Thursday, 26 October 2017 10:29

Algo-Entwurf

Induktion

- für mathematische Beweise

Invarianten

- analog zu Induktion,
  aber für Algorithmen

Invariante INV

- Schritt des Algo konserviert INV
- Anfangs gilt INV
- INV und Ende ⇒ Ergebnis

Suchen + Sortieren

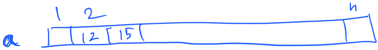
↙ Array
Zugriff: $O(1)$

1.) Suchen in sortierter Folge

2.) ——"—— unsortierter —"—

3.) Sortieren

1.)



- Schlüssel, der gesucht wird: k "key"
- Schlüssel in Folge, lin. ordn.

INV: k liegt in Bereich "links" – "rechts"
falls er überhaupt vorkommt

- wir analysieren Algorithmus
  für den <u>schlechtesten Fall</u>

```
links ← 1 , rechts ← n
while links ≤ rechts do
    mitte ← $\lfloor \frac{links + rechts}{2} \rfloor$
    if k < $a_{mitte}$ then
        rechts ← mitte − 1
```

```
    else if $a_{mitte} < K$ then
            links ← mitte+1
    else
            gefunden (mitte)
nicht gefunden
```

Analyse Anzahl Schlüsselvergleiche T ("time")

$$\boxed{\begin{aligned} T(1) &\le C \qquad \text{(Konstante)} \\ T(n) &\le C + T(n/2) \end{aligned}}$$

teleskopiere:

$$\begin{aligned} T(n) &\le C + T(n/2) \\ &\le C + C + T(n/4) \\ &\le C + C + C + T(n/8) \\ &\le k \cdot C + T(n/2^k) \\ &\le \log_2(n) \cdot C + T(1) \qquad (\text{für } k = \log_2(n)) \end{aligned}$$

Kein Beweis

formalisiere durch Induktion

Behauptung: $T(n) \le (1+\log_2(n))\cdot C$

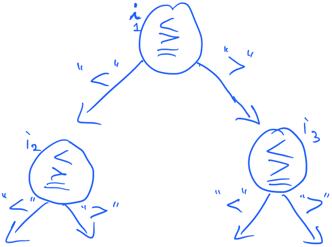
Beweis per Induktion (Übung)

Suche mit Schlüsselvergleichen

untere Schranke: $\log_2(n)$

mächtigere Vergleiche:

ternäre Operation $a_i \boxed{\begin{matrix}<\\>\\=\end{matrix}} k$



i_1
$\begin{matrix}<\\>\\=\end{matrix}$
"<"
">"
i_2
$\begin{matrix}<\\>\\=\end{matrix}$
"<"
">"
i_3
$\begin{matrix}<\\>\\=\end{matrix}$
"<"
">"

Anzahl der Knoten in diesem ↙ Anzahl Positionen im Array
Baum muss mindestens n sein

(sonst schlägt der Algorithmus
fehl wenn der gesuchte Schlüssel
in einer fehlenden Position ist)

Anzahl der Vergleiche im Schlechtesten Fall

= längster Pfad im Baum

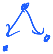
<u>mathematische Frage:</u>

wieviele Knoten kann ein binärer Baum haben wenn der längste Pfad $\leq k$ ist

$k = 0$:

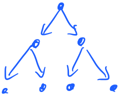
$(n=1)$

$k = 1$:



$(n=3)$

$k = 3$:

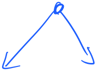


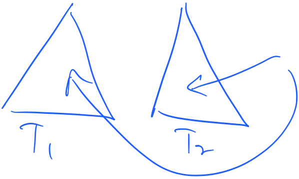
$(n=7)$

<u>Behauptung</u> $n = 2^k - 1$

Beweis per Induktion

beliebiger $\overset{\text{bin.}}{\vee}$ Baum mit Pfad länge k





bin. Baum
mit Pfad länge $\leq k-1$
Ind. Hyp.

$$\Rightarrow \quad n_1 \leq 2^{k-2}-1$$
$$n_2 \leq 2^{k-2}-1$$

$$\Rightarrow \quad n \leq n_1 + n_2 + 1$$
$$\leq 2 \cdot 2^{k-2} - 1$$
$$= 2^{k-1} - 1$$

damit gilt $k \geq \log_2(n)$
für jeden Baum mit n knoten
und Pfad länge $\leq k$

$\rightsquigarrow$ im schlechtesten Fall $\log_2(n)$ Vergleiche

---

Suche in unsortierter Folge

Schlüssel k

Array / Folge $a = (a_1, \dots, a_n)$ (nicht sortiert)

lineare Suche: n Vergleiche
im schlechtesten Fall

untere Schranke:

zwei Sorten von Vergleichen:

- vergleiche zwei Arrayelemente
- vergleiche Schlüssel mit Array element

Gruppen von Array elementen
(Gruppe von miteinander
verglichenen Array elementen)

g Gruppen ⇒ $\geq g$ Vergleiche
mit Schlüssel

Behauptung wir brauchen $\geq n-g$
Vergleiche um g Gruppen von Array elem.
zu erhalten

Beweis Anfang n Gruppen
jeder Vergleich reduziert Anzahl
der Gruppen um höchstens 1.

3. Sortieren

"$a_i \leq a_{i+1}$"

(3.a.) n Vergleiche um zu prüfen

dass Array sortiert ist

3.b.

idee:

```
prüfe ob Array sortiert
wenn "$a_i \nleq a_{i+1}$" dann
    tausche $a_i$, $a_{i+1}$
```

Bubblesort

```
wiederhole n mal:
    for $i = 1 .. n-1$
        if $a_i > a_{i+1}$ tausche $a_i, a_{i+1}$
```

Korrekt da nach i Wiederholungen die letzten i Positionen korrekt sind

Anzahl Vergleiche: $O(n^2)$

3.c.

INV: die letzten Positionen von $i+1 .. n$ sind korrekt (so wie bei der endgültigen Sortierung)

Schritt: finde max im Bereich $1 .. i$
tausche es an Stelle i

"Sortierung durch Auswahl"
(selection sort)

Analyse $O(n^2)$ Vergleiche

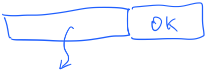
(Beweis als Übung)

Idee:

„goldene Mitte"

finde maximum in weiteren Iterationen schneller als bei der ersten Iteration

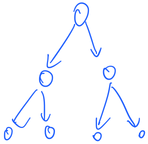
3.d.



Struktur

- max finden
- Struktur wieder herstellen

Struktur: binärer Baum



Nachfolger kleiner

⇝ Wurzel hat Maximum

Wurzel von Teilbäumen sind Maximum des Teilbaum

genannt: Heap („Haufen")