**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

Duomenų struktūros (P175B014) Laboratorinio darbo Nr.3 ataskaita

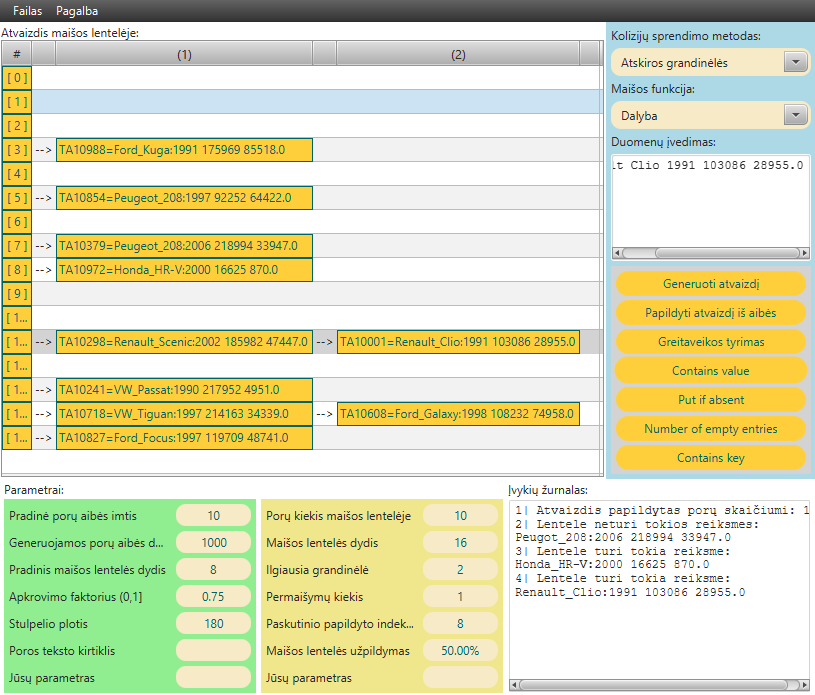
Atliko Martynas Pilinkus gr. IFF-8/1

Programos kodas: <https://github.com/MPilinkus/Lab3>

boolean containsValue(Object value)

**public boolean** containsValue(Object value) {  
 **if** (value == **null**)  
 **return false**;  
 **for** (Node<K, V> n : **table**) {  
 **for**(Node<K, V> nn = n; nn != **null**; nn = nn.**next**) {  
 **if** (nn.**value**.equals(value)){  
 **return true**;  
 }  
 }  
 }  
 **return false**;  
}

Metodui ištestuoti buvo spaudžiamas mygtukas apvestas raudonai žemiau esančiam paveikslėlyje.

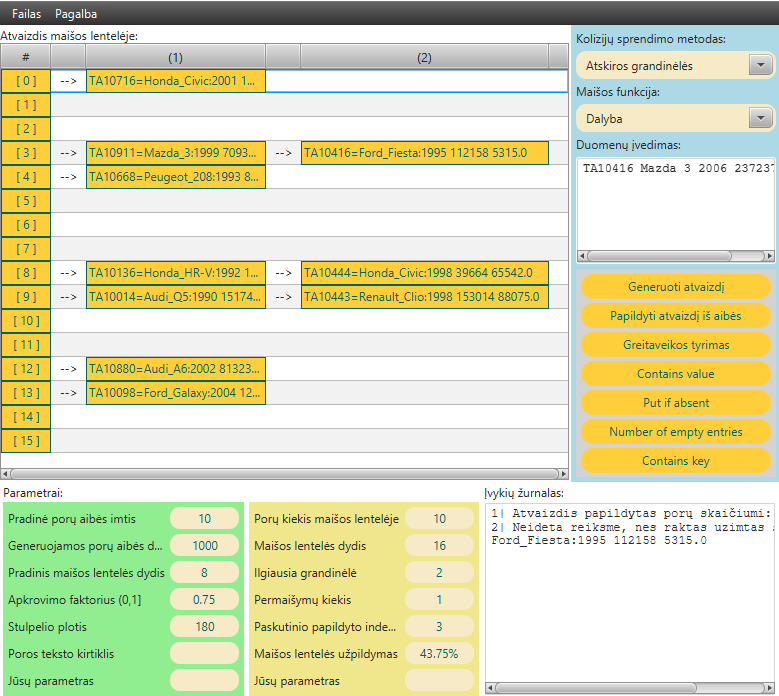


V putIfAbsent(K key, V value)

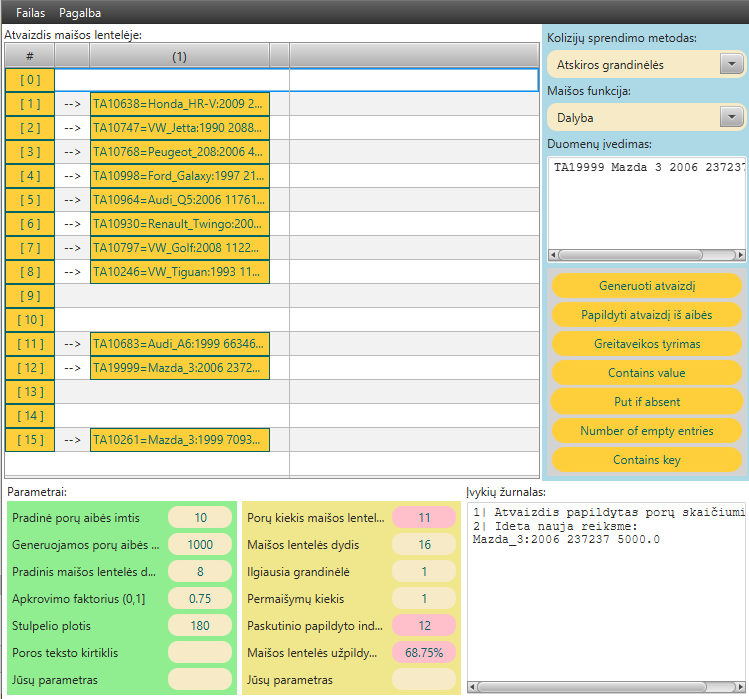
**public** V putIfAbsent(K key, V value) {  
 V val = get(key);  
 **if**(val == **null**) {  
 put(key, value);  
 }  
 **return** val;  
}

Metodui ištestuoti buvo spaudžiamas mygtukas apvestas raudonai žemiau esančiuose paveikslėliuose.

Įdėti nepavyko nes, raktas TA10416 jau užimtas



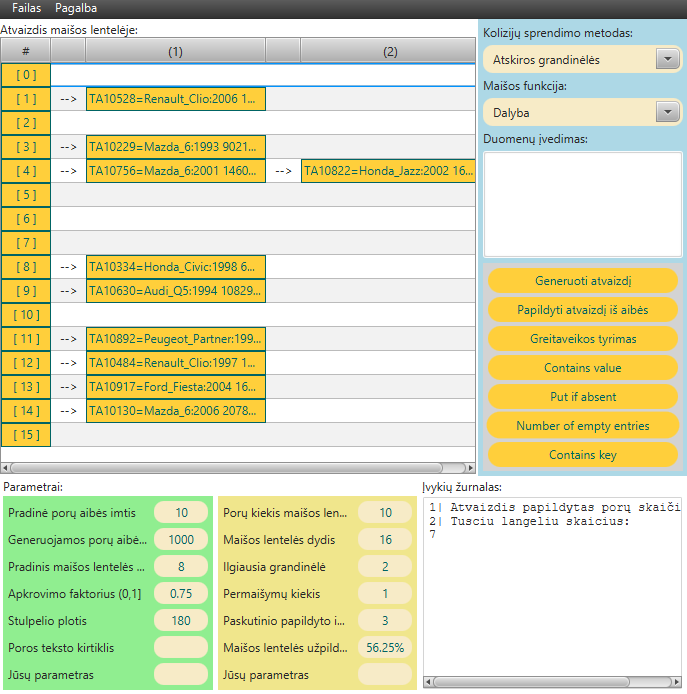
Įdėti pavyko, nes raktas TA19999 nebuvo užimtas

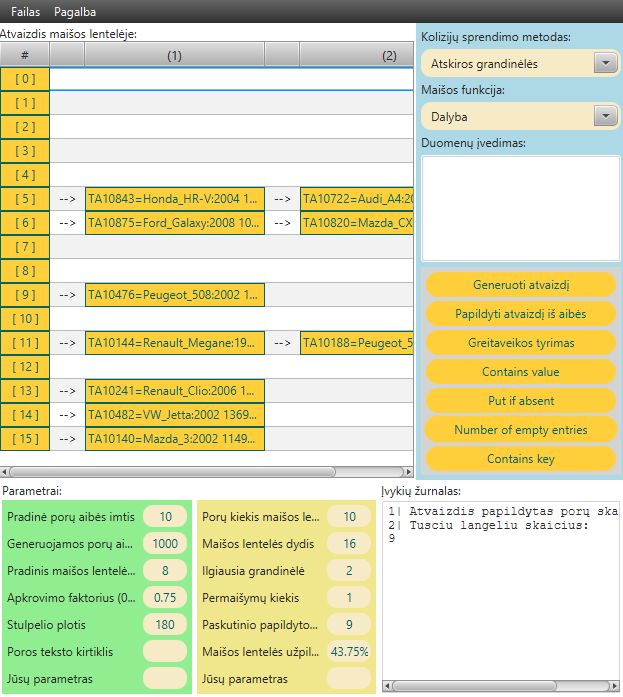


int numberOfEmpties()

**public int** numberOfEmpties() {  
 **return table**.**length** - **chainsCounter**;  
}

Metodui ištestuoti buvo spaudžiamas mygtukas apvestas raudonai žemiau esančiuose paveikslėliuose.

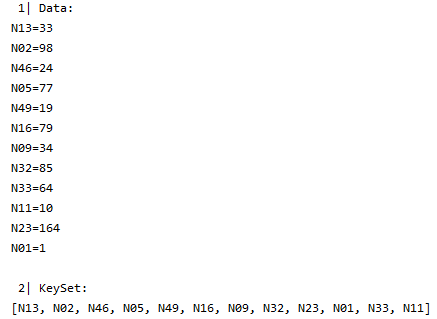




java.util.Set<K> keySet()

**public** Set<K> keySet() {  
 Set<K> keySet = **new** HashSet<K>();  
 **for** (Node<K, V> n : **table**) {  
 **for**(Node<K, V> nn = n; nn != **null**; nn = nn.**next**) {  
 keySet.add(nn.**key**);  
 }  
 }  
 **return** keySet;  
}

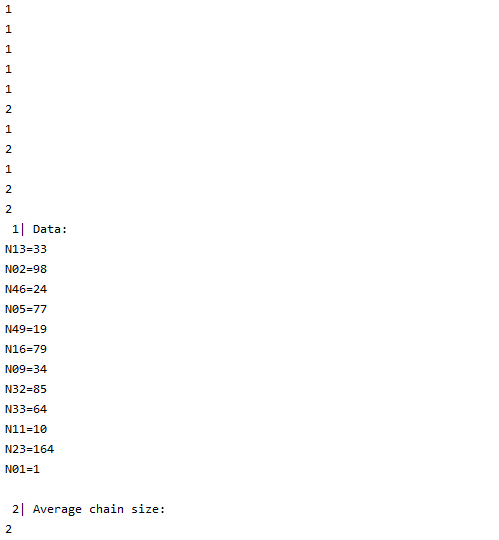
Veikimas matomas toliau įdėtame paveiksliuke:



int averageChainSize()

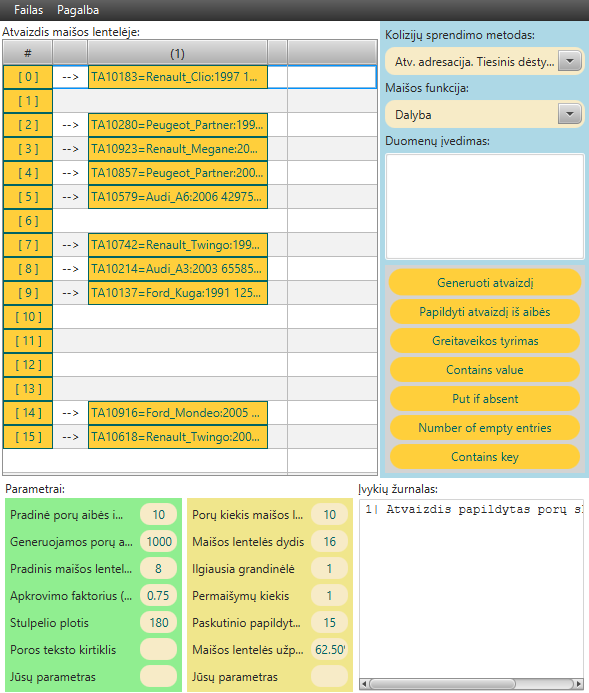
**public int** averageChainSize() {  
 **if** (**chainsCounter** == 0)  
 **return** 0;  
 **return size** / **chainsCounter**;  
}

Toliau esančiam paveikslėlyje parodytas veikimas. Kiekvieną kartą pridėjus elementą paskaičiuojamas vidutinis gradinėlės ilgis.

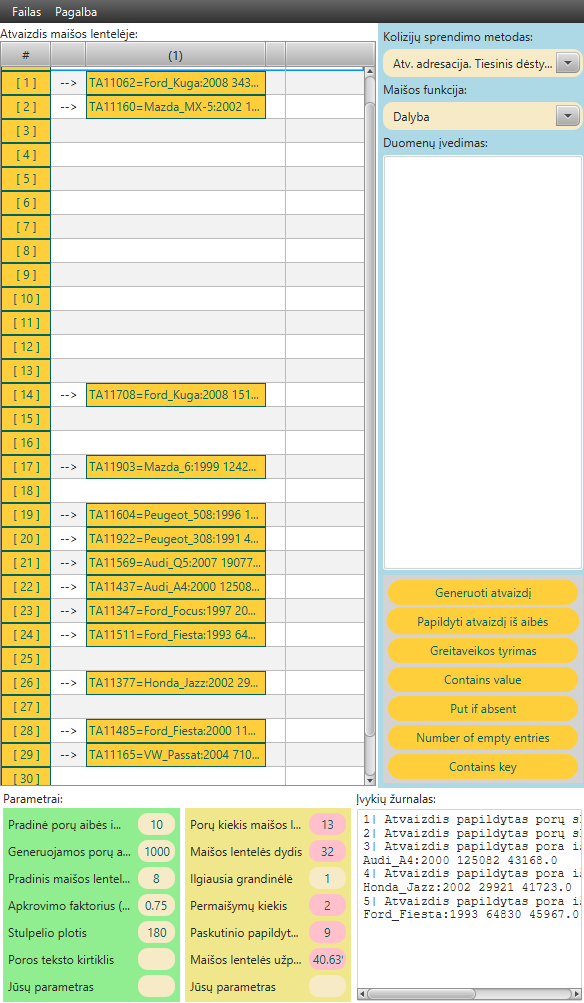


HashMapOa (tiesinis dėstymas)

1 pavyzdys:



2 pavyzdys:



Greitaveikos tyrimas

Matuojant greitaveiką su skirtingu elementų kiekiu siekiant tikslesnių rezultatų su kiekvienu kiekiu greitaveiką skaičiuojama 10 kartų ir imamas vidurkis. Laikai skaičiuojamas sekundėmis.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| kiekis | input | javaContains | customContains | javaRemove | customRemove |
| 5000 | 0.2649 | 0.0015 | 0.0012 | 0.0029 | 0.0015 |
| 10000 | 0.1238 | 0.0012 | 0.0013 | 0.0035 | 0.001 |
| 20000 | 0.1193 | 0.0019 | 0.0023 | 0.0009 | 0.0013 |
| 40000 | 0.1049 | 0.0031 | 0.0054 | 0.0049 | 0.0023 |
| 80000 | 0.0948 | 0.0046 | 0.0061 | 0.0061 | 0.006 |

Rezultatai gan įdomūs. javaContains veikia greičiau, bet panašu, kad mano parašytas remove dažnai lenkia java. Ką vertėtų pastebėti, tai kiekvieną kartą input užtrunka vis mažiau laiko, nors duomenų kiekis tik didėja. Manyčiau, kad tai galai būti dėl to, kad programa optimizuoja save, jeigu kartoja tas pačias operacijas kelis kartus.

Tiriamas atminties sunaudojimas

|  |  |  |
| --- | --- | --- |
| kiekis | Java.hashMap | Custom.hashMap |
| 5000 | 2968808 | 6451568 |
| 10000 | 5930424 | 6089608 |
| 20000 | 5912840 | 6755592 |
| 40000 | 5883536 | 7481808 |
| 80000 | 5870592 | 14804424 |

Iš atminties palyginimo matom, kad java su tuo susitvarko geriau.

Išvados

Laboratorinio darbo metu susipažinau su maišos lentelėmis. Šią struktūrą buvo lengviau suprasti nei medžius. Viskas pavyko.

Laboratorinį darbą atlikau per 14 valandų.