|  |  |
| --- | --- |
| **Российский университет транспорта (МИИТ) Институт транспортной техники и систем управления Кафедра «Управление и защита информации»** | |
| **Отчет по практическому заданию по теме «Структуры данных» по дисциплине «Системы управления базами данных»** | |
|  | Выполнил:  Студент группы ТКИ-442  Жижин А.Д. |
|  | Проверил:  Доцент кафедры УиЗИ к.т.н., с.н.с  Васильева М. А. |
| Москва 2024 | |

Оглавление

[1. Условие задачи 3](#_Toc153275225)

[2. Текст программы на языке С++ 3](#_Toc153275226)

[2.1. ../src/Node.hpp 3](#_Toc153275230)

[2.2. ../src/BST.hpp 5](#_Toc153275231)

[2.3. ../src/main.cpp 8](#_Toc153275232)

[2.4. ../src/CmakeLists.txt 8](#_Toc153275233)

[2.5. ../test/NodeTest.cpp 8](#_Toc153275234)

[2.6. ../test/BSTTest.cpp 9](#_Toc153275235)

[2.7. ../test/CmakeLists.txt 10](#_Toc153275236)

[2.8. CmakeLists.txt 10](#_Toc153275237)

[3. Результат работы программы 11](#_Toc153275238)

1. Условие задачи

Бинарное дерево поиска

Определить класс «Node», включающий следующую информацию:

* Данные любого типа;
* Указатели на поддеревья и родителя;

Определить класс «BST», включающий следующую информацию:

* Указатель на корень дерева;

В программе предусмотреть:

* создание БДП;
* вывод на экран;
* удаление элемента по ключу;
* вставка элемента по ключу;
* поиск данных элемента по ключу;
* поиск минимального и максимального элемента.

1. Текст программы на языке С++

4. 1. ../src/Node.hpp

#pragma once  
#include <iostream>  
#include <sstream>  
#include "BST.hpp"  
  
  
template<typename T>  
class Node {  
 template<typename U>  
 friend class BST;  
private:  
 T data;  
 Node<T> \*left, \*right, \*parent;  
  
public:  
 Node(const Node<T>& other) = delete;  
 explicit Node(T data): left(nullptr), right(nullptr), parent(nullptr), data(data){}  
 ~Node(){  
 this->left = nullptr;  
 this->right = nullptr;  
 if (this->parent != nullptr)  
 {  
 if (this == this->parent->left)  
 this->parent->left = nullptr;  
 else if (this == this->parent->right)  
 this->parent->right = nullptr;  
 }  
 }  
 T getData(){ return data; }  
 Node<T>\* getLeft(){ return left; }  
 Node<T>\* getRight(){ return right; }  
 /\*\*  
 \* @brief swap data between nodes  
 \* @param other - node to swap  
 \*/  
 void dataSwap(Node<T>\* other){  
 auto tmp = this->data;  
 this->data = other->data;  
 other->data = tmp;  
 }  
 /\*\*  
 \* @brief Operator left shift  
 \* @param node - node to output  
 \* @return output stream  
 \*/  
 friend std::ostream& operator<<(std::ostream& out, const Node<T>& node){  
 out << node.data << " ";  
 return out;  
 }  
 /\*\*  
 \* @brief Operator equal  
 \* @param other - node to compare  
 \* @return result of comparison  
 \*/  
 bool operator==(Node<T> const& other) const { return data == other.data; }  
 /\*\*  
 \* @brief Operator not equal  
 \* @param other - node to compare  
 \* @return result of comparison  
 \*/  
 bool operator!=(Node<T> const& other) const { return data != other.data; }  
 /\*\*  
 \* @brief Operator greater  
 \* @param other - node to compare  
 \* @return result of comparison  
 \*/  
 bool operator>(Node<T> const& other) const { return data > other.data; }  
 /\*\*  
 \* @brief Operator lower  
 \* @param other - node to compare  
 \* @return result of comparison  
 \*/  
 bool operator<(Node<T> const& other) const { return data < other.data; }  
};

* 1. ../src/BST.hpp

#pragma once  
#include "Node.hpp"  
template<typename T>  
class Node;  
template<typename T>  
class BST {  
private:  
 Node<T>\* root = nullptr;  
 /\*\*  
 \* @brief func to insert node in tree  
 \* @param node - node to insert  
 \* @param current - buffer for recursion (default = root)  
 \* @param parent - revers link in hierarchy (default = nullptr)  
 \* @return pointer on current obj  
 \*/  
 Node<T>\* inserter(Node<T>\* node, Node<T>\* current, Node<T>\* parent) {  
 if (current == nullptr){  
 current = node;  
 current->parent = parent;  
 return current;  
 }  
 if (\*node < \*current)  
 current->left = inserter(node, current->left, current);  
 else if (\*node > \*current)  
 current->right = inserter(node, current->right, current);  
 return current;  
 }  
 /\*\*  
 \* @brief func to find node in tree  
 \* @param node - node to find  
 \* @param current - buffer for recursion (default = root)  
 \* @return pointer on current obj  
 \*/  
 bool contains(Node<T> \*node, Node<T> \*current) {  
 if (current == nullptr)  
 return false;  
 else if (\*node == \*current)  
 return true;  
 else if (\*node < \*current)  
 contains(node,current->left);  
 else  
 contains(node,current->right);  
 }  
 /\*\*  
 \* @brief func to find minimum node in (sub)tree  
 \* @param node - root node of (sub)tree (default = root)  
 \* @return pointer on minimum node  
 \*/  
 Node<T>\* findMin(Node<T>\* node) {  
 if (node == nullptr || node->left == nullptr)  
 return node;  
  
 return findMin(node->left);  
 }  
 /\*\*  
 \* @brief func to find maximum node in (sub)tree  
 \* @param node - root node of (sub)tree (default = root)  
 \* @return pointer on maximum node  
 \*/  
 Node<T>\* findMax(Node<T>\* node) {  
 if (node == nullptr || node->right == nullptr)  
 return node;  
  
 return findMax(node->right);  
 }  
 /\*\*  
 \* @brief output in increasing order  
 \* @param node - root node of (sub)tree (default = root)  
 \* @param out - output stream  
 \* @return string of nodes in increasing order  
 \*/  
 std::string inOrderTraversal(Node<T>\* node, std::ostringstream& out) {  
 if (node != nullptr) {  
 inOrderTraversal(node->left, out);  
 out << \*node;  
 inOrderTraversal(node->right, out);  
 }  
 return out.str();  
 }  
 /\*\*  
 \* @brief output in post order  
 \* @param node - root node of (sub)tree (default = root)  
 \* @param out - output stream  
 \* @return string of nodes in post order  
 \*/  
 std::string postOrderTraversal(Node<T>\* node, std::ostringstream& out) {  
 if (node != nullptr) {  
 postOrderTraversal(node->left, out);  
 postOrderTraversal(node->right, out);  
 out << \*node;  
 }  
 return out.str();  
 }  
 /\*\*  
 \* @brief func to convert subtree root into leaf  
 \* @param current - root of subtree  
 \* @return pointer on leaf  
 \*/  
 Node<T>\* subtreeToLeaf(Node<T>\* current){  
 if (current->left == nullptr || current->right == nullptr)  
 return current;  
 else if (current->left != nullptr) {  
 current->dataSwap(current->left);  
 subtreeToLeaf(current->left);  
 } else if (current->right != nullptr) {  
 current->dataSwap(current->right);  
 subtreeToLeaf(current->right);  
 }  
 }  
 /\*\*  
 \* @brief func to delete node from tree  
 \* @param data - key to delete  
 \* @param root - root node of (sub)tree (default = root)  
 \*/  
 void nodeRemover(Node<T>\* node, Node<T>\* current) {  
 if (contains(node, current) == false)  
 return;  
 else{  
 if (node->data > current->data)  
 nodeRemover(node, current->right);  
 else if (node->data < current->data)  
 nodeRemover(node, current->left);  
 else{  
 delete subtreeToLeaf(current);  
 }  
 }  
 }  
 /\*\*  
 \* @brief func to delete tree  
 \* @param root - root node of (sub)tree (default = root)  
 \*/  
 void treeRemover(Node<T>\* node){  
 if (node) {  
 treeRemover(node->left);  
 treeRemover(node->right);  
 delete node;  
 }  
 }  
public:  
 BST() { this->root = nullptr; }  
 Node<T>\* getRoot(){ return this->root; }  
 Node<T>\* getMin() { return findMin(this->root); }  
 Node<T>\* getMax() { return findMax(this->root); }  
 /\*\*  
 \* @brief interface for inserter  
 \* @param newData - key to insert  
 \*/  
 void insertNode(const T& newData) {  
 this->root = inserter(new Node<T>(newData), this->root, nullptr);  
 }  
 /\*\*  
 \* @brief interface for contains  
 \* @param dataToFind - key to search  
 \* @return bool - flag - is tree contains it or not  
 \*/  
 bool isContains (T dataToFind) {  
 Node<T>\* tmp = new Node<T>(dataToFind);  
 bool is = contains(tmp, this->root);  
 delete tmp;  
 return is;  
 }  
 /\*\*  
 \* @brief interface for in increase order output  
 \* @return string of nodes in increasing order  
 \*/  
 std::string inOrderPrint() {  
 std::ostringstream buffer{};  
 inOrderTraversal(this->root, buffer);  
 return buffer.str();  
 }  
 /\*\*  
 \* @brief interface for in decrease order output  
 \* @return string of nodes in increasing order  
 \*/  
 std::string postOrderPrint() {  
 std::ostringstream buffer{};  
 postOrderTraversal(this->root, buffer);  
 return buffer.str();  
 }  
 /\*\*  
 \* @brief interface for deleting node from tree  
 \*/  
 void deleteNode(const T& data) {  
 Node<T>\* tmp = new Node<T>(data);  
 nodeRemover(tmp, this->root);  
 delete tmp;  
 }  
 /\*\*  
 \* @brief tree destructor  
 \*/  
 ~BST() { treeRemover(this->root); }  
};

* 1. ../src/main.cpp

#include "BST.hpp"  
  
int main(){  
  
 return 0;  
}

* 1. ../src/CmakeLists.txt

cmake\_minimum\_required(VERSION 3.0)  
project("BinarySearchTree")  
  
set(CMAKE\_CXX\_STANDARD 20)  
  
add\_executable(7th\_semester  
 Node.hpp  
 BST.hpp  
 main.cpp  
)

* 1. ../test/NodeTest.cpp

#include <gtest/gtest.h>  
#include "../src/Node.hpp"  
  
  
TEST(NodeTest, dataConstructorTest) {  
 Node<int>\* node = new Node<int>(0);  
 ASSERT\_EQ(0, node->getData());  
 ASSERT\_EQ(nullptr, node->getLeft());  
 ASSERT\_EQ(nullptr, node->getRight());  
}  
  
TEST(NodeTest, intCompareOperatorTest) {  
 Node<int>\* otherNode = new Node<int>(15);  
 Node<int>\* node = new Node<int>(10);  
  
 ASSERT\_EQ(false, \*node == \*otherNode);  
 ASSERT\_EQ(true, \*node != \*otherNode);  
 ASSERT\_EQ(false, \*node > \*otherNode);  
 ASSERT\_EQ(true, \*node < \*otherNode);  
}  
  
TEST(NodeTest, stringCompareOperatorTest) {  
 Node<std::string>\* node = new Node<std::string>("ozob");  
 Node<std::string>\* otherNode = new Node<std::string>("bozo");  
  
 ASSERT\_EQ(false, \*node == \*otherNode);  
 ASSERT\_EQ(true, \*node != \*otherNode);  
 ASSERT\_EQ(true, \*node > \*otherNode);  
 ASSERT\_EQ(false, \*node < \*otherNode);  
}  
  
TEST(NodeTest, dataSwapTest) {  
 Node<int>\* otherNode = new Node<int>(15);  
 Node<int>\* node = new Node<int>(10);  
 node->dataSwap(otherNode);  
 ASSERT\_EQ(15, node->getData());  
 ASSERT\_EQ(10, otherNode->getData());  
}  
  
int main(int argc, char \*\*argv) {  
 ::testing::InitGoogleTest(&argc, argv);  
  
 return RUN\_ALL\_TESTS();  
}

* 1. ../test/BSTTest.cpp

#include <gtest/gtest.h>  
#include "../src/BST.hpp"  
  
TEST(BSTTest, insertTest) {  
 BST<int>\* tree = new BST<int>();  
 tree->insertNode(10);  
 tree->insertNode(15);  
 tree->insertNode(5);  
  
 ASSERT\_EQ(10, tree->getRoot()->getData());  
 ASSERT\_EQ(5, tree->getRoot()->getLeft()->getData());  
 ASSERT\_EQ(15, tree->getRoot()->getRight()->getData());  
}  
TEST(BSTTest, isContainsTest) {  
 BST<int>\* tree = new BST<int>();  
 tree->insertNode(10);  
 tree->insertNode(15);  
 tree->insertNode(5);  
  
 ASSERT\_EQ(true, tree->isContains(5));  
 ASSERT\_EQ(false, tree->isContains(55));  
}  
TEST(BSTTest, findMinTest) {  
 BST<int>\* tree = new BST<int>();  
 tree->insertNode(10);  
 tree->insertNode(15);  
 tree->insertNode(5);  
  
 ASSERT\_EQ(5, tree->getMin()->getData());  
}  
TEST(BSTTest, findMaxTest) {  
 BST<int>\* tree = new BST<int>();  
 tree->insertNode(10);  
 tree->insertNode(15);  
 tree->insertNode(5);  
  
 ASSERT\_EQ(15, tree->getMax()->getData());  
}  
TEST(BSTTest, removeTest) {  
 BST<int>\* tree = new BST<int>();  
 tree->insertNode(10);  
 tree->insertNode(15);  
 tree->insertNode(5);  
 tree->insertNode(7);  
 tree->insertNode(12);  
 tree->insertNode(3);  
 tree->insertNode(17);  
  
 ASSERT\_EQ(true, tree->isContains(15));  
 tree->deleteNode(15);  
 ASSERT\_EQ(false, tree->isContains(15));  
}

* 1. ../test/CmakeLists.txt

cmake\_minimum\_required(VERSION 3.0)  
set(CMAKE\_CXX\_STANDARD 20)  
project("BSTTest")  
  
enable\_testing()  
add\_executable(test  
 NodeTest.cpp  
 BSTTest.cpp)  
target\_link\_libraries(test gtest gmock)

* 1. CmakeLists.txt

cmake\_minimum\_required(VERSION 3.0)  
project("BinarySearchTreeWithTests")  
  
set(CMAKE\_CXX\_STANDARD 20)  
  
add\_subdirectory(test)  
add\_subdirectory(src)

1. Результат работы программы

|  |
| --- |
|  |
| 1. Результат работы тестов для узла дерева |
|  |
| 1. Результат работы тестов для бинарного дерева поиска |
|  |
| 1. Сводка по работе всех тестов |

|  |
| --- |
|  |
| 1. Анализ памяти с помощью valgrind |

1. UML диаграмма классов

|  |
| --- |
|  |
| 1. Uml диаграмма классов |