
Projet d'Algorithmie
(Rapport d'expérimentations
sur les ABR et les AVL)

Auteur(s) : Florian Legendre



Légendes et Abréviations utilisées

```
1 Ceci est du code source.  
2 Selon les langages, différents mots seront colorés selon  
3 si ce sont des mots clefs ou non (comme int, char, etc.).
```

Listing 1 – Exemple de code source

```
Ceci est un formatage automatique Latex d'un texte copié-collé  
directement depuis un terminal Bash ayant valeur de capture  
d'écran. La coloration correspond à une coloration quelconque  
d'un terminal Bash (les chemins étant habituellement coloré et  
le nom de l'utilisateur aussi comme crex@crex:~$ ...)
```

Listing 2 – Exemple d'une pseudo capture d'écran d'un terminal de commande (type eshell ou Bash)

Table des matières

I	Exercice 1 - Arbres binaires de recherche	3
0.1	Première question : générer des ABR aléatoirement	4
0.2	Deuxième question : Calculer la moyenne des déséquilibres	5
0.3	Troisième question : Sous-suites ordonnées et moyenne des déséquilibres	6
II	Exercice 2 - Arbres AVL	8
1	Implantation d'un module Avl	9
1.1	Implantation des opérations de rotation	9
1.2	Implantation de l'opération "rééquilibrer"	10
1.3	Implantation des opérations d'insertion et de suppression dans un arbre AVL	11
1.4	Opération de recherche dans un AVL	12
2	Expérimentations avec les arbres AVL	13
2.1	Génération aléatoire d'AVL et complexités des opérations implantées	13
2.2	Sous-suites ordonnées et nombre de rotations dans un AVL	14

Première partie

Exercice 1 - Arbres binaires de recherche

Première question : générer des ABR aléatoirement

Question : À partir du module sur les ABR que vous avez réalisé au TP4 et en utilisant la fonction `Random.self_init` du module `Random` qui permet d'initialiser un générateur de nombres aléatoires et la fonction `Random.int` borne qui donne un nombre aléatoire compris entre 0 et borne - 1 (le paramètre borne doit être inférieur à 2^{30}), écrivez une fonction `bst_rnd_create` qui crée un arbre binaire de recherche.

Réponse :

```
1 let bst_rnd_create(bound, treeSize : int * int) : int bst =
2
3   Random.self_init();
4
5   let empty_tree : int bst = empty() in
6   let randABR : int bst ref = ref empty_tree in
7
8   for i=1 to treeSize do
9     let randInt : int = Random.int bound in
10    randABR := bst_linsert(!randABR, randInt);
11  done;
12
13  !randABR
14  ;;
```

Listing 3 – Code source de génération aléatoire d'arbres binaires de recherche

```
# show_int_btree(bst_rnd_create(100, 10));;
      8
     / \
    1   81
     /  \
    17   97
   /  \
  16   81
 /  \
14  35
   /
  29
- : unit = ()

# show_int_btree(bst_rnd_create(100, 10));;
      38
     /  \
    11   61
   /  \ /  \
  6   32 45 66
 /   \   \
1    62 72
- : unit = ()
```

Listing 4 – Exemple de résultat du code ci-dessus

Deuxième question : Calculer la moyenne des déséquilibres

Question : Le déséquilibre d'un arbre est la différence entre la hauteur de son fils gauche et la hauteur de son fils droit. À l'aide de la fonction `bst_rnd_create` estimez le déséquilibre moyen des arbres binaires de recherche construits à partir de suites de nombres entiers aléatoires. Pour avoir des estimations significatives, il est nécessaire de répéter un grand nombre de fois l'estimation de la moyenne faite sur un grand nombre d'arbres.

Réponse :

```
1 let desequilibre(tree : 'a t_btree) : int =
2   height(lson(tree)) - height(rson(tree))
3 ;;
4
5 let avgDesequilibre(sampleSize, treeSize : int * int) : float =
6
7   let sum : float ref = ref 0. in
8
9   for i=1 to sampleSize do
10    sum := !sum +. float_of_int(desequilibre(bst_rnd_create(100, treeSize))
11    )
12   done;
13
14   !sum /. float_of_int(sampleSize)
15 ;;
```

Troisième question : Sous-suites ordonnées et moyenne des déséquilibres

Question : Reprenez le même processus d'estimation du déséquilibre moyen d'arbres binaires de recherche, mais cette fois-ci avec des suites de nombres entiers qui contiennent des sous-suites ordonnées de longueurs variables. Les longueurs de ces sous-suites pourront être choisies aléatoirement ou bien de longueur fixe ou encore de longueurs croissantes ou décroissantes.

Réponse :

```
1 let createSeries(len : int) : int list =
2
3   Random.self_init();
4
5   if (len <= 0)
6   then
7     (
8       let randLowerBound : int = Random.int 1000 in
9       let res : int list ref = ref [] in
10
11      for i=1 to Random.int 100 do
12        res := (randLowerBound+(i-1)) :: !res
13      done;
14
15      !res
16    )
17  else
18    (
19      let randLowerBound : int = Random.int 1000 in
20      let res : int list ref = ref [] in
21
22      for i=1 to len do
23        res := (randLowerBound+(i-1)) :: !res
24      done;
25
26      !res
27    )
28  ;;
29
30
31
32 let bst_rndSeries_create(treeSize, seriesLen : int * int) : int bst =
33
34   Random.self_init();
35
36   let empty_tree : int bst = empty() in
37   let randABR : int bst ref = ref empty_tree in
38   let deltaSizes : int ref = ref treeSize in
39
40   while (!deltaSizes != 0) do
41     let rndSeries : int list = createSeries(seriesLen) in
42
```

```

43   for i=0 to List.length(rndSeries)-1 do
44       if(!deltaSizes > 0)
45           then (
46               randABR := bst_lininsert(!randABR, List.nth rndSeries i);
47               deltaSizes := !deltaSizes - 1;
48           )
49       done;
50
51   done;
52
53   !randABR
54 ;;
55
56 let avgSeriesDesequilibre(sampleSize, treeSize, seriesLenMode : int * int *
57     char) : float =
58
59     let sum : float ref = ref 0. in
60
61     for i=1 to sampleSize do
62         match seriesLenMode with
63         | 'r' -> sum := !sum +. float_of_int(desequilibre(bst_rndSeries_create(
64             treeSize, -1)))
65         | 'f' -> sum := !sum +. float_of_int(desequilibre(bst_rndSeries_create(
66             treeSize, 10)))
67         | 'a' -> sum := !sum +. float_of_int(desequilibre(bst_rndSeries_create(
68             treeSize, 1+i)))
69         | 'd' -> sum := !sum +. float_of_int(desequilibre(bst_rndSeries_create(
70             treeSize, sampleSize-i)))
71         | _ -> failwith("Wrong mode for series length... 'r' for random lengths
72             , 'f' for fixed lengths, 'a' for ascending lengths, and 'd' for
73             descending lengths.");
74     done;
75
76     !sum /. float_of_int(sampleSize)
77 ;;

```

Deuxième partie

Exercice 2 - Arbres AVL

Chapitre 1

Implantation d'un module Avl

Implantation des opérations de rotation

Question : Implantez les rotations à partir des axiomes et des exemples fournis.

Réponse :

Implantation de l'opération "rééquilibrer"

Question : Implantez l'opération `reequilibrer` à partir des axiomes et en supposant que le dés-équilibre d'un arbre est stocké à la racine de cet arbre (proposez une manière de réaliser ce stockage à partir du type `'a t_btreen`)

Réponse :

Implantation des opérations d'insertion et de suppression dans un arbre AVL

Question : Implantez les opérations d'insertion et de suppression dans un arbre AVL.

Réponse :

Opération de recherche dans un AVL

Question : Pouvez-vous réutiliser l'opération de recherche du module Bst ? Si ce n'est pas le cas modifiez votre code afin de pouvoir le faire.

Réponse :

Chapitre 2

Expérimentations avec les arbres AVL

Génération aléatoire d'AVL et complexités des opérations implantées

Question : Définissez une fonction `avl_rnd_create` qui crée des arbres AVL à partir de suites d'entiers aléatoires et vérifiez expérimentalement que les opérations de recherche, d'insertion et de suppression ont bien une complexité en $\theta(\log n)$ où n est la taille de l'arbre.

Réponse :

Sous-suites ordonnées et nombre de rotations dans un AVL

Question : En créant des arbres AVL avec des suites de nombres entiers qui contiennent des sous-suites ordonnées de longueur variable, estimez le nombre moyen de rotations qui sont effectuées pour garder l'arbre équilibré. Comment ce nombre évolue-t-il en fonction de la taille de l'arbre ?

Réponse :
