LOG3430 – Méthodes de test et de validation du logiciel

Laboratoire 2

Tests de partition de catégorie et de flot de données

Département de génie informatique et de génie logiciel

École Polytechnique de Montréal



Soumis par

Roman Zhornytskiy (1899786)

Hakim Payman (1938609)

Gabriel Tagliabracci (1935775)

Groupe : 02

Soumis à Noureddine Kerzazi

Hiver 2020

4. Travail à effectuer

4.1. Pour afficher la liste des nœuds en ordre (inorder traversal), on exploite la propriété de base d’un BST pour lire les nœuds de l’arbre de gauche à droite. Utilisez l’approche de partition de catégories EC pour tester les opérations **d’insertion et d’affichage**. Astuce : considérez la propriété « le BST est trié correctement » comme une des catégories. Dans votre rapport, il faut aller jusqu’au niveau des cas de test, \*\*pas\*\* l’écriture des tests avec l’échafaudage unittest. [4 points]

Cas de test pour l’insertion

V : number.

O : n’importe quoi. Ça peut être un objet ou un String

N : nœud.

V1 = {valeur < 0} [propriétés : number]

V2 = {0 < valeur < ∞} [propriétés : number]

V3 = {valeur = O} [erreur, propriétés : n’est pas un number]

V4 = {valeur = valeur} [la valeur existe déjà dans l’arbre, propriétés : number]

N1 = {N: None}

N2 = {N: not None}

N3 = {N: root} [La racine contient des fils]

N4 = {N: leaf} [Le nœud ne contient pas de fils]

V1N1 -> t1 = < {valeur = -420, N = None}, {-420 was inserted} >

V2N2 -> t2 = < {valeur = 4, N = not None}, {4 was inserted} >

V3N3 -> t3 = < {valeur = "hello", N = root}, {Error in insertion} >

V4N4 -> t4 = < {valeur = 4, N = leaf}, {Error, this value already exists in the BST} >

Cas de test pour l’affichage

B : BST.

B1 = {B: None}

B2 = {B: not None}

B1 -> t1 = < {B = None}, {The BST is empty} >

B2 -> t2 = < {B = 1,2,4}, {1,2,4} >

4.2. Utilisez de nouveau l’approche de partition de catégories, mais cette fois avec AC au lieu de EC, pour l’opération de suppression de nœuds (méthode delete\_node()) dans le fichier BST.py. Allez jusqu’au niveau de l’écriture des tests avec unittest. [6 points]

Cas de test pour la suppression

B : BST.

N : Nœud.

B1 : {B : None}

B2 : {B : not None}

N1 : {N : number} [le nœud est dans l’arbre]

N2 : {N : number} [le nœud n’est pas dans l’arbre]

B1N1 -> t1 = < {B = None, N = None}, {Error} >

B1N2 -> t2 = < {B = None, N = not None}, {Error} > **//impossible à tester, car dès qu’on insère un nœud dans l’arbre, l’arbre n’est plus « None »**

B2N1 -> t3 = < {B = not None, N = None}, {Error} >

B2N2 -> t4 = < {B = not None, N = not None}, {Deleted node with success}>

4.3. Utilisez maintenant l’approche boîte blanche all-P-uses/some-C-uses (flot de données) sur la même opération (méthode delete\_node()) dans le fichier BST.py que 4.2 (suppression de nœuds) en regardant le code source joint à cet énoncé. Allez de nouveau jusqu’au niveau de l’écriture des tests avec unittest. Comparez vos résultats avec ceux de 4.2, qu’estce que vous remarquez? [6 points]

Nous remarquons que nous avons plus de tests pour le cas des tests de boîte blanche.

4.4. Utilisez de nouveau l’approche boîte blanche all-P-uses/some-C-uses (flot de données), cette fois-ci sur la méthode invertTree() dans le fichier BST.py (inversion de l’arbre). Dans votre rapport, il faut aller jusqu’au niveau des cas de test, **\*\*pas\*\*** l’écriture des tests avec l’échafaudage unittest. [4 points]

**Roman : c’est la fonction reversetree et non invertTree qui n’existe pas…**

Cas de test pour l’inversion

B : BST.

N : Nœud.

B1 = {B: None}

B2 = {B: not None}

N1 = {N: None}

N2 = {N: number}

B1N1 -> t1 = < {B = None, N = None}, {return value = None} >

B1N2 -> t2 = < {B = None, N = 3}, {return value = None} >

B2N1 -> t3 = < {B = 1,2,4, **N = None**}, {B = 4,2,1} >

B2N2 -> t4 = < {B = 1,2,4, **N = 1**}, {B = 4,2,1} >

Roman : je ne suis pas trop sûr pour les valeurs de N, car à chaque itération le nœud à inverser change… alors je ne suis pas sûr de la façon de représenter ça…