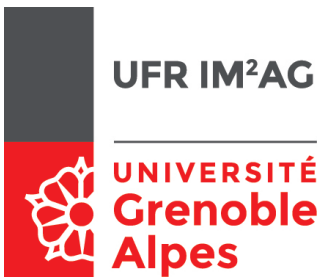
Algorithmique et Analyse d'Algorithmes

L3 Info

Cours 10 : Algorithmique de graphes

Arbre couvrant

Benjamin Wack



2017- 2018

La dernière fois

- Codage
- Entropie
- Algorithme de Huffman

Aujourd'hui

- Rappels sur les graphes
- Aspects algorithmiques
- Problèmes d'optimisation
- Arbre couvrant

Plan

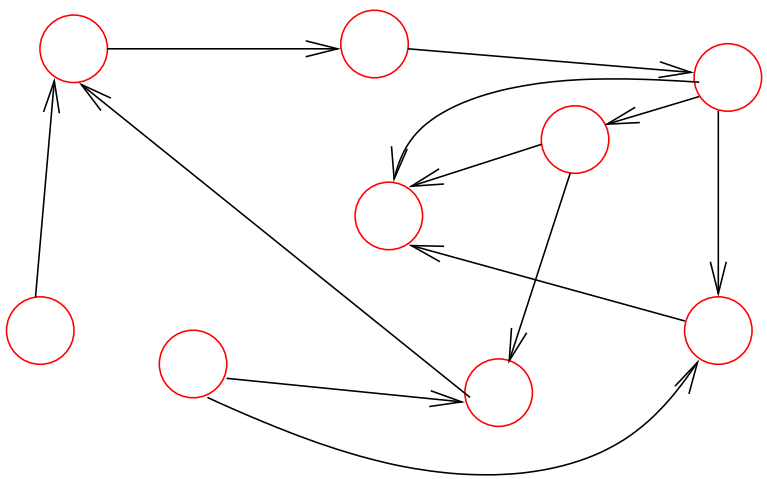
Plan

Notions de base

Graphe

Un graphe est un couple $G = (X, R)$:

- X est l'ensemble des sommets
- R est l'ensemble des arcs : c'est une relation binaire sur X (ensemble de couples de X).



Notions de base

Graphe

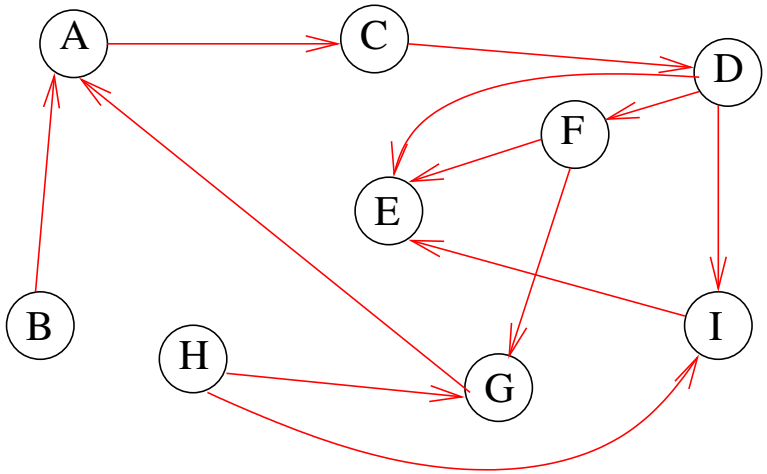
Un graphe est un couple $G = (X, R)$:

- X est l'ensemble des sommets
- R est l'ensemble des arcs : c'est une relation binaire sur X (ensemble de couples de X).

Arc

Un arc est un couple de sommets (x, y).

- x est appelé origine de l'arc ;
- y est appelé extrémité.



Notions de base

Graphe

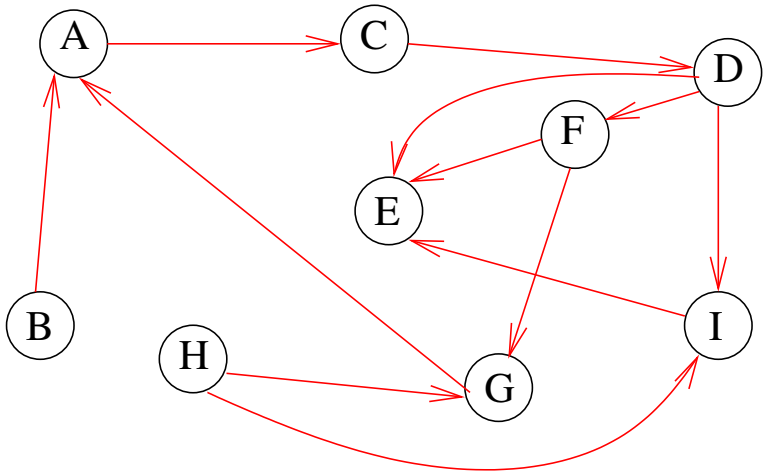
Un graphe est un couple $G = (X, R)$:

- X est l'ensemble des sommets
- R est l'ensemble des arcs : c'est une relation binaire sur X (ensemble de couples de X).

Arc

Un arc est un couple de sommets (x, y).

- x est appelé origine de l'arc ;
- y est appelé extrémité.

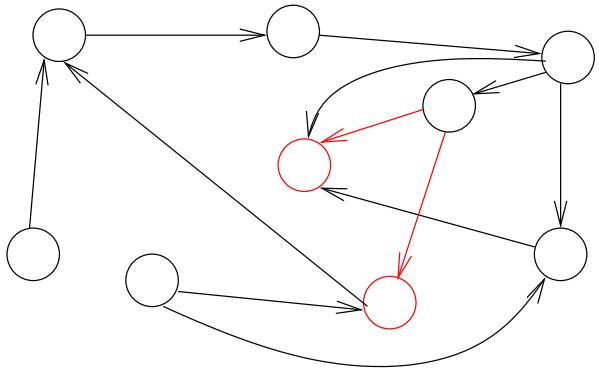


Sur l'exemple : $R = \{(B, A), (A, C), (C, D), (D, E), (D, F), (D, I), \ldots\}$

Voisinages

Successeur

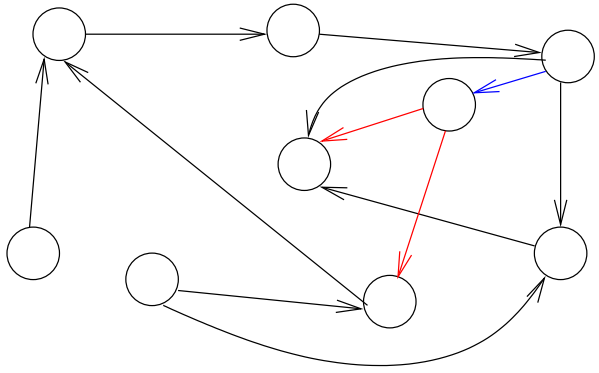
On appelle successeur d'un sommet x tout sommet y tel que (x, y) est un arc du graphe.



Voisinages

Successeur

On appelle successeur d'un sommet x tout sommet y tel que (x, y) est un arc du graphe.



Degré d'un sommet

Le degré d'un sommet x est le nombre d'arcs dont x est origine ou extrémité :

- le demi-degré extérieur est le nombre d'arcs dont x est origine ;
- le demi-degré intérieur est le nombre d'arcs dont x est extrémité.

Degré d'un graphe

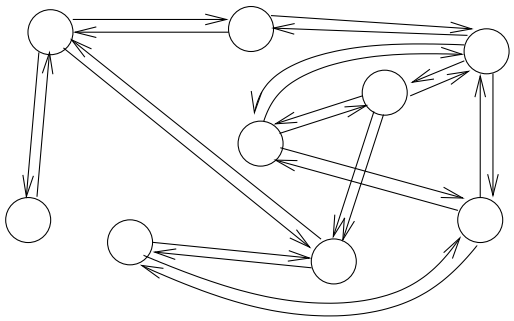
Le degré d'un graphe est le maximum des degrés de ses sommets.

Graphe non orienté

Graphe non orienté

Si R est symétrique
$((x, y) \in R \Leftrightarrow (y, x) \in R)$ alors :

- le graphe est dit non orienté ;
- on appelle arête un couple de sommets, qui correspond à **deux** arcs.

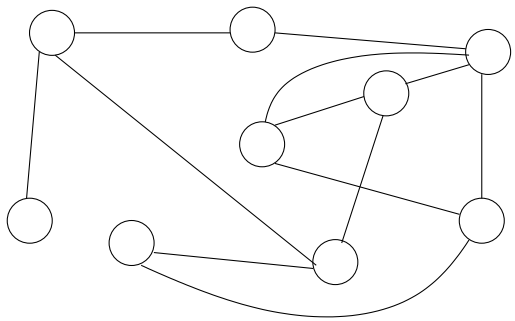


Graphe non orienté

Graphe non orienté

Si R est symétrique
$((x, y) \in R \Leftrightarrow (y, x) \in R)$ alors :

- le graphe est dit non orienté ;
- on appelle arête un couple de sommets, qui correspond à **deux** arcs.

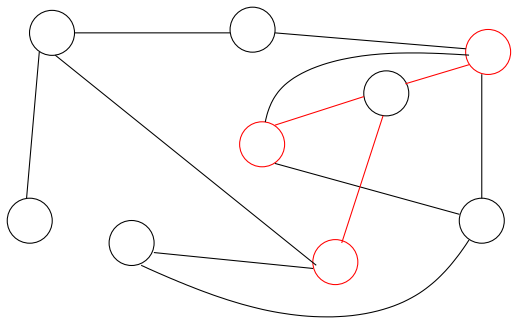


Graphe non orienté

Graphe non orienté

Si R est symétrique
$((x, y) \in R \Leftrightarrow (y, x) \in R)$ alors :

- le graphe est dit non orienté ;
- on appelle arête un couple de sommets, qui correspond à **deux** arcs.



Dans un graphe non orienté :

- on parle de voisin plutôt que de successeur ;
- on ne compte bien sûr qu'une fois chaque arête dans le degré.

Dans toute la suite, on ne considère que des graphes non orientés, mais la plupart des notions s'adaptent facilement.

Étiquettes, pondération

Dans un graphe il est possible d'étiqueter :

- les sommets
- et / ou les arêtes

au moyen d'une fonction de X (respectivement de R) dans un ensemble d'étiquettes donné.

Étiquettes, pondération

Dans un graphe il est possible d'étiqueter :

- les sommets
- et / ou les arêtes

au moyen d'une fonction de X (respectivement de R) dans un ensemble d'étiquettes donné.

Si les étiquettes sont à valeur numérique (entier, réel...) elles peuvent représenter un poids (coût, valeur...) pour les sommets ou les arêtes : on parle alors de graphe pondéré.

Applications

- Réseau (routier, de communication)
  - Calculer un itinéraire
  - Identifier les goulots d'étranglement

Applications

- Réseau (routier, de communication)
  - Calculer un itinéraire
  - Identifier les goulots d'étranglement
- Conflits
  - Chaque sommet représente un processus, une activité...
  - Chaque arête représente une compétition pour une ressource
  - Déterminer le nombre de ressources nécessaires, ou les activités compatibles

Applications

- Réseau (routier, de communication)
  - Calculer un itinéraire
  - Identifier les goulots d'étranglement
- Conflits
  - Chaque sommet représente un processus, une activité...
  - Chaque arête représente une compétition pour une ressource
  - Déterminer le nombre de ressources nécessaires, ou les activités compatibles
- Dépendances (avec un graphe *orienté*)
  - Chaque sommet représente une tâche
  - L'origine de chaque arc doit être réalisée avant son extrémité
  - Détecter une incohérence, calculer un ordre approprié

Applications

- Réseau (routier, de communication)
  - Calculer un itinéraire
  - Identifier les goulots d'étranglement
- Conflits
  - Chaque sommet représente un processus, une activité...
  - Chaque arête représente une compétition pour une ressource
  - Déterminer le nombre de ressources nécessaires, ou les activités compatibles
- Dépendances (avec un graphe *orienté*)
  - Chaque sommet représente une tâche
  - L'origine de chaque arc doit être réalisée avant son extrémité
  - Détecter une incohérence, calculer un ordre approprié
- ...

De façon générale toute relation, orientée ou non, se traduit par un graphe qui permet ensuite de raisonner algorithmiquement sur cette relation.

Plan

Type abstrait

Nom Graphe, Sommet

Utilise Etiquette, bool, int, ensemble

Opérations

$$\begin{aligned} \text{GrapheVide} &: () \to \mathit{Graphe} \\ \text{AjouterSommet} &: \mathit{Etiquette} \times \mathit{Graphe} \to \mathit{Graphe} \\ \text{AjouterArete} &: \mathit{Sommet} \times \mathit{Sommet} \times \mathit{Graphe} \\ &\quad \to \mathit{Graphe} \end{aligned}$$

Type abstrait

Nom Graphe, Sommet
Utilise Etiquette, bool, int, ensemble
Opérations

$$
\begin{array}{rcl}
\text{GrapheVide} & : & () \to \mathit{Graphe} \\
\text{AjouterSommet} & : & \mathit{Etiquette} \times \mathit{Graphe} \to \mathit{Graphe} \\
\text{AjouterArete} & : & \mathit{Sommet} \times \mathit{Sommet} \times \mathit{Graphe} \\
 & & \quad \to \mathit{Graphe} \\
 & & \\
\text{nbSommets} & : & \mathit{Graphe} \to \mathit{int} \\
\text{ensSommets} & : & \mathit{Graphe} \to \mathit{ensemble}(\mathit{Sommet}) \\
\text{existeArete} & : & \mathit{Sommet} \times \mathit{Sommet} \times \mathit{Graphe} \to \mathit{bool} \\
\text{ensVoisins} & : & \mathit{Sommet} \times \mathit{Graphe} \\
 & & \quad \to \mathit{ensemble}(\mathit{Sommet})
\end{array}
$$

+ suppressions, étiquettes des arêtes...

Type abstrait

Nom Graphe, Sommet

Utilise Etiquette, bool, int, ensemble

Opérations

$$
\begin{array}{rcl}
\text{GrapheVide} & : & () \to \mathit{Graphe} \\
\text{AjouterSommet} & : & \mathit{Etiquette} \times \mathit{Graphe} \to \mathit{Graphe} \\
\text{AjouterArete} & : & \mathit{Sommet} \times \mathit{Sommet} \times \mathit{Graphe} \\
 & & \quad \to \mathit{Graphe} \\
 & & \\
\text{nbSommets} & : & \mathit{Graphe} \to \mathit{int} \\
\text{ensSommets} & : & \mathit{Graphe} \to \mathit{ensemble}(\mathit{Sommet}) \\
\text{existeArete} & : & \mathit{Sommet} \times \mathit{Sommet} \times \mathit{Graphe} \to \mathit{bool} \\
\text{ensVoisins} & : & \mathit{Sommet} \times \mathit{Graphe} \\
 & & \quad \to \mathit{ensemble}(\mathit{Sommet})
\end{array}
$$

\+ suppressions, étiquettes des arêtes...

Préconditions Pas de difficulté particulière

Axiomes AjouterArete(x,y,G) rend à la fois x voisin de y et inversement.

Parcours de graphe

Principes communs

On choisit un sommet (origine) par lequel commencer le parcours.
On traite chaque sommet **une seule fois** (attention aux cycles !).

Parcours de graphe

Principes communs

On choisit un sommet (origine) par lequel commencer le parcours.
On traite chaque sommet **une seule fois** (attention aux cycles !).

Parcours en profondeur

On suit un chemin aussi « loin » que possible.
Si nécessaire on revient en arrière pour explorer d'autres chemins.

Parcours de graphe

Principes communs

On choisit un sommet (origine) par lequel commencer le parcours.
On traite chaque sommet **une seule fois** (attention aux cycles !).

Parcours en profondeur

On suit un chemin aussi « loin » que possible.
Si nécessaire on revient en arrière pour explorer d'autres chemins.

Parcours en largeur

On procède par « cercles concentriques » autour d'un sommet x : on traite **tous** ses voisins, puis les sommets situés à une distance 2, etc.

Parcours de graphe

Principes communs

On choisit un sommet (origine) par lequel commencer le parcours.
On traite chaque sommet **une seule fois** (attention aux cycles !).

Parcours en profondeur

On suit un chemin aussi « loin » que possible.
Si nécessaire on revient en arrière pour explorer d'autres chemins.

Parcours en largeur

On procède par « cercles concentriques » autour d'un sommet x : on traite **tous** ses voisins, puis les sommets situés à une distance 2, etc.

Remarque

Parfois on ne parvient pas à visiter tout à partir de l'origine choisie.
Dans ce cas il faut « relancer » à partir d'une autre origine.

Notion de marquage

Les structures linéaires possèdent un ordre de parcours naturel (induit par la structure).

Notion de marquage

Les structures linéaires possèdent un ordre de parcours naturel (induit par la structure).

Dans les arbres on dispose d'une liberté supplémentaire (profondeur ou largeur d'abord...) mais on dispose de deux orientations « naturelles » :

- depuis la racine vers les feuilles
- de la gauche vers la droite

Notion de marquage

Les structures linéaires possèdent un ordre de parcours naturel (induit par la structure).

Dans les arbres on dispose d'une liberté supplémentaire (profondeur ou largeur d'abord...) mais on dispose de deux orientations « naturelles » :

- depuis la racine vers les feuilles
- de la gauche vers la droite

Dans un graphe c'est plus complexe :

- pas de « point d'entrée » (ou de sortie) unique
- possibilité de revenir à un endroit déjà visité

Notion de marquage

Les structures linéaires possèdent un ordre de parcours naturel (induit par la structure).

Dans les arbres on dispose d'une liberté supplémentaire (profondeur ou largeur d'abord...) mais on dispose de deux orientations « naturelles » :

- depuis la racine vers les feuilles
- de la gauche vers la droite

Dans un graphe c'est plus complexe :

- pas de « point d'entrée » (ou de sortie) unique
- possibilité de revenir à un endroit déjà visité

Marquage

Dans un algorithme qui parcourt un graphe, il est fréquent de marquer les sommets déjà traités afin d'assurer la **terminaison** de l'algorithme.

C'est une forme d'étiquetage, qui peut s'ajouter à celui déjà présent s'il y en a un.

Algorithme de parcours générique

```
PARCOURS_GRAPHE( Graphe g, Sommet origine )
L = ListeVide()
Marquer( origine )
Insérer( origine, L )
while ¬ EstVide(L)
    x = Premier(L)
    L = ExtrairePremier(L)
    Traiter( x, g )
    foreach y ∈ ensVoisins(x, g) do
        if y non marqué
            Marquer( y )
            Insérer( y, L )
```

Algorithme de parcours générique

```
PARCOURS_GRAPHE( Graphe g, Sommet origine )
L = ListeVide()
Marquer( origine )
Insérer( origine, L )
while ¬ EstVide(L)
    x = Premier(L)
    L = ExtrairePremier(L)
    Traiter( x, g )
    foreach y ∈ ensVoisins(x, g) do
        if y non marqué
            Marquer( y )
            Insérer( y, L )
```

Si L est une pile

Parcours en profondeur

Si L est une file

Parcours en largeur

Algorithme de parcours générique

```
PARCOURS_GRAPHE( Graphe g, Sommet origine )
L = ListeVide()
Marquer( origine )
Insérer( origine, L )
while ¬ EstVide(L)
    x = Premier(L)
    L = ExtrairePremier(L)
    Traiter( x, g )
    foreach y ∈ ensVoisins(x, g) do
        if y non marqué
            Marquer( y )
            Insérer( y, L )
```

Si L est une pile	Si L est une file
Parcours en profondeur	Parcours en largeur

Variante possible : marquer les nœuds à leur sortie de L (moins efficace).

Plan

Mesure de complexité

Rappel

La complexité d'un algorithme s'exprime en fonction de la taille de la donnée.

Mesure de complexité

Rappel

La complexité d'un algorithme s'exprime en fonction de la taille de la donnée.

La taille d'un graphe dépend de deux paramètres :

- le nombre de sommets V,
- le nombre d'arêtes E,

qui ne sont pas liés linéairement : en général $0 \leq E \leq V^2$.

On précisera donc toujours la complexité d'un algorithme en fonction de E et/ou de V.

Mesure de complexité

Rappel

La complexité d'un algorithme s'exprime en fonction de la taille de la donnée.

La taille d'un graphe dépend de deux paramètres :

- le nombre de sommets V,
- le nombre d'arêtes E,

qui ne sont pas liés linéairement : en général $0 \leq E \leq V^2$.

On précisera donc toujours la complexité d'un algorithme en fonction de E et/ou de V.

Parcours de graphe

L'algorithme de parcours générique vu plus haut est en $\mathcal{O}(E + V)$.
(ce qui revient à écrire $\mathcal{O}(max(E, V))$)

Incidence de la représentation concrète

Selon la représentation choisie, la complexité des opérations élémentaires peut varier :

- Déterminer si deux sommets sont voisins peut être en temps :
  - constant
  - ou proportionnel à V
  - ou même proportionnel à E.
- Parcourir la liste des voisins d'un sommet peut être en temps proportionnel :
  - à son degré
  - ou à V.

Incidence de la représentation concrète

Selon la représentation choisie, la complexité des opérations élémentaires peut varier :

- Déterminer si deux sommets sont voisins peut être en temps :
  - constant
  - ou proportionnel à V
  - ou même proportionnel à E.
- Parcourir la liste des voisins d'un sommet peut être en temps proportionnel :
  - à son degré
  - ou à V.

L'empreinte en mémoire d'une structure de graphe peut elle-même varier entre $E + V$ et V^2.

Il est donc parfois délicat de quantifier la complexité d'un algorithme de graphe en se basant uniquement sur le type abstrait : on énonce ce qui est **possible sous réserve d'une représentation adéquate**.

Plan

Définition

Un problème d'optimisation a les caractéristiques suivantes :

- Une solution est un **sous-ensemble** d'une des données du problème.
- Il existe en général plusieurs solutions **admissibles**.
- À chaque solution (admissible) est associée une **valeur** (en général un coût ou un gain).

Le problème d'optimisation consiste non seulement à trouver une solution admissible, mais à trouver une solution de valeur **minimale** (pour un coût) ou **maximale** (pour un gain).

Définition

Un problème d'optimisation a les caractéristiques suivantes :

- Une solution est un **sous-ensemble** d'une des données du problème.
- Il existe en général plusieurs solutions **admissibles**.
- À chaque solution (admissible) est associée une **valeur** (en général un coût ou un gain).

Le problème d'optimisation consiste non seulement à trouver une solution admissible, mais à trouver une solution de valeur **minimale** (pour un coût) ou **maximale** (pour un gain).

Cette définition est notamment compatible avec celle d'algorithme glouton (ce qui ne veut pas dire qu'il existe un algorithme glouton pour tout problème d'optimisation).

Définition

Un problème d'optimisation a les caractéristiques suivantes :

- Une solution est un **sous-ensemble** d'une des données du problème.
- Il existe en général plusieurs solutions **admissibles**.
- À chaque solution (admissible) est associée une **valeur** (en général un coût ou un gain).

Le problème d'optimisation consiste non seulement à trouver une solution admissible, mais à trouver une solution de valeur **minimale** (pour un coût) ou **maximale** (pour un gain).

Cette définition est notamment compatible avec celle d'algorithme glouton (ce qui ne veut pas dire qu'il existe un algorithme glouton pour tout problème d'optimisation).

Dans le contexte des graphes, un problème d'optimisation consistera souvent à déterminer un ensemble optimal de sommets et/ou d'arcs.

Exemples

- Réseau
  - Itinéraire le plus court, le plus rapide
    (ensemble d'arcs de poids total minimal formant un chemin)

Exemples

- Réseau
  - Itinéraire le plus court, le plus rapide
    (ensemble d'arcs de poids total minimal formant un chemin)
  - Tolérance aux pannes
    (ensemble de sommets de cardinal minimal brisant la connexité du graphe)

Exemples

- Réseau
  - Itinéraire le plus court, le plus rapide
    (ensemble d'arcs de poids total minimal formant un chemin)
  - Tolérance aux pannes
    (ensemble de sommets de cardinal minimal brisant la connexité du graphe)
- Conflits
  - Maximiser le nombre d'activités simultanées
    (ensemble de sommets de valeur maximale sans arête interne)

Exemples

- Réseau
  - Itinéraire le plus court, le plus rapide
    (ensemble d'arcs de poids total minimal formant un chemin)
  - Tolérance aux pannes
    (ensemble de sommets de cardinal minimal brisant la connexité du graphe)
- Conflits
  - Maximiser le nombre d'activités simultanées
    (ensemble de sommets de valeur maximale sans arête interne)
  - Minimiser le nombre de ressources nécessaires
    (coloration des sommets compatible avec les arêtes utilisant un minimum de couleurs)
- ...

Plan

Chaînes, connexité

Chaîne

Dans un graphe non orienté, une chaîne est une suite de sommets $(x_0, x_1, \ldots x_k)$ telle que pour tout $0 < i \leq k$, (x_{i-1}, x_i) est une arête.
Les sommets x_0 et x_k sont les **extrémités** de la chaîne.

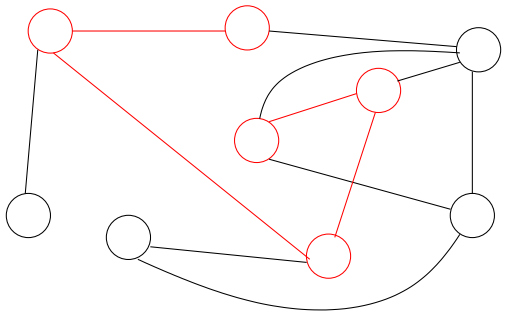
Chaînes, connexité

Chaîne

Dans un graphe non orienté, une chaîne est une suite de sommets $(x_0, x_1, \ldots x_k)$ telle que pour tout $0 < i \leq k$, (x_{i-1}, x_i) est une arête.
Les sommets x_0 et x_k sont les **extrémités** de la chaîne.

Longueur d'une chaîne

La longueur d'une chaîne est le nombre d'arêtes qui composent cette chaîne, c'est-à-dire le nombre de sommets moins un (4 dans cet exemple).



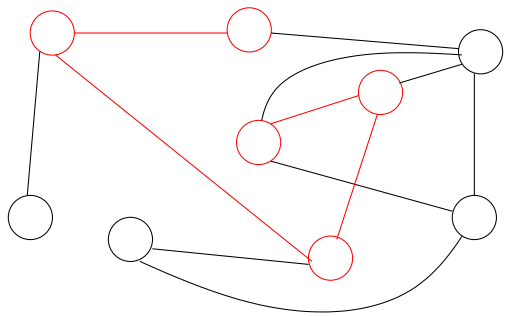
Chaînes, connexité

Chaîne

Dans un graphe non orienté, une chaîne est une suite de sommets $(x_0, x_1, \ldots x_k)$ telle que pour tout $0 < i \leq k$, (x_{i-1}, x_i) est une arête.
Les sommets x_0 et x_k sont les **extrémités** de la chaîne.

Longueur d'une chaîne

La longueur d'une chaîne est le nombre d'arêtes qui composent cette chaîne, c'est-à-dire le nombre de sommets moins un (4 dans cet exemple).



Graphe connexe

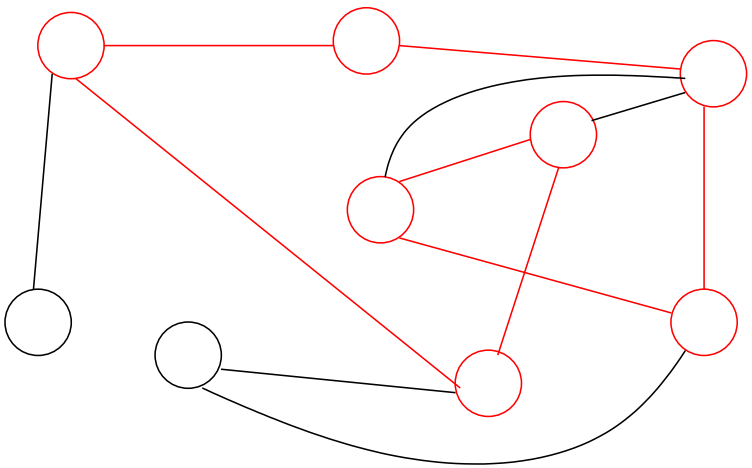
Un graphe est connexe si et seulement si, pour tout couple de sommets x et y, il existe une chaîne entre x et y.

Cycle

Cycle

Un cycle est une chaîne :

- dont les extrémités sont identiques ;
- de longueur k supérieure ou égale à 3 ;
- telle que pour tout $0 \leq i < k-1$, x_i est distinct de x_{i+2}.



Arborescence

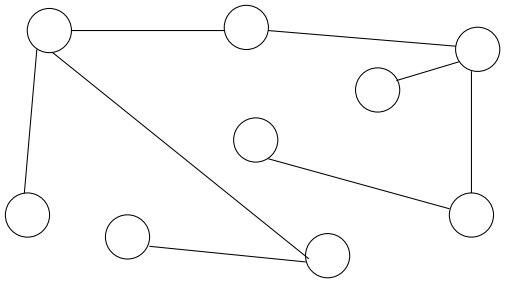
Arbre

Un arbre est un graphe non orienté, connexe et sans cycle.

Arborescence

Arbre

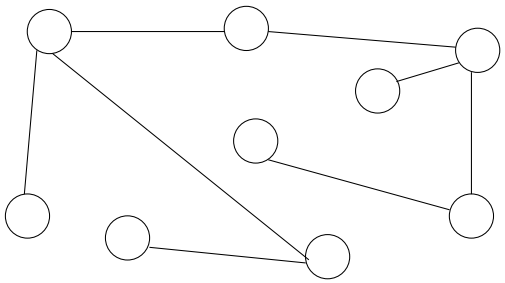
Un arbre est un graphe non orienté, connexe et sans cycle.

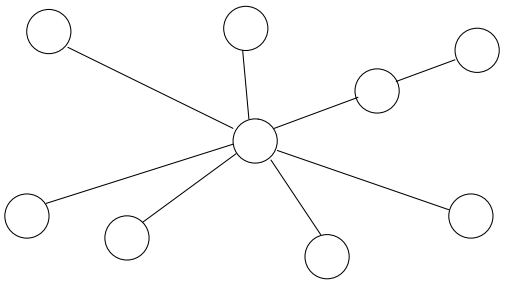


Arborescence

Arbre

Un arbre est un graphe non orienté, connexe et sans cycle.

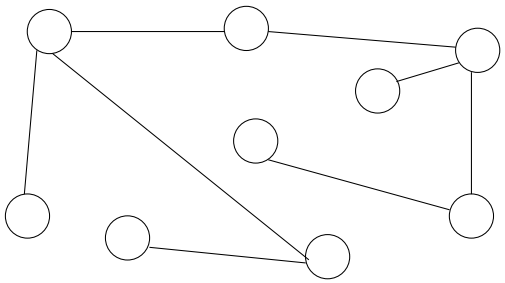
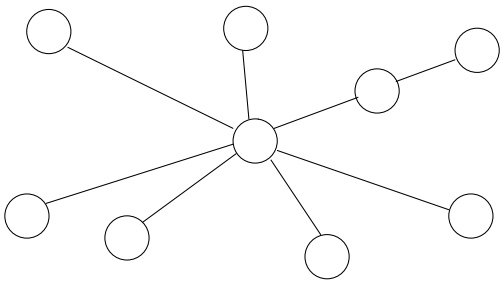
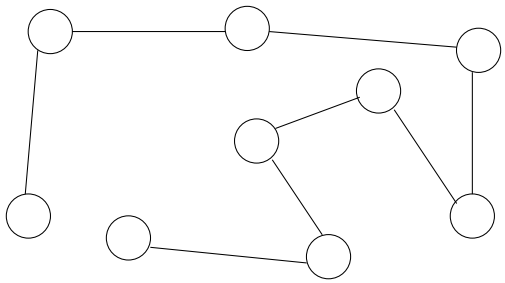




Arborescence

Arbre

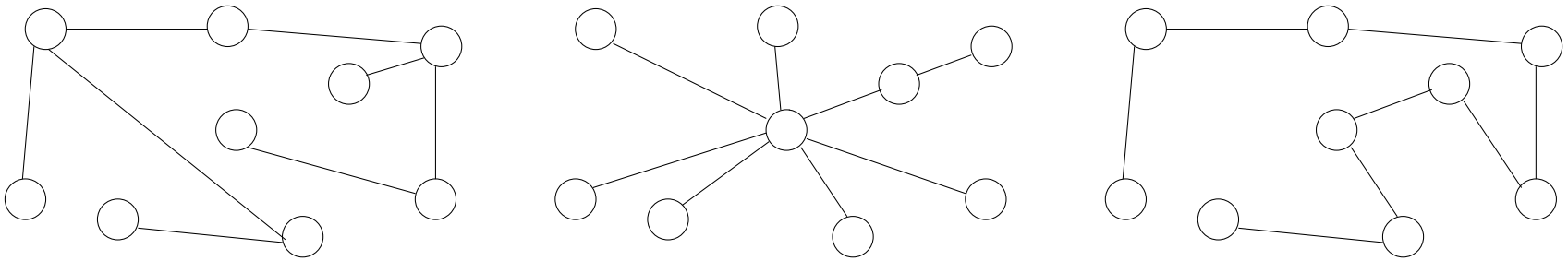
Un arbre est un graphe non orienté, connexe et sans cycle.

Arborescence

Arbre

Un arbre est un graphe non orienté, connexe et sans cycle.



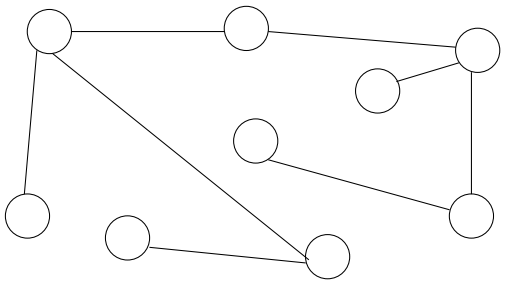
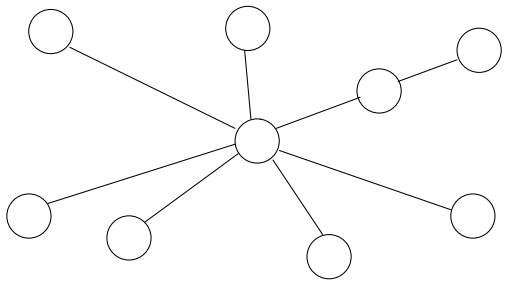
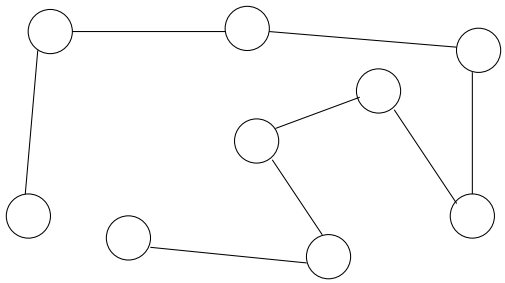
Attention, la notion est un peu différente des arbres définis plus tôt :

- Aucune contrainte d'arité
- Pas de notion de racine

Arborescence

Arbre

Un arbre est un graphe non orienté, connexe et sans cycle.

Attention, la notion est un peu différente des arbres définis plus tôt :

- Aucune contrainte d'arité
- Pas de notion de racine

Propriétés évidentes

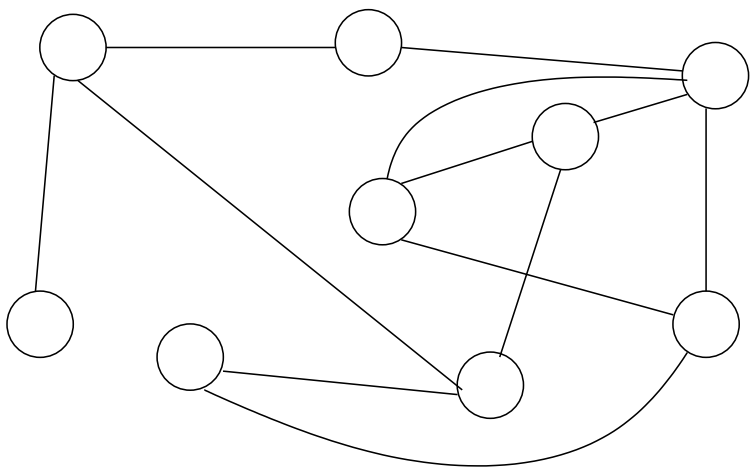
- Entre deux sommets donnés d'un arbre, il existe toujours exactement une chaîne (élémentaire).
- Un arbre à n sommets comporte $n - 1$ arêtes.

Plan

Sous-graphe

Sous-graphe

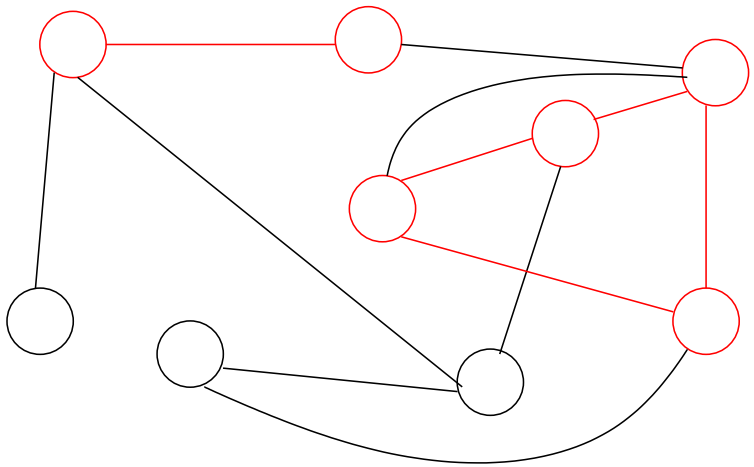
Si $G = (X, R)$, un sous-graphe de G est un graphe $H = (Y, Q)$ avec $Y \subseteq X$ et $Q \subseteq R$. Les extrémités de toutes les arêtes de Q font évidemment partie de Y.



Sous-graphe

Sous-graphe

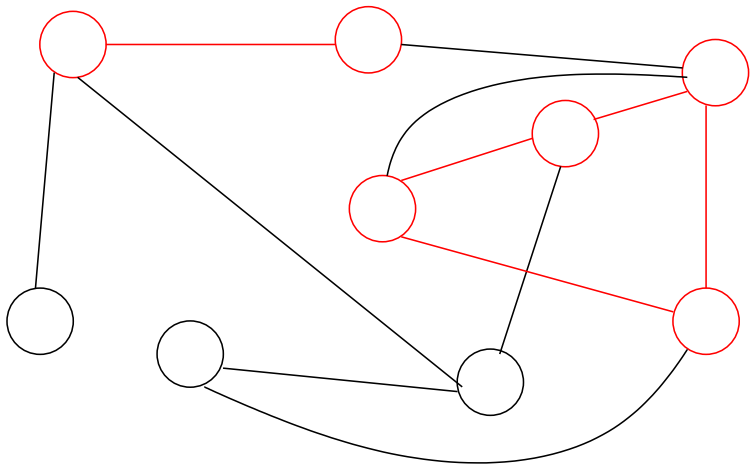
Si $G = (X, R)$, un sous-graphe de G est un graphe $H = (Y, Q)$ avec $Y \subseteq X$ et $Q \subseteq R$. Les extrémités de toutes les arêtes de Q font évidemment partie de Y.



Sous-graphe

Sous-graphe

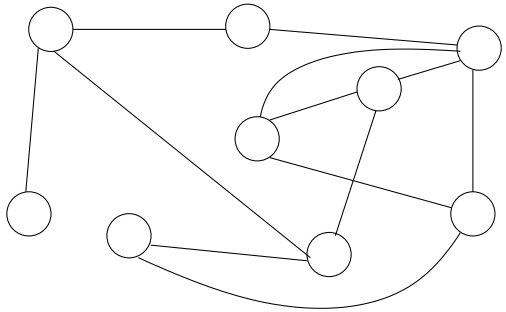
Si $G = (X, R)$, un sous-graphe de G est un graphe $H = (Y, Q)$ avec $Y \subseteq X$ et $Q \subseteq R$. Les extrémités de toutes les arêtes de Q font évidemment partie de Y.



Un sous-graphe de G est dit couvrant s'il contient tous les sommets de G. Attention, un sous-graphe couvrant n'est pas forcément connexe.

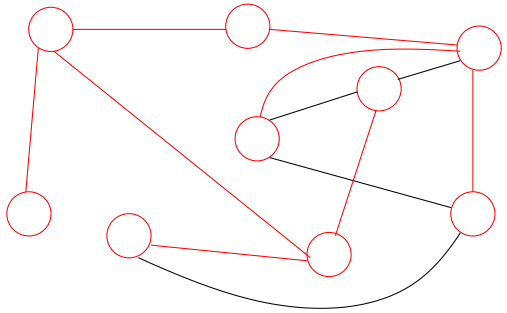
Arbre couvrant

Tout graphe connexe admet un arbre couvrant.



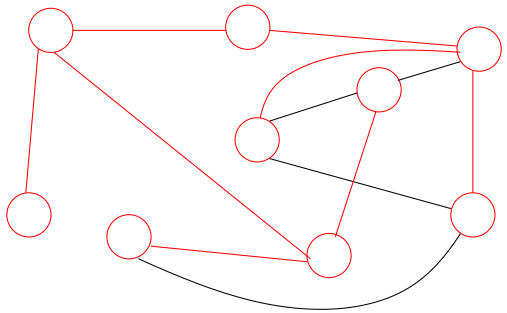
Arbre couvrant

Tout graphe connexe admet un arbre couvrant.



Arbre couvrant

Tout graphe connexe admet un arbre couvrant.

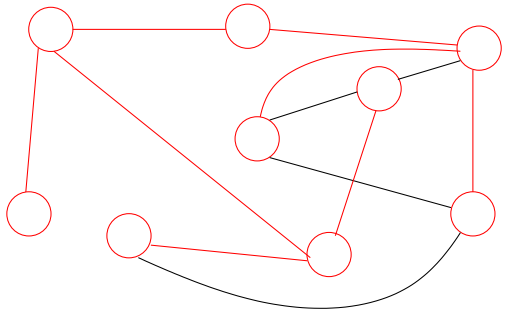


Si A est un sous-graphe couvrant d'un graphe G ayant V sommets, les caractérisations suivantes sont équivalentes :

- A est un arbre couvrant de G
- A est sans cycle et possède $V - 1$ arêtes
- A est connexe et possède $V - 1$ arêtes
- on ne peut pas ajouter une arête à A sans créer un cycle
- on ne peut pas retirer une arête à A sans briser sa connexité

Arbre couvrant

Tout graphe connexe admet un arbre couvrant.



Si A est un sous-graphe couvrant d'un graphe G ayant V sommets, les caractérisations suivantes sont équivalentes :

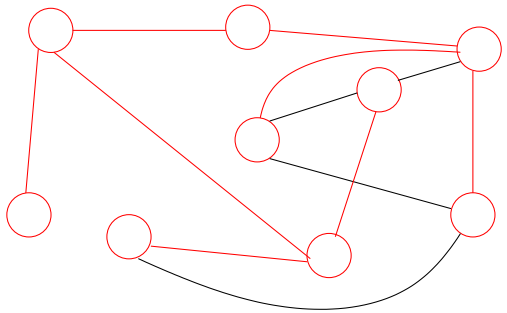
- A est un arbre couvrant de G
- A est sans cycle et **possède** $V-1$ **arêtes**
- A est connexe et **possède** $V-1$ **arêtes**
- on ne peut pas ajouter une arête à A sans créer un cycle
- on ne peut pas retirer une arête à A sans briser sa connexité

Caractérisations particulièrement utiles pour l'écriture d'algorithmes :

- le nombre d'arêtes est un bon critère d'arrêt ;

Arbre couvrant

Tout graphe connexe admet un arbre couvrant.



Si A est un sous-graphe couvrant d'un graphe G ayant V sommets, les caractérisations suivantes sont équivalentes :

- A est un arbre couvrant de G
- A est sans cycle et possède $V - 1$ arêtes
- A est connexe et possède $V - 1$ arêtes
- **on ne peut pas ajouter une arête à A sans créer un cycle**
- **on ne peut pas retirer une arête à A sans briser sa connexité**

Caractérisations particulièrement utiles pour l'écriture d'algorithmes :

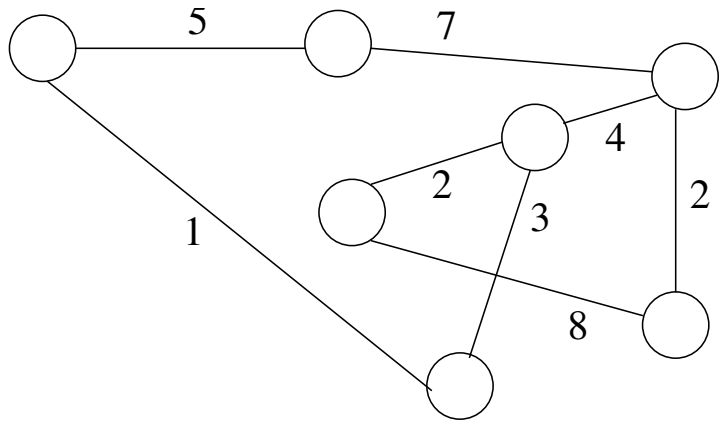
- le nombre d'arêtes est un bon critère d'arrêt ;
- l'absence de cycle (resp. la connexité) est un invariant à maintenir.

Dans un graphe pondéré

Rappel

Un **graphe aux arêtes pondérées** est un graphe muni d'une fonction de poids $f : R \to \mathbb{Z}$ (ou $\mathbb{R}$).

Le **poids d'un sous-graphe** est la somme des poids de ses arêtes.
On recherche alors un arbre couvrant de poids minimal (en anglais *Minimum Spanning Tree*).

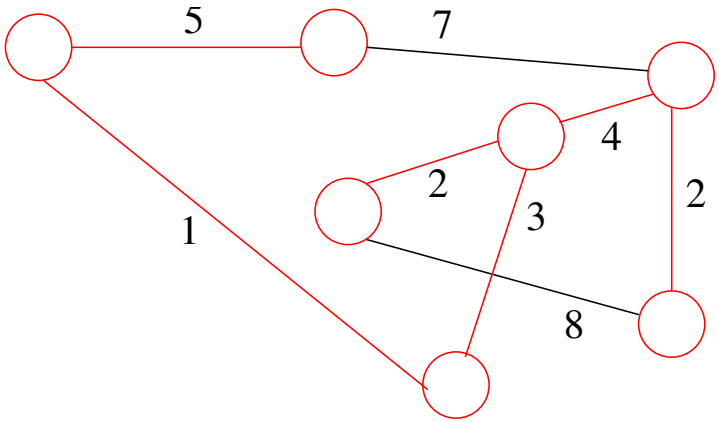


Dans un graphe pondéré

Rappel

Un **graphe aux arêtes pondérées** est un graphe muni d'une fonction de poids $f : R \to \mathbb{Z}$ (ou $\mathbb{R}$).

Le **poids d'un sous-graphe** est la somme des poids de ses arêtes.
On recherche alors un arbre couvrant de poids minimal (en anglais *Minimum Spanning Tree*).



Application

Problème proposé (et résolu) en 1926 par Otakar Borůvka pour la construction de réseaux électriques efficaces.

- Les sommets de G représentent des lieux à connecter : villes, ordinateurs, composants électroniques, etc.
- Les arêtes de G représentent les liens (routes, câbles, tuyaux...) *potentiels* entre ces lieux, avec les coûts effectifs de création (ou d'activation) de ces liens.
- L'arbre couvrant minimal est le réseau le moins coûteux ne laissant aucun lieu isolé.

Application

Problème proposé (et résolu) en 1926 par Otakar Borůvka pour la construction de réseaux électriques efficaces.

- Les sommets de G représentent des lieux à connecter : villes, ordinateurs, composants électroniques, etc.
- Les arêtes de G représentent les liens (routes, câbles, tuyaux...) *potentiels* entre ces lieux, avec les coûts effectifs de création (ou d'activation) de ces liens.
- L'arbre couvrant minimal est le réseau le moins coûteux ne laissant aucun lieu isolé.

Attention : on minimise le coût global du réseau, pas les longueurs des chemins dans l'arbre.
Il existe des variantes du problème contraignant la forme de l'arbre obtenu : degré borné pour chaque sommet ; diamètre borné ; etc.

Plan

Algorithmes de détermination d'un arbre couvrant

Deux idées duales :

Algorithmes de détermination d'un arbre couvrant

Deux idées duales :

$A := (X, \emptyset)$ (le graphe vide)

$A := (X, R)$ (le graphe G)

Algorithmes de détermination d'un arbre couvrant

Deux idées duales :

$A := (X, \emptyset)$ (le graphe vide)
tant que $nbAretes(A) < V - 1$
faire

$A := (X, R)$ (le graphe G)
tant que $nbAretes(A) > V - 1$
faire

Algorithmes de détermination d'un arbre couvrant

Deux idées duales :

$A := (X, \emptyset)$ (le graphe vide)
tant que $nbAretes(A) < V - 1$
faire
    Choisir une arête de G qui ne crée pas de cycle.

$A := (X, R)$ (le graphe G)
tant que $nbAretes(A) > V - 1$
faire
    Choisir une arête de A qui n'est pas indispensable à la connexité.

Algorithmes de détermination d'un arbre couvrant

Deux idées duales :

$A := (X, \emptyset)$ (le graphe vide)
tant que $nbAretes(A) < V - 1$
faire
  Choisir une arête de G qui ne crée pas de cycle.
  Ajouter cette arête à A.

$A := (X, R)$ (le graphe G)
tant que $nbAretes(A) > V - 1$
faire
  Choisir une arête de A qui n'est pas indispensable à la connexité.
  Retirer cette arête à A.

Algorithmes de détermination d'un arbre couvrant

Deux idées duales :

$A := (X, \emptyset)$ (le graphe vide)
tant que $nbAretes(A) < V - 1$
faire
  Choisir une arête de G qui ne crée pas de cycle.
  Ajouter cette arête à A.

$A := (X, R)$ (le graphe G)
tant que $nbAretes(A) > V - 1$
faire
  Choisir une arête de A qui n'est pas indispensable à la connexité.
  Retirer cette arête à A.

Par construction le graphe A obtenu est un arbre couvrant (pour peu que G soit connexe).

Algorithmes de détermination d'un arbre couvrant

Deux idées duales :

$A := (X, \emptyset)$ (le graphe vide)
tant que $nbAretes(A) < V - 1$
faire
  Choisir une arête de G qui ne crée pas de cycle.
  Ajouter cette arête à A.

$A := (X, R)$ (le graphe G)
tant que $nbAretes(A) > V - 1$
faire
  Choisir une arête de A qui n'est pas indispensable à la connexité.
  Retirer cette arête à A.

Par construction le graphe A obtenu est un arbre couvrant (pour peu que G soit connexe).

Critère de choix

- ne pas créer de cycle : assez facile
- vérifier la connexité : moins facile (algorithme **Reverse-Delete**)

Choisir une arête

Toute arête qui ne crée pas de cycle convient.
Pour que l'arbre soit minimal, il faut aussi tenir compte des poids.

Là encore, deux politiques :

Choisir une arête

Toute arête qui ne crée pas de cycle convient.
Pour que l'arbre soit minimal, il faut aussi tenir compte des poids.

Là encore, deux politiques :

Connexité d'abord : Algorithme de Prim

On choisit l'arête de poids minimal **parmi celles incidentes à** A.
En cours d'algorithme A est un **arbre**.

Il suffit qu'une extrémité de l'arête choisie soit hors de A.

Choisir une arête

Toute arête qui ne crée pas de cycle convient.
Pour que l'arbre soit minimal, il faut aussi tenir compte des poids.

Là encore, deux politiques :

Connexité d'abord : Algorithme de Prim

On choisit l'arête de poids minimal **parmi celles incidentes à** A.
En cours d'algorithme A est un **arbre**.

Il suffit qu'une extrémité de l'arête choisie soit hors de A.

Minimalité d'abord : Algorithme de Kruskal

On choisit l'arête de poids minimal **dans tout le graphe**.
En cours d'algorithme A est une **forêt**.

Il faut mémoriser si deux sommets sont dans des **composantes connexes** distinctes : utilisation de *Union-Find*.

Choisir une arête

Toute arête qui ne crée pas de cycle convient.
Pour que l'arbre soit minimal, il faut aussi tenir compte des poids.

Là encore, deux politiques :

Connexité d'abord : Algorithme de Prim

On choisit l'arête de poids minimal **parmi celles incidentes à** A.
En cours d'algorithme A est un **arbre**.

Il suffit qu'une extrémité de l'arête choisie soit hors de A.

Minimalité d'abord : Algorithme de Kruskal

On choisit l'arête de poids minimal **dans tout le graphe**.
En cours d'algorithme A est une **forêt**.

Il faut mémoriser si deux sommets sont dans des **composantes connexes** distinctes : utilisation de *Union-Find*.

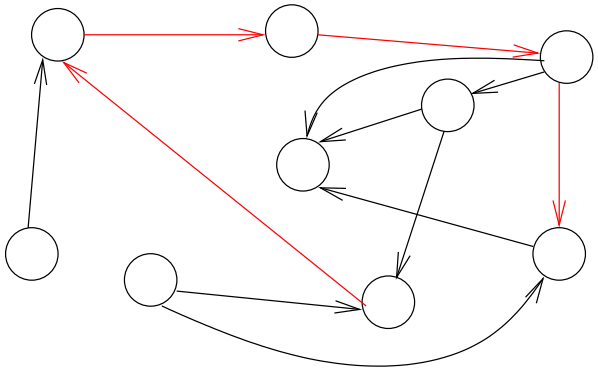
Algorithmes gloutons

Plan

Chemins

Chemin

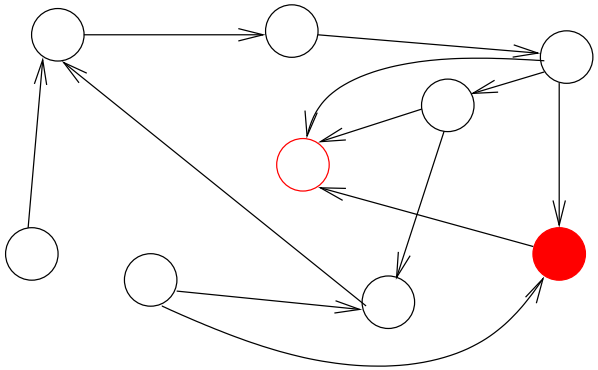
Un chemin est une suite de sommets $(x_0, x_1, \ldots x_k)$ telle que pour tout $0 < i \leq k$, (x_{i-1}, x_i) est un arc du graphe.
x_0 est l'**origine** du chemin, et x_k est son **extrémité**.



Chemins

Chemin

Un chemin est une suite de sommets $(x_0, x_1, \ldots x_k)$ telle que pour tout $0 < i \leq k$, (x_{i-1}, x_i) est un arc du graphe.
x_0 est l'**origine** du chemin, et x_k est son **extrémité**.



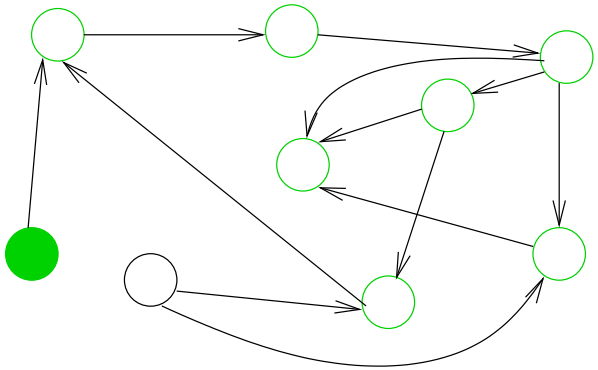
Accessible

Un sommet y est dit accessible à partir d'un sommet x s'il existe un **chemin** d'origine x et d'extrémité y.

Chemins

Chemin

Un chemin est une suite de sommets $(x_0, x_1, \ldots x_k)$ telle que
pour tout $0 < i \leq k$, (x_{i-1}, x_i) est un arc du graphe.
x_0 est l'**origine** du chemin, et x_k est son **extrémité**.



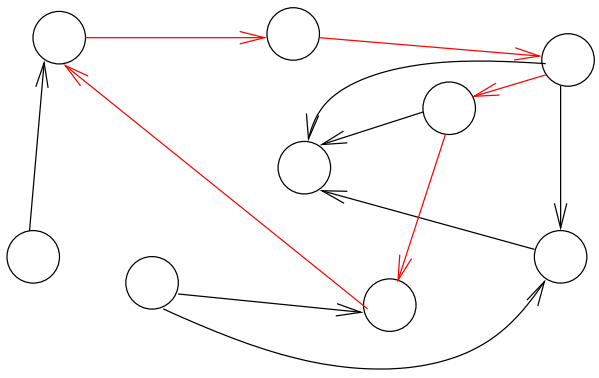
Accessible

Un sommet y est dit accessible à partir d'un sommet x s'il existe un **chemin** d'origine x et d'extrémité y.

Circuits

Circuit

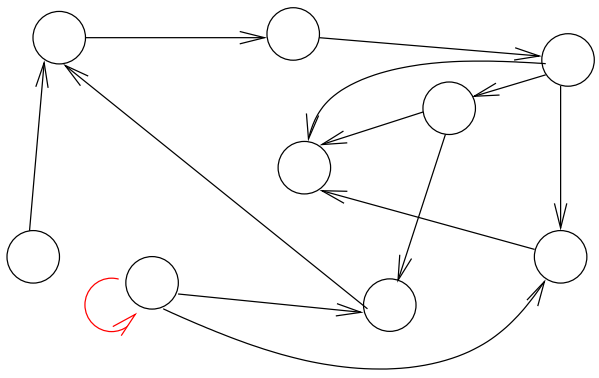
Un circuit est un chemin de longueur non nulle dont l'origine est identique à l'extrémité.
En général, par commodité, on considère que tous les circuits de la forme $(x_i, x_{i+1}, \ldots x_{k-1}, x_0, x_1, \ldots x_i)$ constituent le même circuit.



Circuits

Circuit

Un circuit est un chemin de longueur non nulle dont l'origine est identique à l'extrémité.
En général, par commodité, on considère que tous les circuits de la forme $(x_i, x_{i+1}, \ldots x_{k-1}, x_0, x_1, \ldots x_i)$ constituent le même circuit.



Boucle

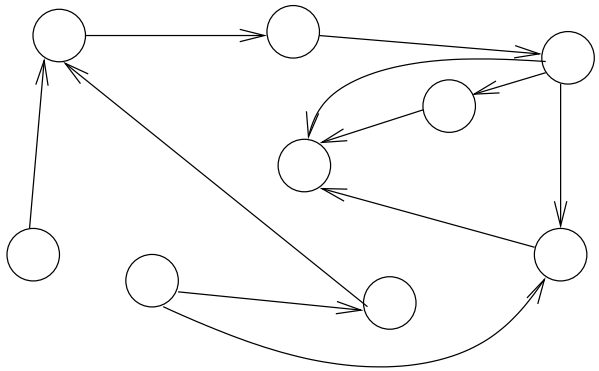
On appelle boucle tout arc dont l'origine est identique à l'extrémité.
Une boucle est un cas particulier de circuit, de longueur 1.

Le problème du tri topologique

Définition

Un tri topologique d'un graphe orienté $G = (X, R)$ est un **ordre total** des sommets de G tel que

pour tout arc $(x, y) \in R$ *alors* x *apparaît avant* y.

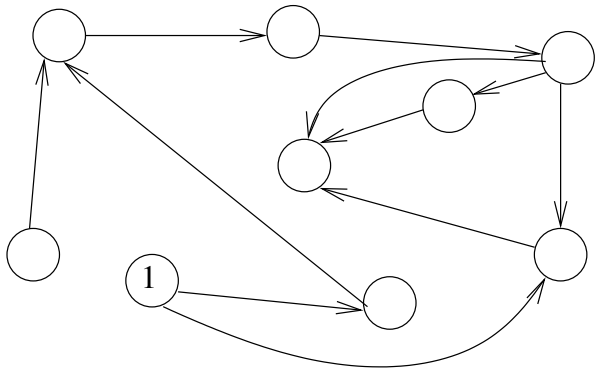


Le problème du tri topologique

Définition

Un tri topologique d'un graphe orienté $G = (X, R)$ est un **ordre total** des sommets de G tel que

pour tout arc $(x, y) \in R$ *alors* x *apparaît avant* y.

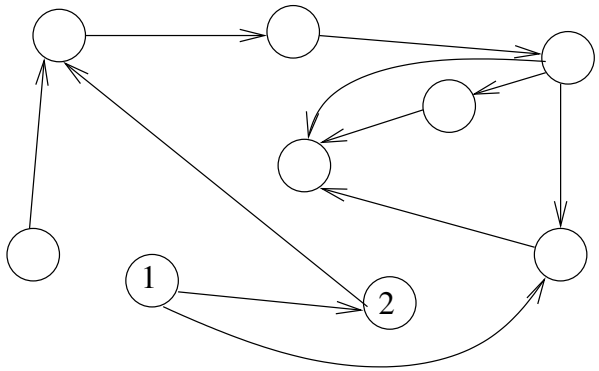


Le problème du tri topologique

Définition

Un tri topologique d'un graphe orienté $G = (X, R)$ est un **ordre total** des sommets de G tel que

pour tout arc $(x, y) \in R$ *alors* x *apparaît avant* y.

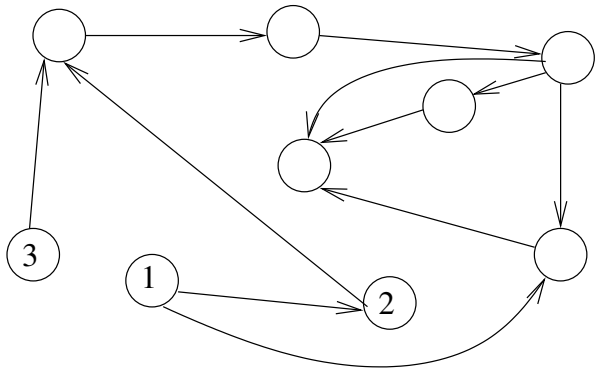


Le problème du tri topologique

Définition

Un tri topologique d'un graphe orienté $G = (X, R)$ est un **ordre total** des sommets de G tel que

pour tout arc $(x, y) \in R$ *alors* x *apparaît avant* y.

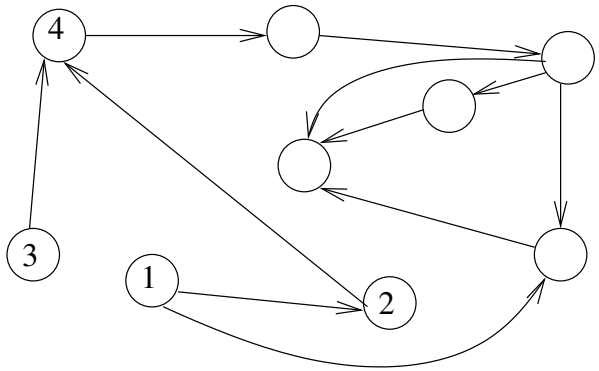


Le problème du tri topologique

Définition

Un tri topologique d'un graphe orienté $G = (X, R)$ est un **ordre total** des sommets de G tel que

pour tout arc $(x, y) \in R$ *alors* x *apparaît avant* y.

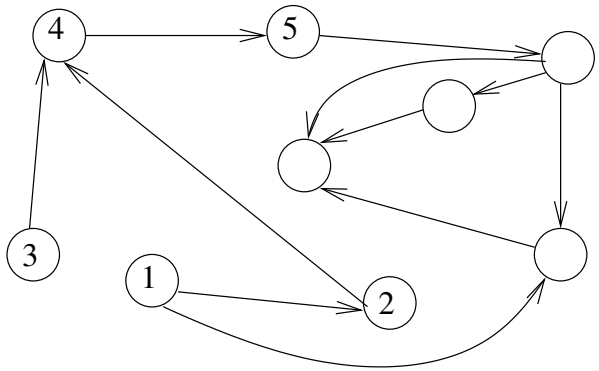


Le problème du tri topologique

Définition

Un tri topologique d'un graphe orienté $G = (X, R)$ est un **ordre total** des sommets de G tel que

pour tout arc $(x, y) \in R$ *alors* x *apparaît avant* y.

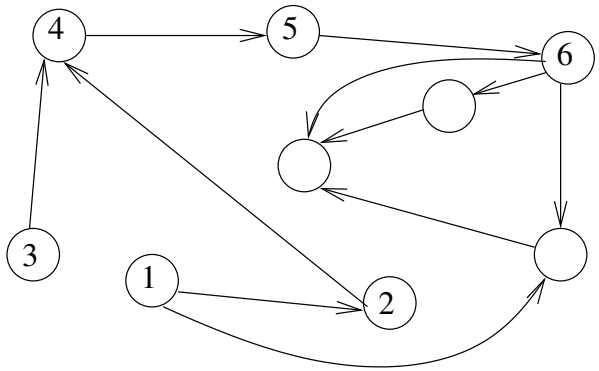


Le problème du tri topologique

Définition

Un tri topologique d'un graphe orienté $G = (X, R)$ est un **ordre total** des sommets de G tel que

pour tout arc $(x, y) \in R$ *alors* x *apparaît avant* y.

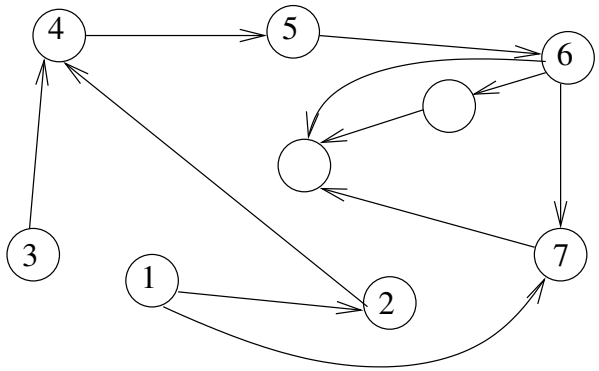


Le problème du tri topologique

Définition

Un tri topologique d'un graphe orienté $G = (X, R)$ est un **ordre total** des sommets de G tel que

pour tout arc $(x, y) \in R$ *alors* x *apparaît avant* y.

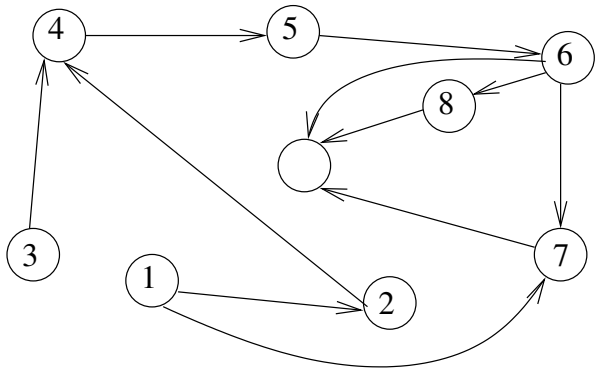


Le problème du tri topologique

Définition

Un tri topologique d'un graphe orienté $G = (X, R)$ est un **ordre total** des sommets de G tel que

pour tout arc $(x, y) \in R$ *alors* x *apparaît avant* y.

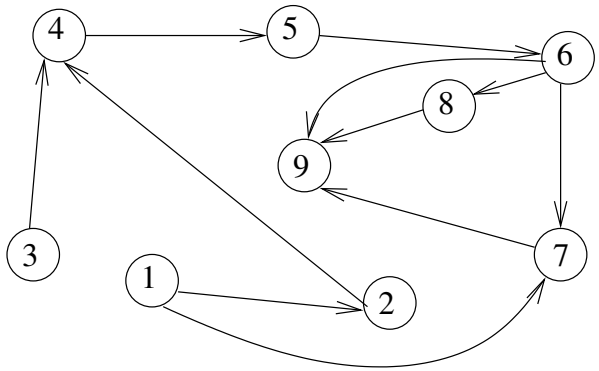


Le problème du tri topologique

Définition

Un tri topologique d'un graphe orienté $G = (X, R)$ est un **ordre total** des sommets de G tel que

pour tout arc $(x, y) \in R$ *alors* x *apparaît avant* y.

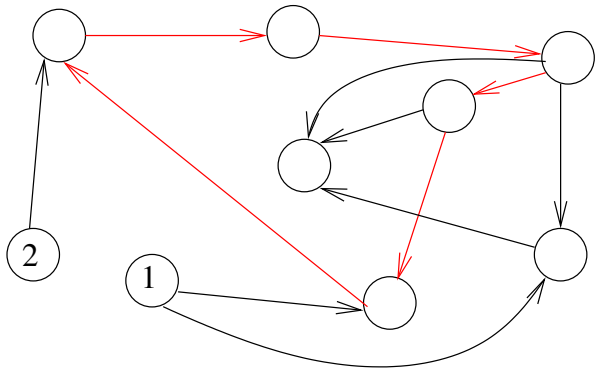


Le problème du tri topologique

Définition

Un tri topologique d'un graphe orienté $G = (X, R)$ est un **ordre total** des sommets de G tel que

pour tout arc $(x, y) \in R$ *alors* x *apparaît avant* y.



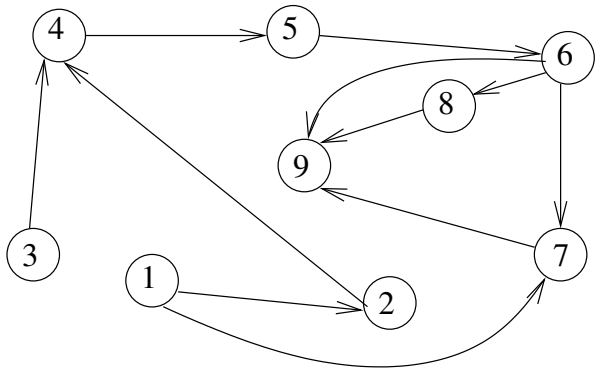
Corollaire : il existe un tri topologique si et seulement si le graphe ne comporte pas de circuit.

Le problème du tri topologique

Définition

Un tri topologique d'un graphe orienté $G = (X, R)$ est un **ordre total** des sommets de G tel que

pour tout arc $(x, y) \in R$ *alors* x *apparaît avant* y.



Corollaire : il existe un tri topologique si et seulement si le graphe ne comporte pas de circuit.

Non unicité du tri topologique

Il existe en général plusieurs ordres corrects.
Dans l'algorithme qui suit, selon la structure choisie pour l'ensemble E on obtient un ordre différent.

Algorithme de tri topologique

TRI_TOPOLOGIQUE (Graphe g)
Données : Un graphe g
Résultat : Une file F contenant un tri topologique des sommets de g
ou une valeur spéciale $\perp$ si c'est impossible

```
E := EnsembleVide()
foreach x ∈ ensSommets(g) do
    if estVide(ensPredecesseurs(x))
        Insérer( x, E )
F := FileVide()
while ¬ estVide(E)
    x := ExtraireElement( E )
    F := Enfiler( x, F )
    foreach y ∈ ensSuccesseurs(x) do
        supprimerArc( x, y, g )
        if estVide(ensPredecesseurs(y))
            Insérer( y, E )
if nbArcs(g) = 0
    return F
else // g comporte un
circuit
    return ⊥
```

Analyse de l'algorithme de tri tpopologique

À propos des prédécesseurs :

- dans certaines représentations les ensembles de prédécesseurs peuvent être immédiatement disponibles ;
- il est aussi possible de précalculer le demi-degré intérieur de chaque sommet (a priori en $\mathcal{O}(E + V)$), puis de le maintenir à jour lors des suppressions d'arcs.

Analyse de l'algorithme de tri tpopologique

À propos des prédécesseurs :

- dans certaines représentations les ensembles de prédécesseurs peuvent être immédiatement disponibles ;
- il est aussi possible de précalculer le demi-degré intérieur de chaque sommet (a priori en $\mathcal{O}(E + V)$), puis de le maintenir à jour lors des suppressions d'arcs.

Aspect glouton

On reconnaît les caractéristiques d'un algorithme glouton :

- la réponse au problème est une séquence ordonnée
- choix définitifs (on ne fait qu'enfiler dans F)
- sommet à enfiler choisi sur un critère local (son demi-degré intérieur)

Analyse de l'algorithme de tri tpopologique

À propos des prédécesseurs :

- dans certaines représentations les ensembles de prédécesseurs peuvent être immédiatement disponibles ;
- il est aussi possible de précalculer le demi-degré intérieur de chaque sommet (a priori en $\mathcal{O}(E + V)$), puis de le maintenir à jour lors des suppressions d'arcs.

Aspect glouton

On reconnaît les caractéristiques d'un algorithme glouton :

- la réponse au problème est une séquence ordonnée
- choix définitifs (on ne fait qu'enfiler dans F)
- sommet à enfiler choisi sur un critère local (son demi-degré intérieur)

Dans le pire des cas :

- chaque sommet est visité : $\mathcal{O}(V)$
- chaque arc sortant de chaque sommet est supprimé : $\mathcal{O}(E)$

d'où une complexité en $\mathcal{O}(E + V)$.

En résumé

Aujourd'hui

- Les graphes et la structure de données correspondante constituent un bon support à beaucoup d'algorithmes
- Plusieurs parcours d'un graphe sont possibles, avec des précautions supplémentaires à cause des **cycles** possibles
- La complexité d'un algorithme de graphes s'exprime en fonction du nombre de **sommets et** du nombre d'**arcs**.
- Les graphes se prêtent à de nombreux algorithmes d'optimisation, parfois réalisables sur le principe glouton mais parfois très complexes.

La prochaine fois

- Recherche d'une sous-chaîne dans un texte
- Algorithme de Knuth-Morris-Pratt