Arbres couvrant de poids minimum

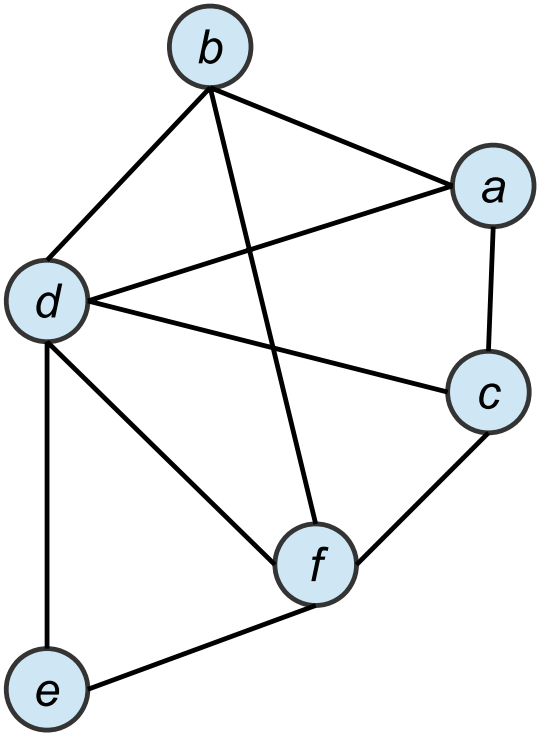
Nadia Brauner et Yann Kieffer

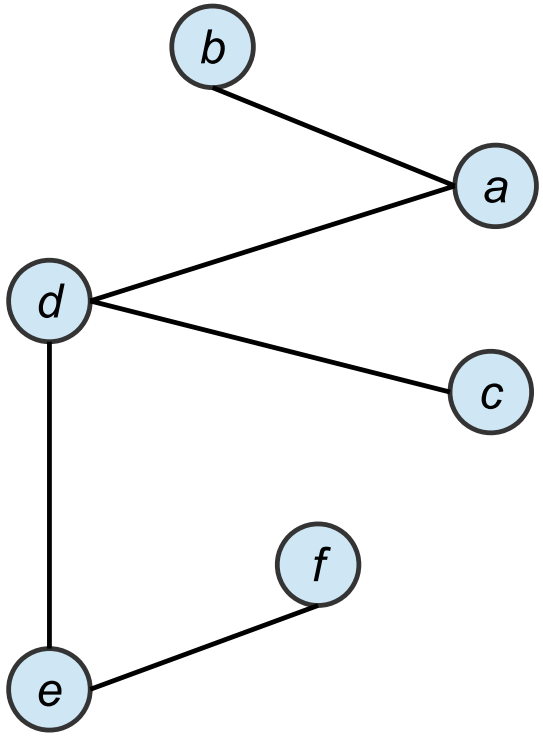
Nadia.Brauner@imag.fr

Arbres couvrants

T est un **arbre couvrant** de G si [*spanning tree*]

- $V(T) = V(G)$ et
- $E(T) \subset E(G)$ et
- T est un arbre.





Arbres couvrants

- Donnez une condition nécessaire et suffisante pour qu'un graphe G admette un arbre couvrant.
- Dans un graphe G d'ordre n, une chaîne élémentaire de longueur $n-1$ est-elle un arbre couvrant ?
- Dans un graphe G d'ordre n, un arbre avec $n-1$ arêtes est-il forcément couvrant ?

Arbres couvrants

Tout graphe connexe contient un arbre couvrant

Algorithme 1 : Arbre couvrant en déconstruisant

Données : $G = (V, E)$ connexe
Résultat : $G' = (V, F)$ un arbre couvrant de G

$F = E$
tant que $G' = (V, F)$ *contient un cycle* **faire**
    soit C un cycle de G' et soit e une arête de C
    $F \leftarrow F \setminus \{e\}$
retourner $G'=(V,F)$

Arbres couvrants

Montrer que cet algorithme renvoie bien un arbre couvrant de G.

1. *tout s'exécute correctement*
2. *en un nombre fini d'étapes*
   - A chaque étape, la cardinalité de F diminue
3. *en cas d'arrêt, on obtient l'objet souhaité*
   - l'instruction dans le **tant que** ne déconnecte pas le graphe
   - A la sortie du **tant que**, le graphe est sans cycle

Arbres couvrants

Tout graphe connexe contient un arbre couvrant

Algorithme 2 : Arbre couvrant en construisant

Données : $G = (V, E)$ connexe
Résultat : $G' = (V, F)$ un arbre couvrant de G
$F = \emptyset$
tant que $G' = (V, F)$ *n'est pas connexe* **faire**
    soit e une arête de E qui relie deux composantes connexes de G'
    $F \leftarrow F \cup \{e\}$
retourner $G'=(V,F)$

Arbres couvrants

Montrer que cet algorithme renvoie bien un arbre couvrant de G.

1. *tout s'exécute correctement*
2. *en un nombre fini d'étapes*
   - A chaque étape, la cardinalité de F augmente et $F \subseteq E$
3. *en cas d'arrêt, on obtient l'objet souhaité*
   - l'instruction dans le **tant que** ne crée pas de cycle
   - A la sortie du **tant que**, le graphe est connexe

Arbres couvrants de poids minimum

Un **graphe pondéré** $G = (V, E, w)$ est un graphe $G = (V, E)$ muni d'une fonction de poids sur les arêtes : $w : E \to \mathbb{R}^+$

[weighed graph]

Arbres couvrants de poids minimum

Arbre couvrant : les arêtes de l'arbre sont des arêtes de G. Les sommets de l'arbres sont exactement les sommets de G.

Poids d'un arbre = somme des poids de ses arêtes

Le problème

Soit $G = (V, E, w)$ un graphe pondéré. Trouver un arbre couvrant de G de poids minimum.

[*Minimum Spanning Tree* (MST)]

Arbres couvrants de poids minimum

Applications

- Relier les composants sur un circuit électronique pour les mettre au même potentiel (minimiser la longueur totale des fils utilisé)
- Création d'un Réseau d'interconnexion électrique entre villes

Arbres couvrants de poids minimum

Algorithme glouton : fait le meilleur choix au moment où il le fait (on ne revient pas sur un choix)

On va construire l'arbre couvrant petit à petit, en s'assurant à chaque étape qu'il reste

- couvrant sans cycle (algorithme de **Kruskal**)
- connexe sans cycle (algorithme de **Prim**)

Arbres couvrants de poids minimum

Augmenter un MST

Méthode générique qui maintient la propriété : l'ensemble A d'arêtes est un sous-ensemble d'un MST

A chaque itération, on ajoute une arête e à A qui maintient la propriété

Arbres couvrants de poids minimum

Comment trouver une telle arête ?

- **coupe** S : partition de V en $(S, V \setminus S)$
- coupe S **respecte** ensemble d'arêtes A si aucune arête de A n'appartient au co-cycle de S
- e est une **arête légère** qui traverse une coupe S si
  - e appartient au co-cycle de S et
  - e est de plus petit poids parmi les arêtes du co-cycle de S

Soit A un sous-ensemble de E inclus dans un MST de G et soit $(S, V \setminus S)$ une coupe qui respecte A et soit e une arête légère de cette coupe. Alors, $A \cup \{e\}$ est inclus dans un MST de G.

Arbres couvrants de poids minimum

Soit A un sous-ensemble de E inclus dans un MST de G et soit $(S, V \setminus S)$ une coupe qui respecte A et soit uv une arête légère de cette coupe. Alors, $A \cup \{uv\}$ est inclus dans un MST de G.

- Soit T un MST qui contient A.
- Si T ne contient pas uv alors $T \cup \{uv\}$ contient un cycle C
- Dans T, il y a un chemin de u à v donc C contient une arête $e' \neq uv$ qui appartient au co-cycle de S.
- uv est une arête légère qui traverse S donc $w(uv) \leq w(e')$
- Soit $T' = T \cup \{uv\} \setminus \{e'\}$.
- On a $w(T') = w(T) - w(e') + w(uv) \leq w(T)$.
- Comme T est un MST, T' est aussi un MST.
- Donc T' est un MST qui contient A et uv.

Arbres couvrants de poids minimum

Algorithme 3 : MST générique

Données : $G = (V, E, w)$ connexe
Résultat : $G_A = (V, A)$ un MST de G

$A = \emptyset$
tant que $G_A = (V, A)$ *n'est pas connexe* **faire**
    soit S une coupe qui respecte A
    soit e une arête légère qui traverse S
    $A \leftarrow A \cup \{e\}$
retourner $G_A = (V, A)$

Arbres couvrants de poids minimum

1. *tout s'exécute correctement*
   - Dans la boucle, A n'est pas connexe, donc S existe
   - G connexe donc e existe
2. *en un nombre fini d'étapes*
   - A chaque étape, la cardinalité de A augmente et $A \subseteq E$
3. *en cas d'arrêt, on obtient l'objet souhaité*
   - Dans la boucle A reste inclus dans un MST (propriété)
   - Donc à la sortie du **tant que**, G_A est connexe, couvrant et inclus dans un MST
   - Donc G_A est un MST

Et si G n'est pas connexe ?

Arbres couvrants de poids minimum

Les algorithmes

- Kruskal
  - graphe $G_A = (V, A)$ couvrant sans cycle,
  - arête valide = arête de plus petit poids qui connecte deux composante connexes de G_A
- Prim
  - A connexe sans cycle
  - arête valide = arête la plus légère entre les sommets couverts par A et les sommets non couverts par A

Algorithme de Kruskal

Algorithme 4 : Algorithme de Kruskal

Données : $G = (V, E, w)$

Résultat : $T = (V, F)$ un MST de G

trier les arêtes de E par poids croissants : $w(e_1) \leq w(e_2) ... w(e_m)$

$F = \emptyset$

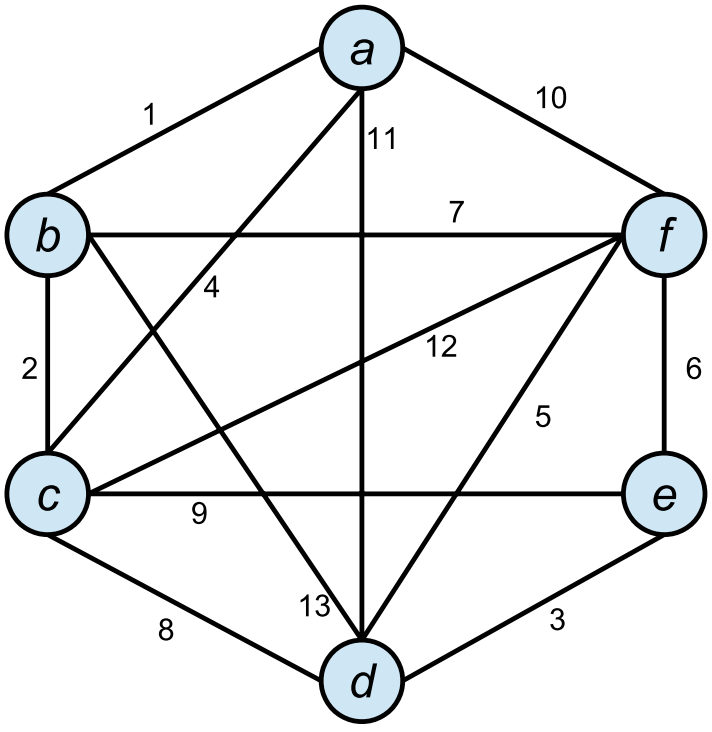
pour $i = 1$ *à* $|E|$ **faire**

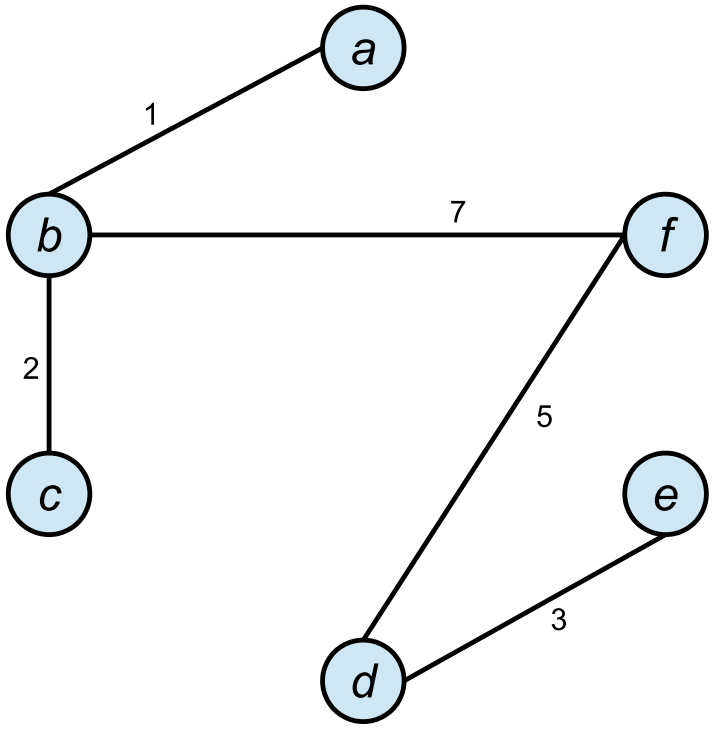
  Si l'ajout de e_i à F ne crée pas de cycle alors

  $F \leftarrow F \cup \{e_i\}$

retourner $T = (V, F)$

Arbres couvrant





Algorithme de Kruskal

Structure de donnée pour gérer les composantes connexe d'un graphe : **Union-Find**

Permet de gérer les partitions d'un ensemble

- construire une partition initiale sur un ensemble d'éléments
- fusionner (*unir*) deux classes de la partition
- savoir si deux éléments sont dans la même classe

Algorithme de Kruskal

Union-Find

Pour cela, il faut choisir un représentant de chaque classe qui permet d'identifier la classe entière.

Les services

- Construire une partition qui pour chaque élément x crée la classe $\{x\}$.
- *find*(x) qui renvoie le représentant de la classe contenant x.
- *union*(x, y) qui fusionne les classes contenant x et y.
  Les paramètres x et y doivent être dans des classes différentes.

Algorithme de Kruskal

Structure **Union Find**

- stocker chaque classe comme un arbre enraciné dans lequel chaque nœud contient une référence vers son nœud parent.
- Le représentant de chaque classe est alors le nœud racine de l'arbre correspondant.
- la racine est le seul nœud qui pointe sur lui même

Algorithme de Kruskal

Complexité de **Union Find**

- *Construire* : complexité n
- *find*(x) suit le chemin de x jusqu'à la racine : complexité profondeur de x
- *union*(x, y) : la racine de l'un devient le parent de la racine de l'autre : complexité max des profondeurs de x et y

Donc la complexité dépend de la hauteur de des arbres

idée : maîtriser la hauteur en utilisant la fonction *rank*

Algorithme de Kruskal

Union Find

Algorithme 5 : *construire*(S)

pour *tous les éléments* x *de* S **faire**
    $parent(x) = x$
    $rank(x) = 0$

Algorithme 6 : *find*(x)

tant que $x \neq parent(x)$ **faire**
    $x \leftarrow parent(x)$
retourner (x)

Algorithme de Kruskal

Algorithme 7 : $union(x, y)$

$r_x \leftarrow find(x)$
$r_y \leftarrow find(y)$
si $rank(r_x) > rank(r_y)$ **alors**
  $parent(r_y) \leftarrow r_x$
sinon
  $parent(r_x) \leftarrow r_y$
  si $rank(r_x) = rank(r_y)$ **alors**
    $rank(r_y) = rank(r_x) + 1$

Algorithme de Kruskal

- $rank(x)$ est la hauteur de la sous-arborescence de racine x
- donc $rank(x) < rank(parent(x))$
- si $rank(x) = k$ alors le sous-arbre de racine x a au moins 2^k sommets
- donc $rank(x) <= log_2 n$
- donc *union* et *find* de complexité $\log(n)$

Algorithme de Kruskal

Algorithme 8 : Algorithme de Kruskal avec union-find

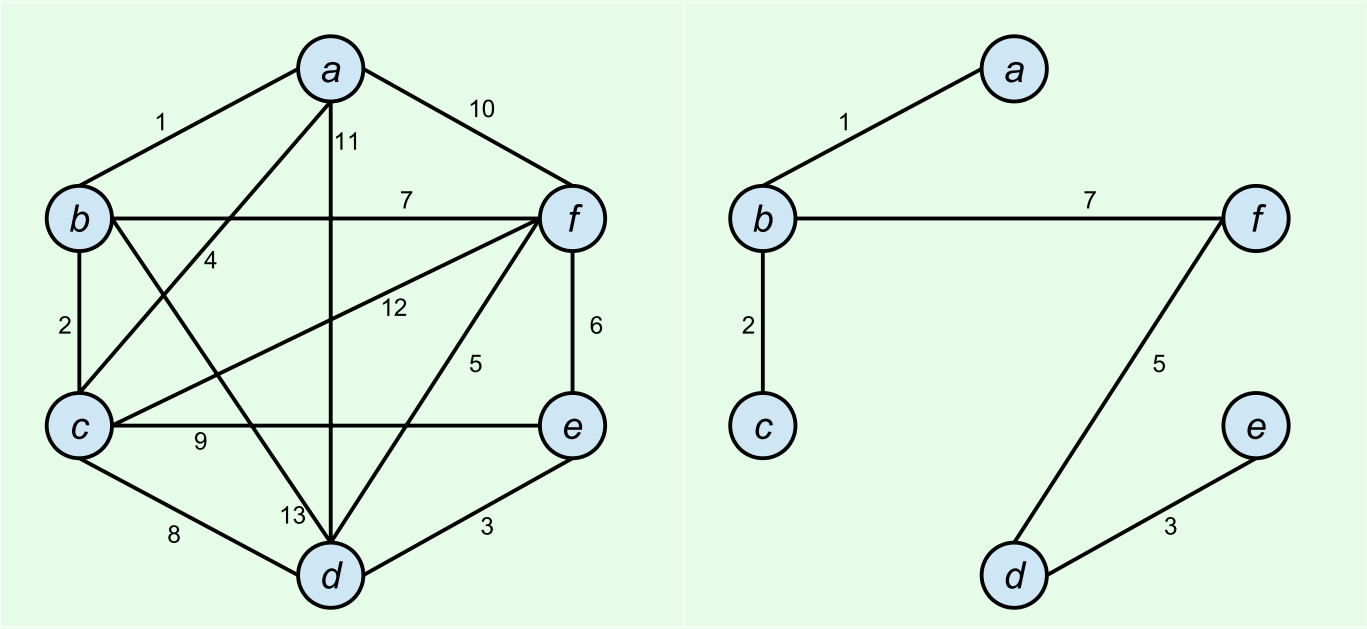
Données : $G = (V, E, w)$
Résultat : $T = (V, F)$ un MST de G

trier les arêtes de E par poids croissants : $w(x_1y_1) \leq w(x_2y_2)...$
$F = \emptyset$
Construire une partition sur V
pour $i = 1$ *à* $|E|$ **faire**
    Si $find(x_i) \neq find(y_i)$
    $F \leftarrow F \cup \{x_iy_i\}$
    $union(x_i, y_i)$
retourner $T = (V, F)$

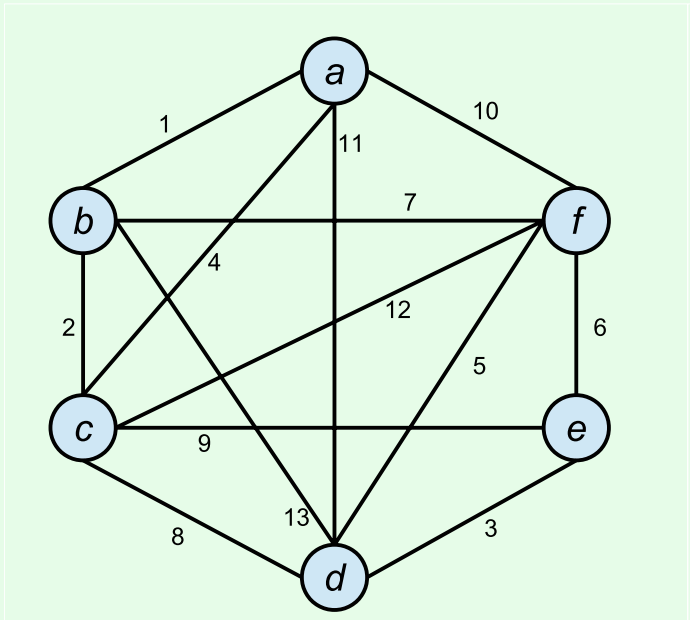
donc Kruskal dominé par le tri des arêtes $O(m \log m)$

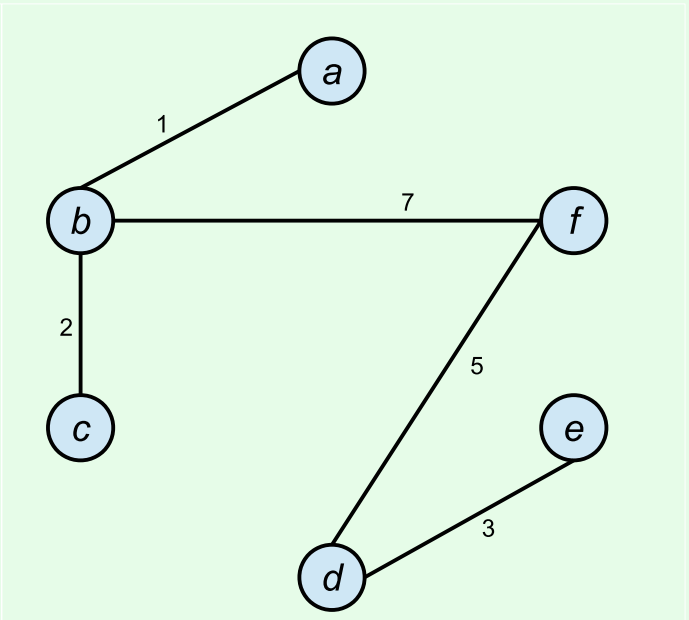
Algorithme de Kruskal

- Montrer que pour tout arbre couvrant T, on peut ordonner les arêtes de G tel que l'algorithme de Kruskal renvoie T
- Sur le graphe G suivant, quelle est la plus petite valeur entière qu'il faut donner à l'arête ab pour qu'elle ne soit obligatoirement plus dans l'arbre couvrant de poids minimum ?



Arbres couvrants de poids minimum





- Comment faites-vous pour trouver cette valeur à partir de l'arbre couvrant T initial ?
- Sur ce même graphe, quelle est la plus grande valeur entière qu'il faut donner à l'arête *ce* pour qu'elle soit obligatoirement dans l'arbre couvrant de poids minimum.

Arbres couvrants de poids minimum

- Que se passe-t-il si le graphe est non connexe pour les deux algos ?
- Que se passe-t-il si on a des poids négatifs (arbre couvrant ou graphe couvrant connexe)

Extension

Arbres de Steiner

Bientôt sur vos écrans...