Int. Journal of Computing and Optimization, Vol. 1, 2014, no. 2, 73 - 93
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ijco.2014.4813

Data Structures for Big Data

Ranjit Biswas

Department of Computer Science
Jamia Hamdard University
Hamdard Nagar, New Delhi–110062, India

Abstract

The present day world dealing with big data (expanding very fast in 4Vs**:** Volume, Varity, Velocity and Veracity) needs **N**ew advanced kind of logical and physical storage structures, **N**ew advanced kind of heterogeneous data structures, **N**ew mathematical theories and **N**ew models for processing giant big data in 4Vs. In the literature, there is no appropriate data structure for big data, no appropriate network topology for big data, no appropriate distributed system for big data. The existing data structures, network topologies and type of distributed system are not sufficiently rich to deal with big data. The existing network topologies like tree topology or bus/ring/star/mesh/hybrid topologies seem to be weak topologies for big data processing. For a success, there is no other way but to develop 'new data structures', 'new type of distributed systems' having a very fast and tremendous extent of mutual compatibility and mutual understanding with the new data structures, 'new type of network topologies' to support the new distributed system, and of course 'new mathematical/logical theory' models. Needless to mention that the next important issue is how to integrate all these 'new' to make ultimately a single and simple scalable system to the laymen users. With these views in mind, Biswas [11, 12, 15] introduced a new special type of fundamental data structure **'atrain'** exclusively for heterogeneous big data (where, the data structure 'train' is exclusively for homogeneous big data) and then introduced a new distributed system called by 'Atrain Distributed System' (**ADS**) which can process big data of any 4V of any momentum. The rich merit of ADS [15] is due to its layout architecture making it infinitely scalable both in breadth (horizontally) and depth (vertically). The 'Atrain Distributed System' (ADS) could be unitier or multitier having a single unique Pilot Computer (PC) and several Distributed Computers (DCs) connected by new type of network topologies called by ‘Multi-horse Cart Topology’ and ‘Cycle Topology’. ADS is completely compatible with the data structure ‘atrain’. The data structure ‘atrain’ [11, 12, 15] is a very powerful dynamic and flexible data structure, being the first data structure introduced exclusively for big data. Thus **‘Data Structures for Big Data’** is to be regarded as a new subject in Big Data Science, not just as a new topic, considering the explosive momentum of the big data. With philosophically eyes, big data could be viewed as a higher order tensor. Based upon the train-atrain notion of Biswas [11, 12, 15], in this paper the author introduces few more data structures for big data **:** homogeneous stacks ‘train stack’ and ‘rT-coach stack’, heterogeneous stacks ‘atrain stack’ and ‘rA-coach stack’, homogeneous queues ‘train queue’ and ‘rT-coach queue’, heterogeneous queues ‘atrain queue’ and ‘rA-coach queue’, homogeneous binary trees ‘train binary tree’ and ‘rT-coach binary tree’, heterogeneous binary trees ‘atrain binary tree’ and ‘rA-coach binary tree’, homogeneous trees ‘train tree’ and ‘rT-coach tree’, heterogeneous trees ‘atrain tree’ and ‘rA-coach tree’, to enrich the subject ‘Data Structures for Big Data’ for big data science.

Mathematics Subject Classification: 68P05, 68M14, 11C20, 15B36

Keywords: train, CD-table, atrain, atrain distributed system (ADS), ‘Multi-horse Cart Topology’ and ‘Cycle Topology’, train stack, atrain stack, rt-coach stack, rA-coach stack, train queue, atrain queue, rT-coach queue, rA-coach queue, train binary tree, atrain binary tree, rT-coach binary tree, rA-coach binary tree, train tree, atrain tree, rT-coach tree and rA-coach tree

1 Introduction

The present world of big data [5 - 7, 9, 10 - 14, 15, 18] are expanding very fast in 4Vs: Volume, Varity, Velocity and Veracity, and also in many more directions. How to deal with explosive momentum of big data, how to process big data in an efficient way within limited resources, etc. are of major concern to the computer scientists now - a - days. Big data has variable mass of 4Vs in various directions, and in this sense it is to be philosophically regarded as a vector, or rather as a tensor. The existing data structures of computer science are neither appropriate nor sufficient to deal with the big data. The homogeneous data structure r-Train and the heterogeneous data structure r-Atrain for big data are introduced by Biswas in [11, 12, 15], being the first attempt to introduce any exclusive data structure for big data. The architecture of a new type of distributed system ‘Atrain Distributed System’ (ADS) exclusively designed for processing big data is then introduced by Biswas in [15]. The huge merit of ADS stands on the fact that both the data structures train and atrain are 100% compatible with the new distributed system ‘ADS’ for processing big data, in particular to coup easily with 4Vs of any momentum.

It is obvious that the **'Data Structures for Big Data'** is to be regarded as a new subject in big data science, not just as a new topic, considering the explosive momentum of the big data in a new universe. One could view big data with philosophical eyes as a higher order tensor. The data structures train and atrain with ADS and the mathematical models like 'solid matrix/latrix', hematrix/helatrix to store big data of any big amount of any datatype have opened easy gates to the world giant organizations and their developers to deal with big data. In this paper we introduce few more new data structures which are **:** homogeneous stacks 'train stack' and 'rT-coach stack'; heterogeneous stacks 'atrain stack' and 'rA-coach stack'; homogeneous queues 'train queue' and 'rT-coach queue'; heterogeneous queues 'atrain queue' and 'rA-coach queue'; homogeneous binary trees 'train binary tree' and 'rT-coach binary tree'; heterogeneous binary trees 'atrain binary tree' and 'rA-coach binary tree'; homogeneous trees 'train tree' and 'rT-coach tree'; heterogeneous trees 'atrain tree' and 'rA-coach tree'; to enrich the subject 'Data Structures for Big Data' with the extended notion of the architecture of the data structures r-atrain and r-train of Biswas [11, 12, 15] for big data. As a pre-requisite to understand the work of this paper, one could see the work of Biswas [11, 12, 15] to get introduced with the basics of r-train, r-atrain and ADS for big data.

2 r-Train Stack and r-Atrain Stack

In this section we introduce two new data structures for big data:

(1) 'r-train stack' or 'train stack' in short, and

(2) 'r-atrain stack' or 'atrain stack' in short.

(The natural number r is suitably predecided and fixed by the programmer depending upon the problem of Big Data under consideration and also upon the organization/industry for which the problem is posed. We do not feel it appropriate to propose r in a r-train or in a r-atrain as a variable).

2.1 Introducing r-Train Stack (or Train Stack, in short)

First of all we define few terminologies.

2.1.1 Coach of a 'Train Stack'

The term coach here is coined from the notion of the data structures train and atrain for big data [11, 12, 15]. By a coach C in a train stack we mean a non-empty larray (however, it could be a null larray) containing data of homogeneous data type.

The following figure shows a coach of a 7-train in a '7-train stack' as a larray.

69.37	5.75	0.48	95.26	732.03	0.40	571.86	e

Figure 1. a coach of a train in a 7-train stack

Suppose that the larray C has m number of elements in it. If each element of C is of size x bytes, then to store the coach C in memory exactly m.x number of consecutive bytes are required.

2.1.2 Status of a Coach and Tagged Coach (TC) of a Train Stack

The **status** s of a coach is a non-negative integer variable which is equal to the number of ε elements present in it (i.e. in its larray) at this point of time. Therefore, $0 \leq s \leq r$. The significance of the variable s is that it informs us about the exact number of free spaces available in the coach at this point of time. If there is no ε element in the larray of the coach C, then the value of s is 0 at this point of time.
If C is a coach, then the corresponding **tagged coach** (TC) is denoted by the notation (C, s) where s is the status of the coach. This means that C is a coach tagged with an information on the total amount of available free space (here it is termed as ε elements) inside it at this time.
For example, consider the coach C in Figure 1. Clearly its corresponding TC will be denoted by (C, 0) as shown below in Figure 2.

69.37	5.75	0.48	95.26	732.03	0.40	571.86	e

Figure 2. the tagged coach of the coach C in a train of a 7-train stack

Consider the following coach H in a train of a 7-train stack represented as a larray in Figure 3 below.

1037	875	0	ε	3	ε	61	e

Figure 3. a coach of a train in a 7-train stack

Clearly its corresponding TC will be denoted by (H, 2) as shown in Figure 4.

1037	875	0	ε	3	ε	61	2	e

Figure 4. the tagged coach of the coach H

2.1.3 r-Train Stack (or, Train Stack)

A ‘r-train stack’ (or, train stack) is a crisp stack of r-trains containing data of homogeneous data type in the whole stack. For a given r-train stack, the value of the natural number r is fixed throughout. For different r-train stacks, the value of r could be different. The r-train stack is thus a linear data structure in which a r-train may be added or removed only at the end, called the TOP of the train stack in LIFO manner as in classical stack. If TOP = 0 (Null), it implies that the train stack is an empty train stack.

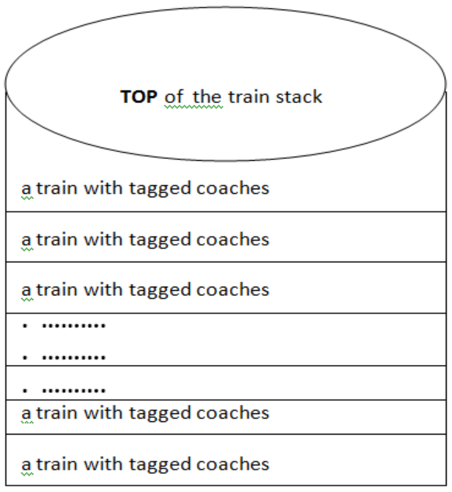
TOP of the train stack
a train with tagged coaches
a train with tagged coaches
a train with tagged coaches
• ....
• ....
• ....
a train with tagged coaches
a train with tagged coaches

Figure 5. a train stack

Thus a train stack T (see Figure 5) may be written as an array of trains as

$$T: \quad T_1, T_2, T_3, \ldots\ldots\ldots, T_n$$

with the implication that the rightmost element is the top element (train T_n). In a train stack, all the trains contain data of identical data type, and so a train stack is to be regarded as a homogeneous data structure.

Insertion and deletion can occur only at the top of the train stack. Two basic operations associated with train stacks are:

(i) **“PUSH”** (to insert a new train into a train stack)
(ii) **“POP”** (to delete a train from a train stack)

The train stack has the following two important operations too, which can be easily implemented using PUSH and POP multiple times, and the properties of the data structure larray:

(iii) **“c - PUSH”** for pushing a coach.
(to insert a tagged coach in a given train of the train stack)

(iv) **“c - POP”** for popping a coach.
(to delete a tagged coach from a given train of the train stack)

2.1.4 rT-Coach Stack (or Coach Stack)

A ‘rt-coach stack’ (or, coach stack in short) is a special case of r-train stack where every train consists of one coach only. For a given rT-coach stack, the value of the natural number r is fixed throughout. For different rT-coach stacks, the value of r could be different. Thus a rT-coach stack is a crisp stack of tagged coaches containing data of homogeneous data type. It is a linear data structure in which a tagged coach may be added or removed only at the end, called the TOP of the train stack in LIFO manner as in classical stack. If TOP = 0 (Null), it implies that the coach stack is an empty coach stack. (The term ‘rT-coach’ signifies that it is a coach of a r-Train).

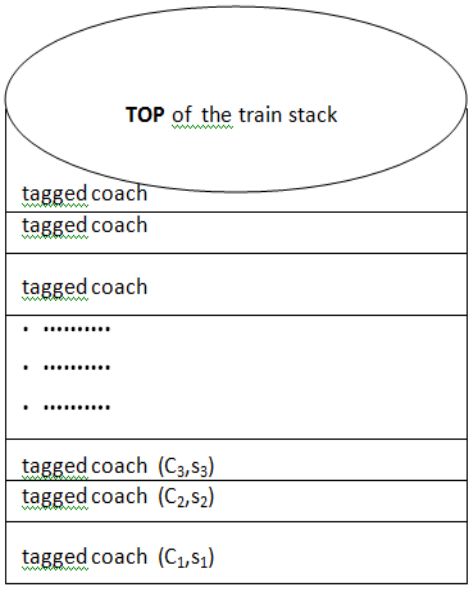

Figure 6. a rT-coach stack

Thus a rT-coach stack T (see Figure 6) may be written as an array of TCs as

$$\mathbf{T}: (C_1, s_1), (C_2, s_2), (C_3, s_3), (C_4, s_4), \ldots\ldots, (C_r, s_r)$$

with the implication that the rightmost element is the top element (tagged coach). In a rT-coach stack all the coaches contain data of identical data type, and so a rT-coach stack is to be regarded as a homogeneous data structure.
Insertion and deletion can occur only at the top of the rT-coach stack. Two basic operations associated with rT-coach stacks are:

(i) **“c - PUSH”** (to insert a new tagged coach into a rT-coach stack)
(ii) **“c - POP”** (to delete a tagged coach from a rT-coach stack)

The rT-coach stack has the following two important operations too, which can be easily implemented using PUSH, POP, c-PUSH, and c-POP operations multiple times, and the properties of the data structure larray:
(iii) **“e - PUSH”** for pushing an element.
(to insert an element into a tagged coach (TC) of a rT-coach stack where status s is not 0, i.e. at least 1).
Insertion of an element (a new passenger) x inside the coach C_i is feasible if x is of same datatype (like other passengers of the coach) and if there is an empty space available inside the coach C_i.
If status of C_j is greater than 0 then data can be stored successfully in the coach, otherwise insertion operation fails here. After each successful insertion, the status s of the coach is to be updated by doing s = s –1.
For insertion of x, we can replace the lowest indexed passenger ε of C_i with x.

(iv) **“e - POP”** for popping an element.
(to delete an element from a tagged coach of a rT-coach stack, retaining the tagged coach in the memory adjusting its status s by doing $s \leftarrow s+1$).
We can delete a data element e_{ij} from the coach C_i. Deletion of a data (passenger) from a coach means replacement of the data by an ε element (of same datatype). Consequently, if $e_{ij}=\varepsilon$, then the question of deletion does not arise. Here it is pre-assumed that e_{ij} is a non-ε member element of the coach C_i. Thus deletion of e_{ij} is done by replacing it by the null element ε, and updating the status s by doing s = s+1. Deletion of a data element (passenger) does not effect the size r of the coach.
For example, consider the tagged coach (C_i, m) where

$$C_i = < e_{i1}, e_{i2}, e_{i3}, e_{i4}, \ldots\ldots, e_{ir} > .$$

If we delete e_{i3} from the coach C_i, then the updated tagged coach will be

$$(C_i, m+1) \quad \text{where } C_i = < e_{i1}, e_{i2}, \varepsilon , e_{i4}, \ldots\ldots\ldots\ldots, e_{ir} >.$$

2.2 Introducing r-Atrain Stack (or Atrain Stack)

Before proceeding to introduce ‘r-atrain stack’, few terminologies needs to be defined:

2.2.1 Code of a Datatype and CD-Table for an Atrain Stack

The concept of CD-Table in atrain stack is analogous to that of the CD-Table introduced in case of r-atrain (atrain) by Biswas [11, 12, 15]. A table is to be created by the user (i.e. by the concerned organization) to fix unique integer code for each datatype which are under use in the organization. This is not an absolute set of codes to be followed universally by every organization, but it is a local document for the concerned organization. For different organizations, this table could be different. But once it is fixed by an organization it should not be altered by this organization, except that addition of new datatypes and corresponding codes may be decided and be incorporated in the table at any stages later retaining the existing records. This table is called **Code of Datatype Table** or **CD-Table** (in short). A sample CD-Table of a hypothetical organization is shown in Table.1 for the sake of understanding. It may be noted that for any organization, for the datatypes character, integer, boolean, etc. the individual space requirement respectively are absolutely fixed. But for a particular organization, for the datatypes String-1, String-2 and String-3 (String type appearing thrice in this case) the space requirement in the above table has been fixed at 10 bytes for one kind, 20 bytes for another and 50 bytes for another kind. Similarly there are four types of file: File - 1, File - 2, File - 3 and File - 4, in the CD- Table and the space requirement for them have been fixed at 100 KB, 1 MB, 10 MB and 25 MB respectively, fixed by choice of the concerned organization (developers).

Table. 1. A hypothetical example of a **CD-Table** of an organization

Sr. No.	**Datatype**	**Space required in bytes (n)**	**Code of Datatype (c)**
1	Character	1	0
2	Integer	2	1
3	Real	4	2
4	String-1	10	3
5	String-2	20	4
6	String-3	50	5
7	File-1	100 KB	6
8	File-2	1 MB	7
9	File-3	10 MB	8
10	File-4	25 MB	9
11	………………	……………	…….
12	………………	……………	…….

2.2.2 Coach of an Atrain Stack

As seen in subsection 2.1.1 earlier, all the coaches in a train stack contain data of homogeneous datatype across the train. In an atrain stack although a coach stores homogeneous data only (not heterogeneous data), but different coaches of an atrain stack store data elements of different datatypes. Thus the data structure atrain stack is a kind of heterogeneous data structure. For constructing a coach for an atrain stack in an organization, we must know in advance the datatype of the data to be stored in it. For this we have to look at the CD-Table of the organization, and reserve space accordingly for r number of data.

Suppose that the larray A has r number of elements in it of a given datatype. If each element of A is of size x bytes (refer to CD-Table) then to store the coach C in memory exactly r.x number of consecutive bytes are required, and accordingly the coach be created by the programmer (concerned organization). In our discussion henceforth, by the phrase "datatype of a coach" we shall always mean the datatype of the data elements of the coach. A coach stores and can store only homogeneous data (i.e. data of identical datatype), but datatype may be different for different coaches in an atrain stack.

By a coach C in an atrain stack we mean a non-empty larray (however, it could be a null larray) containing data of homogeneous data type.
The following two figures shows two coaches C_1 and C_2 of an atrain stack T.

69.37	5.75	0.48	95.26	732.03	0.40	571.86	e

Figure 6. a coach C_1 of a 7-atrain stack T

#	*	@	ε	&	ε	#	e

Figure 7. another coach C_2 of the 7-atrain stack T

2.2.3 Status of a Coach and Tagged Coach (TC) of an Atrain Stack

The **status** s of a coach in an atrain stack is a pair of information (c, n), where c is a non-negative integer variable which is the code of datatype (with reference to the concerned CD-Table) of the data to be stored in this coach and n is a non-negative integer variable which is equal to the number of ε elements present in it (i.e. in its larray) at this point of time. Therefore, $0 \leq n \leq r$. In the status s = (c, n) of a coach, the information c is called the **"code-status"** of the coach and the information n is called the **"availability-status"** of the coach at this time. The significance of the variable n is that it informs us about the exact number of free spaces available in the coach at this point of time. If there is no ε element at this time in the larray of the coach C, then the value of n is 0. Thus, without referring the CD-Table, the status of a coach can not be and should not be fixed.
If C is a coach in an atrain stack, then the corresponding tagged coach (TC) is denoted by the notation [C, s], where s = (c, n) is the status of the coach. This means that C is a coach tagged with the following two information:
(i) one signifying the datatype of the data of the coach.
(ii) the other reflects the total amount of available free spaces (here it is termed as ε elements) inside the coach at this time.

(Thus the concept of TC for atrain stack is slightly different from the concept of TC for train stack defined earlier in section 2.1.2.).

For example, consider Figure 6. and Figure 7. of two coaches C_1 and C_2 respectively of the atrain stack T. Then the following two figures (in Figure 8. and Figure 9.) shows the two tagged coaches C_1 and C_2 of the atrain stack T.

69.37	5.75	0.48	95.26	732.03	0.40	571.86	e, (2,0)

Figure 8. the tagged coach of the coach C_1 in the 7-atrain stack T

#	*	@	ε	&	ε	#	e, (0,2)

Figure 9. the tagged coach of the coach C_2 in the 7-atrain stack T

2.2.4 r-Atrain Stack (or, Atrain Stack)

An ‘atrain stack’ is a crisp stack of r-atrains containing data of heterogeneous data type. Here, each coach is homogeneous internally containing data of identical data type, but different coaches have data of different data types. For a given r-atrain stack, the value of the natural number r is fixed throughout. For different r-atrain stacks, the value of r could be different. The datatype in an atrain stack may vary from coach to coach (unlike in train stacks), but in a coach all data must be homogeneous i.e. of identical datatype. Thus each coach is homogeneous, although the atrain stack is heterogeneous. In this sense, ‘atrain stack’ is regarded as a heterogeneous data structure. (In a homogeneous data structure the data elements considered are all of the same datatype, like in array, linked list, train, etc., but in a heterogeneous data structure data elements are of various datatypes as in atrain).

Analogous to the notion of a train stack, an atrain stack is also a linear data structure in which an atrain may be added or removed only at the end, called the TOP of the atrain stack in LIFO manner as in classical stack (see Figure 10). If TOP = 0 (Null), it implies that the atrain stack is an empty atrain stack.

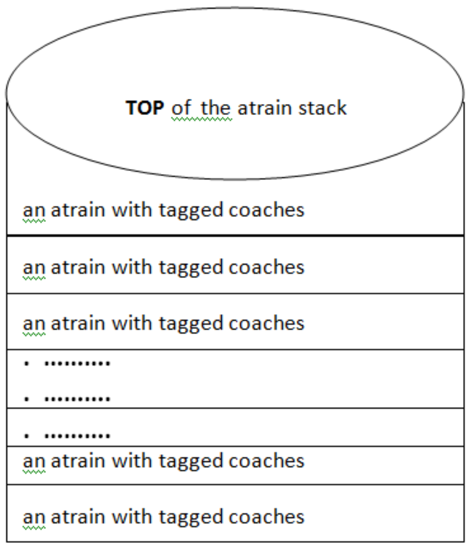
TOP of the atrain stack
an atrain with tagged coaches
an atrain with tagged coaches
an atrain with tagged coaches
an atrain with tagged coaches
an atrain with tagged coaches

Figure 10. an atrain stack

Thus an atrain stack T (see Figure 10) may be written as an array of atrains as

$$T: \quad T_1, T_2, T_3, \ldots\ldots\ldots, T_n$$

with the implication that the rightmost element is the top element (atrain T_n).

Insertion and deletion can occur only at the top of the atrain stack. Two basic operations associated with atrain stacks are:

(i) **"PUSH"** (to insert a new atrain into an atrain stack)
(ii) **"POP"** (to delete an atrain from an atrain stack)

The atrain stack has the following two important operations too, which can be easily implemented using PUSH and POP multiple times, and the properties of the data structure larray :

(iii) **"c - PUSH"** for pushing a coach.
(to insert a tagged coach in a given atrain of the atrain stack)
(iv) **"c - POP"** for popping a coach.
(to delete a tagged coach from a given atrain of the atrain stack)

It is important to note that in this heterogeneous data structure atrain stack, the size of coaches in bytes are not same, although each coach accommodates r number of data elements (including ε which is of same data type as explained in [12, 15]). It depends upon the code from the CD-Table. Therefore, each time we push a new atrain in an atrain stack (assuming that there is no overflow), the TOP gets updated accordingly in a dynamic way. Similarly, each time we POP an atrain from an atrain stack, the TOP gets updated accordingly.

2.2.5 rA-Coach Stack (or Coach Stack)

A 'rA-coach stack' (or, coach stack in short) is a special case of r-atrain stack where every atrain consists of one coach only. For a given rA-coach stack, the value of the natural number r is fixed throughout. For different rA-coach stacks, the value of r could be different. Thus a rA-coach stack is a crisp stack of tagged coaches (of atrain) containing data of heterogeneous data type. It is a linear data structure in which a tagged coach may be added or removed only at the end, called the TOP of the rA-coach stack in LIFO manner as in classical stack. If TOP = 0 (Null), it implies that the rA-coach stack is an empty rA-coach stack. (The term 'rA-coach' signifies that it is a coach of a r-Atrain).

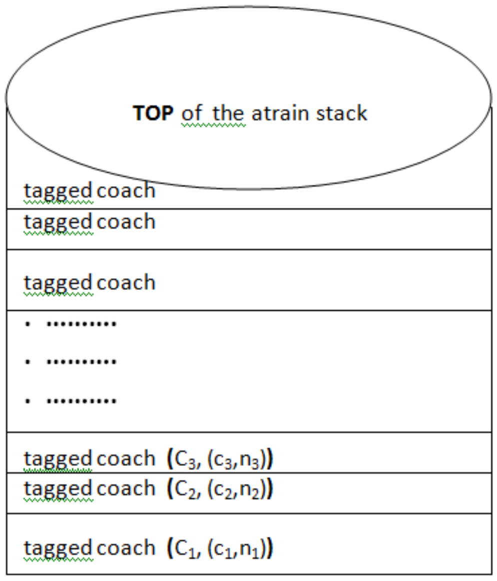

Figure 11. a rA-coach stack

Thus a rA-coach stack T(see Figure 11) may be written as an array of TCs as

$$T: \quad (C_1,s_1), \quad (C_2,s_2), \quad (C_3,s_3), \quad (C_4,s_4), \ldots\ldots\ldots,(C_r,s_r)$$

i.e. as $T: (C_1,(c_1,s_1)), (C_2,(c_2,s_2)), (C_3,(c_3,s_3)), (C_4,(c_4,s_4)), \ldots\ldots\ldots, (C_r,(c_r,s_r))$, with the implication that the rightmost element is the top element (TC).

Insertion and deletion can occur only at the top of the rA-coach stack. Two basic operations associated with rA-coach stacks are:

(i) **"c - PUSH"** (to insert a new tagged coach into a rA-coach stack)

(ii) **"c - POP"** (to delete a tagged coach from a rA-coach stack)

The rA-coach stack has the following two important operations too, which can be easily implemented using PUSH, POP, c-PUSH, and c-POP operations multiple times, and the properties of the data structure larray:

(iii) **"e - PUSH"** for pushing an element.
(to insert an element into a tagged coach (TC) of a rA-coach stack where status s is not 0, i.e. at least 1).
Insertion of an element (a new passenger) x inside the coach C_i is feasible if x is of same datatype (like other passengers of the coach) and if there is an empty space available inside the coach C_i.
If status of C_j is greater than 0 then data can be stored successfully in the coach, otherwise insertion operation fails here. After each successful insertion, the status s of the coach is to be updated by doing s = s –1.
For insertion of x, we can replace the lowest indexed passenger ε of C_i with x.

(iv) **"e - POP"** for popping an element.
(to delete an element from a tagged coach of a rA-coach stack, retaining the tagged coach in the memory adjusting its status s by doing $s \leftarrow s+1$).
We can delete a data element e_{ij} from the coach C_i. Deletion of a data (passenger) from a coach means replacement of the data by an ε element (of same datatype). Consequently, if $e_{ij} = \varepsilon$, then the question of deletion does not arise. Here it is pre-assumed that e_{ij} is a non-ε member element of the coach C_i in the atrain stack.
For j = 1, 2, ..., r, deletion of e_{ij} is done by replacing it by the null element ε, and updating the availability-status n by doing n = n+1. Deletion of a data element (passenger) does not effect the size r of the coach.
For example, consider the tagged coach [C_i, (c_i,m)] where

$$C_i = < e_{i1}, e_{i2}, e_{i3}, e_{i4}, \ldots\ldots\ldots, e_{ir} > .$$

If we delete e_{i3} from the coach C_i, then the updated tagged coach will be [C_i, (c_i, m+1)] where $C_i = < e_{i1}, e_{i2}, \varepsilon, e_{i4}, \ldots\ldots\ldots, e_{ir} >$.

It can be noted that the data structure rT-coach stack is a special case of the data structure r-train stack, the data structure rA-coach stack is a special case of the data structure r-atrain stack. The classical stack (in Computer Science) is a '1A-coach stack', special case of the data structures rA-coach stack for r = 1.

3 r-Train Queue and r-Atrain Queue

In this section we introduce two new data structures for big data:

(1) 'r-train queue' or 'train queue' in short, and
(2) 'r-atrain queue' or 'atrain queue' in short.

The concept of the terminologies: coach, status and tag in a train queue are similar to those of a coach in a train stack as discussed in subsection 2.1.1. The concept of the terminologies: coach, status, tag, CD-table in an atrain queue are similar to those of a coach in an atrain stack as discussed earlier in the subsections 2.2.2 and 2.2.3.

A ‘r-train queue’ (or, train queue) is a crisp queue of r-trains containing data of homogeneous data type across the whole queue. It is a linear data structure in which deletion of a train can take place only at one end called the ‘front’ and insertion of a train can take place only at the other one end called the ‘rear’, in the manner of FIFO list, as shown in Figure 12 below.

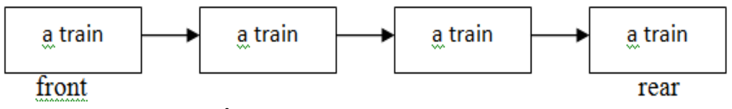

Figure 12. a train queue

In the larray of a train queue, there are two pointer variables: FRONT and REAR. The FRONT contains the location of the front TC and the REAR contains the location of the rear TC. If FRONT = Null, then the train queue is empty. Whenever a TC is deleted from the train queue, the value of FRONT gets incremented by 1. Similarly, whenever a TC is added to the train queue, the value of REAR gets incremented by 1.

A ‘rT-coach queue’ is a crisp queue of tagged coaches (of trains) containing data of homogeneous data type. It is a linear data structure in which deletion of tagged coach can take place only at one end called the ‘front’ and insertion of a tagged coach can take place only at the other one end called the ‘rear’, in the manner of FIFO list, as shown in Figure 13 below.

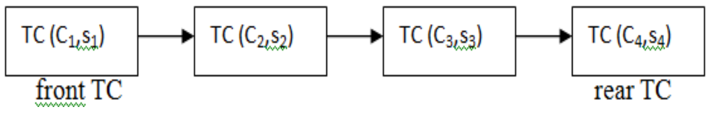

Figure 13. a rT-coach queue

A ‘r-atrain queue’ (or, atrain queue) is a crisp queue of r-atrains containing data of heterogeneous data type, where each coach is homogeneous in itself. Datatypes of different coaches are different. Thus it is a linear data structure in which deletion of an atrain can take place only at one end called the ‘front’ and insertion of an atrain can take place only at the other one end called the ‘rear’, in the manner of FIFO list, as shown in Figure 14 below.

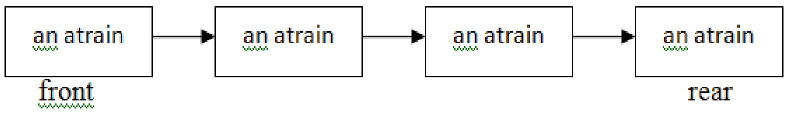

Figure 14. an atrain queue

The datatype in an atrain queue may vary from coach to coach (unlike in train queue), but in a coach all data must be homogeneous i.e. of identical datatype. Thus each coach is homogeneous, although the atrain queue is heterogeneous. In this sense, ‘atrain queue’ is regarded as a heterogeneous data structure.

A ‘rA-coach queue’ is a crisp queue of tagged coaches (as defined in sections 2.2.2 and 2.2.3) containing data of heterogeneous data type (see Figure 15).

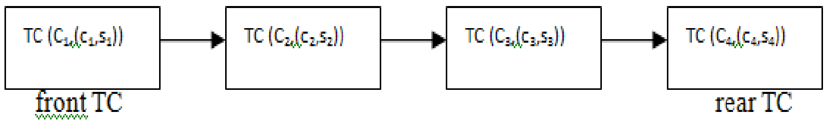

Figure 15. a rA-coach queue

In an atrain queue or rA-coach queue, different TCs are of different sizes depending upon the CD-table concerned.

It can be noted that the data structure rT-coach queue is a special case of the data structure r-train queue, the data structure rA-coach queue is a special case of the data structure r-atrain queue. The classical queue is a special case of the data structures r-train queue and r-atrain queue.

4 r-Train Binary Tree and r-Atrain Binary Tree

In this section we introduce two new non-linear data structures for big data:

(1) ‘r-train binary tree’ or ‘train binary tree’ in short, and
(2) ‘r-atrain binary tree’ or ‘atrain binary tree’ in short.

They are basically extension of the notion of classical binary tree, and are defined in a similar way in the context of train-atrain architecture for big data.
The concept of the terminologies: coach, status and tag in a train binary tree or train tree are similar to those of a coach in a train stack as discussed in subsection 2.1.1. The concept of the terminologies: coach, status, tag, CD-table in an atrain binary tree or atrain tree are similar to those of a coach in an atrain stack as discussed earlier in the subsections 2.2.2 and 2.2.3.

A r-train binary tree (or train binary tree, in short) T is defined as a finite set of r-trains called *train-nodes* such that:

(a) T is empty (called the *null tree* or *empty tree*), or
(b) T contains a distinguished train-node R called the *root-train* of T, and the remaining train-nodes of T form an ordered pair of disjoint train binary trees T_1 and T_2.

If T does contain a root-train R, then the two train binary trees T_1 and T_2 are called respectively *left train subtree* and *right train subtree* of the train binary tree R. If T_1 is non-empty then its root-train is called the left successor of R and if T_2 is non-empty then its root-train is called the right successor of R (see Figure 16). The concept of left child train, right child train, parent train, sibling trains, etc. can also be visualized in a classical way.

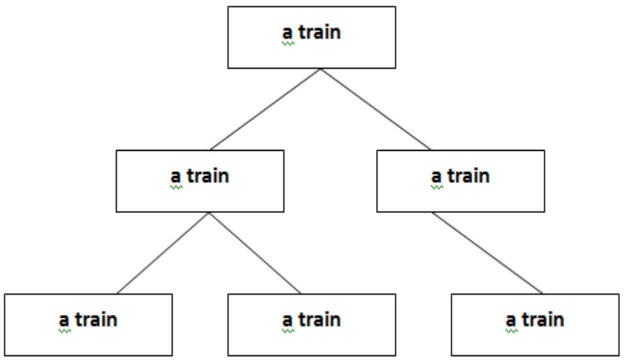

Figure 16. a train binary tree

A rT-coach binary tree T is defined as a finite set of tagged train coaches called *coach-nodes* such that:

(a) T is empty (called the *null tree* or *empty tree*), or
(b) T contains a distinguished coach-node R called the *root-coach* of T, and the remaining coach-nodes of T form an ordered pair of disjoint rT-coach binary trees T_1 and T_2 (see Figure 17).

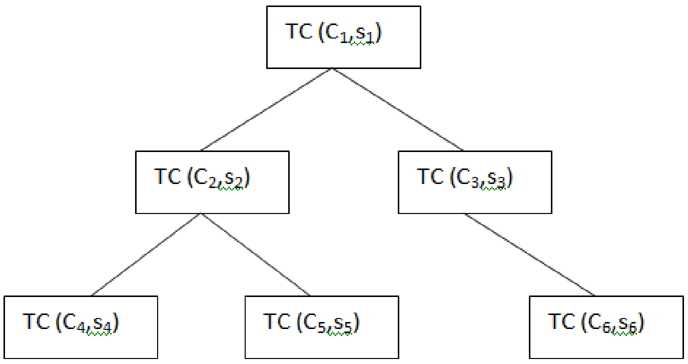

Figure 17. a rT-coach binary tree

A r-atrain binary tree (or, atrain binary tree, in short) can also be defined in an analogous way (see Figure 18) with tagged coaches as defined in section 2.2.3.

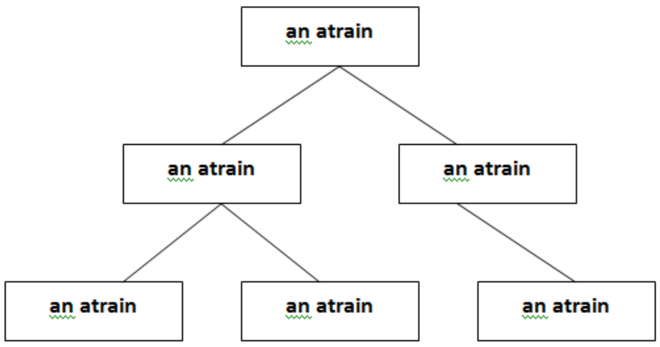

Figure 18. an atrain binary tree

A rA-coach binary tree T is defined as a finite set of tagged atrain coaches called *coach-nodes* such that:

(a) T is empty (called the *null tree* or *empty tree*), or
(b) T contains a distinguished coach-node R called the *root-coach* of T, and the remaining coach-nodes of T form an ordered pair of disjoint rA-coach binary trees T_1 and T_2 (see Figure 19).

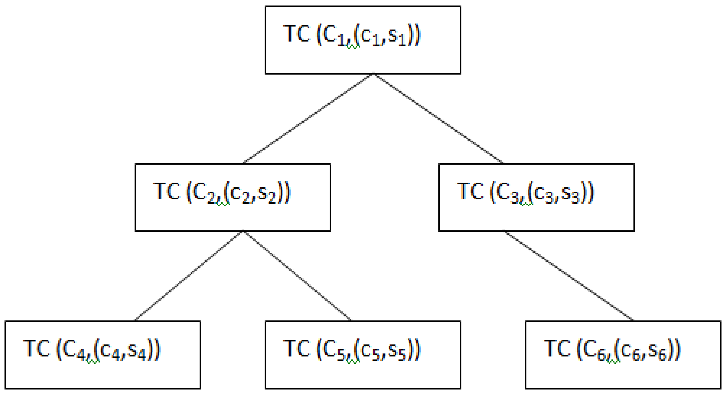

Figure 19. a rA-coach binary tree

A r-train tree (or train tree in short) T is defined as a non-empty finite set of r-trains called *train-nodes* such that:
(a) T contains a distinguished train-node R called the *root-train* of T, and
(b) the remainingtrain-nodes of T form an ordered collection of zero or more number of disjoint train trees T_1, T_2, ….., T_m.

The train trees T_1, T_2, …..,T_m are called train subtrees of the *root-train* R, and the root-trains of T_1, T_2, ….., T_m are called successors of R. The concept of a children trains, parent train, sibling trains, etc. can be visualized in a classical way.

A r-atrain tree (or atrain tree, in short), rT-coach tree and rA-coach tree can be defined in an analogous way. The data structures r-train, r-atrain, r-train tree, and r-atrain tree should not be confused with any of the data structures (basically for multidimensional data) of the group : X-tree, R-tree, R+-trees, R*-trees, M-tree, KD-tree, PQ-tree, SPQR-tree, Hilbert R-tree, etc. etc.

It may be noted that the data structure rT-coach binary tree is a special case of the data structure r-train binary tree, the data structure rA-coach binary tree is a special case of the data structure r-atrain binary tree. The classical binary tree is a special case of the data structures r-train binary tree and r-atrain binary tree. Similarly the classical tree is a special case of the data structures r-train tree and r-atrain tree. The ADS (unitier or multitier) with the new type of network topologies 'Multi-horse Cart Topology' and 'Cycle Topology' defined by Biswas in [15] is the appropriate distributed system to implement atrain tree for big data of any 4Vs of any momentum as the ADS is scalable upto any desired extent both in breadth and depth.

5 Conclusion

Today's supercomputer or multiprocessor system which can provide huge parallelism has become the dominant computing platform (through the proliferation of multi-core processors). In most of the giant business organizations, the system has to deal with Big Data of homogeneous or heterogeneous nature for which the data structures of the existing literature can not always lead to the desired optimal satisfaction. The existing data structures of Computer Science are neither appropriate nor sufficient to deal with big data. The very common and frequent operations like Insertion, deletion, searching, etc. are required to be much faster even if the Big Data be of heterogeneous types. Such situations require some way or some method which work more efficiently than the simple rudimentary data structures such as arrays, linked lists, hash tables, binary search trees, dictionary, etc. Obviously, there is a need of a dash of creativity of a new or better performed heterogeneous data structure which at the same time must be of rudimentary in nature. The data structure 'r-train' (or 'train' in short) is a homogeneous data structure can deal with homogeneous Big Data very efficiently. The data structure **'r-atrain'** (or, atrain) is a robust kind of dynamic heterogeneous data structure which encapsulates the merits of the arrays and of the linked lists, but are different from them and can deal with heterogeneous Big Data. The strong merit of the atrain distributed system (**ADS**) introduced in [15] is that it is an infinitely scalable architecture (along breadth and depth both) of distributed system for processing big data of any momentum of 4Vs. It is so constructed that it has 100% compatibility with the data structure atrain (and train) exclusively discovered for big data. The data structures train tree and atrain tree can be well implemented in atrain distributed systems for big data. Although big data is explosively expanding with 4Vs, the subject has not been growing in the same pace. One of the reasons is that the existing data structures are not the appropriate data structures for big data. The first attempt to introduce a data structure for big data is the discovery of the data structures atrain (and train) in [11, 12, 15], with the distributed system ADS of new type of network topologies 'Multi-horse Cart Topology' and 'Cycle Topology'. The **'Data Structures for Big Data'** is to be regarded as a new subject, not just as a new topic in Big Data Science, considering the explosive momentum of the big data. Consequently, in this paper we introduce few more new data structures for big data which are : homogeneous stacks 'train stack' and 'rT-coach stack', heterogeneous stacks 'atrain stack' and 'rA-coach stack', homogeneous queues 'train queue' and 'rT-coach queue', heterogeneous queues 'atrain queue' and 'rA-coach queue', homogeneous binary trees 'train binary tree' and 'rT-coach binary tree', heterogeneous binary trees 'atrain binary tree' and 'rA-coach binary tree', homogeneous trees 'train tree' and 'rT-coach tree', heterogeneous trees 'atrain tree' and 'rA-coach tree', to enrich the new subject 'Data Structures for Big Data' with the extended notion of the architecture of the data structures r-atrain (atrain) and r-train (train) of Biswas [11, 12, 15] for big data. The natural number r is fixed for a stack / queue / binary tree or tree, but could be different for different stacks / queues / binary trees or trees. All these big data structures can be well implemented in ADS.

The "Big Data Universe" needs intensive attention not only just for finding solutions to the present problem of explosive expansion, but also for the tremendous future expansion. It is fact [15] that the present 4Ns are lagging behind the present 4Vs. Everyday's big data is bigger than the previous day's big data in the big data universe. The data structure atrain for big data and the distributed system ADS for big data can deal with any amount of 4Vs because of its scalability property upto any desired amount both in breadth and depth. However, for the future big data of the 'Big Data Universe', the train/atrain data structures can be further extended embedded with the families of data structures proposed in this paper to create new big data structures viz. m-train of r-trains, m-atrain of r-atrains, m-train of r-train stacks, m-atrain of r-atrain stacks, m-train of r-train queues, m-atrain of r-atrain queues, m-train of r-train binary trees, m-atrain of r-atrain binary trees, etc. depending upon the 4Vs of future big data. Consequently, the corresponding distributed system **EADS** ('Extended ADS') of several designs can be created by developing new methods on how to integrate multiple ADSs to make an EADS, designing new type of Hybrid Topologies of 'Multi-horse Cart Topology', 'Cycle Topology' and other existing classical topologies.

Our next work will also be to study these big data structures with sufficient examples, to introduce the data structures ‘train graph’ and ‘atrain graph’ considering the big data in the topologies of communication network, in particular while a large number of communication networks logically merge together to form a singly network and to work efficiently as a single network as it seems to the laymen users. An important problem to the computer scientists is to develop a new but simple big data language called by “**ADSL**’" (Atrain Distributed System Language) which can easily be used by any laymen user at the PC of an ADS to download unlimited amount of relevant big data of any heterogeneous datatype (if not of homogeneous datatype) from the cyberspace, to upload unlimited amount of big data of any heterogeneous datatype (or homogeneous datatype) in the cyberspace, to store the downloaded big data online in the coaches of PC/DCs of ADS in an organized way of atrain (train) architecture, to answer any query be it arithmetical or statistical or relational query or imprecise query on big data.

References

[1] Andrew S. Tanenbaum, Computer Networks (3rd Edition), Pearson Education India, 1993.

[2] Ashu M. G. Solo, Multidimensional Matrix Mathematics: Part 1-6, Proceedings of the World Congress on Engineering 2010 Vol III, WCE 2010, June 30 - July 2, 2010, London, ISBN: 978-988-18210-8-9, ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online).

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

[4] Christian Krattenthaler and Michael Schlosser, A New Multidimensional Matrix Inverse with Applications to Multiple q-series, Discrete Mathematics, Volume 204, Issues 1 - 3, 6 June 1999, Pages 249 - 279.
http://dx.doi.org/10.1016/s0012-365x(98)00374-4

[5] David Feinleib, “Big Data Demystified: How Big Data Is Changing The Way We Live, Love And Learn”, The Big Data Group Publisher, LLC San Francisco, USA, 2013.

[6] Erik Thomsen, OLAP Solutions: Building Multidimensional Information Systems, 2nd Edition. John Wiley & Sons. USA, 2002.

[7] Jeffrey Needham, Disruptive Possibilities: How Big Data Changes Everything, O’reilly Publisher, Cambridge, 2013.

[8] Joel L Franklin, Matrix Theory, Mineola, N.Y., 2000.

[9] Jules J Berman, Principles of Big Data: Preparing, Sharing, and Analyzing Complex Information, Morgan Kaufmann (Elsevier) Publisher, USA, 2013.

[10] Phil Simon, Too Big to Ignore: The Business Case for Big Data, John Wiley & Sons, New Jersey, 2013.

[11] Ranjit Biswas, Heterogeneous Data Structure "R-Atrain", INFORMATION: An International Journal (Japan), Vol.15 (2) February'2012, pp 879-902 (©2012 International Information Institute of Japan & USA).

[12] Ranjit Biswas, Heterogeneous Data Structure "r-Atrain", Chapter-12 in "Global Trends in Knowledge Representation and Computational Intelligence" by B. K. Tripathy and D. P. Acharjya, IGI Global, USA, 2013. (DOI: 10.4018/978–1–4666–4936–1, ISBN13: 9781466649361,
ISBN10: 1466649364, EISBN13: 9781466649378).

[13] Ranjit Biswas, Region Algebra, Theory of Objects & Theory of Numbers, International Journal of Algebra, Vol. 6 (8), 2012, 1371 – 1417.

[14] Ranjit Biswas, Theory of Solid Matrices & Solid Latrices, Introducing New Data Structures MA, MT: for Big Data, International Journal of Algebra, Vol.7 (16) (2013) pp 767 – 789, http://dx.doi.org/10.12988/ija.2013.31093.

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

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

[17] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, Introduction to Algorithms, Second Edition, MIT Press and McGraw-Hill, 2001.

[18] Viktor Mayer - Schönberger and Kenneth Cukier, BIG DATA: A Revolution That Will Transform How We Live, Work, and Think, Eamon Dolan/Houghton Mifflin Harcourt Publisher, 2013.

Received: August 15, 2014; Published: November 12, 2014