Suchalgorithmen

Weiterführende Literatur:

- Saake und Sattler, *Algorithmen und Datenstrukturen*, Seite 120-123
- *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 17
- *Wikipedia-Artikel „Binäre Suche“*

Suchen Allgemein

Eine der wichtigsten und häufigsten Aufgaben in der Informatik ist das Suchen. Die Suche in unsortierter Datenmenge ist langwierig. Es gibt keine Möglichkeit, das ideal zu gestalten. Das *Suchen in sortierten Daten* ist sinnvoll, denn es kann optimiert werden.

Suchen in sortierten Daten

Möglichkeiten:

Sequenzielle Suche: Es wird eine Folge vom *ersten Element* an durchlaufen. Die Suche ist beendet, wenn das gesuchte Element gefunden ist oder die *gesamte Folge* ohne Ergebnis durchlaufen wurde.

ersten Element

gesamte Folge

Binäre Suche: Wir „verkleinern“ die zu durchsuchenden Menge durch Herausgreifen eines Elements aus der Folge und vergleichen, ob mein gesuchtes Element vor oder nach diesem Element in der Folge liegt. Es muss nur noch eine Hälfte durchsucht werden. In dieser fahren wir analog fort.

Ein Bewertungskriterium für das Suchverfahren ist der Berechnungsaufwand. Wie viele Schritte sind durchschnittlich nötig, um eine Folge zu durchlaufen?[1]

Binäre Suche

Die binäre Suche ist ein Algorithmus, der auf einem Feld (also meist „in einer Liste“) sehr effizient ein gesuchtes Element findet bzw. eine zuverlässige Aussage über das Fehlen dieses Elementes liefert. Voraussetzung ist, dass die Elemente in dem Feld *sortiert* sind. Der Algorithmus basiert auf einer einfachen Form des Schemas *Teile und Herrsche*, zugleich stellt er auch einen *Greedy-Algorithmus* dar. Ordnung und spätere Suche müssen sich auf denselben Schlüssel beziehen.[2]

Iterativer Ansatz

```
7   public final static int KEIN_SCHLÜSSEL = -1;
8
9   public static int suche(int[] zahlen, int schlüssel) {
10    int links = 0, rechts = zahlen.length - 1;
```

[1] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 16.
[2] *Wikipedia-Artikel „Binäre Suche“*.

```
11      while (links <= rechts) {
12        int mitte = (links + rechts) / 2;
13        if (zahlen[mitte] == schlüssel) {
14          return mitte;
15        } else if (zahlen[mitte] > schlüssel) {
16          rechts = mitte - 1;
17        } else {
18          links = mitte + 1;
19        }
20      }
21      return KEIN_SCHLÜSSEL;
22    }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/suche/BinaereSuche.java`

Rekursiver Ansatz

```
24    private static int sucheRekursiv(int[] zahlen, int links, int rechts, int
      ↪  schlüssel) {
25      if (links > rechts) {
26        return KEIN_SCHLÜSSEL;
27      }
28      int mitte = links + (rechts - links) / 2;
29      if (schlüssel < zahlen[mitte]) {
30        return sucheRekursiv(zahlen, links, mitte - 1, schlüssel);
31      }
32      if (schlüssel > zahlen[mitte]) {
33        return sucheRekursiv(zahlen, mitte + 1, rechts, schlüssel);
34      }
35      return mitte;
36    }
37
38    public static int sucheRekursiv(int[] zahlen, int schlüssel) {
39      return sucheRekursiv(zahlen, 0, zahlen.length - 1, schlüssel);
40    }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/suche/BinaereSuche.java`

Komplexität

Bei der Binären muss im schlechtesten Fall nicht die gesamte Folge durchsucht werden. Nach dem ersten Teilen der Folge bleiben nur noch $\frac{n}{2}$ Elemente, nach dem zweiten Schritt $\frac{n}{4}$, nach dem dritten $\frac{n}{8}$ usw. Allgemein bedeutet dies, dass im i-ten Durchlauf maximal $\frac{n}{2^i}$ Elemente zu durchsuchen sind. Entsprechend werden $\log_2 n$ Schritte benötigt.[3]

Um in einem Feld mit n Einträgen die An- oder Abwesenheit eines Schlüssels festzustellen, werden maximal $\lceil \log_2(n+1) \rceil = \lfloor \log_2(n) + 1 \rfloor$ Vergleichsschritte benötigt. Somit hat die binäre Suche in der Landau-Notation ausgedrückt die Zeitkomplexität $\mathcal{O}(\log\ n)$.[4]

[3] Saake und Sattler, *Algorithmen und Datenstrukturen*, Seite 122.
[4] *Wikipedia-Artikel „Binäre Suche“*.

Literatur

[1] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2. Sortieren, Suchen, Komplexität*. https://www.studon.fau.de/file2566441_download.html.

[2] Gunter Saake und Kai-Uwe Sattler. *Algorithmen und Datenstrukturen. Eine Einführung in Java*. 2014.

[3] *Wikipedia-Artikel „Binäre Suche“*. https://de.wikipedia.org/wiki/Binäre_Suche.