Hochschule Offenburg
University of Applied Sciences

Algorithmen und Datenstrukturen
4. Suchen

Prof. Dr. Klaus Dorer

Übersicht

- Suchen
  - Sequentielle Suche
  - Binäre Suche
  - Exponentielle Suche
  - Interpolationssuche

Quellen

- Thomas Ottmann und Peter Widmayer, Algorithmen und Datenstrukturen, 4. Auflage, Spektrum, Berlin, 2002
- Stefan Trahasch, Vorlesungsfolien Algorithmen und Datenstrukturen, Hochschule Offenburg
- www.wikipedia.de

Ziele

- Relevante Begriffe kennen
- Verschiedene Suchverfahren kennen und implementieren können
- Komplexität der Verfahren kennen und einordnen können

Problemstellung

- **Ziel**
  Zur effizienten Verwaltung einer (dynamischen) Menge von Datensätzen sind geeignete Datenstrukturen und Algorithmen gesucht.
- **Datensätze** (records) bestehen aus
  - Suchschlüssel (search key) und
  - Nutzdaten (value)
- **Beispiele**

Telefonbuch

Meier	Michael	Hauptstr. 17	921566
Müller	Franz	Mozartweg 15	765322
Baier	Caroline	Bergstr. 23	876599
Schmidt	Monika	Seeweg 1	445369

Wörterbuch

sehen	see; look
sprechen	speak; talk
gehen	go; walk
hören	hear; listen

Suchen

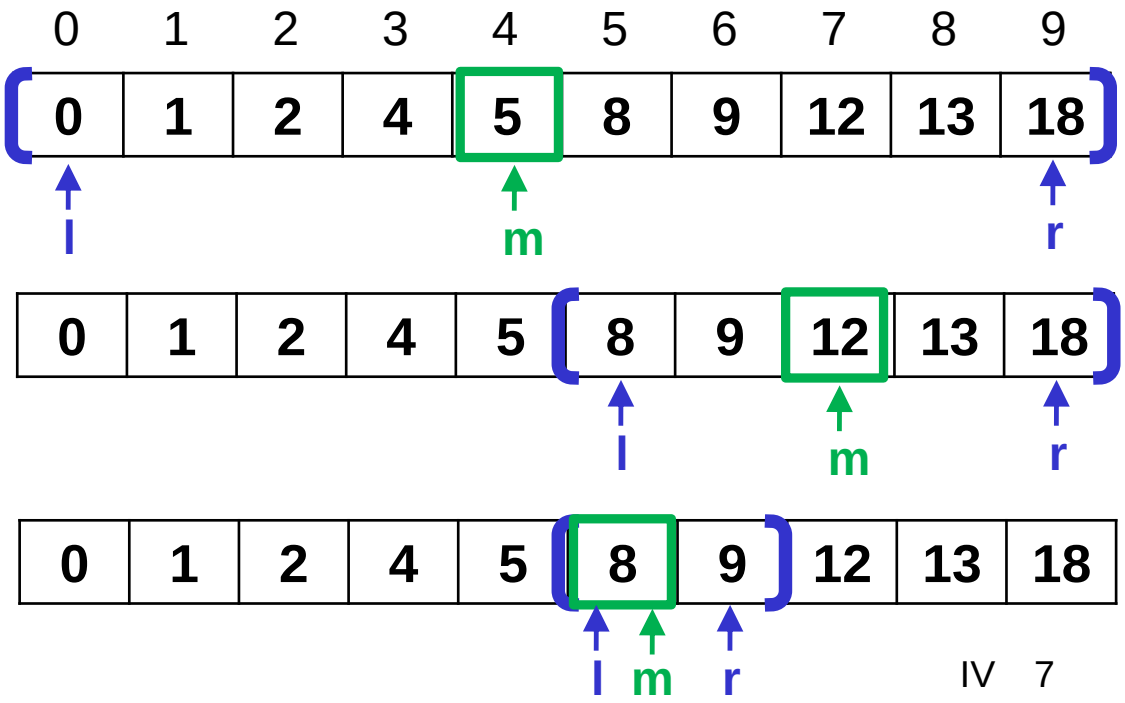
- Finde einen Schlüssel in einer Menge von Daten
  - Erfolgreich: Schlüssel wurde gefunden
  - Erfolglos: Schlüssel nicht in Menge vorhanden
- Elementare Suchverfahren
  - Nur Vergleiche zwischen Schlüsseln erlaubt
  - Keine Arithmetischen Operationen (wie später beim Hashing)
- Suche in
  - Unsortierten Mengen
  - Sortierten Mengen

Sequentielle Suche

- Durchlaufen aller Elemente, bis
  - der Schlüssel gefunden wurde (erfolgreich)
  - das Ende der Liste erreicht ist (erfolglos)
- Daten müssen nicht sortiert sein
- Daten können in verketteter Liste vorliegen, da kein direkter Zugriff auf das i te Element nötig ist
- Komplexität
  -
  - Operationen im schlechtesten Fall:
  - Operationen im Mittel (erfolgreich):
  - Operationen im Mittel (erfolglos):

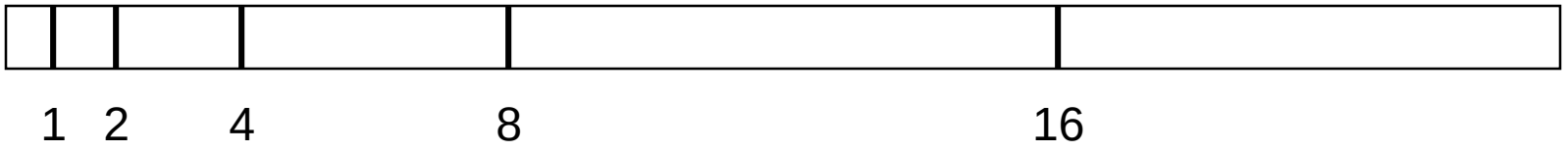
Binäre Suche

- Divide and Conquer
  - Falls Liste leer erfolglos ansonsten betrachte mittleres Element m
  - Falls m > gesuchter Schlüssel k durchsuche linke Hälfte
  - Falls m < k durchsuche rechte Hälfte
  - Sonst ist m der Schlüssel zum gesuchten Element
- Voraussetzungen
  - sortierte Daten
  - direkter Zugriff auf i tes Element
- Beispiel
  - Suche nach Schlüssel 8
- Komplexität
  -



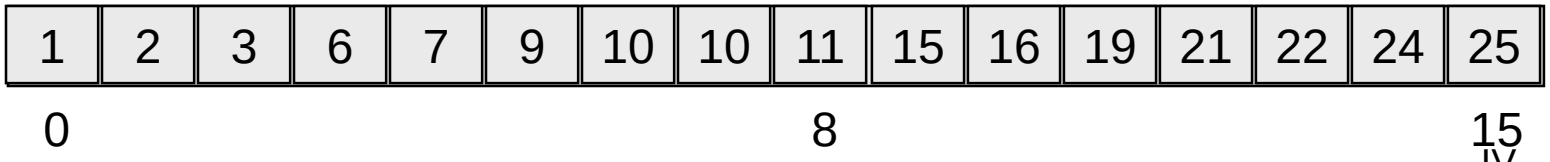
Exponentielle Suche

- Binäre Suche setzt voraus, dass die Größe des Suchbereichs bekannt ist
- Wenn n sehr groß ist kann es sinnvoll sein, zuerst den Suchbereich zu bestimmen
  - Beim ersten Element starten und dann Index jeweils verdoppeln
    - i = i + i;
  - Solange bis a[i/2] < k <= a[i]
  - Dann binäre Suche in diesem Intervall
- Komplexität
  -



Interpolationssuche

- Sucht man im Telefonbuch nach dem Namen „Bauer“, so schlägt man vorne auf, bei „Wild“ hinten
  - Binärsuche: m = l + ½(r-l)
  - Interpolationssuche: ersetze ½ durch die erwartete Position des Schlüssels (k – a[l]) / (a[r] – a[l])
  - Verlangt arithmetische Operationen auf dem Schlüssel
- Komplexität
  - O (n) im schlechtesten Fall
  - O(log log n) im Mittel, wenn Schlüssel gleichverteilt sind
- Beispiel
  - Suche nach 24
  - Binärsuche 1 2 3 4
  - Interpolationssuche 1



Vergleich

- Vergleich der Laufzeit
  - N = Anzahl Elemente in der Liste
  - Loops = Anzahl Suchvorgänge mit ca. 50% Fehlschlägen
  - * im Bereich der Messungenauigkeit
  - ** geschätzt

Verfahren	N=10.000 Loops = 10.000	N=1.000.000 Loops = 1.000.000
Sequentiell	5,44 s	54.400 s (15 h)**
Binär	0,01* s	2,53 s
Exponentiell	0,01* s	3,59 s
Interpolation	0,01* s	2,15 s

Suchen in Java

- Die Klassen Arrays und Collections bieten bereits Methoden für die binäre Suche

```
// make sure namesArray/namesList are sorted before

int index = Arrays.binarySearch(namesArray, "Kroos");

int index = Collections.binarySearch(namesList, "Kroos");
```

Zusammenfassung

Strategie	Voraussetzung	Kurzbeschreibung	Komplexität Zeit
sequentiell	keine	Elemente werden nacheinander mit dem Suchelement verglichen	O(n)
binär	Elemente geordnet, wahlfreier Zugriff	Rekursive Suche in entweder linker oder rechter Hälfte je nach Wert des mittleren Elements (gleich große Hälften)	O(log n)
exponentiell	Elemente geordnet, wahlfreier Zugriff	Für sehr großes n wird vor einer binären Suche der Suchbereich auf eine Länge 2^i eingeschränkt	O(log k) für Suchschlüssel k
Interpolation	Elemente geordnet, wahlfreier Zugriff, Subtraktion auf Schlüsseln definiert	Berechne wahrscheinliche Position, sonst wie binäre Suche	O(n) schlechtester Fall O(log log n) Mittel