Algorithmen

Definition

Ein Algorithmus ist eine präzise (d.h. in einer festgelegten Sprache) verfasste endliche Beschreibung eines allgemeinen Verfahrens unter Verwendung ausführbarer (Verarbeitungs-) Schritte.

Im Alltag sind Algorithmen z.B.

- Bedienungsanleitungen
- Bauanleitungen
- Kochrezepten

Terminierung:
- Ein Algorithmus heißt terminierend, wenn er (bei jeder erlaubten Eingabe) nach endlich vielen Schritten abbricht.

Determinismus:
- Ein deterministischer Ablauf bedeutet, dass zu jedem Zeitpunkt der jeweils nächste Schritt eindeutig bestimmt ist

- Ein Algorithmus liefert ein determiniertes Ergebnis, wenn er zu denselben Eingaben stets dieselben Ausgaben erzeugt

Wichtige Klasse von Algorithmen:
Terminierende Algorithmen mit determiniertem Ergebnis. Diese definieren jeweils eine Ein- und Ausgabefunktion:

f: Eingabewerte → Ausgabewerte

Beispiel: Addition zweier Binärzahlen

Text Beschreibung:
- Es werden jeweils die Bit-Stellen mit gleicher Wertigkeit addiert, beginnend bei der Stelle mit der niedrigsten Wertigkeit

- Bei jeder Stellen-Addition eine Ergebnis-Stelle und einen Übertrag (für die nächsthöhere Bit-Stelle) erzeugen

Algebraische Beschreibung:

$$S_i = A_i \neq B_i \neq C_i$$
$$C_{i+1} = C_i \cdot (A_i \neq B_i) + (A_i \cdot B_i)$$

Nicht - deterministischer Ablauf:

Suchen nach einer Karte in einem Kartenstapel:

1. Teile den Kartenstapel willkürlich in zwei Stapel auf 1. Teile den Kartenstapel willkürlich in zwei Stapel auf

2. Suche entweder zuerst in dem einen oder in dem anderen Stapel nach der Karte

3. Wenn die Karte nicht gefunden wurde, dann durchsuche den noch nicht durchsuchten Stapel

→ Das Ergebnis ist determiniert (die Karte wird auf jeden Fall gefunden)

Nicht - determiniertes Ergebnis:

- Ziehe zufällig eine Karte aus dem Stapel

WICHTIG

- Nicht jedes Problem ist durch einen Algorithmus lösbar (Berechenbarkeit)

- Nicht jedes durch einen Algorithmus lösbare Problem kann bei vertretbarem Aufwand (Rechenzeit und Speicherplatz) mit Hilfe eines Computers umgesetzt werden.

Elementare Operationen
Operationen, die nicht weiter aufgeschlüsselt werden (z.B. schneide Fleisch in Würfel)

Sequentielle Ausführung
Hintereinander-Ausführung von Sequentielle Ausführung: Hintereinander Ausführung von Schritten (z.B. Bringe Wasser zum Kochen, dann gib Nudeln hinzu)

Parallele Ausführung
Gleichzeitige Ausführung von Schritten (z.B. Ich schneide das Fleisch, Du das Gemüse)

Bedingte Ausführung
Ein Schritt wird nur ausgeführt, wenn eine Bedingung erfüllt ist (z.B. Wenn Soße zu dünn, dann füge Mehl hinzu)

Schleife
Wiederholung von Schritten bis Endbedingung erfüllt ist

Unterprogramme
Eine Bearbeitungsvorschrift, die aufgerufen wird. Nach der Ausführung des Unterprogramms führt der Algorithmus an der Stelle fort, an der zum Unterprogramm gewechselt wurde. (z.B. Bereite Soße zu nach Rezept auf Seite 42)

Rekursion
Anwendung desselben Prinzips auf kleinere oder einfachere Teilprobleme. Solange, bis diese direkt gelöst werden können.

Notationen für Algorithmen:

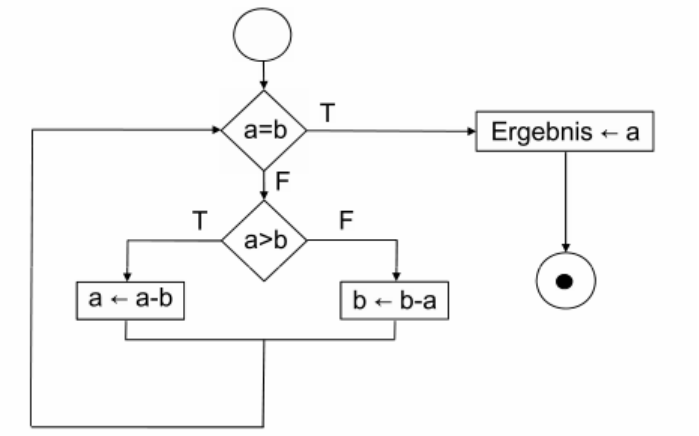
- Flussdiagramme
- Struktogramme
- Pseudocode

Flussdiagramme

Elemente des Flussdiagramms:

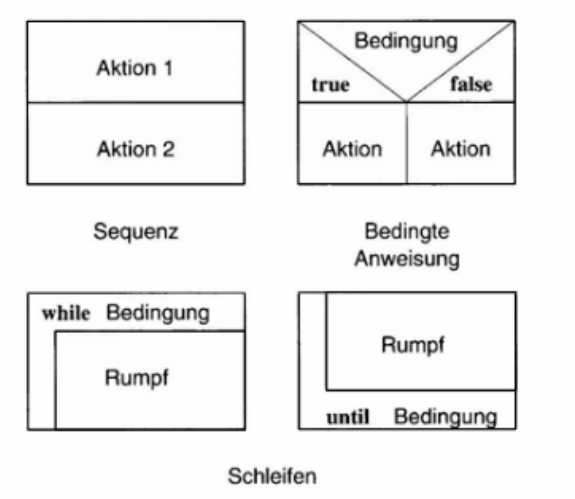
- Anfang (Kreis): Hier beginnt die Ausführung des Algorithmus
- elementare Operation (Rechteck)
- Schrittfolge (Pfeil)
- Bedingung (Karo mit zwei Abzweigungen (T=True (Bedingung erfüllt), F=False(Bedingung nicht erfüllt))
- Unterprogramm (Recheck mit doppelter Seitenlinie)
- Schleife: Wird durch Pfeile zwischen Ende und Start der Schleife realisiert
- Ende (Kreis mit Punkt): Hier ist der Algorithmus beendet

Bsp.: Euklidscher Algorithmus für den ggT (größter gemeinsamer Teiler)



Struktogramme

- Elementare Aktionen entsprechen Rechtecken
- Konstrukte können ineinander verschachtelt werden



Pseudocode

Der Pseudocode ist wie der Quelltext eines Computerprogramms strukturiert. Er besitzt genau wie alle gängigen Programmiersprachen bestimmte Anweisungen. Beendet werden diese immer mit dem Anweisungsnamen rückwärts geschrieben. Ausnahme sind Programmaufrufe, sie enden mit end.

1. Bedingung(1)

```
if <Bedingung> then
Schritt
fi
```

2. Bedingung(2)

```
if <Bedingung> then
SchrittA
else
SchrittB
fi
```

3. Schleifen (Post – Checked, die Bedingung wird am Ende geprüft)

```
do
SchrittA
SchrittB
od while <Bedingung>
```

4. Schleifen (Pre – Checked, die Bedingung wird am Anfang geprüft)

```
do while <Bedingung>
SchrittA SchrittA
SchrittB
od
```

5. Schleifen (Zählschleife, Durchlauf wird wiederholt, bis Wert erreicht ist)

```
for <Initialisierung> to <Maximalwert> do
SchrittA
SchrittB
od
```

6. Unterprogramme (Rekursion) (→ s. Übungsaufgabe 22)

```
function <Name> (<Parameter1>,<Parameter2>,…)
SchrittA
SchrittB
End
```

7. Funktionen (Unterprogramme, die Werte zurückgeben)
   (→ s. Übungsaufgabe 23,24)

```
function <Ergebnis> = <Name> (<Parameter1>,…)
SchrittA
SchrittB
<Ergebnis> = <Wert>
end
```

<u>Die Türme von Hanoi als Beispiel einer rekursiven Funktion</u>

Spielregeln:

- Es darf nur die jeweils oberste Scheibe einzeln bewegt werden
- Es darf niemals eine größere Scheibe auf eine kleinere gelegt werden

Die Vereinfachung des Problems ist mit Hilfe einer Rekursion möglich. Stellt man sich vor, dass nur eine Scheibe zu bewegen ist, so kann man jene von Start direkt auf Ziel bewegen.

Jenes kann man direkt in Pseudocode schreiben als:

```
if  Anzahl = 1
     Bewege Scheibe von Start auf Ziel
```

Nun muss man sich überlegen, wie man darstellt, dass 2 Scheiben bewegt werden. Dazu würde man folgende Spielzüge brauchen:

1. Scheibe 1 von Start auf Ablage
2. Scheibe 2 von Start auf Ziel
3. Scheibe 1 von Ablage auf Ziel

Als nächstes muss man sich überlegen, wie man dem Programm mitteilt, wie es die Scheiben zu bewegen hat. Dazu muss man es in zwei Ebenen teilen. Ruft man es auf, wird zunächst geprüft, ob nur eine Ebene vorhanden ist. Dazu benutzt man die if Anweisung. Ist diese falsch, wird durch einen erneuten Programmaufruf eine neue Ebene angelegt. In dieser ist die if- Anweisung dann richtig und die entsprechende Scheibe wird bewegt.
Wie diese bewegt wird, entnimmt man aus dem Ablauf.
Der Programmtext, der genutzt wird, sieht daher folgendermaßen aus

```
function Turmbewegung (Anzahl)
      if Anzahl = 1
            Bewege Scheibe von Start auf Ziel
      else
            function Turmbewegung(Anzahl - 1)
            (…)
      fi
end
```

Um jetzt zwischen den Funktionsaufrufen zu unterscheiden, nutzt man Variablen. Das ist in diesem Fall wichtig, weil das Unterprogramm `function Turmbewegung(Anzahl - 1)`, beim ersten Aufruf die Scheibe 1 ja nicht von Start auf Ziel, sondern von Start auf Ablage legen soll. Daher erweitert man den Pseudocode um die drei Variablen.

```
function Turmbewegung (Anzahl, Start, Ziel, Ablage)
      if Anzahl = 1
            Bewege Scheibe von Start auf Ziel
      else
            function Turmbewegung(Anzahl - 1)
            (…)
      fi
end
```

Ruft man jetzt das Programm auf, so werden die Variablen so gesetzt, wie sie vorhanden sind, d.h. Start ist Start, Ablage ist Ablage und Ziel ist Ziel. Die if- Anweisung wird übersprungen, jetzt wird das Unterprogramm aufgerufen. Hier müssen jetzt die Variablen so gesetzt werden, dass sie im neuen Programmtext Start für Start und Ablage für Ziel liefern:

```
function Turmbewegung (Anzahl, Start, Ziel, Ablage)
      if Anzahl = 1
            Bewege Scheibe von Start auf Ziel
      else
            function Turmbewegung(Anzahl - 1, Start, Ablage, Ziel)
            (…)
      fi
end
```

Würde das Unterprogramm jetzt ablaufen, würde folgendes passieren: Die if Anweisung ist richtig, weil nur eine Scheibe da ist. Jetzt wird diese von Start auf Ziel gelegt, wobei Ziel hier durch den Variablentausch die Ablage ist. Das Unterprogramm wäre damit beendet.

Als nächstes soll Scheibe 2 von Start auf Ziel gelegt werden. Es muss also in der oberen Ebene, in der die Anzahl = 2 ist, passieren. Start und Ziel sind durch die Variablen bereits richtig zugewiesen, daher kann die Anweisung direkt in den Programmtext geschrieben werden.

```
function Turmbewegung (Anzahl, Start, Ziel, Ablage)
      if Anzahl = 1
            Bewege Scheibe von Start auf Ziel
      else
            function Turmbewegung(Anzahl - 1, Start, Ablage, Ziel)
            Bewege Scheibe von Start auf Ziel
            (…)
      fi
end
```

Der dritte und letzte Zug findet wieder auf der zweiten Ebene statt und wird durch ein Unterprogramm umgesetzt. Hierbei verfährt man wie beim ersten Unterprogramm, indem man sich überlegt, wie die Scheibe bewegt wird (hier von Ablage auf Ziel) und die Variablen dementsprechend übergibt. Es wird also Start zu Ablage und Ziel bleibt Ziel.

```
function Turmbewegung (Anzahl, Start, Ziel, Ablage)
      if Anzahl = 1
            Bewege Scheibe von Start auf Ziel
      else
            function Turmbewegung(Anzahl - 1, Start, Ablage, Ziel)
            Bewege Scheibe von Start auf Ziel
            function Turmbewegung(Anzahl - 1, Ablage, Ziel, Start)
      fi
end
```

Jetzt führt das Programm die Schritte entsprechend den Vorgaben für eine und zwei Scheiben aus. Setzt man eine größere Anzahl ein, werden solange Unterprogramme ausgeführt, bis die Ebene Anzahl = 1 erreicht ist.

Anmerkung: Der Algorithmus für Anzahl = 3 findet sich in einer Extra txt-Datei, weil er von der Breite her nicht in dieses Dokument passt.