DHBW Lörrach
Duale Hochschule
Baden-Württemberg

Algorithmen und Komplexität
TIF 21 A/B
Dr. Bruno Becker

2. Abstrakte Datentypen

Abstrakte Datentypen

- **Einführung und Beispiele**
- Arrays
- Verkettete Listen
- Stacks & Queues

Einführung

- **Datenstruktur** bezeichnet bestimmte Art, Daten im Speichers eines Computers zu verwalten.
- Bietet **Operationen** an zum Lesen, Einfügen, Löschen,…
- **Abstrakte Datentypen** beschreiben Verhalten von Datenstrukturen, die von jeglichen Implementierungsdetails abstrahiert
- **Datenstrukturen** können als **statische** (d.h. feste Größe) oder **dynamische** (d.h. variable Größe) Strukturen umgesetzt werden

Beispiele

- Felder (Arrays) - statisch
  - 1-dimensional: Vektor a[0..n-1]
  - 2-dimensional: Tabelle a[0,..n-1][0,..m-1]
- Listen – dynamisch
  - Einfach oder doppelt verkettet
- Mengen
- Schlangen
- Bäume
- Graphen

Abstrakte Datentypen

- Einführung und Beispiele
- **Arrays**
- Verkettete Listen
- Stacks & Queues

Array (Feld)

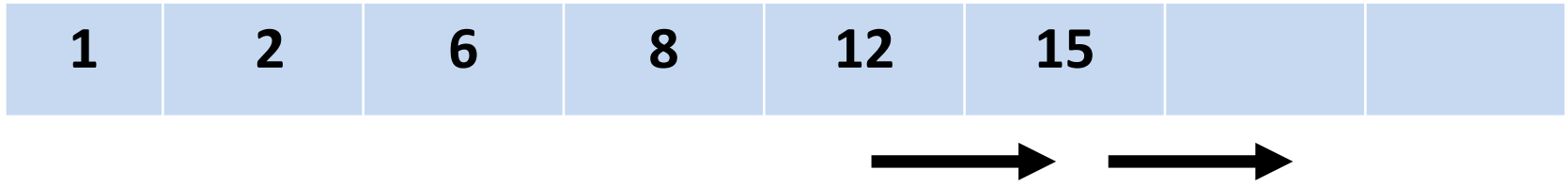
- Feste Anzahl von Elementen gleichen Typs
- Schneller wahlfreier Zugriff auf jedes Element
- Zugriffszeit unabhängig von Position für jedes *i von 0 bis N*: *a[i]*
- Länge (Anzahl Elemente) wird bei Erzeugung festgelegt, **danach nicht veränderbar**

➔ Statische Datenstruktur

Array – Einfügen und Löschen von Elementen

- Einfügen in unsortiertes Array: Besetze nächste freie Position
  z.B. am Ende

- Sortiertes Sortiertes Array? Beispiel: Füge Element mit Schlüssel **9** ein



Aufwand: ?

- Löschen an Position i?

➔ **Einfügen und Löschen im Array teuer, außer am Ende!**

Arrays: Suche

- Binärsuche im sortierten Array: **sehr schnell!**
- Aber: Array muss sortiert gehalten werden!
- Problem: Einfügen, Löschen kosten linearen Aufwand

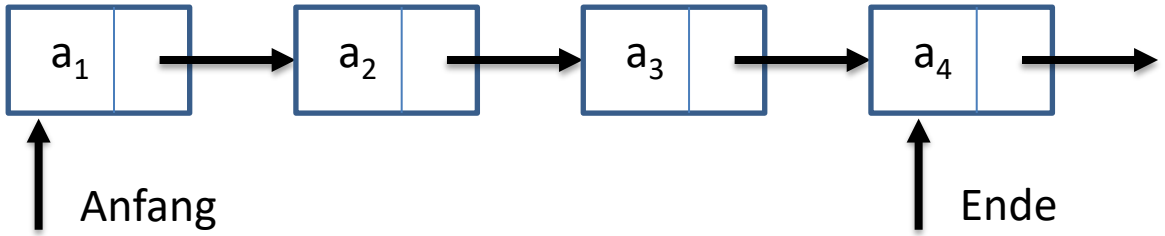
- **Bei häufigen Veränderungen nicht effizient**!

Abstrakte Datentypen

- Einführung und Beispiele
- Arrays
- **Verkettete Listen**
- Stacks & Queues

Verkettete Listen

- **Dynamische Datenstruktur:** Speicher für Elemente wird bei Bedarf reserviert (allokiert), theoretisch unbegrenzt.
- Elemente haben einen gegebenen Datentyp (*record*) mit Zeiger auf Nachfolger.
- Elemente werden in linearer Liste gespeichert.
- Zusätzliche Zeiger auf Anfang oder Ende
- Liste kann nur sequentiell durchlaufen werden, **kein** direkter Zugriff



Verkettete Liste: Implementierung

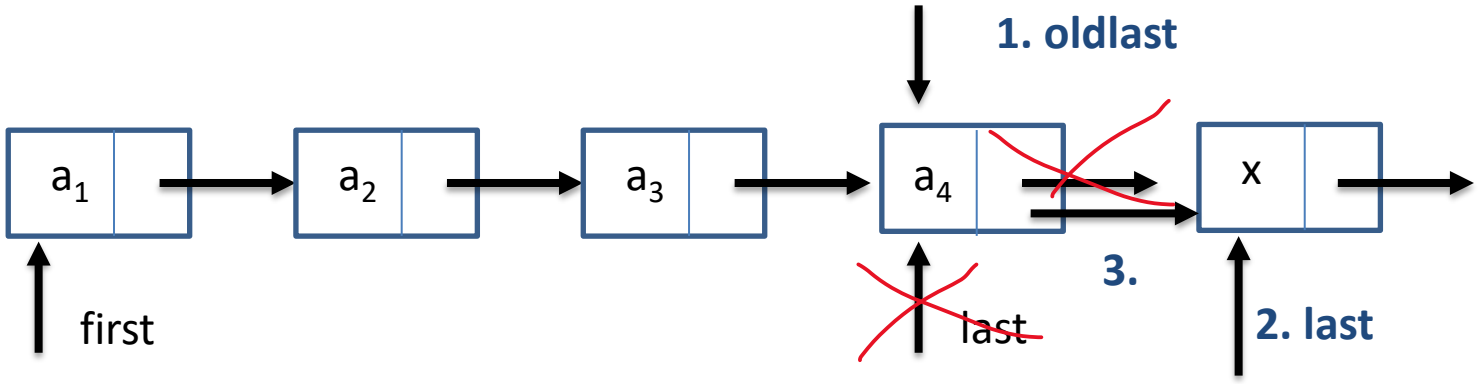
- Einfach verkettete Liste von *Items in Java:*

```
public class Listofitems {
  private Node start = null; // erstes Element
  private class Node {
      Item item;
      Node next;
   }
}
```

- Welche Operationen werden nun benötigt?
- *Einfügen (x,L); Löschen (x,L); Suchen p-tes Element in Liste Suchen(p,L):x;*

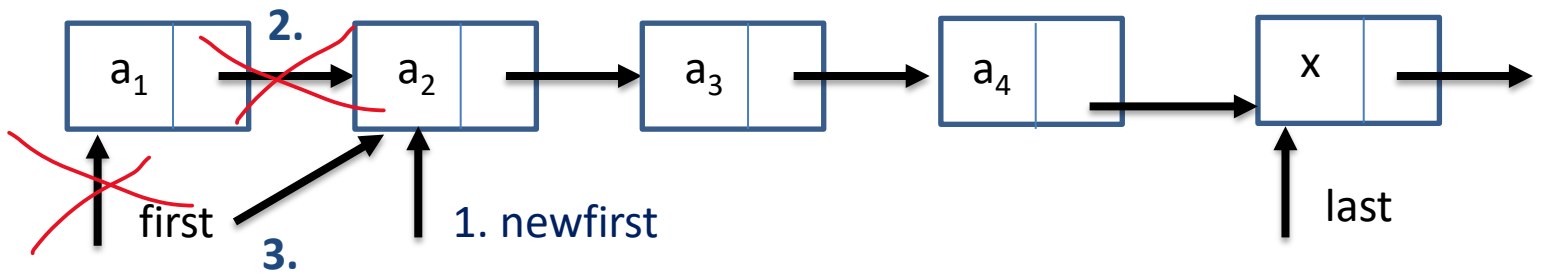
Verkettete Liste: Einfügen eines Elementes

- Wo soll eingefügt werden? Am Anfang oder Ende oder zwischendrin?
- Beispiel Einfügen am Ende (item x):
  1. Node „oldlast“ zeigt auf letztes Element : Node oldlast = last
  2. Erzeuge neuen Knoten für das Ende: last = new Node(); last.item = x;
  3. Erzeuge eine Referenz vom Ende der Liste auf den neuen Knoten:
     oldlast.next = last;



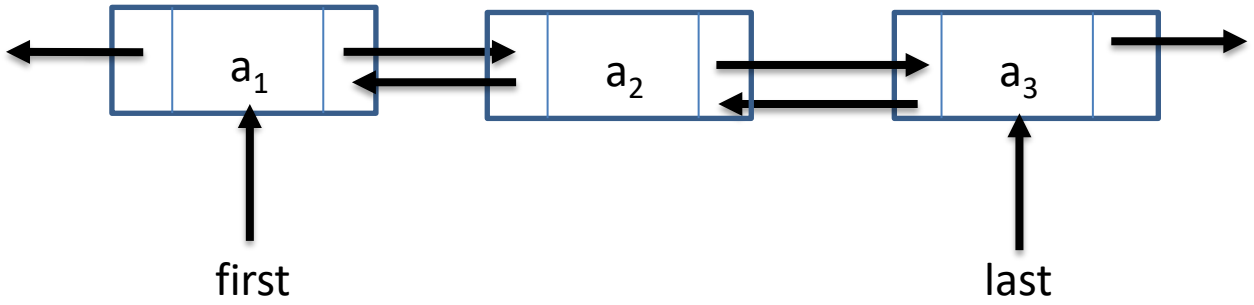
Verkettete Liste: Löschen eines Elementes

- Löschen am Anfang der Liste:
- Beispiel Einfügen am Ende (item x):
  1. Setze Zeiger auf neuen Anfang der Liste Node newfirst = first.next;
  2. First.next = null;
  3. First = newfirst;
- Wie funktioniert Löschen am Ende?
  - Problem: Zugriff auf Vorgänger



Doppelt Verkettete Liste

- Variante: Doppelt verkettete Liste: Zeiger auf Vorgänger und Nachfolger



→Damit Löschen am Ende einfacher

- Bei Such- und Änderungsoperationen Sonderfälle beachten:
  - Zeiger auf *Null,* z.B. bei leerer Liste
  - Variante: *Dummies (d.h. Elemente ohne Inhalt)* am Anfang und Ende (s. Übungsblatt)

Verkettete Liste: Aufwandsbetrachtungen

- Einfügen an *bekannter* Position (d.h. Zeiger auf das Element)
  - Konstanter Aufwand
- Einfügen an unbekannter Position (d.h. kein Zeiger auf das Element)
  - Zuerst Einfügestelle suchen, dann Einfügen
  - Beispiel sortierte List: Linearer Aufwand (bezogen auf #Elemente)
- Löschen an *bekannter* Position (d.h. Zeiger auf das Element)
  - Konstanter Aufwand
- Löschen an unbekannter Position (d.h. kein Zeiger auf das Element)
  - Analog zum Einfügen
- Suche nach Element
  - Erfolglose Suche bei sortierter Liste schneller als bei unsortierter Liste

Abstrakte Datentypen

- Einführung und Beispiele
- Arrays
- Verkettete Listen
- **Stacks & Queues**

Stack

- **Stack *(Stapel, Keller):*** Überfüllter Bus, Tellerstapel,…
  Last in, First out (LIFO)
- **Operationen*:***
  - Letzter Kunde (Element) wird bedient
  - Neuer Kunde (Element) wird vorne angehängt
- Beispiel: Erkennen wohlgeformter Klammerausdrücke (WKA)
  - Definition eines WKA: $\{\}, (),$ ist WKA. Sind w_1 und w_2 WKA, dann sind auch w_1w_2, $\{w_1\}$ WKAs.
  - Zeichenreihe wird zeichenweise gelesen.
    - Ist das Zeichen eine öffnende Klammer, wird sie auf einen Keller gelegt
    - Ist das Zeichen eine schließende Klammer, wird sie vom Keller geholt

Stack: Implementierung

- Typisches minimales API eines Stacks *von Items*

```
public class Stackofitems

  void push (Item item) // Legt neues Element auf den Stack
  Item pop() // Entfernt oberstes Element vom Stack und gibt es zurück
  boolean isEmpty() // ist der Stack leer?
```

- Implementierung als einfach verkettete Liste mit Zeiger auf Anfang
  - Die Operationen push und pop arbeiten am Anfang der Liste

Stacks : Anwendungen

- Compiler: Syntaxanalyse (siehe Beispiel Klammerausdrücke)
- Auswertung arithmetischer Ausdrücke
- Implementierung rekursiver Funktionen
- Java-VM ist eine *Stack-Maschine*
- Undo-Funktion im Anwendungen
- Back-Button im Browser….

Queues (Schlangen)

- **Warteschlangen:** Supermarktkasse, Auto-Maut, Druckaufträge,…
  First in, First out (FIFO)
- **Operationen*:***
  - Erster Kunde (Element) wird bedient
  - Neuer Kunde (Element) wird hinten angehängt

Queue Implementierung

- Typisches minimales API eines Stacks *von Items*

```
public class Queueofitems

  void enqueue (Item item) // Fügt neues Element in Queue ein
  Item dequeue() // Entfernt ältestes (letztes) Element aus Queue und gibt es zurück
  boolean isEmpty() // ist die Queue leer?
```

- Implementierung als einfach verkettete Liste mit Zeiger auf Anfang und Ende
  - Die Operationen enqueue am Ende; dequeue am Anfang
  - Könnte man bei dieser Implementierung Reihenfolge auch umdrehen?

Queues : Anwendungen

- Betriebssysteme: Wartende Prozesse
- Druckerwarteschlange
- Kommunikationstechnik: Puffer für zu sendende Daten
- Webserver: Request-Queue
- Ticketing-System
- …