International Journal of Advances in Computer Science & Its Applications
Volume 6 : Issue 2 *[ISSN 2250-3765]*
Publication Date : 31 August, 2016

AVL and Red-Black tree as a single balanced tree

Lynda Bounif, Djamel Eddine Zegour

***Abstract* – Since the invention of AVL trees in 1962 and Red-black trees in 1978, researchers were divided in two separated communities, AVL supporters and Red-Black ones. Indeed, AVL trees are commonly used for retrieval applications whereas Red Black trees are used in updates operations, so, the choice of a structure must be done firstly even if the operations are not known to be searches or updates. That is the main reason why we propose a common tree with the same complexity and memory space, representing both an AVL and a Red-Black tree, this new tree allows to gather together the two communities on one hand, and to expand the scope of AVL and Red-Black tree applications on the other hand.**

***Keywords* - Balanced trees, AVL, Red-Black, Partitioning.**

I. Introduction

Binary search trees are very efficient for many applications in computer science but have poor worst-case performance [1].We can make the case that when a tree is perfect balanced it takes a long time to traverse down it because it has height at most Log (n) [2], unfortunately keeping a perfect balance of a tree is rude and so expensive in practice, that's why balanced trees are introduced, where costs are guaranteed to be logarithmic while ensuring that the tree remains almost balanced, examples of these trees are: AVL trees and Red-Black trees.

Both AVL trees and Red-Black trees are very popular data structures. They are the most used self-balancing binary search trees. Indeed, they are implemented and integrated in many programming languages like JAVA and C++. They are also used generally to implement dictionaries and associative arrays.

After the invention of AVL trees, other propositions are made in order to improve performance or to give simplify algorithms: Foster gives complements studies expanded to the original AVL tree. [3], the second improved AVL tree is the generalization of AVL trees which allows unbalances up to a small integer [4].Another main structure is a one-sided height-balanced tree (OSHB) which makes a restriction on heights of node's sons so that the right son never has smaller height than the left one, insertion and deletion algorithms are in O ($\log^2 n$) time [5] [6], after that optimum algorithms for OSHB tree are proposed in O (Log n) time with more complicated algorithms than the original AVL trees [7].

The original data structure of Red-Black trees is invented in 1972 by Rudolf Bayer [8] under the name: "Symmetric Binary B-trees", a few years after, a new form of the original structure is proposed [9] where tree balance is expressed using red and black colors, this structure is difficult to understand and implement, therefore many works appeared in order to simplify the algorithms, AA tree is a powerful simplification of Red-Black trees with the same performance and much more simplicity and simple coding [10], moreover several simple implementations of Red-Black trees can be found in [11] [12].

Recently the majority of works in terms of AVL and Red-Black trees aims basically to simplify rather than anything else, [13] introduces a new simpler insertion and deletion algorithms for AVL tree by using virtual nodes and a brief study of AVL trees using this concept is presented in [14], then [15] gives a new algorithms and shows how easily maintain the balance factor after an updating operation. When it comes to Red-Black trees a revisited version has been proposed [16] where the code is considerably reduced compared to the implementation proposed in [17] where a complete C# implementation of Red-Black trees is presented including all the implementation details for insertion and deletion.

In 2015 rank balanced trees [18] is presented as a proper subset of Red-Black tree similar to AVL tree with best performances with eight rank rules, but in addition to the obligation to satisfy loads of inequalities corresponding to the number of insertions and deletion, this structure isn't implemented.

Even though AVL trees are old, they are still strong and efficient for many applications, In 1985 [19] presents a new implementation of AVL tree without changing it and demonstrate the power of the basic AVL tree, also in [18] the authors have said that the design and analysis of balanced trees is still a rich area, not yet fully explored and "AVL tree is anything but passé".

AVL tree is used recently in many applications; it is used in cloud computing environment for task scheduling [20]. [21] Proposes an algorithm for indexing the keyword extracted from the web documents along with their context based on AVL tree. [22] Used them in Wireless sensor network to provide a security protocol. Also [23] has used this structure in data mining classification for a decision tree induction. In the other hand because of the hardness to understand Red Black tree it is not used us the AVL tree since both the structures are O (log n). It is used in a real time Data Addressing of Control System which reduces the time of the consumption of searching and updating operations taking place frequently and overcome the problem of synchronization cycle. [24]

In this paper, section 2 introduces the famous existing balanced trees AVL and Red-Black trees, and then gives a comparison between them. Section 3 will present our new data structure. Section 4 shows the benefit of our data structure, and finally section 5 concludes our work.

Lynda Bounif, Djamel Eddine Zegour
LCSI Laboratory, National School of computer science ESI (ex INI) Algeria.

II. Review of AVL and Red-Black trees

A. AVL trees

It is the first balanced binary data structure. It's named after its two inventors, G.M. Adelson-Velskii and E.M. Landis [2]. In an AVL tree, the heights of the two child sub-trees of any node differ by at most one, so each node alone in the tree represents an AVL tree, A balance factor is then added in each node in order to maintain the balance of the tree, it can takes only the values: 0, -1 or +1, so the balancing actions have to be done when the balance factor becomes 2 or -2 [25].

Figure 1 gives an example of AVL tree.

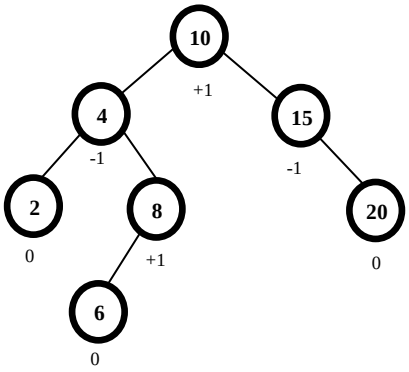


Figure 1. Example of an AVL tree

B. Red-Black trees

A red-black tree is a binary search tree with one extra bit of storage per node: its color, which can be either Red or Black, it is binary representation of a 2-3 tree.

In a Red-Black tree, any path contains the same number of black nodes and does not contain two consecutive red nodes. A color field is then added in each node as is shown in figure 2.

A binary search tree is a red-black tree if it satisfies the following red-black properties [26]:

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

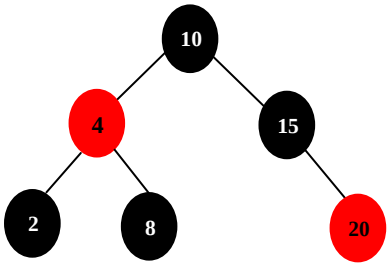


Figure 2. Example of a Red-Black tree

C. Comparison between AVL and Red-Black trees

Several comparison studies of balanced trees especially for AVL and Red-Black tree are available; the most important one is [27], it takes in considerations these two factors: the height of the tree per operation and the number of rotations, and varies the rate of search, insertion and deletion operation, the results of these studies, shows that AVL tree is more efficient that Red-Black tree, especially in lookup-intensive applications, because AVL tree is more balanced than Red-Black trees, It has a height bounded by 1.44 Log (n) where n is the number of nodes in the tree and log the base-two logarithm.

AVL tree is more balanced than the Red-Black tree since it maximum's height 1.44 Log (n) where in Red-Black tree it is 2 Log (n+1). Consequently, AVL trees perform well in lookup-intensive applications.

Both AVL trees and Red-Black trees perform at most one restructuring in insert operation. Restructuring consists in one or two rotations. AVL trees perform at most Log (n) restructurings in delete operation while Red-Black trees perform at most two restructurings. Consequently, Red-Black trees perform better than AVL trees for update-intensive applications.

Here after a table witch gives the important differences between AVL and Red-Black tree

TABLE I. Comparison between AVL and Red-Black trees

Worst case	AVL	Red-Black
Height	1.44 Log (n)	2 log (n+1)
Updates complexity	O(Log (n))	O(Log (n))
Retrieval Complexity	O(Log (n))	O(Log (n))
Rotations for insert	2	2
Rotations for delete	Log (n)	3

III. The new data structure

A. Description

The new data structure is a binary search tree partitioned in classes. Each class is in fact a sub tree holding an AVL tree of height h or h-1. The root node of this sub tree is a class node; the other nodes are simple. Furthermore, the new structure is perfectly balanced considering only class nodes.

Beside the data field, a node contains a byte to designate both its kind and its height. The height of a node is in fact the depth of the sub-tree rooted at this node inside the class it belongs.

Formally, the new structure should respect theses four rules:

1. Every node must be either a simple node or a class node.
2. Every class must have a height equals to h-1 or h-2.
3. Every direct path from any node to a leaf must contains the same number of class nodes.

Figure 3 shows an example of our new data structure when h = 3, we can see that the height of every class is 2 or 1; moreover every direct path from the root to a leaf has exactly two classes.

It is straightforward to observe that for h = ∞ the new structure generates an AVL Tree. Indeed, there is only one class node which is the root of the class. All the others are simple nodes. In practice, there is no limit for the height of the unique class and each path from any node to a leaf contains the same number of class nodes (0 or 1).

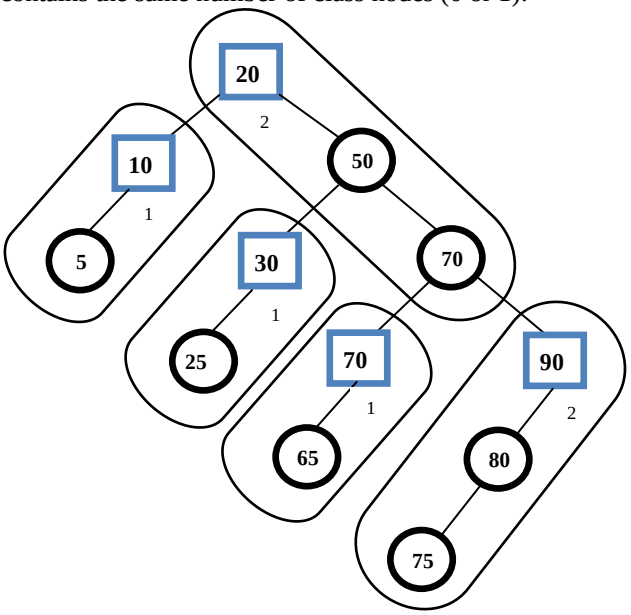


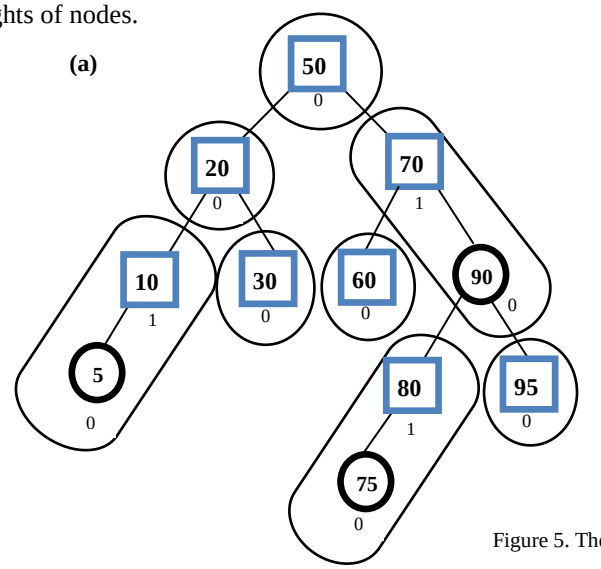
Figure 3. Representation of the new struture

It is also pretty straightforward to notice that for h = 2 the new structure generates a data structure equivalent to a Red-Black tree. Indeed, in the definition above, by replacing simple nodes by red nodes and class nodes by black nodes, we obtain exactly the definition of a Red-Black tree.

When the height of a class is equal to 0, this means the black node has not a red child. When the height of a class is equal to 1, this means that the black node has one or two red children. All the simple nodes have 0 as height. Classes define mathematically a partition on the tree. In other words, the intersection of any two classes is empty and the union of all the classes gives all the items in the tree.

In order to simplify the presentation in the figures below, class nodes are represented inside squares and simple nodes inside circles. Furthermore, classes are surrounded.

Figure 4(a) shows the new structure as a structure equivalent to an AVL tree. There is only one class containing an AVL tree. Values under the nodes designate heights of nodes.



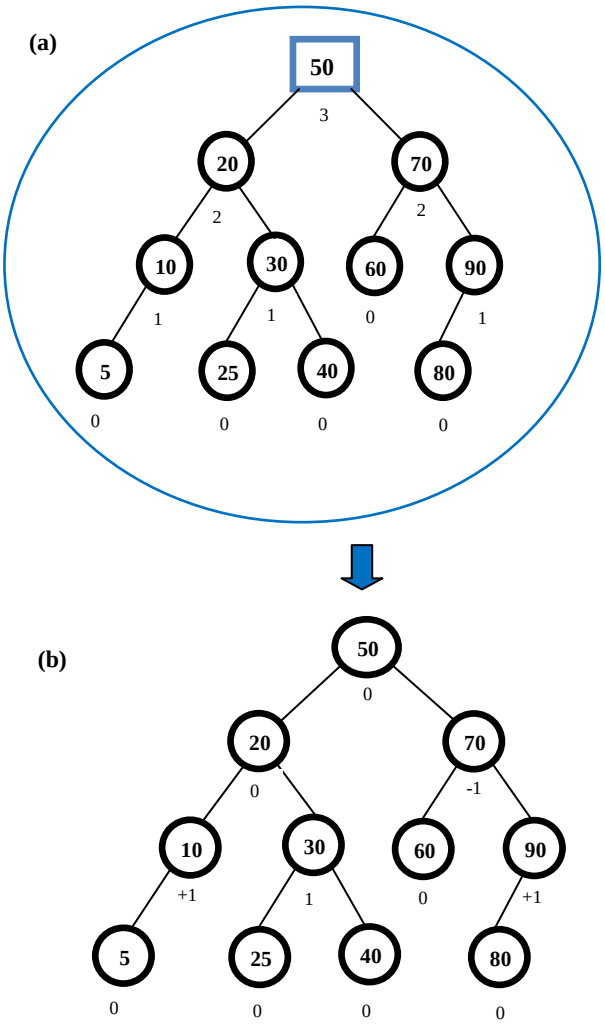


Figure 4. The new structure as an AVL tree

Figure 4(b) shows the corresponding AVL tree. Values under nodes designate balances.

Figure 5(a) shows the new structure as a structure equivalent to a Red-Black tree. 5(b) shows the corresponding Red-Black tree. Class nodes stand for black nodes and simple nodes for red ones. Indeed, every class contains an AVL tree of height 0 or 1.

From bottom to the root, we have the following classes {80, 75}, {95}, {10, 5}, {30} {60}, {70, 90}, {20}, {50}.

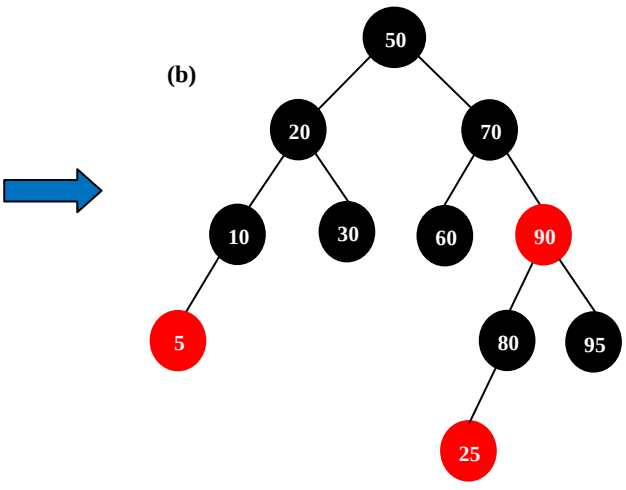


Figure 5. The new structure as a Red-Black tree

B. Implementation

The proposed tree is a balanced binary search tree, characterized by a byte of storage which contains both the kind and the height of a node; this byte represents the color in a Red-Black tree and the balance in an AVL tree. Thus, the first bit defines its kind (simple or class); the seven others bits define its height.

We can notice that when the new structure generates an AVL tree, there is a unique class that can contain a binary search tree of height 2^7, i.e. a class with 2^{128} elements.

IV. Benefit of the new structure

Our new structure can be applied in many real applications where AVL or Red-Black trees are used since it is equivalent to both of them, it can be a very efficient structure for real time systems, in this context the authors of [28] demonstrates the usefulness of using both AVL and Red-Black trees in the priority queue for Dynamic Data-Driven Application Systems: when the system anticipates intensive search operations the system will convert the tree to AVL, when the system anticipates intensive updates operations it convert the tree to Red-black.

V. Conclusion

Balanced binary search trees are pertinent data structure for many applications in computer science. In this paper we presented a unique balanced tree for both AVL and Red-Black trees. A single parameter allows switching from one structure to another. This is accomplished through a binary search tree partitioned into classes that are in fact AVL sub-trees. When the height of the class is infinite, it is an AVL tree. When the height is 0 or 1, it is a structure equivalent to a Red-Black tree. The presented tree is simple and clear to understand and implement which allows expanding applications.

Acknowledgment

A major expression of thanks and gratitude must go to my supervisor Prof. Djamel Eddine Zegour for the patient guidance, encouragement and pertinent advices.

References

[1] Robert. Sedgewick and Addison. Wesley. “Algorithms in Java, Parts 1-4. Professional. 768 pages”. 23 juil. 2002.

[2] AdelsonVelskii, M., and Evgenii Mikhailovich Landis. “An algorithm for the organization of information”. Dokl. Akad. Nauk SSSR 146, pp. 263-266. English translation in Soviet Math. Dokl. 3, pp. 1259-1262, 1962.

[3] Foster, Caxton C. "Information retrieval: information storage and retrieval using AVL trees." *Proceedings of the 1965 20th national conference*. ACM, 1965.

[4] Foster, Caxton C. "A generalization of AVL trees." *Communications of the ACM* 16.8 (1973): 513-517.

[5] Hirschberg, Daniel S. "An insertion technique for one-sided height-balanced trees." *Communications of the ACM* 19.8 (1976): 471-473.

[6] Kosaraju, S. Rao. "Insertions and deletions in one-sided height-balanced trees." *Communications of the ACM* 21.3 (1978): 226-227.

[7] Räihä, Kari-Jouko, and Stuart H. Zweben. "An optimal insertion algorithm for one-sided height-balanced binary search trees." *Communications of the ACM*22.9 (1979): 508-512.

[8] Bayer, R [1972] “Symmetric Binary B-Trees”: Data structure and maintenance algorithms. Acta Informatica 1(4):290-306, 1972.

[9] Guibas, Leo J., and Robert Sedgewick. "A dichromatic framework for balanced trees." *19th Annual Symposium on Foundations of Computer Science*. IEEE, 1978.

[10] Andersson, Arne. "Balanced search trees made simple." *Algorithms and Data Structures*. Springer Berlin Heidelberg, 1993. 60-71.

[11] Okasaki, Chris. *Purely functional data structures*. Cambridge University Press, 1999.

[12] Kahrs, Stefan. "Red-black trees with types." *Journal of functional programming*11.04 (2001): 425-432.

[13] Tripathi, Rajeev R. Kumar. "Balancing of AVL tree using virtual node." *RN* 10 (2010): 20.

[14] Chauhan, Shivani, et al. "A brief study of balancing of AVL tree." *International Journal of Research* 1.11 (2014): 406-408.

[15] Goutam Mondal .“A New Way of Inserting and Deleting the Node To and From the AVL search tree” International Journal of Advance Research in Computer Science and Management Studies”,2014: 191-194.

[16] Sedgewick, Robert. "Left-leaning red-black trees." *Dagstuhl Workshop on Data Structures*. 2008.

[17] Wiener, Richard. "Generic Red-Black Tree and its C# Implementation." *Journal of Object Technology* 4.2 (2005): 59-80.

[18] Haeupler, Bernhard, Siddhartha Sen, and Robert E. Tarjan. "Rank-balanced trees." *ACM Transactions on Algorithms (TALG)* 11.4 (2015): 30.

[19] Tsakalidis, Athanasios K. "AVL-trees for localized search." *Information and Control* 67.1 (1985): 173-194.

[20] Chiu, Chuan-Feng, et al. "Task Scheduling Based on Load Approximation in Cloud Computing Environment." *Future Information Technology*. Springer Berlin Heidelberg, 2014. 803-808.

[21] Tyagi, Nidhi, Rahul Rishi, and R. P. Aggarwal. "Context based Web Indexing for Storage of Relevant Web Pages." *International Journal of Computer Applications Vol40* (2012).

[22] Boumerzoug, Hayette, Boucif Amar Bensaber, and Ismail Biskri. "A key management method based on an avl tree and ecc cryptography for wireless sensor networks." *Proceedings of the 7th ACM symposium on QoS and security for wireless and mobile networks*. ACM, 2011.

[23] Bhukya, Devi Prasad, and S. Ramachandram. "Decision tree induction: an approach for data classification using AVL-tree." *International Journal of Computer and Electrical Engineering* 2.4 (2010): 660-665.

[24] Sicheng, An, et al. "Advanced Red-Black Algorithm for Real-Time Data Addressing of Control System." *Sensors & Transducers (1726-5479)* 178.9 (2014).

[25] Brijendra Kumar Joshi, “Data Structures and Algorithms in C++”, Tata McGraw-Hill Education, 2010, 360 pages

[26] Jeff Edmonds, How to think about algorithms, Cambridge University Press,2008, USA

[27] Štrbac-Savić, Svetlana, and Milo Tomašević. "Comparative performance evaluation of the AVL and red-black trees." *Proceedings of the Fifth Balkan Conference in Informatics*. ACM, 2012.

[28] Ng, Chetan Kumar, et al. "Improving system predictability and performance via hardware accelerated data structures." *Procedia Computer Science* 9 (2012): 1197-1205.