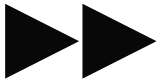


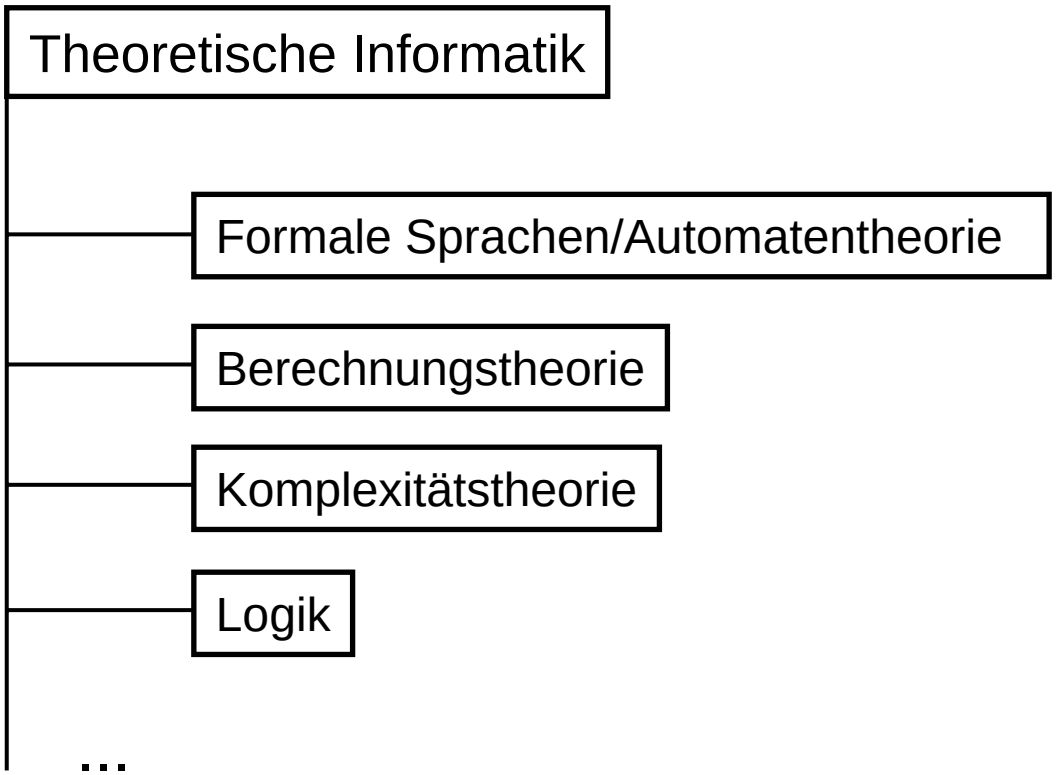
Theoretische Informatik

Wintersemester 2005/06

Steffen Lange

Theoretische Informatik

Einordnung/Motivation



Theoretische Informatik

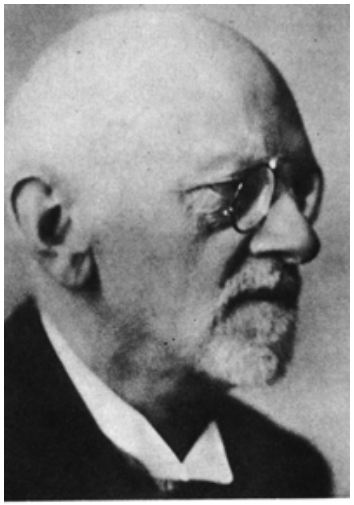
Berechnungstheorie

Welcherart Probleme sind überhaupt mit Hilfe eines Computers lösbar?
Von welchen Problemen sollte man die Finger lassen?
...

> Geben Sie einem Computer die richtige Software und er wird tun, was immer Sie wünschen. Die Maschine selbst mag ihre Grenzen haben, doch für die Möglichkeiten von Software gibt es keine Grenzen.
>
> Time-Magazine, 1984

Theoretische Informatik

Berechnungstheorie



Vortrag von David Hilbert auf dem Internationalen Mathematikerkongreß 1900 in Paris

Problem Nr. 10:
Eine diophantische Gleichung mit irgendwelchen Unbekannten, natürlichzahligen Exponenten und ganzzahligen Koeffizienten sei vorgelegt: “Man soll ein Verfahren angeben, nach welchem sich mittels einer endlichen Anzahl von Operationen entscheiden läßt, ob diese Gleichung in den ganzen Zahlen lösbar ist.”

Beispiele für Diophantische Gleichungen

$ax^2 + bx + c = 0$ (/* mit a,b und c aus den ganzen Zahlen */)

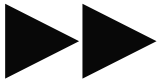
$x^2 + y^2 = z^2$

$x^n + y^n = z^n$ (/* mit n > 2 und n aus den natürlichen Zahlen */)

$x^2 – (a^2 – 1)\, y^2 = 1$ (/* mit a aus den ganzen Zahlen *)

...

Hinweis: x, y und z sind Variablen



Berechnungstheorie

Es gibt keinen Algorithmus, der folgendes leistet:

- der Algorithmus akzeptiert als Eingabe eine Beschreibung einer diophantischen Gleichung mit irgendwelchen Unbekannten, natürlichzahligen Exponenten und ganzzahligen Koeffizienten
- falls diese Gleichung eine Lösung in den ganzen Zahlen hat, so liefert der Algorithmus das Resultat „ja“
- andernfalls liefert der Algorithmus das Resultat „nein“

Theoretische Informatik

Berechnungstheorie

gegeben:

- Beschreibung von endlich vielen Fliesentypen (Quadrate unterteilt in vier farbige Dreiecke)
- beliebig viele Fliesen jedes Typs

Frage: Kann man jedes Rechteck mit Fliesen dieser Typen kacheln?

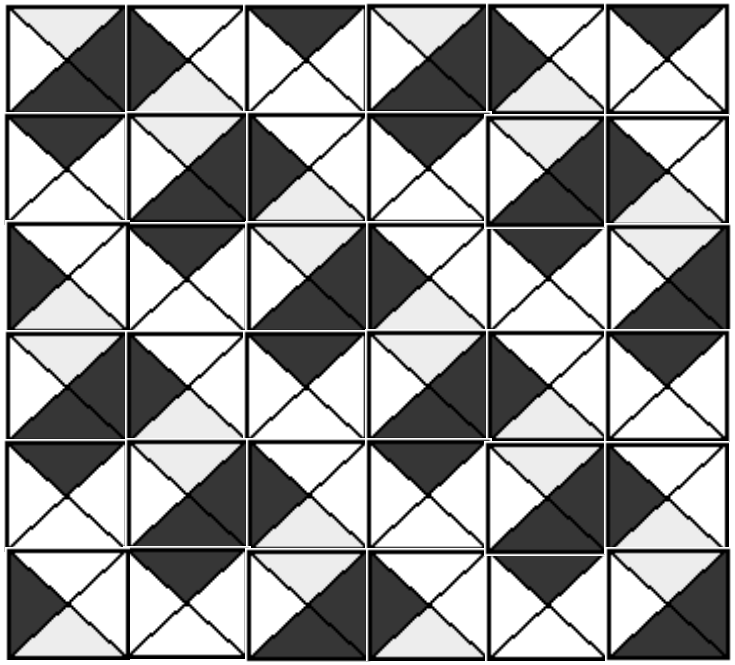
Randbedingung

- Fliesen haben eine feste Orientierung
- Farben aneinanderstoßender Fliesen müssen übereinstimmen

Theoretische Informatik

Berechnungstheorie

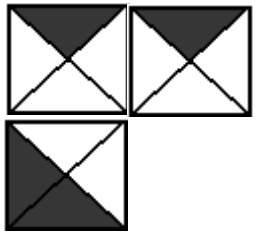
Fliesentypen, mit denen es klappt

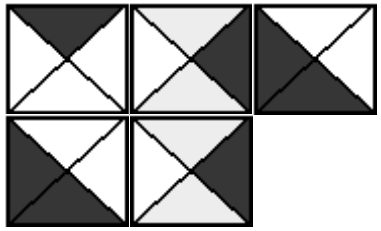


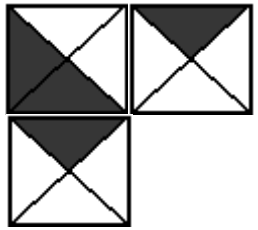
Theoretische Informatik

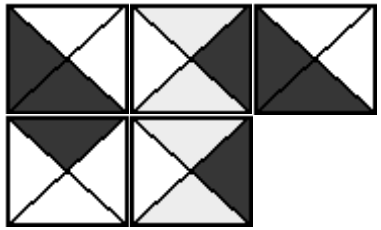
Berechnungstheorie

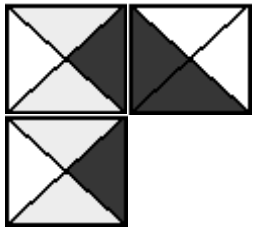
Fliesentypen, mit denen es nicht klappt











Theoretische Informatik

Berechnungstheorie

Es gibt keinen Algorithmus, der folgendes leistet:

- der Algorithmus akzeptiert als Eingabe eine Beschreibung von endlich vielen Fliesentypen
- falls sich jedes Rechteck mit Fliesen dieser Typen kacheln läßt, so liefert der Algorithmus das Resultat „ja“
- andernfalls liefert der Algorithmus das Resultat „nein“

Theoretische Informatik

Berechnungstheorie

es seien die folgenden Funktionsdefinitionen gegeben

```
int f1(int n) {
        while( n!= 1 ) {
                if ( n%2==0 ) n = n/2;
                else n = 3n+1;
                }
        return(n);
}
```

```
int f2(int n) {
        return(1);
}
```

Welche Funktionswerte werden für n = 1, 2, 3, ... berechnet?

f1(1) = 1	f2(1) = 1,
f1(2) = 1	f2(2) = 1,
f1(3) = 1	f2(3) = 1,
...	...

Theoretische Informatik

Berechnungstheorie

Collatz Problem:

```
int f1(int n) {
        while( n!= 1 ) {
                if ( n%2==0 ) n = n/2;
                else n = 3n+1;
                }
        return(n);
}
```

```
int f2(int n) {
        return(1);
}
```

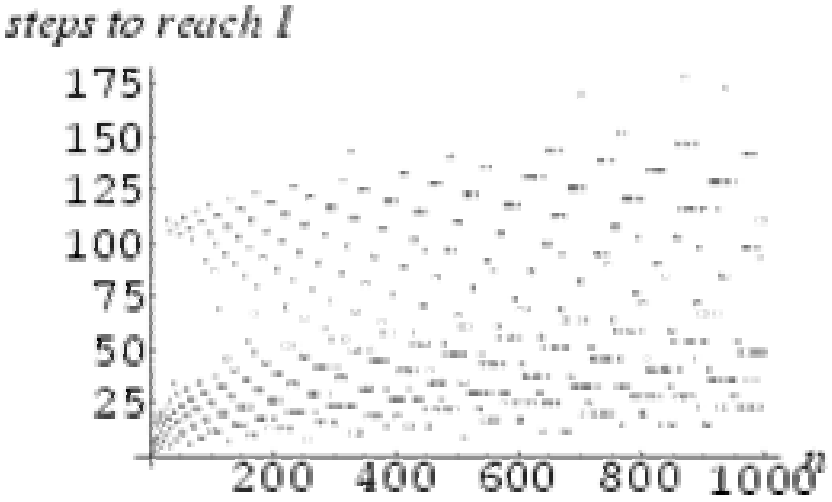
Frage: Gilt für alle $n \geq 1$: f1(n) = f2(n)?

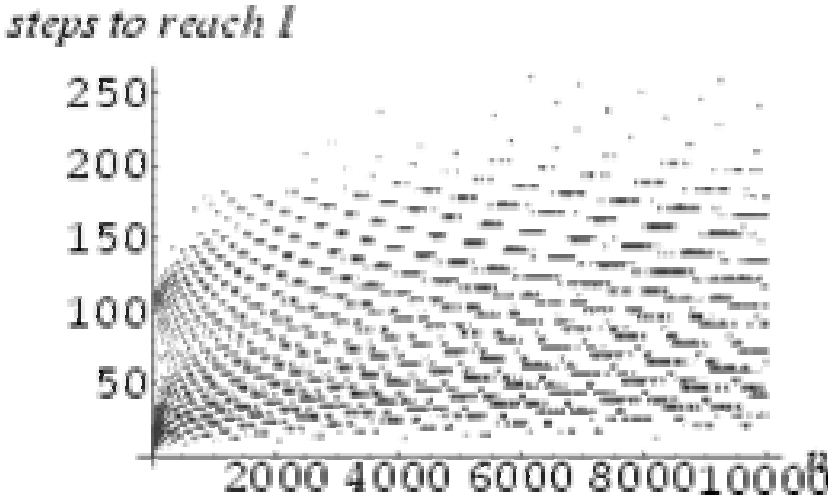
... bisher ist **unklar**, wie diese Frage zu beantworten ist

Theoretische Informatik

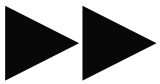
Berechnungstheorie

```
int f1(int n) {
        while( n!= 1 ) {
                if ( n%2==0 ) n = n/2;
                else n = 3n+1;
                }
        return(n);
}
```





Theoretische Informatik



Komplexitätstheorie

... Einschränkung auf lösbare Probleme

Anhand welcher Kriterien beurteilt man die Güte von Algorithmen zur Lösung eines Problems?
Wie gut können Algorithmen zur Lösung eines Problems sein?
Kann man einem Problem ansehen, ob es einen effizienten Algorithmus zu seiner Lösung gibt?
...

Theoretische Informatik

Komplexitätstheorie

Problem 1

Input: Eine Straßenkarte mit Entfernungsangaben an Straßenabschnitten und eine Stadt A.
Output: Für jede Stadt B, die Länge einer kürzesten „Reiseroute“, die von A nach B geht.

Problem 2

Input: Eine Straßenkarte mit Entfernungsangaben an Straßenabschnitten.
Output: Die Länge einer kürzesten “Rundreise”, die in einer Stadt A beginnt, durch jede Stadt führt und in A endet.

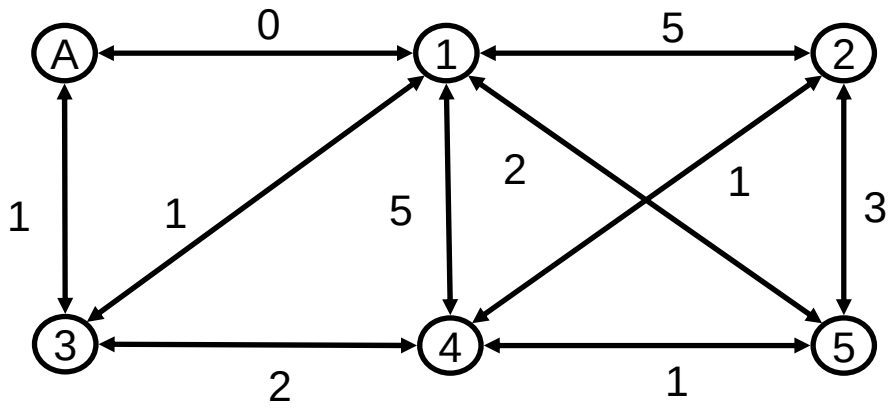
Aufgabe: Gib jeweils einen möglichst effizienten Algorithmus an, der das entsprechende Problem löst.

Theoretische Informatik

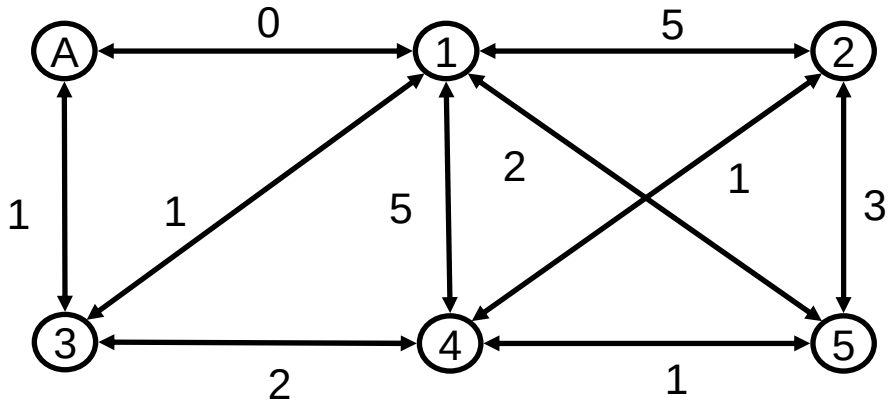
Komplexitätstheorie

Problem 1

Input: Eine Straßenkarte mit Entfernungsangaben an Straßenabschnitten und eine Stadt A.
Output: Für jede Stadt B, die Länge einer kürzesten „Reiseroute“, die von A nach B geht.

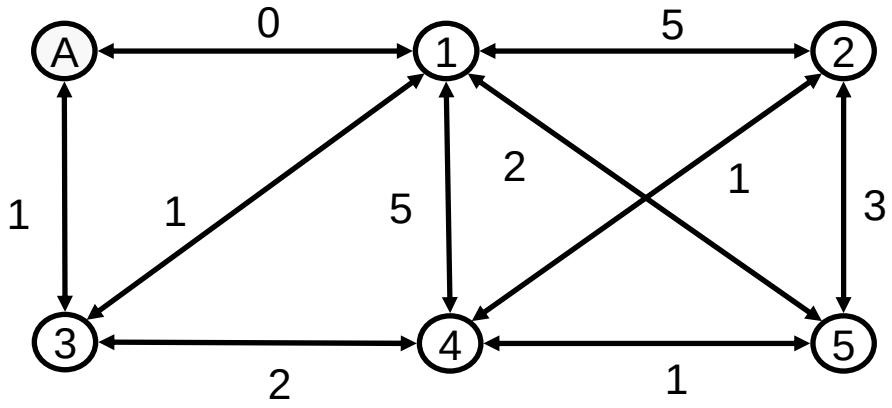


Komplexitätstheorie



- es sei X die Menge aller Städte B, für die eine kürzeste Reiseroute von A nach B bekannt ist
- es sei S die Menge der verbleibenden Städte

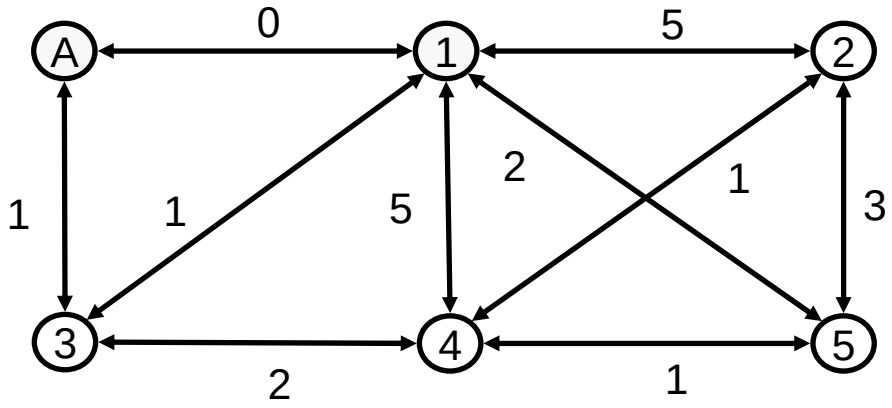
algorithmische Idee:
- bestimme eine Stadt B‘ in S mit minimaler Entfernung zu A
- füge B‘ zu X hinzu und streiche B‘ aus S



Schritt 0:

X = { A(0) }
S = { 1, 2, 3, 4, 5 }

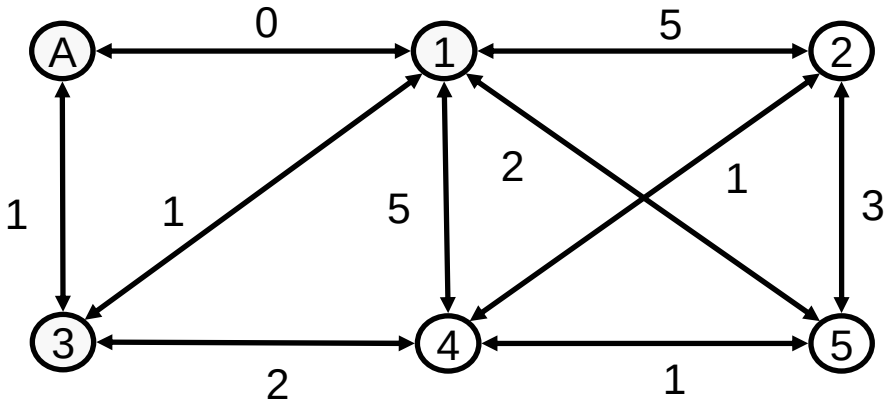
Hinweis: in Klammern – Länge eines kürzesten Wegs von A zu jeweiligen Stadt B



Schritt 1:

X = { A(0), 1(0) }
S = { 2, 3, 4, 5 }

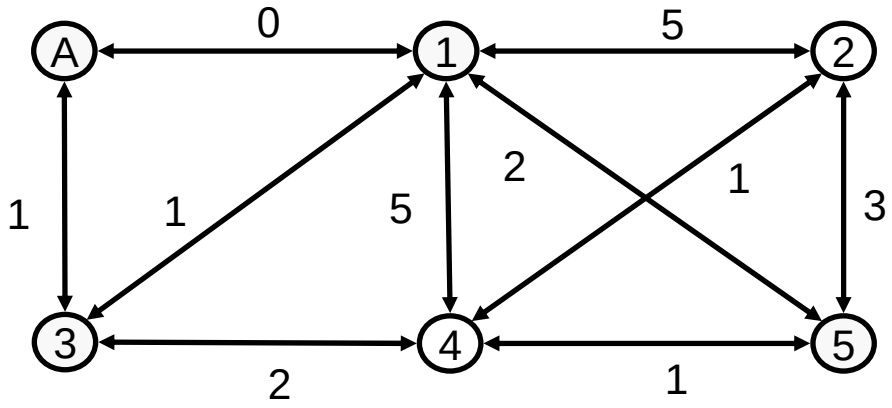
Hinweis: in Klammern – Länge eines kürzesten Wegs von A zu jeweiligen Stadt B



Schritt 2:

X = { A(0), 1(0), 3(1) }
S = { 2, 4, 5 }

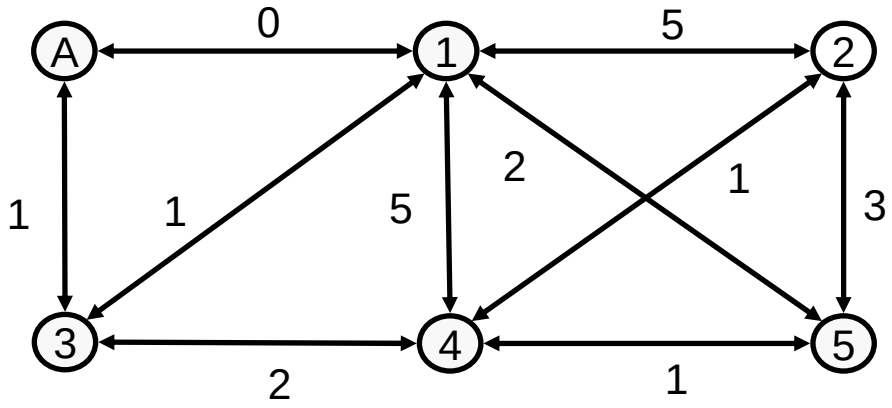
Hinweis: in Klammern – Länge eines kürzesten Wegs von A zu jeweiligen Stadt B



Schritt 3:

X = { A(0), 1(0), 3(1), 5(2) }
S = { 2, 4 }

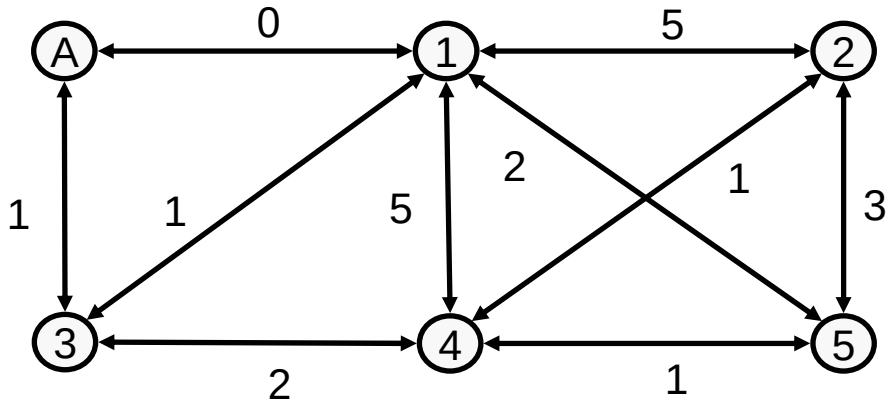
Hinweis: in Klammern – Länge eines kürzesten Wegs von A zu jeweiligen Stadt B



Schritt 4:

X = { A(0), 1(0), 3(1), 5(2), 4(3) }
S = { 2 }

Hinweis: in Klammern – Länge eines kürzesten Wegs von A zu jeweiligen Stadt B



Schritt 5:

X = { A(0), 1(0), 3(1), 5(2), 4(3), 2(4) }
S = { }

Hinweis: in Klammern – Länge eines kürzesten Wegs von A zu jeweiligen Stadt B

wenn man diese Idee geschickt implementiert ...

- kann man zeigen, daß zur Bestimmung aller kürzesten „Reiserouten“ cn^2 viele Rechenschritte ausreichen
  (/* dabei ist n die Anzahl der Städte und c eine Konstante */)

Problem 2

Input: Eine Straßenkarte mit Entfernungsangaben an Straßenabschnitten.
Output: Eine “Rundreise” minimaler Länge, die in einer Stadt A beginnt, durch jede Stadt führt und in A endet.

Annahme: jede Stadt ist mit jeder Stadt durch eine Straße verbunden

Hinweis: diese Annahme macht das Problem höchstens einfacher ...

Komplexitätstheorie

algorithmische Idee:

- wähle eine Stadt A
- bilde sukzessive alle Rundreisen $r_1, r_2, ..., r_m$, die in A beginnen, in denen jede Stadt genau einmal vorkommt und die in A enden
- setze min = r_1 und i = 2
- solange $i \leq m$:
  - prüfe, ob länge(r_i) < länge(min)
  - falls ja, setze min = r_i
  - setze i = i + 1
- gib min aus

... hoffnungslos ineffizient

es sei n die Anzahl der Städte

... dann gibt es mehr als 2^n viele Rundreisen und folglich benötigt obiger Algorithmus mehr als 2^n viele Rechenschritte

... es ist kein „qualitativ besserer“ Algorithmus bekannt

schlimmer: es wird allgemein vermutet, daß es überhaupt keinen „qualitativ besseren“ Algorithmus gibt

... „qualitativ bessere“ Algorithmen gibt es nur für Varianten von Problem 2

Problem 2‘

Input: Eine Straßenkarte mit Entfernungsangaben an Straßenabschnitten.
Output: Eine “Rundreise”, die in einer Stadt A beginnt, durch jede Stadt führt und in A endet und die **höchstens doppelt so lang** wie eine minimale Rundreise ist.

Theoretische Informatik

Komplexitätstheorie

Arbeitszeit eines Rechners mit 10^6 Op/sec

Rechenschritte	n = 10	n = 20	n = 30	n = 40	n = 50
n	0,00001 s	0,00002 s	0,00003 s	0,00004 s	0,00005 s
n^2	0,0001 s	0,0004 s	0,0009 s	0,0016 s	0,0025 s
n^3	0,001 s	0,008 s	0,027 s	0,064 s	0,125 s
n^5	0,1 s	3,2 s	24,3 s	1,7 min	5,2 min
2^n	0,001 s	1,0 s	17,9 min	12,7 Tage	35,7 Jahre
3^n	0.059 s	58 min	6.5 Jahre	3855 Jahr-hunderte	$2*10^8$ Jahr-hunderte

Hinweis: n – Maß für die Problemgröße

Komplexitätstheorie

Problemgröße, die in einer Stunde bewältigt werden kann

Rechenschritte	heute	100-fache Geschwindigkeit	1000-fache Geschwindigkeit
n	N_1	100 N_1	1000 N_1
n^2	N_2	10 N_2	31,6 N_2
n^3	N_3	4,6 N_3	10 N_3
n^5	N_4	2,5 N_4	3,9 N_5
2^n	N_5	N_5 + 6,6	N_5 + 9,9
3^n	N_6	N_6 + 4,2	N_6 + 6,3

Hinweis: n – Maß für die Problemgröße

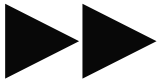
Theoretische Informatik

Formale Sprachen/Automatentheorie

- Seitenbeschreibungssprachen wie HTML, XML, ...
- höhere Programmiersprachen wie C++, Java, ...

... es geht jeweils darum, geeignete sprachliche Ausdrucksmittel zur Verfügung zu stellen, um Dokumente, Programme, ... zu beschreiben

Theoretische Informatik



Formale Sprachen/Automatentheorie

Beispiel Seitenbeschreibungssprachen (/* ... Fokus: WWW */)

Server

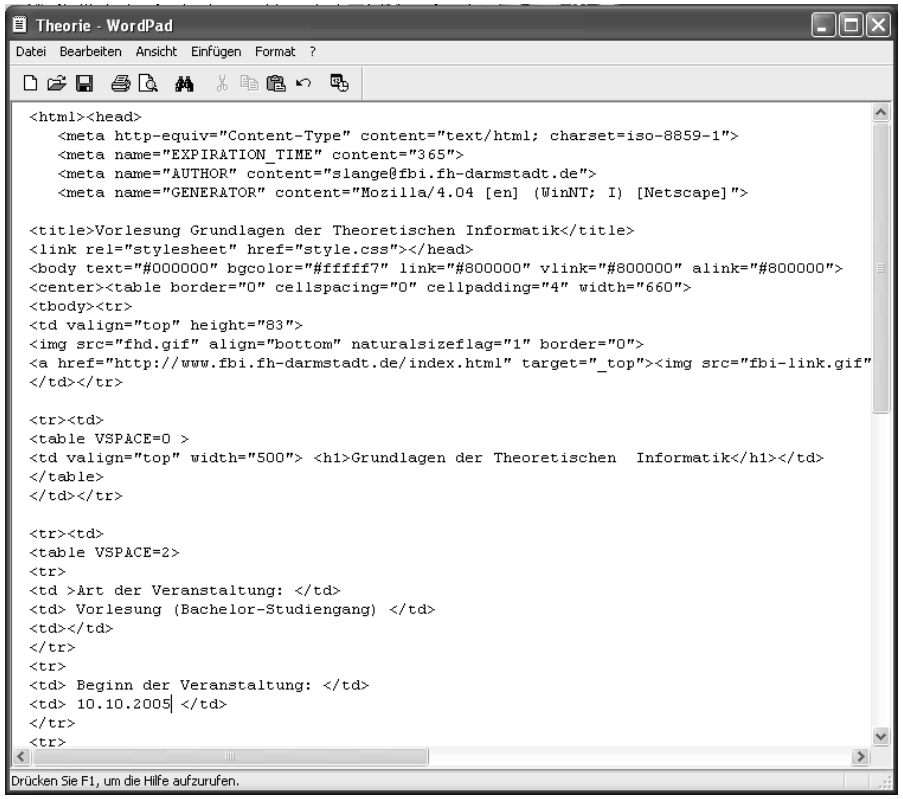
- stellt Dokumente zur Verfügung

Browser

- interpretiert diese Dokumente

Theoretische Informatik

Formale Sprachen/Automatentheorie



solcherart Dokumente produzieren wir
(/* Syntax */)

solcherart Resultate wollen wir
(/* Semantik */)

Theoretische Informatik

Formale Sprachen/Automatentheorie

Mensch

- muß die zur Beschreibung von Dokumenten zur Verfügung stehenden Ausrucksmittel kennen
- muß „verstehen“, welcherart „Effekte“ die jeweils verwendeten Ausdrucksmittel bewirken

Maschine

- muß die bereitgestellten Dokumente – sofern sie korrekt formuliert sind – sinnvoll interpretieren

höhere Programmiersprachen

Mensch

- formuliert seine Algorithmen in einer höheren Programmiersprache (/* Source-Code /*)

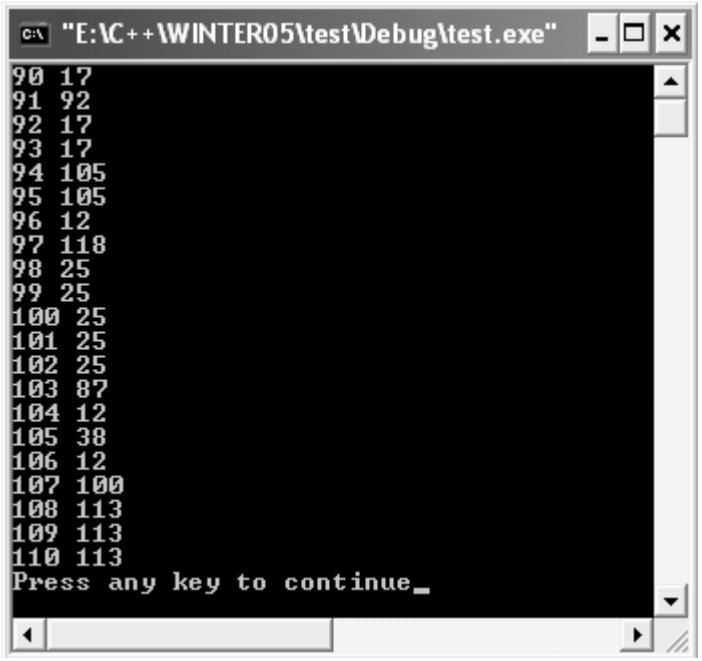
Maschine

- kann Source-Code in Maschine-Code übersetzen und Maschinen-Code ausführen

Theoretische Informatik

Formale Sprachen/Automatentheorie

```
#include <iostream>
using namespace std;
int f1(int n) {
	int z = 0;
	while( n!= 1 ) {
		if ( n%2==0 ) n = n/2;
		else n = 3*n+1;
		++z;
	}
	return(z);
}
void main() {
	for (int i = 90; i <= 110; ++i)
		cout << i <<" " << f1(i) << endl;
}
```

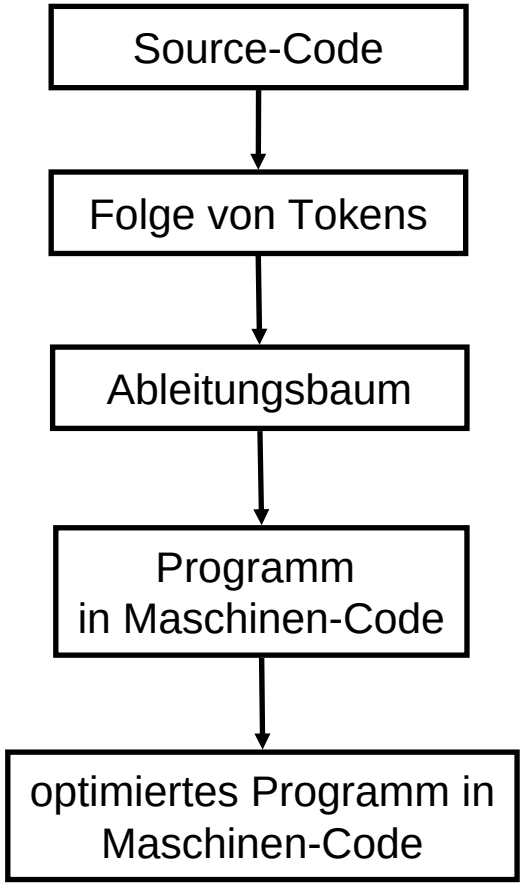


solcherart Source-Code produzieren wir
(/* Syntax */)

solcherart Resultate wollen wir
(/* Semantik */)

Theoretische Informatik

Formale Sprachen/Automatentheorie



lexikalische Analyse

syntaktische Analyse

semantische Analyse und
Codegenerierung

Codeoptimierung

Theoretische Informatik

Formale Sprachen/Automatentheorie

Mensch

- muß die zur Beschreibung von Algorithmen zur Verfügung stehenden Ausrucksmittel kennen
- muß „verstehen“, welcherart „Effekte“ die jeweils verwendeten Ausdrucksmittel bewirken

Maschine

- muß Programme in Source-Code – sofern sie korrekt formuliert sind – in Maschinen-Code übersetzen und Programme in Maschinen-Code ausführen

Relevante Fragestellung

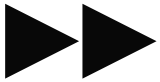
- Welche Beschreibungsmittel verwendet man, um zu präzisieren, welche syntaktischen Konstrukte überhaupt zulässig sind?

... wenigstens zwei Aspekte

- Menschen sollten diese Beschreibungen verstehen können
- Maschinen sollten herausfinden können, ob die im jeweiligen Dokument bzw. Source-Code verwendeten syntaktischen Konstrukte **zulässig** sind

... möglichst Beschreibungsmittel verwenden, so daß man beiden Aspekten gerecht wird

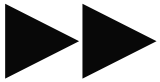
Theoretische Informatik



Gliederung der Vorlesung

0. Grundbegriffe
1. Formale Sprachen/Automatentheorie
    - 1.1. Grammatiken
    - 1.2. Reguläre Sprachen
    - 1.3. Kontext-freie Sprachen
2. Berechnungstheorie
    - 2.1. Entscheidungsprobleme
    - 2.2. Berechenbarkeitsmodelle
    - 2.3. Die Churchsche These
    - 2.4. Unentscheidbarkeit
3. Komplexitätstheorie
    - 3.1. Nicht-deterministische Turing Maschinen
    - 3.1 Komplexitätsmaße
    - 3.2. Das P=NP? Problem

Theoretische Informatik



Organisatorisches

- Vorlesung
  - wöchentlich
  - Folien im Netz
    http://www.fbi.fh-darmstadt.de/~slange
- Übung
  - 14-tägig (/* in Gruppen */)
  - erste Übungen in der nächsten Woche
  - Aufgabenblätter im Netz
    http://www.fbi.fh-darmstadt.de/~slange
- Sprechstunde
  - wöchentlich (/* Mi 10:00 – 11:00 Uhr, D15, 207a */)

Theoretische Informatik

Literatur

- J.E. Hopcroft, J.E. Ullmann, Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie, Addison-Wesley, Bonn, 1990.
- U. Schöning, Theoretische Informatik – kurz gefaßt, Spektrum Akademischer Verlag, Heidelberg, 1997.
- U. Hedtstück, Einführung in die Theoretische Informatik, Oldenburg Verlag, München, 2000.
- I. Wegener, Theoretische Informatik – eine algorithmen-orientierte Einführung, Teubner Verlag, Stuttgart, 1999.
- I. Wegener, Kompendium Theoretische Informatik – eine Ideen-sammlung, Teubner Verlag, Stuttgart, 1999.
- J. Hromkovic, Theoretische Informatik, Teubner Verlag, Stuttgart, 2002.