TP — Colle 11 : Isomorphisme d'arbres

L'objet de ce TP est d'écrire une fonction permettant de tester si deux arbres sont ou non isomorphes.

On considère le type arbre suivant :

```
type arbre = F of int | Noeud of int*arbre list;;
```

Voici un exemple :

```
(Noeud(0,[Noeud(1,[Noeud(3,[F(6);F(7);F(8)])]);Noeud(2,[Noeud(4,[F(9);F(10);F(11)]);
Noeud(5,[Noeud(12,[F(13);F(14)])])])]))
```

Dans tout le problème, chaque sommet de l'arbre (i.e. chaque noeud et chaque feuille) est indexé par son rang dans l'énumération "étage par étage" (parcours en largeur) en numérotant les sommets à partir de 0.

On dit que deux arbres a_1 et a_2 sont isomorphes s'il existe une bijection σ de l'ensemble des sommets de a_1 (noté V_1) dans l'ensemble des sommets de a_2 (noté V_2) telle que la racine de a_1 soit envoyée sur la racine de a_2 et:

$$\forall (i,j) \in V_1^2, i \text{ et } j \text{ sont reliés dans } a_1 \text{ ssi } \sigma(i) \text{ et } \sigma(j) \text{ sont reliés dans } a_2$$

1 Fonctions de base utiles

▷ **Question 1.** Écrire une fonction **nb_sommets** qui compte le nombre de sommets (noeuds plus feuilles d'un arbre). ◁

▷ **Question 2.** Écrire une fonction **etage** qui parcourt l'arbre étage par étage (parcours en largeur) et renvoie un tableau de couples de la forme `int * arbre` tel que la case i du tableauf contient le numéro de l'étage du sommet i ainsi que l'arbre enraciné en i. Pour simuler une file, on pourra se contenter d'utiliser une liste avec insertion à la fin ou reprendre une des implémentations vues dans le TP sur les files. ◁

▷ **Question 3.** Écrire une fonction **tri** qui trie une liste d'éléments prise en entrée selon une relation d'ordre compare, elle aussi prise en entrée. ◁

▷ **Question 4.** Écrire une fonction **comparuple** qui prend en argument deux listes ℓ_1 et ℓ_2 et qui renvoie `true` ssi ℓ_1 est avant ℓ_2 dans l'ordre lexicographique. ◁

▷ **Question 5.** Écrire une fonction **egalensemble** qui prend en argument deux listes de listes ℓ_1 et ℓ_2 et qui renvoie `true` ssi ℓ_1 est égale à ℓ_2. ◁

2 Présentation de l'algorithme

L'algorithme va construire une numérotation des sommets des deux arbres passés en entrée. La numérotation obtenue par l'algorithme correspondra à un isomorphisme entre les arbres (deux sommets isomorphes auront le même numéro). On conservera la numérotation "étage par étage" tout le long de l'algorithme pour pouvoir identifier facilement les sommets manipulés tout en construisant une nouvelle numérotation répondant au problème. La nouvelle numérotation est initialisée en mettant 0 à tous les sommets des arbres puis elle est mise à jour étage par étage en commençant par le niveau le plus éloigné de la racine.

1. La première étape de l'algorithme consiste à numéroter les sommets de chaque graphe dans l'ordre étage par étage.
2. On initialise ensuite un nouveau tableau de numérotation des sommets pour chaque arbre passé en entrée. On initialise tous les sommets à 0.
3. A chaque étape (i.e. pour chaque étage), en supposant que l'étage $i+1$ a déjà été mis à jour, l'algorithme va mettre à jour les sommets de l'étage i de la manière suivante :
   (a) On commence par associer à chaque sommet de l'étage i une liste construite de la façon suivante :
      i. si c'est une feuille alors on associe la liste vide à ce sommet.
      ii. si ce n'est pas une feuille alors on associe la liste triée des numéros des voisins de ce sommet situés à l'étage $i+1$ (c'est la liste de ses fils).

(b) On considère alors la liste triée (par ordre lexicographique) des listes des sommets de l'étage i. On a donc construit deux listes de listes, une pour chaque arbre, et on vérifie que ce sont les mêmes. Si ce n'est pas le cas alors les arbres ne sont pas isomorphes et si c'est bien le cas, alors on met à jour la numérotation des sommets de l'étage i. Pour cela, on associe à chaque sommet le rang (selon l'ordre lexicographique) de sa liste parmi toutes les listes des sommets de l'étage i. On peut alors passer à l'étage $i-1$.

4. Si à la fin on accepte les deux racines alors les arbres sont isomorphes et la numérotation donne un isomorphisme.

▷ **Question 6.** Appliquer l'algorithme sur l'exemple donné en début de sujet. ◁

▷ **Question 7.** Montrer que deux graphes enracinés en r et r' sont isomorphes ssi il existe une bijection γ de l'ensemble des fils de r vers l'ensemble des fils de r' telle que pour chaque fils de r, noté i alors les arbres enracinés en i et en $\gamma(i)$ sont isomorphes. ◁

▷ **Question 8.** En déduire la correction de l'algorithme décrit ci-dessus. ◁

3 Implémentation de l'algorithme

▷ **Question 9.** Écrire une fonction **boucle** de signature : `'a->'b array->('a*arbre)array->'b list list` qui prend en entrée un entier correspondant au numéro de l'étage que l'on est en train de traiter, le tableau correspondant à la numérotation courante des étages et le tableau obtenu par la fonction **etage** de la première section et renvoie la liste des listes de l'étage i. ◁

▷ **Question 10.** Écrire une fonction **bouclesource**, similaire à la fonction **boucle** qui renvoie, associé à chaque liste de la liste, le numéro du sommet d'où elle provient :
`bouclesource : 'a ->'b array -> ('a * arbre) array ->('b list * int) list` ◁

▷ **Question 11.** Écrire une fonction **maj_num** qui prend en entrée le tableau de numérotation, la liste des couples renvoyée par la fonction **bouclesource** pour l'étage i et qui met à jour la numérotation des sommets de l'etage i en fonction de leur rang dans la liste renvoyée par la fonction **bouclesource**.
`maj_num : int array -> ('a list * int) list -> unit` ◁

▷ **Question 12.** Écrire la fonction **iso** qui renvoie `true` ssi les deux arbres passés en entrée sont isomorphes.
`iso : arbre -> arbre -> bool` ◁