Red-Black Tree Test Plan

Version 1.0

Document History:

Author: Anindita Sarkar
Create Date: 25-Oct-05
Reviewed By:
Review Date:
Last Updated By:
Last Update Date:

Objective:

This document presents a test plan for a Red-Black BST data type. It also refers to the corresponding test cases for the test plan items. Finally, it discusses a regression setup and identifies the various user applications in the regression.

Regression Setup:

The regression setup will comprise[1] the following:

1. Regression Applications – these are either test by themselves or they will be used to drive the tests through the scenarios.

   In the simpler scenarios, the data & instruction may be embedded in the application itself. So every application then will represent a test case. In Big and Random scenarios, however, generic applications need to be written that'll use 'generated' input data / instruction to drive the test.

   In every application the Scenario will be maintained within the banner the 'Purpose' of the test.

2. Regression Tests – these are various inputs sets comprising –
   a. Data Sequences (as Key and Info pairs) and / or
   b. Instruction Sequences – Insert or Find

3. Regression Goldens – these are expected outputs in every case as created from messages and / or Tree Prints. Possible messages include –
   a. Created Tree
   b. Find Pass: <Key, Info>
   c. Find Fail: <Key, Info>
   d. Insert Pass: <Key, Info>
   e. Insert Fail (Duplicate): <Key, Info>
   f. Valid Tree
   g. Released Tree

The Regression setup will have a folder structure following the scenario classification. Multiple tests within the same classification will be number serially from 1 as 'Test 1', 'Test 2', 'Test 3' etc. Every folder will contain the following:

1. Application
2. Input File (.txt file)
3. Output File (.txt file)
4. Performance File (.txt file), if any, will include the memory & time performance information.

All files in a folder will bear the same name.

[1] Normally, regression setup also comprises an automation script that can be invoked from a single command for all tests to run through the respective applications, generate the output and regress against the golden. We have now kept such activities out of the scope as you are not familiar with scripting.

Scenarios:

Various scenarios have been enumerated here[2].

1. **Functional Tests** – test for the primary functionality of the data type.
    a. Create / Destroy Tests – these are for testing various constructors & the destructor.
        i. Automatic Tree
        ii. Dynamic Tree
    b. Insertion Tests
        i. Normal (Common) Cases – without duplicates
            - Tree arising from Arbitrary Sequence of 5 Insertions
            - Tree arising from Arbitrary Sequence of 10 Insertions
            - Tree arising from Arbitrary Sequence of 15 Insertions
        ii. Duplicate Insertion (3 cases)
    c. Find Tests
        i. Existing Keys (5 cases)
        ii. Missing Keys (5 cases)
    d. Print Tests
    e. Traversal Tests / Sorting Order
    f. Visitor Tests[3]
2. **API Tests** – test all APIs for the data type (library). All variants for an API (depending on default parameters) need to be tested.
    a. Create – wraps the constructor
    b. Insert
    c. Find
    d. Validate
    e. Print
    f. Release – wraps the destructor
    g. Visitor
3. **Directed Tests, Corner Cases and Tests from Implementation**.
    a. Empty Tree
    b. Tree with a Single Node
    c. Creating all scenarios for Tree rotation, color shifts and root splits – enumerate & elaborate[4] from the Red-Black Tree algorithms
    d. Multiple Trees within the same scope
        i. Disjoint Lifetime
        ii. Overlapped Lifetime
    e. Negative Tests
        i. No-Copy Test – the Tree cannot be copied
        ii. No-Assignment Test – the Tree cannot be assigned.
    f. Big Tests[5].

[2] Anindita, complete the possible scenarios. Include cases for Deletion as well.
[3] We are yet to introduce the 'visitor' functionality in the Red-Black Tree. So skip these tests till we provide such support.
[4] Anindita, please complete.

i. Tree arising from long (~500) monotonic sequence of insertions
- Increasing Case
- Decreasing Case

ii. Tree arising from long alternating sequences of insertions

4. **Use-Model / Extension Tests** – if the Tree is specialized by the user. For example,
   a. Add a 'name' to a Tree.
   b. Change the type of Key in the Data nodes
   c. Change the type of Info in the Data nodes
   d. Allow for multiple Keys by accommodating a resolution on secondary Keys
5. **Random & Huge Tests**[6].
   a. Pseudo-Random Tests – the data for such tests should be generated beforehand using the random generator and stored. The size of every test should be between 10,000 and 1,000,000. Each category should have at least 5 tests.
      i. No-Duplicate Inserts & Exists-only Finds
      ii. Free Inserts & Exists-only Finds
      iii. No-Duplicate Inserts & Free Finds
      iv. Free Inserts & Free Finds
   b. Random Test – these tests will generate random test data every time the test is run. The random runs should be of the order of 100,000.
6. **Low Memory Tests**.

[5] Anindita, identify more big tests.
[6] Write a generator for the random data in every case. This will generate the Key-Info pairs and the Insert-Find sequences. The generation process needs to confirm the constraints on the randomness (like Free-Inserts and Exists-only Find).

Quality and Performance:

The following quality parameters need to be achieved by every positive Golden.

1. Zero Memory Leak
2. Zero Memory Access Error

The following performance parameters need to be measured for Big and Random Tests:

1. Peak Memory
2. Memory / Key-Info pair
3. Time / Insert
4. Time / Find