

# 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

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

V : Valide, Invalide

Un V valide est une valeur entière plus grande ou égale à 0. Un V invalide est une valeur inférieure à 0.

p : Valide, Invalide

Un p valide est un nombre réel entre 0 et 1 inclusivement. Un p invalide est toute valeur en dehors de cet intervalle.

Pour « Each Choice », nous nous sommes assuré que chacune de nos classes était utilisée pour V et pour p au moins une fois.

testSimpleEC\_1() : On s'assure que la création d'un graphe avec un V valide de 15 et un p invalide de 1.01 lance une exception de type `IllegalArgumentException`.

testSimpleEC\_2() : On s'assure que la création d'un graphe avec un V invalide de -1 et un p valide de 0.32 lance une exception de type `IllegalArgumentException`.

Pour « All Combinations », nous avons testé chaque combinaison de classe possible de V et p.

testSimpleAC\_1() : On s'assure que la création d'un graphe avec un V valide de 15 et un p invalide de 1.01 lance une exception de type `IllegalArgumentException`.

testSimpleAC\_2() : On s'assure que la création d'un graphe avec un V invalide de -6 et un p valide de 0.32 lance une exception de type `IllegalArgumentException`.

testSimpleAC\_3() : On s'assure que la création d'un graphe avec un V invalide de -3 et un p invalide de 1.03 lance une exception de type IllegalArgumentException.

testSimpleAC\_4() : On s'assure que la création d'un graphe avec un V valide de 4 et un p valide de 0.45 renvoie bien un graphe simple avec 4 sommets.

Le minimum possible de V est 0 mais il ne possède pas de maximum. Par contre, la variable p possède un minimum de 0, une moyenne de 0.5 et un maximum de 1. Nous avons décidé que le min+ et max- ont une différence de 0.1 par rapport au min ou au max.

testSimpleLimite\_MinV() : On s'assure que la création d'un graphe avec un V minimum de 0 et un p valide de 0.45 renvoie bien un graphe simple avec 0 sommet.

testSimpleLimite\_MinPlusV() : On s'assure que la création d'un graphe avec un V minimum+ de 1 et un p valide de 0.45 renvoie bien un graphe simple avec 1 sommet.

testSimpleLimite\_MinP() : On s'assure que la création d'un graphe avec un V valide de 4 et un p minimum de 0 renvoie bien un graphe simple avec 4 sommets et 0 arrêtes car la probabilité qu'il y ait une arrête entre deux sommets est nulle.

testSimpleLimite\_MinPlusP() : On s'assure que la création d'un graphe avec un V valide de 4 et un p minimum+ de 0.1 renvoie bien un graphe simple avec 4 sommets.

testSimpleLimite\_MoyP() : On s'assure que la création d'un graphe avec un V valide de 4 et un p moyen de 0.5 renvoie bien un graphe simple avec 4 sommets.

testSimpleLimite\_MaxMoinsP() : On s'assure que la création d'un graphe avec un V valide de 4 et un p maximum- de 0.9 renvoie bien un graphe simple avec 4 sommets.

testSimpleLimite\_MaxP() : On s'assure que la création d'un graphe avec un V valide de 4 et un p maximum de 1 renvoie bien un graphe simple avec 4 sommets.

## Graphe Bipartite

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

V1 : Valide, Invalide

V2 : Valide, Invalide

Un V valide est une valeur entière plus grande ou égale à 0. Un V invalide est une valeur inférieure à 0.

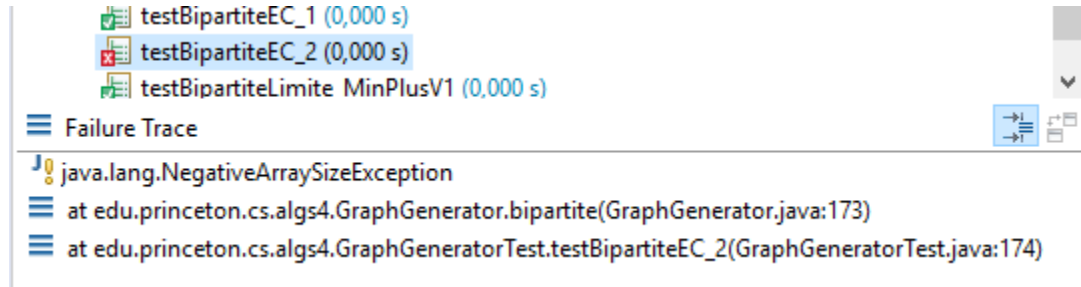
p : Valide, Invalide

Un p valide est un nombre réel entre 0 et 1 inclusivement. Un p invalide est toute valeur en dehors de cet intervalle.

Pour « Each Choice », nous nous sommes assuré que chacune de nos classes était utilisée pour V1, V2 et pour p au moins une fois.

testBipartiteEC\_1() : On s'assure que la création d'un graphe avec un V1 valide de 15, un V2 valide de 10 et un p invalide de 1.01 lance une exception de type IllegalArgumentException.

testBipartiteEC\_2() : On s'assure que la création d'un graphe avec un V1 invalide de -3, un V2 invalide de -4 et un p valide de 0.22 lance une exception de type IllegalArgumentException.



```
testBipartiteEC_1 (0,000 s)
testBipartiteEC_2 (0,000 s)
testBipartiteLimite MinPlusV1 (0,000 s)

Failure Trace

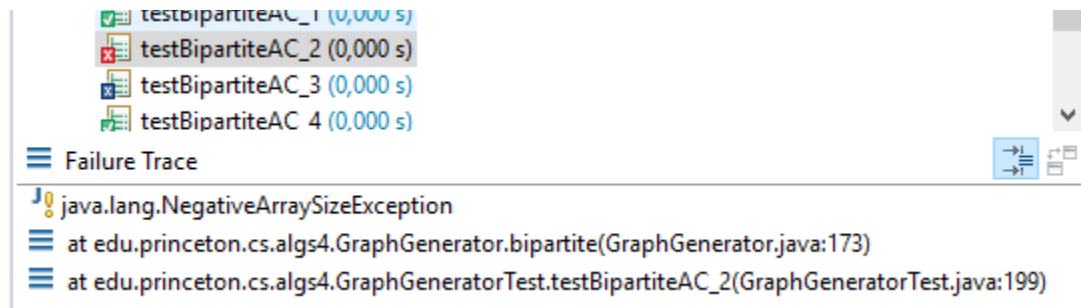
java.lang.NegativeArraySizeException
    at edu.princeton.cs.algs4.GraphGenerator.bipartite(GraphGenerator.java:173)
    at edu.princeton.cs.algs4.GraphGeneratorTest.testBipartiteEC_2(GraphGeneratorTest.java:174)
```

Par contre, ce test n'as pas passé. En effet, la fonction a plutôt retourné un NegativeArraySizeException. La fonction ne respecte donc pas la spécification parfaitement.

Pour « All Combinations », nous avons testé chaque combinaison de classe possible de V1, V2 et p.

testBipartiteAC\_1() : On s'assure que la création d'un graphe avec un V1 invalide, un V2 invalide et un p invalide lance une exception de type IllegalArgumentException.

testBipartiteAC\_2() : On s'assure que la création d'un graphe avec un V1 invalide, un V2 invalide et un p valide de 0.32 lance une exception de type IllegalArgumentException.



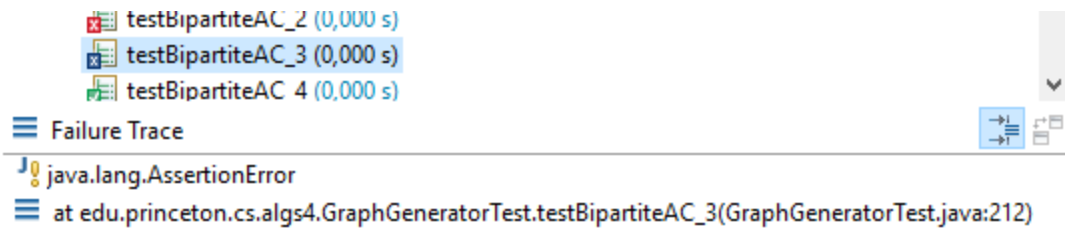
```
testBipartiteAC_1 (0,000 s)
testBipartiteAC_2 (0,000 s)
testBipartiteAC_3 (0,000 s)
testBipartiteAC_4 (0,000 s)

Failure Trace

java.lang.NegativeArraySizeException
    at edu.princeton.cs.algs4.GraphGenerator.bipartite(GraphGenerator.java:173)
    at edu.princeton.cs.algs4.GraphGeneratorTest.testBipartiteAC_2(GraphGeneratorTest.java:199)
```

Par contre, ce test n'as pas passé. En effet, la fonction a plutôt retourné un NegativeArraySizeException. La fonction ne respecte donc pas la spécification parfaitement.

testBipartiteAC\_3() : On s'assure que la création d'un graphe avec un V1 invalide, un V2 valide de 1 et un p valide de 0.32 lance une exception de type IllegalArgumentException.



```
testBipartiteAC_2 (0,000 s)
testBipartiteAC_3 (0,000 s)
testBipartiteAC_4 (0,000 s)

Failure Trace
java.lang.AssertionError
at edu.princeton.cs.algs4.GraphGeneratorTest.testBipartiteAC_3(GraphGeneratorTest.java:212)
```

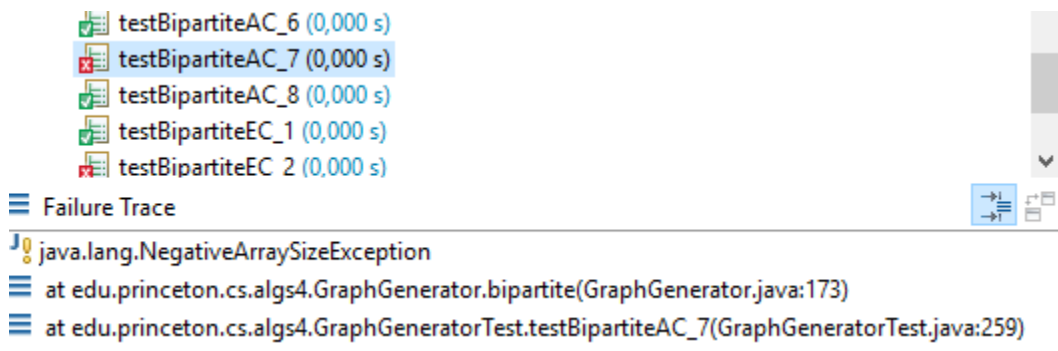
Par contre, ce test n'as pas passé. En effet, la fonction a été exécutée sans problème alors qu'elle aurait dû rencontrer une erreur à un certain point. La fonction ne respecte donc pas la spécification parfaitement.

testBipartiteAC\_4() : On s'assure que la création d'un graphe avec un V1 invalide, un V2 valide de 1 et un p invalide lance une exception de type IllegalArgumentException.

testBipartiteAC\_5() : On s'assure que la création d'un graphe avec un V1 valide de 1, un V2 invalide et un p invalide lance une exception de type IllegalArgumentException.

testBipartiteAC\_6() : On s'assure que la création d'un graphe avec un V1 valide de 1, un V2 valide de 1 et un p invalide lance une exception de type IllegalArgumentException.

testBipartiteAC\_7() : On s'assure que la création d'un graphe avec un V1 valide de 1, un V2 invalide et un p invalide lance une exception de type IllegalArgumentException.



```
testBipartiteAC_6 (0,000 s)
testBipartiteAC_7 (0,000 s)
testBipartiteAC_8 (0,000 s)
testBipartiteEC_1 (0,000 s)
testBipartiteEC_2 (0,000 s)

Failure Trace
java.lang.NegativeArraySizeException
at edu.princeton.cs.algs4.GraphGenerator.bipartite(GraphGenerator.java:173)
at edu.princeton.cs.algs4.GraphGeneratorTest.testBipartiteAC_7(GraphGeneratorTest.java:259)
```

Par contre, ce test n'as pas passé. En effet, la fonction a plutôt retourné un NegativeArraySizeException. La fonction ne respecte donc pas la spécification parfaitement.

testBipartiteAC\_8() : On s'assure que la création d'un graphe avec un V1 valide de 1, un V2 valide de 2 et un p valide de 0.32 renvoie bien un graphe bipartite avec 3 sommets.

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+ pour ceux-là. Par contre, la variable p possède un minimum de 0, une moyenne de 0.5 et un maximum de 1. Nous avons décidé que le min+ et max- ont une différence de 0.1 par rapport au min ou au max.

testBipartiteLimite\_MinV1() : On s'assure que la création d'un graphe avec un V1 minimum de 0, un V2 valide de 2 un p valide de 0.45 renvoie bien un graphe bipartite avec 2 sommets.

testBipartiteLimite\_MinPlusV1() : On s'assure que la création d'un graphe avec un V1 minimum+ de 1, un V2 valide de 2 un p valide de 0.45 renvoie bien un graphe bipartite avec 3 sommets.

testBipartiteLimite\_MinV2() : On s'assure que la création d'un graphe avec un V1 valide de 0, un V2 minimum de 0 un p valide de 0.45 renvoie bien un graphe bipartite avec 2 sommets.

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 les deux variables, la classe invalide comprend les valeurs négatives.

Pour « Each Choice », nous nous sommes assurés 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.

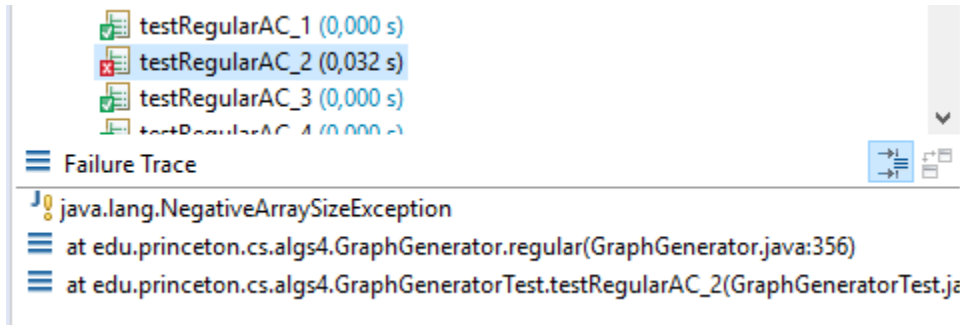
testRegularEC\_2() : On vérifie ici les valeurs impaires, c'est à dire V et K impairs. On s'assure que la création du graphe lance une exception de type IllegalArgumentException.

testRegularEC\_3() : On vérifie ici les valeurs invalides, c'est à dire V et K invalides. 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.

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



Par contre, ce test n'as pas passé. En effet, la fonction a plutôt retourné un NegativeArraySizeException. La fonction ne respecte donc pas la spécification parfaitement.

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

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

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

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

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

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

testRegularAC\_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.

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

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

testRegularLimite\_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.