Algorithmique des graphes

3 — Graphes orientés suite

Marie-Pierre Béal (Cours d'Anthony Labarre)

Université Gustave Eiffel

Motivation pour le calcul de la fermeture transitive

- Il est parfois utile d'enrichir la structure d'un graphe de manière à pouvoir répondre à certaines requêtes sur ce graphe plus rapidement ;

Motivation pour le calcul de la fermeture transitive

- Il est parfois utile d'enrichir la structure d'un graphe de manière à pouvoir répondre à certaines requêtes sur ce graphe plus rapidement ;
- Par exemple : calcul répété de tous les descendants d'un sommet ;

Motivation pour le calcul de la fermeture transitive

- Il est parfois utile d'enrichir la structure d'un graphe de manière à pouvoir répondre à certaines requêtes sur ce graphe plus rapidement ;
- Par exemple : calcul répété de tous les descendants d'un sommet ;

Définition 1

La **fermeture transitive** d'un graphe orienté $G = (V, A)$ est le graphe orienté $G' = (V, A')$ avec $A' = \{(u, v) \mid v \in \text{ descendants}(u) \text{ dans } G\}$.

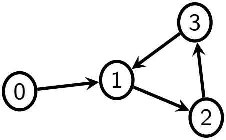
Motivation pour le calcul de la fermeture transitive

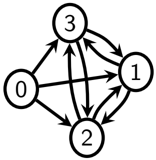
- Il est parfois utile d'enrichir la structure d'un graphe de manière à pouvoir répondre à certaines requêtes sur ce graphe plus rapidement ;
- Par exemple : calcul répété de tous les descendants d'un sommet ;

Définition 1

La **fermeture transitive** d'un graphe orienté $G = (V, A)$ est le graphe orienté $G' = (V, A')$ avec $A' = \{(u, v) \mid v \in$ descendants(u) dans $G\}$.

Exemple 1





G fermeture transitive de G

Calcul de la fermeture transitive

- Idée simple : rajouter un arc connectant chaque sommet à tous ses descendants ;

La complexité de cet algorithme se calcule simplement : on lance $|V|$ parcours, lesquels requièrent chacun $O(|V| + |A|)$ opérations, et l'on a donc un algorithme en $O(|V|^2 + |A||V|)$ opérations.

Calcul de la fermeture transitive

- Idée simple : rajouter un arc connectant chaque sommet à tous ses descendants ;
- Les descendants se calculent avec un simple parcours.

La complexité de cet algorithme se calcule simplement : on lance $|V|$ parcours, lesquels requièrent chacun $O(|V| + |A|)$ opérations, et l'on a donc un algorithme en $O(|V|^2 + |A||V|)$ opérations.

Calcul de la fermeture transitive

- Idée simple : rajouter un arc connectant chaque sommet à tous ses descendants ;
- Les descendants se calculent avec un simple parcours.

Algorithme 1 : FermetureTransitive(G)

Entrées : un graphe orienté connexe G.
Sortie : la fermeture transitive de G.

$F \leftarrow$ GrapheOrienté(G.sommets());
pour chaque $u \in G$.*sommets()* **faire**
  pour chaque $v \in$ LargeurOrienté(F, u) **faire**
    si $u \neq v$ **alors** F.ajouter_arc(u, v) ;
renvoyer F;

La complexité de cet algorithme se calcule simplement : on lance $|V|$ parcours, lesquels requièrent chacun $O(|V| + |A|)$ opérations, et l'on a donc un algorithme en $O(|V|^2 + |A||V|)$ opérations.

Connexité dans les graphes orientés

Définition 2

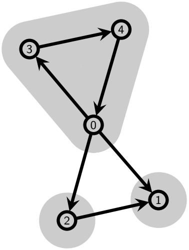
Une **composante fortement connexe** H d'un graphe orienté G est un ensemble maximal de sommets de G tel que pour toute paire de sommets u, v de H il existe un chemin de u à v et il existe un chemin de v à u.

Connexité dans les graphes orientés

Définition 2

Une **composante fortement connexe** H d'un graphe orienté G est un ensemble maximal de sommets de G tel que pour toute paire de sommets u, v de H il existe un chemin de u à v et il existe un chemin de v à u.

Exemple 2



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 mettre chaque sommet dont l’exploration **se termine** dans une pile.

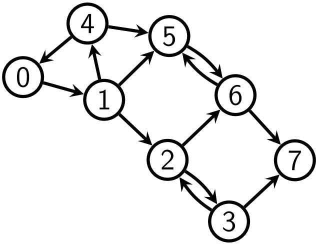
Parcours en profondeur “daté”

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l’exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

Parcours en profondeur “daté”

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l’exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

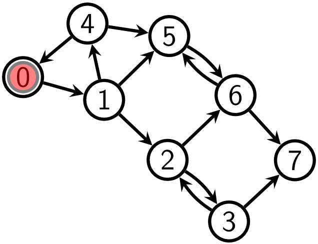
Exemple 3



Parcours en profondeur “daté”

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l’exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

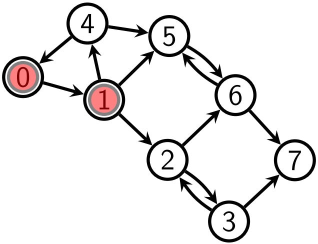
Exemple 3



Parcours en profondeur “daté”

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l'exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

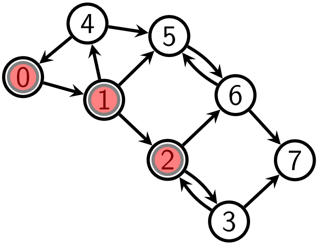
Exemple 3



Parcours en profondeur “daté”

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l’exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

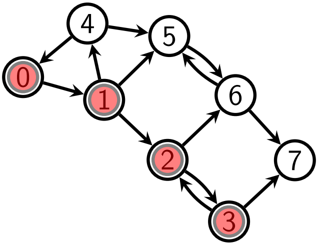
Exemple 3



Parcours en profondeur “daté”

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l’exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

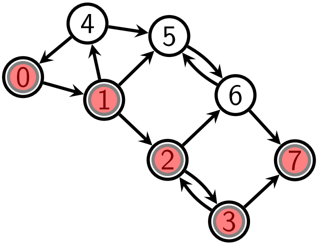
Exemple 3



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l'exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

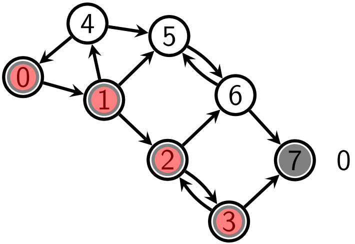
Exemple 3



Parcours en profondeur “daté”

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l’exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

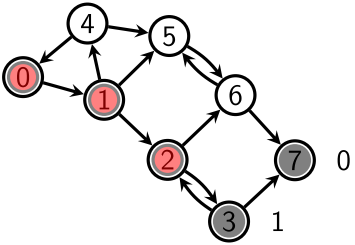
Exemple 3



Parcours en profondeur “daté”

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l’exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

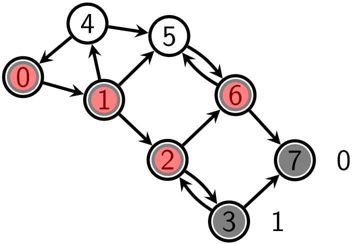
Exemple 3



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l'exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

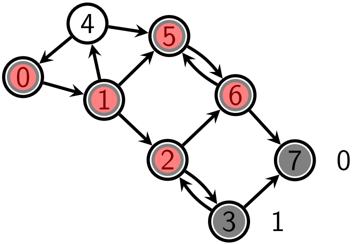
Exemple 3



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l'exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

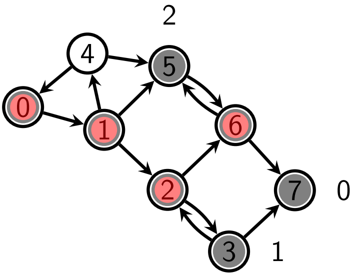
Exemple 3



Parcours en profondeur “daté”

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l’exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

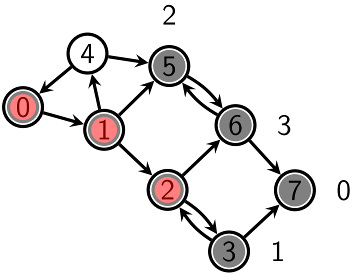
Exemple 3



Parcours en profondeur “daté”

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l’exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

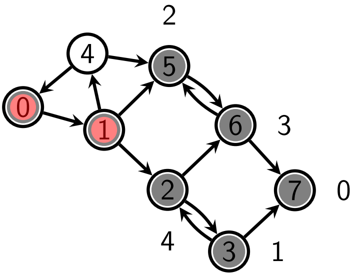
Exemple 3



Parcours en profondeur “daté”

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l’exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

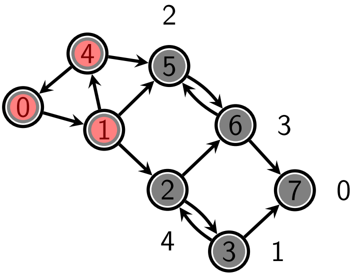
Exemple 3



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l'exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

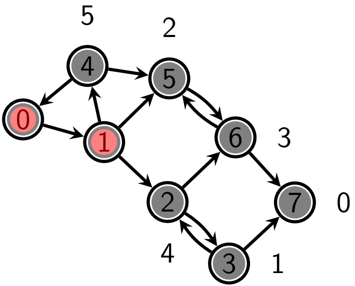
Exemple 3



Parcours en profondeur "daté"

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l'exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

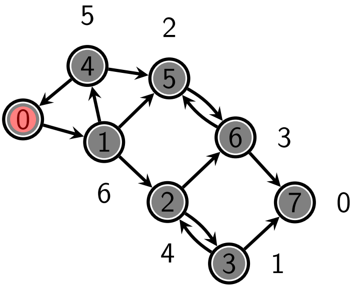
Exemple 3



Parcours en profondeur “daté”

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l’exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

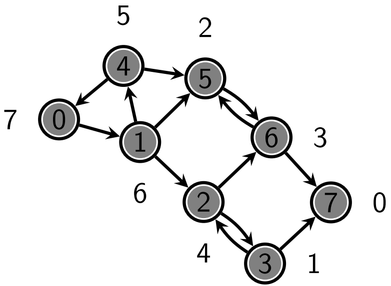
Exemple 3



Parcours en profondeur “daté”

- La première étape consiste à parcourir le graphe en profondeur ;
- Lors de ce parcours, on va mettre chaque sommet dont l’exploration **se termine** dans une pile.
- À la fin du parcours, tous les sommets seront donc dans la pile.

Exemple 3



L'algorithme du parcours en profondeur daté

Algorithme 2 : $\textsc{ProfondeurDates}(G)$

Entrées : un graphe orienté G
Résultat : pile contient les sommets de G dans l'ordre de la fin de visite

visité $\leftarrow$ tableau(G.nombre_sommets(), FAUX);
pile $\leftarrow$ pile() ; `// on empile un sommet à la fin de sa visite;`
pour chaque $u \in G$*.sommets()* **faire**
  si *visité[u]* $=$ FAUX **alors** $\textsc{ProfondeurDatesRec}(G, u,$ visité, pile) ;
renvoyer pile

L'algorithme du parcours en profondeur daté

Algorithme 3 : PROFONDEURDATESREC(G, départ, visité, pile)

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

Résultat : pile contient les sommets dans l'ordre la fin de visite

visité[départ] $\leftarrow$ VRAI ; `// marquer le début de l'exploration`;

pour chaque $v \in G$*.successeurs(départ)* **faire**

  si *visité[v]* $=$ FAUX **alors** PROFONDEURDATESREC(G, v, visité, pile) ;

pile.empiler(départ);

  `// on empile le sommet à la fin de l'exploration`;

Parcours renversé

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

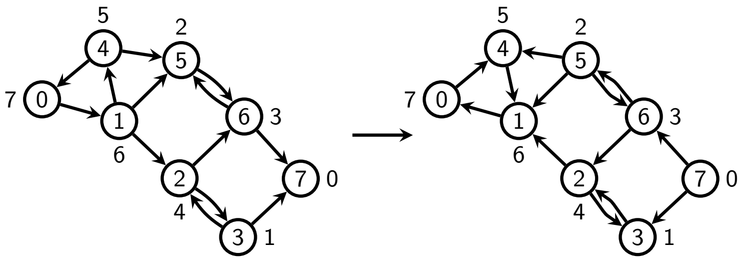
Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours en profondeur 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 en profondeur s'effectue par date de fin décroissante. Chaque arbre de la forêt obtenue est une composante fortement connexe de G.

Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours en profondeur 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 en profondeur s'effectue par date de fin décroissante. Chaque arbre de la forêt obtenue est une composante fortement connexe de G.

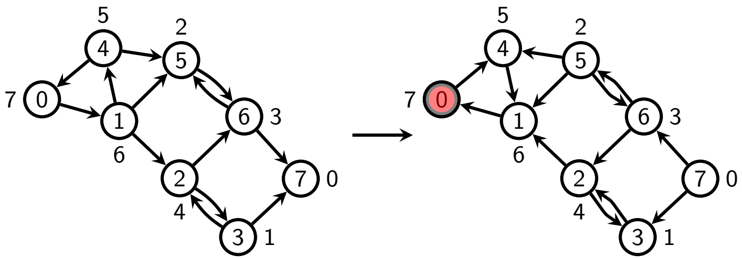
Exemple 4



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours en profondeur 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 en profondeur s'effectue par date de fin décroissante. Chaque arbre de la forêt obtenue est une composante fortement connexe de G.

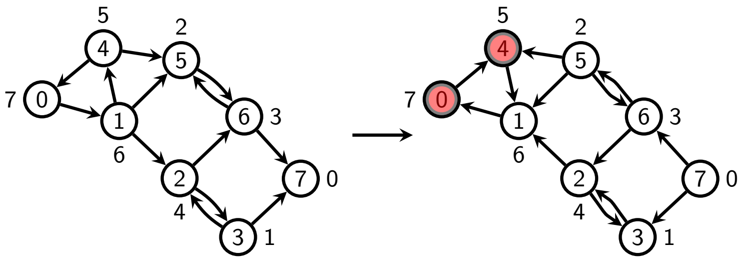
Exemple 4



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours en profondeur 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 en profondeur s'effectue par date de fin décroissante. Chaque arbre de la forêt obtenue est une composante fortement connexe de G.

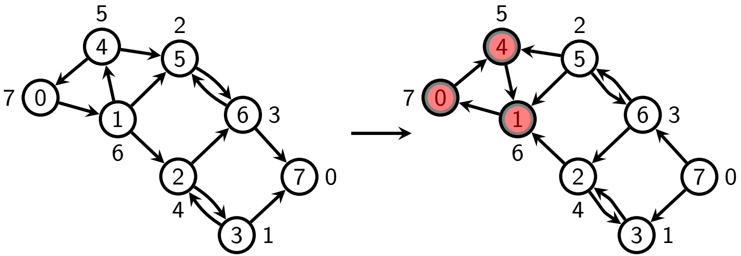
Exemple 4



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours en profondeur 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 en profondeur s'effectue par date de fin décroissante. Chaque arbre de la forêt obtenue est une composante fortement connexe de G.

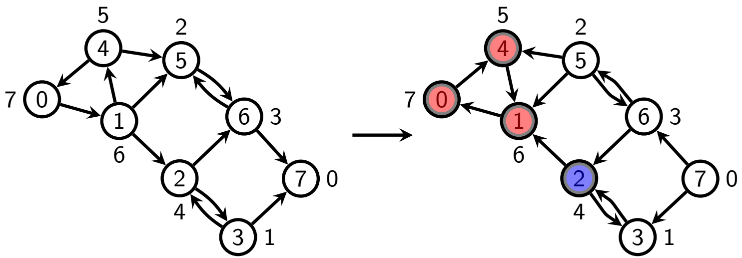
Exemple 4



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours en profondeur 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 en profondeur s'effectue par date de fin décroissante. Chaque arbre de la forêt obtenue est une composante fortement connexe de G.

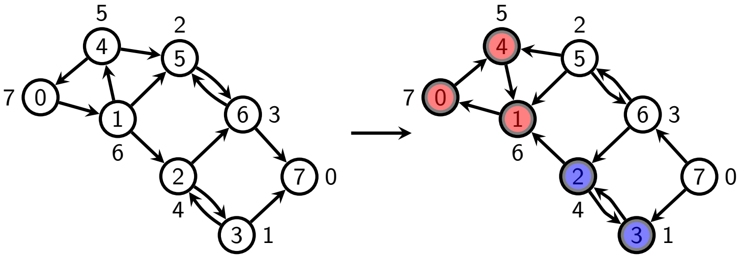
Exemple 4



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours en profondeur 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 en profondeur s'effectue par date de fin décroissante. Chaque arbre de la forêt obtenue est une composante fortement connexe de G.

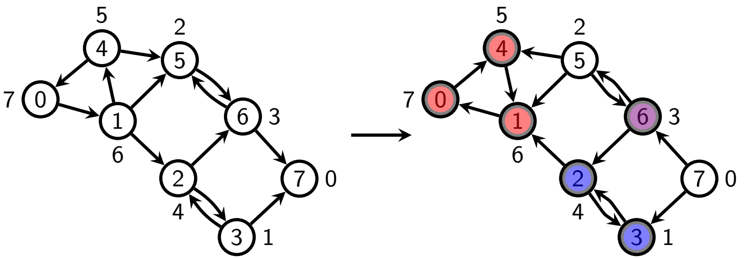
Exemple 4



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours en profondeur 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 en profondeur s'effectue par date de fin décroissante. Chaque arbre de la forêt obtenue est une composante fortement connexe de G.

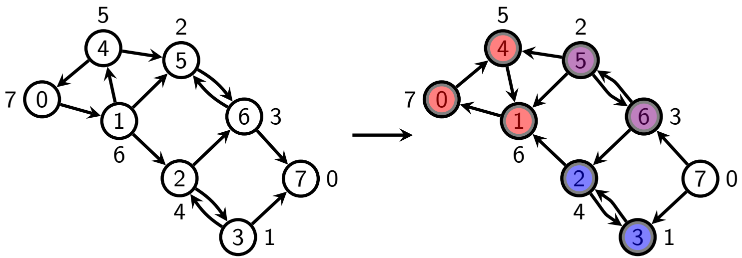
Exemple 4



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours en profondeur 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 en profondeur s'effectue par date de fin décroissante. Chaque arbre de la forêt obtenue est une composante fortement connexe de G.

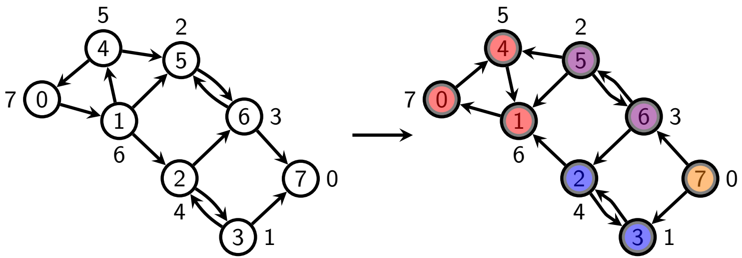
Exemple 4



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours en profondeur 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 en profondeur s'effectue par date de fin décroissante. Chaque arbre de la forêt obtenue est une composante fortement connexe de G.

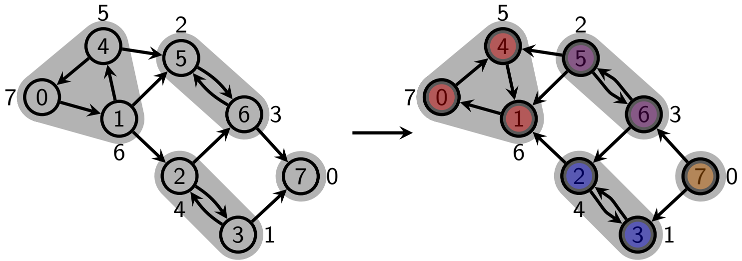
Exemple 4



Parcours renversé

- La deuxième étape de l'algorithme consiste à effectuer un parcours en profondeur 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 en profondeur s'effectue par date de fin décroissante. Chaque arbre de la forêt obtenue est une composante fortement connexe de G.

Exemple 4



L'algorithme de Kosaraju-Sharir

Algorithme 4 : KosarajuSharir(G)

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

```
CFC ← liste() ;          // une liste de listes de sommets;
pile ← ProfondeurDates(G);
G′ ← renverser_arcs(G);
visités ← tableau(G.nombre_sommets(), faux);
pour chaque v ∈ pile (de haut en bas) faire
    si ¬ visités[v] alors
        composante ← ProfondeurOrientéRec(G′, v, visités);
        CFC.ajouter_en_fin(composante);
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. 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. parcours de G' : $O(|V| + |A'|) = O(|V| + |A|)$;
- $\Rightarrow$ total : $O(|V| + |A|)$;

Complexité

- La complexité de l'algorithme de Kosaraju-Sharir se calcule aisément :
  1. parcours daté : $O(|V| + |A|)$;
  2. renversement : $O(|V| + |A|)$;
  3. parcours de G' : $O(|V| + |A'|) = O(|V| + |A|)$;
- $\Rightarrow$ total : $O(|V| + |A|)$;
- 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. parcours de G' : $O(|V| + |A'|) = O(|V| + |A|)$;
- $\Rightarrow$ total : $O(|V| + |A|)$;
- Peut-on faire mieux ?
  1. il existe un algorithme ne réalisant qu'un seul parcours [2] $\Rightarrow$ même complexité, mais plus rapide.

Preuve de l'algorithme de Kosaraju-Sharir

- On note r la racine d'une composante C lors de l'exploration de G'. Si u est dans C, il existe une chemin de r à u dans G' donc il existe une chemin de u à r dans G.

Preuve de l'algorithme de Kosaraju-Sharir

- On note r la racine d'une composante C lors de l'exploration de G'. Si u est dans C, il existe une chemin de r à u dans G' donc il existe une chemin de u à r dans G.
- On note date_fin(u) le numéro de u à la fin de sa visite lors de l'exploration de G.

Preuve de l'algorithme de Kosaraju-Sharir

- On note r la racine d'une composante C lors de l'exploration de G'. Si u est dans C, il existe une chemin de r à u dans G' donc il existe une chemin de u à r dans G.
- On note $\text{date\_fin}(u)$ le numéro de u à la fin de sa visite lors de l'exploration de G.
- On a donc $\text{date\_fin}(r) > \text{date\_fin}(u)$ pour u dans C distinct de r.

Preuve de l'algorithme de Kosaraju-Sharir

- Pour u dans C, il existe une chemin de u à r dans G. Si u est exploré avant r dans G alors on aurait date_fin(u) $>$ date_fin(r), contradiction.

Preuve de l'algorithme de Kosaraju-Sharir

- Pour u dans C, il existe une chemin de u à r dans G. Si u est exploré avant r dans G alors on aurait $\text{date\_fin}(u) > \text{date\_fin}(r)$, contradiction.
- Donc r est exploré avant u dans G. Comme $\text{date\_fin}(u) < \text{date\_fin}(r)$, u est exploré au cours de l'exploration de r dans G. Donc il existe un chemin de r à u dans G.

Preuve de l'algorithme de Kosaraju-Sharir

- Pour u dans C, il existe une chemin de u à r dans G. Si u est exploré avant r dans G alors on aurait date_fin$(u) >$ date_fin(r), contradiction.
- Donc r est exploré avant u dans G. Comme date_fin$(u) <$ date_fin(r), u est exploré au cours de l'exploration de r dans G. Donc il existe un chemin de r à u dans G.
- Donc pour tous les sommets u, v de C il existe un chemin de u à v et une chemin de v à u dans G. De plus il n'existe pas de sommet w dans une autre composante C' calculée avant C tel que il existe un chemin de w à r dans G'.

Preuve de l'algorithme de Kosaraju-Sharir

- Pour u dans C, il existe une chemin de u à r dans G. Si u est exploré avant r dans G alors on aurait date_fin(u) $>$ date_fin(r), contradiction.
- Donc r est exploré avant u dans G. Comme date_fin(u) $<$ date_fin(r), u est exploré au cours de l'exploration de r dans G. Donc il existe un chemin de r à u dans G.
- Donc pour tous les sommets u, v de C il existe un chemin de u à v et une chemin de v à u dans G. De plus il n'existe pas de sommet w dans une autre composante C' calculée avant C tel que il existe un chemin de w à r dans G'.
- Donc C est une composante fortement connexe (maximale) de G.

Le graphe des composantes fortement connexes

Définition 3

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 3

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 3

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 ;
- $A(H) = \{(C_i, C_j) \mid \exists u \in C_i, v \in C_j : (u, v) \in A(G)\}$.

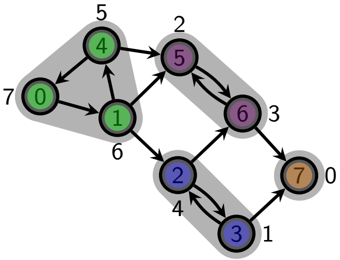
Le graphe des composantes fortement connexes

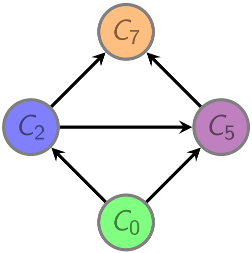
Définition 3

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 5





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 4

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

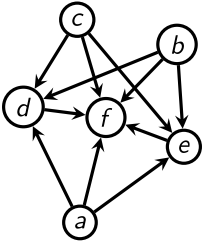
Ordres topologiques

Définition 4

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 6

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

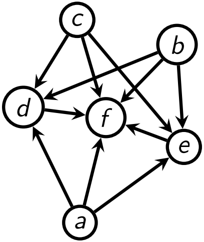
- Un algorithme simple et intuitif dû à Kahn [1] nous permet de produire un ordre topologique comme suit :
  1. placer les **sources** (sommets de degré entrant nul) en premier lieu ;
  2. retirer les sources du graphe et recommencer.

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.

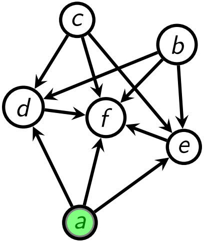
Déroulement de l'algorithme de Kahn

Exemple 7



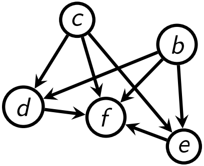
Déroulement de l'algorithme de Kahn

Exemple 7



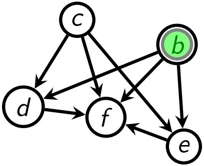
Déroulement de l'algorithme de Kahn

Exemple 7



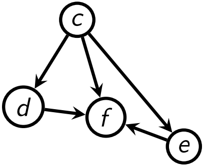
Déroulement de l'algorithme de Kahn

Exemple 7



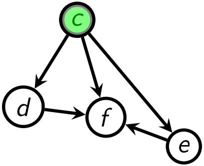
Déroulement de l'algorithme de Kahn

Exemple 7



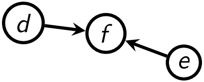
Déroulement de l'algorithme de Kahn

Exemple 7



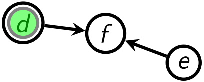
Déroulement de l'algorithme de Kahn

Exemple 7



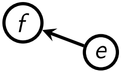
Déroulement de l'algorithme de Kahn

Exemple 7



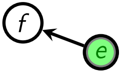
Déroulement de l'algorithme de Kahn

Exemple 7



Déroulement de l'algorithme de Kahn

Exemple 7



Déroulement de l'algorithme de Kahn

Exemple 7

Déroulement de l'algorithme de Kahn

Exemple 7

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 5 : KAHN(G)

Entrées : un graphe orienté acyclique G.
Sortie : les sommets de G ordonnés selon un ordre topologique.

```
/* stocker les degrés entrants et les sources                    */
résultat ← liste();
sources ← pile();
degrés_entrants ← tableau(G.nombre_sommets(), 0);
pour chaque v ∈ G.sommets() faire
    degrés_entrants[v] ← G.degré_entrant(v);
    si degrés_entrants[v] = 0 alors sources.empiler(v);
/* dépiler les sources, les ajouter au résultat, et empiler
   les nouvelles sources                                         */
tant que sources.pas_vide() faire
    u ← sources.dépiler();
    résultat.ajouter_en_fin(u);
    pour chaque v ∈ G.successeurs(u) faire
        degrés_entrants[v] ← degrés_entrants[v] −1;
        si degrés_entrants[v] = 0 alors sources.empiler(v);
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.