Duomenų struktūrų lab2 ataskaita

Parengė:

Vytenis Kriščiūnas IFF-1/1

Tiriamieji metodai

BstSet ir AvlSet containsAll() metodai.

Klasės BstSet: containsAll() metodas:

```java
public boolean containsAll(Set<E> set) {

    for (E element : set){
        if (contains(element) == false){
            return false;
        }
    }
    return true;
}
```

Metodas patikrina ar aibėje egzistuoja visi elementai, esantys aibėje set.

Klasė AvlSet paveldi containsAll() metodą iš BstSet klasės.

Asimptotinis sudėtingumas

Class BstSet: containsAll()

Asimptotinis sudėtingumas $O(\log_2(n))$.

Class AvlSet: containsAll()

Asimptotinis sudėtingumas $O(\log_2(n))$.

Greitaveikos testavimo metodika

Testavimo klasė:

```java
package edu.ktu.ds.lab2.demo;

import edu.ktu.ds.lab2.utils.*;
import org.openjdk.jmh.annotations.*;
import org.openjdk.jmh.infra.BenchmarkParams;
import org.openjdk.jmh.runner.Runner;
import org.openjdk.jmh.runner.RunnerException;
import org.openjdk.jmh.runner.options.Options;
import org.openjdk.jmh.runner.options.OptionsBuilder;

import java.util.concurrent.TimeUnit;

4 inheritors
@BenchmarkMode(Mode.AverageTime)
@State(Scope.Benchmark)
@OutputTimeUnit(TimeUnit.MICROSECONDS)
@Warmup(time = 1, timeUnit = TimeUnit.SECONDS)
@Measurement(time = 1, timeUnit = TimeUnit.SECONDS)
public class Benchmark {

    4 usages  4 inheritors
    @State(Scope.Benchmark)
    public static class FullSet {

        3 usages
        Car[] cars;
        3 usages
        AvlSet<Car> avlSet;
        3 usages
        BstSet<Car> bstSet;

        16 usages
        @Setup(Level.Iteration)
        public void generateElements(BenchmarkParams params) {
            cars = Benchmark.generateElements(Integer.parseInt(params.getParam( key: "elementCount")));
        }

        40 usages
        @Setup(Level.Invocation)
        public void fillCarSet(BenchmarkParams params) {
            addElements(cars, bstSet = new BstSet<>());
            addElements(cars, avlSet = new AvlSet<>());
        }
    }
```

```
8 usages
@Param({"4000", "8000", "16000", "32000", "64000"})
public int elementCount;

1 usage
Car[] cars;

28 usages
@Setup(Level.Iteration)
public void generateElements() {
    cars = generateElements(elementCount);
}
2 usages
static Car[] generateElements(int count) {
    return new CarsGenerator().generateShuffle(count, shuffleCoef: 1.0);
}

10 usages
@org.openjdk.jmh.annotations.Benchmark
public void BstSetContainsAll(FullSet carSet) {
   carSet.bstSet.containsAll(carSet.bstSet);
}

10 usages
@org.openjdk.jmh.annotations.Benchmark
public void AvlSetContainsAll(FullSet carSet) {
    carSet.avlSet.containsAll(carSet.avlSet);
}
2 usages
public static void addElements(Car[] carArray, SortedSet<Car> carSet) {
    for (int i = 0; i < carArray.length; i++) {
        carSet.add(carArray[i]);
    }
}

public static void main(String[] args) throws RunnerException {
    Options opt = new OptionsBuilder()
            .include(Benchmark.class.getSimpleName())
            .forks( i: 1)
            .build();
    new Runner(opt).run();
}
```

Pasirenku elementų kiekius testavimui. Sukuriu elementus pagal pasirinktą skaičių ir juos sudedu į BstSet ir AvlSet kintamuosius. Tada apsirašau metodus AvlSetContainsAll() ir BstSetContainsAll(), kuriuose naudoju containsAll() metodus. Galiausiai pradėjus greitaveikos testavimą stebiu gautus rezultatus.

Kompiuterio parametrai

Procesorius:

AMD FX-6300 six-core, greitis – 3.50 Ghz., 3 branduoliai, apdorojimas – 64-bit, talpykla – 8 MB.

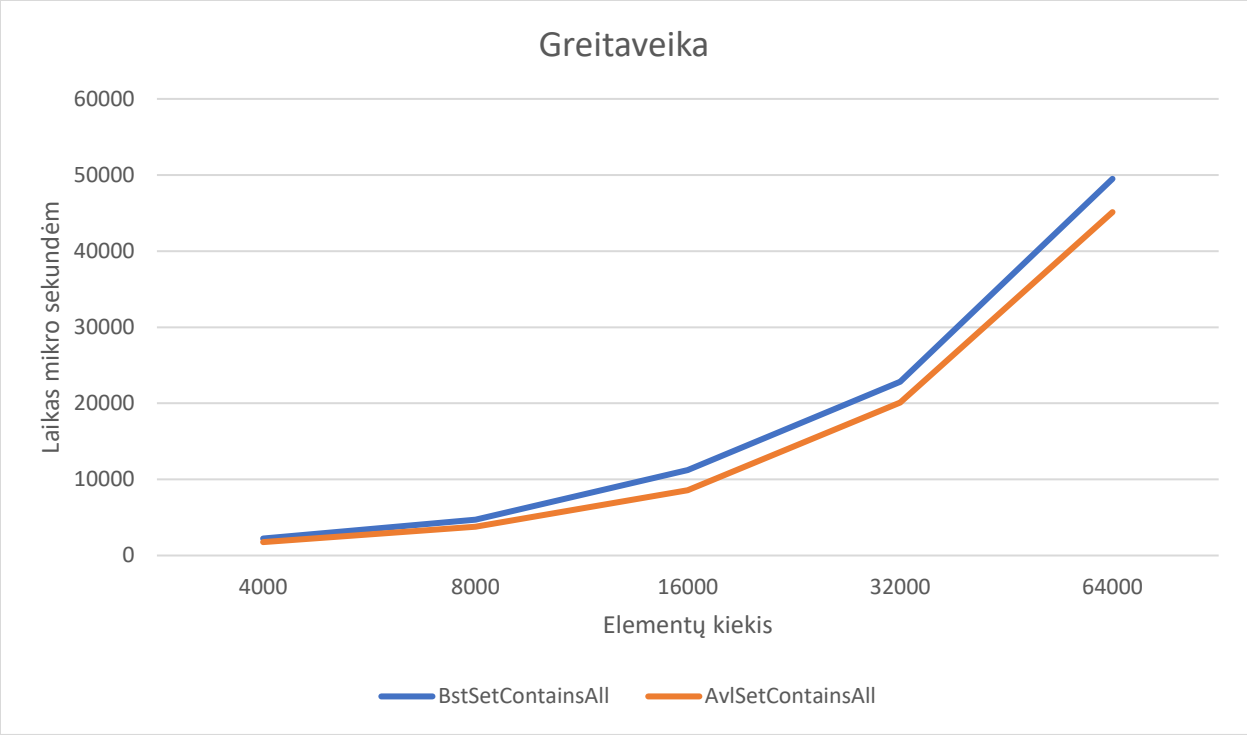
Atmintis:

16 GB

Talpa:

SSD 224 GB

Algoritmų/ metodų vykdymo laiko priklausomybės nuo įvesties duomenų kiekio grafikas



Išvados

Pagal gautus rezultatus akivaizdu, kad AvlSet klasės containsAll() metodas yra greitesnis nei BstSet klasės containsAll() metodas, nors asimptotiniai sudėtingumai nesiskiria balansuoto medžio metodas turi atlikti mažiau veiksmų nei nebalansuoto medžio metodas. Kuo didesnis elementų kiekis tuo didesnis greičių skirtumas yra matomas.