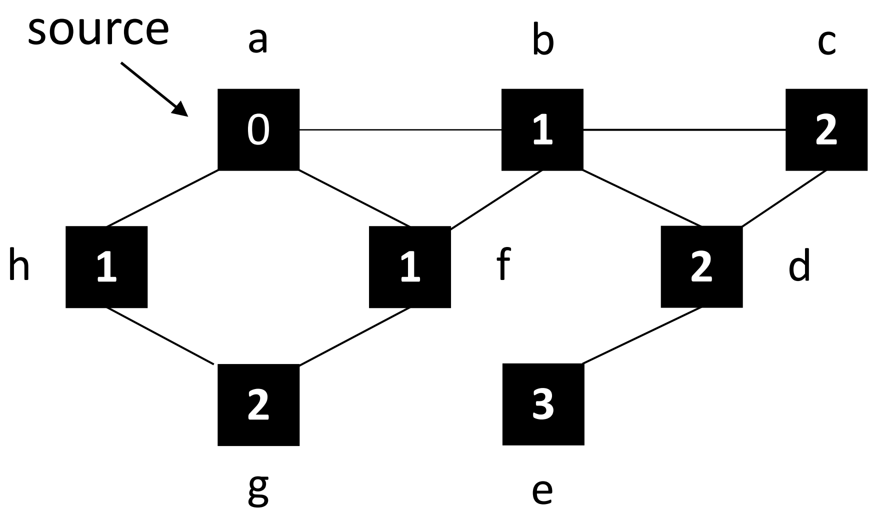
- On peut montrer que
  - PeL(G, s) visite chaque sommet accessible depuis la source s

Validité du Parcours en largeur

- On peut montrer que
  - PeL(G, s) visite chaque sommet accessible depuis la source s
  - A la fin de l'algorithme
    - $d[i] = \delta(s, i)$ pour tout $i \in S$
    - Pour tout $i \neq s$ accessible depuis s, un des plus courts chemins de s à i est le plus court chemin de s à $p[i]$ complété par l'arc / arête $(p[i], i)$

Propriétés du Parcours en largeur

- Si $F = < i_1, i_2, \dots, i_f >$ où $i_1 = \text{tête}(F)$ et $i_f = \text{queue}(F)$, alors
  $d(i_f) \leq d(i_1) + 1$
  $d(i_k) \leq d(i_{k+1})$ pour $k = 1, \dots, f - 1$

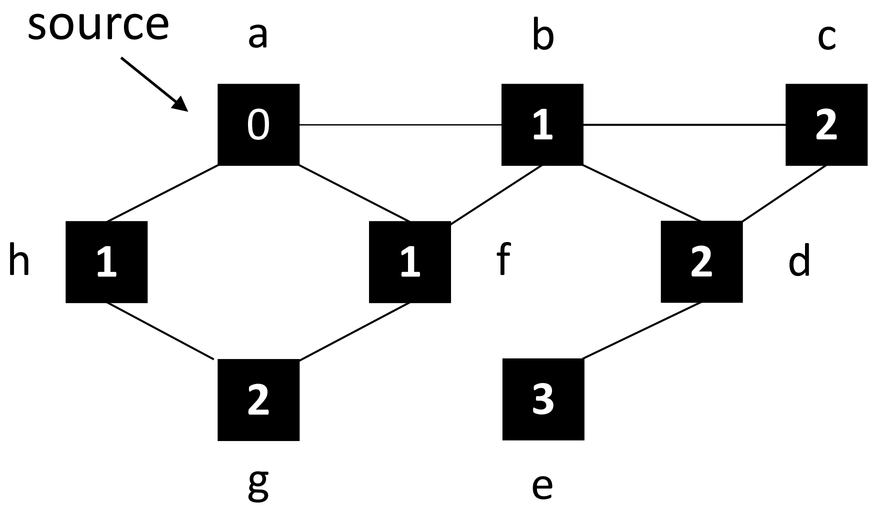


Propriétés du Parcours en largeur

- Si $F = < i_1, i_2, \dots, i_f >$ où $i_1 = \text{tête}(F)$ et $i_f = \text{queue}(F)$, alors

  $d(i_f) \leq d(i_1) + 1$

  $d(i_k) \leq d(i_{k+1})$ pour $k = 1, \dots, f-1$



F = **b f h**
$d[i]$ = 1 1 1
$p[i]$ = a a a

F = **f h c d**
$d[i]$ = 1 1 2 2
$p[i]$ = a a b b

F = **h c d g**
$d[i]$ = 1 2 2 2
$p[i]$ = a b b f

F = **g e**
$d[i]$ = 2 3
$p[i]$ = f d

Propriétés du Parcours en largeur

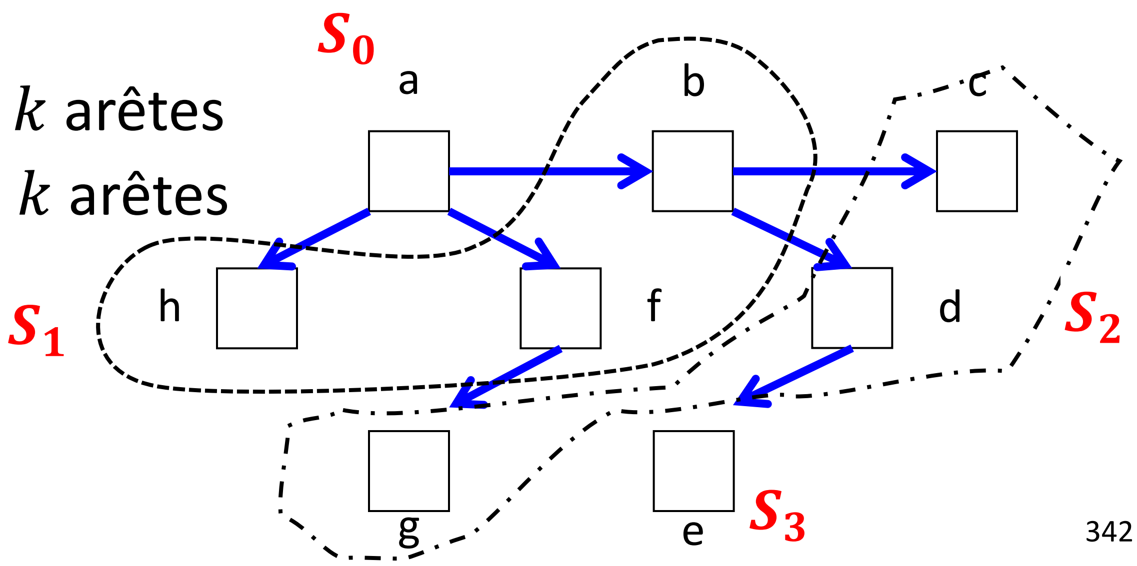
- Soit $G = (S, A)$ un graphe non orienté
- On note G_s la composante connexe contenant s et S_k les sommets à une distance k de s, alors

Propriétés du Parcours en largeur

- Soit $G = (S, A)$ un graphe non orienté
- On note G_s la composante connexe contenant s et S_k les sommets à une distance k de s, alors
  - $PeL(G, s)$ visite tous les sommets et toutes les arêtes de G_s
  - Les arêtes retenues par le parcours forment un arbre couvrant T_s de G_s
  - Pour chaque sommet $i \in S_k$
    - Le chemin de s à i dans T_s a (exactement) k arêtes
    - Chaque chemin de s à i dans G a au moins k arêtes

Propriétés du Parcours en largeur

- Soit $G = (S, A)$ un graphe non orienté
- On note G_s la composante connexe contenant s et S_k les sommets à une distance k de s, alors
  - $PeL(G, s)$ visite tous les sommets et toutes les arêtes de G_s
  - Les arêtes retenues par le parcours forment un arbre couvrant T_s de G_s
  - Pour chaque sommet $i \in S_k$
    - Le chemin de s à i dans T_s a (exactement) k arêtes
    - Chaque chemin de s à i dans G a au moins k arêtes



Parcours en profondeur de $G = (S, A)$

- Principe similaire au PeL (exploration systématique)

  $\rightarrow$ Stratégie d'exploration différente

  - Parcourir l'ensemble des sommets du graphe en privilégiant **la profondeur**

    $\rightarrow$ on va le plus loin possible

Parcours en profondeur de $G = (S, A)$

- Principe similaire au PeL (exploration systématique)
  $\rightarrow$ Stratégie d'exploration différente
  - Parcourir l'ensemble des sommets du graphe en privilégiant **la profondeur**
    $\rightarrow$ on va le plus loin possible
  - Soit i le sommet le plus récemment découvert
    - Explorez toutes les arêtes incidentes à i
    - **Revenir en arrière** et fixer $\boldsymbol{i} \leftarrow$ **p[i]**, puis recommencer

Parcours en profondeur de $G = (S, A)$

- Principe similaire au PeL (exploration systématique)

  $\rightarrow$ Stratégie d'exploration différente

  - Parcourir l'ensemble des sommets du graphe en privilégiant **la profondeur** $\rightarrow$ on va le plus loin possible
  - Soit i le sommet le plus récemment découvert
    - Explorez toutes les arêtes incidentes à i
    - **Revenir en arrière** et fixer $\boldsymbol{i} \leftarrow$ **p[i]**, puis recommencer
  - Continuez jusqu'à ce que tous les sommets accessibles à partir de la source soient découverts
  - S'il reste des sommets non découverts, l'un d'entre eux est choisi comme nouvelle source et la recherche est répétée

Parcours en profondeur de $G = (S, A)$

- En entrée de l'algorithme
  - Le graphe $G = (S, A)$, **orienté ou non**
  - Pas nécessairement besoin d'un sommet source
- En sortie de l'algorithme
  - pour tout $i \in S$
    - $d[i] = \text{date de découverte du sommet } i$ (passage du blanc au gris)
    - $f[i] = \text{date de fin de traitement du sommet } i$ (passage du gris au noir)
    - $p[i] = j$, $i \text{ a été découvert en scannant la liste d'adjacence de } j$
- L'algorithme utilise la même convention des couleurs
- Utilise aussi une variable globale, ***temps***

Parcours en profondeur de $G = (S, A)$

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

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

```
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
```

Initialisation

Traitement

Parcours en profondeur de $G = (S, A)$

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

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

```
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
```

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

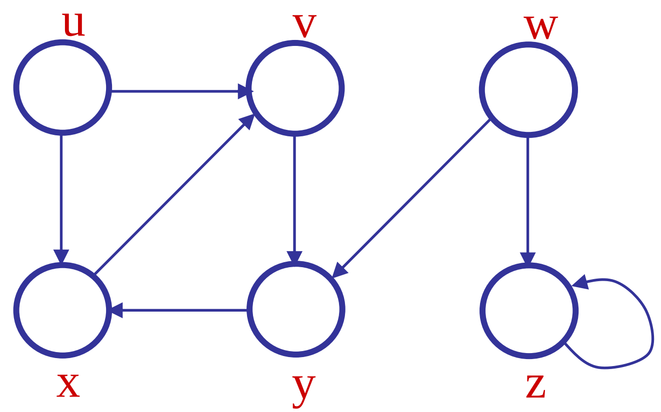
Procédure **Visiter**(x)

Traitement

```
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;
```

Parcours en profondeur de $G = (S, A)$

- Exemple



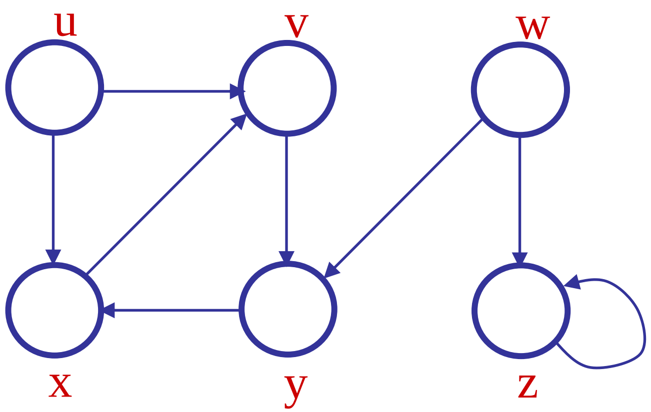
pour x $\in$ S faire
c[x] $\leftarrow$ Blanc ; p[x] $\leftarrow$ Nul ;
fin pour
temps $\leftarrow$ 1;

$temps = 1$

i	u	v	w	x	y	z
$p[i]$			Nul			

Parcours en profondeur de $G = (S, A)$

- Exemple



pour x ∈ S faire
    c[x] ← Blanc ;   p[x] ← Nul ;
fin pour
temps ← 1;

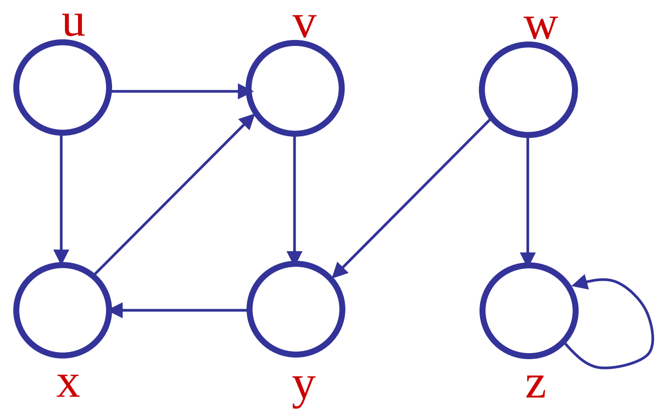
$temps = 1$

i	u	v	w	x	y	z
$p[i]$			Nul			

$\boxed{\text{pour x} \in \text{S faire}}$ ⟶ x = u
    si(c[x] = Blanc) alors
        Visiter(x) ;
    fin si
fin pour

Parcours en profondeur de $G = (S, A)$

- Exemple



pour x ∈ S faire
c[x] ← Blanc ;   p[x] ← Nul ;
fin pour
temps ← 1;

$temps = 1$

i	u	v	w	x	y	z
$p[i]$			Nul			

$\boxed{\text{pour x} \in \text{S faire}}$ ⟶ x = u
si(c[x] = Blanc) alors
Visiter(x) ;
fin si
fin pour

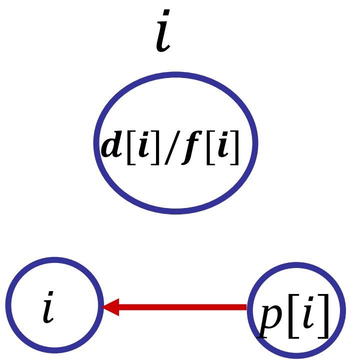
↓

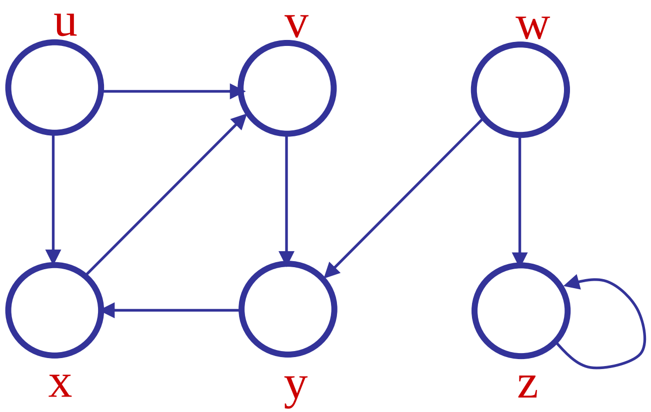
Visiter(u)

Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(u)





$temps = 1$

i	u	v	w	x	y	z
$p[i]$			Nul			

Procédure **Visiter**(x)

c[x] ← Gris ;

d[x] ← temps ; temps ← temps + 1;

pour y ∈ $Adj[\mathrm{x}]$ faire

si (c[y] = Blanc) alors

p[y] ← x;

Visiter(y);

fin si

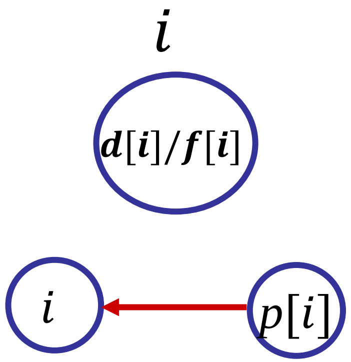
fin pour

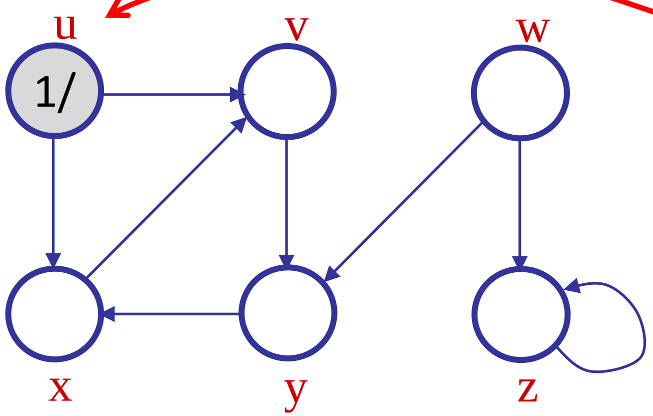
c[x] ← Noir;

f[x] ← temps ; temps ← temps + 1;

Parcours en profondeur de $G = (S, A)$

- Exemple





$temps = 1\ 2$

i	u	v	w	x	y	z
$p[i]$			Nul			

Visiter(u)

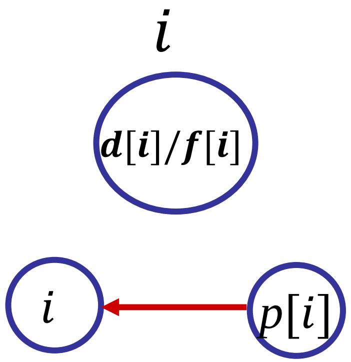
Procédure **Visiter**(x)

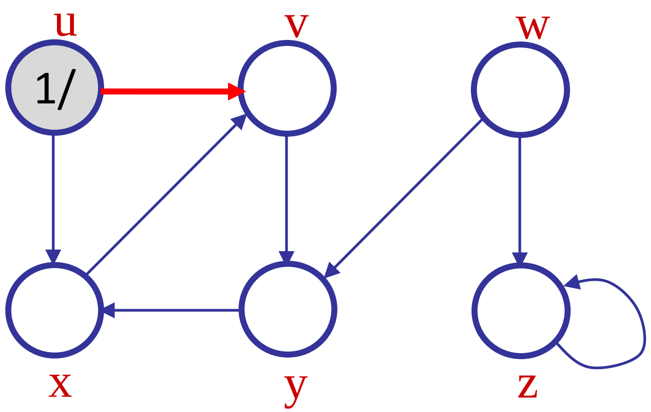
c[x] ← Gris ;
d[x] ← temps ;   temps ← temps + 1;
pour y ∈ $Adj[\mathrm{x}]$ faire
    si (c[y] = Blanc) alors
        p[y] ← x;
        Visiter(y);
    fin si
fin pour
c[x] ← Noir;
f[x] ← temps ;   temps ← temps + 1;

Parcours en profondeur de $G = (S, A)$

- Exemple

$\boxed{\text{Visiter(u)}}$





$temps = 2$

i	u	v	w	x	y	z
$p[i]$		u				

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

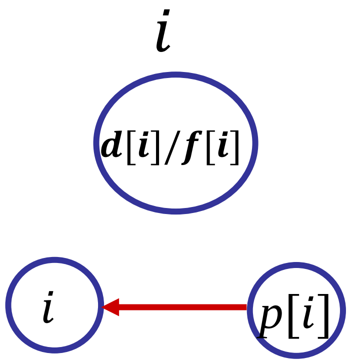
$\boxed{\text{Visiter(v)}}$

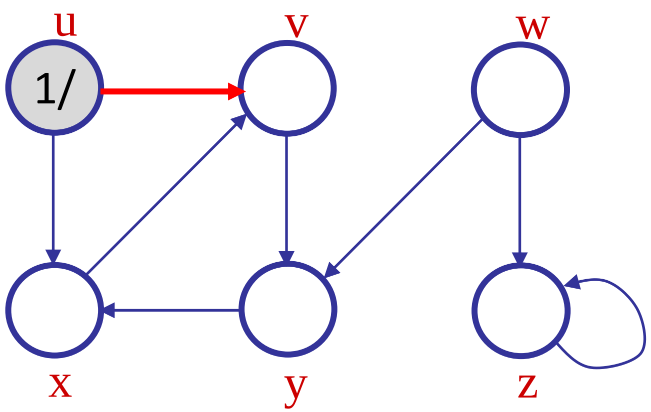
Parcours en profondeur de $G = (S, A)$

Visiter(u)

$\boxed{\text{Visiter(v)}}$

- Exemple





$temps = 2$

i	u	v	w	x	y	z
$p[i]$		u				

Procédure **Visiter**(x)

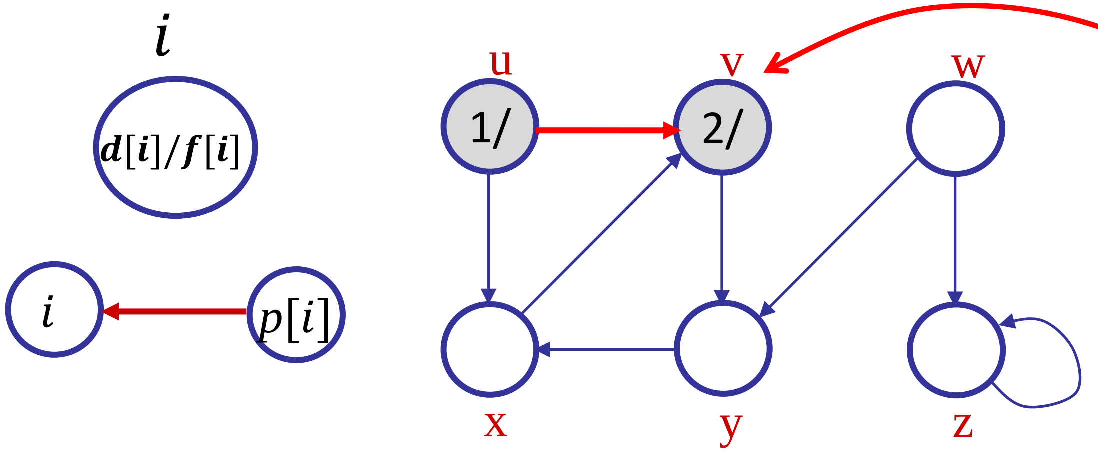
c[x] ← Gris ;
d[x] ← temps ; temps ← temps + 1;
pour y ∈ Adj[x] faire
si (c[y] = Blanc) alors
p[y] ← x;
Visiter(y);
fin si
fin pour
c[x] ← Noir;
f[x] ← temps ; temps ← temps + 1;

Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(u)

Visiter(v)



$temps = 2\ \ 3$

i	u	v	w	x	y	z
$p[i]$		u				

Procédure **Visiter**(x)

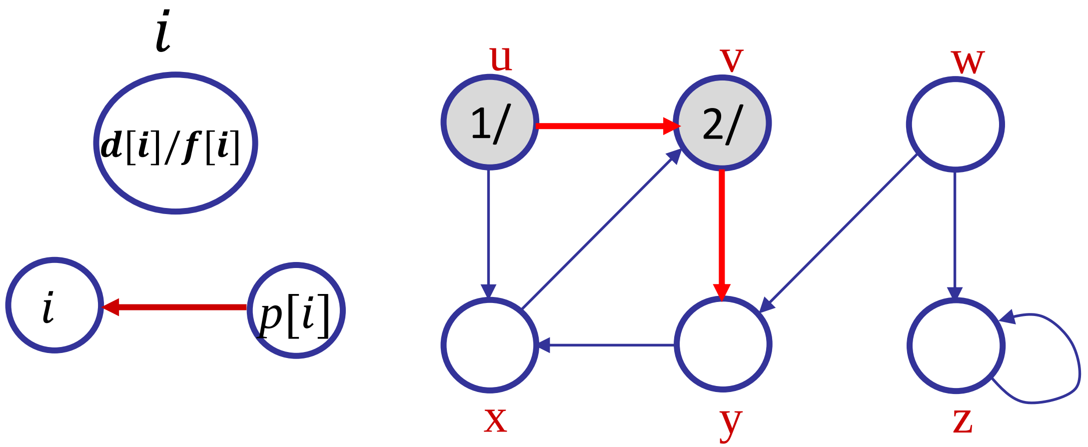
c[x] ← Gris ;
d[x] ← temps ; temps ← temps + 1;
pour y ∈ Adj[x] faire
si (c[y] = Blanc) alors
p[y] ← x;
Visiter(y);
fin si
fin pour
c[x] ← Noir;
f[x] ← temps ; temps ← temps + 1;

Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(u)

$\boxed{\text{Visiter(v)}}$



$temps = 3$

i	u	v	w	x	y	z
$p[i]$		u			v	

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;

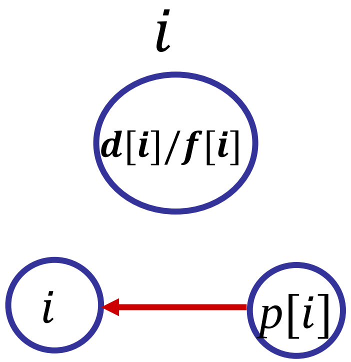
$\boxed{\text{Visiter(y)}}$

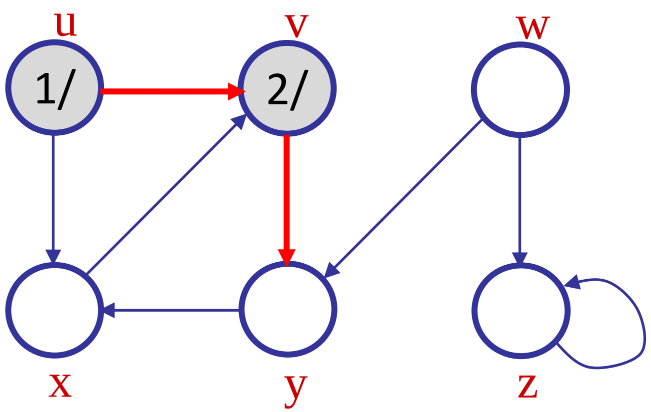
Parcours en profondeur de $G = (S, A)$

Visiter(u) Visiter(v)

$\boxed{\text{Visiter(y)}}$

- Exemple





$temps = 3$

i	u	v	w	x	y	z
$p[i]$		u			v	

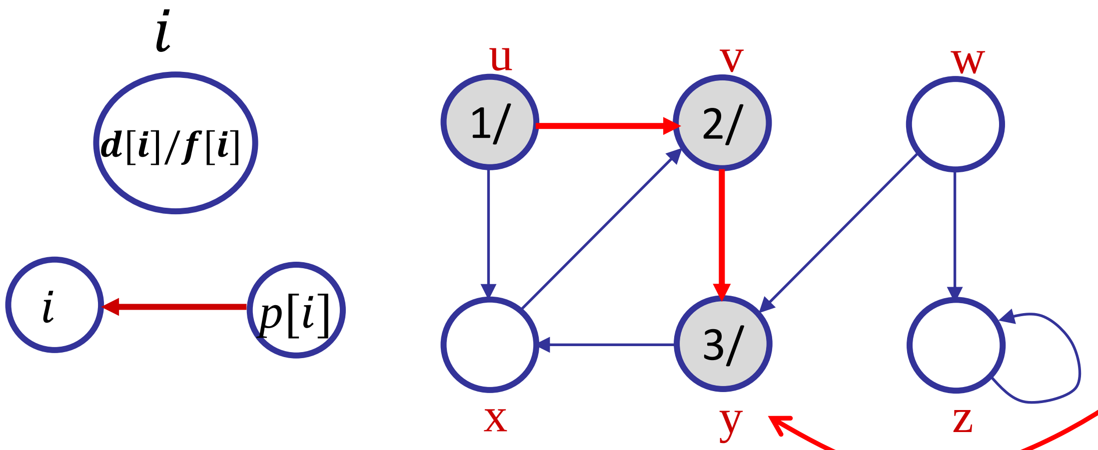
Procédure **Visiter**(x)
  c[x] ← Gris ;
  d[x] ← temps ;   temps ← temps + 1;
  pour y ∈ Adj[x] faire
    si (c[y] = Blanc) alors
      p[y] ← x;
      Visiter(y);
    fin si
  fin pour
  c[x] ← Noir;
  f[x] ← temps ;   temps ← temps + 1;

Parcours en profondeur de $G = (S, A)$

Visiter(u) Visiter(v)

$\boxed{\text{Visiter(y)}}$

- Exemple



$temps = 3 \;\; 4$

i	u	v	w	x	y	z
$p[i]$		u			v	

Procédure **Visiter**(x)

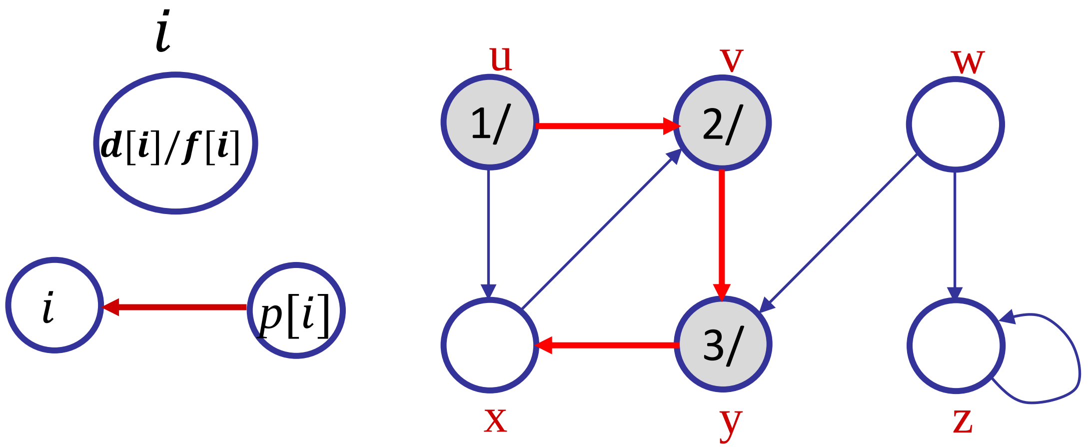
c[x] ← Gris ;
d[x] ← temps ; temps ← temps + 1;
pour y ∈ $Adj[\text{x}]$ faire
  si (c[y] = Blanc) alors
    p[y] ← x;
    Visiter(y);
  fin si
fin pour
c[x] ← Noir;
f[x] ← temps ; temps ← temps + 1;

Parcours en profondeur de $G = (S, A)$

Visiter(u) Visiter(v)

$\boxed{\text{Visiter(y)}}$

- Exemple



$temps = 4$

i	u	v	w	x	y	z
$p[i]$		u		y	v	

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;

$\boxed{\text{Visiter(x)}}$

Parcours en profondeur de $G = (S, A)$

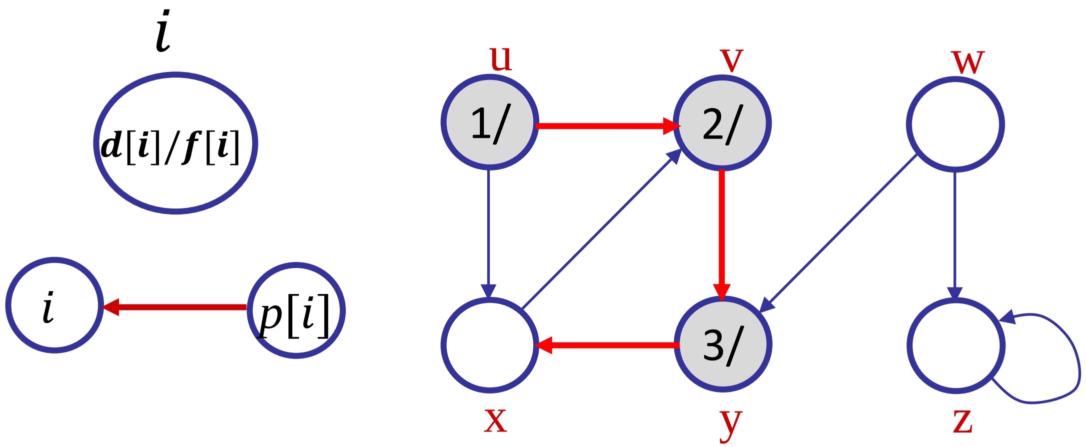
Visiter(u)

Visiter(v)

Visiter(y)

$\boxed{\text{Visiter(x)}}$

- Exemple



$temps = 4$

i	u	v	w	x	y	z
$p[i]$		u		y	v	

Procédure **Visiter**(x)

c[x] ← Gris ;

d[x] ← temps ; temps ← temps + 1;

pour y ∈ $Adj[\text{x}]$ faire

si (c[y] = Blanc) alors

p[y] ← x;

Visiter(y);

fin si

fin pour

c[x] ← Noir;

f[x] ← temps ; temps ← temps + 1;

Parcours en profondeur de $G = (S, A)$

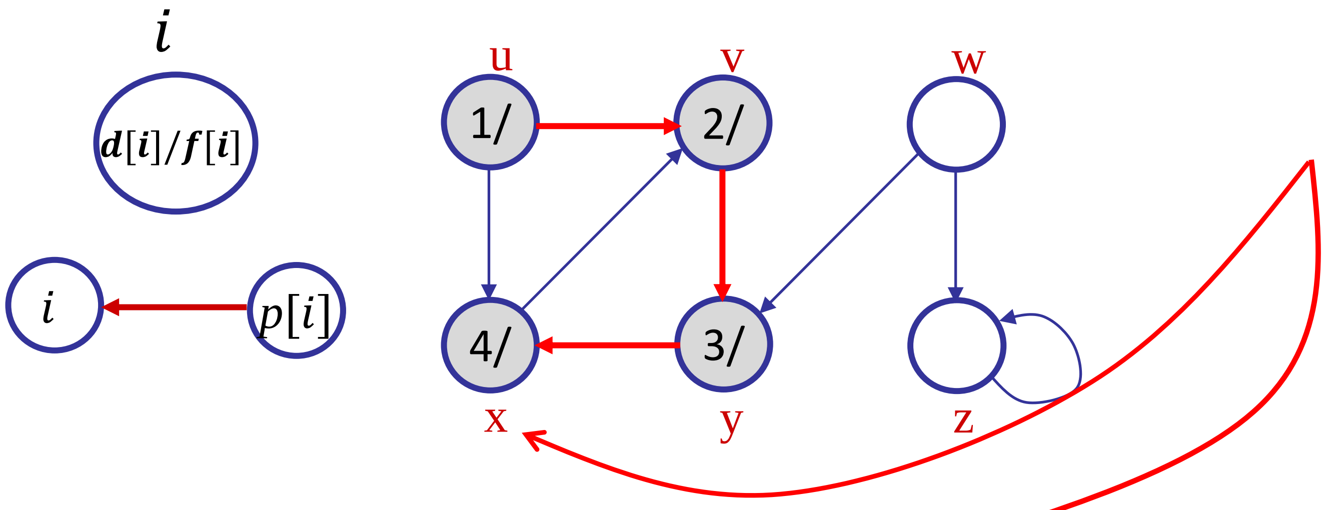
Visiter(u)

Visiter(v)

Visiter(y)

$\boxed{\text{Visiter(x)}}$

- Exemple



$temps = 4\ 5$

i	u	v	w	x	y	z
$p[i]$		u		y	v	

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;

Parcours en profondeur de $G = (S, A)$

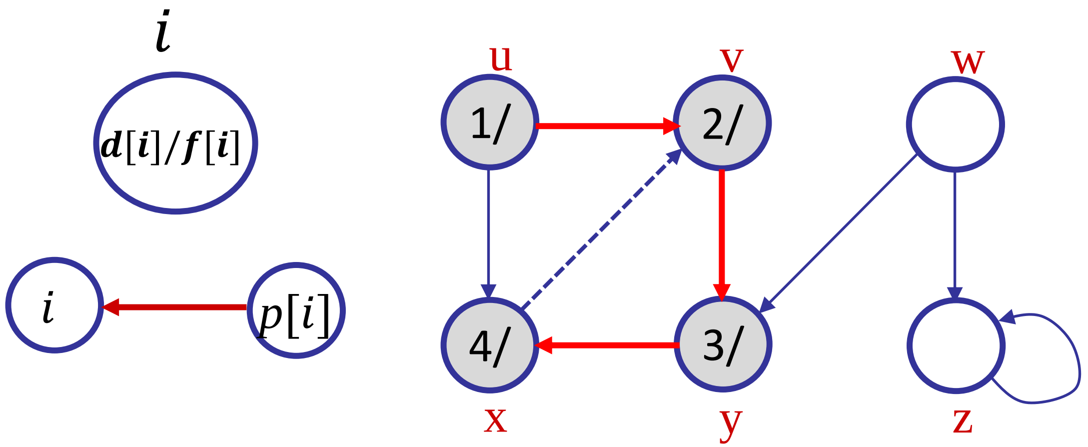
Visiter(u)

Visiter(v)

Visiter(y)

Visiter(x)

- Exemple



$temps = 5$

i	u	v	w	x	y	z
$p[i]$		u		y	v	

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;

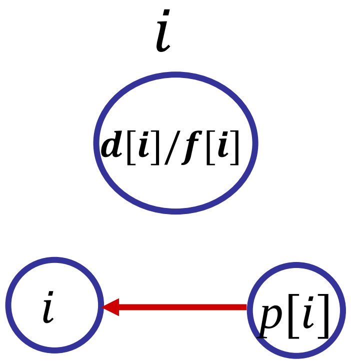
f[x] ← temps ;   temps ← temps + 1;

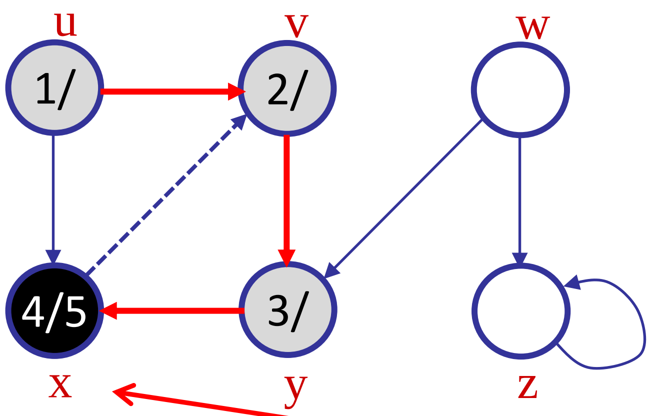
$\varnothing$

Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(u)
Visiter(v)
Visiter(y)
$\boxed{\text{Visiter(x)}}$





$temps = 5\ 6$

i	u	v	w	x	y	z
$p[i]$		u		y	v	

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

Parcours en profondeur de $G = (S, A)$

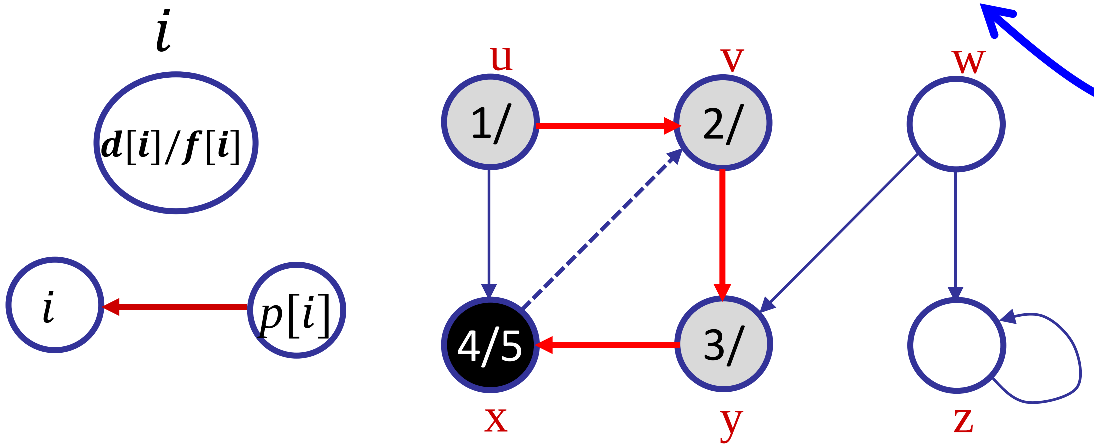
Visiter(u)

Visiter(v)

Visiter(y)

Visiter(x)

- Exemple



$temps = 6$

i	u	v	w	x	y	z
$p[i]$		u		y	v	

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;

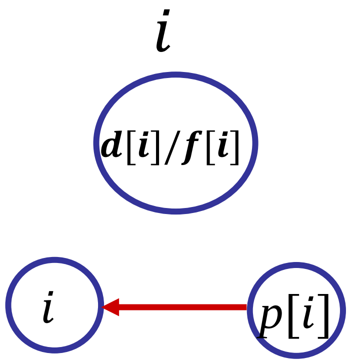
f[x] ← temps ; temps ← temps + 1;

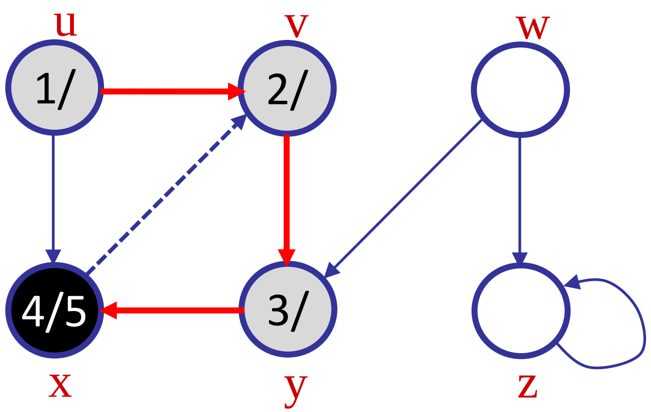
Parcours en profondeur de $G = (S, A)$

Visiter(u)

Visiter(v) $\boxed{\text{Visiter(y)}}$

- Exemple





$temps = 6$

i	u	v	w	x	y	z
$p[i]$		u		y	v	

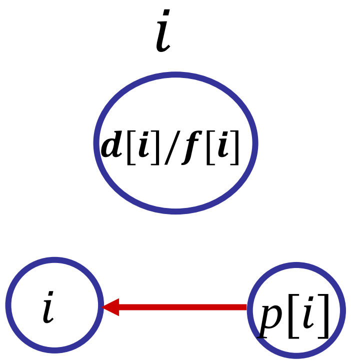
Procédure **Visiter**(x)
- c[x] ← Gris ;
- d[x] ← temps ; temps ← temps + 1;
- pour y ∈ Adj[x] faire
  - si (c[y] = Blanc) alors
    - p[y] ← x;
    - **Visiter**(y);
  - fin si
- fin pour
- c[x] ← Noir;
- f[x] ← temps ; temps ← temps + 1;

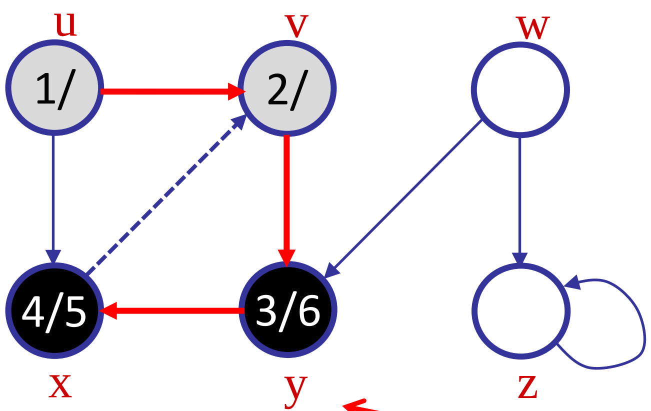
$\boxed{\varnothing}$

Parcours en profondeur de $G = (S, A)$

Visiter(u)
Visiter(v)
$\boxed{\text{Visiter(y)}}$

- Exemple





$temps = 6\ 7$

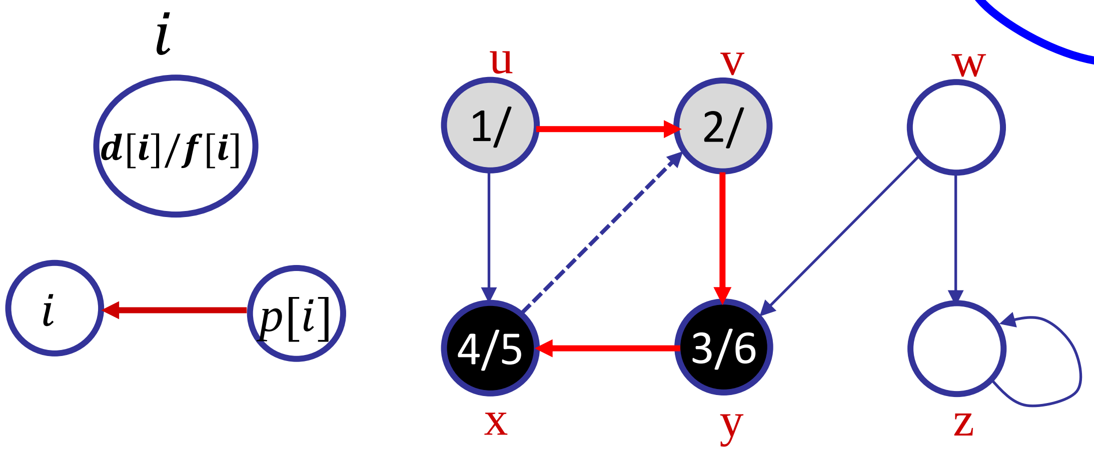
i	u	v	w	x	y	z
$p[i]$		u		y	v	

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

Parcours en profondeur de $G = (S, A)$

Visiter(u)
Visiter(v)
Visiter(y)

- Exemple



$temps = 7$

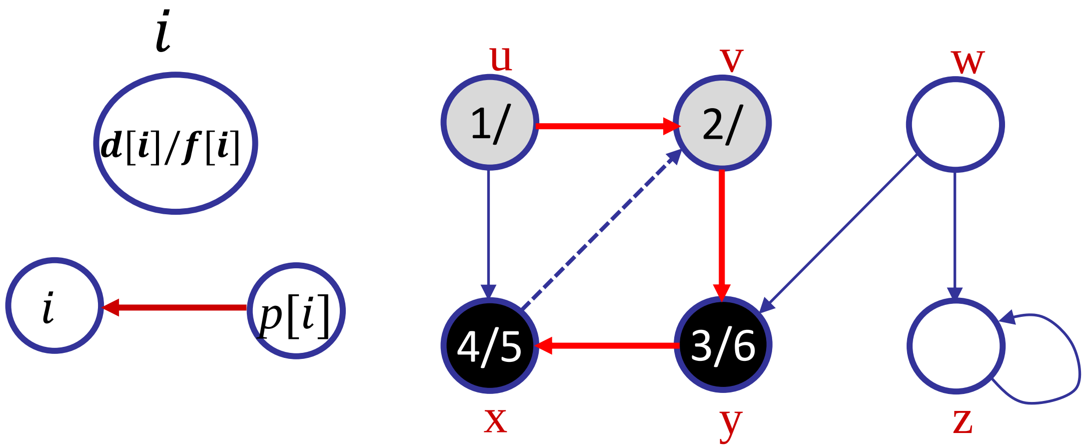
i	u	v	w	x	y	z
$p[i]$		u		y	v	

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

Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(u) | Visiter(v)



$temps = 7$

i	u	v	w	x	y	z
$p[i]$		u		y	v	

Procédure **Visiter**(x)

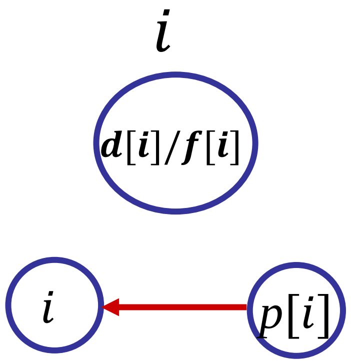
c[x] ← Gris ;
d[x] ← temps ; temps ← temps + 1;
pour y ∈ Adj[x] faire
si (c[y] = Blanc) alors
p[y] ← x;
Visiter(y);
fin si
fin pour
c[x] ← Noir;
f[x] ← temps ; temps ← temps + 1;

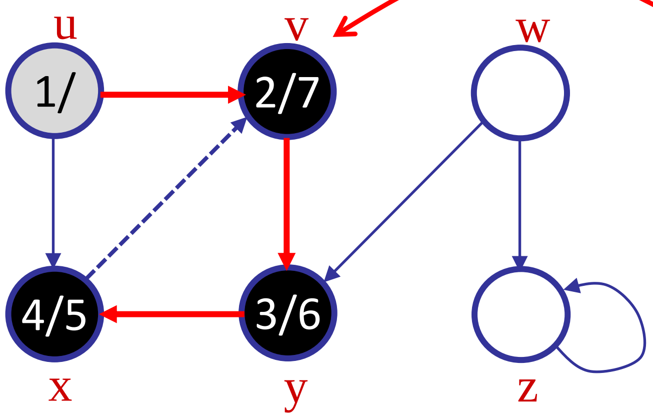
$\boxed{\varnothing}$

Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(u) $\boxed{\text{Visiter(v)}}$





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

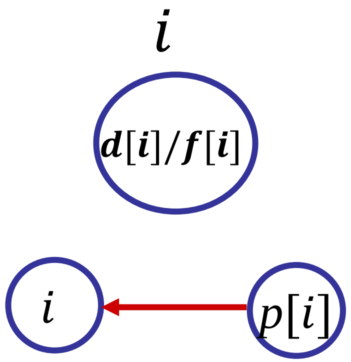
$temps = 7\ \ 8$

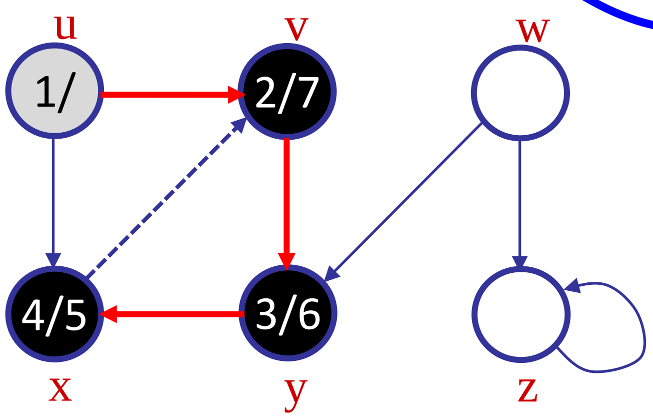
i	u	v	w	x	y	z
$p[i]$		u		y	v	

Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(u) $\boxed{\text{Visiter(v)}}$





$temps = 8$

i	u	v	w	x	y	z
$p[i]$		u		y	v	

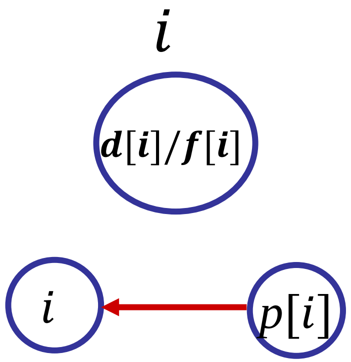
Procédure **Visiter**(x)

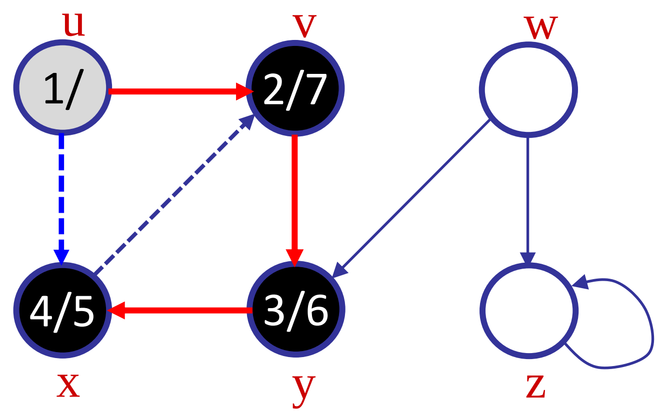
c[x] ← Gris ;
d[x] ← temps ; temps ← temps + 1;
pour y ∈ Adj[x] faire
si (c[y] = Blanc) alors
p[y] ← x;
Visiter(y);
fin si
fin pour
c[x] ← Noir;
f[x] ← temps ; temps ← temps + 1;

Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(u)





$temps = 8$

i	u	v	w	x	y	z
$p[i]$		u		y	v	

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

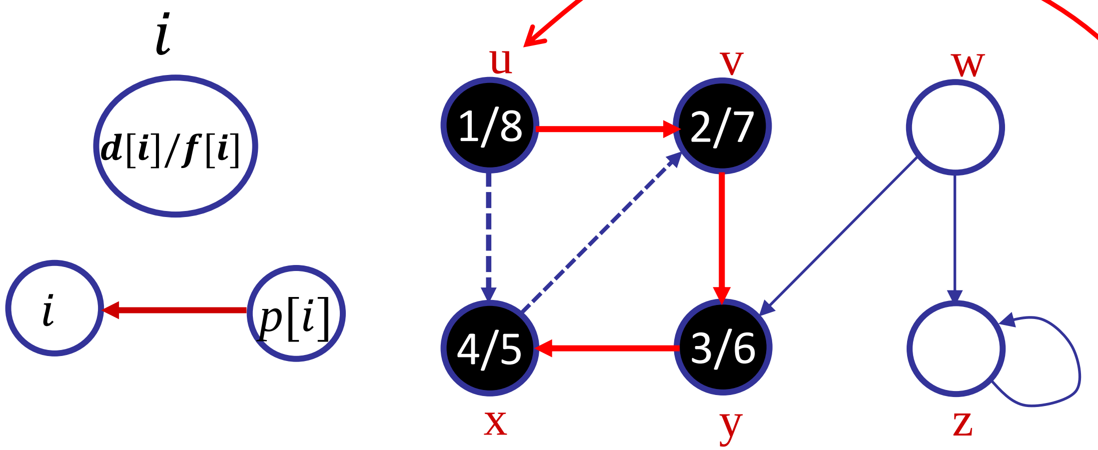
c[x] ← Noir;

f[x] ← temps ; temps ← temps + 1;

$\varnothing$

Parcours en profondeur de $G = (S, A)$

- Exemple



Visiter(u)

$temps = 8\ \ 9$

i	u	v	w	x	y	z
$p[i]$		u		y	v	

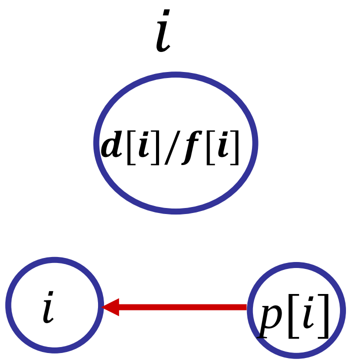
Procédure **Visiter**(x)
    c[x] ← Gris ;
    d[x] ← temps ;   temps ← temps + 1;
    pour y ∈ $Adj[\mathrm{x}]$ faire
        si (c[y] = Blanc) alors
            p[y] ← x;
            Visiter(y);
        fin si
    fin pour
    c[x] ← Noir;
    f[x] ← temps ;   temps ← temps + 1;

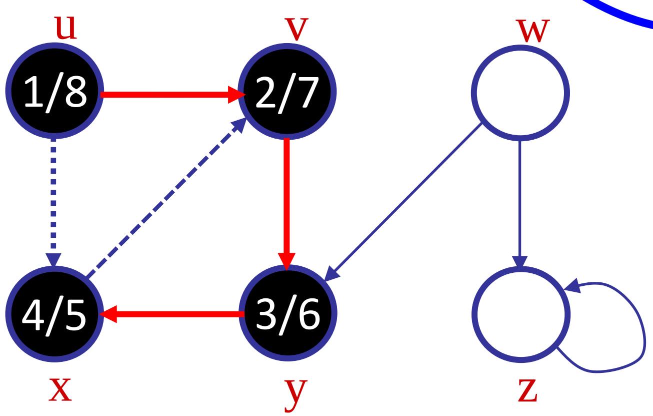
Parcours en profondeur de $G = (S, A)$

- Exemple

$\varnothing$

Visiter(u)





$temps = 9$

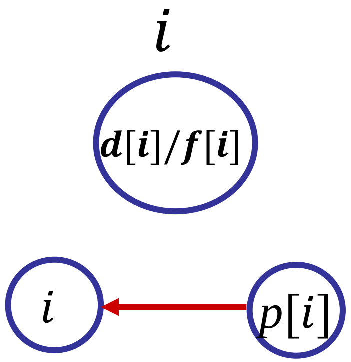
i	u	v	w	x	y	z
$p[i]$		u		y	v	

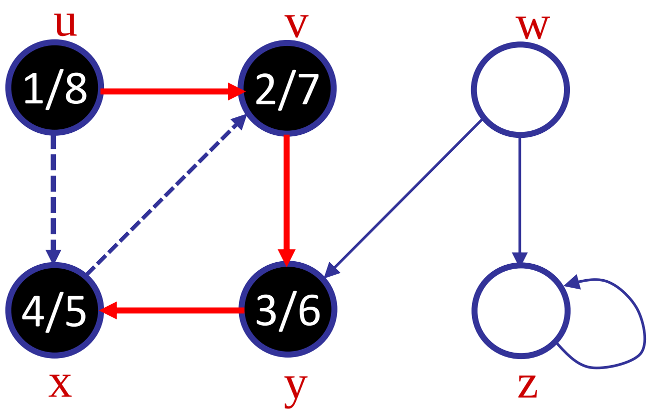
Procédure **Visiter**(x)

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;

Parcours en profondeur de $G = (S, A)$

- Exemple





$temps = 9$

i	u	v	w	x	y	z
$p[i]$		u		y	v	

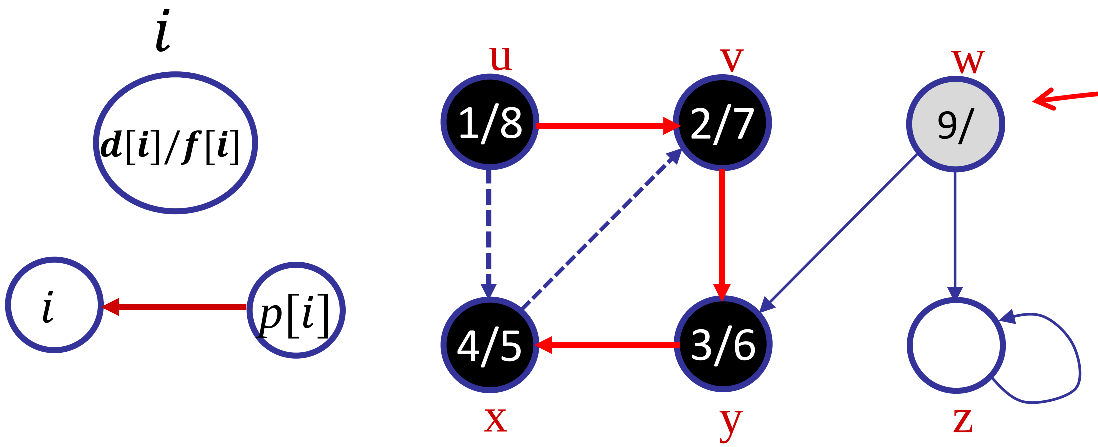
```
pour x ∈ S faire
        c[x] ← Blanc ;    p[x] ← Nul ;
    fin pour
    temps ← 1;
```

```
pour x ∈ S faire                      x = w
        si(c[x] = Blanc) alors
            Visiter(x) ;
        fin si
    fin pour
```

Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(w)



$temps = 9 \;\; 10$

i	u	v	w	x	y	z
$p[i]$		u		y	v	

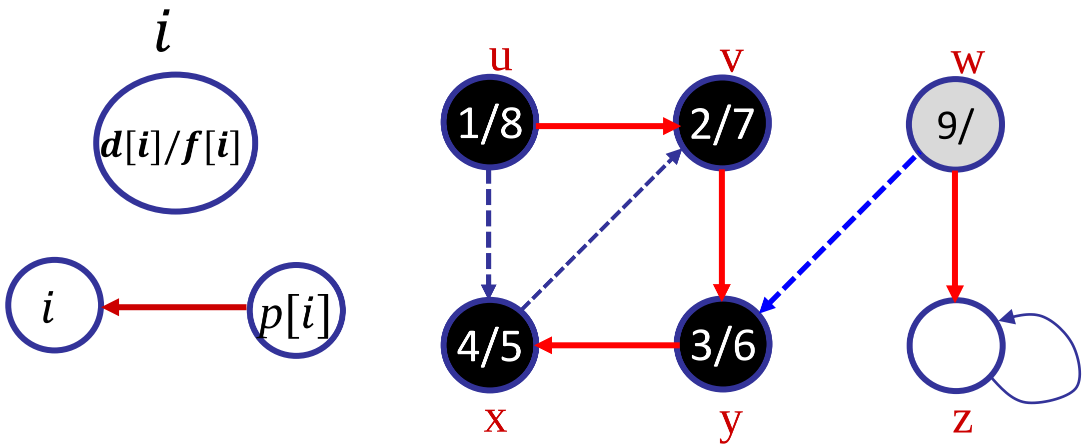
Procédure **Visiter**(x)

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;

Parcours en profondeur de $G = (S, A)$

- Exemple

$\boxed{\text{Visiter(w)}}$



$temps = 10$

i	u	v	w	x	y	z
$p[i]$		u		y	v	w

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;

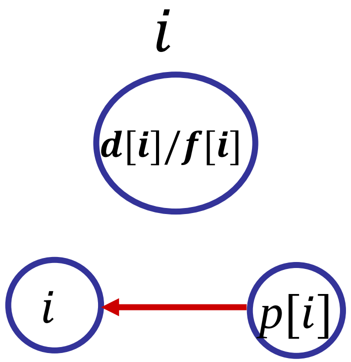
$\boxed{\text{Visiter(z)}}$

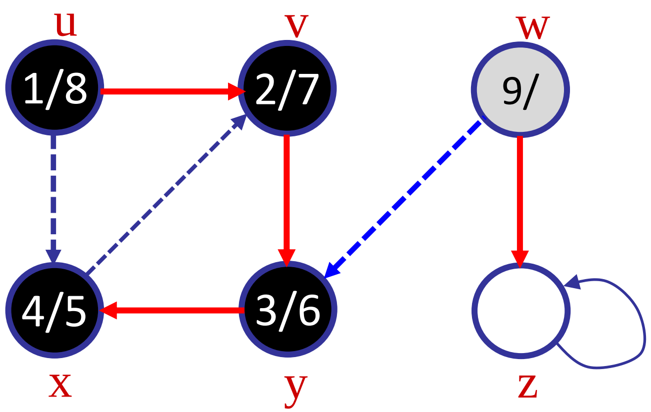
Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(w)

$\boxed{\text{Visiter(z)}}$





$temps = 10$

i	u	v	w	x	y	z
$p[i]$		u		y	v	w

Procédure **Visiter**(x)

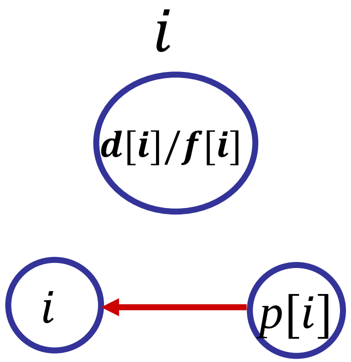
c[x] ← Gris ;
d[x] ← temps ;   temps ← temps + 1;
pour y ∈ $Adj[\mathrm{x}]$ faire
si (c[y] = Blanc) alors
p[y] ← x;
Visiter(y);
fin si
fin pour
c[x] ← Noir;
f[x] ← temps ;   temps ← temps + 1;

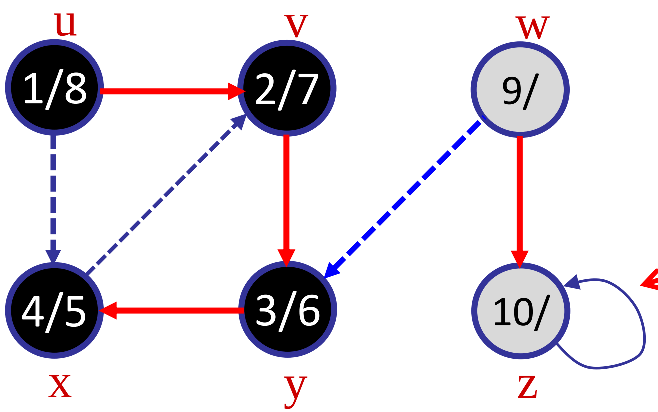
Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(w)

$\boxed{\text{Visiter(z)}}$





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;

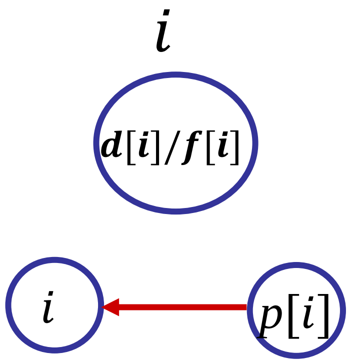
$temps = 10\ \ 11$

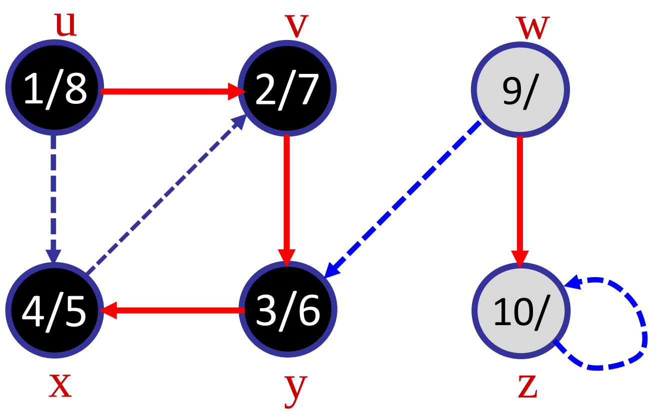
i	u	v	w	x	y	z
$p[i]$		u		y	v	w

Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(w) $\boxed{\text{Visiter(z)}}$





$temps = 11$

i	u	v	w	x	y	z
$p[i]$		u		y	v	w

Procédure **Visiter**(x)

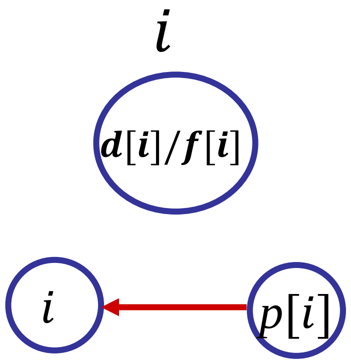
c[x] ← Gris ;
d[x] ← temps ; temps ← temps + 1;
pour y ∈ Adj[x] faire
si (c[y] = Blanc) alors
p[y] ← x;
Visiter(y);
fin si
fin pour
c[x] ← Noir;
f[x] ← temps ; temps ← temps + 1;

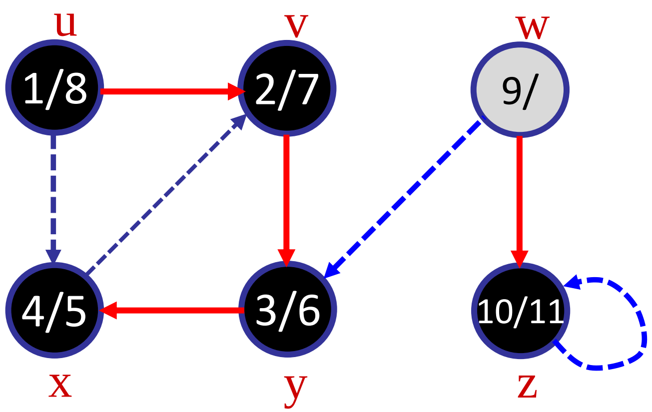
$\boxed{\emptyset}$

Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(w) $\boxed{\text{Visiter(z)}}$





$temps = 11\ \ 12$

i	u	v	w	x	y	z
$p[i]$		u		y	v	w

Procédure **Visiter**(x)

c[x] ← Gris ;

d[x] ← temps ; temps ← temps + 1;

pour y ∈ $Adj[\mathrm{x}]$ faire

si (c[y] = Blanc) alors

p[y] ← x;

Visiter(y);

fin si

fin pour

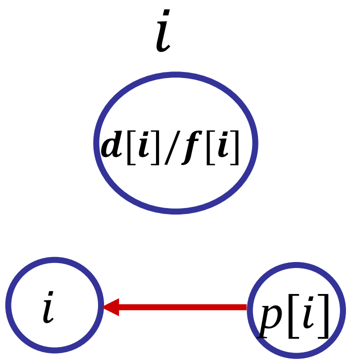
c[x] ← Noir;

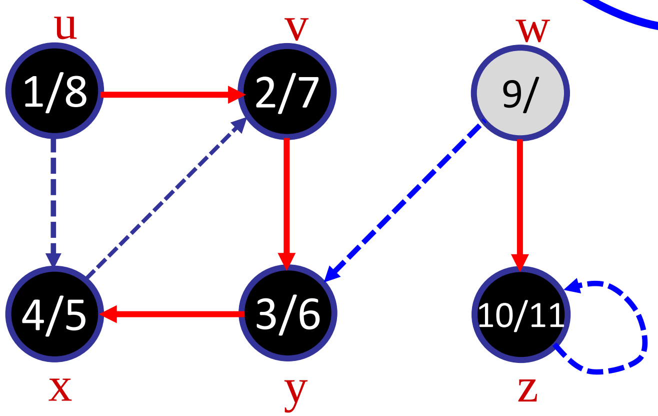
f[x] ← temps ; temps ← temps + 1;

Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(w) $\boxed{\text{Visiter(z)}}$





$temps = 12$

i	u	v	w	x	y	z
$p[i]$		u		y	v	w

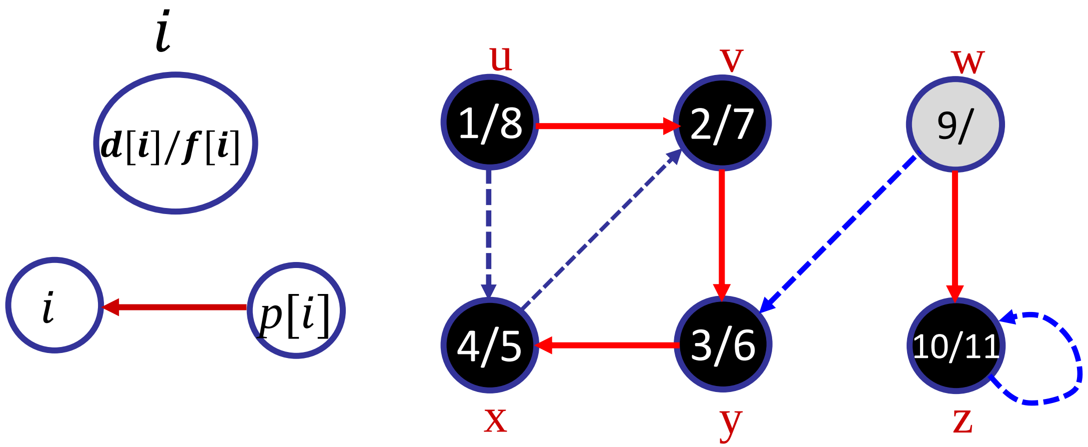
Procédure **Visiter**(x)

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;

Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(w)



$temps = 12$

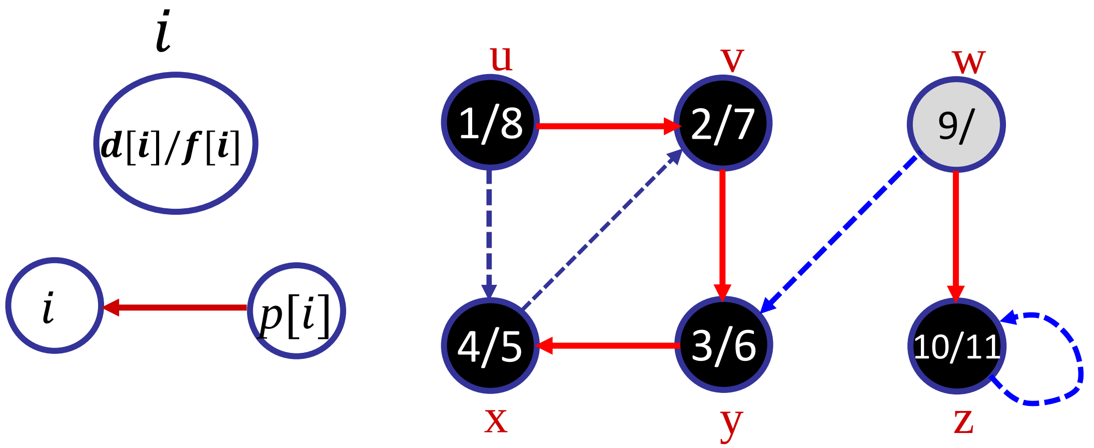
i	u	v	w	x	y	z
$p[i]$		u		y	v	w

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

Parcours en profondeur de $G = (S, A)$

- Exemple

Visiter(w)



$temps = 12$

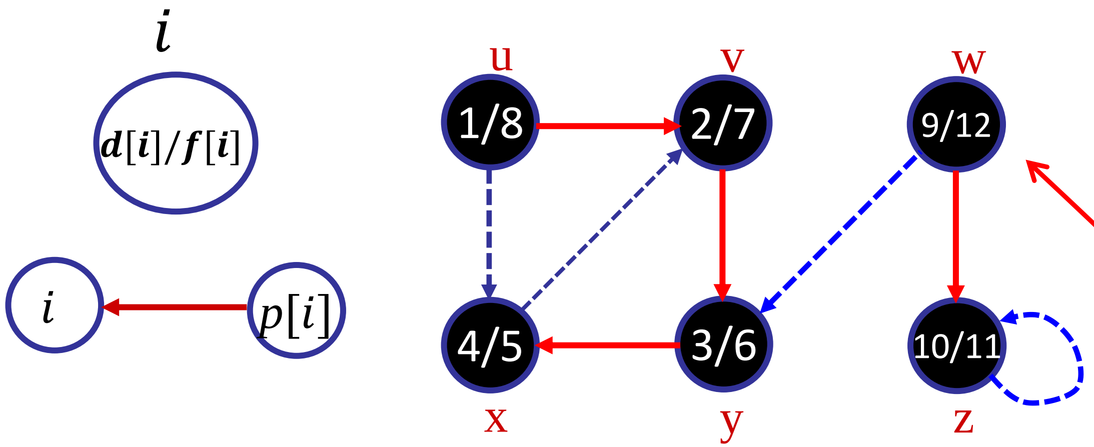
i	u	v	w	x	y	z
$p[i]$		u		y	v	w

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;

$\emptyset$

Parcours en profondeur de $G = (S, A)$

- Exemple



Visiter(w)

$temps = 12 \;\; 13$

i	u	v	w	x	y	z
$p[i]$		u		y	v	w

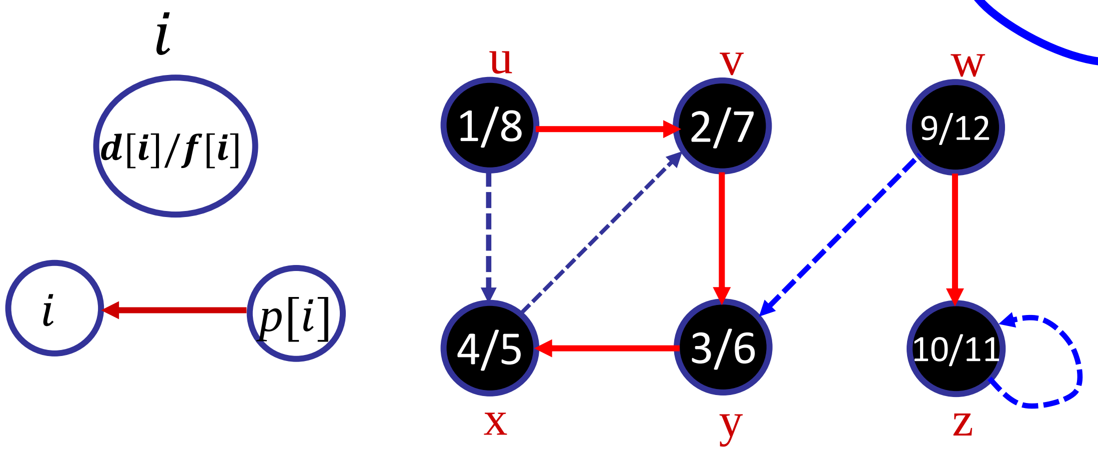
Procédure **Visiter**(x)
  c[x] ← Gris ;
  d[x] ← temps ; temps ← temps + 1;
  pour y ∈ Adj[x] faire
    si (c[y] = Blanc) alors
      p[y] ← x;
      Visiter(y);
    fin si
  fin pour
  c[x] ← Noir;
  f[x] ← temps ; temps ← temps + 1;

Parcours en profondeur de $G = (S, A)$

- Exemple

$\varnothing$

Visiter(w)



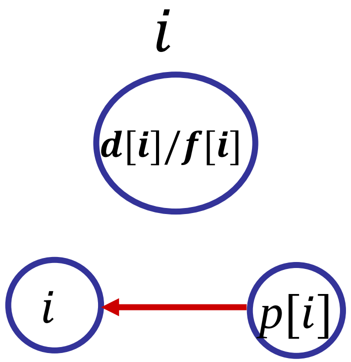
$temps = 13$

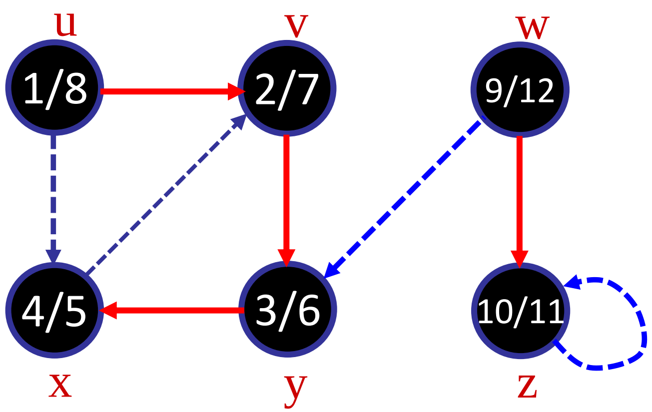
i	u	v	w	x	y	z
$p[i]$		u		y	v	w

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;

Parcours en profondeur de $G = (S, A)$

- Exemple





$temps = 13$

i	u	v	w	x	y	z
$p[i]$		u		y	v	w

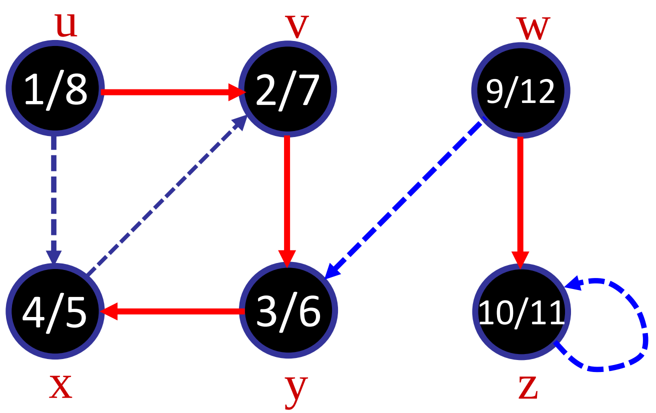
pour x ∈ S faire
c[x] ← Blanc ; p[x] ← Nul ;
fin pour
temps ← 1;

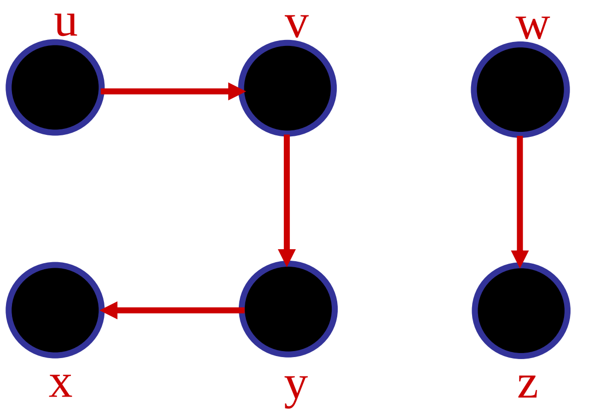
$\boxed{\text{pour x} \in \text{S faire}}$ → ∅
si(c[x] = Blanc) alors
Visiter(x) ;
fin si
fin pour

Stop

Parcours en profondeur de $G = (S, A)$

- Exemple





Forêt du parcours en profondeur

Le sous-graphe des prédécesseurs p de $G = (S, A)$ est $G_p = (S, A_p)$ où

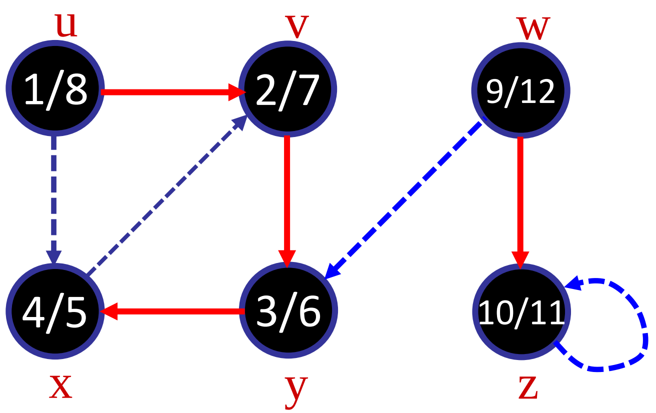
$\rightarrow A_p = \{(p[i], i) \in A : i \in S, p[i] \neq Nul\}$

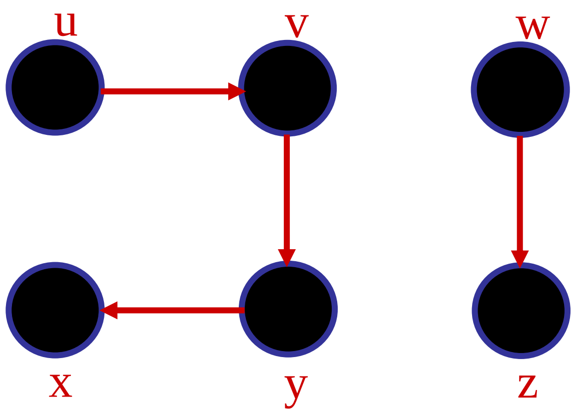
Le sous-graphe G_p est **un ensemble d'arbres**

Les arêtes dans A_p sont appelées arêtes d'arbre

Parcours en profondeur de $G = (S, A)$

- Exemple





Forêt du parcours en profondeur

Le sous-graphe des prédécesseurs p de $G = (S, A)$ est $G_p = (S, A_p)$ où

$$\rightarrow A_p = \{(p[i], i) \in A : i \in S, p[i] \neq Nul\}$$

Le sous-graphe G_p est **un ensemble d'arbres**

Les arêtes dans A_p sont appelées arêtes d'arbre

Remarque : la forêt peut être constituée d'un seul arbre (!)

Parcours en profondeur de $G = (S, A)$

- Terminaison
  - On peut montrer que le parcours en profondeur se termine
  - La terminaison provient du fait qu'on appelle la fonction **Visiter** au plus $|S|$ fois et que chaque fonction Visiter s'appelle elle-même au plus $|S|$ fois.

Parcours en profondeur de $G = (S, A)$

- Terminaison
  - On peut montrer que le parcours en profondeur se termine
  - La terminaison provient du fait qu’on appelle la fonction **Visiter** au plus $|S|$ fois et que chaque fonction Visiter s’appelle elle-même au plus $|S|$ fois.
- L’ordre dans lequel les sommets sont examinés influe sur la sortie de l’algorithme
- En pratique, cet ordre nous importe peu car les sorties possibles sont équivalentes

Parcours en profondeur de $G = (S, A)$

- Complexité
  - Le parcours en profondeur se fait en $O(|S| + |A|)$

Parcours en profondeur de $G = (S, A)$

- Complexité
  - Le parcours en profondeur se fait en $O(|S| + |A|)$

- On applique exactement $|S|$ fois la procédure Visiter
  - Visiter est appelée une fois pour chaque sommet $i \in S$ (Blanc) lorsqu'il est colorié en Gris la première fois
  - Pendant tout le parcours, la boucle dans Visiter s'exécute en $\sum_{i \in S} |Adj(i)| = O(|A|)$

Parcours en profondeur de $G = (S, A)$

- **Théorème** des parenthèses
  - Soit l'intervalle $I_s = [\ d[s], f[s]\ ]$ défini pour $s \in S$.

    Alors, pour tout $i, j \in S$, une et une seule des trois conditions suivantes est vérifiée

    1. $I_i \cap I_j = \emptyset$, et ni i ni j n'est un descendant de l'autre
    2. $I_i \subseteq I_j$, et i est un descendant de j
    3. $I_j \subseteq I_i$, et j est un descendant de i

Parcours en profondeur de $G = (S, A)$

- **Théorème** des parenthèses
  - Soit l'intervalle $I_s = [\, d[s], f[s] \,]$ défini pour $s \in S$.

    Alors, pour tout $i, j \in S$, une et une seule des trois conditions suivantes est vérifiée

    1. $I_i \cap I_j = \emptyset$, et ni i ni j n'est un descendant de l'autre
    2. $I_i \subseteq I_j$, et i est un descendant de j
    3. $I_j \subseteq I_i$, et j est un descendant de i
  - Remarques

$$I_i \cap I_j = \emptyset \leftrightarrow \begin{cases} d(i) < f(i) < d(j) < f(j), \text{ou} \\ d(j) < f(j) < d(i) < f(i) \end{cases}$$

$$I_j \subseteq I_i \leftrightarrow d(i) < d(j) < f(j) < f(i)$$

$$I_i \subseteq I_j \leftrightarrow d(j) < d(i) < f(i) < f(j)$$

Parcours en profondeur de $G = (S, A)$

- **Théorème** des parenthèses : **démonstration**
  - Si $d(j) < f(i) \rightarrow j$ descendant de i

    Découverte de j plus récente que $i \rightarrow$ traitement de j se termine avant celui de i, d'où $[d(j), f(j)] \subseteq [d(i), f(i)]$

Parcours en profondeur de $G = (S, A)$

- **Théorème** des parenthèses : **démonstration**
  - Si $d(j) < f(i) \rightarrow j$ descendant de i

    Découverte de j plus récente que $i \rightarrow$ traitement de j se termine avant celui de i, d'où $[d(j), f(j)] \subseteq [d(i), f(i)]$
  - Si $d(i) < f(j)$, on raisonne de manière analogue en inversant les rôles de i et j

Parcours en profondeur de $G = (S, A)$

- **Théorème** des parenthèses : **démonstration**
  - Si $d(j) < f(i) \rightarrow j$ descendant de i

    Découverte de j plus récente que $i \rightarrow$ traitement de j se termine avant celui de i, d'où $[d(j), f(j)] \subseteq [d(i), f(i)]$
  - Si $d(i) < f(j)$, on raisonne de manière analogue en inversant les rôles de i et j
  - Si $f(i) < d(j) \rightarrow d(i) < f(i) < d(j) < f(j)$

    $\rightarrow [d(j), f(j)] \cap [d(i), f(i)] = \emptyset$

  $\rightarrow$ Aucun des deux sommets n'a été découvert pendant que l'autre était gris, donc aucun sommet n'est un descendant de l'autre

Parcours en profondeur de $G = (S, A)$

- **Théorème** des parenthèses : **conséquences**
- Soit l'intervalle $I_s = [d(s), f(s)]$, alors pour tout $i, j \in S$ on a

$$I_i \cap I_j \in \{\emptyset, I_i, I_j\}$$

  - $I_i \cap I_j = I_j \leftrightarrow I_j \subseteq I_i$
  - $I_i \cap I_j = I_i \leftrightarrow I_i \subseteq I_j$

Parcours en profondeur de $G = (S, A)$

- **Théorème** des parenthèses : **conséquences**
- Soit l'intervalle $I_s = [d(s), f(s)]$, alors pour tout $i, j \in S$ on a
$$I_i \cap I_j \in \{\emptyset, I_i, I_j\}$$
  - $I_i \cap I_j = I_j \leftrightarrow I_j \subseteq I_i$
  - $I_i \cap I_j = I_i \leftrightarrow I_i \subseteq I_j$
- Cas impossible
$$I_i \cap I_j \notin \{\emptyset, I_i, I_j\} \leftrightarrow \begin{cases} d(i) < d(j) < f(i) < f(j) \\ d(j) < d(i) < f(j) < f(i) \end{cases}$$
- Un sommet j est un descendant propre de i ssi
$$I_j \subseteq I_i \leftrightarrow d(i) < d(j) < f(j) < f(i)$$

Parcours en profondeur de $G = (S, A)$

- **Théorème** des parenthèses : **comme des parenthèses (!)**

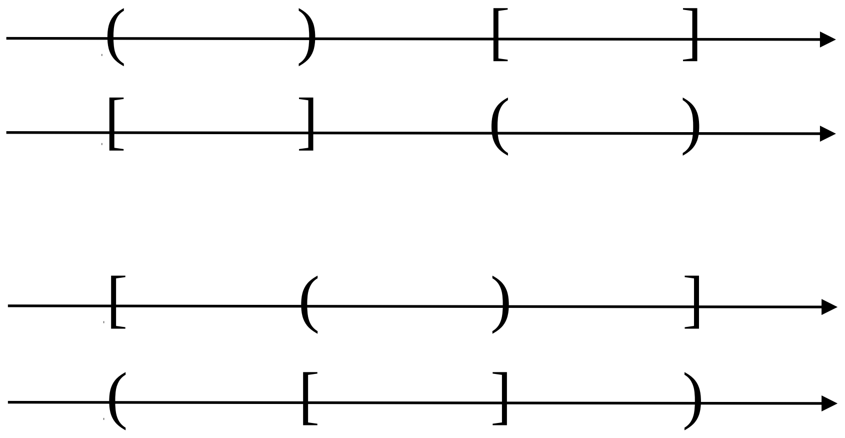
  $d(i) = ($, $f(i) =)$

  $d(j) = [$, $f(j) =]$

Parcours en profondeur de $G = (S, A)$

- **Théorème** des parenthèses : **comme des parenthèses (!)**

  $d(i) = ($, $f(i) =)$
  $d(j) = [$, $f(j) =]$



- Cas possibles (corrects)
  - $I_i \cap I_j = \emptyset \leftrightarrow (\quad)[\quad]$ ou $[\quad](\quad)$
  - $I_i \subseteq I_j \qquad \leftrightarrow \quad [\,(\quad)\,]$
  - $I_j \subseteq I_i \qquad \leftrightarrow \quad (\,[\quad]\,)$

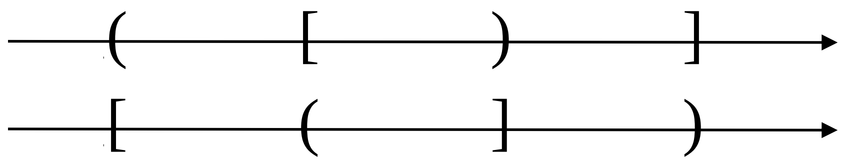
Parcours en profondeur de $G = (S, A)$

- **Théorème** des parenthèses : **comme des parenthèses (!)**

  $d(i) = ($, $f(i) =)$
  $d(j) = [$, $f(j) =]$

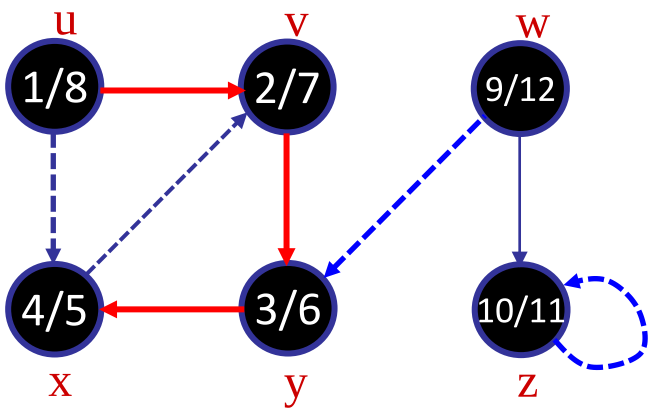
- Cas impossibles (incorrects)

  - $I_i \cap I_j \notin \{\emptyset, I_i, I_j\} \leftrightarrow$ ([ )] [( ])



Parcours en profondeur de $G = (S, A)$

- **Théorème** des parenthèses : **sur l'exemple précédent**



$$(u(v(y(xx)y)v)u)(w(zz)w)$$

Parcours en profondeur de $G = (S, A)$

- Classification des arcs
  - Arcs de liaison ⇔ arcs de la forêt

  $\rightarrow$ (i, j) est un arc de liaison si j a été découvert pendant le parcours de l'arc (i, j)

Parcours en profondeur de $G = (S, A)$

- Classification des arcs
  - Arcs de liaison ⇔ arcs de la forêt

  $\rightarrow$ (i, j) est un arc de liaison si j a été découvert pendant le parcours de l'arc (i, j)
  - Arc arrière ⇔ (i, j) reliant i à un ancêtre j dans un arbre

  $\rightarrow$ remarque : les boucles sont considérées comme des arcs arrière

Parcours en profondeur de $G = (S, A)$

- Classification des arcs
  - Arcs de liaison ⇔ arcs de la forêt

  $\rightarrow$ (i, j) est un arc de liaison si j a été découvert pendant le parcours de l'arc (i, j)
  - Arc arrière ⇔ (i, j) reliant i à un ancêtre j dans un arbre

  $\rightarrow$ remarque : les boucles sont considérées comme des arcs arrière
  - Arc avant ⇔ (i, j) pas de liaison et qui relie i à un descendant j dans un arbre

Parcours en profondeur de $G = (S, A)$

- Classification des arcs
  - Arcs de liaison ⇔ arcs de la forêt

  $\rightarrow$ (i, j) est un arc de liaison si j a été découvert pendant le parcours de l'arc (i, j)
  - Arc arrière ⇔ (i, j) reliant i à un ancêtre j dans un arbre

  $\rightarrow$ remarque : les boucles sont considérées comme des arcs arrière
  - Arc avant ⇔ (i, j) pas de liaison et qui relie i à un descendant j dans un arbre
  - Arc transverse : tous les autres arcs

  $\rightarrow$ ils peuvent relier deux sommets d'un même arbre, du moment que l'un des sommets n'est pas un ancêtre de l'autre

  $\rightarrow$ ils peuvent aussi relier deux sommets appartenant à des arbres différents

Parcours en profondeur de $G = (S, A)$

- Classification des arcs
  - Arcs de liaison ⇔ arcs de la forêt
  - Arc arrière ⇔ (i, j) reliant i à un ancêtre j dans un arbre
  - Arc avant ⇔ (i, j) pas de liaison et qui relie i à un descendant j dans un arbre
  - Arc transverse : tous les autres arcs

- Un arc (i, j) est un
  - Arc arrière ssi $d(j) < d(i) < f(i) < f(j)$

Parcours en profondeur de $G = (S, A)$

- Classification des arcs
  - Arcs de liaison ⇔ arcs de la forêt
  - Arc arrière ⇔ (i, j) reliant i à un ancêtre j dans un arbre
  - Arc avant ⇔ (i, j) pas de liaison et qui relie i à un descendant j dans un arbre
  - Arc transverse : tous les autres arcs

- Un arc (i, j) est un
  - Arc arrière ssi $d(j) < d(i) < f(i) < f(j)$
  - Arc avant ssi $d(i) < d(j) < f(j) < f(i)$

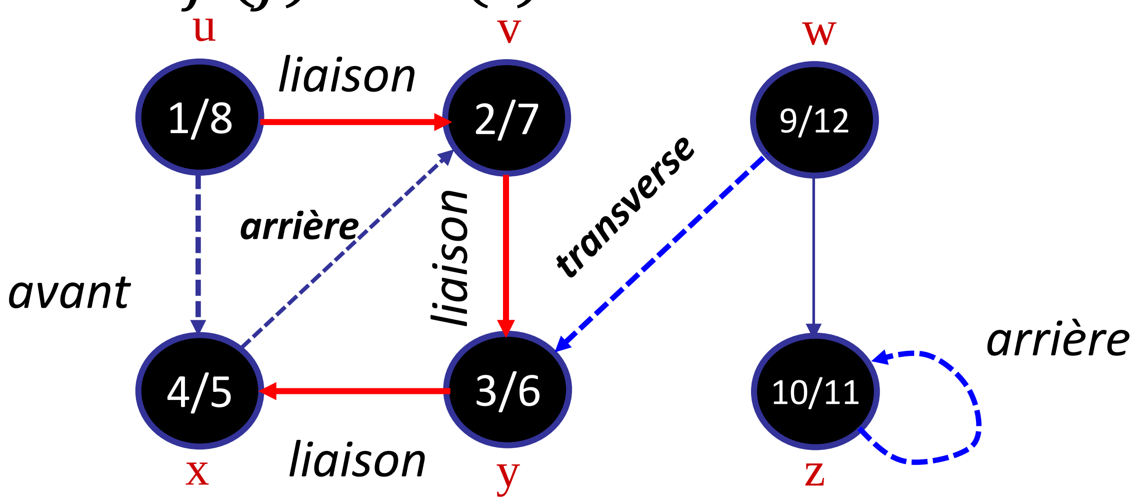
Parcours en profondeur de $G = (S, A)$

- Classification des arcs
  - Arcs de liaison ⇔ arcs de la forêt
  - Arc arrière ⇔ (i, j) reliant i à un ancêtre j dans un arbre
  - Arc avant ⇔ (i, j) pas de liaison et qui relie i à un descendant j dans un arbre
  - Arc transverse : tous les autres arcs

- Un arc (i, j) est un
  - Arc arrière ssi $d(j) < d(i) < f(i) < f(j)$
  - Arc avant ssi $d(i) < d(j) < f(j) < f(i)$
  - Arc de traverse ssi $f(j) < d(i)$

Parcours en profondeur de $G = (S, A)$

- Classification des arcs : **sur l'exemple**
  - Liaison si l'arc est dans la forêt
  - Arc arrière ssi $d(j) < d(i) < f(i) < f(j)$
  - Arc avant ssi $d(i) < d(j) < f(j) < f(i)$
  - Arc de traverse ssi $f(j) < d(i)$

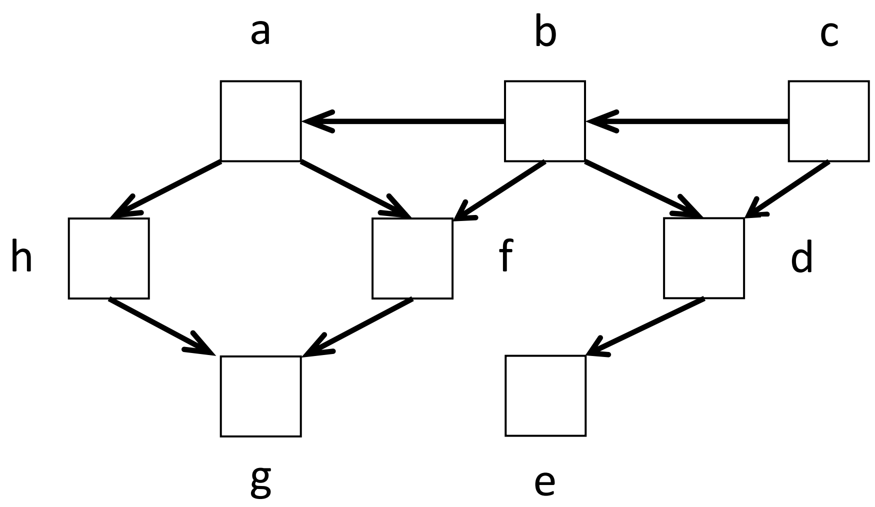


(Exemple d') Application du PeP

- Graphe orienté acyclique

  $\rightarrow$ Tri topologique

  - Applications
    - Problèmes de séquencement
    - Ordonnancement
    - Gestion de projet
    - Planification



- Graphe orienté acyclique
  - Graphe $G = (S, A)$ orienté dans lequel il n'y a pas de circuit

Graphe orienté acyclique

- Soit $G = (S, A)$ un graphe orienté sans circuit. Soit un sommet $i \in S$
  - On dit que i est une **source** si $d^{-}(i) = 0$
  - On dit que i est un **puits** si $d^{+}(i) = 0$
- Propriétés d'un graphe sans circuit
  - G sans circuit ⇔ tout chemin de longueur > 0 est **élémentaire**
  - G sans circuit ⇔ il existe une source s et un puits p dans S
  - G sans circuit ⇔ tout sous-graphe de G est sans circuit

Graphe orienté acyclique

- Soit $G = (S, A)$ un graphe orienté acyclique et soit R la relation d'accessibilité définie pour tous $i, j \in S$ par

$$i \; R \; j \leftrightarrow \text{il existe un chemin } \textbf{de } i \text{ à } j$$

$\rightarrow$ R est une relation d'ordre partielle

- Rappels : une relation d'ordre R est une relation binaire réflexive, antisymétrique et transitive : pour tous $i, j, k \in S$
  - $i \; R \; i$ (réflexivité)
  - ($i \; R \; j$ et $j \; R \; i$) $\Rightarrow i = j$ (antisymétrie)
  - ($i \; R \; j$ et $j \; R \; k$) $\Rightarrow i \; R \; k$ (transitivité)

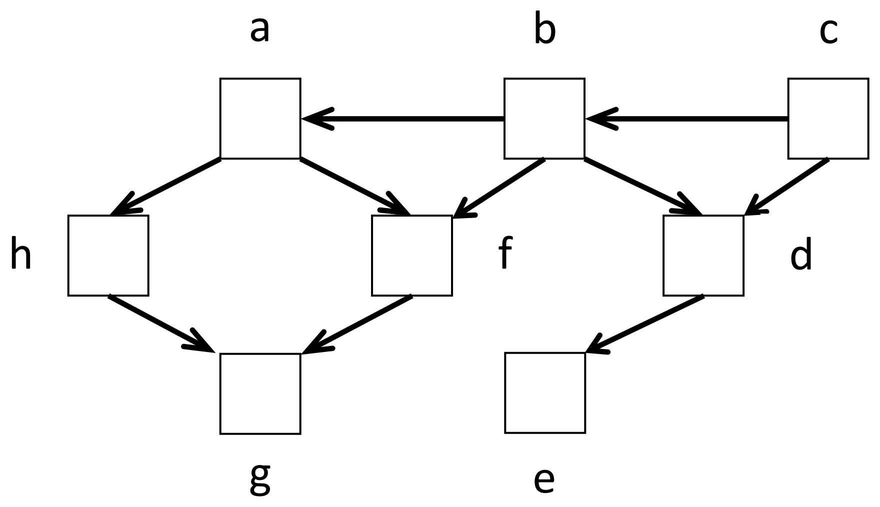
$\rightarrow$ mais on peut avoir i et j tels que $i \not{R} \; j$ **et** $j \not{R} \; i$

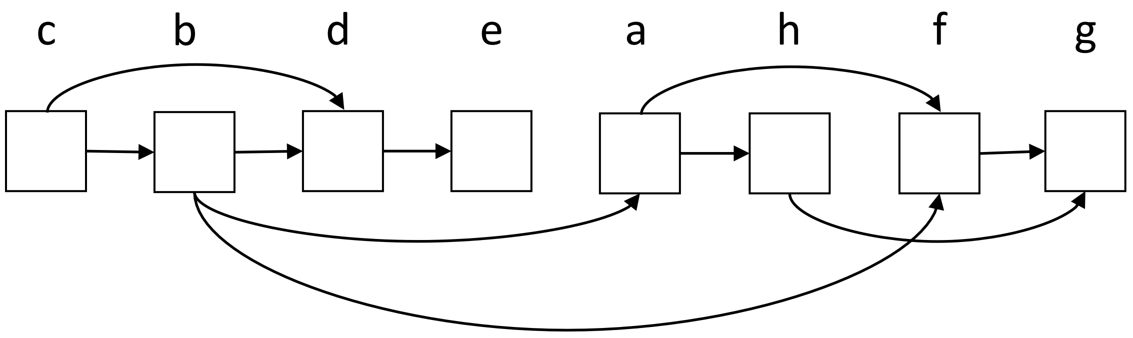
$\rightarrow$ ordre total : soit $i \; R \; j$, soit $j \; R \; i$, pour tous i et j, $i \neq j$

Tri topologique

- Etant donné un graphe orienté acyclique modélisant une relation d’ordre partiel, trouver une relation d’ordre **totale** entre les sommets qui **respecte l’ordre partiel**

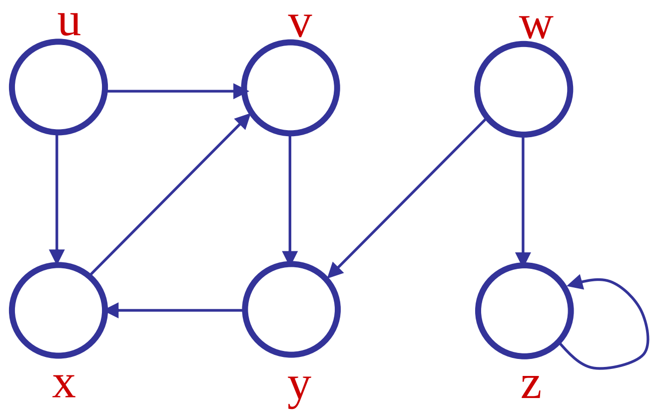
  $\rightarrow$ trouver un ordre des sommets tel qu'un sommet soit toujours visité avant ses successeurs

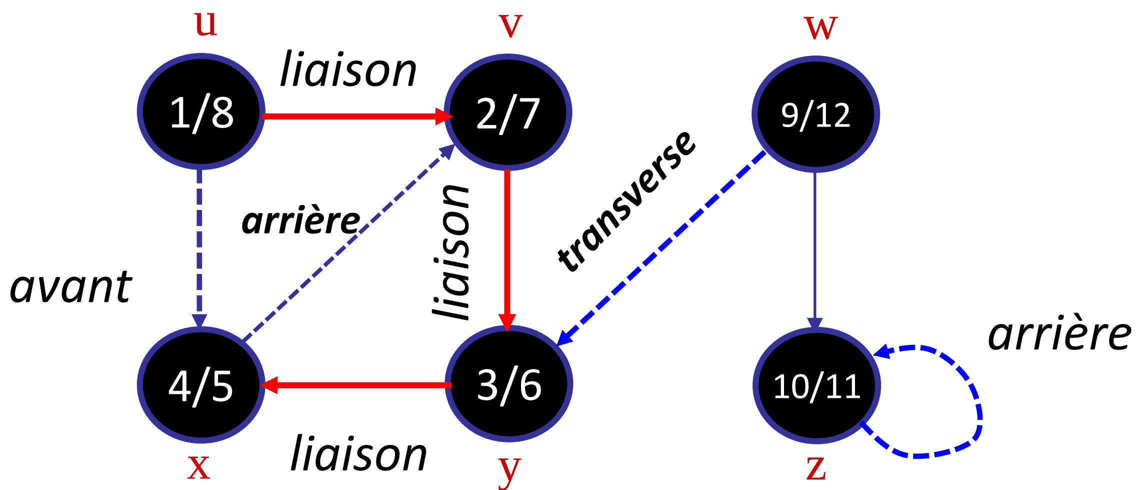




Tri topologique – classification des arcs

- Le type d'un arc (i,j) de $G = (S,A)$ peut être identifié lorsqu'il est exploré via le PeP
- L'identification est basée sur la **couleur** du sommet j
  - Blanc : arc de l'arbre
  - Gris : arc arrière
  - Noir : arc avant ou transverse ou croisement



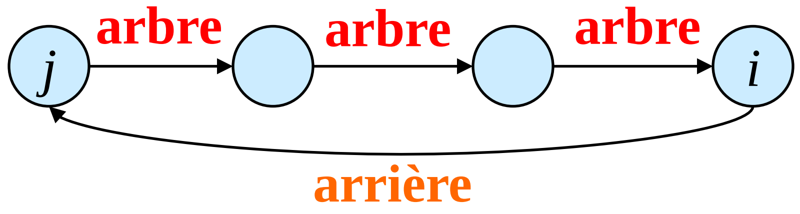


Détection d'un circuit

- **Proposition :** G possède un circuit si et seulement si **il existe un arc arrière** dans un parcours en profondeur de G
- Après le PeP on a
  - $d[i] =$ début de découverte
  - $f[i] =$ fin de traitement
  - Un arc (i, j) est un
    - arc d'arbre ou arc avant ssi $d(i) < d(j) < f(j) < f(i)$
    - **arc arrière** ssi $d(j) < d(i) < f(i) < f(j)$
    - arc de traverse ssi $f(j) < d(i)$

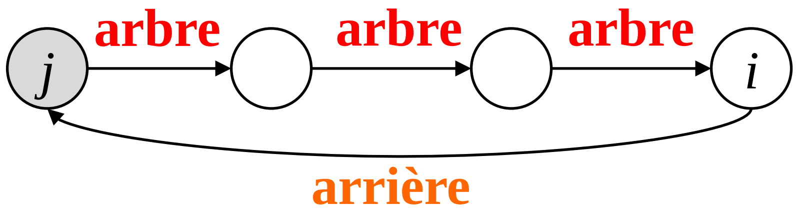
Détection d'un circuit

- **Proposition :** G possède un circuit si et seulement si **il existe un arc arrière** dans un parcours en profondeur de G
- Preuve
  - $\Rightarrow$: Montrons que l'existence d'un arc arrière $\Rightarrow$ cycle
    - Supposons qu'il y ait un arc arrière (i, j).
    - Alors j est l'ancêtre de i dans une forêt profondeur
    - Donc il y a un chemin π_{ji} de j à i
    - Donc $\pi_{ji} + (i, j)$ est un cycle



Détection d'un circuit

- **Proposition :** G possède un circuit si et seulement si **il existe un arc arrière** dans un parcours en profondeur de G
- Preuve
  - $\Leftarrow$: Montrons qu'un cycle implique l'existence d'un arc arrière
    - Soient π un cycle dans G, j le **premier** sommet découvert de π
    - Soit (i, j) l'arc qui forme π
    - Dans le PeP : au moment $d[j]$ les sommets de π forment un chemin de j à i composé de **sommets blancs**
    - Donc i est un descendant de j dans la forêt PeP
    - Donc (i, j) est un arc arrière



Tri topologique – méthode

$Tri - Topologique(G)$

1. Appeler $PeP(G)$ pour calculer les dates de fin $f[i]$ pour tout $i \in S$
2. Chaque fois que le traitement d'un sommet est fini, insérer le sommet **au début d'une liste chaînée**
3. **Retourner** la liste chaînée de sommets

Tri topologique – algorithme

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

Entrée : Un graphe $G = (S, A)$

Sortie : $\boldsymbol{L}$ **Liste chaînée de l'ordre linéaire**

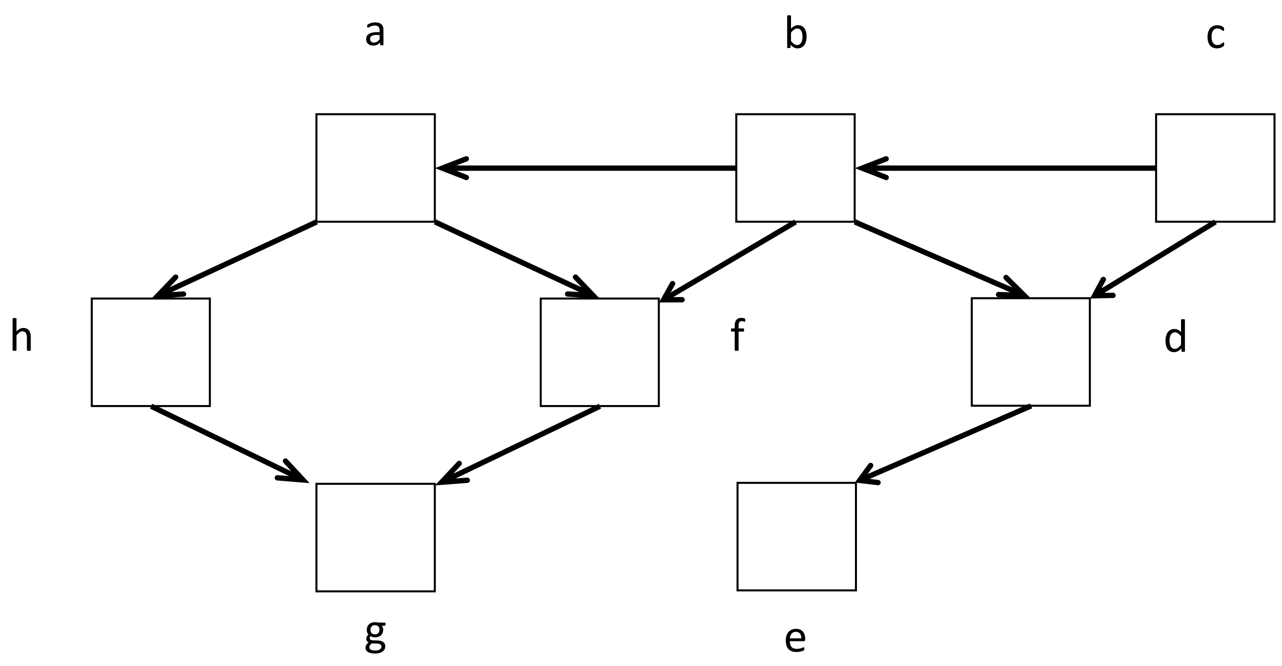
```
procédure PEP (Entrée : G
                 Sortie : L)
   pour x ∈ S faire
      c[x] ← Blanc ;   p[x] ← Nul ;
   fin pour
   temps ← 1;  L ← ∅ ;
   pour x ∈ S faire
      si(c[x] = Blanc) alors
         Visiter(x) ;
      fin si
   fin pour
```

$d[i]$: 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

```
Procédure Visiter(x)
   c[x] ← Gris ;
   d[x] ← temps ;    temps ← temps + 1;
   pour y ∈ V⁺(x) faire
      si (c[y] = Blanc) alors
         p[y] ← x;
         Visiter(y);
      fin si
   fin pour
   c[x] ← Noir;      L ← {x} + L ;
   f[x] ← temps ;    temps ← temps + 1;
```

Complexité : $O(n + m)$

Tri topologique – Exemple

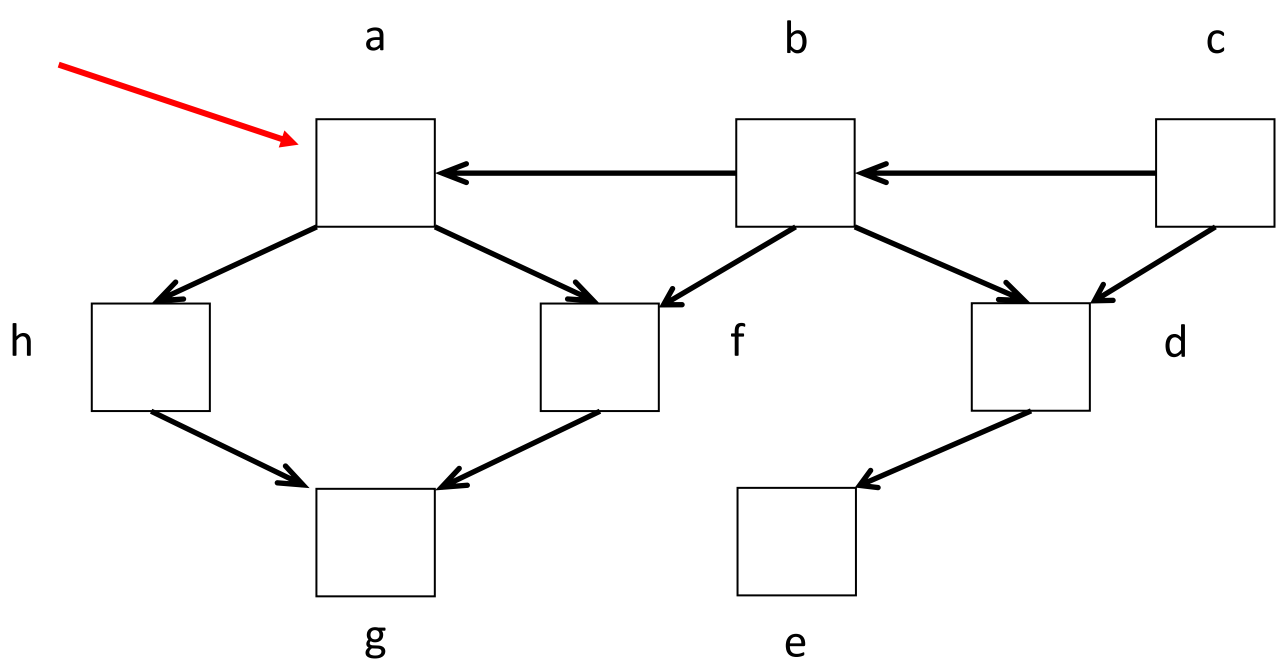


Liste chaînée

$\varnothing$

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

Tri topologique – Exemple



Liste chaînée

$\varnothing$

procédure PEP (Entrée : G **Sortie : L**)

pour x $\in$ S faire

c[x] $\leftarrow$ Blanc ; p[x] $\leftarrow$ Nul ;

fin pour

temps $\leftarrow$ 1; **L $\leftarrow \varnothing$;**

pour x $\in$ S faire

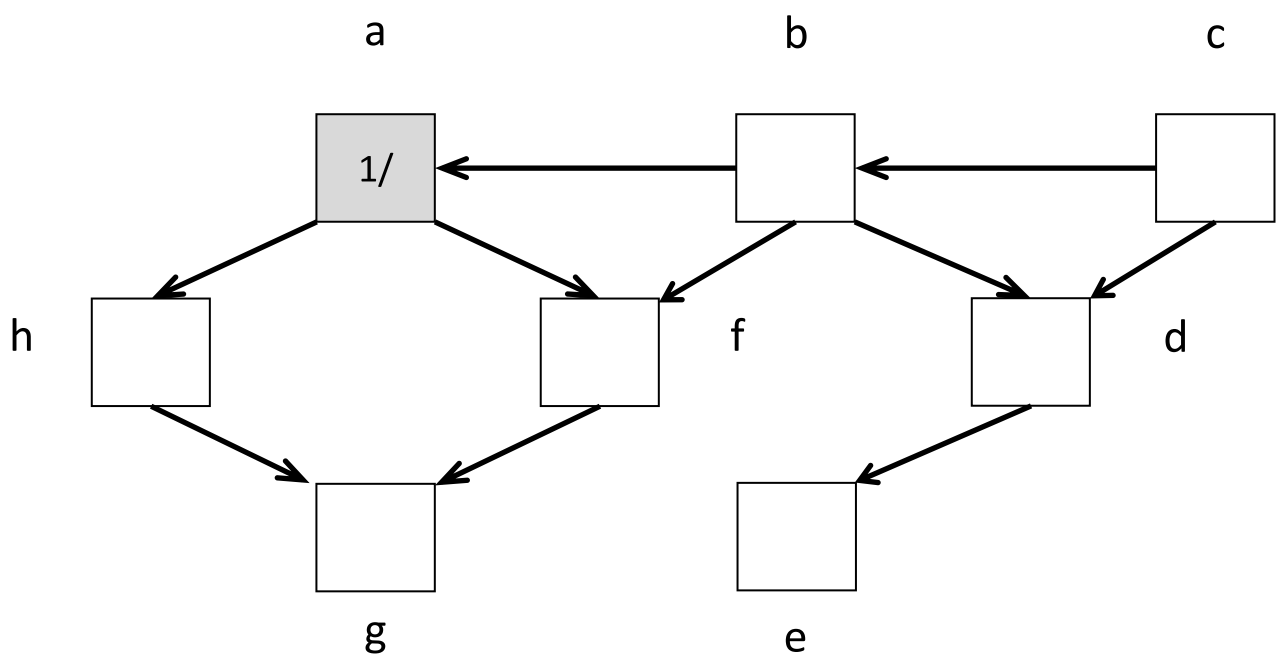
si(c[x] = Blanc) alors

Visiter(x) ;

fin si

fin pour

Tri topologique – Exemple



Liste chaînée

$\varnothing$

Procédure **Visiter**(x)

c[x] ← Gris ;

d[x] ← temps ; temps ← temps + 1;

pour y ∈ $\boldsymbol{V^+(\mathbf{x})}$ faire

si (c[y] = Blanc) alors

p[y] ← x;

Visiter(y);

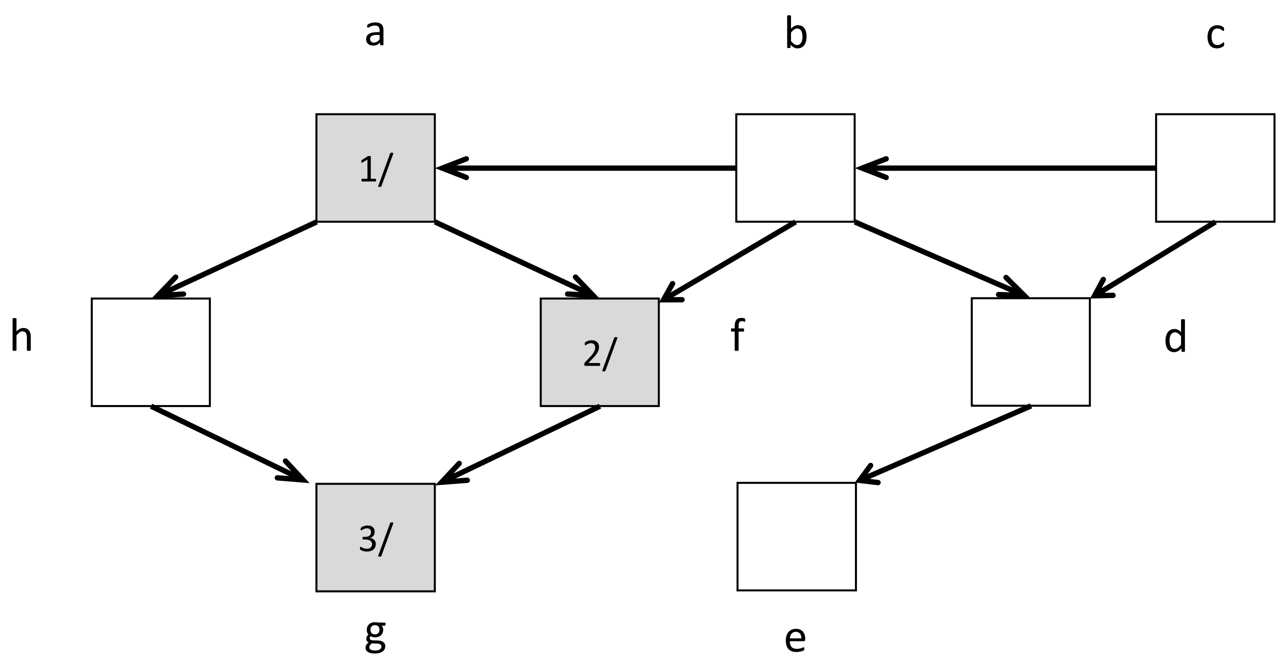
fin si

fin pour

c[x] ← Noir; **L ← {x} + L ;**

f[x] ← temps ; temps ← temps + 1;

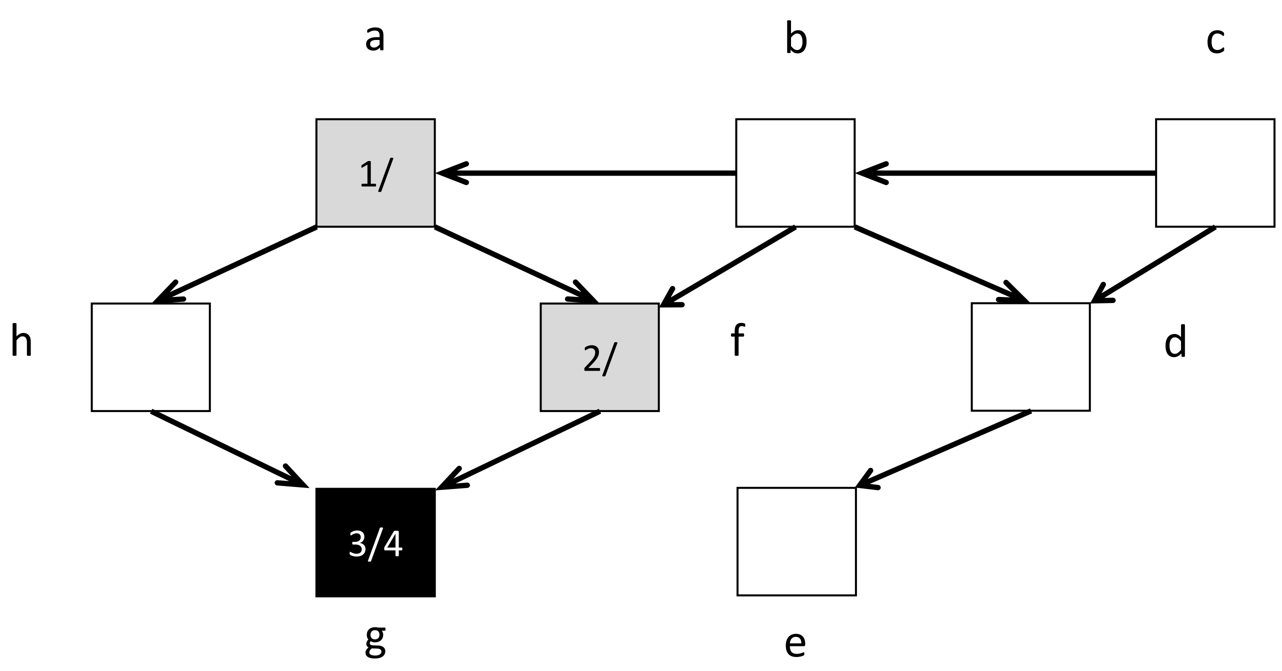
Tri topologique – Exemple



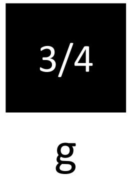
Liste chaînée
$\varnothing$

Procédure **Visiter**(x)
    c[x] ← Gris ;
    d[x] ← temps ; temps ← temps + 1;
    pour y ∈ $V^+(\mathbf{x})$ faire
        si (c[y] = Blanc) alors
            p[y] ← x;
            Visiter(y);
        fin si
    fin pour
    c[x] ← Noir; **L ← {x} + L ;**
    f[x] ← temps ; temps ← temps + 1;

Tri topologique – Exemple



Liste chaînée



Procédure **Visiter**(x)

c[x] ← Gris ;

d[x] ← temps ; temps ← temps + 1;

pour y ∈ $\boldsymbol{V^+(x)}$ faire

si (c[y] = Blanc) alors

p[y] ← x;

Visiter(y);

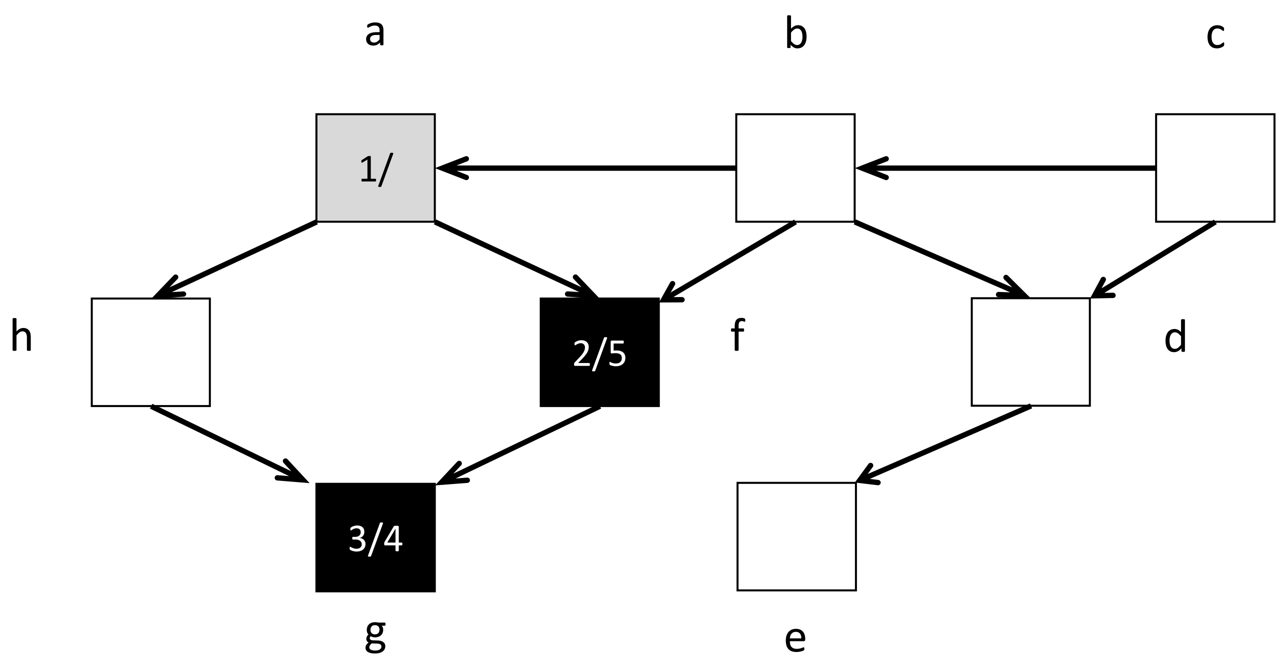
fin si

fin pour

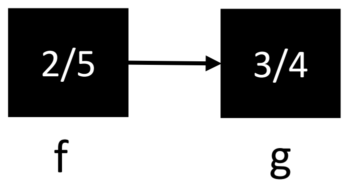
c[x] ← Noir; **L ← {x} + L ;**

f[x] ← temps ; temps ← temps + 1;

Tri topologique – Exemple



Liste chaînée



Procédure **Visiter**(x)

 c[x] ← Gris ;

 d[x] ← temps ;   temps ← temps + 1;

 pour y ∈ $V^+(x)$ faire

  si (c[y] = Blanc) alors

   p[y] ← x;

   Visiter(y);

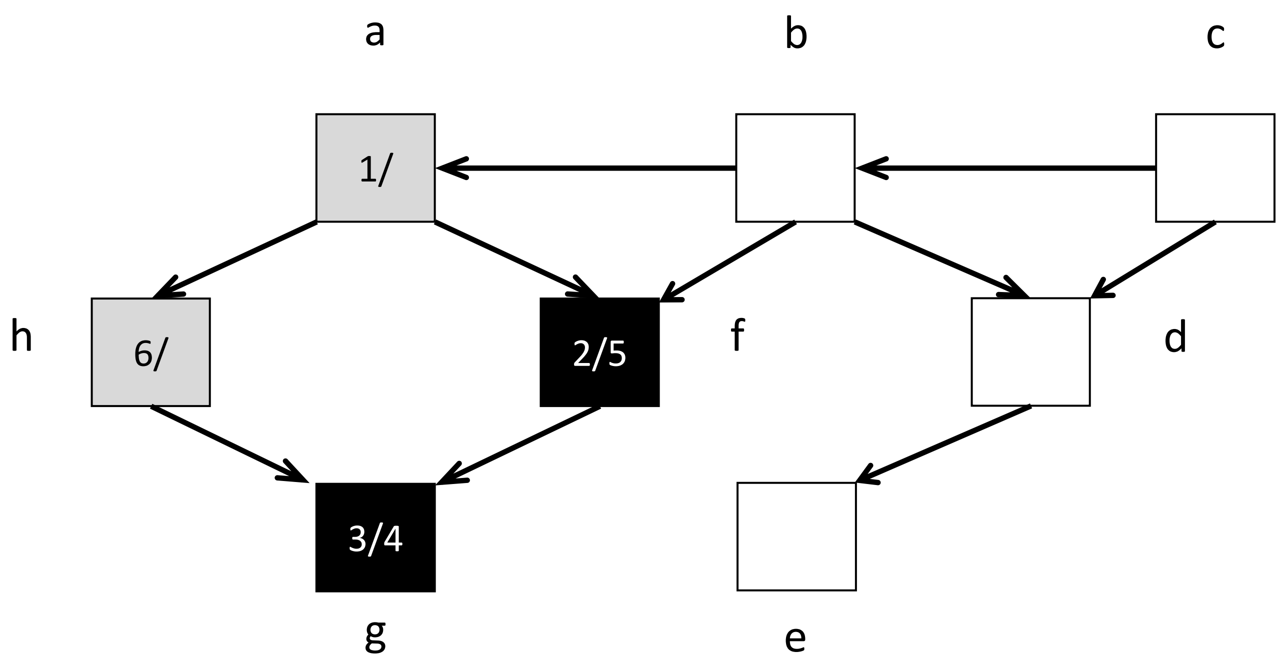
  fin si

 fin pour

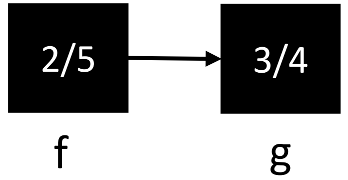
 c[x] ← Noir;   **L ← {x} + L ;**

 f[x] ← temps ;   temps ← temps + 1;

Tri topologique – Exemple



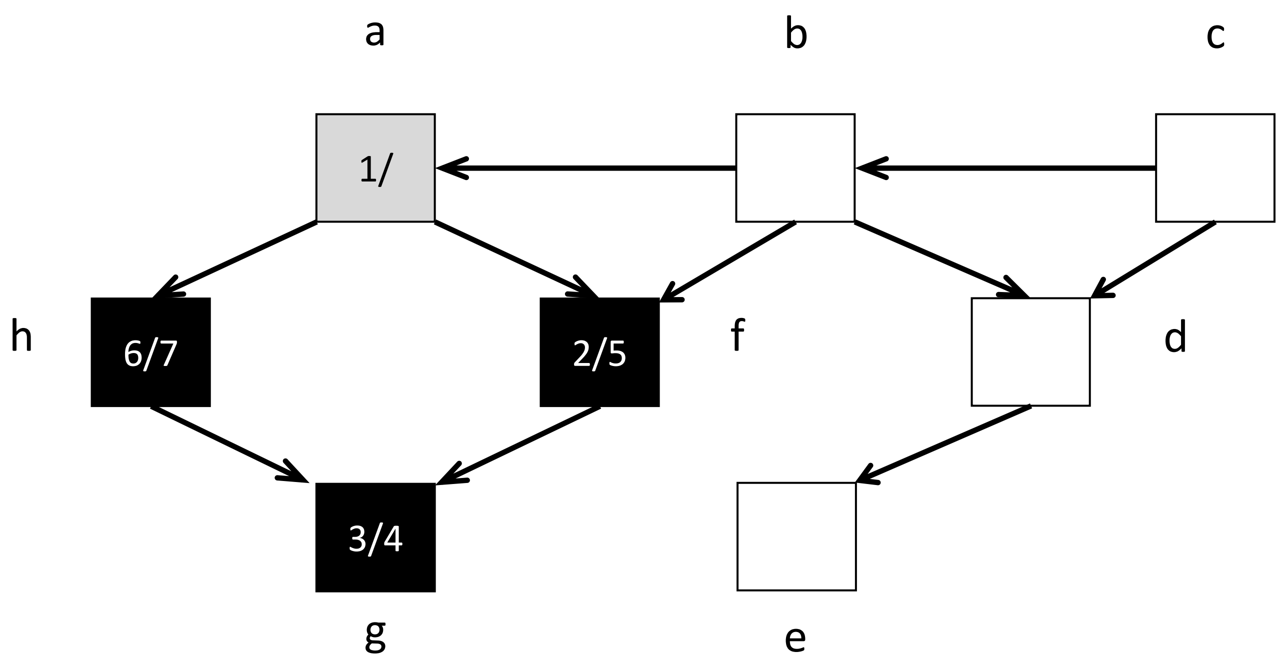
Liste chaînée



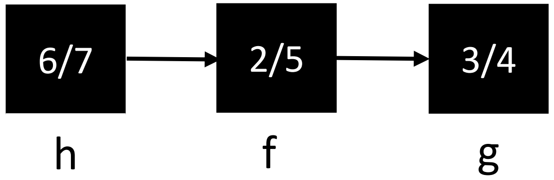
Procédure **Visiter**(x)

    c[x] ← Gris ;
    d[x] ← temps ; temps ← temps + 1;
    pour y ∈ $V^{+}(\mathbf{x})$ faire
        si (c[y] = Blanc) alors
            p[y] ← x;
            Visiter(y);
        fin si
    fin pour
    c[x] ← Noir; **L ← {x} + L ;**
    f[x] ← temps ; temps ← temps + 1;

Tri topologique – Exemple



Liste chaînée



Procédure **Visiter**(x)

c[x] ← Gris ;

d[x] ← temps ;   temps ← temps + 1;

pour y ∈ $V^+(x)$ faire

si (c[y] = Blanc) alors

p[y] ← x;

Visiter(y);

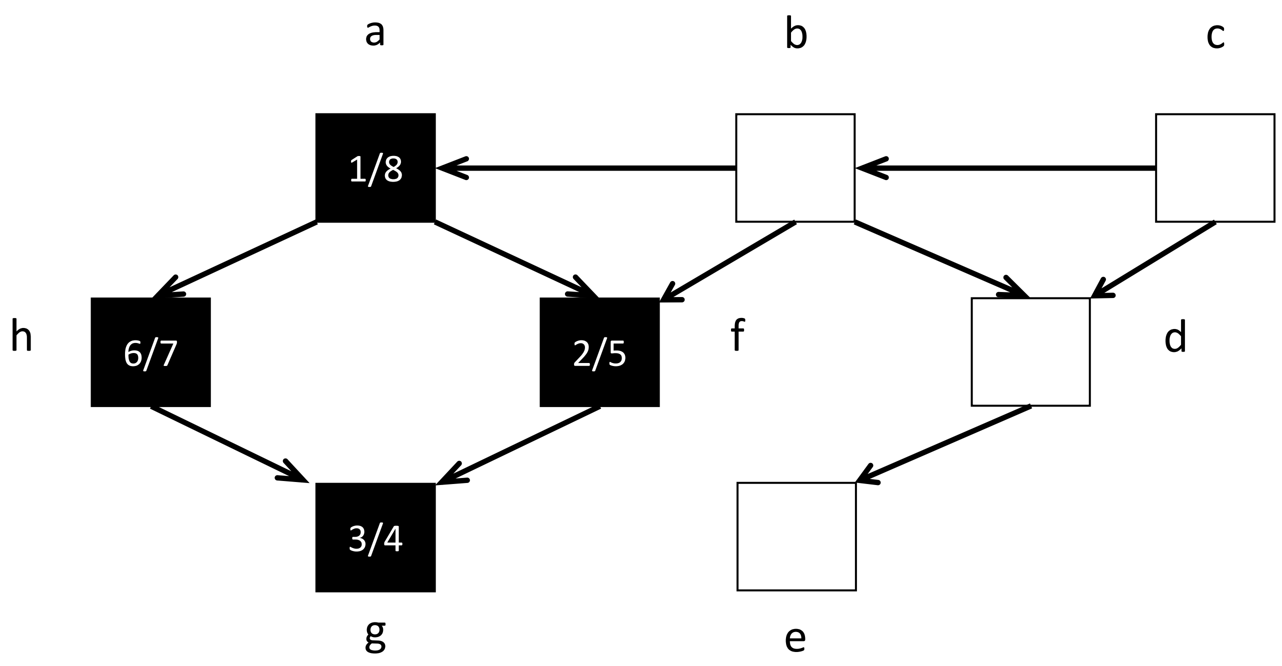
fin si

fin pour

c[x] ← Noir;   **L ← {x} + L ;**

f[x] ← temps ;   temps ← temps + 1;

Tri topologique – Exemple



Procédure **Visiter**(x)

c[x] ← Gris ;

d[x] ← temps ;   temps ← temps + 1;

pour y ∈ $V^+(x)$ faire

si (c[y] = Blanc) alors

p[y] ← x;

Visiter(y);

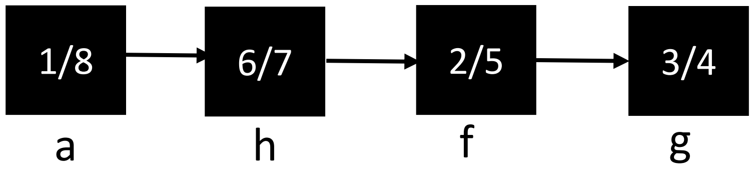
fin si

fin pour

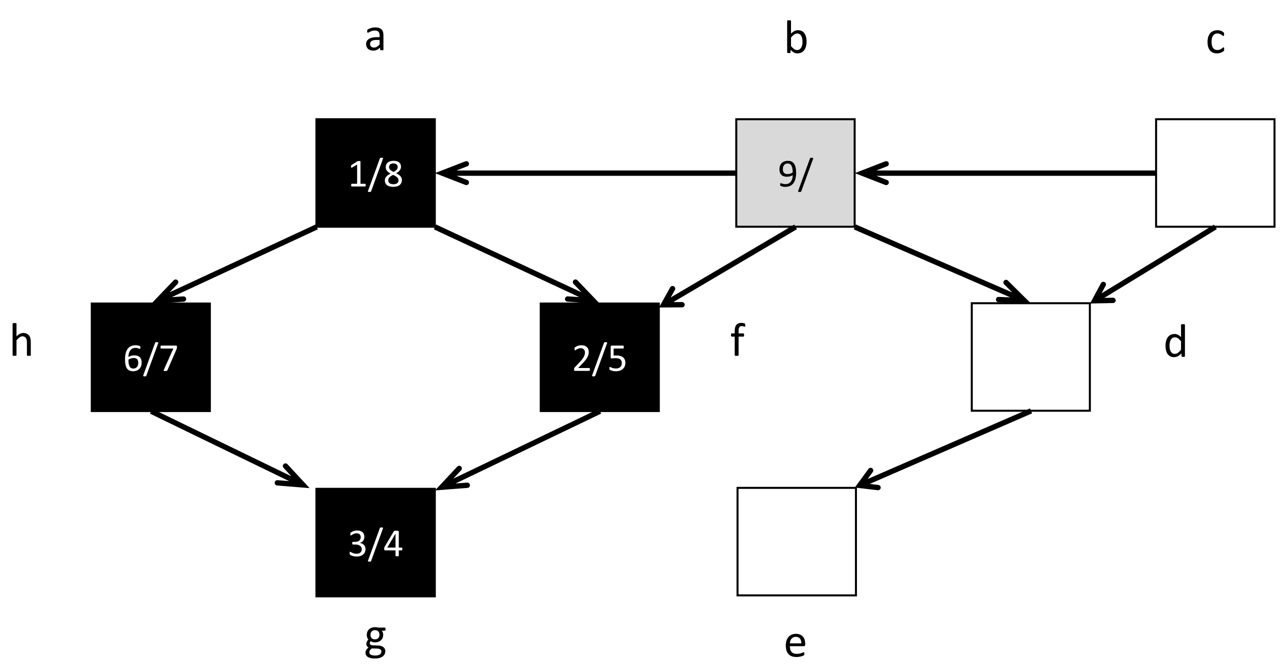
c[x] ← Noir;   **L ← {x} + L ;**

f[x] ← temps ;   temps ← temps + 1;

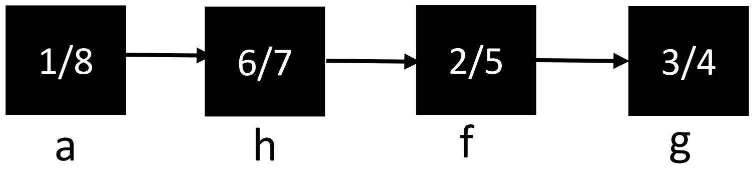
Liste chaînée



Tri topologique – Exemple

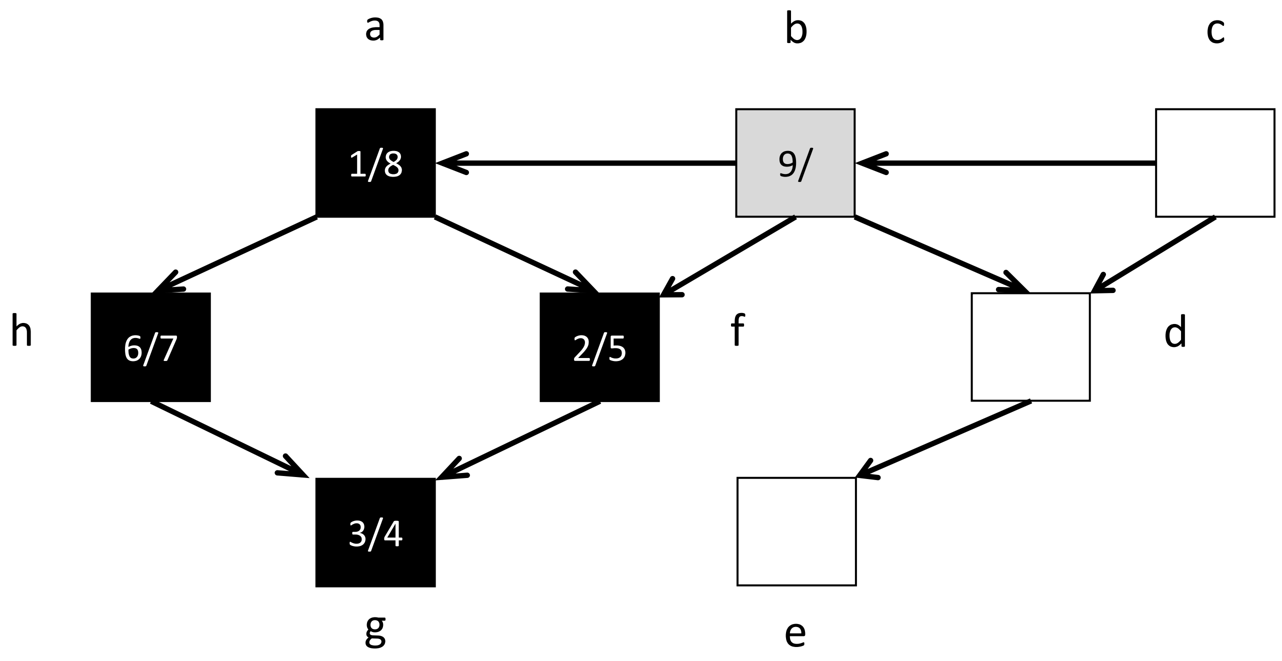


Liste chaînée

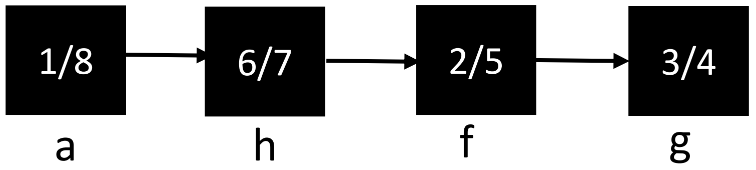


procédure PEP (Entrée : G
Sortie : L)
pour x ∈ S faire
c[x] ← Blanc ; p[x] ← Nul ;
fin pour
temps ← 1; **L ← ∅ ;**
pour x ∈ S faire
si(c[x] = Blanc) alors
Visiter(x) ;
fin si
fin pour

Tri topologique – Exemple



Liste chaînée



Procédure **Visiter**(x)

c[x] ← Gris ;

d[x] ← temps ;   temps ← temps + 1;

pour y ∈ $\boldsymbol{V^{+}(\mathbf{x})}$ faire

si (c[y] = Blanc) alors

p[y] ← x;

Visiter(y);

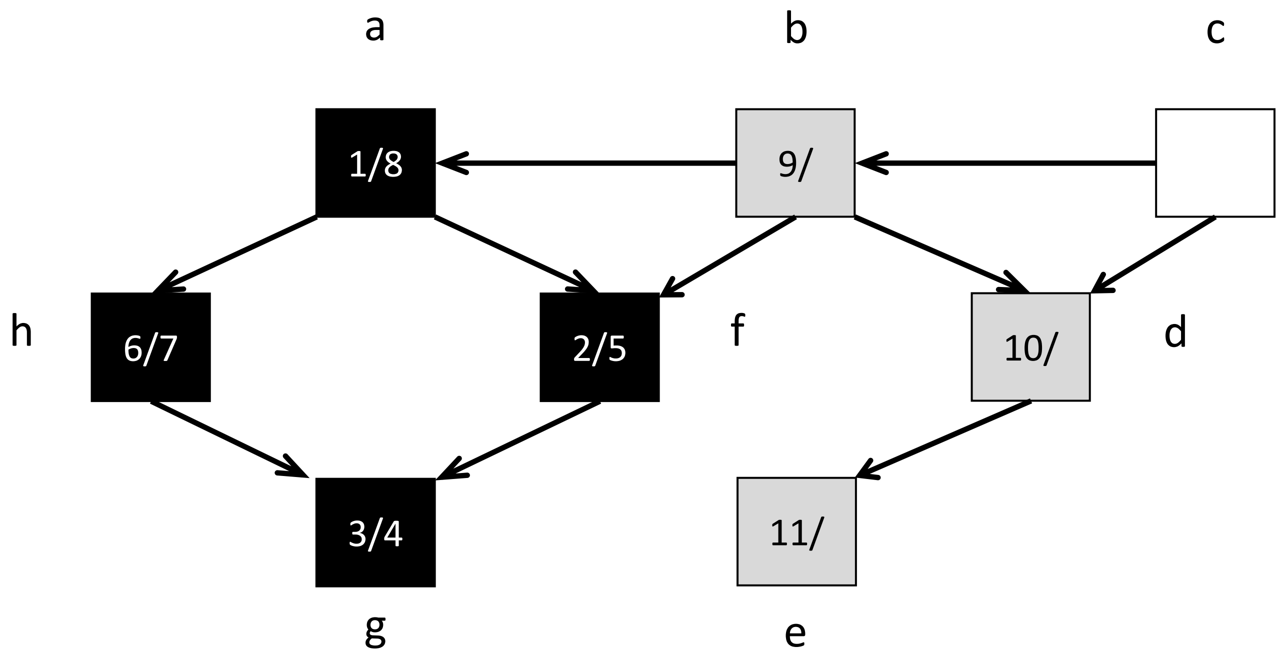
fin si

fin pour

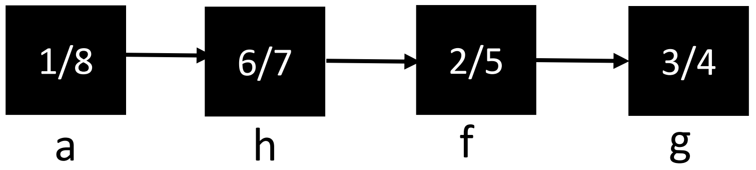
c[x] ← Noir;   **L ← {x} + L ;**

f[x] ← temps ;   temps ← temps + 1;

Tri topologique – Exemple



Liste chaînée



Procédure **Visiter**(x)

c[x] ← Gris ;

d[x] ← temps ; temps ← temps + 1;

pour y ∈ $\boldsymbol{V^{+}(\mathbf{x})}$ faire

si (c[y] = Blanc) alors

p[y] ← x;

Visiter(y);

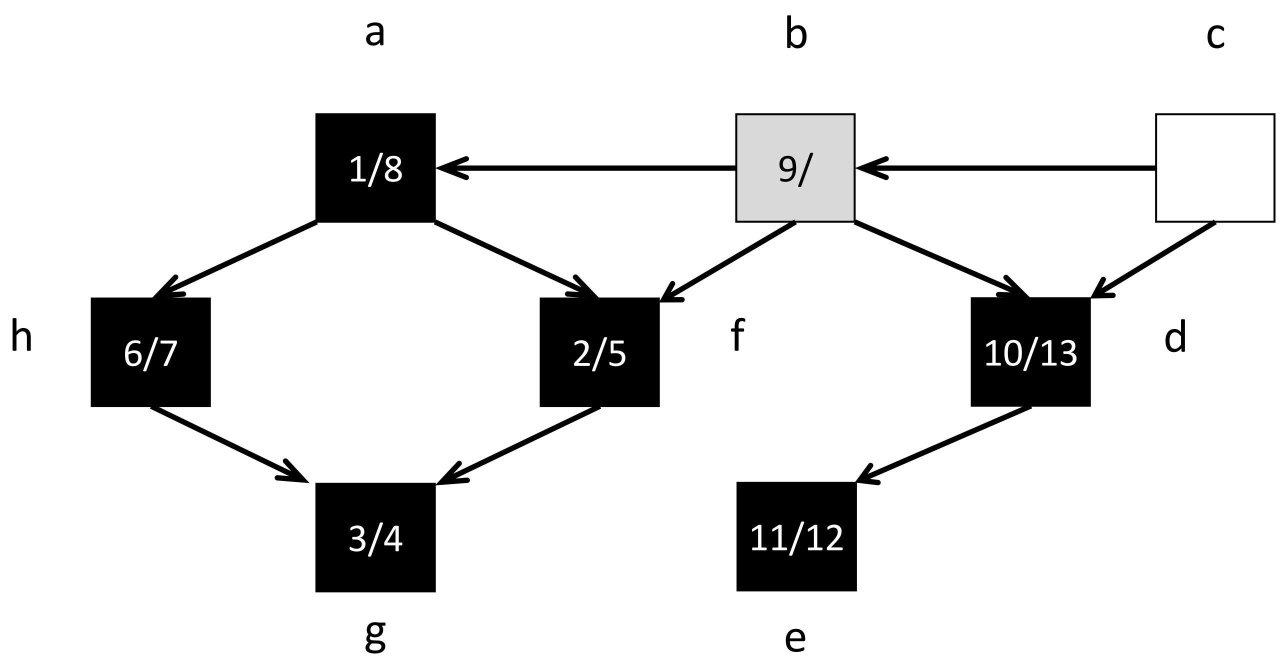
fin si

fin pour

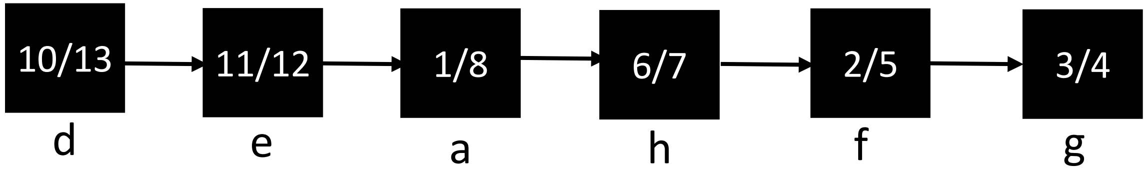
c[x] ← Noir; **L ← {x} + L ;**

f[x] ← temps ; temps ← temps + 1;

Tri topologique – Exemple



Liste chaînée



Procédure **Visiter**(x)

c[x] ← Gris ;

d[x] ← temps ;   temps ← temps + 1;

pour y ∈ $V^{+}(\mathbf{x})$ faire

si (c[y] = Blanc) alors

p[y] ← x;

Visiter(y);

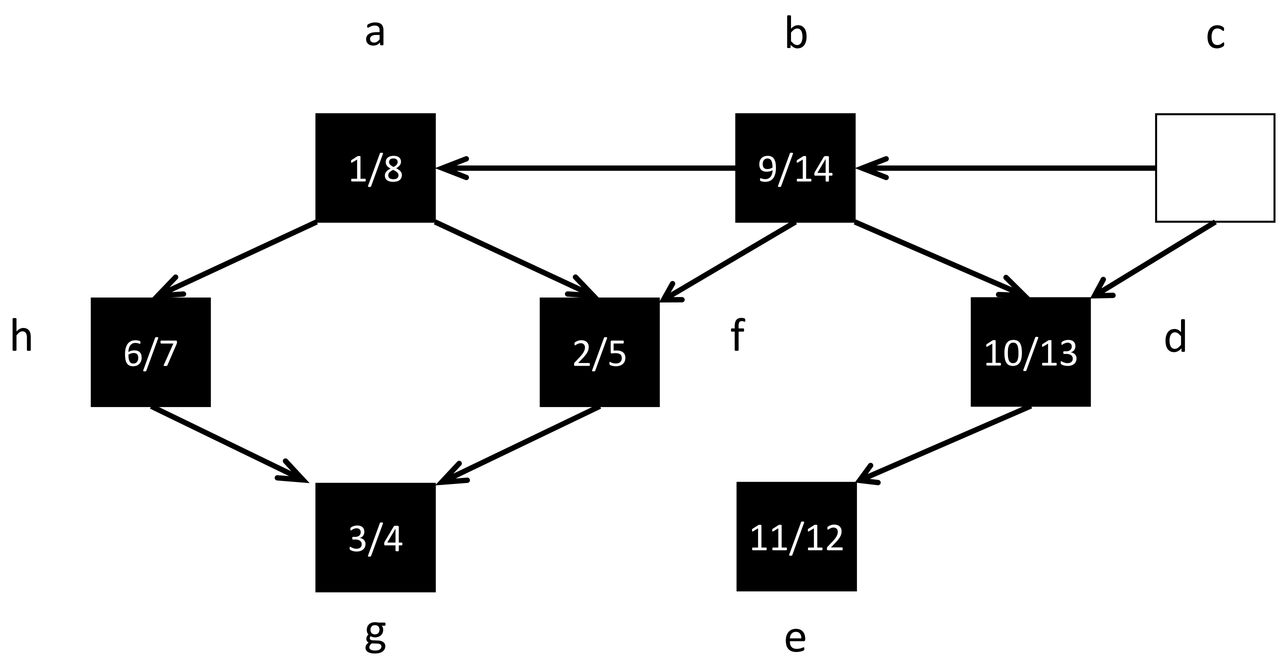
fin si

fin pour

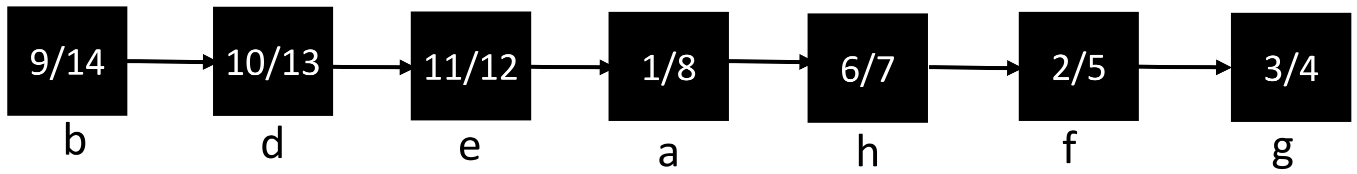
c[x] ← Noir;   **L ← {x} + L ;**

f[x] ← temps ;   temps ← temps + 1;

Tri topologique – Exemple



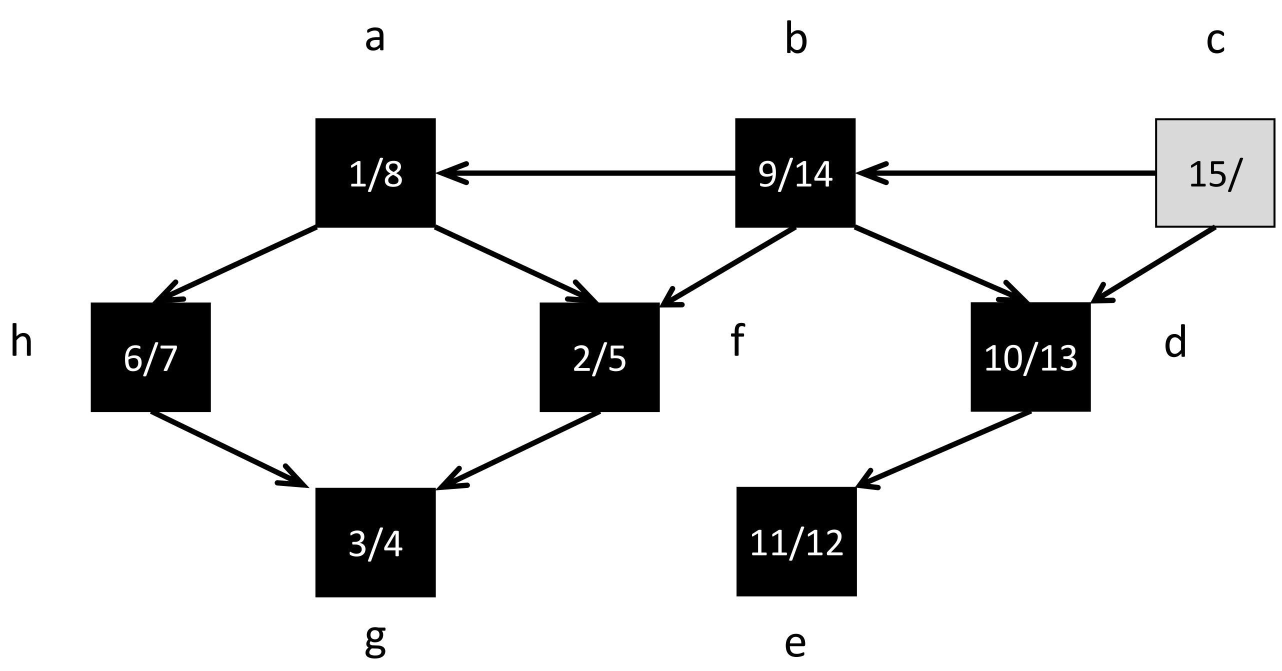
Liste chaînée



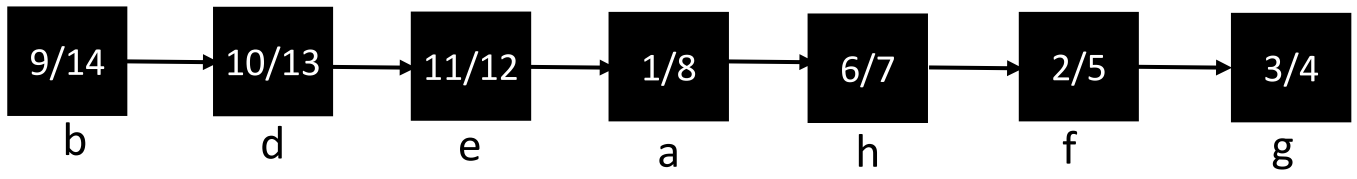
Procédure **Visiter**(x)

    c[x] ← Gris ;
    d[x] ← temps ;   temps ← temps + 1;
    pour y ∈ $\boldsymbol{V^+(x)}$ faire
        si (c[y] = Blanc) alors
            p[y] ← x;
            Visiter(y);
        fin si
    fin pour
    c[x] ← Noir;   **L ← {x} + L ;**
    f[x] ← temps ;   temps ← temps + 1;

Tri topologique – Exemple

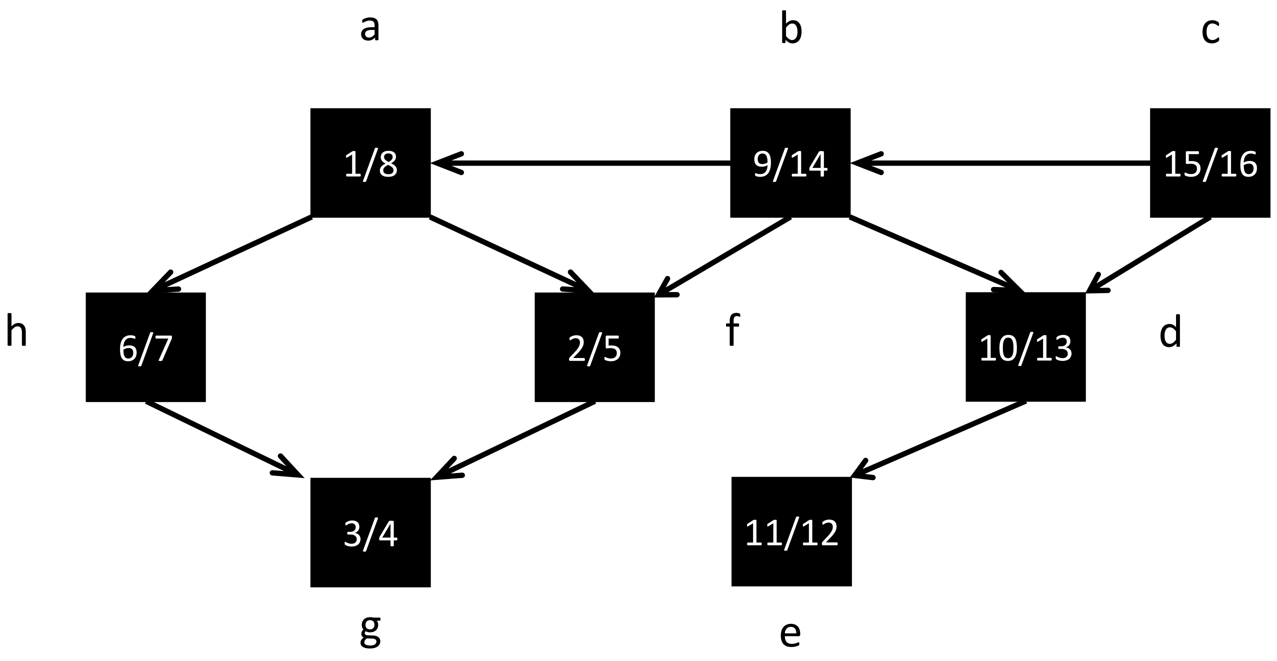


Liste chaînée




```
Procédure Visiter(x)
    c[x] ← Gris ;
    d[x] ← temps ;    temps ← temps + 1;
    pour y ∈ V⁺(x) faire
        si (c[y] = Blanc) alors
            p[y] ← x;
            Visiter(y);
        fin si
    fin pour
    c[x] ← Noir;       L ← {x} + L ;
    f[x] ← temps ;    temps ← temps + 1;
```

Tri topologique – Exemple



Procédure **Visiter**(x)

c[x] ← Gris ;

d[x] ← temps ;   temps ← temps + 1;

pour y ∈ $\boldsymbol{V^{+}(\mathbf{x})}$ faire

si (c[y] = Blanc) alors

p[y] ← x;

Visiter(y);

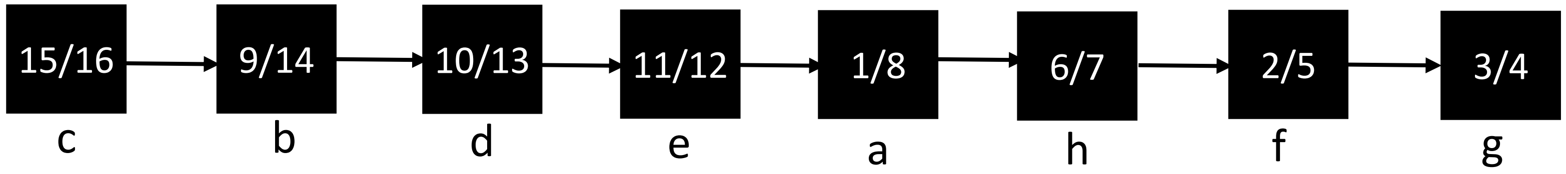
fin si

fin pour

c[x] ← Noir;   **L ← {x} + L ;**

f[x] ← temps ;   temps ← temps + 1;

Liste chaînée



Tri topologique – validité de l'algorithme

- **Théorème :** l'algorithme $Tri - Topologique(G, L)$ réalise le tri topologique d'un graphe orienté acyclique $G = (S, A)$
- Preuve
  - On a juste besoin de montrer $(i, j) \in A \rightarrow f(j) < f(i)$
  - Quand on explore (i, j), **quelles sont les couleurs de *i* et *j* ?**
  - $c(i) = Gris$
    - $\boldsymbol{c(j) = Gris}$ **?**
      - *Non*, car alors j serait l'ancêtre de i
      - $\Rightarrow (i, j)$ est un arc arrière
      - $\Rightarrow$ contradiction car G n'a aucun arc arrière par définition

Tri topologique – validité de l'algorithme

- **Théorème :** l'algorithme $Tri - Topologique(G, L)$ réalise le tri topologique d'un graphe orienté acyclique $G = (S, A)$
- Preuve
  - On a juste besoin de montrer $(i, j) \in A \rightarrow f(j) < f(i)$
  - Quand on explore (i, j), **quelles sont les couleurs de *i* et *j* ?**
  - $c(i) = Gris$
    - $\boldsymbol{c(j) = Blanc}$ **?**
      - Alors il devient descendant de i
      - Par le théorème de parenthèse, $d[i] < d[j] < \boldsymbol{f[j] < f[i]}$

Tri topologique – validité de l’algorithme

- **Théorème :** l’algorithme $Tri - Topologique(G, L)$ réalise le tri topologique d’un graphe orienté acyclique $G = (S, A)$
- Preuve
  - On a juste besoin de montrer $(i, j) \in A \rightarrow f(j) < f(i)$
  - Quand on explore (i, j), **quelles sont les couleurs de *i* et *j* ?**
  - $c(i) = Gris$
    - $\boldsymbol{c(j) = Noir}$ **?**
      - Alors j est dejà terminé
      - Puisque nous explorons (i, j), nous n’avons pas encore terminé i
      - Donc, $\boldsymbol{f[j]<f[i]}$

Structure d'un graphe sans circuit

- Notion de **niveau** dans un graphe orienté acyclique $G = (S, A)$
  - $\forall i \in S$ on définit $Niveau(i)$ le nombre d'arcs maximum d'un chemin élémentaire se terminant en i
- **Théorème** : soit $G = (S, A)$ un graphe orienté
  - G est sans circuit ⇔ S admet une partition $S = N_0 \cup N_1 \cup \ldots \cup N_p$ telle que

$$\forall i \in S, \forall k \leq p, i \in N_k \leftrightarrow Niveau(i) = k$$

→ Hiérarchie des sommets du graphe

Hiérarchie

Entrée : un graphe orienté sans circuit $G = (S, A)$

Sortie : les niveaux N_k

$I = S, k = 0$

Tant que $I \neq \emptyset$ **faire**

$$N_k = \{i \in I : \boldsymbol{V^{-}_{G(I)}(i)} = \emptyset\}$$

$$I = I - N_k$$

$$k + 1$$

Fin tant que

$\boldsymbol{V^{-}_{G(I)}(i)}$ **: ensemble des prédécesseurs de $\boldsymbol{i}$ dans $\boldsymbol{G(I)}$**

Hiérarchie – Exemple

$I = S, k = 0$

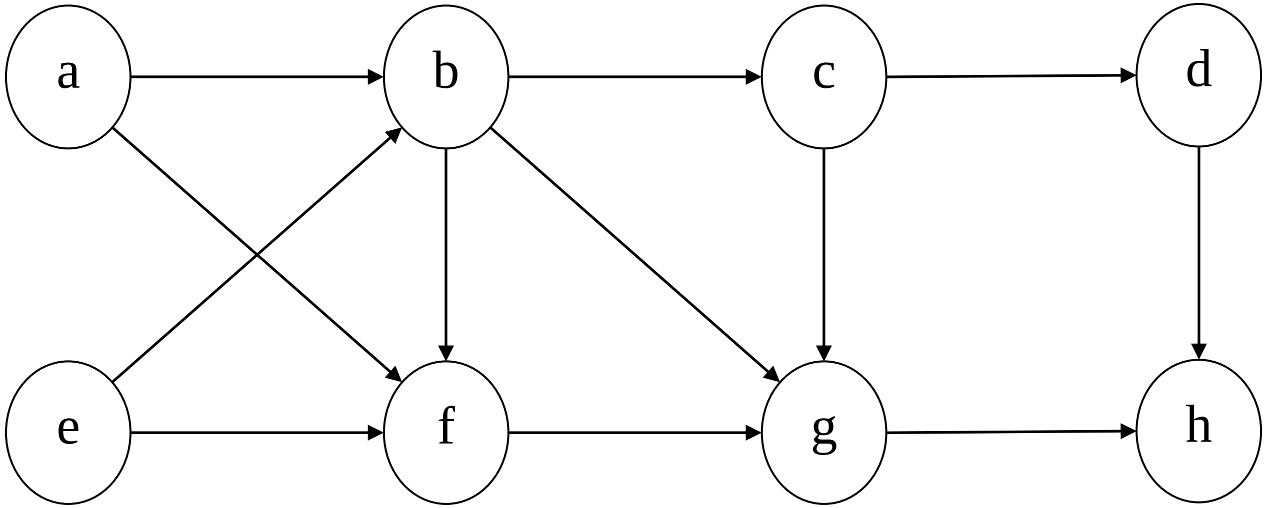
Tant que $I \neq \emptyset$ **faire**

$$N_k = \left\{i \in I : \boldsymbol{V^-_{G(I)}(i)} = \emptyset\right\}$$

$$I = I - N_k$$

$$k + 1$$

Fin tant que



$I = S \qquad G(I) = G$

Hiérarchie – Exemple

$I = S, k = 0$

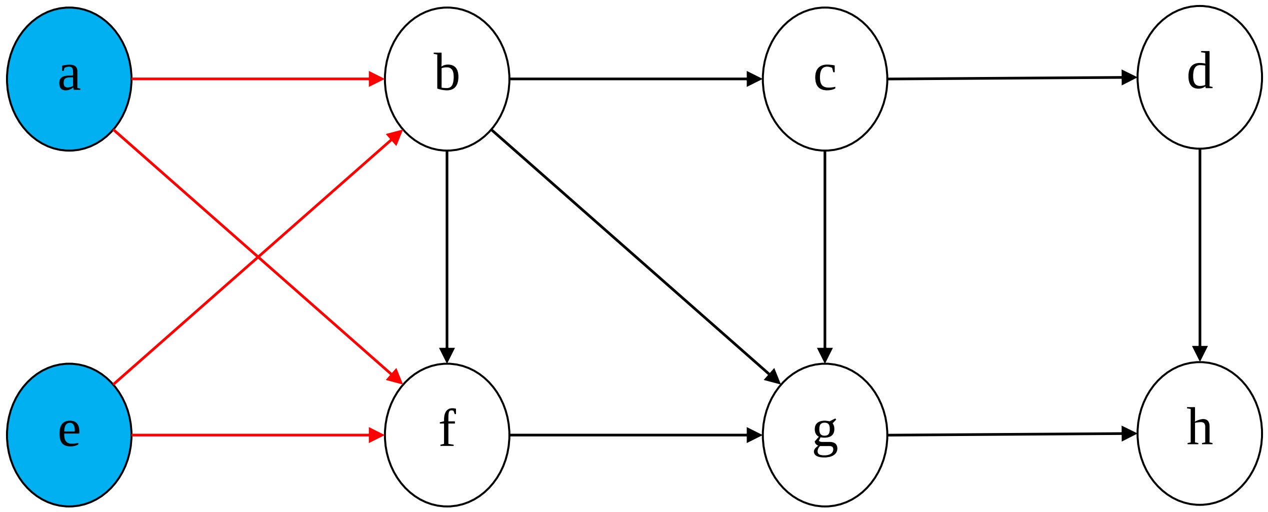
Tant que $I \neq \emptyset$ **faire**

$N_k = \{i \in I : \boldsymbol{V^-_{G(I)}(i)} = \emptyset\}$

$I = I - N_k$

$k + 1$

Fin tant que



$N_0 = \{i \in I : V^-_{G(I)}(i) = \emptyset\} = \{a, e\}$

$I = \{a, e\}$

$G(I) = (S - I, A - \{(a, b), (a, f), (e, b), (e, f)\})$

Hiérarchie – Exemple

$I = S, k = 0$

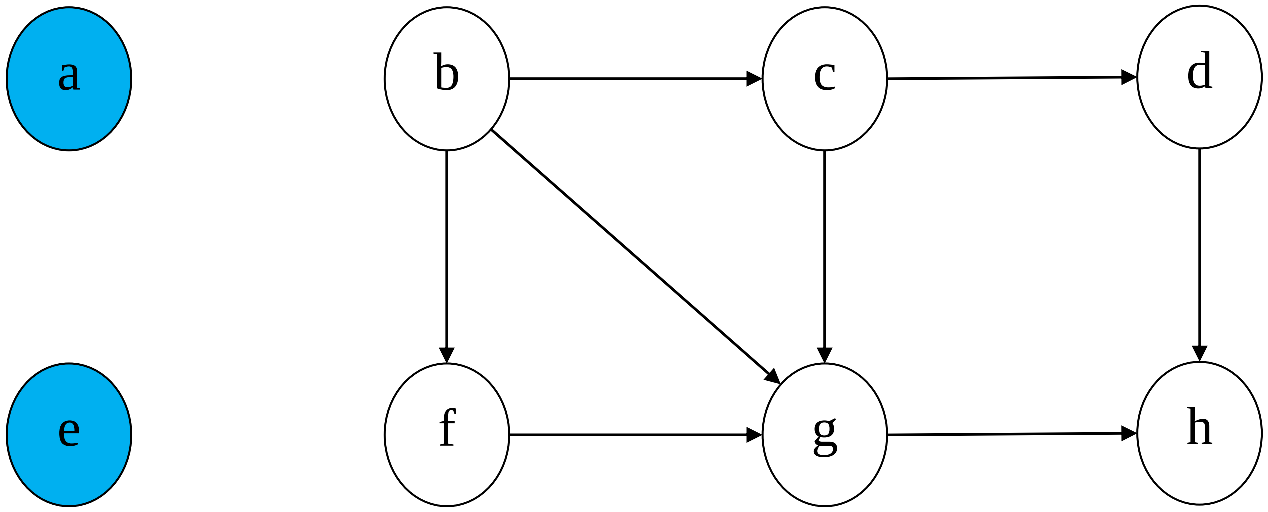
Tant que $I \neq \emptyset$ **faire**

$N_k = \{i \in I : \boldsymbol{V^-_{G(I)}(i)} = \emptyset\}$

$I = I - N_k$

$k + 1$

Fin tant que



$N_0 = \{i \in I : V^-_{G(I)}(i) = \emptyset\} = \{a, e\}$

$I = \{a, e\}$

$G(I) = (S - I, A - \{(a,b), (a,f), (e,b), (e,f)\})$

Hiérarchie – Exemple

$I = S, k = 0$

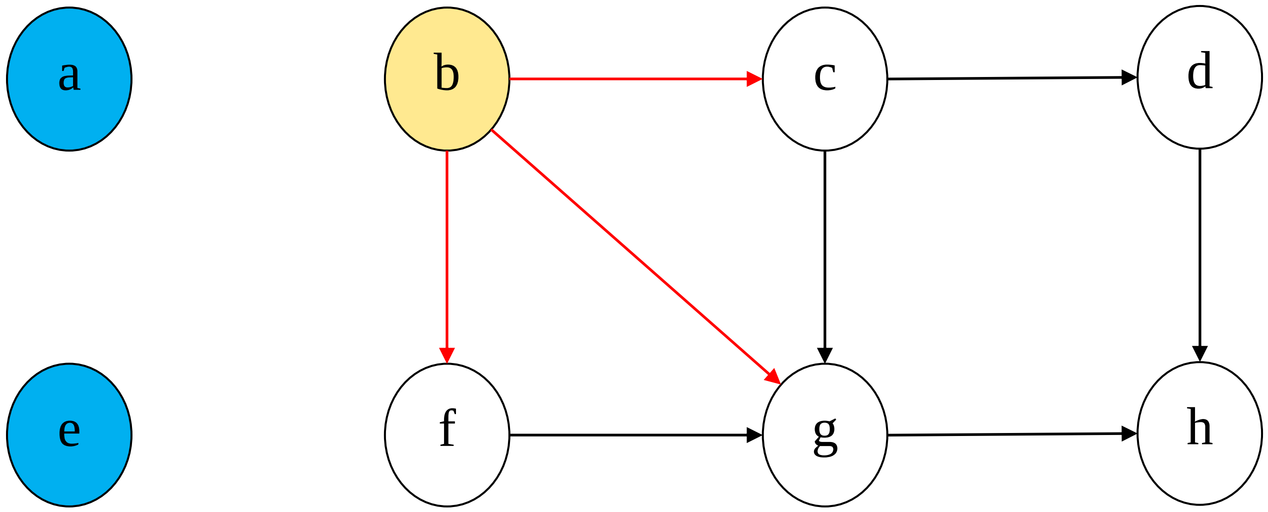
Tant que $I \neq \emptyset$ **faire**

$N_k = \{i \in I : \boldsymbol{V^-_{G(I)}(i)} = \emptyset\}$

$I = I - N_k$

$k + 1$

Fin tant que



$N_1 = \{i \in I : V^-_{G(I)}(i) = \emptyset\} = \{b\}$

$I = \{a, e, b\}$

$G(I) = (S - I, A - \{(a,b), (a,f), (e,b), (e,f), (b,c), (b,f), (b,g)\})$

Hiérarchie – Exemple

$I = S, k = 0$

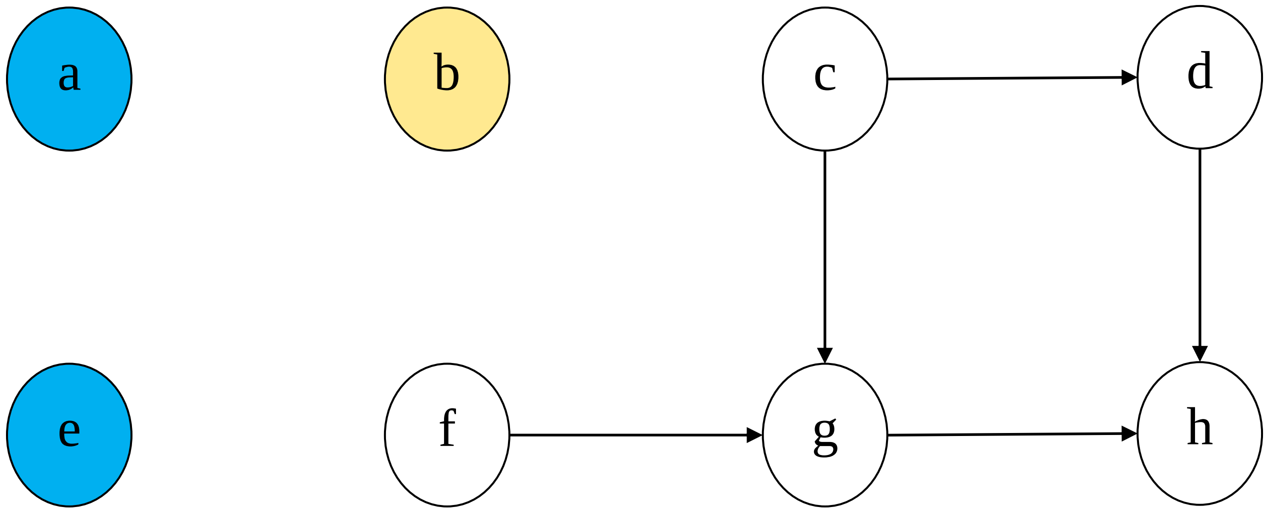
Tant que $I \neq \emptyset$ **faire**

$$N_k = \{i \in I : \boldsymbol{V^{-}_{G(I)}(i)} = \emptyset\}$$

$$I = I - N_k$$

$$k + 1$$

Fin tant que



$N_1 = \{i \in I : V^{-}_{G(I)}(i) = \emptyset\} = \{b\}$

$I = \{a, e, b\}$

$G(I) = (S - I, A - \{(a,b), (a,f), (e,b), (e,f), (b,c), (b,f), (b,g)\})$

Hiérarchie – Exemple

$I = S, k = 0$

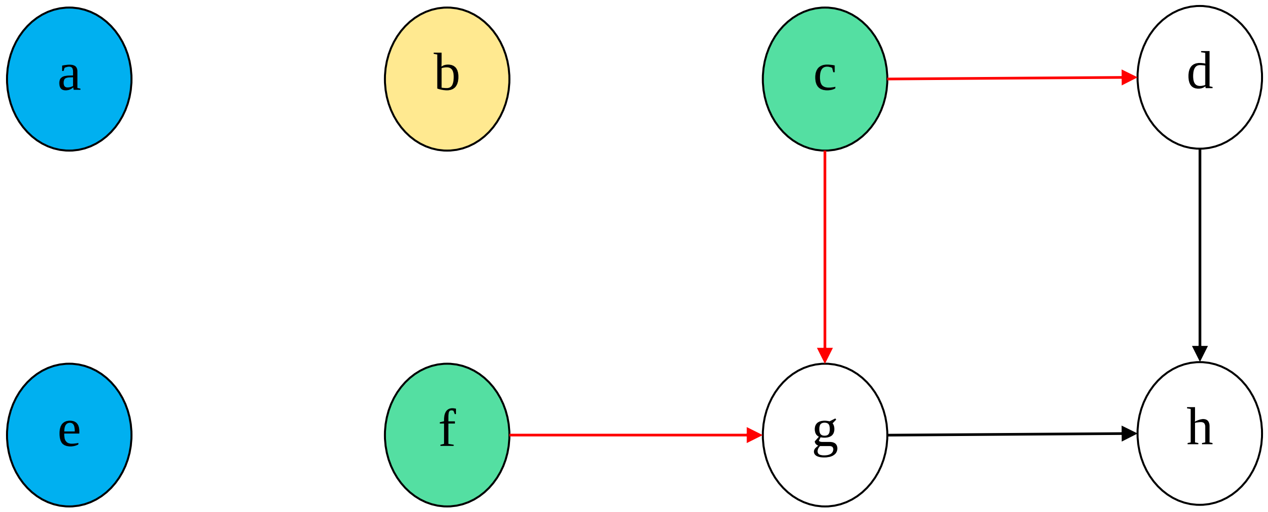
Tant que $I \neq \emptyset$ **faire**

$N_k = \{i \in I : \boldsymbol{V^-_{G(I)}(i)} = \emptyset\}$

$I = I - N_k$

$k + 1$

Fin tant que



$N_2 = \{i \in I : V^-_{G(I)}(i) = \emptyset\} = \{c, f\}$

$I = \{a, e, b, c, f\}$

$G(I) = (S - I, A - \{(a,b), (a,f), (e,b), (e,f), (b,c), (b,f), (b,g), (c,d), (c,g), (f,g)\})$

Hiérarchie – Exemple

$I = S, k = 0$

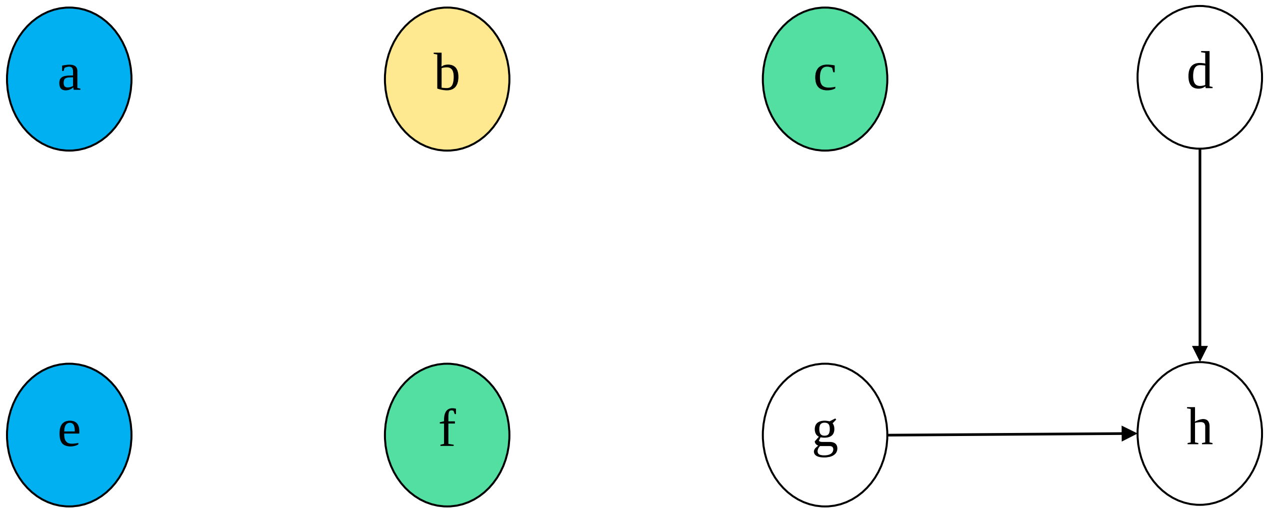
Tant que $I \neq \emptyset$ **faire**

$$N_k = \left\{ i \in I : \boldsymbol{V^{-}_{G(I)}(i)} = \emptyset \right\}$$

$$I = I - N_k$$

$$k + 1$$

Fin tant que



$$N_2 = \left\{ i \in I : V^{-}_{G(I)}(i) = \emptyset \right\} = \{c, f\}$$

$$I = \{a, e, b, c, f\}$$

$$G(I) = (S - I, A - \{(a,b), (a,f), (e,b), (e,f), (b,c), (b,f), (b,g), (c,d), (c,g), (f,g)\})$$

Hiérarchie – Exemple

$I = S, k = 0$

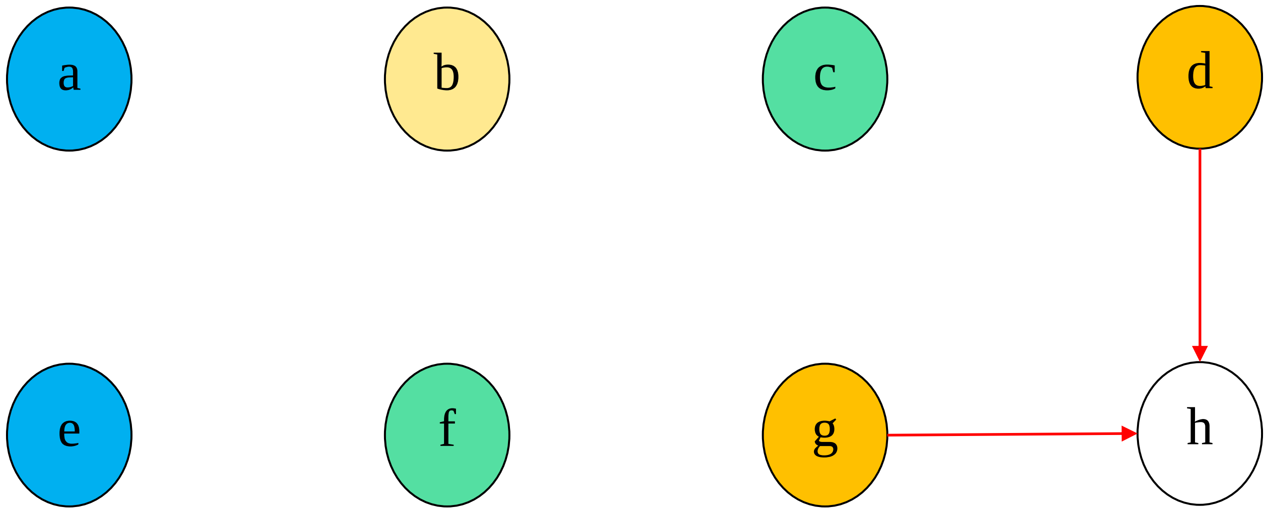
Tant que $I \neq \emptyset$ **faire**

$N_k = \{i \in I : \boldsymbol{V^{-}_{G(I)}(i)} = \emptyset\}$

$I = I - N_k$

$k + 1$

Fin tant que



$N_3 = \{i \in I : V^{-}_{G(I)}(i) = \emptyset\} = \{d, g\}$

$I = \{a, e, b, c, f, d, g\}$

$G(I) = (S - I, A - \{(a,b), (a,f), (e,b), (e,f), (b,c), (b,f), (b,g), (c,d), (c,g), (f,g), (d,h), (g,h)\})$

Hiérarchie – Exemple

$I = S, k = 0$

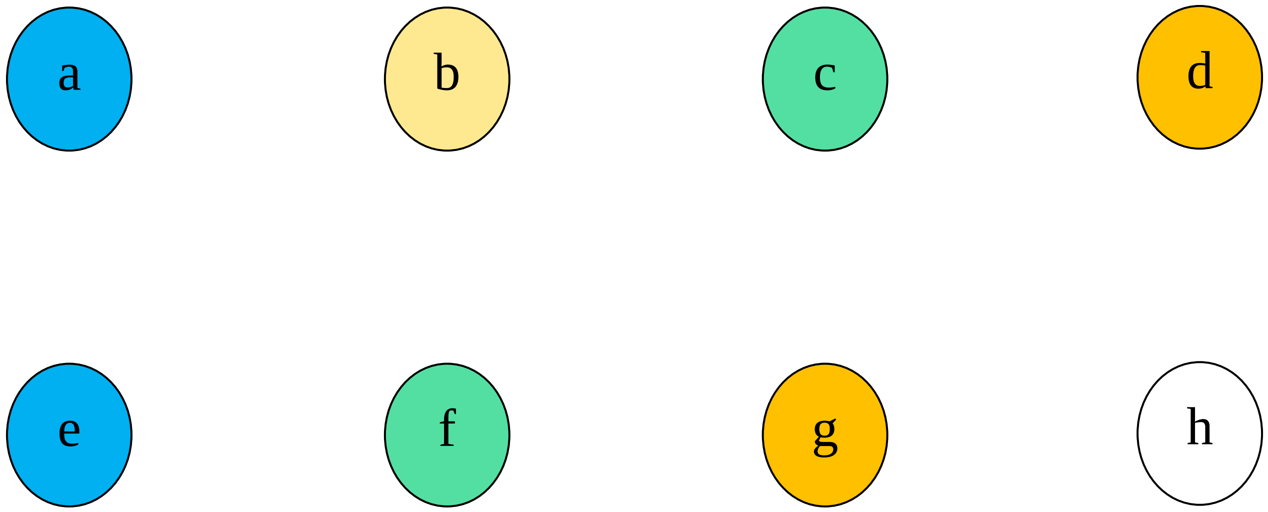
Tant que $I \neq \emptyset$ **faire**

$N_k = \{i \in I: \boldsymbol{V^-_{G(I)}(i)} = \emptyset\}$

$I = I - N_k$

$k + 1$

Fin tant que



$N_3 = \{i \in I: V^-_{G(I)}(i) = \emptyset\} = \{d, g\}$

$I = \{a, e, b, c, f, d, g\}$

$G(I) = (S - I, A - \{(a,b), (a,f), (e,b), (e,f), (b,c), (b,f), (b,g), (c,d), (c,g), (f,g), (d,h), (g,h)\})$

Hiérarchie – Exemple

$I = S, k = 0$

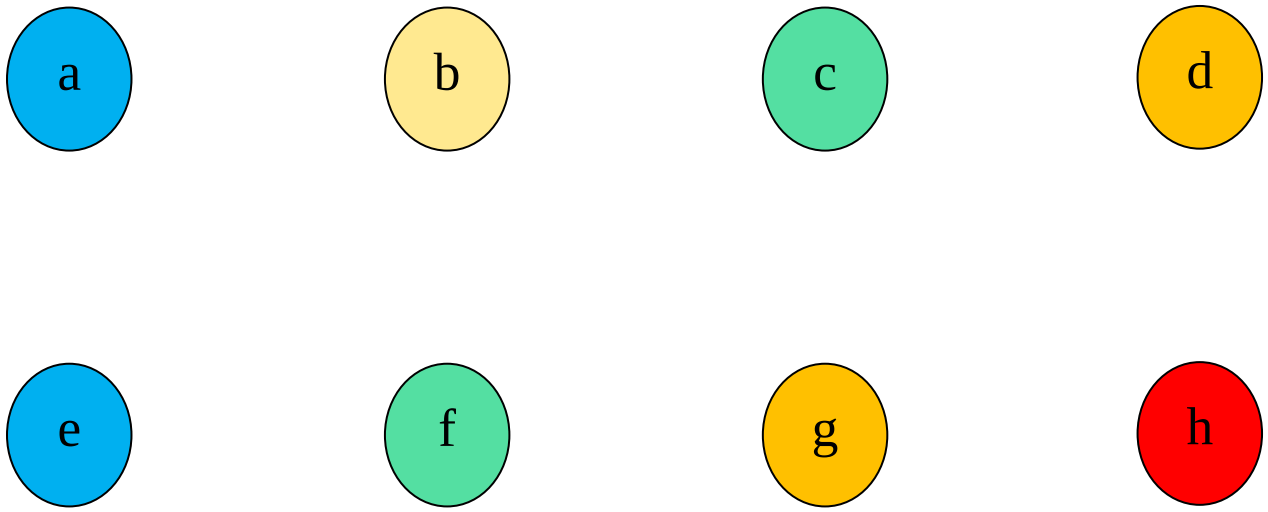
Tant que $I \neq \emptyset$ **faire**

$$N_k = \{i \in I : \boldsymbol{V^-_{G(I)}(i)} = \emptyset\}$$

$$I = I - N_k$$

$$k + 1$$

Fin tant que



$N_4 = \{i \in I : V^-_{G(I)}(i) = \emptyset\} = \{h\}$

$I = \{a, e, b, c, f, d, g, h\}$

$G(I) = (S - I, A - \{(a,b),(a,f),(e,b),(e,f),(b,c),(b,f),(b,g),(c,d),(c,g),(f,g),(d,h),(g,h)\})$

Hiérarchie – Exemple

$I = S, k = 0$

Tant que $I \neq \emptyset$ **faire**

$$N_k = \left\{ i \in I : \boldsymbol{V^{-}_{G(I)}(i)} = \emptyset \right\}$$

$$I = I - N_k$$

$$k + 1$$

Fin tant que