IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

PA 2021/22

Programação Avançada

Enunciado de Laboratório

4

Laboratório Avaliado: ADT Binary Search Tree e Unit Testing

Objetivos:

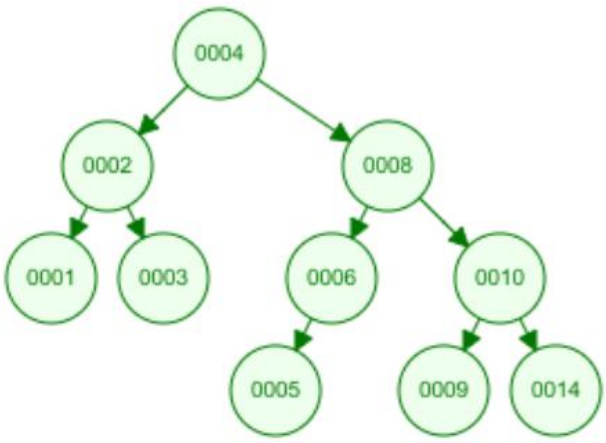
- Utilização/manipulação do ADT **Binary Search Tree** para inteiros;
- Criação de testes unitários em JUnit;
- Implementação de métodos adicionais.

Introdução:

Comece por clonar o projeto existente no GitHub no seguinte endereço:
https://github.com/estsetubal-pa-geral/Lab4Template.git

Nível 1 – Criação de testes unitários para a classe IntegerBST

1.1. Crie uma classe de testes unitários para a classe **IntegerBST**, adicionando um atributo deste tipo e configure o método **setUp()** de modo a inicializar a seguinte BST antes da invocação de cada teste unitário:



Crie as seguintes UTs e execute a bateria de testes:

1.2. **test_elementExists_afterInsert() –** Verifica se os elementos 7, 13 e 0 existem após a sua inserção;

1.3. **test_elementDoNotExist_afterRemove() –** Verifica se os elementos 4, 2 e 1 não existem após a sua remoção.

1.4. Pretende-se testar a correção do método **inOrder()**. Para tal, crie uma única UT denominada **isCorrect_inOrder_withSeveralTrees()** que verifica se os elementos devolvidos e a sua ordem estão corretos após:

- Árvore inicial;
- Após inserir 17;
- Após remover 4.

Na implementação da UT, utilize a seguinte função auxiliar:

```
private boolean equals(Iterable<Integer> actual, Integer... expected) {
        int i = 0;
        int count = 0;
        for(int item : actual) {
            if(item != expected[i++]) return false;
            count++;
        }

        return count == expected.length;
    }
```

Nível 2 – Implementação e teste do método *sum()*

2.1 Implemente o método **sum()** que devolve a soma dos elementos na BST.

2.2 Crie as UTs:

- **isThrown_Exception_onSumOnEmptyTree()** – Verifica se a exceção é lançada após esvaziar a árvore;
- **sum_isCorrect_afterInsertAndRemove()** – Verifica se a soma dos valores é calculada corretamente após:
  - Árvore inicial;
  - Após inserir 0;
  - Após inserir 7;
  - Após remover 5.

Nível 3 – Implementação dos métodos relativos a nós internos da árvore

3.1 Implemente o método **isInternal()** que verifica se um determinado *nó* de uma árvore é interno.

3.2 Implemente o método **sumInternals().**

3.3 Crie as UTs:

- **isThrown_Exception_onSumInternalsOnEmptyTree()** – Verifica se a exceção é lançada após esvaziar a árvore;
- **sumInternals_isCorrect_afterInsertAndRemove()** – Verifica se os valores são devolvidos corretos após:
  - Árvore inicial;
  - Após inserir 15;
  - Após remover 4.

Nível 4 – Implementação do método *breadthOrder()*

4.1 Implemente o método **breadthOrder**() (*algoritmo-base fornecido em baixo*).

4.2 – Crie a UT **breadthOrder_isCorrect_afterInsertAndRemove()** que verifica se os elementos devolvidos e a ordem pela qual aparecem estão corretas após:

a. Árvore inicial;
b. Após remover 4;
c. Após remover 5.

```
Breadth-Traversal (bst):
    Queue q = <empty_queue>
    q.enqueue(bst.root)
    while(!q.isEmpty()):
        node = q.dequeue()
        //visit node's element
        If node.left exists:
            q.enqueue(node.left)
        If node.right exists:
            q.enqueue(node.right)
```

Nível 5 – Implementação dos métodos greaterThan, coutGreaterThan e testes unitários

Implemente os seguintes métodos (*já definidos na classe*):

5.1 **greaterThan()** – Devolve o conjunto dos elementos na BST que são maiores que o parâmetro enviado ao método.

5.2 **countGreaterThan()** – Devolve o número de elementos na BST que são maiores que o valor enviado ao método.

Implemente a UT:

5.3 **countGreaterThan_isCorrect_afterInsertAndRemove()** – Verifica se, ao fazer a contagem de valores maiores que 5, o resultado é o esperado.

- Árvore Inicial;
- Após remover 2;
- Após remover 10;
- Após Adicionar -1;
- Após Adicionar 123.

(fim de enunciado)