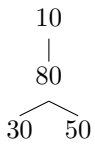
Ciências ULisboa | Informática

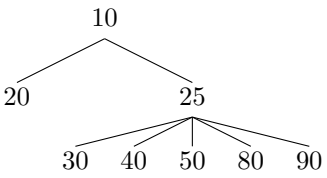
VVS (2018/2019)
Assignment 1

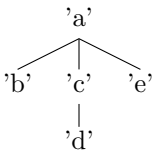
1 Introduction

This assignment goal is to prepare a set of unit tests to verify an implementation of a n-ary tree data structure (denoted *n-trees*).

Some examples of n-trees:







In this case, we are only interested in trees with the following invariant:

A prefix traversal of the tree results is a strict increasing sequence of elements.

So, in the above examples, only the second and third n-trees satisfy the invariant. Notice also that the invariant implies that there are no repeated elements inside a n-tree, and that all elements should be comparable.

The provided source code implements a restricted version where each node has a maximum capacity of children.

The insertion and deletion method must satisfy the invariant. Also, in the SUT, n-trees try to use as much as possible the current node's capacity before making a new level below. It was not mandatory to make a balanced tree and, so, there was some freedom in the implementation of these operations. An important part of the tests will be to verify if all code paths on these methods maintain the invariant.

Import maven project `vvs_1819_assignment_1` to access the data structure implementation.

2 Requirements

The entire test set must satisfy the requirements produced by the following criteria:

1. Line and Branch Coverage for all public methods;
2. Edge-Pair Coverage and at least 50% coverage for Prime Path Coverage for method `insert`;
3. All-Coupling-Use Coverage for method `delete` and its private methods;
4. Select and apply one Logic-based test coverage for method `insert`, justify your option.

For methods `equals` and `equalTrees`, include a test set based of Input State Partitioning, namely Base Choice Coverage, using the following characteristics:

1. Tree 1 is empty
2. Tree 2 is empty
3. Tree 2 is null
4. Tree 1 intersection of Tree 2 is empty/full/partial

Verify your test set via program mutation using PIT. Write a report comparing the mutation coverage achieved by each criteria.

Use JUnit QuickCheck to create an n-tree random generator. Test the following properties:

1. Given a set of elements, shuffling their order of insertion does not break the n-tree's invariant
2. If you remove all elements from a tree, the n-tree must be empty
3. Given a n-tree, inserting and then removing the same element will not modify other elements
4. Given a n-tree, inserting all its elements again will produce no effect
5. Given a n-tree, inserting an element already there several times over will produce no effect

For each coverage criteria paradigm, include its test cases in different classes or even different packages (if you have several classes per testing paradigm), all of them properly named. There's no problem if you repeat test cases between packages.

For each unit test include appropriate Java comments that detail what test requirements it satisfies.

3 Deliver

Upload a zip of the group's work into the course's moodle webpage until 23:59 of May 5. The zip file should be named `vvs01_XX.zip` where `XX` is your group's number.

The zip must include:

- the maven project with just the source code and `pom.xml` (no binaries or your IDE's metadata);
- a pdf report with your analysis of the several coverage criteria and with a list, if any, of the faults corrected during your analysis, explaining, for each fault, what was the subsequent failure and which test caught it.