

Les arbres binaires de recherche (ABR)

Mise en garde

La tentation est grande de prendre de recopier tel quel un algorithme trouvé dans le cours ou sur le net. Ce n'est pas une bonne idée !

De même, pour l'examen n'apprenez pas par cœur les algorithmes vus au cours.

Vous devriez, plutôt que de recopier ou mémoriser du *code*, **comprendre** et retenir les *principes moteurs* de l'algorithme et vous **entraîner** à reconstituer celui-ci.

Le document *AlgorithmesABR* reprend les « principes moteurs » des principaux algorithmes sur les ABR.

Exercices obligatoires

A Code Runner

Faites les *codeRunner* sur les ABR. Suivez les niveaux : 1, 2 et puis 3.

Le document *CodeRunner_ABRTestes* donne une visualisation des arbres testés dans ce *codeRunner*.

B Implémentation d'un ABR d'entiers

Vous allez implémenter un arbre binaire de recherche dont chaque nœud contiendra un entier. Dans cet arbre, tout élément situé dans un sous-arbre de gauche devra être inférieur à l'élément racine de cet arbre. Tout élément situé dans un sous-arbre de droite devra lui être supérieur ou égal.

B1 Assurez-vous d'avoir bien compris l'algorithme d'insertion dans un ABR.

Voici le lien d'un programme de démo :

<https://www.cs.usfca.edu/~galles/visualization/BST.html>

Soyez actif.

Par exemple, commencez par dessiner **sur papier** un ABR en partant de l'arbre vide puis en faisant des ajouts (12 5 8 17 2 ...) :

(Nb : ici, on ne s'intéresse pas encore à l'implémentation et aux classes *Noeud* et *Arbre*, représentez un nœud par un simple rond contenant sa valeur)

Ensuite vérifiez votre ABR en utilisant le programme de démo.

B2 Complétez la classe *ABRDEntiers*. Les « objets » contenus dans l'arbre sont des entiers.

La classe *TestABRDEntiers* permet de tester cette classe.

B3 Il existe plusieurs variantes pour l'algorithme de suppression dans un ABR.

Expliquez en français l'algorithme de suppression utilisé dans le programme de démo

<https://www.cs.usfca.edu/~galles/visualization/BST.html>

C Arbre équilibré

C1 Ecrivez la méthode `hauteur()` de la classe *ABRDEntiers*.

La classe *TestABRDEntiers* teste les arbres mis en exemple dans le document de présentation *ABR*.

Exercices défis

B3 Implémentez la méthode `supprime()` de la classe *ABRDEntiers*.

Voici quelques indications :

Algorithme proposé :

Si le nœud qui contient l'entier à supprimer n'a pas de fils droit :

Le nœud est remplacé par son fils gauche (qui pourrait être vide).

Attention c'est donc le parent du nœud à supprimer qui remplace un de ses 2 nœuds.

Sinon (le nœud qui contient l'entier à supprimer a un fils droit)

L'entier à supprimer est remplacé par le plus petit entier retrouvé dans son fils droit et le nœud qui contient ce plus petit entier est supprimé.

Attention, au final, ce n'est pas le nœud qui contient l'entier à supprimer qui est supprimé, cet entier est juste remplacé.

Pour suivre cet algorithme, nous vous proposons de d'abord écrire les méthodes `min()` et `supprimeMin()`. La classe de test permet de tester ces 2 méthodes séparément.

La méthode `min()` peut s'écrire de façon itérative, réfléchissez : où se trouve le plus petit entier dans un arbre?

La méthode `supprimeMin()` est plus simple que la méthode `supprime()`. Le nœud qui contient le plus petit entier n'a pas de fils gauche, réfléchissez !

B4 ❤ La méthode `taille()` de la classe *ABRDEntiers* a été implémentée en $O(N)$. Elle peut s'implémenter en $O(1)$ en ajoutant un attribut `taille`.

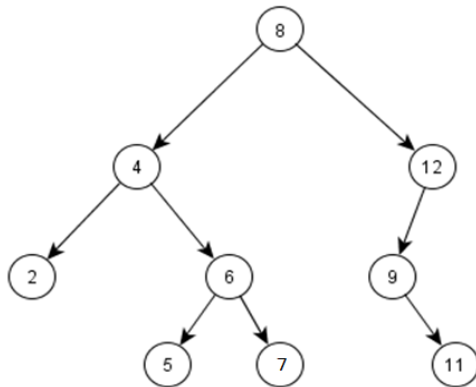
```
public int taille(){  
    return taille;  
}
```

Ajoutez l'attribut `taille` et modifiez les méthodes `insere()`, `supprimeMin()` et `supprime()` afin qu'elles modifient l'attribut en cas de réussite de l'opération.

La classe *TestABRDEntiers* permet de tester cette méthode.

B5 La méthode `toArray()` de la classe *ABRDEntiers* va remplir une table avec les entiers contenus dans l'arbre. La taille logique de cette table doit correspondre à sa taille physique. Les entiers devront y apparaître par ordre croissant :

Exemple :



2	4	5	6	7	8	9	11	12
---	---	---	---	---	---	---	----	----

Dans l'exemple ci-dessus, l'arbre contient 9 entiers, le sous-arbre de gauche en contient 5 et le sous-arbre de droite en contient 3.

L'attribut `taille` permet de trouver facilement l'indice où placer l'entier situé dans la racine de l'arbre.

La méthode `toArray()` construit la table et appelle une méthode **private** à 4 paramètres : un nœud, la table, un indice de début et un indice de fin.

Ces indices permettent de délimiter la partie de la table où placer l'arbre (le sous-arbre) sur lequel la méthode est appelée.

La classe *TestABRDEntiers* permet de tester cette méthode.