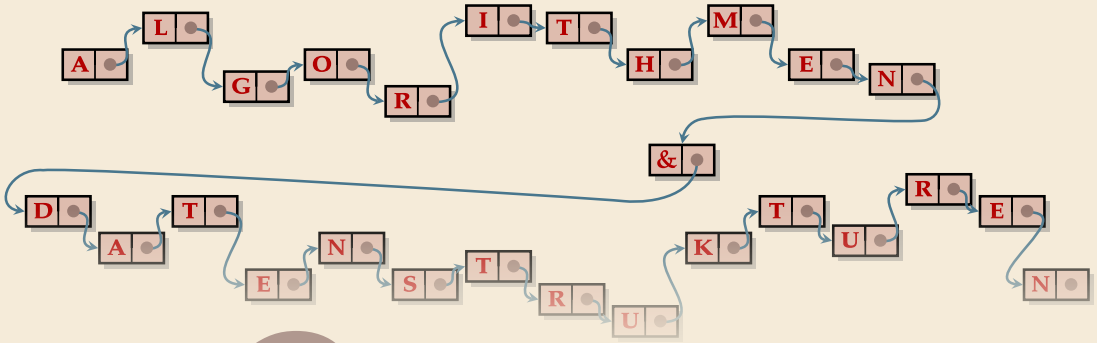


2 Programmierumgebung

Algorithmen & Datenstrukturen · Sommersemester 2026

Prof. Dr. Sebastian Wild

CS210 (Sommersemester 2026)
Philipps-Universität Marburg
version `2026-04-20 09:31 H`

Outline

2 Programmierumgebung

Unser Spagat

- ADS ist halb Theorie, halb Praxis
- einerseits lernen Sie asymptotische Analysen und abstrakte Modelle
  - ohne dass wir ein Mathematik-Modul daraus machen wollen
- andererseits werden Sie die Toolbox der Algorithmik auch *einsetzen*
  - aber Syntax und Details von Java sollen nicht im Vordergrund stehen

$\rightsquigarrow$ an manchen Stellen erlauben wir uns Abkürzungen (an anderen absichtlich *nicht*)

2.1 Standard Library

Vokabular einer Programmiersprache

Programmiersprachen als Fremdsprachen

- eine Programmiersprache definiert die “Grammatikregeln” einer Sprache
- eine interessante Unterhaltungen braucht aber auch reichhaltiges Vokabular und idiomatische Redewendungen!

Programmierbibliotheken

- In Programmiersprachen liefern das Bibliotheken
- in der Regel (hauptsächlich) in der Sprache selbst geschrieben
- kapselt Interaktion mit Betriebssystem

Standardbibliothek

- Teil des “Lieferumfangs”, immer verfügbar
- Beispiele: Java Runtime Library, C++ Standard Template Library, Python Standard Library

Java Runtime Library

- alle Klassen in `java.lang` und `java.util` (und weitere packages)
- z. B. auch `System.out.println()` etc.
- insgesamt über 4400 Klassen und Interfaces
  (Java 21, `https://docs.oracle.com/en/java/javase/21/docs/api/allclasses-index.html`)
  - kann etwas überwältigend sein . . .
  - *niemand* kennt alle diese Klassen auswendig!
  - aber: gute Programmierer:innen bekommen ein Gespür dafür, für welche Funktionalität es eine Library-Lösung geben *müsste*

  ⇝ um diese dann zu finden und korrekt zu verwenden gibt es Werkzeuge

Hier müssen Sie von Fall zu Fall entscheiden:

- von KI machen lassen
  - ⇝ beim nächsten Mal wieder Hilfe nötig
  - ⇝ OK für Aufgaben, die nur einmal kommen und einfach sind
- lernen, wie die Bibliothek funktioniert
  - beim nächsten Mal schneller
  - ⇝ Vokabular erweitert

2.2 Algs4

Algs4

"Algs4" = Sedgewick & Wayne: Algorithms 4th edition

"`algs4`" = Java-Bibliothek mit Hilfsklassen

`https://github.com/kevin-wayne/algs4`

Bequemlichkeit

- Standardbibliothek per definitionem *one-size-fits-all*
  - ⇝ keine Rücksicht auf Anfänger:innen . . .

- außerdem: backwards compatibility oft zwingend

⇝ Designfehler schwer zu beheben

Beispiele:

- `Date` ist mutable

  `https://docs.oracle.com/javase/tutorial/datetime/iso/legacy.html`

- Methoden in `Vector`, `Hashtable`, `Stack` sind `synchronized`

⇝ Für einige nützliche Funktionen verwenden wir die vereinfachten Wrapper von Sedgewick & Wayne

- verfübar unter `https://algs4.cs.princeton.edu/code/` sowie auf GitHub
- kompakter und lesbarer als direkt via Runtime Library
- grundsätzlich für Programmieraufgaben erlaubt*

UNIX input/output stream model

Abstraktion von Ein- und Ausgabe: Streams

- i.W. unverändert in Benutzung seit 1970er Jahre in UNIX
- flexibel und einfach
- in Java Runtime Library
  `System.in` und `System.out`

≠ command-line arguments

Design„fehler“

- `System.in` hat keine bequemen Methoden um primitive Daten einzulesen

⇝ `new java.util.Scanner(System.in)`

- `System.out` ist langsam für große Ausgaben (automatisches Flush bei `'\n'`)

⇝ `new java.io.PrintStream(new java.io.BufferedOutputStream(System.out))`

- beide sind byte-basiert, nicht Zeichen-basiert

Insbesondere `System.in` ist unbequem . . .

StdIn

edu.princeton.cs.algs4.StdIn aus algs4.jar ist ein bequemerer Wrapper für System.in

public class StdIn

methods for reading individual tokens from standard input		
boolean	isEmpty()	*is standard input empty (or only whitespace)?*
int	readInt()	*read a token, convert it to an* int*, and return it*
double	readDouble()	*read a token, convert it to a* double*, and return it*
boolean	readBoolean()	*read a token, convert it to a* boolean*, and return it*
String	readString()	*read a token and return it as a* String
methods for reading characters from standard input		
boolean	hasNextChar()	*does standard input have any remaining characters?*
char	readChar()	*read a character from standard input and return it*
methods for reading lines from standard input		
boolean	hasNextLine()	*does standard input have a next line?*
String	readLine()	*read the rest of the line and return it as a* String
methods for reading the rest of standard input		
int[]	readAllInts()	*read all remaining tokens and return them as an* int *array*
double[]	readAllDoubles()	*read all remaining tokens and return them as a* double *array*
boolean[]	readAllBooleans()	*read all remaining tokens and return them as a* boolean *array*
String[]	readAllStrings()	*read all remaining tokens and return them as a* String *array*
String[]	readAllLines()	*read all remaining lines and return them as a* String *array*
String	readAll()	*read the rest of the input and return it as a* String

- StdIn besteht aus static methods
- edu.princeton.cs.algs4.In ist eine Objekt-orientierte Variante
  - gleiche Methoden
  - Konstrukturen für Dateien, URLs, . . .

https://introcs.cs.princeton.edu/java/15inout/

StdOut

- edu.princeton.cs.algs4.`StdOut`
  - ähnliche Methoden wie `System.out`
    spart ein paar Zeichne beim Tippen 😉
- edu.princeton.cs.algs4.`Out`
  - wie `In` Objekt-orientierte Version
  - Konstrukturen für Dateien, Sockets, Streams

Beispiel

```java
import edu.princeton.cs.algs4.*; // algs4 Klassen

class Average {
    public static void main(String[] args) {
        double sum = 0.0;
        int n = 0;
        while (!StdIn.isEmpty()) {
            double x = StdIn.readDouble();
            sum += x;
            n++;
        }
        StdOut.println(sum/n);
    }
}
```

`algs4.jar` einbinden

Kompilieren: `javac -cp algs4.jar Average.java`

Ausführen: `java -cp algs4.jar:. Average`

2.3 Malen nach Zahlen

StdDraw

Nur Text ist etwas dröge . . . viele spannende Dinge sind ebenso einfach!

Zum Beispiel einfache **Graphiken**

`public class StdDraw` *(basic drawing commands)*
`void line(double x0, double y0, double x1, double y1)`
`void point(double x, double y)`

viele weitere Methoden ⇝ `https://introcs.cs.princeton.edu/java/15inout/`

`public class StdDraw` *(basic control commands)*	
`void setCanvasSize(int w, int h)`	*create canvas in screen window of width* `w` *and height* `h` *(in pixels)*
`void setXscale(double x0, double x1)`	*reset x-scale to (*`x0`*,* `x1`*)*
`void setYscale(double y0, double y1)`	*reset y-scale to (*`y0`*,* `y1`*)*
`void setPenRadius(double radius)`	*set pen radius to* `radius`

Malen nach Zufallszahlen

Ein einfacher Zufallsprozess:

1. Starte mit 3 Punkten $p_0 = (0, 0)$, $p_1 = (1, 0)$ und $p_2 = (\frac{1}{2}, \sqrt{3/4})$ (gleichseitges Dreieck)
2. Wähle zufälligem Punkt $p \in \{p_0, p_1, p_2\}$
3. Wähle zufälligem Punkt $q \in \{p_0, p_1, p_2\}$
4. Setze p auf den Mittelpunkt zwischen p und q
5. Male Punkt p, zurück zu 3.

```java
import edu.princeton.cs.algs4.*;

class Chaos {
    public static void main(String[] args) {
        int trials = Integer.parseInt(args[0]);
        double    c = Math.sqrt(3.0) / 2.0;
        double[] cx = {0.0, 1.0, 0.5 };
        double[] cy = {0.0, 0.0, c   };
        StdDraw.setPenRadius(0.01);
        double x = 0, y = 0;
        for (int t = 0; t < trials; ++t) {
            int r = (int) (Math.random() * 3);
            x = (x + cx[r]) / 2;
            y = (y + cy[r]) / 2;
            StdDraw.point(x, y); // Punkt malen
        }
    }
}
```

`edu.princeton.cs.algs4.Math` ein weiterer Wrapper,
z. B. einfachere `Math.random()` $\overset{\mathcal{D}}{=}$ Uniform$(0, 1)$

Allgemeines Iterated Function System (IFS)

Das *Sierpinksi-Dreieck* ist ein Beispiel für eine ganze Familie von solchen Prozessen.

- In jedem Schritt aktualisieren wir den aktuellen Punkt $p = (x, y)$ anhand einer linearen Funktion $f : \mathbb{R}^2 \to \mathbb{R}^2$: $(x, y) \mapsto \big(f_x(x, y), f_y(x, y)\big)$
- f in jedem Schritt zufällig aus einer Liste von Funktionen gewählt.

Funktion	Wahrscheinlichkeit	$f_x(x, y)$	$f_y(x, y)$
f_0 (p_0)	$\frac{1}{3}$	$\frac{1}{2}x$	$\frac{1}{2}y$
f_1 (p_1)	$\frac{1}{3}$	$\frac{1}{2}x + \frac{1}{2}$	$\frac{1}{2}y$
f_2 (p_2)	$\frac{1}{3}$	$\frac{1}{2}x + \frac{1}{4}$	$\frac{1}{2}y + \frac{1}{2}\sqrt{3/4}$ ← ≈ 0.433

- Allgemein:

$$x \mapsto (e_R)^T \cdot A \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \qquad y \mapsto (e_R)^T \cdot B \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

e_R zufälliger Einheitsvektor

Sierpinski: $R \overset{\square}{=} \text{Uniform}\{1, 2, 3\}$ $A = \begin{pmatrix} 0.5 & 0 & 0 \\ 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.25 \end{pmatrix}$ $B = \begin{pmatrix} 0 & 0.5 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0.5 & 0.433 \end{pmatrix}$

IFS Implementierung

```java
import edu.princeton.cs.algs4.*;

public class IFS {
    public static void main(String[] args) {
        // number of iterations
        int trials = Integer.parseInt(args[0]);
        // prob. distribution for functions
        double[] dist = StdArrayIO.readDouble1D();
        // update matrices
        double[][] cx = StdArrayIO.readDouble2D();
        double[][] cy = StdArrayIO.readDouble2D();

        // current value of (x, y)
        double x = 0.0, y = 0.0;
        // for faster drawing, delay update to draw()
        StdDraw.enableDoubleBuffering();
        for (int t = 0; t < trials; t++) {
            // random index according to prob. dist.
            int r = StdRandom.discrete(dist);
            // compute new coordinates
            double x0, y0;
            x0 = cx[r][0]*x + cx[r][1]*y + cx[r][2];
            y0 = cy[r][0]*x + cy[r][1]*y + cy[r][2];
            x = x0;
            y = y0;
            // draw the resulting point
            StdDraw.point(x, y);
        }
        // show final picture
        StdDraw.show();
    }
}
```

- verwendet weitere Bequemlichkeits-Wrapper:
  - `edu.princeton.cs.algs4.StdArrayIO` liest Vektoren und Matrizen ein
  - Format: Anzahl (Zeilen und) Spalten gefolgt von Einträgen
  - `edu.princeton.cs.algs4.StdRandom.discrete` zieht diskrete Zufallsvariable gegeben die Wahrscheinlichkeitsgewichte

sierpinsky.txt

```
3
    0.333    0.333    0.334
3 3
    0.5    0    0
    0.5    0    0.5
    0.5    0    0.25
3 3
    0    0.5    0
    0    0.5    0
    0    0.5    0.433
```

Iterated function systems

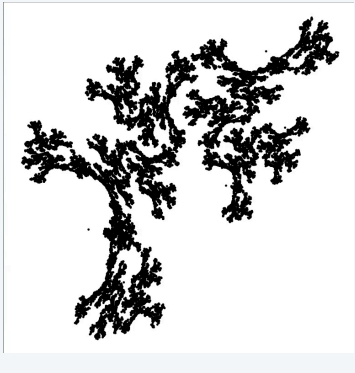
What happens when we change the rules?

probability	*new x*	*new y*
40%	$.31x - .53y + .89$	$-.46x - .29y + 1.10$
15%	$.31x - .08y + .22$	$.15x - .45y + .34$
45%	$.55y + .01$	$.69x - .20y + .38$

IFS.java (Program 2.2.3) is a *data-driven* program that takes the coefficients from *standard input.*

```
% more coral.txt
3
   0.40  0.15  0.45
3 3
   0.307692 -0.531469  0.8863493
   0.307692 -0.076923  0.2166292
   0.000000  0.545455  0.0106363
3 3
  -0.461538 -0.293706  1.0962865
   0.153846 -0.447552  0.3383760
   0.692308 -0.195804  0.3808254
```

```
% java IFS 10000 < coral.txt
```



Iterated function systems

Another example of changing the rules

probability	*new x*	*new y*
2%	0.5	$.27y$
15%	$-.14x + .26y + .57$	$.25x + .22y - .04$
13%	$.17x - .21y + .41$	$.22x + .18y + .09$
70%	$.78x + .03y + .11$	$-.03x + .74y + .27$

```
% more barnsley.txt
4
  .02 .15 .13 .70
4 3
  .000  .000  .500
 -.139  .263  .570
  .170 -.215  .408
  .781  .034  .1075
4 3
  .000  .270  .000
  .246  .224 -.036
  .222  .176  .0893
 -.032  .739  .270
```

```
% java IFS 10000 < barnsley.txt
```

Iterated function systems

Simple iterative computations yield patterns that are remarkably similar to those found in the natural world.

Q. What does computation tell us about nature?

Q. What does nature tell us about computation?

20th century sciences. Formulas.

21st century sciences. Algorithms?

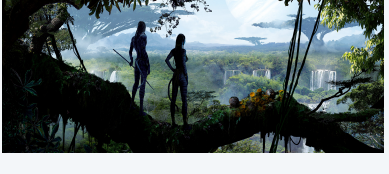
Note. You have seen many practical applications of integrated function systems, in movies and games.

an IFS fern

a real fern

a real plant

an IFS plant



2.4 Mehr zu entdecken

StdAudio

edu.princeton.cs.algs4.StdAudio erlaubt einfache Soundausgabe

public class StdAudio	
int SAMPLE_RATE	*44,100 Hz*
void play(String filename)	*play the given .wav file*
void playInBackground(String filename)	*play the given .wav file in a background thread*
void play(double[] samples)	*play the given samples*
void play(double sample)	*play sample*
void save(String filename, double[] a)	*save to a .wav file*
double[] read(String filename)	*read from a .wav file*

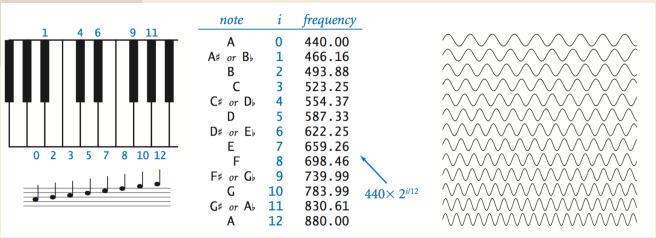
API for our library of static methods for standard audio

Mini-Synthesizer

```java
import edu.princeton.cs.algs4.*;

public class PlayThatTune {
    public static void main(String[] args) {
        int SAMPLING_RATE = 44100;
        while (!StdIn.isEmpty()) {
            // Read and play one note.
            int pitch = StdIn.readInt();
            double duration = StdIn.readDouble();
            double hz = 440 * Math.pow(2, pitch / 12.0);
            int n = (int) (SAMPLING_RATE * duration);
            double[] a = new double[n+1];
            for (int i = 0; i <= n; i++)
                a[i] = Math.sin(2*Math.PI * i * hz
                        / SAMPLING_RATE);
            StdAudio.play(a);
        }
    }
}
```

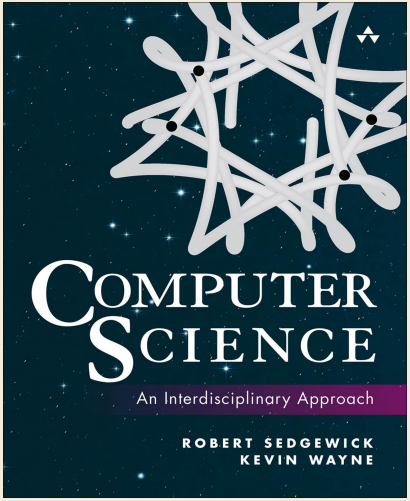
- Spannender Physik-Exkurs:
  Was sind Geräusche und Töne?
- Sinus-Schwingung für reine Töne
  440 Hz $\rightsquigarrow$ Kammerton
  (eingestrichenes A)
- `StdAudio.play(double[])` spielt gegebene Werte als Ton
- Soundkarte erwartet gewisse Sample-Rate (Werte pro Sekunde)



https://introcs.cs.princeton.edu/java/15inout/

Programming with a Purpose

Viele weitere kreative Beispiele und Anwendungen bei Sedgewick & Wayne



Für Algorithmen und Datenstrukturen nur tangential relevant, aber macht Spaß 😎