1º Projecto
Verificação e Validação de Software

Gonçalo Cardoso fc46784

Pedro Gaspar fc46411

Carla Mata fc45109

1.Line and Branch Coverage

Para a parte do line and branch foram detectados erros em dois métodos: toList() e no size(). Em ambos os casos apanhamos o erro para quando a árvore está vazia. Os testes que apanharam os erros foram:

Method Tested	Test Name	Test Case Values	Expected Value	Value Returned
size()	testEmptyTree()	[]	0	1

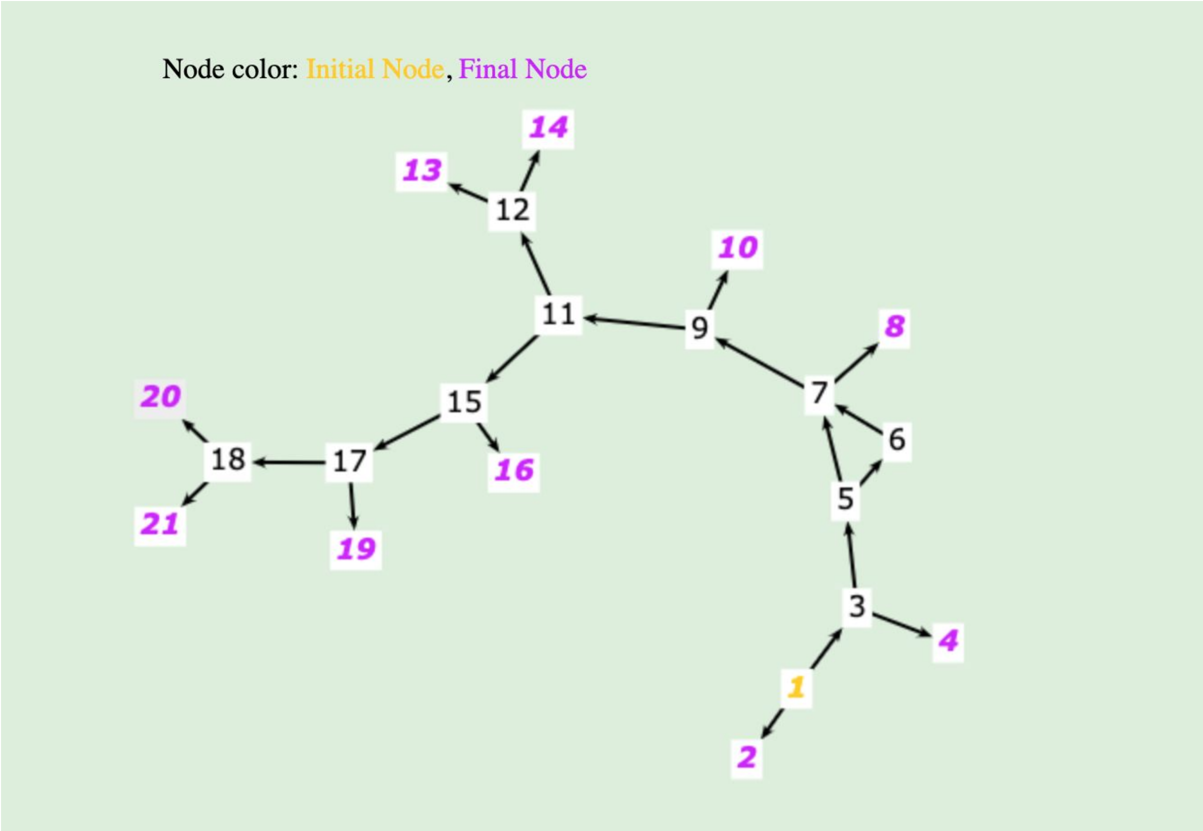
Cada um dos métodos foi alterado para ter em conta estes casos.

Foi achado um caso que é unreachable no branch coverage que é justificado no Logic-Based Coverage, nomeadamente na secção dos infeasible, sendo este o segundo deles.

2.Insert

<u>**nota**:</u> Os testes têm antes da sua declaração quais test requirements eles cobrem, excluindo repetições.

2.1 insert(T elem)



Legenda:
^= nó inicial
*= nó final
bold = cobertos

2.2 Edge-Pair Coverage

TR(NC) = { **[1]^, [2]*, [3], [4]*, [5], [6], [7], [8]*, [9], [10]*, [11], [12], [13]*, [14]*, [15], [16]***, **[17], [18], [19]*, [20]*, [21]*** }

TR(EC) = **TR(NC)** U { **[1,2]**, **[1,3]**, **[3,4]**, **[3,5]**, **[5,6]**, **[5,7]**, **[6,7]**, **[7,8]**, **[7,9]**, **[9,10]**, **[9,11]**, **[11,12]**, **[11,15]**, **[12,13]**, **[12,14]**, **[15,16]**, **[15,17]**, **[17,18]**, **[17,19]**, **[18,20]**, **[18,21]** }

TR(EPC) = **TR(EC)** U { **[1,3,4]**, **[1,3,5]**, **[3,5,6]**, **[3,5,7]**, **[5,6,7]**, **[5,7,8]**, **[5,7,9]**, **[6,7,8]**, **[6,7,9]**, **[7,9,10]**, **[7,9,11]**, **[9,11,15]**, **[9,11,12]**, **[11,12,13]**, **[11,12,14]**, **[11,15,16]**, **[11,15,17]**, **[15,17,18]**, **[15,17,19]**, **[17,18,20]**, **[17,18,21]** }

Test Name	Test Case Values (tree,insertedValue)	Expected Value	Value Returned	Requirements covered

testInsertEmptyTree()	([],1)	[1]	[1]	[1],[2],[1,2]
testInsertIsLeaf()	([1],2)	[1:[2]]	[1:[2]]	[5,7],[3,5,7], [5,7,8]
testInsertRepetitions()	([],1) ([1],1) → segunda inserção do elemento 1	[1]	[1]	[3],[4],[1,3],[3,4], [1,3,4]
testSwapElementsAtRoot()	([1],0)	[0:[1]]	[0:[1]]	[5],[6],[7],[8],[3,5],[5, 6],[6,7],[7,8], [1,3,5],[3,5,6], [5,6,7],[6,7,8]
testInsertLowerElementInNonEmptyTree()	([1:[3]], 2)	[1:[2][3]]	[1:[2][3]]	[9], [10], [11], [12], [13], [7,9], [9,10], [9,11], [11,12], [12,13], [7,9,10], [7,9,11] [9,11,12], [11,12,13]
testNewLevel()	([1:[10][20]],5)	[1:[5][10][20]]	[1:[5][10][20]]	[15],[16], [11,15], [15,16] [9,11,12], [9,11,15], [11,15,16], [11,15,17]
testIfNodeCapacityIsFullAndElemLagerThanAll( )	([10:[15:[16]][17][20:[30][40]]],3 0) → delete do 30 [10:[15:[16]][17][20:[40]]],30) → insert do 30	[10:[15:[16]][17][20:[30][40]]]	[10:[15:[16]][17][20:[30] [40]]]	[17], [18], [19] ,[20], [21] [15,17], [17,18] ,[18,20] [18,21], [15,16,17], [17,18,20], [17,18,21]
testCapacityNotFull()	([10:[17][20][21:[23][25][26:[35][40][50]]]],20) → delete do 20 ([10:[17][21:[23][25][26:[35][40][50]]]],22) → insert do 22	[10:[17][21:[22:[23]][25][26 :[35][40][50]]]]	[10:[17][21:[22:[23]][25] [26:[35][40][50]]]]	[11,12,14]
testAtRoot()	([10:[15:[16]][17][20:[30][40]]],9)	[9:[10:[15][16]][17][20:[30][40]]]	[9:[10:[15][16]][17][20:[30][40]]]	[6,7,9]

2.3 Prime Path Coverage (pelo menos 50%)

TR(PPC) = { [1,3,5,6,7,9,11,15,17,18,20], [1,3,5,6,7,9,11,15,17,18,21], **[1,3,5,7,9,11,15,17,18,20]**, **[1,3,5,7,9,11,15,17,18,21]**, [1,3,5,6,7,9,11,15,17,19], **[1,3,5,7,9,11,15,17,19]**, [1,3,5,6,7,9,11,15,16], [1,3,5,6,7,9,11,12,13], [1,3,5,6,7,9,11,12,14], **[1,3,5,7,9,11,15,16]**, **[1,3,5,7,9,11,12,14]**, **[1,3,5,7,9,11,12,13]**, **[1,3,5,6,7,9,10]**, **[1,3,5,6,7,8]**, **[1,3,5,7,9,10]**, **[1,3,5,7,8]**, **[1,3,4]**, **[1,2]** }

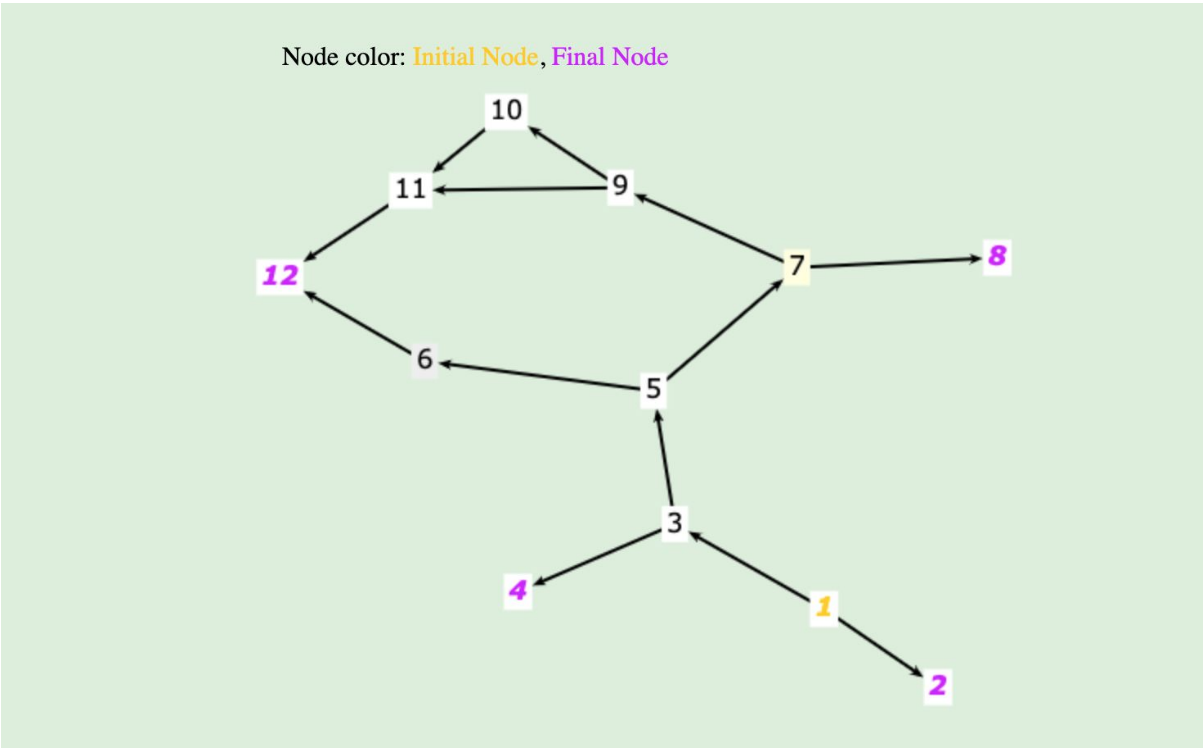
(12/18) * 100% = **66.66(6)% > 50%**

Test Name	Test Case Values (tree,insertedValue)	Expected Value	Value Returned	Requirements covered
testInsertEmptyTree()	([],1)	[1]	[1]	[1,2]
testInsertIsLeaf()	([1],2)	[1:[2]]	[1:[2]]	[1,3,5,7,8]
testInsertRepetitions()	([],1) ([1],1) → segunda inserção do elemento 1	[1]	[1]	[1,3,4]
testSwapElementsAtRoot()	([1],0)	[0:[1]]	[0:[1]]	[1,3,5,6,7,8]
testInsertLowerElementInNonEmptyTree()	([1:[3]], 2)	[1:[2][3]]	[1:[2][3]]	[1,3,5,7,9,10], [1,3,5,7,9,11,12,13]
testNewLevel()	([1:[10][20]],5)	[1:[5][10][20]]	[1:[5][10][20]]	[1,3,5,7,9,11,15,16]
testIfNodeCapacityIsFullAndElemLagerThanAll()	([10:[15:[16]][17][20:[30][40]]],30) → delete do 30 [10:[15:[16]][17][20:[40]]],30) → insert do 30	[10:[15:[16]][17][20:[30][40]]]	[10:[15:[16]][17][20:[30][40]]]	[1,3,5,7,9,11,15,17,18,21], [1,3,5,7,9,11,15,17,18,20]

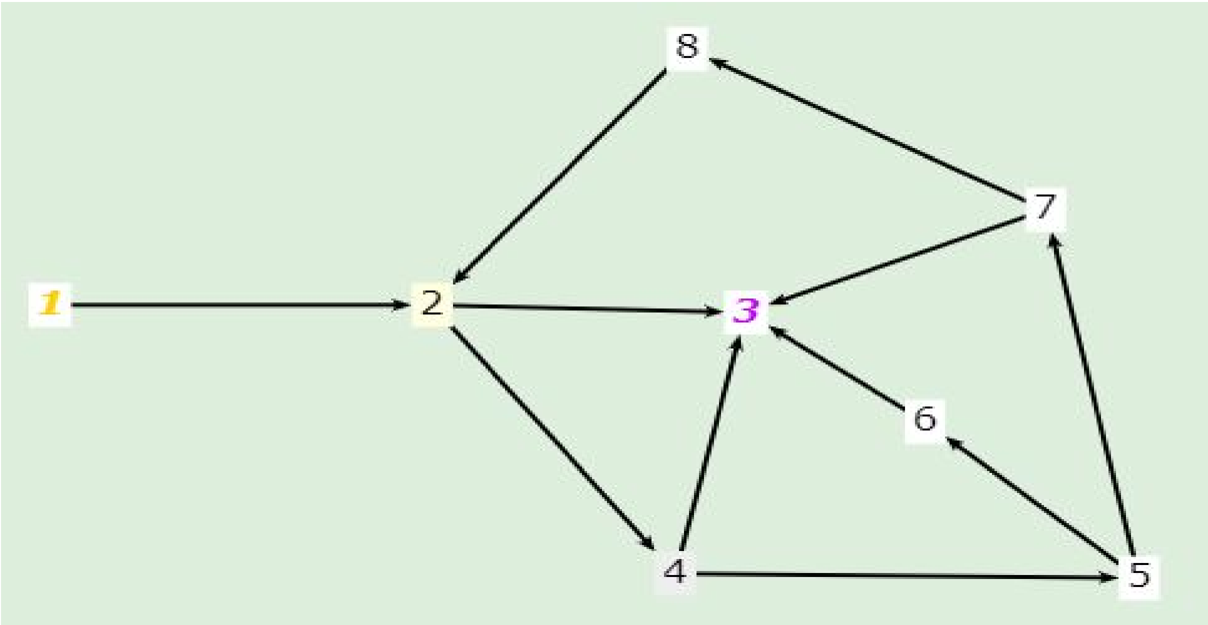
testCapacityNotFull()	([10:[17][20][21:[23][25][26:[35][40][50]]]],20) → delete do 20 ([10:[17][21:[23][25][26:[35][40][50]]]],22) → insert do 22	[10:[17][21:[22:[23]][25][26:[35][40][50]]]]	[10:[17][21:[22:[23]][25][26:[35][40][50]]]]	[1,3,5,7,9,11,12,14]
testAtRoot()	([10:[15:[16]][17][20:[30][40]]],9)	[9:[10:[15][16]][17][20:[30][40]]]	[9:[10:[15][16]][17][20:[30][40]]]	[1,3,5,6,7,9,10]

3.All-Coupling-Use Coverage

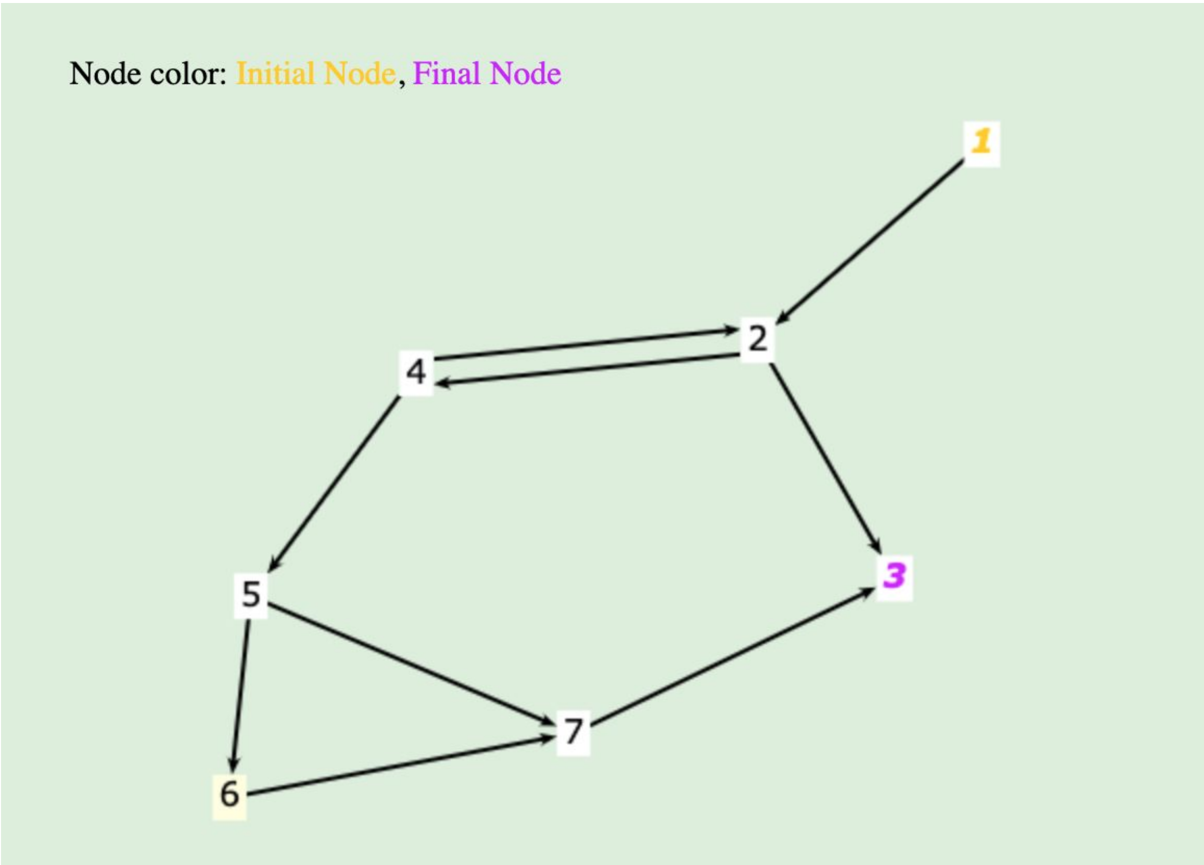
3.1 delete(T elem)



3.2 proposePosition(T elem)



3.3 compact(ArrayNTree<T>[] children)



last-def (nós)	first-use(nós)
elem(delete): {1}	elem(proposePosition): {4,5}
index(proposePosition): {1,5,6}	position(delete): {7}
children(delete): {6,11}	children(compact): {4}

Coupling du-pairs (last-def -> first-use):

1) **(delete, elem, node 1) -> (proposePosition, elem, node 4)**
2) **(proposePosition, index, node 1) -> (delete, position, node 7)**
3) **(proposePosition, index, node 6) -> (delete, position, node 7)**
4) **(proposePosition, index, node 8) -> (delete, position, node 7)**
5) **(delete, children, node 6) ->(compact, children node 4)**
6) **(delete, children, node 11) ->(compact, children node 4)**

bold = covered

Test Name	Test Case Values (tree,insertedValue)	Expected Value	Value Returned	Requirements covered
testLeafElement ()	([1:[2][3][4][5]], 5)	[1:[2][3][4]]	[1:[2][3][4]]	(delete, elem, node 1) → (proposePosition, elem, node 4) (proposePosition, index, node 8) → (delete, position, node 7) (delete, children, node 11) → (compact, children node 4)
testRootDeletion ()	([1:[2][3][4][5]],1)	[2:[3][4][5]]	[2:[3][4][5]]	(delete, children, node 6) → (compact, children node 4)
testNonExistantElementMiddle()	([1:[2][3][4:[5]]],2)	[1:[3][4:[5]]]	[1:[3][4:[5]]]	(proposePosition, index, node 1) → (delete, position, node 7)
testLowerElementThenAllTree	[1:[3][4:[5]]]	[1:[3][4:[5]]]	[1:[3][4:[5]]]	(proposePosition, index, node 6) -> (delete, position, node 7)

4.Logic-Based Coverage (GACC)

Escolhemos o GACC como critério lógico pois achamos os critérios CoC e RACC pouco praticos devido ao número elevado de requisitos e achamos o isolamento das cláusulas uma forma interessante de testar um método mais complexo como o insert.

Cada predicado (p) e cada cláusula (c) está comentado no código, sendo que existem 13 cláusulas e 10 predicados.

Composição dos predicados e cláusulas:

- p1(c1)
- p2(c2)
- p3(c3)
- p4(c4)
- p5(c5)
- p6(c6 && c7)
- p7(c8)
- p8(c9 && c10)
- p9(c11 || c12)
- p10(c13)

TR(GACC) = { **c1**, **¬c1**, **c2**, **¬c2**, **c3**, **¬c3**, **c4**, **¬c4**, **c5**, **¬c5**, **c6 && ¬c7**, **¬c6 && ¬c7**, ¬ c6 && c7, **c8**, **¬c8** , **c9 && ¬c10**, **¬c9 && ¬c10**, **¬c9 && c10, c11 && c12**, ¬c11 && c12, **c11 && ¬c12**, **c13** , **¬c13** }

bold = covered

Test Name	Test Case Values (tree,insertedValue)	Expected Value	Value Returned	Requirements covered
testInsert()	([],1)	[1]	[1]	c1
testInsertRepetitions()	([],1) ([1],1) → segunda inserção do elemento 1	[1]	[1]	~c1, c2
testSwapElementsAtRoot()	([1],0)	[0:[1]]	[0:[1]]	~c2 , c3, c4
testInsertLowerElementInNonEmptyTree()	([1:[2][3]], 0)	[0:[1:[2]][3]]	[0:[1:[2]][3]]	~c3 , ~c4, c5 ~c5, ~c6 && ~c7, ~c9 && ~c10, c11 && ~c12 ,~c13

testInsertIfSpaceAvailable()	([1:[2][4]],3)	[1:[2][3][4]]	[1:[2][3][4]]	c6 && ~c7
testPlaceBelowLastChild()	([1:[2][7][9]],3)	[1:[2:[3]][7][9]]	[1:[2:[3]][7][9]]	~c9 && c10
testIfNodeCapacityIsFullAndElemLagerThanAll()	(10:[15:[16]][17][20:[30][40]]] ,30) → delete do 30 ([10:[15:[16]][17][20:[40]]] ,,30) → insert do 30	[10:[15:[16]][17][20:[30][40]]]	[10:[15:[16]][17][20:[30][40]]]	c13
testInsertNotFullLevelWithElementlargerThanAllChildrenButNotBiggerThenAllPreviousChildren()	([10:[20:[25][27][28]][30][50:[55][57][58]]],30) → delete do 30 ([10:[20:[25][27][28]][50:[55][57][58]]] ,51) → insert 51	[10:[20:[25][27][28]][50:[51:[55]][57][58]]]	[10:[20:[25][27][28]][50:[51:[55]][57][58]]]	c8,~c8
testHasCapacityAndElementSmallerThenAllRecommendedChildren()	([2:[7][11][15:[17][18][19]][25][30]],30) → delete do 30 ([2:[7][11][15:[17][18][19]][25]],16) → insert do 16	[2:[7][11][15:[16][17][18][19]][25]]	[2:[7][11][15:[16][17][18][19]][25]]	c11 & c12, c9 && ~c10

4.1 Infeasible Cases

1. `¬ c6 && c7` ⇔ `!(nChildren<capacity) && children[position] == null` ⇔ `nChildren>=capacity && children[position] == null`

   Este caso é infeasible pois caso o nó tenha chegado ao limite dos nós filhos então o nó filho sugerido terá que ter um valor, chegando assim a uma contradição na condição.

2. `¬c11 && c12` ⇔ `!(nChildren==capacity) && elem.compareTo(children[position].max())<0` ⇔

⇔ `nChildren!=capacity && elem.compareTo(children[position].max())<0`

Devido à forma como a árvore é construída a primeira cláusula pode ser traduzida para (`nChildren>capacity || nChildren < capacity`). Isto torna a condição equivalente à condição do predicado que o precede. Como tal, não será possível chegar ao predicado nestas condições.

5.Base Choice Coverage

Partitions

Tree1 empty → [et1,¬et1]
Tree2 empty → [et2,¬et2]
Tree2 null → [nt2,¬nt2]
Tree intersection of Tree 2 → [empty,full,partial]

base choice :
[¬et1,¬et2,¬nt2,empty]

TR(BCC) = { **[¬et1,¬et2,¬nt2,empty], [et1,¬et2,¬nt2,empty], [¬et1, et2,¬nt2,empty], [et1,¬et2, nt2,empty], [¬et1,¬et2,¬nt2,partial], [¬et1,¬et2,¬nt2,full]** }

bold = covered

Test Name	Test Case Values (tree,insertedValue)	Expected Value	Value Returned	Requirements covered
testPartialEquals ()	([1:[2][3][4][5]], [2:[3][4][5][6]])	False	False	[~et1,~et2,~nt2,partial]
testEqualsWithSameElements ()	([1:[2][3][4][5]],[1:[2][3][4][5]])	True	True	[~et1,~et2,~nt2,full]
testEqualsWithDifferentIntersections ()	([1:[2][3][4][5]],[6:[7][8][9][10]])	False	False	[~et1,~et2,~nt2,empty]
testTreeNull ()	([1:[2][3][4][5]],null)	False	False	[~et1,~et2,nt2,empty]

testTreeEmpty ()	([1:[2][3][4][5]],[])	False	False	[et1,et2,nt2,empty]
testTree1Empty ()	([],[1:[2][3][4][5]])	False	False	[et1,et2,nt2,empty]

6.Mutations (PIT)

6.1.Line and Branch Coverage

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
2	96%	171/178	93%	158/169

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
startup	1	0%	0/5	0%	0/1
sut	1	99%	171/173	94%	158/168

6.2 Edge-Pair Coverage (insert)

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
2	63%	113/178	59%	99/169

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
startup	1	0%	0/5	0%	0/1
sut	1	65%	113/173	59%	99/168

6.3 Prime Path Coverage (insert)

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
2	63%	113/178	59%	99/169

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
startup	1	0%	0/5	0%	0/1
sut	1	65%	113/173	59%	99/168

6.4 All-Coupling Use Coverage (delete)

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
2	56%	99/178	49%	82/169

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
startup	1	0%	0/5	0%	0/1
sut	1	57%	99/173	49%	82/168

6.5 Logic Based Coverage - GACC (insert)

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
2	63%	113/178	59%	100/169

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
startup	1	0%	0/5	0%	0/1
sut	1	65%	113/173	60%	100/168

6.6 Base Choice Coverage (equals & equalstree)

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
2	44%	79/178	28%	47/169

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
startup	1	0%	0/5	0%	0/1
sut	1	46%	79/173	28%	47/168

Existem diferenças nos resultados entre os diversos tipos de critérios pedidos, isto deve-se a dois factores:

- Existem critérios que cobrem menos métodos que outros, dando uma maior possibilidade aos critérios que cobrem mais métodos de matar mais mutantes.
- Entre os critérios que cobrem os mesmos métodos, existem uns que contêm testes mais exaustivos que outros, apresentando consequentemente melhores resultados.

8. Erros Encontrados

Method Tested	Test Name	Class	Test Case Values	Expected Value	Value Returned
size()	testEmptyTree()	Line and Branch	[]	0	1
iterator<T elem> ()	shuffleTree()	Quickcheck	[]	[]	[[]]

8.1 Alterações Feitas

8.1.1 size()

```java
//colocar no relatório que apanhamos .
public int size() {
    int sum=0;
    if (empty) {
        return 0;
    }
    for(NTree<T> brt : children)
        if (brt!=null)
            sum += brt.size();
    return 1+sum;
}
```

8.1.2 PrefixIterator<T elem>()

```java
 */
public PrefixIterator(ArrayNTree<T> tree) {
    stack = new LinkedList<>();
    if (!tree.isEmpty()) {
        stack.push(tree);
    }

}
```