POLYTECHNIQUE
MONTRÉAL

UNIVERSITÉ
D'INGÉNIERIE

Méthodes de test et de validation du logiciel

LOG3430

TP2 : Tests de partition de catégorie et de flot de données

Thomas Houtart et Ludovic Bonheur
1895556 et 1783113

Remis à :
Noureddine Kerzazi

28 février 2020

Hiver 2020

Question 1

`insert(self, value)`

Paramètre	**Catégorie**
V	Valeur ajouté dans l'arbre
E	premier noeud de l'arbre

Spécifications formelles des tests

$V0 : \{v : \text{None}\}[\text{error}]$

$V1 : \{v : \text{number}\}[\text{validValue}]$

$E0 : \{e : \text{None}\}[\text{emptyRoot}]$

$E1 : \{e : \text{Node}\}[\text{validRoot}]$

Test :

$V0 \rightarrow i0 = \langle\{v = \text{None}, e =?\}, \{\text{error}\}\rangle$

$V1E0 \rightarrow i1 = \langle\{v = 20, e = None\}, \{\text{self.root} = \text{Node(20)}\}\rangle$

$V1E1 \rightarrow i2 = \langle\{v = 10, e = Node(20)\}, \{\text{call \_insert(10, self.root)}\}\rangle$

où
$i0$ = test_when_value_is_not_a_number_should_return_error
$i1$ = test_when_tree_is_empty_should_create_node_for_root
$i2$ = test_when_tree_is_not_empty_should_call_insert_function

`insert(self, value, cur_node)`

Paramètre	**Catégorie**
V	Valeur ajouté dans l'arbre
C	Valeur du noeud courant
L	noeud gauche du noeud courant
R	noeud droit du noeud courrant

Spécifications formelles des tests

$V0 : \{v : \text{None}\}[\text{error}]$

$V1 : \{v : \text{number}\}[\text{validValue}]$

$C0 : \{c : \text{None}\}[\text{error}]$

$C1 : \{c : c == v\}[\text{validNodeValue}]$

$C2 : \{c : c > v\}[\text{validNodeValue}]$

$C3 : \{c : c < v\}[\text{validNodeValue}]$

$L0 : \{l : \text{None}\}[\text{emptyLeftRoot}]$

$L1 : \{l : \text{Node}\}[\text{validLeftRoot}]$

$R0 : \{l : \text{None}\}[\text{emptyRightRoot}]$

$R1 : \{l : \text{Node}\}[\text{validRigthRoot}]$

Test :

$V0 \rightarrow n0 = \langle\{v = \text{None}, c =?, l =?, r =?\}, \{\text{error}\}\rangle$

$C0 \rightarrow n1 = \langle\{v =?, c = \text{None}, l =?, r =?\}, \{\text{error}\}\rangle$

$V1C1 \rightarrow n2 = \langle\{v = 10, c = 10, l =?, r =?\}, \{\text{call print("This Value is already in the tree!!!")}\}\rangle$

$V1C2L0 \rightarrow n3 = \langle\{v = 10, c = 15, l = \text{None}, r =?\}, \{\text{l} == \text{Node(10)} ; \text{l.parent} = \text{cur\_node}\}\rangle$

$V1C2L1 \rightarrow n4 = \langle\{v = 10, c = 15, l = \text{Node}(8), r =?\}, \{\text{call \_insert(10, l}\}\rangle$

$V1C3R0 \rightarrow n5 = \langle\{v = 20, c = 15, l =?, r = \text{None}\}, \{\text{r} == \text{Node(20)} ; \text{l.parent} = \text{cur\_node}\}\rangle$

$V1C3R1 \rightarrow n6 = \langle\{v = 20, c = 15, l =?, r = \text{Node}(16)\}, \{\text{call \_insert(20, r}\}\rangle$

où
$n0$ = test_when_value_is_not_a_number_should_return_error
$n1$ = test_when_current_node_value_is_not_a_number_should_return_error
$n2$ = test_when_value_is_equal_to_current_node_value_should_print_message
$n3$ = test_when_value_is_smaller_then_current_node_value_and_left_node_is_empty_should...
_create_node_with_chosen_value_and_set_current_node_as_his_parent

$n4 =$ test_when_value_is_smaller_then_current_node_value_and_left_node_is_not_empty_should...
_call_insert_with_value_and_left_node
$n5 =$ test_when_value_is_greater_then_current_node_value_and_right_node_is_empty_should...
_create_node_with_chosen_value_and_set_current_node_as_his_parent
$n6 =$ test_when_value_is_greater_then_current_node_value_and_right_node_is_not_empty_should...
_call_insert_with_value_and_right_node

`print(self)`

Paramètre	**Catégorie**
V	Valeur ajouté dans l'arbre
E	Noeud initial de l'arbre

Spécifications formelles des tests

$V0 : \{v : \text{None}\}[\text{error}]$

$V1 : \{v : \text{number}\}[\text{validValue}]$

$E0 : \{e : \text{None}\}[\text{emptyRoot}]$

$E1 : \{e : \text{Node}\}[\text{validRoot}]$

Test :

$V0 \to p0 = \langle\{v = \text{None}, e = ?\}, \{\text{erreur}\}\rangle$

$V1E0 \to p1 = \langle\{v = 20, e = None\}, \{\text{ Nothing }\}\rangle$

$V1E1 \to p2 = \langle\{v = 10, e = Node(20)\}, \{\text{call\_}self.\_print\_tree(self.root)\}\rangle$

$p0 =$ test_when_value_is_not_a_number_should_return_error
$p1 =$ test_when_root_is_none_should_do_nothing
$p2 =$ test_when_root_is_not_none_should_call_print_tree_with_root

`print(self, cur_node)`

Paramètre	**Catégorie**
V	Valeur ajouté dans l'arbre
C	Noeud actuellement dans l'arbre

Spécifications formelles des tests

$V0 : \{v : \text{None}\}[\text{error}]$

$V1 : \{v : \text{number}\}[\text{validValue}]$

$C0 : \{c : \text{None}\}[\text{emptyRoot}]$

$C1 : \{c : \text{Node}\}[\text{validRoot}]$

Test :

$V0 \rightarrow p0 = \langle\{v = \text{None}, c =?\}, \{\text{erreur}\}\rangle$

$V1C0 \rightarrow p1 = \langle\{v = 20, c = None\}, \{\text{ Nothing }\}\rangle$

$$\begin{aligned} V1C1 \rightarrow p2 = \langle\{v = 10, c = Node(20)\}, \{&\text{call\_}self.\_print\_tree(cur\_node.left), \\ &call\ print(str(cur\_node.value)), \\ &call\ self.\_print\_tree(cur\_node.right)\}\rangle \end{aligned}$$

$p0$ = test_when_value_is_not_a_number_should_return_error
$p1$ = test_when_cur_node_is_none_should_do_nothing
$p2$ = test_when_cur_node_is_not_none_should_call_print_tree_with_left_node_and...
_call_print_of_cur_node_value_and_call_print_with_right_node

Question 2

`delete_node(self, node)`

Paramètre	**Catégorie**
N	Noeud à supprimer dans l'arbre
V	Valeur du noeud à supprimer est dans l'arbre
P	Parent du noeud à supprimer dans l'arbre
L	fils gauche du noeud à supprimer
R	fils droit du noeud à supprimer

Spécifications formelles des tests

$N0$: $\{n$: None$\}$[none]

$N1$: $\{n$: Node$\}$[validNode]

$V0$: $\{v$: False$\}$[None]

$V1$: $\{v$: True$\}$[validValue]

$P0$: $\{p$: None$\}$[noParent]

$P1$: $\{p$: Node$\}$[validParent, leftChildIsValidNode]

$P2$: $\{p$: Node$\}$[validParent, rightChildIsValidNode]

$L0$: $\{l$: None$\}$[noLeftNode]

$L1$: $\{l$: Node$\}$[validLeftNode]

$R0$: $\{r$: None$\}$[noLeftRight]

$R1$: $\{r$: Node$\}$[validLeftNode]

Test :

$N0V0P0L0R0 \rightarrow p0 = \langle \{n = \text{”None”}, v = ”False”, p = \text{None}, l = \text{None}, r = \text{None}\}, \{\text{None}\} \rangle$

$N1V0P0L0R0 \rightarrow p1 = \langle \{n = \text{”Node(20)”}, v = ”False”, p = \text{None}, l = \text{None}, r = \text{None}\}, \{\text{None}\} \rangle$

$N1V1P0L0R0 \rightarrow p2 = \langle \{n = \text{”Node(20)”}, v = ”True”, p = \text{None}, l = \text{None}, r = \text{None}\}, \{\text{self.root} = \text{None}\} \rangle$

$N1V1P0L0R1 \rightarrow p3 = \langle \{n = \text{”Node(20)”}, v = ”True”, p = \text{None}, l = \text{None}, r = \text{Node(25)}\}, \{\text{self.root} = \text{r}\} \rangle$

$N1V1P0L1R0 \rightarrow p4 = \langle \{n = \text{”Node(20)”}, v = ”True”, p = \text{None}, l = \text{Node(15)}, r = \text{None}\}, \{\text{self.root} = \text{l}\} \rangle$

$N1V1P0L1R1 \rightarrow p5 = \langle \{n = \text{”Node(20)”}, v = ”True”, p = \text{None}, l = \text{Node(15)}, r = \text{Node(25)}\},$
$\{\text{call min\_value\_node(r)}\,;\ \text{v} = \text{successor.value}\,;\ \text{call delete\_node(successor)}\} \rangle$

$N1V1P1L0R0 \rightarrow p6 = \langle \{n = \text{”Node(20)”}, v = ”True”, p = \text{Node(30)}, l = \text{None}, r = \text{None}\}, \{\text{p.left} = \text{None}\} \rangle$

$N1V1P1L0R1 \rightarrow p7 = \langle \{n = \text{”Node(20)”}, v = ”True”, p = \text{Node(30)}, l = \text{None}, r = \text{Node(25)}\}, \{\text{p.left} = \text{r}\} \rangle$

$N1V1P1L1R0 \rightarrow p8 = \langle \{n = \text{”Node(20)”}, v = ”True”, p = \text{Node(30)}, l = \text{Node(15)}, r = \text{None}\}, \{\text{p.left} = \text{l}\} \rangle$

$N1V1P1L1R1 \rightarrow p9 = \langle \{n = \text{”Node(20)”}, v = ”True”, p = \text{Node(30)}, l = \text{Node(15)}, r = \text{Node(25)}\},$
$\{\text{call min\_value\_node(r)}\,;\ \text{v} = \text{successor.value}\,;\ \text{call delete\_node(successor)}\} \rangle$

$N1V1P2L0R0 \rightarrow p10 = \langle \{n = \text{”Node(20)”}, v = ”True”, p = \text{Node(10)}, l = \text{None}, r = \text{None}\}, \{\text{p.right} = \text{None}\} \rangle$

$N1V1P2L0R1 \rightarrow p11 = \langle \{n = \text{”Node(20)”}, v = ”True”, p = \text{Node(10)}, l = \text{None}, r = \text{Node(25)}\}, \{\text{p.right} = \text{r}\} \rangle$

$N1V1P2L1R0 \rightarrow p12 = \langle \{n = \text{”Node(20)”}, v = ”True”, p = \text{Node(10)}, l = \text{Node(15)}, r = \text{None}\}, \{\text{p.right} = \text{l}\} \rangle$

$N1V1P2L1R1 \rightarrow p13 = \langle \{n = \text{”Node(20)”}, v = ”True”, p = \text{Node(10)}, l = \text{Node(15)}, r = \text{Node(25)}\},$
$\{\text{call min\_value\_node(r)}\,;\ \text{v} = \text{successor.value}\,;\ \text{call delete\_node(successor)}\} \rangle$

$p0 =$ test_when_value_for_deleted_node_is_None_should_return_None

$p1 =$ test_when_value_for_deleted_node_is_Node_but_value_is_not_in_tree_should_return_None

$p2 =$ test_when_value_for_deleted_node_is_Node_in_tree_with_no_parent_and_no_child_should_set_root_to_None

$p3 =$ test_when_value_for_deleted_node_is_Node_in_tree_with_no_parent_and_right_child_should_set_root_to_right_child

$p4 =$ test_when_value_for_deleted_node_is_Node_in_tree_with_no_parent_and_left_child_should_set_root_to_left_child

$p5 =$ test_when_value_for_deleted_node_is_Node_in_tree_with_no_parent_and_two_child_should_set_root_to_min_right_child_node_and_call_delete_node_using_said_node

$p6 =$ test_when_value_for_deleted_node_is_left_child_of_its_parent_and_has_no_child_should_set_parents_left_child_to_None

$p7 =$ test_when_value_for_deleted_node_is_left_child_of_its_parent_and_has_right_child_should_set_parents_left_child_to_right_child

$p8 =$ test_when_value_for_deleted_node_is_left_child_of_its_parent_and_has_left_child_should_set_parents_left_child_to_left_child

$p9 =$ test_when_value_for_deleted_node_is_left_child_of_its_parent_and_has_two_child_should_set_node_to_min_right_child_node_and_call_delete_node_using_said_node

$p10 =$ test_when_value_for_deleted_node_is_right_child_of_its_parent_and_has_no_child_should_set_parents_right_child_to_None

$p11 =$ test_when_value_for_deleted_node_is_right_child_of_its_parent_and_has_right_child_should_set_parents_right_child_to_right_child

$p12 =$ test_when_value_for_deleted_node_is_right_child_of_its_parent_and_has_left_child_should_set_parents_right_child_to_left_child

$p13 =$ test_when_value_for_deleted_node_is_right_child_of_its_parent_and_has_two_child_should_set_node_to_min_right_child_node_and_call_delete_node_using_said_node

Question 3

`delete_node(self, node)`

*les lignes sont par rapport au codes fournie.

Noeuds N	DEF	C-USE	P-USE
noeud 1	node		
noeud 2			node, node.value
noeud 3			
noeud 4	node_parent, node_children	node.parent, node	
noeud 5			node_children
noeud 6			node_parent
noeud 7			node_parent.left, node
noeud 8	node_parent.left		
noeud 9	node_parent.right		
noeud 10	self.root		
noeud 11			node_children
noeud 12			node.left
noeud 13	tmp_child	node.left	
noeud 14	tmp_child	node.right	
noeud 15			node_parent
noeud 16			node_parent.left, node
noeud 17	node_parent.left	tmp_child	
noeud 18	node_parent.right	tmp_child	
noeud 19	self.root	tmp_child	
noeud 20	tmp_child.parent	node_parent	
noeud 21			node_children
noeud 22	successor, node.value	node.right, successor.value, successor	

où :
noeud 1 correspond à la ligne 165
noeud 2 correspond à la ligne 166
noeud 3 correspond à la ligne 167-168

Path1 = {1, 2} avec P-USE(node, 2)
Path2 = {1, 2} avec P-USE(node.value, 2)
Path3 = {1, 2, 4} avec C-USE(node.parent, 4)
Path4 = {1, 2, 4} avec C-USE(node, 4)
Path5 = {1, 2, 4} avec C-USE(node, 4)
Path6 = {1, 2, 4, 5} avec P-USE(node_children, 5)
Path7 = {1, 2, 4, 5, 6} avec P-USE(node_parent, 6)
Path8 = {1, 2, 4, 5, 6, 7} avec P-USE(node_parent.left, 7)
Path9 = {1, 2, 4, 5, 6, 7} avec P-USE(node, 7)
Path10 = {1, 2, 4, 5, 6, 7, 8, 11} avec P-USE(node_children, 11)
Path11 = {1, 2, 4, 5, 6, 7, 8, 11, 12} avec P-USE(node.left, 12)
Path12 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 13} avec C-USE(node.left, 13)
Path13 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 14} avec C-USE(node.right, 14)
Path14 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 14, 15} avec P-USE(node_parent, 15)

Path15 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 14, 15, 16} avec P-USE(node_parent.left, 16)
Path16 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 14, 15, 16} avec P-USE(node, 16)
Path17 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 14, 15, 16, 17} avec C-USE(tmp_child, 17)
Path18 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 14, 15, 16, 18} avec C-USE(tmp_child, 18)
Path19 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 14, 15, 19} avec C-USE(tmp_child, 19)
Path20 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 14, 15, 19, 20} avec C-USE(node_parent, 20)
Path21 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 14, 15, 19, 20, 21} avec C-USE(node_children, 21)
Path22 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 14, 15, 19, 20, 21, 22} avec C-USE(node.right, 22)
Path23 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 14, 15, 19, 20, 21, 22} avec C-USE(successor.value, 22)
Path24 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 14, 15, 19, 20, 21, 22} avec C-USE(successor, 22)

Paths :
PathA = {1, 2, 4, 5, 6, 7, 8, 11, 21, 23} couvre Path 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 21
PathB = {1, 2, 4, 5, 11, 12, 13, 15, 16, 17, 20, 21, 23} couvre Path 11, 12, 13, 14, 15, 16, 17, 20
PathC = {1, 2, 4, 5, 11, 21, 22, 23} couvre Path 22, 23, 24

Jeu de test qui satisfait all-P-Uses some-C-Uses :
$p0 = \langle\{n = ''\text{node(20)}'', v = ''False'', p = \text{node}(30), l = \text{None}, r = \text{None}\}\rangle$
$p1 = \langle\{n = ''\text{node(20)}'', v = ''False'', p = \text{node}(30), l = \text{node}(15), r = \text{None}\}\rangle$
$p2 = \langle\{n = ''\text{node(20)}'', v = ''False'', p = \text{node}(10), l = \text{node}(15), r = \text{node}(25)\}\rangle$

Question 4

`_reversetree(self, cur\_node)`

Variable	Lignes de définition	Lignes d'utilisation	Chemin déf->util.
cur_node	266, 270	267, 271, 272, 273	266-267, 266-267-269-270-271-272-273
cur_node.left	266, 270	267, 271, 272, 273	270-271
cur_node.right	266, 270	267, 271, 272, 273	270-271
_reversetree	266	271, 272	266-267-269-270-271-272

Test

Tableau :

Noead n	DEF	C-USE	P-USE
1	cur_node, _reversetree		
2			cur_node
3			
4	cur_node, cur_node.left, cur_node.right	cur_node, cur_node.left, cur_node.right	cur_node, cur_node.left, cur_node.right

où :
noeud 1 correspond à la ligne 266
noead 2 correspond à la ligne 267
noead 3 correspond à la ligne 268
noeud 4 correspond à la ligne 269-270-271-272-273

On a 2 tests :
Path1 = {1, 2, 3} avec P-USE(cur_node, 2)
Path2 = {1, 2, 4} avec P-USE(cur_node, 2), C-USE(cur_node, 4),
C-USE(cur_node.left, 4), C-USE(cur_node.right, 4) P-USE(cur_node, 4), P-USE(cur_node.left, 4),
P-USE(cur_node.right, 4), P-USE(_reversetree, 4)