LOG3430

Méthodes de test et de validation du logiciel

Groupe 03

Travail pratique #2

Test boîte noire d'une application

Présenté à Rodrigo MORALES

David TREMBLAY 1748125

Taleb SOULI 1770491

Département de génie informatique et génie logiciel

Le 16 octobre 2017
École polytechnique de Montréal

Table des matières

1. Conception des tests boîte noire

Tout comme le travail pratique précédent, la première étape de notre conception de tests a été d'importer les classes Graph ainsi que GraphGenerator provenant des fichiers fournis pour les laboratoires. Contrairement au TP1, seulement trois graphes devaient être testés cette fois-ci. Nous avons donc réalisé notre conception de tests en suivant l'ordre et des graphes et des jeux de tests pour chaque graphe ont ensuite été conçus en trois étapes.

Premièrement, nous avons réfléchi sur les différents cas possibles de retour de fonction et plus précisément sur les cas invalides et valides selon le type de graphe. Nous avons par la suite réalisé un tableau de classes équivalentes à l'aide de ces différents cas et avons choisi des valeurs dans chaque classe équivalente pour construire des cas de tests. Il est à noter que nous n'avons pas regardé l'implémentation des graphes pour réaliser cette étape puisque nous réalisons des tests boîte noire pour ce TP et donc nous testons sans nous soucier de l'implémentation.

La deuxième étape a été de regarder notre tableau de cas de tests pour le graphe et de réaliser un jeu de tests de type **« Each Choice »**. Pour ce faire, nous avons tout simplement pris une valeur de la première classe d'équivalence de chaque paramètre pour réaliser un test puis avons passé à la deuxième valeur et ensuite de suite. Si un paramètre était déjà testé pour chaque classe avant d'avoir fini les autres paramètres, nous avons pris une valeur dans la première classe et recommencé le processus jusqu'à la fin. Cette situation ne s'est pas produite par contre puisque nos cas de tests avaient tous le même nombre de valeurs pour chaque paramètre et le jeu de tests EC était donc identique à nos cas de tests.

Finalement, la dernière étape consistait à réaliser un jeu de tests de type **« All Choice »**. Pour ce faire, nous avons tout simplement testé toutes les combinaisons de valeurs possibles. Les jeux de tests obtenus sont détaillés et analysés dans la suite de ce rapport.

Pour ce qui a trait aux tests eux-mêmes, nous avons utilisé la méthode **assertEquals()** pour vérifier que les paramètres du graphe étaient bien ceux entrés ou des mots clés **expected** suivis du nom d'une classe d'erreur pour confirmer que les bonnes erreurs étaient retournés lors d'entrées invalides. Un exemple pour chacun de ces cas est illustré dans la figure ci-jointe:

```java
    // Test the case where the vertices number is negative for bipartite Probability Graph
    @Test(expected = NegativeArraySizeException.class)
    public void testBipartiteGraphWithProbabilityEC1() {
        int verticesOne = -10;
        int verticesTwo = 0;
        double probability = 0.5;
        Graph graphToTest = GraphGenerator.bipartite(verticesOne, verticesTwo, probability);
    }

    // Test the case where the vertices number is very high for bipartite Probability Graph
    @Test(expected = OutOfMemoryError.class)
    public void testBipartiteGraphWithProbabilityEC2() {
        int verticesOne = Integer.MAX_VALUE;
        int verticesTwo = -7;
        double probability = 0.0;
        Graph graphToTest = GraphGenerator.bipartite(verticesOne, verticesTwo, probability);
    }

    // Test the case where the number of vertices is 0 for bipartite Probability Graph
    @Test public void testBipartiteGraphWithProbabilityEC3() {
        int verticesOne = 0;
        int verticesTwo = 7;
        double probability = 1.0;
        int vertices = 7;
        Graph graphToTest= GraphGenerator.bipartite(verticesOne, verticesTwo, probability);
        assertEquals("Vertices must be "+vertices, vertices, graphToTest.V());
        assertEquals("Type name must be bipartite", "bipartite", graphToTest.getTypeName());
    }

    // Test the case where the probability is greater than 1.0 for bipartite Probability Graph
    @Test(expected = IllegalArgumentException.class)
    public void testBipartiteGraphWithProbabilityEC4() {
        int verticesOne = 10;
        int verticesTwo = 70;
        double probability = 2.0;
        Graph graphToTest = GraphGenerator.bipartite(verticesOne, verticesTwo, probability);
    }
```

Figure 1: Exemple de tests

Nous avons aussi ajouté nos jeux de tests en commentaire ainsi qu'utilisé des noms de tests explicites pour faciliter la compréhension du code et tous les noms de tests finissent par EC ou AC suivi d'un nombre (exemple EC2 et EC3 dans la figure ci-haut référent aux tests 2 et 3 du tableau « Each Choice » pour le graphe biparti avec probabilité).

2. Énumération des tests

Les cas de tests sont énumérés sous forme de tableau dans cette section. Par contre, les jeux de test EC et AC ne s'y trouvent pas puisque les tests AC sont très longs à énumérer et une contrainte de 4 pages maximum doit être respectée. Ils sont toutefois détaillés en commentaire dans le fichier test fourni avec ce rapport. Ils sont aussi analysés dans la section suivante. Il est à noter que puisque nous avons utilisé le même nombre de valeurs pour chaque paramètre, les jeux de tests EC se trouvent à être identiques à nos valeurs choisies pour réaliser ces cas de tests.

Graphe simple avec probabilité

Tableau 1: Cas de tests - Graphe simple avec probabilité

Vertices	Probability
-1	-1.0
0	0.0
5	0.5
Integer.MAX_VALUE	1.0
1000	2.0

Graphe biparti avec probabilité

En raison du fait que ce graphe contient trois paramètres et que donc les tests « All Choice » sont beaucoup plus exhaustifs pour chaque valeur ajoutée au testage, nous n'avons utilisé que trois des cinq cas de tests d'EC pour les tests AC (cas 1, 3 et 5).

Tableau 2: Cas de tests- Graphe biparti avec probabilité

Vertices_ONE	Vertices_TWO	Probability
-10	0	0.5
Integer.MAX_VALUE	-7	0.0
0	7	1.0
10	70	2.0
100	Integer.MAX_VALUE	-3.0

Graphe régulier

Tableau 3: Cas de tests - Graphe régulier

Vertices	K == Degree
-5	1
Integer.MAX_VALUE	0
11	2
10	-2
5	3
0	Integer.MAX_VALUE

3. Analyse des tests

Graphe simple avec probabilité

Ce type de graphe prend en paramètre un nombre de sommets ainsi qu'une probabilité. Pour le paramètre *Vertices*, nous avons testé un nombre négatif, positif de grandeur

petite, positif de grande valeur, la plus grande valeur positive possible ainsi que le cas spécial 0. Pour la probabilité, nous savons qu'elle doit se situer entre 0 et 1.0. Nous avons donc testé une probabilité négative, médiane à l'intervalle d'entrées valides, les valeurs limites 0.0 et 1.0, ainsi qu'une probabilité plus grande que 1.0. Nous obtenons un jeu de tests « Each Choice » de 5 tests et un jeu de tests « All Choice » de 25 tests (5 x 5). Des valeurs de chaque classe d'équivalence sont testées ainsi que les valeurs limites entre celles-ci et tous les tests sont fonctionnels. Nous pouvons donc dire que notre confiance en cette méthode est grande.

Graphe biparti avec probabilité

Ce type de graphe est semblable au précédent mais prend 2 nombres de sommets en paramètre (puisque c'est un graphe biparti). Comme précédemment, nous avons utilisé, pour chacun des sommets, une valeur négative, positive et petite, positive et grande, la valeur maximale positive ainsi que le cas spécial 0 ainsi qu'une probabilité négative, les valeurs limites 0.0 et 1.0, médiane de l'intervalle d'entrées valides ainsi que supérieur à 1.0. Par contre, dû au fait que c'est valeurs nécessiterait 125 tests pour un jeu de tests AC (5 x 5 x 5) nous n'avons utilisé que 3 valeurs pour chaque paramètre. Nous obtenons donc 5 tests pour EC et 18 tests pour AC (3 x 3 x 3). Les valeurs limites sont testées pour les tests EC mais pas toutes les limites sont testées en AC. Malgré tout, chaque classe d'équivalence est testée et nous pouvons dire que notre confiance en la méthode est bonne. Pour plus de rigueur et sans avoir à tester 125 fois, un jeu de tests « Basic Choice » aurait pu être réalisé.

Graphe régulier

Ce type de graphe prend en paramètre un nombre de sommets ainsi que le degré des sommets. Ayant déjà ce type de graphe dans des cours tels que LOG2810: Structures discrètes, nous savions qu'une des conditions pour qu'un tel graphe existe est que V*k soit pair et avons donc décidé de tester ce cas. Nous avons donc deux cas de tests similaires ((11,2) et (5,3)) mais un est valide tandis que l'autre ne l'est pas à cause de cette condition. Sinon, le reste des valeurs sont comme les précédentes. Une valeur négative, positive, valeur limite supérieure et le cas spécial 0. Nous obtenons 6 tests avec EC et 36 tests avec AC (6 x 6). Nous constatons que des valeurs de chaque classe d'équivalence sont testées ainsi que les limites des entrées valides. Encore une fois, les tests permettent d'augmenter notre confiance en la méthode.