- Soit $G = (V, E)$ un graphe pondéré par $w : E \longrightarrow \mathbb{R}$.
  - Un **arbre couvrant** T de G est un ensemble d'arêtes de G qui forme un arbre et qui contient tous les sommets.
  - Son **poids** $w(T)$ est la somme des poids des arêtes de T.
  - Un arbre couvrant dont le poids est le plus petit possible est appelé un **arbre couvrant de poids minimum**.

- Soit G un graphe connexe pondéré par w.
  Alors G possède un arbre couvrant de poids minimum.

  Preuve : Soit $E = \{w(T) \mid T \text{ est un arbre couvrant de } G\}$.

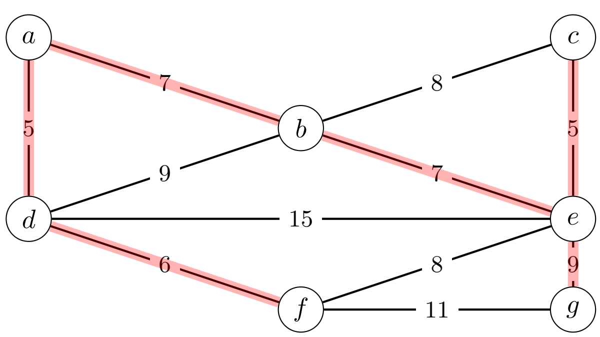
  - $E \neq \emptyset$: l'ensemble des arêtes parcourues dans un parcours de graphe est un arbre couvrant.
  - E est fini.

  Donc E admet bien un minimum.

  Remarque : il n'y a pas forcément unicité d'un arbre couvrant de poids minimum.

- L'algorithme de **Kruskal** est un algorithme glouton permettant d'obtenir un arbre couvrant de poids minimum en sélectionnant les arêtes par poids croissant qui ne créent pas de cycle.

  Exemple : Un graphe avec un arbre couvrant de poids minimum obtenu par l'algorithme de Kruskal (en choisissant, dans l'ordre : ad, ec, df, ab, be, eg).



- L'algorithme de Kruskal renvoie bien un arbre T qui est couvrant et de poids minimum.

  Preuve : Montrons d'abord que T est un arbre couvrant.

  1. T est acyclique : d'après les choix de l'algorithme.
  2. T est connexe et couvrant : soient u et v deux sommets de G. Soit U l'ensemble des sommets accessibles depuis u dans T.
     Supposons $v \notin U$. Comme G est connexe, il existe une arête de G entre U et $V \setminus U$.
     Cette arête aurait dû être ajoutée à T, puisqu'elle ne créée pas de cycle.
     Contradiction: v est donc accessible depuis u dans T.
     Comme c'est vrai pour tout u, v, T est connexe.

  Montrons maintenant que T est de poids minimum.
  Soient T l'arbre obtenu par Kruskal et T^* un arbre de poids minimum.
  Si $T = T^*$, le théorème est démontré.
  Sinon, soit $e^* = \{u, v\} \in T^* \setminus T$. $T^* - e^*$ n'est pas connexe donc possède deux composantes connexes T_u^* (contenant u) et T_v^* (contenant v).
  Comme T est connexe, il existe un chemin C dans T reliant u et v. Ce chemin possède une arête e entre T_u^* et T_v^*.
  Soit $T_2^* = T^* - e^* + e$. T_2^* possède $n - 1$ arêtes et une seule composante connexe : c'est un arbre couvrant.
  De plus, $w(e) \leq w(e^*)$ sinon e^* aurait été ajouté avant e dans l'algorithme de Kruskal. On a donc $w(T^*) \geq w(T_2^*)$.
  On répète ce processus avec T_2^* au lieu de T^*, ce qui nous donne des arbres couvrants T_3^*, T_4^*... jusqu'à obtenir T:

  $$w(T^*) \geq w(T_2^*) \geq w(T_3) \geq ... \geq w(T)$$

  $w(T) \leq w(T^*)$ montre que T est de poids minimum.

- Pour l'implémentation, on suppose l'existence d'une fonction `tri_aretes` qui renvoie la liste des arêtes triées par poids croissant (chaque arête étant un triplet (w, u, v) où w est le poids de l'arête $\{u, v\}$).

```
let chemin (t : int list array) u v =
(* détermine s'il existe un chemin de u à v dans t *)
(* on fait un parcours en profondeur depuis u *)
  let n = Array.length t in
  let visited = Array.make n false in
  let rec aux u =
    if not visited.(u) then (
      visited.(u) <- true;
      List.iter aux t.(u)
    ) in
  aux u;
  visited.(v)

let kruskal (g : int list array) =
    let n = Array.length g in
    let t = Array.make n [] in
    List.iter (fun (w, u, v) ->
        if not (chemin t u v) then (
            t.(u) <- v :: t.(u);
            t.(v) <- u :: t.(v);
    ) tri_aretes g;
    done; t
```

Remarque : on peut aussi utiliser une file de priorité au lieu du tri
Complexité : $O(n \log(n))$ (pour le tri ou les n appel la fonction d'extraction d'un tas).