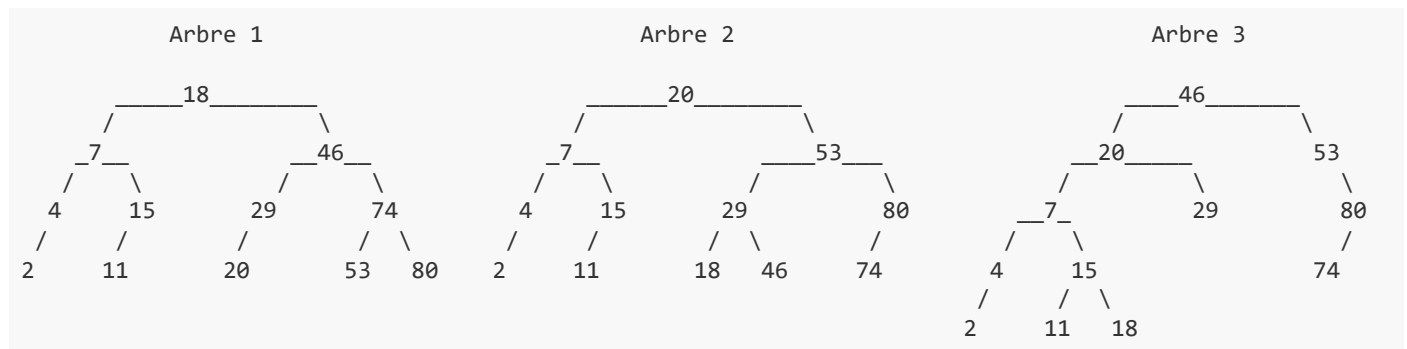


Dans cet exercice, la taille d'un arbre est égale au nombre de ses noeuds et on convient que la hauteur d'un arbre ne contenant qu'un noeud vaut 1.

On utilisera la définition suivante : un arbre binaire de recherche est un arbre binaire, dans lequel on peut comparer les valeurs des noeuds. Ce sont par exemple des nombres entiers, ou des lettres de l'alphabet :

- Si x est un noeud de cet arbre et y est un noeud du **sous-arbre gauche** de x , alors il faut que $y.valeur < x.valeur$
- Si x est un noeud de cet arbre et y est un noeud du **sous-arbre droit** de x , alors il faut que $y.valeur \geq x.valeur$

1. Pour chacun des arbres ci-dessous, justifier pourquoi il est ou non un arbre binaire de recherche.



Une classe **ABR**, qui implémente une structure d'arbre binaire de recherche, possède l'interface suivante :

```
class ABR:
    def __init__(self, valeur, sag, sad):
        self.valeur = valeur # Valeur de la racine
        self.sag = sag       # Sous-arbre gauche
        self.sad = sad       # Sous-arbre droit

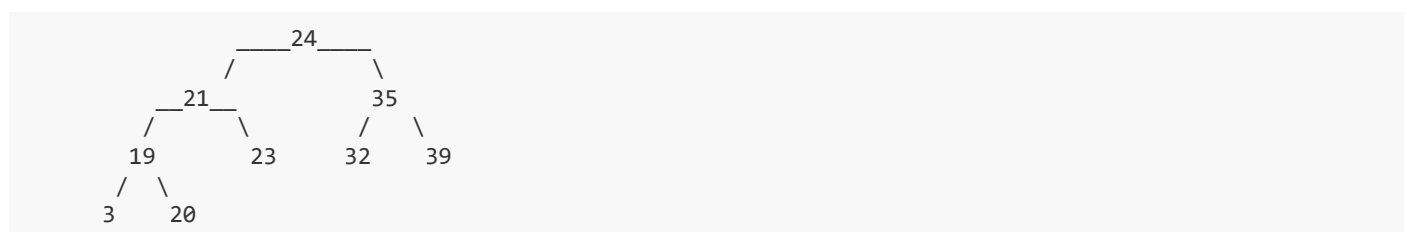
    def inserer_noeud(self, val):
        """ Renvoie un nouvel ABR avec le noeud de valeur 'val' inséré
        comme nouvelle feuille à sa position correcte """
        # Code non étudié dans cet exercice
```

La construction d'un objet **ABR** se fait en insérant progressivement les valeurs à partir de la racine : la méthode **inserer_noeud** (dont le code n'est pas étudié dans cet exercice) place ainsi un noeud à sa "bonne place" comme feuille dans la structure, sans modifier le reste de la structure.

2. En utilisant les méthodes de la classe **ABR** écrire l'instruction Python qui permet d'instancier un objet **a1** ayant un seul noeud (la racine) de valeur 19. Un arbre binaire vide prendra la valeur **None**.

3. Écrire une séquence d'instructions qui permet ensuite d'insérer dans l'objet **a1** les deux feuilles de l'arbre de valeurs 11 et 42.

Selon l'ordre dans lequel les valeurs sont insérées, on construit des ABR ayant des structures différentes. Voici par exemple ci-dessous un ABR **a2** obtenu en créant une instance de la classe **ABR** ayant un seul noeud de valeur 24 puis en insérant successivement les valeurs dans l'ordre suivant : 21, 35, 19, 23, 32, 39, 3, 20



4. Dessiner sur la copie l'ABR (nommé **a3**) que l'on obtiendrait en créant une instance de la classe **ABR** ayant un seul noeud de valeur 3 puis en insérant successivement les valeurs dans l'ordre suivant : 20, 19, 21, 24, 23, 35, 32, 39

5. Donner la hauteur des ABR **a2** et **a3**.

On complète la classe **ABR** avec une méthode **calculer_hauteur** qui renvoie la hauteur de l'arbre.

6. Recopier sur la copie les lignes **10** et **13** en les complétant par des **commentaires** et la ligne **14** en la complétant par **une instruction** dans le code ci-dessous de cette méthode.

On pourra utiliser la fonction Python **max** qui prend en paramètres deux nombres et renvoie le maximum de ces deux nombres.

```
1. def calculer_hauteur(self):
2.     """ Renvoie la hauteur de l'arbre """
3.
4.     if self.sag is None and self.sad is None :
5.         # L'arbre est réduit à une feuille
6.         return 1
7.     elif self.sad is None:
8.         # Arbre avec une racine et seulement un sous-arbre gauche
9.         return 1 + self.sag.calculer_hauteur()
10.    elif self.sag is None:
11.        # à compléter
12.        return 1 + self.sad.calculer_hauteur()
13.    else:
14.        # à compléter
15.        return # à compléter
```

7. Recopier et compléter sur la copie les lignes **3**, **7**, **8** et **10** du code ci-dessous de la méthode **rechercher_valeur**, qui permet de tester la présence ou l'absence d'une valeur donnée dans l'ABR :

```
1. def rechercher_valeur(self, v):
2.     """ Renvoie True si la valeur v est trouvée dans l'ABR False sinon """
3.
4.     if # à compléter
5.         return True
6.     elif v < self.valeur and self.sag is not None:
7.         return self.sag.rechercher_valeur(v)
8.     elif # à compléter
9.         return # à compléter
10.    else:
11.        return # à compléter
```

La différence de hauteur entre l'ABR **a2** et l'ABR **a3** aura des conséquences lors de la recherche d'une valeur dans l'ABR.

On admet que le nombre de fois où la méthode **rechercher_valeur** est appelée pour rechercher la valeur 32 dans l'ABR **a3** est **6**.

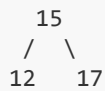
8. Donner le nombre de fois où la méthode **rechercher_valeur** est appelée pour rechercher la valeur 3 dans l'ABR **a2**.

Il existe des algorithmes pour modifier la structure d'un ABR, afin par exemple de diminuer la hauteur d'un ABR ; on s'intéresse aux algorithmes appelés **rotation**, consistant à faire "pivoter" une partie de l'arbre autour d'un de ses noeuds.

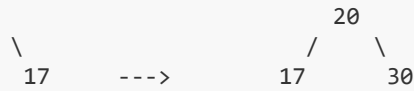
L'exemple ci-dessous permet d'expliquer l'algorithme pour réaliser une rotation droite d'un ABR autour de sa racine :



- On appelle **pivot** le sous-arbre gauche de la racine de l'arbre :



- Le sous-arbre droit du pivot (ici le noeud 17) devient le sous-arbre gauche de la racine :



- La racine ainsi modifiée devient le sous-arbre droit du pivot et la racine du pivot devient la nouvelle racine de l'ABR :



On admet que ces transformations conservent la propriété d'ABR de l'arbre.

La méthode `rotation_droite` ci-après renvoie une nouvelle instance de la classe `ABR`, correspondant à une rotation droite de l'objet de type `ABR` à partir duquel elle est appelée :

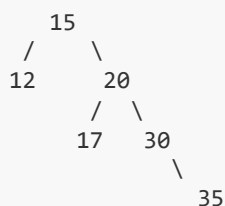
```

1. def rotation_droite(self):
2.     """ Renvoie une instance d'un ABR après une rotation droite
3.         On suppose qu'il existe un sous-arbre gauche """
4.     pivot = self.sag
5.     self.sag = pivot.sad
6.     pivot.sad = self
7.     return ABR(pivot.valeur, pivot.sag, pivot.sad)
  
```

Pour réaliser une rotation gauche, on suivra alors l'algorithme suivant :

- On appelle **pivot** le sous-arbre droit de la racine de l'arbre,
- Le sous-arbre gauche du pivot devient le sous-arbre droit de la racine,
- La racine ainsi modifiée devient le sous-arbre gauche du pivot et la racine du pivot devient la nouvelle racine de l'ABR

9. En suivant les différentes étapes de cet algorithme, dessiner l'arbre obtenu après une rotation gauche de l'ABR suivant :



10. Écrire le code d'une méthode Python `rotation_gauche` qui réalise la rotation gauche d'un ABR autour de sa racine.