Algorithmic analysis of advancement in chaining method with BST & AVL

[1]Lalit kumar Saini, [2]Deepak Verma

[1]Assistant Professor, [2]Assistant Professor-II
[1] Department of CSE, [2]Department of CSE,
[1]Sobhasaria Group of Institutions, Sikar, Rajasthan, India
[2]JECRC University Jaipur, Rajasthan, India

Abstract: Searching is the important & core task in computers. Efficient searching of data is always the main concern. The data structure which accelerates the searching is hashing methods. There we generally do insert, delete and search data with some key. This key is generated by the some of the appropriate hash function i.e. mid square method, folding method etc. The problem of Collision in hashing occurs generally, for that so many data structures (methods) are used like chaining, addressing methods.
This paper first analyzes the separate chaining method additionally with upgraded approaches ie. chaining with balanced binary trees like BST and AVL. Some algorithms are also given to help in implementation for improving the performance of separate chaining and decreases the time of searching when element is searched.

***Index Terms*: Data Structure, Hashing, Separate Chaining, Binary Search Tree, AVL Tree**

1 INTRODUCTION

In the computer science data storage and security is always interesting field of research. So that there are so many ways to organized data in files or in database. Some operations (insert, delete and retrieval) which can generally perform on that stored data are crucial in context of speed and performance[1][2][3] .So from many years researchers try to developed the new techniques, algorithms and data structures to perform these operations as fast as possible and in convenient way. So when the database is larger, complexity is the major concern for these basic operations.
Hashing is the procedure to do these basic operations with optimal time complexity. We know that to retrieve any of the specific data we required the some unique key. So in turn data is stored with its unique key identification. In hashing hash function is used to generate these unique key with the data is stored and it also decreases the time of basic operations.

2 HASHING

Hashing is a technique for storing the data in array like structure called a **hash table** and retrieving data from that .It includes two main tasks **1.** Computation of an index key ***H (K)*** by some of the hash function ***H***. **2.** Stores the corresponding data in hash table with this computed index key ***H (K)***. Index key is the unique key and corresponding data to be stored in hash-table. If ***K*** is a data which is to be stored then ***H (K)*** = Index key (index in hash table).
In hashing selection of the good hash function [1, 2] is one of the major tasks. There are so many hash function schemes are available like folding method, mid Square method, mod method etc.

2.1 Classification of hashing

Hashing can be classified into two categories: Static hashing [1] and Dynamic hashing [1]. In static hashing the hash table is of limited size because the size of the table is predefined, generally done by array. The disadvantage of this technique is wastage of memory. It is used when the record to be stored is less in amount or numbers of records are fixed. In Dynamic hashing hash table has a size of the unlimited capacity, generally done through the link list, binary tree etc.

So memory is wasted. In dynamic hashing the performance does not degrades as the data size increases and size of the table not needed to define previously.

2.2 Collision

Collision is the main problem in above two techniques .The main idea behind the collision is when for two distinct data element K1 and K2 index keys are equal, such that h (K1) =h (K2), means they mapped into the same slot in hash table. So there are generally two methods to handle collision.1. Chaining: An array of the link list. 2. Open Addressing: Array based Implementation.

2.3 Separate chaining

A simple and efficient way for dealing with collision is separate chaining .In that we put all the elements, which is to be hashed ,to the same slot or index in hash table .In other words every link list has the elements that collides to the similar slot. Shows in figure 1.

Hash key=key%table_size
6 = 36%10
8 = 18%10
0= 20%10
9 = 29%10
7= 17%10
9 = 119%10
6 = 66%10
6= 186%10

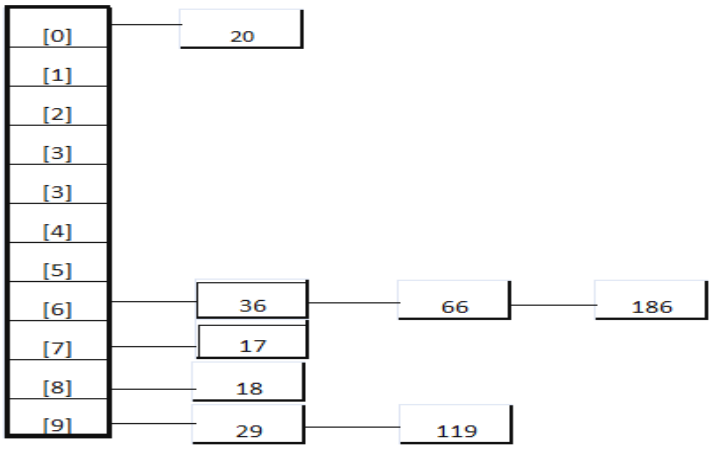

Figure 1. Separate Chaining

2.4 Analysis

2.4.1 Worst case analysis

The worst case appears when all elements are mapped into same key. Then if total no of elements that to be mapped is K and table size =N (total or maximum no of link lists to be created). Then separate chaining no longer works when the every mapped key collides due to same hash index or slot except first element. Then all the elements are in the same link list.

The searching complexity = time taken in search of index in hash table +time taken to search element
The searching complexity = $O(1) + O(n_i)$
$= O(K)$

2.4.2 Best or Average case analysis

The better case is when the distribution of elements is same over all the table indexes in separate chaining.

If elements in the link list at:
Table inde $x_1 = k_1$
Table inde $x_2 = k_2$
Table inde $x_3 = k_3$
.
.
Table $index_n = k_n$
Then total lengths of all lists are $K = k_1+k_2+k_3+k_4+\ldots\ldots+k_n$ *and* $k_1=k_2=k_3=k_4=\ldots\ldots=k_n$

Time complexity of searching of the element in any of the link list = ***O (K/N) or O (K/table size)***

one more case we have when the link list contains single element. In this condition the searching returns with constant amount of time.

By above analysis we easily assume that separate chaining is problematic in worst case and no longer works. There are following limitations of the separate chaining.

- The insertion, deletion & searching take O (K) time in worst case.
- When the data is inserted in the unsorted order that needs more time to search a element rather when data is sorted.

On the basic of above two limitations of separate chaining, we need better data structure for resolve collision.

3. EXTENDED TECHNIQUES

3.1 Literature survey

In last decades so many researchers are trying to find out the solution of decreasing the time complexity of in worst case . Some more extended data structures are created, implemented and analyzed. Daniel Sleator[1] in 1988 describe the working of hash tables and analyze its algorithms with use of Dynamic Hash Tables. Mahima Singh, Deepak Garg[5] in 2009 ompare the hashing collision resolving methods and find out when to use which technique. Saifullahi Aminu Bello, Ahmed Mukhtar Liman [6] in 2014 compares & analyzes the collision resolution techniques such as Quadratic probing and double hashing. Samir Raval, Prakruti Sharma [7] in 2014 somehow briefly tries to describe the use of AVL trees for collision resolution. Dapeng Liu, Shaochun Xu[8] in 2014 proves that close addressing has better stability than open addressing when they are used in an on-line application with a large set of data. Peter Nimbe, Samuel Ofori Frimpong , Michael Opoku[9] in 2014 presents NFO, a new and innovative technique for collision resolution based on single dimensional arrays. Akshay Saxena, Harsh Anand, Tribikram Pradhan[10] in 2015 proposes a hybrid chaining model which is a combination of Binary Search Tree and AVL Tree to achieve a complexity of O (log n). Dr. Vimal P. Parmar, Dr. CK Kumbharana[11] in 2017 , analyze all the techniques for searching the data by their search complexities .

3.2 Extended Technique

Extended technique proposed that each index of hash table contains binary search tree (BST)[4,12] instead of link list. We know that the BST is extended as the data grows.

Structure:

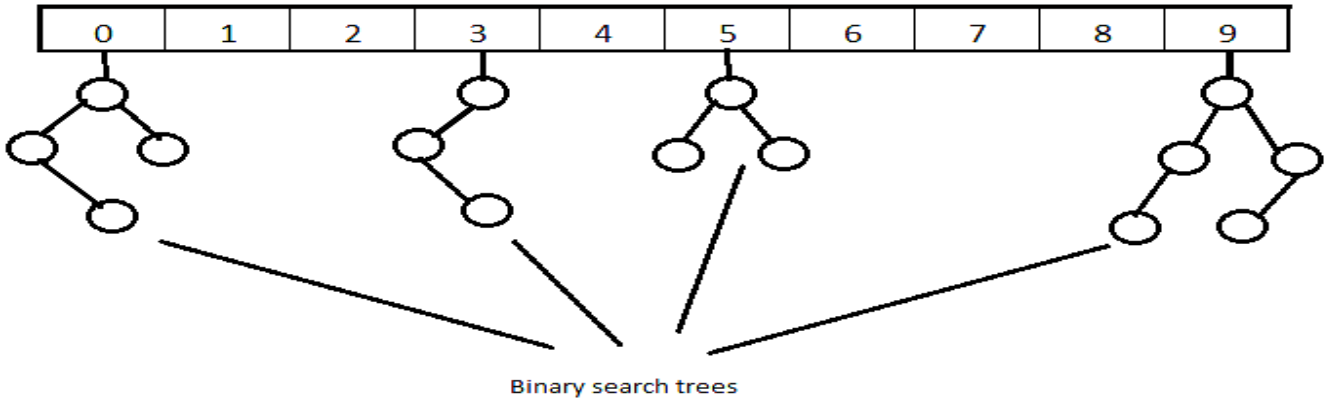

Figure 2. Structure of the hash table with BST (Binary Search Tree)

The binary search tree pre-order traversal gives sorted data and it is expendable when data amount increases. When elements collides and to be hashed on the same index in separate chaining with mod 10 (for Example) .Then the structure will be as follows:

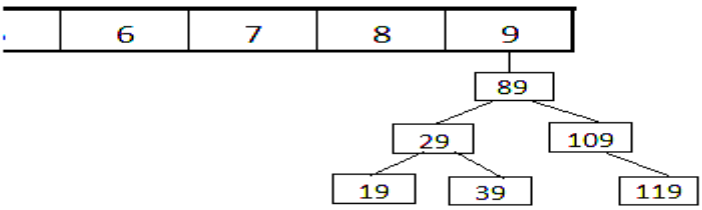

Figure 3. Hash table structure when all data collides at index 9

3.3 Analysis

3.3.1 Worst case

When all the keys are collides and hashed at the same index. (Total elements = **K**)

Case 1:- (Elements comes in Sorted Order)

Where all the keys come in the sorted order then the tree expands either on left side (when keys in descending order) or on right side (when keys come in increasing order). Then the complexity will be:

***O (K)* same as describe in section 2.4.1.**

Case 2:- (Elements comes in Unsorted Order)

When all the elements not in any sorted order and height of tree is minimum. Then the complexity will be

O (log K)

3.3.2 Best Case

When all keys are equally distributed over all the slots or indexes of hash table. Means each BST contains same elements. Most likely then:

Case 1 :- (Worst case of BST)

Where all the keys come in the sorted order, then the complexity will be

O (K/N) or O (K/Table size)

Case 2:-

When all the keys not in any sorted order and height of tree is minimum. Then the complexity will be

O (log (K/N)) or O (log (K/Table size))

So by the above analysis of separate chaining with BST we knew that the complexities of all the operations of hashing is going to be better except case1 of section 3.3.1

So for overcome this only one exceptional case we needs some improved in extended data structure. Next section 4 explains it how?

4. IMPROVED EXTENSION WITH AVL TREE

BST extension is worthless in one case (case1 section3.1.1). To overcome from this particular limitation we use the AVL tree, a Balance Binary tree.

4.1 AVL tree

An **AVL tree** [7] is tree with capability of improvement in deficiencies of binary search tree by self-balancing. In an AVL tree, the range of difference of height between two sub trees is -1 to +1. Lookup, insertion, and deletion all take O(log *n*) time in both the average and worst cases, where n is the number of nodes in the tree prior to the operation[13]. Insertions and deletions may require the tree to be re-balanced by one or more tree rotations.

4.2 Hash Table Structure with AVL tree

A hash table is an array of pointers equals to the table size. Each index of hash table contains the address of the individual or corresponding AVL tree.

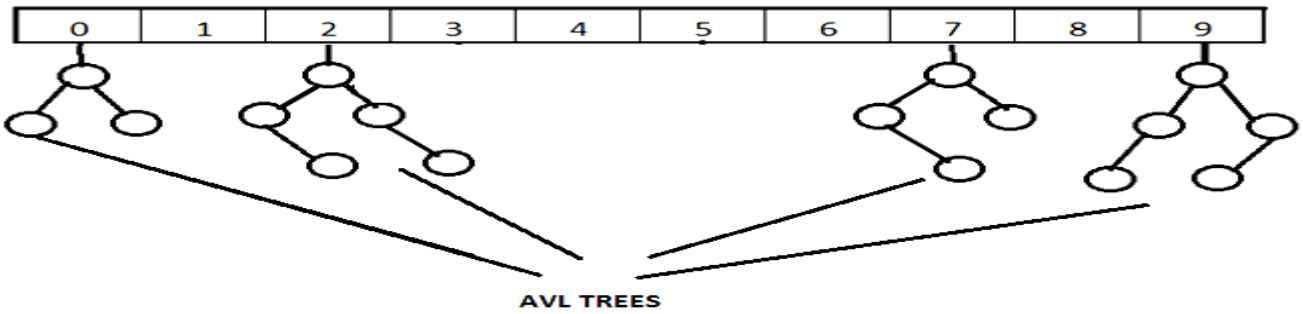

Figure 4. Structure of the hash table with AVL tree

In above figure on index or slot 0, 2, 7 and 9 we have AVL trees instead (from last extension) of BST.

4.3 Structure for hashing with AVL tree

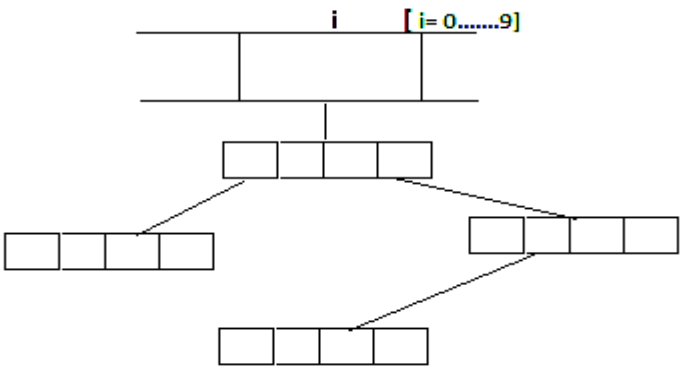

Figure 5. Individual tree structure on single Index with fields

4.4 Operations

There should three basic operations insertion, deletion and searching.

4.4.1 Insertion in Hash Table with AVL Extension

When the element is to be inserted we first calculate the hash index with the help of proper hash function. After that, search the index in hash table. If the hash index i has NULL at tree pointer field then first node of AVL will made and if tree pointer is not NULL then find the appropriate node, whose right sub tree pointer or left sub tree pointer contain the address of this new created node. Then check balancing factor, if unbalance then balanced with applying one or more proper rotations.

Algorithm for insertion

1. Calculate hash index i by hash_function.
2. If hash_index(tree_pointer)=NULL
   Then create the first node of AVL tree.
3. Else
   Search the proper position for insertion.
4. Insert new_node.
5. If balancing factor is unbalanced, balance it with one or more rotations.
6. End

Algorithm 1: Algorithm for inserting key

4.4.2 Deletion in Hash Table with AVL Extension

When the element is to be deleted, calculate the hash index by hash function. After that, search the index in hash table. If the hash index i has NULL at tree pointer field then no data deleted ,return with unsuccessful deletion and if tree pointer is not NULL then find the keys with comparing with node key , if not found again return with unsuccessful deletion . If match found delete it, then balance the tree with one or more rotations.

Algorithm for Deletion

1. Calculate hash index i by hash_function.
2. If hash_index(tree_pointer)=NULL
   Return unsuccessful deletion.
3. Else Search the node
   If key not match
   Return unsuccessful deletion.
   Else
   Delete node and do proper Rotations.
4. End

Algorithm 2: Algorithm for Deleting key

4.4.3 Searching in Hash Table with AVL Extension

When the element is to be searched, as previous, first calculate the hash index with hash function. Then search the index in hash table. If the hash index i has NULL at tree pointer field means not any item inserted at this index .If tree is present then search it in AVL tree of that particular index .If key is in a tree means search successful else return with unsuccessful search.

Algorithm for Searching

1. Calculate hash index i by hash_function.
2. If hash_index(tree_pointer)=NULL
   Then return Unsuccessful_search
3. Else
   Search in the AVL tree of index i.
4. If match found
   Return with searched key
   Or Successful_search
5. Else
   Return Unsuccessful_search
6. End

Algorithm 3: Algorithm for Searching key

4.5 Analysis

Previous section 3.3 analyze that the performance of operation in hashing is better than separate chaining. But in case1 of section 3.3.1 performance was not improved. So for the AVL tree all the operation complexity is ***O (log n)***, where n is the number of element.

4.5.1 Worst case
When all the keys are collides and hashed at the same index.

O (log K)
4.5.2 Best Case
When all keys are equally distributed over all the slots or indexes of hash table.
Means each AVL contains elements = ***K/N or K/Table size*** (maximum elements)
Then

O (log (K/N)) or O (log (K/Table size))

So the single limitation of case1 section 3.3.1, which is in the BST extension, is removed in AVL extension, because the element ordering not takes effect in AVL tree.

5. CONCLUSION

Generally the time complexity of an algorithm is depending upon many factors like numbers of input number of processors, other processes running at a time, CPU Temperature etc. But especially the no. of inputs is a main factor on which maximum analysis is done in this paper. So from the analysis done in last three sections anyone easily states that how the performance of searching and other operations in separate chaining is improved in proposed BST extension and then in AVL extension. Some algorithms are also given to design the proper implementation code. On the other side these two extensions is also shows improved performance when data is in the either in unsorted order or in sorted manner.

6. REFRENCES

[1.] Daniel Sleator "Dynamic Hash Tables" Algorithms and Data Structures , Communications of the ACM , April 1988 Volume 31 Number 4
[2.] Cormen, T. H., Leiserson, C. E., & Rivest, R. L "Introduction to Algorithms" India: Prentice-Hall of India Private Ltd. (2001)
[3.] B. J. M c kenzie, r. Harries and t. Bell, "Selecting a hashing algorithm",Department of Computer Science, University of Canterbury,Christchurch, New Zealand
[4.] Das, Gopal Chandra, Masud, Md. Mehendi, "A hashing technique separate binary tree". Data Science journal, Volume 5, 19 October 2006
[5.] Mahima Singh, Deepak Garg, "Choosing Best Hashing Strategies and Hash Functions",IEEE International Advance Computing Conference (IACC 2009) Patiala, India
[6.] Saifullahi Aminu Bello, Ahmed Mukhtar Liman ,Abubakar Sulaiman Gezawa "Comparative analysis of linear probing, quadratic probing and double hashing techniques for resolving collusion in a hash table", International Journal of Scientific & Engineering Research, Volume 5, Issue 4, April-2014
[7.] Samir Raval, Prakruti Sharma "Improving Hashing with AVL Tree", International Journal of Innovative and Emerging Research in Engineering , Volume 1, Issue 1, 2014
[8.] Dapeng Liu, Shaochun Xu, Zengdi Cui "An Empirical Study on the Performance of Hash Table", IEEE ICIS 2014, June 4-6, 2014, Taiyuan, China
[9.] Peter Nimbe "An Efficient Strategy for Collision Resolution in Hash Tables" International Journal of Computer Applications · August 2014
[10.] Akshay Saxena, Tribikram Pradhan, Harsh Anand ,"A Hybrid Chaining Model with AVL and Binary Search Tree to Enhance Search Speed in Hashing", International Journal of Hybrid Information Technology Vol.8, No.3 (2015), pp.185-194
[11.] Dr. Vimal P. Parmar, Dr. CK Kumbharana, "Designing and implementing data structure with Search algorithm to search any element from a Given list having constant time complexity",International Education & Research Journal [IERJ], Volume : 3,Issue : 1 ,Jan 2017
[12.] Dr. Vimal P. Parmar, Dr. CK Kumbharana,"Comparing Linear Search and Binary Search Algorithms To Search An Element From a Linear List Implemented Through Static Array, Dynamic Array And Linked List" - International Journal of Computer Applications (IJCA) 121(3) 13-17 -July 2015 Volume 121/ Number 3
[13.] Gajender, gaurav, himanshu sharma "AVL tree and hashing", International Journal of Innovative Research in Technology(ijirt) volume 1 issue 7
[14.] Kamlesh Kumar Pandey "A Comparison and Selection on Basic Type of Searching Algorithm in Data Structure" , International Journal of Computer Science and Mobile Computing, Vol.3 Issue.7, July- 2014, pg. 751-758