6.1 Allgemeines zum Suchen

Suchen gehört zu den häufigsten Aufgaben, die Computer im Alltag übernehmen – ob beim Auffinden von Internetadressen, Telefonnummern, Artikeln oder Dateien. Suchalgorithmen spielen daher eine zentrale Rolle in der Informatik.

Grundsätzlich unterscheidet man beim Suchen nach dem Zustand der Daten, die durchsucht werden sollen. Es ergeben sich dabei folgende typische Situationen:

1. Die Daten sind **unsortiert**.
2. Die Daten sind zwar sortiert, das Suchkriterium entspricht aber nicht dem Sortierkriterium.
3. Die Daten sind zwar sortiert, das Suchkriterium entspricht dem Sortierkriterium, allerdings sind die Daten nicht sehr suchfreundlich angeordnet.
4. Die Daten sind suchfreundlich sortiert, zum Beispiel in einem binären Suchbaum oder einem B-Baum, und das Suchkriterium entspricht dem Sortierkriterium.

Zu 1

Sind die Daten unsortiert, kann man sie nur ***sequentiell*** oder ***linear*** durchsuchen. Dabei werden die Elemente *nacheinander* durchsucht – beginnend beim ersten Eintrag, bis entweder das gesuchte Element gefunden oder das Ende der Daten erreicht ist.

Zu 2

Eine Sammlung von Büchern ist nach dem **Kaufdatum** sortiert, es wird aber nach einem bestimmten **Autor** gesucht. Sortierkriterium (Kaufdatum) und Suchkriterium (Autor) stimmen nicht überein. Daher kann nur sequentiell gesucht werden.

Zu 3

In einer Bibliothek stehen in 20 Regalen 10.000 Bücher, sortiert nach Autoren. Das erste Regal enthält die Autoren Aa bis Bm, das zweite Regal Bn bis Ce, das dritte Cf bis Da und so weiter.

Zu 4

In einer Bibliothek stehen in 26 Regalen 10.000 Bücher, sortiert nach Autoren. Das erste Regal enthält alle Autoren mit dem Anfangsbuchstaben A, das zweite Regal B, das dritte C und so weiter.

6.1.1 Vier Beispiele aus dem Alltag

Beispiel 1 aus dem Alltag - Interpolationssuche

Sie stehen vor dem Bücherregal, in dem Sie all die Romane aufbewahren, die Sie seit Jahren gesammelt haben. Die Romane sind alphabetisch nach Autor bzw. Autorin geordnet. Sie wollen nun ein Buch von Karl Melcher lesen. Dann werden Sie sicherlich nicht oben links im Regal bei "A" anfangen zu suchen, aber auch nicht unten rechts bei "Z", sondern werden gezielt irgendwo in der Mitte des Regals mit der Suche anfangen.

Sie haben quasi in Gedanken ***interpoliert***, wo im Regal der beste Einstiegspunkt für Ihre Suche ist.

Beispiel 2 aus dem Alltag - Indexsuche

Sie befinden sich in der Uni-Bibliothek und suchen ein ganz bestimmtes Buch. Leider kennen Sie nicht den oder die Autoren, sondern nur den Titel des Buches: *"Anwendung diverser Suchverfahren im Alltag"*. Früher hatte man für solche Zwecke Karteikästen. In einem Karteikasten waren die Karteikarten alphabetisch nach Autoren sortiert, in dem anderen Karteikasten hatte man die Karten systematisch nach Fachgebieten sortiert.

Sie gehen also zum Karteikasten mit den Fachgebieten, suchen nach dem Reiter "Informatik" und blättern dann - sequentiell - die Karten durch, bis Sie das gesuchte Buch gefunden haben - oder bis Sie beim nächsten Reiter angekommen sind und frustriert feststellen müssen, dass das Buch nicht in der Bibliothek vorhanden ist.

Aber wir nehmen einmal an, dass Sie die Karteikarte mit dem Buch gefunden haben. Auf der Karteikarte ist dann der Standort (Regal, Fach, Nummer) vermerkt, an dem das Buch steht. Diese Information ist dann Ihr ***Index***. Sie stehen auf, gehen zum Regal und suchen dann - wieder sequentiell - dort nach dem Buch.

Beispiel 3 aus dem Alltag - Binäre Suche

Sie wollen den Papierkorb Ihres Rechners löschen, das Betriebssystem unterbricht den Löschvorgang aber, weil Sie für irgendeine Datei keine Rechte besitzen.

Sie wollen jetzt diese Datei identifizieren. Sehr zu Ihrem Ärger befinden sich aber 2.600 Dateien im Papierkorb.

Die beste Strategie ist jetzt eine binäre Suche. Sie markieren die ersten 1.300 Dateien und versuchen, diese zu löschen. Wenn es nicht funktioniert, wissen Sie, dass sich die Problemdatei in diesen 1.300 Dateien befindet.

Also markieren Sie jetzt die obere Hälfte dieser Dateien, also die ersten 650. Der Löschversuch gelingt jetzt. Damit wissen Sie, dass sich die problematische Datei in den zweiten 650 Dateien befindet.

Diese halbieren Sie wieder und versuchen, die oberen 325 Dateien zu löschen und so weiter.

Auf diese Weise kommen Sie nach recht wenigen Schritten zur problematischen Datei und können versuchen, diese auf andere Weise zu löschen.

Kleine Kritik an diesem Beispiel

Handelt es sich bei dem Beispiel 3 wirklich um eine binäre Suche?

Nein, bei einer binären Suche müssen die Daten sortiert vorliegen, was hier nicht der Fall ist. Zwar sind die Dateien irgendwie sortiert, beispielsweise nach Name oder Erstellungsdatum, aber zum Suchen wird diese Sortierung nicht genutzt (Suchkriterium entspricht nicht dem Sortierkriterium).

Vielmehr handelt es sich um das ***Prinzip des binären Eingrenzens***: Wir teilen die Menge und testen, in welcher Hälfte die Problemdatei steckt.

Beispiel 4 aus dem Alltag - Sequentielle Suche

Sie haben von Ihrem Vater eine große Platten-Sammlung geerbt. Ihr Vater war ein sehr ordentlicher Mensch und hatte seine Platten-Sammlung gut sortiert. Allerdings nicht alphabetisch nach Interpreten, sondern nach dem Kaufdatum.

Sie wollen nun nachschauen, ob auch eine Platte von Ihrer Lieblingsband "The Luftschiff" dabei ist.

Wo fangen Sie an mit der Suche? Sie wissen, dass die ersten Platten dieser Gruppe schon 1984 erschienen sind, also zu der Zeit, als Ihr Vater mit dem Plattensammeln anfing. Aber vielleicht hat er diese Platte auch in den 90er Jahren gekauft oder um 2010 oder erst kurz vor seinem Tod im Jahre 2024.

Es ist also völlig egal, ob Sie ganz vorne oder ganz hinten in der Sammlung mit Ihrer Suche anfangen - aber anfangen müssen Sie ja schließlich.

Obwohl die Plattensammlung hoch geordnet ist, müssen Sie hier eine sequentielle Suche durchführen, da das Ordnungssystem nicht Ihrem Suchkriterium entspricht. Für Ihre Suche ist die Plattensammlung völlig unsortiert.

Hätte Ihr Vater mehrere Register oder Karteikästen angelegt, wäre die Suche kein Problem mehr: Sie suchen im Interpreten-Register nach "The Luftschiff" und finden dort das Kaufdatum. Wenn die Platte im Januar 2010 gekauft wurde, müssen Sie nicht ganz vorn mit der Suche anfangen, aber auch nicht ganz hinten, sondern vielleicht im hinteren Viertel der Sammlung. Dort können Sie dann binär oder sequentiell nach der Platte suchen.

6.2 Sequentielle Suche

Bei der sequentiellen (linearen) Suche wird eine Liste Element für Element durchlaufen, bis das gesuchte Element gefunden ist oder das Ende der Liste erreicht wird.

Dieses Verfahren setzt weder eine Sortierung noch eine spezielle Datenstruktur voraus und ist daher universell einsetzbar. Der Algorithmus ist sehr einfach zu implementieren und eignet sich insbesondere für kleine Datenmengen.

Auch große Datenmengen müssen sequentiell durchsucht werden, wenn sie nach einem anderen Kriterium sortiert sind, als es dem Suchkriterium entspricht.

Beispiel: Datensammlung mit 8.000.000 Datensätzen von Personen, die nach Namen sortiert sind. Wenn man jetzt alle Menschen finden will, die am 1. Januar 1991 geboren sind, müsste man den gesamten Datenbestand sequentiell durchsuchen.

6.2.1 Zeitverhalten der sequentiellen Suche

Sei **n** die Anzahl der Elemente in der Datensammlung und **S** die Suchzeit.

Günstigster Fall (Best Case): S = 1

Das gesuchte Element befindet sich an erster Stelle.

Ungünstigster Fall (Worst Case): S = n

Das gesuchte Element steht am Ende der Liste oder kommt gar nicht vor.

Durchschnittlicher Fall (Average case): S = (n+1)/2

In einer Liste mit sechs Elementen wird ein Element nach einem Vergleich gefunden, ein Element nach zwei Vergleichen und so weiter: S = (1+2+3+4+5+6) / 6 = 21/6 = 3,5.

Zeitkomplexität:

Die sequentielle Suche besitzt eine Zeitkomplexität von **O(n)**. Die Laufzeit wächst also linear mit der Anzahl der Elemente.

6.2.2 Implementierung einer sequentiellen Suche

Wir wollen nun eine sequentielle Suche in Java implementieren. Dazu verwenden wir einen int-Array `zahlen` aus 100 Zufallszahlen.

```
public int getIndex(int[] zahlen, int suchzahl)
{
   for (int i = 0; i < zahlen.length; i++)
      if (zahlen[i] == suchzahl)
         return i;
   return -1;
}
```

Diese Methode sucht in dem Array den **Index** der Suchzahl. Falls die Suchzahl nicht in dem Array vorhanden ist, wird **-1** zurückgegeben.