Chapitre 5
Les graphes orientés

Michaël Krajecki

Université de Reims Champagne-Ardenne
michael.krajecki@univ-reims.fr
http://www.univ-reims.fr/crestic

Graphes et algorithmes

Les graphes

- Bibliographie : *Structures de données et algorithmes*, A. Aho, J. Hopcroft, J. Ullman, InterEditions, 1989
- De nombreux problèmes supposent la modélisation de relations arbitraires entre des objets
- Les graphes offrent une solution dans ce cadre
- Les graphes représentent une généralisation des arbres
- On distingue les graphes orientés des graphes non-orientés.

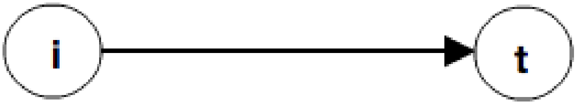
Leonhard Euler

- Le mathématicien suisse L. Euler propose le problème des sept ponts de Königsberg
- Trouver une promenade partant d'un point donné qui passe exactement une fois par chaque pont avant de revenir au point de départ
- Les applications des graphes sont nombreuses :
  - dans le domaine des transports de personnes ou de marchandises
  - dans les réseaux (routage, web sémantique)

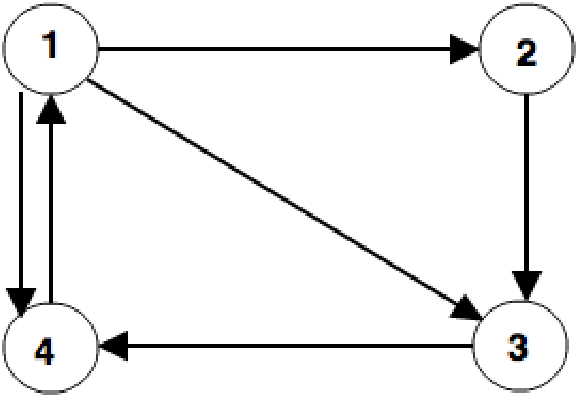
Graphe orienté

Définition (Graphe orienté)

Un graphe orienté G est défini par un ensemble de sommets S et un ensemble d'arcs A. Un arc est une paire ordonnée (i,t) où i désigne le sommet initial et t, le sommet terminal



Exemple : graphe à 4 sommets et 6 arcs



Chemin et circuit

Définition (Chemin et circuit)

Un chemin est défini dans un graphe orienté G(S,A) par une suite de sommets $s_1, s_2, \ldots, s_n$ où $\forall i, s_i \in S$ et $s_i \rightarrow s_{i+1} \in A$. Ce chemin part du sommet s_1 et aboutit au sommet s_n.
La longueur du chemin est défini par le nombre d'arcs empruntés, soit $n-1$. Le chemin de s à s est toujours défini et a une longueur nulle.
Un chemin est dit simple si $\forall i, \forall j \neq i, s_i \neq s_j$ sauf éventuellement pour i=1 et j=n.
Un circuit simple est un chemin simple de longueur strictement positive qui part d'un sommet s_i et aboutit au même sommet s_i.

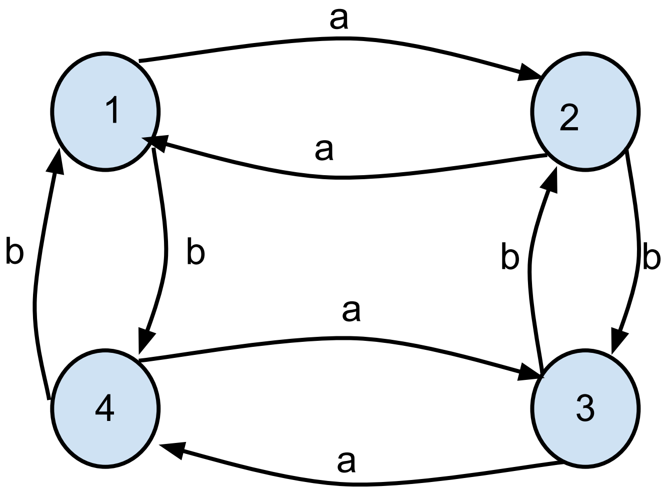
Graphe orienté étiqueté

- Il est souvent utile d'associer des informations supplémentaires à des arcs ou des sommets.
- Par exemple, on peut souhaiter définir la longueur d'un arc ou le nom d'un sommet. On peut également associer à un arc une capacité (comme un débit dans le cas de la modélisation d'un réseau informatique).

Définition (Graphe orienté étiqueté)

Un graphe orienté étiqueté G(S,A) est un graphe orienté où il est possible d'associer une valeur à un arc $a \in A$ ou à un sommet $s \in S$.

Exemple : expression régulière



Opérations élémentaires

- Les graphes peuvent être représentés à l'aide de plusieurs structures de données.
- Le choix d'une structure de données dépend des opérations qui seront appliqués aux graphes
- Les représentations classiques s'appuyent sur une matrice d'adjacence où sur les listes d'adjacence.
- Les principales fonctions de base sont :
  1. Insertion et suppression de sommets ou d'arcs, éventuellement étiquetés.
  2. Lecture des étiquettes associés aux sommets et aux arcs.
  3. Parcours de graphes le long de leurs arcs en suivant l'orientation de ceux-ci.

Parcours de graphes

- Il est souvent utile de d'ordonner les sommets adjacent à un autre pour pouvoir réaliser ensuite des schémas itératifs du type :

Pour chaque sommet t adjacent au sommet s **faire** suite-d-actions(t) **Fpour**

- Nous pouvons définir 2 fonctions de base qui renvoie un indice, c'est à dire, un numéro unique de sommet adjacent à s :
  1. $\mathit{Premier}(s)$: renvoie l'indice du premier sommet adjacent à s.
  2. $\mathit{Suivant}(s, i)$: renvoie l'indice immédiatement supérieur à i parmi les sommets adjacents à s

Plan

Représentation par matrice d'adjacence

- Soit le graphe $G = (S, A)$ où
- $S = \{1, 2, \ldots, n\}$
- Il est possible de définir une matrice de booléens M_a de dimension $n \times n$.
- $\forall i, j, M_a(i, j) = 1$ si et seulement si $\exists a \in A$ tel que $a = i \rightarrow j$.
- Pour des graphes étiquetés, il est possible de définir une matrice valuée
- Exemple (sur les 2 graphes précédents)
- Complexité en espace : $\Omega(n^2)$
- Plutôt pénalisant si le graphe est creux $(|S|^2 >> |A|)$

Plan

Représentation par listes d'adjacence

- Soit le graphe $G = (S, A)$ où
- $S = \{1, 2, \ldots, n\}$
- On associe un tableau $Sommet[1, n]$ où $Sommet[i]$ désigne la liste d'adjacence du sommet i
- La liste d'ajacence au sommet i contient le sommet j si et seulement si $\exists a \in A | a = i \rightarrow j$
- Avantage : complexité en mémoire en lien direct avec le nombre d'arcs
- Inconvénient : test d'existence d'un arc $i \rightarrow j$ en $O(n)$

Variante : représentation par tableau d'adjacence

- Soit le graphe $G = (S, A)$ où
- $S = \{1, 2, \ldots, n\}$ et $|A| = m$
- On associe un tableau $Sommet[1, n]$ où $Sommet[i]$ désigne l'indice j du tableau d'adjacence $Adjacence[1, n + m]$ du premier sommet adjacent à i
- Dans ce cas, $Adjacence[j + 1]$ contient l'indice du deuxième sommet adjacent à i et ainsi de suite
- Jusqu'à $Adjacence[j + k] = 0$ précisant qu'il n'y a plus de sommet adjacent à i
- Remarque : si le graphe est dynamique, la mise à jour de la table d'adjacence peut être coûteuse
- Exemple sur les deux graphes précédents

Plan

Recherche des plus courts chemins depuis une source

- Soit le graphe $G = (S, A)$ où chaque arc a est étiqueté par une valeur positive ou nulle appelé *poids*
- On peut associer cette valeur à une notion de distance entre deux sommets
- On définit $s \in S$ comme étant le noeud source du graphe G
- Problème à résoudre : définir tous les plus courts chemins partant de s
- Le coût d'un chemin est défini par la somme des *poids* le composant

Principe glouton

- Résolution par une méthode gloutonne
- Principe des algorithmes gloutons :
- A chaque étape, l'algorithme sélectionne la solution *localement* optimale
- Attention : en général, une approche gloutonne ne garantit pas la solution *globalement* optimale
- Exemple : les pièces de monnaies

Algorithme de Dijkstra

- Principe : tenir à jour un ensemble E contenant les sommets pour lesquels les distances les plus courtes sont déjà connues
- Initialisation : $E = \{s\}$
- A chaque étape : ajouter le sommet i le plus proche de s
- Comme tous les poids sont positifs ou nuls, il est toujours possible de construire :
  - un chemin minimal depuis s vers i ne passant que par des sommets de E
  - Ce chemin est appelé *raccourci*
- On définit le tableau D contenant toutes les longueurs des meilleurs raccourcis connus
- Quand E contient tous les sommets de S, D contient toutes les distances les plus courtes depuis la source s

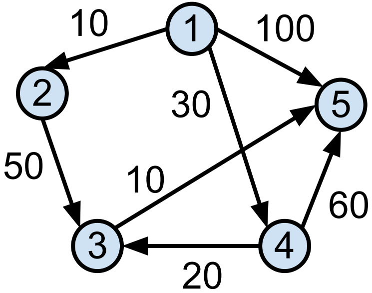
Algorithme de Dijkstra

- Soit le graphe $G = (S, A)$ où
- $S = \{1, 2, \ldots, n\}$ et la source $s = 1$
- On définit la matrice des poids $P[n \times n]$ où
- $P[i, j] = p_{ij}$ avec p_{ij} est l'étiquette associée à l'arc $i \to j$
- S'il n'existe pas d'arc $i \to j$ alors $P[i, j] = \infty$

Algorithme de Dijkstra

Procédure *Dijkstra.*
Début
  $E \leftarrow \{ 1\}$
  Pour *i* ***de*** *2 à n* ***faire***
    $D[i] \leftarrow P[1,i]$
  Fpour
  Pour *i* ***de*** *1 à n-1* ***faire***
    choisir un sommet t de S-E tel que D[t] soit le minimum
    $E \leftarrow E + \{ t\}$
    Pour *chaque sommet s de S-E* ***faire***
      $D[s] \leftarrow \min(D[s], D[t] + P[t,s])$
    Fpour
  Fpour
Fin.

Exemple



Reconstruire le plus court chemin depuis s

- Pour reconstruire le plus court chemin depuis s, il faut définir en plus :
- Un nouveau tableau $C[2 \ldots n]$ de sommets
- $C[i] = j$ signifie alors que le sommet j précède immédiatement i dans le plus court chemin depuis la source $s(= 1)$ vers j
- Au départ : $C[i] \leftarrow 1$ pour tout $i \neq 1$
- La valeur de $C[i]$ est mise à jour lors du calcul de $D[i]$

Algorithme de Dijkstra

Procédure *Dijkstra.*
Début
E ←{ 1}
Pour *i* ***de*** *2 à n* ***faire***
D[i] ←P[1,i] ; C[i] ←1
Fpour
Pour *i* ***de*** *1 à n-1* ***faire***
choisir un sommet t de S-E tel que D[t] soit le minimum
E ←E + { t}
Pour *chaque sommet s de S-E* ***faire***
D[s] ←min(D[s], D[t] + P[t,s])
Si *D[t]+P[t,i] < D[i]* ***alors*** *C[i] ←t*
Fpour
Fpour
Fin.

Optimalité de l'algorithme de Dijkstra

- Montre l'efficacité d'une approche gloutonne dans certains cas
- Pour Dijkstra, l'optimalité locale induit l'optimalité globale grâce à l'inégalité triangulaire vérifiée (car $P[i, j] > 0$)
- Supposons qu'il existe un chemin entre la source s et t plus court que le chemin raccourci calculé à l'aide de E
- Ce chemin emprunte alors un sommet x qui n'est pas dans E
- Dans ce cas, le chemin entre s et x est le plus court chemin possible
- Donc x est nécessairement dans E, en contradiction avec l'hypothèse de départ

Complexité de l'algorithme de Dijkstra

- Supposons $G = (S, A)$ où $|S| = n$ et $|A| = a$
- En utilisant une matrice d'ajacence, la complexité est en $O(n^2)$
- En utilisant des listes d'adjacence, si a est petit devant n
- Il est utile d'utiliser une file de priorité (voir TD) pour ordonner les sommets dans $S - E$
- La mise à jour du tableau D est alors effectuée en $O(a \log n)$

Plan

Recherche des plus courts chemins entre tous le sommets

- Problème de distance minimum entre 2 sommets (noté $DM2S$)
- Exemple : table des distances entre les principales villes d'une région ou d'un pays
- Formellement :
  - Soit le graphe $G = (S, A)$ où
  - On associe à tout arc $d \rightarrow a$ un poids $P[d, a] > 0$
  - Le problème $DM2S$ consiste à déterminer pour chaque couple de sommets ordonnés (d, a) le chemin de plus faible poids entre d et a
- Première solution : appliquer l'algorithme de Dijkstra à tous les sommets de S
- Complexité de l'ordre $O(n^3)$

Algorithme de Floyd

- L'algorithme de Floyd apporte une réponse plus directe au problème $DM2S$
- Soit le graphe $G = (S, A)$ où
- $S = \{1, 2, \dots, n\}$
- On définit la matrice des poids $P[n \times n]$ où
- $P[i, j] = p_{ij}$ avec p_{ij} est l'étiquette associée à l'arc $i \rightarrow j$
- S'il n'existe pas d'arc $i \rightarrow j$ alors $P[i, j] = \infty$
- L'algorithme de Floyd utilise une matrice $L[n \times n]$
- Initialisation : $L[i, j] = P[i, j]$

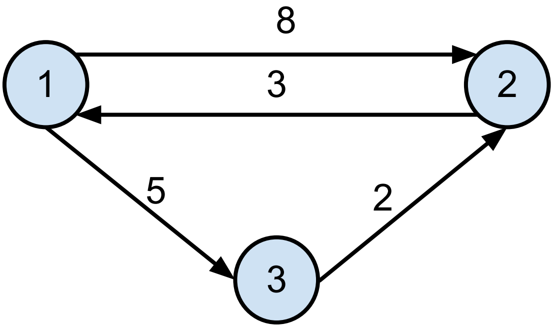
Algorithme de Floyd

- L'algorithme comporte n itérations
- A chaque itération, il met à jour la matrice L
- A l'itération k :
  - l'élément $L[i,j]$ est égal au plus court chemin de i à j
  - n'empruntant que des sommets d'indice inférieur à k
  - mise à jour de L :
  - $L_k[i,j] = min \begin{cases} L_{k-1}[i,j] \\ L_{k-1}[i,k] + L_{k-1}[k,j] \end{cases}$
- Comme aucun indice, ligne ou colonne, k ne peut être modifié à l'itération k
- Il est possible d'utiliser une unique matrice L

Algorithme de Floyd

Procédure *Floyd.*
Début
  Pour i ***de*** *1* ***à*** n ***faire***
    Pour j ***de*** *1* ***à*** n ***faire***
      $L[i,j] \leftarrow P[i,j]$
  Pour i ***de*** *1* ***à*** n ***faire***
    $L[i,i] \leftarrow 0$
  Pour k ***de*** *1* ***à*** n ***faire***
    Pour i ***de*** *1* ***à*** n ***faire***
      Pour j ***de*** *1* ***à*** n ***faire***
        Si $L[i,k] + L[k,j] < L[i,j]$ ***alors*** $L[i,j] \leftarrow L[i,k] + L[k,j]$
Fin.

Exemple



Complexité et chemins

- Complexité en $O(n^3)$
- Comparable à Dijkstra, sauf avec liste de priorité $O(a \log n)$
- Comme pour Dijkstra, l'algorithme de Floyd calcule les plus courts chemins
- Mais ne mémorise pas explicitement les sommets empruntés
- Nouvelle matrice $C[i,j]$:
  - $C[i,j] = k$ si k est le dernier sommet qui a permis à Floyd d'améliorer le plus court chemin entre i et j
  - Si $C[i,j] = 0$, nous pouvons en déduire que le plus court chemin entre i et j et l'arc $i \to j$

Algorithme de Floyd

Procédure *Floyd.*
Début
  Pour i ***de*** *1* *à* n ***faire***
    Pour j ***de*** *1* *à* n ***faire***
      $L[i,j] \leftarrow P[i,j]$
  Pour i ***de*** *1* *à* n ***faire***
    $L[i,i] \leftarrow 0$
  Pour k ***de*** *1* *à* n ***faire***
    Pour i ***de*** *1* *à* n ***faire***
      Pour j ***de*** *1* *à* n ***faire***
        Si $L[i,k] + L[k,j] < L[i,j]$ ***alors***
          $L[i,j] \leftarrow L[i,k] + L[k,j]$
          $C[i,j] \leftarrow k$
        Fsi
Fin.

Tous les sommets d'un plus court chemin

- Pour connaître tous les sommets empruntés lors du plus court chemin entre i et j
- Il est possible de définir une fonction récursive

Fonction *Chemin(i,j :sommet).*
variable *: k :sommet*
Début
  $k \leftarrow C[i,j]$
  Si $k \neq 0$ ***alors***
    Chemin(i,k)
    Affiche(k)
    Chemin(k,i)
  Fsi
Fin.

Plan

Fermeture transitive

- Il est parfois utile de savoir s'il existe un chemin entre i et j
- L'algorithme de Floyd peut être modifié en ce sens
- Il suffit d'utiliser une matrice de booléens, appelée *Fermeture Transitive*
- Cet algorithme est connu sous le nom d'algorithme de Warshall

Algorithme de Warshall

Procédure *Warshall.*
Début
  Pour i ***de*** *1* *à* n ***faire***
    Pour j ***de*** *1* *à* n ***faire***
      Si $i \rightarrow j \in A$ ***alors*** $T[i,j] \leftarrow Vrai$
  Pour k ***de*** *1* *à* n ***faire***
    Pour i ***de*** *1* *à* n ***faire***
      Pour j ***de*** *1* *à* n ***faire***
        Si $!T[i,j]$ ***alors*** $T[i,j] \leftarrow T[i,k]$ *et* $T[k,j]$
Fin.

Plan

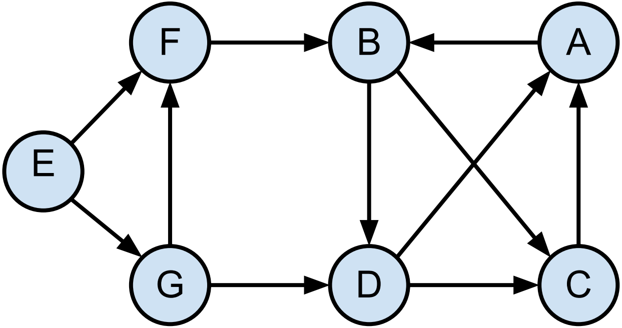
Parcours en profondeur d'abord

Procédure *PacourirGraphe.*
Début
  Pour s **de** 1 *à* n ***faire***
    $etat[s] \leftarrow inexploré$
  Pour s **de** 1 *à* n ***faire***
    Si $etat[s] = inexploré$ ***alors*** *RechercheProfondeur(s)*
Fin.

Parcours en profondeur d'abord

Procédure *RechercheProfondeur(s :sommet).*
variable : *t : sommet*
Début
  $etat[s] \leftarrow$*exploré*
  Pour *chaque sommet t tel que* $s \rightarrow t \in A$ ***faire***
    Si $etat[t] =$ *inexploré* ***alors***
      RechercheProfondeur(t)
Fin.

Exemple



Plan

Forêt de recouvrement des recherches

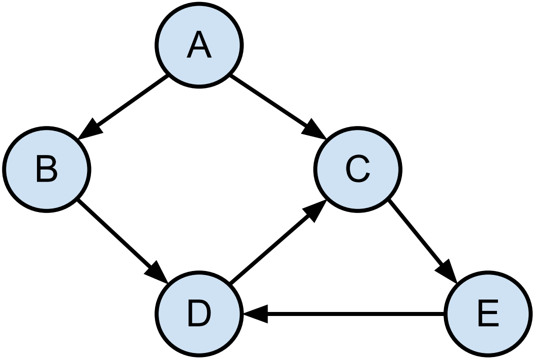
- Le parcours en profondeur permet de découvrir des *arcs d'arbre*
- Il s'agit des arcs empruntés pour découvrir un nouveau sommet
- L'ensemble des arcs d'arbre définissent une *forêt de recouvrement* de la recherche en profondeur
- Il existe un autre type d'arc : les arcs *rétrograde*
- Ils relient un sommet à l'un de ses ancètres dans l'arborescence

Plan

Graphe Orienté Sans Circuit

- Un graphe orienté sans circuit est un cas particulier de graphe orienté
- C'est aussi une généralisation des arbres
- Ils sont utiles pour représenter des expressions arithmétiques répétant des termes
- Comment détecter des circuits ?
- En réalisant un parcours en profondeur d'abord
- S'il existe un arc rétrograde, il existe un circuit (et inversement)

Exemple



Plan

Tri topologique

- Pour obtenir la licence d'Informatique, vous devez valider chaque année
- Pour valider chaque année, vous devez valider chaque semestre
- Pour valider un semestre, vous devez valider des UE
- Pour valider une UE, vous devez vous y inscrire (et réussir les examens)
- Pour vous inscrire dans une UE, vous devez avoir suivi les UE prérequises
- Un graphe orienté sans cycle permet de modéliser un tel ensemble de tâches
- Il existe un arc entre les tâches i et j si la tâche i est un prérequis pour j

Tri topologique

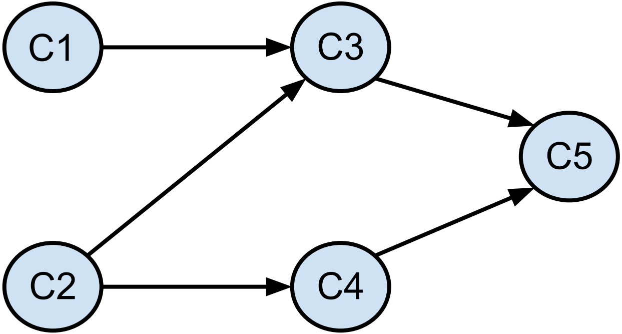
- Dans quel ordre dois-je valider les UE pour obtenir ma licence ?
- Pour répondre à cette question : utilisation d'un *tri topologique*
- Le tri topologique définit un ordre linéaire sur les sommets d'un GOSC tel que
- S'il existe un arc $i \to j$, le sommet i apparaît avant j dans l'ordre linéaire

Tri Topologique

- Les sommets accessibles depuis s sont affichés dans l'ordre inverse du tri topologique

Procédure *TriTopologique(s :sommet).*
variable *: t : sommet*
Début
  etat$[s] \leftarrow$*exploré*
  Pour *chaque sommet t tel que* $s \rightarrow t \in A$ ***faire***
    Si *etat*$[t] =$ *inexploré* ***alors***
      TriTopologique(t)
  Affiche(s)
Fin.

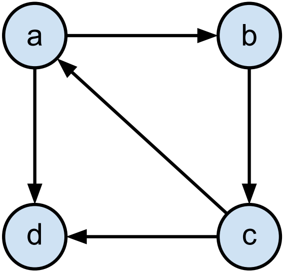
Exemple



Plan

Composantes fortement connexes

- Un ensemble maximal de sommets CC dans lequel il existe un chemin pour tout sommets i et j pris dans CC définit une *composante fortement connexe*
- Le parcours en profondeur d'abord permet de construires les composantes fortement connexes d'un graphe orienté



Composantes fortement connexes

- Soit $G = (S, A)$ un graphe orienté
- Il est possible de construire sur S les classes d'équivalence S_i pour la relation
- "Les sommets s et t sont équivalents si et seulement si il existe un chemin allant de s à t et un chemin allant de t à s"
- Soient A_i les ensembles d'arcs dont les 2 extrémités sont dans les S_i
- Les graphes $G_i = (S_i, A_i)$ représentent les composantes fortement connexes
- Si G ne comporte qu'une composante fortement connexe, il est dit fortement connexe

Composantes fortement connexes

- Il existe des arcs dit *intercomposante*, c'est à dire partant d'une composante fortement connexe et aboutissant dans une autre composante fortement connexe
- Il est possible de définir un graphe *réduit* de G à partir de ses composantes fortement connexes
- Il suffit de regrouper tous les sommets d'une même composante fortement connexe en un seul
- Ne sont alors représentés que les arcs intercomposantes
- Un graphe réduit ne peut comporter de circuit

Construire les composantes fortement connexes

1. Numéroter les sommets de G lors d'une recherche en profondeur
2. Construire le graphe inverse G_r où tous les arcs de G sont inversés dans G_r
3. Effectuer une recherche en profondeur d'abord sur G_r depuis le sommet de rang le plus élevé. Recommencer jusqu'à explorer tous les sommets

- Chaque arbre de la forêt obtenue est une composante fortement connexe

Exemple