

# Algorithmen und Datenstrukturen

## B3. ADTs , Bags, Stack and Queues

Marcel Lüthi and Gabriele Röger

Universität Basel

21. März 2019

# Algorithmen und Datenstrukturen

21. März 2019 — B3. ADTs , Bags, Stack and Queues

## B3.1 Abstrakte Datentypen

## B3.2 Multimengen, Warteschlange und Stapel

## B3.3 Anwendung von Stacks

## B3.4 Priority Queues

## B3.1 Abstrakte Datentypen

## Abstrakte Datentypen : Definition

### Abstrakter Datentyp

Die Beschreibung eines Datentyps durch eine Zusammenfassung von Daten und anwendbaren Operationen.

Beispiele:

- ▶ Integer mit arithmetischen Operationen
- ▶ Komplexe Zahlen mit Operationen add und subtract
- ▶ Mengen mit Operationen union, intersection und setminus
- ▶ Geordnete Sequenz von von Objekten

## Informatikerin des Tages



Barbara Liskov

- ▶ Eine der ersten Frauen in USA mit Doktor in Informatik
- ▶ Gewinnering des Turing Awards
- ▶ Hat Konzept von „Abstrakt Data Types“ eingeführt.

Liskov, Barbara, and Stephen Zilles. Programming with abstract data types. ACM Sigplan Notices. ACM, 1974.

## Abstrakte Datentypen und Klassen

- ▶ Abstrakte Datentypen entsprechen Klassen in OO Programmierung

```
public class Complex {
    private double real;
    private double imag;

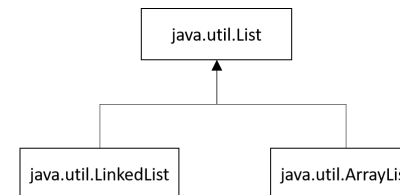
    public Complex(double real, double imag) { ... }
    public Complex(double magnitude, double phase) { ... }

    public Complex add(Complex c1, Complex c2) { ... }
    public Complex subtract(Complex c1, Complex c2) { ... }
    ...
}
```

## Vorteile von Abstrakten Datentypen

- ▶ Nutzer programmiert gegen Schnittstelle
- ▶ **Verwendete Datenstruktur** (Repräsentation) ist versteckt (gekapselt)
  - ▶ Repräsentation kann jederzeit ausgetauscht werden
- ▶ Verständnis auf zwei Ebenen
  - 1 Was macht der Datentyp (Schnittstelle)
  - 2 Wie wird es gemacht (Interne Datenstruktur)
- ▶ Erlaubt komplexe Sachverhalte zu abstrahieren

## Beispiel: Listen in Java



```
interface List<E>:
    E get(int index);
    void add(E element);
    void add(int pos, E element);
    ...
```

### Achtung

Verschiedene Listen haben dieselbe Schnittstelle, aber Operationen haben nicht dieselbe Komplexität.

## Datentypdesign

Wir werden für jeden Datentyp folgende Punkte besprechen

- ▶ Beschreiben der Schnittstelle (API)
- ▶ Beispielanwendungen (Client) die die Schnittstelle nutzen
- ▶ Implementation

## Quiz: Abstrakte Datentypen

- ▶ Ist eine verkettete Liste ein Datentyp oder eine Datenstruktur?
- ▶ Ist ein Array nur eine Datenstruktur oder auch Abstrakter Datentyp?
  - ▶ Was wären die Operationen auf einem Array, welche den ADT Array Charakterisieren?
  - ▶ Welche Datenstruktur würden Sie für die Implementation eines Array Datentyps verwenden?
- ▶ Was ist die Gefahr, bei der Verwendung eines abstrakten Datentyps?

## B3.2 Multimengen, Warteschlange und Stapel

### Ein Besuch in der Mensa



(Teller-)Stapel



Multimenge (von Essen)



Schlange

Stapel, Multimenge und Schlange sind wichtige Datentypen, die wir vom täglichen Leben kennen.

## Multimengen (Bag)

```
class Bag[Item]:
  # Element hinzufügen
  def add(item : Item) -> Item

  # Ist die Multimenge leer?
  def isEmpty() -> bool

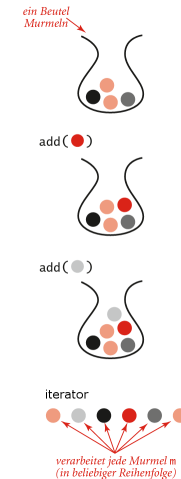
  # Wieviele Elemente sind in der Menge?
  def size() -> int

  # Abstraktion um ueber Elemente zu iterieren
  def iterator() -> Iterator[Item]
}
```

- Anmerkung: Typ Annotation angelehnt an Python Typing Module (PEP 484)

## Multimenge (bag)

- undefinierte Reihenfolge der Elemente
  - Welches Element man nimmt ist undefiniert.
  - Aber: Jedes Element wird nur einmal entnommen
- Nicht zu verwechseln mit Liste / Array, die die Reihenfolge garantieren.



Quelle: Abbildung 1.30 - Algorithms, Sedgewick & Wayne

## Warteschlange (Queue)

```
class Queue[Item] {

  # Element zu Schlange hinzufügen
  def enqueue(item : Item)

  # Element von Schlange entfernen
  def dequeue() -> Item

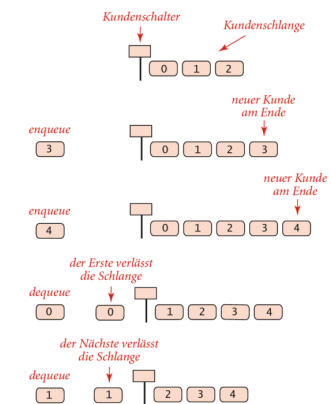
  # Anzahl Elemente in der Schlange
  def size() -> int //

  # Ist die Schlange leer?
  def isEmpty() -> bool

}
```

## Warteschlange (queue)

- Reihenfolge: First in - first out.
  - Elemente werden nur von vorne entnommen
  - Elemente werden nur von hinten hinzugefügt.
- Anwendung:  
Zwischenspeicher von Elementen, ohne dass die Reihenfolge verändert wird.



Quelle: Abbildung 1.31, Algorithms, Sedgewick & Wayne

## Stapel (Stack)

```
class Stack[Item] {

  # Element zu Stapel hinzufügen
  def push(item : Item)

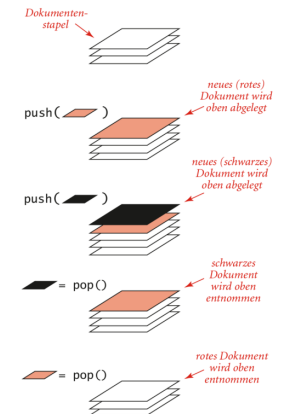
  # Element von Stapel entfernen
  def pop() -> Item // Element entnehmen

  # Ist Stapel leer?
  def isEmpty() -> Boolean

  # Anzahl Element in Stapel
  def size() -> int
}
```

## Stapel (Stack)

- Reihenfolge: last in - first out (LIFO)
  - Jedes element wird oben den Stapel gelegt.
  - Nur oberstes Element kann entfernt werden.
- Anwendung: Stapeln und Schachtelung von Dingen
  - Verschachtelte Funktionen / arithmetische Ausdrücke
  - E-Mail organisation
  - Browser history (back button)



Quelle: Abbildung 1.32, Algorithmen, Sedgewick & Wayne

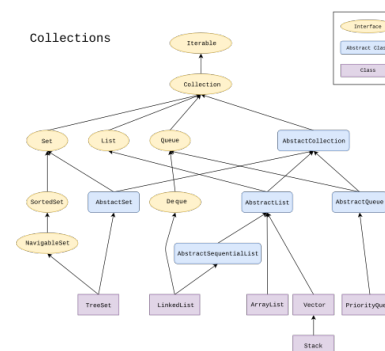
## Multimengen, Warteschlangen und Stapel

- Nichts Neues: Nur Listen mit eingeschränkter Funktionalität
- In Python: Alle Operationen definiert im Datentype List  
Siehe: <https://docs.python.org/3.1/tutorial/datastructures.html>

Einschränkungen helfen Intention und Nutzung klar zu machen und Fehler in Nutzung zu verhindern.

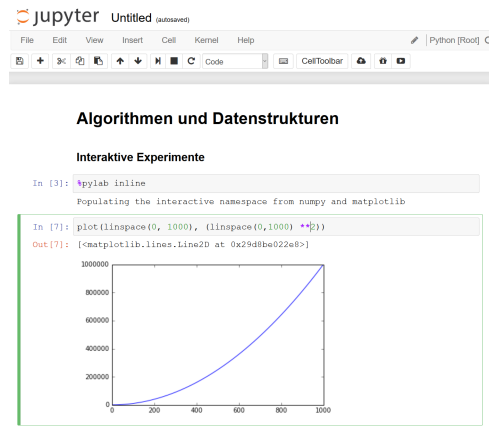
## ADTs in Bibliotheken (Java)

- ADTs sind heute Teil jeder Standardbibliothek



Quelle: By RamImn - Own work, CC BY-SA 4.0,  
<https://commons.wikimedia.org/w/index.php?curid=64043967>

## Beispiele und Implementation



IPython Notebooks: fundamental-adts.ipynb

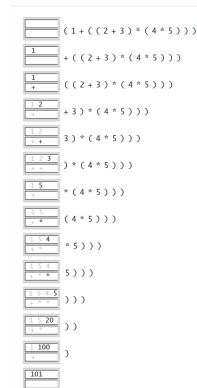
## B3.3 Anwendung von Stacks

## Auswerten arithmetischer Operationen

Beispiel:  $(1 + ((2 + 3) * (4 * 5)))$

### Two-Stack Algorithmus (Dijkstra)

- ▶ Wert: **push** auf Wertestapel
- ▶ Operator: **push** auf Operatorenstapel
- ▶ Linke Klammer: Ignorieren
- ▶ Rechte Klammer: **pop** Operator und zwei Werte
  - ▶ Operation auf Werte anwenden
  - ▶ **push** Resultat der Operation auf Wertestapel



Quelle: <https://algs4.cs.princeton.edu/lectures/13StacksAndQueues-2x2.pdf>

## Warum funktioniert das?

### Beobachtung:

- ▶ Nach Auswertung eines geklammerten Ausdrucks ist der Stack im selben Zustand wie wenn der Wert anstelle des Ausdrucks gestanden hätte.
  - ▶  $(1 + ((2 + 3) * (4 * 5)))$  wird zu  $(1 + (5 * (4 * 5)))$
  - ▶  $(1 + (5 * (4 * 5)))$  wird zu  $(1 + (5 * 20))$
  - ▶  $(1 + (5 * 20))$  wird zu  $(1 + 100)$
  - ▶  $(1 + 100)$  wird zu **101**

## B3.4 Priority Queues

## Vorrangwarteschlangen (Priority Queue)

### Anwendung:

- ▶ Grösste Elemente müssen verarbeitet werden. Nicht alle auf einmal.

### Beispiele:

- ▶ Job-Scheduling (Elemente: Prioritäten von Prozessen)
- ▶ Numerische Berechnung: (Elemente: Berechnungsfehler, die zuerst zu beheben sind)
- ▶ Simulationssysteme (Elemente (Schlüssel): Ereigniszeiten)

## Priority Queue ADT

```
class MaxPQ[Item]:
    # Element einfügen
    def insert(k : Item) -> None

    # Grösstes Element zurueckgeben
    def max() -> Item

    # Grösstes Element entfernen und zurueckgeben
    def delMax() -> Item

    # Ist die Queue leer?
    def isEmpty() -> bool

    # Anzahl Elemente in der Priority Queue
    def size() -> int
```

## Einfache Implementationen

### Arrayrepräsentation (ungeordnet)

- ▶ Insert: Schlüssel zu Array hinzufügen
- ▶ max: Suche grössten Schlüssel
  - ▶ - Swap mit letztem Element
  - ▶ - Siehe: Selection sort

### Arrayrepräsentation (geordnet)

- ▶ Insert: Schlüssel an richtiger Stelle im Array hinzufügen
  - ▶ - Siehe: Insertion sort
- ▶ max: Letztes Element in Array zurückgeben.

Datenstruktur	Einfügen	Grösstes Element entfernen
Ungeordnetes Array	1	$N$
Geordnetes Array	$N$	1

## Beispielclient

**Gegeben:** Sehr grosser Stream von  $N$  Elementen  $N$  so gross, dass Speichern nicht möglich ist.

**Gesucht:**  $M$  grösste Elemente.

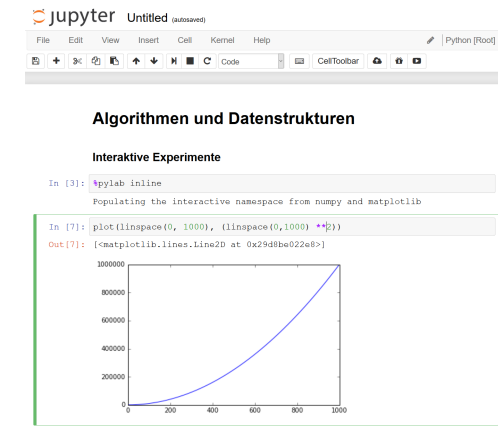
### Einfachste Implementierungen (Nicht praktikabel)

- ▶ Daten werden in Array gespeichert
- ▶ Daten werden sortiert und  $M$  grösste Elemente zurückgegeben

### Bessere Idee

Halte  $M$  grösste Elemente in Priority Queue.

## Implementation



IPython Notebooks: PQ.ipynb

## Komplexität Beispielclient

Implementation	Zeit	Speicher
Sortier-Client	$N \log N$	$N$
PQ (einfache Implementation)	$NM$	$M$

- ▶ Grosse Vorteile in Laufzeit und Speicherkomplexität wenn  $M \ll N$

## Ausblick: Heaps - Ideale Datenstruktur für Priority Queues

### Datenstruktur

Datenstruktur	Einfügen	Grösstes Element entfernen
Geordnetes Array	$N$	1
Ungeordnetes Array	1	$N$
Heap	$\log N$	$\log N$

### Testclient

Implementation	Zeit	Speicher
Sortier-Client	$N \log N$	$N$
PQ (einfache Implementation)	$NM$	$M$
Heap Implementation	$N \log M$	$M$