Infovorkurs, Teil II: Theoretische Informatik

Lutz Büch

08.10.2015

Webseite

Mittlerweile gibt es die Folien und die Übungen online.

Außerdem die Ergebnisse der Umfrage! Ich empfehle auch die Links.

`www.geile-hirnbude.de/vorkurs`

Rückblick

Haben schon folgendes herausgefunden:

- Was ist ein Algorithmus?
- Algorithmen lösen Instanzen eines Problems
- Verschiedene Algorithmen lösen dieselben Probleme
- worst-case Zeitkomplexität - eine Abschätzung für die Laufzeit
- Tricks können Faktoren an Laufzeit sparen
- Um aber *wesentlich* schneller zu werden (bessere Komplexitätsklasse), braucht es oft ganz andere Ansätze

Programmieren

- Informatik ist nicht gleich Programmieren
- Aber Informatik ohne Programmieren ist nichts
- Ich hatte unheimlichen Respekt vor Programmieren
- So viele Konzepte, die man benutzen soll, ohne dass sie einem klar sind!

Konzepte!

- Variablen
- Compiler
- Typen
- Namespace
- Prozess
- Speicher
- Stack
- Subroutine
- Garbage collection
- Initialisierung
- Katzenbilder

Algorithmus, revisited

Zurück zum Algorithmus!

In den 1930ern suchte man nach einem formalen Modell für **Berechenbarkeit**. Was kann man alles *intuitiv* berechnet? Wie kann ich das durch ein einheitliches Konzept darstellen?

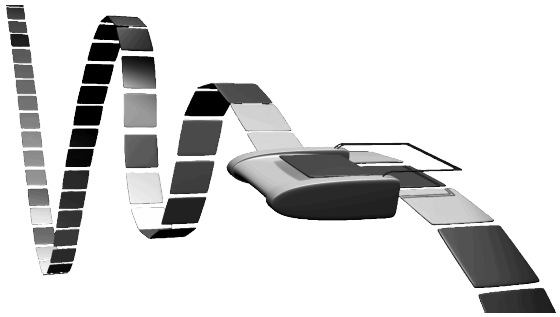
Durch geeignete Kodierung decken berechenbare Funktionen auch Probleme ab wie Suchprobleme, Sortierprobleme, kürzeste Wege in Graphen... - alles was man sich wünscht!

Algorithmus, revisited

Alan Turing lieferte ein solches Konzept noch vor der Konstruktion der ersten programmierbaren Computer in den 1940ern: Die **Turing-Maschine**.

Die erste universelle Rechenmaschine entstand somit zuerst auf dem Papier. Bis heute besteht Konsens, dass dieses Konzept tatsächlich alles praktisch Berechenbare erfasst (Church-Turing-These).

Beschreibung



Bestandteile:

- unendliches Band
- Schreib-Lese-Kopf

Band ist in Zellen aufgeteilt

In jeder Zelle kann ein Buchstabe eines festgelegten Alphabets geschrieben stehen, z.B. das Binäralphabet $\{0, 1\}$

Hinzukommt ein Buchstabe (z.B. \$), der auf dem Band in jeder Zelle vor Beginn der Rechnung steht: $\{0, 1, \$\}$

Schreib-Lese-Kopf

Der Schreib-Lese-Kopf kann schreiben und lesen. Außerdem kann er sich auf dem Band nach links oder rechts bewegen (oder stehenbleiben).

Der Kopf befindet sich immer in einem aus einer festgelegten Menge von **Zuständen**.

Schließlich, am wichtigsten: Der Kopf enthält ein **Programm**. Dieses ist einfach eine Tabelle mit **Regeln**.

Übergangsregeln

Eine Regel sieht immer so aus:
Wenn du

- *dich in* **Zustand q** *befindest, und*
- *auf einer Zelle mit dem* **Buchstaben a** *stehst,*
- *überschreibe ihn mit dem* **Buchstaben b**,
- *gehe in* **Richtung D** *und*
- *nimm den* **Zustand p** *an.*

Kurz: $(q, a) \mapsto (b, D, p)$

Richtungen können L, R und N sein (links, rechts, neutral/stehenbleiben).

Wir nehmen wieder an, dass es zu jedem Paar aus Zustand q und Buchstabe a höchstens eine Übergangsregel $(q, a) \mapsto \ldots$ gibt.

Initialisierung

Die Maschine soll eine Eingabe in eine Ausgabe verwandeln.

Dazu wird das Band zu Anfang mit der Eingabe initialisiert:
$\ldots|\$|\$|\$|\$|\mathbf{0}|1|1|0|1|\$|\$|\$|\$|\ldots$

Der Schreib-Lese-Kopf befindet sich auf der hervorgehobenen **0** und befindet sich in einem ausgezeichneten **Startzustand**.

Die Eingabe muss die Probleminstanz geeignet kodieren. Einfaches Beispiel: Natürliche Zahl in Binärdarstellung

Ausgabe

Sobald die Maschine in einen der ausgezeichneten **Stoppzustände** übergeht, hält sie an.

Als Ausgabe wird dann der längste zusammenhängende String von der aktuellen Position des Kopfes nach rechts gewertet.

Beispiel:
$\ldots|\$|\$|\$|\$|1|0|\mathbf{1}|0|0|\$|\$|1|1|\$|\$|\$|\ldots$

In diesem Fall wäre die Ausgabe 100.

Spielzeugbeispiel

Ersetze alle Einsen durch Nullen und umgekehrt!

Aus 1001011 mache 0110100,
aus 001000 mache 110111 usw...

$(s,0) \mapsto (1,R,s)$
$(s,1) \mapsto (0,R,s)$
$(s,\$) \mapsto (\$,L,z)$
$(z,0) \mapsto (0,L,z)$
$(z,1) \mapsto (1,L,z)$
$(z,\$) \mapsto (\$,R,e)$ mit e Stoppzustand

Wir überprüfen unser Programm mit einer computersimulierten Turingmaschine:
"Tursi" ist eine Java-Implementation einer Turingmaschine
`ais.informatik.uni-freiburg.de/tursi/`

Fragerunde!

www.govote.at, code 54 26 84

Addition von Binärzahlen

Wir wollen Binärzahlen addieren!

Wir starten klein: Addiere 1 zu einer Binärzahl:
Aus 001101 mache 001110,
aus 10100 mache 10101 usw.

Ab hier verwenden wir die vereinfachte Notation von Tursi:
q a b D p statt $(q, a) \mapsto (b, D, p)$

Addition von 1

Zunächst: Springe hinter die Zahl
j 0 0 R j
j 1 1 R j
j $ $ L inc

Dann inkrementiere mit Übertrag:
inc 0 1 L inc0
inc 1 0 L inc
inc0 0 0 L inc0
inc0 1 1 L inc0
inc0 $ $ R end
inc $ 1 N end

Analog geht Subtraktion von 1, aber passe auf bei Eingabe Null!

Addition von Binärzahlen

Baue alles zusammen!
Also Springen, Addition, Subtraktion. Dazu kommt etwas Prüfen

Springe hinter erste Zahl:
jump1 0 0 R jump1
jump1 1 1 R jump1
jump1 $ $ R jump2

Springe hinter zweite Zahl und checke, ob sie größer Null ist:
jump2 0 0 R jump2
jump2 1 1 R jump2ok
jump2ok 0 0 R jump2ok
jump2ok 1 1 R jump2ok
jump2ok $ $ L dec
jump2 $ $ L toend1

Addition von Binärzahlen

Subtrahiere 1 von zweiter Zahl:

dec 1 0 L dec0
dec0 1 1 L dec0
dec0 0 0 L dec0
dec 0 1 L dec
dec0 $ $ L inc

Addiere 1 zu erster Zahl:

inc 0 1 L inc0
inc0 1 1 L inc0
inc0 0 0 L inc0
inc 1 0 L inc
inc $ 1 R jump1
inc0 $ $ R jump1

Addition von Binärzahlen

Springe vor die zweite Zahl, um später zu halten:
toend1 0 0 L toend1
toend1 1 1 L toend1
toend1 $ $ L toend2

Springe vor die erste Zahl, und halte:
toend2 0 0 L toend2
toend2 1 1 L toend2
toend2 $ $ R end

Maschinensprache

Eine Turingmaschine ist Modell moderner elektronischer Computer

Von Neumann hat nach eigener Angabe Inspiration durch die Turingmaschine für seine "Von-Neumann-Architektur" bekommen

Auch auf einem modernen Computer hat man eigentlich nur solch primitive Grundoperationen zur Verfügung!

Höhere Sprachen

Aber wir programmieren doch nicht so!

Wo ist/sind mein/e geliebte/n (verhassten?)...

- Variablen
- Flusskontrolle
- Subroutinen
- Typen
- Datenstrukturen
- Objekte
- ...?

Das sind alles Konzepte "höherer" Programmiersprachen

Höhere Sprachen

Wie können wir höhere Programmiersprachen benutzen?

Dazu brauchen wir ein **Programm** (in Maschinensprache), die **Programme** in diesen höheren Programmiersprachen in **Programme** in der Maschinensprache übersetzen: Einen **Compiler**

Bootstrapping

Um einen Compiler zu erhalten, kann man ihn in Maschinensprache schreiben

Einfacher ist es, man schreibt den Compiler schon in der höheren Sprache!

Aber wie verwandele ich diesen Compiler in Hochsprache in einen Compiler in Maschinensprache, wenn ich dazu einen Compiler in Maschinensprache brauche??

Bootstrapping

Wenn ich einen Compiler in Maschinensprache **habe**, kann ich einen Compiler in Maschinensprache **erhalten**.

Solch einen Ringschluss hatten wir schon mal:
MergeSort funktioniert genau dann, wenn MergeSort funktioniert...

Bootstrapping

Schreibe einen Mini-Compiler in Maschinensprache (niemand will viel Maschinencode schreiben müssen!)

Mit dessen geringen Sprachumfang kann ich immerhin einen nächsten Compiler übersetzen, der einen größeren Sprachumfang hat, usw.

Am Ende steht der Compiler mit vollem Umfang der höheren Sprache, der in Maschinensprache vorliegt.