Red Black Trees using Iterators

Kashish Oberoi
B.Tech in Computer Science and Engineering
PES University
Bangalore, India
kashishoberoi00@gmail.com

Sagar Ratan Garg
B.Tech in Computer Science and Engineering
PES University
Bangalore, India
sagarratangarg@gmail.com

***Abstract*—In this project, we have implemented a generic approach on Red Black Trees, which are an optimization of binary search trees, they are self balancing trees which is taken care by the coloring scheme and rotation policies of the data structure. Red-black trees are a well-known way of implementing balanced 2-3-4 trees as binary trees. The main advantage of Red-Black trees is that in both insertions and deletions a single top-down pass may be used. Red Black trees offer worst-case guarantees for all insertion time, deletion time, and search time. We have used the concept of iterators to perform operations on the Red Black Tree. Iterators play a critical role in connecting algorithms and operations with the Red-Black Tree Structure along with the manipulation of data stored inside. We have used bidirectional iterators to move to the predecessor/ancestor as well as the successor of the given node. The use of iterators, improves readability and usability of the code, it is easier to build over to add more features.For generic implementation, we have used typename to declare the contents of the node in the Red Black Tree.**

I. Introduction

Red Black trees are self-balancing binary search tree where
1) Every node has a color either red or black.
2) The root of the tree is always black.
3) There are no two adjacent red nodes (A red node cannot have a red parent or red child).
4) Every path from a node (including root) to any of its descendant NULL node has the same number of black nodes.
Red-black trees are a well-known way of implementing balanced 2-3-4 trees as binary trees. Red-black trees realize 3- and 4-nodes by connecting binary nodes. While this(at worst) doubles the height of the tree, compared to the associated 2-3-4 tree, it does not affect the number of comparisons a search has to make, and it simplifies the balancing process considerably.

The main advantage of Red-Black trees is that in both insertions and deletions a single top-down pass may be used. Red Black trees offer worst-case guarantees for all insertion time, deletion time, and search time. CPU and Memory are the two main hardware resources through which virtualization can be successfully reached. CPU utilization can be maximized along with the distribution of resources between the tasks. Completely Fair Scheduler (CFS) of Linux is the default scheduler of Linux and it makes sure that equal opportunity is distributed between tasks. Based on minimal virtual time CFS selects it's next process and to perform this efficiently it uses red black trees (RB tree).

The generic approach for the Red-Black Tree ensures that the node can contain various data types. We have used the concepts of the iterator to implement Red-Black Trees. An iterator is an object (like a pointer) that points to a node inside the Red-Black Tree. We can use iterators to move through the contents of the Red-Black Tree. They can be visualized as something similar to a pointer pointing to some location and we can access the content at that particular location using them. Iterators play a critical role in connecting algorithms and operations with the data structure along with the manipulation of data stored inside.

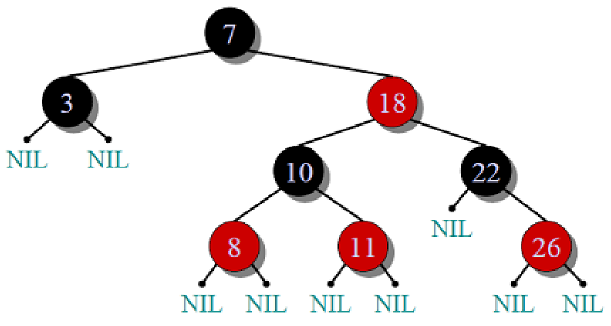

Fig. 1. Red Black Tree

For our implementation we have used bidirectional iterators, Bidirectional iterators are iterators that can be used to access the sequence of elements in a range in both directions (towards the end and the beginning). They are similar to forward iterators, except that they can move in the backward direction also, unlike the forward iterators, which can move only in the forward direction. Some of its salient feature that helps the programmer move through the Red-Black Trees to perform operations are - 1)Usability 2)Equality / Inequality Comparison 3)Dereferencing 4)Incrementable 5)Decrementable. Some of the features that can't be implemented using Bidirectional iterators is the use of offset or provide random access to the data structure which was not the requirement for our implementation and thus useful for implementing Red-Black Trees.

II. Implementation

A. Structural Specification

The structure of a single node as follows:-

```
struct rb_node
{
.        typedef T value_type ;
.        T value ;
.        rb_node * parent ;
.        rb_node * left ;
.        rb_node * right ;
.        bool color ;
}
```

The above structure in the code is preceded with template¡typename T¿ which is used for to implement generic content in the node. Templates are expanded at compiler time. This is like macros. The difference is, compiler does type checking before template expansion. The idea is simple, source code contains only function/class, but compiled code may contain multiple copies of same function/class. The given node is a class templates which are useful when a class defines something that is independent of the data type. Rest of the attributes of the Red Black Trees correspond to the features of a binary tree. Here,
1) *parent is a pointer to the parent node. The parent node is the first ancestor of the given node, for the root node the parent node is always NULL. Having this in the node structure helps in bidirectional movement of the iterator.
2) *left and *right nodes are the child nodes of the node. Here, according to the property of binary search tree, the value in the left node is always less than the value in the right node.
Red Black Trees are different from the binary search trees because of the Boolean bit introduced, that is, the colour of the node. The rest of the operations and algorithms are implemented based on this bit.

The structure of the Red Black Tree:-

```
template¡ typename T ¿
class rb_tree {
.   public:
.        // type definitions:
.        typedef rb_node¡ T ¿ node_type ;
.        typedef T value_type ;
.        typedef rb_iterator¡ node_type ¿ iterator ;

.        //members
.        node_type * root;
.        unsigned long size;
}
```

This structure has two important members, the root pointer which is the pointer to the root node of the Red-Black Tree defined by rb_node. The size of the Red-Black tree is also used to ease computations.

B. Operators Supported

The implementation of generic Bi-directional iterators to move through the Red Black Trees required the support of operators that would assist in accessing the Red-black Trees. The operators supported in our implementation are:-
1) Referencing and De-referencing: In our implementation, referencing refers to returning the pointer value that is the generic value stored in the node and de-referencing must return the iterator itself.
2) Relational Operators: While inserting, searching and performing other basic operations in a binary search tree or in fact a Red-Black trees we need to compare the values contained in the node and the given value. For that feature in our implementation we have defined Relational Operators to have left hand value as the generic type value in the node.
3) Increment/Decrement: In our structural implementation, we saw that we need to have a pointer to the immediate ancestor in the node itself and pointers to its child node to provide bidirectional movement. Similarly, here in operators we need to define the operators increment and decrement to move to the next valid node wherever in the tree structure it may be. If the iterator is at a height h, then we must be able to move to height 0 and H(highest possible) and that is made possible due to these operators.
4) Assignment Operator: The standard Assignment operator would call a copy constructor or copy the whole structure if supported. In our implementation, the node contains the generic type value to which the value on the right hand side should be assigned.

C. Featured Operations

1) Insertion: In Red-Black trees we use coloring and rotation to balance the tree after the insertion. Steps involved-

- Perform standard BST insertion and make the color of newly inserted nodes as RED.
- If x is root, change color of x as BLACK (Black height of complete tree increases by 1)
- Do following if color of x's parent is not BLACK and x is not root.

  . a) If x's uncle is RED
  . (i) Change color of parent and uncle as BLACK.
  . (ii) color of grand parent as RED.
  . (iii) Change x = x's grandparent, repeat steps 2 and 3 for new x

  . b) If x's uncle is BLACK, then there can be four configurations for x, x's parent and x's grandparent(This is similar to AVL Tree)
  . i) Left Left Case (p is left child of g and x is left child of p)
  . ii) Left Right Case (p is left child of g and x is right child of p)
  . iii) Right Right Case (Mirror of case i)

. iv) Right Left Case (Mirror of case ii)

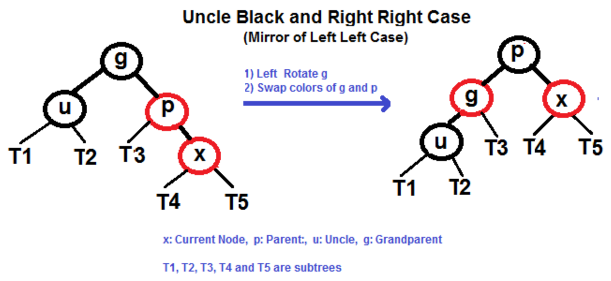
Uncle Black and Right Right Case
(Mirror of Left Left Case)
1) Left Rotate g
2) Swap colors of g and p
g
u
p
x
T1
T2
T3
T4
T5
x: Current Node, p: Parent:, u: Uncle, g: Grandparent
T1, T2, T3, T4 and T5 are subtrees

Fig. 2. Insertion Example

2) Adjusting Red Black Tree: Adjusting the color scheme in accordance to the insertion of a node in a Red Black Tree.

3) Rotations: Left rotation and Right Rotation is basically restructuring the links of the tree to maintain the inorder ordering of keys. A rotation can be performed in O(1) time.

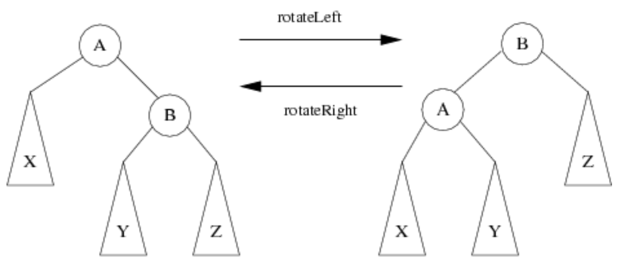

Fig. 3. Rotation

4) Height of the Red-Black Tree: It is similar to evaluating height of a Binary tree. The height of a Red-Black tree is always O(Logn) where n is the number of nodes in the tree. There is also a concept of Black Height in R-B trees, it is number of black nodes on a path from root to a leaf. Leaf nodes are also counted black nodes. From above properties 3 and 4, we can derive, a Red-Black Tree of height h has black-height = h/2.
Other operations like searching, deletion and so can be done by using the structural advantages.

Acknowledgment

This project was supported by Mr NS Kumar, PES University. We are grateful to Mr NS Kumar, professor, in the Department of Computer Science and Engineering. We are extremely thankful and indebted to him for sharing expertise, and sincere and valuable guidance and encouragement extended to our group. We would also like to thank the department of Computer Science and engineering, PES Universiy for providing us such an opportunity.

References

[1]Red-black trees with types,Stefan Kahrs

[2]https://www.cs.princeton.edu/ appel/papers/redblack.pdf

[3]Red Black Trees and It's Application

[4]https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/video-lectures/lecture-10-red-black-trees-rotations-insertions-deletions/lec10.pdf

[5]https://www.geeksforgeeks.org/red-black-tree-set-2-insert/

[6]https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/?ref=lbp

[7]http://fpsalmon.usc.es/manuales/STL/tiny_stl.htm

[8]https://www.programiz.com/dsa/insertion-in-a-red-black-tree