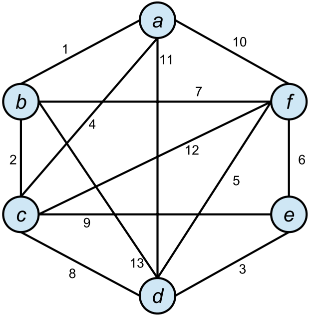
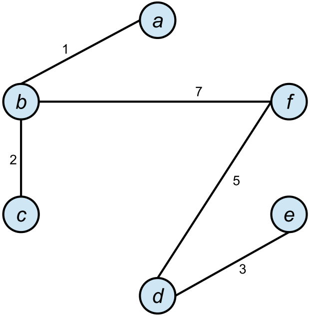
Algorithme de Kruskal

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 $T = (V, F)$

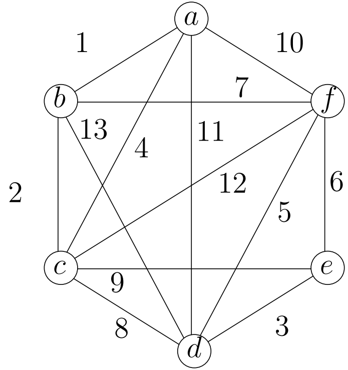
Algorithme de Kruskal : exemple



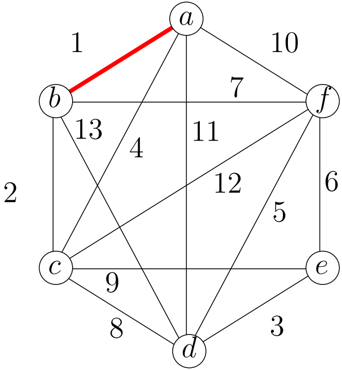


Arbre couvrant de poids $1 + 2 + 3 + 5 + 7 = 18$, c'est l'arbre couvrant de poids minimum renvoyé par l'algorithme de Kruskal (cf détails slide suivant).

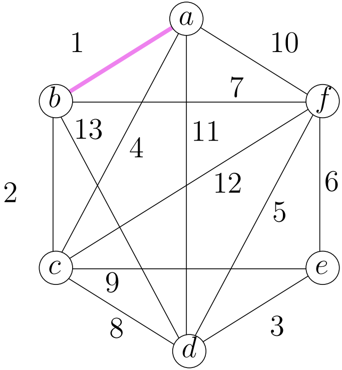
Algorithme de Kruskal : exemple



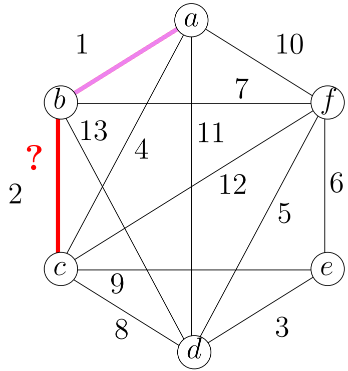
Algorithme de Kruskal : exemple



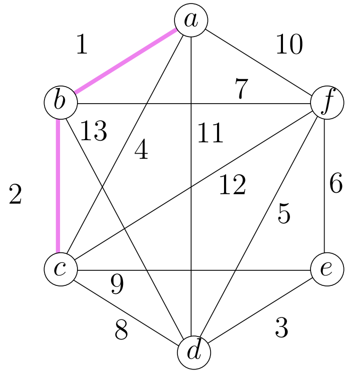
Algorithme de Kruskal : exemple



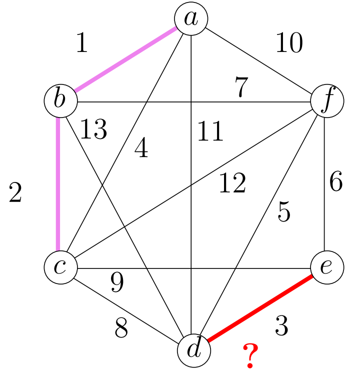
Algorithme de Kruskal : exemple



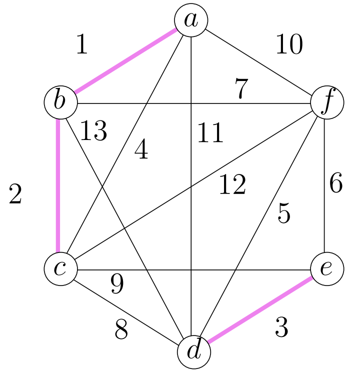
Algorithme de Kruskal : exemple



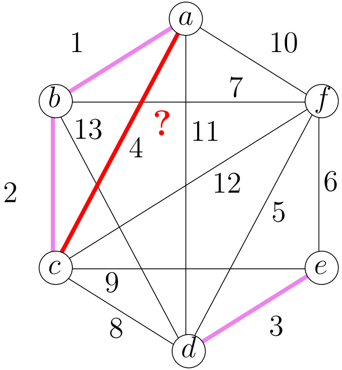
Algorithme de Kruskal : exemple



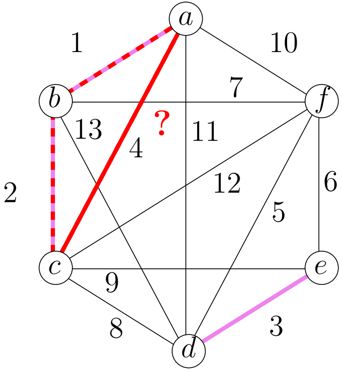
Algorithme de Kruskal : exemple



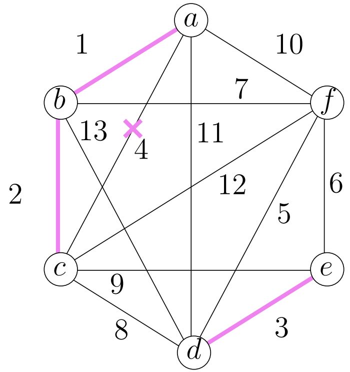
Algorithme de Kruskal : exemple



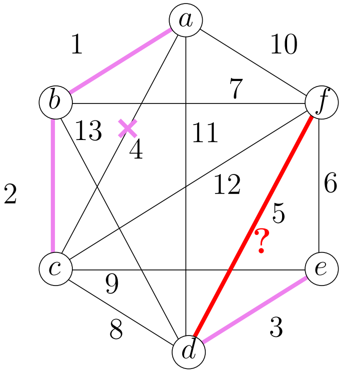
Algorithme de Kruskal : exemple



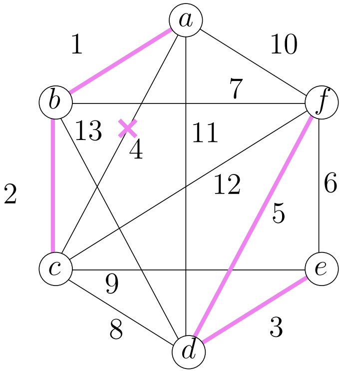
Algorithme de Kruskal : exemple



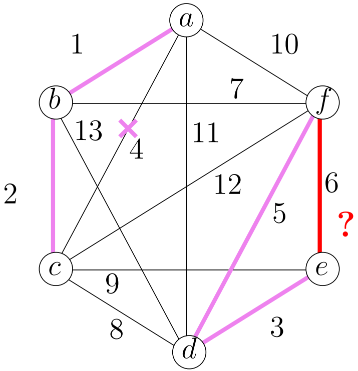
Algorithme de Kruskal : exemple



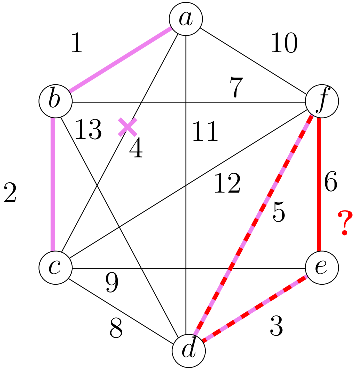
Algorithme de Kruskal : exemple



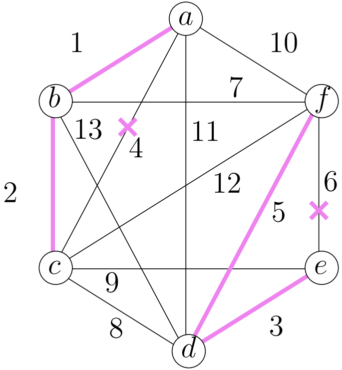
Algorithme de Kruskal : exemple



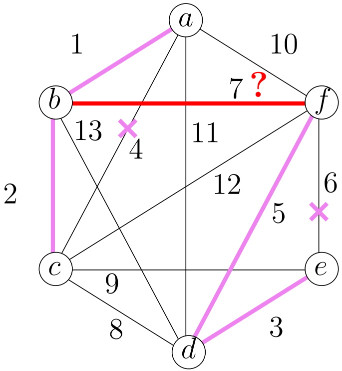
Algorithme de Kruskal : exemple



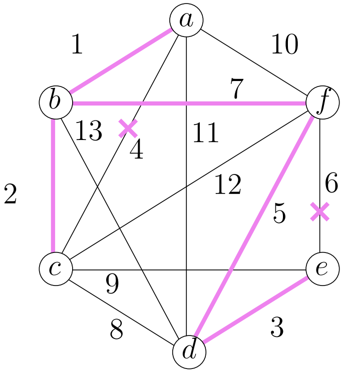
Algorithme de Kruskal : exemple



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

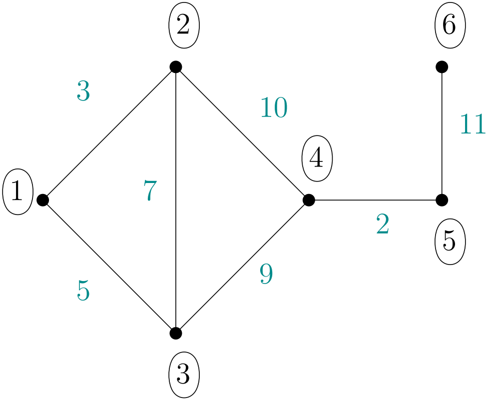
- Construire une partition qui pour chaque élément x crée la classe $\{x\}$.
- $\mathit{find}(x)$ qui renvoie le représentant de la classe contenant x.
- $\mathit{union}(x, y)$ qui fusionne les classes contenant x et y.
  Les paramètres x et y doivent être dans des classes différentes.

Algorithme de Kruskal

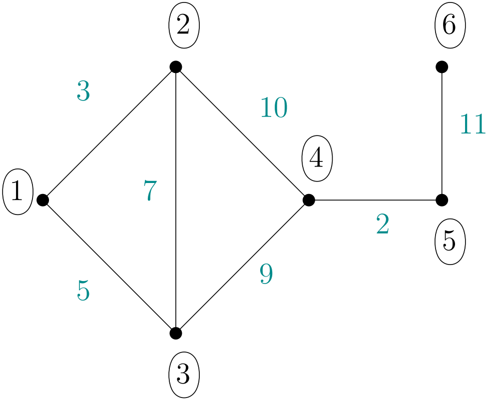
Structure **Union Find**

- 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



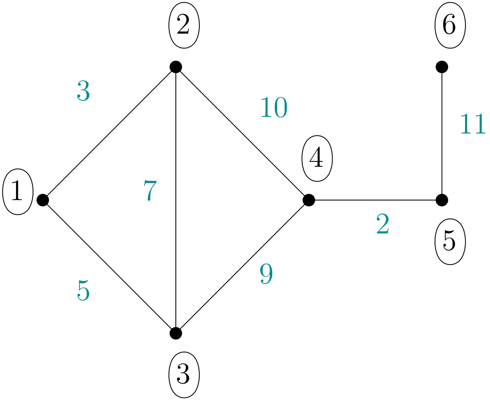
Kruskal avec Union-Find : exemple



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

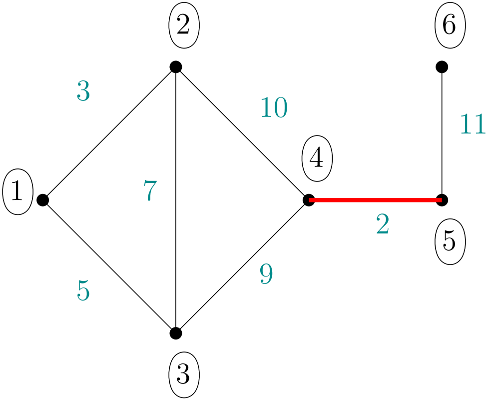


Structure de données Union-Find

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

$v_{rank(v)}$

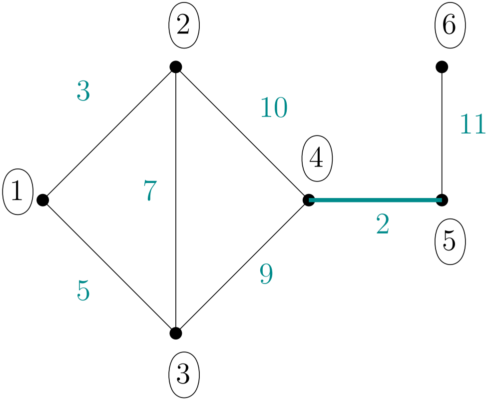
Kruskal avec Union-Find : exemple



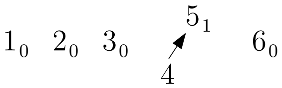
Structure de données Union-Find

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

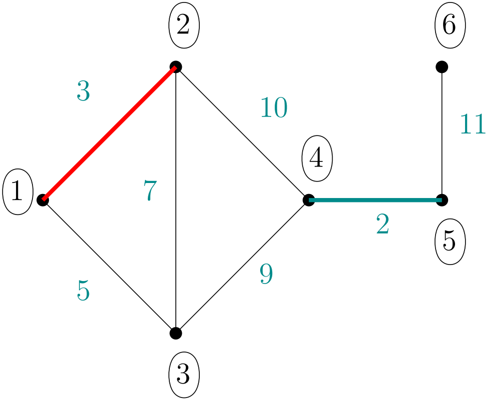
Kruskal avec Union-Find : exemple



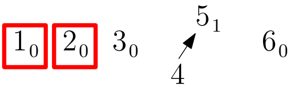
Structure de données Union-Find



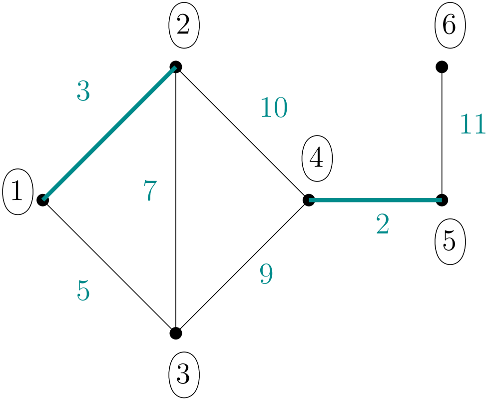
Kruskal avec Union-Find : exemple



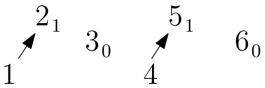
Structure de données Union-Find



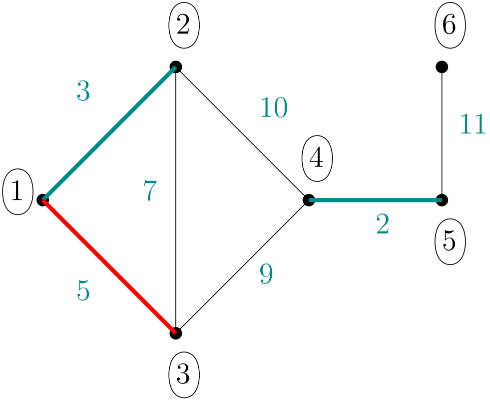
Kruskal avec Union-Find : exemple



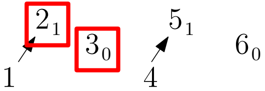
Structure de données Union-Find



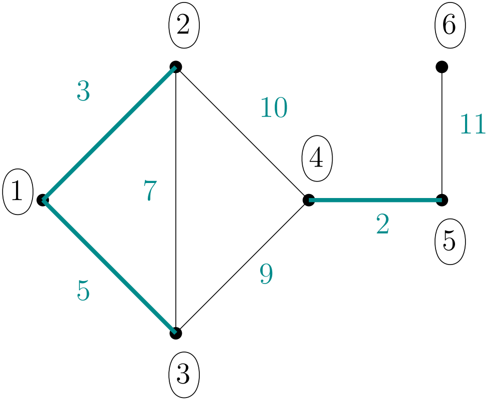
Kruskal avec Union-Find : exemple



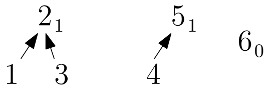
Structure de données Union-Find



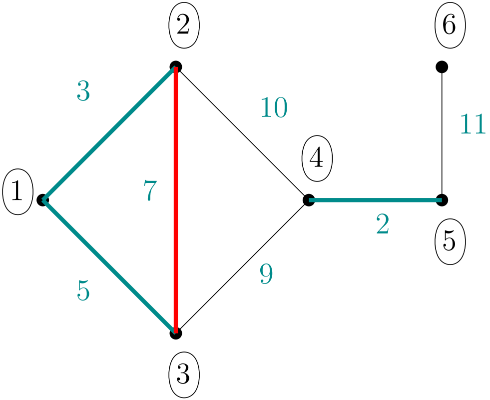
Kruskal avec Union-Find : exemple



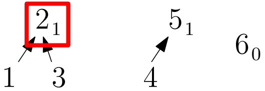
Structure de données Union-Find



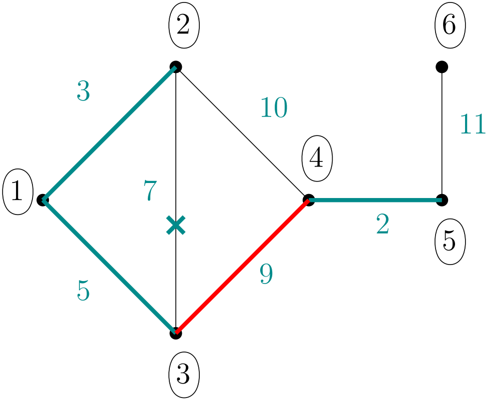
Kruskal avec Union-Find : exemple



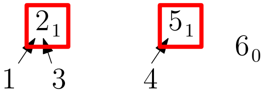
Structure de données Union-Find



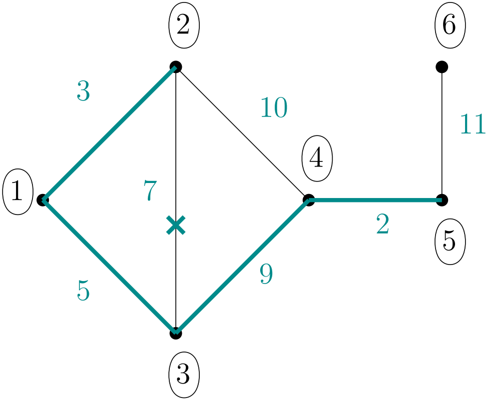
Kruskal avec Union-Find : exemple



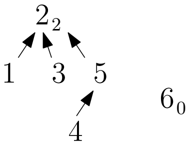
Structure de données Union-Find



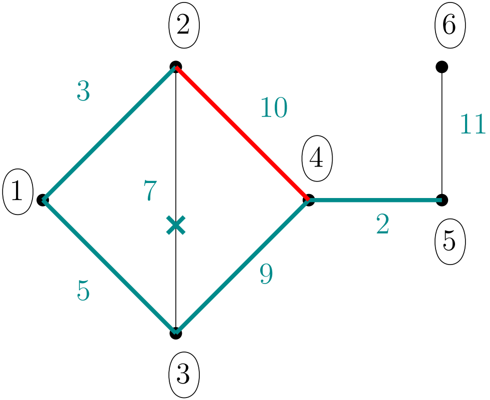
Kruskal avec Union-Find : exemple



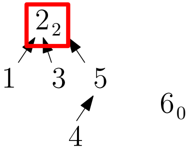
Structure de données Union-Find



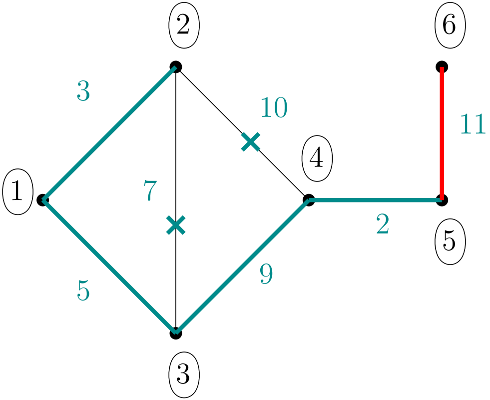
Kruskal avec Union-Find : exemple



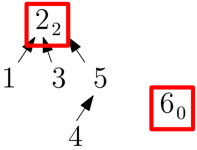
Structure de données Union-Find



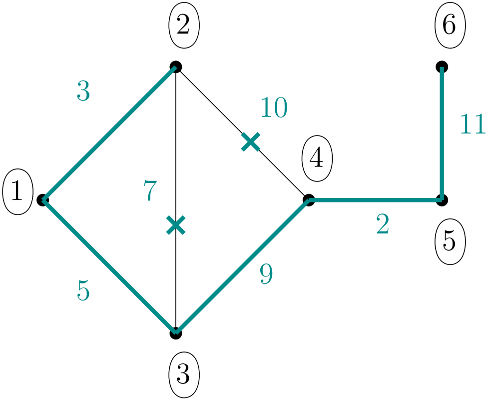
Kruskal avec Union-Find : exemple



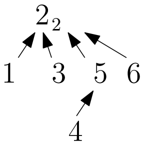
Structure de données Union-Find



Kruskal avec Union-Find : exemple



Structure de données Union-Find



Algorithme de Kruskal

Complexité de **Union Find**

- *Construire* : complexité $O(n)$
- $\mathit{find}(x)$ suit le chemin de x jusqu'à la racine : complexité profondeur de x
- $\mathit{union}(x, y)$: 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 : $w(x_1y_1) \leq w(x_2y_2)$...
$F = \emptyset$
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 $T = (V, F)$