Arbres

Plan

1. Arbres et forêts
2. Arbre enraciné
3. Arbres couvrant de poids minimum

Plan

1 Arbres et forêts

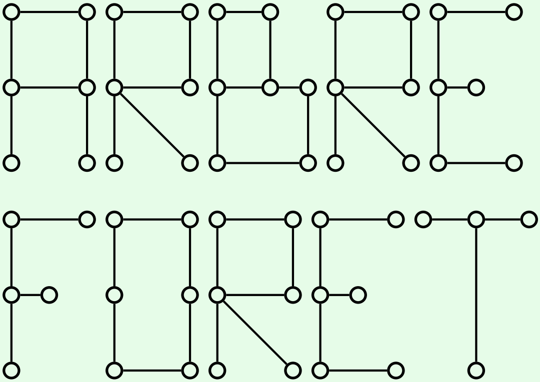
2 Arbre enraciné

3 Arbres couvrant de poids minimum

Graphe acyclique

G **acyclique** : ne contient pas de cycle

Quelles composantes connexes du graphe suivant sont acycliques ?



Graphe acyclique

Un graphe acyclique G à n sommets possède au plus $n - 1$ arêtes.

Graphe acyclique

Un graphe acyclique G à n sommets possède au plus $n-1$ arêtes.

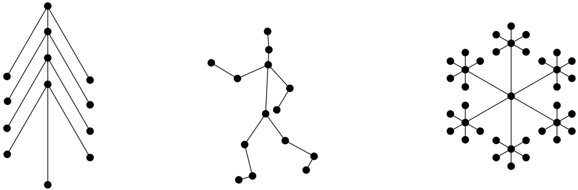
Preuve par induction sur le nombre de sommets du graphe.

- Si G est d'ordre 1, il ne possède aucune arête. ✔
- Supposons la propriété vraie à l'ordre n et établissons-la à l'ordre $n+1$. Soit $G=(V,E)$ acyclique à $n+1$ sommets.
- On sait que (cours précédent) si un graphe est acyclique, alors il possède un sommet, noté x, de degré au plus 1.
- Soit $G'=(V',E')$ le graphe d'ordre n tel que $V'=V\backslash\{x\}$ et E' est égal à E privé de l'arête incidente à x si elle existe.
- Le graphe G' est sans cycle, donc, par l'hypothèse d'induction, il possède au plus $n-1$ arêtes.
- Or $d(x)<2$ impose que E diffère de E' par au plus une arête. Donc $|E|$ est inférieure à n. ✔

Arbres et forêts

Forêt : graphe acyclique.

Arbre : graphe acyclique connexe

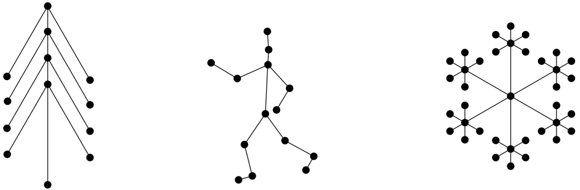


Dessins de *Invitation to mathematics* de Matoušek et Nešetřil

Arbres et forêts

Forêt : graphe acyclique.

Arbre : graphe acyclique connexe



Dessins de *Invitation to mathematics* de Matoušek et Nešetřil

Chaque "bout" (composante connexe) de la forêt est un arbre. . .

Arbres et forêts

Quelques questions pour comprendre

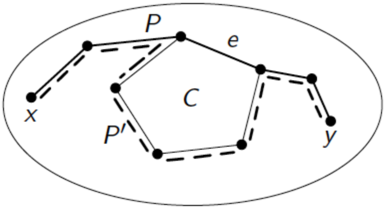
- A quelle condition un arbre est-il un graphe complet ?
- A quelle condition un cycle est-il un graphe complet ?
- Si on retire une arête à un cycle, qu'obtient-on ?
- Un arbre peut-il être Eulérien ?
- Un arbre peut-il être Hamiltonien ?
- Quels arbres contiennent une chaîne Eulérienne ?

Arbres et forêts

Un premier lemme

Si e est une arête d'un cycle C d'un graphe connexe G, alors $G - e$ est connexe.

$\forall x, y \in V(G)$, il existe une xy-chaine P car G est connexe. Si $e \notin E(P)$, alors P est une xy-chaine dans $G - e$. Si $e \in E(P)$ alors en remplaçant dans P l'arête e par la chaine $C - e$ on obtient une xy-chaine P' dans $G - e$. $G - e$ est donc connexe.



Arbres et forêts

Caractérisations d'un Arbre

Jamais deux sans trois

Soit T, un graphe d'ordre n. Deux des propriétés suivantes impliquent la troisième.

1. T est connexe
2. T a $n-1$ arêtes
3. T est acyclique

Arbres et forêts

- $(1) + (3) \Rightarrow (2)$
  - T connexe $\Rightarrow$ T a au moins $n-1$ arêtes
  - T acyclique $\Rightarrow$ T a au plus $n-1$ arêtes

Arbres et forêts

- $(1) + (3) \Rightarrow (2)$
  - T connexe $\Rightarrow$ T a au moins $n-1$ arêtes
  - T acyclique $\Rightarrow$ T a au plus $n-1$ arêtes

- $(1) + (2) \Rightarrow (3)$
  - Soit C un cycle de T.
  - Si on enlève une arête de C, T reste connexe à $n-2$ arêtes. ✘

Arbres et forêts

- (1) + (3) $\Rightarrow$ (2)
  - T connexe $\Rightarrow$ T a au moins $n-1$ arêtes
  - T acyclique $\Rightarrow$ T a au plus $n-1$ arêtes

- (1) + (2) $\Rightarrow$ (3)
  - Soit C un cycle de T.
  - Si on enlève une arête de C, T reste connexe à $n-2$ arêtes. ✘

- (2) + (3) $\Rightarrow$ (1)
  - Soit c le nombre de composantes connexes de T
  - Chaque composante connexe C_i de T a v_i sommets et $v_i - 1$ arêtes (connexe et acyclique)
  - On a donc $\sum_i (v_i - 1) = \sum_i v_i - c = n - c$
  - et $\sum_i (v_i - 1) = n - 1$ (par (2))
  - D'où $c = 1$ et donc T connexe.

Arbres et forêts

Autres caractérisations d'un arbre

Les trois propositions suivantes sont équivalentes

- T est un arbre
- **connexe minimal** : la suppression de toute arête le déconnecte
- **acyclique maximal** : l'ajout de toute arête crée un cycle

Arbres et forêts

Soit $T = (V, E)$ un arbre

- Arbre $\Rightarrow$ Connexe minimal
    - T arbre $\Rightarrow |E| = n - 1$
    - T auquel on enlève une arête a $n - 2$ arêtes donc il ne peut pas être connexe.
    - Donc la suppression de n'importe quelle arête déconnecte T.

Arbres et forêts

Soit $T = (V, E)$ un arbre

- Arbre $\Rightarrow$ Connexe minimal
  - T arbre $\Rightarrow |E| = n - 1$
  - T auquel on enlève une arête a $n - 2$ arêtes donc il ne peut pas être connexe.
  - Donc la suppression de n'importe quelle arête déconnecte T.

- Arbre $\Rightarrow$ Acyclique maximal
  - Soient x et y deux sommets de T tels que $xy \notin E$.
  - T est connexe donc il existe dans T une chaîne C de x à y
  - C est d'extrémités x et y et il ne contient pas l'arête xy
  - Donc C auquel on ajoute xy est un cycle
  - Donc l'ajout de n'importe quelle arête crée un cycle

Arbres et forêts

- Connexe minimal $\Rightarrow$ Arbre
  - Soit T connexe minimal.
  - Si T contient un cycle, la suppression d'une arête de ce cycle ne peut pas le rendre non connexe. Donc T est acyclique.
  - T est acyclique et connexe $\Rightarrow$ T est un arbre.

Arbres et forêts

- Connexe minimal $\Rightarrow$ Arbre
  - Soit T connexe minimal.
  - Si T contient un cycle, la suppression d'une arête de ce cycle ne peut pas le rendre non connexe. Donc T est acyclique.
  - T est acyclique et connexe $\Rightarrow$ T est un arbre.

- Acyclique maximal $\Rightarrow$ Arbre
  - Supposons $T = (V, E)$ acyclique maximal
  - si T est non connexe alors, il existe x et y deux sommets de T tels qu'il n'y a pas dans T de chaîne de x à y (en particulier $xy \notin E$).
  - Donc l'ajout de l'arête xy à T ne crée pas de cycle : contradiction avec l'hypothèse acyclique maximal
  - Donc T est connexe et acyclique $\Rightarrow$ T est un arbre.

Arbres et forêts

Théorème

Soit G acyclique ayant au moins une arête, alors G possède un sommet de degré 1.

Arbres et forêts

Théorème

Soit G acyclique ayant au moins une arête, alors G possède un sommet de degré 1.

C'est (à un détail près) la contraposée de la propriété vue dans le cours précédent :

Si dans un graphe G tout sommet est de degré supérieur ou égal à 2, alors G possède au moins un cycle.

Sur le même principe que la recherche d'un cycle :

Cheminement_Arbre(i)

Données : un sommet i avec $d(i) \geq 1$
Résultat : un sommet de degré 1

Soit j un sommet voisin de i
tant que $d(j) > 1$, **faire**
    soit k un voisin de j différent de i
    $i \leftarrow j$
    $j \leftarrow k$
retourner j

Arbres et forêts

Vérification de Cheminement_Arbre(i)

1. *tout s'exécute correctement*
   - Comme le graphe possède au moins une arête, i et j sont bien définis avant le **tant que**.
   - Dans la boucle, comme $d(j) > 1$, k est bien défini

Arbres et forêts

Vérification de Cheminement_Arbre(i)

1. *tout s'exécute correctement*
   - Comme le graphe possède au moins une arête, i et j sont bien définis avant le **tant que**.
   - Dans la boucle, comme $d(j) > 1$, k est bien défini
2. *en un nombre fini d'étapes*
   - Les sommets visités forment une chaîne.
   - Il n'y a pas d'aller-retour le long d'une arête
   - Si un sommet est visité deux fois et qu'il n'y a pas d'aller retour le long d'une arête alors on obtient un cycle
   - acyclique $\Rightarrow$ un sommet n'est jamais visité deux fois

Arbres et forêts

Vérification de Cheminement_Arbre(i)

1. *tout s'exécute correctement*
   - Comme le graphe possède au moins une arête, i et j sont bien définis avant le **tant que**.
   - Dans la boucle, comme $d(j) > 1$, k est bien défini
2. *en un nombre fini d'étapes*
   - Les sommets visités forment une chaîne.
   - Il n'y a pas d'aller-retour le long d'une arête
   - Si un sommet est visité deux fois et qu'il n'y a pas d'aller retour le long d'une arête alors on obtient un cycle
   - acyclique $\Rightarrow$ un sommet n'est jamais visité deux fois
3. *en cas d'arrêt, on obtient l'objet souhaité*
   - sortie du tant que avec $d(j) = 1$

Arbres et forêts

Vérification de Cheminement_Arbre(i)

1. *tout s'exécute correctement*
   - Comme le graphe possède au moins une arête, i et j sont bien définis avant le **tant que**.
   - Dans la boucle, comme $d(j) > 1$, k est bien défini
2. *en un nombre fini d'étapes*
   - Les sommets visités forment une chaîne.
   - Il n'y a pas d'aller-retour le long d'une arête
   - Si un sommet est visité deux fois et qu'il n'y a pas d'aller retour le long d'une arête alors on obtient un cycle
   - acyclique $\Rightarrow$ un sommet n'est jamais visité deux fois
3. *en cas d'arrêt, on obtient l'objet souhaité*
   - sortie du tant que avec $d(j) = 1$

Remarque : **Cheminement_Arbre(i)** ne retourne pas i

Arbres et forêts

Théorème

Soit G acyclique ayant au moins une arête, alors G admet au moins deux sommets de degré 1.

Arbres et forêts

Théorème

Soit G acyclique ayant au moins une arête, alors G admet au moins deux sommets de degré 1.

preuve par récurrence sur le nombre de sommets du graphe.

- $H(n)$: soit G acyclique à n sommets et au moins une arête. Alors, G admet au moins deux sommets de degré 1.
- Cas de base $n = 2$: graphe composé d'une arête. ✔
- supposons $H(n)$ vraie au rang $n \geq 2$. On veut montrer que $H(n+1)$ est vraie.
- Soit G graphe acyclique d'ordre $n+1$ avec au moins une arête.
- Par le lemme précédent, on sait que G contient un sommet x tel que $d(x) = 1$. Soit $yx \in E$ l'arête incidente à x.

Arbres et forêts

- Considérons $G' = (V/\{x\}, E/\{xy\})$
- G' est acyclique et G' a n sommets.
- Si G' n'a pas d'arête. Alors, y, le voisin de x est de degré 1. ✔
- Si G' a au moins une arête, alors, on peut appliquer $H(n)$. Donc G' a au moins deux sommets de degré 1.
- Dans ce cas, au moins un de ces sommets n'est pas y. Donc G a au moins deux sommets de degré 1. ✔

Arbres et forêts

Une autre preuve

- Soit $C = (V', E')$ une composante connexe de G.
- $\sum_{v \in V'} d(v) = 2|E'| = 2|V'| - 2 \quad$ (C acyclique)
- Comme il n'y a pas de sommet de degré 0 dans C (graphe connexe), il existe au moins deux sommets qui ont un degré égal à 1.

Arbres et forêts

Une autre preuve

- Soit $C = (V', E')$ une composante connexe de G.
- $\sum_{v \in V'} d(v) = 2|E'| = 2|V'| - 2 \quad$ (C acyclique)
- Comme il n'y a pas de sommet de degré 0 dans C (graphe connexe), il existe au moins deux sommets qui ont un degré égal à 1.

Encore une autre preuve

- Soit x, un sommet de G avec $d(x) \geq 1$.
- Soit $y =$ **Cheminement_Arbre(**x**)**.
- Soit $z =$ **Cheminement_Arbre(**y**)**.
- On a $y \neq z$ et $d(y) = d(z) = 1$

Arbres et forêts

Peut-on aller plus loin ?

Est-il vrai que G acyclique avec au moins une arête $\Rightarrow$ il existe au moins 3 sommets de degré 1 ?

Arbres et forêts

Peut-on aller plus loin ?
Est-il vrai que G acyclique avec au moins une arête $\Rightarrow$ il existe au moins 3 sommets de degré 1 ?

non : une chaîne élémentaire

$d(x) = 1$

- Dans G quelconque : **sommet pendant**
- Dans G arbre : **feuille**

Arbres et forêts

A quoi ça sert de savoir ça ?

Les deux assertions suivantes sont équivalentes pour un graphe $G = (V, E)$ et un sommet pendant $v \in V$:

- G est un arbre.
- $G' = (V \setminus \{v\}, E \setminus \{xv\})$ est un arbre. avec $xv \in E$

Arbres et forêts

Encore une caractérisation des arbres

G est un arbre si et seulement si il existe une chaîne élémentaire unique entre chaque paire de sommets de G

preuve ?

Certificat

Question oui/non avec **certificat** :

Est-ce que le graphe G est un arbre ?

1. Pour $S \subseteq V$, $cocycle(S) = \{uv \in E \mid u \in S \text{ et } v \notin S\}$

Certificat

Question oui/non avec **certificat** :

Est-ce que le graphe G est un arbre ?

oui : le nombre d'arêtes et le nombre de composantes connexes

1. Pour $S \subseteq V$, $cocycle(S) = \{uv \in E \mid u \in S \text{ et } v \notin S\}$

Certificat

Question oui/non avec **certificat** :
Est-ce que le graphe G est un arbre ?

oui : le nombre d'arêtes et le nombre de composantes connexes

non : un cycle ou un $S \subsetneq V$ avec[1] $cocycle(S) = \emptyset$

1. Pour $S \subseteq V$, $cocycle(S) = \{uv \in E \mid u \in S \text{ et } v \notin S\}$

Certificat

Question oui/non avec **certificat** :
Est-ce que le graphe G est un arbre ?

oui : le nombre d'arêtes et le nombre de composantes connexes

non : un cycle ou un $S \subsetneq V$ avec[1] $cocycle(S) = \emptyset$

autre oui : ordre $v_1, v_2, ...v_n$ sur les sommets du graphe tels que

- $G_i = (V_i, E_i) \quad i = 1, 2...n$
- $G_1 = G$,
- $G_{i+1} = (V_i \setminus \{v_i\}, E_i \setminus \{xv_i\})$ où v_i est une feuille de G_i et x est l'unique voisin de v_i dans G_i

1. Pour $S \subseteq V$, $cocycle(S) = \{uv \in E \mid u \in S \text{ et } v \notin S\}$

Plan

1. Arbres et forêts
2. Arbre enraciné
3. Arbres couvrant de poids minimum

Arbre enraciné

Arbre enraciné ou **Arborescence**

Souvent, pour manipuler un arbre, nous particularisons un sommet du graphe que nous appelons **racine** (notée r).

Le choix d'une racine revient dans un certain sens à orienter l'arbre, la racine apparaissant comme l'ancêtre commun à la manière d'un arbre généalogique. Le vocabulaire de la théorie des graphes s'en inspire directement : on parle de fils, de père, de frère...

Arbre enraciné

Pour un arbre T de racine r

- **Le père** d'un sommet x est l'unique voisin de x sur le chemin de la racine à x. La racine r est le seul sommet sans père.
- **Les fils** d'un sommet x sont les voisins de x autres que son père.
- Une **feuille** est un sommet sans fils. Les feuilles sont de degré 1.
- La **hauteur** $h(T)$ de l'arbre T est la longueur de la plus longue chaîne de la racine à une feuille.

On retrouve ce que l'on avait vu au Cours Magistral numéro 2.

Plan

1 Arbres et forêts

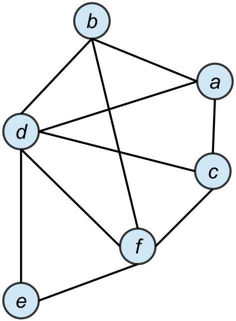
2 Arbre enraciné

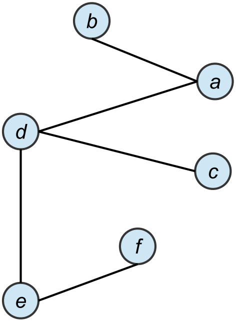
3 Arbres couvrant de poids minimum

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 2 : 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*

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*

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*

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 3 : 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*

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*

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*

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 : $w : E \to \mathbb{R}^+$

[weighted 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

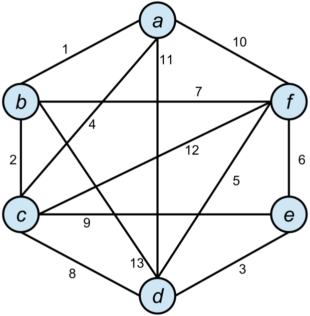
- couvrant sans cycle (algorithme de **Kruskal**)
- connexe sans cycle (algorithme de **Prim**)

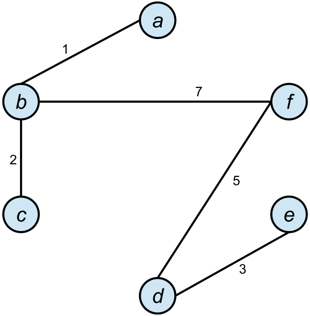
Algorithme de Kruskal

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 $T = (V, F)$

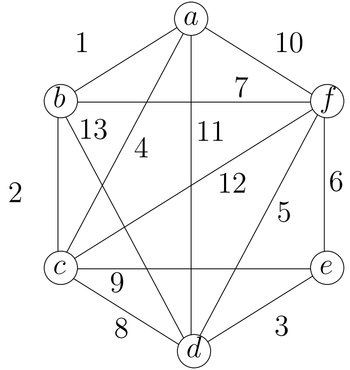
Algorithme de Kruskal : exemple



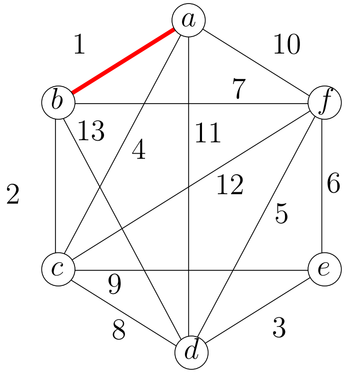


Arbre couvrant de poids $1 + 2 + 3 + 5 + 7 = 18$, c'est l'arbre couvrant de poids minimum renvoyé par l'algorithme de Kruskal (cf détails slide suivant).

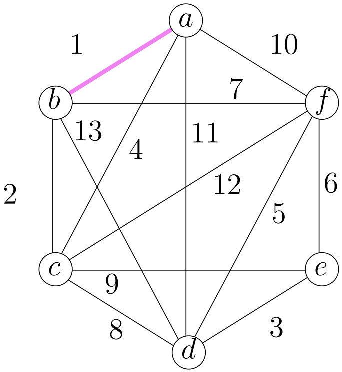
Algorithme de Kruskal : exemple



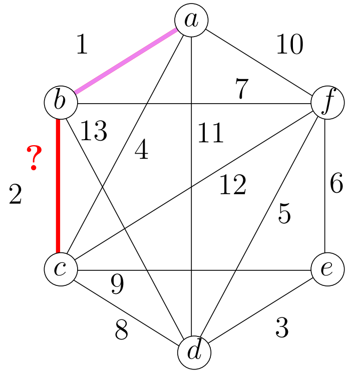
Algorithme de Kruskal : exemple



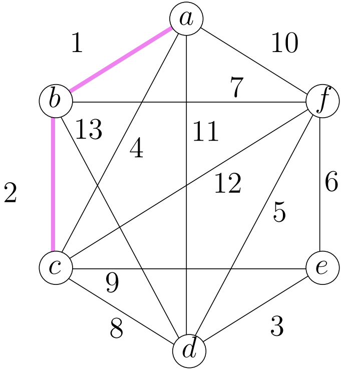
Algorithme de Kruskal : exemple



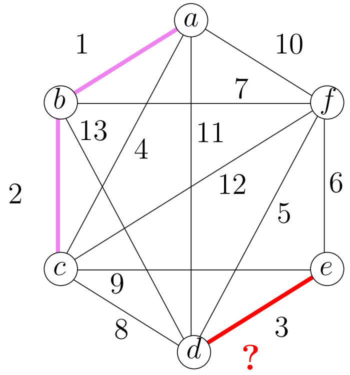
Algorithme de Kruskal : exemple



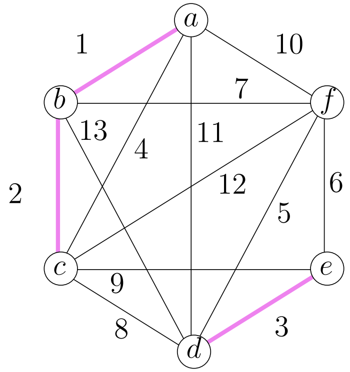
Algorithme de Kruskal : exemple



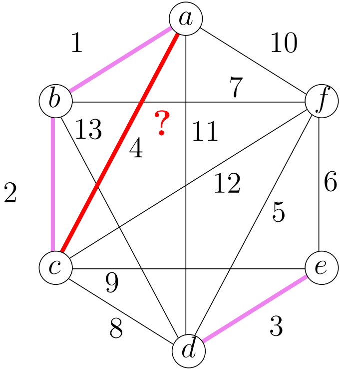
Algorithme de Kruskal : exemple



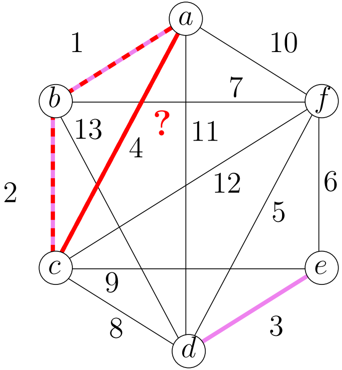
Algorithme de Kruskal : exemple



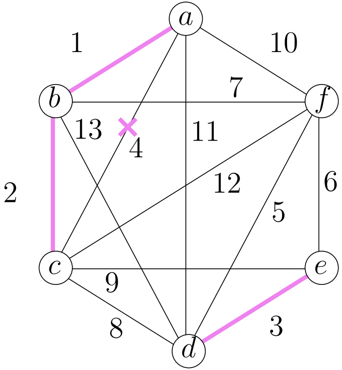
Algorithme de Kruskal : exemple



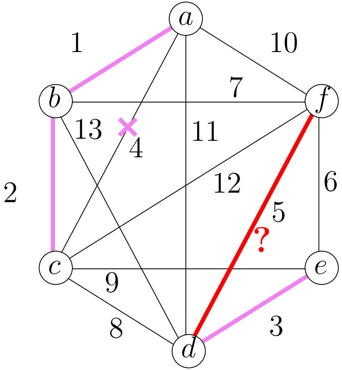
Algorithme de Kruskal : exemple



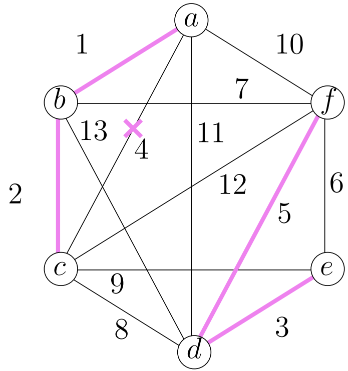
Algorithme de Kruskal : exemple



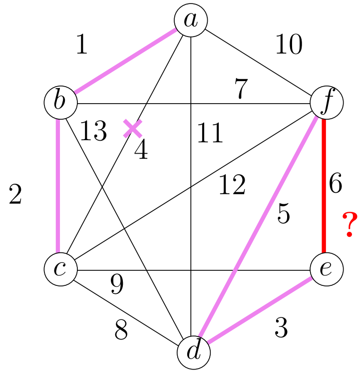
Algorithme de Kruskal : exemple



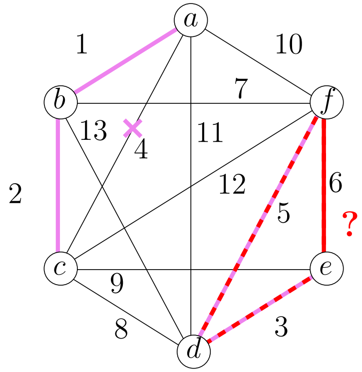
Algorithme de Kruskal : exemple



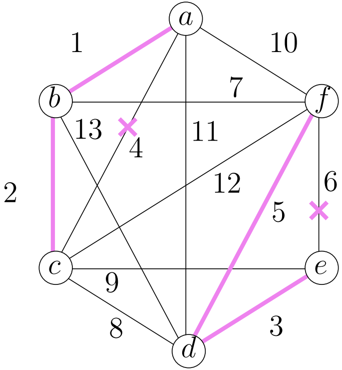
Algorithme de Kruskal : exemple



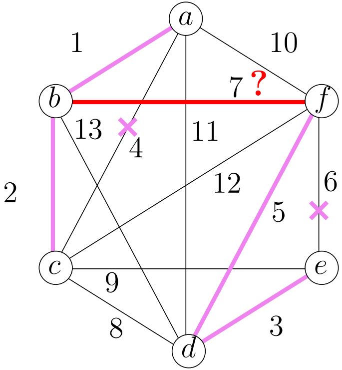
Algorithme de Kruskal : exemple



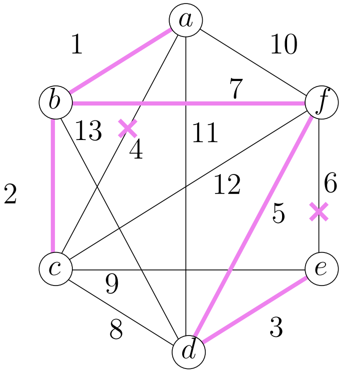
Algorithme de Kruskal : exemple



Algorithme de Kruskal : exemple



Algorithme de Kruskal : exemple



Algorithme de Kruskal

Problème : comment détecter efficacement les cycles ?

Algorithme de Kruskal

Problème : comment détecter efficacement les cycles ?

Cycle si e relie deux sommets qui sont déjà dans la même composante connexe.

Algorithme de Kruskal

Problème : comment détecter efficacement les cycles ?

Cycle si e relie deux sommets qui sont déjà dans la même composante connexe.

Structure de données pour gérer les composantes connexes 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

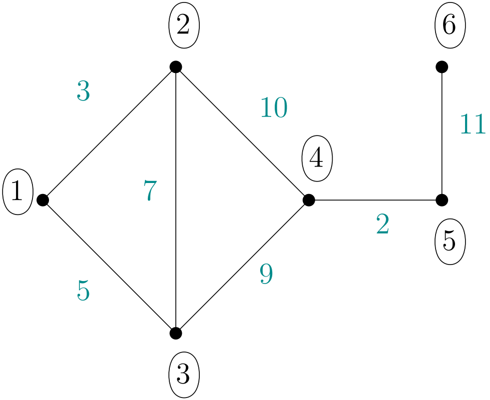
- Construire une partition qui pour chaque élément x crée la classe $\{x\}$.
- $\mathit{find}(x)$ qui renvoie le représentant de la classe contenant x.
- $\mathit{union}(x, y)$ qui fusionne les classes contenant x et y.
  Les paramètres x et y doivent être dans des classes différentes.

Algorithme de Kruskal

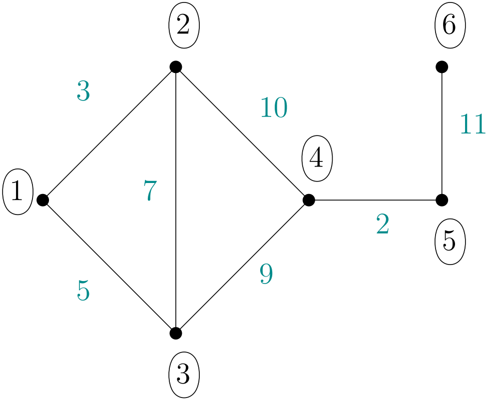
Structure **Union Find**

- 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

Kruskal avec Union-Find : exemple



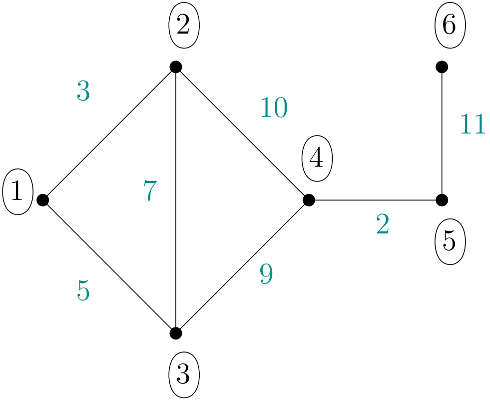
Kruskal avec Union-Find : exemple



Structure de données Union-Find

$1_0 \quad 2_0 \quad 3_0 \quad 4_0 \quad 5_0 \quad 6_0$

Kruskal avec Union-Find : exemple



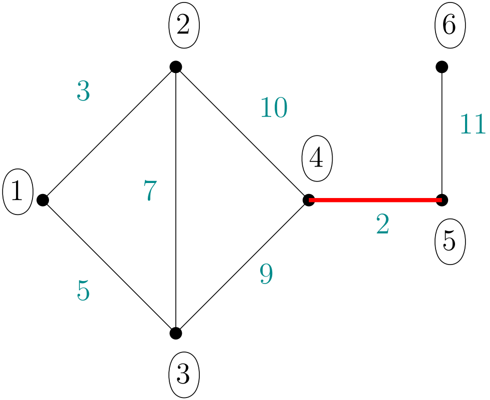
Structure de données Union-Find

$1_0 \quad 2_0 \quad 3_0 \quad 4_0 \quad 5_0 \quad 6_0$

$\uparrow$

$v_{rank(v)}$

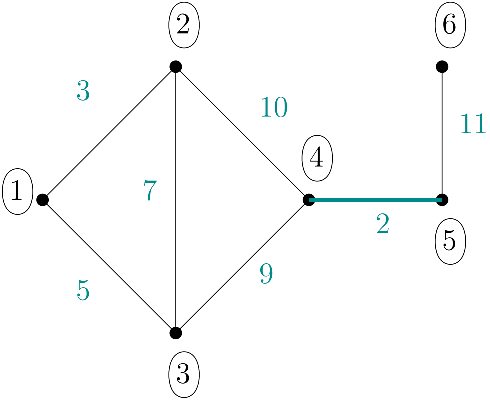
Kruskal avec Union-Find : exemple



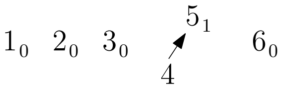
Structure de données Union-Find

1_0 2_0 3_0 $\boxed{4_0}$ $\boxed{5_0}$ 6_0

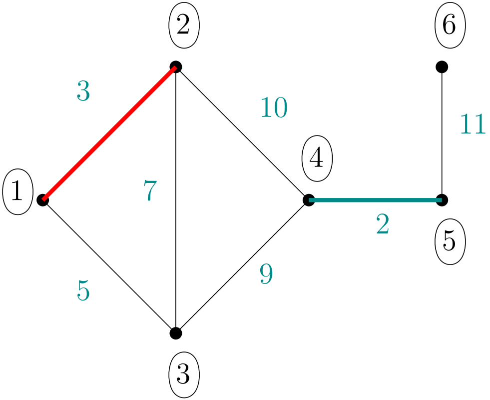
Kruskal avec Union-Find : exemple



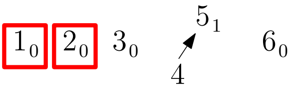
Structure de données Union-Find



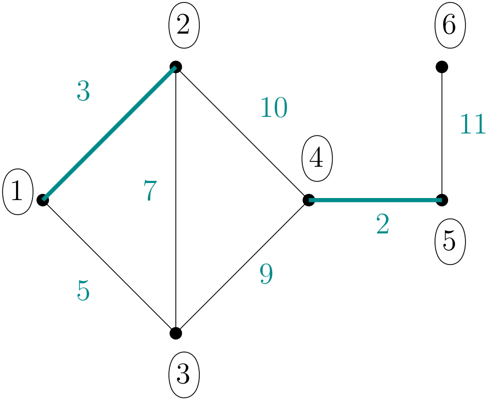
Kruskal avec Union-Find : exemple



Structure de données Union-Find



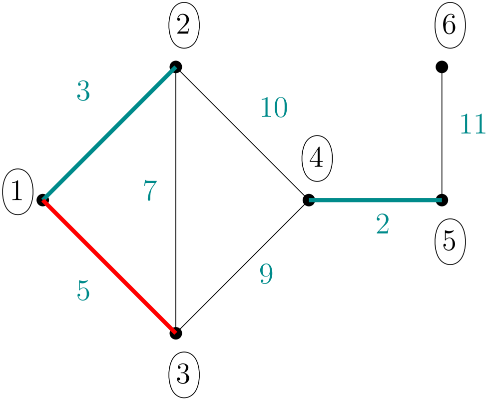
Kruskal avec Union-Find : exemple



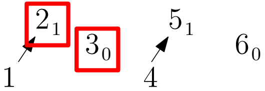
Structure de données Union-Find

$1 \nearrow 2_1 \quad 3_0 \quad 4 \nearrow 5_1 \quad 6_0$

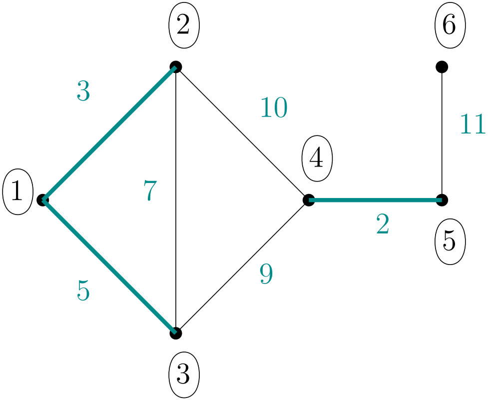
Kruskal avec Union-Find : exemple



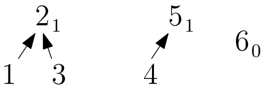
Structure de données Union-Find



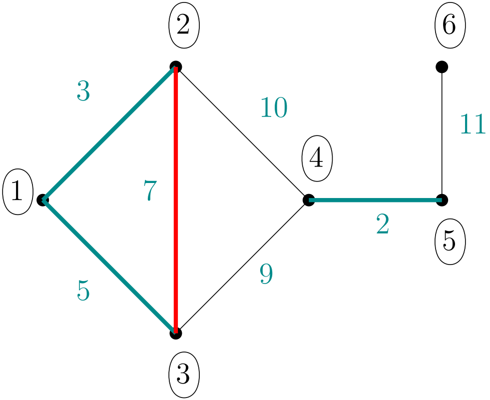
Kruskal avec Union-Find : exemple



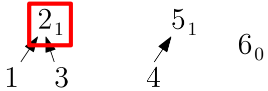
Structure de données Union-Find



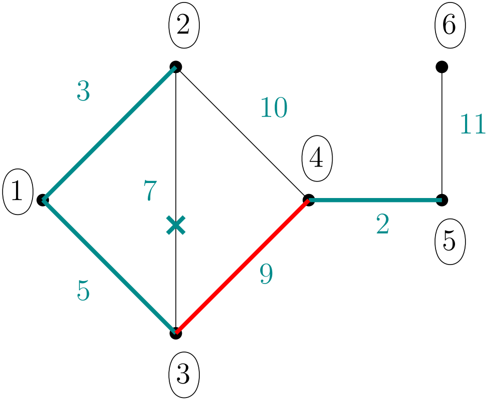
Kruskal avec Union-Find : exemple



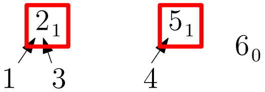
Structure de données Union-Find



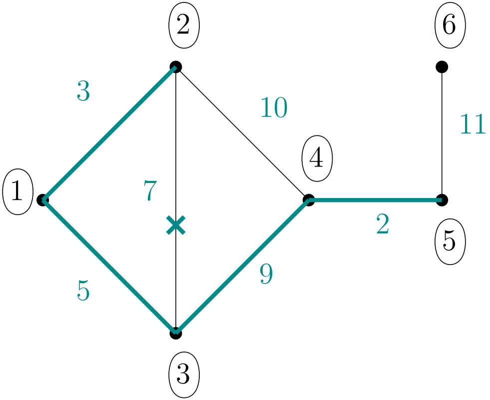
Kruskal avec Union-Find : exemple



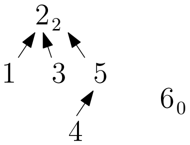
Structure de données Union-Find



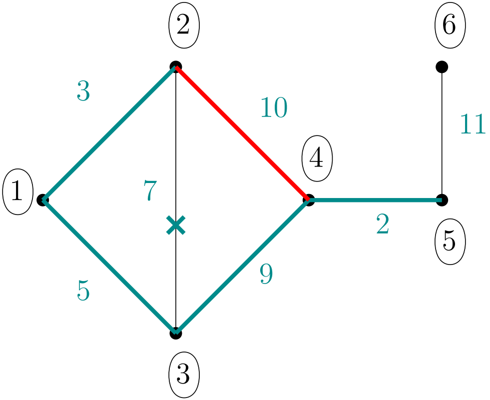
Kruskal avec Union-Find : exemple



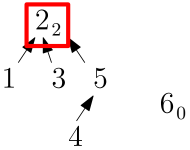
Structure de données Union-Find



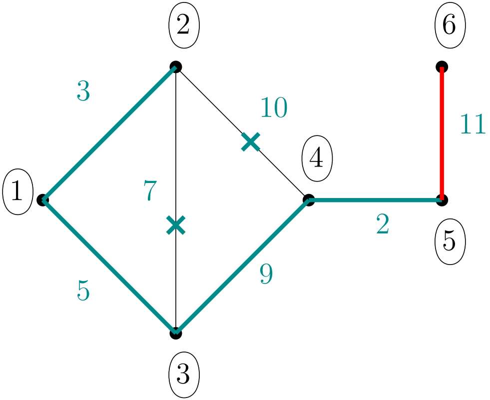
Kruskal avec Union-Find : exemple



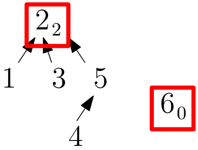
Structure de données Union-Find



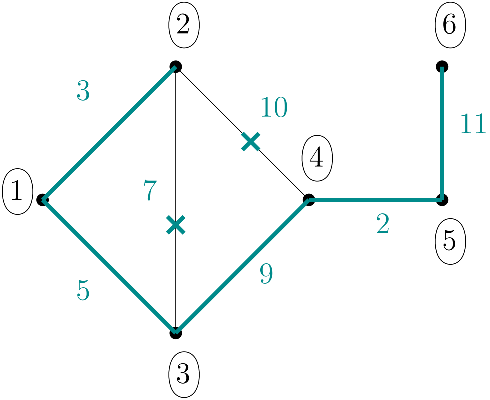
Kruskal avec Union-Find : exemple



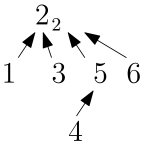
Structure de données Union-Find



Kruskal avec Union-Find : exemple



Structure de données Union-Find



Algorithme de Kruskal

Complexité de **Union Find**

- *Construire* : complexité $O(n)$
- *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**
    $rank(r_y) = rank(r_x) + 1$

Kruskal avec Union-Find

L'efficacité de la détection de cycle lors de l'ajout d'une arête uv dépend maintenant de l'efficacité à trouver le représentant de la composante connexe de u et celle de v, c'est-à-dire remonter jusqu'à leurs racines dans le Union-Find : il faut maîtriser la hauteur de nos arbres.

- $rank(x)$ est en fait la hauteur de la sous-arborescence de racine r
- $\forall x \in V$, si $rank(x) = k$ alors le sous-arbre de racine x a au moins 2^k sommets *(preuve par récurrence sur k)*
- donc $rank(x) \leq log_2 n$

Preuve Union-Find

Si $rank(r) = k$ alors le sous-arbre de racine r a au moins 2^k sommets.

Par récurrence sur k. $k = 0$: au moins 1 sommet, ok.
Hérédité. Soit $k \geq 1$ et r tel que $rank(r) = k + 1$. Montrons que le sous-arbre de racine r a au moins 2^{k+1} sommets. Examinons le moment où le rang de r est passé de k à $k + 1$: c'était lors d'un appel de $union(x, y)$ où les deux représentants $r = r_y$ et r' se sont trouvés de même rang k, et r est devenu le parent de r'.
Par hypothèse de récurrence, r et r' contenaient chacun dans leur arbre au moins 2^k sommets, donc après l'union r contient dans son arbre la somme des deux soit au moins $2^k + 2^k = 2^{k+1}$ sommets.

Algorithme de Kruskal avec Union-Find

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 : $w(x_1 y_1) \leq w(x_2 y_2)$...
$F = \emptyset$
Construire une partition sur V
pour $i = 1$ *à* $|E|$ **faire**
  Si $find(x_i) \neq find(y_i)$
  $F \leftarrow F \cup \{x_i y_i\}$
  $union(x_i, y_i)$
retourner $T = (V, F)$

Arbres couvrants de poids minimum

Une vision plus générale : 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** l'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

Arbres couvrants de poids minimum

Comment trouver une telle arête ?

- **coupe** S : partition de V en $(S, V \setminus S)$
- coupe S **respecte** l'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 9 : 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, G_A n'est pas connexe, donc S existe
    - G connexe donc e existe

Arbres couvrants de poids minimum

1. *tout s'exécute correctement*
   - Dans la boucle, G_A n'est pas connexe, donc S existe
   - G connexe donc e existe
2. *en un nombre fini d'étapes*

Arbres couvrants de poids minimum

1. *tout s'exécute correctement*
   - Dans la boucle, G_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*

Arbres couvrants de poids minimum

1. *tout s'exécute correctement*
   - Dans la boucle, G_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

Arbres couvrants de poids minimum

1. *tout s'exécute correctement*
   - Dans la boucle, G_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 composantes 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

Arbres couvrants de poids maximum

Et si on cherche un arbre couvrant de poids **maximum** ?

Soit H un arbre couvrant
de c-cout maximum $\Leftrightarrow$ de $(-c)$-cout minimum $\Leftrightarrow$ de
$(C - c)$-cout minimum où $C = \max\{c(e) : e \in E(G)\}$.