Algorithmen

Präsentiert von

- Tim Deimel
- Matthias Bohn
- Sebastian Schmalz

Inhaltsverzeichnis

Algorithmen

Algorithmen sind klare, schrittweise Anweisungen zur Lösung eines Problems oder zur Durchführung einer Aufgabe. Sie werden unabhängig von Programmiersprachen formuliert, häufig in Pseudocode oder einfacher Sprache, damit sie universell verständlich und auf unterschiedliche Systeme übertragbar sind.

Ein Algorithmus besteht typischerweise aus:

1. Eingaben: Die benötigten Daten.
2. Verarbeitung: Die schrittweise Bearbeitung dieser Daten.
3. Ausgaben: Das Ergebnis der Verarbeitung.

Wichtige Eigenschaften sind Eindeutigkeit, Endlichkeit, Allgemeinheit und Effizienz. Dadurch sind Algorithmen vielseitig anwendbar und dienen als Grundlage für die Umsetzung in verschiedenen Programmiersprachen.

Suchalgorithmen

Lineare Suche

Zu den einfachen Suchalgorithmen gehört die lineare oder auch sequenzielle Suche. Sie wird in der Regel bei ungeordneten Arrays verwendet und eignet sich vor allem bei eher kleineren Datenmengen.

```
int value;
int index = 0;
while (index < array.Length && array[index] != value)
{
    index++;
}
if (index > array.Length)
{
    return -1;
}
return index;
```

Binäre Suche

Die **binäre Suche** ist ein Algorithmus, der effizient eine bestimmte Zahl oder ein bestimmtes Element in einer **sortierten Liste** findet. Der Name kommt daher, dass die Liste bei jedem Schritt in zwei Hälften geteilt wird ("binär" = zwei).

- **Voraussetzung:** Die Liste muss **sortiert** sein, also in aufsteigender oder absteigender Reihenfolge.
- **Ziel:** Wir suchen ein bestimmtes Element (z. B. eine Zahl) in dieser Liste.

- **Schritte:**
  - Wir schauen uns das Element in der **Mitte der Liste** an.
  - Wenn es das gesuchte Element ist: **Fertig!**
  - Wenn das gesuchte Element kleiner ist, ignorieren wir die rechte Hälfte der Liste (da es dort nicht sein kann).
  - Wenn es größer ist, ignorieren wir die linke Hälfte.
  - Wir wiederholen das Ganze mit der übrig gebliebenen Hälfte.
- **Warum effizient?** Weil die Liste bei jedem Schritt **halbiert** wird. Dadurch wird die Anzahl der zu überprüfenden Elemente schnell sehr klein.

```
int binarySearch(int[] data, int x)
{
    int l = 0;
    int r = data.Length - 1;
    while(l<=r)
    {
        int m = Convert.ToInt32(Math.Floor(Convert.ToDecimal((l + r) / 2)));
        if (data[m] == x) return m;
        else if (x < data[m])
        {
            r = m - 1;
        }
        else l = m + 1;

    }
    return -1;
}
```

Sortieralgorithmen

Bubble Sort

Bubble Sort ist ein einfacher Algorithmus, mit dem man eine Liste von Zahlen (oder anderen Dingen) der Größe nach sortieren kann. Der Name kommt daher, dass größere Werte in der Liste wie Luftblasen langsam nach oben "aufsteigen", während kleinere Werte unten bleiben.

1. **Vergleichen:** Gehe von links nach rechts durch die Liste und vergleiche immer zwei benachbarte Zahlen.
2. **Tauschen:** Wenn die linke Zahl größer ist als die rechte, vertausche die beiden.
3. **Wiederholen:** Mach das für die ganze Liste. Am Ende des ersten Durchgangs steht die größte Zahl ganz hinten, wo sie hingehört.
4. **Nächste Runden:** Wiederhole das Ganze, aber ignoriere in jedem Durchgang das letzte Element, da es bereits an der richtigen Stelle ist.
5. **Fertig:** Mach so lange weiter, bis du keine Zahlen mehr tauschen musst. Die Liste ist dann sortiert.

```
int n = array.Length;

for (int i = 0; i < n - 1; i++)
{
    for (int j = 0; j < n - i - 1; j++)
    {
        if (array[j] > array[j + 1])
        {
            int temp = array[j];
            array[j] = array[j + 1];
            array[j + 1] = temp;
        }
    }
}
```

Bucket Sort

Bucket Sort (Eimer-Sortierung) ist ein Algorithmus, der eine Liste sortiert, indem er die Elemente in verschiedene **Buckets** (Eimer) aufteilt. Diese Eimer repräsentieren Bereiche von Werten, und innerhalb der Eimer werden die Elemente separat sortiert. Schließlich werden die Eimer in der richtigen Reihenfolge zusammengefügt.

1. **Eimer erstellen:** Teile den Wertebereich der Liste in mehrere Eimer (z. B. Intervalle wie 0-9, 10-19, etc.).
2. **Elemente verteilen:** Platziere jedes Element in den passenden Eimer, abhängig von seinem Wert.
3. **Sortieren:** Sortiere die Elemente innerhalb jedes Eimers (z. B. mit einem einfachen Algorithmus wie Insertion Sort).
4. **Zusammenfügen:** Kombiniere die Eimer in der richtigen Reihenfolge, um die sortierte Liste zu erhalten.

```
int n = array.Length;
List[] buckets = new List[n];

for (int i = 0; i < n; i++)
    buckets[i] = new List();

for (int i = 0; i < n; i++)
{
    int bucketIndex = (int)(array[i] * n);
    buckets[bucketIndex].Add(array[i]);
}

for (int i = 0; i < n; i++)
    buckets[i].Sort();

int index = 0;
for (int i = 0; i < n; i++)
{
    foreach (var value in buckets[i])
    {
        array[index++] = value;
    }
}
```

Rekursion

Rekusion erklärt mit Fukultät

Rekursion ist eine Programmiertechnik, bei der eine Funktion sich selbst aufruft, um ein Problem schrittweise zu lösen. Sie wird häufig verwendet, wenn sich ein Problem in kleinere Teilprobleme gleicher Art zerlegen lässt.

Ein klassisches Beispiel ist die Berechnung der **Fakultät** einer Zahl n. Die Fakultät (n!) wird definiert als das Produkt aller positiven ganzen Zahlen von 1 bis n, mit der Bedingung, dass 0! = 1.

Die Fakultät lässt sich rekursiv wie folgt definieren:

- $n! = n \times (n-1)!$ wenn $n > 0$
- $0! = 1$

```
static int Fakultaet(int n)
{
    if (n == 0)
        return 1;
    return n * Fakultaet(n - 1);
}
```

Fibonacci

Der Fibonacci-Algorithmus wird verwendet, um Zahlen in der Fibonacci-Folge zu generieren, einer Sequenz, bei der jede Zahl die Summe der beiden vorherigen Zahlen ist. Die ersten beiden Zahlen sind definiert als 0 und 1. Die Sequenz sieht so aus: 0, 1, 1, 2, 3, 5, 8, 13, ...

Wie funktioniert der Fibonacci-Algorithmus?

1. Iterativ:
   - Beginne mit den ersten beiden Zahlen (0 und 1).
   - Berechne die nächste Zahl, indem du die beiden vorherigen addierst.
   - Wiederhole den Vorgang, bis die gewünschte Position erreicht ist.

2. Rekursiv:
   - Definiere eine Funktion, die die Summe der Ergebnisse für die beiden vorherigen Zahlen zurückgibt.
   - Die Rekursion endet bei den Basisfällen: Fibonacci(0) = 0 und Fibonacci(1) = 1.

Anwendung: Der Fibonacci-Algorithmus wird in der Mathematik, Computerwissenschaft und Naturwissenschaften verwendet, z. B. bei der Modellierung von Wachstumsprozessen.

```
static int FibonacciIterativ(int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;

    int prev1 = 0, prev2 = 1, current = 0;

    for (int i = 2; i <= n; i++)
    {
        current = prev1 + prev2;
        prev1 = prev2;
        prev2 = current;
    }

    return current;
}
```