Algorithme de Boruvka et union find

On rappelle le principe de l'algorithme de Boruvka :
Soit $G = (S, A)$ un graphe connexe, on initialise $F = (S, \emptyset)$.
Tant que F n'est pas connexe :
Trouver toutes les arêtes sures de F (une par CC) et les insérer à F.

1. Rappeler la définition d'une arête sure pour un ensemble de sommets $C \subset S$ d'un graphe $G = (S, A)$.

   Une solution pour gérer les composantes connexes est d'utiliser une structure union find.
   Considérons la structure suivante :

   ```
   type uf = {link : int array; rank : int array; mutable nb : int};;
   ```

   où le champs `nb` désigne le nombre de composantes connexes de la partition.

2. Ecrire une fonction `create : int -> uf` qui initialise une partition pour un ensemble de taille n, $\{0, \ldots, n-1\}$ où chaque élément forme sa propre classe.

3. Ecrire une fonction `find : uf->int->int` qui renvoie un représentant canonique d'un élément dans la partition `uf`, cette fonction utilisera la compression des chemins.

4. Ecrire une fonction `union : uf->int->int->unit` qui modifie la partition afin de réunir les classes des éléments `i` et `j`. Cette fonction optimisera les rangs.

   On définit le type `graphe` (graphe pondéré) par :

   ```
   type graphe = (int*int) list array
   ```

   On considère le code suivant :

   ```
   let mystere uf g=
     let n = Array.length g in
     let a_s = Array.make n (-1,max_int,-1) in
     for i = 0 to n-1 do
       let ri = find uf i in
       let rec aux liste = match liste with
         |[]->()
         |(a,p)::suite when (ri = find uf a ) -> aux suite;
         |(a,p)::suite  -> let x,y,z = a_s.(ri) in if (y > p) then a_s.(ri)<- (a,p,i); aux suite;
       in
       aux g.(i);
     done;
     a_s;;
   ```

5. Que fait cette fonction ?

6. Ecrire une fonction récursive `boruv : graphe->graphe->uf->graphe` qui prend en entrée un graphe connexe `g`, une foret `f` qui est un sous graphe de l'arbre couvrant de poids minimal de `g` contenant tous les sommets de `g` ainsi qu'une partition de l'ensemble des sommets de `g` qui coincide avec les composantes connexes de `f` et qui renvoie l'arbre couvrant de poids minimal de `g`.

7. Ecrire une fonction `boruvka : graphe -> graphe` qui renvoie l'arbre couvrant de poids minimal d'un graphe passé en entrée.