Федеральное государственное автономное

образовательное учреждение

Высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт космических и информационных технологий |
| институт |
| Программная инженерия |
| кафедра |

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  |  | Н. А. Самарин |
|  | |  | подпись, дата |  | инициалы, фамилия |
| Студент | Студент КИ21-17/1Б, |  |  |  | Н. А. Самарин |
|  | номер группы, зачётной книжки |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2024

**СОДЕРЖАНИЕ**

1 Задание............................................................................................................... 3

2 Исходный код разработанного алгоритма..................................................... 3

3 Результат........................................................................................................... 7

**1 Задание**

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

**2 Исходный код разработанного алгоритма**

Листинг 1 – Main.scala

import org.json4s.jackson.JsonMethods.parse  
import org.json4s.{DefaultFormats, JObject}  
import java.io.{FileNotFoundException, NotSerializableException}  
import scala.io.Source  
object Main {  
 def main(args: Array[String]): Unit = {  
 var jsonObjects: Option[List[MapNode]] = None  
 var tree: Option[Tree] = None  
 var continue = true  
 while (continue) {  
 println("1. Загрузить JSON из файла")  
 println("2. Создать дерево по полю")  
 println("3. Выполнить поиск по дереву")  
 println("4. Выполнить линейный поиск")  
 println("5. Сохранить индекс в файл")  
 println("6. Загрузить индекс из файла")  
 println("7. Выход")  
 var choice: Int = -1  
 try {  
 choice = scala.io.StdIn.readInt()  
 } catch {  
 case e: NumberFormatException =>  
 }  
 choice match {  
 case 1 =>  
 println("Введите путь до файла:")  
 val filePath = scala.io.StdIn.readLine()  
 jsonObjects = Some(readJsonFile(filePath))  
 case 2=>  
 if (jsonObjects.isDefined) {  
 println("Введите поле:")  
 val field = scala.io.StdIn.readLine()  
 tree = Some(Tree.makeFrom(jsonObjects.get, field))  
 println(s"Высота: ${tree.get.height}")  
 } else println("Сначала загрузите json")  
 case 3 =>  
 if (tree.isDefined) {

Продолжение листинга 1

println("Введите значение поля:")  
 val value = scala.io.StdIn.readLine()  
 val result = tree.get.find(value)  
 println(s"Результат: $result")  
 println(s"Количество: ${result.length}")  
 println(s"Сравнений: ${tree.get.comparisonCount}")  
 } else println("Сначала постройте дерево")  
 case 4 =>  
 if (jsonObjects.isDefined) {  
 println("Введите поле:")  
 val field = scala.io.StdIn.readLine()  
 println("Введите значение поля:")  
 val value = scala.io.StdIn.readLine()  
 val linearSearch = new LinearSearch(jsonObjects.get, field)  
 val result = linearSearch.find(value)  
 println(s"Результат: $result")  
 println(s"Количество: ${result.length}")  
 println(s"Сравнений: ${linearSearch.comparisonCount}")  
 } else println("Сначала загрузите json")  
 case 5=>  
 if (tree.isDefined) {  
 println("Введите путь до файла:")  
 val filePath = scala.io.StdIn.readLine()  
 Tree.saveIndex(tree.get, filePath)  
 } else println("Сначала постройте дерево")  
 case 6 =>  
 println("Введите путь до файла:")  
 val filePath = scala.io.StdIn.readLine()  
 try {  
 tree = Some(Tree.loadIndex(filePath))  
 println("Загружено дерево")  
 println(s"Индекс построен по ${tree.get.key}")  
 println(s"Высота дерева: ${tree.get.height}")  
 } catch {  
 case e: NotSerializableException => println(e)  
 case e: FileNotFoundException => println(e)  
 }  
 case 7 =>  
 continue = false  
 case \_ =>  
 println("Некорректный ввод")  
 }  
 }  
 }  
 def readJsonFile(filePath: String): List[MapNode] = {  
 val source = Source.fromFile(filePath)  
 val jsonString = try {  
 source.mkString  
 } finally {  
 source.close()  
 }  
 implicit val formats: DefaultFormats.type = DefaultFormats  
 val json = parse(jsonString)  
 val objects: List[JObject] = json.extract[List[JObject]]  
 objects.map { obj =>  
 val dataMap = obj.values  
 MapNode(dataMap)  
 }

Окончание листинга 1

}  
}

Листинг 2 – LinearSearch.scala

class LinearSearch(  
 mapNodes: List[MapNode],  
 keyToIndex: String  
 ) {  
 var comparisonCount: Integer = 0  
 def find(value: Any): List[MapNode] = {  
 comparisonCount = 0  
 var res = List.empty[MapNode]  
 for (node <- mapNodes) {  
 if (node.contains(keyToIndex)) {  
 comparisonCount += 1  
 if (node.get(keyToIndex) == value) {  
 res = res.appended(node)  
 }  
 }  
 }  
 res  
 }  
}

Листинг 3 – Tree.scala

import java.io.{FileInputStream, FileOutputStream, NotSerializableException, Obj  
ectInputStream, ObjectOutputStream}  
class Tree private(  
 var root: Node,  
 var comparisonCount: Integer,  
 keyToIndex: String  
 ) extends Serializable {  
 def key: String = {  
 keyToIndex  
 }  
 def height: Integer = {  
 Node.height(root)  
 }  
 def find(value: Any): List[MapNode] = {  
 comparisonCount = 0  
 find(root, value).map(\_.data)  
 }  
 private def find(node: Node, value: Any, foundNodes: List[Node] = List.empty):  
 List[Node] = {  
 if (node == null) foundNodes  
 else {  
 val nodeValue = node.data.get(keyToIndex)  
 comparisonCount += 1  
 if (value == nodeValue) {  
 val leftFound = find(node.left, value, foundNodes.appended(node))  
 find(node.right, value, leftFound)  
 } else if (value.toString < nodeValue.toString) {  
 find(node.left, value, foundNodes)  
 } else {  
 find(node.right, value, foundNodes)  
 }  
 }

Окончание листинга 3

}  
}  
object Tree {  
 def makeFrom(mapNodes: List[MapNode], key: String): Tree = {  
 var root: Node = null  
 for (jnode <- mapNodes) {  
 if (jnode.contains(key)) {  
 root = Node.insert(root, jnode, jsonNode => jsonNode.get(key).toString)  
 }  
 }  
 new Tree(root, 0, key)  
 }  
 def saveIndex(index: Tree, filePath: String): Unit = {  
 val fileOutputStream = new FileOutputStream(filePath)  
 val objectOutputStream = new ObjectOutputStream(fileOutputStream)  
 objectOutputStream.writeObject(index)  
 objectOutputStream.close()  
 fileOutputStream.close()  
 }  
 def loadIndex(filePath: String): Tree = {  
 val fileInputStream = new FileInputStream(filePath)  
 val objectInputStream = new ObjectInputStream(fileInputStream)  
 val tree = objectInputStream.readObject()  
 objectInputStream.close()  
 fileInputStream.close()  
 tree match {  
 case tree: Tree => tree  
 case \_ => throw new NotSerializableException("Из данных в файле невозможно  
 создать дерево(")  
 }  
 }  
}

Листинг 4 – Node.scala

class Node(  
 var data: MapNode,  
 private var height: Integer = 1,  
 var left: Node = null,  
 var right: Node = null  
 ) extends Serializable {  
 def balanceFactor: Integer = {  
 Node.height(right) - Node.height(left)  
 }  
 def fixHeight(): Unit = {  
 height = Math.max(Node.height(left), Node.height(right)) + 1  
 }  
 override def toString: String = data.toString  
}  
object Node {  
 def height(node: Node): Integer = {  
 if (node != null) node.height else 0  
 }  
 private def rotateRight(p: Node): Node = {  
 val q = p.left  
 p.left = q.right  
 q.right = p  
 p.fixHeight()  
 q.fixHeight()

Окончание листинга 4

q  
 }  
 private def rotateLeft(q: Node): Node = {  
 val p = q.right  
 q.right = p.left  
 p.left = q  
 q.fixHeight()  
 p.fixHeight()  
 p  
 }  
 private def balance(p: Node): Node = {  
 p.fixHeight()  
 if (p.balanceFactor == 2) {  
 if (p.right.balanceFactor < 0) {  
 p.right = rotateRight(p.right)  
 }  
 return rotateLeft(p)  
 }  
 if (p.balanceFactor == -2) {  
 if (p.left.balanceFactor > 0) {  
 p.left = rotateLeft(p.left)  
 }  
 return rotateRight(p)  
 }  
 p  
 }  
 def insert(p: Node, value: MapNode, key: MapNode => String): Node = {  
 if (p == null) return new Node(value)  
 if (key(value) < key(p.data)) {  
 p.left = insert(p.left, value, key)  
 } else {  
 p.right = insert(p.right, value, key)  
 }  
 balance(p)  
 }  
}

Листинг 5 – MapNode.scala

case class MapNode(jdata: Map[String, Any]) extends Serializable {  
 def get(key: String): Any = {  
 jdata.getOrElse(key, throw new NoSuchElementException(s"No key = $key"))  
 }  
 def contains(key: String): Boolean = {  
 jdata.contains(key)  
 }  
 override def toString: String = jdata.toString()  
}

**3 Результат**

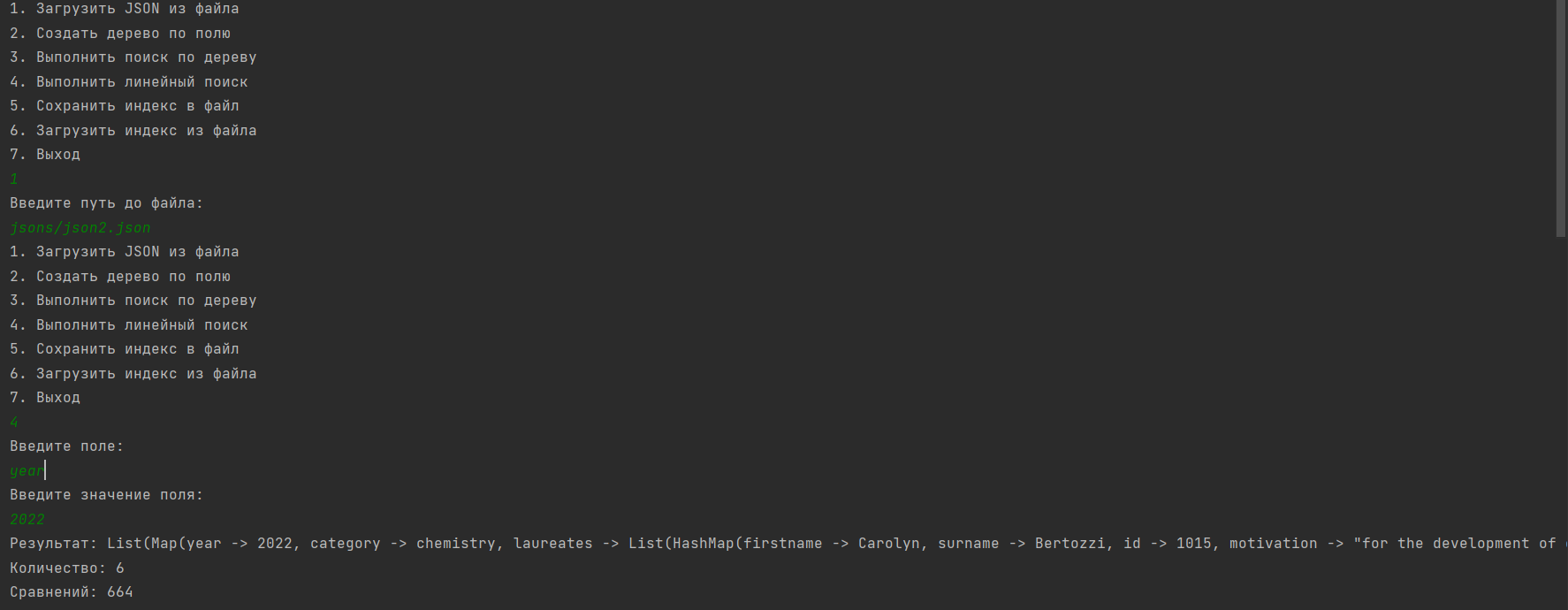


Рисунок 1 – Результат работы алгоритма 1

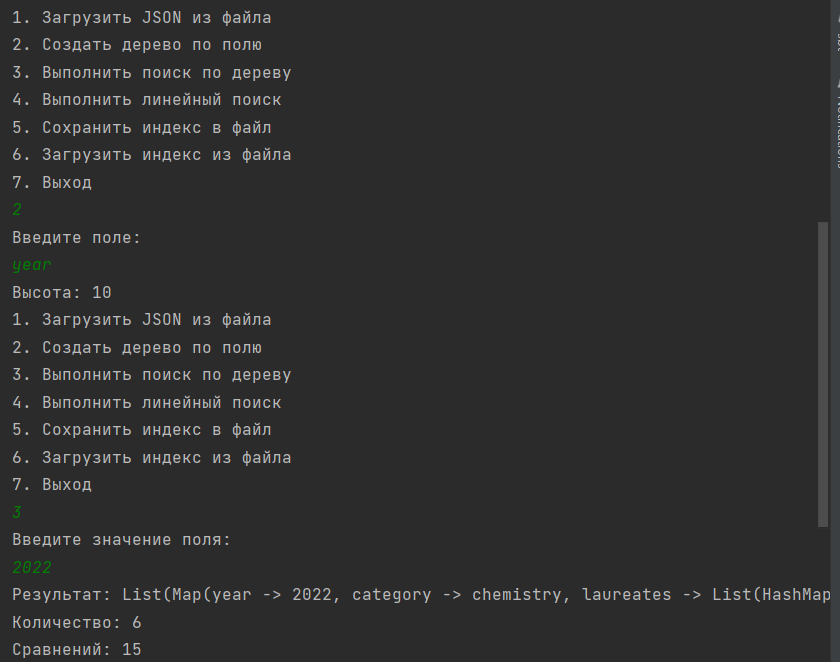


Рисунок 2 – Результат работы алгоритма 2

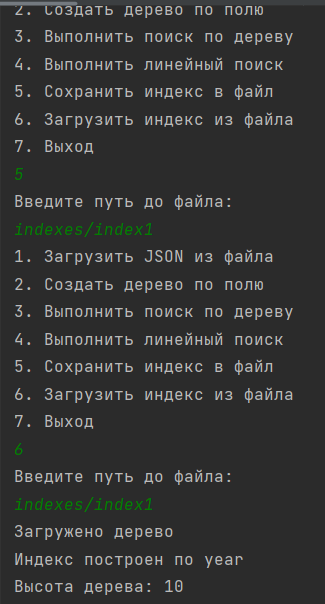


Рисунок 3 – Результат работы алгоритма 3