Rapport TP6

Vincent Chassé 1795836 Gabriel Bourgault 1794069

Nous avons implémenté les cas de tests dans des fichiers distincts pour chacune des classes à tester, soit Bag, BipartiteXExtended, BipartiteX, GraphGenerator, Graph et Stack. Nous avons décidé de ne pas tester StdOut car nous assumons qu'il s'agit d'une classe utilitaire. Ce que nous avons fait cependant, afin d'être en mesure d'effectuer les tests selon l'ordre topologique, c'est créer une nouvelle classe de tests nommée TestComplet. Exécuter cette classe fera en sorte d'appeler tous les tests nécessaires.

Vous retrouverez le diagramme ORD, le diagramme d'ordre topologique, le tableau de CFW et le tableau des niveaux dans des fichiers pdf bien identifiés.

Objectifs et limites des chaque méthode de test

TP1 : Pour le TP1, il nous a suffi de tester une fois chaque fonction afin de satisfaire les critères. Cette méthode est peu efficace et ne permet pas souvent de trouver des défaillances.

TP2 : Ici il fallait faire des tests boîte noire avec la technique catégorie partition (Each Choice). Il s'agissait donc de tester chaque combinaison possible de valeurs d'entrées selon les catégories que nous avions définies. Cette méthode permet de trouver beaucoup de défaillances, mais la quantité de tests à écrire était relativement élevée.

TP3 : Cette fois-ci, il s'agissait d'effectuer des tests boîte blanche. En reprenant les tests du TP2, nous avons analysé la couverture de code avec Jacoco, ce qui s'avère être un outil très intéressant! Avec les branches non-couvertes, nous avons donc créé des tests qui permettaient spécifiquement de les franchir. Bien que toutes les branches soient couvertes, il peut toujours y avoir des erreurs qui ne seront pas trouvées par cette méthode. En fait, il semble que lorsque l'on se fie uniquement à la couverture du code comme métrique, on tombe souvent dans le piège de penser que notre code est parfait.

TP4 : L'objectif était de tester les transitions en s'inspirant de l'arbre de transition. Il fallait donc couvrir tous les chemins de l'arbre de transition. Ces tests permettent de trouver des défaillances au niveau des changements d'état, mais pas nécessairement des défaillances comme les valeurs des paramètres d'entrée.

TP5 : Il fallait ici tester la classe Queue à l'aide d'un tableau de MaDUM. Ces tests se révèlent intéressants, car ils permettent de s'assurer que la valeur de la variable est correcte après chaque étape impliquant potentiellement une transformation. Cependant, cela crée des tests complexes qui s'étendent sur plusieurs lignes et qui ont des chances de rater à plusieurs endroits. Il est mieux d'avoir des tests plus précis qui ne valident qu'une seule chose.

TP6 : Dans ce cas-ci, nous nous sommes inspirés des diagrammes ORD et d'ordre topologique afin de définir un certain ordre de tests. Nous avons majoritairement réutilisé les cas de tests produits lors des TP précédents, alors à ce niveau il n'y a pas beaucoup à dire. Cependant, nous avons dû fouiller pas mal afin de trouver comment définir l'ordre des tests.