International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 1, ISSN No. 2455-2143, Pages 134-138
Published Online May 2020 in IJEAST (http://www.ijeast.com)

A REVIEW ON JAVA HASHMAP AND TREEMAP

Dinesh Bajracharya
Kantipur College of Management and Information Technology,
Kathmandu, Nepal

Abstract - Developing robust, efficient software applications is complicated a task. In built collections available in programming languages are of great value for developing efficient and robust software applications. Java inbuilt collections HashMap and TreeMap are very efficient tools for improving search time in the programs and having understanding about the underlying data structures of these collections is of great value. HashMap is based on the concept of hashing and TreeMap based on Red-Black tree. This article provides basic understanding about HashMap and TreeMap and highlights the situations where HashMap should be used and where TreeMap should be.

Keywords: HashMap, TreeMap, HashTable, Red-Black Tree, ResultSet

I. INTRODUCTION

Proper organization of data in software applications is very necessary for easy access and modification of data. Unorganized data will be just result in wastage of time during search and modify processes. Several Data structures are defined for organizing data such that search and modify process can be performed in acceptable time [4]. Data structure serves as Abstract Data types (ADT). Abstract Data Type specifies logical organization of data while data structure is physical organization of data [5]. Data structures like: Stack, Queue, Linked list, HashTable, Tree, Graph can be used to organize data for fast access and easy manipulation. Operations like insertion, deletion, manipulation, searching etc. can be performed easily on those data structures. These data structures help to improve performance software and help programmers to save their software development time.

Programming languages like C/C++, Java, Python, C# have several inbuilt classes or collection based on theory of data structures like stack, queue, linked list, hashtable, tree. These inbuilt collection objects can be used to develop robust software easily and free programmers from burden of devoting time developing data structures on their own. Benefits of those collections can be understood by understanding the underlying data structure on which implementation of those collections are based.

Different collection objects can be used for same purpose, that is to store data and perform search, edit operations. But the performance of considered collection may differs from each other and having knowledge of their performances will be of great value [1]. This article focuses to understand how Java inbuilt collections: HashMap and TreeMap perform with real data and which one is better choice in what type of situation.

II. THEORIES

Hashing

Hashing is a technique of distributing elements in an array such that elements can be searched in O(1) time. Hashing uses hash function to map elements in the array, this function takes elements as input and generates integer value which corresponds to the position for the element in the array. Same hash function is used to find the position of the element. A good hash function will distribute elements evenly in the array [2].

Creating a hash function which can distribute elements evenly in an array is challenging task as it is very possible that two or more elements hash to the same position in the array, that is the position generated by the hash function for two or more elements is same. This situation is known as hash collision.

Several techniques like division method, multiplication method are developed for generating hash value. These methods generate hash value for elements in the hash table (hash table is like an ordinary array). A hash table has j number of slots (buckets) for 'm' number of elements, j takes values 0,1,2,…..,n, where n is the size of table. Hash value generated for an element will be among the values of j. A value generated for an element will be its location in the hash table. Collision occurs when two elements

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 1, ISSN No. 2455-2143, Pages 134-138
Published Online May 2020 in IJEAST (http://www.ijeast.com)

hash to same value. To reduce collision, secondary hash table S_j can be created. This secondary hash table S_j corresponds to the j^{th} slot of the primary hash table. If an element hashes to slots j of primary table, it will be stored in the S_j, the secondary hash table. All the elements with hash value j will be stored in S_j. Here two different hash functions will be used, first (outer hash function) one to get slot value in primary table then another (inner hash function) to get hash value in secondary hash table [11].

Generally, hash table stores key, value pairs; key represent identifier for the value, and value can be simple data or complex object. Hash Table contains primary and secondary tables to reduce collisions. The performance of hash table depends on two factors: load factor and capacity. Capacity is the size of primary hash table (that is number of slots in primary table) and load factor, α, is average number of elements stored in each slot.

Hash Table performs in constant time, O(1) for insertion, search, delete operations. Hash table with a good hash function performs very fast if the number of slots is more than keys.

Red-Black Tree:

A red black tree is a binary search tree in which node is colored either red or black. A node includes five fields for storing information color value of a node, reference to right and left children, value of key and pointer to parent node. Red-black tree is a balanced tree as any path of red-black tree is not more than twice as long as any other path [7].

A binary search tree is a red-black tree if it satisfies the following red-black properties:

- a node is either red or black in color
- root is black
- Every leaf (NIL) is black
- If a node is red, then both of its children are black
- For each node, all paths from the node to descendent leaves contain the same number of black nodes.

Height of a Red-Black tree is not more than 2lg(n+1) where n is this number of internal nodes in the tree. Any element in a tree can be searched in O(lgn) time and Insertion and deletion of keys can be done in O(lgn) time [10]. Elements in Red-Black tree are stored in the natural ordering of the elements and there is no wastage of storage space in Red-Black tree.

Java HashMap

HashMap is in-built collection in Java based on Hash table implementation. The implementation results in constant-time performance for get (retrieving values) and put (storing values) operations. Time required to iterate through the HashMap is proportional to the capacity of the HashMap instance plus its size. Performance of a HashMap instance is affected by two parameters: initial capacity and load factor. The capacity specifies the number of buckets in the hash table and the load factor is a measure which provides number of elements to store in the hash table (how full is hash table) before its capacity is automatically increased [10]. HashMap allows one null key and any number of null values. HashMap does not allow duplicate keys and each key can be associated or mapped to only one value [6].

Default value of load factor is 0.75 (lies between 0 and 1) and default capacity is 16. Higher value of load factor means less wastage of memory space but slow performance; lower value of load factor results in efficient performance with more wastage of memory. HashMap entry contains pairs of key and value, int hash of the key and a pointer to the next entry. Each entry occupies 32 bytes (12 bytes header+ 16 bytes data + 4 bytes padding) memory space. HashMap of size 'S' will occupy 32 * S bytes for entries storage. In addition, it will use 4 * C bytes for array entries, 'C' is the map capacity. A HashMap instance will occupy 32 * SIZE + 4 * Capacity bytes, but theoretically, its size limit could be equal to 8 * Size bytes (2 arrays of keys and values with no space wasted) [13].

Java TreeMap

TreeMap is in-built data structure based on a working principle of Red-Black tree. TreeMap stores key, value pairs. Data in the TreeMap is ordered according to the natural ordering of its keys. Several methods are defined in TreeMap and methods containsKey (for searching key), get (getting values for supplied key) and put (storing keys, values) are guaranteed to execute in log(n) time. TreeMap cannot have null key, but allows multiple number of null values [9].

Each node of Red-Black tree contains key, value, pointers to left and right children, pointer to parent and a Boolean color flag. A node occupies 12 bytes for header, 20 bytes for five object fields and 1 byte for the flag; in total 40 bytes including 7 bytes for alignment. The total memory consumed by a TreeMap is 40 + SIZE bytes, and this is approximately same as the memory requirement for a HashMap. In case of CPU time consumption, TreeMap is worse than HashMap provided the load factor of HashMap is

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 1, ISSN No. 2455-2143, Pages 134-138
Published Online May 2020 in IJEAST (http://www.ijeast.com)

enough small. But TreeMap is better in finding next, previous entries [13].

Lots of methods for TreeMap and HashMap are similar and methods considered for study are:

- **public v replace(K key, V value)**: This method is used to replace value of specified key. This method takes two parameters: first is key element for which value is to be replaced, second is the new value for the key. It returns previous value of the provided key or returns null if provided key is not mapped to the map.
- **remove(Key)**: Remove method removes value mapped for the particular key from the map. This method takes one parameter, that is, key for which mapped value is to be removed. It returns previous value for the provided key if key exists otherwise returns null.
- **get()**: To retrieve the value mapped by a specific key in the map, get() method is used. It takes key as parameter for which value is to be retrieved. It returns null if the map has no value mapped for the provided key.
- **containsKey()**: This method is used to see if provided key is mapped in the map or not. It takes the key element and returns true if the element is mapped in the map. It returns true if key is present otherwise false.
- **Put()**: To insert key, value pair in the map, put method is used. If key exists in the map, old value will be replaced by new value for the provided key. It returns null on passing new key, value and returns previous value on passing existing key.

System.nanoTime

This method returns the current value of the most precise available system timer in nanoseconds [12]. This method provides nanosecond precision, but not necessarily nanosecond resolution. This method can be used to measure time taken by a program, method, a block of statement, or a statement. Following pseudocode shows how to measure time taken by code to execute:

```
long t1 = System.nanoTime();
///code
long t2 = System.nanoTime();
long duration = t2 – t1;    ////(time take to execute by code)
```

III. RELATED STUDY

Lucas PenzeyMoong has found that HashMap does insertion and search operation in constant time but performance of HashMap can be affected by the size of structure which grows over time. Two factors that affect the performance of HashMap are: load and capacity. Capacity means the number of buckets (slots) created by the hashing function and load means fullness of each bucket. More buckets will be created as the number of elements in the structure grows. Java HashMap preforms well if the load is less than 75%, this results in over-allocation of memory than required by the HashMap. A TreeMap guarantees logarithmic cost (O(lg(n)) for insertion, search operations and its performance are directly related with the height of the tree. TreeMap should be used if number of elements are not known in advance as there is no wastage of memory, but if speed is main concern HashMap is better choice [8].

HashMap uses hash function to organize and store elements on the array-based data structure. Most of operations like put, get, contains, remove are performed in constant time of O(1). HashMap does not maintain order of elements and is not possible to search max, min, next, pervious elements in HashMap. HashMap suffers from memory wastages as it uses more memory than required to hold data. TreeMap does not wastage memory space, it takes O(log(n)) time to perform put, get, contains and remove operations and maintains natural order of the elements. Both TreeMap and HashMap do not support duplicate keys. If sufficient memory space is available and faster operations is requirement then HashMap is better option than TreeMap; but if memory space is limited, elements are added and removed quite frequently and natural ordering of keys is important then TreeMap should be considered [3].

IV. EXPERIMENTAL AND RESULTS

Programs were developed with HashMap and TreeMap collections, MySQL database was used as data source to populate considered collections, Java's in-built System.nanoTime() method was used to record execution time of considered methods of the collections. Several tests were performed: first with four different sets of data to perform load, get, containskey, put, remove operation for both HashMap and TreeMap objects and second with five sets containing randomly generated records for performaning get, containsKey, put, remove, and replace operations for both HashMap and TreeMap object.

Data for the experiment was first extracted from MySQL database, then stored in ResultSet object and

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 1, ISSN No. 2455-2143, Pages 134-138
Published Online May 2020 in IJEAST (http://www.ijeast.com)

from ResultSet oject loaded to HashMap and TreeMap objects. Separate programs were written for HashMap and TreeMap.

Java in-built method nanoTime() of System class was used to record execution time of each considered operations. For each operation start time (before start of execution) and end time (immediately after the execution of operation completed) were recorded and difference between two is the duration of time required for the execution of the method.

Table 1. Execution time (in nanoSeconds) for get, containsKey, put, remove operatons with first set of records

HashMap	**To Records**	**Number of records searched**	**get()**	**ContainsKey()**	**put()**	**remove()**
Set I	210000	50	787.8049	2004.878	1820	6424
Set II	210000	50	966.66	1500	2300	10500
Set III	210000	50	991.07	1760.71	1080	7600
Set IV	210000	50	950	1187.5	1320	8767
TreeMap						
Set I	210000	50	2066	1997.56	4320	13324
Set II	210000	50	2133	1633	5320	21625
Set III	210000	50	2760	2750	3020	16933.33
Set IV	210000	50	3416.36	3187.27	3200	19933.33

Table 2. Execution time (in nanoSeconds) for put, get, containsKey, remove, replace operations with randomly generated records

HashMap	**Total records**	**Number of Records Searched**	**Time taken to load data**	**get()**	**containsKey()**	**put()**	**remove()**	**replace()**
Set I	2000	100	18119500	689	753	1200	3000	2410
Set II	5000	200	28556400	672.5	848.5	1720	2820	1880
Set III	10000	500	34011700	632.2	1021	3580	2660	1740
Set IV	20000	1000	128072800	672.3	689	720	3760	2220
Set V	139318	2000	806737700	664.15	788.7	1040	4520	1700
TreeMap								
Set I	2000	100	3.76714+E	1625	1280	1420	5660	2910
Set II	5000	200	3.77+E	1631.5	1558	2960	5870	2680
Set III	10000	500	3.76712+E	1437.4	1136	1780	6530	2500
Set IV	20000	1000	3.77115+E	2150.1	1832	2720	10280	3410
Set V	139318	2000	3.76716+E	1807.05	1296	3040	7040	2610

V. DISCUSSION

The experimental results depicted in Tables 1 and 2 showed that HashMap collection of Java which is based on the hashing technique is superior than TreeMap collection based on the Red-Black tree for all five considered methods. For HashMap collections, get method was found faster than containsKey method, but opposite was true for TreeMap for same sets of data and different sets of data. Get method of HashMap was very fast than get method of TreeMap but for other operations differences between two were not seen to be very big for same set of data but seen better performance in HashMap for different set of data. Loading HashMap with any size data was very fast in HashMap than TreeMap. Get method used to search key was seen to be consistent even with increase in the number of search keys for HashMap. Load time for HashMap increases with increase in number of records, but that was not the case TreeMap object.

The result of this study is found to be consistent with results shown in the related studies. HashMap implementation based on hashing technique if superior

for storing, deleting, searching operations than TreeMap.

These collections HashMap and TreeMap will be of great value in improving performance of programs written in Java. Searching data in the Java ResultSet object is a slow process as search has to be peformed sequentially. So, loading data into any one collection object (HashMap or TreeMap) will result in efficient performance. The choice of the in-built collection depends on the need of the users, space and ordering vs speed.

VI. CONCLUSION

Java collections like HashMap and TreeMap are of great value, these collections help to develop efficient programs easily and gain efficiency in searching. HashMap which is based on the theory of Hashing is definitely superior for search, insert, remove, replace operations than TreeMap based on the concept of Red-Black tree. But if data is required in ordered format and searching back and forth according to the natural order of objects, then TreeMap is best choice compared to HashMap.

VII. REFERENCES

[1] Johari A., (2020), *Java Collections - Interface, List, Queue, Sets in Java With Examples*. Retrieved from edureka: https://www.edureka.co/blog/java-collections/

[2] Drozdek A., (2000), *Data Structures and Algorithms in Java*. New Delhi: Thomson Learning Brooks/Cole.

[3] Baeldung, 2020, *Java TreeMap vs HashMap*. Retrieved from Badeldung: https://www.baeldung.com/java-treemap-vs-hashmap

[4] Kruse L.R., Leung P.B.,Tondo L.C., (2000), Data Structures and Program Design in C, New Delhi: Prentice-Hall of India

[5] Yedidya Ll, Moshe J. A., Aaron M.T., (1998), Data Structures using C and C++, New Jersey: Prentice-Hall, Inc.

[6] Vogel L., (2019), *Java Collections - Tutorial*. Retrieved from Vogella: https://www.vogella.com/tutorials/JavaCollections/article.html

[7] Morris J., (2020), Red-Black Trees, https://www.cs.auckland.ac.nz/software/AlgAnim/red_black.html

[8] Lucas P., (2019), *Navigating Java Maps: TreeMap vs. HashMap vs. Linked HashMap*. Retrieved from Medium: https://medium.com/swlh/navigating-java-maps-treemap-vs-hashmap-vs-linked-hashmap-c97e6d248ecf

[9] Java Platform, (2020), *TreeMap (Java™ Platform, Standard Edition 8)*. Retrieved from Java™ Platform, Standard Edition 8

[10] Java Platform, (2020), *HashMap (Java™ Platform, Standard Edition 8)*. Retrieved from Java™ Platform, Standard Edition 8: https://docs.oracle.com/javase/8/docs/api/overview-summary.html

[11] Cormen H.T., Leiserson E.C., Rivest L. R., Stein C., (2002), *Introduction to Algorithms.* MA: Prentice-Hall of India Private Limited.

[12] TutorialsPoint, (2020), *Java.lang.System.nanoTime() Method*. Retrieved from TutorialsPoint Simplyeasylearning: https://www.tutorialspoint.com/java/lang/system_nanotime.htm

[13] Mikhail V., (2013), Aayushi, J. (2020, Feb 27). *Java Collections - Interface, List, Queue, Sets in Java With Examples*. Retrieved from edureka: https://www.edureka.co/blog/java-collections/