Datenstrukturen, Algorithmen und Programmierung 2

Amin Coja-Oghlan

June 10, 2022

Lehrstuhl Informatik 2
Fakultät für Informatik

Sortieren in linearer Zeit

Nicht-vergleichbasierte Sortieralgorithmen

- vergleichbasierte Sortieralgorithmen benötigen $\Omega(n \log n)$ Vergleiche
- der Vorteil vergleichbasierter Algorithmen ist ihre allgemeine Anwendbarkeit
- in dieser Vorlesung lernen wir *schnellere* Sortieralgorithmen für spezielle Sortierprobleme kennen

Sortieren in linearer Zeit

Sortieren via Zählen

- gegeben ist ein Array $A = (A_1, \ldots, A_n)$
- jedes Arrayelement ist mit einem Schlüssel aus einer Menge

$$S = \{s_1, \ldots, s_k\}$$

  versehen
- die Schlüssel sind vergleichbar
- die Arrayelemente sollen anhand der Schlüssel sortiert werden

Sortieren in linearer Zeit

`CountingSort(`$\boldsymbol{A}$`)`

1. Lege ein Hilfsarray $\boldsymbol{C} = (C_1, \ldots, C_k)$ an, so daß C_i die Zahl der Vorkommnisse von s_i in $\boldsymbol{A}$ enthählt.
2. Verwende das Hilfsarray, um die Gesamtzahlen C'_i der Elemente mit Schlüsseln $s_1, \ldots, s_i$ zu bestimmen.
3. Reserviere Speicherplatz $\boldsymbol{B} = (B_1, \ldots, B_n)$ für das Ausgabearray
4. Für $j = n, \ldots, 1$
5. ermittle den Schlüssel σ von A_j
6. setze $B_{C'_\sigma} = A_j$
7. verringere C'_σ um 1

Sortieren in linearer Zeit

Analyse von `CountingSort`

- `CountingSort` sortiert die Eingabe in Zeit $O(n) + O(k)$
- wenn $k = o(n \log n)$, ist `CountingSort` also schneller als vergleichsbasierte Sortieralgorithmen
- `CountingSort` ist ein stabiler Sortieralgorithmus
- D.h. Elemente mit demselben Schlüssel werden in ihrer ursprünglichen Reihenfolge ausgegeben
- die Stabilität wird dadurch sichergestellt, daß Schritt 4 die Elemente von hinten nach vorn durchgeht

Sortieren in linearer Zeit

Radixsort

- wir nehmen an, daß die Eingabeelemente mit einer Folge von d Schlüsseln aus $\mathcal{S}$ versehen sind
- Beispiel: Zahlen- oder Buchstabenfolgen fester Länge
- direktes Anwenden von `CountingSort` benötigt Zeit $O(n) + O(k^d)$

Sortieren in linearer Zeit

`RadixSort`($\boldsymbol{A}$)

1. für $i = d, \ldots, 1$
2. sortiere $\boldsymbol{A}$ nach der i-ten Komponente des Schlüssels mit `CountingSort`

Sortieren in linearer Zeit

Analyse von Radixsort

- weil `CountingSort` stabil ist, funktioniert `RadixSort` korrekt
- die Laufzeit ist $O(d \cdot n) + O(d \cdot k)$.
- schon für moderate d, k ist dies eine deutliche Verbesserung

Sortieren in linearer Zeit

Zusammenfassung

- `CountingSort` und `RadixSort` sind einfache aber effektive Algorithmen
- statt derselben Schlüsselmenge $\mathcal{S}$ können bei `RadixSort` auch verschiedene Schlüsselmengen verwendet werden
- Anwendungsbeispiele: Matrikelnummern, Daten (Tag–Monat–Jahr)