Algorithmique des graphes

5 — Graphes orientés, la suite

Anthony Labarre

3 mars 2021

Université
Gustave Eiffel

Récapitulatif

Jusqu'ici, on a vu :

- des graphes non orientés, et :

Récapitulatif

Jusqu'ici, on a vu :

- des graphes non orientés, et :
  - les parcours (largeur, profondeur) et les arbres associés ;

Récapitulatif

Jusqu'ici, on a vu :

- des graphes non orientés, et :
  - les parcours (largeur, profondeur) et les arbres associés ;
  - le calcul des composantes connexes ;

Récapitulatif

Jusqu'ici, on a vu :

- des graphes non orientés, et :
  - les parcours (largeur, profondeur) et les arbres associés ;
  - le calcul des composantes connexes ;
  - la détection de cycle ;

Récapitulatif

Jusqu'ici, on a vu :

- des graphes non orientés, et :
  - les parcours (largeur, profondeur) et les arbres associés ;
  - le calcul des composantes connexes ;
  - la détection de cycle ;
  - la reconnaissance des graphes bipartis ;

Récapitulatif

Jusqu'ici, on a vu :

- des graphes non orientés, et :
  - les parcours (largeur, profondeur) et les arbres associés ;
  - le calcul des composantes connexes ;
  - la détection de cycle ;
  - la reconnaissance des graphes bipartis ;
- des graphes pondérés, et :

Récapitulatif

Jusqu'ici, on a vu :

- des graphes non orientés, et :
  - les parcours (largeur, profondeur) et les arbres associés ;
  - le calcul des composantes connexes ;
  - la détection de cycle ;
  - la reconnaissance des graphes bipartis ;
- des graphes pondérés, et :
  - le calcul d'ACPM et FCPM ;

Récapitulatif

Jusqu'ici, on a vu :

- des graphes non orientés, et :
  - les parcours (largeur, profondeur) et les arbres associés ;
  - le calcul des composantes connexes ;
  - la détection de cycle ;
  - la reconnaissance des graphes bipartis ;
- des graphes pondérés, et :
  - le calcul d'ACPM et FCPM ;
  - le calcul de plus courts chemins (pour les poids ≥ 0) ;

Récapitulatif

Jusqu'ici, on a vu :

- des graphes non orientés, et :
  - les parcours (largeur, profondeur) et les arbres associés ;
  - le calcul des composantes connexes ;
  - la détection de cycle ;
  - la reconnaissance des graphes bipartis ;
- des graphes pondérés, et :
  - le calcul d'ACPM et FCPM ;
  - le calcul de plus courts chemins (pour les poids ≥ 0) ;
- des graphes orientés, et :

Récapitulatif

Jusqu'ici, on a vu :

- des graphes non orientés, et :
  - les parcours (largeur, profondeur) et les arbres associés ;
  - le calcul des composantes connexes ;
  - la détection de cycle ;
  - la reconnaissance des graphes bipartis ;
- des graphes pondérés, et :
  - le calcul d'ACPM et FCPM ;
  - le calcul de plus courts chemins (pour les poids ≥ 0) ;
- des graphes orientés, et :
  - les parcours (largeur, profondeur) et les arbres associés ;

Récapitulatif

Jusqu'ici, on a vu :

- des graphes non orientés, et :
  - les parcours (largeur, profondeur) et les arbres associés ;
  - le calcul des composantes connexes ;
  - la détection de cycle ;
  - la reconnaissance des graphes bipartis ;
- des graphes pondérés, et :
  - le calcul d'ACPM et FCPM ;
  - le calcul de plus courts chemins (pour les poids ≥ 0) ;
- des graphes orientés, et :
  - les parcours (largeur, profondeur) et les arbres associés ;
  - la détection de cycle ;

Récapitulatif

Jusqu'ici, on a vu :

- des graphes non orientés, et :
  - les parcours (largeur, profondeur) et les arbres associés ;
  - le calcul des composantes connexes ;
  - la détection de cycle ;
  - la reconnaissance des graphes bipartis ;
- des graphes pondérés, et :
  - le calcul d'ACPM et FCPM ;
  - le calcul de plus courts chemins (pour les poids ≥ 0) ;
- des graphes orientés, et :
  - les parcours (largeur, profondeur) et les arbres associés ;
  - la détection de cycle ;
  - le calcul de fermeture transitive ;

Correction de l'algorithme de Dijkstra

- On a vu la fois passée l'algorithme de Dijkstra, qui construit les plus courts chemins d'un sommet donné vers les autres sommets du graphe ;
- Prouvons aujourd'hui que cet algorithme fonctionne (reprenez le pseudocode !) ;

Correction de l'algorithme de Dijkstra

Soit T l'ensemble des sommets déjà traités, et :

- $\delta(a, b) =$ la distance **réelle** entre a et b ;
- $\varepsilon(a, b) =$ la distance **estimée** (par l'algorithme) entre a et b ;

Correction de l'algorithme de Dijkstra

Soit T l'ensemble des sommets déjà traités, et :

- $\delta(a, b) =$ la distance **réelle** entre a et b ;
- $\varepsilon(a, b) =$ la distance **estimée** (par l'algorithme) entre a et b ;

Montrons par **induction** sur $|T|$ que les estimations sont correctes.

Correction de l'algorithme de Dijkstra

Soit T l'ensemble des sommets déjà traités, et :

- $\delta(a, b) =$ la distance **réelle** entre a et b ;
- $\varepsilon(a, b) =$ la distance **estimée** (par l'algorithme) entre a et b ;

Montrons par **induction** sur $|T|$ que les estimations sont correctes.

1. **cas de base :** $T = \{s\}$, et $\varepsilon(s, s) = 0 = \delta(s, s)$;

Correction de l'algorithme de Dijkstra

Soit T l'ensemble des sommets déjà traités, et :

- $\delta(a, b) =$ la distance **réelle** entre a et b ;
- $\varepsilon(a, b) =$ la distance **estimée** (par l'algorithme) entre a et b ;

Montrons par **induction** sur $|T|$ que les estimations sont correctes.

1. **cas de base :** $T = \{s\}$, et $\varepsilon(s, s) = 0 = \delta(s, s)$;
2. **induction :**

Correction de l'algorithme de Dijkstra

Soit T l'ensemble des sommets déjà traités, et :

- $\delta(a, b) =$ la distance **réelle** entre a et b ;
- $\varepsilon(a, b) =$ la distance **estimée** (par l'algorithme) entre a et b ;

Montrons par **induction** sur $|T|$ que les estimations sont correctes.

1. **cas de base :** $T = \{s\}$, et $\varepsilon(s, s) = 0 = \delta(s, s)$;
2. **induction :**
   - hypothèse d'induction (HI) :

Correction de l'algorithme de Dijkstra

Soit T l'ensemble des sommets déjà traités, et :

- $\delta(a, b) =$ la distance **réelle** entre a et b ;
- $\varepsilon(a, b) =$ la distance **estimée** (par l'algorithme) entre a et b ;

Montrons par **induction** sur $|T|$ que les estimations sont correctes.

1. **cas de base :** $T = \{s\}$, et $\varepsilon(s, s) = 0 = \delta(s, s)$;
2. **induction :**
   - hypothèse d'induction (HI) : $\varepsilon(s, v) \leq \delta(s, v) \ \forall \ v \in T$.

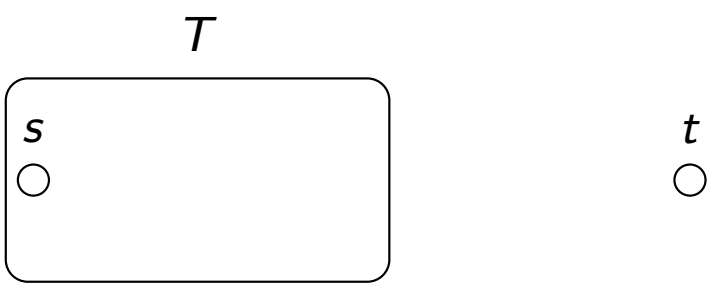
Correction de l'algorithme de Dijkstra

Soit T l'ensemble des sommets déjà traités, et :

- $\delta(a, b) =$ la distance **réelle** entre a et b ;
- $\varepsilon(a, b) =$ la distance **estimée** (par l'algorithme) entre a et b ;

Montrons par **induction** sur $|T|$ que les estimations sont correctes.

1. **cas de base :** $T = \{s\}$, et $\varepsilon(s, s) = 0 = \delta(s, s)$;
2. **induction :**
   - hypothèse d'induction (HI) : $\varepsilon(s, v) \leq \delta(s, v)\ \forall\ v \in T$.
   - à prouver : $\varepsilon(s, t) \leq \delta(s, t)$ (t est le prochain sommet à traiter) ;



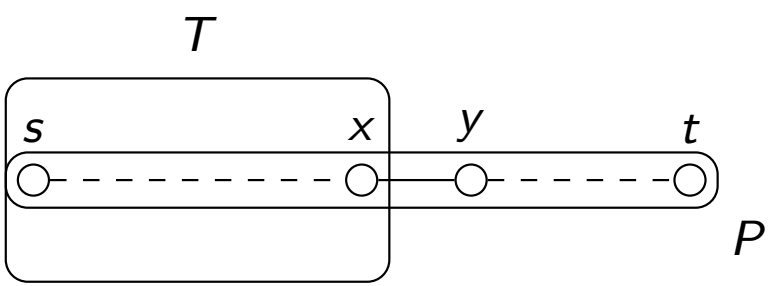
Correction de l'algorithme de Dijkstra

Soit T l'ensemble des sommets déjà traités, et :

- $\delta(a, b) =$ la distance **réelle** entre a et b ;
- $\varepsilon(a, b) =$ la distance **estimée** (par l'algorithme) entre a et b ;

Montrons par **induction** sur $|T|$ que les estimations sont correctes.

1. **cas de base :** $T = \{s\}$, et $\varepsilon(s, s) = 0 = \delta(s, s)$;
2. **induction :**
   - hypothèse d'induction (HI) : $\varepsilon(s, v) \leq \delta(s, v)\ \forall\ v \in T$.
   - à prouver : $\varepsilon(s, t) \leq \delta(s, t)$ (t est le prochain sommet à traiter) ;



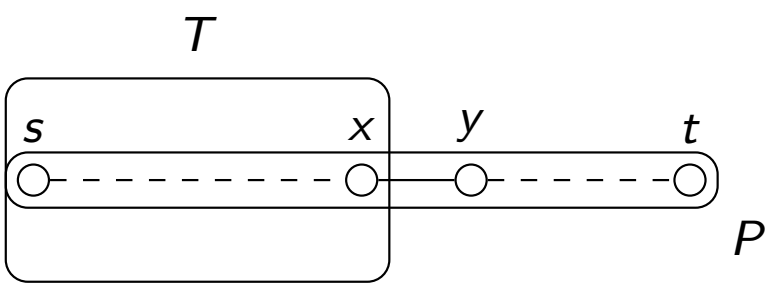
Correction de l'algorithme de Dijkstra

Soit T l'ensemble des sommets déjà traités, et :

- $\delta(a, b) =$ la distance **réelle** entre a et b ;
- $\varepsilon(a, b) =$ la distance **estimée** (par l'algorithme) entre a et b ;

Montrons par **induction** sur $|T|$ que les estimations sont correctes.

1. **cas de base :** $T = \{s\}$, et $\varepsilon(s, s) = 0 = \delta(s, s)$;
2. **induction :**
   - hypothèse d'induction (HI) : $\varepsilon(s, v) \leq \delta(s, v)\ \forall\ v \in T$.
   - à prouver : $\varepsilon(s, t) \leq \delta(s, t)$ (t est le prochain sommet à traiter) ;



$$\delta(s, t) \geq \delta(s, y)$$

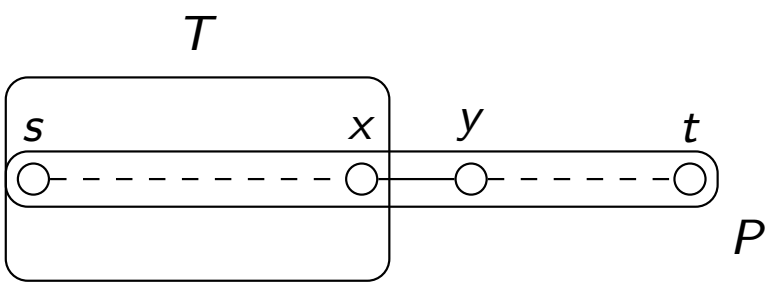
Correction de l'algorithme de Dijkstra

Soit T l'ensemble des sommets déjà traités, et :

- $\delta(a, b) =$ la distance **réelle** entre a et b ;
- $\varepsilon(a, b) =$ la distance **estimée** (par l'algorithme) entre a et b ;

Montrons par **induction** sur $|T|$ que les estimations sont correctes.

1. **cas de base :** $T = \{s\}$, et $\varepsilon(s, s) = 0 = \delta(s, s)$;
2. **induction :**
   - hypothèse d'induction (HI) : $\varepsilon(s, v) \leq \delta(s, v)\ \forall\ v \in T$.
   - à prouver : $\varepsilon(s, t) \leq \delta(s, t)$ (t est le prochain sommet à traiter) ;



$$\begin{aligned}\delta(s, t) &\geq \delta(s, y) \\ &= \delta(s, x) + w(\{x, y\})\end{aligned}$$

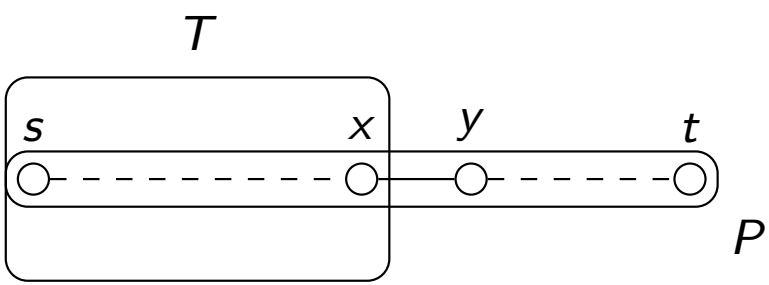
Correction de l'algorithme de Dijkstra

Soit T l'ensemble des sommets déjà traités, et :

- $\delta(a, b) =$ la distance **réelle** entre a et b ;
- $\varepsilon(a, b) =$ la distance **estimée** (par l'algorithme) entre a et b ;

Montrons par **induction** sur $|T|$ que les estimations sont correctes.

1. **cas de base :** $T = \{s\}$, et $\varepsilon(s, s) = 0 = \delta(s, s)$;
2. **induction :**
   - hypothèse d'induction (HI) : $\varepsilon(s, v) \leq \delta(s, v)\ \forall\ v \in T$.
   - à prouver : $\varepsilon(s, t) \leq \delta(s, t)$ (t est le prochain sommet à traiter) ;



$$\begin{aligned}\delta(s, t) &\geq \delta(s, y) \\ &= \delta(s, x) + w(\{x, y\}) \\ &\geq \varepsilon(s, x) + w(\{x, y\})\end{aligned}$$

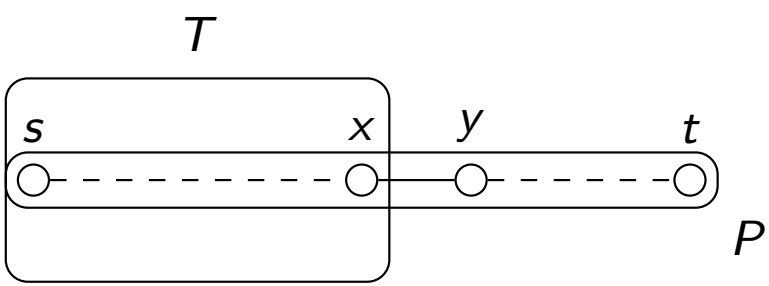
Correction de l'algorithme de Dijkstra

Soit T l'ensemble des sommets déjà traités, et :

- $\delta(a, b) =$ la distance **réelle** entre a et b ;
- $\varepsilon(a, b) =$ la distance **estimée** (par l'algorithme) entre a et b ;

Montrons par **induction** sur $|T|$ que les estimations sont correctes.

1. **cas de base :** $T = \{s\}$, et $\varepsilon(s, s) = 0 = \delta(s, s)$;
2. **induction :**
   - hypothèse d'induction (HI) : $\varepsilon(s, v) \leq \delta(s, v)\ \forall\ v \in T$.
   - à prouver : $\varepsilon(s, t) \leq \delta(s, t)$ (t est le prochain sommet à traiter) ;



$$\begin{aligned}\delta(s,t) &\geq \delta(s,y)\\ &= \delta(s,x) + w(\{x,y\})\\ &\geq \varepsilon(s,x) + w(\{x,y\})\\ &\geq \varepsilon(s,y)\end{aligned}$$

□

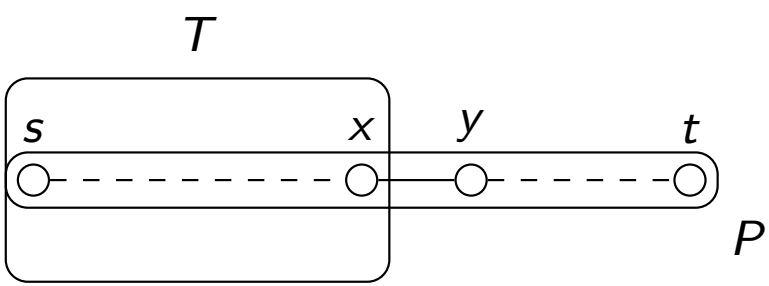
Correction de l'algorithme de Dijkstra

Soit T l'ensemble des sommets déjà traités, et :

- $\delta(a, b) =$ la distance **réelle** entre a et b ;
- $\varepsilon(a, b) =$ la distance **estimée** (par l'algorithme) entre a et b ;

Montrons par **induction** sur $|T|$ que les estimations sont correctes.

1. **cas de base :** $T = \{s\}$, et $\varepsilon(s, s) = 0 = \delta(s, s)$;
2. **induction :**
   - hypothèse d'induction (HI) : $\varepsilon(s, v) \leq \delta(s, v)\ \forall\ v \in T$.
   - à prouver : $\varepsilon(s, t) \leq \delta(s, t)$ (t est le prochain sommet à traiter) ;



$$\begin{aligned}
\delta(s, t) &\geq \delta(s, y) \\
&= \delta(s, x) + w(\{x, y\}) \\
&\geq \varepsilon(s, x) + w(\{x, y\}) \\
&\geq \varepsilon(s, y) \\
&\geq \varepsilon(s, t).
\end{aligned}$$

□

Connexité dans les graphes orientés

Définition 1

Une **composante fortement connexe** H d'un graphe orienté G est un ensemble maximal de sommets dont toute paire est reliée par un chemin orienté.

Connexité dans les graphes orientés

Définition 1

Une **composante fortement connexe** H d'un graphe orienté G est un ensemble maximal de sommets dont toute paire est reliée par un chemin orienté. H est **faiblement connexe** s'il s'agit d'une composante connexe de la version non-orientée de G, mais pas d'une composante fortement connexe de G.

Connexité dans les graphes orientés

Définition 1

Une **composante fortement connexe** H d'un graphe orienté G est un ensemble maximal de sommets dont toute paire est reliée par un chemin orienté. H est **faiblement connexe** s'il s'agit d'une composante connexe de la version non-orientée de G, mais pas d'une composante fortement connexe de G.

À cause de l'orientation, il n'est donc plus correct de dire qu'un graphe orienté **fortement connexe** est forcément "en un seul morceau".

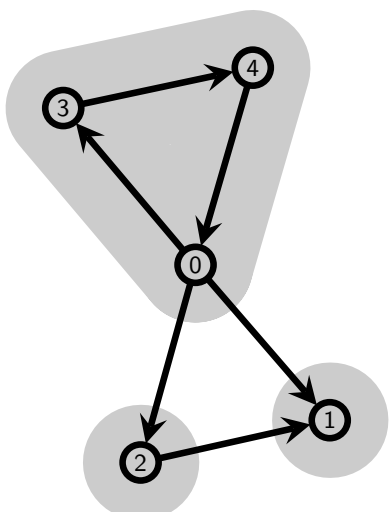
Connexité dans les graphes orientés

Définition 1

Une **composante fortement connexe** H d'un graphe orienté G est un ensemble maximal de sommets dont toute paire est reliée par un chemin orienté. H est **faiblement connexe** s'il s'agit d'une composante connexe de la version non-orientée de G, mais pas d'une composante fortement connexe de G.

À cause de l'orientation, il n'est donc plus correct de dire qu'un graphe orienté **fortement connexe** est forcément "en un seul morceau".

Exemple 1



Identification des CFC

- Sur base des algorithmes déjà vus, comment faire pour identifier les CFC ?

Identification des CFC

- Sur base des algorithmes déjà vus, comment faire pour identifier les CFC ?
- On pourrait :

Identification des CFC

- Sur base des algorithmes déjà vus, comment faire pour identifier les CFC ?
- On pourrait :
  1. calculer les descendants de chaque sommet par un simple parcours ;

Identification des CFC

- Sur base des algorithmes déjà vus, comment faire pour identifier les CFC ?
- On pourrait :
  1. calculer les descendants de chaque sommet par un simple parcours ;
  2. regrouper les paires de sommets mutuellement accessibles (donc dans les deux sens) ;

Identification des CFC

- Sur base des algorithmes déjà vus, comment faire pour identifier les CFC ?
- On pourrait :
  1. calculer les descendants de chaque sommet par un simple parcours ;
  2. regrouper les paires de sommets mutuellement accessibles (donc dans les deux sens) ;
- Ça fonctionne, mais c'est lent : on doit faire $|V|$ parcours ;

Identification des CFC

- Sur base des algorithmes déjà vus, comment faire pour identifier les CFC ?
- On pourrait :
  1. calculer les descendants de chaque sommet par un simple parcours ;
  2. regrouper les paires de sommets mutuellement accessibles (donc dans les deux sens) ;
- Ça fonctionne, mais c'est lent : on doit faire $|V|$ parcours ;
- L'algorithme de Kosaraju-Sharir qu'on va voir le fait en deux parcours ;

Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;

Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;

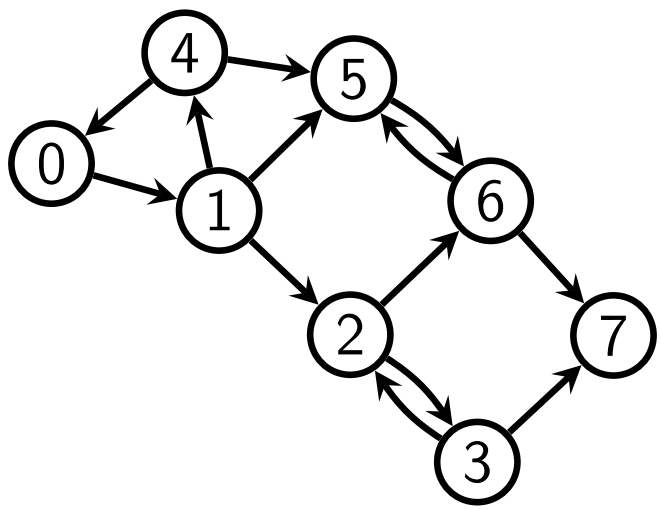
Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

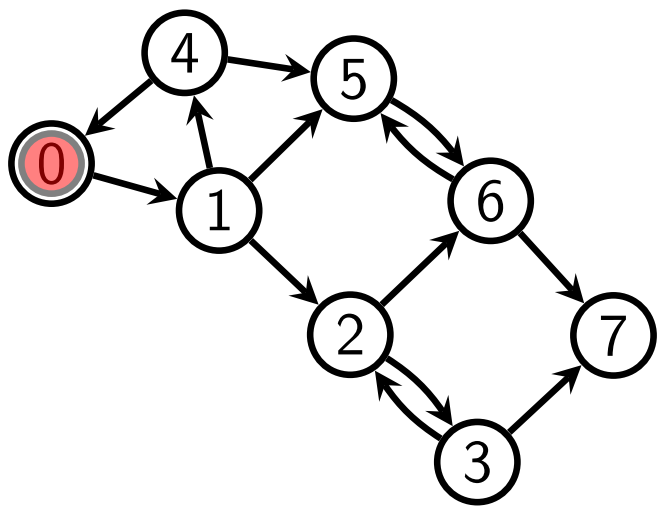
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

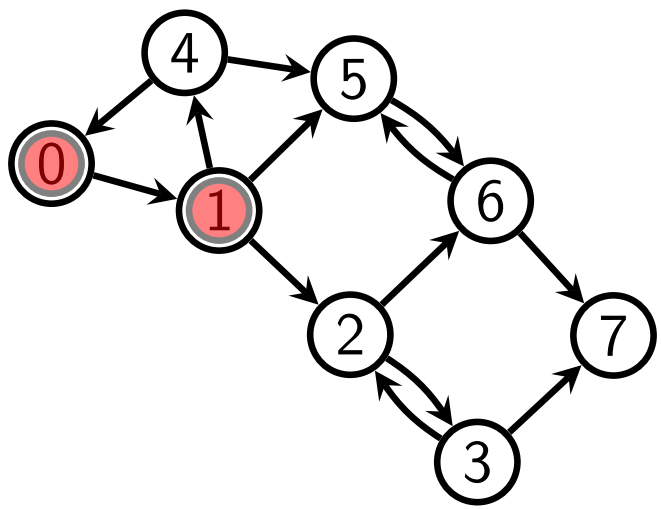
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

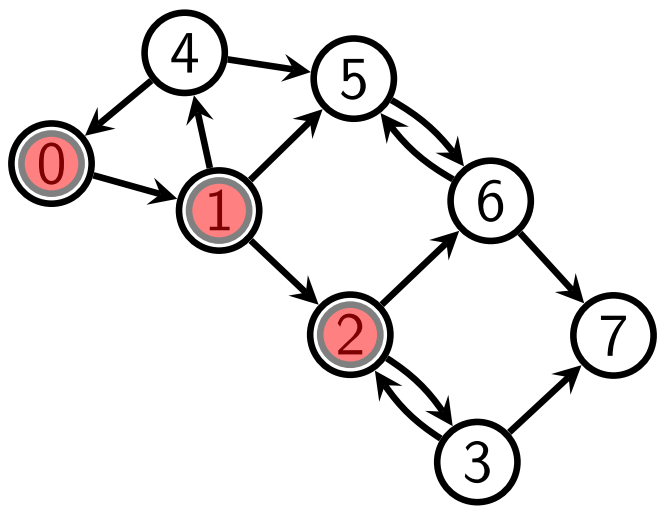
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

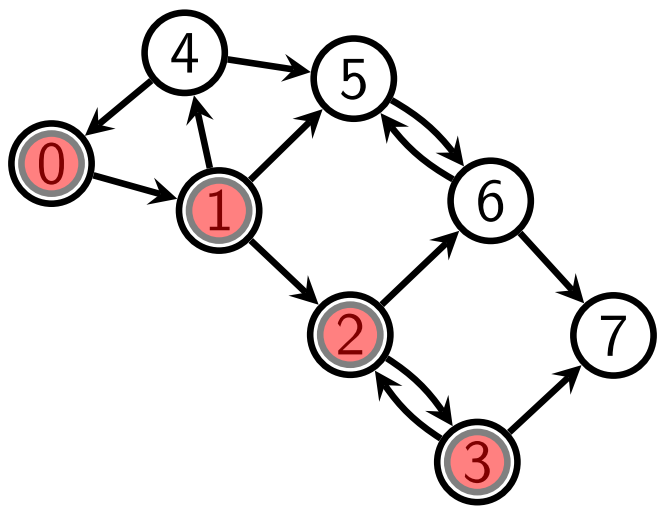
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

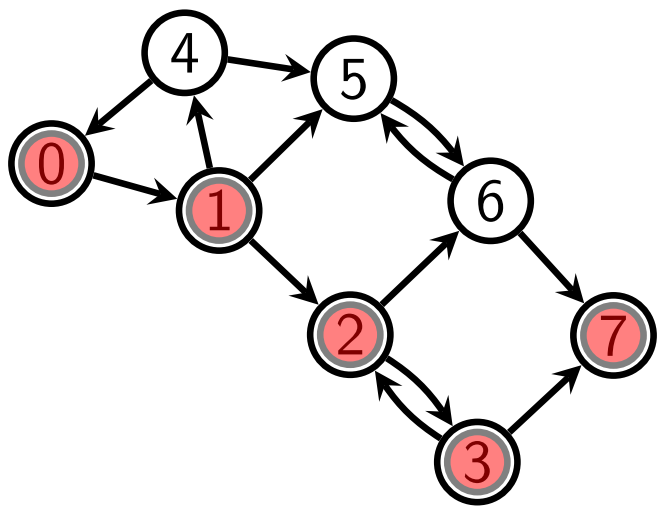
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

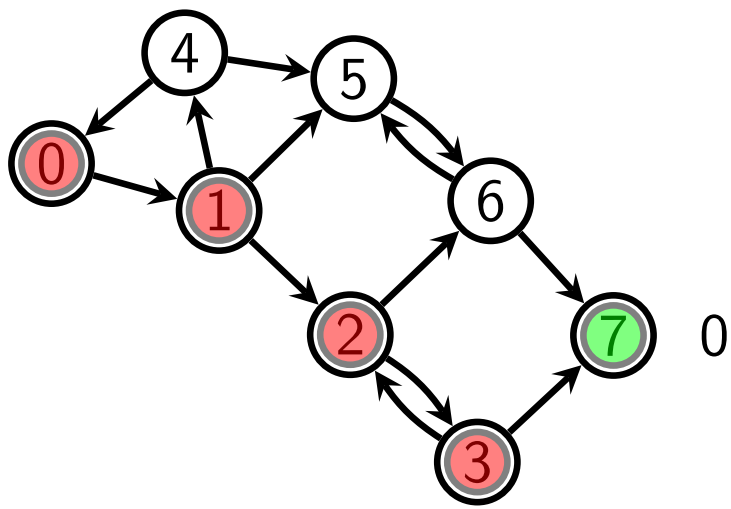
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

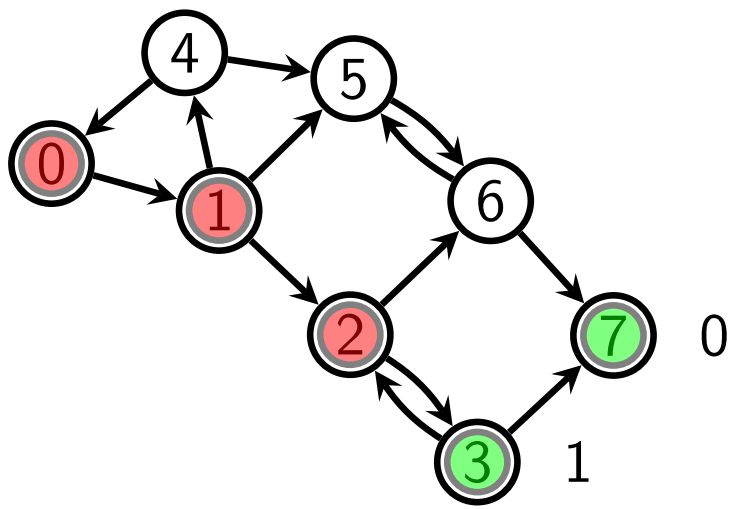
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

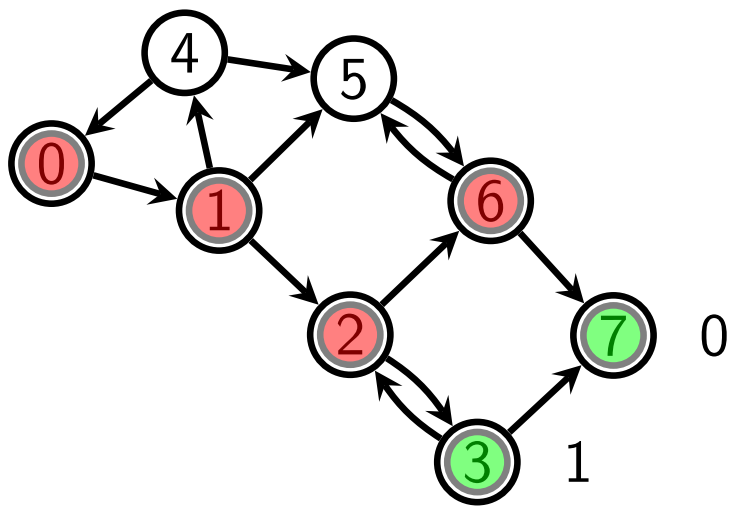
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

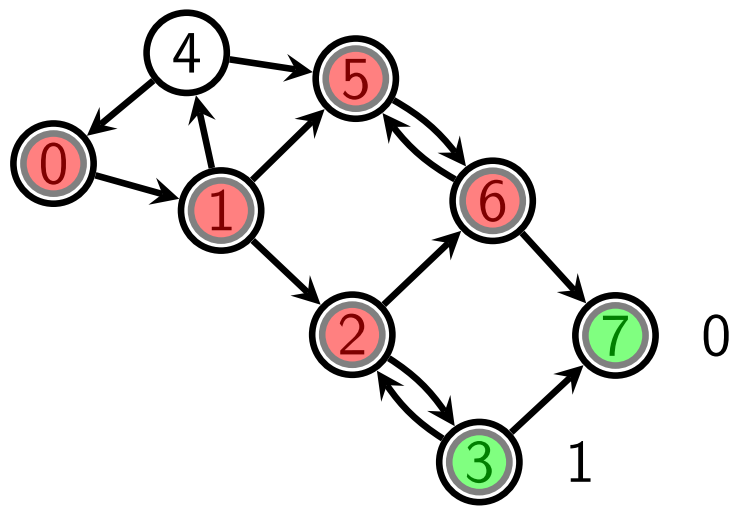
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

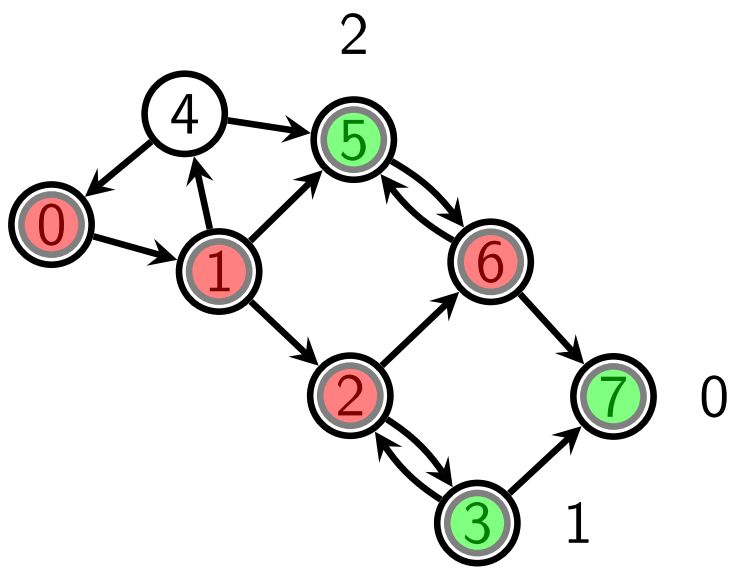
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

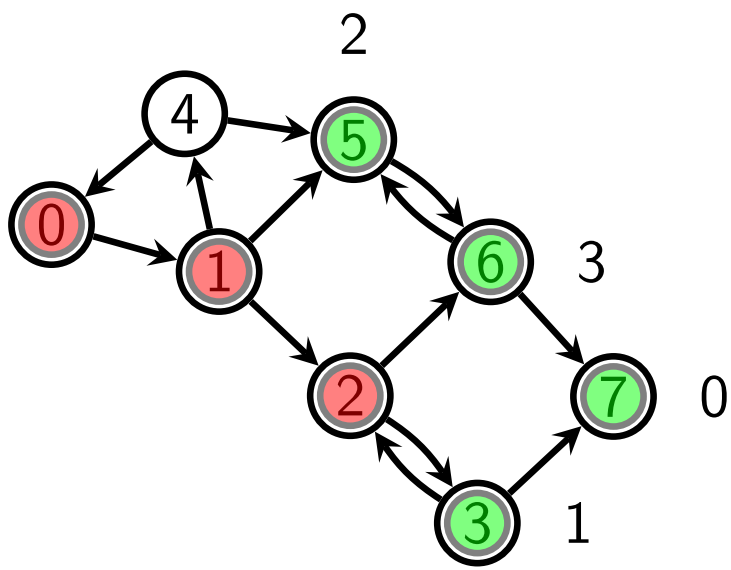
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

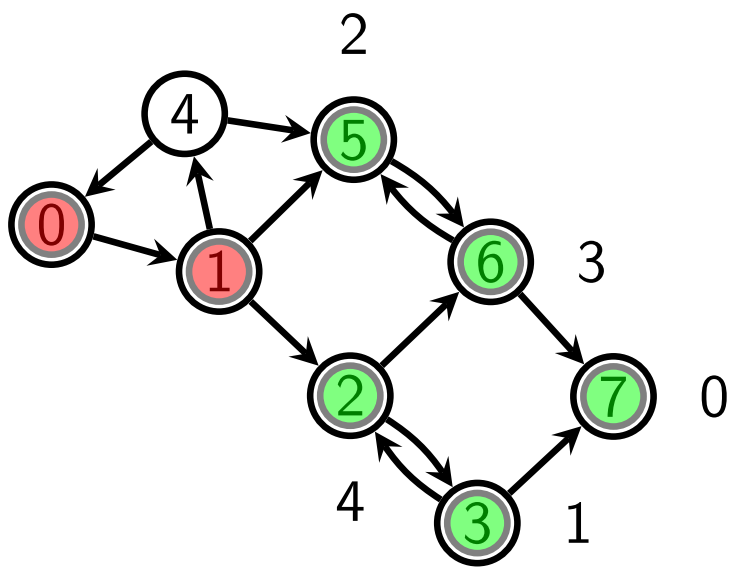
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

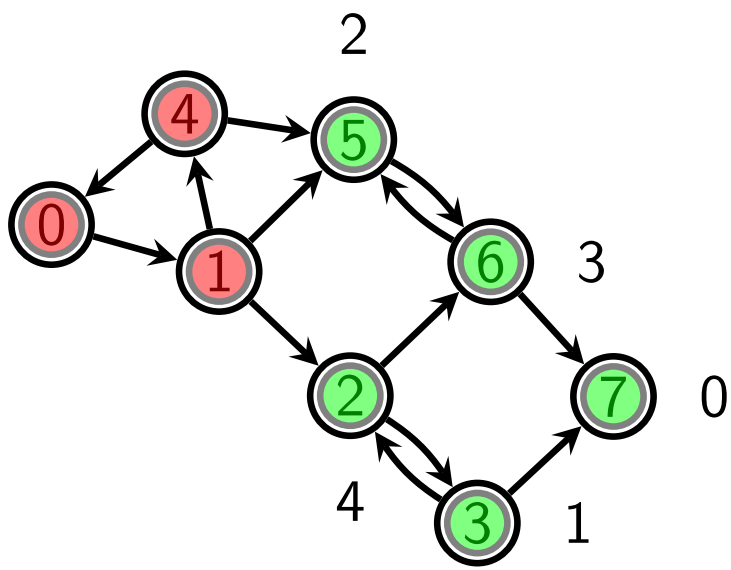
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

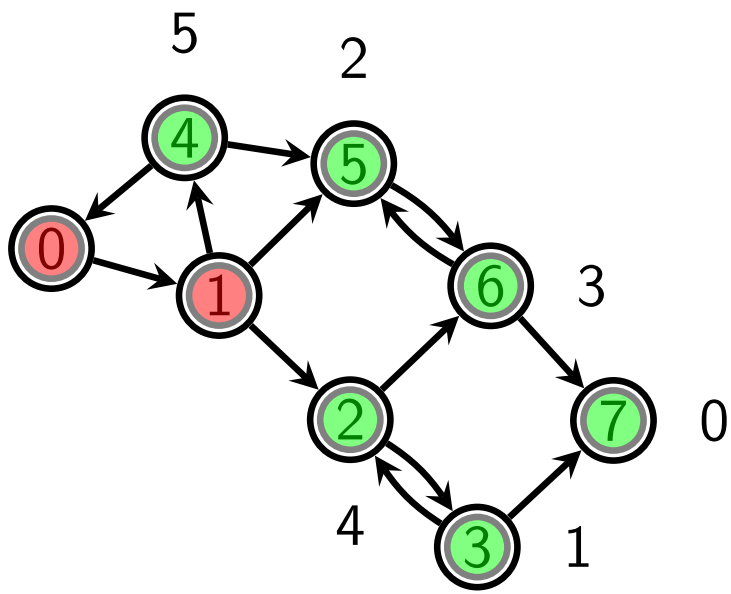
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

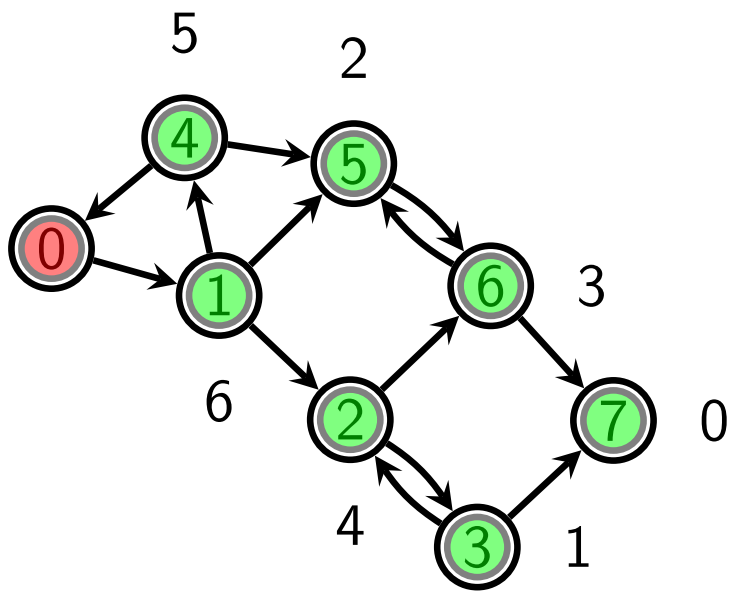
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

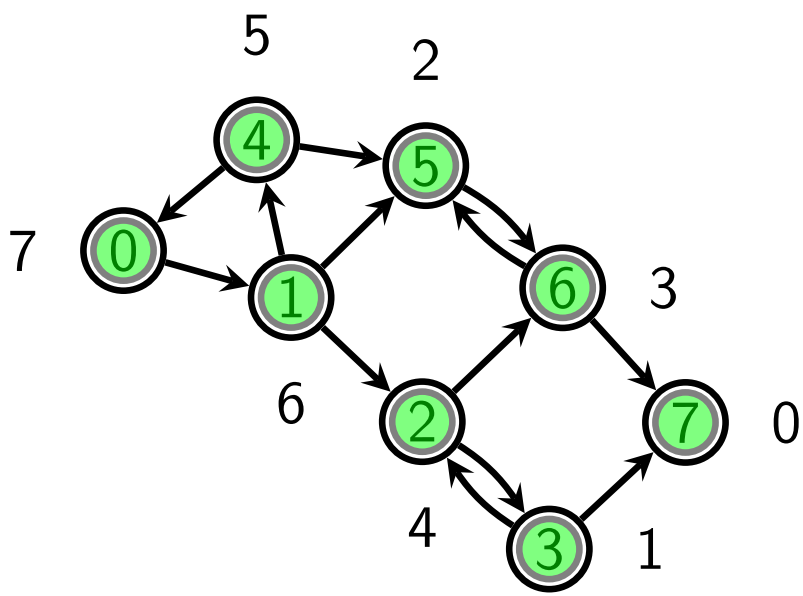
Exemple 2



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va "dater" les sommets en fonction du moment où leur exploration **se termine** ;
- À la fin du parcours, tous les sommets seront donc numérotés ;

Exemple 2



L'algorithme du parcours en profondeur daté

L'algorithme suivant doit être lancé sur tous les sommets du graphe (par exemple par un algorithme auxiliaire ProfondeurDatesAux) :

Algorithme 1 : ProfondeurDates(G, départ, dates, instant)

Entrées : un graphe orienté G, un sommet de départ, un tableau de dates, et un instant.

Résultat : dates contient les dates de fin de visite des sommets de G accessibles à partir du sommet de départ dans l'ordre où le parcours en profondeur les a découverts.

```
1 dates[départ] ← 0 ;     // marquer le début de l'exploration;
2 pour chaque v ∈ G.successeurs(départ) faire
3 |   si dates[v] = NIL alors ProfondeurDates(G, v, dates, instant) ;
4 dates[départ] ← instant ;   // marquer la fin de l'exploration;
5 instant ← instant + 1;
```

Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours sur le graphe "renversé" ; c'est-à-dire le graphe G' tel que $(u, v) \in A(G) \Leftrightarrow (v, u) \in A(G')$;

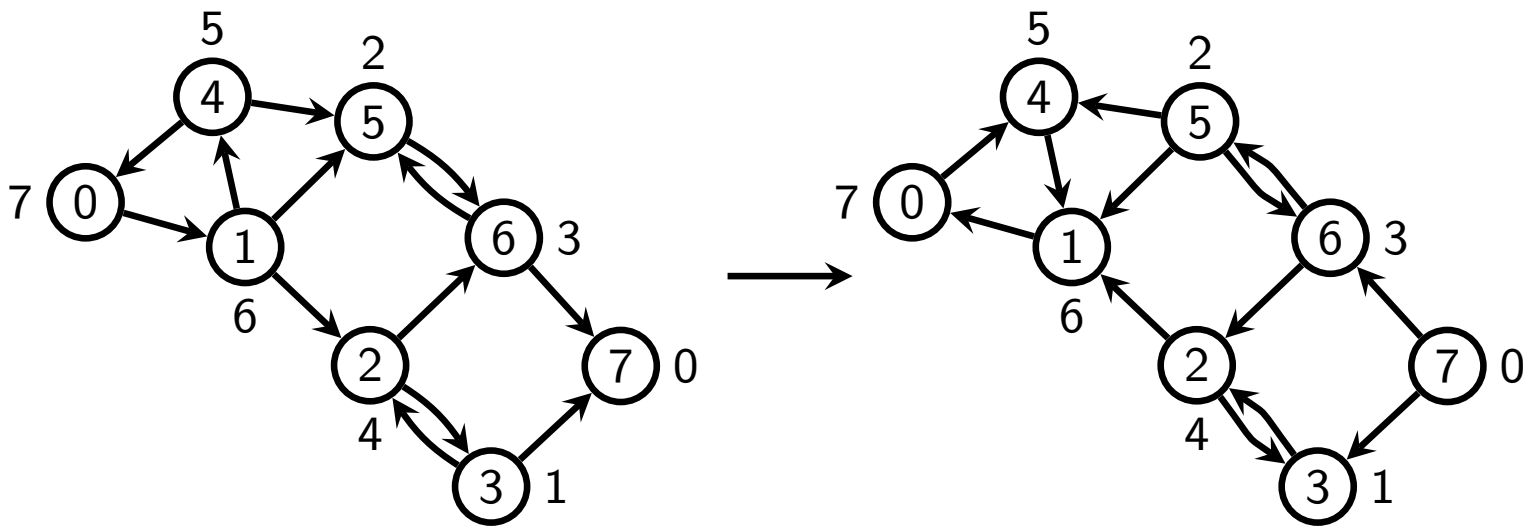
Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours sur le graphe "renversé" ; c'est-à-dire le graphe G' tel que $(u, v) \in A(G) \Leftrightarrow (v, u) \in A(G')$;
- Ce parcours s'effectue par date de fin décroissante ;

Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours sur le graphe "renversé" ; c'est-à-dire le graphe G' tel que $(u, v) \in A(G) \Leftrightarrow (v, u) \in A(G')$;
- Ce parcours s'effectue par date de fin décroissante ;

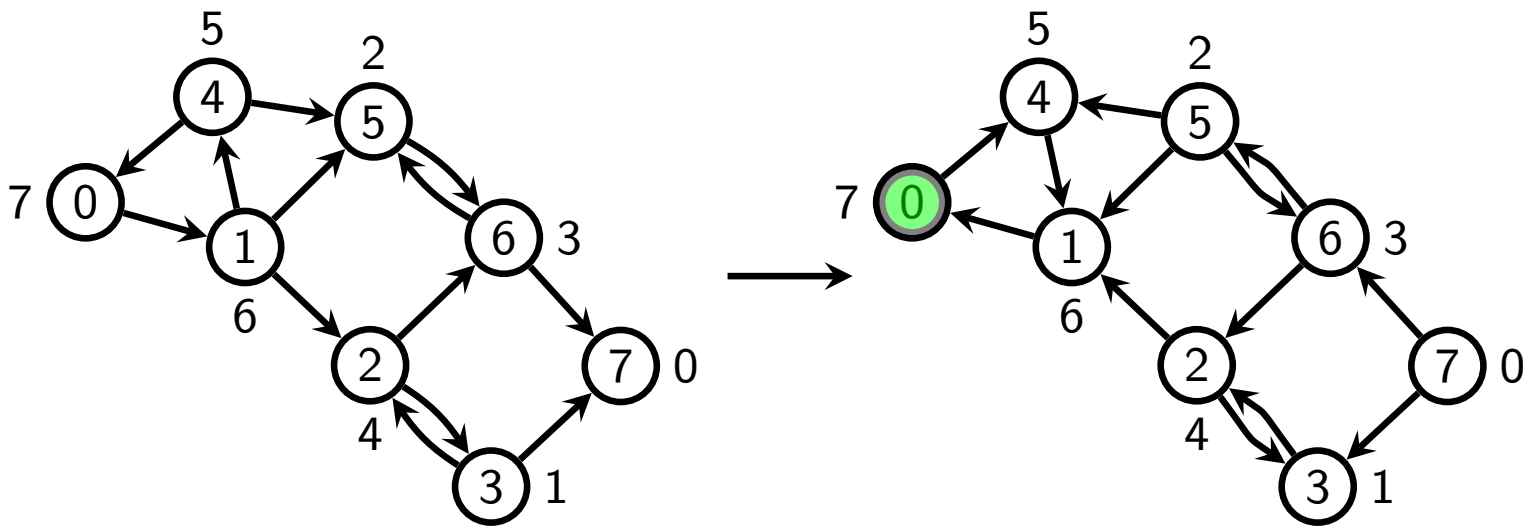
Exemple 3



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours sur le graphe "renversé" ; c'est-à-dire le graphe G' tel que $(u, v) \in A(G) \Leftrightarrow (v, u) \in A(G')$;
- Ce parcours s'effectue par date de fin décroissante ;

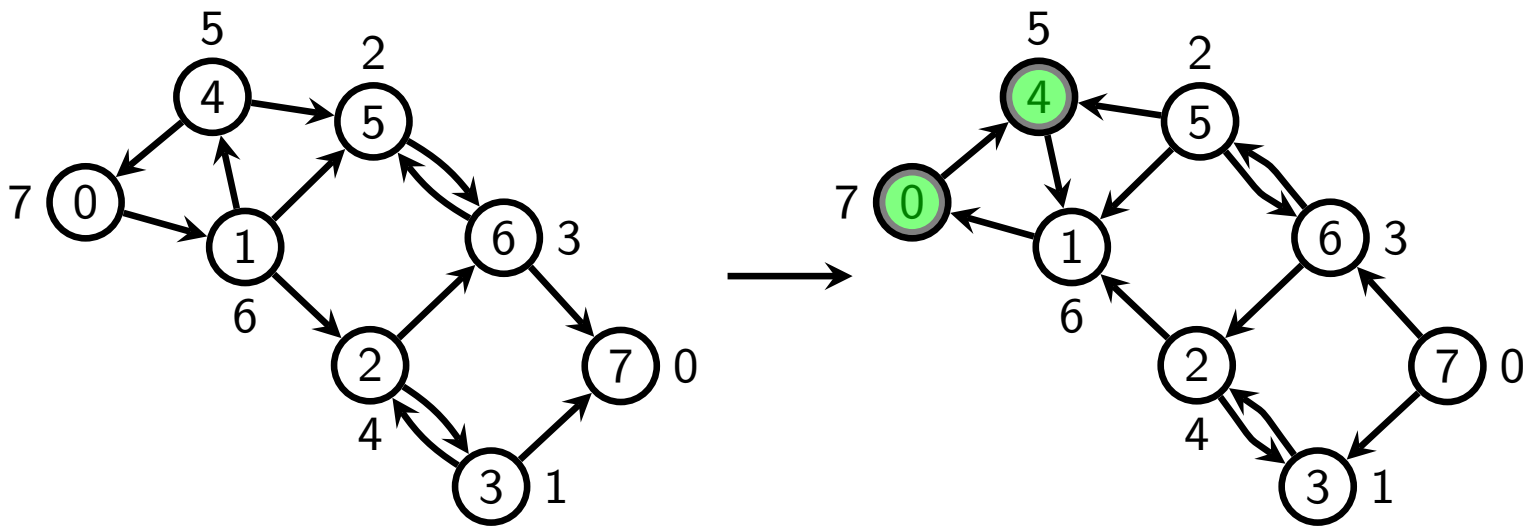
Exemple 3



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours sur le graphe "renversé" ; c'est-à-dire le graphe G' tel que $(u, v) \in A(G) \Leftrightarrow (v, u) \in A(G')$;
- Ce parcours s'effectue par date de fin décroissante ;

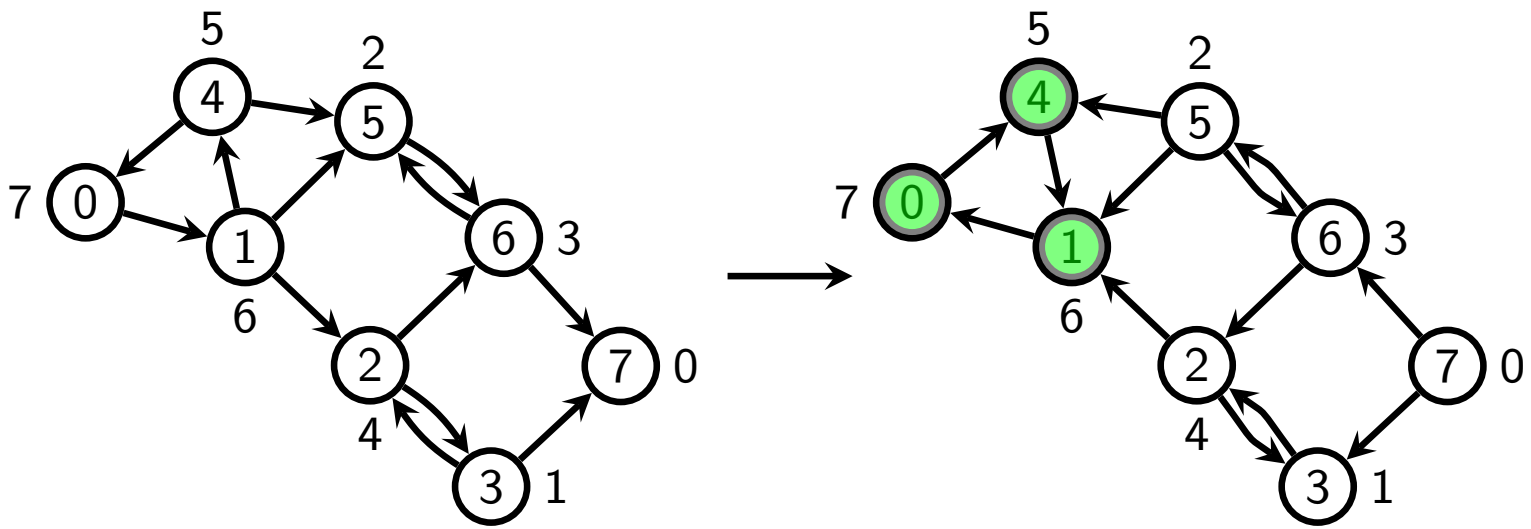
Exemple 3



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours sur le graphe "renversé" ; c'est-à-dire le graphe G' tel que $(u, v) \in A(G) \Leftrightarrow (v, u) \in A(G')$;
- Ce parcours s'effectue par date de fin décroissante ;

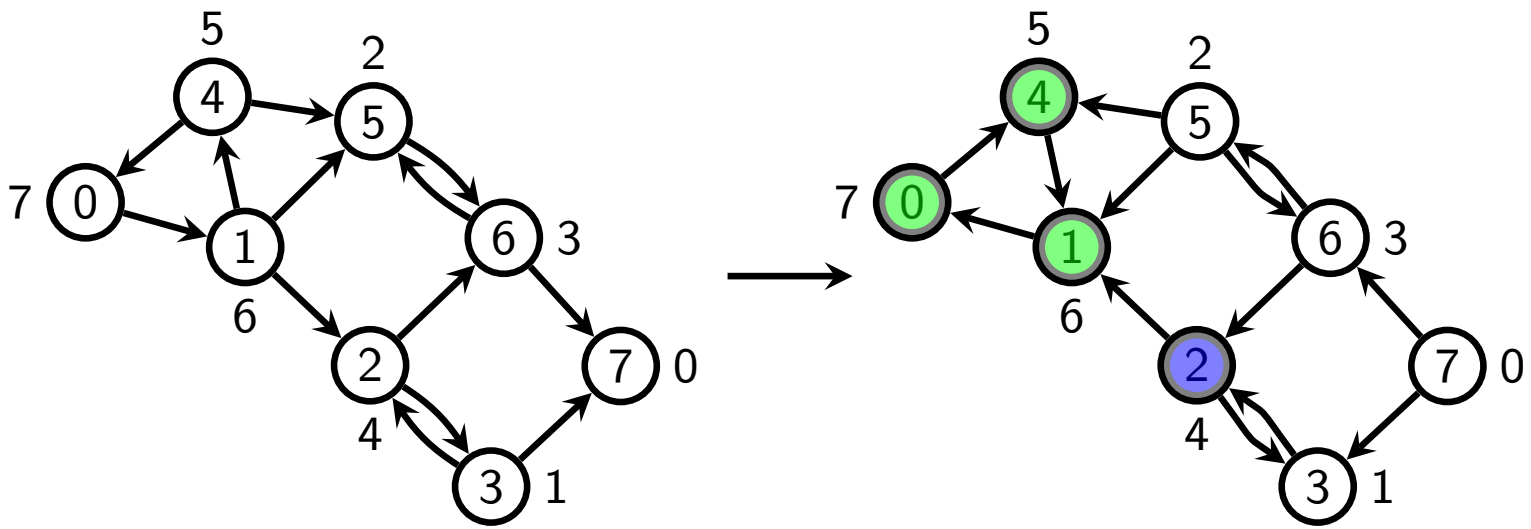
Exemple 3



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours sur le graphe "renversé" ; c'est-à-dire le graphe G' tel que $(u, v) \in A(G) \Leftrightarrow (v, u) \in A(G')$;
- Ce parcours s'effectue par date de fin décroissante ;

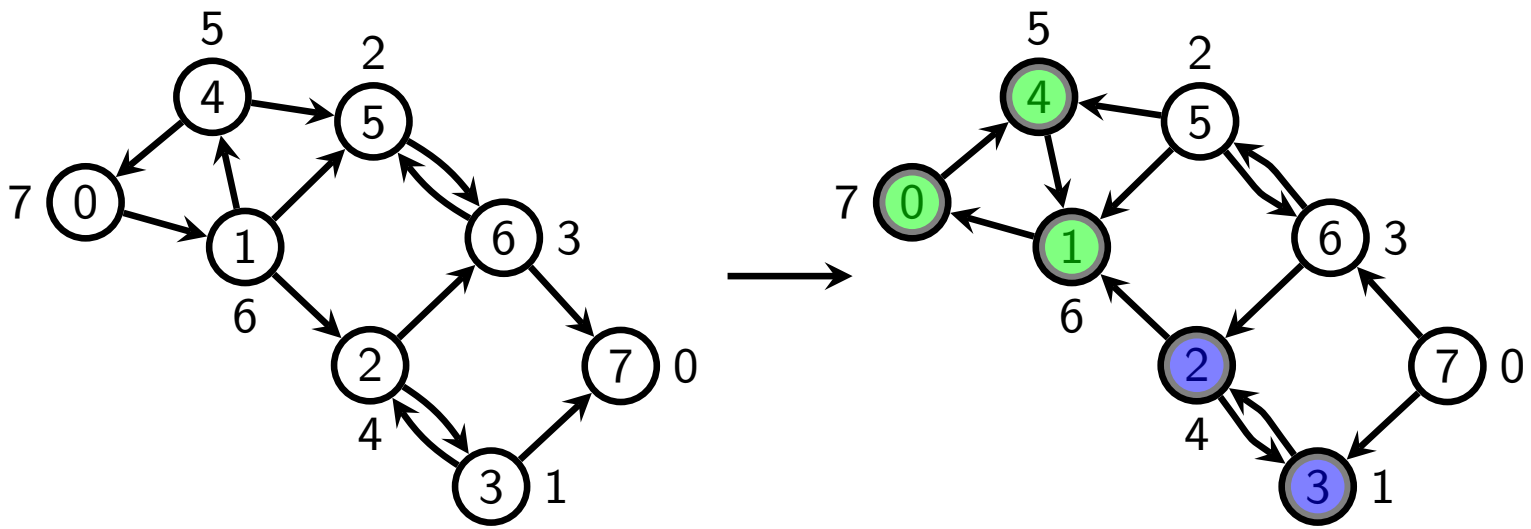
Exemple 3



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours sur le graphe "renversé" ; c'est-à-dire le graphe G' tel que $(u, v) \in A(G) \Leftrightarrow (v, u) \in A(G')$;
- Ce parcours s'effectue par date de fin décroissante ;

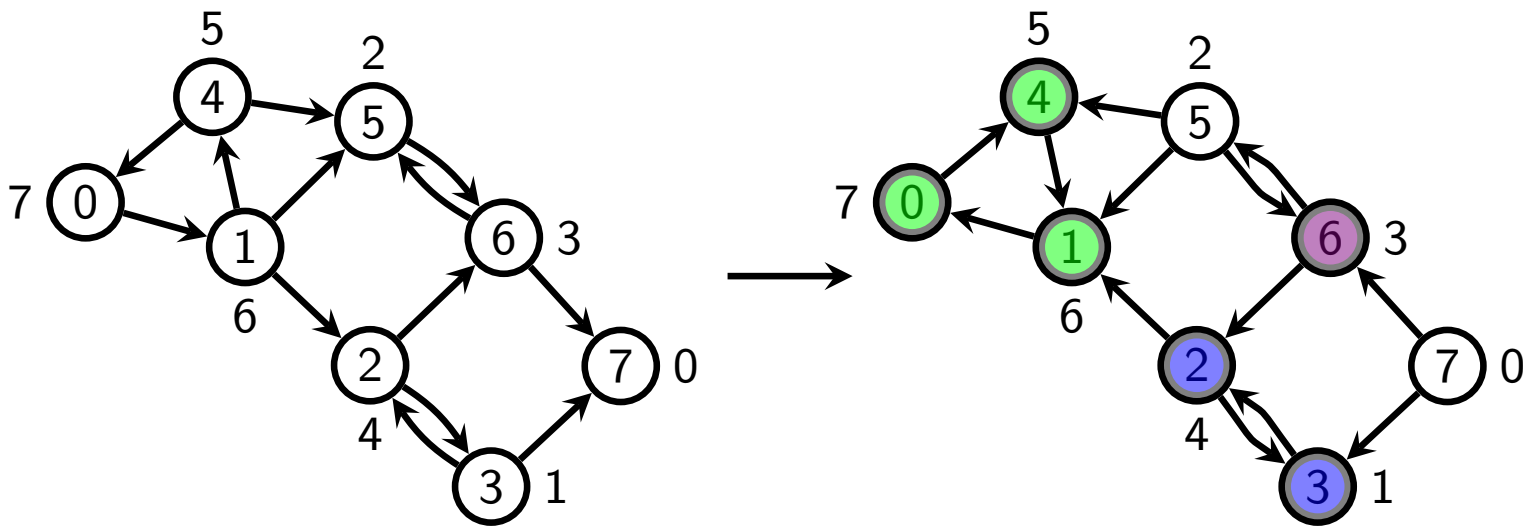
Exemple 3



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours sur le graphe "renversé" ; c'est-à-dire le graphe G' tel que $(u, v) \in A(G) \Leftrightarrow (v, u) \in A(G')$;
- Ce parcours s'effectue par date de fin décroissante ;

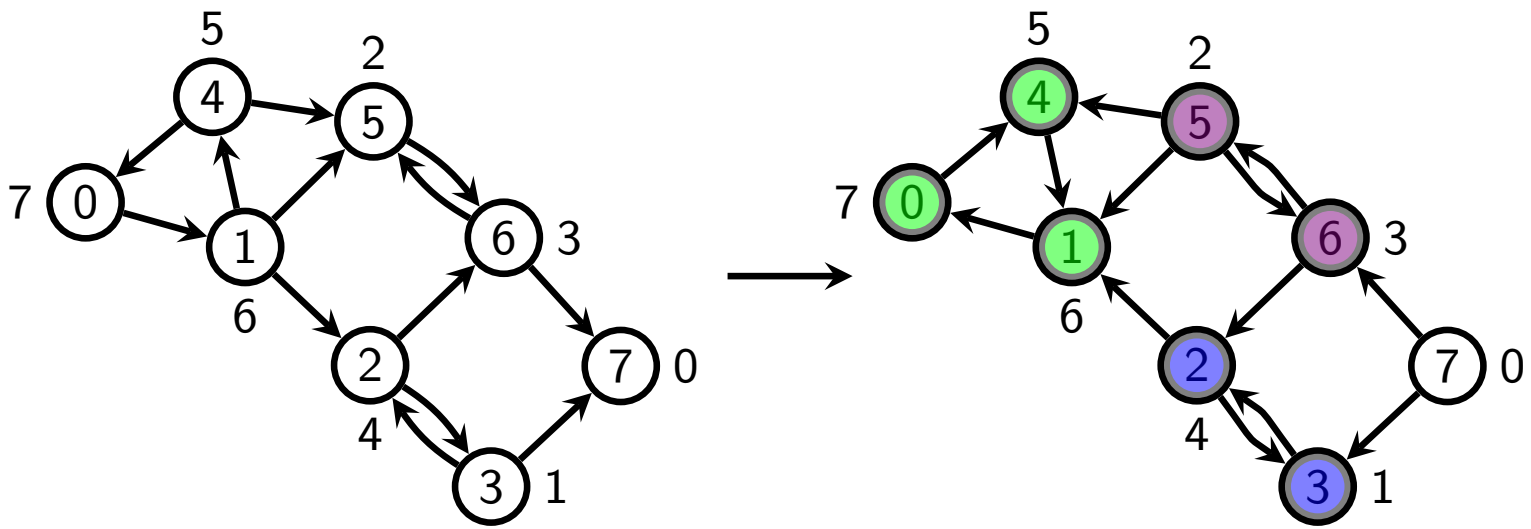
Exemple 3



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours sur le graphe "renversé" ; c'est-à-dire le graphe G' tel que $(u, v) \in A(G) \Leftrightarrow (v, u) \in A(G')$;
- Ce parcours s'effectue par date de fin décroissante ;

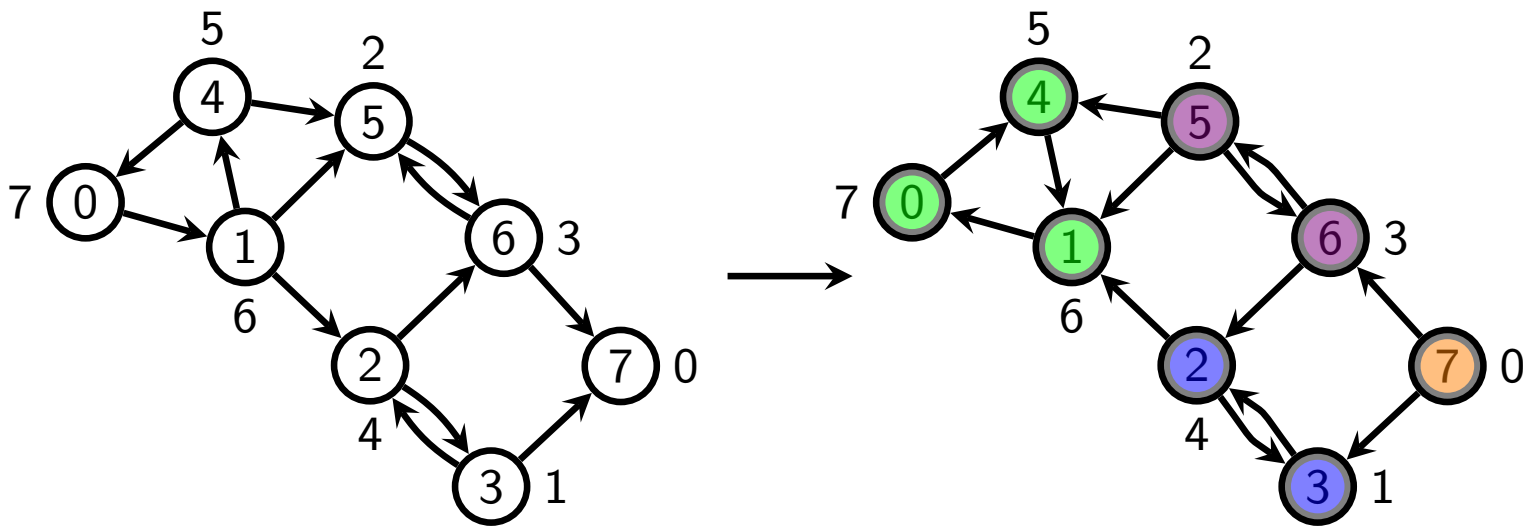
Exemple 3



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours sur le graphe "renversé" ; c'est-à-dire le graphe G' tel que $(u, v) \in A(G) \Leftrightarrow (v, u) \in A(G')$;
- Ce parcours s'effectue par date de fin décroissante ;

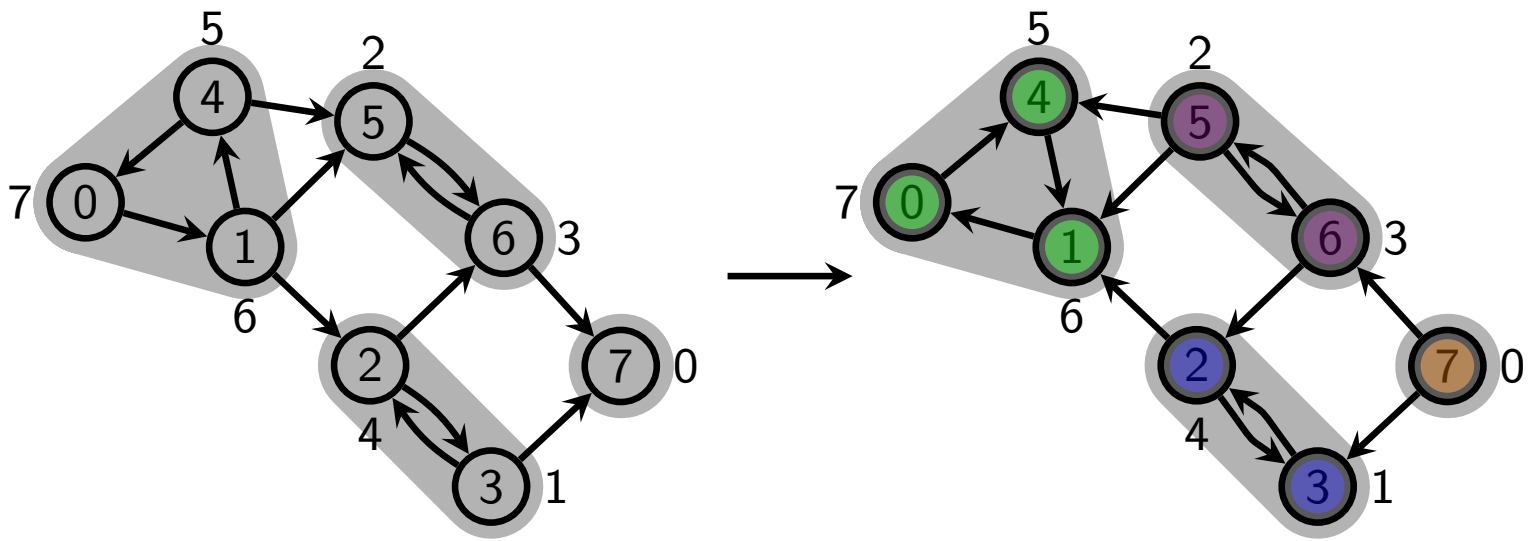
Exemple 3



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours sur le graphe "renversé" ; c'est-à-dire le graphe G' tel que $(u, v) \in A(G) \Leftrightarrow (v, u) \in A(G')$;
- Ce parcours s'effectue par date de fin décroissante ;

Exemple 3



. . . et c'est fini !

L'algorithme de Kosaraju-Sharir proprement dit

Algorithme 2 : KosarajuSharir(G)

Entrées : un graphe orienté G.
Sortie : les composantes fortement connexes de G.

```
1 CFC ← liste();
2 dates ← ProfondeurDatesAux(G);
3 G' ← renverser_arcs(G);
4 visités ← tableau(G.nombre_sommets(), FAUX);
5 pour chaque v ∈ renverser(trier_sommets_par_date(G'.sommets(),
    dates)) faire
6 |   si ¬ visités[v] alors
7 |   |   CFC.ajouter_en_fin(ProfondeurOrienté(G', v, visités));
8 renvoyer CFC;
```

Complexité

- La complexité de l'algorithme de Kosaraju-Sharir se calcule aisément :

Complexité

- La complexité de l'algorithme de Kosaraju-Sharir se calcule aisément :
  1. parcours daté : $O(|V| + |A|)$;

Complexité

- La complexité de l'algorithme de Kosaraju-Sharir se calcule aisément :
  1. parcours daté : $O(|V| + |A|)$;
  2. renversement : $O(|V| + |A|)$;

Complexité

- La complexité de l'algorithme de Kosaraju-Sharir se calcule aisément :
  1. parcours daté : $O(|V| + |A|)$;
  2. renversement : $O(|V| + |A|)$;
  3. tri : $O(|V| \log |V|)$;

Complexité

- La complexité de l'algorithme de Kosaraju-Sharir se calcule aisément :
  1. parcours daté : $O(|V| + |A|)$;
  2. renversement : $O(|V| + |A|)$;
  3. tri : $O(|V| \log |V|)$;
  4. parcours de G' : $O(|V| + |A'|) = O(|V| + |A|)$;

Complexité

- La complexité de l'algorithme de Kosaraju-Sharir se calcule aisément :
  1. parcours daté : $O(|V| + |A|)$;
  2. renversement : $O(|V| + |A|)$;
  3. tri : $O(|V| \log |V|)$;
  4. parcours de G' : $O(|V| + |A'|) = O(|V| + |A|)$;
- $\Rightarrow$ total : $O(|V| \log |V| + |A|)$;

Complexité

- La complexité de l'algorithme de Kosaraju-Sharir se calcule aisément :
  1. parcours daté : $O(|V| + |A|)$;
  2. renversement : $O(|V| + |A|)$;
  3. tri : $O(|V| \log |V|)$;
  4. parcours de G' : $O(|V| + |A'|) = O(|V| + |A|)$;
- $\Rightarrow$ total : $O(|V| \log |V| + |A|)$;
- Peut-on faire mieux ?

Complexité

- La complexité de l'algorithme de Kosaraju-Sharir se calcule aisément :
  1. parcours daté : $O(|V| + |A|)$;
  2. renversement : $O(|V| + |A|)$;
  3. tri : $O(|V| \log |V|)$;
  4. parcours de G' : $O(|V| + |A'|) = O(|V| + |A|)$;
- $\Rightarrow$ total : $O(|V| \log |V| + |A|)$;
- Peut-on faire mieux ?
  1. si le premier parcours empile les sommets au lieu de les numéroter, pas besoin de tri $\Rightarrow O(|V| + |A|)$;

Complexité

- La complexité de l'algorithme de Kosaraju-Sharir se calcule aisément :
  1. parcours daté : $O(|V| + |A|)$;
  2. renversement : $O(|V| + |A|)$;
  3. tri : $O(|V| \log |V|)$;
  4. parcours de G' : $O(|V| + |A'|) = O(|V| + |A|)$;
- $\Rightarrow$ total : $O(|V| \log |V| + |A|)$;
- Peut-on faire mieux ?
  1. si le premier parcours empile les sommets au lieu de les numéroter, pas besoin de tri $\Rightarrow O(|V| + |A|)$;
  2. il existe un algorithme ne réalisant qu'un seul parcours [2] $\Rightarrow$ même complexité, mais plus rapide ;

Quelques intuitions sur l'algorithme de Kosaraju-Sharir

- Sans (pour l'instant) prouver sa correction, demandons-nous comment et pourquoi cet algorithme fonctionne ;

Quelques intuitions sur l'algorithme de Kosaraju-Sharir

- Sans (pour l'instant) prouver sa correction, demandons-nous comment et pourquoi cet algorithme fonctionne ;
- Intuitivement, le premier parcours permet de savoir quels sommets sont accessibles à partir de chaque sommet du graphe ;

Quelques intuitions sur l'algorithme de Kosaraju-Sharir

- Sans (pour l'instant) prouver sa correction, demandons-nous comment et pourquoi cet algorithme fonctionne ;
- Intuitivement, le premier parcours permet de savoir quels sommets sont accessibles à partir de chaque sommet du graphe ;
- La numérotation nous donne aussi des informations ; on constate que :

$$\forall\, u \in V(G) : \quad \text{date\_fin}(u) = 1 + \max_{v \,\in\, \text{descendants}(u)} \text{date\_fin}(v).$$

Quelques intuitions sur l'algorithme de Kosaraju-Sharir

- Sans (pour l'instant) prouver sa correction, demandons-nous comment et pourquoi cet algorithme fonctionne ;
- Intuitivement, le premier parcours permet de savoir quels sommets sont accessibles à partir de chaque sommet du graphe ;
- La numérotation nous donne aussi des informations ; on constate que :

$$\forall\, u \in V(G) : \quad \text{date\_fin}(u) = 1 + \max_{v \,\in\, \text{descendants}(u)} \text{date\_fin}(v).$$

- Donc : v est un descendant de $u \Rightarrow \text{date\_fin}(v) < \text{date\_fin}(u)$;

Quelques intuitions sur l'algorithme de Kosaraju-Sharir

- Sans (pour l'instant) prouver sa correction, demandons-nous comment et pourquoi cet algorithme fonctionne ;
- Intuitivement, le premier parcours permet de savoir quels sommets sont accessibles à partir de chaque sommet du graphe ;
- La numérotation nous donne aussi des informations ; on constate que :

$$\forall\, u \in V(G) : \quad \text{date\_fin}(u) = 1 + \max_{v \,\in\, \text{descendants}(u)} \text{date\_fin}(v).$$

- Donc : v est un descendant de $u \Rightarrow \text{date\_fin}(v) < \text{date\_fin}(u)$;
- La réciproque est fausse ;

Quelques intuitions sur l'algorithme de Kosaraju-Sharir

- Maintenant qu'on "sait" quels sommets sont accessibles au départ de chaque sommet u, il faut savoir à partir de quels sommets on peut atteindre chaque sommet u ;

Quelques intuitions sur l'algorithme de Kosaraju-Sharir

- Maintenant qu'on "sait" quels sommets sont accessibles au départ de chaque sommet u, il faut savoir à partir de quels sommets on peut atteindre chaque sommet u ;
- Comme on parcourt le graphe renversé G', x est un descendant de y dans $G' \Leftrightarrow y$ est un descendant de x dans G ;

Quelques intuitions sur l'algorithme de Kosaraju-Sharir

- Maintenant qu'on "sait" quels sommets sont accessibles au départ de chaque sommet u, il faut savoir à partir de quels sommets on peut atteindre chaque sommet u ;
- Comme on parcourt le graphe renversé G', x est un descendant de y dans $G' \Leftrightarrow y$ est un descendant de x dans G ;
- On démarre toujours d'un sommet u de date maximale dans G', pour atteindre des descendants D_u ...

Quelques intuitions sur l'algorithme de Kosaraju-Sharir

- Maintenant qu'on "sait" quels sommets sont accessibles au départ de chaque sommet u, il faut savoir à partir de quels sommets on peut atteindre chaque sommet u ;
- Comme on parcourt le graphe renversé G', x est un descendant de y dans $G' \Leftrightarrow y$ est un descendant de x dans G ;
- On démarre toujours d'un sommet u de date maximale dans G', pour atteindre des descendants D_u ...
- Et donc dans G, D_u est l'ensemble des sommets à partir desquels on peut atteindre u ;

Le graphe des composantes fortement connexes

Définition 2

Soit G un graphe orienté. Le **graphe des composantes fortement connexes** de G est le graphe orienté H défini par :

Le graphe des composantes fortement connexes

Définition 2

Soit G un graphe orienté. Le **graphe des composantes fortement connexes** de G est le graphe orienté H défini par :

- $V(H)$ est l'ensemble $\{C_1, C_2, \ldots, C_p\}$ des composantes fortement connexes de G ;

Le graphe des composantes fortement connexes

Définition 2

Soit G un graphe orienté. Le **graphe des composantes fortement connexes** de G est le graphe orienté H défini par :

- $V(H)$ est l'ensemble $\{C_1, C_2, \ldots, C_p\}$ des composantes fortement connexes de G ;
- $A(H) = \{(C_i, C_j) \mid \exists u \in C_i, v \in C_j : (u, v) \in A(G)\}$.

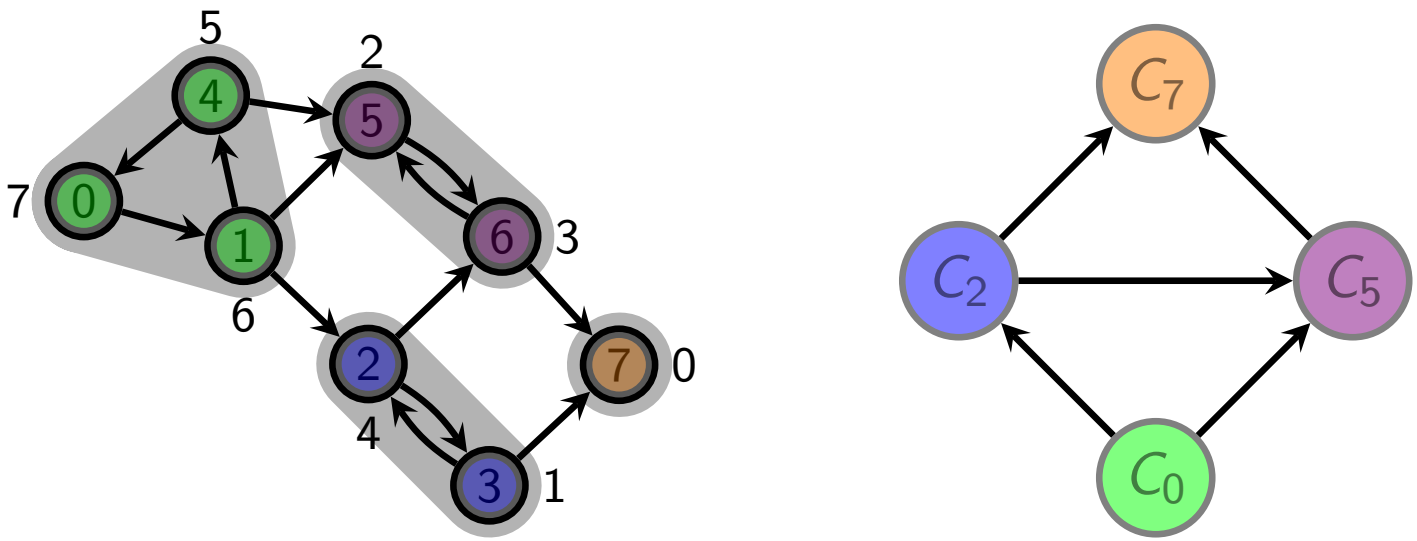
Le graphe des composantes fortement connexes

Définition 2

Soit G un graphe orienté. Le **graphe des composantes fortement connexes** de G est le graphe orienté H défini par :

- $V(H)$ est l'ensemble $\{C_1, C_2, \ldots, C_p\}$ des composantes fortement connexes de G ;
- $A(H) = \{(C_i, C_j) \mid \exists u \in C_i, v \in C_j : (u, v) \in A(G)\}$.

Exemple 4



Graphes orientés acycliques

- Le graphe des composantes fortement connexes est acyclique ;

Graphes orientés acycliques

- Le graphe des composantes fortement connexes est acyclique ;
- En effet, s'il contenait un cycle C, alors toutes les composantes reliées par C seraient mutuellement accessibles et ne devraient donc former qu'une seule composante fortement connexe ;

Graphes orientés acycliques

- Le graphe des composantes fortement connexes est acyclique ;
- En effet, s'il contenait un cycle C, alors toutes les composantes reliées par C seraient mutuellement accessibles et ne devraient donc former qu'une seule composante fortement connexe ;
- De nombreux problèmes deviennent plus simples sur les graphes orientés acycliques (ou DAG (pour **d***irected* **a***cyclic* **g***raphs*)) ;

Tri topologique

- On est déjà capables de reconnaître les DAG ;

Tri topologique

- On est déjà capables de reconnaître les DAG ;
- Si un graphe est un DAG, on peut ordonner ses sommets de manière à rencontrer tous les prédécesseurs d'un sommet avant lui ; c'est ce qu'on appelle un *ordre topologique* ;

Tri topologique

- On est déjà capables de reconnaître les DAG ;
- Si un graphe est un DAG, on peut ordonner ses sommets de manière à rencontrer tous les prédécesseurs d'un sommet avant lui ; c'est ce qu'on appelle un *ordre topologique* ;
- Applications :

Tri topologique

- On est déjà capables de reconnaître les DAG ;
- Si un graphe est un DAG, on peut ordonner ses sommets de manière à rencontrer tous les prédécesseurs d'un sommet avant lui ; c'est ce qu'on appelle un *ordre topologique* ;
- Applications :
  - dans quel ordre réaliser les tâches d'un projet ?

Tri topologique

- On est déjà capables de reconnaître les DAG ;
- Si un graphe est un DAG, on peut ordonner ses sommets de manière à rencontrer tous les prédécesseurs d'un sommet avant lui ; c'est ce qu'on appelle un *ordre topologique* ;
- Applications :
  - dans quel ordre réaliser les tâches d'un projet ?
  - combien de temps le projet va-t-il durer au minimum ?

Tri topologique

- On est déjà capables de reconnaître les DAG ;
- Si un graphe est un DAG, on peut ordonner ses sommets de manière à rencontrer tous les prédécesseurs d'un sommet avant lui ; c'est ce qu'on appelle un *ordre topologique* ;
- Applications :
  - dans quel ordre réaliser les tâches d'un projet ?
  - combien de temps le projet va-t-il durer au minimum ?
  - ...

Ordres topologiques

Définition 3

Un **ordre topologique** pour un graphe orienté acyclique G est un ordonnancement L de ses sommets dans lequel tout sommet apparaît après ses prédécesseurs.

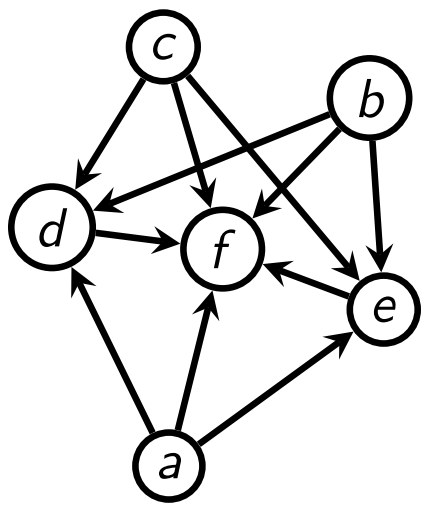
Ordres topologiques

Définition 3

Un **ordre topologique** pour un graphe orienté acyclique G est un ordonnancement L de ses sommets dans lequel tout sommet apparaît après ses prédécesseurs.

Exemple 5

Voici un DAG pour lequel l'ordre (a, b, c, d, e, f) est un ordre topologique :



Algorithme de Kahn

- Un algorithme simple et intuitif dû à Kahn [1] nous permet de produire un ordre topologique comme suit :

Algorithme de Kahn

- Un algorithme simple et intuitif dû à Kahn [1] nous permet de produire un ordre topologique comme suit :
  1. placer les **sources** (sommets de degré entrant nul) en premier lieu ;

Algorithme de Kahn

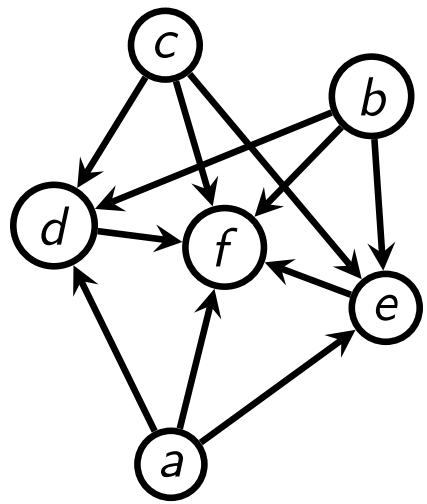
- Un algorithme simple et intuitif dû à Kahn [1] nous permet de produire un ordre topologique comme suit :
  1. placer les **sources** (sommets de degré entrant nul) en premier lieu ;
  2. retirer les sources du graphe et recommencer ;

Algorithme de Kahn

- Un algorithme simple et intuitif dû à Kahn [1] nous permet de produire un ordre topologique comme suit :
  1. placer les **sources** (sommets de degré entrant nul) en premier lieu ;
  2. retirer les sources du graphe et recommencer ;
- Si l'on arrive à "vider" le graphe de cette manière, le résultat est un ordre topologique ; sinon le graphe possède un cycle.

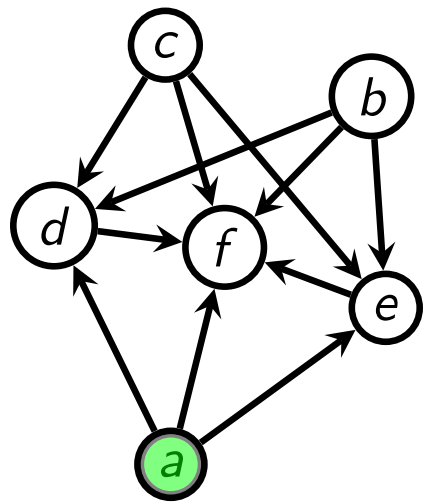
Déroulement de l'algorithme de Kahn

Exemple 6



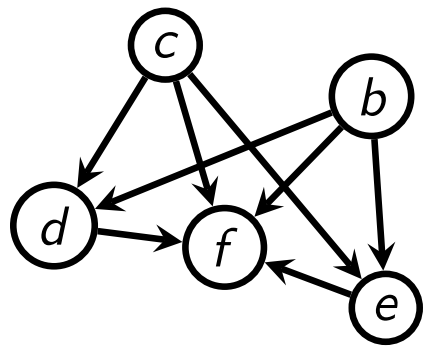
Déroulement de l'algorithme de Kahn

Exemple 6



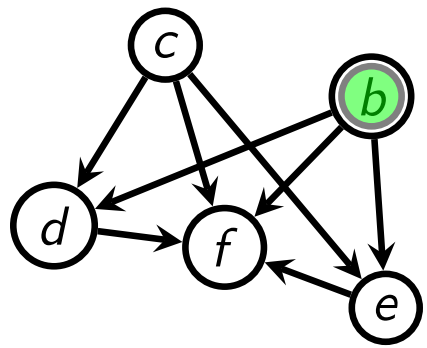
Déroulement de l'algorithme de Kahn

Exemple 6



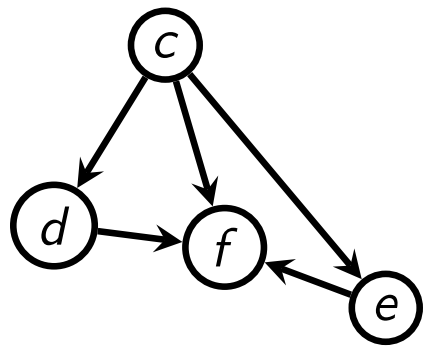
Déroulement de l'algorithme de Kahn

Exemple 6



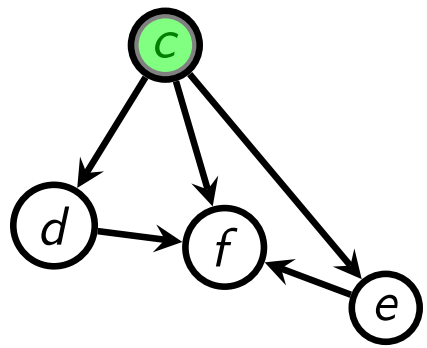
Déroulement de l'algorithme de Kahn

Exemple 6



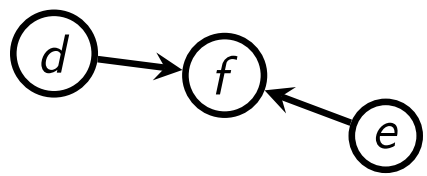
Déroulement de l'algorithme de Kahn

Exemple 6



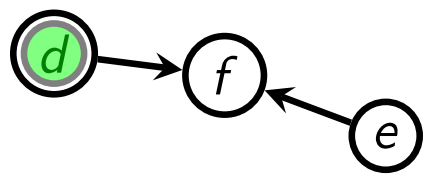
Déroulement de l'algorithme de Kahn

Exemple 6



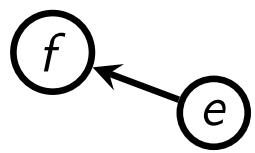
Déroulement de l'algorithme de Kahn

Exemple 6



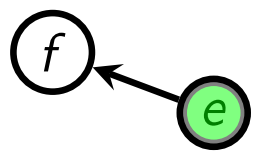
Déroulement de l'algorithme de Kahn

Exemple 6



Déroulement de l'algorithme de Kahn

Exemple 6



Déroulement de l'algorithme de Kahn

Exemple 6

Déroulement de l'algorithme de Kahn

Exemple 6

Implémentation de l'algorithme de Kahn

- L'algorithme de Kahn est simple à implémenter, mais il faut faire attention à sa complexité ;

Implémentation de l'algorithme de Kahn

- L'algorithme de Kahn est simple à implémenter, mais il faut faire attention à sa complexité ;
- La suppression répétée de sommets et d'arcs coûte cher ;

Implémentation de l'algorithme de Kahn

- L'algorithme de Kahn est simple à implémenter, mais il faut faire attention à sa complexité ;
- La suppression répétée de sommets et d'arcs coûte cher ;
- Au lieu de faire ça, on va stocker les degrés entrants à part et les décrémenter ;

L'algorithme de Kahn proprement dit

Algorithme 3 : KAHN(G)

Entrées : un graphe orienté acyclique G.
Sortie : les sommets de G ordonnés topologiquement.

```
   /* stocker les degrés entrants et les sources                    */
1  résultat ← liste();
2  sources ← pile();
3  degrés_entrants ← tableau(G.nombre_sommets(), 0);
4  pour chaque v ∈ G.sommets() faire
5  |   degrés_entrants[v] ← G.degré_entrant(v);
6  |   si degrés_entrants[v] = 0 alors sources.empiler(v);
   /* dépiler les sources, les ajouter au résultat, et empiler
      les nouvelles sources                                         */
7  tant que sources.pas_vide() faire
8  |   u ← sources.dépiler();
9  |   résultat.ajouter_en_fin(u);
10 |   pour chaque v ∈ G.successeurs(u) faire
11 |   |   degrés_entrants[v] ← degrés_entrants[v] −1;
12 |   |   si degrés_entrants[v] = 0 alors sources.empiler(v);
13 renvoyer résultat;
```

Complexité de l'algorithme de Kahn

- L'algorithme de Kahn se contente de parcourir le graphe, en maintenant le tableau degrés_entrants en $O(1)$ par opération ;

Complexité de l'algorithme de Kahn

- L'algorithme de Kahn se contente de parcourir le graphe, en maintenant le tableau degrés_entrants en $O(1)$ par opération ;
- Donc sa complexité dépend directement de l'implémentation du graphe :

Complexité de l'algorithme de Kahn

- L'algorithme de Kahn se contente de parcourir le graphe, en maintenant le tableau degrés_entrants en $O(1)$ par opération ;
- Donc sa complexité dépend directement de l'implémentation du graphe :
  - si l'on choisit une matrice d'adjacence, on a du $O(|V|^2)$.

Complexité de l'algorithme de Kahn

- L'algorithme de Kahn se contente de parcourir le graphe, en maintenant le tableau degrés_entrants en $O(1)$ par opération ;
- Donc sa complexité dépend directement de l'implémentation du graphe :
  - si l'on choisit une matrice d'adjacence, on a du $O(|V|^2)$.
  - si l'on choisit une liste d'adjacence, on a du $O(|V| + |A|)$.

Bibliographie

[1] A. B. Kahn.

Topological sorting of large networks.

Communications of the ACM, 5(11) :558–562, November 1962.

[2] Robert Endre Tarjan.

Depth-first search and linear graph algorithms.

SIAM Journal on Computing, 1(2) :146–160, 1972.