Travail Pratique #2

Tests combinatoires et couverture de code

Marc-André Jolicoeur 1846304
Paul-Arthur Thiéry 1969108

Présenté à

Hiba Bagane

Département de Génie Informatique et Logiciel
Polytechnique de Montréal

15 Octobre 2018

1. Retour sur le TP1

Dans le cadre du TP1, nous avons appris à utiliser Mocha et Chai afin de réaliser des tests unitaires en Node.js. La difficulté relevait principalement de la compréhension du code lié aux appels d'Api afin d'en créer des stubs, et de l'apprentissage de Mocha et Chai (notamment identifier quel outil était le plus adapté à notre situation, parmi le *Mock*, le *Stub* et le *Spy)*. Par exemple, pour simuler les appels API de notre méthode, nous avons utilisé un *Stub*, mais un *Mock* aurait aussi pu être utilisé. De plus, nous avons du écrire de nombreux tests afin de tester tous les paramètres des objets créés durant les différentes étapes.
De plus, l'orthographe et la syntaxe des méthodes de Mocha et Sinon.js nous ont causé des problèmes. En effet, n'étant pas familier aux tests unitaires, nous avons fouillé dans la documentation de Mocha et Sinon.js pour trouver les méthodes utiles pour nos tests.

Au final, ce premier Lab a demandé une assez grande charge de travail et une bonne compréhension du code fourni mais nous a permis de découvrir des outils modernes et flexibles pour la réalisation de test unitaires

2. Partie 4.1

Mise en contexte

Dans le cadre du laboratoire, nous avons dû tester les méthodes de génération de graphes simples, bipartis et réguliers de la classe *GraphGenerator*. Nous avons d'abord utilisé la méthode en boite noire pour élaborer des tests de catégorie-partition qui satisfont le critère d'adéquation **EC** (*Each Choice*). Puis en boite blanche pour élaborer des tests AC (*All Choice*).

Pour élaborer des tests de catégorie-partition, il faut d'abord identifier les paramètres et les catégories impliqués dans la fonction. Il faut ensuite trouver l'ensemble des valeurs possible, les valeurs limites (min et max) ainsi que des valeurs à l'extérieur des limites (min-, max+). Les tests à effectuer dépendent du critère d'adéquation choisi.

Pour satisfaire le critère EC, chaque choix de paramètre (min-, min, possible, max max+) qui peut avoir une influence sur la sortie doit être testé. Pour satisfaire AC, tous les choix de paramètres doivent être testé avec tous les autres choix. Par contre, dans les deux cas, si un des paramètres cause une erreur ou une exception, on ne doit pas le tester avec tout les autres paramètres puisqu'on est certain que la sortie est erronée.

Pertinence des jeux de tests

Prenons *simple()* en exemple pour expliquer le processus de test. La fonction prend deux paramètres en entrés, V pour *Vertices* et E pour *Edges*. Le graphe peut avoir n'importe quel nombre de sommets, entre 0 et l'infini. Par contre, le nombre de d'arrêtes, lui, dépend du nombre de sommets. En effet, on trouve que E doit se situé entre 0 et $\frac{V*(V-1)}{2}$.

Ainsi, une fois les limites établies ($V_{min} = 0, V_{max} = \infty, E_{min} = 0, E_{max} = \frac{V*(V-1)}{2}$), on trouve des valeurs hors de ces limites pour tester les entrées erronées ($V_{min-} - 1,\ E_{min-} = -1,\ E_{max+} = \frac{V*(V-1)}{2} + 1$). Puis on établi un intervalle de donnée possible ($V_{possible} = [1, \infty],\ E_{possible} = [1, \frac{V*(V-1)}{2}]$). On se retrouve alors avec plusieurs choix de valeurs pour chaque variable. On peut ensuite les énumérer

V : min- (-1), min (0), V(1 à 2147483647[1])

E : min- (-1), min (0), E(1 à $\frac{V*(V-1)}{2} - 1$), max ($\frac{V*(V-1)}{2}$), max+ ($\frac{V*(V-1)}{2} + 1$)

V1 = { V = -1 } {erreur}
V2 = { V = 0 } {correct}
V3 = { V = 2147483647 } { correct }
E1 = { E = -1 } {erreur}
E2 = { E = 0 } { correct }
E3 = { E = 1 – ($\frac{V*(V-1)}{2} - 1$) } { correct }
E4 = { E = $\frac{V*(V-1)}{2}$ } { correct }
E5 = { E = $\frac{V*(V-1)}{2} + 1$ } {erreur}

[1] Valeur maximale pour un *int*