Comparison of Hash-tables and Red-Blacks Trees

Corneliu Rotari[1]
corneliu.rotari@stud.acs.upb.ro

University Politehnica of Bucharest, Romania

Abstract. This paper studies the comparison in performance (time and complexity) of a self - blanacing trees (red - black trees) and a hash tables. Each ADT (Abstarct Data Type) will be studied based on *Insertion, Deletion (minim and maxim), Modification and Search of the specific element*.

Keywords: Red-Black Tree · Hash-Table · Complexity · Open-address tables · Linked Hash-Tables · Insertion · Deletion · Data-Structures

1 Introduction

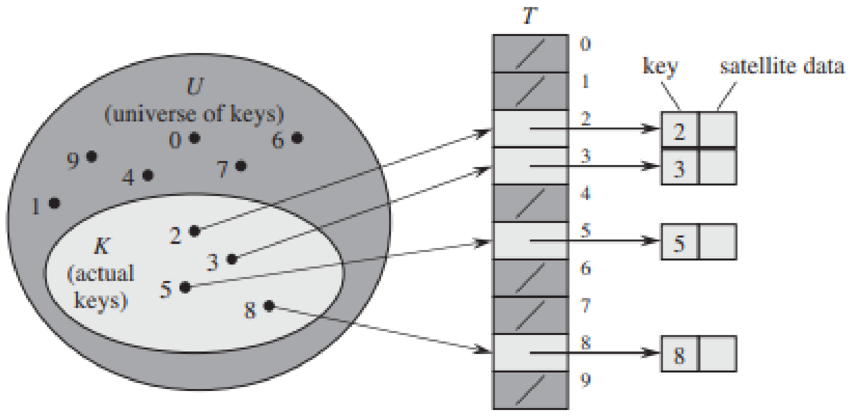
1.1 The Problem Statement

Sets are as fundamental to computer science as they are to mathematics. Whereas mathematical sets are unchanging, the sets manipulated by algorithms can grow, shrink, or otherwise change over time. [1] Therefore, we are looking to find out which Data Structure is more usefull and in which situation:

- Insertion.
- Deletion (minim and maxim).
- Diplaing a specific element (minim and maxim).
- Modification of a given element.
- Displaing all the elements.

Hash-Tables is an array of fixed size containing data items with unique keys, together with a function called a hash function that maps keys to indices in the table. The Hash Function is the part that can create conflict by collision in the keys of the hash table. There are effective techniques for resolving the conflict:

1. Hash-Table types:
   - Open Address - All elements occupy the hash table itself, no elements are stored outside.
   - Chaining - placing all the elements that hash to the same slot into the same linked list.
2. Different types of Hash Functions.



[1]

Fig. 1. Hash-Table Representation

Red-Black Trees are a self-balancing binary search tree in which each node contains an extra bit of information for denoting the color of the node, either red or black. This is needed to ensure that the tree is balanced during any operations performed on the tree like insertion, deletion, etc. In a binary search tree, the searching, insertion and deletion take $O(log_2n)$ time in the average case, $O(1)$ in the best case and $O(n)$ in the worst case.

A red-black tree is a binary tree that satisfies the following red-black properties:[1]

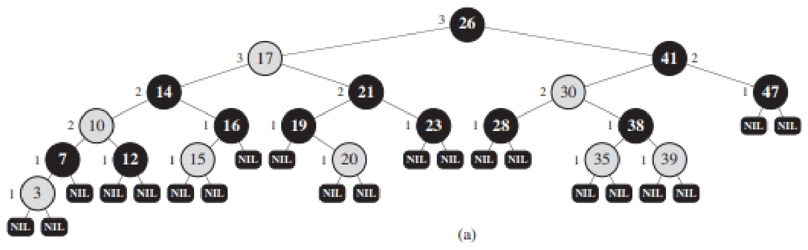
1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

I have noticed that the resizable hash table is lower on performance, because its all computation goes towards maintaing the load factor.I tried to change the load factor and the initial capacity, but the results where not much better.

1.2 Solutions Overview

Practical Applications There are a lot of situations when both Data Structures are suited for a job, but there are applications that bring the advantages to the top:

- Hash-Tables
  - Compilers - For storing and identifying the keywords in the programming languages.



[1]

Fig. 2. A red-black tree with black nodes darkened and red nodes shaded.

  - Caching - The operation optimized for this cache is fast lookup.
- Red-Black Trees [4]
  - Linux Kernel (The Completely Fair Scheduler) - Represents tasks in a tree and finds out which task to run next.
  - Database Engines - Data indexing in database engines uses RB trees directly or indirectly.

Chosen Solutions Every Hash table variation will be compared between every other algorithm and a red-black tree. In consequence there will be $(nrOfHashTable)^{nrOfHashFunctions}$ studied comparisons:

- Hash-Tables.
  - Open Address.
  - Chaining.
  - Fixed size of buckets.
  - Resizable Table.
- Hash functions.
  - The division method.
  - The multiplication method.
  - Universal hashing.

1.3 Evaluation Criteria

In order to test each Data Structure, particularized tests to test the average case, and especially the worst case scenarios.

Each type of test has 5-10 tests, generated in `python` and `java` for a large number of operations. The size of the inputs will vary from $[10.000; 1.000.000]$ operations per test. Types of tests:

1. **Ascending elements** - Sorted elements.
2. **Descending elements** - Descending sorted elements.
3. **Random elements** - Random distributed numbers using random function.

4. **Elements hash to the same location -** using different methods of hashing, the test set the input to the same bucket or same buckets on different resizes.
5. **BTS specific order elements -** The multiple insertion and deletion of the root element.

Each method will be tested for memory usage, efficiency of the building process and the efficiency of the query solving process.

Test structure:

- **N** = number of queries on the next lines.
- Types of queries:
  - **0 a** - Insert 'a' in the set.
  - **1 a** - Delete 'a' from the set.
  - **2 a** - Check if 'a' exists in the set.
  - **3 a b** - Element 'a' (if it exists) is replaced by 'b'.
  - **4** - Display all the elements in the set.

2 Solution

2.1 Red-Black Trees

Functionality : I made a generic implementation of an Red-black tree in Java, because it is vastly used in the `java.util.*` framework. The coloring is used to minimise the number of rotations and to ensure that on each path from the root till each leaf there is a equal amount of black nodes.

Complexity Analysis : each operation on the data structure is of $O(log(n))$, which makes it easy to work with. The only down side of the self-balancing tree are the rotations and the re coloring, but those are on average $O(1)$, because most of the time recoloring affects the current node and it's parent.

Advantages and Disadvantages : The biggest advantage is it has the same complexity for each operation, which in average doesn't need to pay for the moving the whole tree to adapt. On the other hand it has costly performance to an perfect hash-table.

2.2 Hash Tables

Functionality There are a variety of hash-tables, but I chose to implement the most popular ones **Open Addressing**, **Resizable**, **Collision with linked list** and others.

Complexity Analysis : The Complexity of the Hash Table is in general $O(1)$, but including resizing it is a lot more costly.

Advantages and Disadvantages Some features are statically constructed for a better performance review. In consequences some dynamical features are sacrificed (e.g the Open Address Hash table has the maximum buckets generated). The biggest advantages are the approximately constant time on addition, deletion, indexing.

3 Evaluation

3.1 Test construction

As mentioned in the first section, the test were constructed to the on random input, on each data structure's weakness, and in an ascending and descending order. Half of the tests are constructed in `python` to simplify the construction of ascending, descending and random test. The other half are constructed in `java` for a easier access to data structures implementation. I constructed the tests in a way to test the performance form the average case till the worst case.

3.2 Computer Specifications

The computer on which I made the analysis is an Acer Aspire 7 with this specifications :

- Processor - AMD Ryzen 7 5700U with Radeon Graphics 1.80 GHz
- RAM - 16.0 GB, 8 cores
- 64-bit operating system, x64-based processor
- OpenJDK 19 for compilation and running the java program.

3.3 Interpretation

I run each program for 2 cycles, before the performance registry, for an better elimination of external factors. After multiple rounds of comparison between the data structures there are 2 evident contestants. 3

I have tried to reduce the load factor and to increase the intial capacity of the resizable Hash-table, but the performance was not significantly changing. 4

In consequnece the rest of the comparasions will be made on the :

- Red-black tree
- Open Address Hash-Map

In the figures 5 5 there are the results of the comparison of these 2 Data Structures.

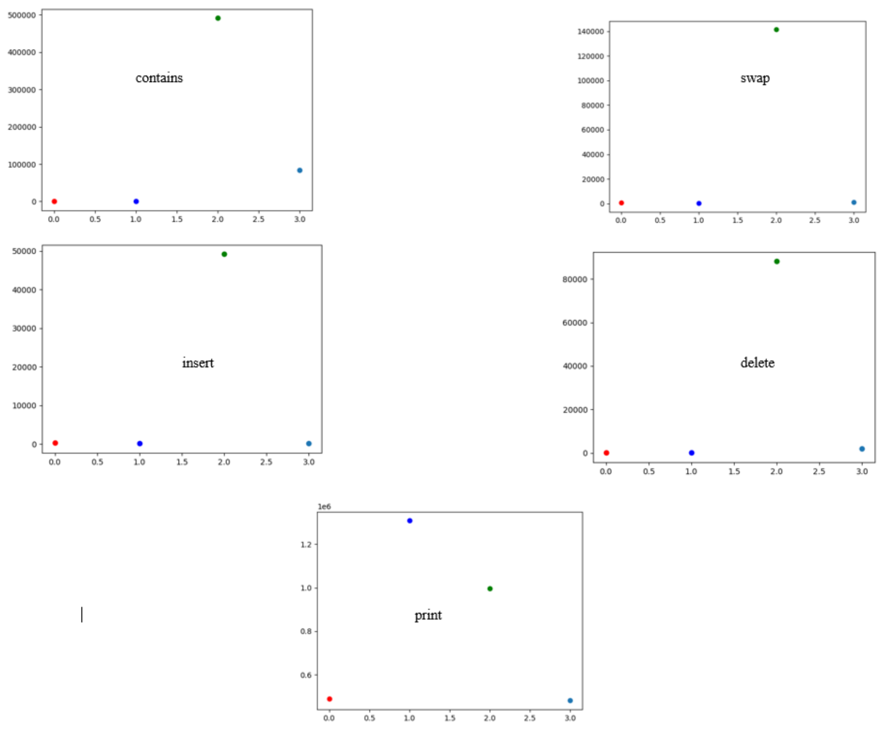


Fig. 3. Red - RBTree, Blue - Open Addr, Green - Resizble, Gray - Collision

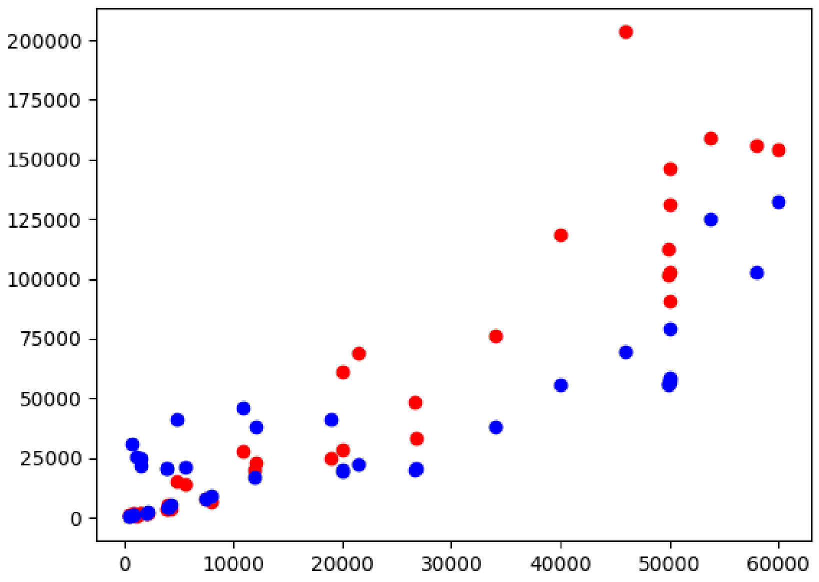


Fig. 4. Red - Initial 16, Blue - Initial 50000

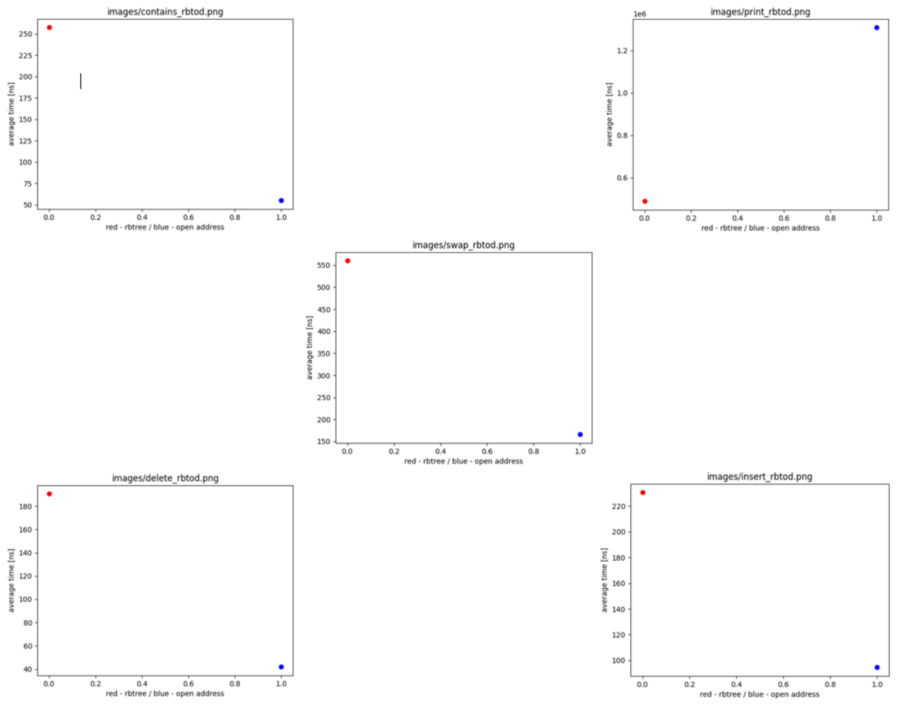


Fig. 5. Comparasion RedBlackTree and OpenAddressing

4 Conclusion

In conclusion, each Data-Structure has it's own use cases and should be used in according to the type of mass operations that they use. In practical uses cases if there are an extensive queries for search and a insertion, with the minimal cost o space complexity it is better to use a self - balancing tree, with the down side of computational power and time complexity. On the other hand if our priority is time complexity and to minimize computational power with the cost of space on a lot of operations of insertion and deletion it is better to use an Hash table. Personally I found that self-balancing trees are more reliable, and can outperform Hash-Tables in most situations.

References

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd ed.). The MIT Press.
2. Red–black tree
   `https://cs.kangwon.ac.kr/~leeck/file_processing/red_black_tree.pdf`. Last accessed 25 Nov 2022.
3. Pat Morin: Hash Tables. Carleton University (2001)
   `https://cglab.ca/~morin/teaching/5408/notes/hashing.pdf` Last accessed 25 Nov 2022.
4. Red-Black Trees Aplications.
   `https://www.baeldung.com/cs/red-black-trees-applications`