**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**Modulio P175B014 „Duomenų struktūros“**

Laboratorinio darbo ataskaita

**Antras laboratorinis darbas (tema „Greitaveika“)**

|  |  |
| --- | --- |
|  | **Dėstytoja**  Lekt. Eglė Butkevičiūtė |
|  |
| **Studentė**  Simona Ragauskaitė IFF-0/3 |

**KAUNAS, 2021**

Turinys

[1. Tiriamieji metodai 3](#_Toc86615232)

[2. Tiriamųjų metodų asimptotinis sudėtingumas 4](#_Toc86615233)

[3. Greitaveikos testavimo metodika 5](#_Toc86615234)

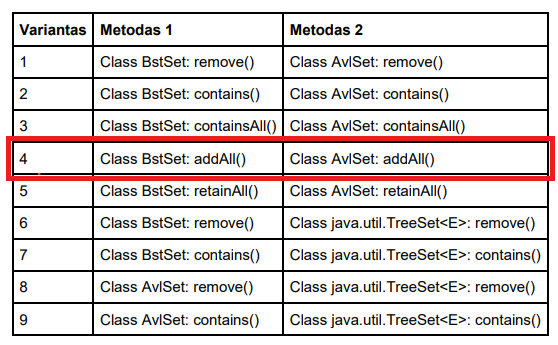
[4. Kompiuterio parametrai, kuriuo atliktas testavimas 7](#_Toc86615235)

[5. Algoritmų vykdymo laiko priklausomybės nuo duomenų kiekio grafikas 8](#_Toc86615236)

[6. Išvados 9](#_Toc86615237)

# Tiriamieji metodai

4 variantas



# Tiriamųjų metodų asimptotinis sudėtingumas

BstSet klasė yra dvejetainis paieškos medis, o AvlSet klasė realizuota AVL-medžio principu. Šiose klasėse metodas addAdd(Set<E>list) prie esančio sąrašo prideda aibėje esančius elementus, kuri yra perduodama per metodo parametrus. Šio metodo asimptotinis sudėtingumas abiejų klasių yra O(m\* log2n).

# Greitaveikos testavimo metodika

Testavimo algoritmas

public class Benchmark {  
  
 @State(Scope.*Benchmark*)  
 public static class FullSet {  
 Car[] cars;  
 BstSet<Car> carSet;  
 @Setup(Level.*Iteration*)  
 public void generateElements(BenchmarkParams params) {  
 cars = Benchmark.*generateElements*(Integer.*parseInt*(params.getParam("elementCount")));  
 }  
 }  
  
 @Param({"2000", "4000", "8000", "12000", "24000"})  
 public int elementCount;  
  
 static Car[] *cars*;  
  
 @Setup(Level.*Iteration*)  
 public void generateElements() {  
 *cars* = *generateElements*(elementCount);  
 }  
  
 static Car[] generateElements(int count) {  
 return new CarsGenerator().generateShuffle(count, 1.0);  
 }  
  
 @org.openjdk.jmh.annotations.Benchmark  
 public BstSet<Car> addAllBst() {  
 BstSet<Car> carSet = new BstSet<>(Car.*byPrice*);  
 addAllBstElements(*cars*, carSet);  
 return carSet;  
 }  
  
 @org.openjdk.jmh.annotations.Benchmark  
 public AvlSet<Car> addAllAvl() {  
 AvlSet<Car> carSet = new AvlSet<>(Car.*byPrice*);  
 addAllAvlElements(*cars*, carSet);  
 return carSet;  
 }  
  
 private void addAllAvlElements(Car[] cars, AvlSet<Car> carSet) {  
 Set<Car> list = new AvlSet<>();  
 for(Car car:cars) {  
 list.add(car);  
 }  
 carSet.addAll(list);  
 }  
  
 private void addAllBstElements(Car[] cars, BstSet<Car> carSet) {  
 Set<Car>list = new BstSet<>();  
 for(Car car:cars) {  
 list.add(car);  
 }  
 carSet.addAll(list);  
 }  
  
 public static void main(String[] args) throws RunnerException {  
 Options opt = new OptionsBuilder()  
 .include(Benchmark.class.getSimpleName())  
 .forks(1)  
 .build();  
 new Runner(opt).run();  
 }  
}

# Kompiuterio parametrai, kuriuo atliktas testavimas

|  |  |
| --- | --- |
| Procesorius | Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz |
| Darbinė atmintis | 16.0 GB |

# Algoritmų vykdymo laiko priklausomybės nuo duomenų kiekio grafikas

|  |  |  |
| --- | --- | --- |
|  | **Sąrašas papildomas sąrašu** | |
| **Elementų kiekis** | **BstSet.addAll(),μs** | **AvlSet.addAll(),μs** |
| 2000 | 544,256 | 729,448 |
| 4000 | 1357,865 | 1716,044 |
| 6000 | 2327,069 | 2903,324 |
| 8000 | 3404,791 | 4022,715 |
| 10000 | 4611,862 | 5454,949 |
| 12000 | 5730,988 | 6655,082 |
| 24000 | 13474,929 | 15378,022 |

# Išvados

Iš gautų greitaveikos rezultatų ir grafiko BstSet klasėje esantis addAll() metodas buvo greitesnis. Šios klasės vidutinis asimptotinis – O(m\*log2n) (reikia peržiūrėt tik kažkokią dalį elementų), o blogiausias – O(n\*m) (reikia peržiūrėti visus elementus). O AvlSet klasėje esantis addAll() metodo vidutinis asimptotinis sudėtingumas – O(2\*m\*log2n), o blogiausias – O(2\*n\*m). Jis yra dvigubai sudėtingesnis, nes medyje reikia atlikti reikiamus pasukimus.