M1 Info - Graphes et programmation dynamique

Cours 4 - Arbre couvrant de poids minimum

Prim et Kruskal

Semestre d'Automne 2022-2023 - Université Côte D'azur

Christophe Crespelle
christophe.crespelle@univ-cotedazur.fr

Arbre couvrant de poids minimum

<u>Cadre :</u> graphes simples $G = (V, E)$ (sans boucles, sans aretes multiples)

- non orientes
- ponderes : $w : E \mapsto \mathbb{R}$

Arbre couvrant de poids minimum

Cadre : graphes simples $G = (V, E)$ (sans boucles, sans aretes multiples)

- non orientes
- ponderes : $w : E \mapsto \mathbb{R}$

Definitions :

Un **arbre couvrant** d'un graphe $G = (V, E)$ non oriente est un arbre $T = (V, E_T)$ avec $E_T \subseteq E$.

Arbre couvrant de poids minimum

Cadre : graphes simples $G = (V, E)$ (sans boucles, sans aretes multiples)

- non orientes
- ponderes : $w : E \mapsto \mathbb{R}$

Definitions :

Un **arbre couvrant** d'un graphe $G = (V, E)$ non oriente est un arbre $T = (V, E_T)$ avec $E_T \subseteq E$.

Dans un graphe pondere par une fonction de poids reelle w, le **poids** $w(T)$ d'un arbre couvrant T est la somme des poids de ses aretes : $w(T) = \sum_{e \in E_T} w(e)$.

Arbre couvrant de poids minimum

<u>**Cadre :**</u> graphes simples $G = (V, E)$ (sans boucles, sans aretes multiples)

- non orientes
- ponderes : $w : E \mapsto \mathbb{R}$

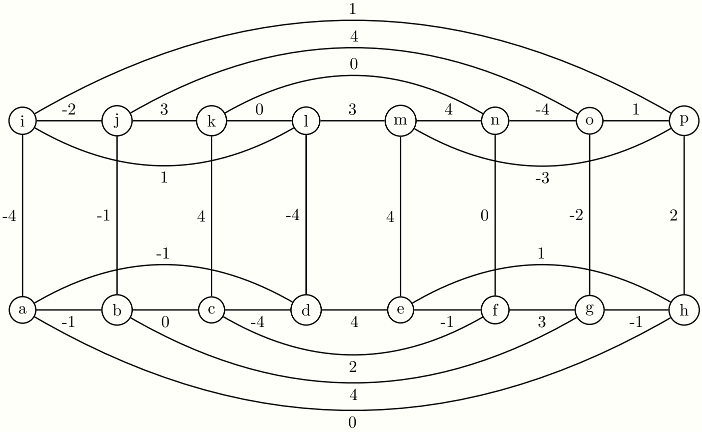
<u>**Definitions :**</u>

Un **arbre couvrant** d'un graphe $G = (V, E)$ non oriente est un arbre $T = (V, E_T)$ avec $E_T \subseteq E$.

Dans un graphe pondere par une fonction de poids reelle w, le **poids** $w(T)$ d'un arbre couvrant T est la somme des poids de ses aretes : $w(T) = \sum_{e \in E_T} w(e)$.

<u>Observation :</u> un graphe G admet un arbre couvrant ssi G est connexe.

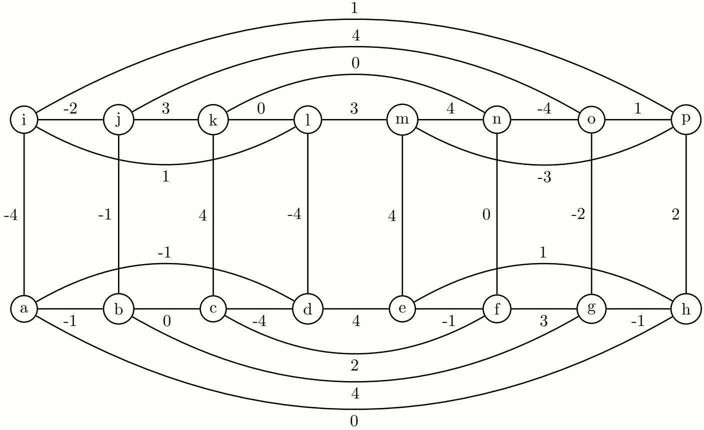
Exemple



Arbre couvrant de poids minimum

<u>Le probleme</u>

- **Entrée :** un graphe $G = (V, E)$ non oriente et pondere
- **Sortie :** un arbre couvrant T de G de poids minimum.



Quelques variations du probleme

- de poids reels a poids positifs

Quelques variations du probleme

- de poids reels a poids positifs
  - si poids negatifs, ajouter $-w_{min}$ a toutes les aretes

Quelques variations du probleme

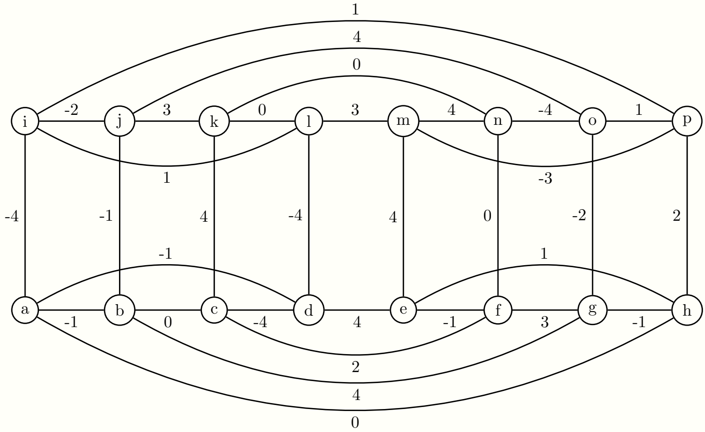
- de poids reels a poids positifs
  - si poids negatifs, ajouter $-w_{min}$ a toutes les aretes
  - $\tilde{w}(T) = w(T) - (n-1) \cdot w_{min}$

Quelques variations du probleme

- de poids reels a poids positifs
  - si poids negatifs, ajouter $-w_{min}$ a toutes les aretes
  - $\tilde{w}(T) = w(T) - (n-1) \cdot w_{min}$
  - $argmin\{\tilde{w}(T)\} = argmin\{w(T)\}$

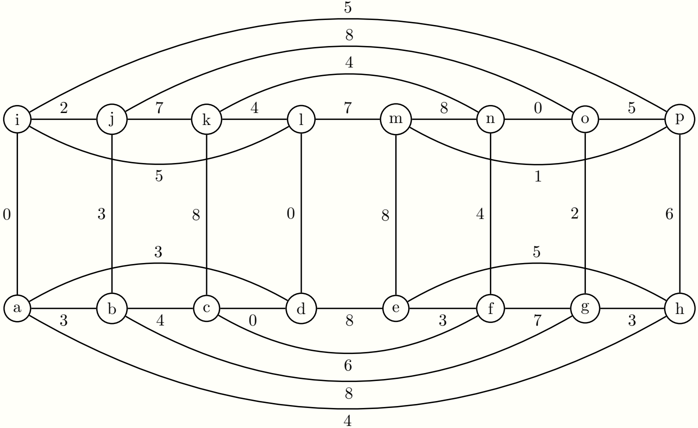
Quelques variations du probleme

- de poids reels a poids positifs
  - si poids negatifs, ajouter $-w_{min}$ a toutes les aretes
  - $\tilde{w}(T) = w(T) - (n-1) \cdot w_{min}$
  - $argmin\{\tilde{w}(T)\} = argmin\{w(T)\}$



Quelques variations du probleme

- de poids reels a poids positifs
  - si poids negatifs, ajouter $-w_{min}$ a toutes les aretes
  - $\tilde{w}(T) = w(T) - (n-1) \cdot w_{min}$
  - $argmin\{\tilde{w}(T)\} = argmin\{w(T)\}$



Quelques variations du probleme

- de poids reels a poids positifs :
- de poids maximum a poids minimum

Quelques variations du probleme

- de poids reels a poids positifs :
- de poids maximum a poids minimum
  - prendre oppose du poids sur toutes les aretes

Quelques variations du probleme

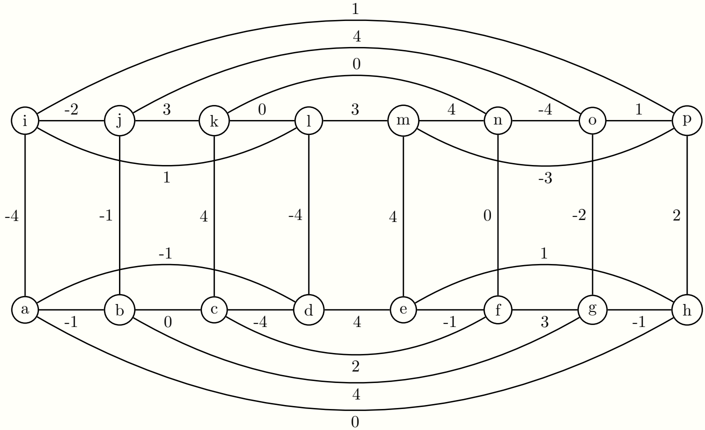
- de poids reels a poids positifs :
- de poids maximum a poids minimum
  - prendre oppose du poids sur toutes les aretes
  - $w^{-}(T) = -w(T)$

Quelques variations du probleme

- de poids reels a poids positifs :
- de poids maximum a poids minimum
  - prendre oppose du poids sur toutes les aretes
  - $w^{-}(T) = -w(T)$
  - $argmax\{w(T)\} = argmin\{w^{-}(T)\}$

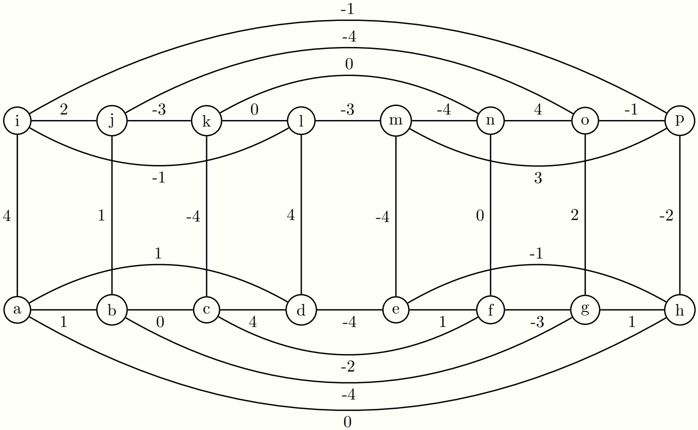
Quelques variations du probleme

- de poids reels a poids positifs :
- de poids maximum a poids minimum
  - prendre oppose du poids sur toutes les aretes
  - $w^{-}(T) = -w(T)$
  - $argmax\{w(T)\} = argmin\{w^{-}(T)\}$



Quelques variations du probleme

- de poids reels a poids positifs :
- de poids maximum a poids minimum
  - prendre oppose du poids sur toutes les aretes
  - $w^{-}(T) = -w(T)$
  - $argmax\{w(T)\} = argmin\{w^{-}(T)\}$



Quelques variations du probleme

Conclusion

- poids reels ou poids positifs
- arbre de poids maximum ou arbre de poids minimum

Quelques variations du probleme

Conclusion

- poids reels ou poids positifs
- arbre de poids maximum ou arbre de poids minimum
- tous ces problemes se ramenent a :

$$\boxed{\textbf{Arbre couvrant de poids } \underline{\textbf{minimum}} \textbf{ avec } \underline{\textbf{poids positifs}}}$$

Une propriete fondamentale des arbres couvrants

Lemme 1 : Soit G un graphe et T un arbre couvrant de G. Soit une arete uv de G qui n'est pas dans T : $uv \in E(G) \setminus E(T)$. Soit e' une arete sur le chemin (unique) de u a v dans T. Alors l'arbre $T' = (T \setminus \{e'\}) \cup \{uv\}$ est aussi un arbre couvrant de G.

Une propriete fondamentale des arbres couvrants

Lemme 1 : Soit G un graphe et T un arbre couvrant de G. Soit une arete uv de G qui n'est pas dans T : $uv \in E(G) \setminus E(T)$. Soit e' une arete sur le chemin (unique) de u a v dans T. Alors l'arbre $T' = (T \setminus \{e'\}) \cup \{uv\}$ est aussi un arbre couvrant de G.

Démonstration.

Soient $x, y \in V^2$ et soit C le chemin de x a y dans T.

$\square$

Une propriete fondamentale des arbres couvrants

<u>**Lemme 1 :**</u> Soit G un graphe et T un arbre couvrant de G. Soit une arete uv de G qui n'est pas dans T : $uv \in E(G) \setminus E(T)$. Soit e' une arete sur le chemin (unique) de u a v dans T. Alors l'arbre $T' = (T \setminus \{e'\}) \cup \{uv\}$ est aussi un arbre couvrant de G.

Démonstration.

Soient $x, y \in V^2$ et soit C le chemin de x a y dans T.

- Si $e' \notin E(C)$, alors C est aussi un chemin de x a y dans T'.

$\square$

Une propriete fondamentale des arbres couvrants

Lemme 1 : Soit G un graphe et T un arbre couvrant de G. Soit une arete uv de G qui n'est pas dans T : $uv \in E(G) \setminus E(T)$. Soit e' une arete sur le chemin (unique) de u a v dans T. Alors l'arbre $T' = (T \setminus \{e'\}) \cup \{uv\}$ est aussi un arbre couvrant de G.

Démonstration.

Soient $x, y \in V^2$ et soit C le chemin de x a y dans T.

- Si $e' \notin E(C)$, alors C est aussi un chemin de x a y dans T'.
- Si $e' \in E(C)$,
  - alors $C = C_1, a, b, C_2$ avec $ab = e'$ et avec $C1$ et C_2 qui sont aussi des chemins dans T' (car $e' \notin C_1$ et $e' \notin C_2$).

$\square$

Une propriete fondamentale des arbres couvrants

<u>Lemme 1 :</u> Soit G un graphe et T un arbre couvrant de G. Soit une arete uv de G qui n'est pas dans T : $uv \in E(G) \setminus E(T)$. Soit e' une arete sur le chemin (unique) de u a v dans T. Alors l'arbre $T' = (T \setminus \{e'\}) \cup \{uv\}$ est aussi un arbre couvrant de G.

Démonstration.

Soient $x, y \in V^2$ et soit C le chemin de x a y dans T.

- Si $e' \notin E(C)$, alors C est aussi un chemin de x a y dans T'.
- Si $e' \in E(C)$,
  - alors $C = C_1, a, b, C_2$ avec $ab = e'$ et avec $C1$ et C_2 qui sont aussi des chemins dans T' (car $e' \notin C_1$ et $e' \notin C_2$).
  - Soit C_{uv} le chemin de u a v dans T, on a $C_{uv} = A, a, b, B$, avec A et B qui sont aussi des chemins dans T'.

$\square$

Une propriete fondamentale des arbres couvrants

Lemme 1 : Soit G un graphe et T un arbre couvrant de G. Soit une arete uv de G qui n'est pas dans T : $uv \in E(G) \setminus E(T)$. Soit e' une arete sur le chemin (unique) de u a v dans T. Alors l'arbre $T' = (T \setminus \{e'\}) \cup \{uv\}$ est aussi un arbre couvrant de G.

Démonstration.

Soient $x, y \in V^2$ et soit C le chemin de x a y dans T.

- Si $e' \notin E(C)$, alors C est aussi un chemin de x a y dans T'.
- Si $e' \in E(C)$,
  - alors $C = C_1, a, b, C_2$ avec $ab = e'$ et avec $C1$ et C_2 qui sont aussi des chemins dans T' (car $e' \notin C_1$ et $e' \notin C_2$).
  - Soit C_{uv} le chemin de u a v dans T, on a $C_{uv} = A, a, b, B$, avec A et B qui sont aussi des chemins dans T'.
  - D'ou $P_{ba} = B, v, u, A$ est un chemin de b a a dans T'

$\square$

Une propriete fondamentale des arbres couvrants

Lemme 1 : Soit G un graphe et T un arbre couvrant de G. Soit une arete uv de G qui n'est pas dans T : $uv \in E(G) \setminus E(T)$. Soit e' une arete sur le chemin (unique) de u a v dans T. Alors l'arbre $T' = (T \setminus \{e'\}) \cup \{uv\}$ est aussi un arbre couvrant de G.

Démonstration.

Soient $x, y \in V^2$ et soit C le chemin de x a y dans T.

- Si $e' \notin E(C)$, alors C est aussi un chemin de x a y dans T'.
- Si $e' \in E(C)$,
  - alors $C = C_1, a, b, C_2$ avec $ab = e'$ et avec $C1$ et C_2 qui sont aussi des chemins dans T' (car $e' \notin C_1$ et $e' \notin C_2$).
  - Soit C_{uv} le chemin de u a v dans T, on a $C_{uv} = A, a, b, B$, avec A et B qui sont aussi des chemins dans T'.
  - D'ou $P_{ba} = B, v, u, A$ est un chemin de b a a dans T'
  - Et on a donc un chemin C_1, P^r_{ba}, C_2 de x a y dans T', ou P^r_{ba} est le chemin inverse de P_{ba}.

□

Une propriete fondamentale des arbres couvrants

Lemme 1 : Soit G un graphe et T un arbre couvrant de G. Soit une arete uv de G qui n'est pas dans T : $uv \in E(G) \setminus E(T)$. Soit e' une arete sur le chemin (unique) de u a v dans T. Alors l'arbre $T' = (T \setminus \{e'\}) \cup \{uv\}$ est aussi un arbre couvrant de G.

Démonstration.

Soient $x, y \in V^2$ et soit C le chemin de x a y dans T.

- Si $e' \notin E(C)$, alors C est aussi un chemin de x a y dans T'.
- Si $e' \in E(C)$,
  - alors $C = C_1, a, b, C_2$ avec $ab = e'$ et avec $C1$ et C_2 qui sont aussi des chemins dans T' (car $e' \notin C_1$ et $e' \notin C_2$).
  - Soit C_{uv} le chemin de u a v dans T, on a $C_{uv} = A, a, b, B$, avec A et B qui sont aussi des chemins dans T'.
  - D'ou $P_{ba} = B, v, u, A$ est un chemin de b a a dans T'
  - Et on a donc un chemin C_1, P^r_{ba}, C_2 de x a y dans T', ou P^r_{ba} est le chemin inverse de P_{ba}.

$\Longrightarrow$ T' est connexe et a le meme nombre d'aretes que T. $\square$

Faire croitre une foret (de maniere sure)

Principe d'un algorithme glouton

- Construit une solution incrementalement

Faire croitre une foret (de maniere sure)

Principe d'un algorithme glouton

- Construit une solution incrementalement
- En maximisant un objectif a chaque etape

Faire croitre une foret (de maniere sure)

Principe d'un algorithme glouton

- Construit une solution incrementalement
- En maximisant un objectif a chaque etape
- Ne revient jamais sur ses choix

Faire croitre une foret (de maniere sure)

Principe d'un algorithme glouton

- Construit une solution incrementalement
- En maximisant un objectif a chaque etape
- Ne revient jamais sur ses choix

Definitions :

(Arete sure pour F) Si $F \subseteq E$ est inclus dans un arbre couvrant, on dit qu'une arete $e \in E \setminus F$ est **sure** lorsque $F \cup \{e\}$ est inclus dans un arbre couvrant.

Faire croitre une foret (de maniere sure)

Principe d'un algorithme glouton

- Construit une solution incrementalement
- En maximisant un objectif a chaque etape
- Ne revient jamais sur ses choix

Definitions :

(Arete sure pour F) Si $F \subseteq E$ est inclus dans un arbre couvrant, on dit qu'une arete $e \in E \setminus F$ est **sure** lorsque $F \cup \{e\}$ est inclus dans un arbre couvrant.

(Coupe) Une coupe d'un graphe G est une bipartition $(A, V \setminus A)$ de ses sommets.

Faire croitre une foret (de maniere sure)

Principe d'un algorithme glouton

- Construit une solution incrementalement
- En maximisant un objectif a chaque etape
- Ne revient jamais sur ses choix

Definitions :

(Arete sure pour F) Si $F \subseteq E$ est inclus dans un arbre couvrant, on dit qu'une arete $e \in E \setminus F$ est **sure** lorsque $F \cup \{e\}$ est inclus dans un arbre couvrant.

(Coupe) Une coupe d'un graphe G est une bipartition $(A, V \setminus A)$ de ses sommets.

(Coupe respectant un sous-ensemble d'aretes F) Une coupe (X, Y) de G respecte un sous-ensemble d'aretes $F \subseteq E$ ssi il n'existe aucune arete de F traversant (X, Y) :
$\{uv \mid u \in X \text{ et } v \in Y \text{ et } xy \in F\} = \varnothing$.

Faire croitre une foret (de maniere sure)

Lemme 2 : Soit G un graphe pondere par w. Soit $F \subseteq E$ un sous-ensemble d'aretes inclus dans un arbre couvrant de G et soit une coupe (X, Y) respectant F. Si e est une arete de poids minimum parmi celles traversant (X, Y) alors e est sure pour F.

Faire croitre une foret (de maniere sure)

Lemme 2 : Soit G un graphe pondere par w. Soit $F \subseteq E$ un sous-ensemble d'aretes inclus dans un arbre couvrant de G et soit une coupe (X, Y) respectant F. Si e est une arete de poids minimum parmi celles traversant (X, Y) alors e est sure pour F.

Démonstration.

- Soit $e = xy$, avec $x \in X$ et $y \in Y$.

□

Faire croitre une foret (de maniere sure)

Lemme 2 : Soit G un graphe pondere par w. Soit $F \subseteq E$ un sous-ensemble d'aretes inclus dans un arbre couvrant de G et soit une coupe (X, Y) respectant F. Si e est une arete de poids minimum parmi celles traversant (X, Y) alors e est sure pour F.

Démonstration.

- Soit $e = xy$, avec $x \in X$ et $y \in Y$.
- Soit T un arbre couvrant de G qui contient F et pas e.

□

Faire croitre une foret (de maniere sure)

Lemme 2 : Soit G un graphe pondere par w. Soit $F \subseteq E$ un sous-ensemble d'aretes inclus dans un arbre couvrant de G et soit une coupe (X, Y) respectant F. Si e est une arete de poids minimum parmi celles traversant (X, Y) alors e est sure pour F.

Démonstration.

- Soit $e = xy$, avec $x \in X$ et $y \in Y$.
- Soit T un arbre couvrant de G qui contient F et pas e.
- On construit un arbre couvrant T' de G qui contient $F \cup \{e\}$.

$\square$

Faire croitre une foret (de maniere sure)

Lemme 2 : Soit G un graphe pondere par w. Soit $F \subseteq E$ un sous-ensemble d'aretes inclus dans un arbre couvrant de G et soit une coupe (X, Y) respectant F. Si e est une arete de poids minimum parmi celles traversant (X, Y) alors e est sure pour F.

Démonstration.

- Soit $e = xy$, avec $x \in X$ et $y \in Y$.
- Soit T un arbre couvrant de G qui contient F et pas e.
- On construit un arbre couvrant T' de G qui contient $F \cup \{e\}$.
- Soit C l'unique chemin de x a y dans T.

□

Faire croitre une foret (de maniere sure)

Lemme 2 : Soit G un graphe pondere par w. Soit $F \subseteq E$ un sous-ensemble d'aretes inclus dans un arbre couvrant de G et soit une coupe (X, Y) respectant F. Si e est une arete de poids minimum parmi celles traversant (X, Y) alors e est sure pour F.

Démonstration.

- Soit $e = xy$, avec $x \in X$ et $y \in Y$.
- Soit T un arbre couvrant de G qui contient F et pas e.
- On construit un arbre couvrant T' de G qui contient $F \cup \{e\}$.
- Soit C l'unique chemin de x a y dans T.
- C contient une arete $x'y'$ avec $x' \in X$ et $y' \in Y$.

□

Faire croitre une foret (de maniere sure)

Lemme 2 : Soit G un graphe pondere par w. Soit $F \subseteq E$ un sous-ensemble d'aretes inclus dans un arbre couvrant de G et soit une coupe (X, Y) respectant F. Si e est une arete de poids minimum parmi celles traversant (X, Y) alors e est sure pour F.

Démonstration.

- Soit $e = xy$, avec $x \in X$ et $y \in Y$.
- Soit T un arbre couvrant de G qui contient F et pas e.
- On construit un arbre couvrant T' de G qui contient $F \cup \{e\}$.
- Soit C l'unique chemin de x a y dans T.
- C contient une arete $x'y'$ avec $x' \in X$ et $y' \in Y$.
- Or d'apres le Lemme 1, $T' = (T \setminus \{x'y'\}) \cup \{xy\}$ est un arbre couvrant de G.

□

Faire croitre une foret (de maniere sure)

Lemme 2 : Soit G un graphe pondere par w. Soit $F \subseteq E$ un sous-ensemble d'aretes inclus dans un arbre couvrant de G et soit une coupe (X, Y) respectant F. Si e est une arete de poids minimum parmi celles traversant (X, Y) alors e est sure pour F.

Démonstration.

- Soit $e = xy$, avec $x \in X$ et $y \in Y$.
- Soit T un arbre couvrant de G qui contient F et pas e.
- On construit un arbre couvrant T' de G qui contient $F \cup \{e\}$.
- Soit C l'unique chemin de x a y dans T.
- C contient une arete $x'y'$ avec $x' \in X$ et $y' \in Y$.
- Or d'apres le Lemme 1, $T' = (T \setminus \{x'y'\}) \cup \{xy\}$ est un arbre couvrant de G.
- Comme (X, Y) respecte F, $x'y' \notin F$ et donc $F \subseteq E(T')$.

$\square$

Faire croitre une foret (de maniere sure)

Lemme 2 : Soit G un graphe pondere par w. Soit $F \subseteq E$ un sous-ensemble d'aretes inclus dans un arbre couvrant de G et soit une coupe (X, Y) respectant F. Si e est une arete de poids minimum parmi celles traversant (X, Y) alors e est sure pour F.

Démonstration.

- Soit $e = xy$, avec $x \in X$ et $y \in Y$.
- Soit T un arbre couvrant de G qui contient F et pas e.
- On construit un arbre couvrant T' de G qui contient $F \cup \{e\}$.
- Soit C l'unique chemin de x a y dans T.
- C contient une arete $x'y'$ avec $x' \in X$ et $y' \in Y$.
- Or d'apres le Lemme 1, $T' = (T \setminus \{x'y'\}) \cup \{xy\}$ est un arbre couvrant de G.
- Comme (X, Y) respecte F, $x'y' \notin F$ et donc $F \subseteq E(T')$.
- Par definition de e, $w(xy) \leq w(x'y')$. D'ou $w(T') \leq w(T)$.

$\square$

Algorithme de Prim

Idee :

- On part d'un arbre T
  - ne contenant aucune arete : $A = \varnothing$
  - contenant un unique noeud r (sa racine) choisi arbitrairement

Algorithme de Prim

Idee :

- On part d'un arbre T
  - ne contenant aucune arete : $A = \varnothing$
  - contenant un unique noeud r (sa racine) choisi arbitrairement
- A tout moment, les aretes selectionnees A forment un arbre T enracine en r.
  - c.a.d. que le sous graphe forme par A est **connexe**
  - et acyclique ! (bien sur)

Algorithme de Prim

Idee :

- On part d'un arbre T
  - ne contenant aucune arete : $A = \varnothing$
  - contenant un unique noeud r (sa racine) choisi arbitrairement
- A tout moment, les aretes selectionnees A forment un arbre T enracine en r.
  - c.a.d. que le sous graphe forme par A est **connexe**
  - et acyclique ! (bien sur)
- On fait grossir T en ajoutant une arete e de poids minimum parmi celles traversant la coupe $(V(T), V \setminus V(T))$.
  - e est sure d'apres le Lemme 2.

Algorithme de Prim

Idee :

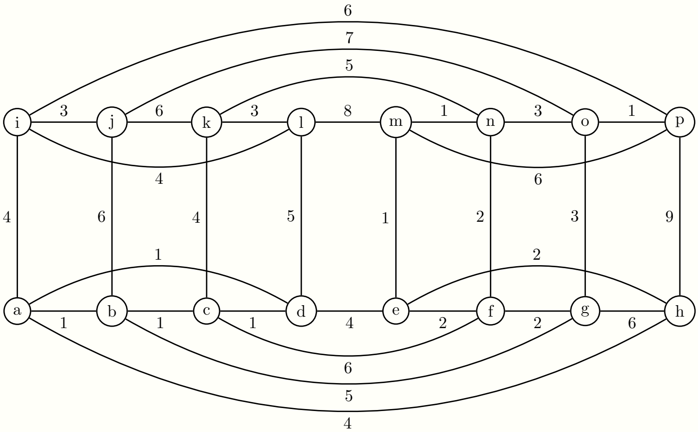
- On part d'un arbre T
  - ne contenant aucune arete : $A = \varnothing$
  - contenant un unique noeud r (sa racine) choisi arbitrairement
- A tout moment, les aretes selectionnees A forment un arbre T enracine en r.
  - c.a.d. que le sous graphe forme par A est **connexe**
  - et acyclique ! (bien sur)
- On fait grossir T en ajoutant une arete e de poids minimum parmi celles traversant la coupe $(V(T), V \setminus V(T))$.
  - e est sure d'apres le Lemme 2.
- On s'arete lorsque $V(T) = V$.

Algorithme de Prim

Algorithme 1 : Prim(G,r)

Trois tableaux *poids*, *pere* et *coul* de taille n;
pour u *de* 0 *a* $n-1$ **faire**
    $poids[u] \leftarrow +\infty$; $pere[u] \leftarrow \perp$; $coul[u] \leftarrow blanc$;
fin
$poids[r] \leftarrow 0$; $Q \leftarrow V$;
tant que $Q \neq \emptyset$ **faire**
    $u \leftarrow min_{poids}(Q)$; $Q \leftarrow Q \setminus \{u\}$; $coul[u] \leftarrow noir$;
    pour $v \in N(u)$ **faire**
        si $coul[v] = blanc$ *et* $w(u,v) < poids[v]$ **alors**
            $poids[v] \leftarrow w(u,v)$; $pere[v] \leftarrow u$;
        fin
    fin
fin
retourner *pere*

Exemple d'execution de Prim



Preuve de correction de l'algorithme de Prim

Notation : B est l'ensemble des noeuds blancs et N l'ensemble des noeuds noirs.

Preuve de correction de l'algorithme de Prim

Notation : B est l'ensemble des noeuds blancs et N l'ensemble des noeuds noirs.
Remarque : Dans l'algorithme, $V(T)$ est l'ensemble N des noeuds noirs et T est decrit par le tableau *pere* : les aretes de T sont les aretes $\{u, pere(u)\}$, pour u un noeud noir.

Preuve de correction de l'algorithme de Prim

Notation : B est l'ensemble des noeuds blancs et N l'ensemble des noeuds noirs.

Remarque : Dans l'algorithme, $V(T)$ est l'ensemble N des noeuds noirs et T est decrit par le tableau *pere* : les aretes de T sont les aretes $\{u, pere(u)\}$, pour u un noeud noir.

Observation 1 : Initialement, tous les noeuds sont places dans Q (ligne 5). Un noeud est blanc lorsqu'il est present dans Q et devient noir, pour toujours, lorsqu'il sort de Q (ligne 7).

Preuve de correction de l'algorithme de Prim

Notation : B est l'ensemble des noeuds blancs et N l'ensemble des noeuds noirs.

Remarque : Dans l'algorithme, $V(T)$ est l'ensemble N des noeuds noirs et T est decrit par le tableau *pere* : les aretes de T sont les aretes $\{u, pere(u)\}$, pour u un noeud noir.

Observation 1 : Initialement, tous les noeuds sont places dans Q (ligne 5). Un noeud est blanc lorsqu'il est present dans Q et devient noir, pour toujours, lorsqu'il sort de Q (ligne 7).

Propriete 1 : Pour tout noeud blanc v, $poids[v]$ est le poids minimum d'une arete reliant v a un noeud noir.

Preuve de correction de l'algorithme de Prim

Notation : B est l'ensemble des noeuds blancs et N l'ensemble des noeuds noirs.

Remarque : Dans l'algorithme, $V(T)$ est l'ensemble N des noeuds noirs et T est decrit par le tableau *pere* : les aretes de T sont les aretes $\{u, pere(u)\}$, pour u un noeud noir.

Observation 1 : Initialement, tous les noeuds sont places dans Q (ligne 5). Un noeud est blanc lorsqu'il est present dans Q et devient noir, pour toujours, lorsqu'il sort de Q (ligne 7).

Propriete 1 : Pour tout noeud blanc v, $poids[v]$ est le poids minimum d'une arete reliant v a un noeud noir.

Démonstration.

□

Preuve de correction de l'algorithme de Prim

Notation : B est l'ensemble des noeuds blancs et N l'ensemble des noeuds noirs.

Remarque : Dans l'algorithme, $V(T)$ est l'ensemble N des noeuds noirs et T est decrit par le tableau *pere* : les aretes de T sont les aretes $\{u, pere(u)\}$, pour u un noeud noir.

Observation 1 : Initialement, tous les noeuds sont places dans Q (ligne 5). Un noeud est blanc lorsqu'il est present dans Q et devient noir, pour toujours, lorsqu'il sort de Q (ligne 7).

Propriete 1 : Pour tout noeud blanc v, $poids[v]$ est le poids minimum d'une arete reliant v a un noeud noir.

Démonstration.

Initialement tous les noeuds sont blancs et ont un poids infini.

□

Preuve de correction de l'algorithme de Prim

Notation : B est l'ensemble des noeuds blancs et N l'ensemble des noeuds noirs.
Remarque : Dans l'algorithme, $V(T)$ est l'ensemble N des noeuds noirs et T est decrit par le tableau *pere* : les aretes de T sont les aretes $\{u, pere(u)\}$, pour u un noeud noir.
Observation 1 : Initialement, tous les noeuds sont places dans Q (ligne 5). Un noeud est blanc lorsqu'il est present dans Q et devient noir, pour toujours, lorsqu'il sort de Q (ligne 7).
Propriete 1 : Pour tout noeud blanc v, $poids[v]$ est le poids minimum d'une arete reliant v a un noeud noir.

Démonstration.

Initialement tous les noeuds sont blancs et ont un poids infini. Lorsqu'un noeud u devient noir (ligne 7), la boucle ligne 8 maintient le poids $poids[v]$ des noeuds blancs v comme le poids minimum d'une arete reliant v a un noeud noir. □

Preuve de correction de l'algorithme de Prim

<u>**Lemme 3 :**</u> A la ligne 7, l'arete $\{u, pere(u)\}$ est l'arete de poids minimum traversant la coupe (N, B).

Démonstration.

$\square$

Preuve de correction de l'algorithme de Prim

<u>**Lemme 3 :**</u> A la ligne 7, l'arete $\{u, pere(u)\}$ est l'arete de poids minimum traversant la coupe (N, B).

Démonstration.

- A la ligne 7, le noeud u est selectionne comme le noeud de Q de poids minimum.

$\square$

Preuve de correction de l'algorithme de Prim

Lemme 3 : A la ligne 7, l'arete $\{u, pere(u)\}$ est l'arete de poids minimum traversant la coupe (N, B).

Démonstration.

- A la ligne 7, le noeud u est selectionne comme le noeud de Q de poids minimum.
- D'apres observation 1 et propriete 1, u est le noeud blanc dont le poids minimum $poids[u]$ d'une arete le reliant a un noeud noir est minimum.

□

Preuve de correction de l'algorithme de Prim

Lemme 3 : A la ligne 7, l'arete $\{u, pere(u)\}$ est l'arete de poids minimum traversant la coupe (N, B).

Démonstration.

- A la ligne 7, le noeud u est selectionne comme le noeud de Q de poids minimum.
- D'apres observation 1 et propriete 1, u est le noeud blanc dont le poids minimum $poids[u]$ d'une arete le reliant a un noeud noir est minimum.
- A la ligne 10, $pere[v]$ est maintenu de sorte que pour tout noeud blanc v, l'arete $\{v, pere[v]\}$ a poids $poids[v]$.

□

Preuve de correction de l'algorithme de Prim

Lemme 3 : A la ligne 7, l'arete $\{u, pere(u)\}$ est l'arete de poids minimum traversant la coupe (N, B).

Démonstration.

- A la ligne 7, le noeud u est selectionne comme le noeud de Q de poids minimum.
- D'apres observation 1 et propriete 1, u est le noeud blanc dont le poids minimum $poids[u]$ d'une arete le reliant a un noeud noir est minimum.
- A la ligne 10, $pere[v]$ est maintenu de sorte que pour tout noeud blanc v, l'arete $\{v, pere[v]\}$ a poids $poids[v]$.
- Ainsi, a la ligne 7, on a bien que l'arete $\{u, pere(u)\}$ est l'arete de poids minimum traversant la coupe (N, B).

$\square$

Preuve de correction de l'algorithme de Prim

Lemme 3 : A la ligne 7, l'arete $\{u, pere(u)\}$ est l'arete de poids minimum traversant la coupe (N, B).

Démonstration.

- A la ligne 7, le noeud u est selectionne comme le noeud de Q de poids minimum.
- D'apres observation 1 et propriete 1, u est le noeud blanc dont le poids minimum $poids[u]$ d'une arete le reliant a un noeud noir est minimum.
- A la ligne 10, $pere[v]$ est maintenu de sorte que pour tout noeud blanc v, l'arete $\{v, pere[v]\}$ a poids $poids[v]$.
- Ainsi, a la ligne 7, on a bien que l'arete $\{u, pere(u)\}$ est l'arete de poids minimum traversant la coupe (N, B).

$\square$

$\Longrightarrow$ D'apres le lemme 2, l'algorithme de Prim est correct.

Analyse de complexite de l'algorithme de Prim

- Toutes les instructions sont en temps $O(1)$, sauf

Analyse de complexite de l'algorithme de Prim

- Toutes les instructions sont en temps $O(1)$, sauf
  - $u \leftarrow min_{poids}(Q)$, a la ligne 7

Analyse de complexite de l'algorithme de Prim

- Toutes les instructions sont en temps $O(1)$, sauf
  - $u \leftarrow min_{poids}(Q)$, a la ligne 7
  - $poids[v] \leftarrow w(u,v)$, a la ligne 10

Analyse de complexite de l'algorithme de Prim

- Toutes les instructions sont en temps $O(1)$, sauf
  - $u \leftarrow min_{poids}(Q)$, a la ligne 7
  - $poids[v] \leftarrow w(u, v)$, a la ligne 10
  - qui dependent de la structure de donnees pour extraire le min

Analyse de complexite de l'algorithme de Prim

- Toutes les instructions sont en temps $O(1)$, sauf
  - $u \leftarrow min_{poids}(Q)$, a la ligne 7
  - $poids[v] \leftarrow w(u, v)$, a la ligne 10
  - qui dependent de la structure de donnees pour extraire le min
    - meme structure que pour Dijkstra !

Analyse de complexite de l'algorithme de Prim

- Toutes les instructions sont en temps $O(1)$, sauf
  - $u \leftarrow min_{poids}(Q)$, a la ligne 7
  - $poids[v] \leftarrow w(u, v)$, a la ligne 10
  - qui dependent de la structure de donnees pour extraire le min
    - meme structure que pour Dijkstra !
- au total : $O(n + m)$ + temps utilisation structure de donnees

Analyse de complexite de l'algorithme de Prim

- Toutes les instructions sont en temps $O(1)$, sauf
  - $u \leftarrow min_{poids}(Q)$, a la ligne 7
  - $poids[v] \leftarrow w(u, v)$, a la ligne 10
  - qui dependent de la structure de donnees pour extraire le min
    - meme structure que pour Dijkstra !
- au total : $O(n + m)$ + temps utilisation structure de donnees

Differentes implementations de la file de priorite

Analyse de complexite de l'algorithme de Prim

- Toutes les instructions sont en temps $O(1)$, sauf
  - $u \leftarrow min_{poids}(Q)$, a la ligne 7
  - $poids[v] \leftarrow w(u, v)$, a la ligne 10
  - qui dependent de la structure de donnees pour extraire le min
    - meme structure que pour Dijkstra !
- au total : $O(n + m)$ + temps utilisation structure de donnees

Differentes implementations de la file de priorite

- Tableau des poids : $O(n^2)$
  - Total algo : $O(n^2)$

Analyse de complexite de l'algorithme de Prim

- Toutes les instructions sont en temps $O(1)$, sauf
  - $u \leftarrow min_{poids}(Q)$, a la ligne 7
  - $poids[v] \leftarrow w(u, v)$, a la ligne 10
  - qui dependent de la structure de donnees pour extraire le min
    - meme structure que pour Dijkstra !
- au total : $O(n + m)$ + temps utilisation structure de donnees

Differentes implementations de la file de priorite

- Tableau des poids : $O(n^2)$
  - Total algo : $O(n^2)$
- Tas binaire : $O((n + m) \log n)$
  - Total algo : $O(m \log n)$ (graphe connexe)

Analyse de complexite de l'algorithme de Prim

- Toutes les instructions sont en temps $O(1)$, sauf
  - $u \leftarrow min_{poids}(Q)$, a la ligne 7
  - $poids[v] \leftarrow w(u, v)$, a la ligne 10
  - qui dependent de la structure de donnees pour extraire le min
    - meme structure que pour Dijkstra !
- au total : $O(n + m)$ + temps utilisation structure de donnees

Differentes implementations de la file de priorite

- Tableau des poids : $O(n^2)$
  - Total algo : $O(n^2)$
- Tas binaire : $O((n + m) \log n)$
  - Total algo : $O(m \log n)$ (graphe connexe)
- Tas fibonacci : $O(m + n \log n)$
  - Total algo : $O(m + n \log n)$

Algorithme de Kruskal

Idee :

- On part d'un ensemble d'aretes vide $A = \varnothing$

Algorithme de Kruskal

Idee :

- On part d'un ensemble d'aretes vide $A = \varnothing$
- On considere les aretes de G une par une par **poids croissant**

Algorithme de Kruskal

Idee :

- On part d'un ensemble d'aretes vide $A = \varnothing$
- On considere les aretes de G une par une par **poids croissant**
- On teste si l'arete courante cree un cycle avec celles de A :

Algorithme de Kruskal

Idee :

- On part d'un ensemble d'aretes vide $A = \varnothing$
- On considere les aretes de G une par une par **poids croissant**
- On teste si l'arete courante cree un cycle avec celles de A :
  - Si oui, on la rejette

Algorithme de Kruskal

Idee :

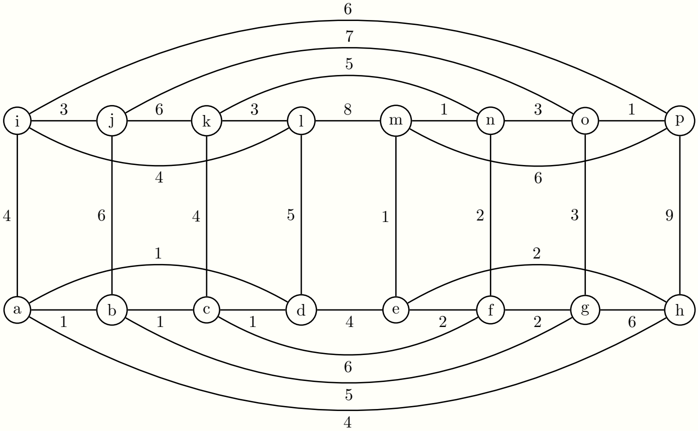
- On part d'un ensemble d'aretes vide $A = \varnothing$
- On considere les aretes de G une par une par **poids croissant**
- On teste si l'arete courante cree un cycle avec celles de A :
  - Si oui, on la rejette
  - Si non, on l'inclut dans A

Algorithme de Kruskal

Algorithme 2 : Kruskal(G)

$A \leftarrow \varnothing$; $\mathcal{F} \leftarrow \varnothing$;
pour *u de* 0 *a* $n-1$ **faire**
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{\{u\}\}$;
fin
Trier les aretes de G par w croissant;
pour *chaque arete uv de G par poids croissant* **faire**
    si $FIND_{\mathcal{F}}(u) \neq FIND_{\mathcal{F}}(v)$ **alors**
        $A \leftarrow A \cup \{uv\}$;
        $UNION_{\mathcal{F}}(u, v)$;
    fin
fin
retourner A

Exemple d'execution de Kruskal



Preuve de correction de l'algorithme de Kruskal

Decoule directement du Lemme suivant.

Lemme 4 : A la ligne 8, l'arete uv est sure pour A.

Preuve de correction de l'algorithme de Kruskal

Decoule directement du Lemme suivant.

Lemme 4 : A la ligne 8, l'arete uv est sure pour A.

Démonstration.

- Par le lemme 1, il suffit de trouver une coupe C telle que :
  - C respecte A et
  - uv est de poids min parmi les aretes traversant C

□

Preuve de correction de l'algorithme de Kruskal

Decoule directement du Lemme suivant.

Lemme 4 : A la ligne 8, l'arete uv est sure pour A.

Démonstration.

- Par le lemme 1, il suffit de trouver une coupe C telle que :
  - C respecte A et
  - uv est de poids min parmi les aretes traversant C
- La coupe $C = (F_u, V \setminus F_u)$ convient.
  - ou F_u est la partie de $\mathcal{F}$ contenant u

□

Preuve de correction de l'algorithme de Kruskal

Decoule directement du Lemme suivant.

Lemme 4 : A la ligne 8, l'arete uv est sure pour A.

Démonstration.

- Par le lemme 1, il suffit de trouver une coupe C telle que :
  - C respecte A et
  - uv est de poids min parmi les aretes traversant C
- La coupe $C = (F_u, V \setminus F_u)$ convient.
  - ou F_u est la partie de $\mathcal{F}$ contenant u
- C respecte A car les aretes de A ont leur deux extremites dans la meme partie de $\mathcal{F}$

□

Preuve de correction de l'algorithme de Kruskal

Decoule directement du Lemme suivant.

Lemme 4 : A la ligne 8, l'arete uv est sure pour A.

Démonstration.

- Par le lemme 1, il suffit de trouver une coupe C telle que :
  - C respecte A et
  - uv est de poids min parmi les aretes traversant C
- La coupe $C = (F_u, V \setminus F_u)$ convient.
  - ou F_u est la partie de $\mathcal{F}$ contenant u
- C respecte A car les aretes de A ont leur deux extremites dans la meme partie de $\mathcal{F}$
- Les aretes xy telles que $w(x, y) < w(u, v)$ ne traversent pas C.

$\square$

Preuve de correction de l'algorithme de Kruskal

Decoule directement du Lemme suivant.

Lemme 4 : A la ligne 8, l'arete uv est sure pour A.

Démonstration.

- Par le lemme 1, il suffit de trouver une coupe C telle que :
  - C respecte A et
  - uv est de poids min parmi les aretes traversant C
- La coupe $C = (F_u, V \setminus F_u)$ convient.
  - ou F_u est la partie de $\mathcal{F}$ contenant u
- C respecte A car les aretes de A ont leur deux extremites dans la meme partie de $\mathcal{F}$
- Les aretes xy telles que $w(x, y) < w(u, v)$ ne traversent pas C.
  - En effet, xy a ete consideree avant par l'algorithme. Ainsi, xy a

$\square$

Preuve de correction de l'algorithme de Kruskal

Decoule directement du Lemme suivant.

Lemme 4 : A la ligne 8, l'arete uv est sure pour A.

Démonstration.

- Par le lemme 1, il suffit de trouver une coupe C telle que :
  - C respecte A et
  - uv est de poids min parmi les aretes traversant C
- La coupe $C = (F_u, V \setminus F_u)$ convient.
  - ou F_u est la partie de $\mathcal{F}$ contenant u
- C respecte A car les aretes de A ont leur deux extremites dans la meme partie de $\mathcal{F}$
- Les aretes xy telles que $w(x, y) < w(u, v)$ ne traversent pas C.
  - En effet, xy a ete consideree avant par l'algorithme. Ainsi, xy a
    - soit ete incluse dans A,
    - soit ete rejetee car x et y sont dans la meme partie de $\mathcal{F}$

$\square$

Preuve de correction de l'algorithme de Kruskal

Decoule directement du Lemme suivant.

Lemme 4 : A la ligne 8, l'arete uv est sure pour A.

Démonstration.

- Par le lemme 1, il suffit de trouver une coupe C telle que :
  - C respecte A et
  - uv est de poids min parmi les aretes traversant C
- La coupe $C = (F_u, V \setminus F_u)$ convient.
  - ou F_u est la partie de $\mathcal{F}$ contenant u
- C respecte A car les aretes de A ont leur deux extremites dans la meme partie de $\mathcal{F}$
- Les aretes xy telles que $w(x,y) < w(u,v)$ ne traversent pas C.
  - En effet, xy a ete consideree avant par l'algorithme. Ainsi, xy a
    - soit ete incluse dans A,
    - soit ete rejetee car x et y sont dans la meme partie de $\mathcal{F}$
  - Dans les deux cas, x et y sont encore dans la meme partie de F

□

Analyse de complexite de l'algorithme de Kruskal

- Le trie ligne 5 prend un temps $O(m \log m) = O(m \log n)$

Analyse de complexite de l'algorithme de Kruskal

- Le trie ligne 5 prend un temps $O(m \log m) = O(m \log n)$
- Le reste de la complexite de l'algo depend entierement de la structure utilisee pour UNION-FIND :
  - m fois *FIND*
  - $n - 1$ fois *UNION*

Analyse de complexite de l'algorithme de Kruskal

- Le trie ligne 5 prend un temps $O(m \log m) = O(m \log n)$
- Le reste de la complexite de l'algo depend entierement de la structure utilisee pour UNION-FIND :
  - m fois *FIND*
  - $n - 1$ fois *UNION*
- Avec l'implementation simple de UNION-FIND :
  - *FIND* : temps $O(1)$ pire des cas
  - *UNION* : temps total de $O(n \log n)$ pour n unions

Analyse de complexite de l'algorithme de Kruskal

- Le trie ligne 5 prend un temps $O(m \log m) = O(m \log n)$
- Le reste de la complexite de l'algo depend entierement de la structure utilisee pour UNION-FIND :
  - m fois *FIND*
  - $n - 1$ fois *UNION*
- Avec l'implementation simple de UNION-FIND :
  - *FIND* : temps $O(1)$ pire des cas
  - *UNION* : temps total de $O(n \log n)$ pour n unions
- Total pour l'algo : $O((m+n) \log n) = O(m \log n)$
  car G connexe

Propriete fondamentale des arbres couvrants de poids min

Theoreme :
La sequence triee des poids des aretes est la meme pour tous les arbres couvrants de poids minimum de G.

Démonstration.

$\square$