TP : Files de priorité persistantes

Nous avons vu en cours des files de priorités impératives, qui ne permettent pas de garder l'historique des opérations.

Nous allons donc implémenter des files de priorité fonctionnelles, à l'aide d'arbres.

Nos arbres seront définis par le type suivant :

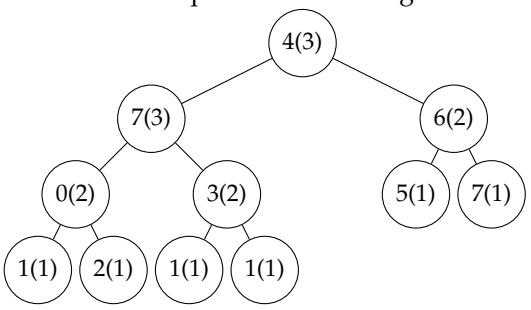
```
type 'a data = {Priorite: int; Valeur: 'a}

type 'a arbre =
|Nil
|Noeud of 'a arbre * 'a * 'a arbre * int
```

Ce dernier entier sera le *rang* du nœud : il représente la distance entre le nœud et la feuille vide la plus à droite.

EXEMPLE

On a écrit entre parentèses les rangs des nœuds de l'arbre suivant.



On appellera rang d'un tas le rang de se racine ; par convention, le tas vide aura un rang nul.

1. Écrire une fonction `rang : 'a arbre -> int` qui calcule le rang d'un tas.
2. Écrire une fonction `faire_tas` qui, étant donnés une valeur `v` et sa priorité `p` et deux tas `a` et `b` dont tous les nœuds ont une priorité inférieure, crée un tas de racine `p,v` et dont le fils droit est le tas ayant le plus petit rang.
3. Montrer que tout tas t créé uniquement avec `faire_tas` a un rang inférieur à $\lg(|t|+1)$.
   On pourra montrer que pour tous $m, n \in \mathbb{N}$, $\lg(\min(m,n)+1)+1 \leqslant \lg(m+n+2)$.

Si on peut avoir des algorithmes en $O(\mathrm{rg}(t))$, on aura donc des algorithmes performants.

4. L'opération la plus importante est la fusion de deux tas `a` et `b`.
   En supposant que la racine de `a` `pv` est de priorité supérieure à celle de `b`, on fusionne le fils droit de `a` avec `b`, ce qui donne un tas `t`, puis on crée le tas de racine `pv` ayant pour fils le fils gauche de `a` et `t`.
5. Montrer que cette fonction renvoie bien un tas, et montrer que la complexité est inférieure à la somme des rangs des deux arbres.
6. Écrire alors les fonctions d'insertion et d'extraction de l'élément de plus grande priorité.
7. Une fonction d'augmentation de priorité serait compliquée. Une solution est d'ajouter à nouveau la valeur avec sa nouvelle priorité, créant un doublon. L'écrire quand même.
8. Écrire une fonction de tri par tas d'une liste.