TP2

Test Combinatoire et Couverture de Code

Jacob Dorais

Billy Bouchard

Gr 02

Un Travail présenté à :

Hiba Bagane

Département Génie informatique et Logiciel

Polytechnique Montreal

le lundi 15 octobre

1 Retour sur le TP 1

1.1 Le language utilisé

Le travail pratique précédent a été fait en javascript sous le module nodejs. Il est très pertinent d'apprendre à utiliser les différentes librairies de test pour nodejs, car elles sont beaucoup plus souvent utilisées maintenant. De plus, le langage web devient de plus en plus le langage le plus utilisé en programmation. NodeJs permettait d'utiliser des librairies comme Sinon, Chai et Sinon-Chai qui sont simple d'utilisation et extrêmement puissante pour faire plusieurs types différents d'objets. Avec une bonne lecture des librairies, il était facile de comprendre comment et à quoi ces différents objets servent. Le fait que le langage soit faiblement typé simplifiait beaucoup les différentes opérations. On devait moins s'attarder sur les types, mais plus sur les méthodes et les tests.

1.2 Le code fourni

Le code fourni était définitivement trop complexe pour le but de l'apprentissage. En effet, le but du TP était d'apprendre les bases des méthodes de test, notamment les objets utilisés lors des tests, mais aussi les différents types de tests à effectuer en fonctions de différentes méthodes. La quantité de classes données, ainsi que leur contenu était beaucoup trop complexe pour que le seul but du lab soit d'apprendre les tests. Malgré qu'aucun cours sur les promesses n'a été données, il fallait maintenant implémenter des tests qui devait simuler (stub) certaines promesses. Les fonctions dans lesquels les tests étaient effectués appelaient d'autres fonctions qui utilisait la même méthode. Cette longue transition rendait difficile de percevoir comment tester chacune de ces méthodes de la bonne facons. La grosseur de ces méthodes privées rendait aussi compliqué de comprendre les tests à effectuer. On pouvait alors difficilement identifier tous les tests qui devait être effectué pour chacune des différentes méthodes, car elle dépendait de plusieurs autres méthodes qui elles étaient privées et donc peu accessible. Il aurait été préférable d'avoir seulement une fonction privée à tester.

1.3 L'information fourni

L'information qui était donnée au début du TP était peu suffisante afin d'en comprendre sa complexité. Il manquait beaucoup d'information pour expliquer comment le code fourni fonctionnait ce qui a causé une grande perte de temps à comprendre le code fourni. Cependant, la tache était bien décrite et elle permettait très bien de connaitre les fonctions à tester. On pouvait facilement suivre notre progression dans les fonctions à tester et de s'assurer de notre compréhension

1.4 Le tp en général

Le niveau de difficulté du TP était bon, mais peut être un peu trop long à réaliser. En effet, il aurait été beaucoup plus pertinent de tester les moins de paramètre d'initialisation (nous aurions appris la même chose qu'il y ait 15 propriétés ou 3). Cependant, pour apprendre les différent type de stub, spy et mock, le TP a vraiment permis de comprendre les cas d'utilisation de chacun et de mieux comprendre leurs utilités

2 Expliquez comment vous avez obtenu vos cas de test pour la partie 4.1

2.1 Graph Simple avec varible E et V

Dans tous les cas de test, la variable V est le nombre de noeuds que le graphe doit posséder.

Dans le cas de la fct `simple(int V, int E)` la variable E stipule le nombre de liens entre chaque noeud V_x. Par conséquent, il est impossible d'avoir plus d'arrête que la somme de 0 jusqu'au nombre de noeuds, car sinon on aurait plus que 1 arrête entre 2 noeuds et le graph ne serait plus un graph simple. dans le tableau suivant, on peut voir les différentes valeurs que peuvent prendre les 2 variables :

Choix		propriétés
v_1	v < 0	erreur
v_2	v $== 0$	ok, vide
v_3	v > 0	ok
e_1	e $== 0$	ok
e_2	$0 <$ e $< \sum_0^{V-1}$	ok
e_3	e $> \sum_0^{V-1}$	erreur
e_4	e < 0	erreur

2.1.1 Each choice

Nous avons simplement fais 5 choix de test dans lesquels nous testons tous les e et tout les v de facon indépendante (en ne combinant rien pour tester les erreurs) : V1, V2E1, V3E2, E3 et E4

2.1.2 All choice

Pour le All Choice, nous avons ajouté toutes les combinaisons possibles sauf dans les cas ou le V est une erreur (il devenait alors inutile de tester si e allait échouer ou réussir) pour un total de 9 tests.

2.2 Graph simple avec e et p

Dans le cas de la fct `simple(int V, double P)` la variable p stipule la probabilité d'un lien entre deux neouds V_i et V_j. Par conséquent, lorsqu'il vaut 0, il est impossible d'avoir des liens entre les noeuds, lorsqu'elle vaut 1 il est sûr et certain que le graphe sera complet. Voici un tableau des valeurs possible des variables :

Choix		propriétés
v_1	v < 0	erreur
v_2	v $== 0$	ok, vide
v_3	v > 0	ok
p_1	p $== 0$	ok
p_2	$0 <$ p < 1	ok
p_3	p $== 1$	erreur
p_4	p < 0	erreur
p_5	p > 0	erreur

2.2.1 Each Choice

Pour le each choice, nous avons simplement testé les 4 cas d'erreurs avec une valeur ok pour sa valeur combiné, puis fais 2 test différents pour les les 4 autres valeurs ok à tester.

2.2.2 All Choice

De la même facon, nous avons testé toutes les combinaisons entre elles sauf pour le v1 que nous avons testé seul car il ferais échouer tout les autres test.

2.3 Graph Biparti v1, v2, e

le graph biparti permet d'obtenir un graphe qui contient 2 sous graphe de v1 et v2 nombre de noeuds respectivement avec un total de e liens entre eux. Nous avons analysé le problème de la facon

suivantes :

Choix		propriétés
$v1_1$	v < 0	erreur
$v1_2$	v $== 0$	ok, vide
$v1_3$	v > 0	ok
$v2_1$	v < 0	erreur
$v2_2$	v $== 0$	ok, vide
$v2_3$	v > 0	ok
e_1	e < 0	ok
e_2	$0 <$ e $<$ v1 $\times$ v2	ok
e_3	e $>$ v1 $\times$ v2	erreur

2.3.1 Each Choice

Pour le each choice, nous avons testé les erreurs avec des valeurs ok et testé tout les autres valeur en minimisant le nombre de test nécéssaires.

2.3.2 All Choice

Pour le All choice, nous avons testé de manière similaire aux autres méthodes, c'est-à-dire que nous avons testé toutes les combinaisons possible qui ne génèrent pas d'erreurs de manière individuelle.

2.4 Graph Biparti v1, v2, p

Le graphe biparti permet d'obtenir un graphe qui contient 2 sous graphes de v1 et v2 et une probabilité de p pour la possibilité d'avoir un lien entre les différents noeuds. Nous avons analysé le

problème de la facon suivantes :

Choix		propriétés
$v1_1$	v < 0	erreur
$v1_2$	v $== 0$	ok, vide
$v1_3$	v > 0	ok
$v2_1$	v < 0	erreur
$v2_2$	v $== 0$	ok, vide
$v2_3$	v > 0	ok
p_1	p < 0	ok
p_2	$0 \leq$ p ≤ 1	ok
p_3	p > 1	erreur

2.5 Graph régulier

Pour le graphe régulier, le paramètre v revenait. Cependant, le paramètre k s'ajoutait. Il s'agit du nombre d'arc par noeuds (donc le niveau du graph). Ce dernier ne peux pas être plus grand que le nombre de noeud dans le cas d'un graphe simple. De plus, lorsque l'on multiplie le nombre de noeud avec le nombre d'arc, le total ne doit pas être impair. Voici les détails de notre analyse :

Choix		propriétés
v_1	v < 0	erreur
v_2	v $== 0$	ok, vide
v_3	v > 0	ok
k_1	k < 0	erreur
k_2	$0 <$ k $<$ v-1	erreur
k_3	k $>$ v - 1	ok
k_4	k $== 0$	ok
k_5	(k $\times$ v) *modulo* 2 $== 1$	erreur

Nous avons procédé de la même facon que les autres test afin de trouver les test all choice et each choice a partir du tableau suivant. Pour le cas de test où nous effectuons le choix k2, nous nous somme attendu à une erreur, mais la fonction n'en a pas généré : notre test échoue. Nous supposons que la fonction retourne un graphe qui n'est pas simple et donc autorise les multigraphe.