МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



**Дніпровський національний університет  
залізничного транспорту імені академіка В. Лазаряна**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №5**

**з дисципліни «Алгоритми та структури даних»**

**на тему: «Дерева»**

Виконав: студент гр. ПЗ1911

Сіньков Г.О.

Прийняла: ас. каф. КІТ

Куроп’ятник О. С.

Дніпро, 2020

**Лабораторна робота №5**

**Тема.** Дерева.

**Мета**. Ознайомитися з поняттям дерева. Отримати практичні навички реалізації бінарного дерева пошуку.

**Постановка задачі**

Розробити програму мовою java для реалізації бінарного дерево та операцій додавання, видалення, пошуку та перегляду інформації про певний об’єкт(вивід інформаційної частини), симетричного обходу.

Клас дерева та вузла представити двома різними класами. Атрибути класу вузла повинні мати модифікатори доступу private або вузол має бути представлений як внутрішній клас класу дерева. Вузол повинен містити посилання на лівий та правий нащадки, поле ключа, поле інформаційної частини, не містити посилання на батьківський вузол. Значення ключа повинно обраховуватися на основі значення інформаційної частини. При додаванні нового вузла зі значенням ключа, яке вже є в дереві, виконати заміну інформаційної частини. Тип даних інформаційної частини визначити за параметризованим типом з індивідуального завдання до лабораторної роботи 4.

В програмі реалізувати текстове меню для виконання усіх операцій над деревом в довільному порядку. Для кожного пункту меню передбачити зворотній зв'язок у вигляді результату виконання операції та/або повідомлення про успішне виконання операції.

**Текст програми**

Node.java

package com.company;  
  
public class Node {  
 private int value; // ключ узла  
 private Node leftChild; // Левый узел потомок  
 private Node rightChild; // Правый узел потомок  
  
 public void printNode() { // Вывод значения узла в консоль  
 System.*out*.println(" Выбранный узел имеет значение :" + value);  
 }  
  
 public int getValue() {  
 return this.value;  
 }  
  
 public void setValue(final int value) {  
 this.value = value;  
 }  
  
 public Node getLeftChild() {  
 return this.leftChild;  
 }  
  
 public void setLeftChild(final Node leftChild) {  
 this.leftChild = leftChild;  
 }  
  
 public Node getRightChild() {  
 return this.rightChild;  
 }  
  
 public void setRightChild(final Node rightChild) {  
 this.rightChild = rightChild;  
 }  
}

BineryTree.java

package com.company;  
import java.util.Stack;  
  
public class BinaryTree {  
 private Node rootNode; // корневой узел  
  
 public BinaryTree() { // Пустое дерево  
 rootNode = null;  
 }  
  
 public Node findNodeByValue(int value) { // поиск узла по значению  
 Node currentNode = rootNode;  
 while (currentNode.getValue() != value) { // поиск покуда не будет найден элемент или не будут перебраны все  
 if (value < currentNode.getValue()) { // движение влево?  
 currentNode = currentNode.getLeftChild();  
 } else { //движение вправо  
 currentNode = currentNode.getRightChild();  
 }  
 if (currentNode == null) { // если потомка нет,  
 return null;  
 }  
 }  
 return currentNode; // возвращаем найденный элемент  
 }  
  
 public void insertNode(int value) { // метод вставки нового элемента  
 Node newNode = new Node();  
 newNode.setValue(value);  
 if (rootNode == null) { // если корневой узел не существует  
 rootNode = newNode;  
 }  
 else { // корневой узел занят  
 Node currentNode = rootNode; // начинаем с корневого узла  
 Node parentNode;  
 while (true) // мы имеем внутренний выход из цикла  
 {  
 parentNode = currentNode;  
 if(value == currentNode.getValue()) { // если такой элемент в дереве уже есть, не сохраняем его  
 return;  
 }  
 else if (value < currentNode.getValue()) { // движение влево?  
 currentNode = currentNode.getLeftChild();  
 if (currentNode == null){ // если был достигнут конец цепочки,  
 parentNode.setLeftChild(newNode); // то вставить слева и выйти из методы  
 return;  
 }  
 }  
 else { // Или направо?  
 currentNode = currentNode.getRightChild();  
 if (currentNode == null) { // если был достигнут конец цепочки,  
 parentNode.setRightChild(newNode); //то вставить справа  
 return;  
 }  
 }  
 }  
 }  
 }  
  
 public boolean deleteNode(int value) // Удаление узла с заданным ключом  
 {  
 Node currentNode = rootNode;  
 Node parentNode = rootNode;  
 boolean isLeftChild = true;  
 while (currentNode.getValue() != value) { // начинаем поиск узла  
 parentNode = currentNode;  
 if (value < currentNode.getValue()) { // Определяем, нужно ли движение влево?  
 isLeftChild = true;  
 currentNode = currentNode.getLeftChild();  
 }  
 else { // или движение вправо?  
 isLeftChild = false;  
 currentNode = currentNode.getRightChild();  
 }  
 if (currentNode == null) {  
 System.*out*.println("Такого елемента нету или уже был удален!");  
 return false; // yзел не найден  
 }  
 }  
  
 if (currentNode.getLeftChild() == null && currentNode.getRightChild() == null) { // узел просто удаляется, если не имеет потомков  
 if (currentNode == rootNode) // если узел - корень, то дерево очищается  
 rootNode = null;  
 else if (isLeftChild)  
 parentNode.setLeftChild(null); // если нет - узел отсоединяется, от родителя  
 else  
 parentNode.setRightChild(null);  
 }  
 else if (currentNode.getRightChild() == null) { // узел заменяется левым поддеревом, если правого потомка нет  
 if (currentNode == rootNode)  
 rootNode = currentNode.getLeftChild();  
 else if (isLeftChild)  
 parentNode.setLeftChild(currentNode.getLeftChild());  
 else  
 parentNode.setRightChild(currentNode.getLeftChild());  
 }  
 else if (currentNode.getLeftChild() == null) { // узел заменяется правым поддеревом, если левого потомка нет  
 if (currentNode == rootNode)  
 rootNode = currentNode.getRightChild();  
 else if (isLeftChild)  
 parentNode.setLeftChild(currentNode.getRightChild());  
 else  
 parentNode.setRightChild(currentNode.getRightChild());  
 }  
 else { // если есть два потомка, узел заменяется преемником  
 Node heir = receiveHeir(currentNode);// поиск преемника для удаляемого узла  
 if (currentNode == rootNode)  
 rootNode = heir;  
 else if (isLeftChild)  
 parentNode.setLeftChild(heir);  
 else  
 parentNode.setRightChild(heir);  
 }  
 return true; // элемент успешно удалён  
 }  
  
 // метод возвращает узел со следующим значением после передаваемого аргументом.  
 // для этого он сначала переходим к правому потомку, а затем  
 // отслеживаем цепочку левых потомков этого узла.  
 private Node receiveHeir(Node node) {  
 Node parentNode = node;  
 Node heirNode = node;  
 Node currentNode = node.getRightChild(); // Переход к правому потомку  
 while (currentNode != null) // Пока остаются левые потомки  
 {  
 parentNode = heirNode;// потомка задаём как текущий узел  
 heirNode = currentNode;  
 currentNode = currentNode.getLeftChild(); // переход к левому потомку  
 }  
 // Если преемник не является  
 if (heirNode != node.getRightChild()) // правым потомком,  
 { // создать связи между узлами  
 parentNode.setLeftChild(heirNode.getRightChild());  
 heirNode.setRightChild(node.getRightChild());  
 }  
 return heirNode;// возвращаем приемника  
 }  
  
 public void printInOrder(){ // Семитричний обход  
 inOrder(rootNode);  
 }  
  
 void inOrder(Node current) { // рекурсивно обходим дерево  
 if (current != null) {  
 inOrder(current.getLeftChild());  
 System.*out*.println(current.getValue() + " "); // Здесь может быть все, что угодно  
 inOrder(current.getRightChild());  
 }  
 }  
  
 public void printTree() { // метод для вывода дерева в консоль  
 Stack globalStack = new Stack(); // общий стек для значений дерева  
 globalStack.push(rootNode);  
 int gaps = 32; // начальное значение расстояния между элементами 32  
 boolean isRowEmpty = false;  
 String separator = "-----------------------------------------------------------------";  
 System.*out*.println(separator);// черта для указания начала нового дерева  
 while (isRowEmpty == false) {  
 Stack localStack = new Stack(); // локальный стек для задания потомков элемента  
 isRowEmpty = true;  
  
 for (int j = 0; j < gaps; j++)  
 System.*out*.print(' ');  
 while (globalStack.isEmpty() == false) { // покуда в общем стеке есть элементы  
 Node temp = (Node) globalStack.pop(); // берем следующий, при этом удаляя его из стека  
 if (temp != null) {  
 System.*out*.print(temp.getValue()); // выводим его значение в консоли  
 localStack.push(temp.getLeftChild()); // соохраняем в локальный стек, наследники текущего элемента  
 localStack.push(temp.getRightChild());  
 if (temp.getLeftChild() != null ||  
 temp.getRightChild() != null)  
 isRowEmpty = false;  
 }  
 else {  
 System.*out*.print(" ");// - если элемент пустой "\_\_"  
 localStack.push(null);  
 localStack.push(null);  
 }  
 for (int j = 0; j < gaps \* 2 - 2; j++)  
 System.*out*.print(' ');  
 }  
 System.*out*.println();  
 gaps /= 2;// при переходе на следующий уровень расстояние между элементами каждый раз уменьшается  
 while (localStack.isEmpty() == false)  
 globalStack.push(localStack.pop()); // перемещаем все элементы из локального стека в глобальный  
 }  
 System.*out*.println(separator);// подводим черту  
 }  
}

Main.java

package com.company;  
  
import java.util.Scanner;  
  
public class Main {  
  
 public static void main(String[] args) {  
 int menu;  
 BinaryTree tree = new BinaryTree();  
 do{  
 System.*out*.println("1. Добавление");  
 System.*out*.println("2. Вивод");  
 System.*out*.println("3. Поиск");  
 System.*out*.println("4. Удаление");  
 System.*out*.println("5. Симметричный обход");  
 System.*out*.println("6. Выход");  
 Scanner scanner = new Scanner(System.*in*);  
 menu = scanner.nextInt();  
  
 switch (menu){  
 case 1:  
 tree.insertNode(23);  
 tree.insertNode(45);  
 tree.insertNode(90);  
 tree.insertNode(10);  
 tree.insertNode(53);  
 tree.insertNode(65);  
 //break;  
 case 2:  
 tree.printTree();  
 break;  
 case 3:  
 Node foundNode = tree.findNodeByValue(90);  
 foundNode.printNode();  
 break;  
 case 4:  
 tree.deleteNode(90);  
 tree.printTree();  
 break;  
 case 5:  
 tree.printInOrder();  
 break;  
 default:  
 System.*out*.println("Неверний номер пункта меню!");  
 }  
  
 }while (menu != 6);  
 }  
}

Висновок: Дерево - одна з найпоширеніших структур даних в програмуванні. Воно являє собою деревоподібну структуру у вигляді кореня і пов'язаних вузлів, з'єднаних ребрами.

Такі структури як дерева мають переваги як масивів, так і списків. Вставка і видалення в деревах так само швидко як і в списках, а пошук працює як в масивах. Тому дерева заслужили повагу і широке застосування в інформаційних системах.