LOG1000- Ingénierie logicielle Processus de développement d'un projet logiciel open source

Travail pratique #4

Stephanie Mansour (1935596)

Mariam Sarwat (1928777)

Section : 02

École Polytechnique de Montréal

23 novembre 2018

E1) Diagramme de flot de contrôle

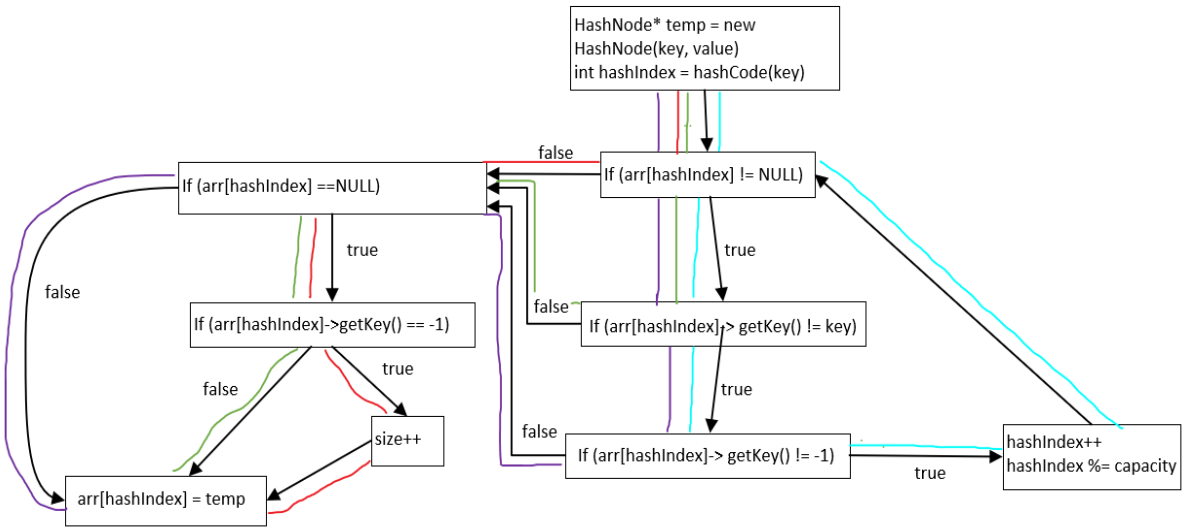

Figure 2 Diagramme de flot de la méthode void HashMap::insertNode(int key, int value) et les chemins possible pour couvrir les branches.

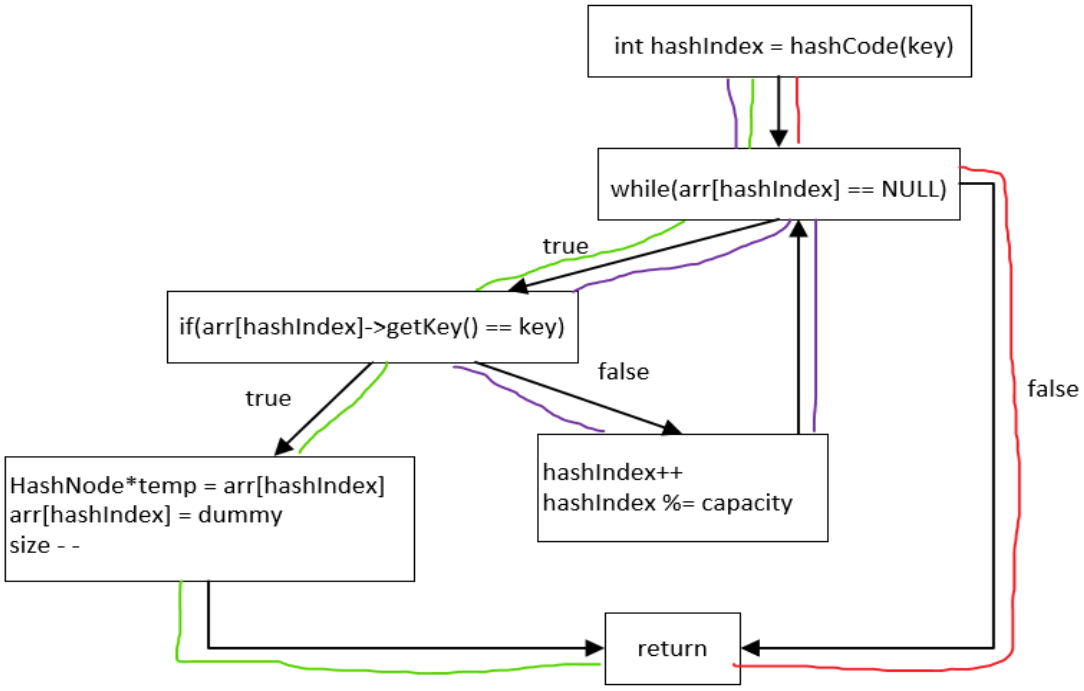

Figure 1 Diagramme de flot de la méthode int HashMap::deleteNode(int key) et les chemins possible pour couvrir les branches.

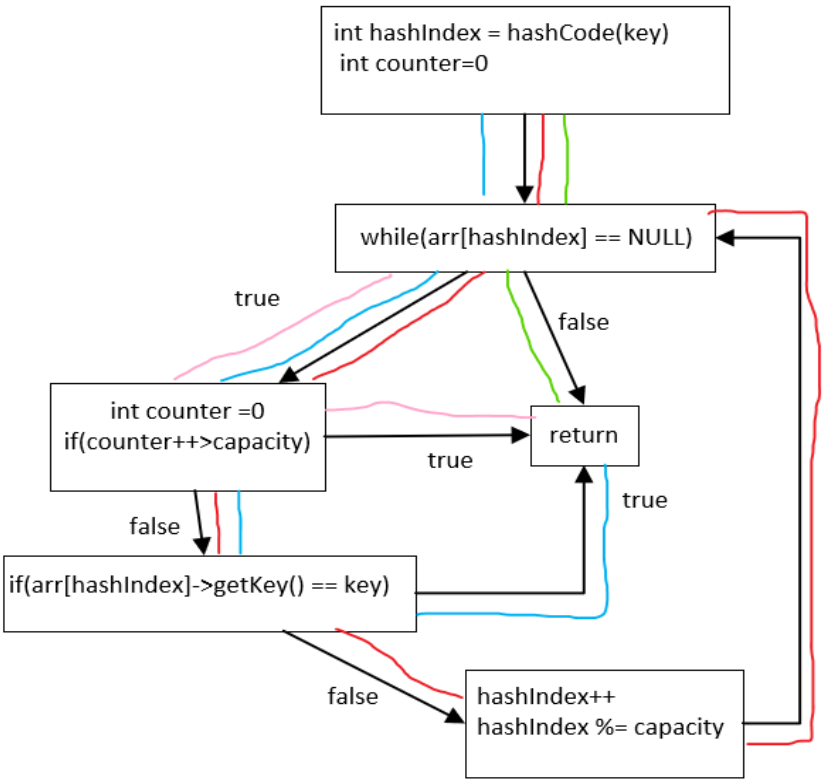

Figure 3 Diagramme de flot de la méthode int HashMap::get(int key) et les chemins possible pour couvrir les branches.

La complexité cyclomatique est calculer avec les deux approches suivantes :

i. M = NombrePointDeDecisions + 1

ii. M = NombreAretes – NombreNoeuds + 2

1.1. Pour la méthode HashMap::insertNode(int key, int value) :

i. M = 5 + 1 = 6

ii. M = 13 – 9 + 2 = 6

1.2. Pour la méthode HashMap::deleteNode(int key) :

i. M = 2 + 1 = 3

ii. M = 7 – 6 + 2 = 3

1.3. Pour la méthode HashMap::get(int key) :

i. M = 3 + 1 = 4

ii. M = 8 – 6 + 2 = 4

E2) Cas de tests et jeu de données

Méthode HashMap::insertNode(int key, int value) :

Comme cette méthode n'a pas de variable de retour, les variables pris en entrée ne vont pas affecter la sortie. Cette méthode permet seulement d'insérer une valeur et une clé qui lui est assigner dans un tableau.

Entrée	Sortie
key = 5, value = 2	Pas de sortie
key = 8, value = 0	Pas de sortie

Méthode HashMap::deleteNode(int key) :

Soit on prend en considération le tableau suivant :

key = 5, value = 2	key = 10, value = 1	key = 8, value = 0	key = 11, value = 5

Entrée	Sortie
key = 5	Retour = 2 En effet la méthode va enlever la première case du tableau et retournera la valeur correspondant au key pris en paramètre
key = 15	Retour = 0 En effet, c'est a cause que le key = 15 ne se retrouve pas dans le tableau donc il est contre-productif d'essai de l'enlevé du tableau

Méthode HashMap::get(int key) :

Soit on reprend le tableau précédent pour cette méthode.

Entrée	Sortie
key = 5	Retour value = 2 La méthode va retourner la valeur correspondant au key pris en paramètre
key = 9	Retour = 0 Car le tableau ne contient pas un key qui est égale à 9 (donc il n'y pas un valeur a retourner)

E3-E4) Implémentation et exécution des tests unitaires et débogage

Voici le résultat obtenu après avoir implémenter nos tests pour la première fois.

```
[masare@l4714-06 tests (master)] $ ./tests
insertTest::test1Segmentation fault (core dumped)
[masare@l4714-06 tests (master)] $
```

Nous avons donc supposé qu'il y avait des bugs qui nous laissaient même pas tester le code. Nous sommes donc aller révisé le fichier HashMap.cpp

Nous avons trouvé un bug à la ligne 46 : on a placé un OR au lieu d'un AND, pour laisser la possibilité d'entré dans la condition 'if' et avons obtenu le résultat suivant :

```
[masare@l4714-06 tests (master)] $ ./tests
insertTest::test1 : assertion
insertTest::test2 : OK
getTest::test1 : assertion
getTest::test2Segmentation fault (core dumped)
[masare@l4714-06 tests (master)] $
```

Ensuite, nous avons encore supposé qu'il y a avait un autre bug qui nous laissait pas exécuter nos tests. Après avoir révisé la fonction get de HashMap, nous avons réalisé que la condition de la boucle 'while' à la ligne 90 était écrite de tel sorte qu'il est impossible de rentrer dans celle-ci. Nous avons donc changer cette ligne : while (arr[hashIndex] != NULL) et avons tester de nouveau :

```
[masare@l4714-06 tests (master)] $ ./tests
insertTest::test1 : OK
insertTest::test2 : assertion
getTest::test1 : OK
getTest::test2 : OK
getTest::test3 : OK
deleteTest::test1Segmentation fault (core dumped)
[masare@l4714-06 tests (master)] $
```

Encore une fois, nous avons vérifié s'il y a eu un bug dans la méthode delete qui ne nous laissait pas exécuter nos tests. Nous avons réalisé que la condition de la boucle 'while' à la ligne 59 était écrite de tel sorte qu'il est impossible de rentrer dans celle-ci. Nous avons donc changer cette ligne : while(arr[hashIndex] != NULL) et nous avons tester de nouveau :

```
[masare@l4714-06 tests (master)] $ ./tests
insertTest::test1 : OK
insertTest::test2 : assertion
getTest::test1 : OK
getTest::test2 : OK
getTest::test3 : OK
deleteTest::test1 : OK
deleteTest::test2 : OK
deleteTest::test3 : OK
insertTest.h:53:Assertion
Test name: insertTest::test2
assertion failed
- Expression: 0 == hashMap->get(1)

Failures !!!
Run: 8   Failure total: 1   Failures: 1   Errors: 0
[masare@l4714-06 tests (master)] $
```

Comme l'indique la capture d'écran ci-dessus, nous avons fait une erreur dans notre test 2 pour la méthode insert, nous l'avons donc modifier pour créer le test voulu :

```
[masare@l4714-06 tests (master)] $ ./tests
insertTest::test1 : OK
insertTest::test2 : OK
getTest::test1 : OK
getTest::test2 : OK
getTest::test3 : OK
deleteTest::test1 : OK
deleteTest::test2 : OK
deleteTest::test3 : OK
OK (8)
[masare@l4714-06 tests (master)] $
```

Tous les tests passent, comme voulu. La capture suivante indique la réponse obtenue en exécutant le main :

```
[masare@l4714-06 TP4 (master)] $ ./EXEC
key = 1  value = 1
key = 2  value = 3
2
3
1
0
0[masare@l4714-06 TP4 (master)] $
```

E5) Contribution au projet Ring

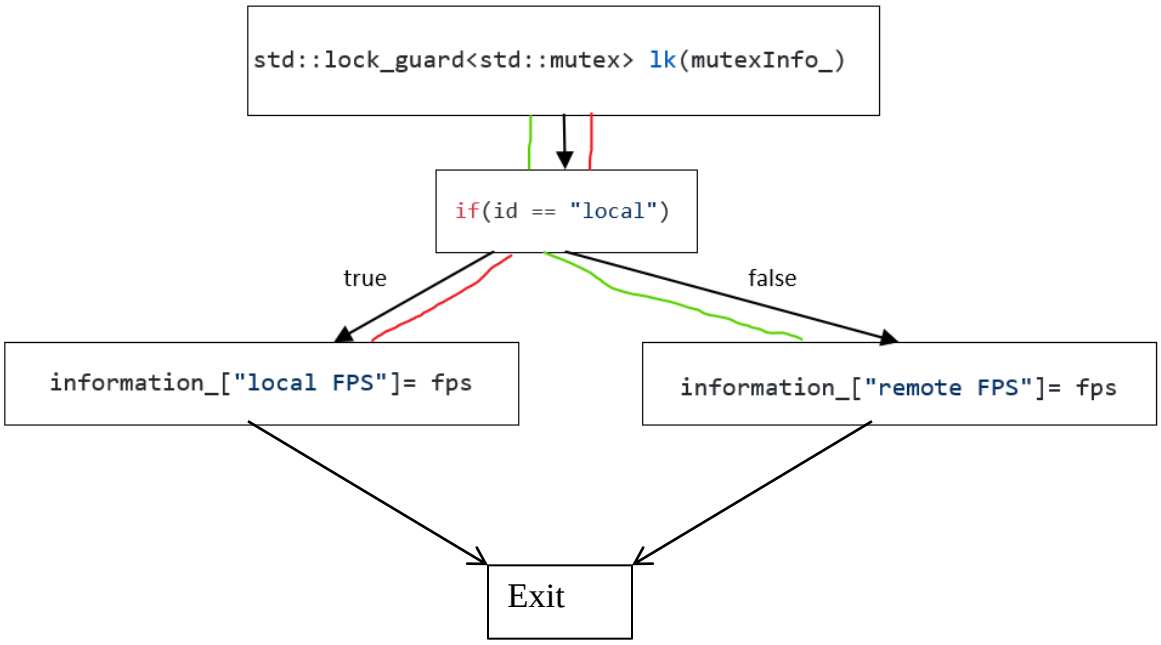

Figure 4 Diagramme de flot de la méthode Void Smartools::setFrameRate(const std::string& id, const std::string& fps) et les chemins possible pour couvrir les branches.

La complexité cyclomatique est calculée avec les deux approches suivantes :

iii. M = NombrePointDeDecisions + 1 = 1 + 1 = 2

iv. M = NombreAretes – NombreNoeuds + 2 = 5 - 5 + 2 = 2

Les tests qui étaient conçues couvrent tous les chemins que nous avons trouvé : un qui insère un ''local'' comme ID et l'autre un ''remote''. Par contre, pour aller plus loin, on pourrait insérer un ID qui n'est pas un string pour être s'assurer que la méthode s'échoue.