Dr. Jürg M. Stettbacher
Neugutstrasse 54
CH-8600 Dübendorf

Telefon: +41 43 299 57 23
E-Mail: dsp@stettbacher.ch

Algorithmen

Konzept und Anwendungen in der Informatik

Version 2.14
2017-09-18

Zusammenfassung: Konzept und Eigenschaften von Algorithmen werden vorgestellt und mit Blick auf Anwendungen in der Informatik erläutert. Es wird gezeigt, dass sich ein Computer als algorithmische Maschine auffassen lässt.

Inhaltsverzeichnis

1 Zweck

Dieses Dokument verfolgt die nachstehenden Ziele:

- Einführen des grundsätzlichen Konzepts von *Algorithmen*.
- Erläutern der Attribute *deterministisch*, *stochastisch*, *terminierend*.
- Aufzeigen der Verwendung von *formalen Sprachen* zur Beschreibung von Algorithmen.
- Illustration des Konzepts anhand von *Beispielen*, insbesondere zu iterativen und rekursiven Algorithmen.
- Der Leser sollte nach der Lektüre beschreiben können, was ein Algorithmus ist und welches seine Merkmale sind. Er sollte das Konzept selbst in praktischen Beispielen anwenden können.

2 Einleitung

Ein Algorithmus ist im Grund nichts anderes als eine Anleitung, die beschreibt, wie etwas zu machen ist. Ein typisches Beispiel ist ein Kochrezept. Es definiert, welche Zutaten in welchen Mengen notwendig sind und es gibt an, in welcher Weise sie zu Verarbeitung sind, so dass am Ende eine fertige Speise resultiert.

Im engeren Sinn, also in der Mathematik und Informatik, ist der Algorithmus eine Vorschrift, die bei gleicher Ausgangslage immer exakt das selbe Resultat liefert. Algorithmen sind also deterministisch. Es besteht kein Interpretationsspielraum, wie das beispielsweise bei einem Kochrezept oft der Fall ist. Aus diesem Grund wählt man in der Regel eine formale Sprache zur Beschreibung eines Algorithmus.

Beispiel Ein alltäglicher Algorithmus in der Mathematik ist die schriftliche Addition von zwei natürlichen[1] Dezimalzahlen[2]:

Erster Summand		**6**	**3**	**5**	**7**
Zweiter Summand	**+**	**4**	**6**	**7**	**2**
Übertrag		1	1		
Resultat	**1**	**1**	**0**	**2**	**9**

Die Vorschrift für das Vorgehen lautet:

1. Schreibe die beiden Zahlen rechtsbündig untereinander, so dass Ziffern mit der gleichen Wertigkeit spaltenweise untereinander stehen.
2. Zeichne einen Strich unter die beiden Zahlen und reserviere darunter Platz für zwei Zeilen, die obere für den *Übertrag*, die untere für das *Resultat*.
3. Beginne mit der Spalte kleinster Wertigkeit, also ganz rechts.
4. Bilde die Summe aller Ziffern dieser Spalte. Berücksichtige auch den *Übertrag*, falls einer vorhanden ist.
5. Nimm von der Ziffernsumme die Ziffer mit der Wertigkeit 1 und schreibe sie in der selben Spalte in die *Resultat*-Zeile.
6. Nimm von der Ziffernsumme die Ziffer mit der Wertigkeit 10 und schreibe sie in der nächsten Spalte links unter dem Strich in die *Übertrag*-Zeile.

[1] Die Menge $\mathbb{N}$ der natürlichen Zahlen umfasst 1, 2, 3, 4, ...

[2] Bei der Addition von zwei Dezimalzahlen ist sicher gestellt, dass jeweils maximal ein einstelliger Übertrag anfällt. Insbesondere bei Binärzahlen ist das nicht der Fall.

7. Falls in der nächsten Spalte links Ziffern stehen, so wähle diese Spalte und fahre bei Punkt 4 fort.

8. Fertig: Das Resultat der Summe steht auf der *Resultat*-Zeile.

3 Begriffe

Deterministisch Ein deterministischer *Prozess* ist ein Ablauf von Teilschritten, die auf Grund der Ausgangslage in einer genau definierten Reihenfolge durchlaufen werden. Ein deterministisches *Resultat* ergibt sich zwingend und reproduzierend aus einer gegebenen Ausgangslage. Das Gegenteil von deterministisch ist stochastisch.

Typische Algorithmen, wie das Additionsbeispiel von oben, sind deterministische Prozesse mit deterministischen Resultaten. Die Vorschrift für die Ausführung ist mit Bedacht so gewählt, dass keine Wahlfreiheit besteht, sondern jeder Teilschritt soll eindeutig und unmissverständlich sein.

Allerdings gibt es einige (eher seltene) Anwendungen, wo bewusst stochastische Teilschritte in einen Algorithmus eingebaut werden. Dies tut man zum Beispiel dann, wenn der Algorithmus zu einander in Beziehung stehende Daten bearbeiten soll, und wenn der Aufwand für die Bearbeitung vom ersten gewählten Datenelement abhängt. Um zu verhindern, dass in diesem Fall der Algorithmus (zufällig) immer im ungünstigsten Fall läuft, ist es oft vorteilhaft, das erste Datenelement randomisiert auszuwählen. Der Algorithmus ist dann nicht deterministisch, weil die Ausführungszeit auf den selben Daten bei wiederholter Ausführung unterschiedlich sein kann. Das Resultat ist aber dennoch deterministisch. Oft wird dieses Verfahren in Sortier-Algorithmen angewandt.

Stochastisch In einem stochastischen Prozess ist der Ablauf durch Wahrscheinlichkeiten definiert. Für ein stochastisches Resultat kann eine Wahrscheinlichkeitsverteilung angegeben werden.

Terminierend Ein terminierender Prozess ist ein Ablauf, der in jedem Fall nach endlich vielen Teilschritten zum Ziel führt. Das Gegenteil von terminierend ist unendlich. Das Additionsbeispiel von oben ist terminierend, denn für jede Stelle der beiden Zahlen wird die in der Vorschrift beschriebene Schleife genau einmal durchlaufen.

Ein nicht terminierender Algorithmus wäre zum Beispiel einer, der alle Primzahlen sucht. Da es unendlich viele Primzahlen gibt, würde der betreffende Algorithmus nie fertig werden. Terminierend wäre er dagegen dann, wenn man zum Beispiel alle bis zu 10-stelligen Primzahlen suchen würde. Dies ist eine endliche Menge.

Teilschritt Ein Teilschritt eines Prozesses ist eine Operation, die im übergeordneten Kontext sinnvollerweise nicht weiter unterteilt wird. Teilschritte werden daher als atomar angesehen. Im Additionsbeispiel besteht der Algorithmus aus total 8 Teilschritten.

4 Notation

Weil Programmiersprachen weit verbreitete und allgemein bekannte formale Sprachen sind, werden Algorithmen häufig in einer Programmiersprache oder einer daran angelehnten Notation formuliert. Damit stehen die typischen Konstrukte der jeweiligen Programmiersprache zur Verfügung. Im allgemeinen sind das Bedingungen und Schleifen.

Die folgende Tabelle listet einige willkürlich ausgewählte Beispiele auf. Die Kleinbuchstaben a, b bezeichnen Variablen, die einen Wert haben, Grossbuchstaben A, B bezeichnen Anweisungen.

Typ	Anweisung	C-Notation

Bedingte Anweisung	Falls $(a = 0)$,
	führe A aus.
		`if (a == 0) {`
		`  A;`
		`}`

Bedingte Anweisung mit Alternative	Falls $(a = 0)$,
	führe A aus,
	sonst führe B aus.
		`if (a == 0) {`
		`  A;`
		`} else {`
		`  B;`
		`}`

Mehrfach bedingte Anweisung	Falls $((a = 0)$ und $(b = 0))$,
	führe A aus.
		`if ((a == 0) && (b == 0)) {`
		`  A;`
		`}`

Unbedingte Schleife	Wiederhole A zehn mal.	`for (k=0; k<10; k++) {`
		`  A;`
		`}`

Bedingte Schleife	Solange $(a < 10)$, führe A aus.	`while (a < 10) {`
		`  A;`
		`}`

Die Teilschritte eines Algorithmus werden typischerweise sequenziell ausgeführt, so wie es bei einer Programmiersprache üblich ist. In Spezialfällen kann allerdings auch eine parallele Verarbeitung vorgesehen werden. Und selbstverständlich kann man Algorithmen hierarchisch aufbauen, indem wiederum analog zu den bekannten formalen Hochsprachen Funktionen, Methoden oder Unteralgorithmen definiert werden, die dann mehrfach verwendbar sind.

5 Beispiele

Heron-Verfahren Heron von Alexandria beschrieb im 1. Jh. n. Chr. einen Algorithmus, mit dem man die Wurzel einer Zahl $X \geq 0$ beliebig genau annähern kann. Grundlage: Es sei ξ eine Schätzung von $\sqrt{X}$. Dann ist das Polynom $p(\xi) = \xi^2 - X$ ein Fehlermass, dessen Betrag es zu minimieren gilt. Mit Hilfe des Newton-Verfahrens können wir die Nullstelle von $p(\xi)$ iterativ annähern (Erklärungen dazu sind im Anhang, Kapitel 7 zu finden):

$$\xi_{n+1} = \frac{\xi_n}{2} + \frac{X}{2\xi_n}$$

Damit folgt der Algorithmus von Heron, wobei wir als Abbruchkriterium für den Schätzfehler die obere Limite ε vorschreiben:

1. Wähle den Zähler $n = 0$ und den Startwert $\xi_n = 1$.
2. Falls der Betrag des Fehlermasses $|\xi_n^2 - X| < \varepsilon$ ist, gehe zu Punkt 6.
3. Berechne ξ_{n+1} aus ξ_n mit Hilfe der Beziehung:

$$\xi_{n+1} = \frac{\xi_n}{2} + \frac{X}{2\xi_n}$$

4. Inkrementiere n.
5. Gehe zu Punkt 2.
6. Ende: Das Fehlermass $|p(\xi_n)|$ hat die Limite ε unterschritten. Der Wert ξ_n ist die finale Schätzung für $\sqrt{X}$.

Mit den Werten $X = 3$ und $\varepsilon = 0.001$ wollen wir den Algorithmus prüfen:

n	ξ_n	$p(\xi_n)$	ξ_{n+1}
0	1.00000	-2.00000	2.00000
1	2.00000	1.00000	1.75000
2	1.75000	0.06250	1.73214
3	1.73214	0.00032	

Bereits nach $n = 3$ Iterationen erreichen wir $p(\xi_3) < \varepsilon$. Damit stoppt der Algorithmus. Der resultierende Schätzwert beträgt $\xi_3 = 1.73214$. Verglichen mit dem wahren Wert $\sqrt{3} = 1.73205$ liegt der Schätzfehler ungefähr bei $9 \cdot 10^{-5}$.

Wir wollen noch anmerken, dass das Heronverfahren deterministisch und terminierend ist. Letzteres geht aus der Konvergenzeigenschaft des Newton-Verfahrens hervor.

Euklid'scher Algorithmus Wir bezeichnen mit dem Operator $\mathrm{ggT}(x, y)$ den grössten gemeinsamen Teiler (ggT) der ganzen Zahlen x und y. Wir wollen hier nur Werte $x \geq 0$ und $y \geq 0$ ansehen. Euklid beschrieb im Jahr 300 v. Chr. einen Algorithmus zum Auffinden des ggT. Das Verfahren basiert auf den folgenden vier Sätzen:

1. Der ggT von zwei gleichen Zahlen ist die Zahl selbst. Beispiel: $\mathrm{ggT}(8, 8) = 8$.

$$\boxed{\mathrm{ggT}(x, x) = x}$$

2. Der ggT von zwei Zahlen ist unabhängig davon, in welcher Reihenfolge die Zahlen angegeben werden. Beispiel: $\mathrm{ggT}(21, 35) = \mathrm{ggT}(35, 21) = 7$.

$$\boxed{\mathrm{ggT}(x, y) = \mathrm{ggT}(y, x)}$$

3. Falls eines der Argument oder beide null sind, so gilt:

$$\boxed{\mathrm{ggT}(x, 0) = x \qquad \mathrm{ggT}(0, 0) = 0}$$

4. Es sei $\mathrm{ggT}(x, y) = \lambda$ und $x < y$. Das bedeutet, dass $x = \alpha_1 \cdot \lambda$ und $y = \alpha_2 \cdot \lambda$ mit zwei natürlichen Konstanten α_1 und α_2. Oder einfacher ausgedrückt: x und y sind Vielfache von λ. Damit folgt, dass auch $y - x = \alpha_2 \cdot \lambda - \alpha_1 \cdot \lambda = (\alpha_2 - \alpha_1) \cdot \lambda$ ein Vielfaches von λ ist.
   Beispiel: $\mathrm{ggT}(21, 35) = \mathrm{ggT}(21, 14) = 7$.

$$\boxed{\mathrm{ggT}(x, y) = \mathrm{ggT}(x, y - x) \qquad \text{mit } x < y}$$

In einer an C angelehnten Notation lautet Euklids Algorithmus so:

```
ggT(x,y) {
         if (y == 0) { return(x); }
         if (x == 0) { return(y); }
         if (x < y) {
             return ggT(x, y-x);
         } else {
             return ggT(x-y, y);
         }
}
```

Beachten Sie, dass der Algorithmus rekursiv formuliert wurde. Das heisst, dass der Algorithmus aus sich selbst heraus aufgerufen wird, aber mit jeweils veränderten Argumenten. Wir testen das Verfahren:

```
ggT(21,35) :
    1. Ebene     ggT(21, 35) = ggT(21, 35-21) = ggT(21, 14)
    2. Ebene     ggT(21, 14) = ggT(21-14, 14) = ggT(7, 14)
    3. Ebene     ggT(7, 14)  = ggT(7, 14-7)   = ggT(7, 7)
    4. Ebene     ggT(7, 7)   = ggT(7, 7-7)    = ggT(7, 0)
    5. Ebene     return(7)
```

Der Test liefert das richtige Resultat. Auch hier finden wir, dass das Verfahren deterministisch und terminierend ist.

6 Maschinen

Computer sind algorithmische Maschinen. Sie wurden für die Abarbeitung von Algorithmen entwickelt und arbeiten selbst nach dem gleichen, strengen Prinzip. Das soll im Folgenden erläutert werden. Der Kern eines Computers im weitesten Sinn[3] ist der Prozessor.

Prozessoren Mikroprozessoren und Mikrocontroller[4] verarbeiten nur ganz elementare Instruktionen, wie sie in typischen Algorithmen vorkommen. Die meisten Prozessoren kennen Instruktionen wie diese:

- Daten verschieben (Speicher $\longleftrightarrow$ Register, I/O $\longleftrightarrow$ Register, ...).
- Logische Instruktionen (bitweise NOT, AND, OR, XOR, ...).
- Arithmetische Instruktionen (Addition, Subtraktion, Multiplikation, ...).
- Schiebe- und Rotationsbefehle (Registerinhalt nach links oder rechts schieben, ...).
- Sprungbefehle, unbedingt oder bedingt (Aufruf und Beenden von Subroutinen, Sprung an eine Adresse, Sprung in Abhängigkeit eines Flags[5]).
- Evt. einige weitere.

Diese Instruktionen sind Prozessor-Befehle (Assembler-Sprache). Alle Hochsprachen (C/C++, Java, PHP, etc.) werden vom Compiler oder Interpreter in Prozessor-Befehle übersetzt. Der Prozessor selbst kennt *keine* Hochsprach-Elemente (`for, if, then, else,` ...). Diese werden statt dessen aus elementaren Prozessor-Instruktionen zusammen gesetzt.

Beispiel: Wir betrachten die folgende Addition in der Sprache C, wobei a und b initialisierte Variablen seien und p ein Pointer auf eine weitere Variable vom selben Typ.

[3] Die überwältigende Mehrzahl aller Prozessoren wird nicht in klassischen Computern eingesetzt, sondern in sog. Embedded Systemen. Dazu gehören unter anderem alle Arten von alltäglichen Geräten, die deren Innerem Mikroprozessoren arbeiten. Beispiele sind Autos, Küchen- und Haushaltgeräte, elektrische Zahnbürsten, Fotoapparate, Telefone, medizinische Geräte, industrielle Maschinen, usw.

[4] Unter einem Mikrocontroller versteht man einen Mikroprozessor, der mit zusätzlichen Baublöcken kombiniert ist. Mikrocontroller sind in sog. Embedded Systemen weit verbreitet. Nebst dem eigentlichen Prozessorkern können sie zum Beispiel Timer, serielle Schnittstellen, A/D-Wandler, etc. enthalten.

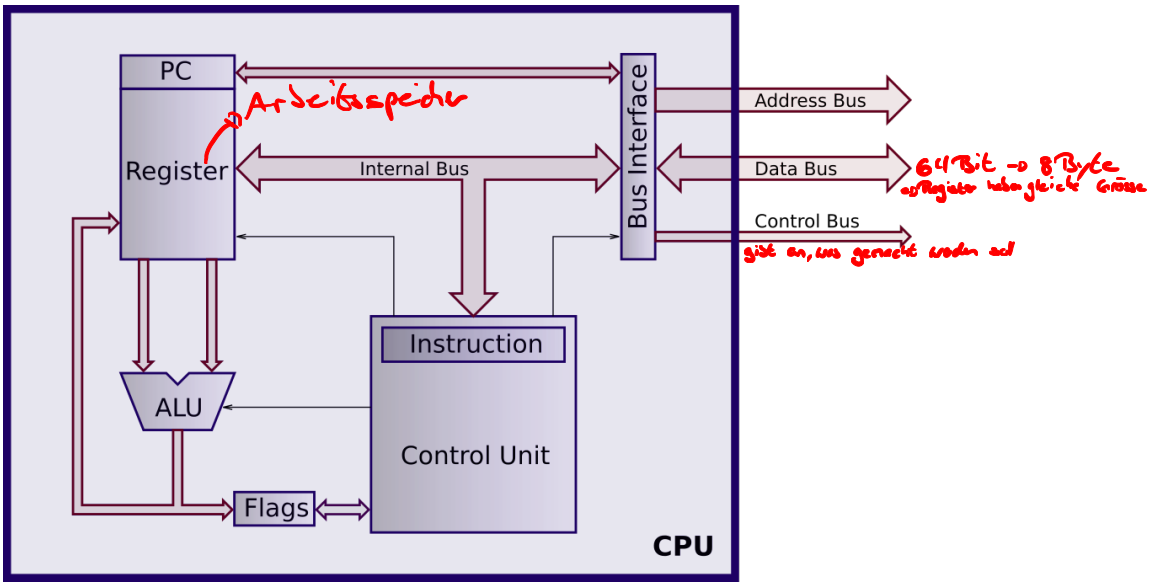
[5] Praktisch jeder Prozessor speichert gewisse Zustände oder Resultate von vorhergehenden Instruktionen in sog. Flags. In Abhängigkeit dieser Flags können bedingte Sprünge ausgeführt werden. Die Flags zeigen zum Beispiel an, ob das Resultat einer arithmetischen Instruktion null wurde, ob bei einer Addition oder Subtraktion ein Überlauf stattgefunden hat, ob die Parität einer Zahl gerade oder ungerade ist, usw.

```
*p = a+b;
```

Der C-Compiler macht daraus die folgende Sequenz von Prozessor-Befehlen:

```
1  Berechne die Adresse der Variablen a.
2  Lade den Wert der Variable a vom Speicher in
   ein internes Register r1.
3  Berechne die Adresse der Variablen b.
4  Lade den Wert der Variable b vom Speicher in
   ein internes Register r2.
5  Berechne die Adresse des Pointers p.
6  Lade den Wert des Pointers p vom Speicher in
   ein internes Register r3.
7  Addiere die Register r1 und r2, speichere das
   Resultat in Register r1.
8  Schreibe den Inhalt von Register r1 an die
   Adresse, die in Register r3 steht.
```

Die folgende Abbildung zeigt die wichtigsten Elemente im Innenleben eines Mikroprozessors. Der Program Counter (PC) adressiert einen Befehl im externen Speicher. Dieser Befehl wird ins Instruction Register in der Control Unit geladen, wo die Ausführung des Befehls gesteuert wird. Arithmetische und logische Instruktionen werden in der ALU (Arithmetic and Logic Unit) ausgeführt. Die Operanden und das Resultat stehen üblicherweise im Register File.



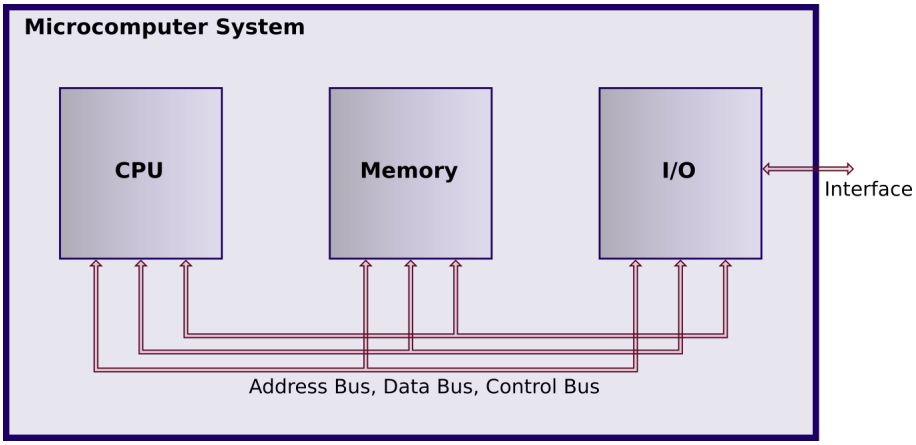
Nach dem Einschalten führt der Prozessor Befehle aus und zwar so lange, bis das System wieder abgeschaltet wird. Der Ablauf eines jeden Befehls ist wie folgt (Prozessor-Algorithmus):

Befehlsausführung im Prozessor

1. Befehl vom Speicher einlesen. Die Adresse steht im Register PC.
2. PC = PC+1
3. Befehl decodieren.
4. Befehl ausführen.
5. Zurück zu Punkt 1.

Prozessorsystem Damit der Prozessor (CPU[6]) arbeiten kann, ist er in einem Prozessorsystem eingebettet. Zu einem Mikroprozessorsystem gehören die folgenden Komponenten:

- Speicher (RAM[7], ROM[8], Flash[9], ...).
- I/O[10] (Disk, USB, Video, Audio, ...).
- Busse[11] (Daten, Adressen, Steuerleitungen).



6 Als CPU (Central Processing Unit) bezeichnet man den Rechenkern eines Computersystems, also in der Regel den Mikroprozessor.

7 Ein RAM (Random Access Memory) ist im Betrieb des Prozessorsystems les- und schreibbar. Der Speicher ist flüchtig. Das heisst, dass alle Daten im Speicher nach einem Spannungsunterbruch verloren sind.

8 Aus einem ROM (Read-Only Memory) kann nur gelesen werden. Der Speicher ist permanent, selbst bei einem Spannungsunterbruch bleiben die Daten im Speicher erhalten.

9 Flash-Speicher sind permanent, behalten ihre Daten also auch über einen Spannungsunterbruch hinweg. Der Inhalt eines Flashs kann aber in einem speziellen Modus gelöscht und neu beschrieben werden. Das ist bei einem klassischen ROM nicht möglich.

10 Unter einer I/O-Komponente (Input/Output) versteht man eine Verbindung mit der Aussenwelt über welche Daten oder Information ausgetauscht werden können. I/Os sind typischerweise Datenverbindungen, Anschlüsse für Bedienungselemente und Anzeigen, usw.

11 Als Bus bezeichnet man eine Datenverbindung, an der mehrere Teilnehmer angeschlossen sein können. Im vorliegenden Fall sind nebst der CPU mindestens Speicher und I/O mit dem Bus verbunden.

Datenformate Darstellung von Daten im Prozessor:

- Binär, 2-er System.
- Bit $b \in \{0, 1\}$, eine Binärziffer.
- Byte, 8 Bit (Oktett), Zahlenbereich 0 bis 255.
  Bei vielen Prozessoren ist das Byte die kleinste adressierbare Einheit.
- Wort, mehrere Bytes, die gemeinsam verarbeitet werden.
  Übliche Beispiele: 16 Bit, 24 Bit, 32 Bit, 64 Bit, 128 Bit.
  Die Wortbreite ist abhängig von der Prozessorarchitektur oder von der Programmiersprache.

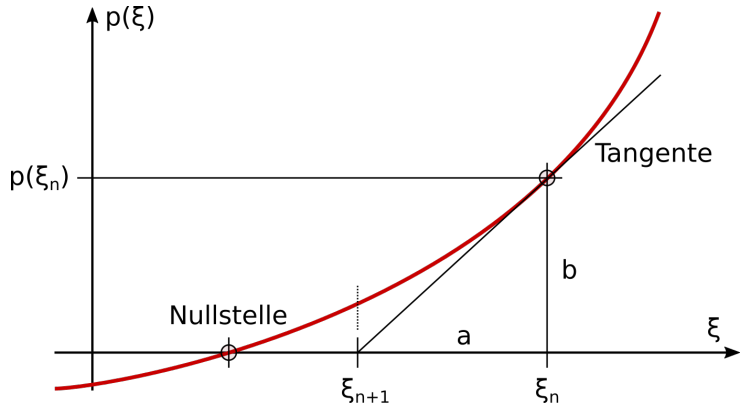
Jeder Prozessor hat eine generische Wortbreite. Sie entspricht in der Regel der Anzahl Leitungen auf dem Datenbus.

Speichergrössen Die Speichergrösse ist gegeben durch die Anzahl Leitungen auf dem Adressbus. Diese definieren, wie viele verschiedene binäre Adressen damit darstellbar sind.

Adressierung	
8 Adressleitungen	2^8 Byte = 256 Bytes
10 Adressleitungen	2^{10} Byte = 1'024 Bytes = 1 kB
16 Adressleitungen	2^{16} Byte = 65'536 Bytes = 64 kB
20 Adressleitungen	2^{20} Byte = 1'048'576 Bytes = 1 MB
30 Adressleitungen	2^{30} Byte = 1'073'741'824 Bytes = 1 GB
32 Adressleitungen	2^{32} Byte = 4'294'967'296 Bytes = 4 GB
40 Adressleitungen	2^{40} Byte = 1'099'511'627'776 Bytes = 1 TB

7 Anhang

Erklärung zum Heron-Verfahren Beim Heron-Verfahren wird auf Grund der Schätzung ξ das Fehlermass $p(\xi) = \xi^2 - X$ bestimmt. Grafisch lässt sich $p(\xi)$ ähnlich wie in der folgenden Abbildung darstellen. Gesucht ist jene Schätzung ξ bei dem das Fehlermass $p(\xi) = 0$ wird.



Wir wenden das Newton-Verfahren an: Es soll für ein gegebenes ξ_n mit dem Fehlermass $p(\xi_n) \neq 0$ eine verbesserte Schätzung ξ_{n+1} ermittelt werden, so dass das Fehlermass $p(\xi_{n+1})$ kleiner wird als $p(\xi_n)$. Gemeinsam bilden ξ_n und $p(\xi_n)$ einen Punkt auf der roten Kurve. In diesem Punkt wird nun eine Tangente an die Kurve gelegt. Die Tangente bildet mit den Strecken a und $b = p(\xi_n)$ ein rechtwinkliges Dreieck. Nun lässt sich die Steigung $s(\xi_n)$ der Tangente berechnen:

$$s(\xi_n) = \frac{b}{a} = p'(\xi_n)$$

Dabei ist $p'(\xi_n)$ die Ableitung von $p(\xi)$ nach ξ im Punkt ξ_n. Sie ist $p'(\xi_n) = 2 \cdot \xi_n$. Damit können wir nun die Strecke a berechnen:

$$a = \frac{b}{p'(\xi_n)} = \frac{p(\xi_n)}{p'(\xi_n)} = \frac{\xi_n^2 - X}{2 \cdot \xi_n}$$

Mit Hilfe der Strecke a kann man jetzt eine neue Schätzung ξ_{n+1} generieren, die näher an der gesuchten Nullstelle $p(\xi) = 0$ der Fehlerfunktion zu liegen kommt. Nämlich:

$$\begin{aligned} \xi_{n+1} &= \xi_n - a \\ &= \xi_n - \frac{\xi_n^2 - X}{2 \cdot \xi_n} = \xi_n - \frac{\xi_n^2}{2 \cdot \xi_n} + \frac{X}{2 \cdot \xi_n} \end{aligned}$$

Und schliesslich:

$$\boxed{\xi_{n+1} = \frac{\xi_n}{2} + \frac{X}{2 \cdot \xi_n}}$$

8 Übungsaufgaben

8.1 Sieb des Eratosthenes

Der im 3. Jh. v. Chr. lebende Universalgelehrte Eratosthenes von Kyrene gab einem schon vorher bekannten Verfahren zum Auffinden von Primzahlen seinen Namen. Die Methode funktioniert so:

Man schreibt, beginnend bei der Zahl 2, alle natürlichen Zahlen je auf eine Karte. Aus praktischen Gründen notiert man natürlich nur endlich viele Zahlen, zum Beispiel bis zu $N = 1000$. Damit findet man dann nur die Primzahlen bis zu dieser Grenze.

Nun ist die 2 die erste (a priori bekannte) Primzahl. Jedes Vielfache von 2 ist demnach keine Primzahl. Man entfernt folglich alle entsprechenden Karten, nämlich 4, 6, 8, 10, 12, etc. Nach der 2 ist 3 die nächste vorhandene Karte. Sie ist eine Primzahl, weil sie noch nicht entfernt wurde. Man entfernt wiederum die Vielfachen 6, 9, 12, 15, usw., wobei einige davon schon vorher ausgeschieden wurden. Die wiederum nächste Karte ist die 5, denn die 4 entfiel bereits. Also ist 5 eine Primzahl und man entfernt von den verbleibenden Karten alle Vielfachen von 5. Das geht so weiter, bis zur Zahl N. Alle Karten, die dann noch übrig sind, enthalten Primzahlen.

In dieser Aufgabe wollen wir das Sieb des Eratosthenes als nicht-rekursiven Algorithmus behandeln.

(a) Beschreiben Sie in Worten den Algorithmus von Eratosthenes.

(b) Beschreiben Sie den Algorithmus von Eratosthenes mit Hilfe einer an C angelehnten formalen Sprache. Verwenden Sie die allgemeine Obergrenze N.

(c) Ist der Algorithmus deterministisch, stochastisch, terminierend?
Begründen Sie Ihre Antworten.

8.2 Fakultät

Die mathematische Fakultät $n!$ für Zahlen $n \geq 1$ ist folgendermassen definiert[12]:

$$n! = \prod_{k=1}^{n} k = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$$

In dieser Aufgaben suchen wir einen rekursiven Algorithmus für die Berechnung der Fakultät $n!$, wobei n positive Ganzzahlen (Integer) seien.

[12] Streng genommen gilt für die Fakultät $n!$ der Bereich $n \geq 0$, wobei $0! = 1$ definiert ist.

(a) Beschreiben Sie den rekursiven Algorithmus für die Fakultät $n!$ in Worten.

(b) Implementieren Sie den rekursiven Algorithmus für die Fakultät $n!$ in C.

(c) Ist der Algorithmus deterministisch, stochastisch, terminierend?
Begründen Sie Ihre Antworten.

(d) Was passiert, wenn Sie mit Ihrer Implementation zum Beispiel 50! berechnen wollen?

8.3 Nicht terminierender Algorithmus

Unter welcher Bedingung ist der Heron-Algorithmus nicht terminierend?