**Report**

# 1.

O construtor do iterador está mal, a raiz é pushed para a stack sem verificar se a tree é emtpy. Isto fazia com que caso a tree fosse vazia, e não haver elementos para iterar, como a stack não estava vazia, a função hasNext retornava true. Este erro foi encontrado no teste testArrayNTreeEquals2EmptyNTreesda classe ArrayNTreeLineBranchCoverageTest.

É impossível cobrir todos os branches do método equalTrees. O branch “one == other” já é coberto no método equals e que caso aconteça, o método equalTrees não é chamado. O branch e “one == null || other == null” também pois, o one aqui seria o this, que não poderia ser null se não ao tentar chamar o método equals seria lançada uma NullPointerException, e o other também nunca poderia ser null pois a condição “other instanceof NTree” não se verificaria e o método equalTrees não seria chamado.

É também impossível cobrir todos os branches do método proposePosition. No if “if (children[i].data.compareTo(elem)<0)” é impossível cobrir o branch em que esta condição não se verifica, isto porque todas as outras condições (ser igual ou maior do que 0) já foram verificadas em partes anteriores do código. Portanto para obter cobertura total, o código teria de ser escrito com if, else if e else, visto que no fim não há outra alternativa a “children[i].data.compareTo(elem)<0”.

# 2.

Como o método equals é apenas um return, o seu grafo de controlo de fluxo seriam apenas dois nós. Portanto, para testar este método, foi feito o grafo para o método privado equalTrees, que é onde estão as instruções mais relevantes para a comparação de igualdade entre duas NTrees.

## 2.1. Edge-Pair Coverage

TR(EPC) = { [0,1], [0,2,9], **[0,2,3]**, **[2,3,4]**, **[3,4,5]**, **[3,4,7]**, **[4,5,4]**, **[4,5,6]**, **[4,7,8]**, **[4,7,9]**, **[5,4,5]**, **[5,4,7]** }

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nome do teste | Valores  (this,other) | Resultado esperado | Caminho | Requerimentos cobertos |
| testArrayNTreeEqualsFirstChildDiff | ([1:[2]], [1:[3]]) | false | [0,2,3,4,5,4,5,6] | [0,2,3] [2,3,4] [3,4,5] [4,5,4] [5,4,5] [4,5,6] |
| testArrayNTreeEqualsEmptyTrees | ([],[]) | true | [0,2,3,4,7,8] | [0,2,3] [2,3,4] [3,4,7] [4,7,8] |
| testArrayNTreeEqualsSingleEqualRoots | ([1],[1]) | true | [0,2,3,4,5,4,7,8] | [0,2,3] [2,3,4]  [3,4,5] [4,5,4]  [5,4,7] [4,7,8] |
| testArrayNTreeEqualsOneEmptyTree | ([1], []) | false | [0,2,3,4,7,9] | [0,2,3] [2,3,4]  [3,4,7] [4,7,9] |

Passar por [0,1] é impossível pq isso e verificado antes de se chamar equalTrees

Passar por [0,2,9] também porque se one não pode ser null porque é o this e o other tmb n pq faz se o instanceof e null não é, logo não passa para o equalTrees

Estes dois casos já foram testados no Line/Branch Coverage, mas a cobertura so aparece no método equals

## 2.2 Prime Path Coverage

Comecemos por determinar os prime paths (marcados a negrito):

(paths marcados a vermelho são caminhos impossíveis)

[0] -> **[0,1]!** [0,2] -> [0,2,3] **[0,2,9]!** -> [0,2,3,4] -> [0,2,3,4,5] [0,2,3,4,7] -> **[0,2,3,4,5,6]! [0,2,3,4,7,8]! [0,2,3,4,7,9]!**

[1]!

[2] -> [2,9]![2,3] -> [2,3,4] -> [2,3,4,5] [2,3,4,7] -> [2,3,4,5,6]! [2,3,4,7,8]! [2,3,4,7,9]!

[3] -> [3,4] -> [3,4,5] [3,4,7] -> [3,4,5,6]! [3,4,7,8]! [3,4,7,9]!

[4] -> [4,5] [4,7] -> **[4,5,4]\*** [4,5,6]! [4,7,8]! [4,7,9]!

[5] -> [5,4] [5,6]! -> **[5,4,5]\*** [5,4,7] -> **[5,4,7,8]!** **[5,4,7,9]!**

[6]!

[7] -> [7,8]! [7,9]!

[8]!

[9]!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nome do teste | Valores  (this,other) | Resultado esperado | Caminho | Requerimentos cobertos |
| testArrayNTreeEqualsRootDifferent | ([1],[2]) | false | [0,2,3,4,5,6] | [0,2,3,4,5,6] |
| testArrayNTreeEqualsEmptyTrees | ([],[]) | true | [0,2,3,4,7,8] | [0,2,3,4,7,8] |
| testArrayNTreeEqualsOneEmptyTree | ([1], []) | false | [0,2,3,4,7,9] | [0,2,3,4,7,9] |
| testArrayNTreeEquals2EqualTreesWith2Elems | ([1:[2]], [1:[2]]) | true | [0,2,3,4,5,4,5,4,7,8] | [4,5,4] [5,4,5] [5,4,7,8] |
| testArrayNTreeEqualsEqualRootsOnlyRootTreeAndRootWithChildTree | ([1:[2]], [1]) | false | [0,2,3,4,5,4,7,9] | [4,5,4] [5,4,7,9] |

\*Para a representação dos valores foi usada a representação toString dos objetos ArrayNTree

# 3.

Diff All-coupliung-use para all-coupling-du-paths?

A função delete, invoca os métodos privados proposePosition e compact isLeaf e isEmpty são só returns.

Para a função proposePosition:

|  |  |
| --- | --- |
| Last-def | First-use |
| elem: {1} | elem: {d,e} |
| index: {f,h} | position: {8,9} |

Através da tabela, é possível extrair os seguintes pares (last-def -> first-use):

* (delete, elem, (1) line 204) -> (proposePosition, elem, (d,e) line 268)
* (proposePosition, (f) index, line 273) -> (delete, position, (8,9) line 225)
* (proposePosition, (h) index, line 278) -> (delete, position, (8,9) line 225)

As variaveis de paremetro e retorno, nos first-uses são arcos, logo são responsáveis os nos finais dos arcos. Mas isto vai fazer com que os pares fiquem iguais (1,e) e (1,d) o caminho é mm pq o arco representa um único if.

Para o compact, não há return coupling porque é um método void

# 4.

Seguindo as seguintes características:

1. Tree 1 is empty – [T1Empty, T1NotEmpty]

2. Tree 2 is empty – [T2Empty, T2NotEmpty]

3. Tree 2 is null - [T2Null, T2NotNull]

4. Tree 1 intersection of Tree 2 is empty/full/partial – [E,F,P]

O caso base seria aquele que seria o mais comum num ambiente de execução normal do sistema, portanto sugere-se: (T1NotEmpty, T2NotEmpty, T2NotNull, P). Portanto, a partir do caso base, os requerimentos para testar são:

(T1NotEmpty, T2NotEmpty, T2NotNull, P), (**T1Empty**, T2NotEmpty, T2NotNull, P),

(T1NotEmpty, **T2Empty**, T2NotNull, P), (T1NotEmpty, T2NotEmpty, **T2Null**, P),

(T1NotEmpty, T2NotEmpty, T2NotNull, **F**), (T1NotEmpty, T2NotEmpty, T2NotNull, **E**).