

Datenstrukturen und Algorithmen

Vorlesung am D-Math (CSE) der ETH Zürich

Felix Friedrich

FS 2017

Willkommen!

Vorlesungshomepage:

<http://lec.inf.ethz.ch/DA/2017>

Das Team:

Assistenten

Alexander Pilz

Daniel Hupp

Lukas Humbel

Dozent

Felix Friedrich

1. Einführung

Algorithmen und Datenstrukturen, drei Beispiele

Ziele der Vorlesung

- Verständnis des Entwurfs und der Analyse grundlegender Algorithmen und Datenstrukturen.
- Vertiefter Einblick in ein modernes Programmiermodell (mit C++).
- Wissen um Chancen, Probleme und Grenzen des parallelen und nebenläufigen Programmierens.

Ziele der Vorlesung

Einerseits

- Unverzichtbares Grundlagenwissen aus der Informatik.

Andererseits

- Vorbereitung für Ihr weiteres Studium und die Praxis.

Inhalte der Vorlesung

Datenstrukturen / Algorithmen

Begriff der Invariante, Kostenmodell, Landau Symbole

Algorithmenentwurf, Induktion

Suchen und Auswahl, Sortieren

Dynamic Programming Graphen, Kürzeste Wege, Backtracking, Flow

Wörterbücher: Hashing und Suchbäume, AVL

Sortiernetzwerke, parallele Algorithmen

Randomisierte Algorithmen (Gibbs/SA), Multiskalen

Geometrische Algorithmen, High Performance LA

Programmieren mit C++

RAII, Move Konstruktion, Smart Pointers, Constexpr, user defined literals

Templates und Generische Programmierung

Exceptions

Funktoren und Lambdas

Promises and Futures

Threads, Mutexs and Monitors

Parallel Programming

Parallelität vs. Concurrency, Speedup (Amdahl/-Gustavson), Races, Memory Reordering, Atomic Registers, RMW (CAS,TAS), Deadlock/Starvation

Literatur

Algorithmen und Datenstrukturen, *T. Ottmann, P. Widmayer*, Spektrum-Verlag, 5. Auflage, 2011

Algorithmen - Eine Einführung, *T. Cormen, C. Leiserson, R. Rivest, C. Stein*, Oldenbourg, 2010

Introduction to Algorithms, *T. Cormen, C. Leiserson, R. Rivest, C. Stein*, 3rd ed., MIT Press, 2009

The C++ Programming Language, *B. Stroustrup*, 4th ed., Addison-Wesley, 2013.

The Art of Multiprocessor Programming, *M. Herlihy, N. Shavit*, Elsevier, 2012.

1.2 Algorithmen

[Cormen et al, Kap. 1; Ottman/Widmayer, Kap. 1.1]

Algorithmus

Algorithmus: wohldefinierte Berechnungsvorschrift, welche aus Eingabedaten (*input*) Ausgabedaten (*output*) berechnet.

Beispielproblem

Input : Eine Folge von n Zahlen (a_1, a_2, \dots, a_n)

Output : Eine Permutation $(a'_1, a'_2, \dots, a'_n)$ der Folge $(a_i)_{1 \leq i \leq n}$, so dass
 $a'_1 \leq a'_2 \leq \dots \leq a'_n$

Mögliche Eingaben

$(1, 7, 3), (15, 13, 12, -0.5), (1) \dots$

Jedes Beispiel erzeugt eine *Probleminstanz*.

Beispiele für Probleme in der Algorithmik

- Routenplanung: Shortest Path
- Kryptographie / Digitale Signaturen
- Stundenpläne / Arbeitspläne: Linear Programming
- DNA Matching: Dynamic Programming
- Fabrikationspipeline: Topologische Sortierung
- Geometrische Probleme, z.B. Konvexe Hülle

Charakteristik

- Extrem grosse Anzahl potentieller Lösungen
- Praktische Anwendung

Datenstrukturen

- Organisation der Daten, zugeschnitten auf die Algorithmen die auf den Daten operieren
- Programme = Algorithmen + Datenstrukturen.

Sehr schwierige Probleme

- NP-vollständige Probleme: Keine bekannte effiziente Lösung (Fehlen einer solchen Lösung ist aber unbewiesen!)
- Beispiel: Travelling Salesman Problem

Ein Traum

- Wären Rechner unendlich schnell und hätten unendlich viel Speicher ...
- ... dann bräuchten wir die Theorie der Algorithmen (nur) für Aussagen über Korrektheit (incl. Terminierung).

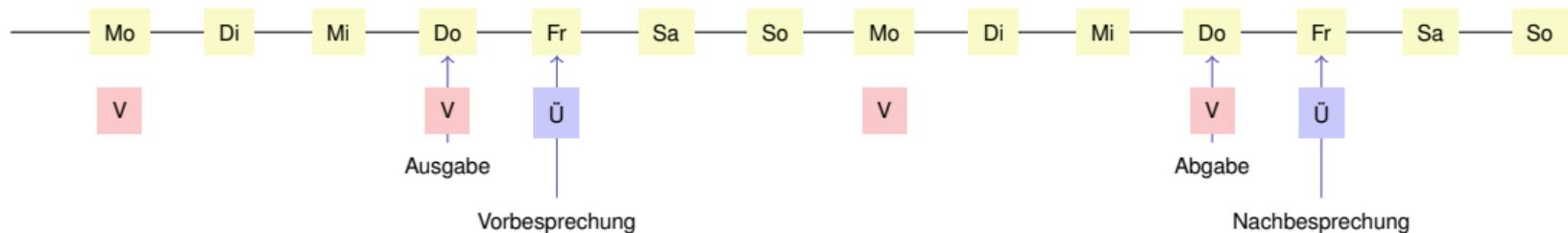
Die Realität

Ressourcen sind beschränkt und nicht umsonst:

- Rechenzeit → Effizienz
- Speicherplatz → Effizienz

1.3 Organisation

Ablauf des Übungsbetriebes



- Übungsblattausgabe zur Vorlesung am Donnerstag (online).
- Vorbesprechung am Freitag.
- Bearbeitung der Übung bis spätestens am Donnerstag darauf.
- Nachbesprechung der Übung am Freitag darauf. Feedback zu den Abgaben innerhalb einer Woche nach Nachbesprechung.

Codeboard

Codeboard ist eine Online-IDE: Programmieren im Browser!

- Sie können Beispiele ausprobieren, ohne dass Sie irgendwelche Tools installieren müssen.
- Auch für die Übungen eingesetzt.

Jetzt mit C++14

```
1 #include <tests.h> // remove slashes at beginning of
2 #include <iostream>
3
4
5
6
7
8
9
10
11 int main ()
12 {
13     Averager avg;
14     int n;
15     std::cin >> n;
16     for (int i = 0; i < n; ++i) {
17         double input;
18         std::cin >> input;
19         avg.add_value(input);
20         std::cout << avg.average_value() << " ";
21     }
22     std::cout << "\n";
23     return 0;
24 }
```

Task 2a: Averager

Not submitted yet

Solve the task and hand in your solution using the green "Submit" button.

Task Description

Write a class **Averager** that computes averages of given values of type **double**. It shall offer at least the members **add_value** (which provides another value of type **double** to the class) and **average_value** (which returns the average of the values provided so far as a **double**).

[based on: Exam Summer 2012, ex.6]

This will display the output.

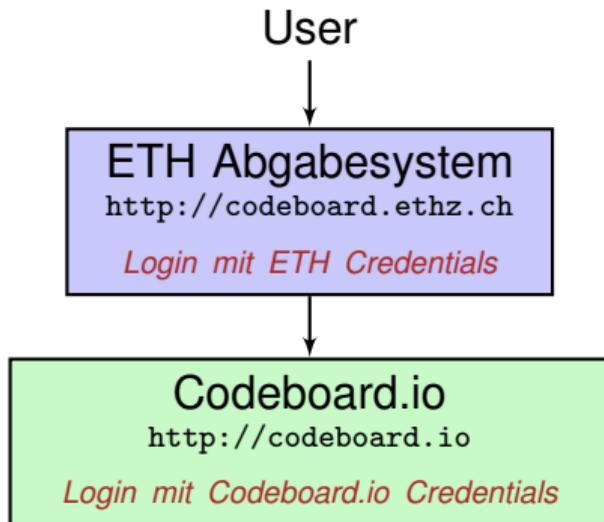
Input to your program (press Enter to send)

User: Anonymous [Sign in](#) to save your progress. Role: Project user. Info: Submissions are forwarded to external platform. [codeboard.io](#)

Codeboard @ETH

Codeboard besteht aus zwei unabhängigen Systemen, die miteinander kommunizieren:

- **Das ETH Abgabesystem:**
Ermöglicht es uns, ihre Aufgaben zu bewerten
- **Die Online IDE:** Die Programmierumgebung



Codeboard

Codeboard.io Registrierung

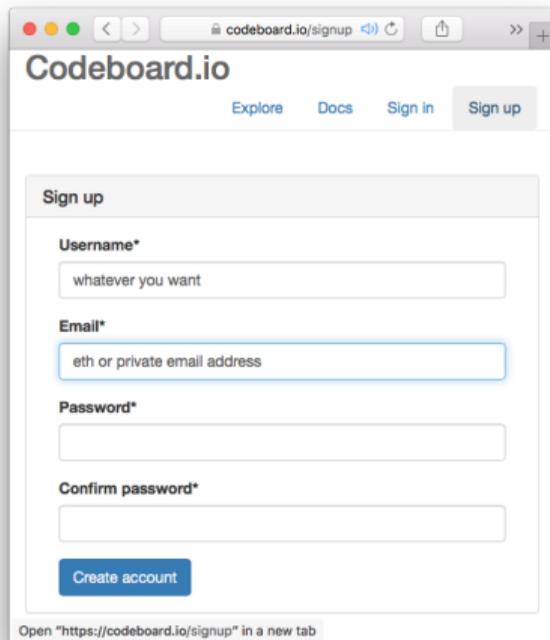
Gehen Sie auf <http://codeboard.io> und erstellen Sie dort ein Konto, bleiben Sie am besten eingeloggt.

Einschreibung in Übungsgruppen

Gehen Sie auf <http://codeboard.ethz.ch/da> und schreiben Sie sich dort in eine Übungsgruppe ein.

Codeboard.io Registrierung

Falls Sie noch keinen **Codeboard.io** Account haben ...



The image shows a browser window with the URL `codeboard.io/signup`. The page title is "Codeboard.io" and it has navigation links for "Explore", "Docs", "Sign in", and "Sign up". The "Sign up" form contains the following fields:

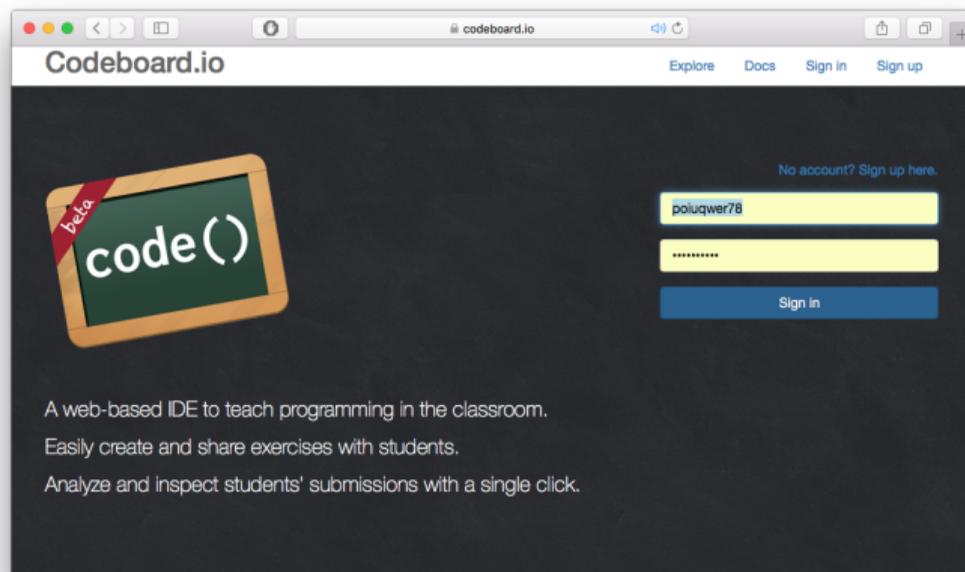
- Username***: A text input field containing the text "whatever you want".
- Email***: A text input field containing the text "eth or private email address".
- Password***: An empty text input field.
- Confirm password***: An empty text input field.

At the bottom of the form is a blue button labeled "Create account". Below the form, a status bar indicates "Open 'https://codeboard.io/signup' in a new tab".

- Wir verwenden die Online IDE **Codeboard.io**
- Erstellen Sie dort einen Account, um Ihren Fortschritt abzuspeichern und später Submissions anzuschauen
- Anmeldedaten können beliebig gewählt werden! *Verwenden Sie nicht das ETH Passwort.*

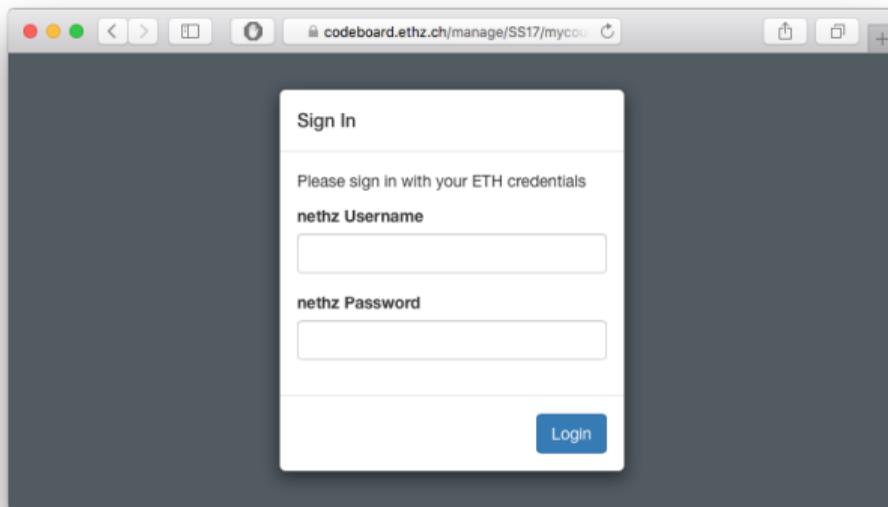
Codeboard.io Login

Falls Sie schon einen Account haben, loggen Sie sich ein:



Einschreibung in Übungsgruppen - I

- Besuchen Sie `http://codeboard.ethz.ch/da`
- Loggen Sie sich mit Ihrem nethz Account ein.

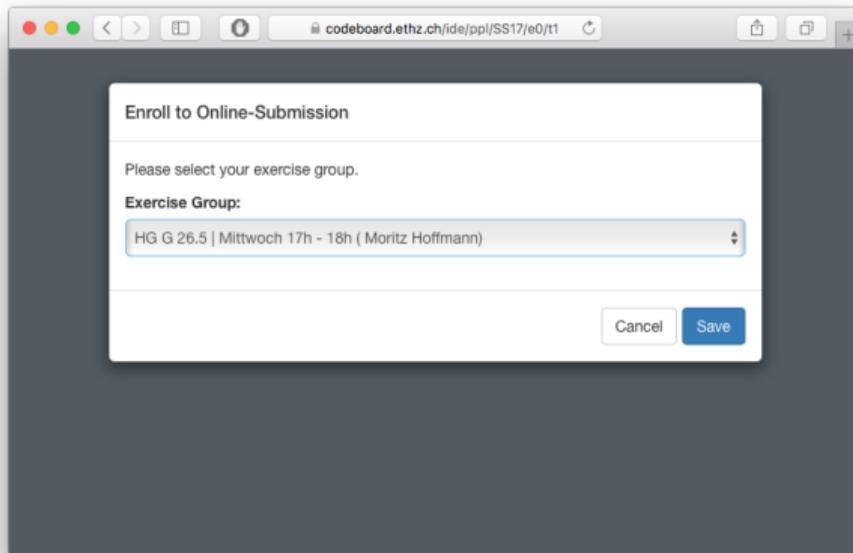


The image shows a browser window with the address bar containing `codeboard.ethz.ch/manage/SS17/mycodeboard`. The main content area displays a 'Sign In' form with the following elements:

- Title: Sign In
- Instruction: Please sign in with your ETH credentials
- Field: nethz Username (text input)
- Field: nethz Password (password input)
- Button: Login

Einschreibung in Übungsgruppen - II

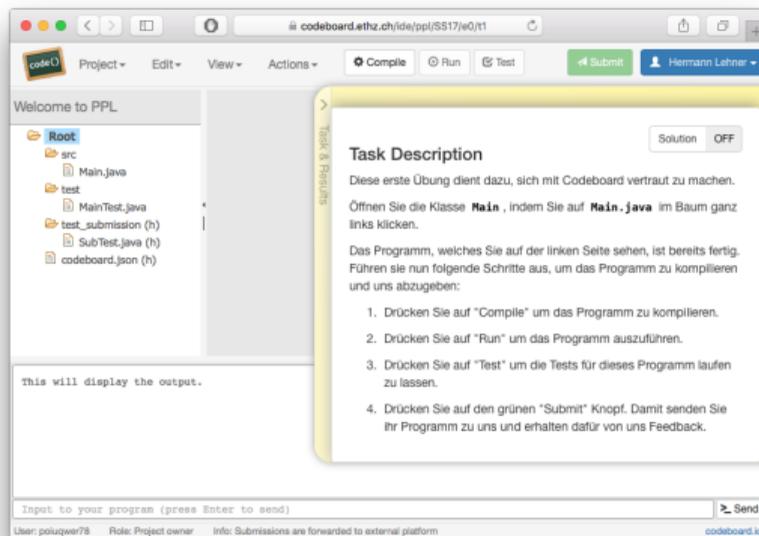
Schreiben Sie sich in diesem Dialog in eine Übungsgruppe ein.



The image shows a browser window with the URL `codeboard.ethz.ch/ide/pp/SS17/e0/t1`. A modal dialog titled "Enroll to Online-Submission" is displayed. The dialog contains the text "Please select your exercise group." followed by a label "Exercise Group:" and a dropdown menu. The dropdown menu is currently open, showing the selected option "HG G 26.5 | Mittwoch 17h - 18h (Moritz Hoffmann)". At the bottom right of the dialog, there are two buttons: "Cancel" and "Save".

Die erste Übung

Sie sind nun eingeschrieben und die erste Übung ist geladen. Folgen Sie den Anweisungen in der gelben Box. Das Übungsblatt auf der Kurshomepage enthält weitere Anweisungen und Erklärungen.



The screenshot shows the Codeboard IDE interface. At the top, there is a navigation bar with a 'code' logo, a 'Project' dropdown, and buttons for 'Edit', 'View', 'Actions', 'Compile', 'Run', and 'Test'. A green 'Submit' button and a user profile 'Herrmann Lehner' are also visible. The main area is divided into three sections: a file explorer on the left showing a project structure with folders 'src' and 'test', and files 'Main.java', 'MainTest.java', 'test_submission (h)', 'SubTest.java (h)', and 'codeboard.json (h)'; a central editor area; and a bottom section for program input. A prominent yellow box on the right side contains the 'Task Description' for the first exercise. The description includes instructions to open 'Main.java', compile, run, and test the program, and finally submit it. A 'Solution' toggle is set to 'OFF'.

codeboard.ethz.ch/ide/spl/SS17/w0/t1

code Project Edit View Actions Compile Run Test Submit Herrmann Lehner

Welcome to PPL

Root

- src
- Main.java
- test
- MainTest.java
- test_submission (h)
- SubTest.java (h)
- codeboard.json (h)

This will display the output.

Input to your program (press Enter to send) Send

User: polzwe78 Role: Project owner Info: Submissions are forwarded to external platform codeboard.io

Task Description Solution OFF

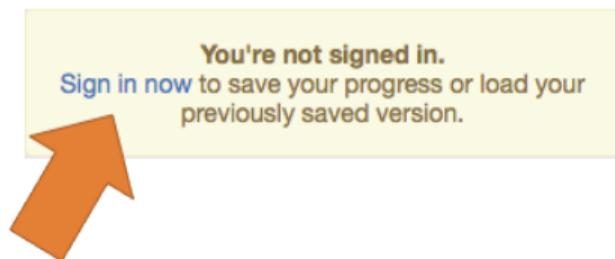
Diese erste Übung dient dazu, sich mit Codeboard vertraut zu machen. Öffnen Sie die Klasse **Main**, indem Sie auf **Main.java** im Baum ganz links klicken.

Das Programm, welches Sie auf der linken Seite sehen, ist bereits fertig. Führen sie nun folgende Schritte aus, um das Programm zu kompilieren und uns abzugeben:

1. Drücken Sie auf "Compile" um das Programm zu kompilieren.
2. Drücken Sie auf "Run" um das Programm auszuführen.
3. Drücken Sie auf "Test" um die Tests für dieses Programm laufen zu lassen.
4. Drücken Sie auf den grünen "Submit" Knopf. Damit senden Sie Ihr Programm zu uns und erhalten dafür von uns Feedback.

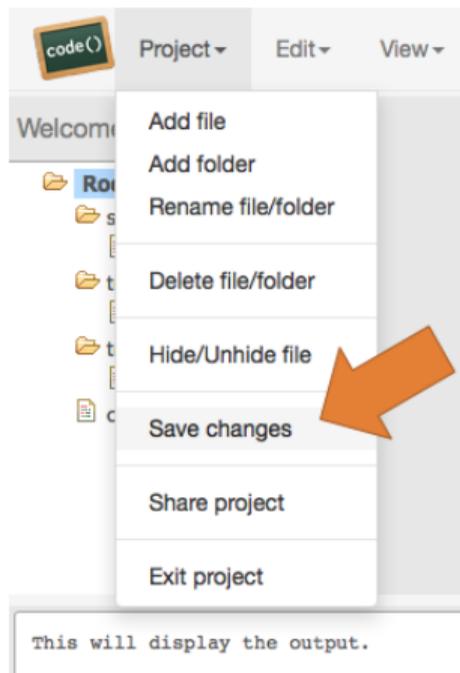
Die erste Übung - Codeboard.io Login

Falls Sie diese Nachricht sehen, klicken Sie auf [Sign in now](#) und melden Sie sich dort mit ihrem **Codeboard.io** Account ein.



Die erste Übung - Fortschritt speichern!

Achtung! Speichern Sie ihren Fortschritt regelmässig ab. So können Sie jederzeit an einem anderen Ort weiterarbeiten.



Zu den Übungen

- Seit HS 2013 für Prüfungszulassung kein Testat mehr erforderlich.
- Bearbeitung der wöchentlichen Übungsserien ist freiwillig, wird aber **dringend** empfohlen!

Relevantes für die Prüfung

Prüfungsstoff für die Endprüfung (in der Prüfungssession 2017) schliesst ein

- Vorlesungsinhalt (Vorlesung, Handout) und
- Übungsinhalte (Übungsstunden, Übungsblätter).

Prüfung (120 min) ist schriftlich. Hilfsmittel: vier A4-Seiten (bzw. 2 A4-Blätter doppelseitig) entweder handgeschrieben oder mit Fontgrösse minimal 11 Punkt.

In Ihrem und unserem Interesse

Bitte melden sie frühzeitig, wenn Sie Probleme sehen, wenn Ihnen

- die Vorlesung zu schnell, zu schwierig, zu ist
- die Übungen nicht machbar sind ...
- Sie sich nicht gut betreut fühlen ...

Kurz: wenn Ihnen irgendetwas auf dem Herzen liegt.



1.4 Altägyptische Multiplikation

Altägyptische Multiplikation

Beispiel 1: Altägyptische Multiplikation¹

Berechnung von $11 \cdot 9$

11		9
22		4
44		2
88		1
<hr/>		
99		—

9		11
18		5
36		2
72		1
<hr/>		
99		

- 1 Links verdoppeln, rechts ganzzahlig halbieren.
- 2 Gerade Zahl rechts \Rightarrow Zeile streichen.
- 3 Übrige Zeilen links addieren.

¹Auch bekannt als Russische Bauernmultiplikation

Vorteile

- Kurze Beschreibung, einfach zu verstehen.
- Effizient für Computer im Dualsystem: Verdoppeln = Left Shift, Halbieren = Right Shift

Beispiel

left shift $9 = 01001_2 \rightarrow 10010_2 = 18$

right shift $9 = 01001_2 \rightarrow 00100_2 = 4$

Fragen

- Funktioniert das immer? (z.B. für negative Zahlen)
- Wenn nicht, wann?
- Wie beweist man die Korrektheit?
- Besser als die "Schulmethode"?
- Was heisst "gut"? Lässt sich Güte anordnen?
- Wie schreibt man das Verfahren unmissverständlich auf?

Beobachtung

Wenn $b > 1$, $a \in \mathbb{Z}$, dann:

$$a \cdot b = \begin{cases} 2a \cdot \frac{b}{2} & \text{falls } b \text{ gerade,} \\ a + 2a \cdot \frac{b-1}{2} & \text{falls } b \text{ ungerade.} \end{cases}$$

Terminierung

$$a \cdot b = \begin{cases} a & \text{falls } b = 1, \\ 2a \cdot \frac{b}{2} & \text{falls } b \text{ gerade,} \\ a + 2a \cdot \frac{b-1}{2} & \text{falls } b \text{ ungerade.} \end{cases}$$

Rekursiv funktional notiert

$$f(a, b) = \begin{cases} a & \text{falls } b = 1, \\ f(2a, \frac{b}{2}) & \text{falls } b \text{ gerade,} \\ a + f(2a, \frac{b-1}{2}) & \text{falls } b \text{ ungerade.} \end{cases}$$

Funktion programmiert

```
// pre: b>0
// post: return a*b
int f(int a, int b){
    if(b==1)
        return a;
    else if (b%2 == 0)
        return f(2*a, b/2);
    else
        return a + f(2*a, (b-1)/2);
}
```

Korrektheit

$$f(a, b) = \begin{cases} a & \text{falls } b = 1, \\ f(2a, \frac{b}{2}) & \text{falls } b \text{ gerade,} \\ a + f(2a \cdot \frac{b-1}{2}) & \text{falls } b \text{ ungerade.} \end{cases}$$

Zu zeigen: $f(a, b) = a \cdot b$ für $a \in \mathbb{Z}$, $b \in \mathbb{N}^+$.

Beweis per Induktion

Anfang: $b = 1 \Rightarrow f(a, b) = a = a \cdot 1.$

Hypothese: $f(a, b') = a \cdot b'$ für $0 < b' \leq b$

Schritt: $f(a, b + 1) \stackrel{!}{=} a \cdot (b + 1)$

$$f(a, b + 1) = \begin{cases} f(2a, \overbrace{\frac{b+1}{2}}^{\leq b}) = a \cdot (b + 1) & \text{falls } b \text{ ungerade,} \\ a + f(2a, \underbrace{\frac{b}{2}}_{\leq b}) = a + a \cdot b & \text{falls } b \text{ gerade.} \end{cases}$$



Endrekursion

Die Rekursion lässt sich *endrekursiv* schreiben

```
// pre: b>0
// post: return a*b
int f(int a, int b){
    if(b==1)
        return a;
    else if (b%2 == 0)
        return f(2*a, b/2);
    else
        return a + f(2*a, (b-1)/2);
}
```



```
// pre: b>0
// post: return a*b
int f(int a, int b){
    if(b==1)
        return a;
    int z=0;
    if (b%2 != 0){
        --b;
        z=a;
    }
    return z + f(2*a, b/2);
}
```

Endrekursion \Rightarrow Iteration

```
// pre: b>0
// post: return a*b
int f(int a, int b){
    if(b==1)
        return a;
    int z=0;
    if (b%2 != 0){
        --b;
        z=a;
    }
    return z + f(2*a, b/2);
}
```



```
int f(int a, int b) {
    int res = 0;
    while (b != 1) {
        int z = 0;
        if (b % 2 != 0){
            --b;
            z = a;
        }
        res += z;
        a *= 2; // neues a
        b /= 2; // neues b
    }
    res += a; // Basisfall b=1
    return res;
}
```

Vereinfachen

```
int f(int a, int b) {  
    int res = 0;  
    while (b != 1) {  
        int z = 0;  
        if (b % 2 != 0){  
            --b; → Teil der Division  
            z = a; → Direkt in res  
        }  
        res += z;  
        a *= 2;  
        b /= 2;  
    }  
    res += a; → in den Loop  
    return res;  
}
```



```
// pre: b>0  
// post: return a*b  
int f(int a, int b) {  
    int res = 0;  
    while (b > 0) {  
        if (b % 2 != 0)  
            res += a;  
        a *= 2;  
        b /= 2;  
    }  
    return res;  
}
```

Invarianten!

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
    int res = 0;
    while (b > 0) {
        if (b % 2 != 0){
            res += a;
            --b;
        }
        a *= 2;
        b /= 2;
    }
    return res;
}
```

Sei $x = a \cdot b$.

Hier gilt $x = a \cdot b + res$

Wenn hier $x = a \cdot b + res \dots$

\dots dann auch hier $x = a \cdot b + res$
 b gerade

Hier gilt $x = a \cdot b + res$

Hier gilt $x = a \cdot b + res$ und $b = 0$

Also $res = x$.

Zusammenfassung

Der Ausdruck $a \cdot b + res$ ist eine *Invariante*.

- Werte von a , b , res ändern sich, aber die Invariante bleibt "im Wesentlichen" unverändert:
- Invariante vorübergehend durch eine Anweisung zerstört, aber dann darauf wieder hergestellt.
- Betrachtet man solche Aktionsfolgen als atomar, bleibt der Wert tatsächlich invariant
- Insbesondere erhält die Schleife die Invariante (*Schleifeninvariante*), wirkt dort wie der Induktionsschritt bei der vollständigen Induktion
- Invarianten sind offenbar mächtige Beweishilfsmittel!

Weiteres Kürzen

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
    int res = 0;
    while (b > 0) {
        if (b % 2 != 0)
            res += a;
        a *= 2;
        b /= 2;
    }
    return res;
}
```



```
// pre: b>0
// post: return a*b
int f(int a, int b) {
    int res = 0;
    while (b > 0) {
        res += a * (b%2);
        a *= 2;
        b /= 2;
    }
    return res;
}
```

Analyse

```
// pre: b>0
// post: return a*b
int f(int a, int b) {
    int res = 0;
    while (b > 0) {
        res += a * (b%2);
        a *= 2;
        b /= 2;
    }
    return res;
}
```

Altägyptische Multiplikation entspricht der Schulmethode zur Basis 2.

$$\begin{array}{r} 1\ 0\ 0\ 1 \times 1\ 0\ 1\ 1 \\ \hline 1\ 0\ 0\ 1 \quad (9) \\ 1\ 0\ 0\ 1 \quad (18) \\ \hline 1\ 1\ 0\ 1\ 1 \\ 1\ 0\ 0\ 1 \quad (72) \\ \hline 1\ 1\ 0\ 0\ 0\ 1\ 1 \quad (99) \end{array}$$

Effizienz

Frage: Wie lange dauert eine Multiplikation von a und b ?

■ Mass für die Effizienz

- Gesamtzahl der elementaren Operationen: Verdoppeln, Halbieren, Test auf "gerade", Addition
- Im rekursiven Code: maximal 6 Operationen pro Aufruf

■ Wesentliches Kriterium:

- Anzahl rekursiver Aufrufe oder
 - Anzahl Schleifendurchläufe(im iterativen Fall)
- $\frac{b}{2^n} \leq 1$ gilt für $n \geq \log_2 b$. Also nicht mehr als $6 \lceil \log_2 b \rceil$ elementare Operationen.

1.5 Schnelle Multiplikation von Zahlen

[Ottman/Widmayer, Kap. 1.2.3]

Beispiel 2: Multiplikation grosser Zahlen

Primarschule:

<i>a</i>	<i>b</i>		<i>c</i>	<i>d</i>	
6	2	·	3	7	
<hr/>					
			1	4	<i>d · b</i>
			4	2	<i>d · a</i>
			6		<i>c · b</i>
	1	8			<i>c · a</i>
<hr/>					
=	2	2	9	4	

$2 \cdot 2 = 4$ einstellige Multiplikationen. \Rightarrow Multiplikation zweier n -stelliger Zahlen: n^2 einstellige Multiplikationen

Beobachtung

$$\begin{aligned}ab \cdot cd &= (10 \cdot a + b) \cdot (10 \cdot c + d) \\&= 100 \cdot a \cdot c + 10 \cdot a \cdot c \\&\quad + 10 \cdot b \cdot d + b \cdot d \\&\quad + 10 \cdot (a - b) \cdot (d - c)\end{aligned}$$

Verbesserung?

a	b		c	d	
6	2	.	3	7	
<hr/>					
			1	4	$d \cdot b$
			1	4	$d \cdot b$
			1	6	$(a - b) \cdot (d - c)$
			1	8	$c \cdot a$
	1	8			$c \cdot a$
<hr/>					
=	2	2	9	4	

→ 3 einstellige Multiplikationen.

Grosse Zahlen

$$6237 \cdot 5898 = \underbrace{62}_{a'} \underbrace{37}_{b'} \cdot \underbrace{58}_{c'} \underbrace{98}_{d'}$$

Rekursive / induktive Anwendung: $a' \cdot c'$, $a' \cdot d'$, $b' \cdot c'$ und $c' \cdot d'$ wie oben berechnen.

→ $3 \cdot 3 = 9$ statt 16 einstellige Multiplikationen.

Verallgemeinerung

Annahme: zwei n -stellige Zahlen, $n = 2^k$ für ein k .

$$\begin{aligned}(10^{n/2}a + b) \cdot (10^{n/2}c + d) &= 10^n \cdot a \cdot c + 10^{n/2} \cdot a \cdot c \\ &+ 10^{n/2} \cdot b \cdot d + b \cdot d \\ &+ 10^{n/2} \cdot (a - b) \cdot (d - c)\end{aligned}$$

Rekursive Anwendung dieser Formel: Algorithmus von Karatsuba und Ofman (1962).

Analyse

$M(n)$: Anzahl einstelliger Multiplikationen.

Rekursive Anwendung des obigen Algorithmus \Rightarrow
Rekursionsgleichung:

$$M(2^k) = \begin{cases} 1 & \text{falls } k = 0, \\ 3 \cdot M(2^{k-1}) & \text{falls } k > 0. \end{cases}$$

Teleskopieren

Iteratives Einsetzen der Rekursionsformel zum Lösen der Rekursionsgleichung.

$$\begin{aligned}M(2^k) &= 3 \cdot M(2^{k-1}) = 3 \cdot 3 \cdot M(2^{k-2}) = 3^2 \cdot M(2^{k-2}) \\ &= \dots \\ &\stackrel{!}{=} 3^k \cdot M(2^0) = 3^k.\end{aligned}$$

Beweis: Vollständige Induktion

Hypothese H:

$$M(2^k) = 3^k.$$

Induktionsanfang ($k = 0$):

$$M(2^0) = 3^0 = 1. \quad \checkmark$$

Induktionsschritt ($k \rightarrow k + 1$):

$$M(2^{k+1}) \stackrel{\text{def}}{=} 3 \cdot M(2^k) \stackrel{\text{H}}{=} 3 \cdot 3^k = 3^{k+1}.$$



Vergleich

Primarschulmethode: n^2 einstellige Multiplikationen.

Karatsuba/Ofman:

$$M(n) = 3^{\log_2 n} = (2^{\log_2 3})^{\log_2 n} = 2^{\log_2 3 \log_2 n} = n^{\log_2 3} \approx n^{1.58}.$$

Beispiel: 1000-stellige Zahl: $1000^2/1000^{1.58} \approx 18$.

Bestmöglicher Algorithmus?

Wir kennen nun eine obere Schranke $n^{\log_2 3}$.

Es gibt praktisch (für grosses n) relevante, schnellere Algorithmen.
Die beste obere Schranke ist nicht bekannt.

Untere Schranke: $n/2$ (Jede Ziffer muss zumindest einmal angeschaut werden).

1.6 Finde den Star

Konstruktiv?

Übung: Finde ein schnelleres Multiplikationsverfahren.

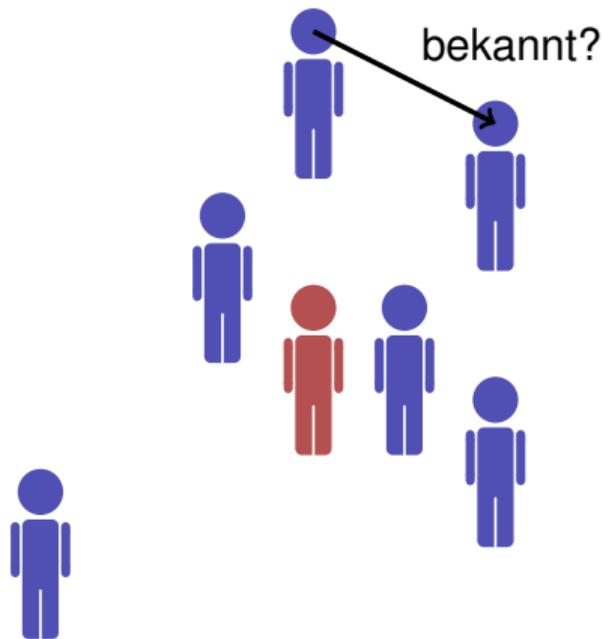
Unsystematisches Suchen einer Lösung \Rightarrow 😞.

Betrachten nun ein konstruktiveres Beispiel.

Beispiel 3: Finde den Star!

Raum mit $n > 1$ Personen.

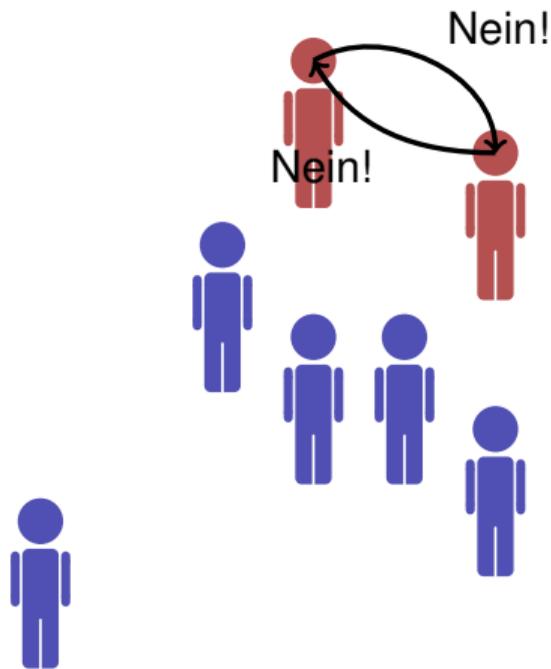
- **Star:** Person, die niemanden anderen kennt, jedoch von allen gekannt wird.
- **Elementare Operation:** Einzige erlaubte Frage an Person A : "Kennst Du B ?" ($B \neq A$)



Problemeigenschaften

- Möglich: kein Star anwesend.
- Möglich: ein Star.
- Mehr als ein Star möglich?

Annahme: Zwei Stars S_1, S_2 .
 S_1 kennt $S_2 \Rightarrow S_1$ kein Star.
 S_1 kennt S_2 nicht $\Rightarrow S_2$ kein Star. \perp



Naive Lösung

Frage jeden über jeden.

Resultat:

	1	2	3	4
1	-	Ja	Nein	Nein
2	Nein	-	Nein	Nein
3	Ja	Ja	-	Nein
4	Ja	Ja	Ja	-

Star ist 2.

Anzahl Operationen (Fragen): $n \cdot (n - 1)$.

Geht das besser?

Induktion: zerlege Problem in kleinere Teile.

- $n = 2$: Zwei Fragen genügen.
- $n > 2$: Schicke eine Person A weg. Finde den Star unter $n - 1$ Personen. Danach überprüfe A mit $2 \cdot (n - 1)$ Fragen.

Gesamt

$$F(n) = 2(n - 1) + F(n - 1) = 2(n - 1) + 2(n - 2) + \dots + 2 = n(n - 1).$$

Kein Gewinn. 😞

Verbesserung

Idee: Vermeide, den Star rauszuschicken.

- Frage eine beliebige Person A im Raum, ob sie B kennt.
- Falls ja: A ist kein Star.
- Falls nein: B ist kein Star.
- Zum Schluss bleiben 2 Personen, von denen möglicherweise eine Person X der Star ist. Wir überprüfen mit jeder Person, die draussen ist, ob X ein Star sein kann.

Analyse

$$F(n) = \begin{cases} 2 & \text{für } n = 2, \\ 1 + F(n-1) + 2 & \text{für } n > 2. \end{cases}$$

Teleskopieren:

$$F(n) = 3 + F(n-1) = 2 \cdot 3 + F(n-2) = \dots = 3 \cdot (n-2) + 2 = 3n - 4.$$

Beweis: Übung!

Bei vielen Problemen lässt sich ein induktives oder rekursives Lösungsmuster erarbeiten, welches auf der stückweisen Vereinfachung eines Problems basiert. Weiteres Beispiel in der nächsten Stunde.