Datenstrukturen und Algorithmen

Peter Rossmanith

Theoretische Informatik, RWTH Aachen

3. Juni 2024

RWTH AACHEN UNIVERSITY

Organisatorisches

Vorlesung:
Peter Rossmanith

Sprechstunde: Mittwochs 10:00–11:00, Raum 4104 Infozentrum

Übung:
Daniel Mock, Daniel Cloerkes, Matthias Gehnen, Nils Speetzen

`dsal@lists.rwth-aachen.de`

Nur diese E-Mailadresse!!!!

Sprechen Sie mit uns persönlich und verwenden Sie E-Mail nur wenn es nicht anders geht!

Organisatorisches

Vorlesung: Donnerstags, 10:30–12:00 Uhr, Audimax
Freitags, 10:30–12:00 Uhr, H01

Globalübung: Mittwochs, 12:30–14:00, H03
Beginn in der dritten Woche

Tutorübungen: (ab 15. April)

- Montags, 8:30–10:00
- Montags, 14:30–16:00
- Montags, 16:30–18:00

Anmeldung zu Übungsgruppen in RWTH-Online
Bis heute abend!

Ausgabe des Übungsblatts über Moodle

Alle Materialen im Moodle-Lernraum

Organisatorisches

Klausuren:

31.7.2024 (1. Klausur)
2.9.2024 (2. Klausur)

Anmeldung und Abmeldung: über RWTH-Online

Zulassungskriterien:

50% der Punkte aus den Hausaufgaben

Verbesserung der Klausurnote durch Mini-Tests in den Tutorübungen (bis zu zwei Notenstufen, leicht zu schaffen)

Erstellung der Hausaufgaben in Gruppen von vier Personen
und Abgabe über Moodle.

Übersicht

Übersicht

Einordnung

Etwas Vorwissen kann bei Algorithmen und Datenstrukturen helfen. Diese Vorlesungen sind nützlich:

- Programmierung
- Diskrete Strukturen
- Analysis
- Stochastik

Einordnung

Vorlesungen, die Datenstrukturen und Algorithmen ergänzen:

- Berechenbarkeit und Komplexität
- Formale Systeme, Automaten, Prozesse
- Numerisches Rechnen
- Effiziente Algorithmen

Übersicht

Drei sehr umfassende Bücher


T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein.
Introduction to Algorithms.
The MIT Press, 2d edition, 2001.
https://www.degruyter.com/document/doi/10.1515/9783110522013/html


T. Ottmann and P. Widmayer.
Algorithmen und Datenstrukturen.
Spektrum Verlag, 2002.


K. Mehlhorn and P. Sanders.
Algorithms and Data Structures: The Basic Toolbox.
Springer, 2008.

Weitere interessante Bücher


A. V. Aho, J. E. Hopcroft, and J. D. Ullman.
The Design and Analysis of Computer Algorithms.
Addison-Wesley, 1974.


S. Skiena.
The Algorithm Design Manual.
Telos, 1997.

Bücher für die Analyse von Algorithmen


R. L. Graham, D. E. Knuth, and O. Patashnik.
Concrete Mathematics.
Addison-Wesley, 1989.


R. Sedgewick and P. Flajolet.
An Introduction to the Analysis of Algorithms.
Addison-Weseley Publishing Company, 1996.

Bücher für die Analyse von Algorithmen


A. Steger.
Diskrete Strukturen I. Kombinatorik, Graphentheorie, Algebra.
Springer-Verlag, 2001.


T. Schickinger und A. Steger.
Diskrete Strukturen II. Wahrscheinlichkeitsrechnung und Statistik.
Springer-Verlag, 2001.

Speziellere Bücher


Warren S. H.
Hacker's Delight.
Addison-Wesley Longman, 2002.


J. L. Hennessy and D. A. Patterson.
Computer Architecture: A Quantitative Approach.
The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufman Publishers, 3rd edition, 1990.

Die Klassiker


D. E. Knuth.
Seminumerical Algorithms, volume 2 of *The Art of Computer Programming*.
Addison-Wesley, 2nd edition, 1969.


D. E. Knuth.
Fundamental Algorithms, volume 1 of *The Art of Computer Programming*.
Addison-Wesley, 2d edition, 1973.


D. E. Knuth.
Sorting and Searching, volume 3 of *The Art of Computer Programming*.
Addison-Wesley, 1973.

Übersicht

Komplexität von Algorithmen

Eine zentrale Frage:

Zwei verschiedene Algorithmen lösen dasselbe Problem.
Welcher ist schneller?

Unterscheide:

1. Die Laufzeit eines konkreten Algorithmus
2. Die Geschwindigkeit mit der sich ein Problem lösen läßt
   $\rightarrow$ Komplexitätstheorie

Komplexität von Algorithmen

Gegeben: Algorithmus A und Algorithmus B

Wir wählen eine bestimmte Eingabe.

Ergebnis:

- Algorithmus A benötigt 10 Sekunden
- Algorithmus B benötigt 15 Sekunden

Ist Algorithmus A schneller? Er ist auf dieser Eingabe schneller.

Es gibt aber viele andere mögliche Eingaben.

Worst-Case–Komplexität von Algorithmen

Lösung:

Die Laufzeit eines Algorithmus ist nicht eine Zahl, sondern eine Funktion $\mathbf{N} \to \mathbf{N}$.

Sie gibt die Laufzeit des Algorithmus für jede Eingabelänge an.

Die Laufzeit für Eingabelänge n ist die schlechteste Laufzeit aus allen Eingaben mit Länge n.

Wir nennen dies die Worst-Case–Laufzeit.

Sequentielle Berechenbarkeitshypothese

Church'sche These:

> *Die Menge der algorithmisch lösbaren Probleme ist für alle vernünftigen Berechnungsmodelle identisch.*

→ Vorlesung Berechenbarkeit und Komplexität

„Theorem“

Sequentielle Berechenbarkeitshypothese:

Die benötigte Laufzeit für ein Problem auf zwei unterschiedlichen sequentiellen Berechenbarkeitsmodellen unterscheidet sich nur um ein Polynom.

Ein beliebiges Polynom ist für uns zu grob!

Die RAM (Random Access Machine)

Für die Laufzeitanalyse legen wir uns auf ein Modell fest:
Die Random Access Machine (RAM).

- Einem normalen von Neumann–Computer ähnlich
- Ein sehr einfaches Modell
- Algorithmus auf RAM und Computer: Anzahl der Anweisungen unterscheidet sich nur um konstanten Faktor

Die RAM (Random Access Machine)

Eine RAM besteht aus:

1. Einem Speicher aus Speicherzellen 1, 2, 3,...
2. Jede Speicherzelle kann beliebige Zahlen speichern
3. Der Inhalt einer Speicherzelle kann als Anweisung interpretiert werden
4. Einem Prozessor der einfache Anweisungen ausführt
5. Anweisungen: Logische und Arithmetische Verknüpfungen, (bedingte) Sprünge
6. Jede Anweisung benötigt konstante Zeit

Grenzen des RAM-Modells

- Jede Speicherzelle enthält beliebige Zahlen:
  Unrealistisch, wenn diese sehr groß werden
- Jede Anweisung benötigt gleiche Zeit:
  Unrealistisch, wegen Caches, Sprüngen

Daumenregel: Bei n Speicherzellen, $O(\log n)$ bits verwenden.

```
void quicksort(int N)
{
    int i,j,l,r,k,t;
    l=1; r=N;
    if(N>M)
      while(1) {
        i=l-1; j=r; k=a[j];
        do {
            do { i++; } while(a[i]<k);
            do { j--; } while(k<a[j]);
            t=a[i]; a[i]=a[j]; a[j]=t;
        } while(i<j);
        a[j]=a[i]; a[i]=a[r]; a[r]=t;
        if(r-i ≥ i-l) {
            if(i-l>M) { push(i+1,r); r=i-1; }
            else if(r-i>M) l=i+1;
            else if(stack_is_empty) break;
            else pop(l,r);
        }
        else
            if(r-i>M) { push(l,i-1); l=i+1; }
            else if(i-l>M) r=i-1;
            else if(stack_is_empty) break;
            else pop(l,r);
      }
    for(i=2;i ≤ N;i++)
        if(a[i-1]>a[i]) {
            k=a[i]; j=i;
            do { a[j]=a[j-1]; j--; } while(a[j-1]>k);
            a[j]=k;
        }
}
```

RWTH AACHEN UNIVERSITY

Quicksort – Ein Beispiel

Statt einer Worst-Case–Analyse führen wir eine Average-Case–Analyse durch.

Die Eingabe besteht aus den Zahlen $1, \ldots, n$ in zufälliger Reihenfolge.

Wie lange benötigt Quicksort im Durchschnitt zum Sortieren?

Es sind $O(n \log n)$ Schritte auf einer RAM.

Für eine genauere Analyse, müssen wir das Modell genau festlegen.

```
sw -4(r29),r30
add r30,r0,r29
sw -8(r29),r31
subui r29,r29,#464
sw 0(r29),r2
sw 4(r29),r3
sw 8(r29),r4
sw 12(r29),r5
...
sw 40(r29),r12
lw r11,(r30)
lw r9,4(r30)
slli r1,r11,#0x2
lhi r8,((_a+4)>>16)&0xffff
addui r8,r8,(_a+4)&0xffff
lhi r12,((_a)>>16)&0xffff
addui r12,r12,(_a)&0xffff
add r7,r1,r12
sgt r1,r11,r9
beqz r1,L8
addi r4,r30,#-408
add r10,r0,r4
L11:
addi r3,r8,#-4
L51:
add r2,r0,r7
lw r5,(r7)
L15:
addi r3,r3,#4
L49:
lw r1,(r3)
slt r1,r1,r5
bnez r1,L49
addi r3,r3,#4
addi r3,r3,#-4
addi r2,r2,#-4
L50:
lw r31,(r2)
slt r1,r5,r31
bnez r1,L50
addi r2,r2,#-4
addi r2,r2,#4
lw r6,(r3)
sw (r3),r31
sltu r1,r3,r2
bnez r1,L15
sw (r2),r6
lw r12,(r3)
sw (r2),r12
lw r12,(r7)
sw (r3),r12
sub r1,r7,r3
srai r5,r1,#0x2
sub r1,r3,r8
srai r2,r1,#0x2
sge r1,r5,r2
beqz r1,L24
sw (r7),r6
sgt r1,r2,r9
beqz r1,L25
addi r1,r3,#4
sw (r4),r1
addi r4,r4,#4
sw (r4),r7
addi r4,r4,#4
j L11
addi r7,r3,#-4
L25:
sgt r1,r5,r9
beqz r1,L34
addi r8,r3,#4
j L51
addi r3,r8,#-4
L24:
sgt r1,r5,r9
beqz r1,L32
addi r1,r3,#-4
sw (r4),r8
addi r4,r4,#4
sw (r4),r1
addi r4,r4,#4
j L11
addi r8,r3,#4
L32:
sgt r1,r2,r9
bnez r1,L11
addi r7,r3,#-4
L34:
seq r1,r4,r10
bnez r1,L8
addi r4,r4,#-4
lw r7,(r4)
addi r4,r4,#-4
j L11
lw r8,(r4)
L8:
slli r1,r11,#0x2
lhi r2,((_a)>>16)&0xffff
addui r2,r2,(_a)&0xffff
add r7,r1,r2
addi r3,r2,#8
sleu r1,r3,r7
beqz r1,L40
addi r4,r2,#4
L42:
lw r1,(r4)
lw r2,(r3)
sgt r1,r1,r2
beqz r1,L41
add r5,r0,r2
add r2,r0,r3
addi r31,r3,#-4
L44:
lw r12,(r31)
sw (r2),r12
addi r31,r31,#-4
lw r1,(r31)
sgt r1,r1,r5
bnez r1,L44
addi r2,r2,#-4
sw (r2),r5
L41:
addi r3,r3,#4
sleu r1,r3,r7
bnez r1,L42
addi r4,r4,#4
L40:
lw r2,0(r29)
lw r3,4(r29)
lw r4,8(r29)
...
lw r12,40(r29)
lw r31,-8(r30)
add r29,r0,r30
jr r31
lw r30,-4(r30)
```

Quicksort auf dem DLX-Prozessor

Die Laufzeit im Durchschnitt beträgt

$$10A_N + 4B_N + 2C_N + 3D_N + 3E_N + 2S_N + 3N + 6,$$

Schritte, wobei

$$\begin{aligned}
A_N &= \frac{2N - M}{M + 2} \\
B_N &= \frac{1}{6}(N+1)\left(2H_{N+1} - 2H_{M+2} + 1 - \frac{6}{M+2}\right) + \frac{1}{2} \\
C_N &= N + 1 + 2(N+1)(H_{N+1} - H_{M+2}) \\
D_N &= (N+1)(1 - 2H_{M+1}/(M+2)) \\
E_N &= \frac{1}{6}(N+1)M(M-1)/(M+2) \\
S_N &= (N+1)/(2M+3) - 1.
\end{aligned}$$

und $H_n = 1 + 1/2 + 1/3 + \cdots + 1/n$.

RWTH AACHEN UNIVERSITY

O-Notation

Definition

$$O(f) = \left\{ g \colon \mathbf{N} \to \mathbf{R} \;\middle|\; \exists c \in \mathbf{R}^+ \, \exists N \in \mathbf{N} \, \forall n \geq N \colon |g(n)| \leq c|f(n)| \right\}$$
$$\Omega(f) = \left\{ g \colon \mathbf{N} \to \mathbf{R} \;\middle|\; \exists c \in \mathbf{R}^+ \, \exists N \in \mathbf{N} \, \forall n \geq N \colon |g(n)| \geq c|f(n)| \right\}$$
$$\Theta(f) = \left\{ g \colon \mathbf{N} \to \mathbf{R} \;\middle|\; \exists c_1, c_2 \in \mathbf{R}^+ \, \exists N \in \mathbf{N} \, \forall n \geq N \colon c_1|f(n)| \leq |g(n)| \leq c_2|f(n)| \right\}.$$

Informell:

- $g = O(f) \;\approx\;$ g wächst langsamer als f
- $g = \Omega(f) \;\approx\;$ g wächst schneller als f
- $g = \Theta(f) \;\approx\;$ g wächst so schnell wie f

RWTH AACHEN UNIVERSITY

O-Notation – Alternative Definition

Eine weitere Möglichkeit, $O(f)$, $\Omega(f)$ und $\Theta(f)$ zu definieren, falls $f(n) = 0$ nur für endlich viele n zutrifft.

$$O(f) = \left\{ g : \mathbf{N} \to \mathbf{R} \;\middle|\; \limsup_{n\to\infty} \frac{|g(n)|}{|f(n)|} \text{ existiert} \right\}$$

$$\Omega(f) = \left\{ g : \mathbf{N} \to \mathbf{R} \;\middle|\; \liminf_{n\to\infty} \frac{|g(n)|}{|f(n)|} \neq 0 \right\}$$

$$\Theta(f) = \left\{ g : \mathbf{N} \to \mathbf{R} \;\middle|\; \liminf_{n\to\infty} \frac{|g(n)|}{|f(n)|} \neq 0 \text{ und } \limsup_{n\to\infty} \frac{|g(n)|}{|f(n)|} \text{ existiert} \right\}$$

O-Notation

Das folgende Theorem besagt, daß beide Definitionen für die O-Notation übereinstimmen.

Theorem

Es seien $f, g: \mathbf{N} \to \mathbf{R}$ *zwei Funktionen. Es sei nur endlich oft* $f(n) = 0$.
Dann existiert der Grenzwert $\limsup_{n\to\infty} |g(n)|/|f(n)|$ *genau dann, wenn es Zahlen* $c, N > 0$ *gibt, so daß* $|g(n)| \leq c|f(n)|$ *für alle* $n \geq N$ *gilt.*

Beweis.

„$\Longrightarrow$“
$\limsup_{n\to\infty} |g(n)|/|f(n)| = c$
$\Rightarrow c + \epsilon \geq |g(n)|/|f(n)|$ und $f(n) \neq 0$ bis auf endlich viele Ausnahmen.
$\Rightarrow$ ab einem $N \in \mathbf{N}$, gilt $c + \epsilon \geq |g(n)|/|f(n)|$.
Insbesondere gilt dann auch $|g(n)| \leq (c + \epsilon)|f(n)|$.

„$\Longleftarrow$“ Es gebe nun ein $N > 0$ und ein $c > 0$, so daß $\forall n \geq N$: $|g(n)| \leq c|f(n)|$.
Dann gilt $0 \leq |g(n)|/|f(n)| \leq c$ oder $g(n) = 0$ für alle $n \geq N$.
Es sei $a_n = |g(n)|/|f(n)|$. Diese Folge liegt in $[0, c]$.
Bolzano-Weierstraß $\Rightarrow$ größter Häufungswert.
Dann existiert auch $\limsup_{n\to\infty} |g(n)|/|f(n)|$. □

Theorem

Es seien $f, g\colon \mathbf{N} \to \mathbf{R}$ *zwei Funktionen und* $c \in \mathbf{R}$ *eine Konstante. Dann gilt das folgende:*

1. $O(cf(n)) = O(f(n))$
2. $O(f(n)) + O(g(n)) = O(|f(n)| + |g(n)|)$
3. $O(f(n) + g(n)) = O(f(n)) + O(g(n))$
4. $O(f(n))O(g(n)) = O(f(n)g(n))$
5. $O(f(n)g(n)) = f(n)O(g(n))$
6. $\sum_{k=1}^{n} O(f(k)) = O(nf(n))$ *falls* $|f(n)|$ *monoton steigt*
7. $f(O(1)) = O(1)$

AACHEN UNIVERSITY

Wie beweisen wir eine Gleichung der Form

$$O(f(n)) = O(g(n))$$

oder allgemein eine Gleichung, mit O-Notation auf der linken *und* rechten Seite?

In Wirklichkeit ist es $O(f(n)) \subseteq O(g(n))$.

Wir beweisen:
Für jedes $\hat{f}(n) = O(f(n))$ gilt $\hat{f}(n) = O(g(n))$.

Wir beweisen $O(f(n)) + O(g(n)) = O(|f(n)| + |g(n)|)$:

Beweis.

Sei $\hat{f}(n) = O(f(n))$ und $\hat{g}(n) = O(g(n))$ beliebig.
Dann gilt

$$|\hat{f}(n)| \leq c|f(n)| \text{ und } |\hat{g}(n)| \leq c|g(n)|$$

für ein c und große n.

$$\Rightarrow |\hat{f}(n)| + |\hat{g}(n)| \leq c(|f(n)| + |g(n)|)$$

Daraus folgt $O(|f(n)|) + O(|g(n)|) = O(|f(n)| + |g(n)|)$.
Es gilt aber $O(f(n)) = O(|f(n)|)$ und $O(g(n)) = O(|g(n)|)$. □

Schätze $\log(n^2 + 3n + 5)$ ab.

Theorem

$f : \mathbf{N} \to \mathbf{R}$ *und* $g : \mathbf{R} \to \mathbf{R}$.

$\lim_{n\to\infty} f(n) = 0$.

$g : \mathbf{R} \to \mathbf{R}$ *in einer Umgebung des Ursprungs* k *mal stetig differenzierbar.*

Dann gilt

$$g(f(n)) = \sum_{i=0}^{k-1} g^{(i)}(0)\frac{f(n)^i}{i!} + O(f(n)^k).$$

$$\log(n^2 + 3n + 5) = 2\log(n) + \log(1 + 3/n + 5/n^2) = 2\log(n) + O(1/n)$$

Beweis.

Der Satz von Taylor besagt:

$$g(z) = \sum_{i=0}^{k-1} \frac{g^{(i)}(0)}{i!} z^i + R_{k-1}$$

mit

$$R_{k-1} = \frac{g^{(k)}(\xi)}{k!} z^k \text{ für ein } \xi \text{ mit } |\xi| \leq |z|,$$

falls z nahe bei 0.
Für uns heißt das:

$$g(f(n)) = \sum_{i=0}^{k-1} \frac{g^{(i)}(0)}{i!} (f(n))^i + R_{k-1}$$

bis auf endlich viele n, da $n \to \infty$.
Es bleibt zu zeigen, daß $R_{k-1} = O(f(n)^k)$. □

AACHEN UNIVERSITY

Beweis.

$$R_{k-1} = \frac{g^{(k)}(\xi)}{k!} f(n)^k \text{ für ein } \xi \in [-|f(n)|, |f(n)|],$$

falls $f(n)$ nahe bei 0.

$$\Rightarrow R_{k-1} \leq \max_{\xi \in [-\epsilon, \epsilon]} \frac{g^{(k)}(\xi)}{k!} f(n)^k$$

für ein $\epsilon > 0$ bis auf endlich viele n.

Da $g^{(k)}$ stetig und $[-\epsilon, \epsilon]$ kompakt ist, existiert das Maximum nach dem Satz von Weierstraß.

Also gilt $R_{k-1} = O(f(n)^k)$. □

Beispiele

$$\frac{1}{1+1/n} = 1 + O\Big(\frac{1}{n}\Big)$$
$$\frac{1}{1+1/n} = 1 - \frac{1}{n} + O\Big(\frac{1}{n^2}\Big)$$
$$\frac{1}{1+1/n} = 1 - \frac{1}{n} + \frac{1}{n^2} + O\Big(\frac{1}{n^3}\Big)$$

$$\sqrt{n+1} = \sqrt{n} \cdot \sqrt{1+1/n} = \sqrt{n}(1 + O(1/n)) = \sqrt{n} + O(1/\sqrt{n})$$

Java

Wir verwenden oft Java für Datenstrukturen und Algorithmen.

Die Vorlesung ist aber von der Programmiersprache unabhängig.

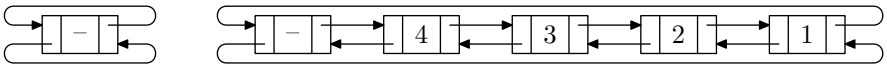
Lernziel sind die einzelnen Algorithmen und Datenstrukturen, nicht ihre Umsetzung in Java.

Für das Verständnis der Vorlesung sind Kenntnisse in einer objektorientierten Sprache notwendig.

Übersicht

Doppelt verkettete Listen

- Zusätzlicher Listenkopf
- Daten in Knoten gespeichert
- Zeiger zum Vorgänger und Nachfolger
- Zyklisch geschlossen

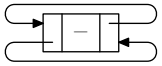


Geeignet für kleine assoziative Arrays.

Für Adjazenzlistendarstellung sehr geeignet.

RWTH AACHEN UNIVERSITY

Der Konstruktor muß den Listenkopf head erzeugen.
Der Vorgänger und Nachfolger von head ist head selbst.



Java

```
public class ADList⟨K, D⟩ extends AbstractMap⟨K, D⟩ {
  protected Listnode⟨K, D⟩ head;
  public ADList() {
    head = new Listnode⟨K, D⟩(null, null);
    head.pred = head;
    head.succ = head;
  }
```

Java

```
public class Listnode⟨K, D⟩ {
    K key;
    D data;
    Listnode⟨K, D⟩ pred, succ;
    Listnode(K k, D d) {
        key = k; data = d; pred = null; succ = null; }
    void delete() {
        pred.succ = succ; succ.pred = pred; }
    void copy(Listnode⟨K, D⟩ n) {
        key = n.key; data = n.data; }
    void append(Listnode⟨K, D⟩ newnode) {
        newnode.succ = succ; newnode.pred = this;
        succ.pred = newnode; succ = newnode; }
```

Programm in Python

```
class Node :
    def __init__(self, key, value) :
        self.key = key
        self.value = value
        self.pred = self
        self.succ = self
```

Programm in Python

```
    def insert(self, x) :
        self.succ.pred = x
        x.succ = self.succ
        self.succ = x
        x.pred = self
```

Programm in Python

```
def remove(self) :
    self.pred.succ = self.succ
    self.succ.pred = self.pred
```

Programm in Python

```
def allnodes(self) :
    node = self.succ
    while node.key :
        yield node
        node = node.succ
```

Java

```
public void append(K k, D d) {
    head.pred.append(new Listnode⟨K, D⟩(k, d));
}
```

Java

```
public void prepend(K k, D d) {
    head.append(new Listnode⟨K, D⟩(k, d));
}
```

Java

```
public void delete(K k) {
  Listnode⟨K, D⟩ n = findnode(k);
  if(n ≠ null) n.delete();
}
```

Java

```
public D find(K k) {
    Listnode⟨K, D⟩ n = findnode(k);
    if(n == null) return null;
    return n.data;
}
```

Java

```
protected Listnode⟨K, D⟩ findnode(K k) {
    Listnode⟨K, D⟩ n;
    head.key = k;
    for(n = head.succ; !n.key.equals(k); n = n.succ) { }
    head.key = null;
    if(n == head) return null;
    return n;
}
```

AACHEN VERSITY

Einfach verkettete Listen

- Kein Zeiger auf Vorgänger
- Einfachere Datenstruktur
- Operationen können komplizierter sein

Frage: Wie kann ein Element *vor* einen gegebenen Knoten eingefügt werden?

Ersetzen und alten Knoten anfügen!

Vorsicht: Eine gefährliche Technik.

Listen – Laufzeit

Manche Operation sind schnell, manche langsam...

Operation	Laufzeit
append()	$\Theta(1)$
delete()*	$\Theta(1)$
delete()	$\Theta(n)$
find()	$\Theta(n)$
insert()	$\Theta(n)$

* direktes Löschen eines Knotens, nicht über Schlüssel

Java

```
public class Stack⟨D⟩ {
  private ADList⟨Object, D⟩ stack;
  private int size;
  public Stack() { stack = new ADList⟨Object, D⟩(); size = 0; }
  public boolean isempty() { return size == 0; }
  public D pop() {
    D x = stack.firstnode().getData();
    stack.firstnode().delete();
    size−−;
    return x;
  }
  public void push(D x) { stack.prepend(null, x); size++; }
  public int size() { return size; }
}
```

AACHEN UNIVERSITY

Java

```
public class Queue⟨D⟩ {
  private ADList⟨Object, D⟩ queue;
  private int size;
  public Queue() { queue = new ADList⟨Object, D⟩(); size = 0; }
  public boolean isempty() { return size == 0; }
  public D dequeue() {
    D x = queue.lastnode().getData();
    queue.lastnode().delete();
    size−−;
    return x;
  }
  public void enqueue(D x) { queue.prepend(null, x); size++; }
  public int size() { return size; }
}
```

AACHEN UNIVERSITY