Tutorium 5: Sortieren

Matthias Schimek | 3. Juni 2017

TUTORIUM ZUR VORLESUNG ALGORITHMEN I IM SS17

Gliederung

1. Letztes Blatt
2. Sortieren
3. Aufgaben

- *Dummy-Header*
- 1. Aufgabe Aufgabenstellung
- wesentliche Funktionen programmieren

Grundlagen

Laufzeit

- best case
- average case
- worst case

Speicherverbrauch

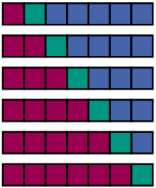
- inplace

Reihenfolge

- stabil: bei gleichen Element bleibt Reihenfolge erhalten

Insertionsort

Procedure insertionSort(a : **Array** $[1..n]$ **of** Element)
    for $i := 2$ **to** n **do**
        invariant $a[1] \leq \cdots \leq a[i-1]$
        move $a[i]$ to the right place


[1]

- langsam: $\mathcal{O}(n^2)$
- inplace
- stabil

[1] Folien 'Algorithmen I', KIT

Selectionsort

```
Function selectionSort(A : array [1...n] of ℕ) : array [1...n] of ℕ
    for i := 1 to n do
        min = i
        for j := i + 1 to n do
            if A[min] > A[j] then
                min := j
            end
        end
        tmp := A[min]
        A[min] := A[i]
        A[i] := tmp
    end
    return A
```

- langsam $\mathcal{O}(n^2)$
- inplace
- stabil

Selectionsort


```
Function selectionSort(A : array [1...n] of ℕ) : array [1...n] of ℕ
    for i := 1 to n do
        min = i
        for j := i + 1 to n do
            if A[min] > A[j] then
                min := j
            end
        end
        tmp := A[min]
        A[min] := A[i]
        A[i] := tmp
    end
    return A
```

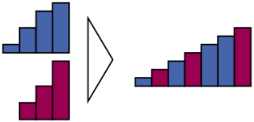
- langsam $\mathcal{O}(n^2)$
- inplace
- stabil

Mergesort

Idee: Teile und Herrsche

Function mergeSort($\langle e_1, \ldots, e_n \rangle$) : Sequence **of** Element
  if $n = 1$ **then return** $\langle e_1 \rangle$ // base case
  else return merge(mergeSort($\langle e_1, \ldots, e_{\lfloor n/2 \rfloor} \rangle$),
   mergeSort($\langle e_{\lfloor n/2 \rfloor + 1}, \ldots, e_n \rangle$))

Gegeben:
zwei sortierte Folgen a und b
Berechne:
sortierte Folge der Elemente aus a und b



[2]

- schnell: $\mathcal{O}(n \log n)$
- auch im Worstcase

---

[2] Folien 'Algorithmen I', KIT

Quicksort

Function quickSort(s : Sequence **of** Element) : Sequence **of** Element
  if $|s| \leq 1$ **then return** s
  pick "some" $p \in s$
  $a := \langle e \in s : e < p \rangle$
  $b := \langle e \in s : e = p \rangle$
  $c := \langle e \in s : e > p \rangle$
  return concatenation of quickSort(a), b, and quickSort(c) [3]

- average case: $\mathcal{O}(n \log n)$
- worst case: $\mathcal{O}(n^2)$
- man kann optimieren

[3] Folien 'Algorithmen I', KIT

Quicksort

Function quickSort(s : Sequence **of** Element) : Sequence **of** Element
  if $|s| \leq 1$ **then return** s
  pick "some" $p \in s$
  $a := \langle e \in s : e < p \rangle$
  $b := \langle e \in s : e = p \rangle$
  $c := \langle e \in s : e > p \rangle$
  return concatenation of quickSort(a), b, and quickSort(c) [3]

- average case: $\mathcal{O}(n \log n)$
- worst case: $\mathcal{O}(n^2)$
- man kann optimieren

[3] Folien 'Algorithmen I', KIT

Procedure qSort(a : **Array of** Element; $\ell, r : \mathbb{N}$)
  if $\ell \geq r$ **then return**
  k:= pickPivotPos(a, ℓ, r)
  m:= partition(a, ℓ, r, k)
  qSort($a, \ell, m-1$)
  qSort($a, m+1, r$)

[4]

- jetzt 'inplace'?
- man kann optimieren

[4] Folien 'Algorithmen I', KIT

Procedure qSort(a : **Array of** Element; $\ell, r : \mathbb{N}$)
  if $\ell \geq r$ **then return**
  k:= pickPivotPos(a, ℓ, r)
  m:= partition(a, ℓ, r, k)
  qSort($a, \ell, m-1$)
  qSort($a, m+1, r$) [4]

- jetzt 'inplace'?
- man kann optimieren

[4] Folien 'Algorithmen I', KIT

- Problem: schlimmstenfalls n rekursive Aufrufe $\Rightarrow$ stackoverflow
- Lösung: halbrekursive Implementierung

Procedure qSort(a : **Array of** Element; $\ell, r : \mathbb{N}$)
    while $r - \ell + 1 > n_0$ **do**
        k:= pickPivotPos(a, ℓ, r)
        m:= partition(a, ℓ, r, k)
        if $m < (\ell + r)/2$ **then** qSort($a, \ell, m-1$); ℓ:= $m+1$
        else qSort($a, m+1, r$); r:= $m-1$
    insertionSort($a[\ell..r]$) [5]

[5] Folien 'Algorithmen I', KIT

- Problem: schlimmstenfalls n rekursive Aufrufe $\Rightarrow$ stackoverflow
- Lösung: halbrekursive Implementierung

Procedure qSort(a : **Array of** Element; $\ell, r : \mathbb{N}$)
  while $r - \ell + 1 > n_0$ **do**
    k:= pickPivotPos(a, ℓ, r)
    m:= partition(a, ℓ, r, k)
    if $m < (\ell + r)/2$ **then** qSort($a, \ell, m-1$); ℓ:= $m+1$
    else qSort($a, m+1, r$); r:= $m-1$
  insertionSort($a[\ell..r]$) [5]

[5] Folien 'Algorithmen I', KIT

Sortieren - Theorie

Satz: Deterministische, vergleichsbasierte Sortieralgorithmen brauchen $n \log n - \mathcal{O}(n)$ Vergleiche im schlechtesten Fall

$\Rightarrow$ Mergesort in dieser Hinsicht optimal

Autovermietung

Du bist der Manager eines Autoverleihs. Deine Firma besitzt $k \in \mathbb{N}$ unterschiedliche Fahrzeugtypen, die alle verliehen werden können. Von jedem Fahrzeugtyp $t \in \{1 \dots k\}$ besitzt die Firma $c_t \in \mathbb{N}$ Fahrzeuge. Es liegen die n nächsten Buchungen(Abholzeitpunkt, Rückgabezeitpunkt, Fahrzeugtyp) vor. Überprüfe, ob mit den vorhandenen Fahrzeugen alle Buchungen erfüllt werden können. Zum aktuellen Zeitpunkt sind keine Fahrzeuge verliehen. Ein Fahrzeug kann, ab dem Moment, in dem es zurückgegeben wird, sofort wieder verliehen werden.

Finde einen Algorithmus in $\mathcal{O}(n \log n)$

Autovermietung

Du bist der Manager eines Autoverleihs. Deine Firma besitzt $k \in \mathbb{N}$ unterschiedliche Fahrzeugtypen, die alle verliehen werden können. Von jedem Fahrzeugtyp $t \in \{1 \dots k\}$ besitzt die Firma $c_t \in \mathbb{N}$ Fahrzeuge. Es liegen die n nächsten Buchungen(Abholzeitpunkt, Rückgabezeitpunkt, Fahrzeugtyp) vor. Überprüfe, ob mit den vorhandenen Fahrzeugen alle Buchungen erfüllt werden können. Zum aktuellen Zeitpunkt sind keine Fahrzeuge verliehen. Ein Fahrzeug kann, ab dem Moment, in dem es zurückgegeben wird, sofort wieder verliehen werden.

Finde einen Algorithmus in $\mathcal{O}(n \log n)$

Kreativaufgabe

Gegeben sei ein Array mit n verschiedenen Elementen (unsortiert, aber mit Ordnung) und eine Medianfunktion, die für ein (Teil-)Array mit m Elementen den Median deterministisch in $\mathcal{O}(m)$ berechnet.

- Finde einen Algorithmus, der das $\frac{1}{3}$-Perzentil deterministisch in $\mathcal{O}(n)$ berechnet.
- Finde einen Algorithmus, der die $\frac{1}{3^{k-1}}, \frac{1}{3^{k-2}}, \ldots, \frac{1}{3}$-Perzentile deterministisch in $O(n)$ berechnet. (Nicht in $O(nk)$!)

Kreativaufgabe

Gegeben sei ein Array mit n verschiedenen Elementen (unsortiert, aber mit Ordnung) und eine Medianfunktion, die für ein (Teil-)Array mit m Elementen den Median deterministisch in $\mathcal{O}(m)$ berechnet.

- Finde einen Algorithmus, der das $\frac{1}{3}$-Perzentil deterministisch in $\mathcal{O}(n)$ berechnet.
- Finde einen Algorithmus, der die $\frac{1}{3^{k-1}}, \frac{1}{3^{k-2}}, \ldots, \frac{1}{3}$-Perzentile deterministisch in $O(n)$ berechnet. (Nicht in $O(nk)$!)

Kreativaufgabe

Gegeben sei ein Array mit n verschiedenen Elementen (unsortiert, aber mit Ordnung) und eine Medianfunktion, die für ein (Teil-)Array mit m Elementen den Median deterministisch in $\mathcal{O}(m)$ berechnet.

- Finde einen Algorithmus, der das $\frac{1}{3}$-Perzentil deterministisch in $\mathcal{O}(n)$ berechnet.
- Finde einen Algorithmus, der die $\frac{1}{3^{k-1}}, \frac{1}{3^{k-2}}, \dots, \frac{1}{3}$-Perzentile deterministisch in $O(n)$ berechnet. (Nicht in $O(nk)$!)