

## TD n°2

### Arbres binaires de recherche

Le type de donnée “**arbre**” sera utilisé pour indiquer l’ensemble des arbres binaires étiquetés (ABE) par des clés de type **entier**. La taille d’un ABE  $a$  est par définition le nombre de clés contenues dans  $a$ . Nous rappelons qu’un arbre binaire de recherche (ABR)  $a$  est un ABE tel que tout nœud interne  $b$  de  $a$  contient une clé

- supérieure ou égale à toutes les clés contenues dans le sous-arbre gauche de  $b$ ;
- inférieure strictement à toutes les clés contenues dans le sous-arbre droit de  $b$ .

**Exercice 1** Peut-on utiliser la procédure suivante pour tester si un ABE  $a$  est un ABR ?

```
1 Fonction estABR( $a$  : arbre) : booléen
2 début
3   si (estVide( $a$ )) alors
4     | retourner vrai;
5   si (estVide(G( $a$ ))) alors
6     |  $gauche := clé(a)$ ;
7   sinon
8     |  $gauche := clé(G(a))$ ;
9   si (estVide(D( $a$ ))) alors
10    |  $droite := clé(a)$ ;
11  sinon
12    |  $droite := clé(D(a))$ ;
13  retourner ( $gauche \leq clé(a) \leq droite \wedge estABR(G(a)) \wedge estABR(D(a))$ );
14 fin
```

Sinon écrivez une fonction qui teste si un ABE  $a$  est un ABR.

**Exercice 2** [Accès au  $k$ -ème élément] Soit  $a$  un ABR.

**Question 1.** Proposez une méthode pour trouver le  $k$ -ème plus grand élément de  $a$  en utilisant :

- une procédure **VisiteGRD**( $a$  : arbre) qui implemente un parcours infixe de l’arbre  $a$
- un tableau d’entiers  $T$ .

Quelle est la complexité de cette méthode en fonction de la taille de  $a$  ?

**Question 2.** Proposez une fonction **RecherchePos**( $a$  : arbre,  $k$  : entier) : booléen  $\times$  entier, basé sur un parcours infixe de l’arbre  $a$ , telle que si  $(b, n) = RecherchePos(a, k)$  alors

- si la taille de  $a$  est inférieure à  $k$  alors  $b = faux$  et  $n$  est la taille de  $a$ ;
- si la taille de  $a$  n’est pas inférieure à  $k$  alors  $b = vrai$  et  $n$  est la valeur de la  $k$ -ème clé.

Quelle est la complexité de cette méthode ?

**Question 3.** En supposant qu’on dispose d’une fonction **taille**( $a$  : arbre) : entier calculant la taille d’un arbre  $a$  en  $\Theta(1)$ , proposez une fonction **RecherchePos**( $a$  : arbre,  $k$  : entier) : arbre telle que si  $b = RecherchePos(a, k)$  alors

- si  $\text{taille}(a) < k$ , alors  $b = \text{ArbreVide}()$  ;
  - si  $\text{taille}(a) \geq k$ , alors  $b$  est le sous-arbre de  $a$  tel que  $\text{clé}(b)$  est la  $k$ -ème clé dans l'arbre  $a$ .
- Évaluez la complexité de la procédure en fonction de la hauteur de l'arbre  $a$ .

**Question 4.** En acceptant de modifier légèrement la structure des ABR (en ajoutant des informations supplémentaires dans les noeuds de l'arbre), pouvez-vous proposer une fonction  $\text{taille}(a : \text{arbre})$  : entier ayant complexité  $\Theta(1)$  ? Vérifiez que votre modification ne change pas la complexité des fonctions d'insertion et de suppression d'un nœud dans un ABR.

**Exercice 3 [Recherche du successeur]** Modifiez légèrement la définition d'un ABR de sorte que toutes les clés soient distinctes. On appelle *strict* un ABR qui satisfait cette définition. Soit  $a$  un ABR strict. Le successeur d'un nœud  $x$  (s'il existe) est par définition le nœud  $y$  ayant la plus petite clé parmi celles plus grandes que  $\text{clé}(x)$ .

**Question 1.** Montrer rigoureusement que dans  $a$  :

- (a) l'élément minimum se trouve à un nœud sans fils gauche.
- (b) si un nœud a un fils droit, son successeur est le minimum de son sous-arbre droit. En déduire que si un nœud a un fils droit, alors son successeur n'a pas de fils gauche.
- (c) si un nœud n'a pas de fils droit, son successeur, s'il existe, est le premier de ses ancêtres dont le fils gauche est aussi l'un de ses ancêtres.

**Question 2.** Écrire une fonction  $\text{succ}(b : \text{arbre}) : \text{arbre}$  qui retourne le sous-arbre de  $a$  dont la racine est le successeur de  $b$ . Quelle est la complexité en temps de cet algorithme ?