Optimisation en informatique – RCP 104

Cédric Bentz

EPN 5 (Informatique), CNAM, Paris

(Contribution aux supports : Agnès Plateau)

Plan

Routage de données dans un réseau

La notion de "routage" de données fait référence à l'itinéraire à emprunter dans un réseau pour acheminer ces données d'un nœud origine o à un nœud destination d.

Si ces nœuds sont dans (ou représentent) 2 sous-réseaux différents, alors il est nécessaire de passer par un routeur, appelé **passerelle**.

En terme de modélisation, l'accès à un routeur ou à un nœud du même sous-réseau se fait par un lien direct (arc) : en pratique, les temps de réponse des liens de chaque sous-réseau sont négligeables devant les temps de réponse des liens entre sous-réseaux.

Pour déterminer le meilleur routage possible entre deux nœuds o et d, il faut donc déterminer la séquence de routeurs à utiliser : en pratique, on peut identifier les sommets du graphe modélisant le réseau considéré avec ses sous-réseaux.

Routage de données dans un réseau

La notion de "routage" de données fait référence à l'itinéraire à emprunter dans un réseau pour acheminer ces données d'un nœud origine o à un nœud destination d.

Si ces nœuds sont dans (ou représentent) 2 sous-réseaux différents, alors il est nécessaire de passer par un routeur, appelé **passerelle**.

En terme de modélisation, l'accès à un routeur ou à un nœud du même sous-réseau se fait par un lien direct (arc) : en pratique, les temps de réponse des liens de chaque sous-réseau sont négligeables devant les temps de réponse des liens entre sous-réseaux.

Pour déterminer le meilleur routage possible entre deux nœuds o et d, il faut donc déterminer la séquence de routeurs à utiliser : en pratique, on peut identifier les sommets du graphe modélisant le réseau considéré avec ses sous-réseaux.

Routage de données dans un réseau

La notion de "routage" de données fait référence à l'itinéraire à emprunter dans un réseau pour acheminer ces données d'un nœud origine o à un nœud destination d.

Si ces nœuds sont dans (ou représentent) 2 sous-réseaux différents, alors il est nécessaire de passer par un routeur, appelé **passerelle**.

En terme de modélisation, l'accès à un routeur ou à un nœud du même sous-réseau se fait par un lien direct (arc) : en pratique, les temps de réponse des liens de chaque sous-réseau sont négligeables devant les temps de réponse des liens entre sous-réseaux.

Pour déterminer le meilleur routage possible entre deux nœuds o et d, il faut donc déterminer la séquence de routeurs à utiliser : en pratique, on peut identifier les sommets du graphe modélisant le réseau considéré avec ses sous-réseaux.

Routage de données dans un réseau

La notion de "routage" de données fait référence à l'itinéraire à emprunter dans un réseau pour acheminer ces données d'un nœud origine o à un nœud destination d.

Si ces nœuds sont dans (ou représentent) 2 sous-réseaux différents, alors il est nécessaire de passer par un routeur, appelé **passerelle**.

En terme de modélisation, l'accès à un routeur ou à un nœud du même sous-réseau se fait par un lien direct (arc) : en pratique, les temps de réponse des liens de chaque sous-réseau sont négligeables devant les temps de réponse des liens entre sous-réseaux.

Pour déterminer le meilleur routage possible entre deux nœuds o et d, il faut donc déterminer la séquence de routeurs à utiliser : en pratique, on peut identifier les sommets du graphe modélisant le réseau considéré avec ses sous-réseaux.

Routage de données dans un réseau – formalisation

À chaque lien est associé son temps de réponse estimé, et l'objectif est donc de faire transiter, avec un temps de réponse total aussi petit que possible, des données d'un nœud origine o vers un nœud destination d, voire vers tous les autres nœuds du réseau.

Dans le graphe correspondant à ce réseau, on associe une longueur positive à chaque arc (temps de réponse), et on cherche un chemin du sommet o au sommet d qui minimise sa longueur totale : un tel chemin est appelé **plus court chemin** de o à d.

Comme toutes les longueurs sont positives ou nulles, on peut obtenir un tel chemin à l'aide de l'algorithme de Dijkstra (1959).

Routage de données dans un réseau – formalisation

À chaque lien est associé son temps de réponse estimé, et l'objectif est donc de faire transiter, avec un temps de réponse total aussi petit que possible, des données d'un nœud origine o vers un nœud destination d, voire vers tous les autres nœuds du réseau.

Dans le graphe correspondant à ce réseau, on associe une longueur positive à chaque arc (temps de réponse), et on cherche un chemin du sommet o au sommet d qui minimise sa longueur totale : un tel chemin est appelé **plus court chemin** de o à d.

Comme toutes les longueurs sont positives ou nulles, on peut obtenir un tel chemin à l'aide de l'algorithme de Dijkstra (1959).

Routage de données dans un réseau – formalisation

À chaque lien est associé son temps de réponse estimé, et l'objectif est donc de faire transiter, avec un temps de réponse total aussi petit que possible, des données d'un nœud origine o vers un nœud destination d, voire vers tous les autres nœuds du réseau.

Dans le graphe correspondant à ce réseau, on associe une longueur positive à chaque arc (temps de réponse), et on cherche un chemin du sommet o au sommet d qui minimise sa longueur totale : un tel chemin est appelé **plus court chemin** de o à d.

Comme toutes les longueurs sont positives ou nulles, on peut obtenir un tel chemin à l'aide de l'algorithme de Dijkstra (1959).

Routage de données dans un réseau – table de routage

Définition

Une **table de routage** *associée à un nœud donné est un tableau qui, pour chaque destination possible à partir de ce nœud, indique le nom (en réalité, l'adresse IP) du prochain nœud (en général, du routeur) à atteindre pour se rendre à cette destination.*

Si on note o ce sommet et d la destination, alors une telle table indique le sommet suivant dans un plus court chemin de o à d.

Routage de données dans un réseau – table de routage

Définition

Une **table de routage** *associée à un nœud donné est un tableau qui, pour chaque destination possible à partir de ce nœud, indique le nom (en réalité, l'adresse IP) du prochain nœud (en général, du routeur) à atteindre pour se rendre à cette destination.*

Si on note o ce sommet et d la destination, alors une telle table indique le sommet suivant dans un plus court chemin de o à d.

Illustration – calcul d'une table de routage à l'aide de l'algorithme de Dijkstra (1/6)

L'algorithme de Dijkstra permet de déterminer, de façon efficace, tous les plus courts chemins issus d'un sommet donné (ici, x_1).

Pour cela, il associe à chaque sommet x_i deux informations : λ_i et p_i, et traite ensuite les sommets un par un (chacun étant traité une et une seule fois), pour calculer ces informations.

Au moment où un sommet x_i est traité, λ_i est égal à la longueur d'un plus court chemin de x_1 à x_i (et ne changera plus ensuite), et p_i est le sommet précédant x_i dans un tel plus court chemin.

Le sommet x_1 est traité d'abord, puisque $\lambda_1 = 0$ et p_1 n'existe pas.

Illustration – calcul d'une table de routage à l'aide de l'algorithme de Dijkstra (1/6)

L'algorithme de Dijkstra permet de déterminer, de façon efficace, tous les plus courts chemins issus d'un sommet donné (ici, x_1).

Pour cela, il associe à chaque sommet x_i deux informations : λ_i et p_i, et traite ensuite les sommets un par un (chacun étant traité une et une seule fois), pour calculer ces informations.

Au moment où un sommet x_i est traité, λ_i est égal à la longueur d'un plus court chemin de x_1 à x_i (et ne changera plus ensuite), et p_i est le sommet précédant x_i dans un tel plus court chemin.

Le sommet x_1 est traité d'abord, puisque $\lambda_1 = 0$ et p_1 n'existe pas.

Illustration – calcul d'une table de routage à l'aide de l'algorithme de Dijkstra (1/6)

L'algorithme de Dijkstra permet de déterminer, de façon efficace, tous les plus courts chemins issus d'un sommet donné (ici, x_1).

Pour cela, il associe à chaque sommet x_i deux informations : λ_i et p_i, et traite ensuite les sommets un par un (chacun étant traité une et une seule fois), pour calculer ces informations.

Au moment où un sommet x_i est traité, λ_i est égal à la longueur d'un plus court chemin de x_1 à x_i (et ne changera plus ensuite), et p_i est le sommet précédant x_i dans un tel plus court chemin.

Le sommet x_1 est traité d'abord, puisque $\lambda_1 = 0$ et p_1 n'existe pas.

Illustration – calcul d'une table de routage à l'aide de l'algorithme de Dijkstra (1/6)

L'algorithme de Dijkstra permet de déterminer, de façon efficace, tous les plus courts chemins issus d'un sommet donné (ici, x_1).

Pour cela, il associe à chaque sommet x_i deux informations : λ_i et p_i, et traite ensuite les sommets un par un (chacun étant traité une et une seule fois), pour calculer ces informations.

Au moment où un sommet x_i est traité, λ_i est égal à la longueur d'un plus court chemin de x_1 à x_i (et ne changera plus ensuite), et p_i est le sommet précédant x_i dans un tel plus court chemin.

Le sommet x_1 est traité d'abord, puisque $\lambda_1 = 0$ et p_1 n'existe pas.

Illustration – calcul d'une table de routage à l'aide de l'algorithme de Dijkstra (2/6)

Algorithme 1 : Algorithme de Dijkstra (dans un graphe orienté)

- $\{$*Début des initialisations*$\}$
  $\lambda_1 = 0$, et tous les sommets sauf x_1 sont à traiter ;
- **Pour** $i = 2$ à nombre de sommets **faire**
  - **Si** l'arc (x_1, x_i) existe **alors** $\lambda_i = longueur_{1i}$ et $p_i = x_1$;
  - **Sinon** $\lambda_i = +\infty$ et $p_i =$ non défini ;
  - **Fin Si**
- **Fin Pour** $\{$*Fin des initialisations*$\}$
- **Tant que** il reste des sommets à traiter **faire**
  - Déterminer x_j à traiter tel que $\lambda_j = min_{x_i \text{ à traiter}}\{\lambda_i\}$;
  - x_j n'est plus à traiter ;
  - **Pour** tout arc (x_j, x_i) avec x_i à traiter **faire**
    - **Si** $\lambda_i > \lambda_j + longueur_{ji}$ **alors** $\lambda_i = \lambda_j + longueur_{ji}$ et $p_i = x_j$;
    - **Fin Si**
  - **Fin Pour**
- **Fin Tant que**

Illustration – calcul d'une table de routage à l'aide de l'algorithme de Dijkstra (3/6)

Exemple illustratif

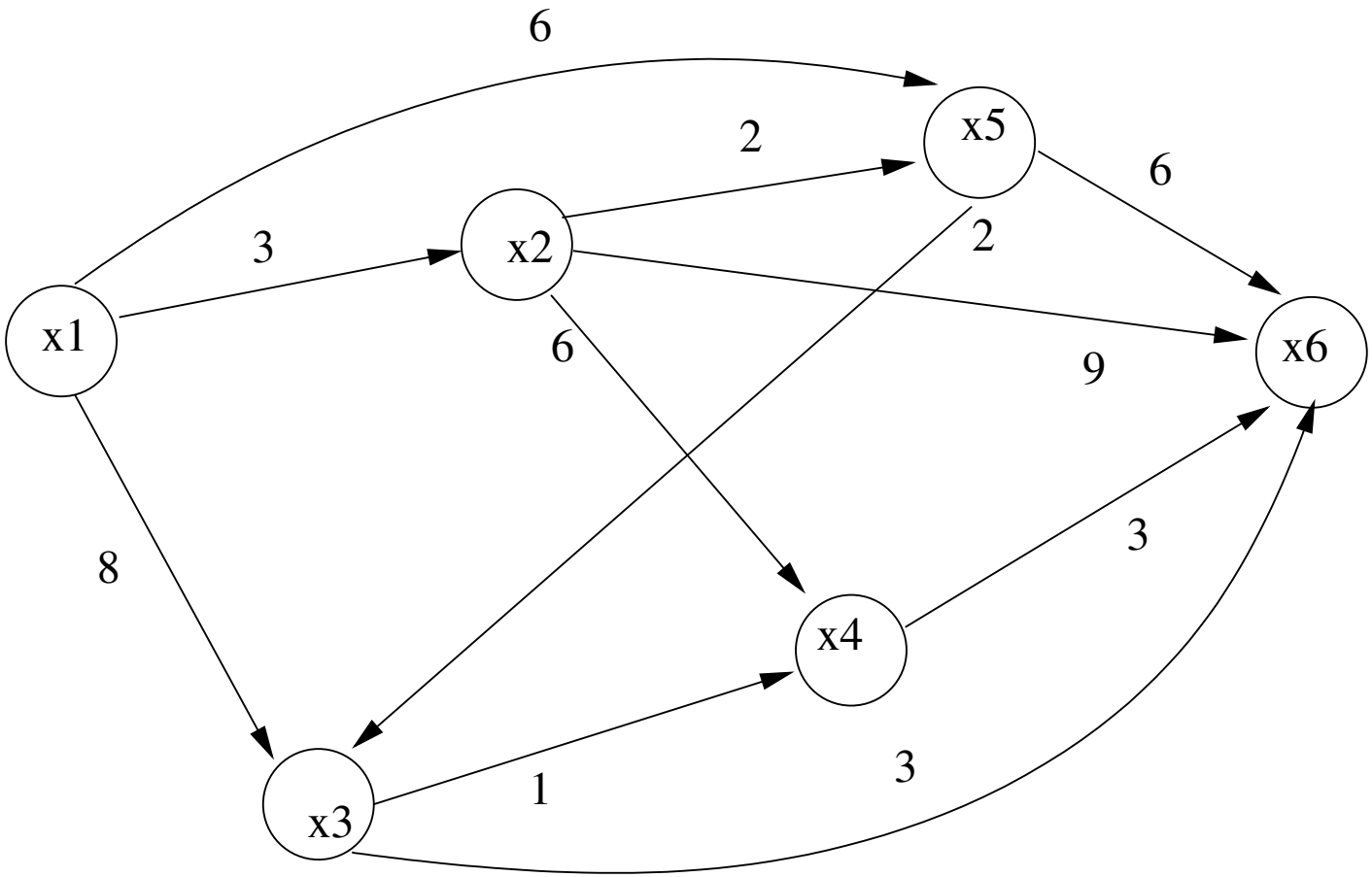


Illustration – calcul d'une table de routage à l'aide de l'algorithme de Dijkstra (4/6)

	(p_2, λ_2)	(p_3, λ_3)	(p_4, λ_4)	(p_5, λ_5)	(p_6, λ_6)	À traiter
x_1	$(x_1,3)$	$(x_1,8)$	$(-,+\infty)$	$(x_1,6)$	$(-,+\infty)$	$x_2,\ x_3,\ x_4,\ x_5,\ x_6$
x_2	$(x_1,3)^*$	$(x_1,8)$	$(x_2,9)$	$(x_2,5)$	$(x_2,12)$	$x_3,\ x_4,\ x_5,\ x_6$
x_5	$(x_1,3)^*$	$(x_5,7)$	$(x_2,9)$	$(x_2,5)^*$	$(x_5,11)$	$x_3,\ x_4,\ x_6$
x_3	$(x_1,3)^*$	$(x_5,7)^*$	$(x_3,8)$	$(x_2,5)^*$	$(x_3,10)$	$x_4,\ x_6$
x_4	$(x_1,3)^*$	$(x_5,7)^*$	$(x_3,8)^*$	$(x_2,5)^*$	$(x_3,10)$	x_6
x_6	$(x_1,3)^*$	$(x_5,7)^*$	$(x_3,8)^*$	$(x_2,5)^*$	$(x_3,10)^*$	$\emptyset$

Illustration – calcul d'une table de routage à l'aide de l'algorithme de Dijkstra (5/6)

Arborescence des plus courts chemins

$\Rightarrow$ Un et un seul chemin de x_1 à tout x_i :

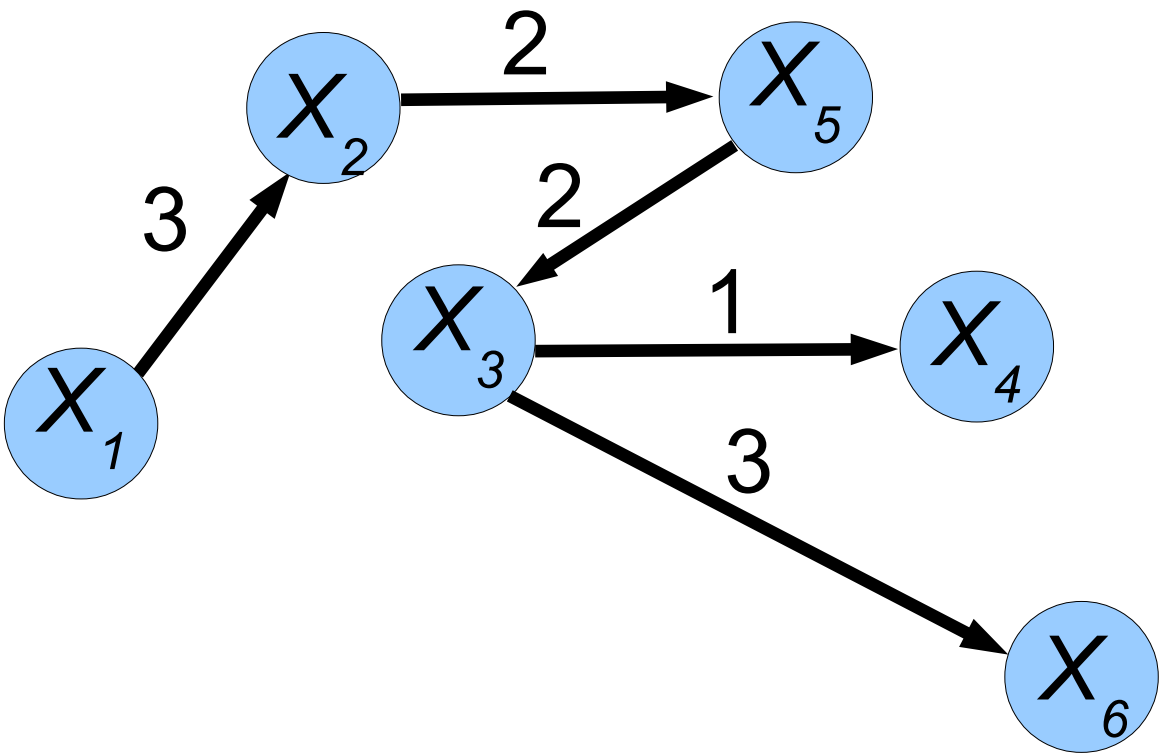


Illustration – calcul d'une table de routage à l'aide de l'algorithme de Dijkstra (6/6)

Table de routage (construite à partir de l'arborescence précédente)

Destination	Sommet suivant à atteindre
x_2	x_2
x_3	x_2
x_4	x_2
x_5	x_2
x_6	x_2

Plus court chemin avec longueurs ≥ 0 et flot à coût min.

Calculer un plus court chemin d'un sommet origine o vers un sommet destination d dans un graphe avec des longueurs ≥ 0 peut en fait se ramener à acheminer une unité de flot de o à d à coût minimum : coûts unitaires $=$ longueurs des arcs, et capacités ≥ 1.

Ainsi, au lieu d'utiliser l'algorithme de Dijkstra, on pourrait utiliser n'importe quelle méthode de résolution efficace pour le flot à coût minimum (comme la programmation linéaire, par exemple).

Si le graphe est non orienté, il suffit de remplacer chaque arête par 2 arcs opposés de même longueur et ayant les mêmes extrémités.

Plus court chemin avec longueurs ≥ 0 et flot à coût min.

Calculer un plus court chemin d'un sommet origine o vers un sommet destination d dans un graphe avec des longueurs ≥ 0 peut en fait se ramener à acheminer une unité de flot de o à d à coût minimum : coûts unitaires $=$ longueurs des arcs, et capacités ≥ 1.

Ainsi, au lieu d'utiliser l'algorithme de Dijkstra, on pourrait utiliser n'importe quelle méthode de résolution efficace pour le flot à coût minimum (comme la programmation linéaire, par exemple).

Si le graphe est non orienté, il suffit de remplacer chaque arête par 2 arcs opposés de même longueur et ayant les mêmes extrémités.

Plus court chemin avec longueurs ≥ 0 et flot à coût min.

Calculer un plus court chemin d'un sommet origine o vers un sommet destination d dans un graphe avec des longueurs ≥ 0 peut en fait se ramener à acheminer une unité de flot de o à d à coût minimum : coûts unitaires $=$ longueurs des arcs, et capacités ≥ 1.

Ainsi, au lieu d'utiliser l'algorithme de Dijkstra, on pourrait utiliser n'importe quelle méthode de résolution efficace pour le flot à coût minimum (comme la programmation linéaire, par exemple).

Si le graphe est non orienté, il suffit de remplacer chaque arête par 2 arcs opposés de même longueur et ayant les mêmes extrémités.

Arbres et arbres de connexion (1/6)

Définition

Un **arbre** *est un graphe non orienté dans lequel il existe un unique chemin non orienté (***chaîne***) entre toute paire de sommets.*

Remarque

Les arbres (non orientés) et les arborescences (orientées) sont des structures très utilisées en informatique : arbres de syntaxe (compilation), arborescences de fichiers (système), arbres binaires de recherche / tas (structures de données), etc.

Remarque

Un arbre contient une arête de moins que de sommets.

Arbres et arbres de connexion (1/6)

Définition

Un **arbre** *est un graphe non orienté dans lequel il existe un unique chemin non orienté (*__chaîne__*) entre toute paire de sommets.*

Remarque

Les arbres (non orientés) et les arborescences (orientées) sont des structures très utilisées en informatique : arbres de syntaxe (compilation), arborescences de fichiers (système), arbres binaires de recherche / tas (structures de données), etc.

Remarque

Un arbre contient une arête de moins que de sommets.

Arbres et arbres de connexion (1/6)

Définition

Un **arbre** *est un graphe non orienté dans lequel il existe un unique chemin non orienté (*__chaîne__*) entre toute paire de sommets.*

Remarque

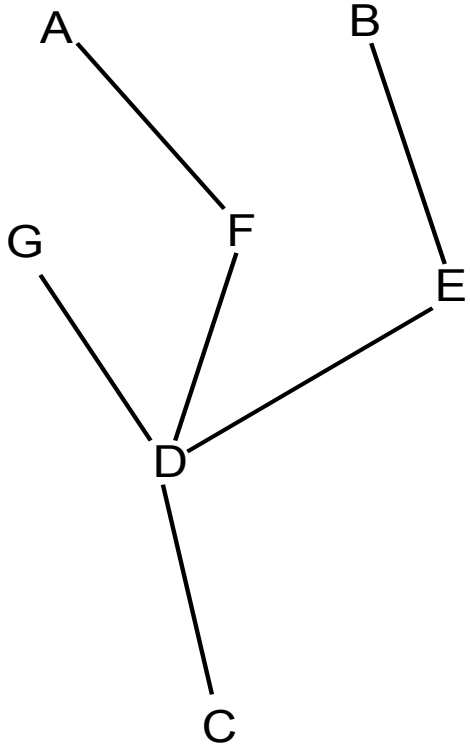
Les arbres (non orientés) et les arborescences (orientées) sont des structures très utilisées en informatique : arbres de syntaxe (compilation), arborescences de fichiers (système), arbres binaires de recherche / tas (structures de données), etc.

Remarque

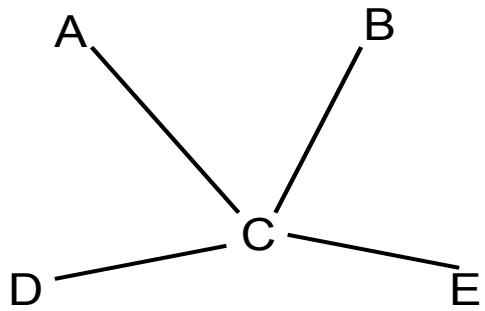
Un arbre contient une arête de moins que de sommets.

Arbres et arbres de connexion (2/6)

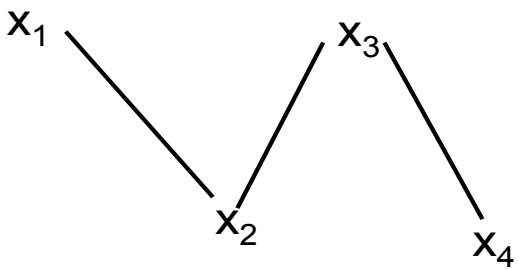
Exemples d'arbres



Arbre G_1



Arbre G_2



Arbre G_3

Arbres et arbres de connexion (3/6)

On considère un ensemble d'utilisateurs d'un réseau, qui doivent être connectés entre eux pour pouvoir communiquer.

On modélise ces utilisateurs par les sommets d'un graphe, et on associe à chaque paire d'utilisateurs une arête munie d'un poids.

Cette arête représente tou(te)s les liens/liaisons nécessaires aux 2 utilisateurs concernés pour pouvoir communiquer entre eux, et son poids est la somme des coûts de réservation de ces liens/liaisons.

Arbres et arbres de connexion (3/6)

On considère un ensemble d'utilisateurs d'un réseau, qui doivent être connectés entre eux pour pouvoir communiquer.

On modélise ces utilisateurs par les sommets d'un graphe, et on associe à chaque paire d'utilisateurs une arête munie d'un poids.

Cette arête représente tou(te)s les liens/liaisons nécessaires aux 2 utilisateurs concernés pour pouvoir communiquer entre eux, et son poids est la somme des coûts de réservation de ces liens/liaisons.

Arbres et arbres de connexion (3/6)

On considère un ensemble d'utilisateurs d'un réseau, qui doivent être connectés entre eux pour pouvoir communiquer.

On modélise ces utilisateurs par les sommets d'un graphe, et on associe à chaque paire d'utilisateurs une arête munie d'un poids.

Cette arête représente tou(te)s les liens/liaisons nécessaires aux 2 utilisateurs concernés pour pouvoir communiquer entre eux, et son poids est la somme des coûts de réservation de ces liens/liaisons.

Arbres et arbres de connexion (4/6)

Définition

Le graphe obtenu est un graphe **complet**, *c'est-à-dire qu'il existe une arête entre toute paire de sommets.*

Dans cette représentation, il est en théorie possible qu'un même lien physique intervienne dans plusieurs arêtes, et donc que son coût puisse être compté plusieurs fois : c'est une représentation agrégée, utilisée pour des raisons de simplicité et de taille.

Une telle représentation est donc basée sur l'hypothèse implicite que l'erreur d'estimation du coût total (due à cette agrégation) est négligeable devant ce coût total.

Arbres et arbres de connexion (4/6)

Définition

Le graphe obtenu est un graphe **complet**, *c'est-à-dire qu'il existe une arête entre toute paire de sommets.*

Dans cette représentation, il est en théorie possible qu'un même lien physique intervienne dans plusieurs arêtes, et donc que son coût puisse être compté plusieurs fois : c'est une représentation agrégée, utilisée pour des raisons de simplicité et de taille.

Une telle représentation est donc basée sur l'hypothèse implicite que l'erreur d'estimation du coût total (due à cette agrégation) est négligeable devant ce coût total.

Arbres et arbres de connexion (4/6)

Définition

Le graphe obtenu est un graphe **complet**, *c'est-à-dire qu'il existe une arête entre toute paire de sommets.*

Dans cette représentation, il est en théorie possible qu'un même lien physique intervienne dans plusieurs arêtes, et donc que son coût puisse être compté plusieurs fois : c'est une représentation agrégée, utilisée pour des raisons de simplicité et de taille.

Une telle représentation est donc basée sur l'hypothèse implicite que l'erreur d'estimation du coût total (due à cette agrégation) est négligeable devant ce coût total.

Arbres et arbres de connexion (5/6)

Définition

Un **arbre de connexion** *est une structure permettant de relier entre eux tous les utilisateurs.*

Pour obtenir une telle structure, il faut qu'il existe au moins une chaîne entre toute paire de sommets.

Il est par ailleurs inutile qu'il en existe plus qu'une : sinon, on pourrait construire à partir de cette structure une structure moins chère reliant également les utilisateurs.

Il faut et il suffit donc qu'il existe une unique chaîne entre toute paire de sommets : c'est pourquoi on l'appelle **arbre** de connexion.

Arbres et arbres de connexion (5/6)

Définition

Un **arbre de connexion** *est une structure permettant de relier entre eux tous les utilisateurs.*

Pour obtenir une telle structure, il faut qu'il existe au moins une chaîne entre toute paire de sommets.

Il est par ailleurs inutile qu'il en existe plus qu'une : sinon, on pourrait construire à partir de cette structure une structure moins chère reliant également les utilisateurs.

Il faut et il suffit donc qu'il existe une unique chaîne entre toute paire de sommets : c'est pourquoi on l'appelle **arbre** de connexion.

Arbres et arbres de connexion (5/6)

Définition

Un **arbre de connexion** *est une structure permettant de relier entre eux tous les utilisateurs.*

Pour obtenir une telle structure, il faut qu'il existe au moins une chaîne entre toute paire de sommets.

Il est par ailleurs inutile qu'il en existe plus qu'une : sinon, on pourrait construire à partir de cette structure une structure moins chère reliant également les utilisateurs.

Il faut et il suffit donc qu'il existe une unique chaîne entre toute paire de sommets : c'est pourquoi on l'appelle **arbre** de connexion.

Arbres et arbres de connexion (5/6)

Définition

Un **arbre de connexion** *est une structure permettant de relier entre eux tous les utilisateurs.*

Pour obtenir une telle structure, il faut qu'il existe au moins une chaîne entre toute paire de sommets.

Il est par ailleurs inutile qu'il en existe plus qu'une : sinon, on pourrait construire à partir de cette structure une structure moins chère reliant également les utilisateurs.

Il faut et il suffit donc qu'il existe une unique chaîne entre toute paire de sommets : c'est pourquoi on l'appelle **arbre** de connexion.

Arbres et arbres de connexion (6/6)

Définition

Dans un graphe non orienté où un poids est associé à chaque arête, un **arbre couvrant de poids minimum** *est un arbre contenant tous les sommets de ce graphe, et dont la somme des poids des arêtes est la plus petite possible.*

Remarque

Un arbre de connexion dont le coût total est aussi faible que possible est donc un arbre couvrant de poids minimum dans le graphe complet associé aux utilisateurs.

Arbres et arbres de connexion (6/6)

Définition

Dans un graphe non orienté où un poids est associé à chaque arête, un **arbre couvrant de poids minimum** *est un arbre contenant tous les sommets de ce graphe, et dont la somme des poids des arêtes est la plus petite possible.*

Remarque

Un arbre de connexion dont le coût total est aussi faible que possible est donc un arbre couvrant de poids minimum dans le graphe complet associé aux utilisateurs.

Calcul efficace d'un arbre de connexion

Remarque

On peut calculer un arbre couvrant de poids minimum dans un graphe non orienté (complet ou non) efficacement, et notamment via un algorithme glouton appelé **algorithme de Kruskal**.

Remarque

*Si on n'utilise pas la représentation agrégée, calculer un arbre de connexion de coût aussi faible que possible ne peut plus se faire via cet algorithme, ni même de façon efficace (si $P \neq NP$) : en effet, le problème obtenu est alors NP-difficile (***arbre de Steiner***).*

Calcul efficace d'un arbre de connexion

Remarque

On peut calculer un arbre couvrant de poids minimum dans un graphe non orienté (complet ou non) efficacement, et notamment via un algorithme glouton appelé **algorithme de Kruskal**.

Remarque

*Si on n'utilise pas la représentation agrégée, calculer un arbre de connexion de coût aussi faible que possible ne peut plus se faire via cet algorithme, ni même de façon efficace (si $P \neq NP$) : en effet, le problème obtenu est alors NP-difficile (***arbre de Steiner***).*

Principe de l'algorithme de Kruskal

Définition (Algorithme de Kruskal)

Algorithme glouton qui construit, dans un graphe non orienté, un arbre couvrant de poids minimum arête par arête en introduisant, itérativement et sans créer de **cycle** *(= chaîne dont les extrémités sont reliées par une arête), ces arêtes par ordre croissant des poids.*

Remarque

L'algorithme de Kruskal gère un ensemble d'arêtes sélectionnées qui représente, à chaque itération, un sous-ensemble des arêtes d'un arbre couvrant de poids minimum.

À la fin de l'algorithme de Kruskal, les arêtes sélectionnées forment un arbre couvrant de poids minimum.

Principe de l'algorithme de Kruskal

Définition (Algorithme de Kruskal)

Algorithme glouton qui construit, dans un graphe non orienté, un arbre couvrant de poids minimum arête par arête en introduisant, itérativement et sans créer de **cycle** *(= chaîne dont les extrémités sont reliées par une arête), ces arêtes par ordre croissant des poids.*

Remarque

L'algorithme de Kruskal gère un ensemble d'arêtes sélectionnées qui représente, à chaque itération, un sous-ensemble des arêtes d'un arbre couvrant de poids minimum.

À la fin de l'algorithme de Kruskal, les arêtes sélectionnées forment un arbre couvrant de poids minimum.

Description de l'algorithme de Kruskal

Données : arêtes a_1, a_2, ..., a_m du graphe triées dans l'ordre croissant de leurs poids, soit $p(a_1) \leq p(a_2) \leq \ldots \leq p(a_m)$.

Initialisation : aucune arête sélectionnée et $i = 1$.

Tant que il y a strictement moins que (nombre de sommets-1) arêtes sélectionnées **faire**

  Si l'arête a_i ne forme pas de cycle avec les arêtes déjà sélectionnées **alors** sélectionner cette arête ;

  Fin Si

  $i = i + 1$;

Fin Tant que

Sortie : l'ensemble des arêtes sélectionnées forme un arbre couvrant de poids minimum.

Description de l'algorithme de Kruskal

Données : arêtes a_1, a_2, ..., a_m du graphe triées dans l'ordre croissant de leurs poids, soit $p(a_1) \leq p(a_2) \leq \ldots \leq p(a_m)$.

Initialisation : aucune arête sélectionnée et $i = 1$.

Tant que il y a strictement moins que (nombre de sommets-1) arêtes sélectionnées **faire**

  Si l'arête a_i ne forme pas de cycle avec les arêtes déjà sélectionnées **alors** sélectionner cette arête ;

  Fin Si

  $i = i + 1$;

Fin Tant que

Sortie : l'ensemble des arêtes sélectionnées forme un arbre couvrant de poids minimum.

Description de l'algorithme de Kruskal

Données : arêtes a_1, a_2, ..., a_m du graphe triées dans l'ordre croissant de leurs poids, soit $p(a_1) \leq p(a_2) \leq \ldots \leq p(a_m)$.

Initialisation : aucune arête sélectionnée et $i = 1$.

Tant que il y a strictement moins que (nombre de sommets-1) arêtes sélectionnées **faire**

Si l'arête a_i ne forme pas de cycle avec les arêtes déjà sélectionnées **alors** sélectionner cette arête ;
Fin Si
$i = i + 1$;

Fin Tant que

Sortie : l'ensemble des arêtes sélectionnées forme un arbre couvrant de poids minimum.

Description de l'algorithme de Kruskal

Données : arêtes a_1, a_2, ..., a_m du graphe triées dans l'ordre croissant de leurs poids, soit $p(a_1) \leq p(a_2) \leq \ldots \leq p(a_m)$.

Initialisation : aucune arête sélectionnée et $i = 1$.

Tant que il y a strictement moins que (nombre de sommets-1) arêtes sélectionnées **faire**

> **Si** l'arête a_i ne forme pas de cycle avec les arêtes déjà sélectionnées **alors** sélectionner cette arête ;
> **Fin Si**
> $i = i + 1$;

Fin Tant que

Sortie : l'ensemble des arêtes sélectionnées forme un arbre couvrant de poids minimum.

Description de l'algorithme de Kruskal

Données : arêtes a_1, a_2, ..., a_m du graphe triées dans l'ordre croissant de leurs poids, soit $p(a_1) \leq p(a_2) \leq \ldots \leq p(a_m)$.

Initialisation : aucune arête sélectionnée et $i = 1$.

Tant que il y a strictement moins que (nombre de sommets-1) arêtes sélectionnées **faire**

> **Si** l'arête a_i ne forme pas de cycle avec les arêtes déjà sélectionnées **alors** sélectionner cette arête ;
> **Fin Si**
> $i = i + 1$;

Fin Tant que

Sortie : l'ensemble des arêtes sélectionnées forme un arbre couvrant de poids minimum.

Description de l'algorithme de Kruskal

Données : arêtes a_1, a_2, ..., a_m du graphe triées dans l'ordre croissant de leurs poids, soit $p(a_1) \leq p(a_2) \leq \ldots \leq p(a_m)$.

Initialisation : aucune arête sélectionnée et $i = 1$.

Tant que il y a strictement moins que (nombre de sommets-1) arêtes sélectionnées **faire**

> **Si** l'arête a_i ne forme pas de cycle avec les arêtes déjà sélectionnées **alors** sélectionner cette arête ;
> **Fin Si**
> $i = i + 1$;

Fin Tant que

Sortie : l'ensemble des arêtes sélectionnées forme un arbre couvrant de poids minimum.

Description de l'algorithme de Kruskal

Données : arêtes a_1, a_2, ..., a_m du graphe triées dans l'ordre croissant de leurs poids, soit $p(a_1) \leq p(a_2) \leq \ldots \leq p(a_m)$.

Initialisation : aucune arête sélectionnée et $i = 1$.

Tant que il y a strictement moins que (nombre de sommets-1) arêtes sélectionnées **faire**

> **Si** l'arête a_i ne forme pas de cycle avec les arêtes déjà sélectionnées **alors** sélectionner cette arête ;
> **Fin Si**
> $i = i + 1$;

Fin Tant que

Sortie : l'ensemble des arêtes sélectionnées forme un arbre couvrant de poids minimum.

Illustration – calcul d'un arbre de connexion à l'aide de l'algorithme de Kruskal (1/12)

Les arêtes qui n'apparaissent pas ont un poids ≥ 7.

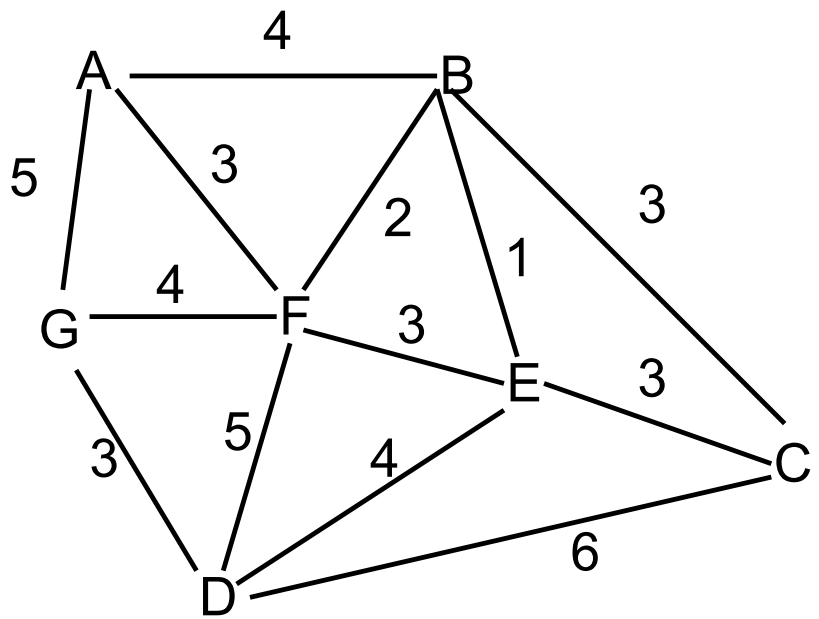


Illustration – calcul d'un arbre de connexion à l'aide de l'algorithme de Kruskal (2/12)

Tri des arêtes par poids croissants : $p([B,E]) = 1, p([B,F]) = 2, p([A,F]) = 3, p([B,C]) = 3, p([C,E]) = 3, p([D,G]) = 3, p([E,F]) = 3, p([D,E]) = 4, p([F,G]) = 4, p([A,B]) = 4, p([A,G]) = 5, p([F,D]) = 5, p([D,C]) = 6.$

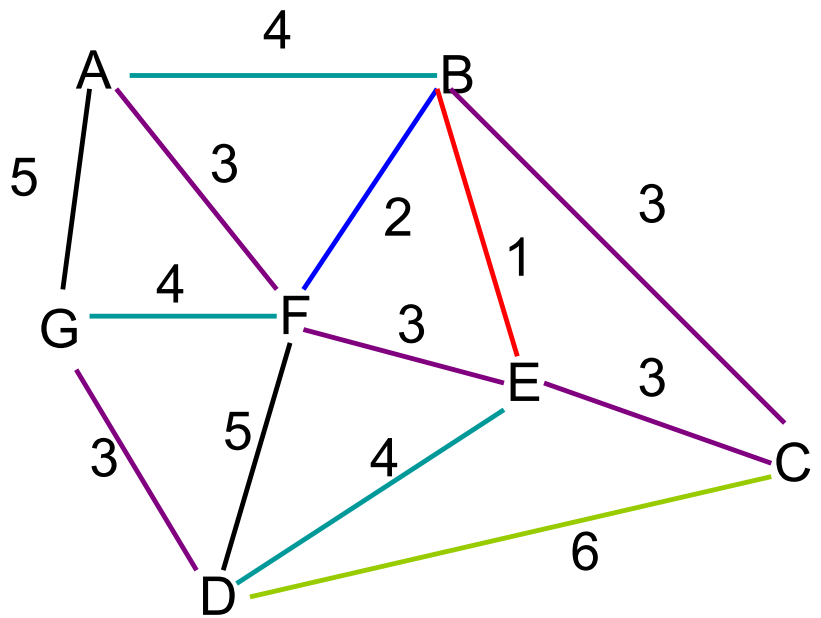


Illustration – calcul d'un arbre de connexion à l'aide de l'algorithme de Kruskal (3/12)

Initialisation : aucune arête sélectionnée et $i = 1$.

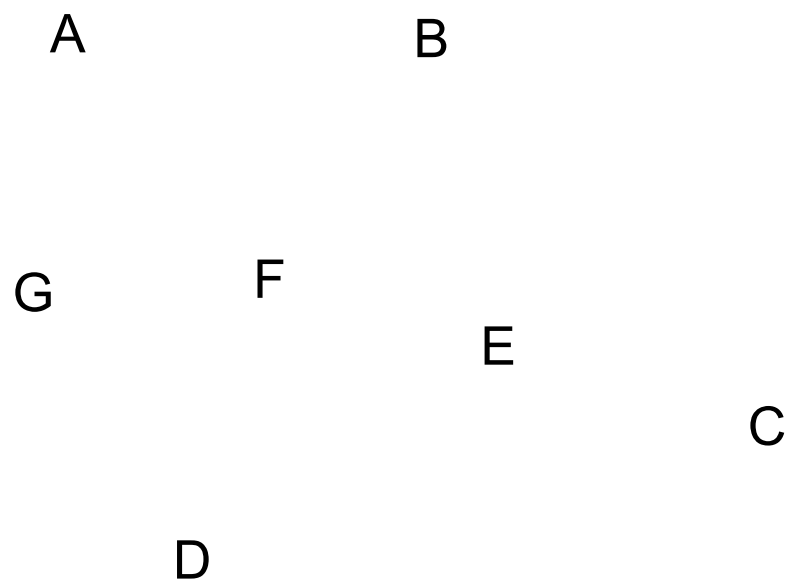


Illustration – calcul d'un arbre de connexion à l'aide de l'algorithme de Kruskal (4/12)

Liste des arêtes par poids croissants (itération 1) : $[B, E]$
$, [B, F], [A, F], [B, C], [C, E], [D, G], [E, F], [D, E], [F, G], [A, B], [A, G], [F, D], [D, C]$

Ajout de $a_1 = [B, E]$?

Oui, car cet ajout ne crée pas de cycle $\Rightarrow$ une arête ajoutée et $i = 2$.

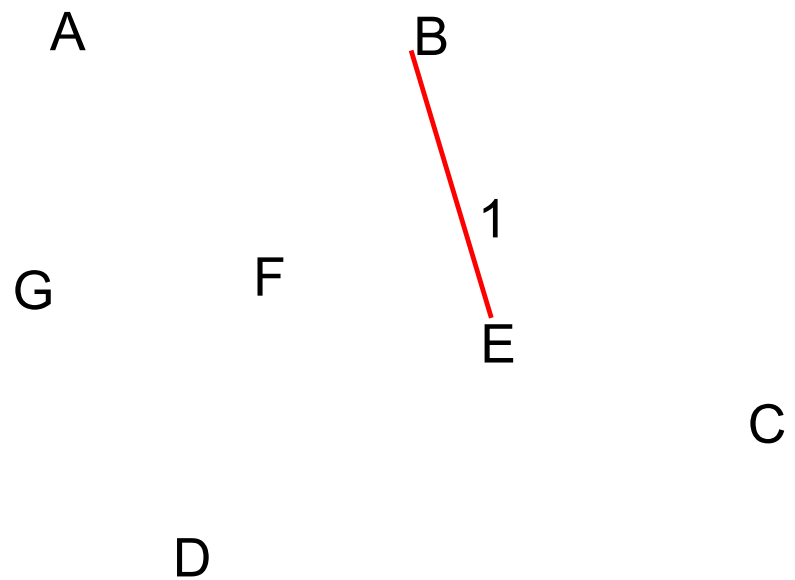


Illustration – calcul d'un arbre de connexion à l'aide de l'algorithme de Kruskal (5/12)

Liste des arêtes par poids croissants (itération 2) : $[B, E]$, $[B, F]$, $[A, F], [B, C], [C, E], [D, G], [E, F], [D, E], [F, G], [A, B], [A, G], [F, D], [D, C]$

Ajout de $a_2 = [B, F]$? Oui, car cet ajout ne crée pas de cycle $\Rightarrow$ deux arêtes ajoutées et $i = 3$.

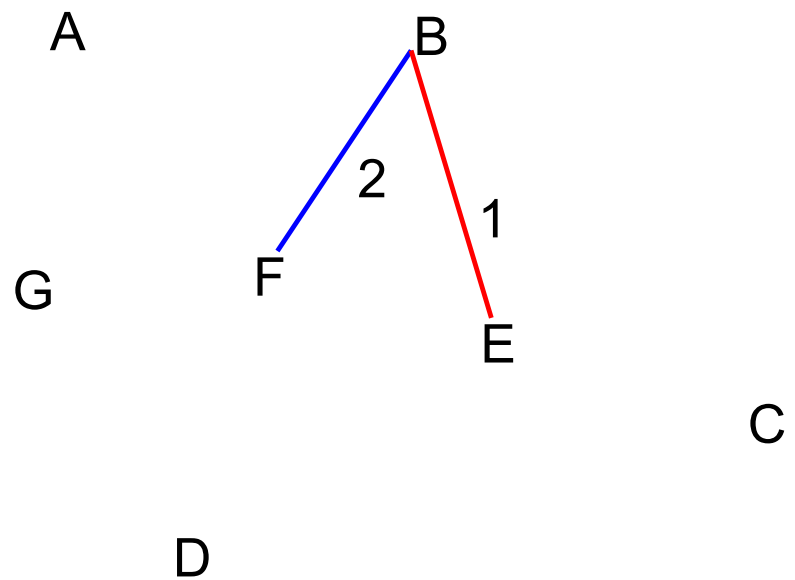


Illustration – calcul d'un arbre de connexion à l'aide de l'algorithme de Kruskal (6/12)

Liste des arêtes par poids croissants (itération 3) : $[B, E], [B, F]$, $[A, F]$, $[B, C], [C, E], [D, G], [E, F], [D, E], [F, G], [A, B], [A, G], [F, D], [D, C]$

Ajout de $a_3 = [A, F]$? Oui, car cet ajout ne crée pas de cycle $\Rightarrow$ trois arêtes ajoutées et $i = 4$.

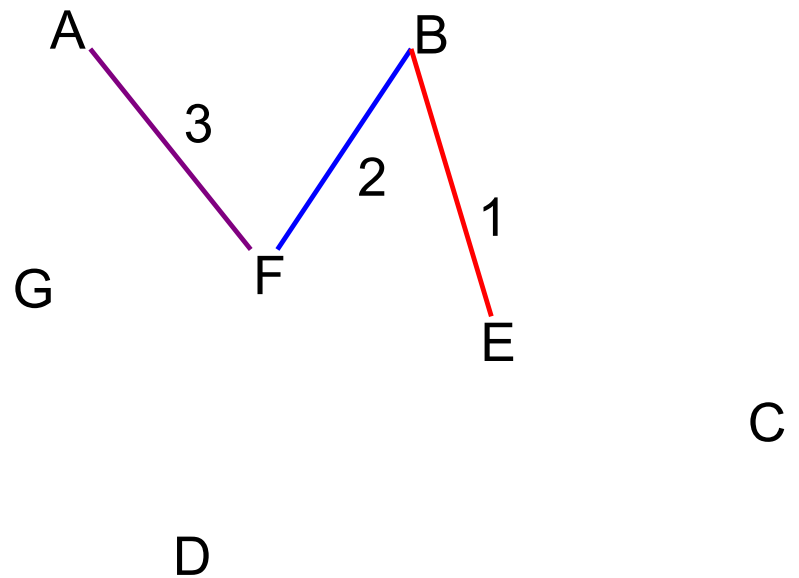


Illustration – calcul d'un arbre de connexion à l'aide de l'algorithme de Kruskal (7/12)

Liste des arêtes par poids croissants (itération 4) : $[B, E], [B, F], [A, F], [B, C]$, $[C, E], [D, G], [E, F], [D, E], [F, G], [A, B], [A, G], [F, D], [D, C]$

Ajout de $a_4 = [B, C]$? Oui, car cet ajout ne crée pas de cycle

$\Rightarrow$ quatre arêtes ajoutées et $i = 5$.

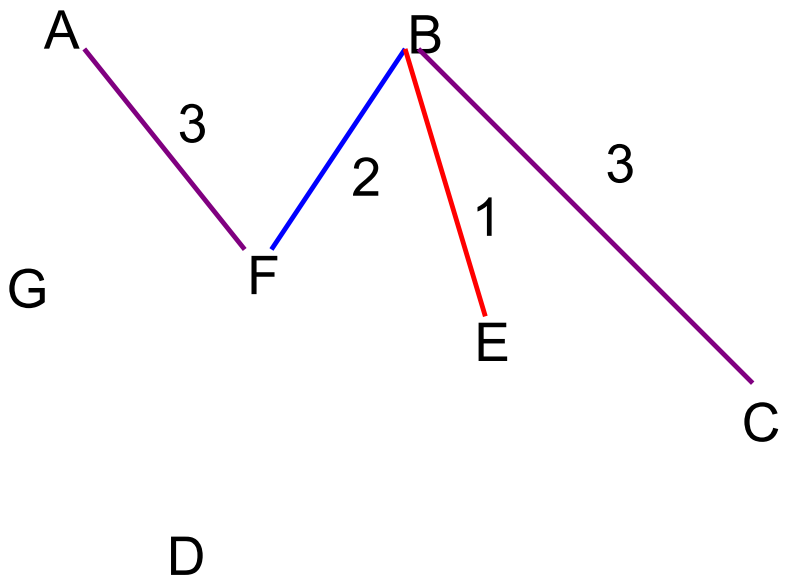


Illustration – calcul d'un arbre de connexion à l'aide de l'algorithme de Kruskal (8/12)

Liste des arêtes par poids croissants (itération 5) :

$[B, E], [B, F], [A, F], [B, C], [C, E]$,
$[D, G], [E, F], [D, E], [F, G], [A, B], [A, G], [F, D], [D, C]$

Ajout de $a_5 = [C, E]$? Non, car cela créerait un cycle $\Rightarrow i = 6$.

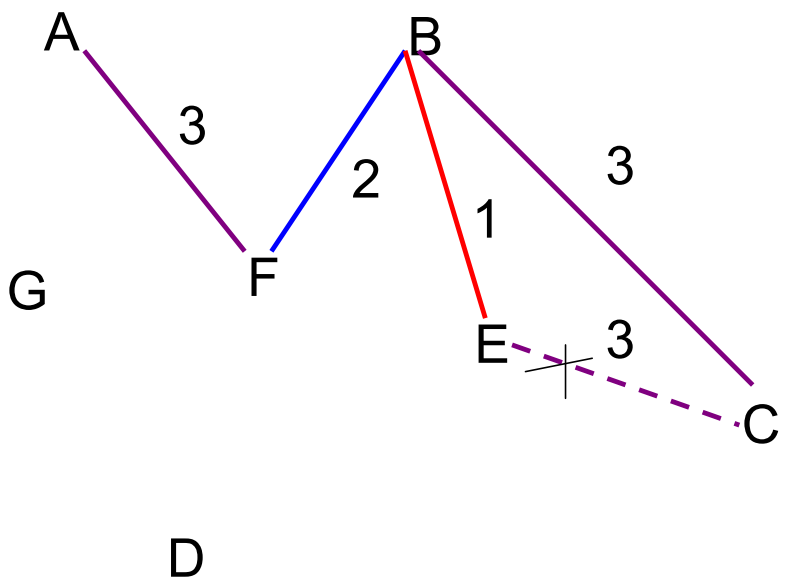


Illustration – calcul d'un arbre de connexion à l'aide de l'algorithme de Kruskal (9/12)

Liste des arêtes par poids croissants (itération 6) :
$[B, E], [B, F], [A, F], [B, C], [C, E], [D, G]$,
$[E, F], [D, E], [F, G], [A, B], [A, G], [F, D], [D, C]$

Ajout de $a_6 = [D, G]$? Oui, car cet ajout ne crée pas de cycle

$\Rightarrow$ cinq arêtes ajoutées et $i = 7$.

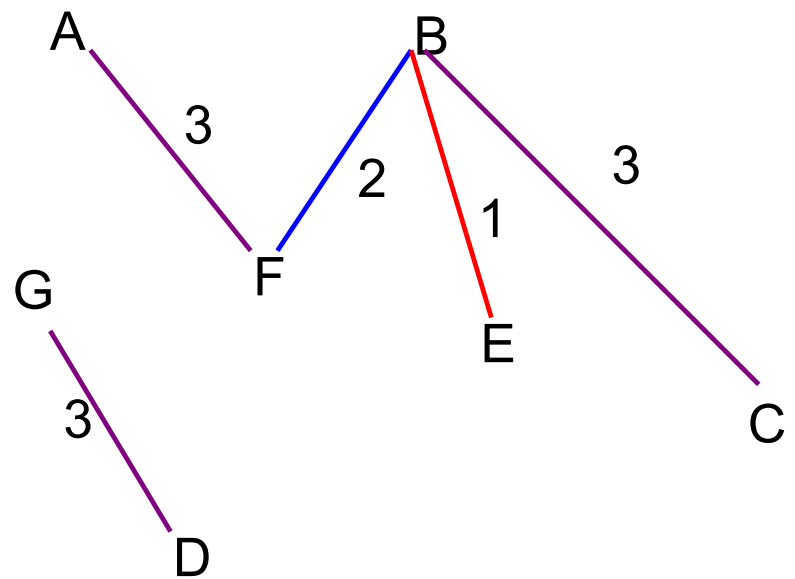


Illustration – calcul d'un arbre de connexion à l'aide de l'algorithme de Kruskal (10/12)

Liste des arêtes par poids croissants (itération 7) :
$[B, E], [B, F], [A, F], [B, C], [C, E], [D, G], [E, F]$,
$[D, E], [F, G], [A, B], [A, G], [F, D], [D, C]$

Ajout de $a_7 = [E, F]$? Non, car cela créerait un cycle $\Rightarrow i = 8$.

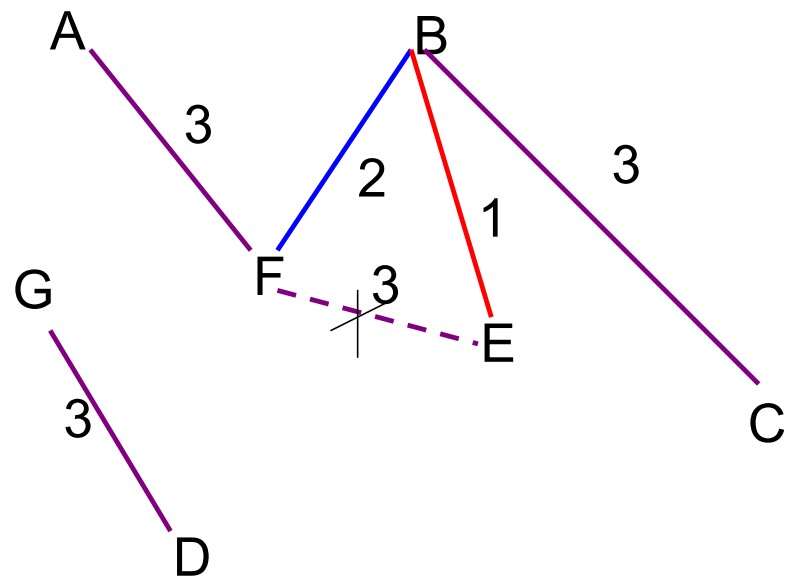


Illustration – calcul d'un arbre de connexion à l'aide de l'algorithme de Kruskal (11/12)

Liste des arêtes par poids croissants (itération 8) :
$[B, E], [B, F], [A, F], [B, C], [C, E], [D, G], [E, F], [D, E]$,
$[F, G], [A, B], [A, G], [F, D], [D, C]$

Ajout de $a_8 = [D, E]$? Oui, car cet ajout ne crée pas de cycle.

STOP car critère d'arrêt atteint : nombre d'arêtes sélectionnées $= 6 = 7 - 1$.

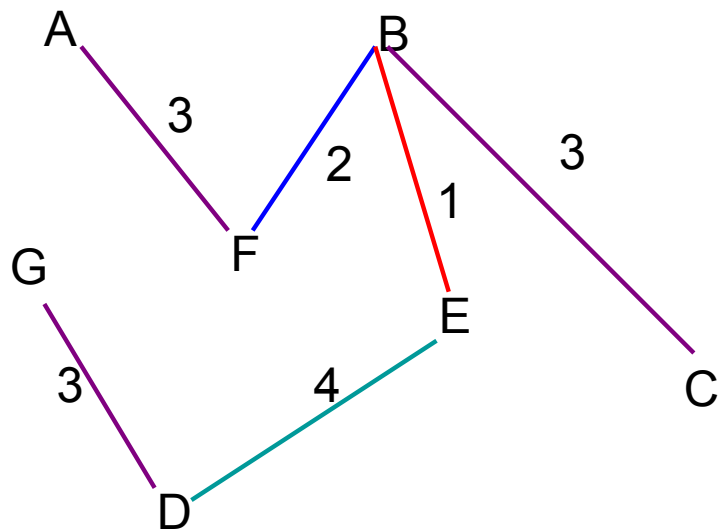
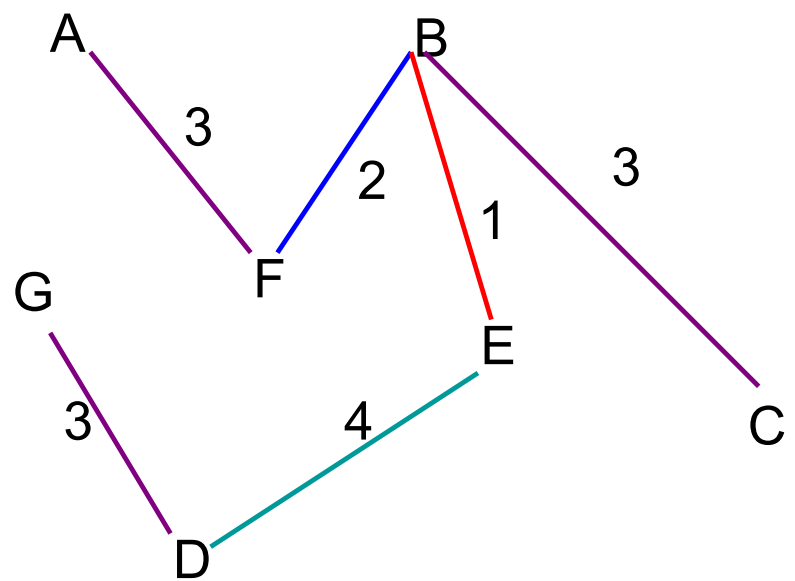


Illustration – calcul d'un arbre de connexion à l'aide de l'algorithme de Kruskal (12/12)

Obtention d'un arbre couvrant de poids minimum égal à :
$p([B,E])+p([B,F])+p([A,F])+p([B,C])+p([D,G])+p([E,D])$
$= 1+2+3+3+3+4 = 16$



Arbre couvrant de poids minimum vs arborescence des plus courts chemins

ATTENTION : dans un graphe non orienté, calculer l'arborescence des plus courts chemins issus d'un sommet donné n'a aucune raison de donner le même résultat que calculer un arbre couvrant de poids minimum.

Par exemple, l'arborescence des plus courts chemins issus de A dans l'exemple précédent contient les 6 arêtes suivantes : $[A, B], [A, F], [A, G], [B, C], [B, E], [F, D]$, et est de poids 21.

Arbre couvrant de poids minimum vs arborescence des plus courts chemins

ATTENTION : dans un graphe non orienté, calculer l'arborescence des plus courts chemins issus d'un sommet donné n'a aucune raison de donner le même résultat que calculer un arbre couvrant de poids minimum.

Par exemple, l'arborescence des plus courts chemins issus de A dans l'exemple précédent contient les 6 arêtes suivantes : $[A, B], [A, F], [A, G], [B, C], [B, E], [F, D]$, et est de poids 21.