Федеральное государственное автономное

образовательное учреждение

Высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | Институт космических и информационных технологий | | институт | | Кафедра программной инженерии | | кафедра | |

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №4**

|  |
| --- |
| Аппликативы и алгоритмы |
| тема |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  |  | К. В. Богданов |
|  | |  | подпись, дата |  | инициалы, фамилия |
| Студент | КИ21-16/2Б, 032153882 |  |  |  | Д. А. Скоробогатов |
|  | номер группы, зачётной книжки |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2024

**СОДЕРЖАНИЕ**

[1 Цель 3](#_Toc157723631)

[2 Задачи 3](#_Toc157723632)

[3 Ход выполнения 4](#_Toc157723633)

[3.1 Написание скрипта 4](#_Toc157723634)

[4 Выводы 7](#_Toc157723635)

[Список использованных источников 8](#_Toc157723636)

# Цель

Имеется набор документов, представленных в виде массива одноуровневых JSON. Массив хранится в файле. Требуется:

1. по набору документов и заданному полю сформировать бинарное сбалансированное дерево поиска (поисковый индекс); в том случае, если в документе поле не представлено, он в индекс не попадает

2. сохранить индекс в отдельный файл в любом формате на ваше усмотрение; предусмотреть подгрузку индекса из файла

3. реализовать утилиту, производящую поиск по набору документов как с использованием индекса, так и без него (последовательным перебором); условие поиска - строгое соответствие; утилита должна выводить найденные документы (если таковые есть), а также количество операций сравнения, которые понадобились для их обнаружения

4. ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ: предусмотреть возможность корректировки поискового индекса при добавлении или удалении документа (без полного переиндексирования)

# Задачи

* Познакомиться с работой с JSON в Scala;
* Написать скрипт для своего варианта;
* Написать отчёт.

# Ход выполнения

## Написание скрипта

Листинг 1 – Исходный код программы

import masks.JsonObject

import scala.annotation.tailrec

case class AVLNode(var key: Int,

var left: Option[AVLNode] = None,

var right: Option[AVLNode] = None,

var parent: Option[AVLNode] = None,

var balance: Int = 0,

value: JsonObject = JsonObject())

class AVLTree(var root: Option[AVLNode] = None) {

private def treeRotateR(x: AVLNode): Unit = {

val y: Option[AVLNode] = x.left

x.left = y.get.right

if (y.get.right.isDefined) {

y.get.right.get.parent = Some(x)

}

y.get.parent = x.parent

if (x.parent.isEmpty) {

root = y

} else if (x.parent.get.right.contains(x)) {

x.parent.get.right = y

} else {

x.parent.get.left = y

}

y.get.right = Some(x)

x.parent = y

}

private def treeRotateL(x: AVLNode): Unit = {

val y: Option[AVLNode] = x.right

x.right = y.get.left

if (y.get.left.isDefined) {

y.get.left.get.parent = Some(x)

}

y.get.parent = x.parent

if (x.parent.isEmpty) {

root = y

} else if (x.parent.get.left.contains(x)) {

x.parent.get.left = y

} else {

x.parent.get.right = y

}

y.get.left = Some(x)

x.parent = y

}

private def treeRotateLR(x: AVLNode): Unit = {

treeRotateL(x.left.get)

treeRotateR(x)

}

Окончание листинга 1

private def treeRotateRL(x: AVLNode): Unit = {

treeRotateR(x.right.get)

treeRotateL(x)

}

private def avlRestoreBalance(x: AVLNode): Unit = {

x.balance = treeHeight(x.right) - treeHeight(x.left)

if (x.balance < -1) {

if (treeHeight(x.left.get.left) > treeHeight(x.left.get.right)) {

treeRotateR(x)

} else {

treeRotateLR(x)

}

}

if (x.balance > 1) {

if (treeHeight(x.right.get.right) > treeHeight(x.right.get.left)) {

treeRotateL(x)

} else {

treeRotateRL(x)

}

}

}

def insert(key: Int, value: JsonObject = JsonObject()): Unit = {

AVLInsert(AVLNode(key, value = value))

}

private def AVLInsert(x: AVLNode): Unit = {

treeInsert(x)

var current: Option[AVLNode] = Some(x)

while (current.isDefined) {

avlRestoreBalance(current.get)

current = current.get.parent

}

}

private def treeInsert(z: AVLNode): Unit = {

val y = treeSearchInexact(z.key)

z.parent = y

if (y.isEmpty) {

root = Some(z)

} else if (z.key < y.get.key) {

y.get.left = Some(z)

} else if (z.key > y.get.key) {

y.get.right = Some(z)

}

}

def searchWithSteps(key: Int): (Option[AVLNode], Int) = {

@tailrec

def searchNode(node: Option[AVLNode], key: Int, steps: Int): (Option[AVLNode], Int) = {

node match {

case Some(n) =>

if (key == n.key) (Some(n), steps)

else if (key < n.key) searchNode(n.left, key, steps + 1)

else searchNode(n.right, key, steps + 1)

case None => (None, steps)

Окончание листинга 1

}

}

searchNode(root, key, 0)

}

private def treeSearchInexact(k: Int): Option[AVLNode] = {

var y: Option[AVLNode] = None

var x = root

while (x.isDefined && k != x.get.key) {

y = x

if (k < x.get.key) {

x = x.get.left

} else {

x = x.get.right

}

}

if (x.isDefined) {

y = x

}

y

}

private def treeHeight(node: Option[AVLNode]): Int = {

node match {

case Some(n) => 1 + Math.max(treeHeight(n.left), treeHeight(n.right))

case None => 0

}

}

def displayTree(): Unit = {

def displayNode(node: Option[AVLNode], level: Int): Unit = {

if (node.isDefined) {

displayNode(node.get.right, level + 1)

println(" " \* level + node.get.key)

displayNode(node.get.left, level + 1)

}

}

displayNode(root, 0)

}

def countNodes(): Int = {

def countNodesRec(node: Option[AVLNode]): Int = {

node match {

case Some(n) => 1 + countNodesRec(n.left) + countNodesRec(n.right)

case None => 0

}

}

countNodesRec(root)

}

}

Пример выполнения скрипта представлен на рисунке 1.

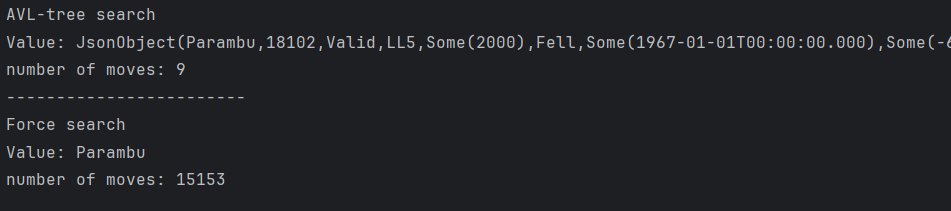


Рисунок 1 – Пример выполнения скрипта

# Выводы

В ходе выполнения практической работы:

* Познакомиться с работой с JSON в Scala;
* Написать скрипт для своего варианта;
* Написан отчёт.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. СТО 7.5-07-2021 Система менеджмента качества. Общие требования построению, изложению и оформлению документов учебной деятельности. – Взамен СТО 4.2-07-2014 ; введ. 07.12.2021. – Красноярск : СФУ 2021. – 61 с.