Recherche de Chemin de poids Minimal

Et utilisation de Jgraph

Nous avons créé un modèle personnel pour le substituer au DefaultGraphModel afin de fournir des services de manipulmation de graphe comme degre+ et ièmesucc.

Nous voulons utiliser ce modèle pour la création d'un graphe valué. Une fois le graphe créé nous voulons réaliser une recherche de chemin de poids minimal entre deux sommets désignés. Le chemin ainsi trouvé sera indiqué par une coloration spécifique des arêtes qui le composent.

Votre application doit permettre de modifier le poids d'un arc afin de donner la possibilité de changer le chemin de poids minimal.

Vous devez rendre un **rapport papier** sur votre réalisation. Les sources seront transmises par voie électronique. N'oubliez pas de préciser vos noms sur le rapport ainsi que dans les noms d'archives transmises et dans les fichiers contenus dans l'archive

Vous trouverez en annexe un document de présentation d'un algorithme de recherche de ce chemin de poids minimal.

Vous trouverez sous gi/LO42/TP un interface "MonGraphModel" et une classe "Default MonGraphModel " qui sont des exemples simplifiés de ce que vous pouvez faire pour répondre à la question du TP précédent.

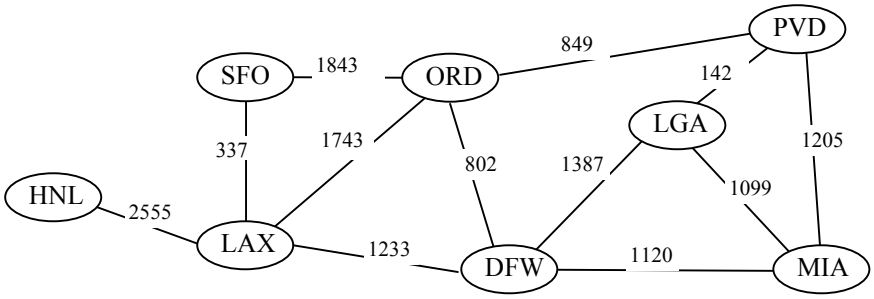
ANNEXE

Source : http://www.iro.umontreal.ca/~hamelsyl/chemins4.pdf

Chemin de poids minimal entre deux sommets

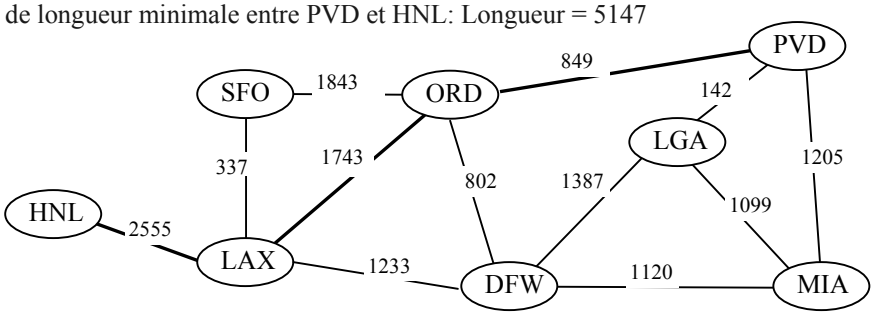
Graphe avec poids

- Dans un graphe avec poids, chaque arête garde en mémoire une valeur numérique, appelée le poids de l'arête
- Le poids d'une arête peut représenter une distance, un coût, …
- Exemple:
  - Dans ce graphe, le poids d'une arête représente la distance entre deux aéroports



- Étant donné un graphe et deux sommets u et v, on veut trouver le chemin de poids total minimal entre u et v.
  - on va appeler la longueur d'un chemin la somme des poids des arêtes faisant parties de ce chemin
  - le chemin de poids minimal (ou longueur minimale) entre u et v sera appelé la distance entre u et v

- Exemple:
  Chemin de longueur minimale entre PVD et HNL: Longueur = 5147

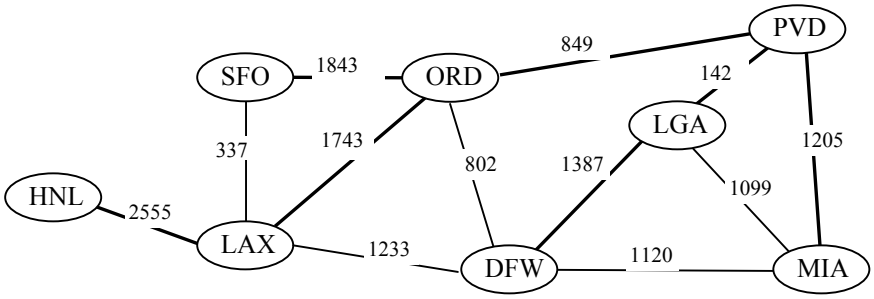


Propriétés

Propriété 1:
Un sous-chemin d'un chemin de coût minimal est un chemin de coût minimal

Propriété 2:
Étant donné un sommet v d'un graphe non-orienté et connexe G, il existe un arbre couvrant composé des chemins de coût minimal entre v et tous les autres sommets du graphe

Exemple: Arbres des chemins de coût minimal de PVD

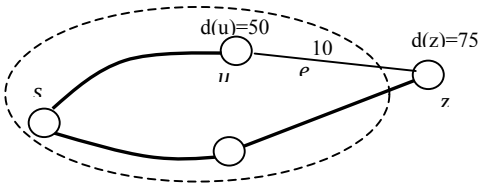


Algorithme de Dijkstra

- La distance entre un sommet v et un sommet s est le poids total minimal d'un chemin entre v et s
- L'algorithme de Dijkstra est un algorithme glouton qui calcule les distances entre un sommet v et tous les autres sommets d'un graphe
- Ici, on va assumer:
  - Le graphe est connexe
  - Le graphe est non-orienté
  - Les poids des arêtes sont non-négatifs
- On va faire grossir un "nuage" de sommets, contenant au départ v et couvrant éventuellement tous les sommets
- On va donner une étiquette d(u) à chaque sommet, représentant la distance entre v et u dans le sous graphe constitué des sommets dans le nuage et des arêtes adjacentes
- À chaque étape:
  - On ajoute au nuage le sommet u extérieur au nuage qui a la plus petite étiquette d(u)
  - On met à jour les étiquettes des sommets adjacents à u

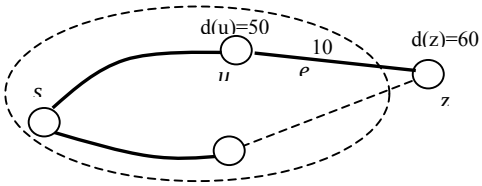
Relaxation des arêtes

- On appelle la mise à jour des étiquettes des sommets, après l'ajout d'un sommet u dans le nuage, la relaxation des arêtes
- Considérons une arête e=(u,z) telle que
  u est le sommet qu'on vient d'ajouter au nuage
  z n'est pas dans le nuage

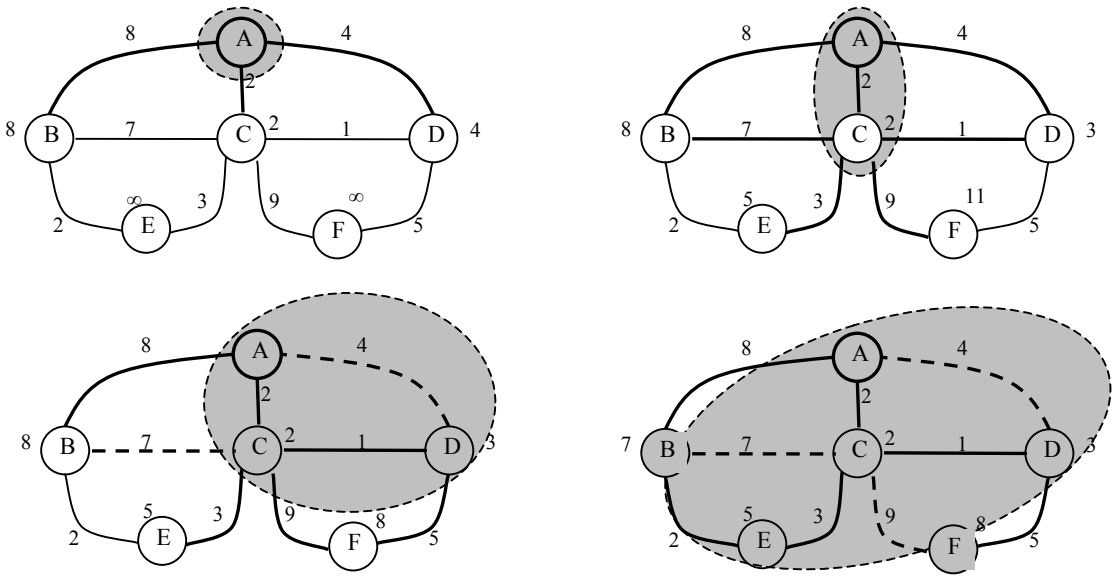


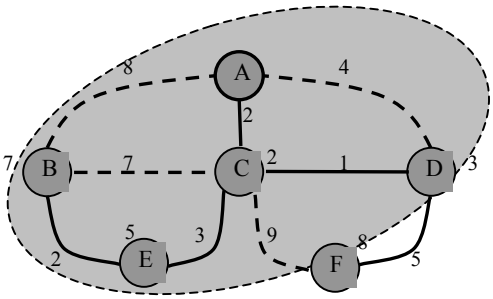
- La relaxation de e, consiste à mettre à jour la distance d(z) comme suit :

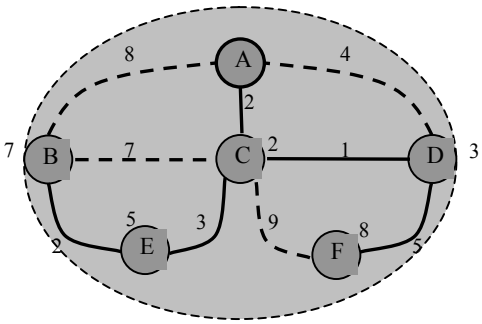
  $d(z) \leftarrow \min\{ d(z), d(u) + poids(e) \}$



Exemple:







Analyse de complexité de l'algorithme de Dijkstra

Insérer les sommets et leurs étiquettes correspondantes dans une liste avec priorités, prend un temps *O*(nlogn) si on insère les éléments un à un, ou un temps *O*(n) si on utilise une construction de bas en haut

À chaque tour de boucle TANT QUE, on prend un temps *O*(logn) pour enlever un sommet u du monceau, et un temps *O*(degré(u)logn) pour exécuter la procédure de relaxation

Le temps total d'exécution de la boucle "tant que" est donc de :

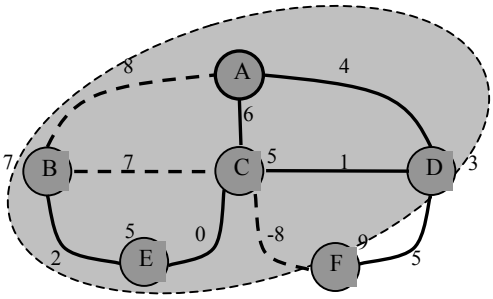
$$\sum_{v \in G} (1+deg(v)) \log \log n$$

La complexité en temps de Dijkstra est de *O*((n+m) logn)

Pourquoi, Dijkstra ne fonctionne pas avec des poids négatifs ?

- L'algorithme de Dijkstra est un algorithme glouton qui ajoute des sommets selon un ordre croissant de leur distance à v

  - Si un nœud ayant une arête incidente de poids négatif, est ajouté au nuage tardivement lors de l'algorithme alors, cela peut causer des problèmes pour la distance des sommets déjà dans le nuage



La vrai distance pour C est 1, mais C est déjà dans le nuage avec d(C)=5 !