Algorithmen und Datenstrukturen

Master

Suchen

Lineare Suche

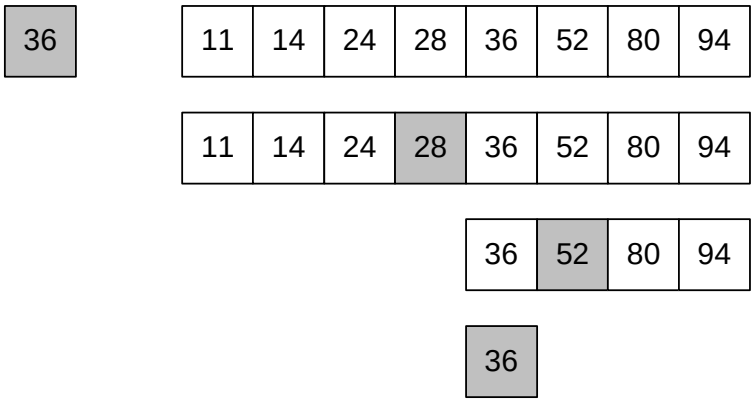
- Daten werden sequentiell mit dem Schlüssel verglichen.
- Worst Case Aufwand: O(N)

- Vorsortierung nicht notwendig

Binärsuche

- Nur bei sortierten Daten
- Daten müssen im direkten Zugriff stehen, d.h. nur bei Array anwendbar
- Prinzip: Datenbereich wird schrittweise um die Hälfte eingeschränkt
- Aufwand O(log N)

Binärsuche



Fibonacci-Suche

- Analog zur binären Suche
- Kommt allerdings mit Addition und Subtraktion aus (keine Division!)

$$F(0) = 0, F(1) = 1,$$
$$F(n) = F(n-1) + F(n-2) \text{ für } n \geq 2$$

- Aufwand abschätzbar $O(1{,}618^n)$

Exponetielle Suche

- Binäre/Fibonacci Suche gehen davon aus, dass die Anzahl der Elemente bei Beginn der Suche bekannt ist.
- Für Fälle, wo der Suchbereich zwar endlich aber praktisch unbegrenzt ist, ist es günstig, zunächst eine obere Grenze für den zu durchsuchenden Bereich zu bestimmen.
- Weitere Suche dann wieder binär.

Exponetielle Suche

- Der verfügbare Suchbereich wird zunächst eingegrenzt durch

$$a\left[\frac{i}{2}\right] < k < a[i]$$

- Das erreicht man durch fortlaufendes Erhöhen des Index um sich selbst
  - i=0;
  - while (k>a[i]) i = i + i;

Selbstorganisierende lineare Listen

- Idee: verschiebe häfig verwendete Einträge nach vorn
  - MF (move to front): mach zuletzt gesuchtes Element zum Ersten in der Liste
  - T (transpose): vertausche Element mit dem unmittelbar vorangehenden, wenn darauf zugegriffen wurde
  - FC (frequency count): ordne die Elemente nach der Häufigkeit der Zugriffe – zusätzlicher Zähler pro Element

Selbstorganisierende lineare Listen

- MF und FC sind i.A. besser als T
- FC kostet zusätzlich Speicherplatz
- MF kann kurzfristig durch selten zugegriffene Elemente aus dem Tritt kommen – Korrektur erfolgt langsam

Übung 3a

- Verwende wieder die CArray Klasse für 1000 zufällige Elemente
  - Füge eine neue Variable compCount hinzu
  - Suche nach 10 zufällig bestimmten Werten
    - Linear (unsortiertes Array)
    - Binary Search (sortiertes Array)
  - Und gib jeweils die Anzahl der Vergleiche aus
  - Implementiere auch eine Funktion min() und max()

Übung 3b

- Aus Übung 2 wird ein sortiertes Array (absteigend) gewonnen
- Implementiere eine Fibonacci-Suche in diesem Array und gib für 3 Elemente die Such-Indizes an!