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 a utilisé les différentes librairies de test pour nodejs, car elles sont beaucoup plus souvent utilisée maintenant. De plus, le language web devient de plus en plus le language le plus utilisé en programmation. NodeJs permettait d'utilisé des librairies comme Sinon, Chai et Sinon-Chai qui sont simple d'utilisation et extrêmement puissantes pour faire plusieurs types différents d'objets. Avec une bonne lecture des librairies, il était facile de comprendre comment et a quoi ces différents objets servent. Le fait que le language soit faiblement typé simplifiait beaucoup les différentes opérations. On devait moins s'attarder sur les types mais plus sur les méthodes et les test.

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 base des méthodes de test, notamment les objets utilisé lors des tests, mais aussi les différents type de test a 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 test. Malgré qu'aucun cours sur les promesses est été données, il fallait maintenant implémenter des test qui devait simuler (stub) certaines promesses. Les fonctions dans lesquels les test étaient effectuer appelais d'autre fonctions qui utilisait la même méthodes. Cette longue transistions rendais 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 test à effectuer. On pouvait alors difficilement identifier tous les test qui devait être effectuer pour chacune des différentes méthodes car elles dépendait de plusieurs autres méthodes qui elles étaient privé et donc peu accessible. Il aurait été preférable d'avoir seulement une fonction privée a testé.

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 se qui a causer une grande perte de temps a comprendre le code fourni. Cependant, la tache était bien décrite et elle permettait très bien de connaitre les fonctions a tester. On pouvait facilement suivre notre progression dans les fonctions a tester et de s'assurer de notre compréhension

1.4 Le tp en général

Le niveau de difficulté du tp était bon mais peu être un peu trop long a 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 propriété 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 varible V est le nombre de noeud que le graph doit posséder. Dans le cas de la fct `simple(int V, int E)` la variable E stipule le nombre de lien entre chaque noeud V_x. Par conséquent, il est impossible d'avoir plus d'arrête que la somme de 0 jusqu'au nombre de noeud, car sinon on aurait plus que 1 arrête entre 2 noeud et le graph ne serait plus un graph simple. dans le tableau suivant on peu voir les différentes valeur que peut 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 on test tout 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 ajouter tous les combinaisons possibles sauf dans les cas ou le V était une erreur (il devenait alors inutile de tester si e allait echouer ou reussir) pour un total de 9 test

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 sur et certain que le graph 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 tester les 4 cas d'erreures avec une valeur ok pour sa valeur combiner puis fais 2 test différents pour les les 4 autres valeurs ok a tester

2.2.2 All Choice

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

2.3 Graph Biparti v1, v2, e

le graph biparti permet d'obtenir un graph qui contient 2 sous graph de v1 et v2 nombre de noeuds respectivement avec un total de e liens entre eux. Nous avons analyser le problem 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 × v2	ok
e_3	e > v1 × v2	erreur

2.3.1 Each Choice

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

2.3.2 All Choice

pour le All choice, nous avons tester les erreurs avec des valeurs ok(pas 2 erreurs entres elles) et tester tout les autres valeurs en les combinant avec toutes les valeurs des autres variables possible

2.4 Graph Biparti v1, v2, p

le graph biparti permet d'obtenir un graph qui contient 2 sous graph de v1 et v2 et une probabilité de p pour la possibilité d'avoiur un lien entre les différents noeuds. Nous avons analyser le problem 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 ≤ p ≤ 1	ok
p_3	p > 1	erreur

2.5 Graph régulier

Pour le graph 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 etre plus grand que le nombre de noeud dans le cas d'un graph parafit. 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	ok
k_2	0 < k < v-1	ok
k_3	k > v - 1	erreur
k_4	k == 0	erreur
k_5	(k × 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.