GENERIC PROGRAMMING : UE18CS331

Project ID - 30

GENERIC RED-BLACK TREES

Name	SRN
Anant Thazhemadam	**PES2201800014**
Sampath Sri Harsha Durvasula	**PES2201800019**
Roshini Bhaskar K	**PES2201800122**

1. Abstract

Red Black Trees are self balancing binary search trees with one extra field per node known as *color*. A node's color can either be *red* or *black*. Each node of the tree now contains the fields *parent, left , right, value* and *color.* If a node's child or parent does not exist, then the pointer field of the node is equated to the value NIL. All the leaf nodes are also made to point to NIL which corresponds to a node with all the fields set to nullptr except the color which is set to *black*.

The tree is balanced by setting each node of the tree with either red or black color such that they satisfy the following properties:

- Root node is always black
- Every node is either red or black.
- Every leaf (NIL) is black.
- If a node is red, then both its children are black.
- Every simple path from a node to a descendant leaf contains the same number of black nodes.

Time Complexity (in big *O* notation):

Algorithm	**Average**	**Worst**
Space	*O(N)*	*O(N)*
Search	*O(logN)*	*O(logN)*
Insert	*O(logN)*	*O(logN)*
Delete	*O(logN)*	*O(logN)*

2. Implementation

The Red-Black tree itself is represented by a class named **RBTree**. An **RBTree** maintains nodes, which are represented by the type **RBNode**.

♦ ***class RBTree***

Clients can create a tree with any type and can also define the predicate for the tree. By default, the type is set to int and the predicate is set to std::less<T>. The **RBTree** class also has one static smart pointer (*unique_ptr*) called *NIL*. This is abstracted to behave similar to a nullptr and a non-existent node in this implementation.

- **Special Functions:**
  RBTree deals with dynamic memory allocation. Thus, special functions must be created, to make the class canonical. Constructors that either create an empty tree, or a tree with a given node as root, copy constructor, copy assignment constructor, move constructor, move assignment constructor, and destructor have all been defined for this class.

- **Operator functions:**
  Operator functions for output into a stream, equality and inequality checking and have also been defined.

- ***root, begin and end***
  Return an **Iterator** to the root of the tree, first node in the inorder traversal, and *NIL* respectively

- ***is_empty***
  Return a boolean value that signifies if the tree is empty or not

- ***insert***
  An element can be inserted in the tree using a pointer to an RBNode or by value. Insertion occurs based on the predicate provided. After inserting a new node, *insert_fixup* is called to balance the tree based on color orientation. After insertion, an **Iterator** to the newly inserted node is returned.

- ***remove***
  RBNodes can be removed from the tree if either a value, node, or an iterator is provided. A range can also be specified (using two iterators) for removal of nodes. If the node to be deleted is *black* then *remove_fixup* is called to re-balance the tree. In case of non-leaf nodes, the node is replaced by its inorder successor.

- ***search***
  Clients can search the tree for a node using an **RBNode**, value, or iterator.
  If a matching node is found, an **Iterator** to it is returned. Else, an **Iterator** to *NIL* is returned.

- ***display***
  Display the tree in a structured, tree-like manner.
  Note:
  Functions for printing the elements of the tree in preorder, inorder, and postorder have also been created.

- ***bounds - lower and upper***
  Find the lower and upper bound elements in an RBTree, given a value of an RBNode, and return an Iterator to the same. If no such element is found in the tree, return end() instead.

♦ ***class RBNode***

An object of type **RBNode** represents a node which could be stored in an **RBTree**.

- The **RBTree** class is a friend class of **RBNode**.
- **RBNodes** can be created using either the default constructor, or by using the constructor that takes in the node attributes as arguments (with default values defined). Since the **RBNode** class by itself does not dynamically allocate memory, the remaining special functions have been set to default.
- Operator functions for output into a stream, equality and inequality checking and have also been defined.
- As no operation of interest to us is done using the **RBNode** type itself, no other member functions have been defined.

3. Iterators

Iterator support for the **RBTree** class is facilitated by the nested class **Iterator**. This iterator class is implemented to provide a bidirectional iterator for **RBTree**, and it inherits from the iterator class defined in <*iterator*> accordingly.
The default order of traversal adopted by **Iterator** is in-order.

- **Special Functions:**
  Constructors that either initialize an iterator that doesn't point to an actual node, or initialize an iterator to point to a node, along with a copy constructor and copy assignment operator have been created.
    - Since the iterator class doesn't dynamically allocate memory, the remaining special functions have been omitted, or set to default.

- **Operator Functions:**
  Iterator supports the following operator functions:
    - Dereferencing
    - Post and Pre Increment
    - Post and Pre Decrement
    - Equality and Inequality

- **Traversal methods**
  The **Iterator** class also provides different methods for tree traversal using iterators.
    - Level-order Predecessor
    - Level-order Successor
    - Pre-order Predecessor
    - Pre-order Successor
    - In-order Predecessor
    - In-order Successor
    - Post-order Predecessor
    - Post-order Successor

  Note:
  These traversal methods return a pointer to an **RBNode**.

4. Snapshots

- *Tree creation (Insertion)*

```
222     #if 1 // user defined type tree creation
223     struct temp y1 = {5, 13}; // 5 < 13 T/F
224     struct temp y2 = {6, 12};
225     struct temp y3 = {7, 11};
226     struct temp y4 = {8, 10};
227
228
229     RBNode<struct temp> *ny1_t = new RBNode<struct temp>(y1);
230     RBNode<struct temp> *ny2_t = new RBNode<struct temp>(y2);
231     RBNode<struct temp> *ny3_t = new RBNode<struct temp>(y3);
232     RBNode<struct temp> *ny4_t = new RBNode<struct temp>(y4);
233
234
235     RBTree<struct temp,struct temp>::Iterator it_1_t = tree_user->insert(ny1_t);
236     RBTree<struct temp,struct temp>::Iterator it_2_t = tree_user->insert(ny2_t);
237     RBTree<struct temp,struct temp>::Iterator it_3_t = tree_user->insert(ny3_t);
238     RBTree<struct temp,struct temp>::Iterator it_4_t = tree_user->insert(ny4_t);
239
240     cout << *tree_user;
241     #endif
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

roshini@roshini-VirtualBox:~/Desktop/GP/GP_Project/generic-red-black-trees$ ./RBT

└──12 - black          0x559abe57af60
    ├──11 - black              0x559abe57af90
    │   ├──10 - red            0x559abe57afc0
    └──13 - black              0x559abe57af30
```

- *Pre and post decrement and Pre and post increment on Iterators*

```
242
243     #if 1 // pre post inc/dec
244     cout <<  "Value : " << it_1_t;
245     cout << --it_1_t; //pre decrement
246     cout << it_1_t--; //post decrement
247     cout << it_1_t++; //post increment
248     #endif
249
250     #if 0 // search n delete
251     //cout << tree->search(it_3);
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

roshini@roshini-VirtualBox:~/Desktop/GP/GP_Project/generic-red-black-trees$ ./RBT

└──12 - black          0x559a401a5f60
    ├──11 - black              0x559a401a5f90
    │   ├──10 - red            0x559a401a5fc0
    └──13 - black              0x559a401a5f30

Value : 13      black
12      black
12      black
11      black
roshini@roshini-VirtualBox:~/Desktop/GP/GP_Project/generic-red-black-trees$
```

- ***Has next, next, has prev and prev functions on Iterators***

```
278
279     #if 1  // iterators
280     cout <<  "Value : " << it_1_t;
281     cout << boolalpha;
282     cout <<  it_1_t.hasnext() <<"\n";
283     cout << it_1_t.next().next();
284     cout << it_1_t.hasprev()<< "\n";
285     #endif
286
287     #if 0 //returns address of the nodes
288     cout << it_11.preorder_successor();
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
roshini@roshini-VirtualBox:~/Desktop/GP/GP_Project/generic-red-black-trees$ ./RBT

└──12 - black          0x561505b70f60
    ├──11 - black              0x561505b70f90
    │   ├──10 - red            0x561505b70fc0
    └──13 - black              0x561505b70f30

Value : 13      black
true
0       black
false
```

- ***Tree traversal using iterators (returns the node address)***

```
287     #if 1 //returns address of the nodes
288     cout << it_3_t;
289     cout << it_3_t.preorder_successor() <<"\n";
290     cout << it_1_t.postorder_predecessor(tree_user) << "\n";
291     cout << it_1_t.preorder_predecessor(tree_user) << "\n";
292     #endif
293     return 0;
294 }
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
roshini@roshini-VirtualBox:~/Desktop/GP/GP_Project/generic-red-black-trees$ ./RBT

└──12 - black          0x55ffb3e69f60
    ├──11 - black              0x55ffb3e69f90
    │   ├──10 - red            0x55ffb3e69fc0
    └──13 - black              0x55ffb3e69f30

11      black
0x55ffb3e69fc0
0x55ffb3e69ee0
0x55ffb3e69ee0
```

- ***Deletion on RBTree***

```
253     tree_user->remove(ny1_t); // deleting using node
254     tree_user->remove(it_3_t); // deleting using iterator
255     cout << "After deleting nodes -\n";
256     cout << *tree_user;
257     tree_user->delete_tree();
258     cout << "After deleting tree -\n";
259     #endif
260
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
roshini@roshini-VirtualBox:~/Desktop/GP/GP_Project/generic-red-black-trees$ ./RBT

└──12 - black           0x559422130f60
    ├──11 - black               0x559422130f90
    │   ├──10 - red             0x559422130fc0
    └──13 - black               0x559422130f30

After deleting nodes -

└──12 - black           0x559422130f60
    ├──10 - red         0x559422130fc0

After deleting tree -
roshini@roshini-VirtualBox:~/Desktop/GP/GP_Project/generic-red-black-trees$
```

- ***Searching using predicate(by default it is less than)***

```
251     cout << *tree;
252     cout << tree->search(it_4); //iterator
253     cout << tree->search(ny4); // node
254     cout << tree->search(1000); //value
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
└──7 - black            0x560da9514fe0
    ├──3 - red          0x560da9514f80
    │   ├──1 - black            0x560da9514fb0
    │   └──5 - black            0x560da9515010
    │       ├──4 - red          0x560da95150d0
    └──11 - red         0x560da9514f50
        ├──8 - black            0x560da9515040
        └──12 - black           0x560da9515070
            └──15 - red         0x560da95150a0

4       red
4       red
nullptr
```

- ***Copy constructor and Assignment***

```
269
270     RBTree<int> *tree2 (tree); //copy ctr
271     cout << *tree2;
272
273     tree2 = tree; //copy assignment
274     cout << *tree2;
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
└──7 - black            0x557f5a6dffe0
    ├──3 - red          0x557f5a6dff80
    │   ├──1 - black            0x557f5a6dffb0
    │   └──5 - black            0x557f5a6e0010
    │       ├──4 - red          0x557f5a6e00d0
    └──11 - red         0x557f5a6dff50
        ├──8 - black            0x557f5a6e0040
        └──12 - black           0x557f5a6e0070
            └──15 - red         0x557f5a6e00a0


└──7 - black            0x557f5a6dffe0
    ├──3 - red          0x557f5a6dff80
    │   ├──1 - black            0x557f5a6dffb0
    │   └──5 - black            0x557f5a6e0010
    │       ├──4 - red          0x557f5a6e00d0
    └──11 - red         0x557f5a6dff50
        ├──8 - black            0x557f5a6e0040
        └──12 - black           0x557f5a6e0070
            └──15 - red         0x557f5a6e00a0
```

- ***Move constructor and assignment***

```
276     RBTree<int> tree2 = std::move(tree1); //move ctr
277     cout << tree2;
278     cout << "--\n";
279     cout << "Original tree : "<< tree1;
280
281
282     #if 0  // iterators
283     cout <<  "Value : " << it_1_t;
284     cout << boolalpha;
285     cout <<  it_1_t.hasnext() <<"\n";
286     cout << it_1_t.next().next();
287     cout << it_1_t.hasprev()<< "\n";
288     #endif
```

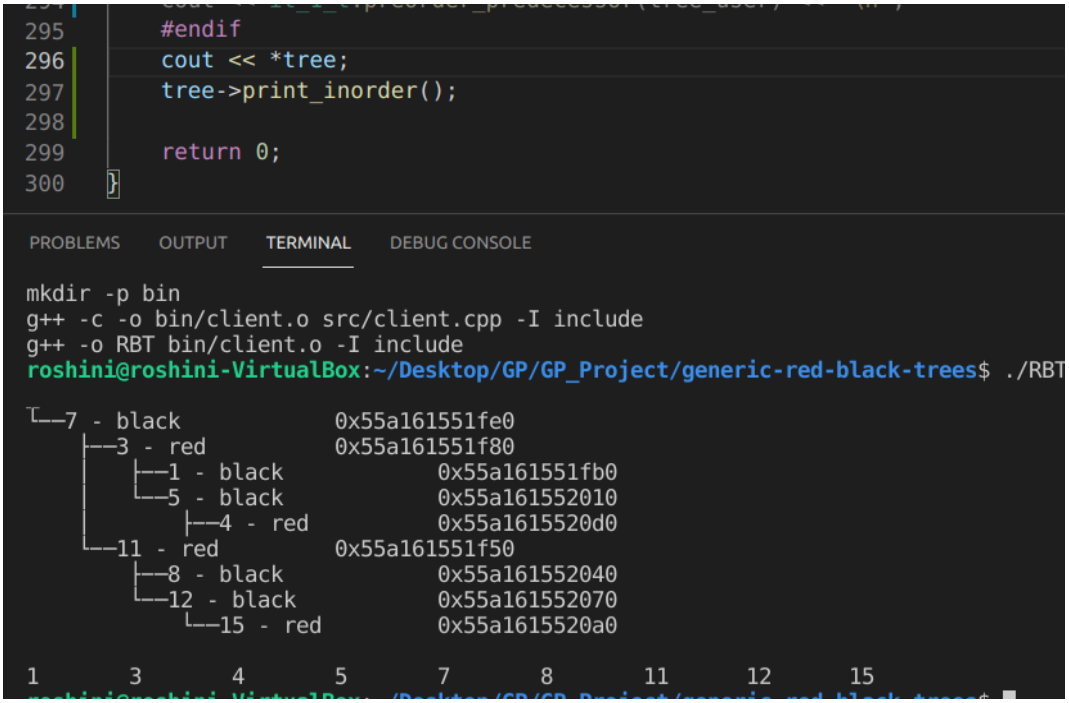
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
g++ -o RBT bin/client.o -I include
roshini@roshini-VirtualBox:~/Desktop/GP/GP_Project/generic-red-black-trees$ ./RBT

└──4 - black            0x559b87cbc1f0
    ├──2 - red          0x559b87cbc130
    │   ├──1 - black            0x559b87cbc160
    │   └──3 - black            0x559b87cbc190
    └──10 - red         0x559b87cbc100
        ├──8 - black            0x559b87cbc250
        │   ├──6 - red          0x559b87cbc280
        │   └──9 - red          0x559b87cbc220
        └──13 - black           0x559b87cbc1c0

--
Original tree : _
```

- ***Tree traversals (inorder of the tree printed here)***



5. References

i. Introduction to Algorithms, 3rd Edition - Chapter 13
ii. https://en.cppreference.com/w/cpp
iii. https://en.wikipedia.org/wiki/Red%E2%80%93black_tree