Generic BinarySearch Tree

Abstract **: This paper represents the implementation of GenericBinarySearch Tree library built using C++. All necessary Operator overloading is implemented here. Implementation is quite similar to a normal c++ library module. All the Implementation details and also the necessary information for the user to use this module are described here. This class also has an Iterator which traverses trees by level order. Each Node is represented by the private class named TreeStruct . This class is a friend of BinaryTree i.e. main class , hence users cannot modify the TreeStruct class.**

I. Introduction

This module is built using C++ and it is built in a generic way. Users can pass any type of collection of objects and it takes all the objects and builds a binary search tree and returns an object to the user. In case if objects passed do not support '<' operator then there is also provision of providing ranks to this class so that it uses this corresponding rank to compare . Rank provided here is of integer type. In case if a user wishes to make an entire entire collection to act as a node of Tree he/she can do so, which is explained below. There is also a method for adding a node and deleting a node provided. There is a method called BalanceTree(), where users can get a height balanced binary search tree and there are three different kinds of tree traversal for viewing the tree. There is also an iterator class provided to BinaryTree, here users can make use of this. Iterator traversal is by level order queue is used for level ording. Copy and Move constructor and Copy and Move operator is also provided

Each node is represented by the class called TreeStruct which takes objects to be stored and ranks if rank exists. Operators like '<' , '!=', '==' are overloaded. This is a private class where clients cannot get access to this class. It has a friend class which is BinaryTree So only the BinaryTree class can get all access to its private members. This class has left, right, up pointers to build a tree.

Note : Once Tree gets Created users cannot modify the value of the node

2. Constructor and destructor

There are totally 7 different constructor including copy and move constructor. Users can provide different kind of input i.e.

- Can provide collection objects , where each object in the collection can act as node
- Can provide collection of objects and array of ranks for comparing between objects
- Can provide entire collection to act as node with or without rank
- Empty constructor
- Copy and Move Constructor

One destructor which will delete the allocated node

3. *Methods available to users*

1. ***inOrder(), preOrder() and postOrder()***
   These are used for displaying the tree

2. ***void addNode(const T& a, bool, int rank);***
   This is the template function which takes object as first parameter and if rank is provided then bool value is set to true else false

3. ***bool deleteNode(const T& obj)***
   This function is a template function which is used to delete the node specified

4. ***Void BalanceTree()***
   This function is used to balance the unbalanced Tree. Time complexity is O(n) and Space is also O(n)

5. ***int NoOfNode()***
   This function is used to return the No of nodes present in the Tree at a given time

6. ***bool findNode(const T2&, int rank)***
   This function returns true if given Object present in the tree else false

7. **void erase()**
   This function used to remove all the nodes and clear the given tree

8. ***Iterator begin()***
   This function returns an iterator object to root of the tree

9. ***Iterator end()***
   This is used to denote end of tree traversal

4. *Iterator class*

Iterator class is used for traversing the tree by level order which uses queue for doing this. Here we implemented a forward iterator. begin() function of the BinaryTree class passes the root node of the tree and end() function passes the nullptr to denote the end of traversal Operators like *, ++(pre and post), == , != is overloaded for users' use.

5. *Operators overloaded for BinaryTree*

Here we implemented equality operator **(==, !=)** and + operator
Equality operators used here compare node by node for equality and each node should be at exactly the same position in both the trees provided. If equal this returns true else false .

'+' Operator is used to merge two Binary trees or concat two given trees . This function returns the newly merged tree if created without any kind of error

6. *Additional information*

Data structured used to implement above BinaryTree are vector and queue

We used vector to store the reference of the nodes in copy and move constructors
We used queue for level order traversing and in iterator

7. *Reference*

1. http://www.cplusplus.com/reference/vector/vector/
2. https://en.cppreference.com/w/cpp/container/list
3. https://www.tutorialspoint.com/cpp_standard_library/list.html
4. C++ FAQ -Marshall Cline Greg Lomow Mike Girou
5. The C++ Standard Library_ A Tutorial and Reference (Nicolai M. Josuttis)

Team Members :

1. **Adeesh**
   PES1201700959

2. **Manthan BY**
   PES201700959