|  |  |
| --- | --- |
| **Российский университет транспорта (МИИТ)**  **Институт транспортной техники и систем управления**  **Кафедра «Управление и защита информации»** | |
| **Отчет**  **по практическому заданию**  **по теме «Бинарное Дерево»**  **по дисциплине «Системы управления базами данных»** | |
|  | Выполнил:  Студент группы ТКИ-441  Земсков Н. А.  Моисеенко М. А. |
|  | Проверил:  к.т.н. доцент  Васильева М.А. |
| Москва 2023 | |

Оглавление

[Оглавление 2](#_Toc154500131)

[1 Задание 3](#_Toc154500132)

[2 Описание структуры – Односвязного списка 4](#_Toc154500133)

[2.1 UML-диаграмма 4](#_Toc154500134)

[2.3 Листинг кода 5](#_Toc154500138)

[Tree.h 5](#_Toc154500139)

[Node.h 9](#_Toc154500140)

[test.cpp 9](#_Toc154500141)

[3.1Результат работы программы 12](#_Toc154500142)

[3.2Тестирование программы 13](#_Toc154500143)

[4. Заключение 14](#_Toc154500144)

1. Задание

Разработать структуру данных на языке программирования С++ в ООП парадигме. Реализовать библиотеку классов, обеспечить выполнение операций CRUD (create, read, update, delete) для любого типа данных (template). Переопределить операторы сдвига влево и вправо для возможности чтения и записи из любого потока (консоль, строка, файл). Ни один метод не должен зависеть от консоли или файла, используйте представление в строку. Написать итератор для работы с коллекцией (разработанной структурой данных). По возможности использовать умные указатели (smart pointer), заместо «сырых» (raw). Использование стандартных структур данных возможно в качестве вспомогательных, например, при обходе дерева можно использовать std::vector. Для избежания недоразумений, лучше обсудить использование с преподавателем индивидуально.

Все методы покрыть тестами (модульное тестирование).

Выбранная структура данных: Односвязный список

1. Описание структуры – Односвязного списка
   1. UML-диаграмма

На рисунке 2 представлена UML-диаграмма, показывающая связь классов в проекте.

|  |
| --- |
|  |
| 1. – UML-диаграмма |

* 4. Листинг кода

Tree.h

#ifndef TREE\_H\_INCLUDED

#define TREE\_H\_INCLUDED

#include <iostream>

#include "Node.h"

#include <string>

#include <sstream>

/\*\*

\* @brief Представляет бинарное дерево поиска.

\*

\* @tparam T Тип значений, хранящихся в дереве.

\*/

**template**<**typename** T>

**class** **Tree**

{

Node<T>\* root; /\*\*< Указатель на корневой узел дерева. \*/

Node<T>\* insert\_at\_sub(T i, Node<T>\* p); /\*\*< Вставляет новый узел в поддерево. \*/

Node<T>\* delete\_at\_sub(**const** T& i, Node<T>\* p); /\*\*< Удаляет узел с заданным значением из поддерева. \*/

**int** **countNodes**(Node<T>\* p); /\*\*< Рекурсивно подсчитывает количество узлов в поддереве. \*/

std::string print\_sub(Node<T>\* p, std::ostringstream& out); /\*\*< Рекурсивно формирует строковое представление поддерева. \*/

Node<T>\* minValue(Node<T>\* p); /\*\*< Находит узел с минимальным значением в поддереве. \*/

Node<T>\* maxValue(Node<T>\* p); /\*\*< Находит узел с максимальным значением в поддереве. \*/

Node<T>\* get\_last(Node<T>\* p); /\*\*< Возвращает указатель на последний узел в поддереве. \*/

Node<T>\* get\_first(Node<T>\* p); /\*\*< Возвращает указатель на первый узел в поддереве. \*/

**void** **deleteSubTree**(Node<T>\* p); /\*\*< Рекурсивно удаляет все узлы в поддереве. \*/

**size\_t** t\_size = **0**; /\*\*< Количество элементов в дереве. \*/

**public:**

/\*\*

\* @brief Конструктор по умолчанию для класса Tree.

\*/

Tree() : root{ nullptr } {}

/\*\*

\* @brief Деструктор для класса Tree.

\*/

~Tree()

{

deleteSubTree(root);

}

/\*\*

\* @brief Добавляет новый элемент в дерево.

\*

\* @param i Значение для добавления в дерево.

\*/

**void** add(T i)

{

++t\_size;

root = insert\_at\_sub(i, root);

}

/\*\*

\* @brief Возвращает строковое представление дерева.

\*

\* @return Строка, содержащая значения дерева в порядке возрастания.

\*/

std::string print()

{

std::ostringstream buffer{};

print\_sub(root, buffer);

**return** buffer.str();

};

/\*\*

\* @brief Проверяет, присутствует ли заданное значение в дереве.

\*

\* @param i Значение для проверки.

\* @return True, если значение присутствует, в противном случае - false.

\*/

**bool** **contain**(T i)

{

**return** contain\_sub(i, root);

}

**bool** **contain\_sub**(T i, Node<T>\* p);

/\*\*

\* @brief Удаляет указанный элемент из дерева.

\*

\* @param i Значение для удаления.

\*/

**void** **destroy**(T i)

{

**if** (contain(i))

root = delete\_at\_sub(i, root);

**else**

**return**;

}

/\*\*

\* @brief Получает количество элементов в дереве.

\*

\* @return Количество элементов в дереве.

\*/

**int** **get\_size**()

{

**return** t\_size;

}

/\*\*

\* @brief Получает количество узлов в левом поддереве корня.

\*

\* @return Количество узлов в левом поддереве.

\*/

**int** **getNumberLeftNodes**()

{

**return** countNodes(root->pLeft);

}

/\*\*

\* @brief Получает количество узлов в правом поддереве корня.

\*

\* @return Количество узлов в правом поддереве.

\*/

**int** **getNumberRightNodes**()

{

**return** countNodes(root->pRight);

}

};

**template**<**typename** T>

**void** Tree<T>::deleteSubTree(Node<T>\* p)

{

**if** (p != nullptr)

{

deleteSubTree(p->pLeft);

deleteSubTree(p->pRight);

**delete** p;

}

}

**template**<**typename** T>

**int** Tree<T>::countNodes(Node<T>\* p)

{

**static** **int** nodes;

**if** (p == nullptr)

**return** **0**;

**if** (p->pLeft != nullptr)

{

++nodes;

countNodes(p->pLeft);

}

**if** (p->pRight != nullptr)

{

++nodes;

countNodes(p->pRight);

}

**return** nodes + **1**;

}

**template**<**typename** T>

Node<T>\* Tree<T>::insert\_at\_sub(T i, Node<T>\* p)

{

**if** (p == nullptr)

**return** **new** Node<T>(i);

**if** (i <= p->val)

p->pLeft = insert\_at\_sub(i, p->pLeft);

**else** **if** (i > p->val)

p->pRight = insert\_at\_sub(i, p->pRight);

**return** p;

}

**template**<**typename** T>

std::string Tree<T>::print\_sub(Node<T>\* p, std::ostringstream& out)

{

**if** (p != nullptr)

{

print\_sub(p->pLeft, out);

out << p->val << " ";

print\_sub(p->pRight, out);

}

**return** out.str();

}

**template**<**typename** T>

**bool** Tree<T>::contain\_sub(T i, Node<T>\* p)

{

**if** (p == nullptr)

**return** false;

**else** **if** (i == p->val)

**return** true;

**else** **if** (i <= p->val)

**return** contain\_sub(i, p->pLeft);

**else**

**return** **contain\_sub**(i, p->pRight);

}

**template**<**typename** T>

Node<T>\* Tree<T>::minValue(Node<T>\* p)

{

Node<T>\* current = p;

**while** (current && current->pLeft)

current = current->pLeft;

**return** current;

}

**template**<**typename** T>

Node<T>\* Tree<T>::maxValue(Node<T>\* p)

{

Node<T>\* current = p;

**while** (current && current->pRight)

current = current->pRight;

**return** current;

}

**template**<**typename** T>

Node<T>\* Tree<T>::delete\_at\_sub(**const** T& i, Node<T>\* p)

{

**if** (p == nullptr)

**return** nullptr;

**if** (i < p->val)

p->pLeft = delete\_at\_sub(i, p->pLeft);

**else** **if** (i > p->val)

p->pRight = delete\_at\_sub(i, p->pRight);

**else**

{

**if** (!p->pLeft)

{

Node<T>\* temp = p->pRight;

**delete** p;

--t\_size;

**return** temp;

}

**else** **if** (!p->pRight)

{

Node<T>\* temp = p->pLeft;

**delete** p;

--t\_size;

**return** temp;

}

Node<T>\* temp = minValue(p->pRight);

p->val = temp->val;

p->pRight = delete\_at\_sub(temp->val, p->pRight);

--t\_size;

}

**return** p;

}

#endif // TREE\_H\_INCLUDED

Node.h

#ifndef NODE\_H\_INCLUDED

#define NODE\_H\_INCLUDED

**template**<**typename** T>

**class** **Node**

{

**public:**

Node<T>\* pLeft;

Node<T>\* pRight;

T val;

Node<T>(T val) : val{val}, pLeft{nullptr}, pRight{nullptr} {}

};

#endif

test.cpp

#include "pch.h"

#include <string>

#include "F:\прога\Vasileva111111\Vasileva111111\Tree.h"

TEST(TestCaseName, TestName) {

EXPECT\_EQ(**1**, **1**);

EXPECT\_TRUE(true);

}

TEST(Test\_String, TestPrint\_String) {

Tree<std::string> bst;

bst.add("Zanildo");

bst.add("Helder");

bst.add("Wilson");

bst.add("Ady");

bst.add("Adilson");

bst.add("Patrick");

EXPECT\_EQ(bst.print(), "Adilson Ady Helder Patrick Wilson Zanildo ");

}

TEST(Test\_Int, TestPrint\_Int) {

Tree<**int**> tr;

tr.add(**4**);

tr.add(**6**);

tr.add(**1**);

tr.add(**9**);

tr.add(**2**);

tr.add(**0**);

tr.add(**89**);

tr.add(**12**);

tr.add(**32**);

tr.add(**5**);

tr.add(**22**);

tr.add(**222**);

EXPECT\_EQ(tr.print(), "0 1 2 4 5 6 9 12 22 32 89 222 ");

}

TEST(Test\_Int, TestSizeCorrect\_Int) {

Tree<**int**> tr;

tr.add(**4**);

tr.add(**6**);

EXPECT\_EQ(tr.get\_size(), **2**);

}

TEST(Test\_String, TestSizeCorrect\_String) {

Tree<std::string> tr;

tr.add("4");

tr.add("6");

EXPECT\_EQ(tr.get\_size(), **2**);

}

TEST(Test\_Int, TestAdd\_Int) {

Tree<**int**> tr;

tr.add(**4**);

tr.add(**6**);

EXPECT\_TRUE(tr.contain(**4**));

}

TEST(Test\_String, TestAdd\_String) {

Tree<std::string> tr;

tr.add("4");

tr.add("6");

EXPECT\_TRUE(tr.contain("4"));

}

TEST(Test\_Double, TestAdd\_Double) {

Tree<**double**> tr;

tr.add(**4.33**);

tr.add(**6.343**);

EXPECT\_TRUE(tr.contain(**6.343**));

}

TEST(Test\_Double, TestSizeCorrect\_Double) {

Tree<**double**> tr;

tr.add(**4.33**);

tr.add(**6.343**);

EXPECT\_EQ(tr.get\_size(), **2**);

}

TEST(Test\_Double, TestPrint\_Double) {

Tree<**double**> tr;

tr.add(**4.33**);

tr.add(**6.343**);

EXPECT\_EQ(tr.print(), "4.33 6.343 ");

}

TEST(Test\_Int, TestDestroy\_Int) {

Tree<**int**> tr;

tr.add(**6**);

tr.add(**4**);

tr.destroy(**4**);

EXPECT\_FALSE(tr.contain(**4**));

}

TEST(Test\_Double, TestDestroy\_Double) {

Tree<**double**> tr;

tr.add(**6.343**);

tr.add(**4.4**);

tr.destroy(**4.4**);

EXPECT\_FALSE(tr.contain(**4.4**));

}

TEST(Test\_String, TestDestroy\_String) {

Tree<std::string> tr;

tr.add("6.343");

tr.add("6");

tr.destroy("6");

EXPECT\_FALSE(tr.contain("6"));

}

**3.Работа программы**

3.1Результат работы программы

|  |
| --- |
|  |
|  |
| 1. – Результаты работы программы |

3.2Тестирование программы

Чтобы оценить, что программа работает верно, её нужно протестировать. Результаты работы тестов показаны на рисунке 6.

|  |
| --- |
|  |
| 1. – Результаты работы тестов |

4. Заключение

В ходе задания была изучена теория, связанная с бинарным деревом, были освоены методы работы с ними.

Итогом выполнения задания стала реализация структуры односвязный список на языке программирования C++.