НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«Московский энергетический институт»
Кафедра математического и компьютерного моделирования

«Технологии программирования»

Лабораторная работа
Вариант №15

Выполнил: Сошников С. А.
Группа: А-16-19

Преподаватель: Князев А.В.

Задание на лабораторную работу

Общее:

1) Составить на языке Java описания классов для указанных объектов.

2) Разработать консольную программу, иллюстрирующую использование объектов заданных классов.

3) Отчёт по лабораторной работе должен содержать:

- Титульный лист
- Задание на работу (общее и индивидуальное)
- Описание работы программы
- Алгоритмы выполнения основных операций на псевдокоде
- Тесты
- Распечатки экранов при работе программы
- Листинг программы

Индивидуальное:

15	Сошников С.А.	Упорядоченное множество целых чисел на основе бинарного дерева (включение, объединение).

Описание работы программы

Программа включает в себя два класса: класс узла дерева и класс самого дерева

Класс узла дерева содержит информацию (целое число), а также ссылку на левый и на правый лист (узел)

Класс дерева содержит:

1) информацию о корне дерева
2) метод для вывода дерева (множества) – выводит множество в том виде, в котором оно хранится в дереве (т.е. упорядоченное)
3) включения элемента во множество (добавления в дерево) – если элемент отсутствует во множестве, то добавляет элемент в дерево, соблюдая порядок.
4) метод для проверки на наличие элемента во множестве – обходит дерев, пока не наткнётся на нужный элемент
5) поиска минимума в дерево с заданным корнем – уходит в самое левое поддерево и возвращает оттуда элемент
6) метод для удаления элемента из множества (дерева) - при рекурсивном удалении узла из бинарного дерева нужно рассмотреть три случая: удаляемый элемент находится в левом поддереве текущего поддерева, удаляемый элемент находится в правом поддереве или удаляемый элемент находится в корне. В двух первых случаях нужно рекурсивно удалить элемент из нужного поддерева. Если удаляемый элемент находится в корне текущего поддерева и имеет два дочерних узла, то нужно заменить его минимальным элементом из правого поддерева и рекурсивно удалить **этот** минимальный элемент из правого поддерева. Иначе, если удаляемый элемент имеет один дочерний узел, нужно заменить его потомком.
7) метод для объединения с другим множеством – обходит объединяемое дерево, попутно добавляя из него элементы в объединяющее дерево через метод включения во множество

Описание алгоритмов на псевдокоде

Вывод дерева: если корень null, то выводится соотв. сообщение в консоль, если левое поддерево не null, рекурсивно вызываем алгоритм для левого дерева, после этого выводим элемент, затем если правое поддерево не null, рекурсивно вызываем алгоритм для правого поддерева
Включение: если включаемый элемент уже в дереве, выбрасывается исключение, если корень null, возвращаем узел с включаемым элементом, если включаемый элемент меньше того, что в корне, рекурсивно переходим влево, иначе вправо. В конце возвращаем корень
Проверка на наличие: если корень null, то false, если в корне искомый элемент, то true, если искомый меньше того, что в корне – переходим рекурсивно влево, иначе – вправо
Поиск минимума – если левое поддерево не null, переходим рекурсивно влево, иначе возвращаем элемент корня
Удаление – если корень null, то возврат null, если искомое меньше того, что в корне – рекурсивно уходим влево, иначе вправо, иначе если ни левое, ни правое не null, из правого поддерева достаём минимум, ставим его на место удалённого корня, а в правом поддереве его удаляем
иначе
если левое поддерево не нул, ставим корень левого поддерево на место корня, иначе если правое поддерево не нул, ставим корень правого поддерево на место корня, иначе корень просто удаляем
Объедининие: если корень объединяемого дерево нул, то вывод соотв. сообщения в консоль и завершение работы метода, иначе обход объединяемого дерева с переносом элементов через метод включения в объединяющее дерево

Тесты и распечатки экрана

```
Создали пустое дерево, выведем его
Tree is empty!
Заполним дерево
Введите кол-во элементов для включения
8
Вводите элементы
5
7
1
3
9
2
4
0
Ваше дерево
0
1
2
3
4
5
7
9
Введите кол-во элементов для удаления
```

```
Введите кол-во элементов для удаления
3
Вводите элементы
2
5
7
Ваше дерево
0
1
3
4
9
Протестируем объединение: введите кол-во элементов для второго дерева
6
Вводите элементы
1
2
3
4
5
6
Ваше второе дерево
1
2
3
4
5
6
Объединеяем 1 со 2-ым
Результат объединения
0
1
2
3
4
5
6
9
Программа протестирована
```

```
Создали пустое дерево, выведем его
Tree is empty!
Заполним дерево
Введите кол-во элементов для включения
10
Вводите элементы
-5
-3
0
15
512
3
0
512
-9
-11
Ваше дерево
-11
-9
-5
-3
0
3
15
512
```

```
Введите кол-во элементов для удаления
5
Вводите элементы
-11
-5
0
15
512
Ваше дерево
-9
-3
3
```

```
Протестируем объединение: введите кол-во элементов для второго дерева
5
Вводите элементы
-9
-3
3
5
10
Ваше второе дерево
-9
-3
3
5
10
Объединеяем 1 со 2-ым
Результат объединения
-9
-3
3
5
10
Программа протестирована
```

Листинг программы

```
package com.knyazev;
import java.util.Scanner;


class Node
{
    private int info;
    public Node left = null;
    public Node right = null;
    public Node(int new_info)
    {
        info = new_info;
    }
    public int getInfo()
    {
        return info;
    }
    public void setInfo(int new_info)
    {
        info = new_info;
    }
}

class Tree
{
    public Node root = null;
    public void output(Node start)
    {
        if (this.root == null)
        {
            System.out.println("Tree is empty!");
            return;
        }
        if (start.left != null)
            output(start.left);
        System.out.println(start.getInfo());
        if (start.right != null)
            output(start.right);
    }
    public Node add(Node start, int new_info) throws Exception
    {
        if (check(this.root, new_info))
            throw new Exception("The element is already in set");
        if (start == null)
        {
            return new Node(new_info);
        }
        else if (new_info < start.getInfo())
            start.left = add(start.left, new_info);
        else if (new_info > start.getInfo())
            start.right = add(start.right, new_info);
        return start;
    }
    public boolean check(Node start, int need_info)
    {
        if (start == null)
            return false;
        if (start.getInfo() == need_info)
            return true;
        if (need_info < start.getInfo())
            return check(start.left, need_info);
        else
```

```
            return check(start.right, need_info);
    }
    public Node min(Node start)
    {
        if (start.left == null)
            return start;
        return min(start.left);
    }
    public Node delete(Node start, int need_info)
    {
        if (start == null)
            return null;
        if (need_info < start.getInfo())
            start.left = delete(start.left, need_info);
        else if (need_info > start.getInfo())
            start.right = delete(start.right, need_info);
        else if (start.left != null && start.right != null)
        {
            start.setInfo(min(start.right).getInfo());
            start.right = delete(start.right, start.getInfo());
        }
        else
        {
            if (start.left != null)
                start = start.left;
            else if (start.right != null)
                start = start.right;
            else
                start = null;
        }
        return start;
    }
    public void union(Node another_tree_node)
    {
        if (another_tree_node == null)
        {
            System.out.println("Another tree is empty");
            return;
        }
        if (another_tree_node.left != null)
            union(another_tree_node.left);
        try
        {
            this.root = add(this.root, another_tree_node.getInfo());
        }
        catch (Exception a) {}
        if (another_tree_node.right != null)
            union(another_tree_node.right);
    }
}

public class Main
{

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        Tree test = new Tree();
        System.out.println("Создали пустое дерево, выведем его");
        test.output(test.root);
        System.out.println("Заполним дерево");
        System.out.println("Введите кол-во элементов для включения");
        int k = in.nextInt();
        System.out.println("Вводите элементы");
```

```
        for (int i = 0; i < k; ++i)
        {
            int a = in.nextInt();
            try
            {
                test.root = test.add(test.root, a);
            }
            catch (Exception t) {}
        }
        System.out.println("Ваше дерево");
        test.output(test.root);
        System.out.println("Введите кол-во элементов для удаления");
        k = in.nextInt();
        System.out.println("Вводите элементы");
        for (int i = 0; i < k; ++i)
        {
            int a = in.nextInt();
            test.root = test.delete(test.root, a);
        }
        System.out.println("Ваше дерево");
        test.output(test.root);
        System.out.println("Протестируем объединение: введите кол-во
элементов для второго дерева");
        Tree test2 = new Tree();
        k = in.nextInt();
        System.out.println("Вводите элементы");
        for (int i = 0; i < k; ++i)
        {
            int a = in.nextInt();
            try
            {
                test2.root = test2.add(test2.root, a);
            }
            catch (Exception t) {}
        }
        System.out.println("Ваше второе дерево");
        test2.output(test2.root);
        System.out.println("Объединеяем 1 со 2-ым");
        test.union(test2.root);
        System.out.println("Результат объединения");
        test.output(test.root);
        System.out.println("Программа протестирована");
    }
}
```