I.J. Intelligent Systems and Applications, **2018, 11, 27-35**
Published Online November 2018 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2018.11.04

Sorted r-Train: An Improved Dynamic Data Structure for Handling Big Data

Mohd Abdul Ahad
Department of Computer Science and Engineering, School of Engineering Sciences and Technology,
Jamia Hamdard, New Delhi-110062, India
E-mail:itsmeahad@gmail.com

Ranjit Biswas
Department of Computer Science and Engineering, School of Engineering Sciences and Technology,
Jamia Hamdard, New Delhi-110062, India
E-mail:ranjitbiswas@yahoo.com

Received: 19 March 2018; Accepted: 12 July 2018; Published: 08 November 2018

Abstract—In today's computing era, the world is dealing with big data which has enormously expanded in terms of 7Vs (volume, velocity, veracity, variability, value, variety, visualization). The conventional data structures like arrays, linked list, trees, graphs etc. are not able to effectively handle these big data. Therefore new and dynamic tools and techniques which can handle these big data effectively and efficiently are the need of the hour. This paper aims to provide an enhancement to the recently proposed "dynamic" data structure "r-Train" for handling big data. With the emergence of the "Internet of Things (IoT)" technology, real-time handling of requests and services are pivotal. Therefore it becomes necessary to promptly fetch the required data as and when required from the enormous piles of big data that are generally located at different sites. Therefore an effective searching and retrieval mechanism must be provided that can handle these challenging issues. The primary aim of this proposed refinement is to provide an effective means of insertion, deletion and searching techniques to efficiently handle the big data.

Index Terms—r-Train, trie, HAT, Linked List, Arrays, Big Data, larray.

I. Introduction

A dynamic data structure in the context of this paper is one that can store the data of variable sizes. When we talk about big data one prominent point that is pertinent to address is "How to store this enormous data". As these data are so huge that it cannot be stored in a single storage device, therefore we need multiple storage devices to store these big data. At the same time, we need to look for the storage techniques that will somehow link these storage devices together in one or the other way so that the process of data storage and retrieval remains uncomplicated and time efficient. The critical issue about big data lies in its intricate and heterogeneous nature. The major portion of big data generated today is largely unstructured. This means that there is no fixed structure attached with the data. Furthermore big data possesses several characteristics which are exponentially expanding with time and are often represented by 10 V's of big data in recent years. It is very important to understand the effect of each of these characteristics as they are closely linked to each other and sometimes overlaps with each other. If we are able to effectively identify and analyze these varying characteristics of big data, we can come up with system, tools and techniques to handle these huge datasets in an effective manner. The 10 V's of big data are shown in Fig. 1.

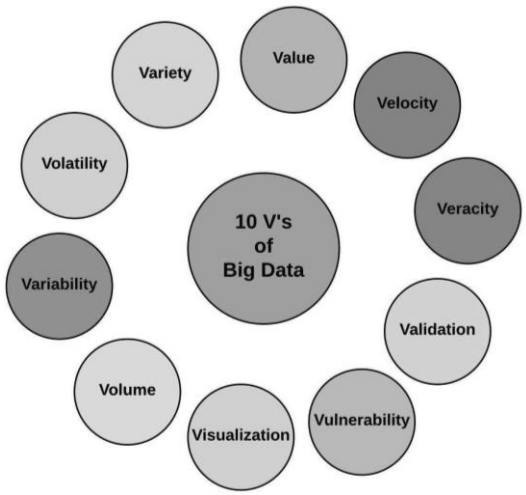

Fig.1. 10 V's of big data

There are several techniques evolved in recent times for handling such huge datasets. In [1], the author proposed a new type of dynamic data structure known as "r-Train". The core idea of the "r-Train" data structure is to store the data items in various "coaches" (which are like data items stored in a linked list). Each coach consists of some data items and the address to reach the next coach, forming a 'train' like structure.

A. r-Train Data Structure: A Brief Introduction

The author of [1] proposed a new dynamic "homogeneous data structure r-Train" which is used to store similar type of data. The main advantage of this data structure that was pointed out by 'R. Biswas' [1] is its ability to be scalable to any desired amount as and when required. The difference of r-Train from the data structure arrays lies in a way that, in array we cannot leave an empty space for a data in-between the elements already present in the array, but in case of r-Train it is quite possible. A coach of the 'r-Train' may consist of one or more empty spaces. The empty space is being represented by a 'ε' (also known as null element) which is treated as of same data type as the other elements of the coach. The definition given by R-Biswas asserts an r-Train as a "linked list of tagged coaches". The term tagged coach means that every coach consists of some data elements (including 'ε') and the information about how many more elements can be stored in it. The various formal terms and definition related to r-Train data structure are given below in brief [1]. However for detailed information about the functioning of "r-Train", one could see [1, 2, 3, 4, 5].

1) larray

A 'larray' is a collection of similar types of elements where one or more elements can be 'ε' element or null elements. Since 'ε' is treated as of same data type, therefore the same amount of space (as that for the other elements) will be reserved in the memory for storing "ε" elements. The length of the larray may be defined as the number of elements (including ε) stored in it [1]. The following are few examples of larray.

i. X = < 2,6,7, ε,8, ε> , the length of 'larray' X is 6
ii. Y = <12, 5, ε, ε, ε, 7, 100, 8>, the length of 'larray' Y is 8
iii. Z = < ε, ε, ε, ε>, the length of 'larray' Z is 4
iv. P = <1, 5, 8, 14, 7>, the length of 'larray' P is 5
v. Q = < 2 >, the length of 'larray' Q is 1
vi. R = < ε >, the length of 'larray' R is 1
vii. S = < >, S is an empty larray.

If we have a 'larray' where all the elements present in it are 'ε' (null) elements, then that array is known as "null larray". Here, larrays 'Z' and 'R' are said to be "null larrays". It can also be observed that the lengths of larray 'Z' and 'R' are '4' and '1' respectively. Therefore it can be concluded that "null larrays" are not unique.

2) Coach of an r-Train

A "Coach (C)" of an "r-Train" may be defined as a pair (LA, add) where "LA" is a "nonempty larray" (can include a "null larray" also) and "add" is the address of the next coach of the "r-Train". However, if the coach "C" is the last or the only coach of the "r-Train" then "add" will contain an invalid address value [1].

3	5	ε	ε	20	ε	24	ε	ε	e2

Fig.2. An Example of a Coach of an "r-Train"

Fig. 2 shows a coach C with the items of 'larray' as < 3, 5, ε, ε,20, ε,24, ε, ε, e2> where e2 being the address of next coach. Therefore while creating a coach it must be noted that the last block of the coach must be reserved for storing the address of the next coach and no element (ε or non- ε) can be stored in it [1].

3) Status of the Coach

It is defined as the number of empty spaces ('ε' elements) in the larray at a given point in time. It is denoted by "s" and has the value in the range between $0 \le s \le r$. Also the value of "s" is dynamic in nature and will change with every insertion or deletion of the element to or from the 'larray'. If the "status" of a coach is "0 (zero)" at any point of time, it signifies that the larray of this coach is full and no more elements can be added in this particular coach [1, 2]. If the "status" of the larray is equal to "r", it means that the larray is a "null larray" with all the elements as 'ε' and we can store up to a maximum of "r" number of elements in the larray [1].

4) Tagged Coach of an r-Train

If C (LA, add) is a "coach" of an "r-Train" then the "tagged coach (TC)" of the coach C, is given by a pair (C,s), where "C" is the "coach" and "s" is its "status". The tagged coach consists of the information about how many empty spaces are available in the coach at any given time. For example : If the Coach C (LA, add) is there with its larray 'LA' given by < 2, 5, 7, 21, ε, ε, 3, ε > then the tagged coach (TC) with respect to the coach C will be TC = (C, 3). As we have three number of "ε" elements present in the larray "LA" [1].

II. General Representation of r-Train Data Structure

As mentioned by the author in [1, 2, 3, 4, 5] an "r-Train" is a homogeneous, dynamic data structure which is highly scalable in nature. It makes the best use of both arrays and linked list data structures to come up with a more powerful and efficient way of managing big data (which arrays and linked lists lack in general) [1]. An "r-Train" may be treated as the "linked list of tagged coaches". The alphabet "r" represents the number of "tagged coaches (TC)" in the "r-Train". The general representation of an r-Train "T" of length "k" is given below in Fig. 3.

$T = < (C_1, S_{C1}), (C_2, S_{C2}), (C_3, S_{C3}), \ldots\ldots., (C_k, S_{ck}) >,$

Where C_1, C_2 ..... C_k are the tagged coaches and S_{c1}, S_{c2} ..... S_{ck} are their corresponding statuses

Also note that C_1, C_2 ..... C_k consists of the pairs (LA_1, add_1), (LA_2, add_2), (LA_3, add_3)....(LA_k, add_k) respectively.

Where LA_1, LA_2, LA_3 ..... LA_k are larrays with "d_{11}, d_{21}, d_{31} ... d_{kr}" are the data items and "add_1, add_2, add_3 ..... add_k" are the addresses of the next coach (i.e. add_1 holds the address of coach C_2, add_2 holds the address of Coach C_3 and so on. And 'add_k' holds an invalid address as it is the last coach of the r-Train) [1].

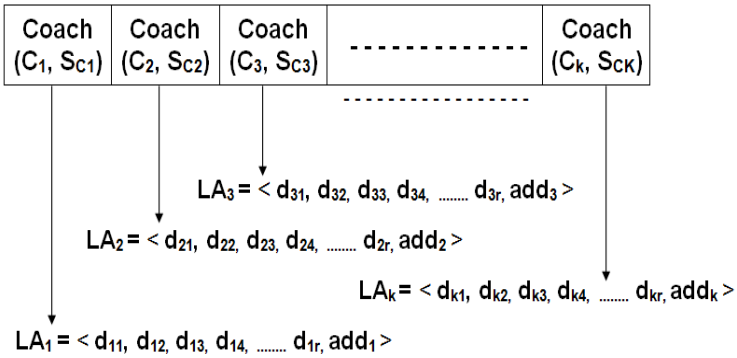

Fig.3. An example of an "r-Train" with "k" number of Coaches

Example: A 4-Train of length 3

Let the 4-Train is denoted by 'T' and is given by

T = < (C_1, 2), (C_2, 1), (C_3, 1), (C_4, 0) >

Where C_1 = < 1, ε, ε, add_1 >, C_2 < 4, 8, ε, add_2 >, C_3 = < 2, 5, ε, add_3 > and C_4 = < 7, 3, 9, Invalid Address >

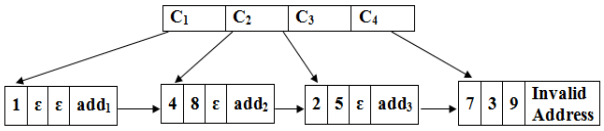

Fig.4. An example of 4-Train of length 3

Fig. 4 shows an example of 4-Train of length 3, similarly, we can create r-Train of any desired length. For further details about r-Train and its implementation, one could see [1].

III. Related Work

There are numerous data structures being proposed by various authors that work well for different needs and situations. The researchers in [6] proposed a "dynamic self adjusting" form of binary tree and gave it the name "splay trees". Here the authors used a restructuring heuristics known as splaying, every time a tree is being accessed. The data structure by the name "AVL Trees" was proposed by "G. M.Velskii & E.M. Landis" [7]. These types of trees are not perfectly balanced but the height of pair of subtrees differs by at most 1. They maintain the "search, insertion and deletion" time of O (log (n)). The researchers in [8] introduced a framework for the pre-processing and management of IoT big data. They also discussed the latest innovations, developments and challenges in managing big data generated from IoT devices. Nikolas Askitis Ranjan Sinha [9] proposed a HAT-trie data structure which is a type of "cache conscious trie" based data structure for strings. The authors in [10] proposed a "random matrix theory (RMT)" based approach to detect anomalies in smart grids. They also identified the system correlation in terms of "Mean Spectral Radius (MSR)". The researchers in [11] implemented the Chornos Software (a kind of time-based database) in C++ language. They claim to increase the processing efficiency by 40- 90 percent by storing the data and algorithms in RAM instead of main memory when compared with MongoDB and MYSQL databases. The authors in [12] reviewed the various techniques used for storing the big data generated from DNA and RNA fragment samples for performing "next-generation sequencing (NGS)". The author in [13] proposed a new data structured named "Hashed -Array-Trees" which combines the merits of arrays, hash tables and trees. The authors in [14] proposed a new dynamic technique for storing multidimensional point data which was based on quadtree like sub-division of space. Gu M et al in [15] presented a comprehensive review of the innovations in "nanophotonics-enabled optical storage techniques" and its uses for storing big data. The authors in [16] talked about the big data storage systems specially catering to high velocity, volumes and varieties of data. They also highlighted the various challenges in way of big data storage security. The authors of [17] presented an algorithm for maintaining "Dynamic AVL Trees". The researchers in [18, 19, 20, 21] discussed about the impact of Big data in the daily lives of humans. The challenges, opportunities, applications and advantages associated with the effective handling of big data were also discussed in their paper. The authors of [22] proposed the various indicators affecting the expansion of big data. In [23], the researchers highlighted the critical barriers and challenges that hinder the analysis of big data due to its heterogeneous and intricate nature. The researchers in [24] proposed a cloud based architecture by combining the advantages of existing BDA's to provide performance enhancements and QoS. The author in [25] discussed the important characteristics of big data in terms of knowledge discovery and highlighted the future prospects of KDD

with big data. In [26], the authors highlighted the need for securing big data and maintaining its authenticity and accuracy for gaining valuable insights from it. The researchers in [27] proposed a model for performing visualization and analytics of big data by transforming the data into structured format and visualizing it through graphs and charts. The authors of [28] proposed a privacy-preserved big data storage technique using "proxy- re-encryption" and anonymity.

IV. Sorted r-Train (The proposed Approach)

This paper proposes a modified approach for storing and searching the data in the "r-Train" data structure so that the data becomes sorted and searching can be performed efficiently and effectively. The "r-Train" data structure can be further improved by making a provision for deleting "any coach" of the r-Train (not only the last coach as in case of conventional r-Train). However, a coach will be deleted only when it has no non- ε elements stored in it. In the proposed approach after every insertion and deletion operation, the data is being sorted so that every data item comes in its proper place in an organized manner (ascending or descending). Since the sorting is done after the insertion or deletion operation, the end-user need not worry about the overhead involved. Furthermore, as we are getting a sorted data, the searching operation can be completed in an identical time complexity as in case of a binary search tree approach. The point to note here is, the data structure r-Train or Sorted r-Train are effective only when we have a huge amount of data or big data. If we have a limited amount of data, then using r-Train or Sorted r-Train will be an overhead to the programmer and thus should not be used.

The steps given below present the proposed modification. (We are assuming that the length of the larray is 'r' (just to present a generalized approach))

1. Let "BEGIN" be the address of the first Coach (C_1).
2. Create the first Coach C_1 with all 'larray' items in it as "ε" elements and "add_1" as the address pointing to an invalid value (since C1 is the only coach we have at this point of time).
   An example of this type of coach will be
   $C_1 = < \varepsilon 1, \varepsilon 2, \varepsilon 3, \varepsilon 4, ... \varepsilon r, add_1 >$
3. Find the "tagged coach (TC_1)" of the coach C_1
   - $TC_1 = (C_1, s)$, where "s" denotes the "status of the coach C_1". At this point of time the value of "s" will be equal to "r", since all the elements in the coach C_1 are "ε" (null elements).

Let us assume that we have the indices first, mid and last, wherein first denotes the index location of the first non- ε element of the larray, last denotes the index location of the last but one element of the larray and mid denotes the index location of ((first +last)/2) of the larray. (Point to note here is that the elements at these locations (first, mid, or last) can be 'ε' as well)

A. Insertion Operation

The "insertion operation" in the "Sorted r-Train" can also be of two types as we have in case of classical "r-Train" data structure.

i. Insertion of a "New data item" in a "Coach".
ii. Insertion of a "New Coach" in the "Sorted r-Train"

a. Insertion of a "New data element" in Sorted r-Train

The insertion of new element is done as per the following algorithm:

Algorithm: LA_Insertion

Let the Status of Coach (C) = Number of "ε" elements in the Coach and let the Length of Coach (C) = Number of Elements that can be stored in the coach = K

```
1.  FOR each New Element, DO
2.  FOR Each Coach in the Sorted r-Train, DO
3.  If Status (S) of Coach (C) = 0
    a. Print " Coach Full, Check the status of Next Coach for Insertion
4.  END IF
5.  If status (S) of Coach = Length of Coach (K)
    a. Insert the Element at LA[0] (First location in the Larray) in the Coach
    b. Update the Status of the Coach as S= S-1
6.  END IF
7.  Else if (0<S < K)
    a. Loc= SearchLocation(LA) (Location where the new element will be inserted
    b. Insert (LA, element, Loc) // insert the element at location Loc
    c. Sort (LA) // Sort the larray
    d. Update the Status of the Coach as S= S-1
8.  End ELSE IF
9.  END FOR
10. END FOR

Function: SearchLocation(LArray)
int SearchLoc(LA)
{
return (LA.IndexOf(ε)) // returns First index of 'ε' in the larray
}
```

Let us take few exmple for showing the insertion opertation.

Example 1:

Let the given larray be
LA = < 3, 5, 7, ε, 26, 32, 39, ε, add >
We can easily find out the following:
Size of the larray = 8 (No. of elements in larray)
Status of the Coach = 2 (No. of 'ε' elements in larray)
Index of larray element '3' = 0
Index of larray element '5' = 1
Index of larray element '7' = 2
Index of larray element 'ε' = 3
Index of larray element '26' = 4
Index of larray element '32' = 5
Index of larray element '39' = 6
Index of larray element 'ε' = 7

Since Status (S) of Coach is > 0, we can insert the element in this coach.

Location (index) for insertion = SearchLocation(LA)
= 3
Let the element to be inserted is '4'.
Therefore '4' will be inserted at index 3 of larray.
So, the larray after insertion becomes
LA = (LA) = < 3, 5, 7, **4**, 26, 32, 39, ε, add >
Now, sort the larray to get the final larray after insertion
Therefore LA = <3, **4**, 5, 7, 26, 32, 39, ε, add >
This completes the insertion operation

Example 2:

Let the given larray be
LA = < 9, 12, 17, 19, 38, 39, 45, 90 ε, add >
From the larray above, we can easily find out the following:
Size of the larray = 9 (No. of elements in larray)
Status of the Coach = 1 (No. of 'ε' elements in larray)
Index of larray element '9' = 0
Index of larray element '12' = 1
Index of larray element '17' = 2
Index of larray element '19' = 3
Index of larray element '38' = 4
Index of larray element '39' = 5
Index of larray element '45' = 6
Index of larray element '90' = 7
Index of larray element 'ε' = 8

Since Status (S) of Coach is > 0, we can insert the element in this coach.

Location (index) for insertion = SearchLocation(LA)
= 8
Let the element to be inserted is '14'.
Therefore 14 will be inserted at index 8 of larray.
So, the larray after insertion becomes
LA = < 9, 12, 17, 19, 38, 39, 45, 90, **14**, add >
Now, sort the larray to get the final larray after insertion
Therefore LA = <9, 12, **14**, 17, 19, 38, 39, 45, 90 add >
This completes the insertion operation

Example 3:

Let the given larray be
LA = < 13, 25, 77, ε, ε, ε, add >
From the given larray, we can easily find out the following:
Size of the larray = 6 (No. of elements in larray)
Status of the Coach = 3 (No. of 'ε' elements in larray)
Index of larray element '13' = 0
Index of larray element '25' = 1
Index of larray element '77' = 2
Index of larray element 'ε' = 3
Index of larray element 'ε' = 4
Index of larray element 'ε' = 5

Since Status (S) of Coach is > 0, we can insert the element in this coach.

Location (index) for insertion = SearchLocation(LA)
= 3
Let the element to be inserted is '44'.
Therefore '4' will be inserted at index 3 of larray.
So, the larray after insertion becomes
LA = (LA) = < 13, 25, 77, **44**, ε, ε, add >
Now, sort the larray to get the final larray after insertion
Therefore LA = <13, 25, **44**, 77, ε, ε, add >
This completes the insertion operation

b. Insertion of new coach at the end in Sorted r-Train

This is very much similar to the insertion technique of r-Train data structure by R. Biswas [1]. Let the new coach to be inserted is C_{k+1}, perform the following steps to insert this coach in the Sorted r-Train

1. Create the new Coach C_{k+1} with all 'r' larray items in it as 'ε'
   An example of this type of coach will be
   C_{k+1} = < *ε1, ε2, ε3, ε4, ... εr , add*$_{k+1}$>
1. Update the pilot (linked list)
2. Update the address part of the Coach C_k and assign the address of Coach C_{k+1} to it.
3. Set the address part of Coach C_{k+1} to an invalid address value.
4. Set the status of Coach C_{k+1} = r.

B. Search Operation

Let us assume that we have a Sorted r-Train as
T = < (C_1,S_{c1}),(C_2,S_{c2}),(C_3,S_{c3}),...,(C_i,S_{ci}) ... (C_k, S_{ck}) >
Where the coach C_i is (LA_i , add_i) where "LA_i" is the larray and "add_i" stores the "address of the next coach C_{i+1}" or an "invalid address" in case C_i is the last coach (when i= k) and S_{ci} denotes the "status of coach C_i", for i = 1,2,3…k.
Let the element to be searched is "num".
There can be two cases when we want to search for a particular element 'num'

1. When we know the coach number of 'num', we can directly go to that coach by visiting the pilot and perform "binary search" on that coach to match the elements of larray with 'num'.
2. When the coach number is not known, we will proceed in the following manner
   - Start from the first coach C1, and perform the operations given below till the element is found.

Let 'LA' be a larray, 'first' and 'last' denotes the index location of first non- ε element and last but one element of the larray respectively (note that 'last' here means 'length of the larray - 1', as the last element of the larray is the address of the next coach which is not taken into account for these calculations)
The pseudo code for searching the element is given below:

Algorithm: LA_Search

```
boolean LA_Search (LA, first, last, num)
1.  {
2.  mid=(first +last)/2;
3.  if (first >last) return false;
4.  found=false;
5.  if(LA[mid] = = num){
6.  Print ("Element Found at Location"+mid);
7.  return true;
8.  }
9.  else if(A[mid] = = 'ε')
10. {
11. found=Search(LA, mid+1, last, num);
12. }
13. else if(LA[mid] > num){
14. found=Search(LA, first, mid-1, num);
15. }
16. else
17. {
18. found=Search(LA, mid+1, last, num);
19. }
20. return found;
21. }
```

Let us take few exmple for showing the search opertation.

Example 1:

Let the given larray be:

LA = < 3, 5, 7, 4, 26, 32, 39, ε, add >
Let the element to be searched = 4
Now from the given larray, we can find first and last indexes.
first = 0
last = 7
mid = (first +last)/2
= (0+7)/2
= 3
Now check if (4 = = LA [mid])
i.e Is (4 = = LA [3]), YES, Since LA [3] = 4
Therefore element Found at index 3. The search operation successfully completes here.

Example 2:

Let the given larray be:

LA = < 13, 25, 37, 46, 60, ε, add >
Let the element to be searched = 60
Now from the given larray, we can find first and last indexes.
first = 0
last = 5
mid = (first +last)/2
= (0+5)/2
= 2
Now check if (60 = = LA [2])
Is (60= = LA [2]), NO, Since LA [2] = 37
Now we will find new value of low (index) as
low = mid+1
Therefore, low = 2+1 = 3
mid = (3+5)/2 = 4
Is (60 = = (LA [4]), YES, Since LA [4] = 60
Therefore element found at index 4. The search operation successfully completes here.

C. Deletion Operation

We can have the following three types of "deletion operations" in the proposed "Sorted r-Train" analogous to classical "r-Train" [1, 2].

1. Deletion of the data item from a Coach in the Sorted r-Train (here deletion operation refers to replacing the value of the deleted item with "ε").
2. Deletion of the last Coach of Sorted r-Train
3. Deletion of a Coach from between the Two Coaches

a. Deletion of the data item from a coach (C)

1. Let the element to be deleted is "k"
2. If the coach number of the element "k" is known, we will directly go to that coach by accessing the pilot and search for "k" using search technique as discussed in previous section (Searching operation).
3. After finding "k" within the coach, we will replace its value by "ε". And change the status of the coach as s = s+1 (An important point to note here is, deletion operation does not have any effect on the size of the coach.)
4. Sort the larray (LA) (so that all 'ε' elements comes at the beginning of the larray)
5. If the coach number of the element "k" is not known, we will start from the first coach C_1 say, and search for the element "k" using the search technique discussed in above section (searching technique)
6. After finding 'k' within any coach C_i (for i= 1, 2, 3, ... n) we will replace its value by "ε".
7. Sort the larray (LA)
8. Change the status of the coach C_i as s= s+1

The sorting of elements after every deletion operation ensures that all the 'ε' elements are always present in the beginning of the larray, this makes the process of insertion and searching less complex.

b. Deletion of the last Coach of Sorted r-Train

The deletion of a coach "C_i" is possible if and only if it is an empty coach (i.e all the elements in the coach are "**ε**" elements, as depicted here under in Fig. 5)

ε	ε	ε	ε	ε	ε	**Invalid Address**

Fig.5. An example of an "Empty Last Coach" of an r-Train

For deleting the last coach C_i, of a Sorted r-Train, we proceed as follows:

a) Update the address "add_{i-1}" of the coach "C_{i-1}" by storing an "invalid address" in it.
b) Delete the "coach (C_i, r)" from the Sorted r-Train
c) Update the pilot.

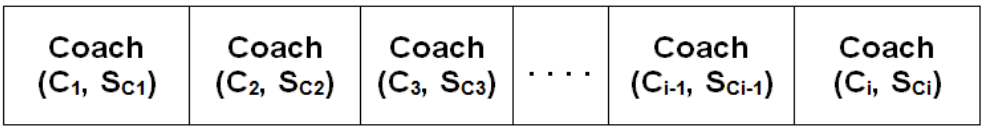

Fig.6. (a) An r-Train before deletion operation

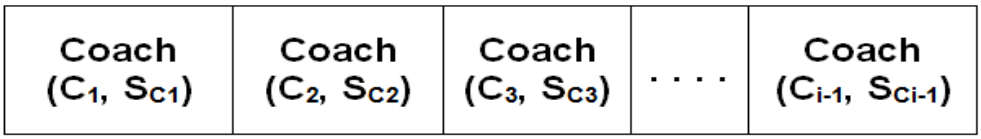

Fig.6. (b) An r-Train after deletion operation is performed

It can be observed from the above Fig. 6(a) and Fig. 6(b) that the coach (C_i,Sc_i) has been deleted.

1) Deletion of a Coach that lies between the Two Coaches in the Sorted r-Train

For deleting the coach 'C_i', of a Sorted r-Train which is between C_{i-1} and C_{i+1}, we proceed as follows:

a) Update the address "add $_{i-1}$" of the coach "C_{i-1}" by storing the address of "C_{i+1}" in it.
b) Delete the coach (C_i, r) from the r-Train
c) Update the pilot.

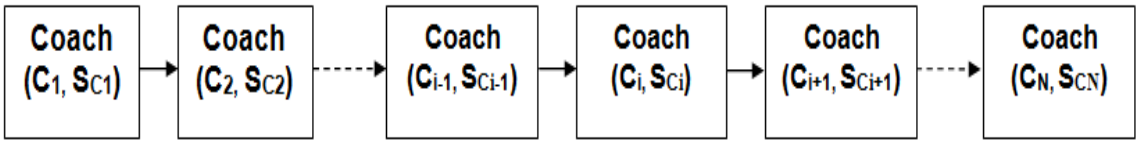

Fig.7. (a) An r-Train before deletion operation

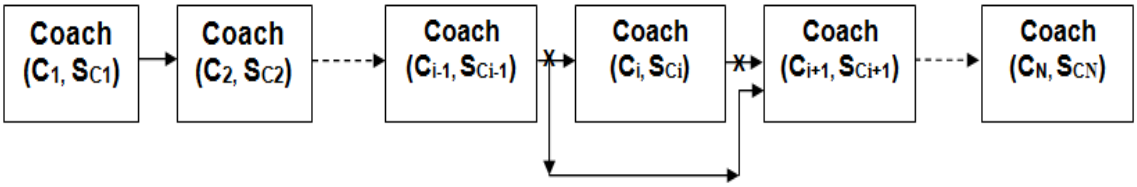

Fig.7. (b) An r-Train after deletion operation

Fig. 7(a) and Fig. 7(b) given above shows the deletion operation. Here the Coach C_i has been deleted from the r-train.

V. Conclusion

The data structure "r-Train" proposed by R. Biswas [1] is a dynamic and highly scalable data structure which can be used for handling big data in an efficient and effective manner. As mentioned in [29], the "r-Train" data structured can be very successfully used for processing the data in parallel for executing various tasks. The proposed modification "Sorted r-Train" has further improved the time required for searching an element stored in the "r-Train" data structure. As the data is being stored in a sorted manner, we can very effectively apply binary search technique (which runs in O (log (n)) time in the worst case) in order to search the data items [30]. Moreover, deletion of any coach is made possible by just adjusting the address fields of the previous and next coach of the deleted coach (analogous to Linked list [30, 31]). The proposed approach of Sorted r-Train data structure can be used to store data generated from IoT devices as it requires real-time analysis and processing. With Sorted r-Train, the data storage and retrieval time can be reduced to a greater extent as compared to the classical r-Train data structure. With the intricate and heterogeneous nature of big data, it is imperative to device new techniques and approaches for effective handling of these huge datasets. Sorted r-Train is an attempt towards the same. In future, the application of sorted r-train data structure for storing big data generated from IoT devices can be explored in order to perform real time query processing and analysis.

References

[1] Biswas, Ranjit. "'Atrain Distributed System"(ADS): An Infinitely Scalable Architecture for Processing Big Data of Any 4Vs." In *Computational Intelligence for Big Data Analysis*, pp. 3-54. Springer, Cham, 2015.

[2] Biswas, Ranjit. "Heterogeneous Data Structure "r-Atrain"." In *Global Trends in Intelligent Computing Research and Development*, pp. 338-359. IGI Global, 2014. doi:10.4018/978-1-4666-4936-1.ch012

[3] Biswas, Ranjit. "Introducing Data Structures for Big Data." In *Effective Big Data Management and Opportunities for Implementation*, pp. 25-52. IGI Global, 2016.

[4] Biswas, Ranjit. "Data Structures for Big Data."

International Journal Computing and Optimization, no. 2, 73-93 (2014).

[5] Biswas, R.,Processing of Heterogeneous Big Data in an Atrain Distributed System (ADS) Using the Heterogeneous Data Structure r-Atrain. *International Journal Computing and Optimization*, 1(1), pp.17-45, (2014). doi: http://dx.doi.org/10.12988/ijco.2014.445

[6] Sleator, Daniel Dominic, and Robert Endre Tarjan. "Self-adjusting binary search trees." *Journal of the ACM (JACM)* 32, no. 3 (1985): 652-686. dio:http://dx.doi.org/10.1145/3828.3835

[7] G.M.Velskii & E.M. Landis. An algorithm for the organization of information. *Soviet Mathemtics Doklady* 3:1259-1263, 1962.

[8] H. Cai, B. Xu, L. Jiang and A. V. Vasilakos, "IoT-Based Big Data Storage Systems in *Cloud Computing: Perspectives and Challenges," in IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 75-87, Feb. 2017. doi: 10.1109/JIOT.2016.2619369

[9] Askitis, Nikolas, and Ranjan Sinha. "HAT-trie: a cache-conscious trie-based data structure for strings." In *Proceedings of the thirtieth Australasian conference on Computer science*-Volume 62, pp. 97-105. Australian Computer Society, Inc., 2007.

[10] He, Xing, Qian Ai, Robert Caiming Qiu, Wentao Huang, Longjian Piao, and Haichun Liu. "A big data architecture design for smart grids based on random matrix theory." *IEEE transactions on smart Grid*, 8, no. 2 (2017): 674-686.

[11] M. Tahmassebpour, "A New Method for Time-Series Big Data Effective Storage," in *IEEE Access*, vol. 5, pp. 10694-10699, 2017. doi: 10.1109/ACCESS.2017.2708080

[12] Bertil Schmidt, Andreas Hildebrandt, Next-generation sequencing: big data meets high performance computing, *Drug Discovery Today*, Volume 22, Issue 4, 2017, Pp: 712-717, doi: https://doi.org/10.1016/j.drudis.2017.01.014.

[13] Sitarski, E.: HATs: Hashed array trees. *Dr. Dobb's Journal* 21(11) (1996), http://www.ddj.com/architect/184409965?pgno=5

[14] Park, Eunhui, and David M. Mount. "A self-adjusting data structure for multidimensional point sets." In *European Symposium on Algorithms*, pp. 778-789. Springer, Berlin, Heidelberg, 2012. doi =http://dx.doi.org/10.1007/978-3-642-33090-2_67

[15] Gu, Min, Xiangping Li, and Yaoyu Cao. "Optical storage arrays: a perspective for future big data storage." *Light: Science & Applications*, 3, no. 5 (2014): e177.

[16] Strohbach M., Daubert J., Ravkin H., Lischka M. Big Data Storage. In: *Cavanillas J., Curry E., Wahlster W. (eds) New Horizons for a Data-Driven Economy*. 2016, Springer, Cham

[17] Van Doren, James R., and Joseph L. Gray. "An algorithm for maintaining dynamic AVL trees." In *Information Systems*, pp. 161-180. Springer, Boston, MA, 1974.

[18] Berman, Jules J. *Principles of big data: preparing, sharing, and analyzing complex information.* Newnes, 2013.

[19] Feinleib, *David. Big Data Demystified: How Big Data is Changing the Way We Live, Love, and Learn*. Big Data Group, 2013.

[20] Needham, J.: Disruptive Possibilities: *How Big Data Changes Everything*. O'reilly Publisher, Cambridge (2013)

[21] Simon, P.: *Too Big to Ignore: The Business Case for Big Data*. John Wiley & Sons, New Jersey (2013)

[22] Makrufa Sh. Hajirahimova, Aybeniz S. Aliyeva, "About Big Data Measurement Methodologies and Indicators", *International Journal of Modern Education and Computer Science(IJMECS)*, Vol.9, No.10, pp. 1-9, 2017.DOI: 10.5815/ijmecs.2017.10.01

[23] Rasim M. Alguliyev, Rena T. Gasimova, Rahim N. Abbasl ,"The Obstacles in Big Data Process", *International Journal of Information Technology and Computer Science(IJITCS)*, Vol.9, No.4, pp.31-38, 2017. dio: 10.5815/ijitcs.2017.04.05

[24] Entesar Althagafy, M. Rizwan Jameel Qureshi,"Novel Cloud Architecture to Decrease Problems Related to Big Data", *International Journal of Computer Network and Information Security (IJCNIS)*, Vol.9, No.2, pp.53-60, 2017. doi: 10.5815/ijcnis.2017.02.07

[25] Mai Abdrabo, Mohammed Elmogy, Ghada Eltaweel, Sherif Barakat,"Enhancing Big Data Value Using Knowledge Discovery Techniques", *International Journal of Information Technology and Computer Science(IJITCS)*, Vol.8, No.8, pp.1-12, 2016. doi: 10.5815/ijitcs.2016.08.01

[26] PankajDeep Kaur, Awal Adesh Monga,"Managing Big Data: A Step towards Huge Data Security", *International Journal of Wireless and Microwave Technologies(IJWMT)*, Vol.6, No.2, pp.10-20, 2016. doi: 10.5815/ijwmt.2016.02.02

[27] Rohit More, R H Goudar,"DataViz Model: A Novel Approach towards Big Data Analytics and Visualization", *International Journal of Engineering and Manufacturing(IJEM)*, Vol.7, No.6, pp.43-49, 2017. doi: 10.5815/ijem.2017.06.04

[28] Liang, Kaitai, Willy Susilo, and Joseph K. Liu. "Privacy-preserving ciphertext multi-sharing control for big data storage." *IEEE transactions on information forensics and security*, 10.8, (2015): 1578-1589.

[29] Alam, B.: Matrix Multiplication using r-Train Data Structure. In: *AASRI Conference on Parallel and Distributed Computing Systems, AASRI (Elsevier) Procedia* 5, 189–193 (2013), doi: 10.1016/j.aasri.2013.10.077

[30] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms, 2nd edn*. MIT Press and McGraw-Hill (2001)

[31] Xiong, Qing, Chanle Wu, Jianbing Xing, Libing Wu, and Huyin Zhang. "A linked-list data structure for advance reservation admission control." In *Networking and mobile computing*, pp. 901-910. Springer, Berlin, Heidelberg, 2005.

Authors' Profiles

Mohd Abdul Ahad is working as an Assistant Professor in the Department of Computer Science and Engineering, School of Engineering Sciences and Technology, Jamia Hamdard, New Delhi, India. He has a rich experience of 9 years in computer Science and Engineering. His research interests include Big Data, IoT, and Distributed Systems. He is a member of IEEE, ACM, ISTE. He has worked as a review and editorial member of several International Journals.

Dr. Ranjit Biswas is working as a Professor in the Department of Computer Science and Engineering, School of Engineering Sciences and Technology, Jamia Hamdard, New Delhi, India. He has a vast experience of more than 33 years in computer Science and Engineering. His research interests include Fuzzy Systems, Soft Sets, Big Data, and Distributed Systems.

How to cite this paper: Mohd Abdul Ahad, Ranjit Biswas, "Sorted r-Train: An Improved Dynamic Data Structure for Handling Big Data", International Journal of Intelligent Systems and Applications(IJISA), Vol.10, No.11, pp.27-35, 2018. DOI: 10.5815/ijisa.2018.11.04

