Datenstrukturen, Algorithmen und Programmierung 2

Amin Coja-Oghlan

June 10, 2022

Lehrstuhl Informatik 2
Fakultät für Informatik

Quicksort

Ziel

Eine Liste $L = (\ell_1, \ldots, \ell_n)$ vergleichbarer Elemente aufsteigend sortieren.

Vergleichbarkeit

- für je zwei Element ℓ_i, ℓ_j gilt entweder $\ell_i < \ell_j$, $\ell_i = \ell_j$ oder $\ell_i > \ell_j$
- die Ordnung ist transitiv: $\ell_h \leq \ell_i$ und $\ell_i \leq \ell_j \Rightarrow \ell_h \leq \ell_j$
- die Ordnung ist antisymmetrisch: $\ell_i \leq \ell_j$ und $\ell_j \leq \ell_i \Rightarrow \ell_i = \ell_j$
- wir haben Zugriff auf eine Funktion, die zwei Element ℓ_i, ℓ_j vergleicht

Quicksort

Beispiele

- ganze, rationale oder reelle Zahlen mit der gewöhnlichen Ordnung
- Zeichenketten mit alphabetischer Ordnung
- Punkte in $\mathbb{R}^2$ mit lexikographischer Ordnung

Effiziente, generische Sortieralgorithmen

- Laufzeit = Zahl der Vergleiche, die der Algorithmus durchführt
- der Algorithmus darf Elemente *nur* miteinander vergleichen
- dadurch ist der Algorithmus auf beliebige vergleichbare Objekte anwendbar

Quicksort

Algorithmus `Quicksort`

1. Für $i = 1, \ldots, n$
2. falls $\ell_i < \ell_1$, füge ℓ_i der Liste K hinzu.
3. falls $\ell_i > \ell_1$, füge ℓ_i der Liste G hinzu.
4. falls $\ell_i = \ell_1$, füge ℓ_i der Liste M hinzu.
5. Wende `Quicksort` rekursiv an, um K und G zu sortieren.
6. Gib K, M, G aus.

Quicksort

Implementation: Sortieren an Ort und Stelle

- wir implementieren `Quicksort`, indem die Elemente in der Eingabeliste miteinander vertauscht werden
- dazu verwenden wir Dijkstra's Partitionierungsverfahren
- dieses vertausche die Elemente des Arrays wie folgt
- gegeben ist ein Array A der Länge n; Pivot ist Element $A[0]$
- Einträge 0 bis ausschließlich i sind kleiner als das Pivot
- Einträge i bis ausschließlich j sind gleich dem Pivot
- Einträge $k+1$ bis $n-1$ sind größer als das Pivot
- *Beachte: die Indizierung der Einträge beginnt bei 0 und endet bei* $n-1$

Quicksort

Dijkstra-Partitionierung

1. setze $i = 0$, $j = 0$, $k = n - 1$
2. solange $j \leq k$
3. falls $A[j] <$ pivot
4. vertausche $A[i]$ und $A[j]$
5. erhöhe i und j um 1
6. sonst, wenn $A[j] >$ pivot
7. vertausche $A[j]$ und $A[k]$
8. verringere k um 1
9. sonst erhöhe j um 1

Quicksort

Beispiel: Dijkstra-Partitionierung

Wir sortieren $(2, 4, 6, 1, 3, 5, 2)$ mit Pivot 2:

$(2, 4, 6, 1, 3, 5, 2)$	$i = 0,\ j = 0,\ k = 6$
$(2, 4, 6, 1, 3, 5, 2)$	$i = 0,\ j = 1,\ k = 6$
$(2, 2, 6, 1, 3, 5, 4)$	$i = 0,\ j = 1,\ k = 5$
$(2, 2, 6, 1, 3, 5, 4)$	$i = 0,\ j = 2,\ k = 5$
$(2, 2, 5, 1, 3, 6, 4)$	$i = 0,\ j = 2,\ k = 4$
$(2, 2, 3, 1, 5, 6, 4)$	$i = 0,\ j = 2,\ k = 3$
$(2, 2, 1, 3, 5, 6, 4)$	$i = 0,\ j = 2,\ k = 2$
$(1, 2, 2, 3, 5, 6, 4)$	$i = 1,\ j = 2,\ k = 2$
$(1, 2, 2, 3, 5, 6, 4)$	$i = 1,\ j = 3,\ k = 2$

Quicksort

Worst case-Laufzeit

- die schlechteste Laufzeit ergibt sich, wenn die Eingabe bereits sortiert ist!
- in diesem Fall wird jedesmal das Pivot mit allen Elementen verglichen
- die Laufzeit ist daher

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

- trotzdem begegnet Quicksort in der Praxis häufig
- wir werden im weiteren entdecken, warum

Quicksort

Zusammenfassung

- Quicksort sortiert eine Eingabeliste vergleichbarer Elemente
- die Sortierung erfolgt an Ort und Stelle
- die Laufzeit ist im schlimmsten Fall $n(n+1)/2$