Algorithmen und Datenstrukturen

Jonas Milkovits

Last Edited: 26. April 2020

Inhaltsverzeichnis

1 Einleitung

Problem im Sinne der Informatik

- Enthält eine Beschreibung der Eingabe
- Enthält eine Beschreibung der Ausgabe
- Gibt **keinen** Übergang von Eingabe und Ausgabe an
- z.B.: Finde den kürzesten Weg zwischen zwei Orten

Probleminstanzen

- Probleminstanz ist eine konkrete Eingabenbelegung, für die entsprechende Ausgabe gewünscht ist
- z.B.: Was ist der kürzeste Weg vom Audimax in die Mensa?

Begriff des Algorithmus

- Endliche Folge von Rechenschritten, der eine Ausgabe in eine Eingabe verwandelt

Anforderungen an Algorithmen

- Spezifizierung der Eingabe und Ausgabe
  - Anzahl und Typen aller Elemente ist definiert
- Eindeutigkeit
  - Jeder Einzelschritt ist klar definiert und ausführbar
  - Die Reihenfolge der Einzelschritte ist festgelegt
- Eindlichkeit
  - Notation hat eine endliche Länge

Eigenschaften von Algorithmen

- Determiniertheit
  - Für gleiche Eingabe stets die gleiche Ausgabe (andere mögliche Zwischenzustände)
- Determinismus
  - Für gleiche Eingabe stets identische Ausführung und Ausgabe
- Terminierung
  - Algorithmus läuft für jede Eingabe nur endlich lange
- Korrektheit
  - Algorithmus berechnet stets die spezifizierte Ausgabe (falls dieser terminiert)
- Effizienz
  - Sparsamkeit im Ressourcenverbrauch (Zeit, Speicher, Energie,...)

Begriff der Datenstrukturen

- Methode, um Datenabzuspeichern und zu organisieren
- Erleichtert Zugriff auf Daten und Modifikation der Daten
- Beinhalten Strukturbestandteile und Nutzerdaten (Payload)
- Sequenzen: Arrays, Listen,...
- Topologische Strukturen: Bäume, Graphen,...

2 Sortieren

2.1 Allgemeine Informationen

Das Sortierproblem

- Ausgangspunkt: Folge von Datensätzen $D_1, D_2, ..., D_n$
- Zu sortierende Elemente heißen auch Schlüssel(werte)
- Ziel: Datensätze so anzuordnen, dass die Schlüsselwerte sukzessive ansteigen/absteigen
- Bedingung: Schlüsselwerte müssen vergleichbar sein
- Durchführung:
  - Eingabe: Sequenz von Schlüsselwerten $< a_1, a_2, ..., a_n >$
  - Engabe ist eine **Instanz** des Sortierproblems
  - Ausgabe: Permutation $< a'_1, a'_2, ..., a'_n >$ derselben Folge mit Eigenschaft $a'_1 \leq ... \leq a'_n$
- Algorithmus **korrekt**, wenn dieser das Problem für alle Instanzen löst

Exkurs: Totale Ordnung

- Sei M eine nicht leere Menge und $\leq \subseteq MxM$ eine binäre Relation auf M
- Das Paar $(M, \leq)$ heißt genau dann totale Relation auf der Menge M, wenn Folgendes erfüllt ist:
  - Reflexivität: $\forall x \in M : x \leq x$
  - Transitivität: $\forall x, y, z \in M : x \leq y \wedge y \leq z \Rightarrow x \leq z$
  - Antisymmetrie: $\forall x, y \in M : x \leq y \wedge y \leq x \Rightarrow x = y$
  - Totalität: $\forall x, y \in M : x \leq y \vee y \leq x$
- z.B.: $\leq$ Ordnung auf natürlichen Zahlen bildet eine totale Ordnung ($1 \leq 2 \leq 3$...)
- z.B.: Lexikographische Ordnung $\leq_{lex}$ ist eine totale Ordnung ($A \leq B \leq C$...)

Vergleichskriterien von Sortieralgorithmen

- Berechnungsaufwand $O(n)$
- Effizient: Best Case vs Average Case vs Worst Case
- Speicherbedarf:
  - in-place (in situ): Zusätzlicher Speicher von der Eingabegröße unabhängig
  - out-of-place: Speichermehrbedarf von Eingabegröße abhängig
- Stabilität: Stabile Verfahren verändern die Reihenfolge von äquivalenten Elementen nicht
- Anwendung als Auswahlfaktor:
  - Hauptoperationen beim Sortieren: Vergleiche und Vertausche
  - Diese Operationen können sehr teuer oder sehr günstig sein, je nach Aufwand
  - Anpassung des Verfahrens abhängig von dem Aufwand dieser Operationen

2.2 Insertion Sort

Idee

- Halte die linke Teilfolge sortiert
- Füge nächsten Schlüsselwert hinzu, indem es an die korrekte Position eingefügt wird
- Wiederhole den Vorgang bis Teilfolge aus der gesamten Liste besteht

Code

```
1 FOR j = 1 TO A.length - 1
2   key = A[j]
3   // Füge A[j] in die sortierte Sequenz A[0...j-1] ein
4   i = j - 1
5   WHILE i ≥ 0 and A[i] > key
6       A[i + 1] = A[i]
7       i = i - 1
8   A[i + 1] = key
```

3 Pseudocode in der Vorlesung AuD

Datentypen

- String
  - Aufbau:

    `"Die Summe ist"`
  - Konkatenation:

    `"Die Summe ist" summe`
- Array
  - `A`: Bezeichung eines Arrays `A`
  - `A[i]` Zugriff auf `(i+1)`-tes Element des Arrays

Methoden

- Rückgabe:

  ```
  return summe
  ```

Schleifen

- While-Schleife

  ```
  WHILE summe <= n
      summe = summe + 1
  ENDWHILE
  ```

Variablen

- Initialisierung

  ```
  summe := 0
  ```