Informatik II - SS 2014

(Algorithmen & Datenstrukturen)

Vorlesung 7 (21.5.2014)

Binäre Suche, Hashtabellen I

UNI FREIBURG

Fabian Kuhn

Algorithmen und Komplexität

Benutze sortiertes Array?

- Teure Operation bei Liste/Array, insbesondere ***find***
- Falls (sobald) sich die Einträge nicht zu sehr ändern, ist *find* die wichtigste Operation!
- Kann man in einem (nach Schlüsseln) sortierten Array schneller nach einem bestimmten Schlüssel suchen?
  - Beispiel: Suche Tel.-Nr. einer Person im Telefonbuch…

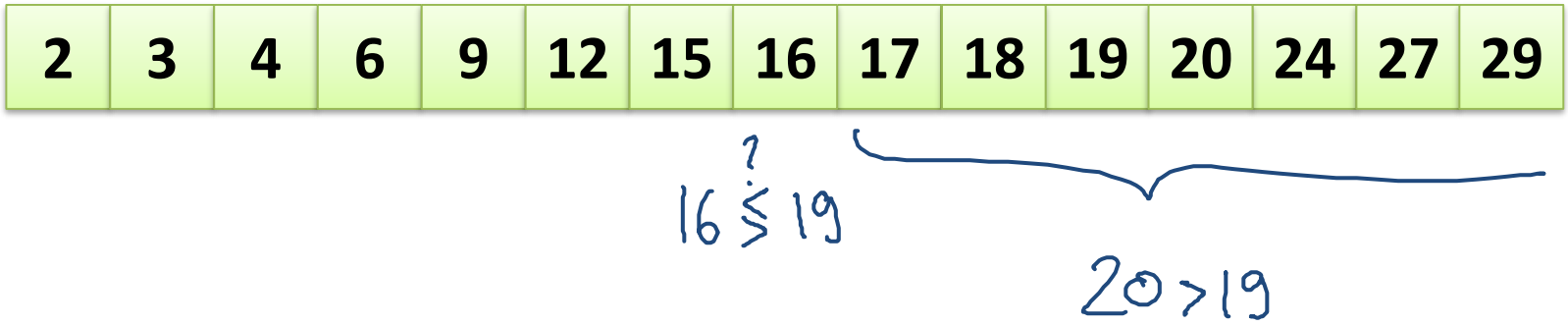
Ideen:

Gehe in die Mitte
und dann entweder links oder rechts

Binäre Suche

Benutze Divide and Conquer Idee!

Suche nach der Zahl (dem Key) 19:



Binäre Suche

2	3	4	6	9	12	15	16	17	18	19	20	24	27	29

Algorithmus (Array A der Länge n, Suche nach Schlüssel x):

```
ℓ = 0; r = n-1
while r > ℓ  do
    m = ⌊(ℓ+r) / 2⌋
    if A[m] < x then    (x ist rechts von m)
        ℓ = m+1
    else if A[m] > x then    (x ist links von m)
        r = m-1
    else
        ℓ = m; r = m
```

$A[\ell] \stackrel{?}{=} x$

Binäre Suche

2	3	4	6	9	12	15	16	17	18	19	20	24	27	29

Algorithmus (Array A der Länge n, Suche nach Schlüssel x):

```
l = 0; r = n – 1;
while r > l do
    m = (l + r) / 2;
    if A[m] < x then
        l = m + 1
    else if A[m] > x then
        r = m – 1
    else
        l = m; r = m
```

Falls Schlüssel x im Array ist, dann gilt am Schluss $A[l] = x$

Ist der Algorithmus korrekt?

UNI FREIBURG

Wie überprüft man das?

- Empirisch: Unit Test oder auch systematischere Tests…
- Formal?
  - Korrektheit ist (meistens) noch wichtiger als Performance!

- **Vorbedingung**
  - Bedingung, welche am Anfang (der Methode / Schleife / …) gilt

- **Nachbedingung**
  - Bedingung, welche am Schluss (der Methode / Schleife / …) gilt

- **Schleifeninvariante**
  - Bedingung welche am Anfang / Ende jedes Schleifendurchlaufs gilt

Ist der Algorithmus korrekt?

```
l = 0; r = n - 1;
while r > l do
    m = (l + r) / 2;
    if A[m] < x then l = m + 1
    else if A[m] > x then r = m - 1
    else l = m; r = m
```

l x r

Vorbedingung

- *Array ist am Anfang sortiert, Array hat Länge n*

Nachbedingung

- *Falls x im Array ist, dann gilt $A[l] = x$*

Schleifeninvariante

- *Falls x im Array ist, dann gilt $A[l] \leq x \leq A[r]$*

Ist der Algorithmus korrekt?

while (r>l) do ...

Vorbedingung

- *Array ist am Anfang sortiert, Array hat Länge n*

```
l = 0; r = n - 1;
```

Schleifeninvariante

$A[0] \leq x \leq A[n-1]$

- *Falls x im Array ist, dann gilt $A[l] \leq x \leq A[r]$*
- Vorbedingung und Zuweisung zu l und r → Schleifeninvariante
  - Invariante gilt am Anfang des ersten Schleifendurchlaufs

Nachbedingung

$A[r] \leq A[l] \leq x \leq A[r]$

A ist sortiert

- *Falls x im Array ist, dann gilt $A[l] = x$*
- Abbruchbedingung while-Schleife → $l \geq r$ und damit $A[l] \geq A[r]$
- Falls x im Array ist, dann folgt aus der Schleifeninvariante und da A sortiert ist, dass $A[l] = A[r]$ und damit $A[l] = x$

Ist der Algorithmus korrekt?

```
l = 0; r = n - 1;
while r > l do
    m = (l + r) / 2;
    if A[m] < x then l = m + 1
    else if A[m] > x then r = m - 1
    else l = m; r = m
```

$A[l] \leq x \leq A[r]$

INV

INV

Schleifeninvariante

- *Falls x im Array ist, dann gilt $A[l] \leq x \leq A[r]$*

$l = m+1$: $l \overset{x}{\sim} m \; r$, $A[m] < x$

$r = m-1$: analog

Terminiert der Algorithmus?

```
l = 0; r = n - 1;
while r > l do
    m = (l + r) / 2;
    if A[m] < x then l = m + 1
    else if A[m] > x then r = m - 1
    else l = m; r = m
```

$\frac{l+r}{2} - \frac{1}{2} \leq m \leq \frac{l+r}{2}$

- Veränderung der Anz. Elemente $(r - l + 1)$ pro Schleifendurchlauf?
  - $l = m + 1$: (neues l)

$$r - (m+1) + 1 \leq r - \left(\frac{l+r}{2} + \frac{1}{2}\right) + 1 = \frac{r-l+1}{2}$$

  - $r = m - 1$:

$$(m-1) - l + 1 \leq \frac{l+r}{2} - 1 - l + 1 = \frac{r-l}{2} < \frac{r-l+1}{2}$$

  - Sonst wird x gefunden und $r - l + 1$ wird 1

Laufzeit

Terminiert der Algorithmus?

- In jedem Schleifendurchlauf wird die Anzahl der Elemente mindestens halbiert.
- Der Algorithmus terminiert!

Laufzeit?

$$T(n) \leq T(\lceil n/2 \rceil) + c, \qquad T(1) \leq c$$

$$\begin{aligned} T(n) &\leq T(n/2) + c \\ &\leq T(n/4) + 2c \\ &\leq T(n/8) + 3c \\ &\vdots \\ &\leq T(n/2^k) + k \cdot c \\ &\vdots \\ &\leq T(1) + c \cdot \log_2 n \leq c(1 + \log n) \end{aligned}$$

Laufzeit Binäre Suche

Der Algorithmus terminiert in Zeit $O(\log n)$.

$T(n) \leq T(\lfloor n/2 \rfloor) + c$ $\qquad$ $T(n) \leq c(1+\log n)$

Verankerung: $T(1) \leq c(1+\log 1) = c$ ✓

Schritt: $T(n) \leq T(\lfloor n/2 \rfloor) + c$

$\leq c(1+\log \lfloor n/2 \rfloor) + c$

$\leq c(1+\log(n/2)) + c$

$= c(1+\log_2 n)$ ✓

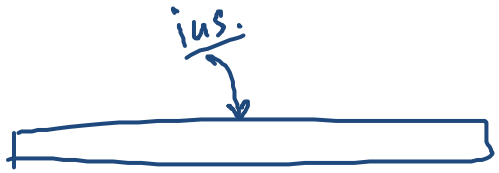
$\log_2(n/2) = \log_2 n - 1$

sind in Zeit $O(\log n)$

Dictionary mit sortiertem Array

Operationen:

- *create*:
  - lege neues Array der Länge *NMAX* an
- *D.find(key)*:
  - **Suche nach *key* mit binärer Suche**
- *D.insert(key, value)*:



  - suche nach *key* und füge neues Element an der richtigen Stelle ein
  - Einfügen: alles dahinter muss um eins nach hinten geschoben werden!
- *D.delete(key)*:
  - suche zuerst nach dem *key* und lösche den Eintrag
  - Löschen: alles dahinter muss um eins nach vorne geschoben werden!

Dictionary mit sortiertem Array

Laufzeiten:

create: $O(1)$

insert: $O(n)$

find: $O(\log n)$

delete: $O(n)$

Können wir alle Operationen schnell machen?

- und das *find* noch schneller?