Algorithmes gloutons ou voraces (greedy algorithms)

Idée: Pour résoudre un problème, on choisit un optimum local sans se soucier des effets que cela aura sur la suite (i.e pas de retour en arrière).

On aimerait que cette stratégie locale nous amène à un optimum global mais ce n'est pas toujours le cas.

Pourquoi intéressant?

- Facile à développer

- Dans certain cas, une preuve d'optimalité garantie l'optimalité de la solution globale trouver par l'algorithme glouton

Caractéristiques générales

On veut résoudre un problème de façon optimale:

1) On a une liste de candidats pour construire notre solution

ex. les arêtes d'un graphe, les pièces de monnaies disponibles

2) La solution sera un sous-ensemble ou multi-ensemble des candidats

3) L'algorithme vorace va maintenir un ensemble de candidats retenus (à la fin contient la solution) et de candidats rejetés

4) Une fonction "solution" regarde si l'ensemble courant de candidats retenus est une solution de notre problème (sans tenir compte de l'optimalité).

ex. A-t-on trouvé un chemin entre A et B? Est-ce que la somme des pièces est n?

5) Une fonction "complétable" qui décide s'il est possible d'ajouter un candidat à l'ensemble de candidats retenus

6) Une fonction "sélection" qui propose parmi les candidats restant celui qui a l'air le plus intéressant (optimum local)

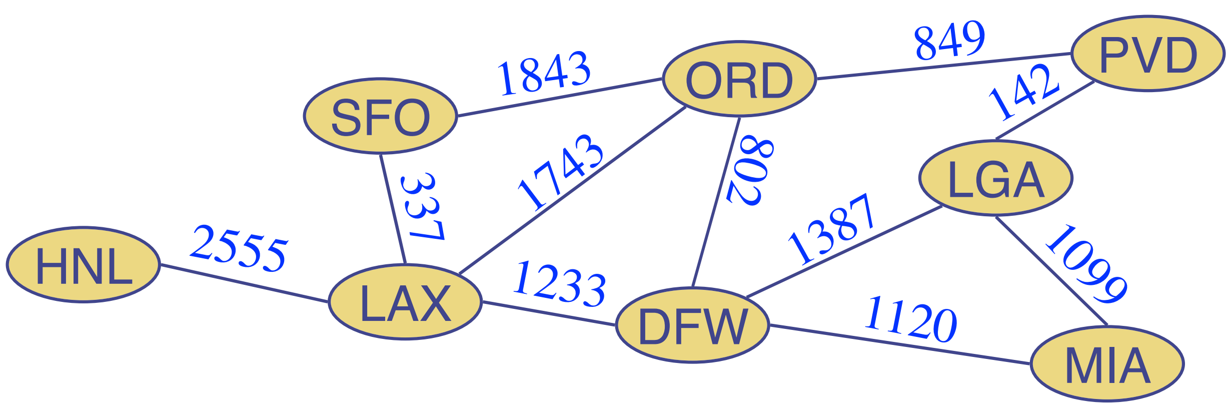
7) Une fonction "objective" (n'apparaît pas dans l'algo) qui donne une valeur à la solution trouvée.

ex. La longueur du chemin entre A et B. Le nombre de pièces utilisées pour faire la monnaie

Graphes

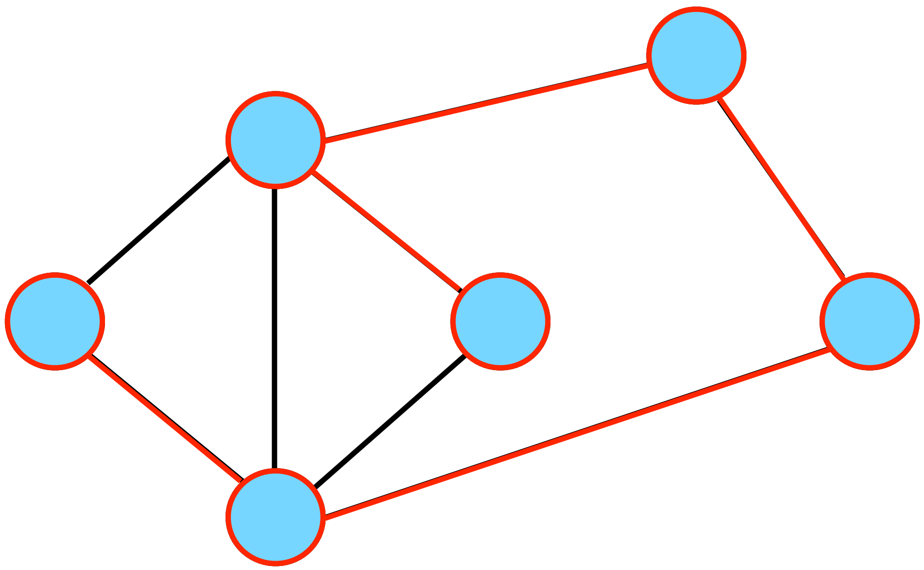
- Un graphe est une paire (N,A), où
  - N est un ensemble de noeuds (appelés sommets)
  - A est un multi-ensemble de paires de sommets appelées arêtes

- Exemple:
  - Chaque sommet représente un aéroport et garde en mémoire le code de 3 lettres représentant cet aéroport
  - Chaque arête représente une route aérienne entre deux villes et garde en mémoire le longueur de cette route



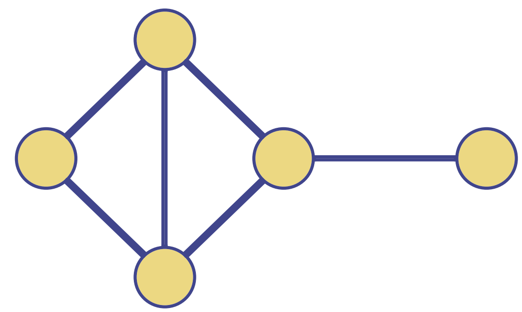
Quelques définitions

- Un sous-graphe S d'un graphe G est un graphe tel que:
  - Les sommets de S forment un sous-ensemble des sommets de G
  - Les arêtes de S forment un sous-ensemble des arêtes de G
  - Un sous-graphe est dit couvrant (spanning) s'il contient tous les sommets de G

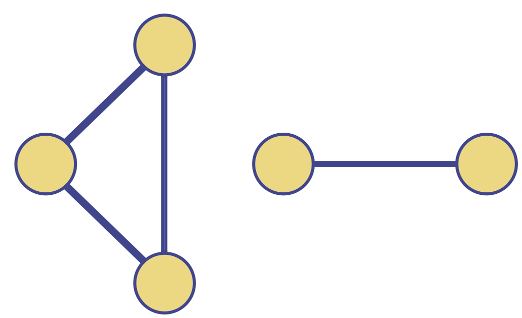


Quelques définitions (suite)

- Un graphe G est dit connexe s'il existe un chemin reliant chaque pair de sommets de G

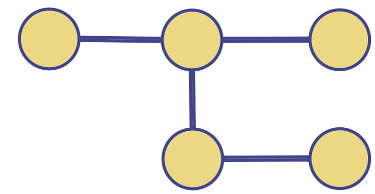


- Une composante connexe d'un graphe G est un sous-graphe connexe maximal de G



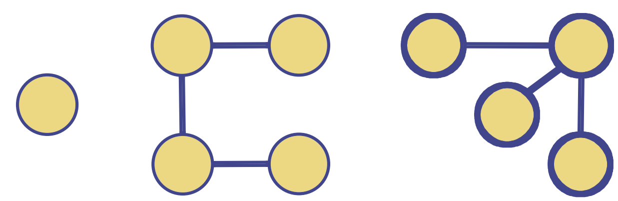
Quelques définitions (suite)

- Un arbre A (non raciné) est un graphe non orienté tel que
  - A est connexe
  - A ne contient pas de cycles



© Goodrich et Tamassia 2004

- Une forêt est un graphe non orienté ne contenant pas de cycles
  - Les composantes connexes d'une forêt sont donc des arbres

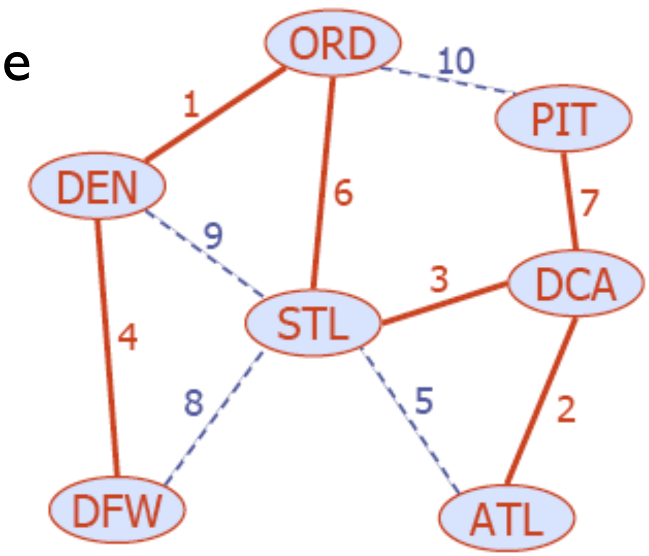


© Goodrich et Tamassia 2004

Arbre couvrant minimal:

- Un arbre couvrant d'un graphe est un sous-graphe couvrant qui est un arbre

- Arbre couvrant minimal (minimum spanning tree):
  - Arbre couvrant d'un graphe avec poids dont le poids total des arêtes est minimal

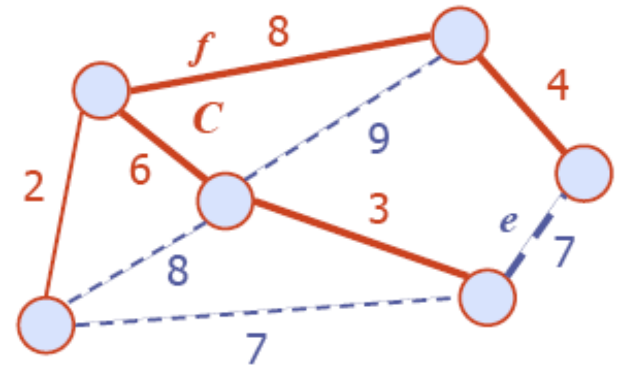


© adapté de Goodrich et Tamassia 2004

Propriété de cycles des ACM:

- **Propriété de cycles:**
  - Soit T un arbre couvrant d'un graphe avec poids G
  - Soit e une arête de G n'appartenant pas à T et soit C, le cycle obtenu lorsqu'on ajoute e à T
  - Si T est minimal, alors on a que pour toutes arêtes f dans C :

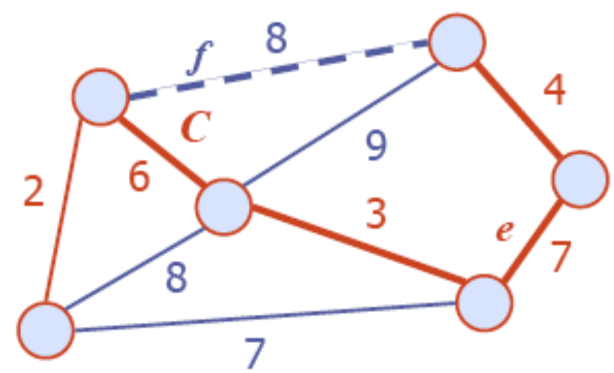
$$poids(f) \leq poids(e)$$



Remplacer f par e nous donne un arbre couvrant de plus petit poids total ...

- **Preuve:**
  - Par contradiction.

    Si poids(f) > poids(e), on obtient un arbre couvrant de plus petit poids en remplaçant l'arête f par l'arête e dans notre arbre T

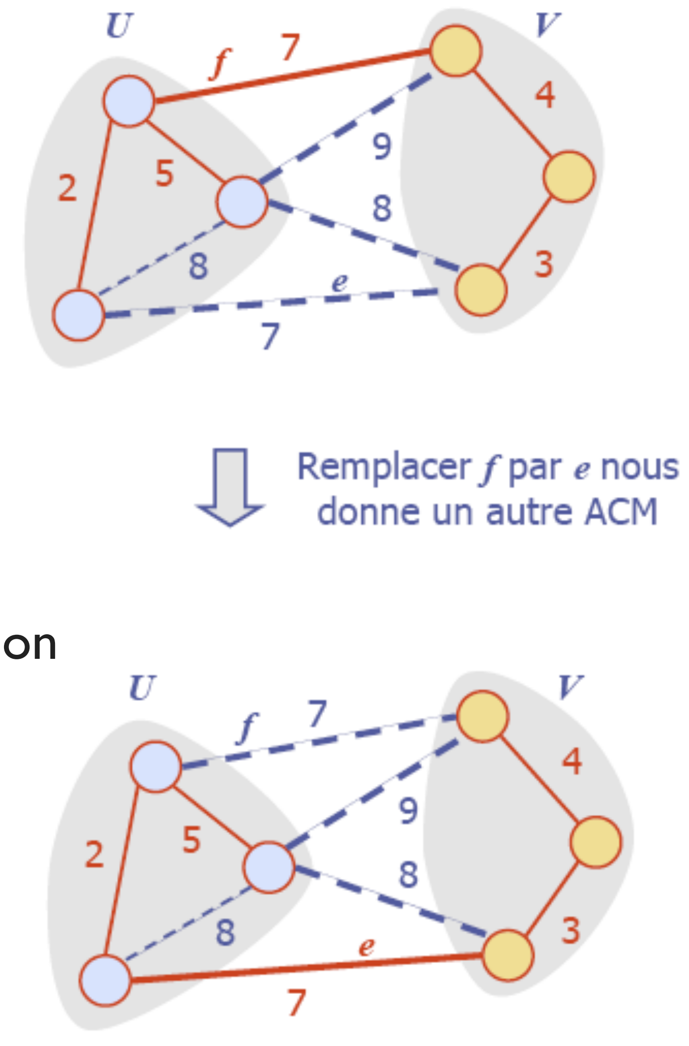


Propriété de partition des ACM:

- Propriété de partition:
  - Considérons une partition des sommets de G en deux ensembles U et V
  - Soit e une arête de poids minimal entre U et V
  - Alors, il existe un arbre couvrant minimal de G contenant e
- Preuve:
  - Soit T un arbre couvrant minimal de G
  - Si T ne contient pas e, soit C le cycle formé par l'addition de e à l'arbre T et soit f, une arête entre U et V
  - Par la propriété de cycles, on a que

    $$poids(f) \leq poids(e)$$

  - Comme on avait pris e de poids minimal, on a que poids(f) = poids(e) et alors on obtient un autre ACM en remplaçant f par e

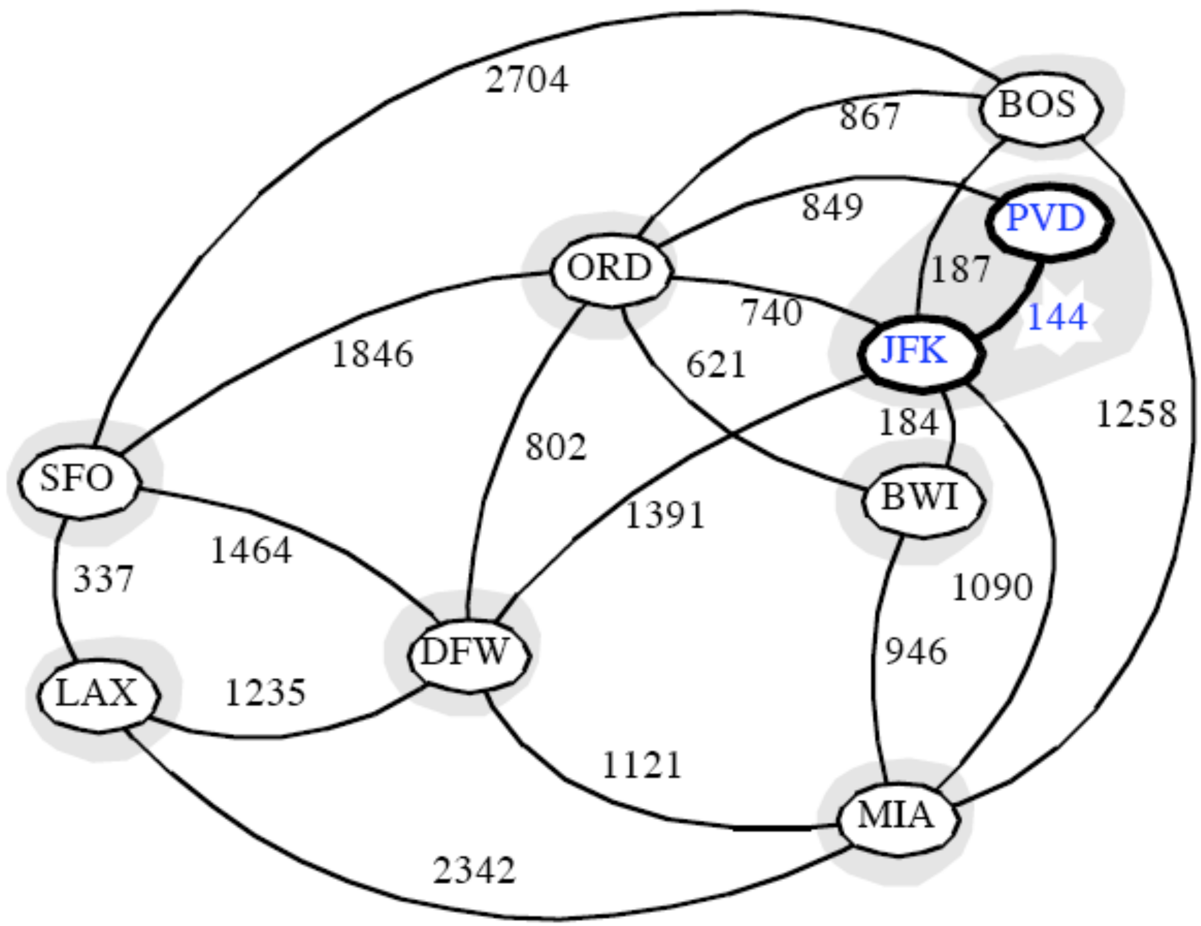


© adapté de Goodrich et Tamassia 2004

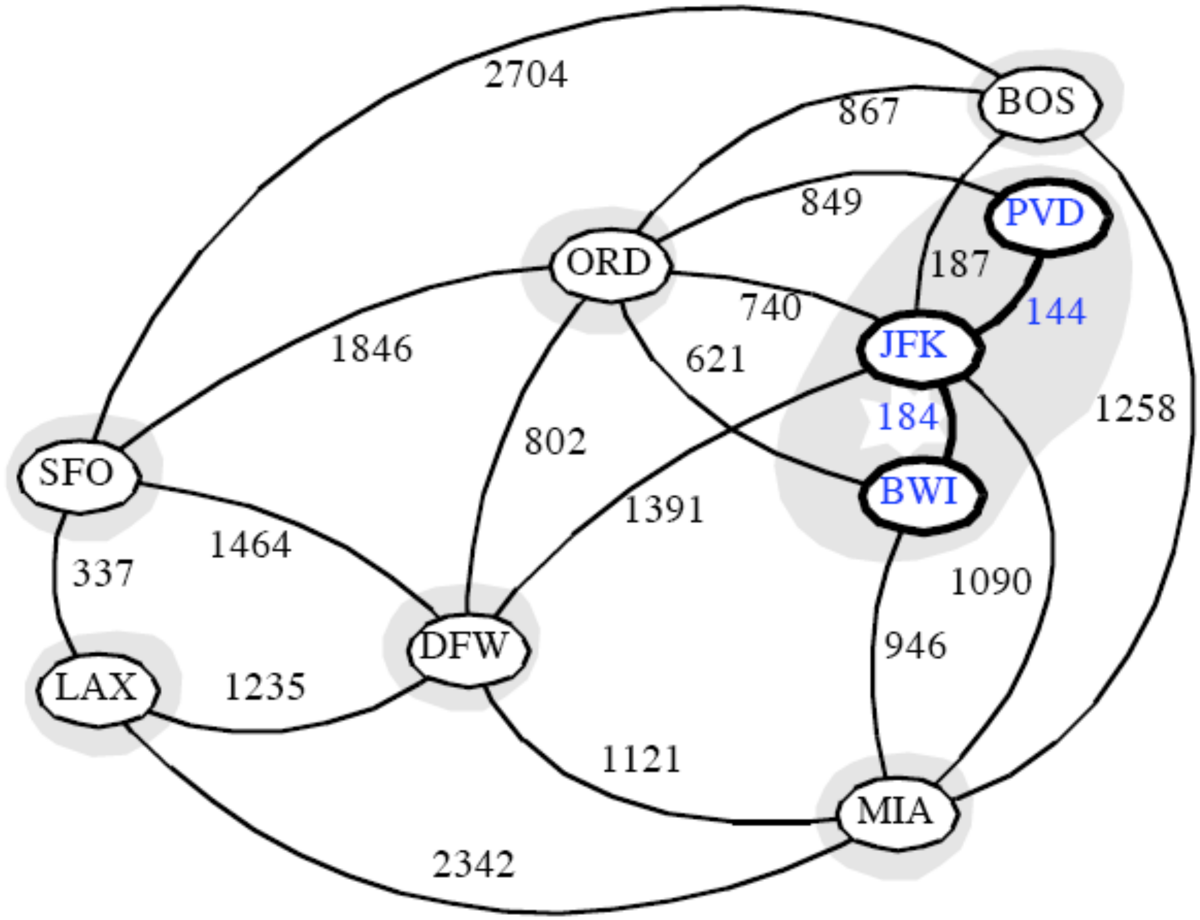
Algorithme Kruskal:

- L'algorithme maintient une forêt d'arbres
- À chaque itération, on choisit l'arête de coût minimal
- Cette arête est acceptée, si elle relit deux arbres distincts, sinon elle est rejetée (pourrait forme un cycle)
- L'algorithme se termine lorsqu'on a un seul arbre

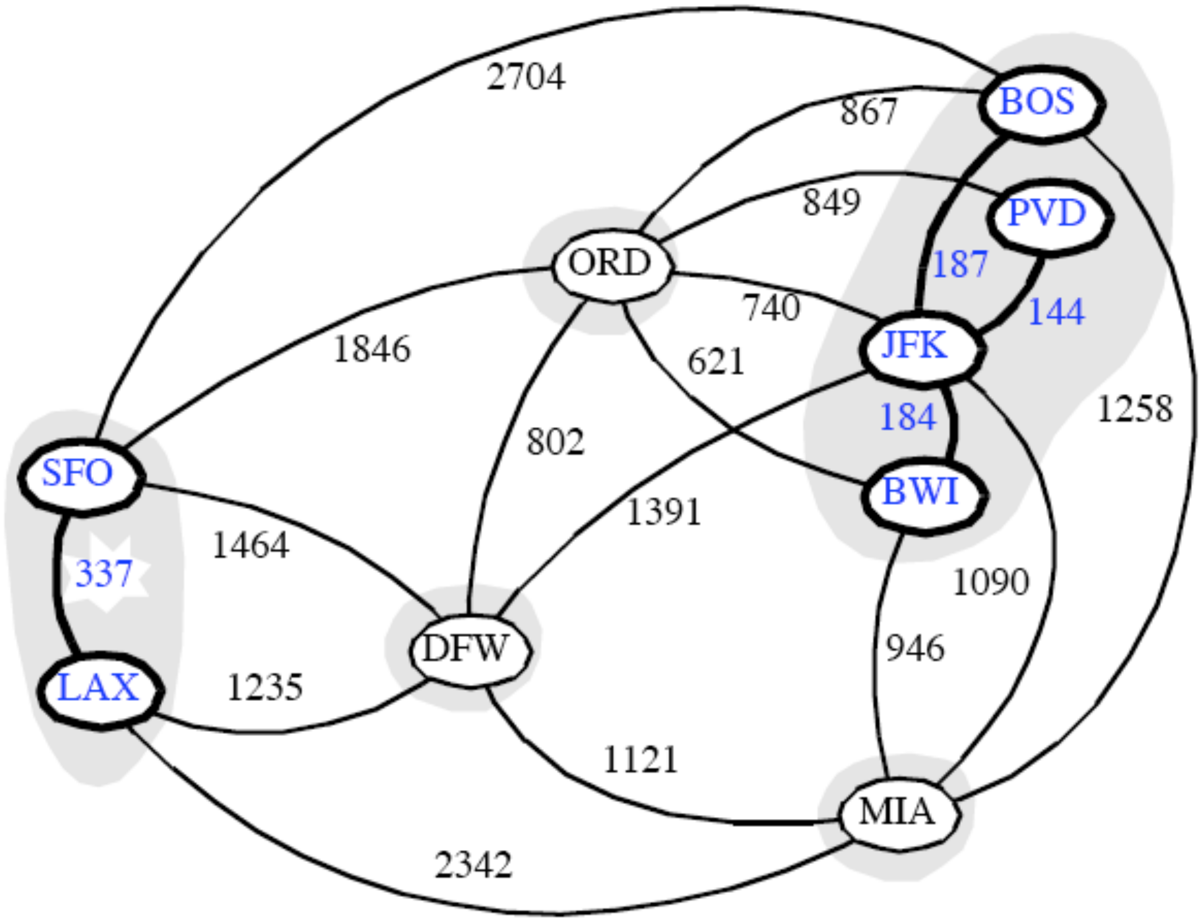
Exemple de Kruskal:



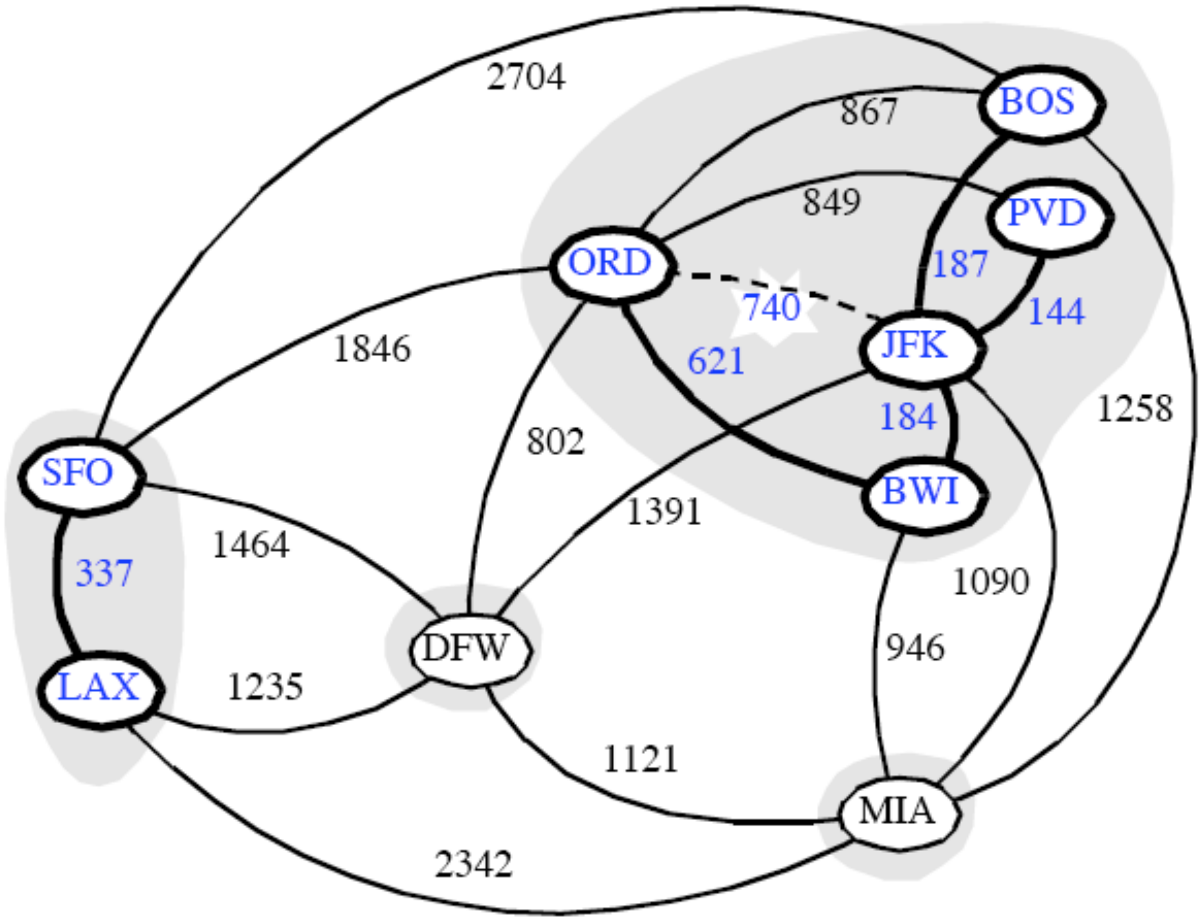
Exemple de Kruskal:



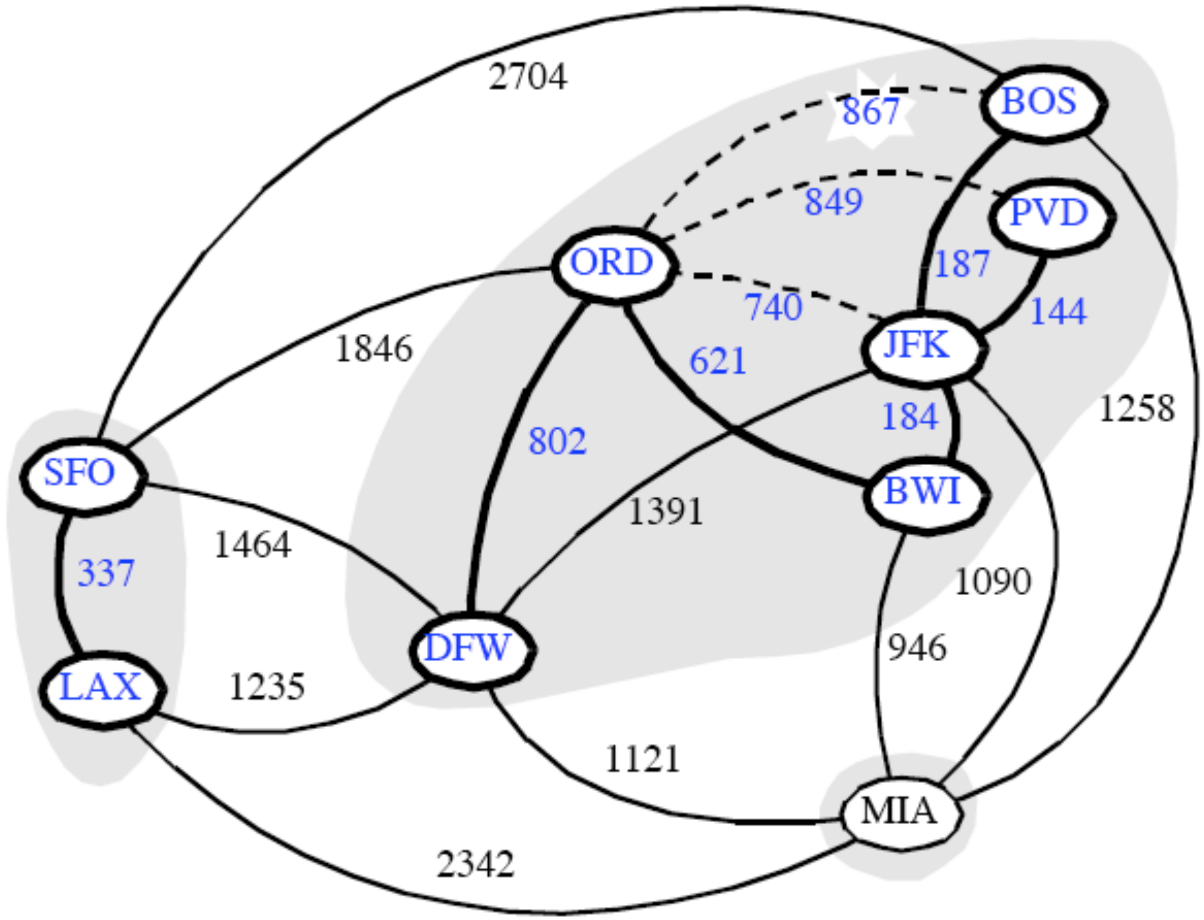
Exemple de Kruskal:



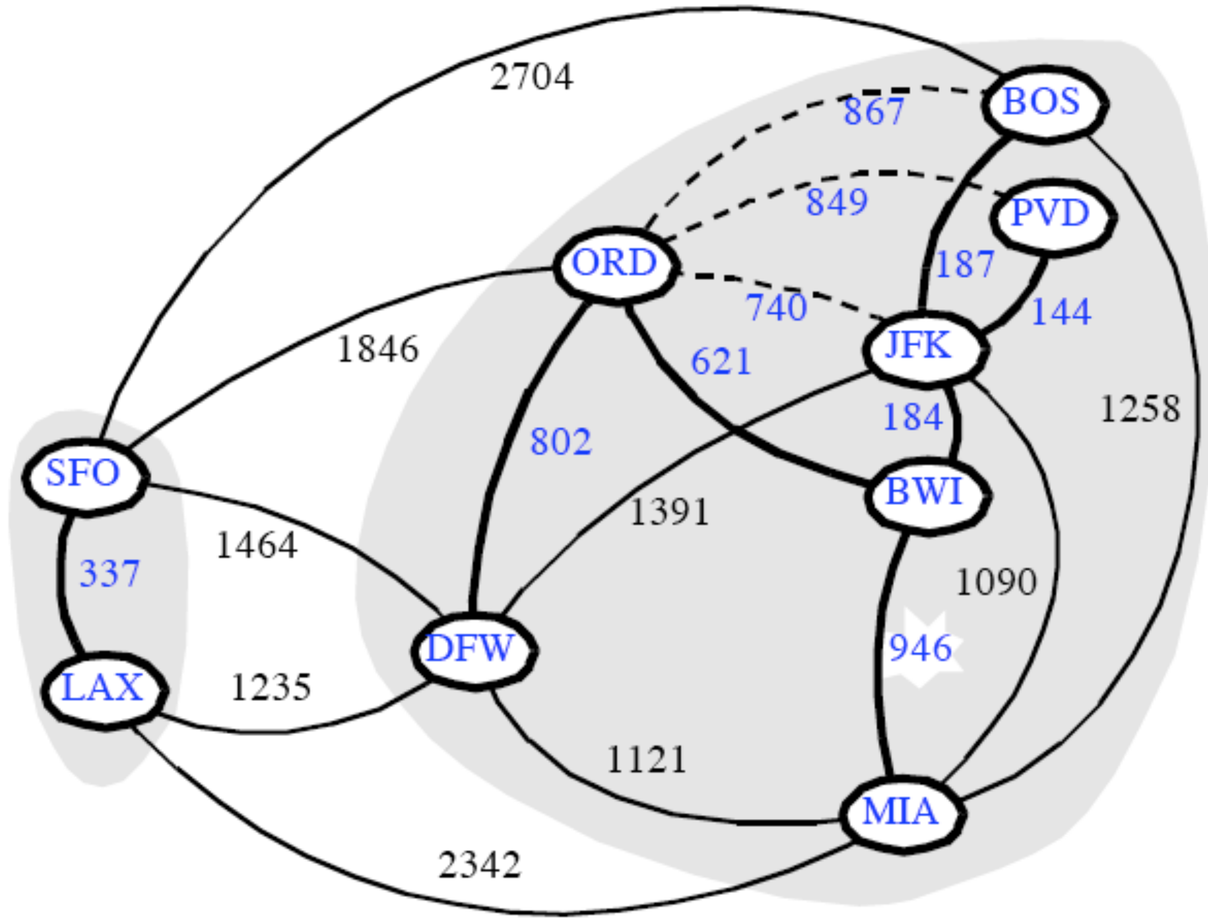
Exemple de Kruskal:



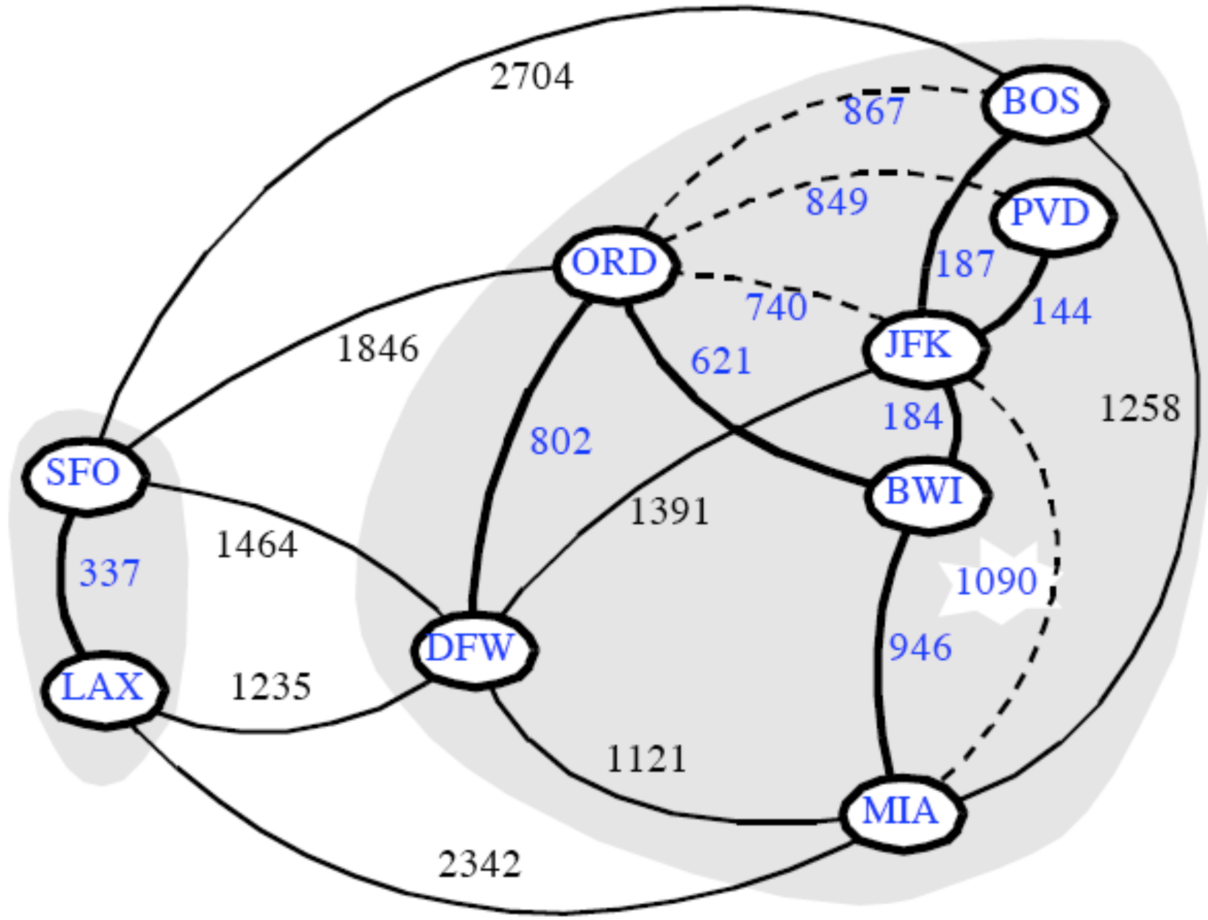
Exemple de Kruskal:



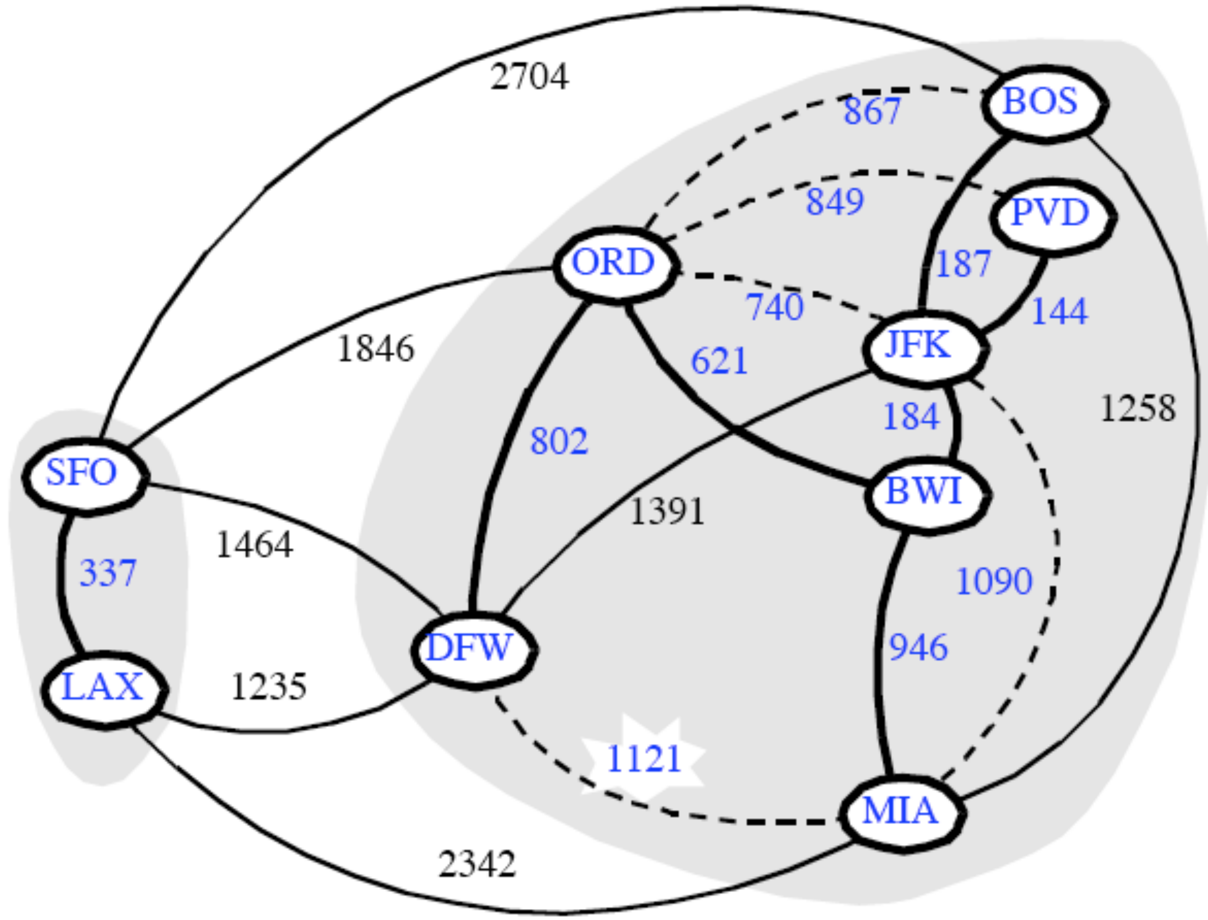
Exemple de Kruskal:



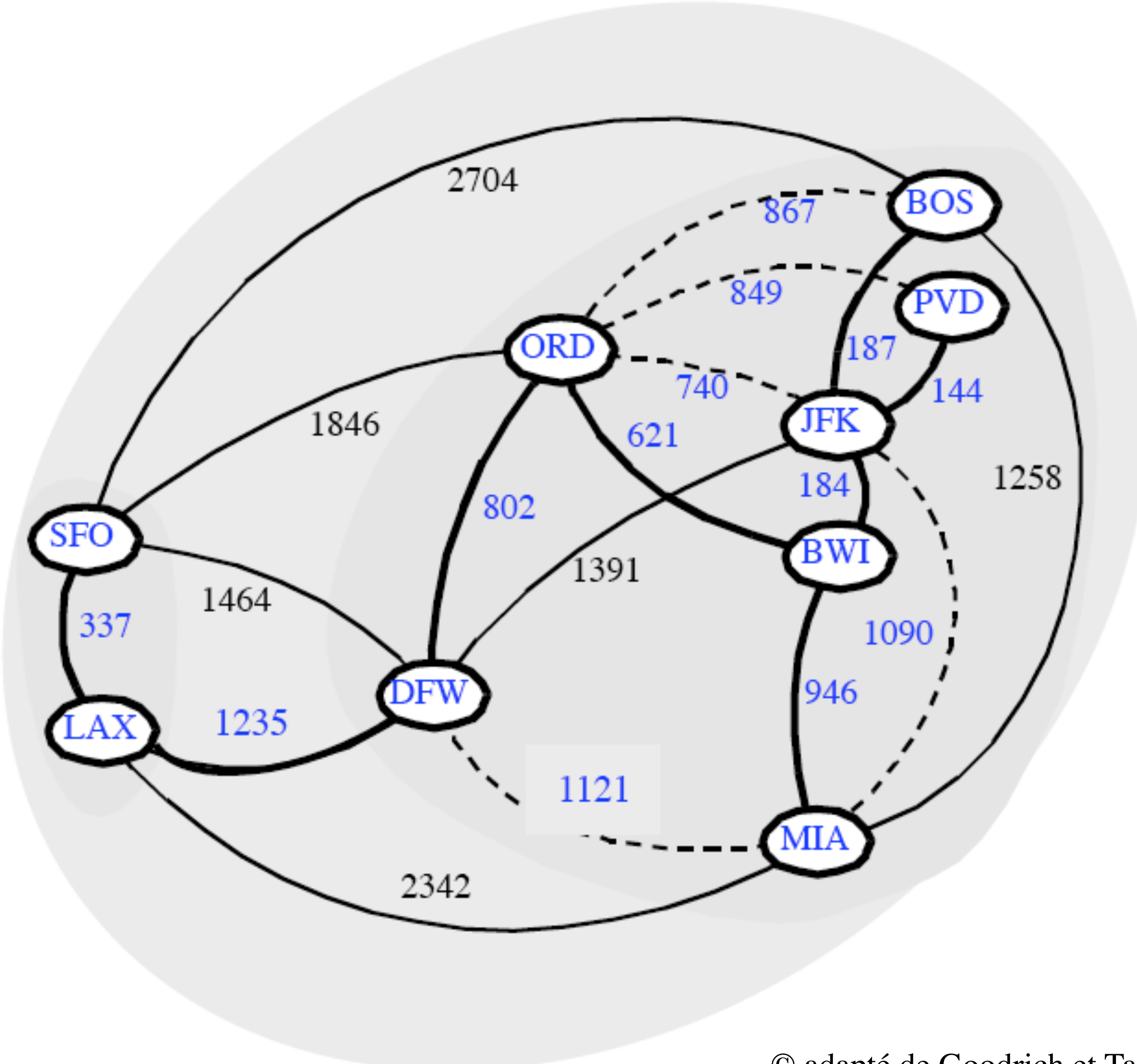
Exemple de Kruskal:



Exemple de Kruskal:



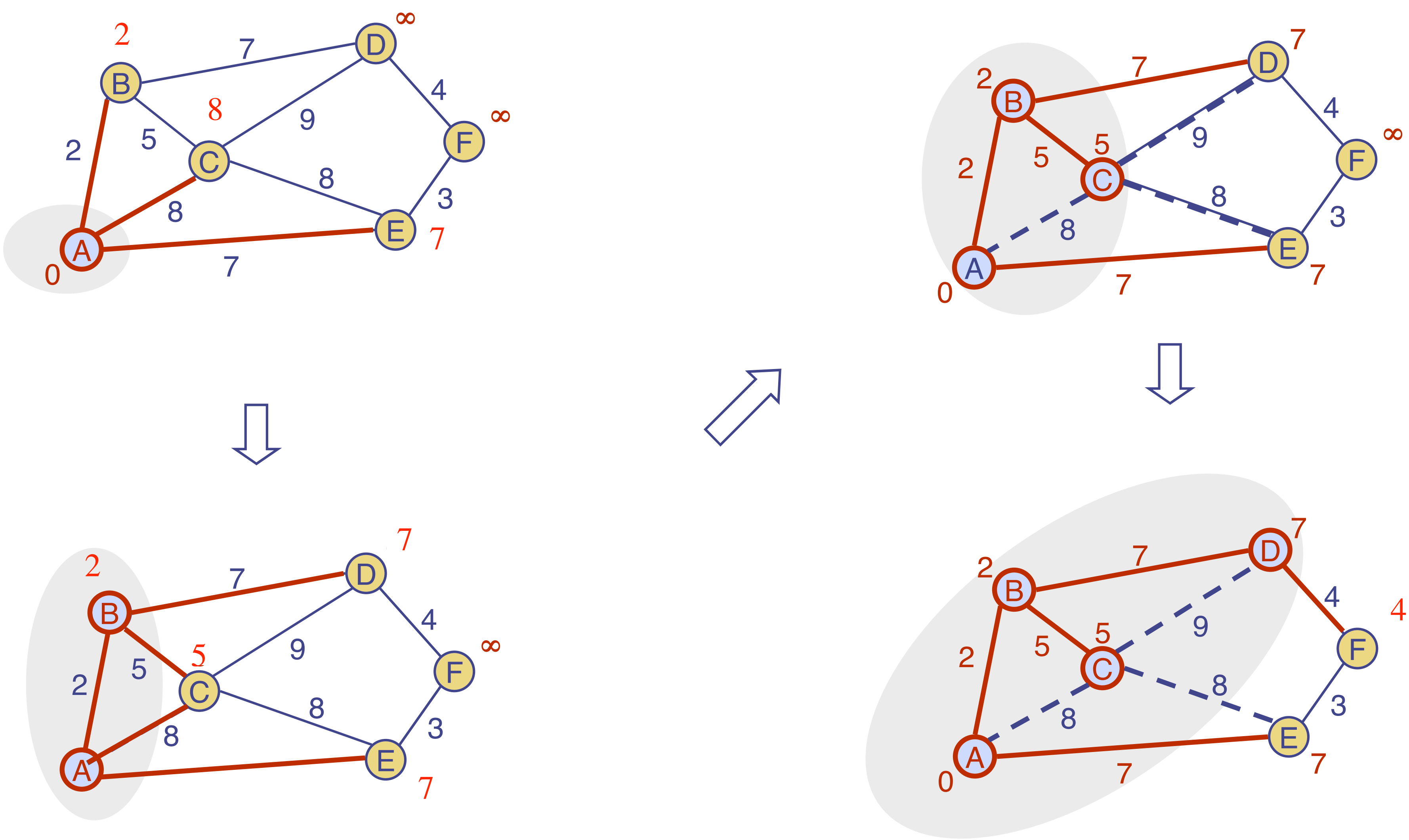
Exemple de Kruskal:



Algorithme de Prim

- On choisit un sommet s aléatoirement qu'on met dans un "nuage" et on construit l'arbre couvrant minimal en faisant grossir le "nuage" d'un sommet à la fois.
- On garde en mémoire à chaque sommet v, une étiquette d(v) qui ici est égale au poids minimal parmi les poids des arêtes reliant v à un sommet à l'intérieur du nuage.
- À chaque étape:
  - On ajoute au nuage le sommet u extérieur ayant la plus petite étiquette d(u)
  - On met à jour les étiquettes des sommets adjacents à u

Exemple:


2 7 D ∞
B 8 4 9
5 F ∞
2 C 8 3
8
A E 7
0 7
2 7 D 7
B 4
2 5 9 F ∞
C 8
5 8 3
8
A E 7
0 7
7 D 7
2 4
B 9
5 F ∞
2 5 C 8
2 8 3
A E 7
0 7
2 7 D 7
B 4 4
5 9
5 C F
2 8
8 3
A E 7
0 7

Exemple (suite)