Algorithmique des structures de données arborescentes

Tp Noté

Le barème est donné à titre **indicatif**. Le sujet comporte 2 pages.

Consignes

1. L'ensemble des fonctions sera sauvegardé dans un fichier de nom `tp-note.ml`.
2. Indiquez vos nom et prénom et votre numéro de groupe en commentaire dans le fichier `tp-note.ml`.
3. Pensez à sauver régulièrement votre travail.
4. À la fin du tp, déposez votre fichier `tp-note.ml` à l'endroit prévu sur Moodle.
5. Créez une archive `<NOM>-<PRENOM>-<GROUPE>.zip` à partir de votre fichier et envoyez votre archive par mail à votre responsable de TD :
   A3 : stefka.gueorguieva@u-bordeaux.fr
   A4 : frederique.carrere@u-bordeaux.fr

Dans les exercices suivants, on utilise le type `'a btree` suivant :

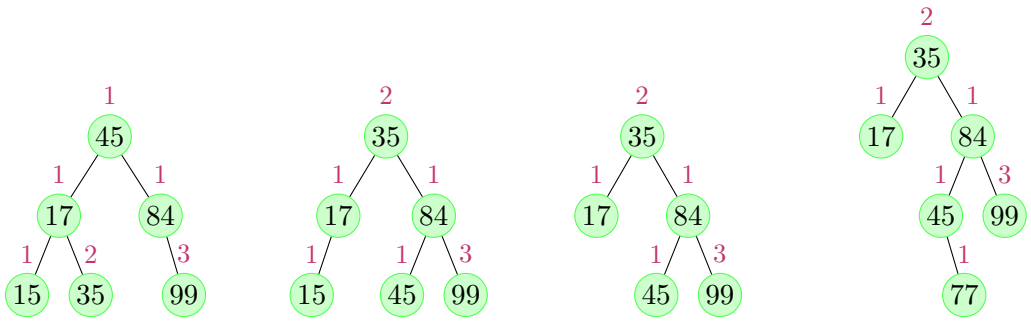
```
type 'a btree = Empty | Node of 'a * 'a btree * 'a btree
```

La complexité dans le pire des cas des fonctions proposées sera prise en compte pour leurs évaluations.

Exercice 1 (3pts)
Etant donné un arbre binaire de recherche t et une liste d'éléments $[e_1\,;e_2\,;\ldots;e_n]$, écrire une fonction `val bst_list_search : 'a btree -> 'a list -> bool = <fun>` qui retourne *true* si la liste d'éléments (de gauche à droite) correspond à la suite des éléments rencontrés lors de la recherche de e_n dans t (e_1 sera donc l'élément à la racine de t). La fonction retourne true si la liste est vide.
Par exemple, si t_4 est l'arbre binaire de recherche illustré sur la Fig. 1(d), l'appel à la fonction `bst_list_search t4 [35;84;45]` va renvoyer *true* et l'appel à la fonction `bst_list_search t4 [35;84;50;77]` va renvoyer *false*.

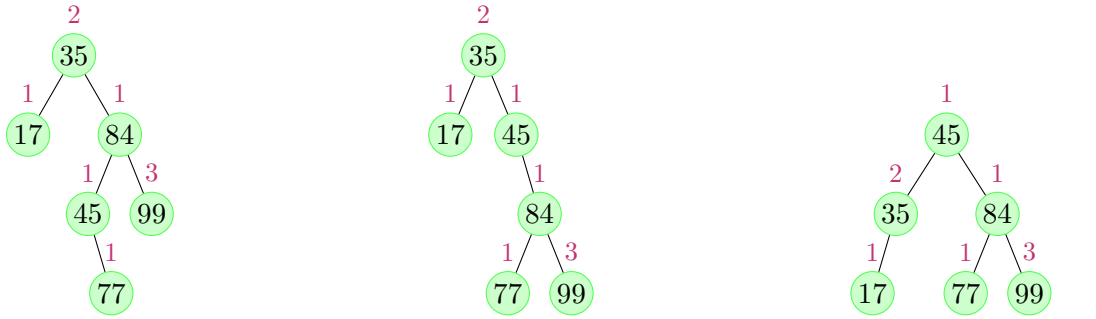
Exercice 2 (4pts)
Un arbre binaire de recherche A **est contenu** dans un arbre binaire de recherche B si tous les éléments de A sont contenus dans B.
Écrire une fonction `val bst_is_contained : 'a btree -> 'a btree -> bool = <fun>` qui teste si un arbre binaire de recherche est contenu dans un autre.
Par exemple, t_3 est contenu dans t_1 et l'appel à la fonction `bst_is_contained t3 t1` va renvoyer *true*.
Les ABRs t_1 et t_3 sont illustrés resp. sur la Fig. 1(a) et la Fig. 1(c).
Par contre, t_4 n'est pas contenu dans t_1 et l'appel à la fonction `bst_is_contained t4 t1` va renvoyer *false*, t_4 est illustré sur la Fig. 1(d).

Exercice 3 (3pts)
Deux arbres binaires de recherche (ABR) sont dit **équivalents** s'ils contiennent les mêmes éléments.
Écrire une fonction `val bst_eq : 'a btree -> 'a btree -> bool = <fun>` qui teste si deux arbres binaires de recherche sont équivalents.
Ainsi par exemple, t_1 est équivalent à t_2 et l'appel à la fonction `bst_eq t1 t2` va renvoyer *true*.
Les ABRs t_1 et t_2 sont illustrés resp. sur la Fig. 1(a) et la Fig. 1(b).



(a) t_1 (b) t_2 (c) t_3 (d) t_4

FIG. 1 : Exemples : t_1 est équivalent à t_2, t_3 est contenu dans t_1, t_4 n'est pas contenu dans t_1, t_3 et t_4 ont des domaines plus petits que t_1.



(a) t_4 (b) (c) t_5

FIG. 2 : Étapes de transformations de t_4 à la recherche auto-adaptative de 45 : (a) A faire une rotation à droite autour du noeud qui contient 84 (b) A faire une rotation à gauche autour du noeud qui contient 35 (c) L'arbre t_5 résultat de la recherche auto-adaptative de 45 dans t_4.

Exercice 4 (4pts)

Un arbre binaire de recherche A est dit **de domaine plus petit** qu'un arbre binaire de recherche B si le plus petit élément de A est supérieur ou égal au plus petit élément de B, et si le plus grand élément de A est inférieur ou égal au plus grand élément de B.

Écrire une fonction `val bst_smaller_domain : 'a btree -> 'a btree -> bool = <fun>` qui teste si un arbre binaire de recherche est de domain plus petit qu'un autre.

Par exemple, t_3 et t_4 ont des domaines plus petits que t_1, et les appels `bst_smaller_domain t3 t1` et `bst_smaller_domain t4 t1` vont renvoyer *true*.

Les ABRs t_1,t_3 et t_4 sont illustrés resp. sur la Fig. 1(a), la Fig. 1(c) et la Fig. 1(d).

Exercice 5 (6pts)

On considère la méthode suivante de **recherche auto-adaptative** dans un arbre binaire de recherche. Soit e un élément présents dans l'arbre. Après toute recherche de l´ élément e, si e est dans la racine on ne fait rien. Sinon, si e est fils gauche de son père p on effectue une rotation à droite autour de p, sinon on effectue la rotation symétrique, une rotation à gauche.

Ainsi l'élément recherché est remonté à la racine de l'arbre par une suite de rotations.

Écrire une fonction `val bst_adaptative_search : 'a btree -> 'a -> 'a btree = <fun>` qui effectue une recherche auto-adaptative.

Par exemple, la recherche `bst_adaptative_search t4 45` va produire l'arbre t_5 de la Fig. 2(c).