LOG 6305: Task 1

Génération de cas de test aléatoire

Giulio Antoniol
Rodrigo Morales
Département Génie Informatique et Génie Logiciel
École Polytechnique de Montréal, Québec, Canada
gulio.antoniol[at]gmail.com
moar82[at]hotmail.com

1 Identification

Student's name: Isnaldo Francisco de Melo jr

Date of the reading note: 30th of January, 2017

2 Introduction

This task covers random testing, which is a black-box software testing that generates several independent inputs. In this task, we implemented a pseudo-random test using the java.util.Random generator to test a Evaluation Triangle software. This test generated homogeneous distributed data to validate the triads of the Triangle software.

According to bibliography, this solution is cheap to implement, easy to understand and actually used in the industry [2]

3 Questions

1. Pourquoi est-il plus complique d'atteindre une couverture elevee lorsque l'intervalle des entiers augmente ? Par exemple, en utilisant des nombres alatoires compris entre 0 et 100 en comparaison de 0 1000.

The branch coverage is not related with the number of generated inputs, but it is related with the diversity among them. Therefore, no simple directly proportional relation — the bigger the interval, the more cases found — can be established. By increasing the interval, more combinations of invalid triads are generated and not necessarily the quantity of valid triads will increase.

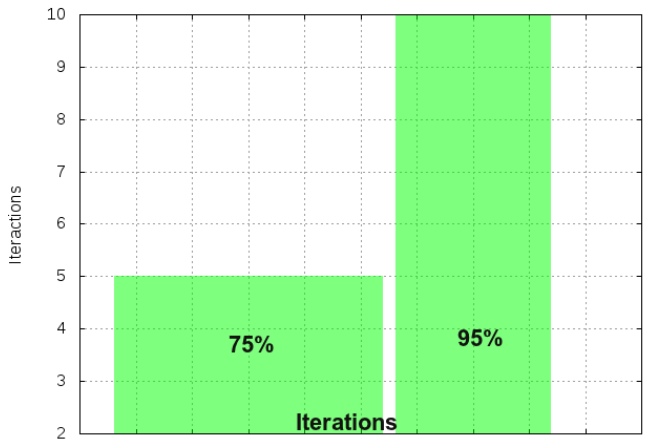
2. Dans vos expriences, tes-vous capable d'atteindre une couverture de 100 percent ? Si ce n'est pas le cas, expliquer pourquoi?

The test was able to reach the 19 cases described according to the random parameters used. While using the interval 0 to 100 the coverage took more time than 0 to 10, and less time than 0 to 1000. As a conclusion, the smaller the interval, the faster seem the coverage. To confirm this information, a log file was written.

On this case then we need k n log(n) test cases on average to cover all targets, or 19/19. This expression can be used to calculate the time to generate the 19 possible values and it is deduced from the scaling law for the Coupon Collector Problem. On this case then we need k n log(n) test cases on average to cover all targets [1].

The fact that 100 percent of the branches was covered does not mean the program has no bugs. It just confirms the branches could be reached with no major problems.

Figure 1: Iteration Plot - mean of several runs



3. A votre avis, quelles sont les limites de la generation de nombres aleatoires comme donnees d'essai et comment pouvons-nous surmonter ces limitations pour atteindre une couverture elevee dans les tests logiciels

The random data generation has limitations because it is not taken in consideration similar cases, which already happen. Therefore, adding more random cases not necessarily increases the coverage. Besides, those similar cases, useless inputs can be generated by the random generation.

An approach to increase the coverage, would be to add some assumptions that could be used to improve the generated inputs.

Another suggestion would be to use Coupons Collector Approach, which is a more advanced probability theory related with winning contests[1].

References

[1] J. W. DURAN and S. C. NTAFOS. Probabilistic analysis of random test generation method for irredundant combinational logic networks. *IEEE Transactions on Computers*, C-24(7):691–695, 1975.

[2] T. Target. Implementing agile in very large enterprises. `http://searchsoftwarequality.techtarget.com/feature/Implementing-Agile-in-very-large-enterprises`. [Online; accessed 19-January-2017].