|  |
| --- |
| **KAUNO TECHNOLOGIJOS UNIVERSITETAS**  **INFORMATIKOS FAKULTETAS** |
| **Modulio P175B014 „Duomenų struktūros“**  Laboratorinio darbo aprašas (ataskaita)  **Trečiasis laboratorinis darbas** |
| **Dėstytojas**  Giedrius Paulikas |
| **Studentas**  Nedas Liaudanskis |
| **KAUNAS, 2022** |

Turinys

[Tiriamieji metodai 2](#_Toc119431724)

[Greitaveikos testavimo metodika 4](#_Toc119431725)

[Kompiuterio parametrai 6](#_Toc119431726)

[Greitaveikos testavimo rezultatai 6](#_Toc119431727)

[Išvados 7](#_Toc119431728)

# Tiriamieji metodai

|  |  |  |
| --- | --- | --- |
| **3** | **Class HashMap: put()** | **Class HashMapOa: put()** |

Pirmiausia atkreipiau dėmesį į taip kaip abu metodai yra aprašyti. Class HashMap: put() metodas atrodo gan paprastas. Pirmiausia patikrina ar atsineštas raktas nėra null ir atsineštas value nėra null. Tada susiranda indeksą, kuriame bus įdėta nauja reikšmė ir patikrina ar tame indekse nėra jau esamos reikšmės, jeigu nėra padidina chainsCounter per vieną dydį. Toliau esantis metodas getInChain suranda grandinėje ar yra elementas su tokiu raktu, jeigu taip, tai jį reikia perrašyti, jeigu ne, tai sukuriamas naujas Node su atsineštu raktu ir reikšme. Jis yra prijungiamas prie esamos grandinės. Visos konstantos sutvarkomos ir metodas baigia savo darbą.

Tuo tarpu Class HashMapOa: put() metodas, pirmiausia patikrina ar atsineštas raktas nėra null ir atsineštas value nėra null. Tada suranda naują poziciją, į kurią bus galima įdėti naują elementą, jeigu pozicija yra lygi -1, tai įvyko klaida ir HashMap sudėti elementai gauna naujus indeksus ir vykdoma put() metodo rekursija. Jeigu position yra skaičius > -1, tai į tą poziciją yra įdedamas naujas raktas ir reikšmė.

Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

**pav. 1 Java HashMap Put() method**

Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

**pav. 2 HasMapOa Put() method**

# Greitaveikos testavimo metodika

Norint atlikti greitaveikos testus reikėjo apsirašyti kelis papildomus dydžius: HashMapOa<String, Car> oaCarsMap, Car RandCar, String RandId. Kaip buvo sukurti šie dydžiai, reikėjo surasti elementą, kurį naudosime metode put() norint į HashMap į dėti tą elementą. Šis elementas buvo gautas generuojant vieną atsitiktinį objektą Car ir objektą Id. HashMap užpildymui taip pat buvo sugeneruota daug atsitiktinių Car objektų ir jų Id, kurie buvo sudėti į List<Car> ir List<String>. Užpildymas HashMap vyko paprastai. Objektai iš List<Car> ir List<String> buvo perkelti į HashMap objektus. Elementų kiekis kito kiekviename teste(pradedant nuo 10000 elementu ir didinant juos po 10000 kiekviename teste iki tol kol elementų skaičius pasiekė 100000).

Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

**pav. 3 Greitaveikos pasiruošimas**

Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

**pav. 4 Greitaveikos metodai**

# Kompiuterio parametrai

Greitaveikos tyrimai buvo įvykdyti naudojant nešiojamą kompiuterį Legion 5 17ACH6H

Su tiesioginiu krovimu iš elektros lizdo.

* AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz
* RAM 16,0 GB
* Windows 10

# Greitaveikos testavimo rezultatai

|  |
| --- |
| Benchmark (elementCount) Mode Cnt Score Error Units  Benchmark.putHashMap 10000 avgt 5 0,401 ± 0,822 us/op  Benchmark.putHashMap 20000 avgt 5 1,679 ± 2,471 us/op  Benchmark.putHashMap 30000 avgt 5 3,080 ± 2,876 us/op  Benchmark.putHashMap 40000 avgt 5 3,346 ± 2,141 us/op  Benchmark.putHashMap 50000 avgt 5 2,927 ± 1,590 us/op  Benchmark.putHashMap 60000 avgt 5 3,224 ± 2,902 us/op  Benchmark.putHashMap 70000 avgt 5 3,183 ± 1,311 us/op  Benchmark.putHashMap 80000 avgt 5 2,929 ± 1,222 us/op  Benchmark.putHashMap 90000 avgt 5 3,433 ± 4,690 us/op  Benchmark.putHashMap 100000 avgt 5 2,979 ± 0,386 us/op  Benchmark.putOaHashMap 10000 avgt 5 0,258 ± 0,465 us/op  Benchmark.putOaHashMap 20000 avgt 5 2,479 ± 3,636 us/op  Benchmark.putOaHashMap 30000 avgt 5 3,908 ± 0,966 us/op  Benchmark.putOaHashMap 40000 avgt 5 5,498 ± 6,691 us/op  Benchmark.putOaHashMap 50000 avgt 5 3,151 ± 7,640 us/op  Benchmark.putOaHashMap 60000 avgt 5 3,388 ± 3,020 us/op  Benchmark.putOaHashMap 70000 avgt 5 4,281 ± 6,347 us/op  Benchmark.putOaHashMap 80000 avgt 5 4,289 ± 6,288 us/op  Benchmark.putOaHashMap 90000 avgt 5 6,744 ± 12,842 us/op  Benchmark.putOaHashMap 100000 avgt 5 2,119 ± 0,350 us/op |

# Išvados

Iš greitaveikos tyrimų rezultatų galiu teigti, jog metodų HashMap: put(), HashMapOa Put() greitaveika yra gan skirtinga. HashMap put() operacija yra visai stabili, norint įvykdyti metodą operacijų nereikia daug. Tuo tarpu metodo HashMapOa put () operacijų skaičius yra didesnis, turi ir šuolius, kurie įvyksta kai reikia Naudoti Rehash() metoda padidinant masyvo dydį dvigubai. Visa tai parodo, jog HashMap put() metodas yra pranašesnis už HashMapOa() put metodą, gan dideliu greitaveikos skirtumu.