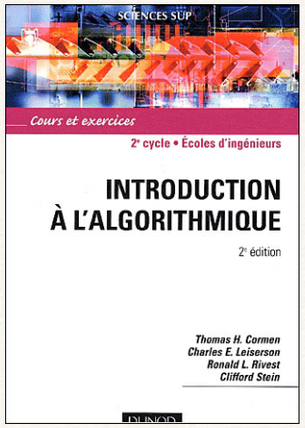


Algorithmique des graphes

Algorithmes élémentaires pour les graphes

Responsable du cours:
Lélia Blin
Université d'Evry
Licence 3 année.

Plan du cours

- Parcours en largeur d'un graphe
- Parcours en Profondeur d'un graphe
- Tri topologique
- Composantes fortement connexes

Blin lélia

Parcours en largeur d'un graphe

Algorithmique des graphes

Parcours en largeur

- Le ***parcours en largeur*** est l'un des algorithmes de parcours les plus simple.
- Il est à la base de nombreux algorithmes importants sur les graphes:
- L'algorithme de **Dijkstra**
  - Calcul des plus courts chemins à origine unique

Parcours en largeur

- Étant donnés un graphe $G = (S, A)$ et un sommet ***origine*** s.
- Le parcours en largeur (PL) emprunte systématiquement les arcs de G pour « découvrir » tous les sommets accessibles depuis s.
- PL calcule la plus petite distance entre chacun des sommet et s l'origine du PL.
- PL construit une « **arborescence de parcours en largeur** »

Parcours en largeur

- Pour tout sommet v accessible depuis s:
  - le chemin reliant s à v dans l'arborescence de parcours en largeur correspond à un « plus court chemin » de s vers v.
- L'algorithme de parcours en largeur tient son nom du fait qu'il:
  - Découvre d'abord tous les sommets situés à une distance k de s
  - Avant de découvrir tout sommet situé à la distance $k + 1$.
  - Ect…

Algorithme de PL

- L'algorithme que nous allons voir fonctionne aussi bien:
  - Sur les graphes orientés.
  - Sur les graphes non orientés.

Principe de PL

- L'algorithme construit une arborescence ayant comme racine le sommet d'origine s.
- A chaque étape un sommet déjà dans l'arborescence
  - Intègre ces voisins qui ne sont pas dans l'arborescence
  - Et ceci en maintenant un plus court chemin.

Implémentation de PL

- L'algorithme colorie chaque sommet en trois couleurs dans l'ordre:
  - blanc (sommet non découvert),
  - gris (sommet découvert dont certains voisins ne le sont pas encore),
  - noir (sommet découvert dont tous les voisin sont découverts).
- Tous les sommets sont blancs au départ, et deviennent gris, puis noirs.

Implémentation de PL

- Quand un sommet est ***découvert*** (*rencontré pour la première fois)*
  - Il perd sa couleur blanche.
- Les sommets gris ou noirs ont donc été découverts.

Implémentation de PL

- La distinction entre gris et noir permet de garantir que la recherche se poursuit *en largeur d'abord*.
- Les sommets gris peuvent avoir représentent la frontière entre les sommets découverts et les sommets non découverts.
- Les sommets noirs sont des sommets dont tous les voisins ont été découvert.

Algorithme PL: variables

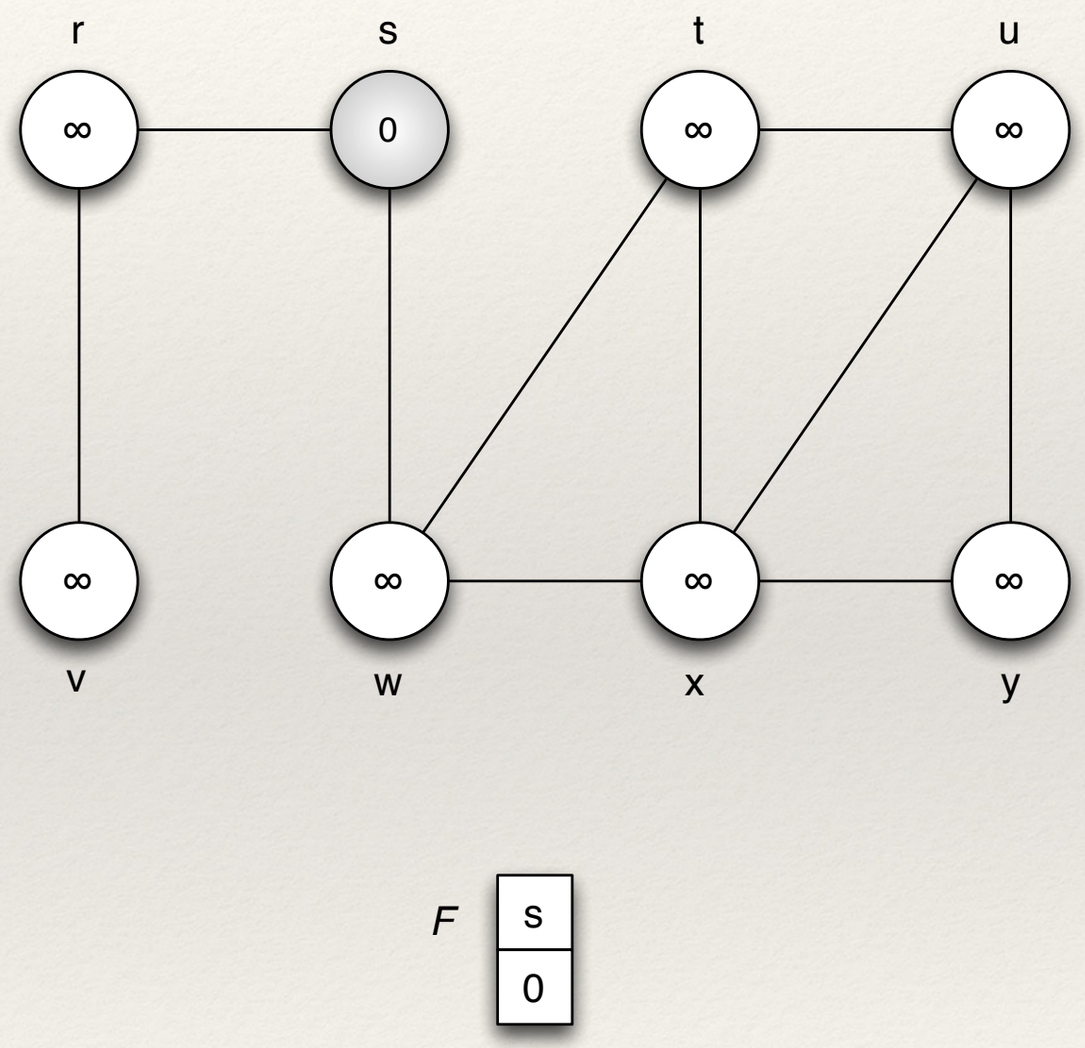
- Le graphe d'entrée $G = (S, A)$ est représenté par des **listes d'adjacences**.
- ***couleur*[*u*]:** La couleur de chaque sommet $u \in S$
- **π[*u*]:** Le parent de chaque sommet $u \in S$
  - Si u n'a pas de parent alors $\pi[u] = \text{NIL}$.
- ***d*[*u*]:** La distance calculée par l'algorithme entre l'origine s et le sommet u.
- L'algorithme a également recours à une file fifo ***F*** pour gérer l'ensemble des sommets gris.

Algorithme PL: pseudo-code

```
PL(G, s)
pour chaque sommet u ∈ S[G] − {s}
    faire couleur[u] ← BLANC
          d[u] ← ∞
          π[u] ← NIL
couleur[s] ← GRIS
d[s] ← 0
π[s] ← NIL
F ← {s}
tant que F ≠ ∅
    faire u ← tête[F]
          pour chaque v ∈ Adj[u]
              faire si couleur[v] = BLANC
                     alors couleur[v] ← GRIS
                           d[v] ← d[u] + 1
                           π[v] ← u
                           ENFILE(F, v)
                           DÉFILE(F)
          couleur[u] ← NOIR
```

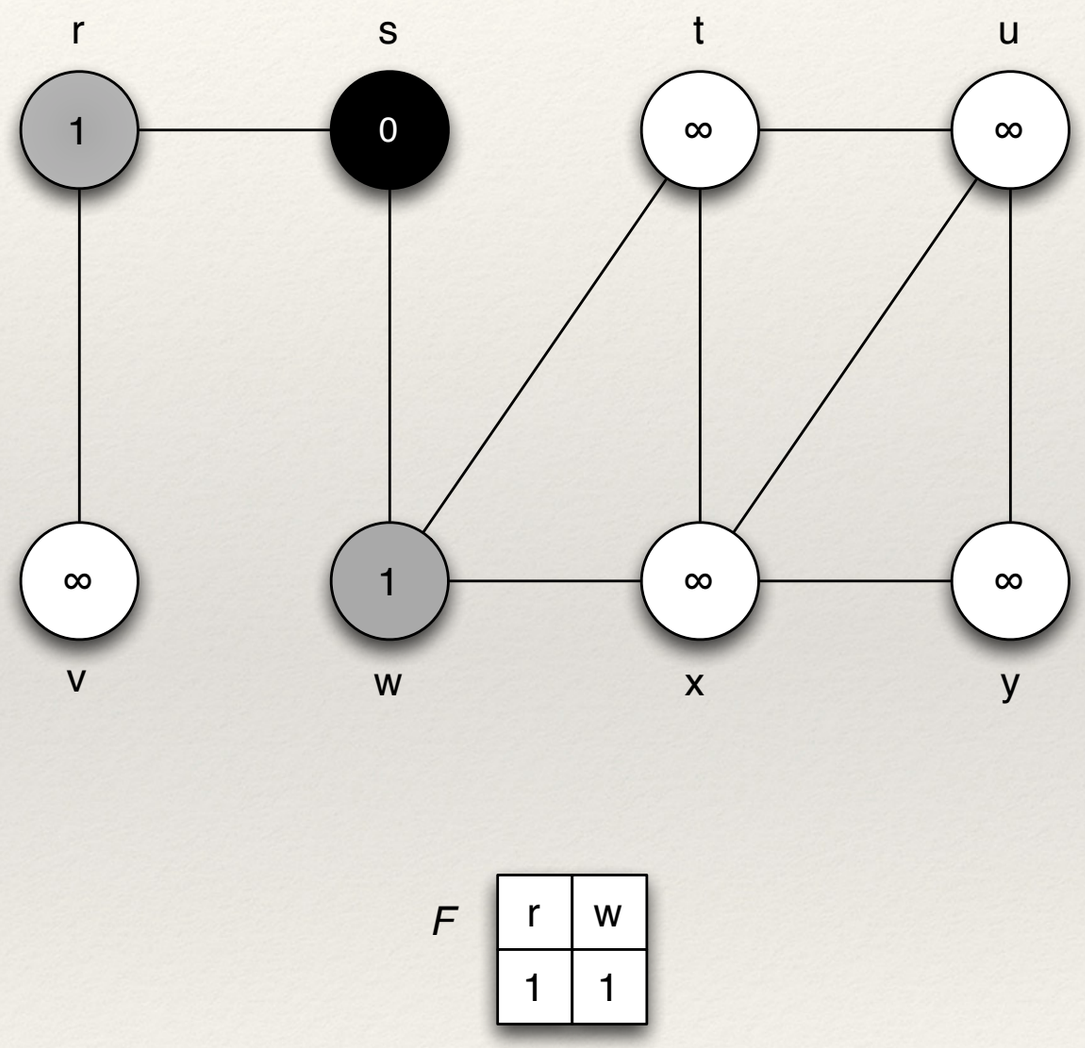
Parcours en largeur: Exemple

```
PL(G, s)
pour chaque sommet u ∈ S[G] − {s}
    faire couleur[u] ← BLANC
          d[u] ← ∞
          π[u] ← NIL
couleur[s] ← GRIS
d[s] ← 0
π[s] ← NIL
F ← {s}
tant que F ≠ ∅
    faire u ← tête[F]
          pour chaque v ∈ Adj[u]
              faire si couleur[v] = BLANC
                     alors couleur[v] ← GRIS
                           d[v] ← d[u] + 1
                           π[v] ← u
                           ENFILE(F, v)
                           DÉFILE(F)
          couleur[u] ← NOIR
```



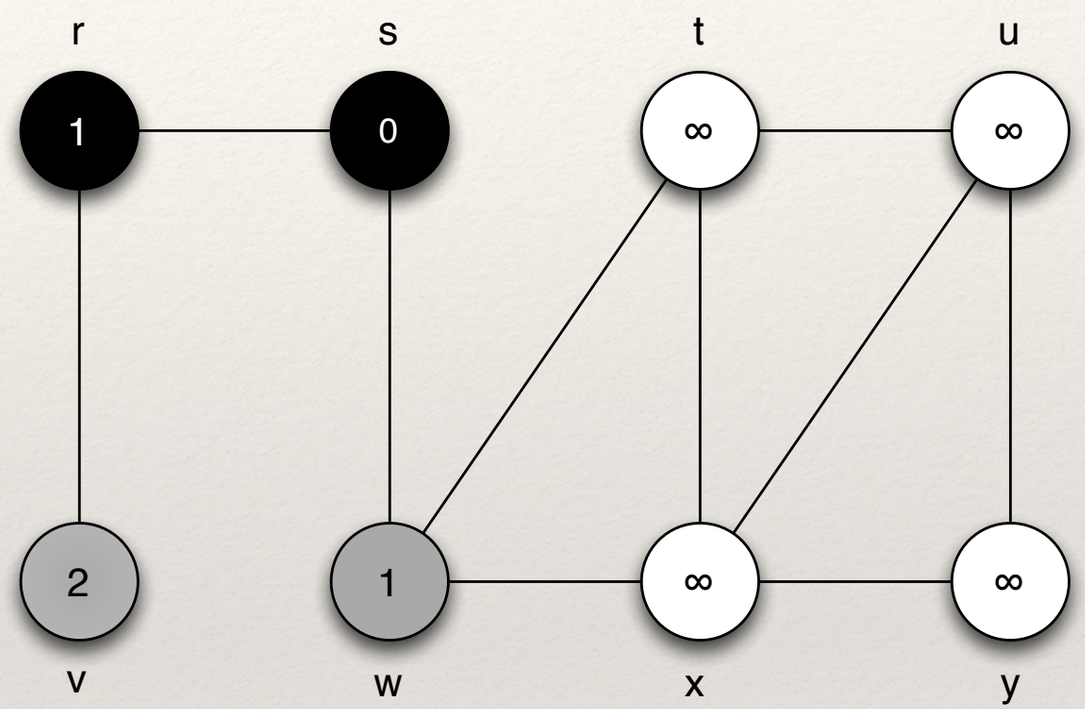
Parcours en largeur: Exemple

```
PL(G, s)
pour chaque sommet u ∈ S[G] − {s}
    faire couleur[u] ← BLANC
          d[u] ← ∞
          π[u] ← NIL
couleur[s] ← GRIS
d[s] ← 0
π[s] ← NIL
F ← {s}
tant que F ≠ ∅
    faire u ← tête[F]
          pour chaque v ∈ Adj[u]
              faire si couleur[v] = BLANC
                     alors couleur[v] ← GRIS
                           d[v] ← d[u] + 1
                           π[v] ← u
                           ENFILE(F, v)
                           DÉFILE(F)
          couleur[u] ← NOIR
```



Parcours en largeur: Exemple

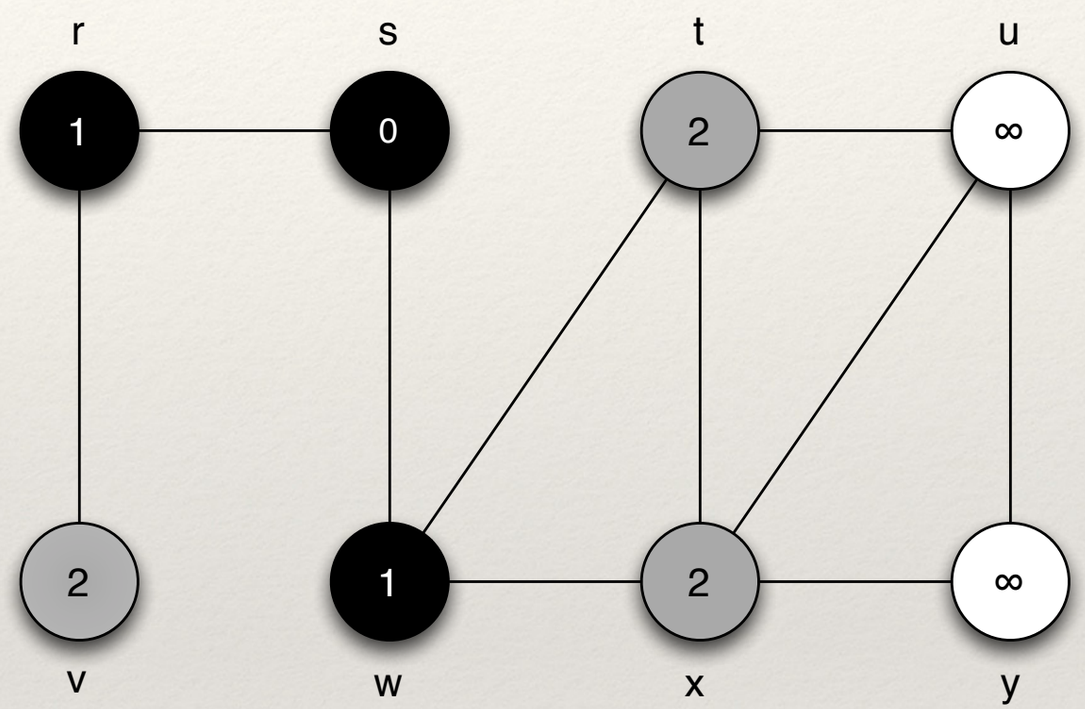
```
PL(G, s)
pour chaque sommet u ∈ S[G] − {s}
    faire couleur[u] ← BLANC
          d[u] ← ∞
          π[u] ← NIL
couleur[s] ← GRIS
d[s] ← 0
π[s] ← NIL
F ← {s}
tant que F ≠ ∅
    faire u ← tête[F]
          pour chaque v ∈ Adj[u]
              faire si couleur[v] = BLANC
                     alors couleur[v] ← GRIS
                           d[v] ← d[u] + 1
                           π[v] ← u
                           ENFILE(F, v)
                           DÉFILE(F)
          couleur[u] ← NOIR
```



F	w	v
	1	2

Parcours en largeur: Exemple

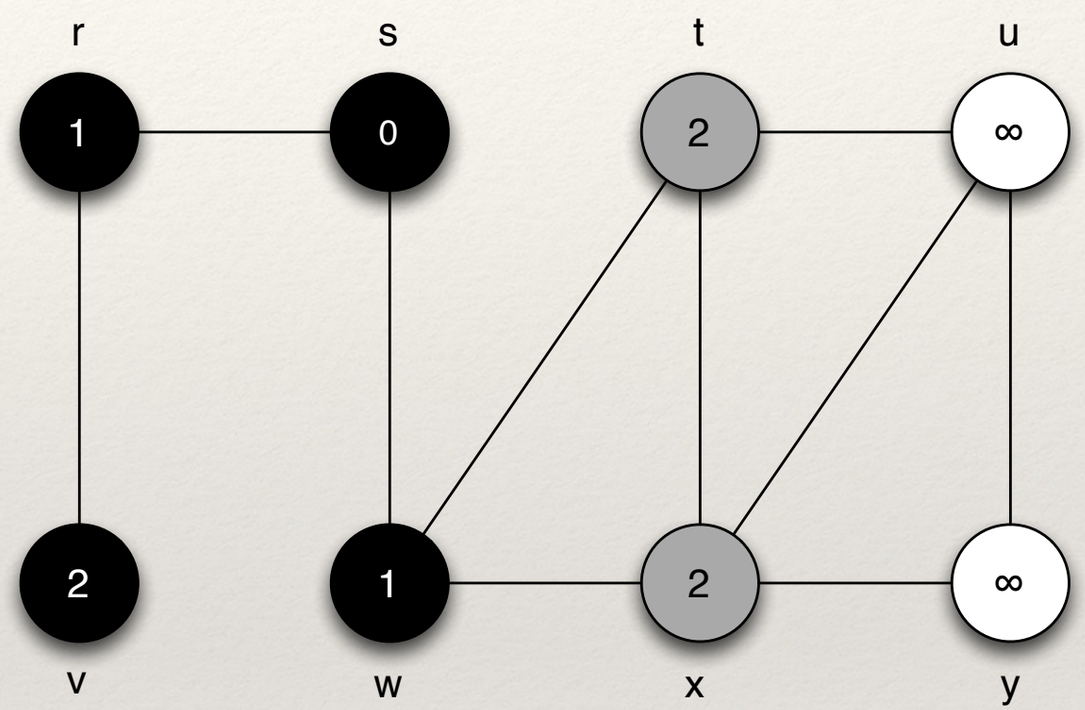
```
PL(G, s)
pour chaque sommet u ∈ S[G] − {s}
    faire couleur[u] ← BLANC
          d[u] ← ∞
          π[u] ← NIL
couleur[s] ← GRIS
d[s] ← 0
π[s] ← NIL
F ← {s}
tant que F ≠ ∅
    faire u ← tête[F]
          pour chaque v ∈ Adj[u]
              faire si couleur[v] = BLANC
                     alors couleur[v] ← GRIS
                           d[v] ← d[u] + 1
                           π[v] ← u
                           ENFILE(F, v)
                           DÉFILE(F)
          couleur[u] ← NOIR
```



F	v	t	x
	2	2	2

Parcours en largeur: Exemple

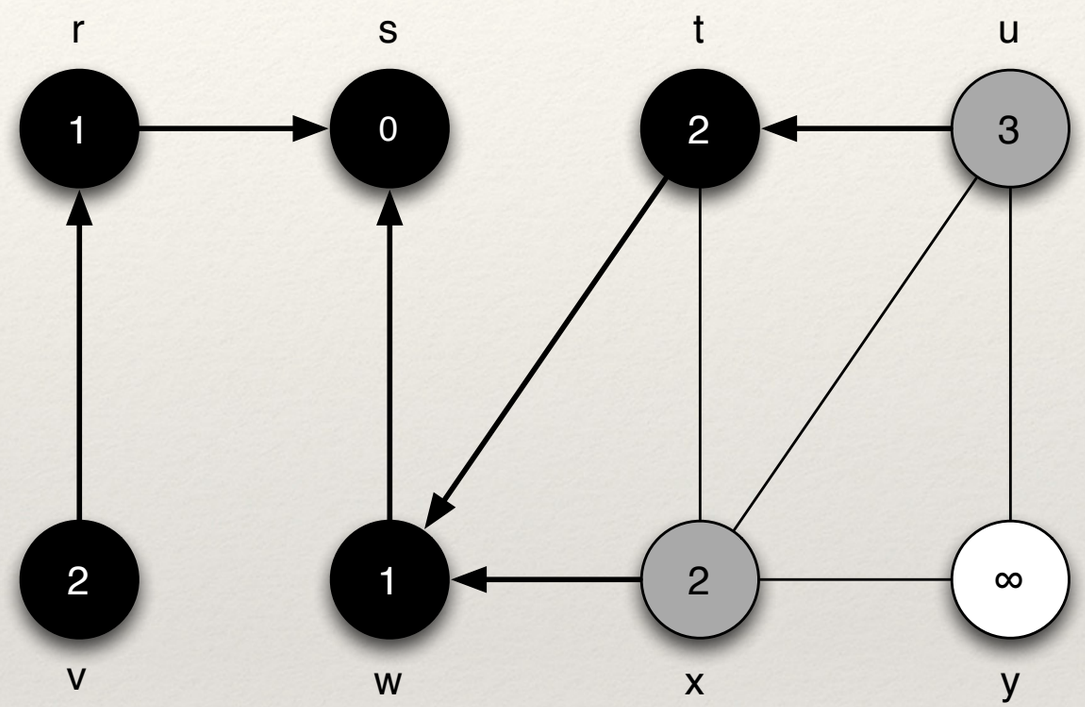
```
PL(G, s)
pour chaque sommet u ∈ S[G] − {s}
    faire couleur[u] ← BLANC
          d[u] ← ∞
          π[u] ← NIL
couleur[s] ← GRIS
d[s] ← 0
π[s] ← NIL
F ← {s}
tant que F ≠ ∅
    faire u ← tête[F]
          pour chaque v ∈ Adj[u]
              faire si couleur[v] = BLANC
                      alors couleur[v] ← GRIS
                            d[v] ← d[u] + 1
                            π[v] ← u
                            ENFILE(F, v)
                            DÉFILE(F)
          couleur[u] ← NOIR
```



F	t	x
	2	2

Parcours en largeur: Exemple

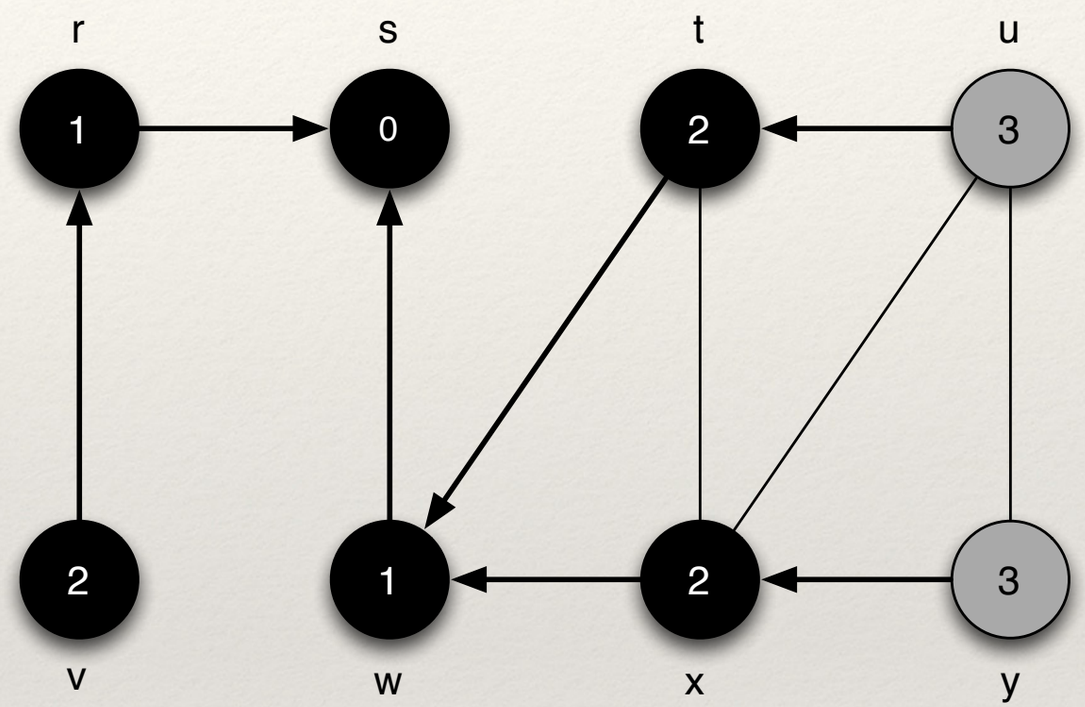
```
PL(G, s)
pour chaque sommet u ∈ S[G] − {s}
    faire couleur[u] ← BLANC
          d[u] ← ∞
          π[u] ← NIL
couleur[s] ← GRIS
d[s] ← 0
π[s] ← NIL
F ← {s}
tant que F ≠ ∅
    faire u ← tête[F]
          pour chaque v ∈ Adj[u]
              faire si couleur[v] = BLANC
                     alors couleur[v] ← GRIS
                           d[v] ← d[u] + 1
                           π[v] ← u
                           ENFILE(F, v)
                           DÉFILE(F)
          couleur[u] ← NOIR
```



F	x	u
	2	3

Parcours en largeur: Exemple

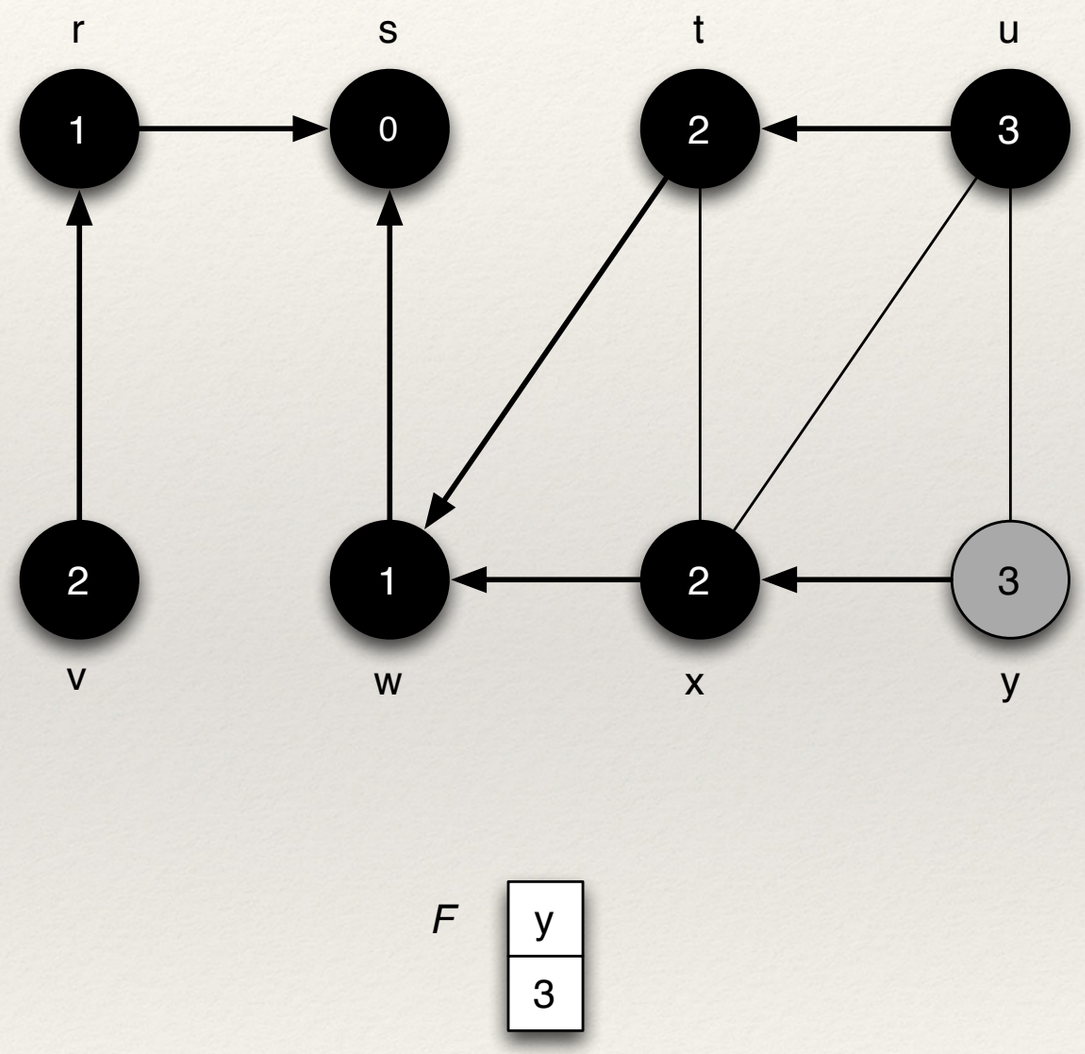
```
PL(G, s)
pour chaque sommet u ∈ S[G] − {s}
    faire couleur[u] ← BLANC
          d[u] ← ∞
          π[u] ← NIL
couleur[s] ← GRIS
d[s] ← 0
π[s] ← NIL
F ← {s}
tant que F ≠ ∅
    faire u ← tête[F]
          pour chaque v ∈ Adj[u]
              faire si couleur[v] = BLANC
                     alors couleur[v] ← GRIS
                           d[v] ← d[u] + 1
                           π[v] ← u
                           ENFILE(F, v)
                           DÉFILE(F)
          couleur[u] ← NOIR
```



F	u	y
	3	3

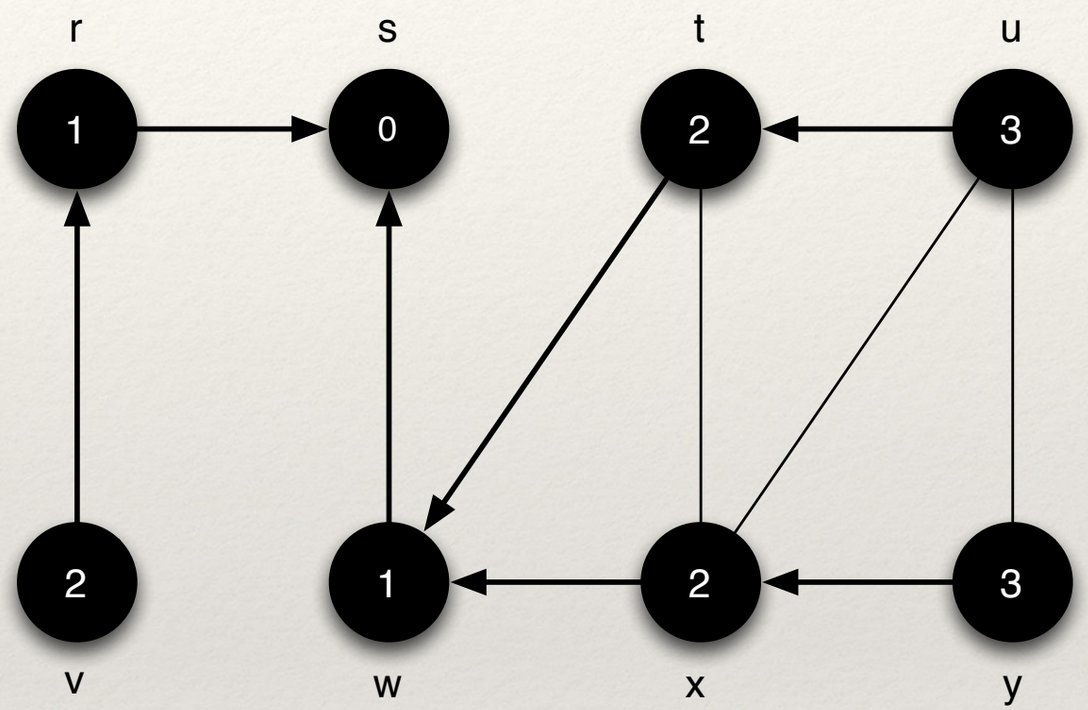
Parcours en largeur: Exemple

```
PL(G, s)
pour chaque sommet u ∈ S[G] − {s}
    faire couleur[u] ← BLANC
          d[u] ← ∞
          π[u] ← NIL
couleur[s] ← GRIS
d[s] ← 0
π[s] ← NIL
F ← {s}
tant que F ≠ ∅
    faire u ← tête[F]
          pour chaque v ∈ Adj[u]
              faire si couleur[v] = BLANC
                     alors couleur[v] ← GRIS
                           d[v] ← d[u] + 1
                           π[v] ← u
                           ENFILE(F, v)
                           DÉFILE(F)
          couleur[u] ← NOIR
```



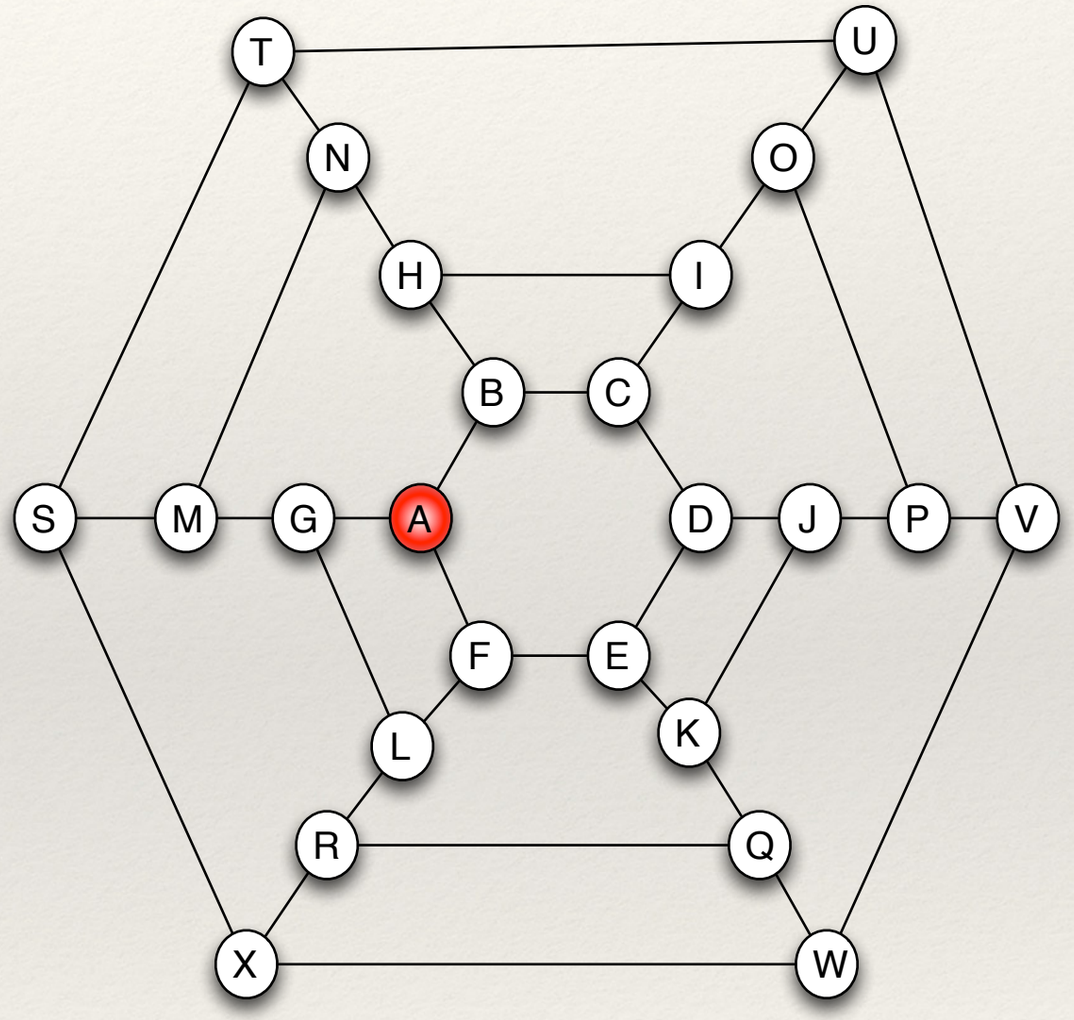
Parcours en largeur: Exemple

```
PL(G, s)
pour chaque sommet u ∈ S[G] − {s}
    faire couleur[u] ← BLANC
          d[u] ← ∞
          π[u] ← NIL
couleur[s] ← GRIS
d[s] ← 0
π[s] ← NIL
F ← {s}
tant que F ≠ ∅
    faire u ← tête[F]
          pour chaque v ∈ Adj[u]
              faire si couleur[v] = BLANC
                      alors couleur[v] ← GRIS
                            d[v] ← d[u] + 1
                            π[v] ← u
                            ENFILE(F, v)
                            DÉFILE(F)
          couleur[u] ← NOIR
```



Parcours en largeur: Autre exemple

```
PL(G, s)
pour chaque sommet u ∈ S[G] − {s}
    faire couleur[u] ← BLANC
          d[u] ← ∞
          π[u] ← NIL
couleur[s] ← GRIS
d[s] ← 0
π[s] ← NIL
F ← {s}
tant que F ≠ ∅
    faire u ← tête[F]
          pour chaque v ∈ Adj[u]
              faire si couleur[v] = BLANC
                     alors couleur[v] ← GRIS
                           d[v] ← d[u] + 1
                           π[v] ← u
                           ENFILE(F, v)
                           DÉFILE(F)
          couleur[u] ← NOIR
```



PL Complexité en temps

```
PL(G, s)
1   pour chaque sommet u ∈ S[G] − {s}
2       faire couleur[u] ← BLANC
3             d[u] ← ∞
4             π[u] ← NIL
5   couleur[s] ← GRIS
6   d[s] ← 0
7   π[s] ← NIL
8   F ← {s}
9   tant que F ≠ ∅
10      faire u ← tête[F]
11            pour chaque v ∈ Adj[u]
12                faire si couleur[v] = BLANC
13                        alors couleur[v] ← GRIS
14                              d[v] ← d[u] + 1
15                              π[v] ← u
16                              ENFILE(F, v)
17                              DÉFILE(F)
18            couleur[u] ← NOIR
```

- Le test de la ligne 12 garantit que:
  - Chaque sommet est enfilé au plus une fois
  - Et par conséquence défilé au plus une fois.
- Les opérations d'enfilement et de défilement sont en $O(1)$,
- le temps total des opérations de file est $O(|A|)$ pour l'ensemble des sommets.

PL Complexité en temps

- Le coût de l'initialisation est $O(|S|)$
- Le temps d'exécution total de PL est donc $O(|S| + |A|)$.
- Conclusion:
  - Le parcours en largeur s'exécute en un temps qui est linéaire par rapport à la taille de la représentation par listes d'adjacences de G.

Distance

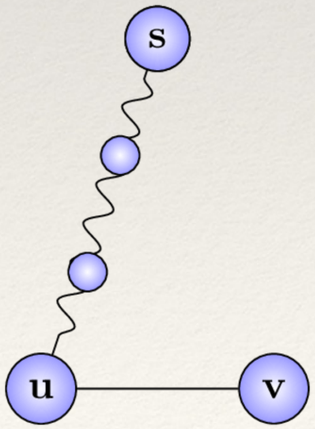
- On définit la ***distance*** d(s, v) de s à v comme étant le nombre minimal d'arêtes (d'arcs) d'un chemin reliant le sommet s au sommet v.
- Si il n'existe aucun chemin de s à v, d(s, v)=∞.

Lemme 1

- Soit G = (S, A) un graphe et soit s ∈ S un sommet arbitraire. Alors, pour toute arête (u, v) ∈ A, on a:
  - $d(s,v) \leq d(s,u) + 1$

Preuve lemme 1

- Si u est accessible depuis s, alors v l'est aussi.
- Dans ce cas, la distance de s à v ne peut pas être plus grande que distance de s à u prolongé par l'arête (u, v), et l'inégalité est donc valable.
- Si u ne peut pas être atteint à partir de s, alors $\mathrm{d}(s, u) = \infty$, et l'inégalité est vérifiée.



Lemme 2

- Soit G = (S, A) un graphe orienté ou non ; on suppose que PL est exécutée sur G à partir d'une origine donnée $s \in S$.
- Alors, quand PL se termine, pour tout sommet $v \in S$, la valeur d[v] calculée par PL vérifie l'inégalité
  - $d[v] \geq d(s, v)$.

Preuve

```
PL(G, s)
 1  pour chaque sommet u ∈ S[G] − {s}
 2      faire couleur[u] ← BLANC
 3            d[u] ← ∞
 4            π[u] ← NIL
 5  couleur[s] ← GRIS
 6  d[s] ← 0
 7  π[s] ← NIL
 8  F ← {s}
 9  tant que F ≠ ∅
10      faire u ← tête[F]
11            pour chaque v ∈ Adj[u]
12                faire si couleur[v] = BLANC
13                        alors couleur[v] ← GRIS
14                              d[v] ← d[u] + 1
15                              π[v] ← u
16                              ENFILE(F, v)
17                              DÉFILE(F)
18            couleur[u] ← NOIR
```

- On utilise une récurrence sur le nombre total de sommets insérés dans la file F.
- Notre hypothèse de récurrence est que d[v] ≥ d(s, v) pour tout v ∈ S.
- La base de la récurrence se situe juste après que s a été placé dans F à la ligne 9 de PL.
- L'hypothèse de récurrence est vérifiée ici, car
  - d[s] = 0 = d(s, s) et
  - d[v] = ∞ ≥ d(s, v)
  - pour tout v ∈ S − {s}.

Preuve

```
PL(G, s)
1   pour chaque sommet u ∈ S[G] − {s}
2       faire couleur[u] ← BLANC
3             d[u] ← ∞
4             π[u] ← NIL
5   couleur[s] ← GRIS
6   d[s] ← 0
7   π[s] ← NIL
8   F ← {s}
9   tant que F ≠ ∅
10      faire u ← tête[F]
11            pour chaque v ∈ Adj[u]
12                faire si couleur[v] = BLANC
13                        alors couleur[v] ← GRIS
14                              d[v] ← d[u] + 1
15                              π[v] ← u
16                              ENFILE(F, v)
17                              DÉFILE(F)
18            couleur[u] ← NOIR
```

- Pour l'étape de récurrence, on considère un sommet v blanc qui est découvert pendant le balayage des successeurs d'un sommet u.
- L'hypothèse de récurrence implique que $d[u] \geq d(s, u)$. Une fois effectuée l'affectation en ligne 15 et d'après le lemme 1, on obtient
  - d[v]=d[u]+1
  - d[v]≥d[s,v]+1
  - d[v]≥d[s,v]

Preuve

```
PL(G, s)
 1  pour chaque sommet u ∈ S[G] − {s}
 2      faire couleur[u] ← BLANC
 3            d[u] ← ∞
 4            π[u] ← NIL
 5  couleur[s] ← GRIS
 6  d[s] ← 0
 7  π[s] ← NIL
 8  F ← {s}
 9  tant que F ≠ ∅
10      faire u ← tête[F]
11            pour chaque v ∈ Adj[u]
12                faire si couleur[v] = BLANC
13                        alors couleur[v] ← GRIS
14                              d[v] ← d[u] + 1
15                              π[v] ← u
16                              ENFILE(F, v)
17                              DÉFILE(F)
18            couleur[u] ← NOIR
```

- Le sommet v est donc inséré dans la file F, et il ne sera plus jamais inséré puisqu'il est colorié en gris et que la clause **alors** des lignes 14–17 n'est exécutée que pour les sommets blancs.
- La valeur de $d[v]$ n'est donc plus modifiée, et l'hypothèse de récurrence est maintenue.

Blin lélia

Parcours en profondeur d'un graphe

Algorithmique des graphes

Parcours en profondeur

- La stratégie suivie par un parcours en profondeur est de descendre plus « profondément » dans le graphe chaque fois que c'est possible.

Parcours en profondeur

- Les arêtes sont explorées à partir du sommet *origine*
- Lorsque tous les arcs d'un sommet v ont été explorés,
  - L'algorithme « revient en arrière » pour explorer les arêtes qui partent du sommet à partir duquel v a été découvert.
  - Ce processus se répète jusqu'à ce que tous les sommets accessibles à partir du sommet origine aient été découverts.

Parcours en profondeur

- Comme dans le parcours en largeur,
  - Les sommets sont coloriés pendant le parcours pour indiquer leur état.
  - Chaque sommet est initialement blanc,
  - Puis gris quand il est ***découvert*** pendant le parcours,
  - Et enfin noirci en fin de traitement, c'est-à-dire quand sa liste d'adjacences a été complètement examinée.
- Cette technique assure que chaque sommet appartient à une arborescence de parcours en profondeur et une seule

Parcours en profondeur

- Le parcours en profondeur ***date*** chaque sommet.
- Chaque sommet v porte deux dates :
  - la première, $d[v]$, marque le moment où v a été découvert pour la première fois et colorié en gris,
  - La seconde, $f[v]$, enregistre le moment ou le parcours a fini d'examiner la liste d'adjacences de v et le colorie en noir.
- Ces dates sont utilisées dans de nombreux algorithmes de graphes et sont utiles pour analyser le comportement du parcours en profondeur.

Parcours en profondeur

- Les dates sont des entiers compris entre 1 et $2+|S|$, puisque découverte et fin de traitement se produisent une fois et une seule pour chacun des $|S|$ sommets.
- Pour tout sommet u *on a* $d[u] < f[u]$

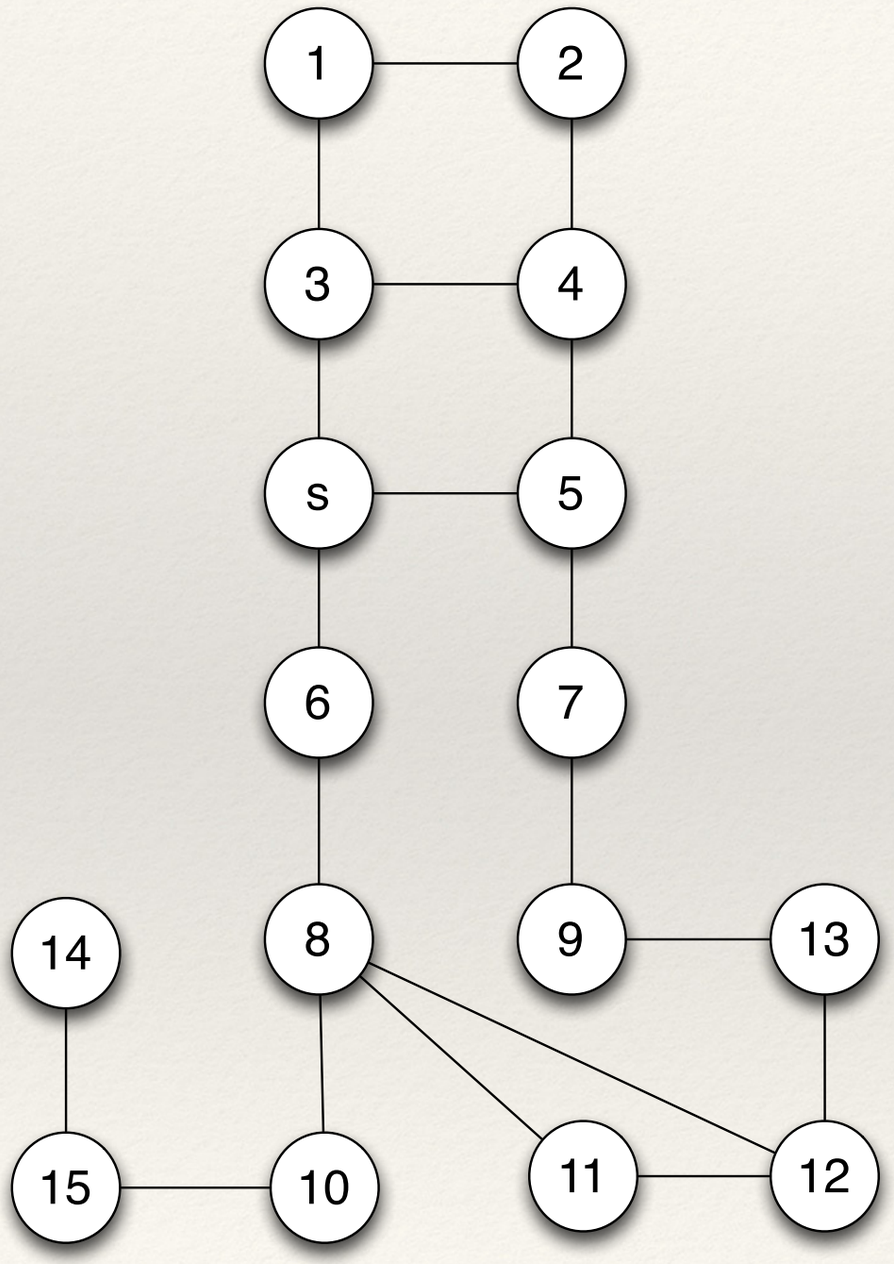
Parcours en profondeur: Pseudo-code

```
PP(G)
1   pour chaque sommet u ∈ S[G]
2       faire couleur[u] ← BLANC
3           π[u] ← NIL
4   date ← 0
5   pour chaque sommet u ∈ S[G]
6       faire si couleur[u] = BLANC
7           alors VISITER-PP(u)
```

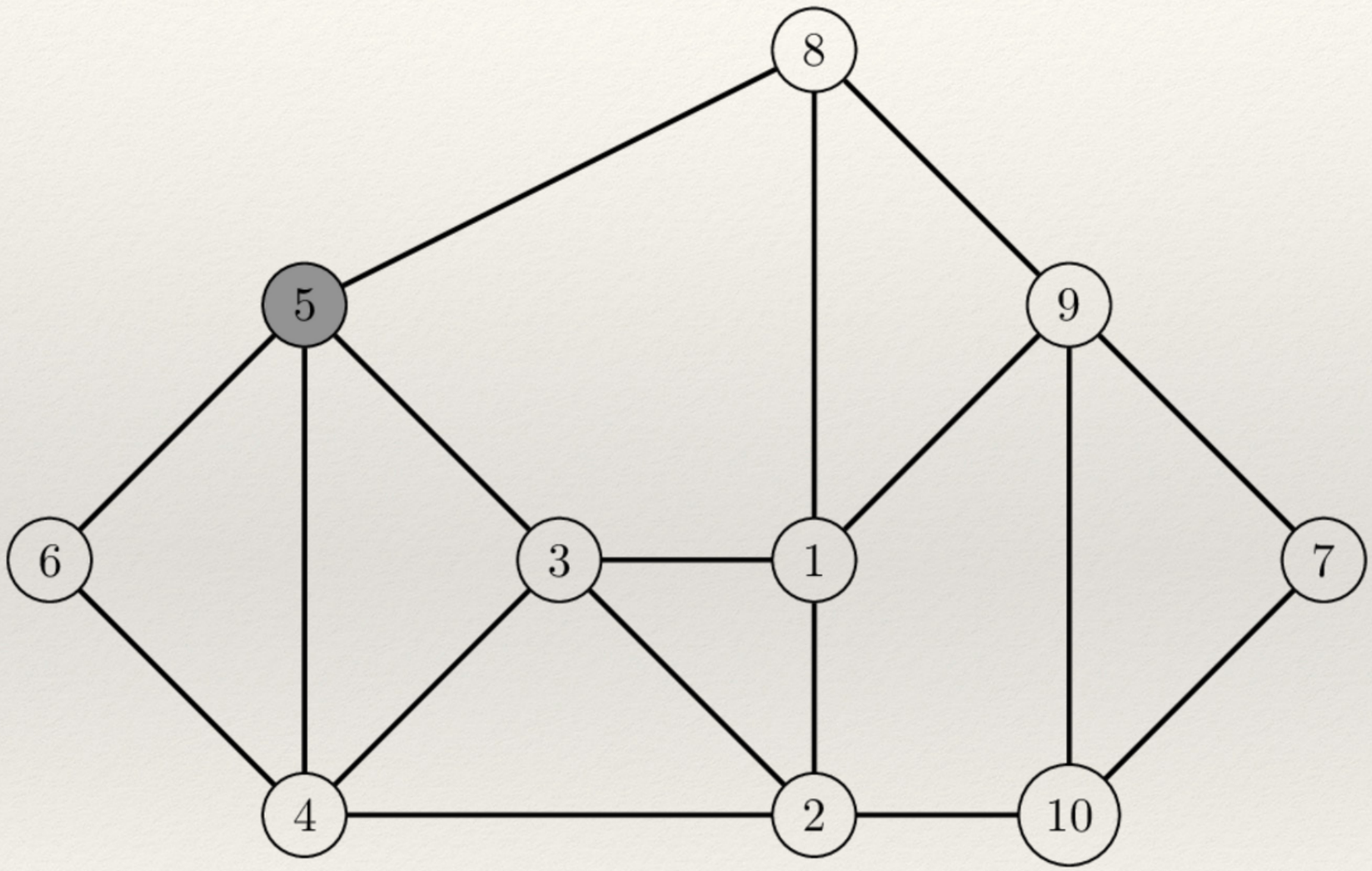
Parcours en profondeur: Pseudo-code

```
VISITER-PP(u)
    couleur[u] ← GRIS ▷ sommet blanc u vient d'être découvert.
    date ← date + 1
    d[u] ← date
    pour chaque v ∈ Adj[u] ▷ Exploration de l'arc (u, v).
        faire si couleur[v] = BLANC
                alors  π[v] ← u
                       VISITER-PP(v)
    couleur[u] ← NOIR ▷ noircir u, car on en a fini avec lui.
    f[u] ← date ← date + 1
```

Parcours en profondeur: exemple



Autre exemple



Blin lélia

Tri topologique

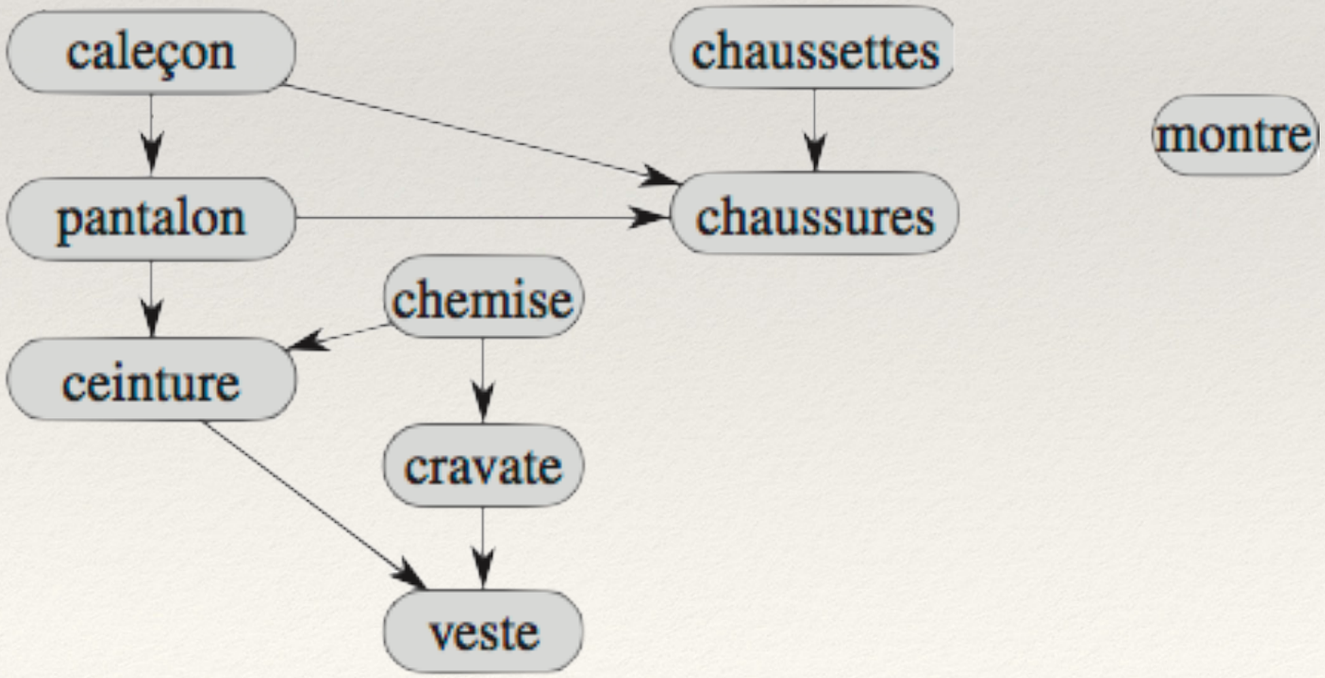
Algorithmique des graphes

Tri topologique

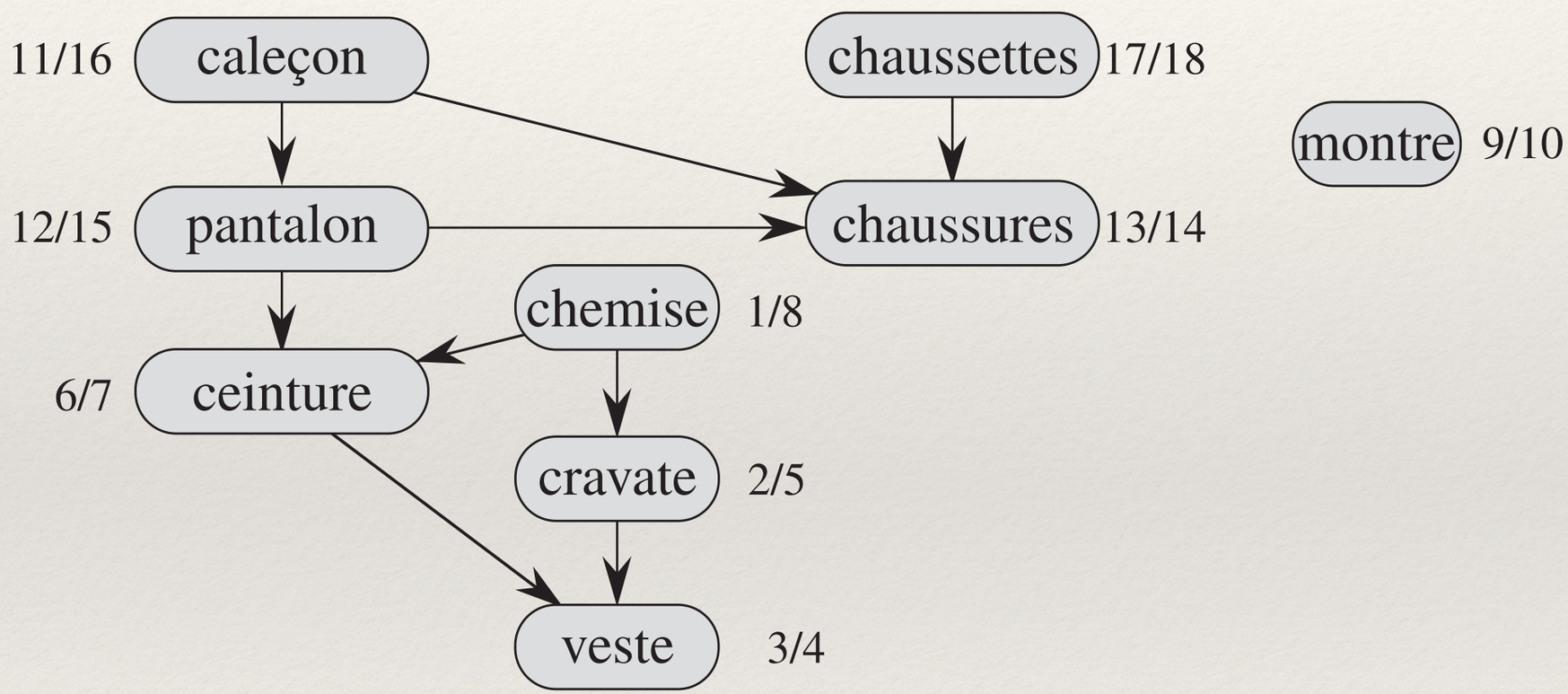
- Le ***tri topologique*** d'un graphe orienté $G = (S, A)$
  - Si le graphe n'a pas de circuit
    - ordonne linéairement tous ses sommets de sorte que si G contient un arc (u, v), u apparaisse avant v dans
  - Si le graphe n'est pas sans circuit,
    - aucun ordre linéaire n'est possible.

Tri topologique: Exemple

- X doit s'habiller le matin.
- Il doit enfiler certains vêtements avant d'autres
  - les chaussettes avant les chaussures.
- D'autres peuvent être mis dans n'importe quel ordre
  - les chaussettes et le pantalon.
- Un arc (u, v) du graphe orienté sans circuit de la figure indique que le vêtement u doit être enfilé avant le vêtement v.

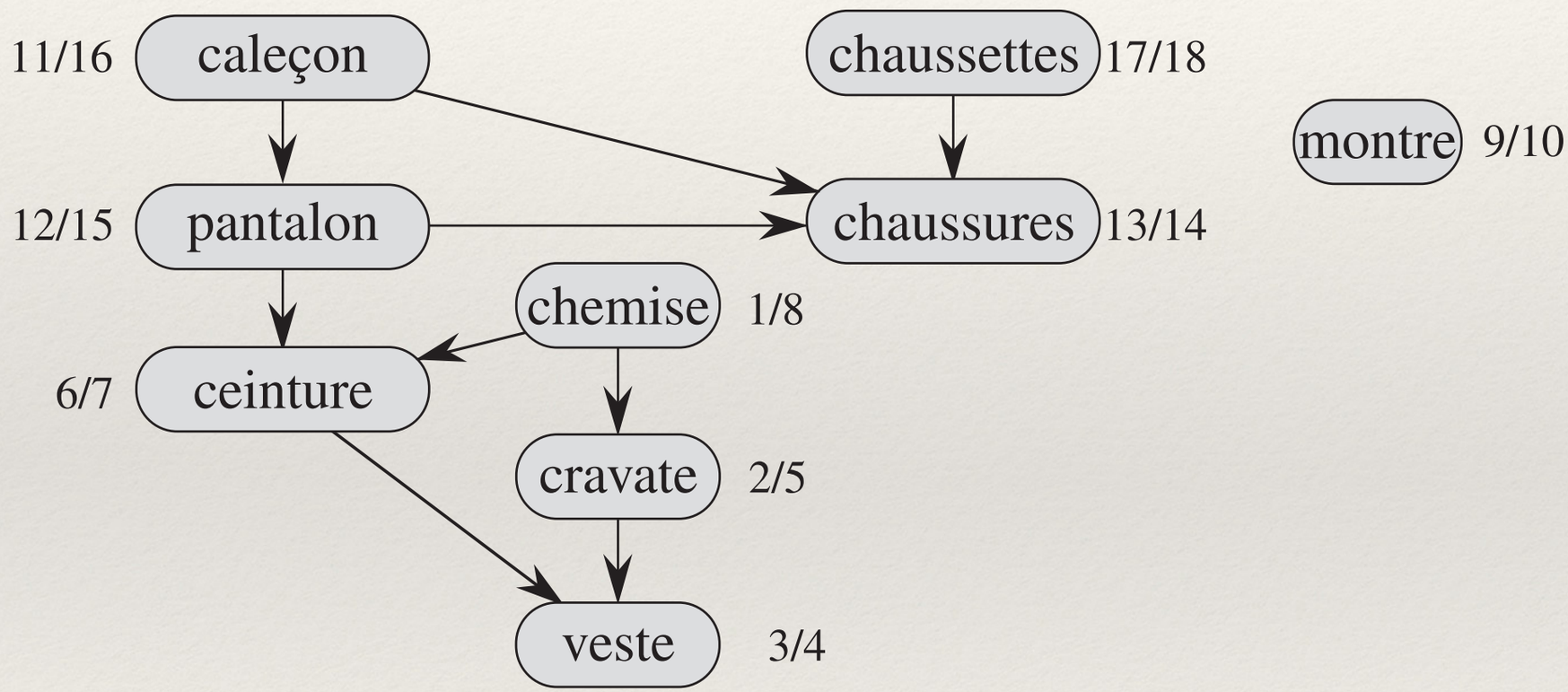


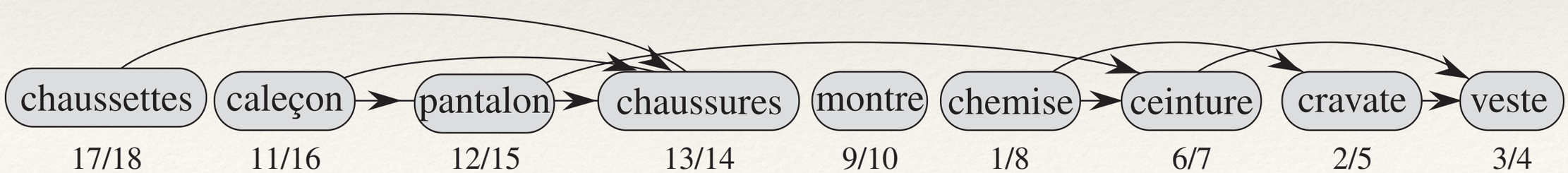
Parcours en profondeur



Tri topologique

- On place les noeud de droite à gauche par date de fin décroissante





Tri topologique: algorithme

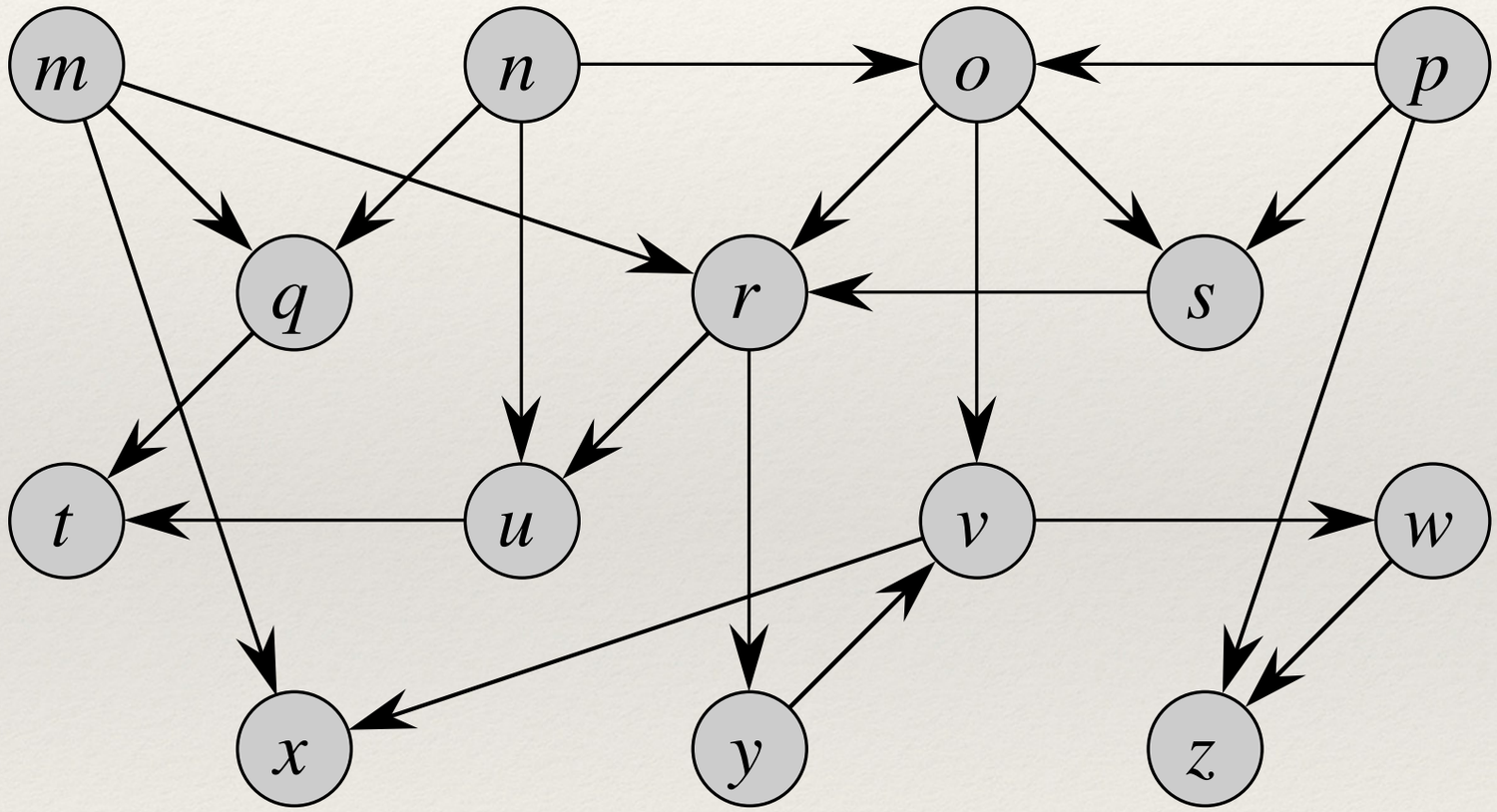
TRI-TOPOLOGIQUE(G)

1 appeler PP(G) pour calculer
les dates de fin de traitement $f[v]$ pour chaque sommet v
2 chaque fois que le traitement d'un sommet se termine,
insérer le sommet début d'une liste chaînée
3 **retourner** la liste chaînée des sommets

Remarque

- Il existe plusieurs parcours en profondeur donc plusieurs tri topologique

Autre exemple



Composantes fortement connexes

Algorithmique des graphes

Lélia Blin

Université d'Evry

Composantes fortement connexes

- Décomposition d'un graphe orienté
  - en composantes fortement connexes.
- Décomposition à l'aide de **deux parcours en profondeur.**
- **Rm: b**eaucoup d'algorithmes de graphe orientés commencent par cette décomposition ;

Transposée

- T***ransposée*** d'un graphe orienté $G = (S,A)$
  - le graphe ${}^{\mathrm{T}}G = (S,{}^{\mathrm{T}}A)$,
  - où ${}^{\mathrm{T}}A = \{(v,u) \in S \times S : (u,v) \in A\}$.
- Autrement dit, ${}^{\mathrm{T}}G$ est obtenu en inversant le sens de tous les arcs de G.

Algorithme

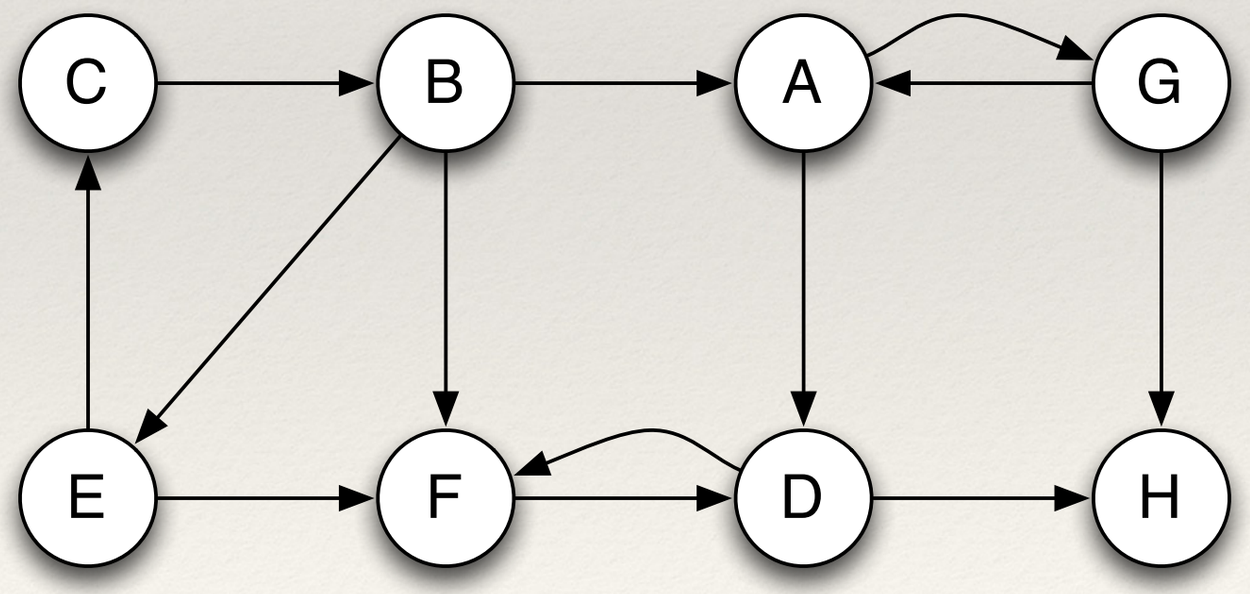
COMPOSANTES-FORTEMENT-CONNEXES(G)

1 appeler PP(G) pour calculer les dates de fin de traitement $f[u]$
  pour chaque sommet u
2 calculer ${}^{\mathrm{T}}G$
3 appeler PP(${}^{\mathrm{T}}G$), mais dans la boucle principale de PP, traiter les sommets
  par ordre de $f[u]$ (calculés en ligne 1) décroissants
4 imprimer les sommets de chaque arborescence de la forêt obtenue
  en ligne 3 en tant que composante fortement connexe distincte

Exemple

COMPOSANTES-FORTEMENT-CONNEXES(G)

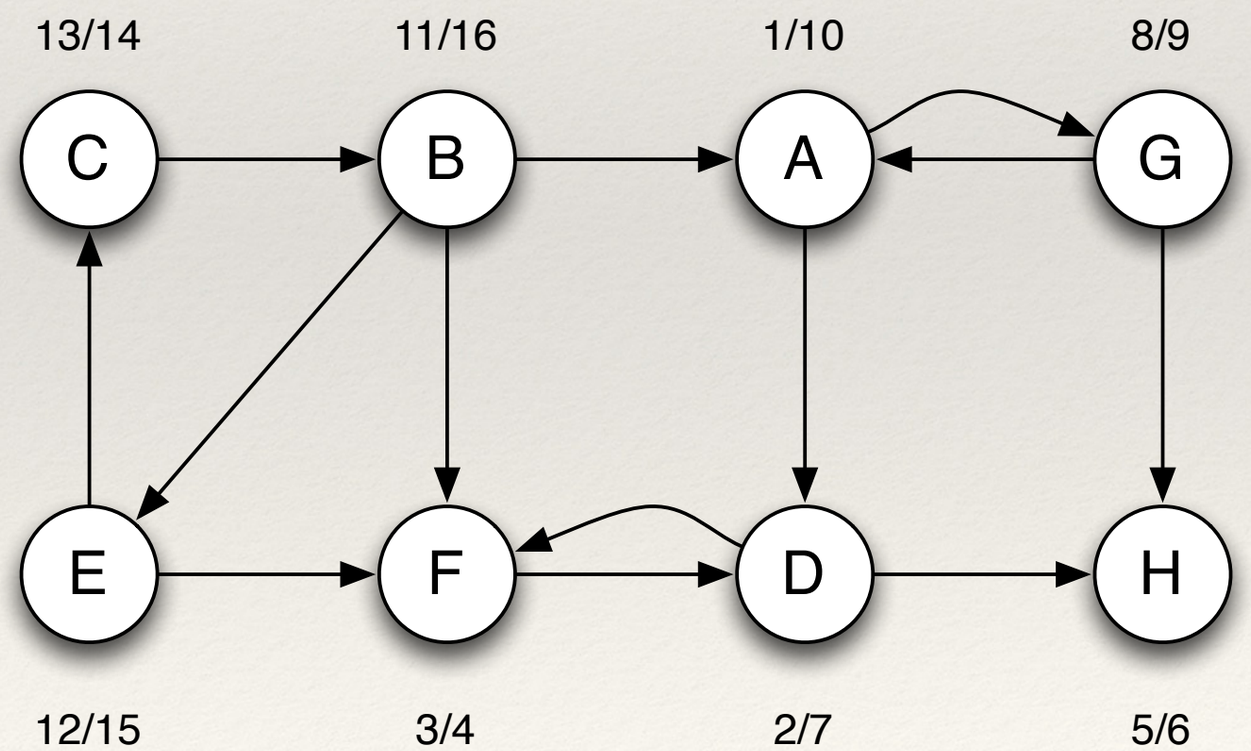
1 appeler PP(G) pour calculer les dates de fin de traitement $f[u]$
  pour chaque sommet u
2 calculer ${}^{\mathrm{T}}G$
3 appeler PP(${}^{\mathrm{T}}G$), mais dans la boucle principale de PP, traiter les sommets
  par ordre de $f[u]$ (calculés en ligne 1) décroissants
4 imprimer les sommets de chaque arborescence de la forêt obtenue
  en ligne 3 en tant que composante fortement connexe distincte



Exemple

COMPOSANTES-FORTEMENT-CONNEXES(G)

1 appeler PP(G) pour calculer les dates de fin de traitement $f[u]$
  pour chaque sommet u
2 calculer ${}^{\mathrm{T}}G$
3 appeler PP(${}^{\mathrm{T}}G$), mais dans la boucle principale de PP, traiter les sommets
  par ordre de $f[u]$ (calculés en ligne 1) décroissants
4 imprimer les sommets de chaque arborescence de la forêt obtenue
  en ligne 3 en tant que composante fortement connexe distincte



Exemple

COMPOSANTES-FORTEMENT-CONNEXES(G)

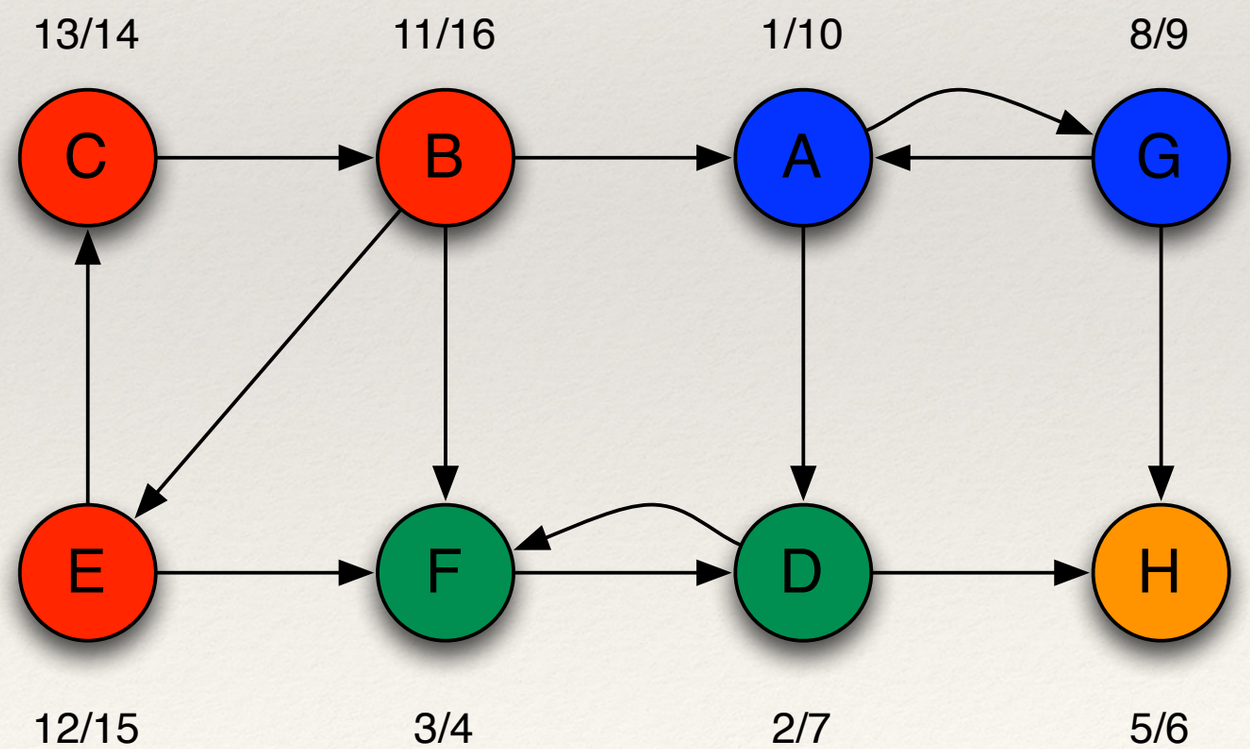
1 appeler PP(G) pour calculer les dates de fin de traitement $f[u]$
pour chaque sommet u

2 calculer ${}^{\mathrm{T}}G$

3 appeler PP(${}^{\mathrm{T}}G$), mais dans la boucle principale de PP, traiter les sommets
par ordre de $f[u]$ (calculés en ligne 1) décroissants

4 imprimer les sommets de chaque arborescence de la forêt obtenue
en ligne 3 en tant que composante fortement connexe distincte

${}^{\mathrm{T}}G$

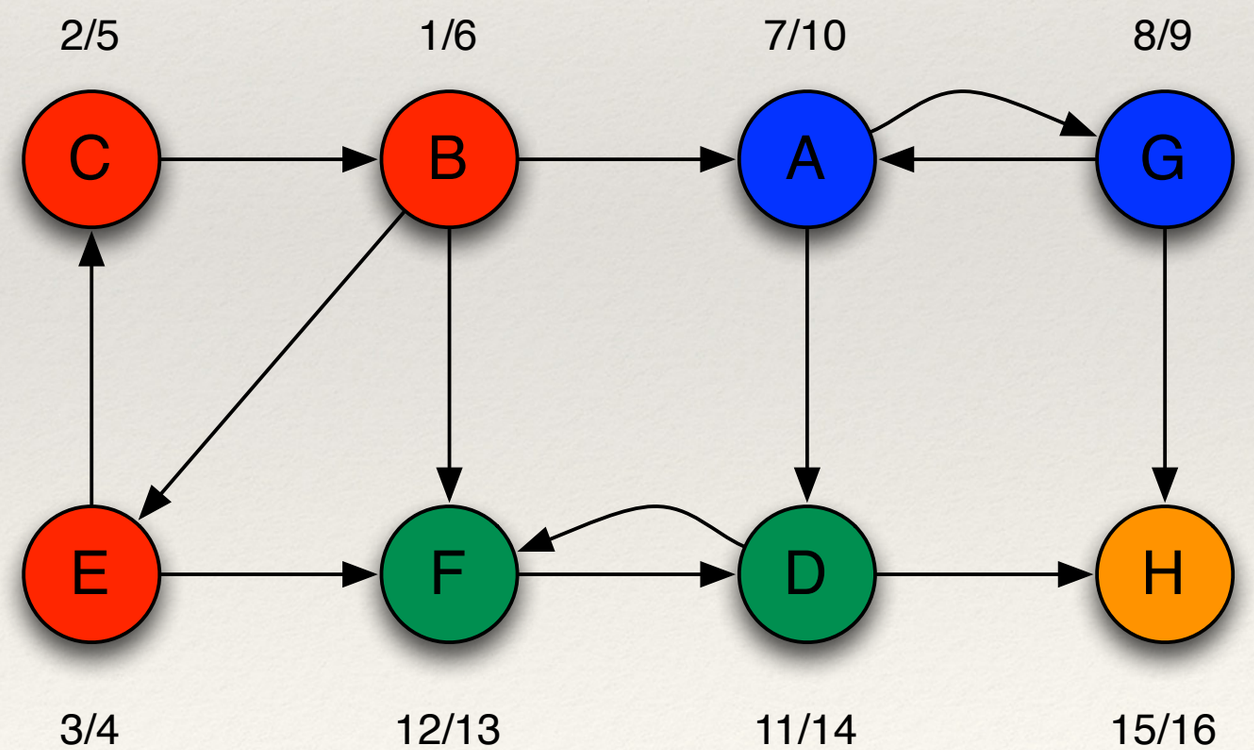


Exemple

COMPOSANTES-FORTEMENT-CONNEXES(G)

1 appeler PP(G) pour calculer les dates de fin de traitement $f[u]$
  pour chaque sommet u
2 calculer ${}^{\mathrm{T}}G$
3 appeler PP(${}^{\mathrm{T}}G$), mais dans la boucle principale de PP, traiter les sommets
  par ordre de $f[u]$ (calculés en ligne 1) décroissants
4 imprimer les sommets de chaque arborescence de la forêt obtenue
  en ligne 3 en tant que composante fortement connexe distincte

PP(ᵀG)



Exemple

COMPOSANTES-FORTEMENT-CONNEXES(G)

1 appeler PP(G) pour calculer les dates de fin de traitement $f[u]$
  pour chaque sommet u
2 calculer ${}^{\mathrm{T}}G$
3 appeler PP(${}^{\mathrm{T}}G$), mais dans la boucle principale de PP, traiter les sommets
  par ordre de $f[u]$ (calculés en ligne 1) décroissants
4 imprimer les sommets de chaque arborescence de la forêt obtenue
  en ligne 3 en tant que composante fortement connexe distincte