**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**Modulio P175B014 „Duomenų struktūros“**

Laboratorinio darbo ataskaita

**Trečias laboratorinis darbas (tema „Maišos lentelės“)**

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

**KAUNAS, 2021**

Turinys

[1. Tiriamieji metodai 3](#_Toc88483778)

[2. Tiriamųjų metodų asimptotinis sudėtingumas 4](#_Toc88483779)

[3. Greitaveikos testavimo metodika 5](#_Toc88483780)

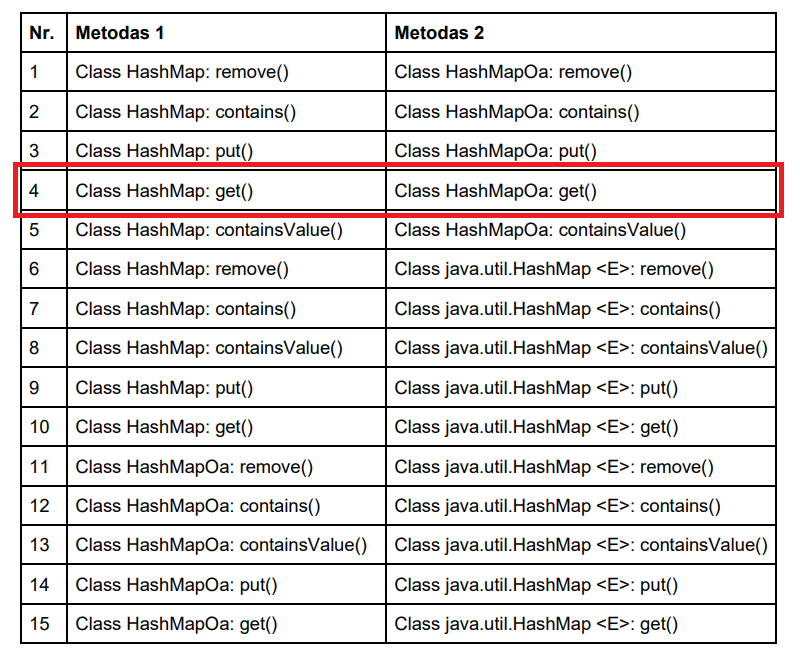
[4. Kompiuterio parametrai, kuriuo atliktas testavimas 7](#_Toc88483781)

[5. Algoritmų vykdymo laiko priklausomybės nuo duomenų kiekio grafikas 8](#_Toc88483782)

[6. Išvados 9](#_Toc88483783)

# Tiriamieji metodai

4 variantas



# Tiriamųjų metodų asimptotinis sudėtingumas

Tiek HashMap, tiek HashMapOa klasėse esantis get() metodas eina per gradinėlę-bloką norėdamas pasiimti reikšmę. Jo vidutinis asimptotinis sudėtingumas yra konstantinis Θ(1) (abejose klasėse), o blogiausiu atveju asimptotinis sudėtingumas – O(n).

# Greitaveikos testavimo metodika

Testavimo algoritmas

public class Benchmark {  
  
 @State(Scope.*Benchmark*)  
 public static class FullMap {  
  
 List<String> ids;  
 List<Car> cars;  
 HashMap<String, Car> carsMap;  
 HashMapOa<String, Car> carsMapOa;  
  
 @Setup(Level.*Iteration*)  
 public void generateIdsAndCars(BenchmarkParams params) {  
 ids = Benchmark.*generateIds*(Integer.*parseInt*(params.getParam("elementCount")));  
 cars = Benchmark.*generateCars*(Integer.*parseInt*(params.getParam("elementCount")));  
 }  
  
 @Setup(Level.*Invocation*)  
 public void fillCarMap(BenchmarkParams params) {  
 carsMap = new HashMap<>(HashManager.HashType.*DIVISION*);  
 carsMapOa = new HashMapOa<>(HashManager.HashType.*DIVISION*);  
 *putMappings*(ids, cars, carsMap);  
 *putMappings*(ids, cars, carsMapOa);  
 }  
 }  
  
 @Param({"10000", "20000", "40000", "80000"})  
 public int elementCount;  
  
 List<String> ids;  
 List<Car> cars;  
  
 @Setup(Level.*Iteration*)  
 public void generateIdsAndCars() {  
 ids = *generateIds*(elementCount);  
 cars = *generateCars*(elementCount);  
 }  
  
 static List<String> generateIds(int count) {  
 return new ArrayList<>(CarsGenerator.*generateShuffleIds*(count));  
 }  
  
 static List<Car> generateCars(int count) {  
 return new ArrayList<>(CarsGenerator.*generateShuffleCars*(count));  
 }  
  
 @org.openjdk.jmh.annotations.Benchmark  
 public void getCarMap(FullMap fullMap) {  
 fullMap.ids.forEach(id -> fullMap.carsMap.get(id));  
 }  
  
 @org.openjdk.jmh.annotations.Benchmark  
 public void getCarMapOa(FullMap fullMapOa) {  
 fullMapOa.ids.forEach(id -> fullMapOa.carsMap.get(id));  
 }  
  
 public static void putMappings(List<String> ids, List<Car> cars, Map<String, Car> carsMap) {  
 for (int i = 0; i < cars.size(); i++) {  
 carsMap.put(ids.get(i), cars.get(i));  
 }  
 }  
  
 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

|  |  |  |
| --- | --- | --- |
|  | **Paimamas elementas iš sąrašo** | |
| **Elementų kiekis** | **HashMap.get(),μs** | **HashMapOa.get(),μs** |
| 10000 | 235,428 | 212,485 |
| 20000 | 1074,348 | 1073,95 |
| 40000 | 2705,888 | 2595,942 |
| 80000 | 6106,383 | 6193,042 |

# Išvados

Iš gautų greitaveikos rezultatų ir grafiko galime matyti, kad abiejų klasių (HashMap ir HashMapOa) testuojamas metodas get() yra O(n) asimptotinis sudėtingumas. Taigi, atitinka teorinį sudėtingumą vidutinį Θ(1), blogiausią – O(n), nes priklauso nuo raktų kiekio O(1\*n). HashMap gali būti sudėti į vieną grandinėlę ir tekti juos visus peržiūrėti, norint pasiimti reikiamą. HashMapOa gali susidaryti vienas blokas, kurį taip pat kaip HashMap reikės peržiūrėti. Be to, testavimo algoritmas buvo atliekamas su atvirojo adresavimo Linear tipu bei hešavimo dalybos tipu (DIVISION – h(x) = |x| mod M).