

**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**DUOMENŲ STRUKTŪROS**

**2 LABORATORINIS DARBAS**

**Atliko Karolis Stasaitis IFF8-9**

**Pagrindinė Klasė:**

package edu.ktu.ds.lab2.cpu;  
  
import edu.ktu.ds.lab2.utils.Ks;  
import edu.ktu.ds.lab2.utils.Parsable;  
  
import java.util.Comparator;  
import java.util.NoSuchElementException;  
import java.util.Random;  
import java.util.Scanner;  
  
public class CPU implements Parsable<CPU> {  
  
 private String CPU\_NAME;  
 private int CPU\_RELEASE\_YEAR;  
 private int CPU\_CORES;  
 private double CPU\_BASE\_MIN;  
 private double CPU\_BASE\_MAX;  
 private double CPU\_PRICE;  
  
 private static String *CPU\_REG\_PREF* = "ID";  
 private static int *CPU\_NUMBER* = 100;  
 private static String *CPU\_ID*;  
  
  
 public CPU()  
 {  
 *CPU\_ID* = *CPU\_REG\_PREF* + (*CPU\_NUMBER*++);  
 }  
 public CPU(String data)  
 {  
  
 }  
 public CPU(Builder builder) {  
 CPU\_NAME = builder.Name;  
 CPU\_RELEASE\_YEAR = builder.Year;  
 CPU\_CORES = builder.Core;  
 CPU\_BASE\_MIN = builder.Base\_Min;  
 CPU\_BASE\_MAX = builder.Base\_Max;  
 CPU\_PRICE = builder.Price;  
 // validate();  
 }  
 public CPU(String cpu\_name, int cpu\_release\_date, int cpu\_core, double cpu\_base\_min, double cpu\_base\_max, double cpu\_price)  
 {  
 CPU\_NAME = cpu\_name;  
 CPU\_RELEASE\_YEAR = cpu\_release\_date;  
 CPU\_CORES = cpu\_core;  
 CPU\_BASE\_MIN = cpu\_base\_min;  
 CPU\_PRICE = cpu\_price;  
 CPU\_BASE\_MAX = cpu\_base\_max;  
 }  
 public boolean validateYear(int minRelease, int maxRelease) {  
 if (CPU\_RELEASE\_YEAR < minRelease || CPU\_RELEASE\_YEAR > maxRelease) {  
 return false;  
 }  
 return true;  
 }  
 public boolean validatePrice(int minPrice, int maxPrice)  
 {  
 if (CPU\_PRICE < minPrice || CPU\_PRICE > maxPrice) {  
 return false;  
 }  
 return true;  
 }  
 @Override  
 public void parse(String dataString) {  
 try{  
 Scanner p = new Scanner(dataString).useDelimiter(",");  
 CPU\_NAME = p.next();  
 CPU\_RELEASE\_YEAR = p.nextInt();  
 CPU\_CORES = p.nextInt();  
 CPU\_BASE\_MIN = Double.*parseDouble*(p.next());  
 CPU\_BASE\_MAX = Double.*parseDouble*(p.next());  
 CPU\_PRICE = Double.*parseDouble*(p.next());  
 }  
 catch (NoSuchElementException e)  
 {  
 Ks.*ern*("Not enough data about CPU" + dataString);  
 }  
 }  
 /\*  
 @Override  
 public int compareTo(CPU car) {  
 return Get\_Cpu\_Id().compareTo(car.Get\_Cpu\_Id());  
 }  
 \*/  
 @Override  
 public String toString()  
 {  
 return String.*format*("%s %d %d %.2f %.2f %.2f", CPU\_NAME, CPU\_RELEASE\_YEAR, CPU\_CORES, CPU\_BASE\_MIN,CPU\_BASE\_MAX, CPU\_PRICE);  
 }  
 /\*  
 public final static Comparator byPrice = (obj1, obj2) -> {  
 double price1 = ((CPU) obj1).Get\_Cpu\_Price();  
 double price2 = ((CPU) obj2).Get\_Cpu\_Price();  
  
 double new\_price = Math.sqrt(price1);  
 double new\_price1 = Math.sqrt(price2);  
 if (new\_price < new\_price1) {  
 return -1;  
 }  
 if (new\_price > new\_price1) {  
 return 1;  
 }  
 return 0;  
 };  
 \*/  
 @Override  
 public int compareTo(CPU o) {  
 double Price = o.CPU\_PRICE;  
  
 if(this.CPU\_PRICE > Price)  
 {  
 return 1;  
 }  
 else if(this.CPU\_PRICE < Price)  
 {  
 return -1;  
 }  
 return 0;  
 }  
 public String Get\_Cpu\_Name(){return CPU\_NAME;}  
 public int Get\_Cpu\_Release\_Year() {return CPU\_RELEASE\_YEAR;}  
 public int Get\_Cpu\_Cores(){return CPU\_CORES;}  
 public double Get\_Cpu\_Base\_Min(){return CPU\_BASE\_MIN;}  
 public double Get\_Cpu\_Price(){return CPU\_PRICE;}  
 public double Get\_Cpu\_Base\_Max(){return CPU\_BASE\_MAX;}  
 public String Get\_Cpu\_Id() { return *CPU\_ID*;};  
 public static class Builder {  
  
 private final static Random *RANDOM* = new Random(1949); // Atsitiktinių generatorius  
 private final static String[][] *MODELS* = { // galimų automobilių markių ir jų modelių masyvas  
 {"Intel", "Quad-Core", "I3", "I5", "I7"},  
 {"AMD", "Ahtlon", "Razen"}  
 };  
 private String Name = "";  
 private int Year = -1;  
 private int Core = -1;  
 private double Base\_Min = -1.0;  
 private double Base\_Max = -1.0;  
 private double Price = -1.0;  
  
 public CPU build() {  
 return new CPU(this);  
 }  
  
 public CPU buildRandom() {  
 int ma = *RANDOM*.nextInt(*MODELS*.length); // markės indeksas 0..  
 int mo = 1 + *RANDOM*.nextInt(*MODELS*.length - 1);  
 int serie = 100 + *RANDOM*.nextInt(9000);  
 return new CPU(*MODELS*[ma][0] + " " + *MODELS*[ma][mo] + '-' + serie,  
 1990 + *RANDOM*.nextInt(25),// metai tarp 1990 ir 2009  
 1 + *RANDOM*.nextInt(8),// rida tarp 6000 ir 228000  
 2.5 + *RANDOM*.nextDouble() \* 4.5,  
 4.5 + *RANDOM*.nextDouble() \* 6.0,  
 200 + *RANDOM*.nextDouble() \* 2000);// kaina tarp 800 ir 88800  
 }  
  
 public Builder Year(int year) {  
 this.Year = year;  
 return this;  
 }  
  
 public Builder model(String model) {  
 this.Name = model;  
 return this;  
 }  
 public Builder Core(int core)  
 {  
 this.Core = core;  
 return this;  
 }  
 public Builder Base\_Min(double base)  
 {  
 this.Base\_Min = base;  
 return this;  
 }  
 public Builder Base\_Max(double base)  
 {  
 this.Base\_Max = base;  
 return this;  
 }  
 public Builder price(double price) {  
 this.Price = price;  
 return this;  
 }  
 }  
  
}

**Avl Klasės realizuoti metodai:**

*/\*\*  
 \* Pašalinamas elementas iš aibės.  
 \*  
 \** ***@param*** *element  
 \*/*@Override  
public void remove(E element) {  
 if(size == 0 || element == null) {  
 throw new UnsupportedOperationException("Element is null or size is zero");  
 }  
 root = removeRecursive(element, (AVLNode<E>)root);  
}  
  
private AVLNode<E> removeRecursive(E element, AVLNode<E> n) {  
 if (n == null) {  
 return n;  
 }  
  
 int h = c.compare(element, n.element);  
  
 //System.out.println(h + " " + element + " " + n.element);  
  
 // System.out.println(element);  
  
 if(h < 0)  
 {  
 n.setLeft(removeRecursive(element, n.getLeft()));  
 if ((height(n.getLeft()) - height(n.getRight())) == 2) {  
 int cmp2 = c.compare(element, n.getLeft().element);  
 n = (cmp2 < 0) ? rightRotation(n) : doubleRightRotation(n);  
 }  
 }  
 else if(h > 0)  
 {  
 n.setRight(removeRecursive(element, n.getRight()));  
 if ((height(n.getRight()) - height(n.getLeft())) == 2) {  
 int cmp2 = c.compare(n.getRight().element, element);  
 n = (cmp2 < 0) ? leftRotation(n) : doubleLeftRotation(n);  
 }  
 }  
 else if(n.getLeft() != null && n.getRight() != null)  
 {  
 System.*out*.println("dis");  
 BstNode<E> Max = getMin(n.getRight());  
  
 n.element = Max.element;  
  
 n.setRight(removeRecursive(n.element, n.getRight()));  
 }  
 else  
 {  
 System.*out*.println("No child" + element);  
 n = (n.getLeft() != null) ? n.getLeft() : n.getRight();  
 size--;  
 if(n == null)  
 {  
 return n;  
 }  
 }  
 n.height = Math.*max*(height(n.getLeft()), height(n.getRight())) + 1;  
 return n;  
  
}

**Klasės Bst realizuoti metodai**

**pollFirst()**

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

**removeAll(BstSet<?> set)**

public boolean removeAll(BstSet<E> c)  
{  
 if(root == null)  
 {  
 return false;  
 }  
 IteratorBst iterator = new IteratorBst(false);  
 Iterator<E> iterator1;  
 while(iterator.hasNext())  
 {  
 E e = iterator.next();  
 iterator1 = c.descendingIterator();  
 while(iterator1.hasNext())  
 {  
 E ee = iterator1.next();  
 int equal = this.c.compare(e, ee);  
 if(equal == 0)  
 {  
 remove(e);  
 break;  
 }  
 }  
 }  
 return true;  
  
}

**SortedSet Metodai:**

@Override  
public SortedSet<E> headSet(E element) {  
 if (element == null) {  
 throw new IllegalArgumentException("Element is null");  
 }  
 SortedSet<E> set = new BstSet<>();  
 IteratorBst iterator;  
 iterator = new IteratorBst(false);  
 while(iterator.hasNext())  
 {  
 E e = iterator.next();  
 int cmp = c.compare(e, element);  
 System.*out*.println("Cmp" + cmp);  
 if(cmp < 0)  
 {  
 set.add(e);  
 }  
 }  
 return set;  
}  
  
*/\*\*  
 \* Grąžinamas aibės poaibis nuo elemento element1 iki element2.  
 \*  
 \** ***@param*** *element1 - pradinis aibės poaibio elementas.  
 \** ***@param*** *element2 - galinis aibės poaibio elementas.  
 \** ***@return*** *Grąžinamas aibės poaibis nuo elemento element1 iki element2.  
 \*/*@Override  
public SortedSet<E> subSet(E element1, E element2) {  
 if (element1 == null || element2 == null) {  
 throw new IllegalArgumentException("Element is null");  
 }  
 SortedSet<E> set = new BstSet<>();  
 IteratorBst iterator;  
 iterator = new IteratorBst(false);  
 while(iterator.hasNext())  
 {  
 E e = iterator.next();  
 int cmp = c.compare(e, element1);  
 System.*out*.println(cmp);  
 int cmp1 = c.compare(element2, e);  
 System.*out*.println(cmp1);  
 if(cmp >= 0 && cmp1 > 0)  
 {  
 set.add(e);  
 }  
  
 }  
 return set;  
}  
  
*/\*\*  
 \* Grąžinamas aibės poaibis iki elemento.  
 \*  
 \** ***@param*** *element - Aibės elementas.  
 \** ***@return*** *Grąžinamas aibės poaibis nuo elemento.  
 \*/*@Override  
public SortedSet<E> tailSet(E element) {  
 if (element == null) {  
 throw new IllegalArgumentException("Element is null");  
 }  
 SortedSet<E> set = new BstSet<>();  
 IteratorBst iterator;  
 iterator = new IteratorBst(true);  
 while(iterator.hasNext())  
 {  
 E e = iterator.next();  
 int cmp = c.compare(e, element);  
 if(cmp >= 0)  
 {  
 set.add(e);  
 }  
 }  
 return set;  
}

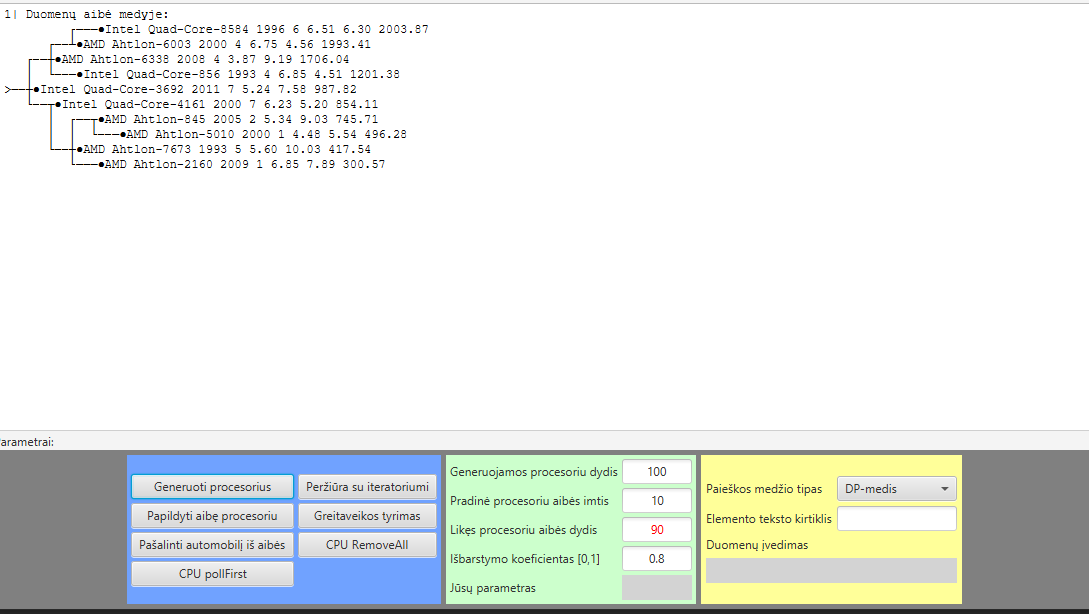
**Realizuotas Iteratoriaus remove metodas(Nepilnai veikiantis)**

@Override  
public void remove() {  
 if(!stack.isEmpty()) {  
 BstNode<E> node = stack.peek();  
 int compare = c.compare(parent.element, node.element);  
  
 if (compare > 0) {  
 root = removeRecursive(parent.left.element, root);  
 } else {  
 root = removeRecursive(parent.right.element, root);  
 }  
 }  
}

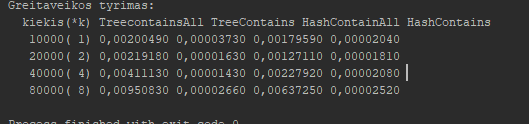
**JUNIT5 Testo Klasė**

package edu.ktu.ds.lab2.cpu;  
  
import org.junit.jupiter.api.BeforeAll;  
import org.junit.jupiter.api.DisplayName;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.*assertTrue*;  
  
public class JUNIT {  
  
 static CPU\_Generator *generator* = new CPU\_Generator();  
 static CPU[] *CPU\_List*;  
 static CPU *cpu* = new CPU();  
 @BeforeAll  
 public static void Create()  
 {  
 *CPU\_List* = *generator*.generateShuffle(30,10, 0.8);  
 *cpu* = *generator*.takeCpu();  
 }  
 @DisplayName("Checking Year")  
 @Test  
 public void Check\_Year()  
 {  
 *assertTrue*(*cpu*.validateYear(2019, 2020), String.*format*("Error year is too low or too high %d", *cpu*.Get\_Cpu\_Release\_Year()));  
 }  
 @DisplayName("Checking Price")  
 @Test  
 public void Check\_Price()  
 {  
 *assertTrue*(*cpu*.validatePrice(1999, 2000), String.*format*("Error price is out of bound %.2f", *cpu*.Get\_Cpu\_Price()));  
 }  
}

**JAVAFX:**



**Greitaveikos rezultatai:**



Hash metodai atlieka greičiausiai, treeSet metodams užima 0(Log n), o hashset 0(1).

Išvados

Medžiai atlieka metodus žymiai efektyviau. JUNIT klasė yra patogi priemonė sutaupyti laiko programuotojui bei atrasti klaidas žymiai greičiau.