# Schnelles Sortieren

Aufgabenblatt 6

07. Mai 2017

**Abstract**

Es sollten ein besonders schnelles Sortierverfahren für eine Spezielle Aufgabenstellung implementiert werden. Dieses Verfahren sollte schneller sein als das zuvor erstellte Quicksort-Verfahren.

Die Problemgrößen gingen wieder von

Das besondere war der Schlüssel, dieser sollte in einem festgelegten Intervall liegen.

Dabei sollten die Zahlen des Intervalls einigermaßen gleichmäßig verteilt sein.

Es ergibt sich also bei z.B. eine Problemgröße von ein Intervall

Somit kommen tausend Zahlen von 7000 bis 8000 als möglicher Schlüssel in Frage.

Um die zuvor festgelegte Eigenschaft des Schlüssels wieder aufzugreifen, die Zahlen seien einigermaßen verteilt, kann man nun die Anzahl der Möglichen Schlüssel auf die Problemgröße Abbilden.

Dies ist somit die Größe Intervalls, in dem der Schlüssel variieren kann.

Das hat zur Folge, dass ein Schlüssel an der dritten Ziffernstelle nur einmal vorkommt.

z.B.

{7062 7187 7256 7378 7451 7557 7663 7779 7873 7928}

Oder

{70040 70147 70229 70384 70498 70564 70681 70760 70861 70948 71001 71135 71254 71338 71484 71532 71669 71730 71892 71975 72018 72188 72266 72358 72498 72551 72649 72728 72895 72983 73060 73177 73228 73399 73468 73597 73698 73748 73826 73953 74032 74129…}

Sortierverfahren welche mit einem vorher bekannten Intervall von Schlüsseln funktioniert sind:

Countingsort und Radixsort.

**Radixsort**

Vorraussetzung

Bei Radixsort wird davon ausgegangen, dass die Schlüssel der zu sortierenden Daten nur aus Zeichen eines endlichen [Alphabets](https://de.wikipedia.org/wiki/Alphabet_(Informatik)) bestehen. Zusätzlich muss eine [Totalordnung](https://de.wikipedia.org/wiki/Totalordnung) zwischen den Zeichen des Alphabets bestehen.

Eine zweite Voraussetzung ist, dass die Länge der Schlüssel durch eine von vornherein bekannte Konstante begrenzt ist, da die Anzahl der Stellen pro Schlüssel eine entscheidende Auswirkung auf die Linearität des Laufzeitverhaltens hat.

Vorgehensweise

Radixsort besteht aus zwei Phasen, die immer wieder abwechselnd durchgeführt werden. Die *Partitionierungsphase* dient dazu, die Daten auf Fächer aufzuteilen, während in der *Sammelphase* die Daten aus diesen Fächern wieder aufgesammelt werden. Beide Phasen werden für jede Stelle der zu sortierenden Schlüssel einmal durchgeführt.

**Partitionierungsphase**

In dieser Phase werden die Daten in die vorhandenen Fächer aufgeteilt, wobei für jedes Zeichen des zugrundeliegenden Alphabets ein Fach zur Verfügung steht. In welches Fach der gerade betrachtete Schlüssel gelegt wird, wird durch das an der gerade betrachteten Stelle stehende Zeichen bestimmt. So wird zum Beispiel die Zahl 352 in das Fach 3 gelegt, wenn gerade die dritte Stelle von hinten betrachtet wird (und sofern 10 als Basis für die Zahldarstellung, d.h. als Radix, gewählt wurde).

**Sammelphase**

Nach der Aufteilung der Daten in Fächer in Phase 1 werden die Daten wieder eingesammelt und auf einen Stapel gelegt. Hierbei wird so vorgegangen, dass zuerst alle Daten aus dem Fach mit der niedrigsten Wertigkeit eingesammelt werden, wobei die Reihenfolge der darin befindlichen Elemente nicht verändert werden darf. Danach werden die Elemente des nächsthöheren Faches eingesammelt und an die schon aufgesammelten Elemente angefügt. Dies führt man fort, bis alle Fächer wieder geleert wurden.

Diese beiden Phasen werden nun für jede Stelle der Schlüssel wiederholt, wobei mit der letzten Stelle begonnen wird und in der letzten Iteration die erste Stelle zum Aufteilen verwendet wird. Nach dem Aufsammeln für die erste Stelle der Schlüssel sind die Daten aufsteigend sortiert.

Alternativ können die Stellen des Schlüssels auch von der höchstwertigen her bearbeitet werden, dabei ist jedoch zu beachten, dass in der Sammelphase die Elemente vom höchstwertigen Fach her vereint werden.[[1]](#endnote-1)

**Countingsort** (von engl. *count* „zählen“)

ist ein stabiles Sortierverfahren, das eine gegebene Folge von {\displaystyle n} natürlichen Zahlen aus einem beschränkten Intervall mit {\displaystyle k} Elementen mit linearem Zeitaufwand (Problemkomplexität {\displaystyle \textstyle O(n+k)}) sortiert. Der Algorithmus arbeitet nicht vergleichsbasiert, sondern zählt die Zahlen der Eingabe - er arbeitet also adressbasiert. Im Vergleich zum vergleichenden Sortieren mit der bestmöglichen Komplexität {\displaystyle \textstyle O(n\log n)} ergibt sich ein Vorteil, wenn die Intervalllänge {\displaystyle k} sehr klein ist gegenüber der Anzahl der zu sortierenden Element {\displaystyle n}.

Gut geeignetes Beispiel: Sortieren aller Einwohner Bayerns (n = 12,85 Mio) nach ihrem Alter (in vollendeten Jahren: k = 0..150).

Schlechtes/nicht umsetzbares Beispiel: Sortieren der Schüler einer Grundschule (n = 300) nach ihrem Nachnamen (Zeichenkette mit max. Länge 40 aus allen möglichen Buchstaben, Umlauten und Sonderzeichen; k ≈ 3540)[[2]](#endnote-2)

1. https://de.wikipedia.org/wiki/Radixsort [↑](#endnote-ref-1)
2. https://de.wikipedia.org/wiki/Countingsort [↑](#endnote-ref-2)