Commencé le	lundi 20 mars 2023, 14:46
État	Terminé
Terminé le	lundi 20 mars 2023, 15:03
Temps mis	17 min 22 s
Note	**10,00** sur 10,00 (**100**%)

Question 1

Correct

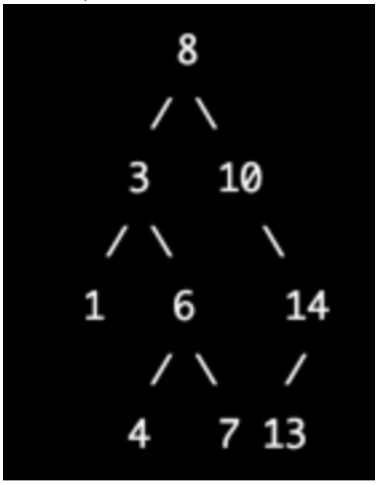
Note de 9,00 sur 9,00

Implémenter dans la classe BST (pour Binary Search Tree), la méthode qui recherche l'ancêtre commun (un BST) d'un arbre de recherche binaire (BST) pour deux étiquettes de nœuds données.

* La méthode prend deux étiquettes de nœuds en entrée et renvoie l'ancêtre commun le plus bas des deux nœuds.

BST<T> findCommonAncestor(T node1, T node2)

Par exemple :



L'ancêtre commun à 1 et 7 est l'arbre de racine 3 ; l'ancêtre commun à 14 et 1 est l'arbre de racine 8

Si l'un des deux labels ne correspond pas à un noeud de l'arbre, l'exception *NoSuchElementException* est levée.

Dans la classe BST, vous pouvez définir toutes les méthodes dont vous avez besoin.

Implement in the public class *BST (for Binary Search Tree)*, the method that searches for the common ancestor (a BST) of a Binary Search Tree (BST) for two given node labels.

* The method takes two node labels as input and returns the lowest common ancestor of the two nodes.

BST<T> findCommonAncestor(T node1, T node2)

The common ancestor of 1 and 7 is root tree 3; the common ancestor of 14 and 1 is root tree 8

If one of the two labels does not match a node in the tree, the NoSuchElementException is thrown.

In the BST class, you can define all the methods you need.

Par exemple:

Test	Résultat

`//Root is ancestor (simple)`
`BST<Integer> bst = exampleBST();`
`assertEquals(8, bst.findCommonAncestor(8,10).getElement());`
	`test : 8 equals 8 ?`
	`true`

`//Subtree is ancestor`
`BST<Integer> bst = exampleBST();`
`assertEquals(6, bst.findCommonAncestor(4, 7).getElement());`
`assertEquals(3, bst.findCommonAncestor(7,1).getElement());`
`assertEquals(3, bst.findCommonAncestor(1,4).getElement());`
	`test : 6 equals 6 ?`
	`true`
	`test : 3 equals 3 ?`
	`true`
	`test : 3 equals 3 ?`
	`true`

`//One is ancestor of the other one`
`BST<Integer> bst = exampleBST();`
`assertEquals(10, bst.findCommonAncestor(10, 14).getElement());`
`assertEquals(14, bst.findCommonAncestor(13, 14).getElement());`
`assertEquals(10, bst.findCommonAncestor(10, 13).getElement());`
	`test : 10 equals 10 ?`
	`true`
	`test : 14 equals 14 ?`
	`true`
	`test : 10 equals 10 ?`
	`true`

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1 package ads.poo2.exams.test3;
```

```
2
3  import java.util.*;
4
5  /**
6   * A class for Binary Search Trees
7   */
8  public class BST<T extends Comparable<? super T>> implements Iterable<T> {
9
10     public static final String ELEMENT_NOT_FOUND = "Element not found";
11
12     private T element;
13     private BST<T> left;
14     private BST<T> right;
15
16     public T getElement() {
17         return element;
18     }
19
20     public BST<T> getLeft() {
21         return left;
22     }
23
24     public void setLeft(BST<T> left) {
25         this.left = left;
26     }
27
28     public BST<T> getRight() {
29         return right;
30     }
31
32     public void setRight(BST<T>right) {
33         this.right = right;
34     }
35
36
37     /**
38      * Cette méthode  recherche l'ancêtre commun d'un arbre binaire de recherche (Binary Search Tree - BST)
39      * pour deux étiquettes de nœuds donnés.
40      * La méthode prend deux nœuds "node1" et "node2" en entrée et renvoie l'ancêtre commun le plus bas des deux nœuds.
41      *
42      * Raise NoSuchElementException if element not in tree
43      * They must pass the following tests:
44      * assertThrows(NoSuchElementException.class,()-> bst.findCommonAncestor(100,8).getElement());
45      * assertThrows(NoSuchElementException.class,()-> bst.findCommonAncestor(1,28).getElement());
46      *
47      * @param node1 label of the depart node
48      * @param node2 label of the target node
49      * @return the common ancestor of the two nodes
50      */
51     public BST<T> findCommonAncestor(T node1, T node2) {
52         if (!this.contains(node1) || !this.contains(node2))
```

	Test	Résultat attendu	Résultat obtenu	

✔	`//same node`
	`BST<Integer> bst = exampleBST();`
	`assertEquals(8, bst.findCommonAncestor(8, 8).getElement());`
	`assertEquals(3, bst.findCommonAncestor(3, 3).getElement());`
		`test : 8 equals 8 ?`
		`true`
		`test : 3 equals 3 ?`
		`true`
			`test : 8 equals 8 ?`
			`true`
			`test : 3 equals 3 ?`
			`true`
				✔

✔	`//Node is not present in the tree`
	`BST<Integer> bst = exampleBST();`
	`assertThrows(NoSuchElementException.class,()-> bst.findCommonAncestor(100,8).getElement());`
	`assertThrows(NoSuchElementException.class,()-> bst.findCommonAncestor(1,28).getElement());`
		`test : java.util.NoSuchElementException equals java.util.NoSuchElementException ?`
		`true`
		`test : java.util.NoSuchElementException equals java.util.NoSuchElementException ?`
		`true`
			`test : java.util.NoSuchElementException equals java.util.NoSuchElementException ?`
			`true`
			`test : java.util.NoSuchElementException equals java.util.NoSuchElementException ?`
			`true`
				✔

✔	`//Root is ancestor (simple)`
	`BST<Integer> bst = exampleBST();`
	`assertEquals(8, bst.findCommonAncestor(8,10).getElement());`
		`test : 8 equals 8 ?`
		`true`
			`test : 8 equals 8 ?`
			`true`
				✔

	Test	Résultat attendu	Résultat obtenu	

✔	`//Subtree is ancestor`
	`BST<Integer> bst = exampleBST();`
	`assertEquals(6, bst.findCommonAncestor(4, 7).getElement());`
	`assertEquals(3, bst.findCommonAncestor(7,1).getElement());`
	`assertEquals(3, bst.findCommonAncestor(1,4).getElement());`
		`test : 6 equals 6 ?`
		`true`
		`test : 3 equals 3 ?`
		`true`
		`test : 3 equals 3 ?`
		`true`
			`test : 6 equals 6 ?`
			`true`
			`test : 3 equals 3 ?`
			`true`
			`test : 3 equals 3 ?`
			`true`
				✔

✔	`//root is the ancestor`
	`BST<Integer> bst = exampleBST();`
	`assertEquals(8, bst.findCommonAncestor(4, 13).getElement());`
	`assertEquals(8, bst.findCommonAncestor(14, 3).getElement());`
		`test : 8 equals 8 ?`
		`true`
		`test : 8 equals 8 ?`
		`true`
			`test : 8 equals 8 ?`
			`true`
			`test : 8 equals 8 ?`
			`true`
				✔

✔	`//One is ancestor of the other one`
	`BST<Integer> bst = exampleBST();`
	`assertEquals(10, bst.findCommonAncestor(10, 14).getElement());`
	`assertEquals(14, bst.findCommonAncestor(13, 14).getElement());`
	`assertEquals(10, bst.findCommonAncestor(10, 13).getElement());`
		`test : 10 equals 10 ?`
		`true`
		`test : 14 equals 14 ?`
		`true`
		`test : 10 equals 10 ?`
		`true`
			`test : 10 equals 10 ?`
			`true`
			`test : 14 equals 14 ?`
			`true`
			`test : 10 equals 10 ?`
			`true`
				✔

✔	`BST<String> bstVocabulary = initVocabulary();`
	`assertEqualsO("arthur", bstVocabulary.findCommonAncestor("genievre", "arthur").getElement());`
	`assertEqualsO("genievre", bstVocabulary.findCommonAncestor("guinevere", "buzzards").getElement());`
	`assertEqualsO("claudius", bstVocabulary.findCommonAncestor("eustace", "claudius").getElement());`
	`assertEqualsO("inspirit", bstVocabulary.findCommonAncestor("jubilation", "hobgoblin").getElement());`
	`assertEqualsO("hello", bstVocabulary.findCommonAncestor("kaleidoscope", "gallimaufry").getElement());`
	`assertEqualsO("eustace", bstVocabulary.findCommonAncestor("eustace", "eustace").getElement());`
		`test : arthur equals arthur ?`
		`true`
		`test : genievre equals genievre ?`
		`true`
		`test : claudius equals claudius ?`
		`true`
		`test : inspirit equals inspirit ?`
		`true`
		`test : hello equals hello ?`
		`true`
		`test : eustace equals eustace ?`
		`true`
			`test : arthur equals arthur ?`
			`true`
			`test : genievre equals genievre ?`
			`true`
			`test : claudius equals claudius ?`
			`true`
			`test : inspirit equals inspirit ?`
			`true`
			`test : hello equals hello ?`
			`true`
			`test : eustace equals eustace ?`
			`true`
				✔

Tous les tests ont été réussis ! ✔

▸ **Montrer / masquer la solution de l'auteur de la question (Java)**

Correct

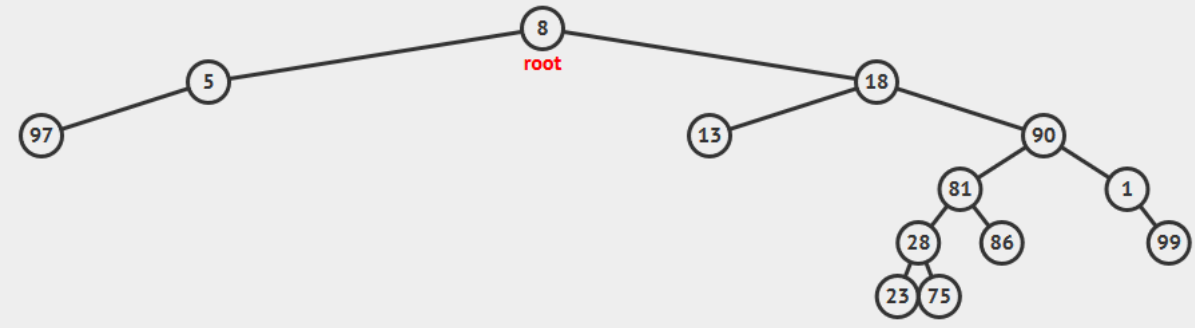
Note pour cet envoi : 9,00/9,00.

Question 2

Correct

Note de 1,00 sur 1,00

The binary tree shown in the picture is a valid **BST** :



Veuillez choisir une réponse.

- Vrai
- Faux ✔

Votre réponse est correcte.

La réponse correcte est : Faux