Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №6

Мова програмування Java

Тема: Java Collections Framework.

Виконав
студент групи ІА-32:
Костінський М. М.

Перевірив:
Лесик В. О.

2025

Тема: Java Collections Framework.

Мета: практичне освоєння роботи з фреймворком колекцій в Java, доцільність вибору конкретних реалізацій та особливості їх роботи.

Хід роботи

Теорія

Червоно-чорне дерево - це різновид самобалансуючого двійкового дерева пошуку. Воно гарантує, що висота дерева залишається логарифмічною відносно кількості вузлів, що забезпечує виконання основних операцій за час O(log n) у найгіршому випадку. Основна ідея структури полягає в тому, що кожен вузол має додатковий атрибут - колір. Під час вставки або видалення вузлів дерево використовує правила, пов'язані з кольорами, для підтримання балансу.

Червоно-чорне дерево має задовольняти наступним п'яти властивостям

- Кожен вузол є або червоним, або чорним. Властивість кореня
- Корінь дерева завжди чорний.
- Усі листки (NIL-вузли, що представляють відсутність дочірнього елемента) є чорними.
- Якщо вузол червоний, то обидва його нащадки мають бути чорними (не може бути двох червоних вузлів поспіль на одному шляху).
- Для кожного вузла всі шляхи від нього до листків (NIL), що є його нащадками, містять однакову кількість чорних вузлів.

Ці властивості гарантують, що найдовший шлях від кореня до листка не більш ніж удвічі довший за найкоротший шлях. Це забезпечує балансування дерева.

При вставці або видаленні вузлів властивості дерева можуть порушуватися. Для відновлення балансу використовуються дві основні операції.

Зміна кольору вузла з червоного на чорний або навпаки. Це найпростіший спосіб відновити властивості, але він не впливає на структуру дерева.

Обертання — це локальна операція, яка змінює структуру дерева, зберігаючи при цьому властивість двійкового дерева пошуку. Обертання змінюють висоту піддерев і дозволяють підняти вузли або опустити для вирівнювання чорної висоти.

Новий вузол вставляється як у звичайне бінарне дерево пошуку на позицію листка. Новий вузол завжди фарбується в червоний колір. Це робиться для того, щоб не порушити властивість, оскільки додавання червоного вузла не змінює кількість чорних вузлів на шляху. Якщо батько нового вузла також червоний, виникає порушення властивості. Для виправлення розглядається колір "дядька" (вузла, суміжного з батьком). Випадок 1: Якщо дядько червоний, то виконується перефарбування батька, дядька та дідуся. Випадок 2: Якщо дядько чорний і новий вузол утворює трикутник, виконується обертання, щоб випрямити ланцюжок. Якщо дядько чорний і вузли утворюють пряму лінію

виконується обертання в протилежний бік та перефарбування батька і дідуся. Після всіх операцій корінь дерева примусово фарбується в чорний колір.

Код:

```
import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;

public class RedBlackTreeDemo {

    private static class Node {
        int data;
        Node parent;
        Node left;
        Node right;
        boolean color;

        public Node(int data) {
            this.data = data;
        }
    }

    private Node root;
    private Node TNULL;

    private final boolean RED = true;
    private final boolean BLACK = false;

    public RedBlackTreeDemo() {
        TNULL = new Node(0);
        TNULL.color = BLACK;
        TNULL.left = null;
        TNULL.right = null;
        root = TNULL;
    }


    private void leftRotate(Node x) {
        Node y = x.right;
        x.right = y.left;
        if (y.left != TNULL) {
            y.left.parent = x;
        }
        y.parent = x.parent;
        if (x.parent == null) {
            this.root = y;
        } else if (x == x.parent.left) {
            x.parent.left = y;
        } else {
            x.parent.right = y;
        }
        y.left = x;
        x.parent = y;
    }

    private void rightRotate(Node x) {
        Node y = x.left;
        x.left = y.right;
        if (y.right != TNULL) {
            y.right.parent = x;
        }
        y.parent = x.parent;
        if (x.parent == null) {
            this.root = y;
```

```
        } else if (x == x.parent.right) {
            x.parent.right = y;
        } else {
            x.parent.left = y;
        }
        y.right = x;
        x.parent = y;
    }

    private void fixInsert(Node k) {
        Node u;
        while (k.parent.color == RED) {
            if (k.parent == k.parent.parent.right) {
                u = k.parent.parent.left;
                if (u.color == RED) {
                    u.color = BLACK;
                    k.parent.color = BLACK;
                    k.parent.parent.color = RED;
                    k = k.parent.parent;
                } else {
                    if (k == k.parent.left) {
                        k = k.parent;
                        rightRotate(k);
                    }
                    k.parent.color = BLACK;
                    k.parent.parent.color = RED;
                    leftRotate(k.parent.parent);
                }
            } else {
                u = k.parent.parent.right;
                if (u.color == RED) {
                    u.color = BLACK;
                    k.parent.color = BLACK;
                    k.parent.parent.color = RED;
                    k = k.parent.parent;
                } else {
                    if (k == k.parent.right) {
                        k = k.parent;
                        leftRotate(k);
                    }
                    k.parent.color = BLACK;
                    k.parent.parent.color = RED;
                    rightRotate(k.parent.parent);
                }
            }
            if (k == root) {
                break;
            }
        }
        root.color = BLACK;
    }

    public void insert(int key) {
        Node node = new Node(key);
        node.parent = null;
        node.data = key;
        node.left = TNULL;
        node.right = TNULL;
        node.color = RED;

        Node y = null;
        Node x = this.root;

        while (x != TNULL) {
            y = x;
            if (node.data < x.data) {
```

```
                x = x.left;
            } else {
                x = x.right;
            }
        }

        node.parent = y;
        if (y == null) {
            root = node;
        } else if (node.data < y.data) {
            y.left = node;
        } else {
            y.right = node;
        }

        if (node.parent == null) {
            node.color = BLACK;
            return;
        }

        if (node.parent.parent == null) {
            return;
        }

        fixInsert(node);
    }

    public void printTree() {
        if (root == TNULL) {
            System.out.println("Дерево порожнє.");
            return;
        }
        printHelper(this.root, "", true);
    }

    private void printHelper(Node root, String indent, boolean last) {
        if (root != TNULL) {
            System.out.print(indent);
            if (last) {
                System.out.print("R----");
                indent += "   ";
            } else {
                System.out.print("L----");
                indent += "|  ";
            }

            String sColor = root.color == RED ? "RED" : "BLACK";
            System.out.println(root.data + "(" + sColor + ")");
            printHelper(root.left, indent, false);
            printHelper(root.right, indent, true);
        }
    }

    public void inorderTraversal() {
        inorderHelper(this.root);
        System.out.println();
    }

    private void inorderHelper(Node node) {
        if (node != TNULL) {
            inorderHelper(node.left);
            System.out.print(node.data + " ");
            inorderHelper(node.right);
        }
    }
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        RedBlackTreeDemo bst = new RedBlackTreeDemo();
        Random rand = new Random();

        System.out.println("=== Червоно-Чорне Дерево (Red-Black Tree) ===");

        while (true) {
            System.out.println("\nОберіть дію:");
            System.out.println("1. Заповнити випадковими числами (Random)");
            System.out.println("2. Заповнити впорядкованими числами (Sorted) -
демонстрація балансування");
            System.out.println("3. Додати число вручну");
            System.out.println("4. Показати дерево");
            System.out.println("5. Очистити дерево (створити нове)");
            System.out.println("0. Вихід");
            System.out.print("Ваш вибір: ");

            String choice = scanner.next();

            switch (choice) {
                case "1":
                    bst = new RedBlackTreeDemo();
                    System.out.print("Введіть кількість елементів: ");
                    int count = scanner.nextInt();
                    int[] randomArr = new int[count];
                    System.out.print("Масив додавання: ");
                    for (int i = 0; i < count; i++) {
                        randomArr[i] = rand.nextInt(100);
                        System.out.print(randomArr[i] + " ");
                        bst.insert(randomArr[i]);
                    }
                    System.out.println("\nГотово! Дерево побудовано.");
                    break;

                case "2":
                    bst = new RedBlackTreeDemo();
                    System.out.print("Введіть кількість елементів: ");
                    int sortedCount = scanner.nextInt();
                    int[] sortedArr = new int[sortedCount];
                    for (int i = 0; i < sortedCount; i++) {
                        sortedArr[i] = rand.nextInt(100);
                    }
                    Arrays.sort(sortedArr);
                    System.out.print("Додаємо в порядку зростання: ");
                    for (int num : sortedArr) {
                        System.out.print(num + " ");
                        bst.insert(num);
                    }
                    break;

                case "3":
                    System.out.print("Введіть ціле число: ");
                    if (scanner.hasNextInt()) {
                        int val = scanner.nextInt();
                        bst.insert(val);
                        System.out.println("Вузол " + val + " додано.");
                    } else {
                        System.out.println("Помилка вводу.");
                        scanner.next();
                    }
                    break;

                case "4":
                    System.out.println("\n--- Структура Дерева ---");
                    bst.printTree();
```

```
                    System.out.println("\n--- In-order обхід (перевірка
сортування) ---");
                    bst.inorderTraversal();
                    break;

                case "5":
                    bst = new RedBlackTreeDemo();
                    System.out.println("Дерево очищено.");
                    break;

                case "0":
                    System.out.println("Завершення роботи.");
                    return;

                default:
                    System.out.println("Невірний вибір. Спробуйте ще раз.");
            }
        }
    }
}
```

Приклад виконання коду:

```
Оберіть дію:
1. Заповнити випадковими числами (Random)
2. Заповнити впорядкованими числами (Sorted) - демонстрація балансування
3. Додати число вручну
4. Показати дерево
5. Очистити дерево (створити нове)
0. Вихід
Ваш вибір: 1
Введіть кількість елементів: 9
Масив додавання: 63 68 47 37 90 61 57 67 34
Готово! Дерево побудовано.
```

```
Оберіть дію:
1. Заповнити випадковими числами (Random)
2. Заповнити впорядкованими числами (Sorted) - демонстрація балансування
3. Додати число вручну
4. Показати дерево
5. Очистити дерево (створити нове)
0. Вихід
Ваш вибір: 4

--- Структура Дерева ---
R----63(BLACK)
   L----47(RED)
   |  L----37(BLACK)
   |  |  L----34(RED)
   |  R----61(BLACK)
   |     L----57(RED)
   R----68(BLACK)
      L----67(RED)
      R----90(RED)

--- In-order обхід (перевірка сортування) ---
34 37 47 57 61 63 67 68 90
```

Висновок: Під час виконання лабораторної роботи було вивчено архітектуру та принципи роботи Java Collections Framework. Було розроблено консольний додаток, який моделює роботу червоно-чорного дерева, реалізовані алгоритми додавання вузлів та автоматичного балансування структури за допомогою методів перефарбування та обертання, що забезпечує дотримання властивостей червоно-чорного дерева.