Hash Table Test Plan

Version 1.0

Document History:

Author: Madhubanti Dasgupta
Create Date: 25-Oct-05
Reviewed By:
Review Date:
Last Updated By:
Last Update Date:

Objective:

This document presents a test plan for a Hash Table 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 or Delete

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

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 Hash Table
        ii. Dynamic Hash Table
    b. Insertion Tests
        i. Normal (Common) Cases – without duplicates
            - Hash Table arising from Arbitrary Sequence of 5 Insertions
            - Hash Table arising from Arbitrary Sequence of 10 Insertions
            - Hash Table arising from Arbitrary Sequence of 15 Insertions
        ii. Normal (Common) Cases – with duplicates (at least 5 in each case)
            - Hash Table arising from Arbitrary Sequence of 15 Insertions
            - Hash Table arising from Arbitrary Sequence of 20 Insertions
            - Hash Table arising from Arbitrary Sequence of 25 Insertions
    c. Find Tests
        i. Existing Keys (5 cases) – unique
        ii. Existing Keys (5 cases) – duplicate / multiplicate
        iii. Missing Keys (5 cases)
    d. Delete Tests[3]
    e. Iterator Tests[4]
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. Delete
    e. Dump – should print all keys in the Hash Table
    f. Release – wraps the destructor
    g. Resize[5]
    h. Iterator

[2] Madhubanti, complete the possible scenarios. Specifically, include cases for Deletion.
[3] Please expand.
[4] We are yet to introduce the 'iterator' functionality in the Hash Table. So skip these tests till we provide such support.
[5] This should resize the table and rehash the existing keys. Add the scenarios when the support for the same is given.

3. **Directed Tests, Collision Cases and Tests from Implementation**.
    a. Empty Hash Table
    b. Hash Table with a Single Node
    c. Creating scenarios for Hash Collisions and all cases for Coalesced Chaining[6].
    d. Multiple Hash Tables within the same scope
        i. Disjoint Lifetime
        ii. Overlapped Lifetime
    e. Negative Tests
        i. No-Copy Test – the Hash Table cannot be copied
        ii. No-Assignment Test – the Hash Table cannot be assigned.
    f. Big Tests[7].
        i. Hash Tables arising from long (~500) sequences of keys that hash to the same or few distinct index values
        ii. Hash Tables arising from long sequences of keys that distribute (moderately) uniformly over the hash value
        iii. Collision behavior for variety of Hash Table sizes
            - Composites
            - Small Primes
            - Large Primes
            - Mersenne Primes
        iv. Collision behavior for variety of Hashing Functions
            - Consider string keys
        v. Collision behavior of Re-Hashing
        vi. Computation behavior for variety of Hashing Functions
            - Cost of remainder operator in hashing
        vii. Hash Table behavior and Hashing Function for non-integer keys
            - String keys
4. **Use-Model / Extension Tests** – if the Hash Table is specialized by the user. For example,
    a. Add a ‘name’ to a Hash Table.
    b. Change the type of Key in the Data nodes
    c. Change the type of Info in the Data nodes
    d. Change the Hash Function
5. **Random & Huge Tests**[8].
    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. Use integer keys.
        i. No-Duplicate Inserts & Exists-only Finds
        ii. Free Inserts & Exists-only Finds

[6] Madhubanti, please complete.
[7] Madhubanti, identify more big tests.
[8] 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).

iii. No-Duplicate Inserts & Free Finds
iv. Free Inserts & Free Finds

b. Pseudo-Random Tests – repeat for String keys
c. 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. Use integer keys.
d. Random Test – repeat for String keys

6. **Low Memory Tests**.

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. Rate of Collision – Collisions / Insert
2. Chain Length – Average length (median-based) of coalesced chains
3. Hash Table Utilization – distribution of indices having:
    a. 1% keys
    b. 5% keys
    c. 10% keys
    d. 25% keys
    e. 50% keys
4. Peak Memory
5. Memory / Key-Info pair
6. Time / Insert
7. Time / Find
8. Time / Delete