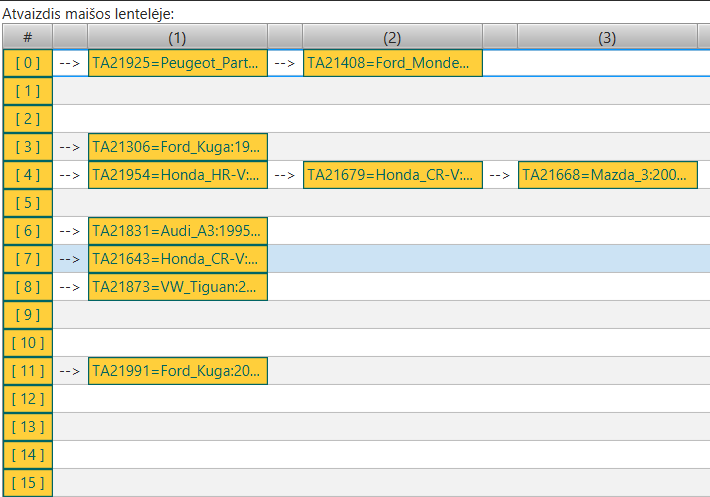
KAUNO TECHNOLOGIJOS UNIVERSITETAS

Duomenų struktūros (P175B014) Laboratorinio darbo Nr.3 ataskaita

Atliko Arnas Švenčionis gr. IFF-8/11

Privalomi metodai:

Pradiniai duomenys:



**Boolean containsValue(Object value)**

public boolean containsValue(Object value){

if(size == 0 || value == null) return false;

for (Node<K, V> node : table) {

if(node != null)

for (Node<K, V> n = node; n != null; n = n.next) {

if(n.value.equals(value))

return true;

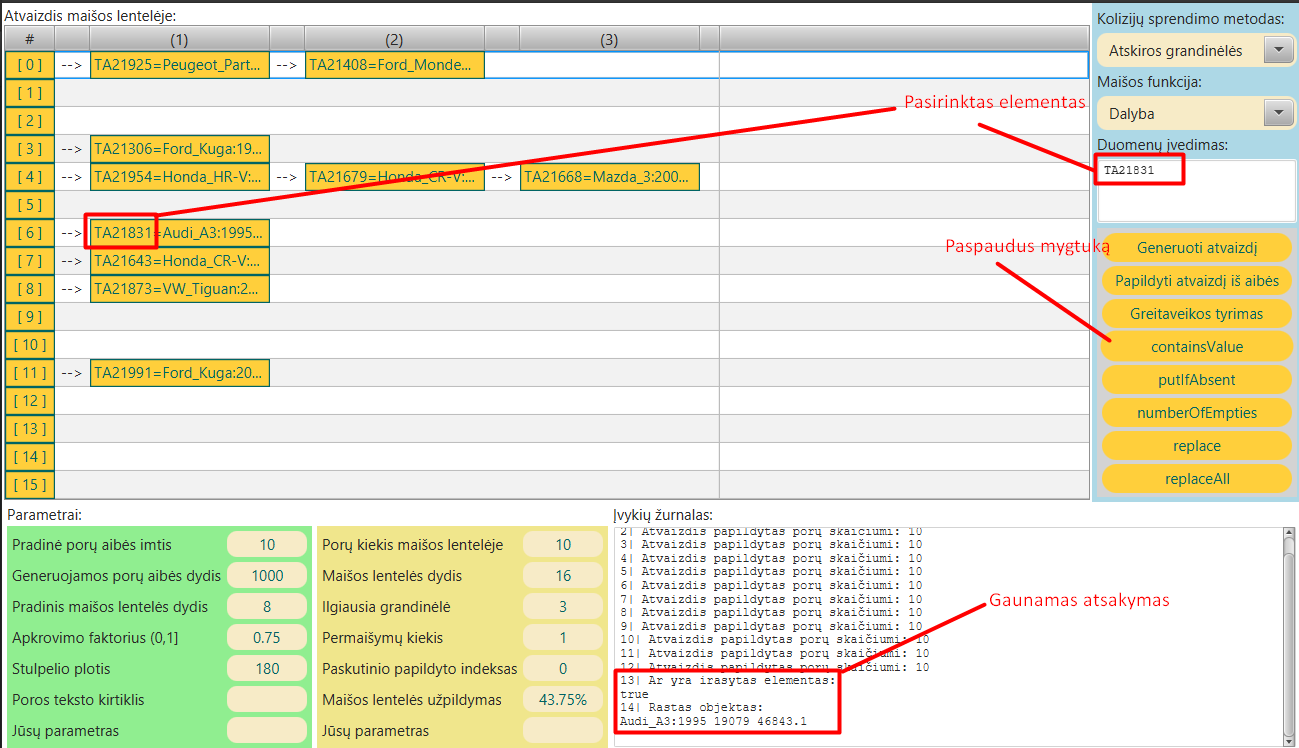
}

}

return false;

}

Rezultatai:



**V putIfAbsent(K key, V value)**

public V putIfAbsent(K key, V value){

if(size == 0 || key == null) return null;

if(get(key) == null){

put(key, value);

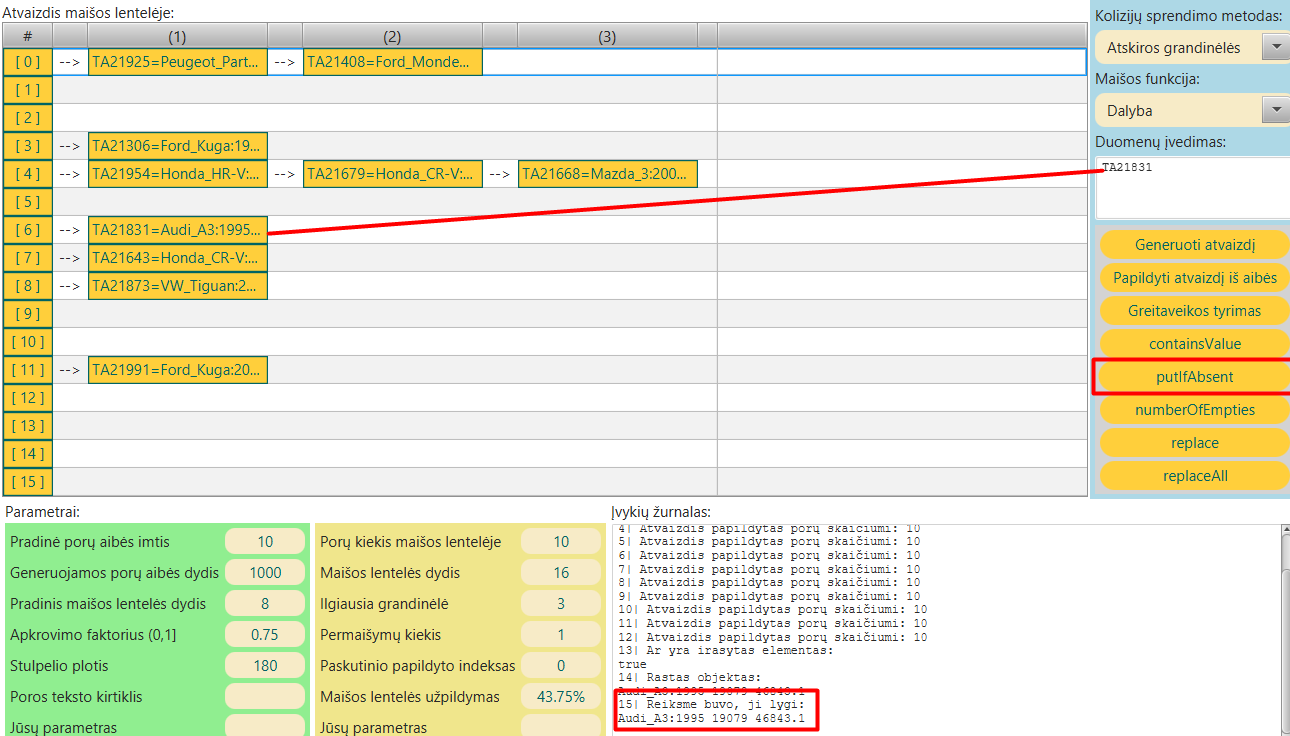
return null;

}

else return get(key);

}

Rezultatai:



**int numberOfEmpties()**

public int numberOfEmpties(){

int temp = 0;

for(Node<K, V> node: table){

if(node == null)

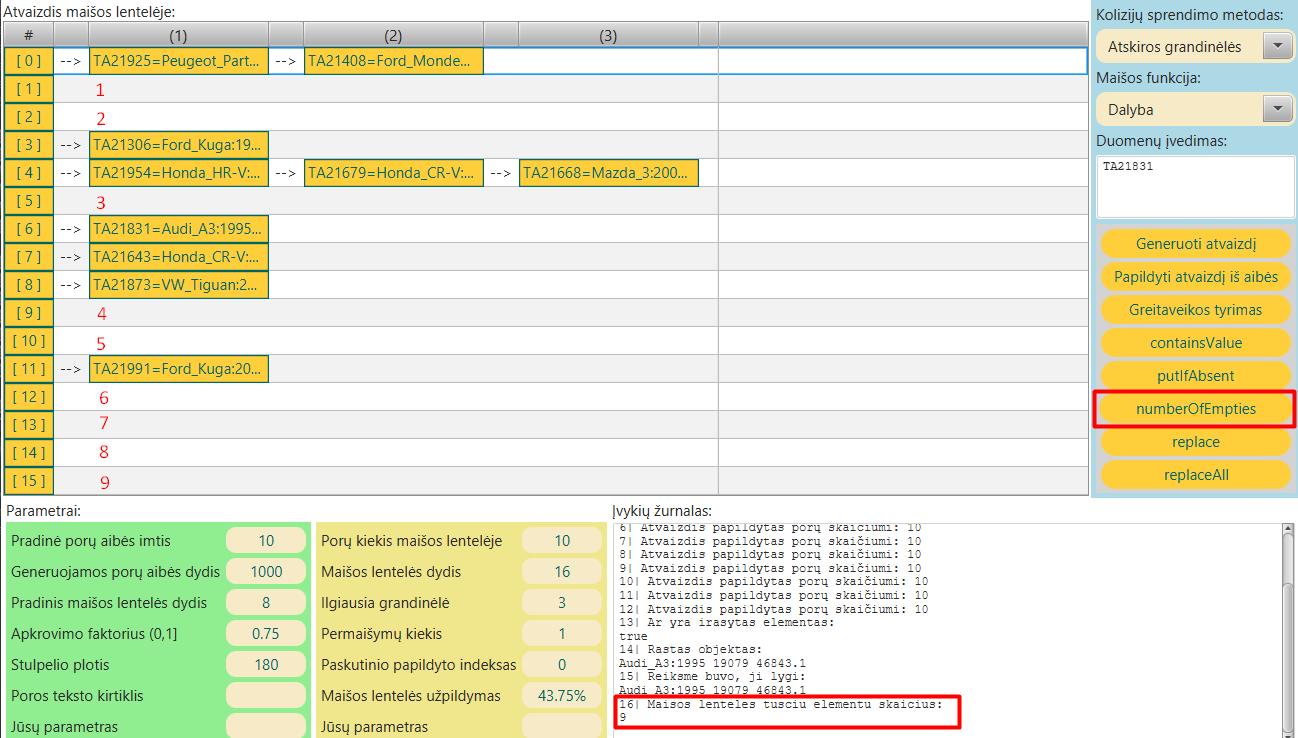
temp++;

}

return temp;

}

Rezultatai:



**Int hash(K key, HashType hashType) (KD)**

private int hash(K key, HashType hashType) {

int h = key.hashCode();

int returnas = -1;

switch (hashType) {

case DIVISION:

returnas = Math.abs(h) % table.length;

break;

case MULTIPLICATION:

double k = (Math.sqrt(5) - 1) / 2;

returnas = (int) (((k \* Math.abs(h)) % 1) \* table.length);

break;

case JCF7:

h ^= (h >>> 20) ^ (h >>> 12);

h = h ^ (h >>> 7) ^ (h >>> 4);

returnas = h & (table.length - 1);

break;

case JCF8:

h = h ^ (h >>> 16);

returnas = h & (table.length - 1);

break;

default:

returnas = Math.abs(h) % table.length;

}

if(table[returnas]!= null)

for(int i = 1; i < table.length;i++){

int ind = returnas + (i\*i);//%table.length;

while(ind >= table.length){

ind -=table.length;

}

//if(ind >= table.length) ind -=table.length;

if(table[ind] == null){

returnas = ind;

break;

}

}

if(returnas == -1){

rehash(table[0]);

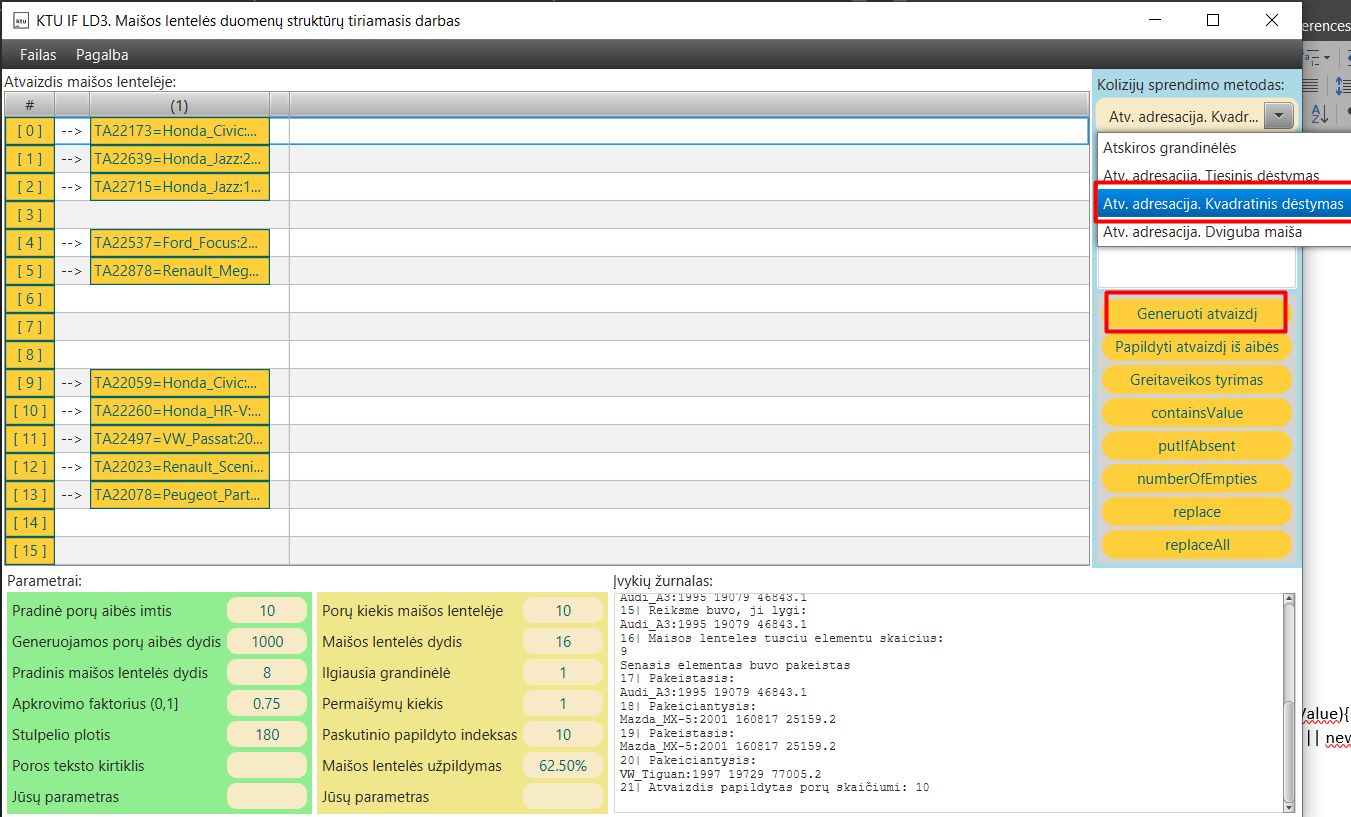
returnas = lastUpdatedChain;

}

return returnas;

}

Rezultatai:



Individualūs metodai:

**replace(K key, V oldValue, V newValue)**

public boolean replace(K key, V oldValue, V newValue){

if(size == 0 || key == null || oldValue == null || newValue == null) return false;

if(!this.contains(key))return false;

if(get(key).equals(oldValue)){

put(key, newValue);

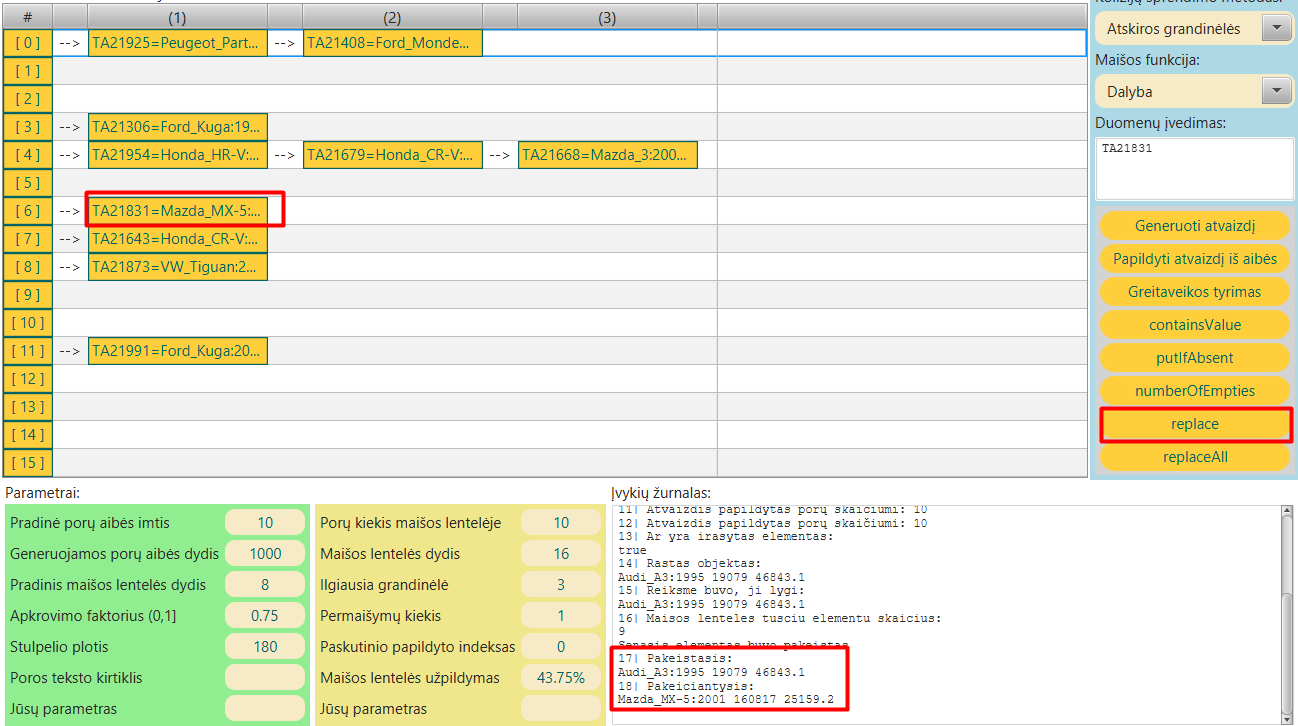
return true;

}

return false;

}

Rezultatai:



**replaceAll(V oldValue, V newValue)**

public void replaceAll(V oldValue, V newValue){

for (Node<K, V> node : table) {

if (node != null) {

for (Node<K, V> n = node; n != null; n = n.next) {

if (n.value.equals(oldValue)) {

put(n.key, newValue);

}

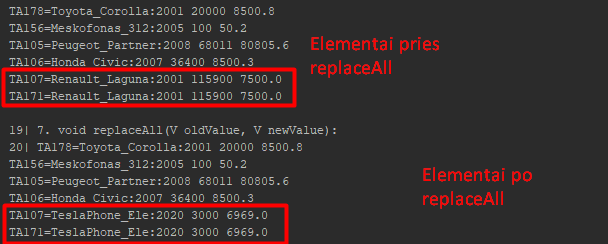
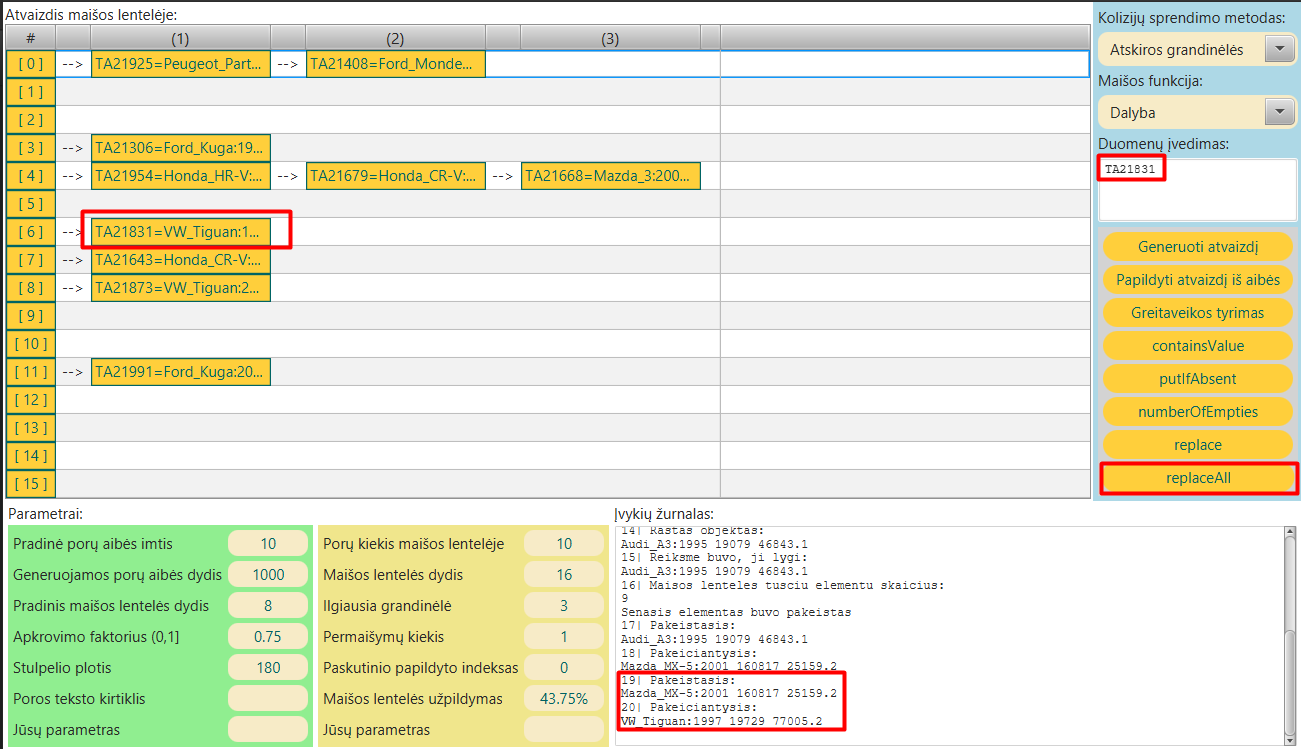
}

}

}

}

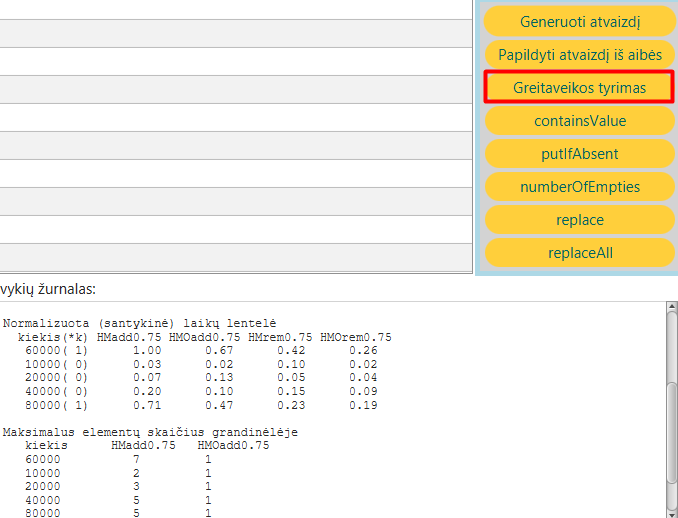
Rezultatai:

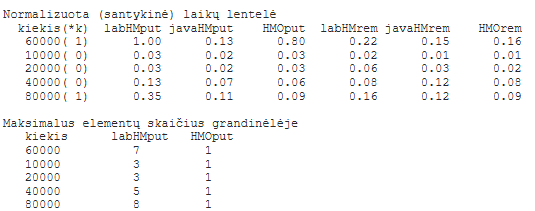
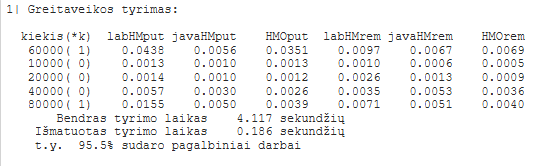


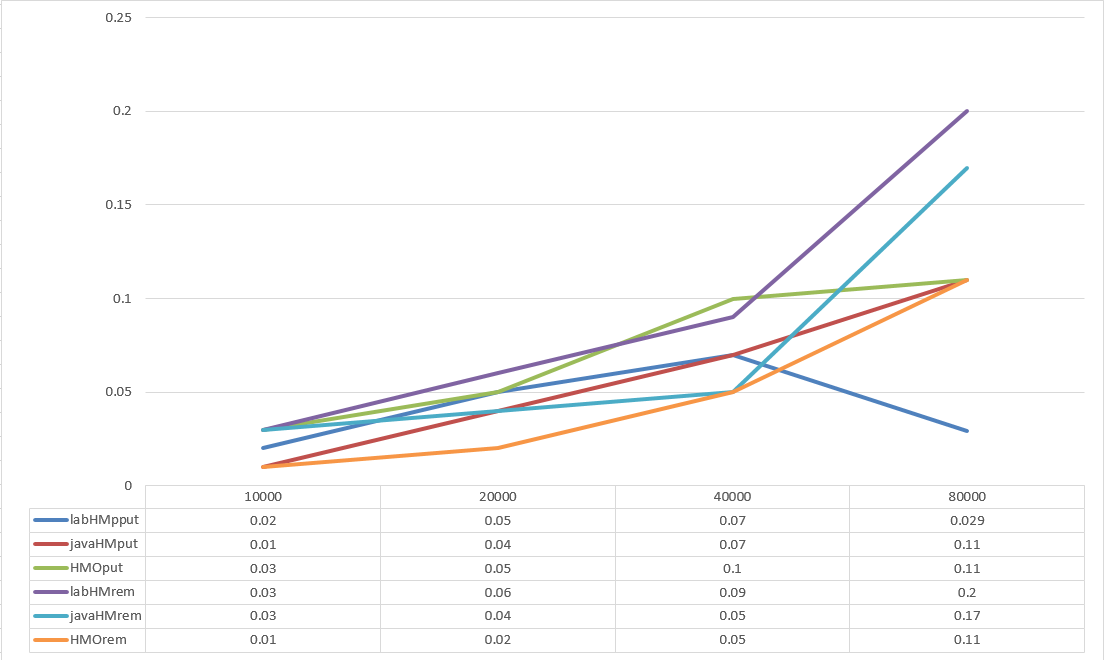
Greitaveika:

Reikia palyginti HashMapOa<String, String> ir HashMap<String, String> put(K key, V value) ir remove(K key) metodus.

Greitaveikos rezultatai:







Išvados:

Laboratorino darbo eiga buvo gana nesudėtinga. Privalomi ir individualūs metodai buvo nesudėtingi, juos pavyko atlikti be kėblmų. Sudėtingiausia dalis buvo HashMapOa klasės sukūrimas ir jos veikimas tiek greitaveikoje, tiek GUI. Ilgai aikinausi ko trūksta, kad ji veiktų ir su ParsableMap interfeisu. Patys maps nebuvo nežinomi – su jais esu anksčiau susidūręs. Tačiau išmokau kas yra Hash lentelės, kaip jos veikia ir kam jos yra naudojamos.

Apytikslė darbo atlikimo trukmė: Individualūs metodai – 1 val., Privalomi metodai – 2 val., Greitaveika – 1 val., GUI pakeitimai – 10 val.