Bucketsort

Weiterführende Literatur:

- *Algorithmen und Datenstrukturen: Tafelübung 11, WS 2018/19*, Seite 65-58
- *Wikipedia-Artikel „Bucketsort“*

BucketSort ; Sortieren durch einfaches Fachverteilen • Voraussetzung: • Werte sind Elemente einer bekannten, endlichen Multimenge • auf der Multimenge ist eine Totalordnung definiert • für jeden möglichen Wert steht ein Fach zur Verfügung • Vorgehen: 1. alle Elemente in das jeweils passende entsprechende Fach einsortieren 2. die Elemente aus den Fächern der Reihe nach einsammeln

BucketSort ist nicht-vergleichsbasiert, denn das Verfahren vergleicht nie zwei Elemente direkt miteinander.

Eigenschaften von BucketSort: • Laufzeitkomplexität: O(n + m) (im Best-, Average- und Worst-Case) • bei n zu sortierenden Werten • und m Fächern (also m möglichen Werten) • benötigt Speicher für die Fächer • Stabilität einfach zu erreichen ; out-of-place Hinweis

In einer Variation des Algorithmus kann man auch jeweils ein Bucket für mehrere mögliche Werte haben. In diesem Fall werden die einzelnen Buckets beim Einfügen mit einem weiteren Sortierverfahren (z.B. InsertionSort) sortiert.[1]

```
3  import java.util.ArrayList;
4  import java.util.Comparator;
5  import java.util.LinkedList;
6  import java.util.List;
7
8  interface Sorter<T> {
9
10   List<T> sort(List<T> arrayToSort);
11 }
12
13 /**
14  * https://github.com/eugenp/tutorials/blob/master/algorithms-
   ↪  sorting/src/main/java/com/baeldung/algorithms/bucketsort/IntegerBucketSorter.java
15  */
16 public class Bucket implements Sorter<Integer> {
17
18   private final Comparator<Integer> comparator;
19
20   public Bucket(Comparator<Integer> comparator) {
21     this.comparator = comparator;
22   }
23
24   public Bucket() {
25     comparator = Comparator.naturalOrder();
26   }
27
28   public List<Integer> sort(List<Integer> arrayToSort) {
29
30     List<List<Integer>> buckets = splitIntoUnsortedBuckets(arrayToSort);
31
```

[1] *Algorithmen und Datenstrukturen: Tafelübung 11, WS 2018/19*, Seite 65-58.

```
32      for (List<Integer> bucket : buckets) {
33        bucket.sort(comparator);
34      }
35
36      return concatenateSortedBuckets(buckets);
37    }
38
39    private List<Integer> concatenateSortedBuckets(List<List<Integer>> buckets) {
40      List<Integer> sortedArray = new LinkedList<>();
41      for (List<Integer> bucket : buckets) {
42        sortedArray.addAll(bucket);
43      }
44      return sortedArray;
45    }
46
47    private List<List<Integer>> splitIntoUnsortedBuckets(List<Integer> initialList)
      ↪ {
48
49      final int max = findMax(initialList);
50      final int numberOfBuckets = (int) Math.sqrt(initialList.size());
51
52      List<List<Integer>> buckets = new ArrayList<>();
53      for (int i = 0; i < numberOfBuckets; i++)
54        buckets.add(new ArrayList<>());
55
56      // distribute the data
57      for (int i : initialList) {
58        buckets.get(hash(i, max, numberOfBuckets)).add(i);
59      }
60      return buckets;
61
62    }
63
64    private int findMax(List<Integer> input) {
65      int m = Integer.MIN_VALUE;
66      for (int i : input) {
67        m = Math.max(i, m);
68      }
69      return m;
70    }
71
72    private static int hash(int i, int max, int numberOfBuckets) {
73      return (int) ((double) i / max * (numberOfBuckets - 1));
74    }
75
76  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/sortier/Bucket.java`

Literatur

[1] *Algorithmen und Datenstrukturen: Tafelübung 11, WS 2018/19*. `https://www.studon.fau.de/file2567217_download.html`. FAU: Lehrstuhl für Informatik 2 (Programmiersysteme).

[2] *Wikipedia-Artikel „Bucketsort“*. `https://de.wikipedia.org/wiki/Bucketsort`.