C++ BST PROJECT

JUNE 2016

Name: *Cezar Angelo*
Surname: *Sas*
ID: *781563*
Mail: *c.sas@campus.unimib.it*

Introduction

After the evaluation of the project, I decided to implement the BST as a list of nodes in which each node has two children. Each node contains a value, which is unique in the tree.

Data Types

The BST is implemented using the nodes as the container for the values inserted, the node is structured as follows:

- Value: a templated variable containing the value of the node, the type is defined at the moment of the tree declaration
- Parent: a pointer to the parent of the node, 0 if the node is root
- Left: a pointer to a node that represents the left child
- Right: a pointer to a node that represents the right child

Implementation

The BST is implemented with the well know binary tree logic:

> *All the values of the nodes in the left subtree are less than the value of the root of the subtree. The values on the right are greater.*

The class implements all the necessary constructors, the default constructor, the destructor and the secondary constructors.

The default constructor builds an empty tree, so a tree with root pointer equals to 0 and size 0. The destructor deletes all the nodes created by the program at runtime. The final constructor is the copy constructor, that takes in a tree and copies all its node in the new tree.

Methods

The class allows users to insert, remove, and check if a value already exist in the tree. All these methods to achieve their goal implements a search in the tree based on the binary tree logic. These methods use the functor passed in by the user at the declaration time. This functor is responsible of telling the BST class if a value A is less/greater than or equals to a value B, so the methods can use a dichotomic search and find the value they are looking for in $O(\log n)$.

After the search is terminated if the value doesn't exist the insert method creates and links a new node into the tree.

The remove method instead unlinks the node to be removed and links the tree to preserve the order of the values. After the unlink process the node is deleted and the size in decreased.

The class also implements the print method which uses the const forward iterator to visit all the nodes with a pre-order visit. The iterator uses a helper method called GET_NEXT_NODE(), this method implements the pre-order visit.

In addition to the BEGIN() and END() methods that returns the iterator start pointer and the end pointer, there are also BEGIN_SUBTREE(VAL) and END_SUBTREE(VAL) that allows users to iterate over a subtree with root val. The decision to implement these two extra methods is that allows to implement the global function SUBTREE(TREE, VAL) in a faster and efficient way, instead of iterate over all the tree to find the value in $O(n)$ we can use the find method and do it $O(\log n)$

The bst.h files implements a global function, called SUBTREE(TREE, VAL), this function takes a tree and a value, if the value is in the subtree then returns a copy of the subtree with root val. This function uses the BEGIN_SUBTREE(VAL) and END_SUBTREE(VAL) in this way can copy a subtree in a faster way.

The BST class uses custom exception extended from the standard library, these exceptions throw a custom message, explaining the error.

Testing

The main.cpp file implements some test for the BST class, the three implemented test runs the same instruction but on different data types:

- Test 1: int
- Test 2: strings
- Test 3: custom made struct representing a point in a 2D space

The tests try to cover each possible scenario in the life of the class, including an insert with duplicates values, remove of values from the tree, copies of the tree, and extraction of a subtree.