

**Kauno technologijos universitetas**

Informatikos fakultetas

BST ir AVL medžiai

P175B014 Duomenų struktūrų antras laboratorinis darbas

**Projekto autorius**

Gustas Klevinskas

**Akademinė grupė**

IFF-8/7

Kaunas, 2019

Turinys

[Savo sukurtos klasės testavimas 3](#_Toc24598587)

[BST metodai 4](#_Toc24598588)

[Medžio aukštis 5](#_Toc24598589)

[BstSet ir AvlSet greitaveika 6](#_Toc24598590)

[add() 6](#_Toc24598591)

[contains() 7](#_Toc24598592)

[TreeSet ir HashSet greitaveika 8](#_Toc24598593)

[BstSet metodai 10](#_Toc24598594)

[headSet() 10](#_Toc24598595)

[subSet() 10](#_Toc24598596)

[tailSet() 11](#_Toc24598597)

[Iteratoriaus remove() 11](#_Toc24598598)

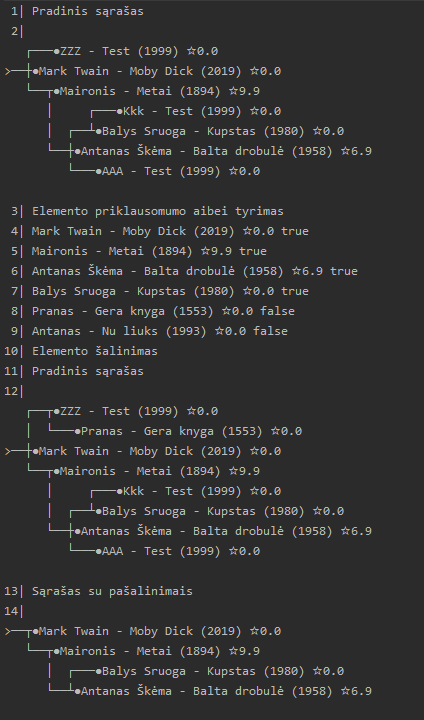
[AvlSet remove() 11](#_Toc24598599)

# Savo sukurtos klasės testavimas

Kodas:

private static void generateBooks() {  
 Book b1 = new Book("Mark Twain", "Moby Dick", 2019);  
 Book b2 = new Book("Maironis", "Metai", 1894);  
 Book b3 = new Book("Antanas Škėma", "Balta drobulė", 1958);  
 Book b4 = new Book("Balys Sruoga", "Kupstas", 1980);  
  
 b2.setRating(9.9f);  
 b3.setRating(6.9f);  
  
 *books* = new Book[]{b1, b2, b3, b4};  
  
 *booksBstSet*.add(b1);  
 *booksBstSet*.add(b2);  
 *booksBstSet*.add(b3);  
 *booksBstSet*.add(b4);  
 *booksBstSet*.add(new Book("ZZZ", "Test", 1999));  
 *booksBstSet*.add(new Book("AAA", "Test", 1999));  
 *booksBstSet*.add(new Book("Kkk", "Test", 1999));  
}  
  
private static void executeTest() {  
 *generateBooks*();  
  
 // Initial set  
 Ks.*oun*("Pradinis sąrašas");  
 Ks.*oun*("\n" + *booksBstSet*.toVisualizedString(""));  
  
 // Check if contains  
 Ks.*oun*("Elemento priklausomumo aibei tyrimas");  
 for (Book i : *books*)  
 Ks.*oun*(i + " " + *booksBstSet*.contains(i));  
 Book b1 = new Book("Pranas", "Gera knyga", 1553);  
 Book b2 = new Book("Antanas", "Nu liuks", 1993);  
 Ks.*oun*(b1 + " " + *booksBstSet*.contains(b1));  
 Ks.*oun*(b2 + " " + *booksBstSet*.contains(b2));  
 *booksBstSet*.add(b1);  
  
 // Delete from set  
 Ks.*oun*("Elemento šalinimas");  
 Ks.*oun*("Pradinis sąrašas");  
 Ks.*oun*("\n" + *booksBstSet*.toVisualizedString(""));  
 *booksBstSet*.remove(b1);  
 *booksBstSet*.remove(new Book("Maironis", "Metai", 1894));  
 *booksBstSet*.remove(new Book("ZZZ", "Test", 1999));  
 *booksBstSet*.remove(new Book("AAA", "Test", 1999));  
 *booksBstSet*.remove(new Book("Kkk", "Test", 1999));  
 Ks.*oun*("Sąrašas su pašalinimais");  
 Ks.*oun*("\n" + *booksBstSet*.toVisualizedString(""));  
}

Rezultatai:



# BST metodai

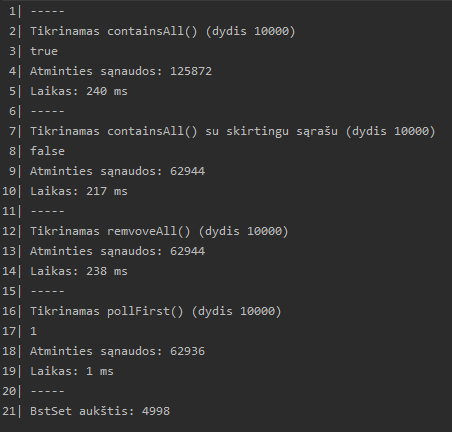
Kodas:

public boolean containsAll(BstSet<?> c) {  
 Iterator i = c.iterator();  
  
 while (i.hasNext()) {  
 if (!contains((E) i.next()))  
 return false;  
 }  
  
 return true;  
}

public void removeAll(BstSet<?> c) {  
 Iterator i = c.iterator();  
  
 while (i.hasNext()) {  
 remove((E) i.next());  
 }  
}

public E pollFirst() {  
 if (size == 0)  
 return null;  
  
 BstNode<E> node = getMin(root);  
 remove(node.element);  
 return node.element;  
}

Rezultatai:

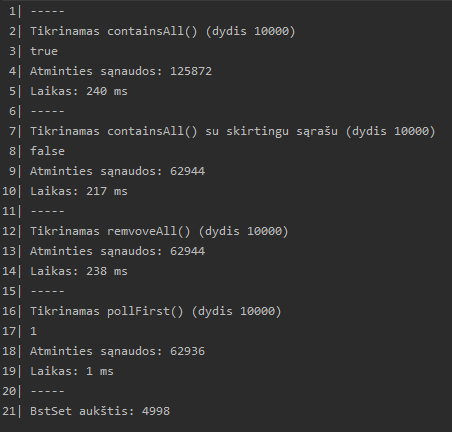


# Medžio aukštis

Kodas:

public int getHeight() {  
 return getHeightRecursive(root);  
}  
  
private int getHeightRecursive(BstNode<E> node) {  
 if (node == null)  
 return -1;  
  
 int leftHeight = getHeightRecursive(node.left);  
 int rightHeight = getHeightRecursive(node.right);  
  
 if (leftHeight > rightHeight)  
 return leftHeight + 1;  
 else  
 return rightHeight + 1;  
}

Rezultatai:



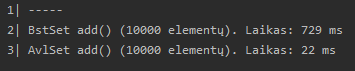
# BstSet ir AvlSet greitaveika

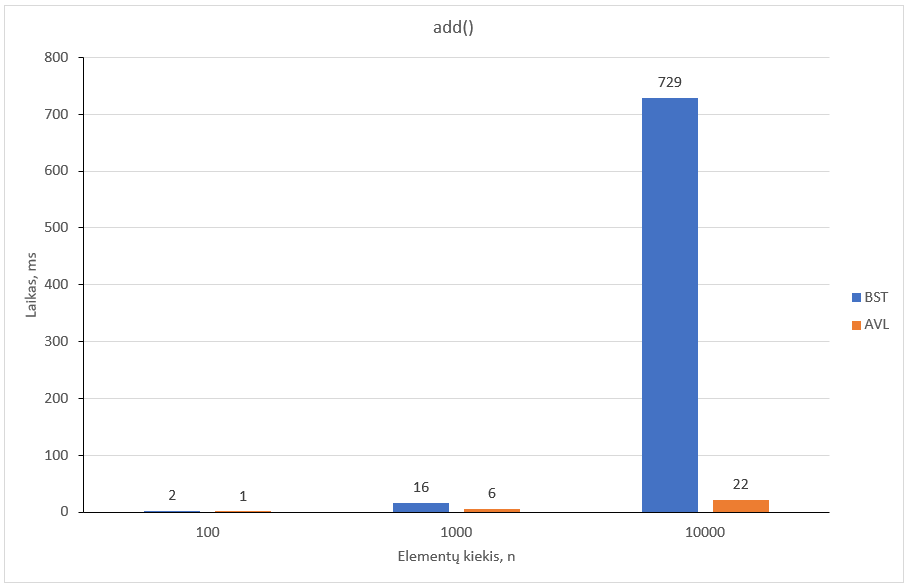
## add()

Kodas:

private static void bstAvlAdd(int n) {  
 BstSet<Integer> bstSet = new BstSet<>();  
 AvlSet<Integer> avlSet = new AvlSet<>();  
  
 long startTime = System.*currentTimeMillis*();  
 for (int i = 0; i < n; i++)  
 bstSet.add(i);  
 long endTime = System.*currentTimeMillis*();  
 *bstTime* = endTime - startTime;  
 Ks.*oun*("-----");  
 Ks.*oun*("BstSet add() (" + n + " elementų). Laikas: " + *bstTime* + " ms");  
  
 startTime = System.*currentTimeMillis*();  
 for (int i = 0; i < n; i++)  
 avlSet.add(i);  
 endTime = System.*currentTimeMillis*();  
 *avlTime* = endTime - startTime;  
 Ks.*oun*("AvlSet add() (" + n + " elementų). Laikas: " + *avlTime* + " ms");  
}

Rezultatai:



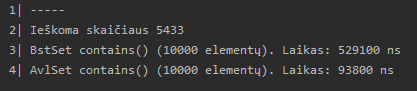


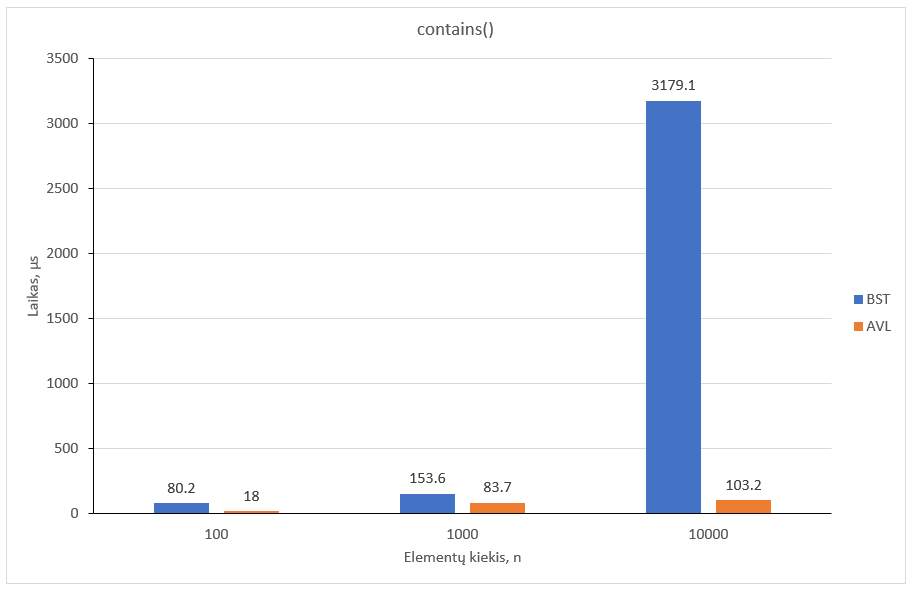
## contains()

Kodas:

private static void bstAvlContains(int n) {  
 BstSet<Integer> bstSet = new BstSet<>();  
 AvlSet<Integer> avlSet = new AvlSet<>();  
  
 for (int i = 0; i < n; i++) {  
 bstSet.add(i);  
 avlSet.add(i);  
 }  
  
 int randomInt = new Random().nextInt(n);  
 long startTime = System.*nanoTime*();  
 bstSet.contains(randomInt);  
 long endTime = System.*nanoTime*();  
 *bstTime* = endTime - startTime;  
 Ks.*oun*("-----");  
 Ks.*oun*("Ieškoma skaičiaus " + randomInt);  
 Ks.*oun*("BstSet contains() (" + n + " elementų). Laikas: " + *bstTime* + " ns");  
  
 startTime = System.*nanoTime*();  
 avlSet.contains(randomInt);  
 endTime = System.*nanoTime*();  
 *avlTime* = endTime - startTime;  
 Ks.*oun*("AvlSet contains() (" + n + " elementų). Laikas: " + *avlTime* + " ns");  
}

Rezultatai:



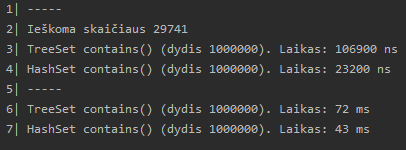


# TreeSet ir HashSet greitaveika

Kodas:

private static void greitaveika2(int n) {  
 TreeSet<Integer> treeSet = new TreeSet<>();  
 HashSet<Integer> hashSet = new HashSet<>();  
 long startTime;  
 long endTime;  
  
 for (int i = 0; i < n; i++) {  
 treeSet.add(i);  
 hashSet.add(i);  
 }  
  
 int randomInt = new Random().nextInt(n);  
 Ks.*oun*("-----");  
 Ks.*oun*("Ieškoma skaičiaus " + randomInt);  
  
 startTime = System.*nanoTime*();  
 treeSet.contains(randomInt);  
 endTime = System.*nanoTime*();  
 Ks.*oun*("TreeSet contains() (dydis " + n + "). Laikas: " + (endTime - startTime) + " ns");  
  
 startTime = System.*nanoTime*();  
 treeSet.contains(randomInt);  
 endTime = System.*nanoTime*();  
 Ks.*oun*("HashSet contains() (dydis " + n + "). Laikas: " + (endTime - startTime) + " ns");  
}  
  
private static void greitaveika3(int n) {  
 TreeSet<Integer> treeSet = new TreeSet<>();  
 HashSet<Integer> hashSet = new HashSet<>();  
 Collection<Integer> collection = new ArrayList<>();  
 long startTime;  
 long endTime;  
  
 for (int i = 0; i < n; i++) {  
 treeSet.add(i);  
 hashSet.add(i);  
 if (i % 5 == 0)  
 collection.add(i);  
 }  
  
 Ks.*oun*("-----");  
  
 startTime = System.*currentTimeMillis*();  
 treeSet.containsAll(collection);  
 endTime = System.*currentTimeMillis*();  
 Ks.*oun*("TreeSet contains() (dydis " + n + "). Laikas: " + (endTime - startTime) + " ms");  
  
 startTime = System.*currentTimeMillis*();  
 treeSet.containsAll(collection);  
 endTime = System.*currentTimeMillis*();  
 Ks.*oun*("HashSet contains() (dydis " + n + "). Laikas: " + (endTime - startTime) + " ms");  
}

Rezultatai:



# BstSet metodai

## headSet()

public Set<E> headSet(E element) {  
 if (element == null) {  
 throw new IllegalArgumentException("Element is null in headSet(E element)");  
 }  
  
 Set<E> newSet = new BstSet<>();  
 headSetRecursive(newSet, element, root);  
 return newSet;  
}  
  
private void headSetRecursive(Set<E> newList, E element, BstNode<E> node) {  
 if (node != null) {  
 if (c.compare(element, node.element) > 0)  
 newList.add(node.element);  
  
 headSetRecursive(newList, element, node.left);  
 headSetRecursive(newList, element, node.right);  
 }  
}

## subSet()

public Set<E> subSet(E from, E to) {  
 if (from == null || to == null) {  
 throw new IllegalArgumentException("Element is null in subSet(E from, E to)");  
 }  
  
 Set<E> newSet = new BstSet<>();  
 subSetRecursive(newSet, from, to, root);  
 return newSet;  
}  
  
private void subSetRecursive(Set<E> newList, E from, E to, BstNode<E> node) {  
 if (node != null) {  
 if ((c.compare(from, node.element) <= 0) && (c.compare(to, node.element) > 0))  
 newList.add(node.element);  
  
 subSetRecursive(newList, from, to, node.left);  
 subSetRecursive(newList, from, to, node.right);  
 }  
}

## tailSet()

public Set<E> tailSet(E element) {  
 if (element == null) {  
 throw new IllegalArgumentException("Element is null in tailSet(E element)");  
 }  
  
 Set<E> newSet = new BstSet<>();  
 tailSetRecursive(newSet, element, root);  
 return newSet;  
}  
  
private void tailSetRecursive(Set<E> newList, E element, BstNode<E> node) {  
 if (node != null) {  
 if (c.compare(element, node.element) <= 0)  
 newList.add(node.element);  
  
 tailSetRecursive(newList, element, node.left);  
 tailSetRecursive(newList, element, node.right);  
 }  
}

## Iteratoriaus remove()

public void remove() {  
 if (!stack.empty()) {  
 BstNode<E> n = stack.pop();  
 parent = (!stack.empty()) ? stack.peek() : root;  
 BstNode<E> node = (ascending) ? n.right : n.left;  
 toStack(node);  
  
 parent = removeRecursive(n.element, parent);  
 }  
}

# AvlSet remove()

Kodas:

public void remove(E element) {  
 root = removeRecursive(element, (AVLNode<E>) root);  
}  
  
private AVLNode<E> removeRecursive(E element, AVLNode<E> n) {  
 if (n == null) {  
 return null;  
 }  
  
 int cmp = c.compare(element, n.element);  
  
 if (cmp < 0) {  
 n.left = removeRecursive(element, n.getLeft());  
 } else if (cmp > 0) {  
 n.right = removeRecursive(element, n.getRight());  
 } else {  
 if ((n.getLeft() == null) || (n.getRight() == null)) {  
 if (n.getLeft() == null)  
 n = n.getRight();  
 else  
 n = n.getLeft();  
 } else {  
 // Node with two children: get the inorder  
 // successor (smallest in the right subtree)  
 AVLNode<E> temp = minValueNode(n.getRight());  
  
 // Copy the inorder successor's data to this node  
 n.element = temp.element;  
  
 // Delete the inorder successor  
 n.right = removeRecursive(temp.element, n.getRight());  
 }  
 }  
 // If the tree had only one node then return  
 if (n == null)  
 return null;  
  
 n.height = Math.*max*(height(n.getLeft()), height(n.getRight())) + 1;  
  
 // Get the balance factor of this node (to check whether  
 // this node became unbalanced)  
 int balance = getBalance(n);  
  
 // If this node becomes unbalanced, then there are 4 cases  
 // Left Left Case  
 if (balance > 1 && getBalance(n.getLeft()) >= 0)  
 return rightRotation(n);  
  
 // Left Right Case  
 if (balance > 1 && getBalance(n.getLeft()) < 0) {  
 n.left = leftRotation(n.getLeft());  
 return rightRotation(n);  
 }  
  
 // Right Right Case  
 if (balance < -1 && getBalance(n.getRight()) <= 0)  
 return leftRotation(n);  
  
 // Right Left Case  
 if (balance < -1 && getBalance(n.getRight()) > 0) {  
 n.right = rightRotation(n.getRight());  
 return leftRotation(n);  
 }  
   
 return n;  
}

Rezultatai:

