Rapport TP2

Vincent Chassé 1795836 Gabriel Bourgault 1794069

Avec les tests que nous avons élaborés, il est facile de constater que la stratégie Each Choice ne permet pas de couvrir de façon très approfondie. Notre stratégie pour All Combinations est de non seulement tester toutes les combinaisons d’entrées selon nos classes, mais aussi de tester les valeurs limites. Dans certains cas, une valeur n’a pas nécessairement de maximum, alors nous testons seulement le min et le min plus. Lorsque nous testons une fonction avec les valeurs limites, nous nous assurons qu’une seule des variables est à la limite à la fois. C’est-à-dire que les autres variables d’entrées ne sont pas des valeurs limites.

## Graphe Simple

testSimpleEC\_1() :

testSimpleEC\_2() :

testSimpleAC\_1() :

testSimpleAC\_2() :

testSimpleAC\_3() :

testSimpleAC\_4() :

testSimpleLimite\_MinV() :

testSimpleLimite\_MinPlusV() :

testSimpleLimite\_MinP() :

testSimpleLimite\_MinPlusP() :

testSimpleLimite\_MoyP() :

testSimpleLimite\_MaxMoinsP() :

testSimpleLimite\_MaxP() :

## Graphe Bipartite

Pour ce type de graphe, nous avons identifié les classes d’entrées suivantes :

V1 : Valide, Invalide

V2 : Valide, Invalide

testBipartiteEC\_1() :

testBipartiteEC\_2() :

testBipartiteAC\_1() :

testBipartiteAC\_2() :

testBipartiteAC\_3() :

testBipartiteAC\_4() :

testBipartiteAC\_5() :

testBipartiteAC\_6() :

testBipartiteAC\_7() :

testBipartiteAC\_8() :

Pour les tests des valeurs limites, étant donné que V1 et V2 n’ont pas de maximum, nous avons seulement pu tester le min et le min plus pour ceux-là. Par contre, nous avons testé le P au complet, car il admet un maximum (1.0).

testBipartiteLimite\_MinV1() :

testBipartiteLimite\_MinPlusV1() :

testBipartiteLimite\_MinV2() :

Vincent

//Séparation

Gab

testBipartiteLimite\_MinPlusV2() : On vérifie ici que la création est valide avec V2 = 1, ce qui est le min plus.

testBipartiteLimite\_MinP() : On vérifie ici que la création est valide avec P = 0.0, ce qui est le min.

testBipartiteLimite\_MinPlusP() : On vérifie ici que la création est valide avec P = 0.1, ce qui est le min plus.

testBipartiteLimite\_MoyP() : On vérifie ici que la création est valide avec P = 0.5, ce qui est la moyenne.

testBipartiteLimite\_MaxMoinsP() : On vérifie ici que la création est valide avec P = 0.9, ce qui est le max moins.

testBipartiteLimite\_MaxP() : On vérifie ici que la création est valide avec P = 1.0, ce qui est le max.

## Graphe Regular

Pour ce type de graphe, nous avons identifié les classes d’entrées suivantes :

V : Pair, Impair, Invalide

K : Pair, Impair, Invalide

Pour Each Choice, nous nous sommes assures que chacune de nos classes était utilisée pour V et pour K.

testRegularEC\_1() : On vérifie ici les valeurs paires, c’est-à-dire V et K pairs.

testBegularEC\_2() : On vérifie ici les valeurs impaires, c’est à dire V et K impairs. On s’assure que la création du graphe lance une exception de type IllegalArgumentException.

testBegularEC\_3() : On vérifie ici les valeurs invalides, c’est à dire V et K invalids. On s’assure que la création du graphe lance une exception de type IllegalArgumentException.

Pour All Combinations, nous avons testé chaque combinaison de classe d’entrée.

testRegularAC\_1() : On vérifie ici que la création est valide avec V pair et K pair.

testBegularAC\_2() : On vérifie ici qu’une exception est lancée avec V pair et K invalide.

testBegularAC\_3() : On vérifie ici que la création est valide avec V pair et K impair.

testBegularAC\_4() : On vérifie ici que la création est valide avec V pair et K impair.

testBegularAC\_5() : On vérifie ici qu’une exception est lancée avec V impair et K invalide.

testBegularAC\_6() : On vérifie ici qu’une exception est lancée avec V invalide et K pair.

testBegularAC\_7() : On vérifie ici qu’une exception est lancée avec V invalide et K pair.

testBegularAC\_8() : On vérifie ici qu’une exception est lancée avec V invalide et K invalide.

testBegularAC\_9() : On vérifie ici qu’une exception est lancée avec V invalide et K impair.

Pour les tests des valeurs limites, étant donné que V et K n’ont pas de maximum, nous avons seulement pu tester le min et le min plus pour les classes pair et impair.

testBegularLimite\_MinPairV() : On vérifie ici que la création est valide avec V = 0, ce qui est le min pour la classe pair.

testBegularLimite\_MinPairPlusV() : On vérifie ici que la création est valide avec V = 2, ce qui est le min plus pour la classe pair.

testBegularLimite\_MinImpairV() : On vérifie ici que la création est valide avec V = 1, ce qui est le min pour la classe impair.

testBegularLimite\_MinImpairPlusV() : On vérifie ici que la création est valide avec V = 3, ce qui est le min plus pour la classe impair.

testBegularLimite\_MinPairK() : On vérifie ici que la création est valide avec K = 0, ce qui est le min pour la classe pair.

testBegularLimite\_MinPairPlusK() : On vérifie ici que la création est valide avec K = 2 ce qui est le min plus pour la classe pair.

testBegularLimite\_MinImpairK() : On vérifie ici que la création est valide avec K = 1, ce qui est le min pour la classe impair.

testBegularLimite\_MinImpairPlusK() : On vérifie ici que la création est valide avec K = 03 ce qui est le min plus pour la classe impair.