Komplexität und O-Notation

Reiner Hüchting

10. April 2023

Themenüberblick

O-Notation

Beispiele: Optimierung von Algorithmen

Themenüberblick

O-Notation

Beispiele: Optimierung von Algorithmen

O-Notation

Bisher: Informelle Komplexitätsabschätzungen

- Laufzeitabschätzungen in Abhängigkeit der Größe einer Datenstruktur
  - z.B. Länge einer Liste oder Anzahl der Elemente eines Baumes

O-Notation

Bisher: Informelle Komplexitätsabschätzungen

- Laufzeitabschätzungen in Abhängigkeit der Größe einer Datenstruktur
  - z.B. Länge einer Liste oder Anzahl der Elemente eines Baumes
- Beobachtung: Laufzeit wird i.d.R. *ungenau* angegeben.
  - z.B. Schleifendurchläufe zählen, aber nicht die Anzahl der Operationen innerhalb der Schleife
  - z.B. geschachtelte Schleifen berücksichtigen, hintereinander ausgeführte Schleifen aber nicht

O-Notation

Bisher: Informelle Komplexitätsabschätzungen

- Laufzeitabschätzungen in Abhängigkeit der Größe einer Datenstruktur
  - z.B. Länge einer Liste oder Anzahl der Elemente eines Baumes
- Beobachtung: Laufzeit wird i.d.R. *ungenau* angegeben.
  - z.B. Schleifendurchläufe zählen, aber nicht die Anzahl der Operationen innerhalb der Schleife
  - z.B. geschachtelte Schleifen berücksichtigen, hintereinander ausgeführte Schleifen aber nicht

Ziel: Formalisierung dieser Ungenauigkeiten

- Wie kommen diese Abschätzungen zustande?
- Welche Operationen müssen gezählt werden?

O-Notation

Beispiel: Maximum einer Liste bestimmen

```java
public static int searchMax(List<Integer> list) {
    int max = Integer.MIN_VALUE;
    for (int i = 0; i < list.size(); i++) {
        max = Math.max(max, list.get(i));
    }
    return max;
}
```

O-Notation

Beispiel: Maximum einer Liste bestimmen

```java
public static int searchMax(List<Integer> list) {
    int max = Integer.MIN_VALUE;
    for (int i = 0; i < list.size(); i++) {
        max = Math.max(max, list.get(i));
    }
    return max;
}
```

Komplexität

- n Schleifendurchläufe (Aufrufe von `Math.max`)
- Komplexitätsklasse: $O(n)$

O-Notation

Beispiel: Differenz zw. Minimum und Maximum bestimmen

```java
public static int diffMinMax(List<Integer> list) {
    int min = Integer.MAX_VALUE;
    int max = Integer.MIN_VALUE;
    for (int i = 0; i < list.size(); i++) {
        min = Math.min(min, list.get(i));
    }
    for (int i = 0; i < list.size(); i++) {
        max = Math.max(max, list.get(i));
    }
    return max - min;
}
```

O-Notation

Beispiel: Differenz zw. Minimum und Maximum bestimmen

```java
public static int diffMinMax(List<Integer> list) {
    int min = Integer.MAX_VALUE;
    int max = Integer.MIN_VALUE;
    for (int i = 0; i < list.size(); i++) {
        min = Math.min(min, list.get(i));
    }
    for (int i = 0; i < list.size(); i++) {
        max = Math.max(max, list.get(i));
    }
    return max - min;
}
```

Komplexität

- $2n$ Aufrufe von `Math.max` oder `Math.min`
- Komplexitätsklasse: $O(n)$
  - Warum nicht $O(2n)$?

O-Notation

Beispiel: Minimale Differenz von Elementen bestimmen

```java
public static int closestPair(List<Integer> list) {
    int minDiff = Integer.MAX_VALUE;
    for (int i = 0; i < list.size(); i++) {
        for (int j = i + 1; j < list.size(); j++) {
            minDiff = Math.min(minDiff, Math.abs(list.
                get(i) - list.get(j)));
        }
    }
    return minDiff;
}
```

O-Notation

Beispiel: Minimale Differenz von Elementen bestimmen

```java
public static int closestPair(List<Integer> list) {
    int minDiff = Integer.MAX_VALUE;
    for (int i = 0; i < list.size(); i++) {
        for (int j = i + 1; j < list.size(); j++) {
            minDiff = Math.min(minDiff, Math.abs(list.
                get(i) - list.get(j)));
        }
    }
    return minDiff;
}
```

Komplexität

- n Durchläufe der äußeren Schleife
- pro Durchlauf: $\leq n$ Durchläufe der inneren Schleife
  - Warum $\leq n$ und nicht genauer?
- Komplexitätsklasse: $O(n^2)$

O-Notation

Beobachtungen

- Komplexitätsklassen geben nur die Größenordnung an.
- Konstante Faktoren und nicht-dominante Terme werden vernachlässigt.

O-Notation

Beobachtungen

- Komplexitätsklassen geben nur die Größenordnung an.
- Konstante Faktoren und nicht-dominante Terme werden vernachlässigt.

Beispiele

$$
\begin{aligned}
O(n) &= O(2n) = O(\frac{n}{2}) \\
O(n^2) &= O(n^2 + n + 1) = O(\left(\frac{n}{2}\right)^2) \\
O(n \log n) &= O(2n \log n + 50n)
\end{aligned}
$$

O-Notation

Intuition:

- Der Unterschied zwischen $O(n)$und $O(2n)$ kann durch schnellere Hardware ausgeglichen werden.
- Ebenso der Unterschied zwischen $O(n^2)$und $O(2(n^2))$.
- Der Unterschied zwischen $O(n)$und $O(n^2)$kann nicht so einfach kompensiert werden.
- Das Verhalten von Polynomen (Funktionen) wird i.W. vom *Leitterm* bestimmt.

O-Notation

Intuition:

- Der Unterschied zwischen $O(n)$und $O(2n)$ kann durch schnellere Hardware ausgeglichen werden.
- Ebenso der Unterschied zwischen $O(n^2)$und $O(2(n^2))$.
- Der Unterschied zwischen $O(n)$und $O(n^2)$kann nicht so einfach kompensiert werden.
- Das Verhalten von Polynomen (Funktionen) wird i.W. vom *Leitterm* bestimmt.

Ziel bei der Entwicklung:

- Komplexitätsklasse möglichst klein halten.
- Komplexität kann nicht durch Hardware ausgeglichen werden!
- Konstante oder lineare Faktoren sind weniger von Bedeutung.

O-Notation

Definition: O-Komplexität

Gegeben eine Funktion $f(n)$, ist $f(n) = O(g(n))$ genau dann, wenn es eine positive Konstante c gibt, so dass für alle $n \geq n_0$ gilt:

$$f(n) \leq c \cdot g(n)$$

O-Notation

Definition: O-Komplexität

Gegeben eine Funktion $f(n)$, ist $f(n) = O(g(n))$ genau dann, wenn es eine positive Konstante c gibt, so dass für alle $n \geq n_0$ gilt:

$$f(n) \leq c \cdot g(n)$$

Intuitiv:

- Falls $f(n) \geq g(n)$ für alle n gilt, dann unterscheiden sich die Funktionen nur durch einen konstanten Faktor.
- Für große n ist $g(n)$ eine gute Abschätzung für $f(n)$.

O-Notation

Definition: O-Komplexität

$$f \in O(g) \Leftrightarrow \exists_{c>0} \exists_{n_0} \forall_{n \geq n_0} : f(n) \leq c \cdot g(n)$$

Intuitiv:

- Die Funktion $f(n)$ wächst nicht schneller als $g(n)$.
- Für fast alle n gilt $f(n) \leq c \cdot g(n)$.

O-Notation

Definition: O-Komplexität

$$f \in O(g) \Leftrightarrow \exists_{c>0} \exists_{n_0} \forall_{n \geq n_0} : f(n) \leq c \cdot g(n)$$

Intuitiv:

- Die Funktion $f(n)$ wächst nicht schneller als $g(n)$.
- Für fast alle n gilt $f(n) \leq c \cdot g(n)$.

Konstante Faktoren sind nicht relevant:

- Bewegen sich im Bereich der Ungenauigkeit, die durch unterschiedliche Hardware entsteht.
- Bieten geringes Optimierungspotenzial.
- Können ggf. durch Hardware ausgeglichen werden.

Themenüberblick

O-Notation

Beispiele: Optimierung von Algorithmen

Themenüberblick

Beispiele: Optimierung von Algorithmen – Vergleich der Elemente zweier Listen

Ziel: Prüfung, ob zwei Listen die gleichen Elemente haben

- Gegeben: Zwei Listen von Zahlen `A` und `B`
- Ergebnis: `true`, falls jedes Element aus Liste `A` auch in Liste `B` vorkommt und umgekehrt.

Beispiele: Optimierung von Algorithmen – Vergleich der Elemente zweier Listen

Ziel: Prüfung, ob zwei Listen die gleichen Elemente haben

- Gegeben: Zwei Listen von Zahlen `A` und `B`
- Ergebnis: `true`, falls jedes Element aus Liste `A` auch in Liste `B` vorkommt und umgekehrt.

Naiver Ansatz

- Durchlaufe Liste `A`.
  - Suche jedes Element in Liste `B`.
- Wiederhole für Liste `B`.

Beispiele: Optimierung von Algorithmen – Vergleich der Elemente zweier Listen

Ziel: Prüfung, ob zwei Listen die gleichen Elemente haben

- Gegeben: Zwei Listen von Zahlen `A` und `B`
- Ergebnis: `true`, falls jedes Element aus Liste `A` auch in Liste `B` vorkommt und umgekehrt.

Naiver Ansatz

- Durchlaufe Liste `A`.
  - Suche jedes Element in Liste `B`.
- Wiederhole für Liste `B`.

Komplexität

- n Durchläufe der äußeren Schleife
- pro Durchlauf: $\leq n$ Durchläufe der inneren Schleife
- Komplexitätsklasse: $O(n^2)$

Beispiele: Optimierung von Algorithmen – Vergleich der Elemente zweier Listen

Ziel: Prüfung, ob zwei Listen die gleichen Elemente haben

- Gegeben: Zwei Listen von Zahlen `A` und `B`
- Ergebnis: `true`, falls jedes Element aus Liste `A` auch in Liste `B` vorkommt und umgekehrt.

Beispiele: Optimierung von Algorithmen – Vergleich der Elemente zweier Listen

Ziel: Prüfung, ob zwei Listen die gleichen Elemente haben

- Gegeben: Zwei Listen von Zahlen `A` und `B`
- Ergebnis: `true`, falls jedes Element aus Liste `A` auch in Liste `B` vorkommt und umgekehrt.

Optimierte Lösung

- Sortiere beide Listen.
- Vergleiche die Listen in einem einzigen Durchlauf.

Beispiele: Optimierung von Algorithmen – Vergleich der Elemente zweier Listen

Ziel: Prüfung, ob zwei Listen die gleichen Elemente haben

- Gegeben: Zwei Listen von Zahlen `A` und `B`
- Ergebnis: `true`, falls jedes Element aus Liste `A` auch in Liste `B` vorkommt und umgekehrt.

Optimierte Lösung

- Sortiere beide Listen.
- Vergleiche die Listen in einem einzigen Durchlauf.

Komplexität

- Sortieren: $O(n \log n)$
- Vergleichen: $O(n)$
- Gesamt: $O(n \log n) + O(n) = O(n \log n)$

Themenüberblick

Beispiele: Optimierung von Algorithmen – Berechnung von Produkten innerhalb von Listen

Ziel: Suche nach dem größten Produkt benachbarter Elemente einer Liste

- Gegeben: Eine Liste von Zahlen der Länge n.
- Ergebnis: Das größte Produkt von m benachbarten Elementen.

Beispiele: Optimierung von Algorithmen – Berechnung von Produkten innerhalb von Listen

Ziel: Suche nach dem größten Produkt benachbarter Elemente einer Liste

- Gegeben: Eine Liste von Zahlen der Länge n.
- Ergebnis: Das größte Produkt von m benachbarten Elementen.

Naiver Ansatz

- Durchlaufe die Liste von Stelle 0 bis $n - m$.
  - Von jeder Position aus berechne das Produkt der nächsten m Elemente.

Beispiele: Optimierung von Algorithmen – Berechnung von Produkten innerhalb von Listen

Ziel: Suche nach dem größten Produkt benachbarter Elemente einer Liste

- Gegeben: Eine Liste von Zahlen der Länge n.
- Ergebnis: Das größte Produkt von m benachbarten Elementen.

Naiver Ansatz

- Durchlaufe die Liste von Stelle 0 bis $n - m$.
  - Von jeder Position aus berechne das Produkt der nächsten m Elemente.

Komplexität

- $n - m \leq n$ Durchläufe der äußeren Schleife
- pro Durchlauf: m Durchläufe der inneren Schleife
- Komplexitätsklasse: $O(n \cdot m)$ (für große m grob $O(n^2)$)

Beispiele: Optimierung von Algorithmen – Berechnung von Produkten innerhalb von Listen

Ziel: Suche nach dem größten Produkt benachbarter Elemente einer Liste

- Gegeben: Eine Liste von Zahlen der Länge n.
- Ergebnis: Das größte Produkt von m benachbarten Elementen.

Beispiele: Optimierung von Algorithmen – Berechnung von Produkten innerhalb von Listen

Ziel: Suche nach dem größten Produkt benachbarter Elemente einer Liste

- Gegeben: Eine Liste von Zahlen der Länge n.
- Ergebnis: Das größte Produkt von m benachbarten Elementen.

Optimierter Ansatz

- Berechne das Produkt der ersten m Elemente.
- Durchlaufe die Liste von Stelle m bis n.
  - Multipliziere das Produkt mit dem Element an Stelle i.
  - Dividiere das Produkt durch das Element an Stelle $i - m$.

Beispiele: Optimierung von Algorithmen – Berechnung von Produkten innerhalb von Listen

Ziel: Suche nach dem größten Produkt benachbarter Elemente einer Liste

- Gegeben: Eine Liste von Zahlen der Länge n.
- Ergebnis: Das größte Produkt von m benachbarten Elementen.

Optimierter Ansatz

- Berechne das Produkt der ersten m Elemente.
- Durchlaufe die Liste von Stelle m bis n.
  - Multipliziere das Produkt mit dem Element an Stelle i.
  - Dividiere das Produkt durch das Element an Stelle $i - m$.

Komplexität

- Berechnung des Anfangsprodukts: $O(m)$
- Schleife $O(n - m)$
- Gesamt: $O(n)$

Themenüberblick

O-Notation

Beispiele: Optimierung von Algorithmen

- Vergleich der Elemente zweier Listen
- Berechnung von Produkten innerhalb von Listen
- Suche nach Strings in einem Text

Beispiele: Optimierung von Algorithmen – Suche nach Strings in einem Text

Ziel: Suche nach einem String in einem Text

- Gegeben: Ein Text (String) der Länge n und ein zu suchender String der Länge m.
- Ergebnis: Alle Positionen, an denen der Suchstring vorkommt.

Beispiele: Optimierung von Algorithmen – Suche nach Strings in einem Text

Ziel: Suche nach einem String in einem Text

- Gegeben: Ein Text (String) der Länge n und ein zu suchender String der Länge m.
- Ergebnis: Alle Positionen, an denen der Suchstring vorkommt.

Naiver Ansatz

- Durchlaufe den gesamten Text.
  - An jeder Position vergleiche den dortigen Teilstring mit dem gesuchten String.

Beispiele: Optimierung von Algorithmen – Suche nach Strings in einem Text

Ziel: Suche nach einem String in einem Text

- Gegeben: Ein Text (String) der Länge n und ein zu suchender String der Länge m.
- Ergebnis: Alle Positionen, an denen der Suchstring vorkommt.

Naiver Ansatz

- Durchlaufe den gesamten Text.
  - An jeder Position vergleiche den dortigen Teilstring mit dem gesuchten String.

Komplexität

- n Durchläufe der äußeren Schleife
- pro Durchlauf: m Schritte für den Vergleich.
- Komplexitätsklasse: $O(n \cdot m)$.

Beispiele: Optimierung von Algorithmen – Suche nach Strings in einem Text

Ziel: Suche nach einem String in einem Text

- Gegeben: Ein Text (String) der Länge n und ein zu suchender String der Länge m.
- Ergebnis: Alle Positionen, an denen der Suchstring vorkommt.

Beispiele: Optimierung von Algorithmen – Suche nach Strings in einem Text

Ziel: Suche nach einem String in einem Text

- Gegeben: Ein Text (String) der Länge n und ein zu suchender String der Länge m.
- Ergebnis: Alle Positionen, an denen der Suchstring vorkommt.

Optimierung

- Durchlaufe den gesamten Text.
  - An jeder Position vergleiche den dortigen Teilstring mit dem gesuchten String.
  - Bei Nicht-Übereinstimmung berechne, wie weit gesprungen werden kann.

Beispiele: Optimierung von Algorithmen – Suche nach Strings in einem Text

Ziel: Suche nach einem String in einem Text

- Gegeben: Ein Text (String) der Länge n und ein zu suchender String der Länge m.
- Ergebnis: Alle Positionen, an denen der Suchstring vorkommt.

Beispiele: Optimierung von Algorithmen – Suche nach Strings in einem Text

Ziel: Suche nach einem String in einem Text

- Gegeben: Ein Text (String) der Länge n und ein zu suchender String der Länge m.
- Ergebnis: Alle Positionen, an denen der Suchstring vorkommt.

Optimierung bei häufiger Suche

- Baue einen Suchindex auf:
- Z.B. ein Präfixbaum, der für mögliche Suchbegriffe die Positionen im Text enthält.
- Kann aus einer Datenbank häufiger Anfragen erstellt werden.
- Kann nebenbei erstellt und aktualisiert werden.