The Ohio State University

PROJECT 3: HASHING IMPLEMENTATION OF MAP

Daniil Gofman

Ansh Pachauri

SW 2: Dev & Dsgn

Paolo Bucci

Yiyang Chen

Shivam Gupta

September 20, 2023

```
 1 import java.util.Iterator;
 2 import java.util.NoSuchElementException;
 3
 4 import components.map.Map;
 5 import components.map.Map1L;
 6 import components.map.MapSecondary;
 7
 8 /**
 9  * {@code Map} represented as a hash table using {@code Map}s for the buckets,
10  * with implementations of primary methods.
11  *
12  * @param <K>
13  *            type of {@code Map} domain (key) entries
14  * @param <V>
15  *            type of {@code Map} range (associated value) entries
16  * @convention <pre>
17  * |$this.hashTable| > 0  and
18  * for all i: integer, pf: PARTIAL_FUNCTION, x: K
19  *     where (0 <= i  and  i < |$this.hashTable|  and
20  *            <pf> = $this.hashTable[i, i+1)  and
21  *            x is in DOMAIN(pf))
22  *   ([computed result of x.hashCode()] mod |$this.hashTable| = i))  and
23  * for all i: integer
24  *     where (0 <= i  and  i < |$this.hashTable|)
25  *   ([entry at position i in $this.hashTable is not null])  and
26  * $this.size = sum i: integer, pf: PARTIAL_FUNCTION
27  *     where (0 <= i  and  i < |$this.hashTable|  and
28  *            <pf> = $this.hashTable[i, i+1))
29  *   (|pf|)
30  * </pre>
31  * @correspondence <pre>
32  * this = union i: integer, pf: PARTIAL_FUNCTION
33  *            where (0 <= i  and  i < |$this.hashTable|  and
34  *                   <pf> = $this.hashTable[i, i+1))
35  *          (pf)
36  * </pre>
37  *
38  * @author Ansh Pachauri
39  *
40  */
41 public class Map4<K, V> extends MapSecondary<K, V> {
42
43     /*
44      * Private members --------------------------------------------------------
45      */
46
47     /**
48      * Default size of hash table.
49      */
50     private static final int DEFAULT_HASH_TABLE_SIZE = 101;
51
52     /**
53      * Buckets for hashing.
54      */
55     private Map<K, V>[] hashTable;
56
57     /**
58      * Total size of abstract {@code this}.
59      */
60     private int size;
61
62     /**
```

```
 63      * Computes {@code a} mod {@code b} as % should have been defined to work.
 64      *
 65      * @param a
 66      *            the number being reduced
 67      * @param b
 68      *            the modulus
 69      * @return the result of a mod b, which satisfies 0 <= {@code mod} < b
 70      * @requires b > 0
 71      * @ensures <pre>
 72      * 0 <= mod  and  mod < b  and
 73      * there exists k: integer (a = k * b + mod)
 74      * </pre>
 75      */
 76     private static int mod(int a, int b) {
 77         assert b > 0 : "Violation of: b > 0";
 78
 79         int mod = a % b;
 80         if (a < 0 && mod != 0) {
 81             mod += b;
 82         }
 83         return mod;
 84     }
 85
 86     /**
 87      * Creator of initial representation.
 88      *
 89      * @param hashTableSize
 90      *            the size of the hash table
 91      * @requires hashTableSize > 0
 92      * @ensures <pre>
 93      * |$this.hashTable| = hashTableSize  and
 94      * for all i: integer
 95      *     where (0 <= i  and  i < |$this.hashTable|)
 96      *   ($this.hashTable[i, i+1) = <{}>)  and
 97      * $this.size = 0
 98      * </pre>
 99      */
100     @SuppressWarnings("unchecked")
101     private void createNewRep(int hashTableSize) {
102         /*
103          * With "new Map<K, V>[...]" in place of "new Map[...]" it does not
104          * compile; as shown, it results in a warning about an unchecked
105          * conversion, though it cannot fail.
106          */
107         this.hashTable = new Map[hashTableSize];
108
109         for (int i = 0; i < hashTableSize; i++) {
110             this.hashTable[i] = new Map1L<K, V>();
111         }
112
113     }
114
115     /*
116      * Constructors ------------------------------------------------------------
117      */
118
119     /**
120      * No-argument constructor.
121      */
122     public Map4() {
123
124         this.createNewRep(DEFAULT_HASH_TABLE_SIZE);
```

```
125
126     }
127
128     /**
129      * Constructor resulting in a hash table of size {@code hashTableSize}.
130      *
131      * @param hashTableSize
132      *            size of hash table
133      * @requires hashTableSize > 0
134      * @ensures this = {}
135      */
136     public Map4(int hashTableSize) {
137
138         this.createNewRep(hashTableSize);
139
140     }
141
142     /*
143      * Standard methods ----------------------------------------------------
144      */
145
146     @SuppressWarnings("unchecked")
147     @Override
148     public final Map<K, V> newInstance() {
149         try {
150             return this.getClass().getConstructor().newInstance();
151         } catch (ReflectiveOperationException e) {
152             throw new AssertionError(
153                     "Cannot construct object of type " + this.getClass());
154         }
155     }
156
157     @Override
158     public final void clear() {
159         this.createNewRep(DEFAULT_HASH_TABLE_SIZE);
160     }
161
162     @Override
163     public final void transferFrom(Map<K, V> source) {
164         assert source != null : "Violation of: source is not null";
165         assert source != this : "Violation of: source is not this";
166         assert source instanceof Map4<?, ?> : ""
167                 + "Violation of: source is of dynamic type Map4<?,?>";
168         /*
169          * This cast cannot fail since the assert above would have stopped
170          * execution in that case: source must be of dynamic type Map4<?,?>, and
171          * the ?,? must be K,V or the call would not have compiled.
172          */
173         Map4<K, V> localSource = (Map4<K, V>) source;
174         this.hashTable = localSource.hashTable;
175         this.size = localSource.size;
176         localSource.createNewRep(DEFAULT_HASH_TABLE_SIZE);
177     }
178
179     /*
180      * Kernel methods ------------------------------------------------------
181      */
182
183     @Override
184     public final void add(K key, V value) {
185         assert key != null : "Violation of: key is not null";
186         assert value != null : "Violation of: value is not null";
```

```
187         assert !this.hasKey(key) : "Violation of: key is not in DOMAIN(this)";
188
189         // Calculate the hash code for the key
190         int hashKey = key.hashCode();
191         // Determine the bucket where the key-value pair should be stored
192         int bucket = mod(hashKey, this.hashTable.length);
193         // Add the key and value to the appropriate bucket in the hash table
194         this.hashTable[bucket].add(key, value);
195         // Increase the size of the hash table
196         this.size++;
197     }
198
199     @Override
200     public final Pair<K, V> remove(K key) {
201         assert key != null : "Violation of: key is not null";
202         assert this.hasKey(key) : "Violation of: key is in DOMAIN(this)";
203
204         // Calculate the hash code for the key
205         int hashKey = key.hashCode();
206
207         // Calculate the bucket where the key should be located
208         int bucket = mod(hashKey, this.hashTable.length);
209
210         // Remove the key-value pair from the hashTable's bucket
211         Pair<K, V> temp = this.hashTable[bucket].remove(key);
212
213         // Decrement the size of the hashTable as a key-value pair was removed
214         this.size--;
215
216         // Return the removed key-value pair
217         return temp;
218     }
219
220     @Override
221     public final Pair<K, V> removeAny() {
222         assert this.size() > 0 : "Violation of: this /= empty_set";
223
224         // Find a non-empty bucket in the hash table
225         int i = 0;
226         while (this.hashTable[i].size() == 0) {
227             i++;
228         }
229
230         // Remove an arbitrary entry from the selected bucket
231         Pair<K, V> removedPair = this.hashTable[i].removeAny();
232
233         // Decrement the size of the hash table
234         this.size--;
235
236         // Return the removed key-value pair
237         return removedPair;
238     }
239
240     @Override
241     public final V value(K key) {
242         assert key != null : "Violation of: key is not null";
243
244         // Ensure that the key is present in the hash table
245         assert this.hasKey(key) : "Violation of: key is in DOMAIN(this)";
246
247         // Initialize variables for the hashed key and bucket index
248         int hashKey = key.hashCode();
```

```
249         int bucket = mod(hashKey, this.hashTable.length);
250
251         // Retrieve the value associated with the key from the hash table
252         V val = this.hashTable[bucket].value(key);
253
254         // Return the retrieved value
255         return val;
256     }
257
258     @Override
259     public final boolean hasKey(K key) {
260         assert key != null : "Violation of: key is not null";
261
262         // Calculate the hash code for the key
263         int hashKey = key.hashCode();
264
265         // Determine the bucket index using the modulo operation
266         int bucket = mod(hashKey, this.hashTable.length);
267
268         // Check if the hash table's bucket at the calculated index contains the key
269         boolean result = this.hashTable[bucket].hasKey(key);
270
271         // Return the result indicating whether the key is present in the hash table
272         return result;
273     }
274
275     @Override
276     public final int size() {
277
278         int size = this.size;
279         return size;
280     }
281
282     @Override
283     public final Iterator<Pair<K, V>> iterator() {
284         return new Map4Iterator();
285     }
286
287     /**
288      * Implementation of {@code Iterator} interface for {@code Map4}.
289      */
290     private final class Map4Iterator implements Iterator<Pair<K, V>> {
291
292         /**
293          * Number of elements seen already (i.e., |~this.seen|).
294          */
295         private int numberSeen;
296
297         /**
298          * Bucket from which current bucket iterator comes.
299          */
300         private int currentBucket;
301
302         /**
303          * Bucket iterator from which next element will come.
304          */
305         private Iterator<Pair<K, V>> bucketIterator;
306
307         /**
308          * No-argument constructor.
309          */
310         Map4Iterator() {
```

```
311             this.numberSeen = 0;
312             this.currentBucket = 0;
313             this.bucketIterator = Map4.this.hashTable[0].iterator();
314         }
315 
316         @Override
317         public boolean hasNext() {
318             return this.numberSeen < Map4.this.size;
319         }
320 
321         @Override
322         public Pair<K, V> next() {
323             assert this.hasNext() : "Violation of: ~this.unseen /= <>";
324             if (!this.hasNext()) {
325                 /*
326                  * Exception is supposed to be thrown in this case, but with
327                  * assertion-checking enabled it cannot happen because of assert
328                  * above.
329                  */
330                 throw new NoSuchElementException();
331             }
332             this.numberSeen++;
333             while (!this.bucketIterator.hasNext()) {
334                 this.currentBucket++;
335                 this.bucketIterator = Map4.this.hashTable[this.currentBucket]
336                         .iterator();
337             }
338             return this.bucketIterator.next();
339         }
340 
341         @Override
342         public void remove() {
343             throw new UnsupportedOperationException(
344                     "remove operation not supported");
345         }
346 
347     }
348 
349 }
350 
```