

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:

Binäre Suche

Benutze Divide and Conquer Idee!

Suche nach der Zahl (dem Key) 19:

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

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):

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 $A[l] = x$

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
```

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?

Vorbedingung

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

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

Schleifeninvariante

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

Nachbedingung

- *Falls x im Array ist, dann gilt $A[l] = x$*
- Abbruchbedingung while-Schleife $\rightarrow l \geq r$ 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
```

Schleifeninvariante

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

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
```

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

- $l = m + 1$:

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

- $r = m - 1$:

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

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

Terminiert der Algorithmus?

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

Laufzeit?

$$T(n) \leq T(\lfloor n/2 \rfloor) + c, \quad T(1) \leq c$$

Laufzeit Binäre Suche

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

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!

Laufzeiten:

create:

insert:

find:

delete:

Können wir alle Operationen schnell machen?

- und das *find* noch schneller?