Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

Computer Programming

«Binary Search Tree»

author	Wojciech Dolibog
instructor	dr inż. Piotr Fabian
year	2022-2023
lab group	even Tuesday, 09:45 – 11:15
deadline	2023-02-07

1 Project's topic

Implement Binary Search Tree defined as a class that supports user-defined data types.

2 Analysis of the task

A Binary Search Tree (BST) is a type of data structure that allows for efficient searching, insertion, and deletion of elements. Each node in the tree has a value, and the left child of a node has a value less than its parent, while the right child has a value greater than its parent.

2.1 Data structures

The main data structure used in a BST implementation is the tree node, which contains the value of the node and pointers to its left and right children. The tree node structure should have at least the following members: user-defined data type value, a pointer to the left child, and a pointer to the right child. Additionally, a BST class should be implemented to manage the overall tree structure, including methods for insertion, deletion, and searching.

2.2 Algorithms

Insertion: To insert a new value into the BST, the algorithm starts at the root of the tree and compares the new value to the value of the current node. If the new value is less than the current node's value, the algorithm moves to the left child. If the new value is greater, the algorithm moves to the right child. This process is repeated until a leaf node is reached, at which point the new value is inserted as a child of that leaf node.

Deletion: To delete a value from the BST, the algorithm first searches for the node containing the value to be deleted. If the node has no children, it is simply removed from the tree. If the node has one child, the child takes the place of the deleted node. If the node has two children, the algorithm must find the smallest value in the right subtree to replace the deleted value.

Searching: To search for a value in the BST, the algorithm starts at the root of the tree and compares the search value to the value of the current node. If the search value is less than the current node's value, the algorithm moves to the left child. If the search value is greater, the algorithm moves to the right child. This process is repeated until the search value is found or a leaf node is reached (indicating that the value is not in the tree).

3 External specification

This is a command line program.You can execute a program by using compiled .exe or .o file or by using make in the e.g. Bash terminal.

4 Internal specification

The program is implemented with object-oriented, structural and functional paradigm.

5 Conclusions

Binary Search Tree (BST) is a powerful data structure that allows for efficient searching, insertion, and deletion of elements. Insertion, deletion, and searching are all basic operations in BST, and their time complexity is O(log n) on average. However, it is important to note that if the tree is not balanced, the time complexity can degrade to O(n) in the worst case. Overall, the BST provides a valuable tool for dealing with large amounts of data, and its efficient algorithms make it a great choice for a wide range of applications.

Appendix
Description of types and functions

Binary Search Tree

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 BST< T > Class Template Reference

Binary Search Tree.

`#include <Bst.h>`

Public Member Functions

- **BST** ()
  Constructor of the binary search tree.
- BST (const BST< T > &other)
  Copy constructor of the binary search tree.
- BST (BST< T > &&other)
  Move constructor of the binary search tree.
- bool isNotEmpty ()
  Checks if binary search tree has any nodes.
- void insert (T _data)
  Inserts new node to the binary search tree.
- T findMinimum ()
  Finds minimum value in the binary search tree.
- T findMaximum ()
  Finds maximum value in the binary search tree.
- bool find (T _data)
  Checks if the node is in the binary search tree.
- void remove (T _data)
  Removes the node from the binary search tree.
- void **erase** ()
  Erases whole binary search tree, sets root pointer to nullptr.
- void **printInOrder** ()
  Prints values of the binary search tree in in-order traversal.
- void **printPreOrder** ()
  Prints values of the binary search tree in pre-order traversal.
- void **printPostOrder** ()
  Prints values of the binary search tree in post-order traversal.

3.1.1 Detailed Description

template<typename T>
class BST< T >

Binary Search Tree.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BST() [1/2]

```
template<typename T >
BST< T >::BST (
            const BST< T > & other )
```

Copy constructor of the binary search tree.

Parameters

other	Binary search tree from which we copy the data

3.1.2.2 BST() [2/2]

```
template<typename T >
BST< T >::BST (
            BST< T > && other )
```

Move constructor of the binary search tree.

Parameters

other	Binary search tree from which we move the data

3.1.3 Member Function Documentation

3.1.3.1 find()

```
template<typename T >
bool BST< T >::find (
            T _data )
```

Checks if the node is in the binary search tree.

Parameters

_data	Value that is checked

Returns

True if the node is found, false otherwise

3.1.3.2 findMaximum()

```
template<typename T >
T BST< T >::findMaximum
```

Finds maximum value in the binary search tree.

Returns

Maximum value

3.1.3.3 findMinimum()

```
template<typename T >
T BST< T >::findMinimum
```

Finds minimum value in the binary search tree.

Returns

Minimum value

3.1.3.4 insert()

```
template<typename T >
void BST< T >::insert (
            T _data )
```

Inserts new node to the binary search tree.

Parameters

_data	Value to initialize the node

3.1.3.5 isNotEmpty()

```
template<typename T >
bool BST< T >::isNotEmpty
```

Checks if binary search tree has any nodes.

Returns

True if has at least one node, false if binary search tree is empty

3.1.3.6 remove()

```
template<typename T >
void BST< T >::remove (
            T _data )
```

Removes the node from the binary search tree.

Parameters

_data	Value that is removed

The documentation for this class was generated from the following files:

- Bst.h
- Bst.cpp

Chapter 4

File Documentation

4.1 Bst.cpp File Reference

Binary Search Tree implementation.

```
#include <iostream>
#include "Bst.h"
```

4.1.1 Detailed Description

Binary Search Tree implementation.

Author

Wojciech Dolibóg

Version

0.1

Date

2023-01-19

Copyright

4.2 Bst.h File Reference

Binary Search Tree class.

```
#include <memory>
```

Classes

- class BST< T >

  Binary Search Tree.

4.2.1 Detailed Description

Binary Search Tree class.

Author

Wojciech Dolibóg

Version

0.1

Date

2023-01-19

Copyright

Copyright (c) 2023

4.3 Bst.h

Go to the documentation of this file.

```
1
12 #pragma once
13
14 #ifndef BST_H
15 #define BST_H
16
17 #include <memory>
22 template <typename T>
23 class BST
24 {
25 private:
30     struct Node
31     {
35         T data;
39         std::unique_ptr<Node> leftChild;
43         std::unique_ptr<Node> rightChild;
48         Node(T _data);
49     };
50
54     std::unique_ptr<Node> root;
60     std::unique_ptr<Node> copyHelper(const std::unique_ptr<Node>& other);
66     void insertHelper(std::unique_ptr<Node>& currentNode, T _data);
72     T findMinHelper(std::unique_ptr<Node>& currentNode);
78     T findMaxHelper(std::unique_ptr<Node>& currentNode);
84     bool findHelper(std::unique_ptr<Node>& currentNode, T _data);
90     void removeHelper(std::unique_ptr<Node>& currentNode, T _data);
95     void eraseHelper(std::unique_ptr<Node>& currentNode);
100     void inOrder(const std::unique_ptr<Node>& currentNode);
105     void preOrder(const std::unique_ptr<Node>& currentNode);
110     void postOrder(const std::unique_ptr<Node>& currentNode);
111
112 public:
116     BST();
121     BST(const BST<T>& other);
126     BST(BST<T>&& other);
```

```
131     bool isNotEmpty();
136     void insert(T _data);
141     T findMinimum();
146     T findMaximum();
152     bool find(T _data);
157     void remove(T _data);
161     void erase();
165     void printInOrder();
169     void printPreOrder();
173     void printPostOrder();
174 };
175
176 #endif
```