Structure de données

Arbre Binaire de Recherche (ABR)

Yannick CHISTEL

Lycée Dumont d'Urville - CAEN

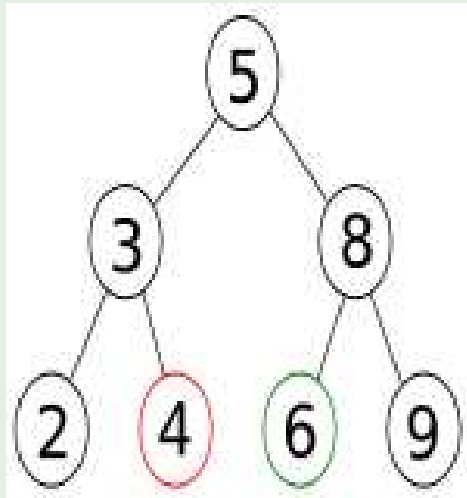
25 janvier 2021

Définition

Un **arbre binaire de recherche** est un arbre binaire tel que pour un nœud fixé de l'arbre, tout nœud de son arbre fils gauche lui est **strictement inférieur** et tout nœud de son arbre fils droit lui est **strictement supérieur**.
De plus, les valeurs sont distinctes, l'arbre ne possède aucun doublon.

Exemple

L'arbre binaire ci-contre est un arbre binaire de recherche (ABR).



La racine de l'arbre a pour valeur 5.
L'arbre fils gauche a pour valeurs 2, 3 et 4 inférieures à 5
L'arbre fils droit a pour valeurs 6, 8 et 9 supérieurs à 5

On remarque :

Le nœud de valeur 4 est dans un arbre fils droit du nœud de valeur 3 et dans un arbre fils gauche du nœud de valeur 5.

Le nœud de valeur 6 est dans un arbre fils gauche du nœud de valeur 8 et dans un arbre fils droit du nœud de valeur 5.

Objet ABR et Nœud

L'implémentation d'un arbre binaire peut se faire avec des listes, dictionnaires ou par programmation objet.

Nous faisons le choix de classes d'objet pour implémenter nos ABR.

- La classe Nœud instancie un objet Nœud avec 3 attributs : valeur, gauche, droit
- La classe ABR instancie un objet arbre avec un seul attribut racine. La racine est soit vide, soit un objet Nœud .

Le code python de cette implémentation est le suivant :

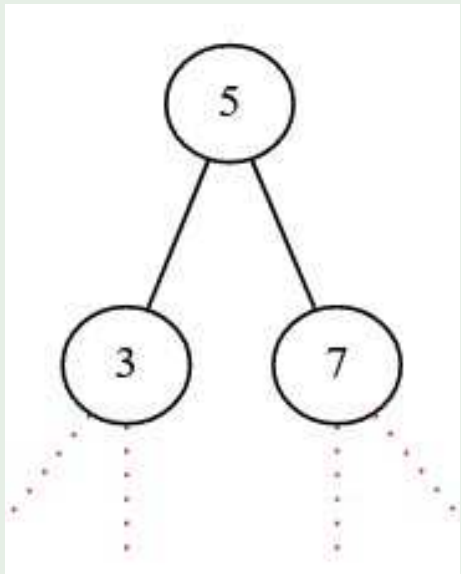
```
1  class Noeud:
2      def __init__(self,valeur,gauche,droit):
3          self.valeur=valeur
4          self.gauche=gauche
5          self.droit=droit
6
7  class ABR:
8      def __init__(self):
9          self.racine=None
```

Exemple :

Création d'un ABR

```
1  # Instanciation d'un objet ABR (vide):
2  a=ABR()
3  print(a)
4  # Premier noeud : racine de l'arbre
5  a.racine=Noeud(5,None,None)
6  print(a)
7  # Arbre gauche non vide :
8  a.racine=Noeud(5,Noeud(3,None,None),None)
9  print(a)
10 # Arbres gauche et droit non vides :
11 a.racine=Noeud(5,Noeud(3,None,None),Noeud(7,None,None))
12 print(a)
```

Arbre



Affichage de l'ABR

```
None
Noeud(5,None,None)
Noeud(5,Noeud(3,None,None),None)
Noeud(5,Noeud(3,None,None),Noeud(7,None,None))
```

Recherche de valeur

Algorithme

Un arbre binaire de recherche est soit vide, soit constitué d'un nœud racine et de 2 arbres binaires de recherche gauche et droit.

La recherche d'une valeur dans un ABR s'appuie sur le parcours récursif de l'arbre. On en donne l'algorithme ci-dessous :

```
1 if arbre vide:
2     return False
3 else:
4     if x < valeur du Noeud visité:
5         return appel récursif avec arbre gauche
6     elif x > valeur Noeud visité:
7         return appel récursif avec arbre droit
8     else:
9         return True
```

Efficacité

La recherche dans ABR avec l'algorithme ci-dessous est efficace lorsque l'arbre est équilibré (bien tassé ou complet). En effet, à chaque appel récursif dans un arbre gauche ou droit, on élimine la moitié des valeurs. Le nombre d'appels correspond au plus à la hauteur de l'arbre.

Par exemple, un ABR complet de hauteur h=20 contient $2^{20} - 1 = 1048575$ valeurs et nécessitera au plus 20 appels récursifs pour trouver une valeur.

Recherche de valeur

En Python

La fonction **appartient** prend en paramètre la valeur x à chercher et l'ABR a.

```
1  def appartient(x,a):
2      if a is None:
3          return False
4      else:
5          if x<a.valeur:
6              return appartient(x,a.gauche)
7          elif x>a.valeur:
8              return appartient(x,a.droit)
9          else:
10             return True
```

Dans la classe ABR, on crée une méthode **appartenir** qui appelle la fonction **appartient**.

```
1  class ABR:
2      def __init__(self):
3          self.racine=None
4
5      def appartenir(self,x):
6          return appartient(x,self.racine)
```

Ajouter une valeur

Algorithme

On donne ci-après le principe de l'ajout d'une valeur :

- Si l'arbre est vide, on crée un Nœud avec la valeur à ajouter ;
- Sinon, on parcourt récursivement l'arbre jusqu'à la bonne position et on ajoute un Nœud avec la valeur donnée.

```
1  def ajoute(x,a):
2      if a is None:
3          return Noeud(x,None,None)
4      else:
5          if x<a.valeur:
6              return Noeud(a.valeur,ajoute(x,a.gauche),a.droit)
7          elif x>a.valeur:
8              return Noeud(a.valeur,a.gauche,ajoute(x,a.droit))
9          else:
10             return a #Noeud(a.valeur,a.gauche,a.droit)
```

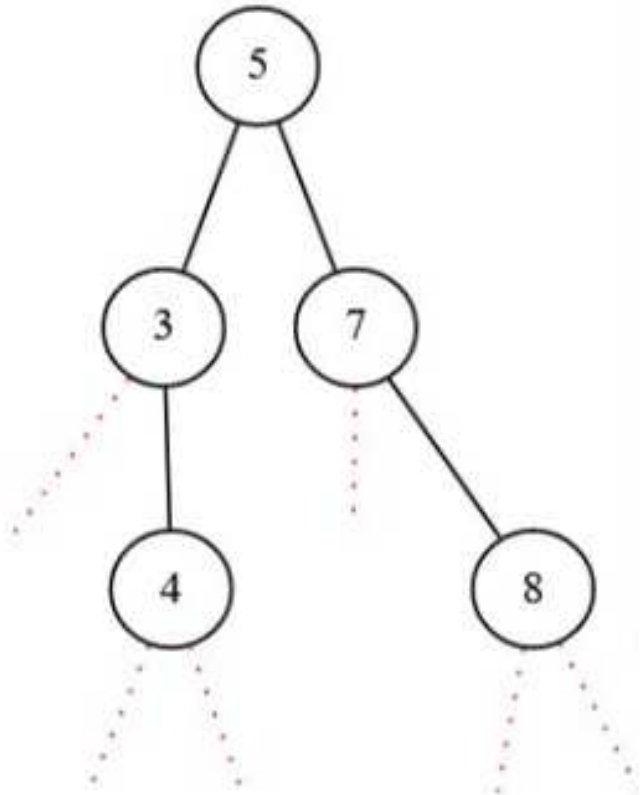
Dans la classe ABR, on crée une méthode **ajouter** qui appelle la fonction **ajoute**.

```
8      def ajouter(self,x):
9          self.racine=ajoute(x,self.racine)
```

Ajouter une valeur

Exemple

```
1  a=ABR()
2  print(a)
3  a.ajouter(5)
4  print(a)
5  a.ajouter(7)
6  print(a)
7  a.ajouter(3)
8  print(a)
9  a.ajouter(8)
10 print(a)
11 a.ajouter(4)
12 print(a)
```




```
None
Noeud(5,None,None)
Noeud(5,None,Noeud(7,None,None))
Noeud(5,Noeud(3,None,None),Noeud(7,None,None))
Noeud(5,Noeud(3,None,None),Noeud(7,None,Noeud(8,None,None)))
Noeud(5,Noeud(3,None,Noeud(4,None,None)),Noeud(7,None,Noeud(8,None,None)))
```