Vorlesung
Algorithmen und Datenstrukturen

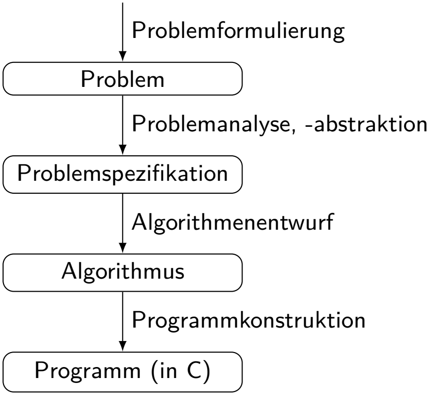
Prof. Dr.-Ing. habil. Heiko Vogler
Lehrstuhl für Grundlagen der Programmierung
Institut für Theoretische Informatik
Fakultät Informatik
Technische Universität Dresden

13. Mai 2022

Inhaltsverzeichnis I

Inhaltsverzeichnis II

Vom Problem zum Programm – Ein Überblick



Ein einfaches Beispiel

Problemformulierung

Problem

Suche die jüngste Person im Raum.

Ein einfaches Beispiel

Problemformulierung

Problem
Suche die jüngste Person im Raum.

Problemanalyse, -abstraktion

- Gibt es überhaupt eine Lösung?
- Wenn es eine Lösung gibt, gibt es eine eindeutige Lösung?
- Wie soll eine Lösung aussehen?
- Wie sollen die Personen bei der Berechnung einer Lösung repräsentiert werden?
- Wie genau soll das Alter einer Person gezählt werden?
- Was passiert, wenn zwei Personen gleich alt sind?

Ein einfaches Beispiel

Abstraktion

1. Jede Person wird durch eine natürliche Zahl i repräsentiert.

Ein einfaches Beispiel

Abstraktion

1. Jede Person wird durch eine natürliche Zahl i repräsentiert.
2. Als Alter rechnen wir nur das Alter in Jahren; jede Person i hat also ein Alter a_i (positive, ganze Zahl).

Ein einfaches Beispiel

Abstraktion

1. Jede Person wird durch eine natürliche Zahl i repräsentiert.
2. Als Alter rechnen wir nur das Alter in Jahren; jede Person i hat also ein Alter a_i (positive, ganze Zahl).
3. Die Zahlen $a_1, a_2, ..., a_n$ sind bekannt.

Ein einfaches Beispiel

Abstraktion

1. Jede Person wird durch eine natürliche Zahl i repräsentiert.
2. Als Alter rechnen wir nur das Alter in Jahren; jede Person i hat also ein Alter a_i (positive, ganze Zahl).
3. Die Zahlen $a_1, a_2, ..., a_n$ sind bekannt.
4. „Jüngste“ Person ist eine Person i, falls für jede Person j gilt: $a_i \leq a_j$.

Ein einfaches Beispiel

Abstraktion

1. Jede Person wird durch eine natürliche Zahl i repräsentiert.
2. Als Alter rechnen wir nur das Alter in Jahren; jede Person i hat also ein Alter a_i (positive, ganze Zahl).
3. Die Zahlen $a_1, a_2, ..., a_n$ sind bekannt.
4. „Jüngste" Person ist eine Person i, falls für jede Person j gilt: $a_i \leq a_j$.
5. Wenn für zwei Personen i und j gilt: $a_i = a_j$, dann wollen wir als Lösung die bzgl. der Folge $a_1, a_2, ..., a_n$ erste Person mit Alter a_i als Lösung angeben.

Ein einfaches Beispiel

Abstraktion

1. Jede Person wird durch eine natürliche Zahl i repräsentiert.
2. Als Alter rechnen wir nur das Alter in Jahren; jede Person i hat also ein Alter a_i (positive, ganze Zahl).
3. Die Zahlen $a_1, a_2, ..., a_n$ sind bekannt.
4. „Jüngste" Person ist eine Person i, falls für jede Person j gilt: $a_i \leq a_j$.
5. Wenn für zwei Personen i und j gilt: $a_i = a_j$, dann wollen wir als Lösung die bzgl. der Folge $a_1, a_2, ..., a_n$ erste Person mit Alter a_i als Lösung angeben.

Problemspezifikation

Gegeben Folge $a_1, a_2, ..., a_n$ von ganzen, positiven Zahlen

Gesucht der kleinste Positionsindex j mit $a_j = \min\{a_1, ..., a_n\}$

Algorithmenentwurf

Ein Algorithmus ist eine (Rechen-)Vorschrift zur Lösung eines Problems.

Algorithmenentwurf

Ein Algorithmus ist eine (Rechen-)Vorschrift zur Lösung eines Problems.

Ein Algorithmus hat folgende Eigenschaften:

Algorithmenentwurf

Ein Algorithmus ist eine (Rechen-)Vorschrift zur Lösung eines Problems.

Ein Algorithmus hat folgende Eigenschaften:

- muss so präzise formuliert sein, dass er im Prinzip maschinell ausgeführt werden kann,

Algorithmenentwurf

Ein Algorithmus ist eine (Rechen-)Vorschrift zur Lösung eines Problems.

Ein Algorithmus hat folgende Eigenschaften:

- muss so präzise formuliert sein, dass er im Prinzip maschinell ausgeführt werden kann,
- ist ein *abstraktes* Objekt,

Algorithmenentwurf

Ein Algorithmus ist eine (Rechen-)Vorschrift zur Lösung eines Problems.

Ein Algorithmus hat folgende Eigenschaften:

- muss so präzise formuliert sein, dass er im Prinzip maschinell ausgeführt werden kann,
- ist ein *abstraktes* Objekt,
- ist unabhängig von der Programmiersprache, in der er geschrieben werden soll,

Algorithmenentwurf

Ein Algorithmus ist eine (Rechen-)Vorschrift zur Lösung eines Problems.

Ein Algorithmus hat folgende Eigenschaften:

- muss so präzise formuliert sein, dass er im Prinzip maschinell ausgeführt werden kann,
- ist ein *abstraktes* Objekt,
- ist unabhängig von der Programmiersprache, in der er geschrieben werden soll,
- ist unabhängig vom Computertyp oder der verwendeten Rechnertechnologie,

Algorithmenentwurf

Ein Algorithmus ist eine (Rechen-)Vorschrift zur Lösung eines Problems.

Ein Algorithmus hat folgende Eigenschaften:

- muss so präzise formuliert sein, dass er im Prinzip maschinell ausgeführt werden kann,
- ist ein *abstraktes* Objekt,
- ist unabhängig von der Programmiersprache, in der er geschrieben werden soll,
- ist unabhängig vom Computertyp oder der verwendeten Rechnertechnologie,
- ist durch einen endlichen Text beschrieben (*Finitheit*),

Algorithmenentwurf

Ein Algorithmus ist eine (Rechen-)Vorschrift zur Lösung eines Problems.

Ein Algorithmus hat folgende Eigenschaften:

- muss so präzise formuliert sein, dass er im Prinzip maschinell ausgeführt werden kann,
- ist ein *abstraktes* Objekt,
- ist unabhängig von der Programmiersprache, in der er geschrieben werden soll,
- ist unabhängig vom Computertyp oder der verwendeten Rechnertechnologie,
- ist durch einen endlichen Text beschrieben (*Finitheit*),
- läuft in einzelnen, wohldefinierten Schritten ab (*Effektivität*),

Algorithmenentwurf

Ein Algorithmus ist eine (Rechen-)Vorschrift zur Lösung eines Problems.

Ein Algorithmus hat folgende Eigenschaften:

- muss so präzise formuliert sein, dass er im Prinzip maschinell ausgeführt werden kann,
- ist ein *abstraktes* Objekt,
- ist unabhängig von der Programmiersprache, in der er geschrieben werden soll,
- ist unabhängig vom Computertyp oder der verwendeten Rechnertechnologie,
- ist durch einen endlichen Text beschrieben (*Finitheit*),
- läuft in einzelnen, wohldefinierten Schritten ab (*Effektivität*),
- nach Ausführung jedes Schrittes ist eindeutig festgelegt, welcher Schritt als nächster ausgeführt wird (*Determiniertheit*) und

Algorithmenentwurf

Ein Algorithmus ist eine (Rechen-)Vorschrift zur Lösung eines Problems.

Ein Algorithmus hat folgende Eigenschaften:

- muss so präzise formuliert sein, dass er im Prinzip maschinell ausgeführt werden kann,
- ist ein *abstraktes* Objekt,
- ist unabhängig von der Programmiersprache, in der er geschrieben werden soll,
- ist unabhängig vom Computertyp oder der verwendeten Rechnertechnologie,
- ist durch einen endlichen Text beschrieben (*Finitheit*),
- läuft in einzelnen, wohldefinierten Schritten ab (*Effektivität*),
- nach Ausführung jedes Schrittes ist eindeutig festgelegt, welcher Schritt als nächster ausgeführt wird (*Determiniertheit*) und
- kommt bei jeder Eingabe in endlich vielen Schritten zu einem Ende (*Terminiertheit*).

Algorithmus MinAlter

Eingabe Eine Folge $a_1, \dots, a_n$ von positiven, ganzen Zahlen.

Ausgabe der kleinste Positionsindex j mit $a_j = \min\{a_1, \dots, a_n\}$.

Verfahren Zusätzliche Variablen: x (für das Alter), i (als Zählvariable);

1. (*Initialisierung*) Setze $j := 1, x := a_j$ und $i := 2$.
2. (*Suchlauf*)
   Solange $i \leq n$ gilt, wiederhole:
   - falls $a_i < x$, setze $j := i$ und $x := a_j$
   - erhöhe i um 1
3. Ausgabe von j als Ergebnis

Programmkonstruktion

anstehende Frage	unsere Antwort
Verfügbare Programmiersprachen?	*C*
Wie kommen Daten in den Rechner?	Altersangaben müssen eingegeben werden.
Durch welche Datenstrukturen werden Daten repräsentiert?	Die Folge $a_1, ..., a_n$ der Zahlen wird in einem Array (Feld) gespeichert.
Wie wird das Ergebnis ausgegeben?	Ausgabe als verständlicher Text.
Wie wird das Programm gegliedert?	Die Operation „Finde Person mit kleinstem Alter“ wird als Funktion beschrieben.

```c
/* JuengstePerson */
#include <stdio.h>

/* maximale Personenzahl */
#define MAX 100
int a[MAX];
int i, j, n;

/* liefert Minimalindex
   im Feld a[0]..a[n-1] */
int MinIndex(int a[], int n)
{ int i, j, x;
  j = 0; x = a[j]; i = 1;
  while (i < n)
  { if  (a[i] < x)
    { j = i;
      x = a[j];
    }
    i = i + 1;
  }
  return j;
}

int main()
{ printf("Anzahl der Personen? ");
  do scanf("%d",&n);
  while ((n < 1) || (n > MAX));

  i = 0;
  while (i < n)
  { printf("Alter der %d. Person? ", i + 1);
    scanf("%d", &a[i]);
    i = i + 1;
  }
  j = MinIndex(a, n);
  printf("\n\n");
  printf("Juengste Person ist Nr. %d\n", j + 1);

  return 0;
}
```

Inhaltsverzeichnis I

Inhaltsverzeichnis II

Sprachen

	Objektsprache	Element der Objektsprache
	natürliche Sprache (z. B. Deutsch)	$\ni$ Satz (z. B. Der Hund läuft schnell.)

Sprachen

Metasprache (Syntax-beschreibungssprache)	Objektsprache	Element der Objektsprache
Grammatik der deutschen Sprache	natürliche Sprache (z. B. Deutsch)	$\ni$ Satz (z. B. Der Hund läuft schnell.)

Sprachen

Metasprache (Syntaxbeschreibungssprache)	Objektsprache	Element der Objektsprache
Grammatik der deutschen Sprache	natürliche Sprache (z. B. Deutsch)	$\ni$ Satz (z. B. Der Hund läuft schnell.)
	Programmiersprache (z. B. C)	$\ni$ Programm (z. B. `/* JuengstePerson */ ...`)

Sprachen

Metasprache (Syntax-beschreibungssprache)	Objektsprache	Element der Objektsprache
Grammatik der deutschen Sprache	natürliche Sprache (z. B. Deutsch)	$\ni$ Satz (z. B. Der Hund läuft schnell.)
- EBNF - Syntaxdiagramme	Programmiersprache (z. B. C)	$\ni$ Programm (z. B. `/* JuengstePerson */ ...`)

Sprachen

Metasprache (Syntax-beschreibungssprache)	Objektsprache	Element der Objektsprache
Grammatik der deutschen Sprache	natürliche Sprache (z. B. Deutsch)	$\ni$ Satz (z. B. Der Hund läuft schnell.)
- EBNF - Syntaxdiagramme	Programmiersprache (z. B. C)	$\ni$ Programm (z. B. `/* JuengstePerson */ ...`)
	formale Sprache (z. B. $L = \{a^n b \mid n \geq 0\}$)	$\ni$ Wort (z. B. $w = aaab$)

Sprachen

Metasprache (Syntax-beschreibungssprache)	Objektsprache	Element der Objektsprache
Grammatik der deutschen Sprache	natürliche Sprache (z. B. Deutsch)	$\ni$ Satz (z. B. Der Hund läuft schnell.)
- EBNF - Syntaxdiagramme	Programmiersprache (z. B. C)	$\ni$ Programm (z. B. `/* JuengstePerson */ ...`)
- Grammatik Typ 2 - Kellerautomaten	formale Sprache (z. B. $L = \{a^n b \mid n \geq 0\}$)	$\ni$ Wort (z. B. $w = aaab$)

Definitionen

Alphabet: nichtleere, endliche Menge (von Symbolen)

Definitionen

Alphabet: nichtleere, endliche Menge (von Symbolen)

Sei Σ ein Alphabet.

Wort über Σ: endliche Folge von Symbolen aus Σ, beliebige Länge $k \geq 0$

Definitionen

Alphabet: nichtleere, endliche Menge (von Symbolen)

Sei Σ ein Alphabet.

Wort über Σ: endliche Folge von Symbolen aus Σ, beliebige Länge $k \geq 0$

leeres Wort: bezeichnet durch ε, Länge 0

Definitionen

Alphabet: nichtleere, endliche Menge (von Symbolen)

Sei Σ ein Alphabet.

Wort über Σ: endliche Folge von Symbolen aus Σ, beliebige Länge $k \geq 0$

leeres Wort: bezeichnet durch ε, Länge 0

Menge aller Wörter über Σ: mit Σ^* bezeichnet

Definitionen

Konkatenation: zweistellige Operation der Zeichenreihenverkettung,
Abbildung $\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$ für jedes $u, v \in \Sigma^*$ definiere

$$u \cdot v = u_1 u_2 \ldots u_m v_1 v_2 \ldots v_n$$

falls $m, n \geq 0$, $u = u_1 u_2 \ldots u_m$, $v = v_1 v_2 \ldots v_n$, $u_i \in \Sigma$ für $1 \leq i \leq m$, und $v_j \in \Sigma$ für $1 \leq j \leq n$.

Definitionen

Konkatenation: zweistellige Operation der Zeichenreihenverkettung,
Abbildung $\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$ für jedes $u, v \in \Sigma^*$ definiere

$$u \cdot v = u_1 u_2 \ldots u_m v_1 v_2 \ldots v_n$$

falls $m, n \geq 0$, $u = u_1 u_2 \ldots u_m$, $v = v_1 v_2 \ldots v_n$, $u_i \in \Sigma$ für $1 \leq i \leq m$, und $v_j \in \Sigma$ für $1 \leq j \leq n$.

n-te Potenz eines Wortes w: w^n mit $w^0 = \varepsilon$ und $w^{n+1} = w^n \cdot w$ $(n \geq 0)$

Definitionen

Konkatenation: zweistellige Operation der Zeichenreihenverkettung, Abbildung $\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$ für jedes $u, v \in \Sigma^*$ definiere

$$u \cdot v = u_1 u_2 \dots u_m v_1 v_2 \dots v_n$$

falls $m, n \geq 0$, $u = u_1 u_2 \dots u_m$, $v = v_1 v_2 \dots v_n$, $u_i \in \Sigma$ für $1 \leq i \leq m$, und $v_j \in \Sigma$ für $1 \leq j \leq n$.

n-te Potenz eines Wortes w: w^n mit $w^0 = \varepsilon$ und $w^{n+1} = w^n \cdot w$ $(n \geq 0)$

(formale) Sprache L (über Σ): Menge von Wörtern über Σ, d. h. $L \in \mathcal{P}(\Sigma^*)$

Definitionen

Konkatenation: zweistellige Operation der Zeichenreihenverkettung, Abbildung $\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$ für jedes $u, v \in \Sigma^*$ definiere

$$u \cdot v = u_1 u_2 \dots u_m v_1 v_2 \dots v_n$$

falls $m, n \geq 0$, $u = u_1 u_2 \dots u_m$, $v = v_1 v_2 \dots v_n$, $u_i \in \Sigma$ für $1 \leq i \leq m$, und $v_j \in \Sigma$ für $1 \leq j \leq n$.

n-te Potenz eines Wortes w: w^n mit $w^0 = \varepsilon$ und $w^{n+1} = w^n \cdot w$ $(n \geq 0)$

(formale) Sprache L (über Σ): Menge von Wörtern über Σ, d. h. $L \in \mathcal{P}(\Sigma^*)$

Seien L_1 und L_2 Sprachen.

Konkatenation (oder Komplexprodukt) von L_1 und L_2: $L_1 \cdot L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$
Beachte: $\emptyset \cdot L = \emptyset = L \cdot \emptyset$ und $L \cdot \{\varepsilon\} = \{\varepsilon\} \cdot L = L$

Definitionen

Konkatenation: zweistellige Operation der Zeichenreihenverkettung,
Abbildung $\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$ für jedes $u, v \in \Sigma^*$ definiere

$$u \cdot v = u_1 u_2 \dots u_m v_1 v_2 \dots v_n$$

falls $m, n \geq 0$, $u = u_1 u_2 \dots u_m$, $v = v_1 v_2 \dots v_n$, $u_i \in \Sigma$ für $1 \leq i \leq m$, und $v_j \in \Sigma$ für $1 \leq j \leq n$.

n-te Potenz eines Wortes w: w^n mit $w^0 = \varepsilon$ und $w^{n+1} = w^n \cdot w$ $(n \geq 0)$

(formale) Sprache L (über Σ): Menge von Wörtern über Σ, d. h. $L \in \mathcal{P}(\Sigma^*)$

Seien L_1 und L_2 Sprachen.

Konkatenation (oder Komplexprodukt) von L_1 und L_2: $L_1 \cdot L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$
Beachte: $\emptyset \cdot L = \emptyset = L \cdot \emptyset$ und $L \cdot \{\varepsilon\} = \{\varepsilon\} \cdot L = L$

Sei L eine Sprache.

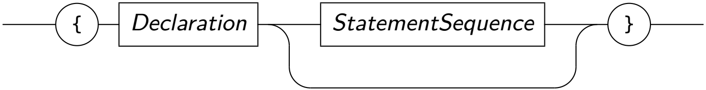
Stern von L: $L^* = \bigcup_{n \geq 0} L^n$, wobei $L^0 = \{\varepsilon\}$, $L^{n+1} = L^n \cdot L$ für jedes $n \geq 0$.
Beachte: $\emptyset^* = \{\varepsilon\}$.

Sprachen

Metasprache (Syntax-beschreibungssprache)	Objektsprache	Element der Objektsprache
Grammatik der deutschen Sprache	natürliche Sprache (z. B. Deutsch)	$\ni$ Satz (z. B. Der Hund läuft schnell.)
- EBNF - Syntaxdiagramme	Programmiersprache (z. B. C)	$\ni$ Programm (z. B. `/* JuengstePerson */ ...`)
- Grammatik Typ 2 - Kellerautomaten	formale Sprache (z. B. $L = \{a^n b \mid n \geq 0\}$)	$\ni$ Wort (z. B. $w = aaab$)

Ein praktisches Beispiel

Block



Aufbau

Ein Syntaxdiagramm besteht aus:

- Ovalen
- Kästchen
- Verbindungen, die aus folgenden Bestandteilen zusammengesetzt sind:
  - Linien (evtl. gebogen)
  - Verzweigungen und Zusammenfassungen.

Aufbau

1. Jedes Syntaxdiagramm hat einen eindeutigen Namen; der Name ist eine syntaktische Variable.

Aufbau

1. Jedes Syntaxdiagramm hat einen eindeutigen Namen; der Name ist eine syntaktische Variable.
2. Jedes Kästchen ist mit dem Namen eines Syntaxdiagrammes beschriftet.

Aufbau

1. Jedes Syntaxdiagramm hat einen eindeutigen Namen; der Name ist eine syntaktische Variable.
2. Jedes Kästchen ist mit dem Namen eines Syntaxdiagrammes beschriftet.
3. Jedes Oval ist mit einem Terminalsymbol beschriftet.

Aufbau

1. Jedes Syntaxdiagramm hat einen eindeutigen Namen; der Name ist eine syntaktische Variable.
2. Jedes Kästchen ist mit dem Namen eines Syntaxdiagrammes beschriftet.
3. Jedes Oval ist mit einem Terminalsymbol beschriftet.
4. An jedem Kästchen und an jedem Oval enden genau zwei Linien.

Aufbau

1. Jedes Syntaxdiagramm hat einen eindeutigen Namen; der Name ist eine syntaktische Variable.
2. Jedes Kästchen ist mit dem Namen eines Syntaxdiagrammes beschriftet.
3. Jedes Oval ist mit einem Terminalsymbol beschriftet.
4. An jedem Kästchen und an jedem Oval enden genau zwei Linien.
5. Es gibt genau einen Strich, der als Anfang markiert ist.

Aufbau

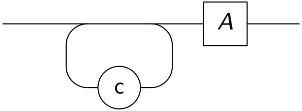
1. Jedes Syntaxdiagramm hat einen eindeutigen Namen; der Name ist eine syntaktische Variable.
2. Jedes Kästchen ist mit dem Namen eines Syntaxdiagrammes beschriftet.
3. Jedes Oval ist mit einem Terminalsymbol beschriftet.
4. An jedem Kästchen und an jedem Oval enden genau zwei Linien.
5. Es gibt genau einen Strich, der als Anfang markiert ist.
6. Es gibt genau einen Strich, der als Ende markiert ist.

Aufbau

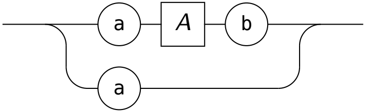
1. Jedes Syntaxdiagramm hat einen eindeutigen Namen; der Name ist eine syntaktische Variable.
2. Jedes Kästchen ist mit dem Namen eines Syntaxdiagrammes beschriftet.
3. Jedes Oval ist mit einem Terminalsymbol beschriftet.
4. An jedem Kästchen und an jedem Oval enden genau zwei Linien.
5. Es gibt genau einen Strich, der als Anfang markiert ist.
6. Es gibt genau einen Strich, der als Ende markiert ist.
7. Linien dürfen sich nicht kreuzen.

Ein weiteres Beispiel

S



A



Algorithmus Rücksprungalgorithmus

Algorithmus Rücksprungalgorithmus

1. Beginne am Eingang des ersten Syntaxdiagramms von U (Startdiagramm).

Algorithmus Rücksprungalgorithmus

1. Beginne am Eingang des ersten Syntaxdiagramms von U (Startdiagramm).
2. Folge den Linien auf einem legalen Weg.
   - Falls dabei der Ausgang erreicht wird, gehe nach Punkt 5.
   - Falls ein Kästchen bzw. Oval erreicht wird, gehe nach Punkt 3.

Algorithmus Rücksprungalgorithmus

1. Beginne am Eingang des ersten Syntaxdiagramms von U (Startdiagramm).
2. Folge den Linien auf einem legalen Weg.
   - Falls dabei der Ausgang erreicht wird, gehe nach Punkt 5.
   - Falls ein Kästchen bzw. Oval erreicht wird, gehe nach Punkt 3.
3. - Falls es sich um ein Oval handelt, notiere das darin enthaltene Terminalzeichen und gehe anschließend zu Punkt 2 zurück.
   - Andernfalls gehe nach Punkt 4.

Algorithmus Rücksprungalgorithmus

1. Beginne am Eingang des ersten Syntaxdiagramms von U (Startdiagramm).
2. Folge den Linien auf einem legalen Weg.
   - Falls dabei der Ausgang erreicht wird, gehe nach Punkt 5.
   - Falls ein Kästchen bzw. Oval erreicht wird, gehe nach Punkt 3.
3. - Falls es sich um ein Oval handelt, notiere das darin enthaltene Terminalzeichen und gehe anschließend zu Punkt 2 zurück.
   - Andernfalls gehe nach Punkt 4.
4. - Falls es sich um ein Kästchen handelt, dann
     - lege eine Kopie der Rücksprungadresse dieses Kästchens oben auf den Keller,
     - suche den Eingang des Diagramms in U auf, welches den Namen trägt, der in dem erreichten Kästchen steht
     - und arbeite an dem Eingang des neuen Diagramms ab Punkt 2 weiter.

Algorithmus Rücksprungalgorithmus

1. Beginne am Eingang des ersten Syntaxdiagramms von U (Startdiagramm).
2. Folge den Linien auf einem legalen Weg.
   - Falls dabei der Ausgang erreicht wird, gehe nach Punkt 5.
   - Falls ein Kästchen bzw. Oval erreicht wird, gehe nach Punkt 3.
3. - Falls es sich um ein Oval handelt, notiere das darin enthaltene Terminalzeichen und gehe anschließend zu Punkt 2 zurück.
   - Andernfalls gehe nach Punkt 4.
4. - Falls es sich um ein Kästchen handelt, dann
     - lege eine Kopie der Rücksprungadresse dieses Kästchens oben auf den Keller,
     - suche den Eingang des Diagramms in U auf, welches den Namen trägt, der in dem erreichten Kästchen steht
     - und arbeite an dem Eingang des neuen Diagramms ab Punkt 2 weiter.
5. - Wenn noch eine Rücksprungadresse adr auf dem Keller liegt, dann
     - gehe zur Stelle, die mit adr gekennzeichnet ist und
     - nehme adr vom Keller und setze die Bearbeitung an dieser Stelle am Punkt 2 fort.
   - Wenn keine Rücksprungadresse auf dem Keller liegt und man sich am Ausgang des Startdiagrammes befindet, dann endet der Erzeugungsprozess hier erfolgreich.

Beispiel einer EBNF-Definition

$\mathcal{E} = (V, \Sigma, S, R)$ mit

Beispiel einer EBNF-Definition

$\mathcal{E} = (V, \Sigma, S, R)$ mit

$V = \{S, A\}$ (syntaktische Variablen)

Beispiel einer EBNF-Definition

$\mathcal{E} = (V, \Sigma, S, R)$ mit

$$V = \{S, A\} \qquad \text{(syntaktische Variablen)}$$

$$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\} \qquad \text{(Terminalsymbole)}$$

Beispiel einer EBNF-Definition

$\mathcal{E} = (V, \Sigma, S, R)$ mit

$V = \{S, A\}$ (syntaktische Variablen)

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ (Terminalsymbole)

$$R: \quad S ::= \hat{\{}\mathrm{c}\hat{\}}A$$

$$A ::= \hat{(}\hat{(}\mathrm{a}A\mathrm{b}\hat{)}\hat{|}\mathrm{a}\hat{)}$$ (EBNF-Regeln)

Definition

Sei V eine Menge von syntaktischen Variablen, und sei Σ eine Menge von Terminalsymbolen mit $V \cap \Sigma = \emptyset$. Die *Menge der EBNF-Terme über V und* Σ, bezeichnet durch $T(\Sigma, V)$, ist die kleinste Menge $T \subseteq \left(V \cup \Sigma \cup \{\hat{\{}, \hat{\}}, \hat{[}, \hat{]}, \hat{(}, \hat{)}, \hat{|}\,\}\right)^*$, so dass folgende Eigenschaften gelten:

1. $V \subseteq T$.
2. $\Sigma \subseteq T$.
3. Wenn $\alpha \in T$, so auch $\hat{(}\alpha\hat{)} \in T, \hat{\{}\alpha\hat{\}} \in T$, $\hat{[}\alpha\hat{]} \in T$.
4. Wenn $\alpha_1, \alpha_2 \in T$, so auch $\hat{(}\alpha_1 \;\hat{|}\; \alpha_2\hat{)} \in T$, $\alpha_1\alpha_2 \in T$.

Definition

Eine *EBNF-Definition* ist ein Tupel $E = (V, \Sigma, S, R)$, wobei

Definition

Eine *EBNF-Definition* ist ein Tupel $E = (V, \Sigma, S, R)$, wobei

- V endliche Menge (syntaktische Variablen)

Definition

Eine *EBNF-Definition* ist ein Tupel $E = (V, \Sigma, S, R)$, wobei

- V endliche Menge (syntaktische Variablen)
- Σ endliche Menge (Terminalsymbole)

Definition

Eine *EBNF-Definition* ist ein Tupel $E = (V, \Sigma, S, R)$, wobei

- V endliche Menge (syntaktische Variablen)
- Σ endliche Menge (Terminalsymbole)
- $S \in V$ (Startsymbol)

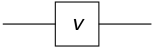
Definition

Eine *EBNF-Definition* ist ein Tupel $E = (V, \Sigma, S, R)$, wobei

- V endliche Menge (syntaktische Variablen)
- Σ endliche Menge (Terminalsymbole)
- $S \in V$ (Startsymbol)
- R endliche Menge von EBNF-Regeln der Form $v ::= \alpha$ mit $v \in V$ und $\alpha \in T(\Sigma, V)$. Weiterhin gilt, dass für jede syntaktische Variable v genau eine EBNF-Regel mit v als linker Seite in R enthalten ist.

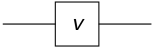
Übersetzung von EBNF-Definitionen in Syntaxdiagramme

1. Sei $v \in V$;

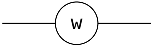
   $trans(v) =$ 

Übersetzung von EBNF-Definitionen in Syntaxdiagramme

1. Sei $v \in V$;

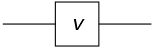
   $trans(v) =$ 

2. Sei $\mathtt{w} \in \Sigma$;

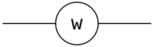
   $trans(\mathtt{w}) =$ 

Übersetzung von EBNF-Definitionen in Syntaxdiagramme

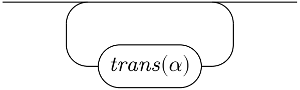
1. Sei $v \in V$;

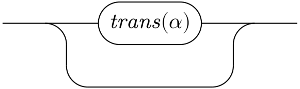
   $trans(v) =$ 

2. Sei $\mathtt{w} \in \Sigma$;

   $trans(\mathtt{w}) =$ 

3. Sei $\alpha \in T(\Sigma, V)$;

   - $trans(\ \hat{\{}\alpha\hat{\}}\ ) =$ 

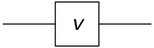
   - $trans(\ \hat{[}\alpha\hat{]}\ ) =$ 

   - $trans(\ \hat{(}\alpha\hat{)}\ ) = trans(\alpha)$

Übersetzung von EBNF-Definitionen in Syntaxdiagramme

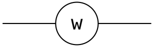
1. Sei $v \in V$;

   $trans(v) =$



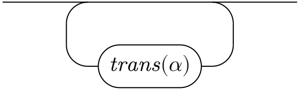
2. Sei $\mathtt{w} \in \Sigma$;

   $trans(\mathtt{w}) =$

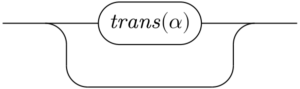


3. Sei $\alpha \in T(\Sigma, V)$;

   - $trans(\ \hat{\{}\alpha\hat{\}}\ ) =$



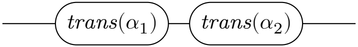
   - $trans(\ \hat{[}\alpha\hat{]}\ ) =$



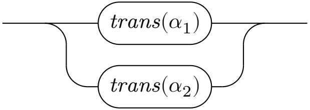
   - $trans(\ \hat{(}\alpha\hat{)}\ ) = trans(\alpha)$

4. Sei $\alpha_1, \alpha_2 \in T(\Sigma, V)$;

   - $trans(\alpha_1\alpha_2) =$



   - $trans(\ \hat{(}\alpha_1 \ \hat{|}\ \alpha_2\hat{)}\ ) =$



Übersetzung von EBNF-Definitionen in Syntaxdiagramme

EBNF-Definition $E = (V, \Sigma, S, R)$ mit

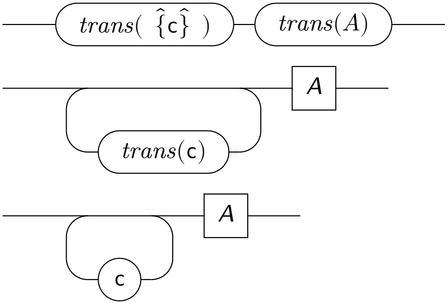
- $V = \{S, A\}$
- $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$
- $R = \{S ::= \hat{\{}\mathsf{c}\hat{\}}A, \quad A ::= \hat{(}\mathsf{a}A\mathsf{b} \mathbin{\hat{|}} \mathsf{a}\hat{)}\}$

Übersetzung von EBNF-Definitionen in Syntaxdiagramme

EBNF-Definition $E = (V, \Sigma, S, R)$ mit

- $V = \{S, A\}$
- $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$
- $R = \{S ::= \hat{\{}\mathsf{c}\hat{\}}A, \quad A ::= \hat{(}\mathsf{a}A\mathsf{b} \hat{|} \mathsf{a}\hat{)}\}$

$S\colon\ \mathit{trans}(\hat{\{}\mathsf{c}\hat{\}}A) = \ldots = \ldots$

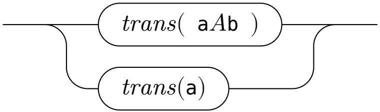


Übersetzung von EBNF-Definitionen in Syntaxdiagramme

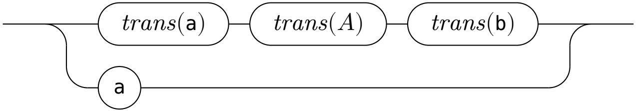
EBNF-Definition $E = (V, \Sigma, S, R)$ mit

- $V = \{S, A\}$
- $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$
- $R = \{S ::= \hat{\{}\mathsf{c}\hat{\}}A, \quad A ::= \hat{(}\mathsf{a}A\mathsf{b} \hat{|} \mathsf{a}\hat{)}\}$

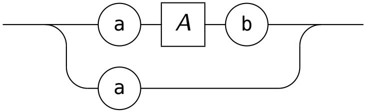
$A\colon \quad trans(\ \hat{(}\hat{(}\mathsf{a}A\mathsf{b}\hat{)} \hat{|}\ \mathsf{a}\ \hat{)}\ ) =$



$=$



$=$

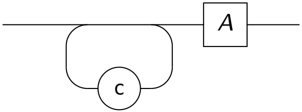


Übersetzung von EBNF-Definitionen in Syntaxdiagramme

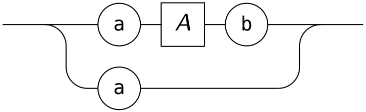
EBNF-Definition $E = (V, \Sigma, S, R)$ mit

- $V = \{S, A\}$
- $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$
- $R = \{S ::= \hat{\{}\mathsf{c}\hat{\}}A, \quad A ::= \hat{(}\mathsf{a}A\mathsf{b} \,\hat{|}\, \mathsf{a}\hat{)}\}$

S



A



Bedeutung einer EBNF-Definition

$$[\![.]\!] : T(\Sigma, V) \to ((V \to \mathcal{P}(\Sigma^*)) \to \mathcal{P}(\Sigma^*))$$

Sei also $\alpha \in T(\Sigma, V)$ und $\rho : V \to P(\Sigma^*)$ eine beliebige Funktion, die jedem $v \in V$ eine formale Sprache über Σ zuordnet. Dann definiere $[\![\alpha]\!](\rho)$ wie folgt:

- Wenn $\alpha = v \in V$, dann gilt $[\![\alpha]\!](\rho) = \rho(v)$.

Bedeutung einer EBNF-Definition

$$[\![.]\!]: T(\Sigma, V) \to ((V \to \mathcal{P}(\Sigma^*)) \to \mathcal{P}(\Sigma^*))$$

Sei also $\alpha \in T(\Sigma, V)$ und $\rho : V \to P(\Sigma^*)$ eine beliebige Funktion, die jedem $v \in V$ eine formale Sprache über Σ zuordnet. Dann definiere $[\![\alpha]\!](\rho)$ wie folgt:

- Wenn $\alpha = v \in V$, dann gilt $[\![\alpha]\!](\rho) = \rho(v)$.
- Wenn $\alpha \in \Sigma$, dann gilt $[\![\alpha]\!](\rho) = \{\alpha\}$.
  Beachte: „{“ und „}“ sind hier übliche Mengenklammern.

Bedeutung einer EBNF-Definition

$$[\![.]\!] : T(\Sigma, V) \to ((V \to \mathcal{P}(\Sigma^*)) \to \mathcal{P}(\Sigma^*))$$

Sei also $\alpha \in T(\Sigma, V)$ und $\rho : V \to P(\Sigma^*)$ eine beliebige Funktion, die jedem $v \in V$ eine formale Sprache über Σ zuordnet. Dann definiere $[\![\alpha]\!](\rho)$ wie folgt:

- Wenn $\alpha = v \in V$, dann gilt $[\![\alpha]\!](\rho) = \rho(v)$.
- Wenn $\alpha \in \Sigma$, dann gilt $[\![\alpha]\!](\rho) = \{\alpha\}$.
  Beachte: „$\{$“ und „$\}$“ sind hier übliche Mengenklammern.
- Wenn $\alpha = \hat{(}\alpha_1\hat{)}$, dann gilt $[\![\alpha]\!](\rho) = [\![\alpha_1]\!](\rho)$.

Bedeutung einer EBNF-Definition

$$[\![.]\!] : T(\Sigma, V) \to ((V \to \mathcal{P}(\Sigma^*)) \to \mathcal{P}(\Sigma^*))$$

Sei also $\alpha \in T(\Sigma, V)$ und $\rho : V \to P(\Sigma^*)$ eine beliebige Funktion, die jedem $v \in V$ eine formale Sprache über Σ zuordnet. Dann definiere $[\![\alpha]\!](\rho)$ wie folgt:

- Wenn $\alpha = v \in V$, dann gilt $[\![\alpha]\!](\rho) = \rho(v)$.
- Wenn $\alpha \in \Sigma$, dann gilt $[\![\alpha]\!](\rho) = \{\alpha\}$.
  Beachte: „{“ und „}“ sind hier übliche Mengenklammern.
- Wenn $\alpha = \widehat{(}\alpha_1\widehat{)}$, dann gilt $[\![\alpha]\!](\rho) = [\![\alpha_1]\!](\rho)$.
- Wenn $\alpha = \widehat{\{}\alpha_1\widehat{\}}$, dann gilt $[\![\alpha]\!](\rho) = ([\![\alpha_1]\!](\rho))^*$, d. h. $[\![\alpha]\!](\rho)$ ist der Stern von $[\![\alpha_1]\!](\rho)$.

Bedeutung einer EBNF-Definition

$$[\![.]\!] : T(\Sigma, V) \to ((V \to \mathcal{P}(\Sigma^*)) \to \mathcal{P}(\Sigma^*))$$

Sei also $\alpha \in T(\Sigma, V)$ und $\rho : V \to P(\Sigma^*)$ eine beliebige Funktion, die jedem $v \in V$ eine formale Sprache über Σ zuordnet. Dann definiere $[\![\alpha]\!](\rho)$ wie folgt:

- Wenn $\alpha = v \in V$, dann gilt $[\![\alpha]\!](\rho) = \rho(v)$.
- Wenn $\alpha \in \Sigma$, dann gilt $[\![\alpha]\!](\rho) = \{\alpha\}$.
  Beachte: „$\{$“ und „$\}$“ sind hier übliche Mengenklammern.
- Wenn $\alpha = \hat{(}\alpha_1\hat{)}$, dann gilt $[\![\alpha]\!](\rho) = [\![\alpha_1]\!](\rho)$.
- Wenn $\alpha = \hat{\{}\alpha_1\hat{\}}$, dann gilt $[\![\alpha]\!](\rho) = ([\![\alpha_1]\!](\rho))^*$, d. h. $[\![\alpha]\!](\rho)$ ist der Stern von $[\![\alpha_1]\!](\rho)$.
- Wenn $\alpha = \hat{[}\alpha_1\hat{]}$, dann gilt $[\![\alpha]\!](\rho) = [\![\alpha_1]\!](\rho) \cup \{\varepsilon\}$.

Bedeutung einer EBNF-Definition

$$[\![.]\!] : T(\Sigma, V) \to ((V \to \mathcal{P}(\Sigma^*)) \to \mathcal{P}(\Sigma^*))$$

Sei also $\alpha \in T(\Sigma, V)$ und $\rho : V \to P(\Sigma^*)$ eine beliebige Funktion, die jedem $v \in V$ eine formale Sprache über Σ zuordnet. Dann definiere $[\![\alpha]\!](\rho)$ wie folgt:

- Wenn $\alpha = v \in V$, dann gilt $[\![\alpha]\!](\rho) = \rho(v)$.
- Wenn $\alpha \in \Sigma$, dann gilt $[\![\alpha]\!](\rho) = \{\alpha\}$.
  Beachte: „{“ und „}“ sind hier übliche Mengenklammern.
- Wenn $\alpha = \widehat{(}\alpha_1\widehat{)}$, dann gilt $[\![\alpha]\!](\rho) = [\![\alpha_1]\!](\rho)$.
- Wenn $\alpha = \widehat{\{}\alpha_1\widehat{\}}$, dann gilt $[\![\alpha]\!](\rho) = ([\![\alpha_1]\!](\rho))^*$, d. h. $[\![\alpha]\!](\rho)$ ist der Stern von $[\![\alpha_1]\!](\rho)$.
- Wenn $\alpha = \widehat{[}\alpha_1\widehat{]}$, dann gilt $[\![\alpha]\!](\rho) = [\![\alpha_1]\!](\rho) \cup \{\varepsilon\}$.
- Wenn $\alpha = \widehat{(}\alpha_1\widehat{|}\alpha_2\widehat{)}$, dann gilt $[\![\alpha]\!](\rho) = [\![\alpha_1]\!](\rho) \cup [\![\alpha_2]\!](\rho)$.

Bedeutung einer EBNF-Definition

$$[\![.]\!] : T(\Sigma, V) \to ((V \to \mathcal{P}(\Sigma^*)) \to \mathcal{P}(\Sigma^*))$$

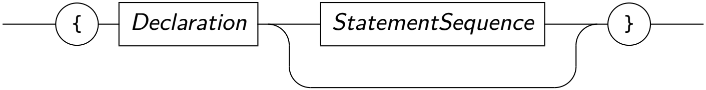
Sei also $\alpha \in T(\Sigma, V)$ und $\rho : V \to P(\Sigma^*)$ eine beliebige Funktion, die jedem $v \in V$ eine formale Sprache über Σ zuordnet. Dann definiere $[\![\alpha]\!](\rho)$ wie folgt:

- Wenn $\alpha = v \in V$, dann gilt $[\![\alpha]\!](\rho) = \rho(v)$.
- Wenn $\alpha \in \Sigma$, dann gilt $[\![\alpha]\!](\rho) = \{\alpha\}$.
  Beachte: „$\{$“ und „$\}$“ sind hier übliche Mengenklammern.
- Wenn $\alpha = \hat{(}\alpha_1\hat{)}$, dann gilt $[\![\alpha]\!](\rho) = [\![\alpha_1]\!](\rho)$.
- Wenn $\alpha = \hat{\{}\alpha_1\hat{\}}$, dann gilt $[\![\alpha]\!](\rho) = ([\![\alpha_1]\!](\rho))^*$, d. h. $[\![\alpha]\!](\rho)$ ist der Stern von $[\![\alpha_1]\!](\rho)$.
- Wenn $\alpha = \hat{[}\alpha_1\hat{]}$, dann gilt $[\![\alpha]\!](\rho) = [\![\alpha_1]\!](\rho) \cup \{\varepsilon\}$.
- Wenn $\alpha = \hat{(}\alpha_1\hat{|}\alpha_2\hat{)}$, dann gilt $[\![\alpha]\!](\rho) = [\![\alpha_1]\!](\rho) \cup [\![\alpha_2]\!](\rho)$.
- Wenn $\alpha = \alpha_1\alpha_2$, dann gilt $[\![\alpha]\!](\rho) = [\![\alpha_1]\!](\rho) \cdot [\![\alpha_2]\!](\rho)$, wobei die Objektsprachen $[\![\alpha_1]\!](\rho)$ und $[\![\alpha_2]\!](\rho)$ durch Konkatenation verknüpft sind.

Definition eines Blocks

$$\langle Block \rangle ::= \{ \langle Declaration \rangle \hat{[} \langle StatementSequence \rangle \hat{]} \}$$

Block

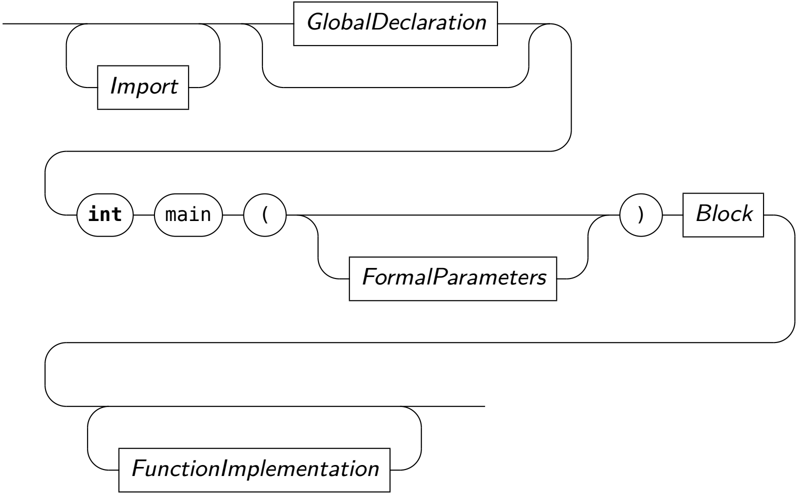


Inhaltsverzeichnis I

Inhaltsverzeichnis II

Aufbau eines Programms

Program



Beispiel: Quadratzahlen summieren

```
/* Summation */
#include <stdio.h>

int n, s;

int main()
{ int i;

  scanf("%d",&n);
  i = 1;
  s = 0;
  while (i <= n)
  { s = s+i*i;
    i = i+1;
  }
  printf("%d",s);
  return 0;
}
```

Spezielle Symbole und Wörter in C

Schlüsselwörter (oder: keywords) sind Wörter mit fester Bedeutung, die mit einem Kleinbuchstaben oder _ beginnen. Schlüsselwörter in C sind z. B.:

break	**case**	**char**	**const**	**do**	**double**
else	**enum**	**float**	**for**	**if**	**int**
long	**return**	**short**	**struct**	**switch**	**typedef**
union	**unsigned**	**void**	**while**		

Operatoren und Begrenzer (oder: punctuators), z. B.:

`{`	`}`	`(`	`)`	`/`	`%`
`&`	`&&`	`->`	`+`	`*`	`#`
`|`	`||`	`;`	`-`		

spezielle Bezeichner (oder: special identifiers), z. B. `main`

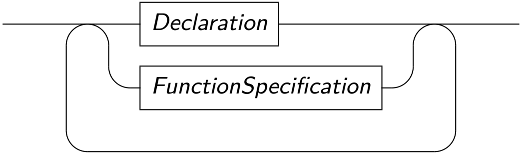
Präprozessor-Token (oder: preprocessing tokens), z. B. `define` und `include`

Bezeichner (Identifier)

- Bezeichner sind Zeichenfolgen
- Bezeichner dürfen Zeichenfolgen beliebiger Länge sein; allerdings sind für die Unterscheidung von zwei Bezeichnern nur die ersten 32 Zeichen signifikant.
- Bezeichner dürfen große und kleine Buchstaben (keine Umlaute!), Ziffern und den Unterstrich (_) enthalten.
- Bezeichner dürfen *nicht* mit einer Ziffer beginnen.
- Schlüsselwörter und einige reservierte Namen dürfen nicht als Bezeichner verwendet werden.

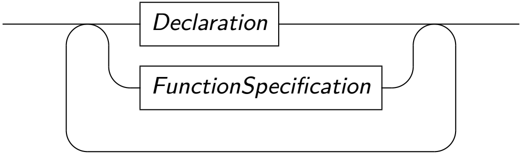
Deklarationen

GlobalDeclaration

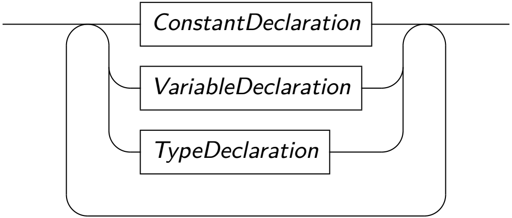


Deklarationen

GlobalDeclaration

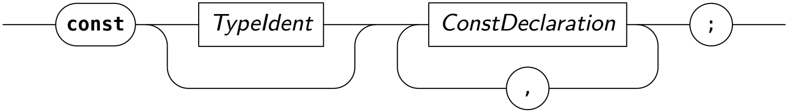


Declaration



Konstantendeklaration

ConstantDeclaration



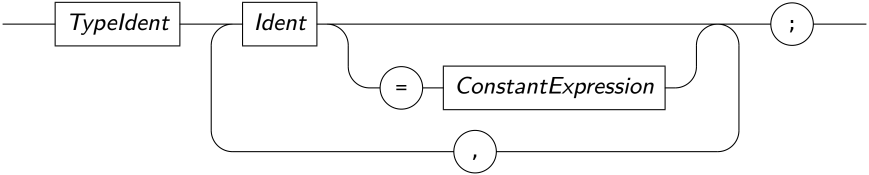
ConstDeclaration

Beispiel:

```
const Zahl1 = 4, Zahl2 = 12;
const Zeichen = 'A';
const double Wert = 27.9;
const float Pi = (float) 3.1415;
```

Variablendeklaration

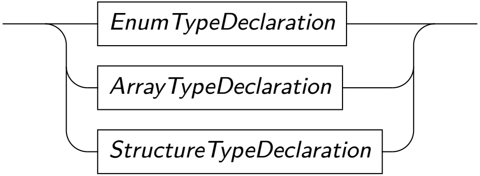
VariableDeclaration



Beispiel: `int n, s = -3, l = 27; /* n = ?, s = -3, l = 27 */`

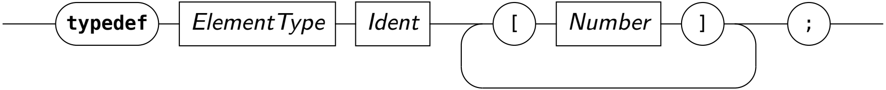
Typdeklaration

TypeDeclaration



Typdeklaration

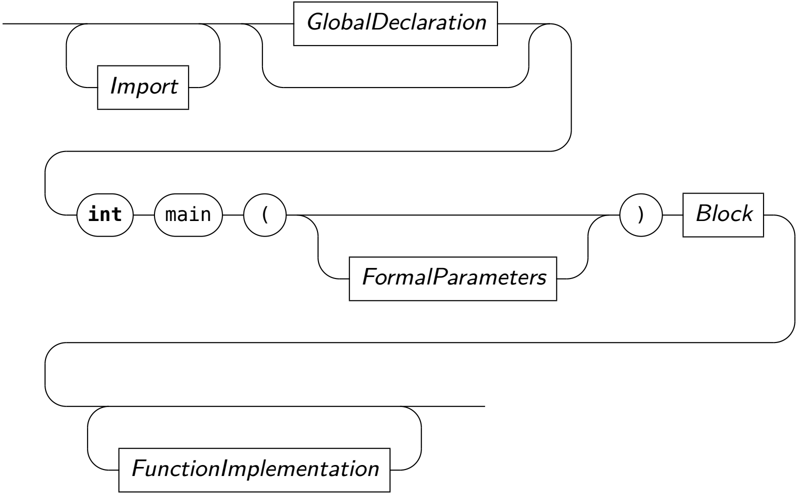
ArrayTypeDeclaration




```
typedef int feld[20];     /* deklariert einen ARRAY-Typ mit
                                 20 Elementen des Typs int;
                                 der Bezeichner des Typs ist feld */

feld A, B;                /* zwei Variablen des Typs feld */
```

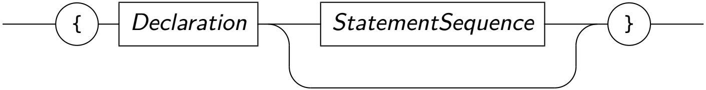
Aufbau eines Programms (Wdh.)

Program

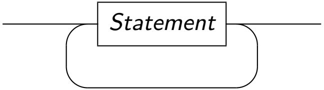


Block einer Funktion

Block



StatementSequence

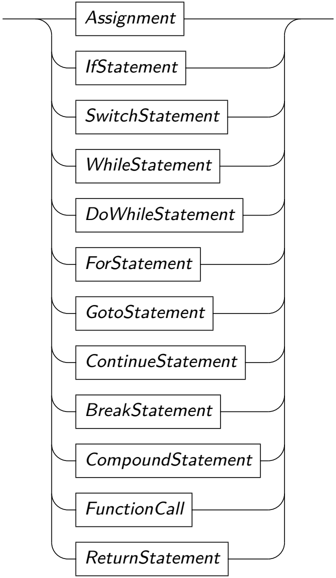


Inhaltsverzeichnis I

Inhaltsverzeichnis II

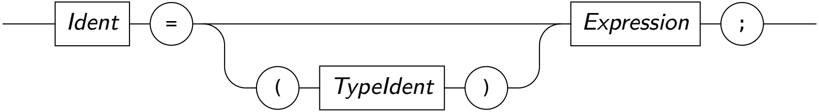
Einfache Kontrollstrukturen von C

Statement



Einfache Kontrollstrukturen von C

Assignment

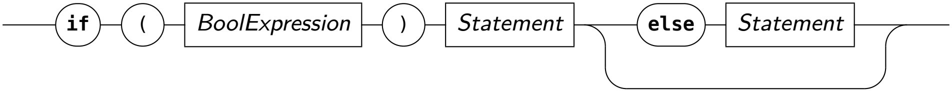


Beispiel:

```
int n, s;
float k;
. . .
k = (float) (n * s);
```

Einfache Kontrollstrukturen von C

IfStatement

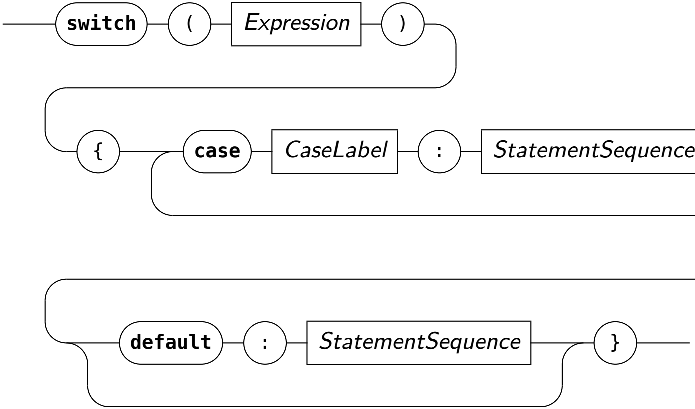


Beispiel:

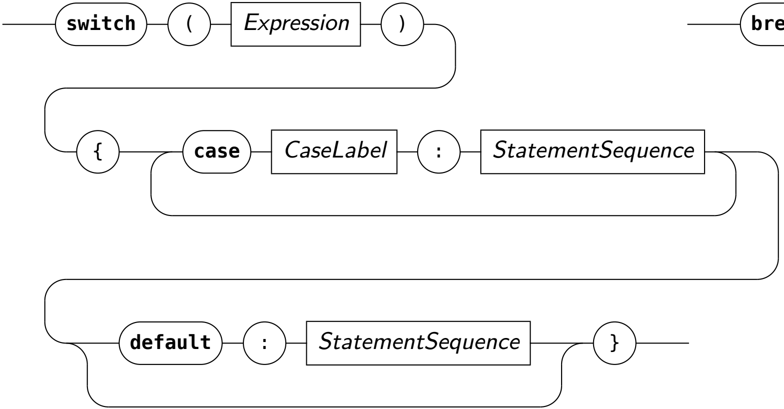
```
if (h == tail) { h = p; q = r; }

if (h == tail) { h = p; q = r; } else { h = r; q = p; }
```

Einfache Kontrollstrukturen von C

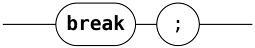
SwitchStatement



SwitchStatement

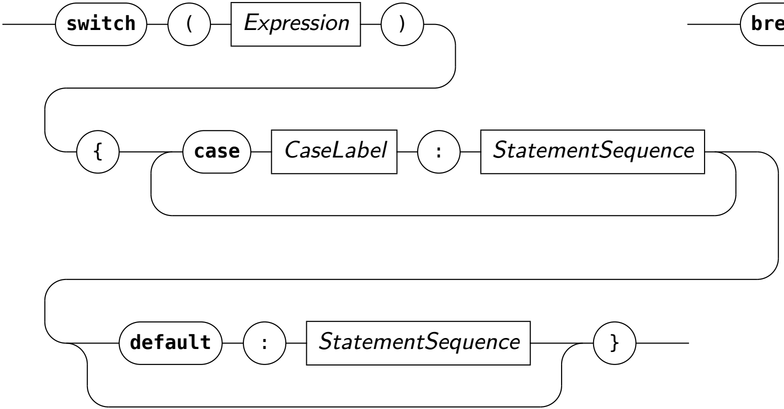


BreakStatement

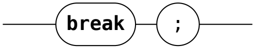


Einfache Kontrollstrukturen von C

SwitchStatement



BreakStatement



Beispiel:

```
switch (exp)
{ case 0:   x = 0;      a = b;      break;
  case 1:   x = x + 1;  a = 2 * b;  break;
  default:  x = 1;      a = 0;             }
```

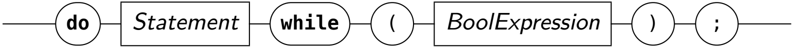
Einfache Kontrollstrukturen von C

WhileStatement

Beispiel: `while (s - 0.5*s > eps) { s = 0.5*s; }`

Einfache Kontrollstrukturen von C

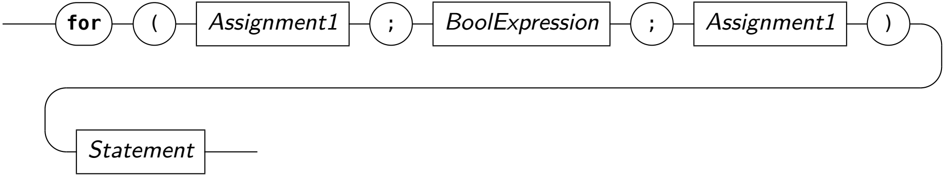
DoWhileStatement



Beispiel: `do { s = 0.5*s; } while (s - 0.5*s > eps);`

Einfache Kontrollstrukturen von C

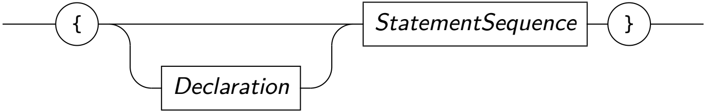
ForStatement



Beispiel:

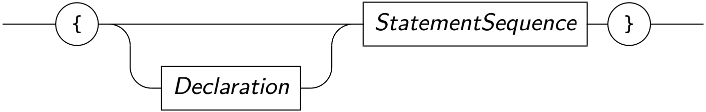
```
for (i = 1; i <= 10; i = i + 1) printf("%d\n", i);
```

CompoundStatement



Einfache Kontrollstrukturen von C

CompoundStatement



Beispiel:

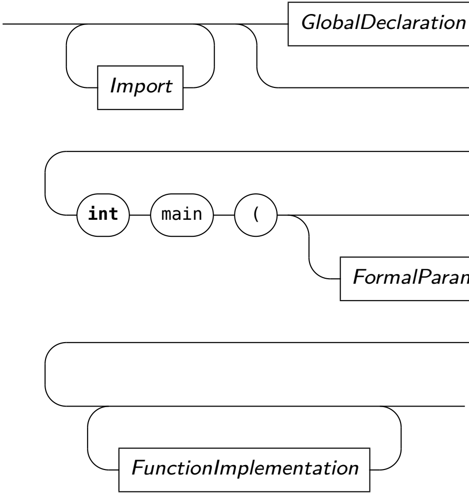
```
#include <stdio.h>          /* Endlosschleife */

int i;

int main()
{ i = 5;
  while (i > 0)             /* Endlosschleife, da innerhalb des CompoundStatements */
  { int i = 8;              /* nur die lokal deklarierte Variable i sichtbar ist,  */
    printf("i = %d\n",i);  /* für Test auf Abbruch aber die globale Variable i    */
    i = i-1;                /* verwendet wird. Außerdem erreicht das innere i nie  */
  }                         /* den Wert 0, da es am Anfang eines jeden Schleifen-  */
  return 0;                 /* durchlaufs erneut auf 8 gesetzt wird.               */
}
```

Inhaltsverzeichnis I

Inhaltsverzeichnis II

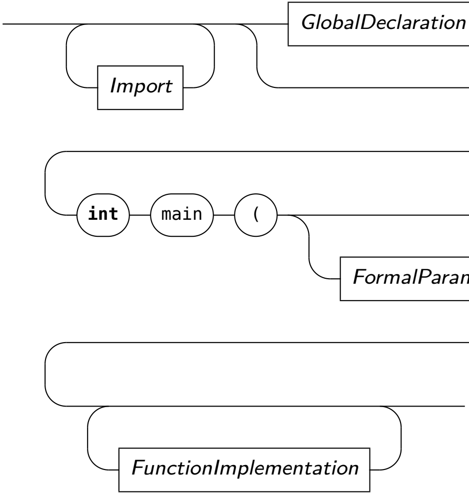
Aufbau eines Programms (Wdh.)

Program

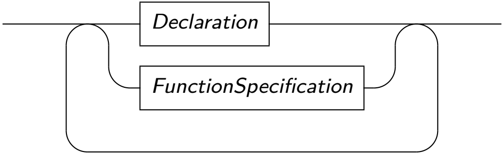


Aufbau eines Programms (Wdh.)

Program



GlobalDeclaration



Funktionen in der Mathematik und in C

	FunctionHeading	*FunctionImplementation*
Mathematik	$f\colon \mathbb{Z} \to \mathbb{Z}$	$f\colon \mathbb{Z} \to \mathbb{Z}$ $f(x) = 3x^2 + 4$

Funktionen in der Mathematik und in C

	FunctionHeading	*FunctionImplementation*
Mathematik	$f\colon \mathbb{Z} \to \mathbb{Z}$	$f\colon \mathbb{Z} \to \mathbb{Z}$ $f(x) = 3x^2 + 4$

C	`int f (int x);`	`int f (int x)`
		`{ return (3*x*x + 4);`
		`}`

Funktionsdeklaration

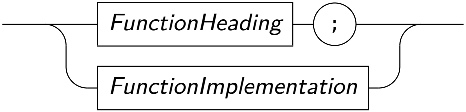
```c
#include <stdio.h>

/* Näherung des Sinus, basierend auf der Taylorentwicklung 5-ten Grades */
float sinus(float x)      /* Funktionsimplementation */
{ return x  -  x * x * x / 6  +  x * x * x * x * x / 120; }

int main()
{ float x1, x2, y1, y2;

  scanf("%f", &x1);
  y1 = sinus(x1);         /* Funktionsaufruf */
  printf("sinus(%f) = %f\n", x1, y1);

  scanf("%f", &x2);
  y2 = sinus(x2);         /* Funktionsaufruf */
    printf("sinus(%f) = %f\n", x2, y2);

  return 0;
}
```

Forward-Deklaration

```c
#include <stdio.h>

/* Näherung des Sinus, basierend auf der Taylorentwicklung 5-ten Grades */
float sinus(float x);     /* Funktionskopf*/

int main()
{ float x1, x2, y1, y2;

  scanf("%f", &x1);
  y1 = sinus(x1);         /* Funktionsaufruf */
  printf("sinus(%f) = %f\n", x1, y1);

  scanf("%f", &x2);
  y2 = sinus(x2);         /* Funktionsaufruf */
    printf("sinus(%f) = %f\n", x2, y2);

  return 0;
}

float sinus(float x)      /* Funktionsimplementation */
{ return x  -  x * x * x / 6  +  x * x * x * x * x / 120; }
```

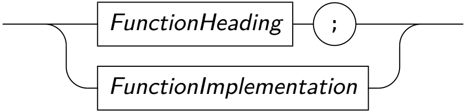
Deklaration von Funktionen

FunctionSpecification

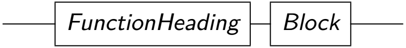


Deklaration von Funktionen

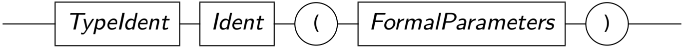
FunctionSpecification



FunctionImplementation

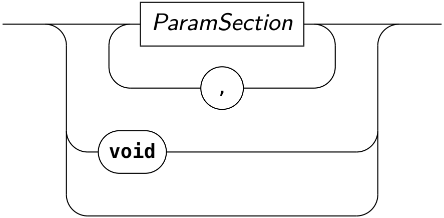


FunctionHeading



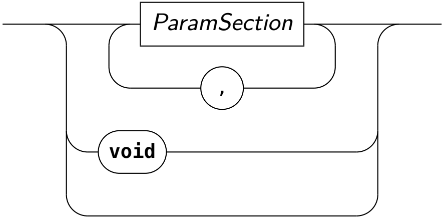
Deklaration von Funktionen

FormalParameters

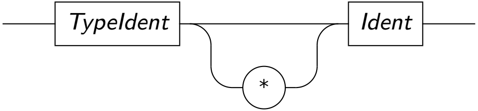


Deklaration von Funktionen

FormalParameters

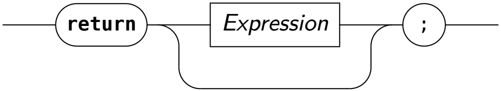


ParamSection



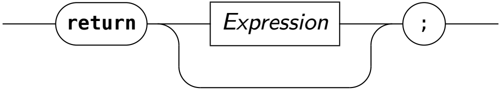
Deklaration von Funktionen

ReturnStatement

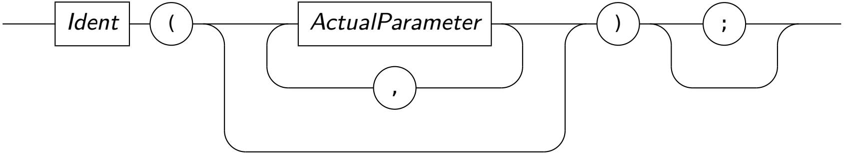


Deklaration von Funktionen

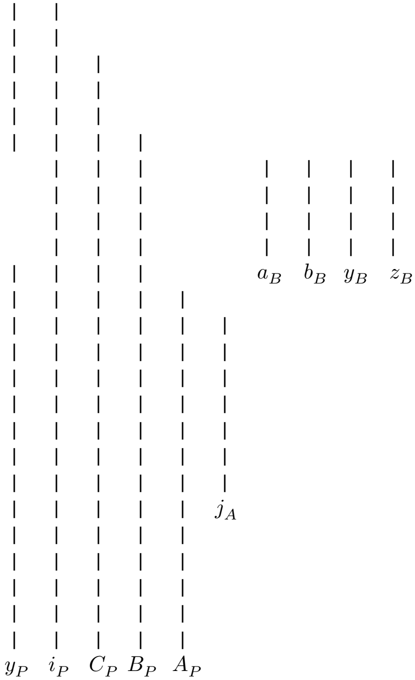
ReturnStatement



FunctionCall



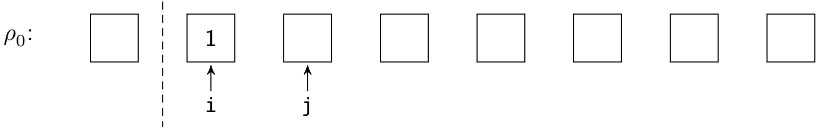
```
int y, i;

void C()
{ i = i - 1; }

void B(int a, int b)
{ int y, z;
  y = 1;
  C();
}

void A()
{ int j;
  y = 2 * y; j = 5;
  if (i < 4)
  { i = i - 1;
    B(i, 6 * j);
  }
}

int main() /* Hauptprogramm */
{ i = 1; y = 2;
  A();
  return 0;
}
```



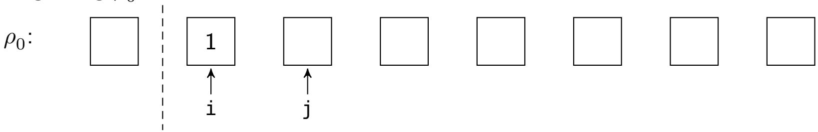
Pulsierender Speicher bei Aufruf von Funktionen

```
int i, j;

void P(int a, int b, int *c)
{ int d, e;
  . . .
  *c = *c + 1;
  . . .
}

int main()
{ i = 1;
  P(4 + 2 * i, 5, &i);
  . . .
  return 0;
}
```

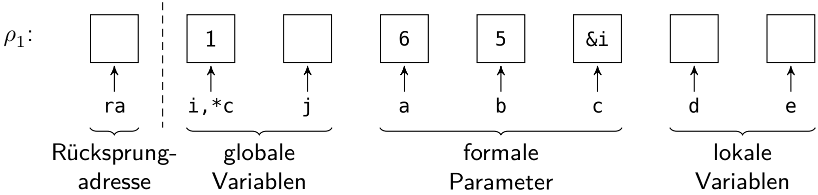
Umgebung ρ_0 vor Aufruf von P:



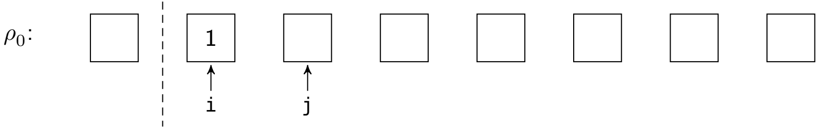
Umgebung ρ_0 vor Aufruf von P:



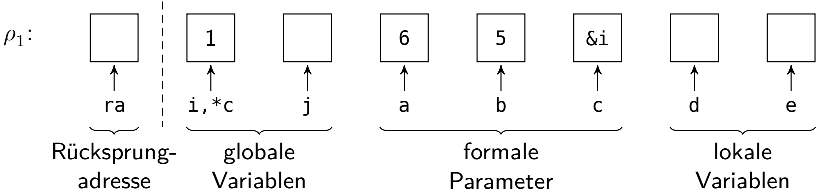
Umgebung ρ_1 nach Aufruf von P:



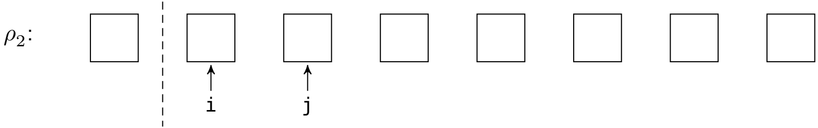
Umgebung ρ_0 vor Aufruf von P:



Umgebung ρ_1 nach Aufruf von P:



Umgebung ρ_2 nach Verlassen von P:



Unwirksame Parameterübergabe

```
int x;

void unwirksam(int a, int b)
{ /*label1*/
  a = a + b;
  /*label2*/
}

int main()
{ x = 3;
  /*label3*/
  unwirksam(x, 4); /*$1*/
  /*label4*/
  return 0;
}
```

Unwirksame Parameterübergabe

```
int x;

void unwirksam(int a, int b)
{ /*label1*/
  a = a + b;
  /*label2*/
}

int main()
{ x = 3;
  /*label3*/
  unwirksam(x, 4); /*$1*/
  /*label4*/
  return 0;
}
```

Haltepunkt	RM	Umgebung 1	 2	 3
label3	–	x 3		
label1	1	x 3	a 3	b 4
label2	1	x 3	a 7	b 4
label4	–	x 3		

Wirksame Parameterübergabe

```
int x;

void wirksam(int *a, int b)
{ /*label1*/
  *a = *a + b;
  /*label2*/
}

int main()
{ x = 3;
  /*label3*/
  wirksam(&x, 4); /*$1*/
  /*label4*/
  return 0;
}
```

Wirksame Parameterübergabe

```
int x;

void wirksam(int *a, int b)
{ /*label1*/
  *a = *a + b;
  /*label2*/
}

int main()
{ x = 3;
  /*label3*/
  wirksam(&x, 4); /*$1*/
  /*label4*/
  return 0;
}
```

Haltepunkt	RM	Umgebung		
		1	2	3
`label3`	–	x		
		3		
`label1`	1	x	a	b
		3	1	4
`label2`	1	x	a	b
		7	1	4
`label4`	–	x		
		7		

Gültigkeitsbereich in rekursiven Funktionen

```c
/*  StaticScope */
#include <stdio.h>

int x, i;

void A();

void B()
{ int x;

  x = 1;
  printf("%d\n", x);
   /*label1*/
  A(); /*$2*/
   /*label2*/
  printf("%d\n", x);
}

void A()
{  /*label3*/
  x = 2*x;
  printf("%d\n", x);
  if (i < 4)
  { i = i+1;
     /*label4*/
    B(); /*$3*/
  }
   /*label5*/
  printf("%d\n", x);
}

int main() /* Hauptprogramm */
{ i = 1;
  x = 2;
   /*label6*/
  A(); /*$1*/
   /*label7*/
  return 0;
}
```

Haltepunkt	RM	Umgebung 1	2	3	4	5
label6	–	x	i			
		2	1			
label3	1	x	i			
		2	1			
label4	1	x	i			
		4	2			
label1	$3:1$		i	x		
		4	2	1		
label3	$2:3:1$	x	i			
		4	2	1		
label4	$2:3:1$	x	i			
		8	3	1		
label1	$3:2:3:1$		i		x	
		8	3	1	1	
label3	$2:3:2:3:1$	x	i			
		8	3	1	1	
label4	$2:3:2:3:1$	x	i			
		16	4	1	1	
label1	$3:2:3:2:3:1$		i			x
		16	4	1	1	1
label3	$2:3:2:3:2:3:1$	x	i			
		16	4	1	1	1
label5	$2:3:2:3:2:3:1$	x	i			
		32	4	1	1	1
label2	$3:2:3:2:3:1$		i			x
		32	4	1	1	1
label5	$2:3:2:3:1$	x	i			
		32	4	1	1	
label2	$3:2:3:1$		i		x	
		32	4	1	1	
label5	$2:3:1$	x	i			
		32	4	1		
label2	$3:1$		i	x		
		32	4	1		
label5	1	x	i			
		32	4			
label7	–	x	i			
		32	4			

```c
#include <stdio.h>
void g(int x, int y, int *z);

void f(int x, int *y)
{ int u;
   /*label1*/
  if (x > 0)
  { f(x-1, &u);   /*$2*/
     /*label7*/
    g(x-1, u, y); /*$3*/
     /*label8*/
  }
  else *y = 1;
   /*label2*/
}

void g(int x, int y, int *z)
{ int u;
   /*label3*/
  if (x > 0)
  { f(x-1, &u);   /*$4*/
     /*label9*/
    *z = u+y;
  }
  else *z = 1;
   /*label4*/
}

int main()
{ int e, a;
  scanf("%d", &e);
   /*label5*/
  f(e, &a);       /*$1*/
  printf("a = %d\n", a);
   /*label6*/
  return 0;
}
```

Haltepunkt	RM	Umgebung								
		1	2	3	4	5	6	7	8	9
`label5`	–	e	a							
		1	?							

Haltepunkt	RM	Umgebung								
		1	2	3	4	5	6	7	8	9
`label5`	–	e	a							
		1	?							
`label1`	1			x	y	u				
		1	?	1	2	?				

Haltepunkt	RM	Umgebung								
		1	2	3	4	5	6	7	8	9
`label5`	–	e	a							
		1	?							
`label1`	1			x	y	u				
		1	?	1	2	?				
`label1`	$2:1$						x	y	u	
		1	?	1	2	?	0	5	?	

Haltepunkt	RM	Umgebung								
		1	2	3	4	5	6	7	8	9
`label5`	–	e	a							
		1	?							
`label1`	1			x	y	u				
		1	?	1	2	?				
`label1`	$2:1$						x	y	u	
		1	?	1	2	?	0	5	?	
`label2`	$2:1$						x	y	u	
		1	?	1	2	1	0	5	?	

Haltepunkt	RM	Umgebung								
		1	2	3	4	5	6	7	8	9
`label5`	–	e	a							
		1	?							
`label1`	1			x	y	u				
		1	?	1	2	?				
`label1`	$2:1$						x	y	u	
		1	?	1	2	?	0	5	?	
`label2`	$2:1$						x	y	u	
		1	?	1	2	1	0	5	?	
`label7`	1			x	y	u				
		1	?	1	2	1				

Haltepunkt	RM	Umgebung								
		1	2	3	4	5	6	7	8	9
`label5`	–	e	a							
		1	?							
`label1`	1			x	y	u				
		1	?	1	2	?				
`label1`	$2:1$						x	y	u	
		1	?	1	2	?	0	5	?	
`label2`	$2:1$						x	y	u	
		1	?	1	2	1	0	5	?	
`label7`	1			x	y	u				
		1	?	1	2	1				
`label3`	$3:1$						x	y	z	u
		1	?	1	2	1	0	1	2	?

Haltepunkt	RM	Umgebung								
		1	2	3	4	5	6	7	8	9
`label5`	–	e	a							
		1	?							
`label1`	1			x	y	u				
		1	?	1	2	?				
`label1`	$2:1$						x	y	u	
		1	?	1	2	?	0	5	?	
`label2`	$2:1$						x	y	u	
		1	?	1	2	1	0	5	?	
`label7`	1			x	y	u				
		1	?	1	2	1				
`label3`	$3:1$						x	y	z	u
		1	?	1	2	1	0	1	2	?
`label4`	$3:1$						x	y	z	u
		1	1	1	2	1	0	1	2	?

Haltepunkt	RM	Umgebung								
		1	2	3	4	5	6	7	8	9
`label5`	–	e	a							
		1	?							
`label1`	1			x	y	u				
		1	?	1	2	?				
`label1`	$2:1$						x	y	u	
		1	?	1	2	?	0	5	?	
`label2`	$2:1$						x	y	u	
		1	?	1	2	1	0	5	?	
`label7`	1			x	y	u				
		1	?	1	2	1				
`label3`	$3:1$						x	y	z	u
		1	?	1	2	1	0	1	2	?
`label4`	$3:1$						x	y	z	u
		1	1	1	2	1	0	1	2	?
`label8`	1			x	y	u				
		1	1	1	2	1				

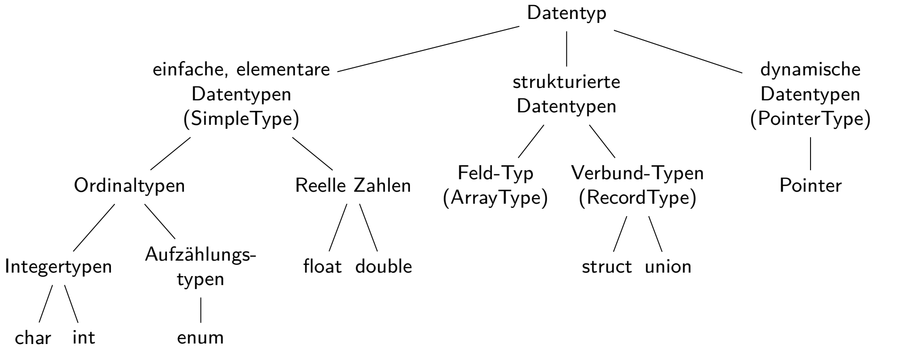
Haltepunkt	RM	Umgebung								
		1	2	3	4	5	6	7	8	9
`label5`	–	e	a							
		1	?							
`label1`	1			x	y	u				
		1	?	1	2	?				
`label1`	$2:1$						x	y	u	
		1	?	1	2	?	0	5	?	
`label2`	$2:1$						x	y	u	
		1	?	1	2	1	0	5	?	
`label7`	1			x	y	u				
		1	?	1	2	1				
`label3`	$3:1$						x	y	z	u
		1	?	1	2	1	0	1	2	?
`label4`	$3:1$						x	y	z	u
		1	1	1	2	1	0	1	2	?
`label8`	1			x	y	u				
		1	1	1	2	1				
`label2`	1			x	y	u				
		1	1	1	2	1				

Haltepunkt	RM	Umgebung								
		1	2	3	4	5	6	7	8	9
`label5`	–	e 1	a ?							
`label1`	1	 1	 ?	x 1	y 2	u ?				
`label1`	$2:1$	 1	 ?	 1	 2	 ?	x 0	y 5	u ?	
`label2`	$2:1$	 1	 ?	 1	 2	 1	x 0	y 5	u ?	
`label7`	1	 1	 ?	x 1	y 2	u 1				
`label3`	$3:1$	 1	 ?	 1	 2	 1	x 0	y 1	z 2	u ?
`label4`	$3:1$	 1	 1	 1	 2	 1	x 0	y 1	z 2	u ?
`label8`	1	 1	 1	x 1	y 2	u 1				
`label2`	1	 1	 1	x 1	y 2	u 1				
`label6`	–	e 1	a 1							

Inhaltsverzeichnis I

Inhaltsverzeichnis II

Übersicht über Datentypen in C



Typ `char`

Wertebereich: -128 bis $+127$ (256 Werte)

`unsigned char`: 0 bis 255

gebräuchlicher Code: ASCII-Code (American Standard Code for Information Interchange)

Typ char

Wertebereich: -128 bis $+127$ (256 Werte)

unsigned char: 0 bis 255

gebräuchlicher Code: ASCII-Code (American Standard Code for Information Interchange)

```
. . .
const c = 'A';
char d;

d = c;
d = '+';
d = 12;     /* !!! */
. . .
```

Zu beachten ist, dass char intern ein *numerischer* Datentyp ist.

Operationen: Alle Operationen für ganze Zahlen, siehe dazu Datentyp int

ASCII-Tabelle

0	NUL	16	DLE	32		48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Beispiel: Parsen einer Zahl

```c
. . .
char ch;
int x;

int main()
{ x = 0;
  scanf("%c", &ch);
  while (('0' <= ch) && (ch <= '9'))
  { x = 10 * x + ch - '0';
    printf("x = %d\n", x);
    scanf(" %c", &ch);
  }
  return 0;
}
```

Taste	ch	x	Ausgabe
0	'0'	$10 * 0 + 48 - 48 = 0$	x = 0
2	'2'	$10 * 0 + 50 - 48 = 2$	x = 2
7	'7'	$10 * 2 + 55 - 48 = 27$	x = 27
b	'b'		

Typ `int`

```
signed short int
unsigned short int
signed int
unsigned int
signed long int
unsigned long int
```

Deklaration von Integer-Konstanten

```
const a = 4, b = 3865;    /* Werte sind vom Typ 'signed int' */
const c = 48726;          /* long int */
const c = 48726u;         /* unsigned int */
const d = -124L;          /* long int */
const d = 324528375;      /* unsigned long int */
```

Operatoren

- **Arithmetische Operationen** (liefern bei ganzzahligen Operanden *immer* ein ganzzahliges Ergebnis):
  - **unäre Operatoren**:

    `++` (Inkrementierung)
    `--` (Dekrementierung)
  - **binäre Operatoren**:
    - **additive Operatoren**:

      `+` (Addition)
      `-` (Subtraktion)
    - **multiplikative Operatoren** (besitzen einen höheren Rang als additive Operatoren):

      `*` (Multiplikation)
      `/` (DIV, Ergebnis der ganzzahligen Division)
      `%` (MOD, Rest der ganzzahligen Division)
    - **bitweise Operationen** (besitzen einen niedrigeren Rang als additive Operationen):

      `<<`, `>>` (bitweises Verschieben nach links bzw. nach rechts)
      `&`, `|`, `^` (bitweise Konjunktion, Disjunktion und Alternative)
- **Vergleichsoperationen**: `==`, `<`, `>`, `<=`, `>=`, `!=`
  Vergleichsoperationen liefern einen Wahrheitswert.
- **Boolesche Operationen** (s. u.)

Boolesche Operatoren

Es stehen folgende Operationen für *logische* Verknüpfungen zur Verfügung:

!	Negation
&&	Konjunktion (UND-Verknüpfung)
||	Disjunktion (ODER-Verknüpfung)

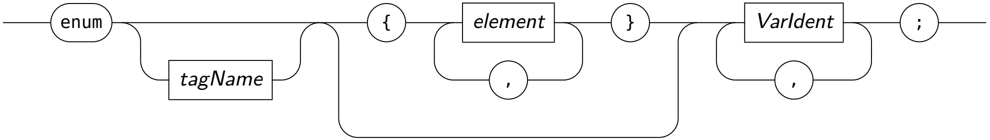
Insbesondere gilt: $!\mathtt{x} = \begin{cases} 0 & \text{wenn } \mathtt{x} \neq 0 \\ 1 & \text{wenn } \mathtt{x} = 0 \end{cases}$

Beispiel: Operationen auf `int`-Zahlen in C:

```c
int a = 4; int b = 7; int c = 0; int d = -3;
int e;

e = !d;      /* e erhält den Wert 0 */
e = !a;      /* e erhält den Wert 0 */
e = !c;      /* e erhält den Wert 1 */
e = !!d;     /* e erhält den Wert 1 */
e = a && b;  /* e erhält den Wert 1 */
e = a && c;  /* e erhält den Wert 0 */
e = !a && c; /* e erhält den Wert 0, Negation hat Vorrang */
e = a || c;  /* e erhält den Wert 1 */
```

Aufzählungstypen (Enumerate)

EnumType

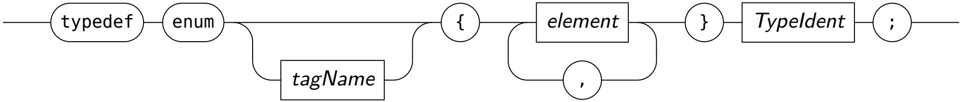


Beispiel:

```
enum {schwarz, weiss} f;
enum colour {rot,gelb,blau} f1,f2;
enum colour farbe;
```

Aufzählungstypen (Enumerate)

EnumTypeDeclaration



Beispiel:

```
. . .
typedef enum Tage {Mo, Di, Mi, Do, Fr, Sa, So} Wochentage;
const enum Tage Sonntag = So;  /* oder: const Wochentage Sonntag = So; */
                               /* aber auch: const Sonntag = So;
                                          bzw.: const int Sonntag = So; */
Wochentage anyday, f;          /* oder auch: enum Tage anyday, f; */
int x;

. . .

anyday = Di;
anyday++;             /*  (anyday == Mi) */
f = anyday;           /*  (f == Mi)  */
x = Do;               /*  (x == 3, automatische Typkonversion von Wochentage zu int) */
f = x + Mi;           /*  (f == Sa) */
f = Do + Sa           /*  (f == 8) */
f = (Wochentage) 4;   /*  (f == Fr) */
f = 12;               /*  (f == 12) */
Sonntag = x;          /*  Falsch! Sonntag als Konstante deklariert! */
```

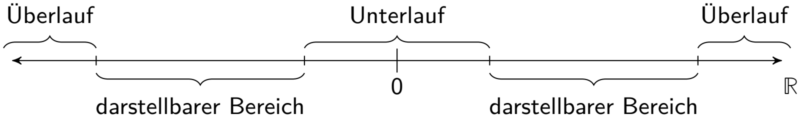
Reelle Zahlen

Grundtypen:

- `float`
- `double`
- `double` kann zu `long double` modifiziert werden

Beispiel Darstellungsweisen für Gleitkommazahlen:

- `37.52`
- `0.0`
- `7.35E13`
- `-0.375E-7`
- $\pm\texttt{0.}\underbrace{a_1 \dots a_n}_{\text{Mantisse}}\texttt{E}\pm\underbrace{b_1 \dots b_k}_{\text{Exponent}}$ mit $a_1 \neq 0, n \geq 1, k \geq 1, a_i, b_j \in \{0, \dots, 9\}$

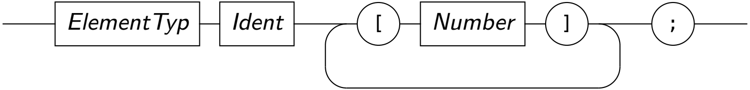


Darstellung, Wertebereich und Genauigkeit

Typ	Byte	Min.	Max.	Genauigkeit
float	4	$\pm$3.4E-38	$\pm$3.4E38	$\geq$ 6 Ziffern
double	8	$\pm$1.7E-308	$\pm$1.7E308	$\geq$ 10 Ziffern
long double	10	$\pm$1.2E-4932	$\pm$1.2E4932	$\geq$ 10 Ziffern

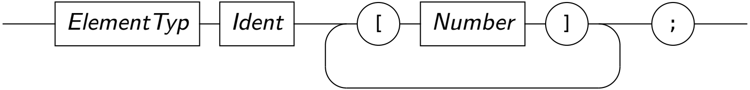
Feld (Array)

ArrayType



Feld (Array)

ArrayType

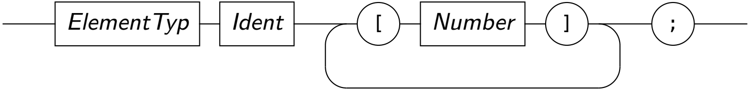


Deklaration von Feldvariablen:

```
int Feld[4];
char Letter[6];

int Feld[4] = {2, 7, 0, -4};
char Letter[6] = {'A', 'B', 'C', 'D', 'E', 'F'};
```

Feld (Array)

ArrayType



Deklaration von Feldvariablen:

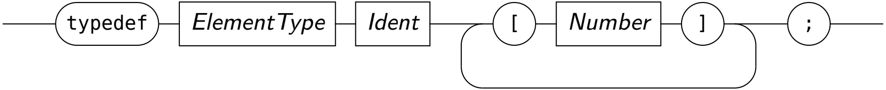
```
int Feld[4];
char Letter[6];

int Feld[4] = {2, 7, 0, -4};
char Letter[6] = {'A', 'B', 'C', 'D', 'E', 'F'};
```

Deklaration von Feldkonstanten:

```
const int Feld[4] = {2, 7, 0, -4};
const char Letter[6] = {'A', 'B', 'C', 'D', 'E', 'F'};
```

Feld (Array)

ArrayTypeDeclaration




```
typedef int array1[4];
typedef char array2[6];

array1 Feld;        /* Feld ist eine Variable des Typs array1 */
array2 Letter;      /* Letter ist eine Variable des Typs array2 */
```

Feld (Array)

Beispiel:

```
enum farben {rot, gruen, blau} color;
   /* color ist eine Variable des Typs enum farben */
float x[100][3];
. . .
color = gruen;
x[87][color] = (float) 3.7;
x[45][rot] = (float) -46.4E-12;
```

Beispiel: Matrixmultiplikation

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}, \qquad B = \begin{pmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nq} \end{pmatrix}.$$

Durch die Multiplikation entsteht die (m, q)-Matrix C:

$$C = \begin{pmatrix} c_{11} & \cdots & c_{1q} \\ \vdots & \ddots & \vdots \\ c_{m1} & \cdots & c_{mq} \end{pmatrix} \text{ mit } c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} .$$

$$\begin{pmatrix} 5 & 0 & -2 \\ 1 & -3 & 4 \end{pmatrix} \cdot \begin{pmatrix} 2 & -1 \\ 5 & 3 \\ 4 & -6 \end{pmatrix} = \begin{pmatrix} 5 \cdot 2 + 0 \cdot 5 + (-2) \cdot 4 & 5 \cdot (-1) + 0 \cdot 3 + (-2) \cdot (-6) \\ 1 \cdot 2 + (-3) \cdot 5 + 4 \cdot 4 & 1 \cdot (-1) + (-3) \cdot 3 + 4 \cdot (-6) \end{pmatrix}$$

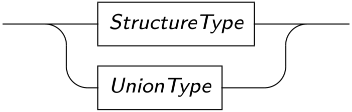
$$= \begin{pmatrix} 10 + 0 + (-8) & -5 + 0 + 12 \\ 2 + (-15) + 16 & -1 + (-9) + (-24) \end{pmatrix} = \begin{pmatrix} 2 & 7 \\ 3 & -34 \end{pmatrix}$$

Beispiel: Matrixmultiplikation

```
. . .
const m = 5, n = 3, q = 4;

float a[5][3],    /* m, n ; in C nur konstante Ausdruecke zugelassen! */
      b[3][4],    /* n, q */
      c[5][4];    /* m, q */
int i, j, k;
float s;

for (i = 0; i < m; i++)
  for (j = 0; j < q; j++)
  { s = 0;
    for (k = 0; k < n; k++)
      s = s + a[i][k] * b[k][j];
    c[i][j] = s;
  }
. . .
```

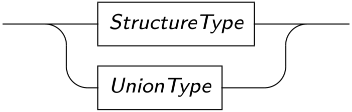
Verbund (Structure, Union)

RecordType

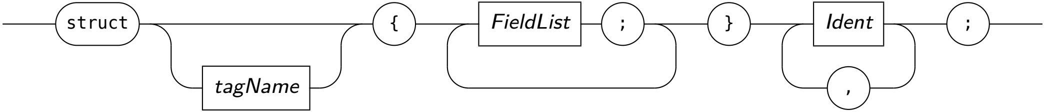


Verbund (Structure, Union)

RecordType

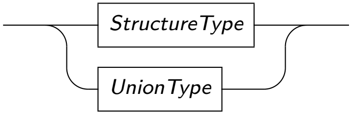


StructureType

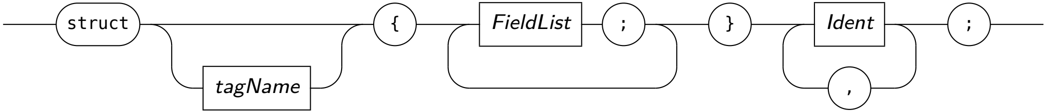


Verbund (Structure, Union)

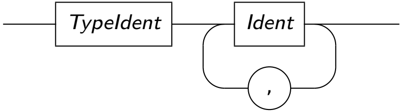
RecordType



StructureType



FieldList



Verbund (Structure, Union)

Beispiele:

```
struct { ... } a, b, c;
```

Verbund (Structure, Union)

Beispiele:

```
struct { ... } a, b, c;

struct beispiel { ... } a, b, c;
struct beispiel x, y, z;
```

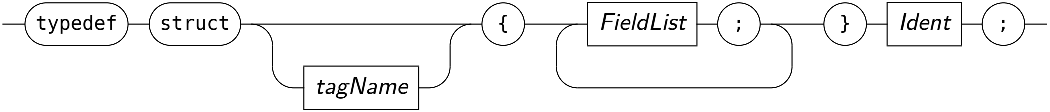
Verbund (Structure, Union)

Beispiele:

```
struct { ... } a, b, c;

struct beispiel { ... } a, b, c;
struct beispiel x, y, z;

struct beispiel_2 {int k, l; float m;} p, q;
```

Verbund (Structure, Union)

StructureTypeDeclaration

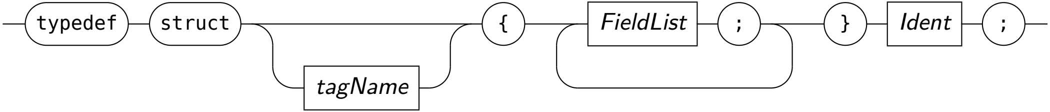


Beispiel: Gegeben sei die folgende Typdeklaration:

```
typedef struct beispiel_3 { ... } mytype;
```

Verbund (Structure, Union)

StructureTypeDeclaration



Beispiel: Gegeben sei die folgende Typdeklaration:

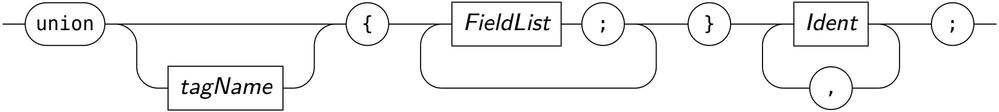
```
typedef struct beispiel_3 { ... } mytype;
```

Dann haben die folgenden beiden Zeilen dieselbe Bedeutung:

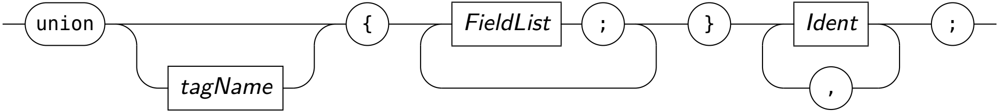
```
struct beispiel_3 x, y, z;
mytype x, y, z;
```

```
/* Beispiel für structure */
. . .
typedef struct personal { char name[30];
                          enum {m, w, i} geschlecht;
                          enum {verh, led, gesch, verw} famstand;
                          unsigned int gehalt;
                          struct {short int tag, monat, jahr;} gebdat;
                        } person;

person egon;
. . .
egon.gehalt = 8000;
strcpy(egon.name, "Maier"); /* kopiert die Zeichen (ASCII-Code) der String-Kon-
                               stanten "Maier" nacheinander auf Adressen ab der
                               Adresse der Variablen egon.name und schreibt auf
                               die nächstfolgende Adresse den Wert 0 */
egon.geschlecht = m;
egon.famstand = led;
egon.gebdat.tag = 22;
egon.gebdat.monat = 12;
egon.gebdat.jahr = 1960;
. . .
```

UnionType



UnionType




```
/* Beispiel für union */
. . .
typedef struct kfz_typ { char hersteller[30];
                         enum {pkw, bus, lkw} art;
                         float preis;
                         union { short int vmax;
                                 short int sitzplaetze;
                                 float zuladung; } eigenschaft; } kfz;

kfz auto1, auto2, auto3;
. . .
auto1.art = pkw;
auto1.eigenschaft.vmax = 180;
```

```
auto2.art = bus;
auto2.eigenschaft.sitzplaetze = 45;

auto3.art = lkw;
auto3.eigenschaft.zuladung = 25.5f;
. . .
```

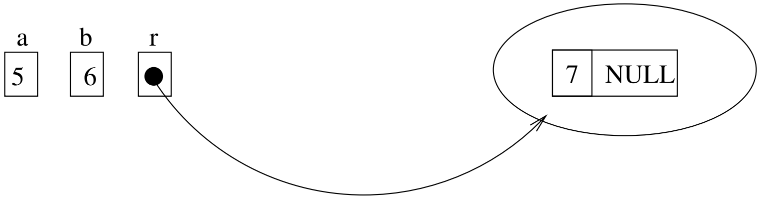
Zeigervariable

```
#include <stdio.h>
#include <stdlib.h>

typedef int feld[100];
typedef feld *P_feld;
P_feld a;

int main()
{
  a = (P_feld) malloc(sizeof(feld));
  . . .
  (*a)[2] = 7;
  . . .
  free(a);
  . . .
}
```

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct ele *zeiger;
typedef struct ele { int zahl;
                     zeiger next;
                   } element;

int a, b; zeiger r;

void A(int x, int y, int *z)
{ int hilf; zeiger p;

  hilf = (x + y) * *z;
  p = (zeiger) malloc(sizeof(element));
  p->zahl = hilf;      /* indirekter Zugriff, Dereferenzierung,  */
  p->next = r;         /* gleichbedeutend mit: (*p).zahl = hilf; */
  r = p;

  if (hilf < 100)
  { *z = *z + 5;       /* z ist Referenzparameter, deshalb hier Zugriff auf den Wert
                          mit *,                                                   */
    A(x + y, 10, z);   /* aber hier rekursiver Aufruf von A mit drittem Parameter als
                          Referenzparameter, also Übergabe einer Adresse, die in z
                          gespeichert ist!                                         */
    *z = 2 * *z;
  }
}

int main()
{ a = 5;
  b = 6;
  r = (zeiger) malloc(sizeof(element));
  r->zahl = 7;         /* gleichbedeutend mit (*r).zahl = 7; */
  r->next = NULL;
  A(a, b, &b);
  return 0;
}
```

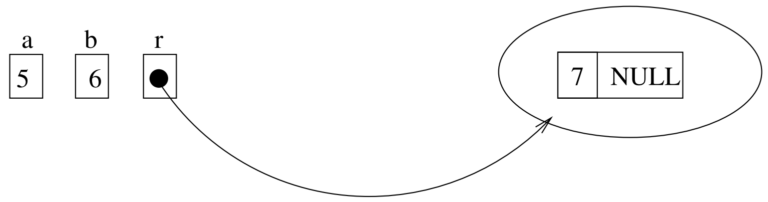
Zeigervariable

Im Hauptprogramm

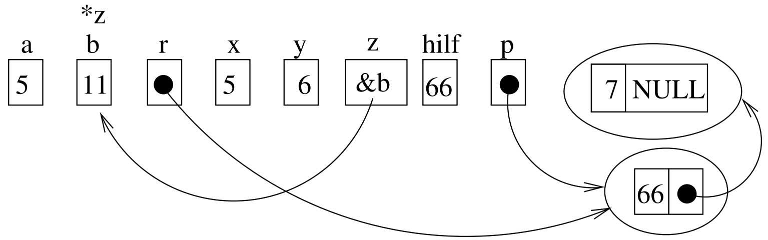


Zeigervariable

Im Hauptprogramm

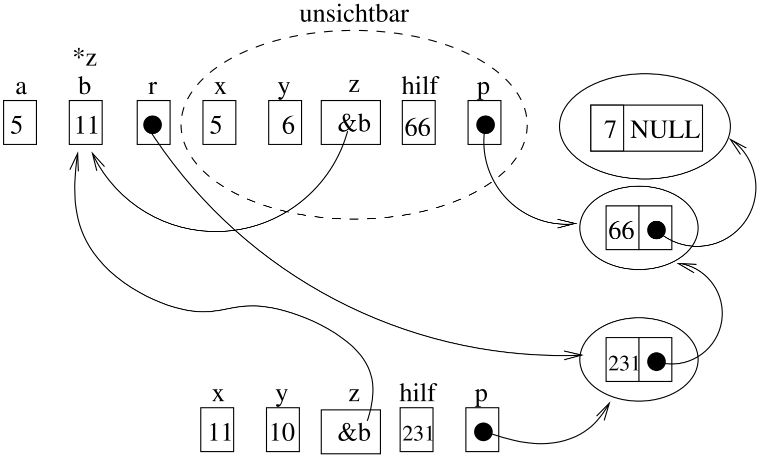


vor dem 2. Aufruf von A



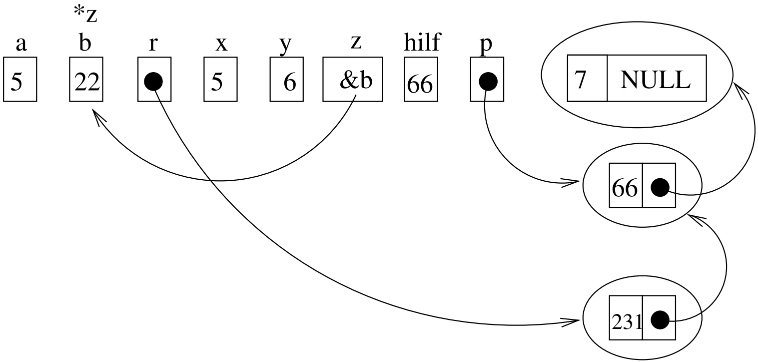
Zeigervariable

vor dem 1. Rücksprung



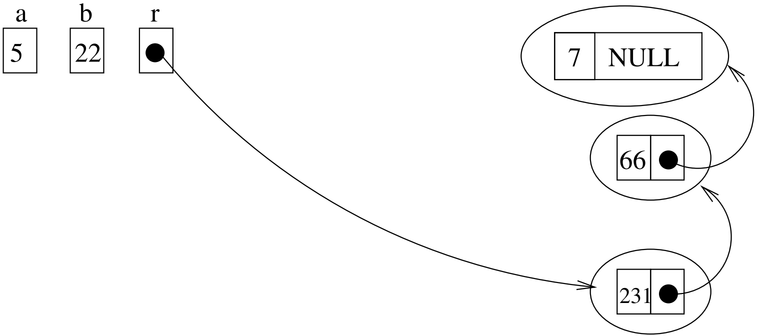
Zeigervariable

nach Verlassen des 2. Aufrufes von A



Zeigervariable

nach Verlassen des 1. Aufrufes von A



Einfach verkettete Listen

```
typedef struct nodeelem *Ptr;
typedef struct nodeelem { int key;
                          Ptr next;
                          int data; } node;
Ptr h,p,q; int n,i;
```

Aufbau einer verketteten Liste

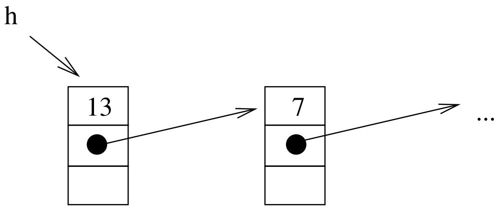
```
scanf("%d", &n);
q = (Ptr) malloc(sizeof(node));
h = q;                          /* h haelt den Listenanfang fest */
q->key = n;
q->next = NULL;
for (i = 1; i <= 3; i++)
{ scanf("%d", &n);
  p = (Ptr) malloc(sizeof(node));
  p->key = n;
  p->next = NULL;
  q->next = p;
  q = p;                        /* q zeigt auf das letzte Element */
}
```

Einfügen in die verkettete Liste ...

...an den Anfang (auf den h zeigt):

```
q = (Ptr) malloc(sizeof(node));
q->next = h;
h = q;
```

vorher:

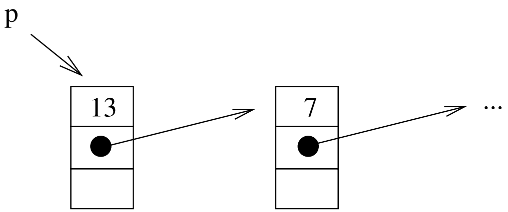


Einfügen in die verkettete Liste ...

...hinter ein durch einen Zeiger p bezeichnetes Objekt:

```
q = (Ptr) malloc(sizeof(node));
q->next = p->next;
p->next = q;
```

vorher:

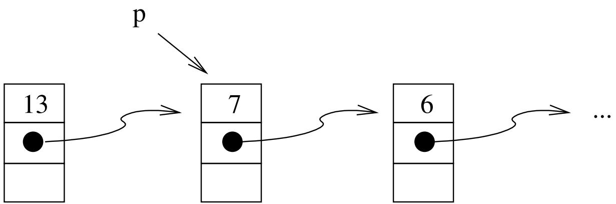


Einfügen in die verkettete Liste ...

...vor ein durch einen Zeiger p bezeichnetes Objekt:

```
q = (Ptr) malloc(sizeof(node));
q->key = p->key;
q->next = p->next;
q->data = p->data;
p->next = q;
p->key = 9;      /* *p ist neues Datenobjekt */
```

vorher:



Ausketten und Archivieren von Elementen

Annahme: Zwei Listen; aus der zweiten Liste den Nachfolger eines durch p bezeichneten Datenobjektes an den Anfang der ersten Liste, bezeichnet durch h, setzen.

```
r = p->next;
p->next = r->next;
r->next = h;
h = r;
```

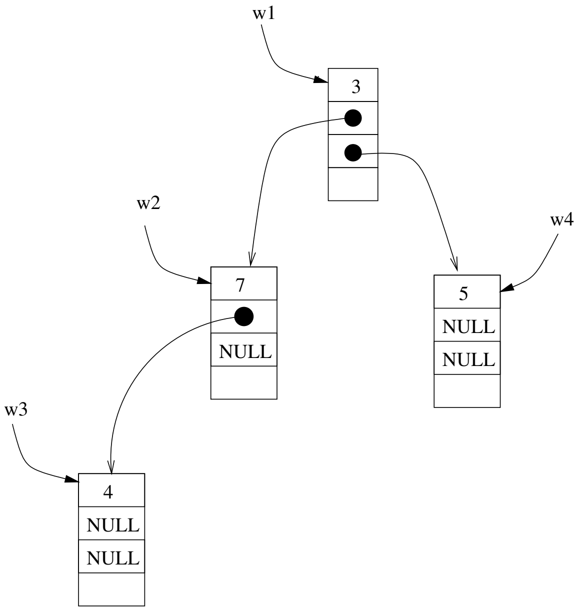
Doppelt verkettete Listen

```
typedef struct nodeelem *LPtr;
typedef struct nodeelem { int key;
                          LPtr next;
                          LPtr prev;
                          ... data;  } node;
LPtr  h, p, q;
```

Aufbau einer doppelt-verketteten Liste

```
scanf("%d", &n);
q = (LPtr) malloc(sizeof(node));
q->key = n;
q->prev = NULL;              /* erstes Element hat keinen Vorgaenger */
h = q;                       /* Listenanfang                         */
for (i = 1; i <= 10; i++)
{ scanf("%d", &n);
  p = (LPtr) malloc(sizeof(node));
  p->key = n;
  q->next = p; p->prev = q;
  q = p;
  q->next = NULL;            /* letztes Element hat keinen Nachfolger */
}
```

Bäume




```
typedef struct nodeelem *BPtr;
typedef struct nodeelem
               { int key;
                 BPtr left, right;
                 ... data;
               } node;
```

Bäume

```c
int hoehe(BPtr wz)
{ int h1, h2;
   /* label1 */
  if (wz == NULL) return 0;
  h1 = hoehe(wz->left);  /* $1 */
   /* label2 */
  h2 = hoehe(wz->right); /* $2 */
   /* label3 */
  return max(h1, h2)+1;
}
```

Bäume

```
/*laengsterPfad*/
typedef struct nodeelem *BPtr;
typedef struct nodeelem { int key;
                          BPtr left, right;
                          ... data;         } node;
. . .
int hoehe(. . .){. . .}       /* Berechnet die Höhe eines binären Baumes.    */
void eingabe(BPtr *wz){. . .}  /* Realisiert Eingabe eines binären Baumes,
                                   Zeiger auf die Wurzel wird zurückgeliefert. */
int main()
{ int h; BPtr w;
  eingabe(&w);
  h = hoehe(w); /* $3 */
  /* label4 */
  . . .
}
```

Haltepunkt	RM	Umgebung											
		1	2	3	4	5	6	7	8	9	10	11	12
label1	3	h1	h2	*wz*									
		?	?	*w1*									
label1	$1:3$				h1	h2	*wz*						
		?	?	*w1*	?	?	*w2*						
label1	$1:1:3$							h1	h2	*wz*			
		?	?	*w1*	?	?	*w2*	?	?	*w3*			
label1	$1:1:1:3$										h1	h2	*wz*
		?	?	*w1*	?	?	*w2*	?	?	*w3*	?	?	*NULL*
label2	$1:1:3$							h1	h2	*wz*			
		?	?	*w1*	?	?	*w2*	0	?	*w3*			
label1	$2:1:1:3$										h1	h2	*wz*
		?	?	*w1*	?	?	*w2*	0	?	*w3*	?	?	*NULL*
label3	$1:1:3$							*wz*	h1	h2	*wz*		
		?	?	*w1*	?	?	*w2*	0	0	*w3*			
label2	$1:3$				h1	h2	*wz*						
		?	?	*w1*	1	?	*w2*						
label1	$2:1:3$							h1	h2	*wz*			
		?	?	*w1*	1	?	*w2*	?	?	*NULL*			

. . .

. . .

Haltepunkt	RM	Umgebung											
		1	2	3	4	5	6	7	8	9	10	11	12
label3	$1:3$				h1	h2	*wz*						
		?	?	*w1*	1	0	*w2*						
label2	3	h1	h2	*wz*									
		2	?	*w1*									
label1	$2:3$				h1	h2	*wz*						
		2	?	*w1*	?	?	*w4*						
label1	$1:2:3$							h1	h2	*wz*			
		2	?	*w1*	?	?	*w4*	?	?	*NULL*			
label2	$2:3$				h1	h2	*wz*						
		2	?	*w1*	0	?	*w4*						
label1	$2:2:3$							h1	h2	*wz*			
		2	?	*w1*	0	?	*w4*	?	?	*NULL*			
label3	$2:3$				h1	h2	*wz*						
		2	?	*w1*	0	0	*w4*						
label3	3	h1	h2	*wz*									
		2	1	*w1*									

Beispiel: Ableitungsbäume

$$r1: S ::= CA$$
$$r2: S ::= A$$
$$r3: C ::= \mathtt{c}C$$
$$r4: C ::= \mathtt{c}$$
$$r5: A ::= \mathtt{a}A\mathtt{b}$$
$$r6: A ::= \mathtt{a}$$

Beispiel: Ableitungsbäume

$r1\colon S ::= CA$

$r2\colon S ::= A$

$r3\colon C ::= \mathtt{c}C$

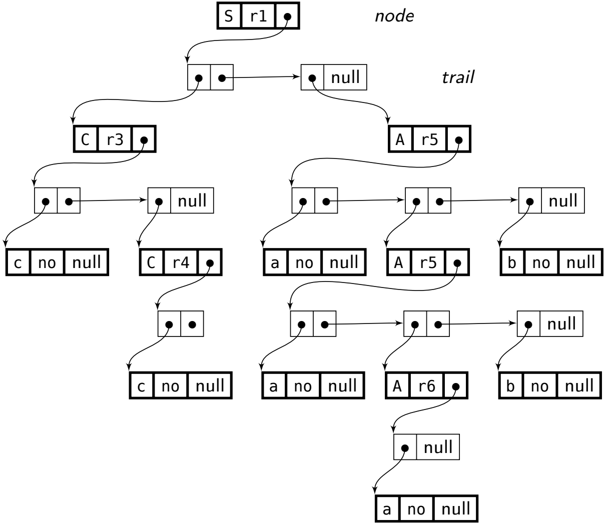
$r4\colon C ::= \mathtt{c}$

$r5\colon A ::= \mathtt{a}A\mathtt{b}$

$r6\colon A ::= \mathtt{a}$

```
typedef enum {S, A, C, a, b, c} set_of_symbols;
typedef enum {r1, r2, r3, r4, r5, r6, no} set_of_rules;

typedef struct nodeelem *pointer_to_node;
typedef struct trailelem *pointer_to_trail;

typedef struct nodeelem { set_of_symbols label;
                          set_of_rules rule;
                          pointer_to_trail next; } node;

typedef struct trailelem { pointer_to_node item;
                           pointer_to_trail next; } trail;
```

Beispiel: ein Ableitungsbaum

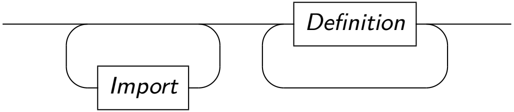


Inhaltsverzeichnis I

Inhaltsverzeichnis II

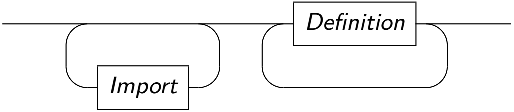
Modularisierungskonzept

DefinitionModule

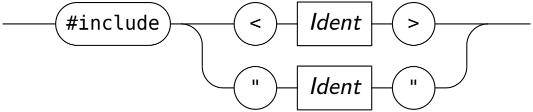


Modularisierungskonzept

DefinitionModule

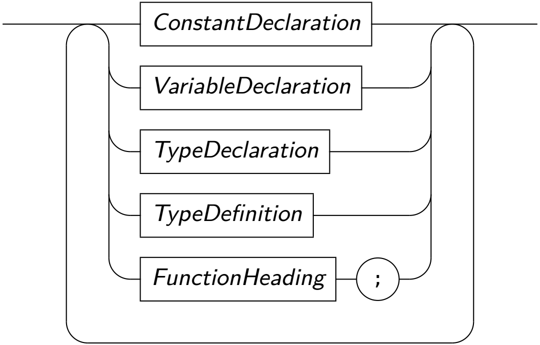


Import



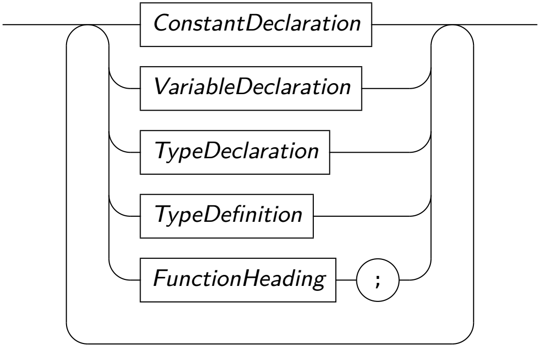
Modularisierungskonzept

Definition



Modularisierungskonzept

Definition



TypeDefinition

```
/* Header-File pushdown.h */

typedef struct ele *pushdown;

void CreatePushdown(pushdown *s);    /* erzeugt einen leeren Keller           */
void Push(pushdown *s, int x);       /* legt ein Element x auf dem Keller ab  */
void Pop(pushdown *s, int *x);       /* entfernt das oberste Element vom
                                        Keller und speichert den Wert auf *x  */
int Empty (pushdown s);              /* testet, ob pushdown leer ist          */
```

```c
/* Header-File pushdown.h */

typedef struct ele *pushdown;

void CreatePushdown(pushdown *s);    /* erzeugt einen leeren Keller           */
void Push(pushdown *s, int x);       /* legt ein Element x auf dem Keller ab  */
void Pop(pushdown *s, int *x);       /* entfernt das oberste Element vom
                                        Keller und speichert den Wert auf *x  */
int Empty (pushdown s);              /* testet, ob pushdown leer ist          */
```

```c
. . .
#include "pushdown.h"

. . .
pushdown t, u;
int x;
. . .
{ . . .
  CreatePushdown(&u);
  CreatePushdown(&t);
  Push(&u, 25);
  Push(&t, 7);
  . . .
  if (!Empty(u))
  { Pop(&u, &x);
    Push(&t, x);
  }
  . . .
}
```

Zusammenwirken des Moduls pushdown

Modul 1

Definitionsmodul

```
/* Header-File pushdown.h */

typedef struct ele *pushdown

void CreatePushdown(pushdown *s); /* erzeugt einen leeren Keller */
void Push(pushdown *s, int x);    /* legt ein Element x auf dem Keller ab*/
void Pop(pushdown *s, int *x);    /* entfernt das oberste Element vom Keller
                                     und speichert den Wert auf *x */
int Empty(pushdown *s);           /* testet, ob pushdown leer ist */
```

Implementierungsmodul

```
/* Implementierungsmodul pushdown.c*/

#include <stdlib.h>
#include "pushdown.h"

typdef struct ele {int key; pushdown next;} element;

void CreatePushdown(pushdown *s)
{ *s=NULL;
}

void Push(pushdown *s, int x)
{ pushdown top;
  top= (pushdown) malloc (sizeof(element)) /* erzeuge neues oberes Kellerelement*/
  top->key= x;
  top->next= *s;
  *s=top;
}

void Pop(pushdown *s, int *x)
{ pushdown top;
  top= *s;
  *x= (*s)->key;
  *s= (*s)->next;
  free(top);                              /* entferne oberes Kellerelement*/
}

int Empty(pushdown s)
{ return s==NULL;
}
```

Import

Modul 2

Definitionsmodul

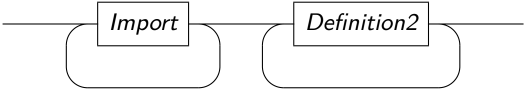
```
...

...
```

Implementierungsmodul

```
...
#include "pushdown.h"

...
pushdown t,u;
int x;
...
{
 CreatePushdown(&u);
 CreatePushdown(&t);
 Push(&u, 25);
 Push(&t, 7);
 ...
 if (!Empty(u))
 { Pop(&u, &x);
   Push(&u, x);
 }
 ...
}
```

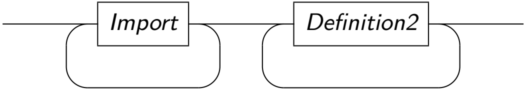
Modularisierungskonzept

ImplementationModule

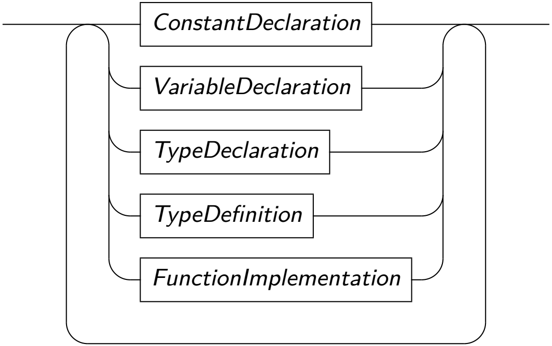


Modularisierungskonzept

ImplementationModule



Definition2



```c
/* Implementierungsmodul pushdown.c */

#include <stdlib.h>
#include "pushdown.h"

typedef struct ele { int key;
                     pushdown next;} element;

void CreatePushdown(pushdown *s)
{ *s = NULL;
}

void Push(pushdown *s, int x)
{ pushdown top;

  top = (pushdown) malloc(sizeof(element));
  top->key = x;
  top->next = *s;
  *s = top;
}

/* 'Pop' nimmt an, dass 's' kein leerer 'pushdown' ist.
void Pop(pushdown *s, int *x)
{ pushdown top;

  top = *s;
  *x = (*s)->key;
  *s = (*s)->next;
  free(top);
}

int Empty(pushdown s)
{ return s==NULL;
}
```

Inhaltsverzeichnis I

Inhaltsverzeichnis II

Teil II – Algorithmische Problemstellungen

Beispiel: Auffinden eines Namens im Telefonbuch

Algorithmus Lineares Suchen

```
Beginnend mit der ersten Seite wird in der Reihenfolge der Seiten das
Telefonbuch durchsucht.
```

Beurteilung: korrekt, aber sehr ineffizient; „linearer Aufwand (in der Anzahl der Einträge)“.

Beispiel: Auffinden eines Namens im Telefonbuch

Algorithmus Binäres Suchen

```
Setze l (erste Seite eines Seitenbereichs)  = Anfangsseite des Telefonbuches
Setze r (letzte Seite eines Seitenbereichs) = Schlussseite des Telefonbuches

Wiederhole folgende Schritte bis Name gefunden bzw. nicht gefunden
   * Besteht der Seitenbereich aus nur einer Seite, dann durchsuche diese.
     Wird Name gefunden, dann Telefonnummer merken und Ende des Suchens.
     Wird Name nicht gefunden, dann Feststellung, dass Name nicht im
      Telefonbuch steht und Ende des Suchens.
   * Schlage Telefonbuch etwa in der Mitte des Seitenbereichs l...r auf; die
     aufgeschlagene Seite habe die Nummer m.
   * Wenn gesuchter Name im Bereich l...m liegen müsste, dann setze r=m,
     andernfalls setze l=m und arbeite mit diesem neuen Bereich weiter.
```

Beurteilung: korrekt und effizient; „logarithmischer Aufwand“.

Beispiel: MinAlter

Algorithmus MinAlter

Eingabe Eine Folge $a_1, \dots, a_n$ von positiven, ganzen Zahlen.

Ausgabe der kleinste Positionsindex j mit $a_j = \min\{a_1, \dots, a_n\}$.

Verfahren Zusätzliche Variablen: x (für das Alter), i (als Zählvariable);

1. (*Initialisierung*) Setze $j := 1, x := a_j$ und $i := 2$.
2. (*Suchlauf*)
   Solange $i \leq n$ gilt, wiederhole:
   - falls $a_i < x$, setze $j := i$ und $x := a_j$
   - erhöhe i um 1
3. Ausgabe von j als Ergebnis

$T\colon \mathbb{N}^{\{0,\ldots,400\}} \to \mathbb{R}$
$T(w)$: Zeit (in ms), um jüngste Person für w zu finden

Konkrete Folge $w \in \mathbb{N}^{\{0,\ldots,400\}}$:

0	1						400
19	22	21	19	18	19	...	20

auf Laptop: 10 ms
auf Großrechner: 0.01 ms

T: $\mathbb{N}^{\{0,\dots,400\}} \to \mathbb{R}$
$T(w)$: Zeit (in ms), um jüngste Person für w zu finden

Konkrete Folge $w \in \mathbb{N}^{\{0,\dots,400\}}$:

0	1						400
19	22	21	19	18	19	...	20

auf Laptop: 10 ms
auf Großrechner: 0.01 ms

$\downarrow$ 1. Abstraktion
(Analyse unabhängig vom Rechner)

T: $\mathbb{N}^{\{0,\dots,400\}} \to \mathbb{N}$
$T(w)$: Anzahl der Rechen-/Vergleichsschritte, um jüngste Person von w zu finden

Konkrete Folge $w \in \mathbb{N}^{\{0,\dots,400\}}$:

0	1						400
19	22	21	19	18	19	...	20

Anzahl der Rechen-/Vergleichsschritte

T: $\mathbb{N}^{\{0,\ldots,400\}} \to \mathbb{R}$
$T(w)$: Zeit (in ms), um jüngste Person für w zu finden

Konkrete Folge $w \in \mathbb{N}^{\{0,\ldots,400\}}$:

0	1						400
19	22	21	19	18	19	...	20

auf Laptop: 10 ms
auf Großrechner: 0.01 ms

1. Abstraktion
(Analyse unabhängig vom Rechner)

T: $\mathbb{N}^{\{0,\ldots,400\}} \to \mathbb{N}$
$T(w)$: Anzahl der Rechen-/Vergleichsschritte, um jüngste Person von w zu finden

Konkrete Folge $w \in \mathbb{N}^{\{0,\ldots,400\}}$:

0	1						400
19	22	21	19	18	19	...	20

Anzahl der Rechen-/Vergleichsschritte

2. Abstraktion
(Analyse des Aufwandes in Abhängigkeit von Problemgröße n)

Beliebige Folge $w \in \bigcup_{n\geq 0} \mathbb{N}^{\{0,\ldots,n-1\}}$:

0	1						$n-1$
a_1	a_2	a_3	a_4	a_5	a_6	...	a_n

T: $\mathbb{N}^{\{0,\dots,400\}} \to \mathbb{R}$
$T(w)$: Zeit (in ms), um jüngste Person für w zu finden

Konkrete Folge $w \in \mathbb{N}^{\{0,\dots,400\}}$:

0	1						400
19	22	21	19	18	19	...	20

auf Laptop: 10 ms
auf Großrechner: 0.01 ms

1. Abstraktion
(Analyse unabhängig vom Rechner)

T: $\mathbb{N}^{\{0,\dots,400\}} \to \mathbb{N}$
$T(w)$: Anzahl der Rechen-/Vergleichsschritte, um jüngste Person von w zu finden

Konkrete Folge $w \in \mathbb{N}^{\{0,\dots,400\}}$:

0	1						400
19	22	21	19	18	19	...	20

Anzahl der Rechen-/Vergleichsschritte

2. Abstraktion
(Analyse des Aufwandes in Abhängigkeit von Problemgröße n)

Beliebige Folge $w \in \bigcup_{n\geq 0} \mathbb{N}^{\{0,\dots,n-1\}}$:

0	1						$n-1$
a_1	a_2	a_3	a_4	a_5	a_6	...	a_n

best case

average case

worst case

$T_{\text{best-case}}(n) = \min\{T(w) \mid w \in \mathbb{N}^n\}$

$T_{\text{average-case}}(n) =$ erwartete Anzahl bei Gleichverteilung aller Eingaben der Lösungen

$T_{\text{worst-case}}(n) = \max\{T(w) \mid w \in \mathbb{N}^n\}$

Beispiel: MinAlter

Operation	Aufwand	Wie oft wird Operation ausgeführt?
Setze $j = 1$	$1\,E$	1
Setze $x = a_j$	$1\,E$	1
Setze $i = 2$	$1\,E$	1
Teste $i \leq n$	$1\,E$	n
Teste $a_i < x$	$1\,E$	$n-1$
Setze $j = i, x = a_j$	$2\,E$	? ($n-1$ bis 0)
Setze $i = i+1$	$1\,E$	$n-1$
Ausgabe	$1\,E$	1

$$T_{\text{best-case}}(n) = (3n+2)$$
$$T_{\text{worst-case}}(n) = 5n$$
$$T_{\text{average-case}}(n) = (3n+2) + 2\sum_{i=2}^{n} \frac{1}{i}$$

Komplexität

Sei $f\colon \mathbb{N} \to \mathbb{N}$, dann definiere

$$O(f) = \{g\colon \mathbb{N} \to \mathbb{N} \mid \text{es gibt } c_1 > 0, c_2 > 0 \text{ und } n_0\colon \text{für jedes } n \geq n_0\colon g(n) \leq c_1 \cdot f(n) + c_2\}$$

$$\Omega(f) = \{g\colon \mathbb{N} \to \mathbb{N} \mid \text{es gibt } c > 0 \text{ und } n_0 > 0\colon \text{für jedes } n > n_0\colon g(n) \geq c \cdot f(n)\}$$

Komplexität

Sei $f\colon \mathbb{N} \to \mathbb{N}$, dann definiere

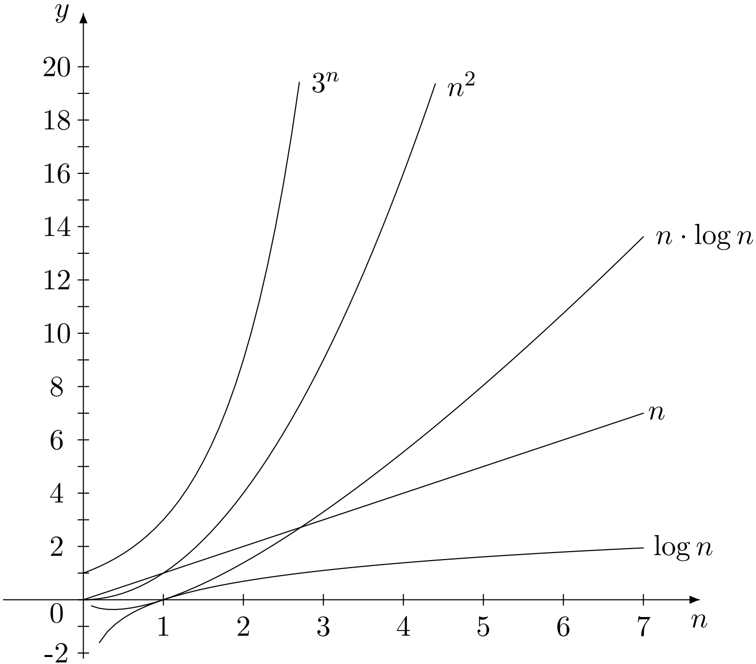
$$O(f) = \{g\colon \mathbb{N} \to \mathbb{N} \mid \text{es gibt } c_1 > 0, c_2 > 0 \text{ und } n_0\colon \text{für jedes } n \geq n_0\colon g(n) \leq c_1 \cdot f(n) + c_2\}$$

$$\Omega(f) = \{g\colon \mathbb{N} \to \mathbb{N} \mid \text{es gibt } c > 0 \text{ und } n_0 > 0\colon \text{für jedes } n > n_0\colon g(n) \geq c \cdot f(n)\}$$

Wichtige und häufig auftretende Wachstumsklassen sind:

- logarithmisches Wachstum: $O(\log(n))$,
- lineares Wachstum: $O(n)$,
- $n \cdot \log(n)$-Wachstum: $O(n \cdot \log(n))$,
- polynomielles Wachstum: $O(n^k)$ für ein $k \in \mathbb{N}$,
- exponentielles Wachstum: $O(2^n)$

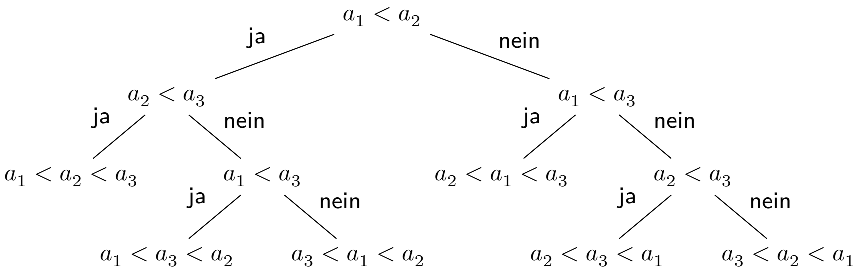
Komplexität



Inhaltsverzeichnis I

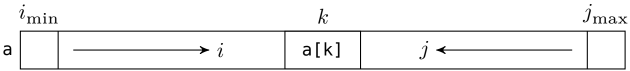
Inhaltsverzeichnis II

Sortieren


$a_1 < a_2$
ja
nein
$a_2 < a_3$
$a_1 < a_3$
ja
nein
ja
nein
$a_1 < a_2 < a_3$
$a_1 < a_3$
$a_2 < a_1 < a_3$
$a_2 < a_3$
ja
nein
ja
nein
$a_1 < a_3 < a_2$
$a_3 < a_1 < a_2$
$a_2 < a_3 < a_1$
$a_3 < a_2 < a_1$

Quicksort

```
void quicksort(int a[], int L, int R)  // L und R bezeichnen die linke bzw. rechte
                                       // Grenze des zu sortierenden Teils von a
{ int i, j, w, x, k;

  i = L;  j = R;                       // i und j durchlaufen a von links bzw. rechts
  k = (L+R) / 2;  x = a[k];            // x wird Pivotelement genannt

  do
  { while (a[i] < x) i = i + 1;
    while (a[j] > x) j = j - 1;
    if (i <= j)
    { w = a[i];                        //
      a[i] = a[j];                     // hier werden a[i] und a[j] getauscht
      a[j] = w;                        //
      i = i + 1;  j = j - 1;
    }
  }
  while (i <= j);

  if (L < j) quicksort(a, L, j);
  if (R > i) quicksort(a, i, R);
}
```



Beispiel

Aufruf: `quicksort(a, 0, 6);`

$i = 0,\ j = 6,\ k = 3,\ x = 5$

Beispiel

Aufruf: `quicksort(a, 0, 6);`
$i = 0,\ j = 6,\ k = 3,\ x = 5$

nach Zeile 9:

$$\begin{array}{cccccccc} a: & 7 & 21 & 9 & 5 & 2 & 3 & 14 \\ & \uparrow & & & & & \uparrow & \\ & i & & & & & j & \end{array}$$

$a[0] = 7 \not< 5 = x$
$a[5] = 3 \not> 5 = x$

Beispiel

Aufruf: `quicksort(a, 0, 6);`

$i = 0,\ j = 6,\ k = 3,\ x = 5$

nach Zeile 9:

$$\begin{array}{cccccccc} a: & 7 & 21 & 9 & 5 & 2 & 3 & 14 \\ & \uparrow & & & & & \uparrow & \\ & i & & & & & j & \end{array}$$

$a[0] = 7 \not< 5 = x$
$a[5] = 3 \not> 5 = x$

nach Zeile 9:

$$\begin{array}{cccccccc} a: & 3 & 21 & 9 & 5 & 2 & 7 & 14 \\ & & \uparrow & & & \uparrow & & \\ & & i & & & j & & \end{array}$$

$a[1] = 21 \not< 5 = x$
$a[4] = 2 \not> 5 = x$

Beispiel

Aufruf: `quicksort(a, 0, 6);`

$i = 0,\ j = 6,\ k = 3,\ x = 5$

nach Zeile 9:

$$\begin{array}{cccccccc} a: & 7 & 21 & 9 & 5 & 2 & 3 & 14 \\ & \uparrow & & & & & \uparrow & \\ & i & & & & & j & \end{array}$$

$a[0] = 7 \not< 5 = x$
$a[5] = 3 \not> 5 = x$

nach Zeile 9:

$$\begin{array}{cccccccc} a: & 3 & 21 & 9 & 5 & 2 & 7 & 14 \\ & & \uparrow & & & \uparrow & & \\ & & i & & & j & & \end{array}$$

$a[1] = 21 \not< 5 = x$
$a[4] = 2 \not> 5 = x$

nach Zeile 9:

$$\begin{array}{cccccccc} a: & 3 & 2 & 9 & 5 & 21 & 7 & 14 \\ & & & \uparrow & \uparrow & & & \\ & & & i & j & & & \end{array}$$

$a[2] = 9 \not< 5 = x$
$a[3] = 5 \not> 5 = x$

Beispiel

Aufruf: `quicksort(a, 0, 6);`

$i = 0,\ j = 6,\ k = 3,\ x = 5$

nach Zeile 9:

$$\begin{array}{cccccccc} a: & 7 & 21 & 9 & 5 & 2 & 3 & 14 \\ & \uparrow & & & & & \uparrow & \\ & i & & & & & j & \end{array}$$

$$a[0] = 7 \not< 5 = x$$
$$a[5] = 3 \not> 5 = x$$

nach Zeile 9:

$$\begin{array}{cccccccc} a: & 3 & 21 & 9 & 5 & 2 & 7 & 14 \\ & & \uparrow & & & \uparrow & & \\ & & i & & & j & & \end{array}$$

$$a[1] = 21 \not< 5 = x$$
$$a[4] = 2 \not> 5 = x$$

nach Zeile 9:

$$\begin{array}{cccccccc} a: & 3 & 2 & 9 & 5 & 21 & 7 & 14 \\ & & & \uparrow & \uparrow & & & \\ & & & i & j & & & \end{array}$$

$$a[2] = 9 \not< 5 = x$$
$$a[3] = 5 \not> 5 = x$$

nach Zeile 9:

$$\begin{array}{cccccccc} a: & 3 & 2 & 5 & 9 & 21 & 7 & 14 \\ & & & \uparrow & \uparrow & & & \\ & & & j & i & & & \end{array}$$

Beispiel

Aufruf: `quicksort(a, 0, 6);`

$i = 0,\ j = 6,\ k = 3,\ x = 5$

nach Zeile 9:

$$\begin{array}{lccccccc} a: & 7 & 21 & 9 & 5 & 2 & 3 & 14 \\ & \uparrow & & & & & \uparrow & \\ & i & & & & & j & \end{array}$$

$a[0] = 7 \not< 5 = x$
$a[5] = 3 \not> 5 = x$

nach Zeile 9:

$$\begin{array}{lccccccc} a: & 3 & 21 & 9 & 5 & 2 & 7 & 14 \\ & & \uparrow & & & \uparrow & & \\ & & i & & & j & & \end{array}$$

$a[1] = 21 \not< 5 = x$
$a[4] = 2 \not> 5 = x$

nach Zeile 9:

$$\begin{array}{lccccccc} a: & 3 & 2 & 9 & 5 & 21 & 7 & 14 \\ & & & \uparrow & \uparrow & & & \\ & & & i & j & & & \end{array}$$

$a[2] = 9 \not< 5 = x$
$a[3] = 5 \not> 5 = x$

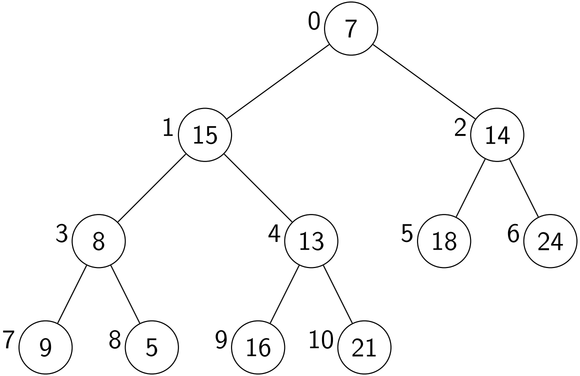
nach Zeile 9:

$$\begin{array}{lccccccc} a: & 3 & 2 & 5 & 9 & 21 & 7 & 14 \\ & & & \uparrow & \uparrow & & & \\ & & & j & i & & & \end{array}$$

Aufrufe: `quicksort(a, 0, 2)`, `quicksort(a, 3, 6)` usw.

Heapsort

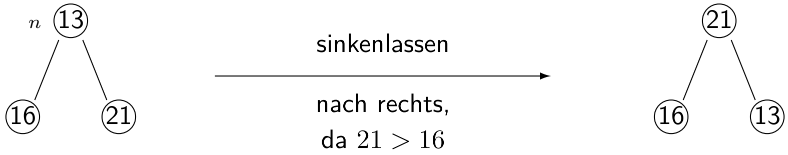
$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	$a[7]$	$a[8]$	$a[9]$	$a[10]$
7	15	14	8	13	18	24	9	5	16	21



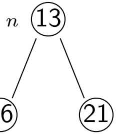
Heapsort

1. Jeder Knoten ist mit einer positiven, ganzen Zahl beschriftet; zwei verschiedene Knoten tragen verschiedene Zahlen.
2. Es gibt eine Ebene t des Baumes, so dass
   (i) alle auf der Ebene t besetzten Positionen linksbündig angeordnet sind,
   (ii) alle Positionen auf Ebene $t-1$ besetzt sind und
   (iii) keine Position der Ebene $t+1$ besetzt ist.
3. Für jeden Knoten n gilt: Wenn n mit h beschriftet ist, dann müssen die Beschriftungen der Nachfolger von n kleiner als h sein ($heap$-Eigenschaft).

Heapsort

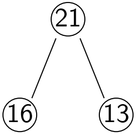


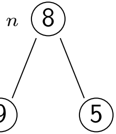
Heapsort



sinkenlassen

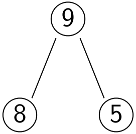
nach rechts,
da $21 > 16$





sinkenlassen

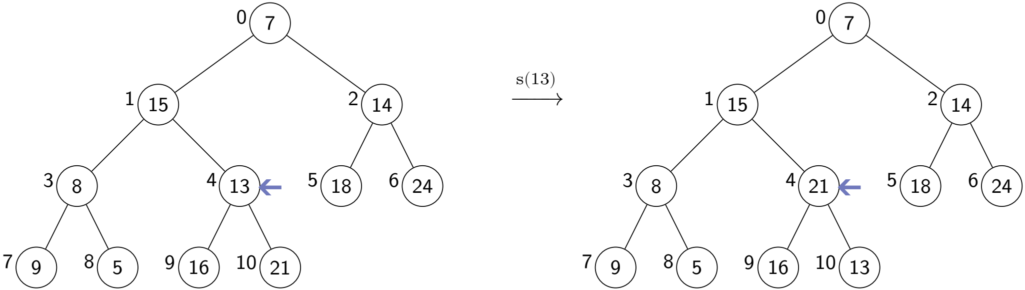
nach links,
da $9 > 5$



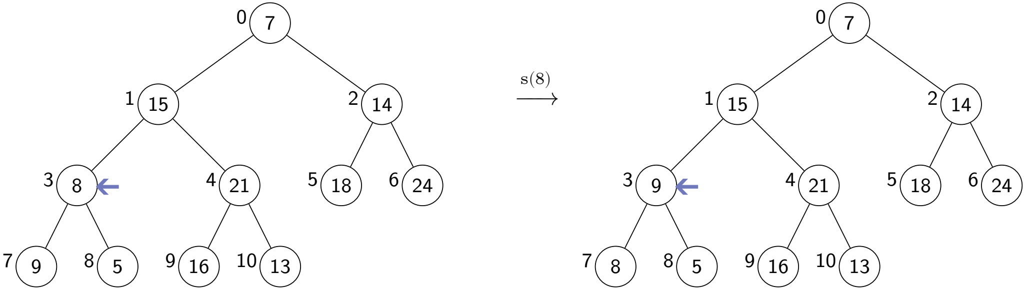
```
. . .
#define K 100

/* lasse a[l] in a[l],a[l+1],...,a[r] hineinsinken */
void sinkenlassen(int a[], int l, int r)
{ int i, j, h, loop;

  i = l;
  h = a[i];
  loop = 1;
  while (loop)
  { j = 2*i+1;                  /* gehe zum linken Nachfolger von i */
    if (j > r)
      break;

    if (j < r)
      if (a[j] < a[j+1])
        j = j+1;                /* rechter Nachfolger a[j+1] ist       */
                                /* groesser als linker Nachfolger a[j] */
    if (h > a[j])
      break;
    else
    { a[i] = a[j];              /* von j nach i sinkenlassen */
      i = j;
    }
  }
  a[i]= h;
}
```

```
void Heapsort(int a[], int n)
{ int li, re, x;

  li = n / 2;
  re = n-1;
  while (li > 0)      /* Phase 1 */
  { li = li-1;                /* rechte Feldgrenze re = n-1 bleibt konstant, */
    sinkenlassen(a, li, re); /* linke Feldgrenze li wird dekrementiert      */
  }
  while (re > 0)      /* Phase 2 */
  { x = a[0];
    a[0] = a[re];
    a[re] = x;
    re = re-1;
    sinkenlassen(a, 0, re);  /* linke Feldgrenze 0 bleibt konstant,       */
  }                          /* rechte Feldgrenze re wird dekrementiert   */
}

int main()
{ int a[K];
  /* Werte fuer a[0] bis a[K-1] eingeben */
  . . .
  Heapsort(a, K);
  . . .
}
```

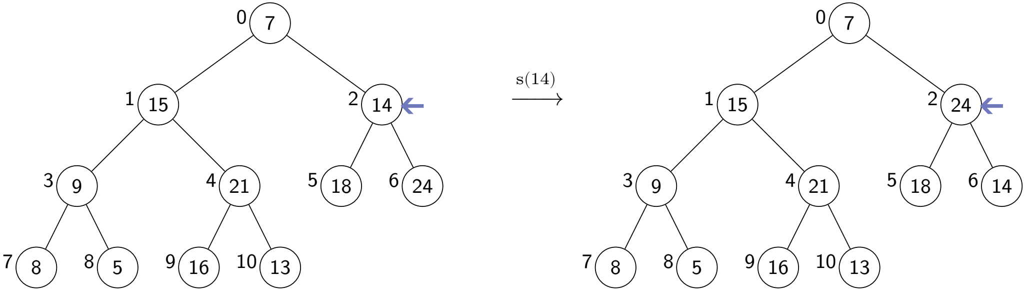
Beispiel, 1. Phase



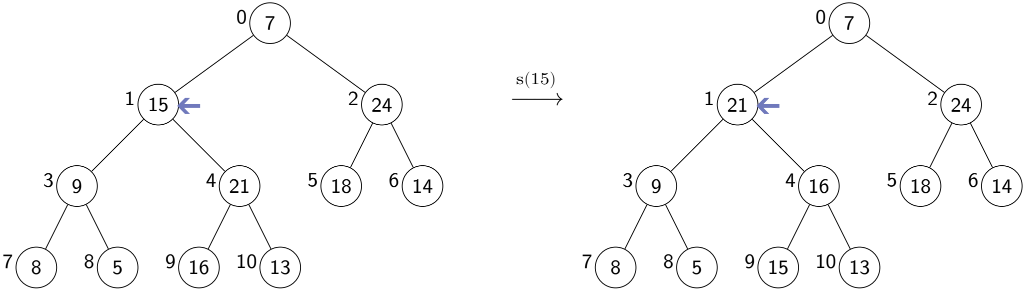
Beispiel, 1. Phase



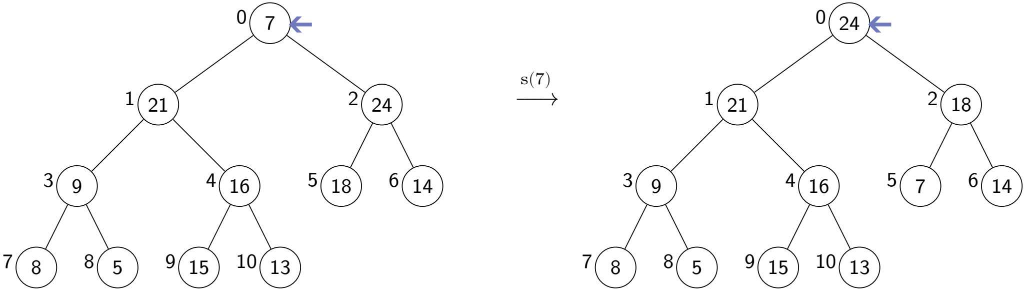
Beispiel, 1. Phase



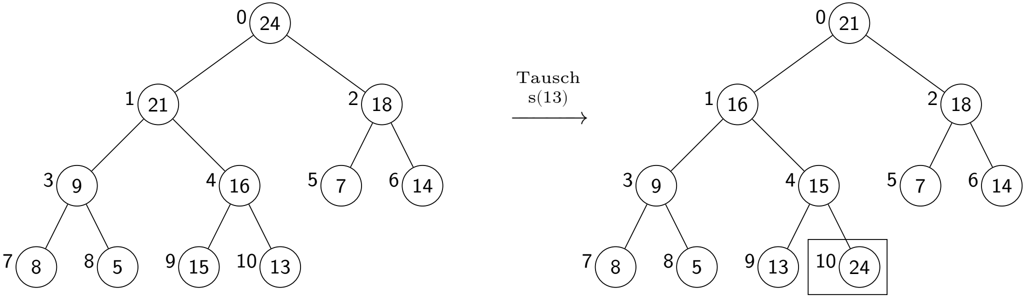
Beispiel, 1. Phase



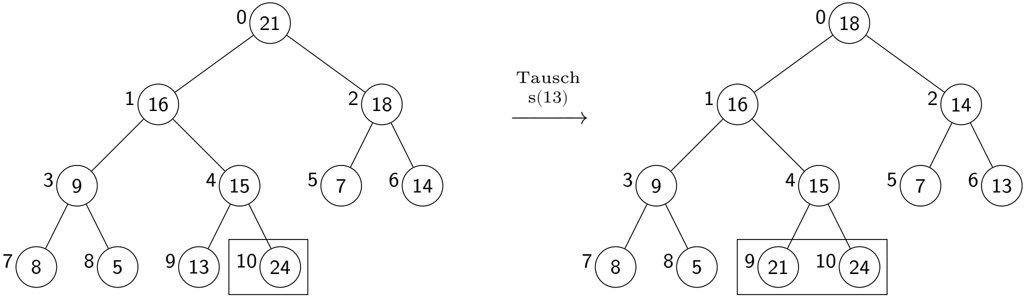
Beispiel, 1. Phase



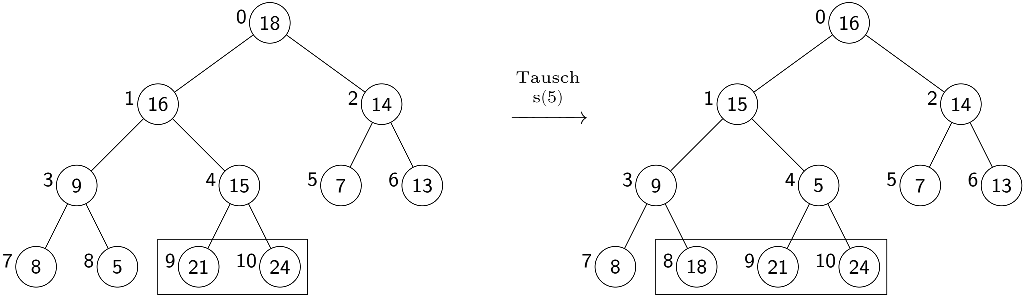
Beispiel, 2. Phase



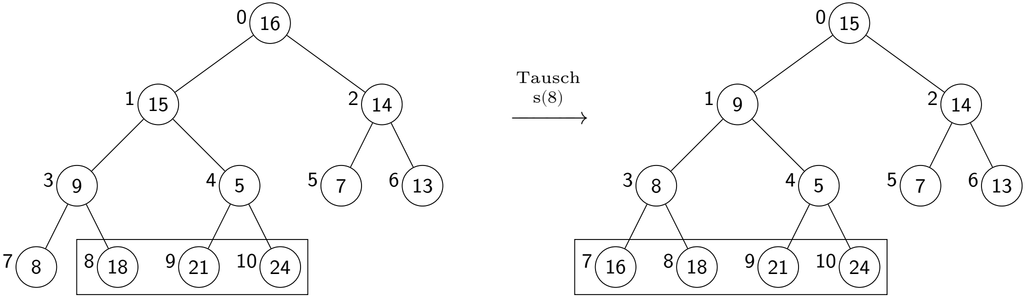
Beispiel, 2. Phase



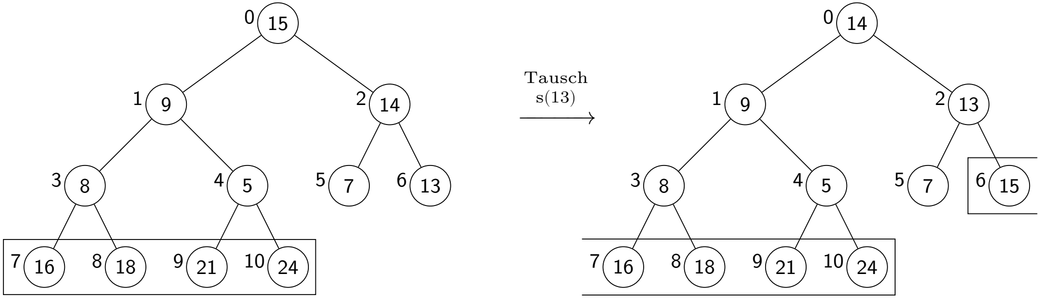
Beispiel, 2. Phase



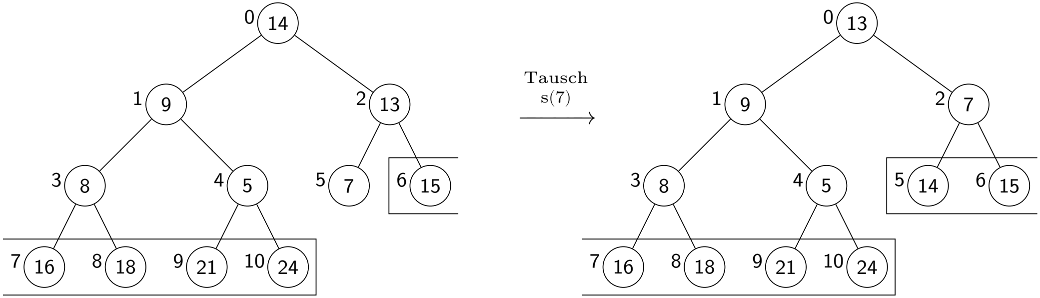
Beispiel, 2. Phase



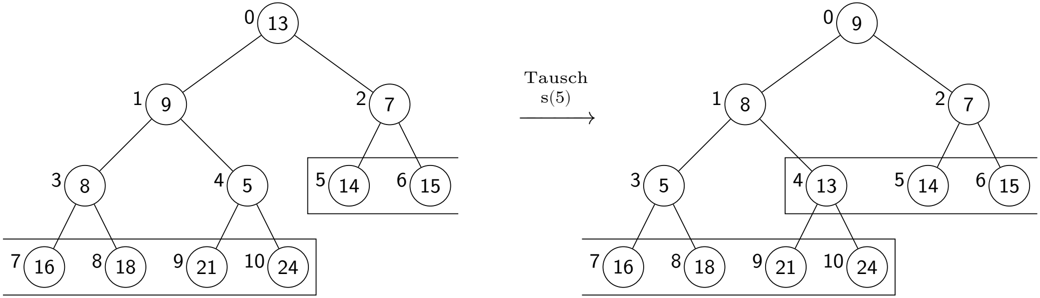
Beispiel, 2. Phase



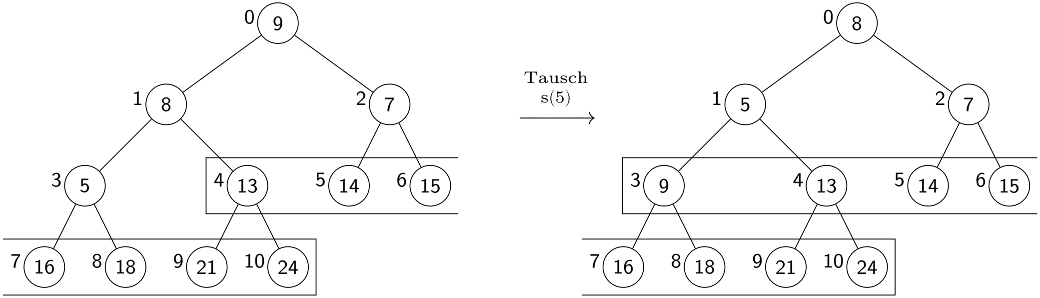
Beispiel, 2. Phase



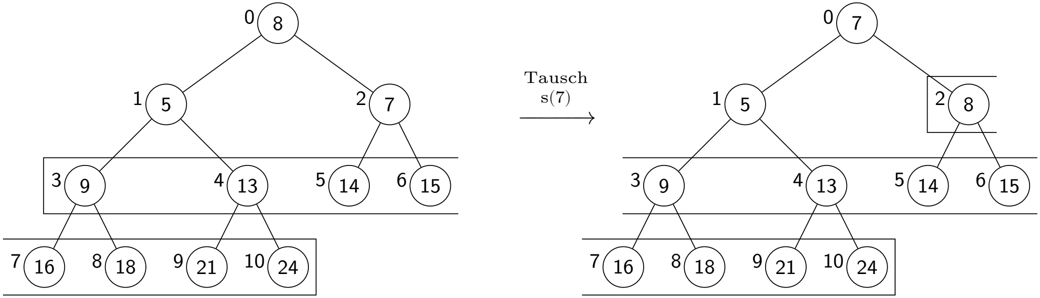
Beispiel, 2. Phase



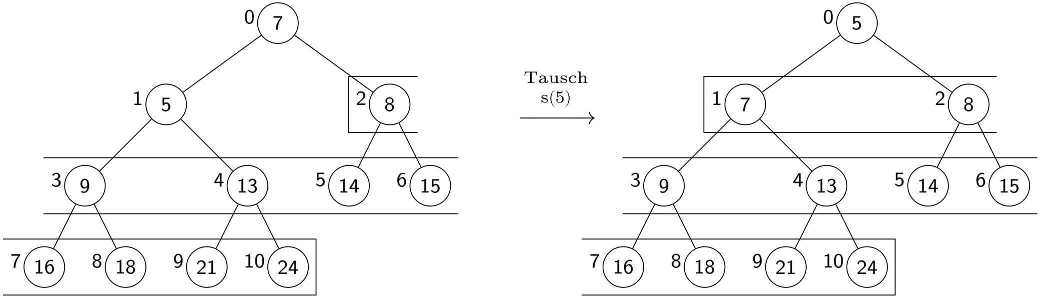
Beispiel, 2. Phase



Beispiel, 2. Phase



Beispiel, 2. Phase



Inhaltsverzeichnis I

Inhaltsverzeichnis II

Suchen und Ersetzen

```
typedef struct Feld { int key;
                      ... contents; } FeldTyp;

FeldTyp F[Flaenge];
int  Wert;
```

Suchen und Ersetzen

```c
typedef struct Feld { int key;
                      ... contents; } FeldTyp;

FeldTyp F[Flaenge];
int  Wert;
```

```c
i = 0;
while ((i < Flaenge) && (F[i].key != Wert))
  i = i+1;

gefunden = (i < Flaenge);
```

Suchen und Ersetzen

```
typedef struct Feld { int key;
                      ... contents; } FeldTyp;

FeldTyp F[Flaenge];
int  Wert;
```

```
i = 0;
while ((i < Flaenge) && (F[i].key != Wert))
  i = i+1;

gefunden = (i < Flaenge);
```

```
FeldTyp F[Flaenge+1];
int Wert;
. . .
i = 0;
F[Flaenge].key = Wert;
while (F[i].key != Wert) i=i+1;
gefunden = (i < Flaenge);
```

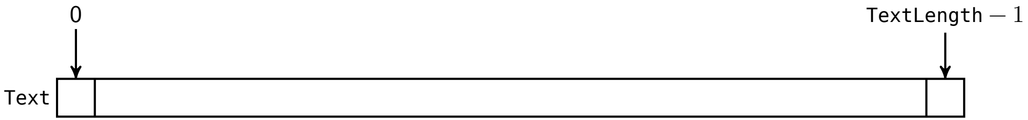
Rekursiver Algorithmus

```
int SearchRec(FeldTyp F[], int links, int rechts, int wert)
{ int pos;

  if (links > rechts)
    return 0;     /* FALSE */
  pos = (links+rechts) / 2;
  if (F[pos].key == wert)
    return 1; /* TRUE */
  if (F[pos].key < wert)
    return SearchRec(F, pos+1, rechts, wert);
  else
    return SearchRec(F, links, pos-1, wert);
}

int main()
{ int gefunden;
  . . .
  gefunden = SearchRec(F,0,Flaenge-1,Wert);
  . . .
}
```

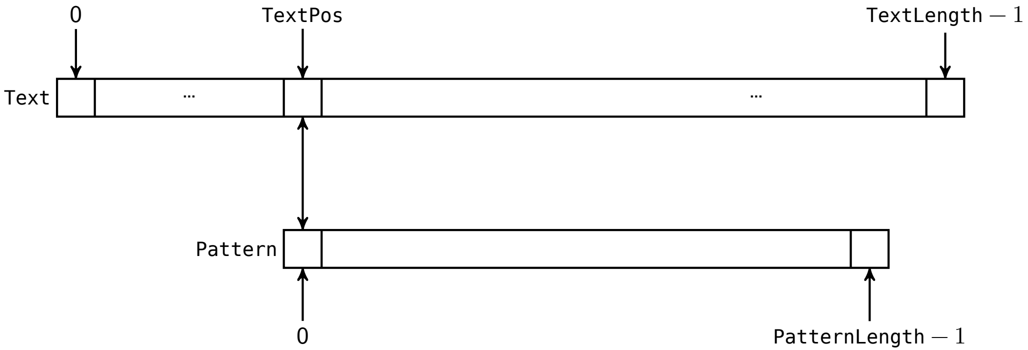
Iterativer Algorithmus

```
gefunden = 0;     /* FALSE */
links = 0; rechts = Flaenge-1;

while ((links <= rechts) && !gefunden)
{ pos = (links+rechts) / 2;
  if (F[pos].key == Wert)
    gefunden = 1;  /* TRUE */
  else
    if (F[pos].key < Wert)
      links = pos+1;
    else
      rechts = pos-1;
}
```

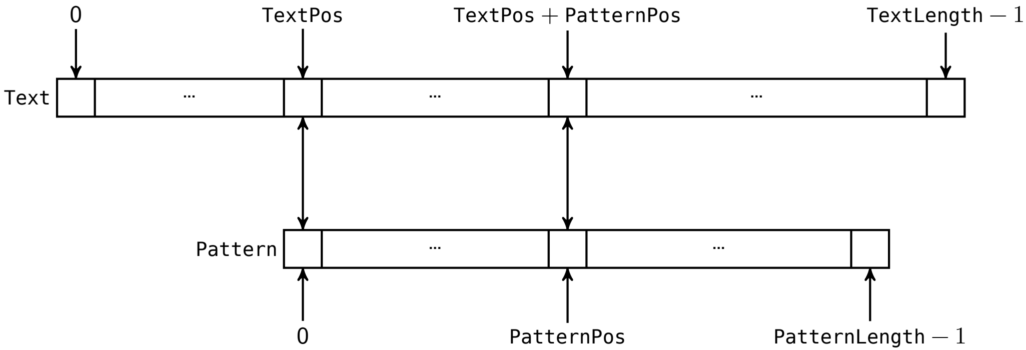
Textsuche



Textsuche



Textsuche



0
TextPos
TextPos + PatternPos
TextLength − 1
Text
...
...
...
Pattern
...
...
0
PatternPos
PatternLength − 1

Naive Textsuche

```
/* alle skalaren Variablen sind vom Typ int, die Strings Text und Pattern
   sind ab 0 bis TextLength-1 bzw. PatternLength-1 indiziert.          */

TextPos = 0; PatternPos = 0;

while ((PatternPos < PatternLength) && (TextPos+PatternLength <= TextLength))
{ PatternPos = 0;
  while ((PatternPos < PatternLength) &&
           (Pattern[PatternPos] == Text[TextPos + PatternPos]))  /* (*) */
    PatternPos = PatternPos + 1;
  TextPos = TextPos + 1;
}

if (TextPos > 0) TextPos = TextPos - 1;
if (PatternPos == PatternLength)
{ gefunden = 1;        /* TRUE */
  printf("Pattern beginnt an Position: %d", TextPos);
}
else
{ gefunden = 0;        /* FALSE */
  printf("Pattern nicht gefunden");
}
```

Verschiebetabelle

Pattern	G	E	G	E	B	E	N
Position	0	1	2	3	4	5	6
Tabelle							

Verschiebetabelle

Pattern	G	E	G	E	B	E	N
Position	0	1	2	3	4	5	6
Tabelle	-1						

```
Textposition:            !
Text:                 ...N I C H ...
Pattern:                 G E G E B E N
Patternposition:         !
- Tabelle[0] := -1
```

Verschiebetabelle

Pattern	G	E	G	E	B	E	N
Position	0	1	2	3	4	5	6
Tabelle	-1	0					

```
Textposition:           !
Text:               ... G G E G ...
Pattern:                G E G E B E N
Patternposition:          !
- Tabelle[1] := 0
```

Verschiebetabelle

Pattern	G	E	G	E	B	E	N
Position	0	1	2	3	4	5	6
Tabelle	-1	0	-1				

```
Textposition:             !
Text:                 G E B I R G E ...
Pattern:              G E G E B E N
Patternposition:          !
- Tabelle[2] := -1
```

Verschiebetabelle

Pattern	G	E	G	E	B	E	N
Position	0	1	2	3	4	5	6
Tabelle	-1	0	-1	0			

```
Textposition:          !
Text:         ... G E G G E N H E I M ...
Pattern:          G E G E B E N
Patternposition:       !
- Tabelle[3] := 0
```

Verschiebetabelle

Pattern	G	E	G	E	B	E	N
Position	0	1	2	3	4	5	6
Tabelle	-1	0	-1	0	2		

```
Textposition:               !
Text:           ... G E G E G E B E N E N F A L L S ...
Pattern:            G E G E B E N
Patternposition:            !
- Tabelle[4] := 2
```

Verschiebetabelle

Pattern	G	E	G	E	B	E	N
Position	0	1	2	3	4	5	6
Tabelle	-1	0	-1	0	2	0	

```
Textposition:                    !
Text:                   ... G E G E B G ...
Pattern:                    G E G E B E N
Patternposition:                 !
- Tabelle[5] := 0
```

Verschiebetabelle

Pattern	G	E	G	E	B	E	N
Position	0	1	2	3	4	5	6
Tabelle	-1	0	-1	0	2	0	0

```
Textposition:                      !
Text:                 ...G E G E B E G  ...
Pattern:                 G E G E B E N
Patternposition:                   !
- Tabelle[6] := 0
```

Algorithmus nach Knuth-Morris-Pratt (KMP)

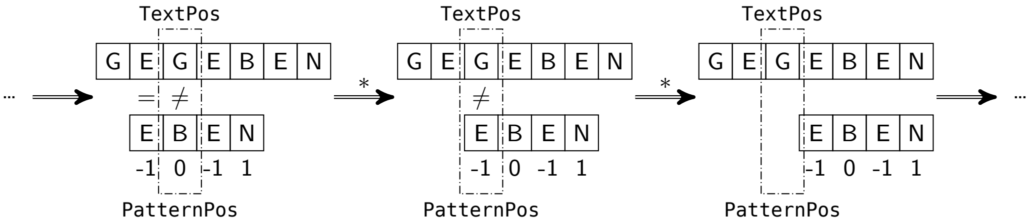
$$l = \max(\{-1\} \cup \{m \mid 0 \leq m \leq j-1 \text{ und } (b_0 \cdots b_{m-1}) = (a_{i+j-m} \cdots a_{i+j-1}) \text{ und } b_m \neq b_j\})$$

$$\text{Tabelle[j]} = \max(\{-1\} \cup \{m \mid 0 \leq m \leq j-1 \text{ und } (b_0 \cdots b_{m-1}) = (b_{j-m} \cdots b_{j-1}) \text{ und } b_m \neq b_j\})$$

	0		i					$i+j$			r
Text:	a_0	$\cdots$	a_i	$\cdots$	a_{i+j-m}	$\cdots$	a_{i+j-1}	a_{i+j}	$\cdots$		a_r
			$=$	$\cdots$	$=$	$\cdots$	$=$	$\neq$			
Pattern:			b_0	$\cdots$	b_{j-m}	$\cdots$	b_{j-1}	b_j	$\cdots$		
					$=$	$\cdots$	$=$	$\neq$			
Pattern:					b_0	$\cdots$	b_{m-1}	b_m	$\cdots$	b_n	

Algorithmus nach Knuth-Morris-Pratt (KMP)

Bei der Suche des Patterns `EBEN` in `GEGEBEN` kommt es zu zwei direkt aufeinander folgenden elementaren Verschiebeoperationen (jeweils mit $*$ gekennzeichnet) bevor `TextPos` bewegt wird:



Algorithmus nach Knuth-Morris-Pratt (KMP)

```
void TabelleBauen()
{ int PatPos;           /* durchlaeuft Pattern bis zum
                           letzten Zeichen */
  int VglInd;           /* Laenge des linken Teilpatterns,
                           das mit Patternanfang uebereinstimmt */

  Tabelle[0] = -1;
  VglInd = 0;

  for (PatPos = 1; PatPos <= PatternLength-1; PatPos = PatPos+1)
  { if (Pattern[PatPos] == Pattern[VglInd])
      Tabelle[PatPos] = Tabelle[VglInd];
    else
      Tabelle[PatPos] = VglInd;

    while ((VglInd >= 0) && (Pattern[PatPos] != Pattern[VglInd]))
      VglInd = Tabelle[VglInd];

    VglInd = VglInd+1;
  }
}
```

Algorithmus nach Knuth-Morris-Pratt (KMP)

```
{ TabelleBauen();
  TextPos = 0; PatternPos = 0;

  while ((PatternPos < PatternLength) && (TextPos < TextLength))
  { while ((PatternPos >= 0) && (Text[TextPos] != Pattern[PatternPos]))
      PatternPos = Tabelle[PatternPos];

    TextPos = TextPos+1;
    PatternPos = PatternPos+1;
  }

  if (PatternPos == PatternLength)
    printf("Pattern gefunden an Position %d", TextPos-PatternLength);
  else
    printf("Pattern nicht gefunden");
}
```

Korrektur von Schreibfehlern

Satz im Aufbau:

Der

Korrektur von Schreibfehlern

Satz im Aufbau:

Der Satttel

Korrektur von Schreibfehlern

Satz im Aufbau:

Der Satttel

Menge C von $k = 3$ Kandidaten für die Ersetzung:

$C = \{\text{Sattel}, \text{Starten}, \text{Staffel}\}$

Korrektur von Schreibfehlern

Satz im Aufbau:

Der Sattel

Menge C von $k = 3$ Kandidaten für die Ersetzung:

$C = \{\text{Sattel}, \text{Starten}, \text{Staffel}\}$

Korrektur von Schreibfehlern

Satz im Aufbau:

Der Sattel

Menge C von $k = 3$ Kandidaten für die Ersetzung:

Korrektur von Schreibfehlern

Satz im Aufbau:

Der Sattel mmit

Menge C von $k = 3$ Kandidaten für die Ersetzung:

$C = \{\text{mir}, \text{mit}, \text{mild}\}$

Korrektur von Schreibfehlern

Satz im Aufbau:

Der Sattel mit

Menge C von $k = 3$ Kandidaten für die Ersetzung:

$$C = \{\mathsf{mir}, \mathsf{mit}, \mathsf{mild}\}$$

Korrektur von Schreibfehlern

Satz im Aufbau:

Der Sattel mit

Menge C von $k = 3$ Kandidaten für die Ersetzung:

Korrektur von Schreibfehlern

Satz im Aufbau:

Der Sattel mit Belleuchtung

Menge C von $k = 3$ Kandidaten für die Ersetzung:

$C = \{\text{Beleuchtung}, \text{Belichtung}, \text{Belüftung}\}$

Korrektur von Schreibfehlern

Satz im Aufbau:

Der Sattel mit Beleuchtung

Menge C von $k = 3$ Kandidaten für die Ersetzung:

$C = \{\text{Beleuchtung}, \text{Belichtung}, \text{Belüftung}\}$

Korrektur von Schreibfehlern

Satz im Aufbau:

Der Sattel mit $\underbrace{\text{Belleuchtung}}_{w}$

Menge C von $k = 3$ Kandidaten für die Ersetzung:

$$C = \{\text{Beleuchtung}, \text{Belichtung}, \text{Belüftung}\}$$

Korrektur von Schreibfehlern

Satz im Aufbau:

Der Sattel mit $\underbrace{\text{Belleuchtung}}_{w}$

Menge C von $k = 3$ Kandidaten für die Ersetzung:

$C = \{\text{Beleuchtung}, \text{Belichtung}, \text{Belüftung}\}$

Berechnung von C:

M: Menge aller korrekt geschriebenen Wörter (Vokabular)

Korrektur von Schreibfehlern

Satz im Aufbau:

Der Sattel mit $\underbrace{\text{Belleuchtung}}_{w}$

Menge C von $k = 3$ Kandidaten für die Ersetzung:

$C = \{\text{Beleuchtung}, \text{Belichtung}, \text{Belüftung}\}$

Berechnung von C:

M: Menge aller korrekt geschriebenen Wörter (Vokabular)

berechne Unterschied $d(w, v) \in \mathbb{N}$ zwischen w und v für jedes $v \in M$

Korrektur von Schreibfehlern

Satz im Aufbau:

Der Sattel mit $\underbrace{\text{Belleuchtung}}_{w}$

Menge C von $k = 3$ Kandidaten für die Ersetzung:

$C = \{\text{Beleuchtung}, \text{Belichtung}, \text{Belüftung}\}$

Berechnung von C:

M: Menge aller korrekt geschriebenen Wörter (Vokabular)

berechne Unterschied $d(w, v) \in \mathbb{N}$ zwischen w und v für jedes $v \in M$

$C =$ Menge der k Wörter aus M mit dem geringsten Unterschied zu w.

Editieroperationen

Die Editieroperationen leisten folgendes:

- Insertion (i) für einen Buchstaben in das Quellwort ein
- Deletion (d) löscht einen Buchstaben aus dem Quellwort
- Substitution (s) ersetzt einen Buchstaben des Quellworts durch einen Buchstaben

Editieroperationen

Die Editieroperationen leisten folgendes:

- Insertion (i)   für einen Buchstaben in das Quellwort ein
- Deletion (d)   löscht einen Buchstaben aus dem Quellwort
- Substitution (s)   ersetzt einen Buchstaben des Quellworts durch einen Buchstaben

1.

i	n	*	t	u	i	t	i	o	n
*	n	u	t	r	i	t	i	o	n
d		i		s					

2.

i	n	t	u	i	t	i	o	n
n	u	t	r	i	t	i	o	n
s	s		s					

3.

i	n	t	u	*	*	*	*	i	t	i	o	n
*	*	*	*	n	u	t	r	i	t	i	o	n
d	d	d	d	i	i	i	i					

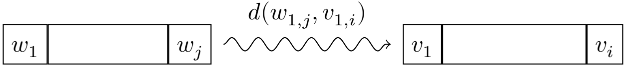
Levenshtein-Distanz

Die Levenshtein-Distanz zwischen w und v, bezeichnet mit $d(w, v)$, ist die Höhe der minimalen Kosten von Editieroperationen (Insertion, Deletion, Substitution), die gebraucht werden, um von w nach v zu kommen.

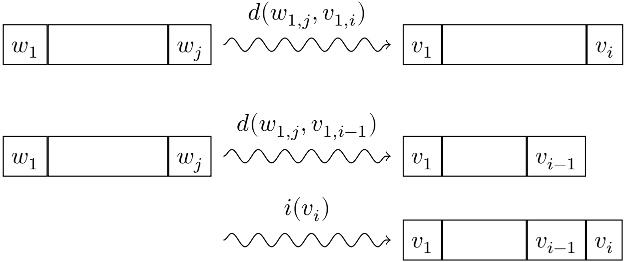
Levenshtein-Distanz

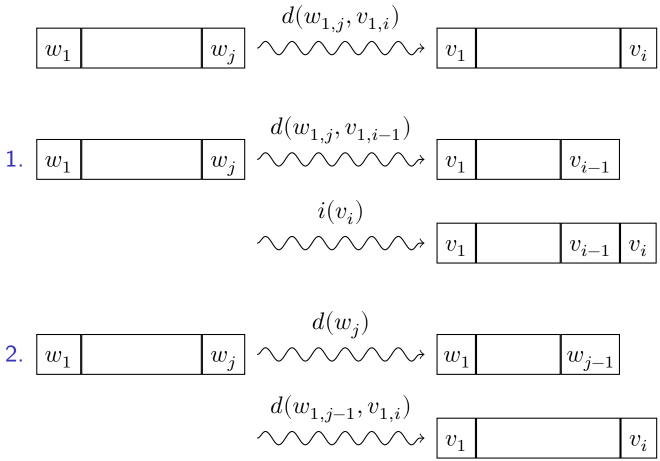
Die Levenshtein-Distanz zwischen w und v, bezeichnet mit $d(w, v)$, ist die Höhe der minimalen Kosten von Editieroperationen (Insertion, Deletion, Substitution), die gebraucht werden, um von w nach v zu kommen.

- $d(w, v) = 0$ genau dann wenn $w = v$,
- $d(w, v) = d(v, w)$ und
- $d(w, v) + d(v, u) \geq d(w, u)$ für jedes Wort u.


w_1
w_j
$d(w_{1,j}, v_{1,i})$
v_1
v_i

1.




$d(w_{1,j}, v_{1,i})$
w_1 w_j v_1 v_i
1.
$d(w_{1,j}, v_{1,i-1})$
w_1 w_j v_1 v_{i-1}
$i(v_i)$
v_1 v_{i-1} v_i
2.
$d(w_j)$
w_1 w_j w_1 w_{j-1}
$d(w_{1,j-1}, v_{1,i})$
v_1 v_i

$$\boxed{w_1}\boxed{\quad\quad}\boxed{w_j} \;\overset{d(w_{1,j}, v_{1,i})}{\rightsquigarrow}\; \boxed{v_1}\boxed{\quad\quad}\boxed{v_i}$$

1. $\boxed{w_1}\boxed{\quad\quad}\boxed{w_j} \;\overset{d(w_{1,j}, v_{1,i-1})}{\rightsquigarrow}\; \boxed{v_1}\boxed{\quad\quad}\boxed{v_{i-1}}$

   $\overset{i(v_i)}{\rightsquigarrow}\; \boxed{v_1}\boxed{\quad\quad}\boxed{v_{i-1}}\boxed{v_i}$

2. $\boxed{w_1}\boxed{\quad\quad}\boxed{w_j} \;\overset{d(w_j)}{\rightsquigarrow}\; \boxed{w_1}\boxed{\quad\quad}\boxed{w_{j-1}}$

   $\overset{d(w_{1,j-1}, v_{1,i})}{\rightsquigarrow}\; \boxed{v_1}\boxed{\quad\quad}\boxed{v_i}$

3. $\boxed{w_1}\boxed{\quad\quad}\boxed{w_j} \;\overset{d(w_{1,j-1}, v_{1,i-1})}{\rightsquigarrow}\; \boxed{v_1}\boxed{\quad\quad}\boxed{v_{i-1}}\boxed{w_j}$

   $\overset{s(w_j, v_i)}{\rightsquigarrow}\; \boxed{v_1}\boxed{\quad\quad}\boxed{v_{i-1}}\boxed{v_i}$

Levenshtein-Distanz

Die minimalen Kosten einer Folge von Editieroperationen, die das Teilwort $w_{1,j}$ in das Teilwort $v_{1,i}$ überführt, ist das Minimum aus

1. $d(w_{1,j}, v_{1,i-1}) + 1$, d.h. den minimalen Kosten einer Folge von Editieroperationen, die $w_{1,j}$ in $v_{1,i-1}$ überführt gefolgt von Insertion von v_i,
2. $d(w_{1,j-1}, v_{1,i}) + 1$, d.h. den minimalen Kosten einer Folge von Editieroperationen, die $w_{1,j-1}$ in $v_{1,i}$ überführt gefolgt von Deletion von w_j,
3. $d(w_{1,j-1}, v_{1,i-1}) + 1$ (falls $w_j \neq v_i$), d.h. den minimalen Kosten einer Folge von Editieroperationen, die $w_{1,j-1}$ in $v_{1,i-1}$ überführt gefolgt von Substitution von w_j durch v_i und
4. $d(w_{1,j-1}, v_{1,i-1})$ (falls $w_j = v_i$), d.h. den minimalen Kosten einer Folge von Editieroperationen, die $w_{1,j-1}$ in $v_{1,i-1}$ überführt.

Bildungsvorschrift der Distanzmatrix

Abkürzung: $d(w_{1,j}, v_{1,i}) \quad \rightsquigarrow \quad d(j,i)$

$$d(0,i) = i \qquad \text{für jedes } 0 \le i \le k,$$

Bildungsvorschrift der Distanzmatrix

Abkürzung: $d(w_{1,j}, v_{1,i}) \quad \rightsquigarrow \quad d(j,i)$

$$d(0,i) = i \qquad \text{für jedes } 0 \leq i \leq k,$$

$$d(j,0) = j \qquad \text{für jedes } 0 \leq j \leq n \text{ und}$$

Bildungsvorschrift der Distanzmatrix

Abkürzung: $d(w_{1,j}, v_{1,i}) \quad \rightsquigarrow \quad d(j,i)$

$$d(0,i) = i \qquad \text{für jedes } 0 \leq i \leq k,$$

$$d(j,0) = j \qquad \text{für jedes } 0 \leq j \leq n \text{ und}$$

$$d(j,i) = \min\{d(j,i-1)+1,\ d(j-1,i)+1,\ d(j-1,i-1) + \left\{ \begin{array}{ll} 1 & \text{wenn } w_j \neq v_i \\ 0 & \text{sonst} \end{array} \right\}\}$$

für jedes $1 \leq j \leq n$ und jedes $1 \leq i \leq k$.

Berechnung der Levenshtein-Distanz (Beispiel)

i (Positionen: $0, ..., 9$) $\longrightarrow$

j (Positionen: $0, ..., 9$) $\downarrow$

v = nutrition, w = intuition

$d(j,i)$		n	u	t	r	i	t	i	o	n
i										
n										
t										
u										
i										
t										
i										
o										
n										

Berechnung der Levenshtein-Distanz (Beispiel)

i (Positionen: $0, \dots, 9$) $\longrightarrow$

j (Positionen: $0, \dots, 9$) $\downarrow$

$d(j,i)$		v: n	u	t	r	i	t	i	o	n
	0	$\rightarrow$ 1	$\rightarrow$ 2	$\rightarrow$ 3	$\rightarrow$ 4	$\rightarrow$ 5	$\rightarrow$ 6	$\rightarrow$ 7	$\rightarrow$ 8	$\rightarrow$ 9
	$\downarrow$									
i	1									
	$\downarrow$									
n	2									
	$\downarrow$									
t	3									
	$\downarrow$									
u	4									
w	$\downarrow$									
i	5									
	$\downarrow$									
t	6									
	$\downarrow$									
i	7									
	$\downarrow$									
o	8									
	$\downarrow$									
n	9									

Berechnung der Levenshtein-Distanz (Beispiel)

i (Positionen: $0, ..., 9$) $\longrightarrow$

j (Positionen: $0, ..., 9$) $\downarrow$

$d(j,i)$		v: n	u	t	r	i	t	i	o	n
	0	$\rightarrow$ 1	$\rightarrow$ 2	$\rightarrow$ 3	$\rightarrow$ 4	$\rightarrow$ 5	$\rightarrow$ 6	$\rightarrow$ 7	$\rightarrow$ 8	$\rightarrow$ 9
	$\downarrow$	$\searrow$	$\searrow$	$\searrow$						
w: i	1	1	$\rightarrow$ 2	$\rightarrow$ 3						
	$\downarrow$									
n	2									
	$\downarrow$									
t	3									
	$\downarrow$									
u	4									
	$\downarrow$									
i	5									
	$\downarrow$									
t	6									
	$\downarrow$									
i	7									
	$\downarrow$									
o	8									
	$\downarrow$									
n	9									

Berechnung der Levenshtein-Distanz (Beispiel)

i (Positionen: $0, ..., 9$) $\longrightarrow$

j (Positionen: $0, ..., 9$) $\downarrow$

$d(j,i)$		v: n	u	t	r	i	t	i	o	n
	0	$\rightarrow$ 1	$\rightarrow$ 2	$\rightarrow$ 3	$\rightarrow$ 4	$\rightarrow$ 5	$\rightarrow$ 6	$\rightarrow$ 7	$\rightarrow$ 8	$\rightarrow$ 9
	$\downarrow$	$\searrow$	$\searrow$	$\searrow$						
w: i	1	1	$\rightarrow$ 2	$\rightarrow$ 3						
	$\downarrow$	$\searrow$	$\searrow$	$\searrow$						
n	2	1	$\rightarrow$ 2	$\rightarrow$ 3						
	$\downarrow$									
t	3									
	$\downarrow$									
u	4									
	$\downarrow$									
i	5									
	$\downarrow$									
t	6									
	$\downarrow$									
i	7									
	$\downarrow$									
o	8									
	$\downarrow$									
n	9									

Berechnung der Levenshtein-Distanz (Beispiel)

i (Positionen: $0, ..., 9$) $\longrightarrow$

j (Positionen: $0, ..., 9$) $\downarrow$

$d(j,i)$			v								
			n	u	t	r	i	t	i	o	n
		0	$\rightarrow$ 1	$\rightarrow$ 2	$\rightarrow$ 3	$\rightarrow$ 4	$\rightarrow$ 5	$\rightarrow$ 6	$\rightarrow$ 7	$\rightarrow$ 8	$\rightarrow$ 9
		$\downarrow$	$\searrow$	$\searrow$	$\searrow$						
	i	1	1	$\rightarrow$ 2	$\rightarrow$ 3						
		$\downarrow$	$\searrow$	$\searrow$	$\searrow$						
	n	2	1	$\rightarrow$ 2	$\rightarrow$ 3						
		$\downarrow$	$\downarrow$	$\searrow$	$\searrow$						
	t	3	2	2	2						
		$\downarrow$									
	u	4									
w		$\downarrow$									
	i	5									
		$\downarrow$									
	t	6									
		$\downarrow$									
	i	7									
		$\downarrow$									
	o	8									
		$\downarrow$									
	n	9									

Berechnung der Levenshtein-Distanz (Beispiel)

i (Positionen: $0, \ldots, 9$) →

j (Positionen: $0, \ldots, 9$) ↓

$d(j,i)$		v: n	u	t	r	i	t	i	o	n
w	0	→ 1	→ 2	→ 3	→ 4	→ 5	→ 6	→ 7	→ 8	→ 9
	↓	↘	↘	↘	↘	↘		↘		
i	1	1	→ 2	→ 3	→ 4	4	→ 5	→ 6	→ 7	→ 8
	↓	↘	↘	↘	↘	↘ ↓	↘	↘	↘	↘
n	2	1	→ 2	→ 3	→ 4	→ 5	5	→ 6	→ 7	7
	↓	↓	↘	↘			↘	↘	↘	↘ ↓
t	3	2	2	2	→ 3	→ 4	→ 5	→ 6	→ 7	→ 8
	↓	↓	↘	↘ ↓	↘	↘	↘	↘	↘	↘
u	4	3	2	→ 3	3	→ 4	→ 5	→ 6	→ 7	→ 8
	↓	↓	↓	↘	↘ ↓	↘		↘		
i	5	4	3	3	→ 4	3	→ 4	→ 5	→ 6	→ 7
	↓	↓	↓	↘	↘	↓	↘			
t	6	5	4	3	→ 4	4	3	→ 4	→ 5	→ 6
	↓	↓	↓	↓	↘	↘	↓	↘		
i	7	6	5	4	4	4	4	3	→ 4	→ 5
	↓	↓	↓	↓	↘ ↓	↘ ↓	↘ ↓	↓	↘	
o	8	7	6	5	5	5	5	4	3	→ 4
	↓	↘ ↓	↓	↓	↘ ↓	↘ ↓	↘ ↓	↓	↓	↘
n	9	8	7	6	6	6	6	5	4	3

Minimale Alignments zwischen intuition und nutrition

1.

i	n	*	t	u	i	t	i	o	n
|	|	|	|	|	|	|	|	|	|
*	n	u	t	r	i	t	i	o	n
d		i		s					

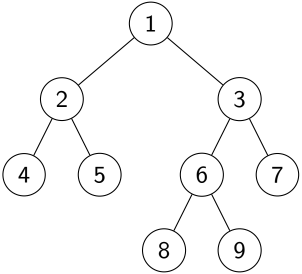
2.

i	n	t	u	i	t	i	o	n
|	|	|	|	|	|	|	|	|
n	u	t	r	i	t	i	o	n
s	s		s					

Inhaltsverzeichnis I

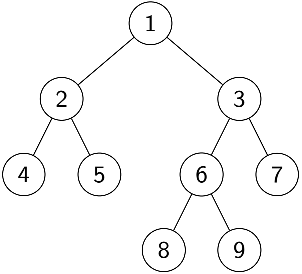
Inhaltsverzeichnis II

Bäume



Wurzel: 1

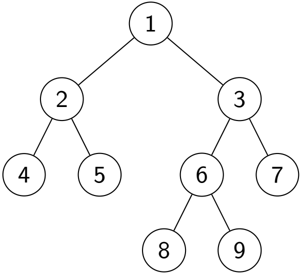
Bäume



Wurzel: 1

Blätter: 4, 5, 8, 9, 7

Bäume

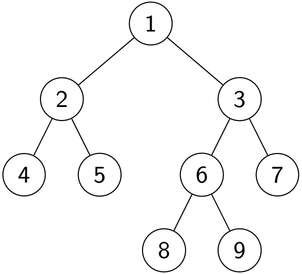


Wurzel: 1

Blätter: 4, 5, 8, 9, 7

innere Knoten: 1, 2, 3, 6

Bäume



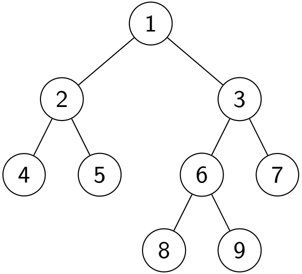
Wurzel: 1

Blätter: 4, 5, 8, 9, 7

innere Knoten: 1, 2, 3, 6

erster Nachfolger: z.B. 6 von 3

Bäume



Wurzel: 1

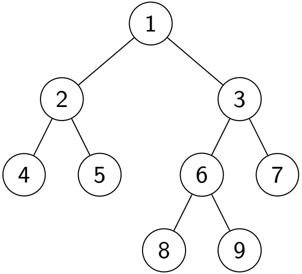
Blätter: 4, 5, 8, 9, 7

innere Knoten: 1, 2, 3, 6

erster Nachfolger: z.B. 6 von 3

zweiter Nachfolger: z.B. 7 von 3

Bäume



Wurzel: 1

Blätter: 4, 5, 8, 9, 7

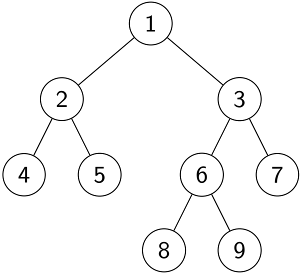
innere Knoten: 1, 2, 3, 6

erster Nachfolger: z.B. 6 von 3

zweiter Nachfolger: z.B. 7 von 3

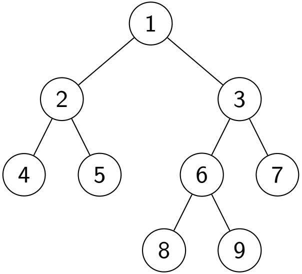
Weg nach n: z.B. Weg nach 6 ist $(1, 3, 6)$

Bäume



Wurzel:	1
Blätter:	4, 5, 8, 9, 7
innere Knoten:	1, 2, 3, 6
erster Nachfolger:	z.B. 6 von 3
zweiter Nachfolger:	z.B. 7 von 3
Weg nach n:	z.B. Weg nach 6 ist $(1, 3, 6)$
Tiefe von n:	z.B. Tiefe von 9 ist 3

Bäume



Wurzel: 1

Blätter: 4, 5, 8, 9, 7

innere Knoten: 1, 2, 3, 6

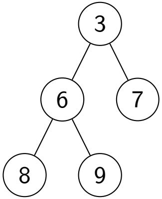
erster Nachfolger: z.B. 6 von 3

zweiter Nachfolger: z.B. 7 von 3

Weg nach n: z.B. Weg nach 6 ist $(1, 3, 6)$

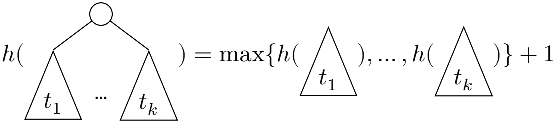
Tiefe von n: z.B. Tiefe von 9 ist 3

Teilbaum von n: z.B. Teilbaum von 3 ist

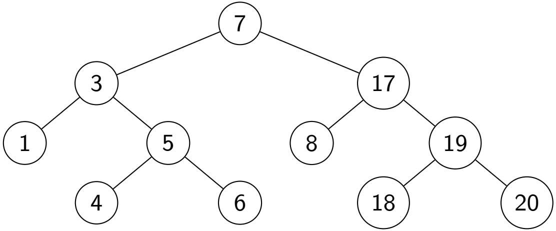


Bäume

$$h(\bigcirc) = 1$$



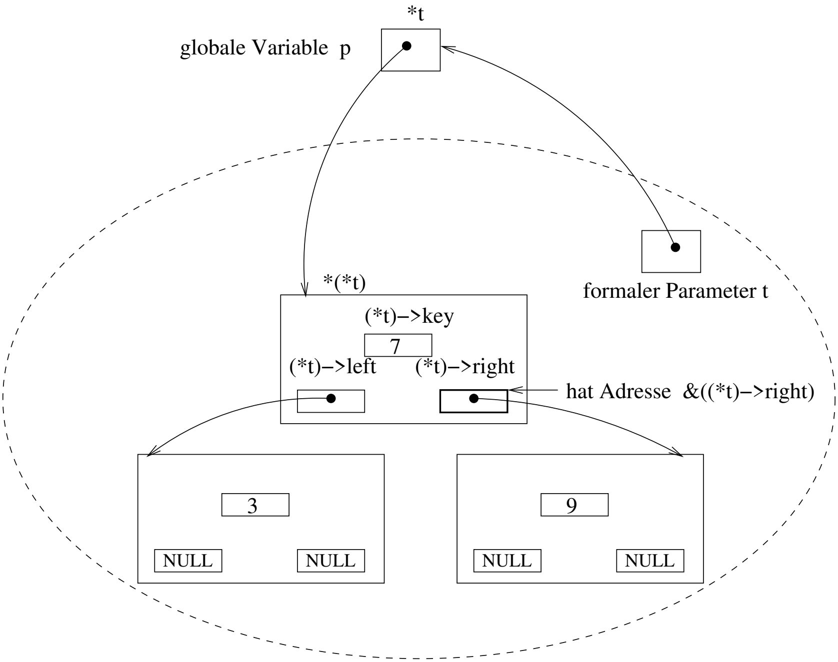
Suchbäume



```
typedef struct Nodeelem *Ptr;
typedef struct Nodeelem { int key;
                          Ptr left, right;
                          . . .    /* beliebige weitere Daten */ } Node;
```

```
void suche (Ptr t, int x)
{ if (t == NULL)
    printf("Element liegt nicht im Baum");
  else
    if (t->key == x)
      printf("Element liegt im Baum");
    else
      if (t->key < x)
        suche(t->right, x);
      else
        suche(t->left, x);
}
```

```
void einfuegen (Ptr *t, int x)
{ Ptr q;
  if (*t == NULL)
  { q = (Ptr)malloc(sizeof(Node));
    q->key = x;
    q->left = NULL;
    q->right = NULL;
    *t = q;
  }
  else
    if ((*t)->key == x)
      printf("Element liegt schon im Baum");
    else
      if ((*t)->key < x)
        einfuegen(&((*t)->right), x);
      else
        einfuegen(&((*t)->left), x);
}
```


*t
globale Variable p
*(*t)
formaler Parameter t
(*t)->key
7
(*t)->left
(*t)->right
hat Adresse &((*t)->right)
3
9
NULL
NULL
NULL
NULL

AVL-Bäume

Ein *AVL-Baum* ist ein Suchbaum, bei dem an jedem Knoten n gilt $b(n) \in \{-1, 0, 1\}$, wobei $b(n)$ der *Balancefaktor* an dem Knoten n ist.

AVL-Bäume

Ein *AVL-Baum* ist ein Suchbaum, bei dem an jedem Knoten n gilt $b(n) \in \{-1, 0, 1\}$, wobei $b(n)$ der *Balancefaktor* an dem Knoten n ist.
Dieser ist wie folgt definiert:

$b(n) = h(t_2) - h(h_1)$ wenn t_1 und t_2 der linke bzw. rechte Teilbaum unter n sind,

AVL-Bäume

Ein *AVL-Baum* ist ein Suchbaum, bei dem an jedem Knoten n gilt $b(n) \in \{-1, 0, 1\}$, wobei $b(n)$ der *Balancefaktor* an dem Knoten n ist.
Dieser ist wie folgt definiert:

$b(n) = h(t_2) - h(h_1)$ wenn t_1 und t_2 der linke bzw. rechte Teilbaum unter n sind,

$b(n) = h(t_2)$ wenn es keinen linken Teilbaum unter n gibt und t_2 der rechte Teilbaum unter n ist,

AVL-Bäume

Ein *AVL-Baum* ist ein Suchbaum, bei dem an jedem Knoten n gilt $b(n) \in \{-1, 0, 1\}$, wobei $b(n)$ der *Balancefaktor* an dem Knoten n ist.
Dieser ist wie folgt definiert:

$b(n) = h(t_2) - h(h_1)$ wenn t_1 und t_2 der linke bzw. rechte Teilbaum unter n sind,

$b(n) = h(t_2)$ wenn es keinen linken Teilbaum unter n gibt und t_2 der rechte Teilbaum unter n ist,

$b(n) = -h(h_1)$ wenn t_1 der linke Teilbaum unter n ist und es keinen rechten Teilbaum unter n gibt und

AVL-Bäume

Ein *AVL-Baum* ist ein Suchbaum, bei dem an jedem Knoten n gilt $b(n) \in \{-1, 0, 1\}$, wobei $b(n)$ der *Balancefaktor* an dem Knoten n ist.
Dieser ist wie folgt definiert:

$b(n) = h(t_2) - h(h_1)$	wenn t_1 und t_2 der linke bzw. rechte Teilbaum unter n sind,
$b(n) = h(t_2)$	wenn es keinen linken Teilbaum unter n gibt und t_2 der rechte Teilbaum unter n ist,
$b(n) = -h(h_1)$	wenn t_1 der linke Teilbaum unter n ist und es keinen rechten Teilbaum unter n gibt und
$b(n) = 0$	wenn es weder einen linken noch einen rechten Teilbaum unter n gibt.

AVL-Bäume

Ein *AVL-Baum* ist ein Suchbaum, bei dem an jedem Knoten n gilt $b(n) \in \{-1, 0, 1\}$, wobei $b(n)$ der *Balancefaktor* an dem Knoten n ist.
Dieser ist wie folgt definiert:

$b(n) = h(t_2) - h(h_1)$ wenn t_1 und t_2 der linke bzw. rechte Teilbaum unter n sind,

$b(n) = h(t_2)$ wenn es keinen linken Teilbaum unter n gibt und t_2 der rechte Teilbaum unter n ist,

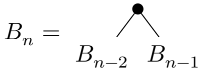
$b(n) = -h(h_1)$ wenn t_1 der linke Teilbaum unter n ist und es keinen rechten Teilbaum unter n gibt und

$b(n) = 0$ wenn es weder einen linken noch einen rechten Teilbaum unter n gibt.

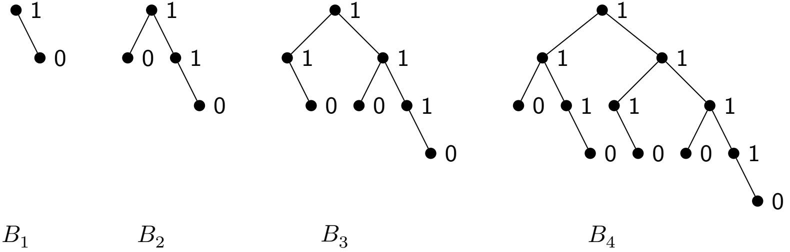
Beachte: Die Höhe eines leeren Teilbaumes ist 0.

AVL-Bäume

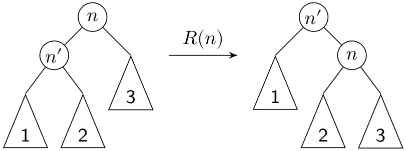
Bildungsgesetz für $n \geq 3$:



Beispiele für Binärbäume, die „gerade noch“ AVL-Bäume sind:

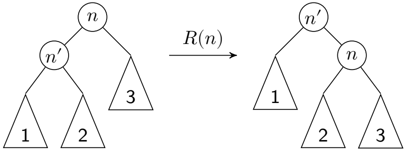


Rechtsrotation:

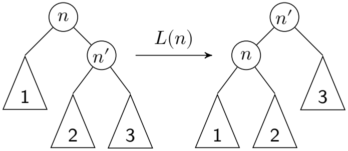


AVL-Bäume

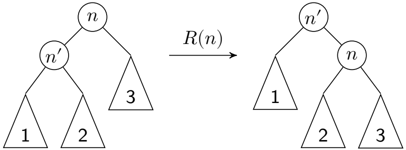
Rechtsrotation:



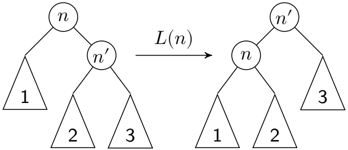
Linksrotation:



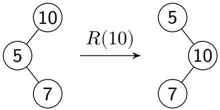
Rechtsrotation:



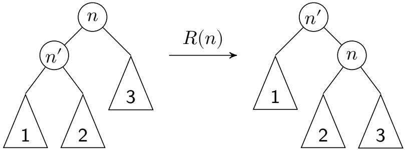
Linksrotation:



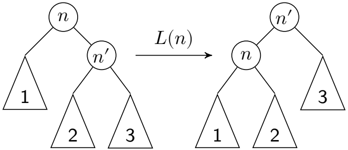
ohne Doppelrotation:



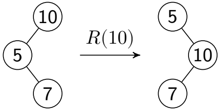
Rechtsrotation:



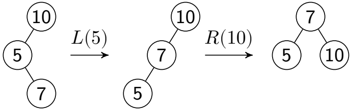
Linksrotation:



ohne Doppelrotation:



mit Doppelrotation:



AVL-Bäume

```
typedef struct Nodeelem *Ptr;
typedef struct Nodeelem { int key;
                          short balance;
                          Ptr left, right; } Node;
```

Algorithmus Einfügen eines Elementes x in AVL-Bäume

Eingabe: ein AVL-Baum `t`, ein Element `x`

Ausgabe: ein AVL-Baum, der die Elemente aus `t` und das Element `x` enthält

Vorgehen:

1. Füge das neue Element `x` mit Balancefaktor 0 als direkten Nachfolger des Knotens `n` als Blatt ein, so dass die Suchbaumeigenschaft erfüllt ist. Aktualisiere `n.balance`.

Algorithmus Einfügen eines Elementes x in AVL-Bäume

Eingabe: ein AVL-Baum `t`, ein Element `x`

Ausgabe: ein AVL-Baum, der die Elemente aus `t` und das Element `x` enthält

Vorgehen:

1. Füge das neue Element `x` mit Balancefaktor 0 als direkten Nachfolger des Knotens `n` als Blatt ein, so dass die Suchbaumeigenschaft erfüllt ist. Aktualisiere `n.balance`.
2. Setze `n` auf den Vorgängerknoten von `n`. (Beachte: `n` ist vom Typ `Ptr`!)
   (a) Falls `x` im linken Unterbaum von `n` eingefügt wurde
      (i) wenn `n->balance == 1` dann `n->balance = 0` und gehe nach 3.
      (ii) wenn `n->balance == 0`, dann `n->balance = -1` und gehe nach 2.
      (iii) wenn `n->balance == -1` und
         - wenn `n->left->balance == -1` dann Rechtsrotation um `n`
         - wenn `n->left->balance == 1`, dann erst eine Linksrotation um `n->left`, dann eine Rechtsrotation um `n`.

AVL-Bäume

Algorithmus Einfügen eines Elementes `x` in AVL-Bäume

Eingabe: ein AVL-Baum `t`, ein Element `x`

Ausgabe: ein AVL-Baum, der die Elemente aus `t` und das Element `x` enthält

Vorgehen:

1. Füge das neue Element `x` mit Balancefaktor 0 als direkten Nachfolger des Knotens `n` als Blatt ein, so dass die Suchbaumeigenschaft erfüllt ist. Aktualisiere `n.balance`.
2. Setze `n` auf den Vorgängerknoten von `n`. (Beachte: `n` ist vom Typ `Ptr`!)
   (a) Falls `x` im linken Unterbaum von `n` eingefügt wurde
      (i) wenn `n->balance == 1` dann `n->balance = 0` und gehe nach 3.
      (ii) wenn `n->balance == 0`, dann `n->balance = -1` und gehe nach 2.
      (iii) wenn `n->balance == -1` und
         - wenn `n->left->balance == -1` dann Rechtsrotation um `n`
         - wenn `n->left->balance == 1`, dann erst eine Linksrotation um `n->left`, dann eine Rechtsrotation um `n`.

   (b) Falls `x` im rechten Unterbaum von `n` eingeführt wurde
      (i) wenn `n->balance == -1`, dann `n->balance = 0` und gehe nach 3.
      (ii) wenn `n->balance == 0`, dann `n->balance = 1` und gehe nach 2.
      (iii) wenn `n->balance == 1` und
         - wenn `n->right->balance == 1` dann Linksrotation um `n`
         - wenn `n->right->balance == -1` dann erst eine Rechtsrotation um `n->right`, dann eine Linksrotation um `n`.

AVL-Bäume

Algorithmus Einfügen eines Elementes `x` in AVL-Bäume

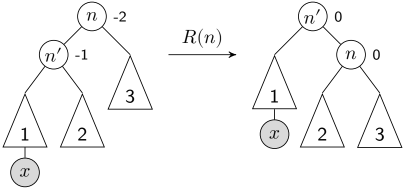
Eingabe: ein AVL-Baum `t`, ein Element `x`

Ausgabe: ein AVL-Baum, der die Elemente aus `t` und das Element `x` enthält

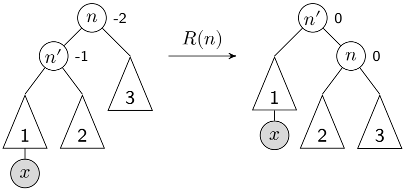
Vorgehen:

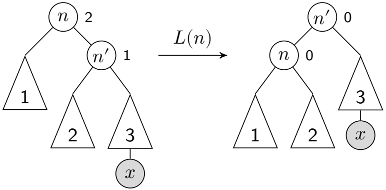
1. Füge das neue Element `x` mit Balancefaktor 0 als direkten Nachfolger des Knotens `n` als Blatt ein, so dass die Suchbaumeigenschaft erfüllt ist. Aktualisiere `n.balance`.
2. Setze `n` auf den Vorgängerknoten von `n`. (Beachte: `n` ist vom Typ `Ptr`!)
   (a) Falls `x` im linken Unterbaum von `n` eingefügt wurde
      (i) wenn `n->balance == 1` dann `n->balance = 0` und gehe nach 3.
      (ii) wenn `n->balance == 0`, dann `n->balance = -1` und gehe nach 2.
      (iii) wenn `n->balance == -1` und
         - wenn `n->left->balance == -1` dann Rechtsrotation um `n`
         - wenn `n->left->balance == 1`, dann erst eine Linksrotation um `n->left`, dann eine Rechtsrotation um `n`.
   (b) Falls `x` im rechten Unterbaum von `n` eingeführt wurde
      (i) wenn `n->balance == -1`, dann `n->balance = 0` und gehe nach 3.
      (ii) wenn `n->balance == 0`, dann `n->balance = 1` und gehe nach 2.
      (iii) wenn `n->balance == 1` und
         - wenn `n->right->balance == 1` dann Linksrotation um `n`
         - wenn `n->right->balance == -1` dann erst eine Rechtsrotation um `n->right`, dann eine Linksrotation um `n`.
3. Gehe zurück zur Wurzel.

AVL-Bäume

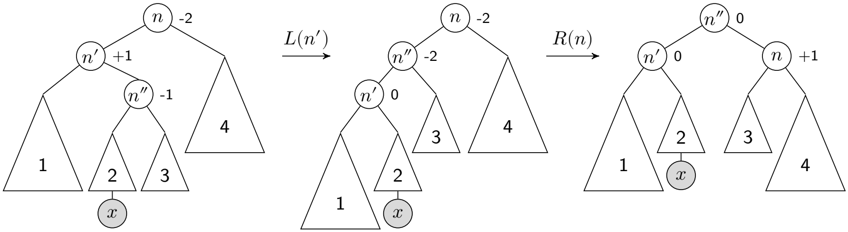


AVL-Bäume

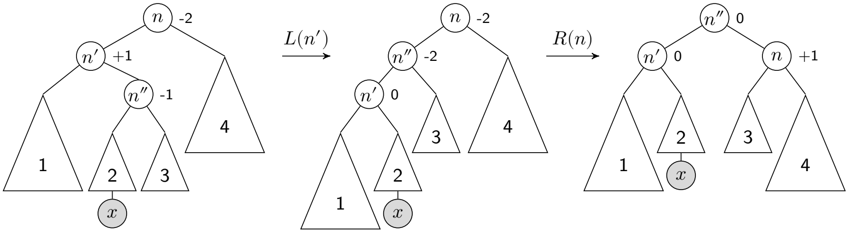




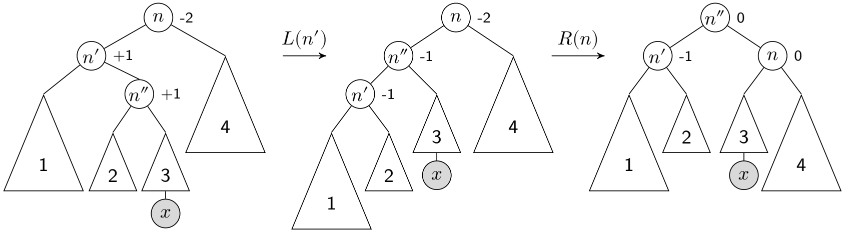
AVL-Bäume



AVL-Bäume

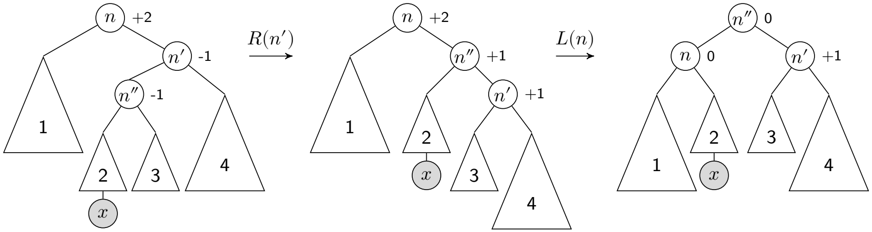


n -2
n' +1
n'' -1
1
2
3
4
x
$L(n')$
n -2
n'' -2
n' 0
1
2
3
4
x
$R(n)$
n'' 0
n' 0
n +1
1
2
3
4
x

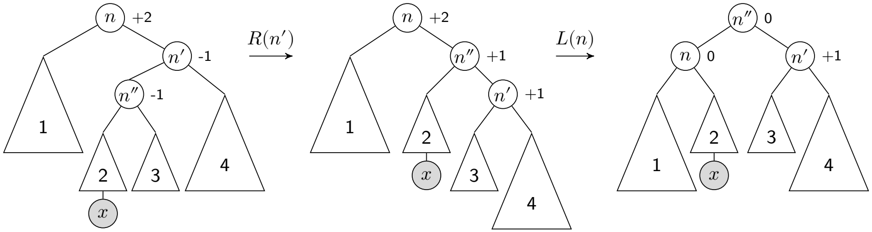


n -2
n' +1
n'' +1
1
2
3
4
x
$L(n')$
n -2
n'' -1
n' -1
1
2
3
4
x
$R(n)$
n'' 0
n' -1
n 0
1
2
3
4
x

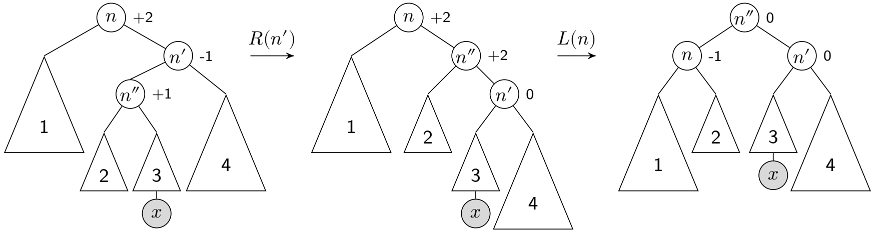
AVL-Bäume



AVL-Bäume



n +2
n′ -1
n″ -1
1
2
3
4
x
R(n′)
n +2
n″ +1
n′ +1
1
2
x
3
4
L(n)
n″ 0
n 0
n′ +1
1
2
x
3
4



n +2
n′ -1
n″ +1
1
2
3
x
4
R(n′)
n +2
n″ +2
n′ 0
1
2
3
x
4
L(n)
n″ 0
n -1
n′ 0
1
2
3
x
4

AVL-Bäume

```
typedef struct Nodeelem *Ptr;          /* Datenstruktur des AVL-Baumes */

typedef struct Nodeelem { int key;
                          short balance;
                          Ptr left, right; } Node;

typedef enum teilbaum{L,R} zweig;

void einfuegen(Ptr *t, int x);         /* Funktion siehe vorn */

Ptr zeiger_von_x(Ptr z_Node, int x);   /* Ermittelt den Zeiger des Knotens im AVL-
                                          Baum, welcher x als Schlüssel hat */

Ptr vorg(Ptr z_Node, Ptr n, zweig *z); /* Ermittelt den Zeiger auf den Vorgänger des
                                          Knotens auf den n zeigt und gibt den Zweig
                                          (Teilbaumseite) an, in dem n liegt, also
                                          *z == L oder *z == R. Gibt es keinen Vor-
                                          gängerknoten, so ist die Rückgabe NULL. */

void rot(Ptr *z_Node, Ptr n, zweig y); /* Führt die Rotation um den Knoten mit Zeiger
                                          n entsprechend Vorschrift aus: y == 'R'
                                          Rechts-, y == 'L' Linksrotation. */

/* Beachte: Dabei kann der Kopfzeiger des (alten) AVL-Baumes geändert werden! */
```

AVL-Bäume

```c
void einfuegen_AVL(Ptr *wurzel, int x)
{ zweig TB; Ptr n;
  . . .
  einfuegen(wurzel, x);
  n = zeiger_von_x(*wurzel, x); n->balance = 0; n = vorg(*wurzel, n, &TB);

  while (n != NULL)
    if (TB == L)
      switch (n->balance)
      { case  1: n->balance = 0; return;
        case  0: n->balance = -1; n = vorg(*wurzel, n, &TB); break;
        case -1: switch (n->left->balance)
                 { case  -1: rot(wurzel, n, R); return;
                   case   1: rot(wurzel, n->left, L);
                             rot(wurzel, n, R); return;
                 }
      }
    else   /* hier gilt TB == R */
      switch (n->balance)
      { case -1: n->balance = 0; return;
        case  0: n->balance = 1, n = vorg(*wurzel, n, &TB);break;
        case  1: switch (n->right->balance)
                 { case  1: rot(wurzel, n, L); return;
                   case -1: rot(wurzel, n->right, R);
                            rot(wurzel, n, L); return;
                 }
      }
}
```

```
void f_balance(Ptr z_Node)
{ if (z_Node == NULL) return;
  z_Node->balance = hoehe(z_Node->right) - hoehe(z_Node->left);
  f_balance(z_Node->right);
  f_balance(z_Node->left);
}
```

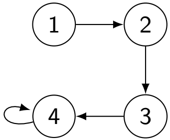
```
f_balance(WURZEL);            /* trägt die Balancewerte in
                                 den initialen Baum ein */
einfuegen_AVL(&WURZEL,WERT);
```

Inhaltsverzeichnis I

Inhaltsverzeichnis II

Graphen

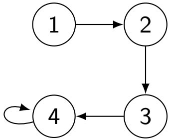
- **gerichteter Graph** $G = (V, E)$
  V endliche Menge (Knoten), oft: $V = \{1, \dots, n\} \subseteq \mathbb{N}$
  E endliche Menge (Kanten)



Graphen

- **gerichteter Graph** $G = (V, E)$
  V endliche Menge (Knoten), oft: $V = \{1, \ldots, n\} \subseteq \mathbb{N}$
  E endliche Menge (Kanten)

- $(v, v) \in E$ **(Schlinge)**

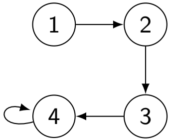


Graphen

- **gerichteter Graph** $G = (V, E)$
  V endliche Menge (Knoten), oft: $V = \{1, \dots, n\} \subseteq \mathbb{N}$
  E endliche Menge (Kanten)

- $(v, v) \in E$ **(Schlinge)**

- $G' = (V', E')$ **Teilgraph von** $G = (V, E)$,
  wenn $V' \subseteq V$ und $E' \subseteq E \cap (V' \times V')$



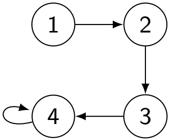
Graphen

- **gerichteter Graph** $G = (V, E)$
  V endliche Menge (Knoten), oft: $V = \{1, \dots, n\} \subseteq \mathbb{N}$
  E endliche Menge (Kanten)

- $(v, v) \in E$ **(Schlinge)**

- $G' = (V', E')$ **Teilgraph von** $G = (V, E)$,
  wenn $V' \subseteq V$ und $E' \subseteq E \cap (V' \times V')$

- $G = (V, E)$ heißt **azyklisch**,
  wenn es keine Knoten v gibt mit $vE^{+}v$.



Graphen

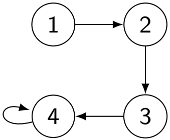
- **gerichteter Graph** $G = (V, E)$
  V endliche Menge (Knoten), oft: $V = \{1, \dots, n\} \subseteq \mathbb{N}$
  E endliche Menge (Kanten)

- $(v, v) \in E$ **(Schlinge)**

- $G' = (V', E')$ **Teilgraph von** $G = (V, E)$,
  wenn $V' \subseteq V$ und $E' \subseteq E \cap (V' \times V')$

- $G = (V, E)$ heißt **azyklisch**,
  wenn es keine Knoten v gibt mit vE^+v .

- **Weg von** v **nach** v': $(v_1, \dots, v_n)$
  $v_i \in V$, $v_1 = v$, $v_n = v'$
  und für jedes $1 \leq i \leq n-1$: $(v_i, v_{i+1}) \in E$
  z.B.: (v) ist Weg von v nach v



Graphen

- **gerichteter Graph** $G = (V, E)$
  V endliche Menge (Knoten), oft: $V = \{1, \dots, n\} \subseteq \mathbb{N}$
  E endliche Menge (Kanten)

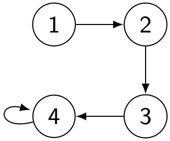
- $(v, v) \in E$ **(Schlinge)**

- $G' = (V', E')$ **Teilgraph von** $G = (V, E)$,
  wenn $V' \subseteq V$ und $E' \subseteq E \cap (V' \times V')$

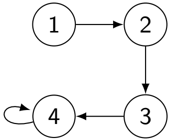
- $G = (V, E)$ heißt **azyklisch**,
  wenn es keine Knoten v gibt mit vE^+v .

- **Weg von** v **nach** v': $(v_1, \dots, v_n)$
  $v_i \in V$, $v_1 = v$, $v_n = v'$
  und für jedes $1 \leq i \leq n-1$: $(v_i, v_{i+1}) \in E$
  z.B.: (v) ist Weg von v nach v

- **innere Kanten eines Weges** $(v_1, \dots, v_n)$: $v_2, \dots, v_{n-1}$



Graphen



- **gerichteter Graph** $G = (V, E)$
  V endliche Menge (Knoten), oft: $V = \{1, \dots, n\} \subseteq \mathbb{N}$
  E endliche Menge (Kanten)

- $(v, v) \in E$ **(Schlinge)**

- $G' = (V', E')$ **Teilgraph von** $G = (V, E)$,
  wenn $V' \subseteq V$ und $E' \subseteq E \cap (V' \times V')$

- $G = (V, E)$ heißt **azyklisch**,
  wenn es keine Knoten v gibt mit vE^+v .

- **Weg von** v **nach** v': $(v_1, \dots, v_n)$
  $v_i \in V$, $v_1 = v$, $v_n = v'$
  und für jedes $1 \leq i \leq n-1$: $(v_i, v_{i+1}) \in E$
  z.B.: (v) ist Weg von v nach v

- **innere Kanten eines Weges** $(v_1, \dots, v_n)$: $v_2, \dots, v_{n-1}$

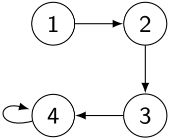
- Graph $G = (V, E)$ ist **ungerichtet**,
  wenn für alle $v, v' \in V$ gilt:
  - entweder es gibt keine Kante $(v, v') \in E$
    und es gibt keine Kante $(v', v) \in E$
  - oder $(v, v') \in E$ und $(v', v) \in E$.

Graphen: Adjazenzmatrix A_G von G

$(n \times n)$-Matrix über $\{0, 1\}$:

$$A_G(i,j) = \begin{cases} 0 & \text{falls } (i,j) \notin E \\ 1 & \text{falls } (i,j) \in E \end{cases}$$

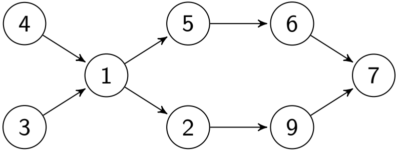
Beispiel:



	1	2	3	4
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	0	0	0	1

Topologisches Sortieren

Gegeben sei ein gerichteter, azyklischer Graph $G = (V, E)$. Eine topologische Sortierung von G ist eine bijektive Abbildung $ord : V \to \{1, \ldots, n\}$ mit $n = |V|$, so dass aus $(v, v') \in E$ folgt $ord(v) < ord(v')$.

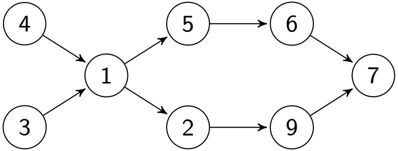


Algorithmus Topologisches Sortieren

```
while (Elemente sind noch übrig)
{ Wähle Element aus, welches keinen Vorgänger hat;
  Dekrementiere die Anzahl der Vorgänger in den Nachfolgern des ausgewählten
    Elements;
  Trage das Element in die gewünschte Ausgabeliste ein;
  Streiche das ausgewählte Element aus der Menge;
}
```

Algorithmus Topologisches Sortieren

```
while (Elemente sind noch übrig)
{ Wähle Element aus, welches keinen Vorgänger hat;
  Dekrementiere die Anzahl der Vorgänger in den Nachfolgern des ausgewählten
    Elements;
  Trage das Element in die gewünschte Ausgabeliste ein;
  Streiche das ausgewählte Element aus der Menge;
}
```



Topologisches Sortieren

```
typedef struct leader *LPtr;
typedef struct trailer *TPtr;

typedef struct leader
              { int key;
                int count;  /* Anzahl der Vorgaenger */
                TPtr trail; /* Liste mit Zeigern zu Nachfolgeelementen
                               bezuegl. der Halbordnung */
                LPtr next;  /* Zeiger zu Element, welches bzgl. seines
                               Erscheinens in der Eingabe das naechste ist */
              } leader;

typedef struct trailer
              { LPtr id;
                TPtr next;
              } trailer;

LPtr p, q, head, tail;
TPtr t;
```

Topologisches Sortieren

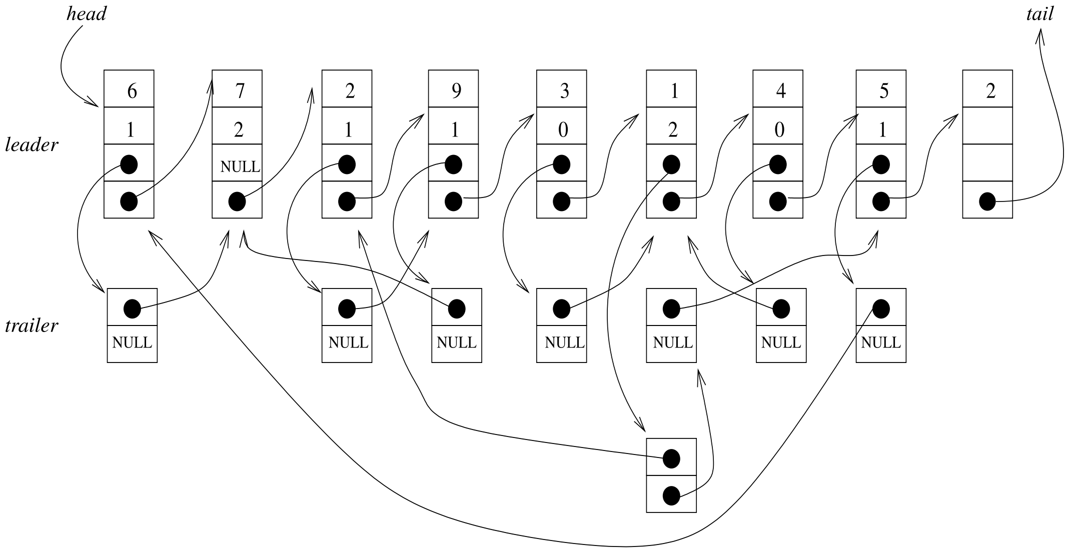
Wir gehen davon aus, dass die Zahlenpaare ohne Klammern und Kommata eingegeben werden und die Eingabe durch "999" abgeschlossen wird, d.h. 6 7 2 9 9 7 3 1 4 1 5 6 1 5 1 2 999.

```
LPtr find(int w)
{ LPtr h;

  h = head;  /* head ist erstes Element der Eingabeliste */
  tail->key = w;
  while (h->key != w) h = h->next; /* Suche Element und setze
                                      Zeiger h auf dieses Element */
  if (h == tail)
  { tail = (LPtr) malloc(sizeof(leader));
    n = n+1;
    h->count = 0;
    h->trail = NULL;
    h->next = tail;
  }
  return h;
}
```

. . .

```
int main()
{ int Done = 1;
    /* Eingabephase, Ende der Eingabe mit "999" als Anfangsknoten */
  head = (LPtr) malloc(sizeof(leader));
  tail = head;
  n = 0;
  printf("Anfangsknoten: ");
  scanf("%d", &x);

  while (Done)
  { printf("Endknoten: ");
    scanf("%d", &y);
    p = find(x);
    q = find(y);
    t = (TPtr) malloc(sizeof(trailer));
    t->id = q;
    t->next = p->trail;
    p->trail = t;
    q->count = q->count + 1;
    printf("\nAnfangsknoten: ");
    scanf("%d", &x);
    if (x == 999) Done = 0;
  }
}
```

Topologisches Sortieren



Zahlenpaare:

$$(6,7)(2,9)(9,7)(3,1)(4,1)(5,6)(1,5)(1,2)$$

Suche Leader ohne Vorgänger

```
p = head;
head = NULL;
while (p != tail)
{ q = p;
  p = q->next;
  if (q->count == 0)
  { q->next = head;
    head = q;
  }
}
```

Ausgabe der topologisch sortierten Folge von Schlüsseln

```
q = head;
printf("Eingebettete lineare Ordnung:\n");
while (q != NULL)   /* drucke dieses Element, lösche es dann */
{ printf(" %d ", q->key);
  n = n-1;
  t = q->trail;
  q = q->next;
    /* verringere den Vorgängerzähler jedes Elements in der
       Liste t der trailer; wird ein Zähler 0, so wird dieses
       Element an den Anfang der Liste q der Leader genommen. */
}

while (t != NULL)
{ p = t->id;
  p->count = p->count-1;
  if (p->count == 0)
  { p->next = q;
    q = p;
  }
  t = t->next;
}
```

```c
/* TopSort */
#include <stdio.h>
#include <stdlib.h>

typedef struct leader *LPtr;
typedef struct trailer *TPtr;

typedef struct leader
               { int key;
                 int count;  /* Anzahl der Vorgaenger */
                 TPtr trail; /* Liste mit Zeigern zu Nachfolgeelementen
                                bezuegl. der Halbordnung */
                 LPtr next;  /* Zeiger zu Element, welches bzgl. seines
                                Erscheinens in der Eingabe das naechste ist */
               } leader;

typedef struct trailer
               { LPtr id;
                 TPtr next;
               } trailer;

LPtr p, q, head, tail;
TPtr t;
int x, y, n;
```

...

...

```
LPtr find(int w)
{ LPtr h;

  h = head;
  tail->key = w;
  while (h->key != w) h = h->next;
  if (h == tail)
  { tail = (LPtr) malloc(sizeof(leader));
    n = n+1;
    h->count = 0;
    h->trail = NULL;
    h->next = tail;
  }
  return h;
}

int main()
{ int Done = 1;
    /* Eingabephase, Ende der Eingabe mit "999" als Anfangsknoten */
  head = (LPtr) malloc(sizeof(leader));
  tail = head;
  n = 0;
  printf("Anfangsknoten: ");
  scanf("%d", &x);
```

...

...

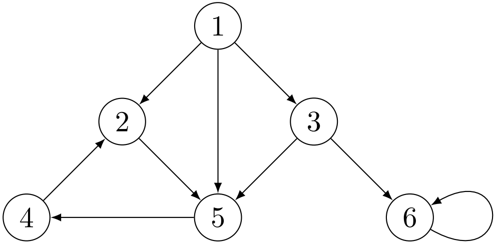
```
  while (Done)
  { printf("Endknoten: ");
    scanf("%d", &y);
    p = find(x);
    q = find(y);
    t = (TPtr) malloc(sizeof(trailer));
    t->id = q;
    t->next = p->trail;
    p->trail = t;
    q->count = q->count+1;
    printf("\nAnfangsknoten: ");
    scanf("%d", &x);
    if (x == 999) Done = 0;
  }

  /* Suche nach Leader ohne Vorgänger */
  p = head;
  head = NULL;
  while (p != tail)
  { q = p;
    p = q->next;
    if (q->count == 0)
    { q->next = head;
      head = q;
    }
  }
```

...

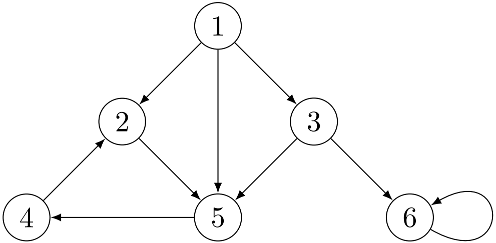
...

```

   /* Ausgabephase */
   q = head;
   printf("Eingebettete lineare Ordnung:\n");
   while (q != NULL) /* drucke jeweils ein Element ohne Vorgaenger */
   { printf(" %d ", q->key);
     n = n-1;
     t = q->trail;
     q = q->next;
     while (t != NULL)
     { p = t->id;
       p->count = p->count-1;
       if (p->count == 0)
       { p->next = q;
         q = p;
       }
       t = t->next;
     }
   }

   if (n != 0)
     printf("\nDiese Liste beschreibt keine partielle Ordnung");
   printf("\n\n");
}
```

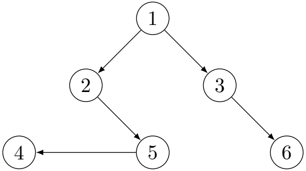
Breiten- und Tiefensuche in Graphen



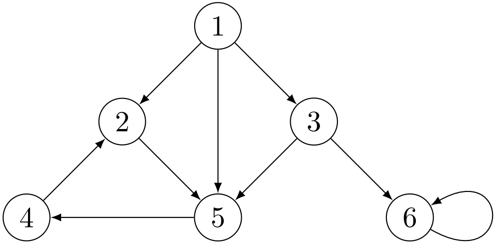
Breiten- und Tiefensuche in Graphen



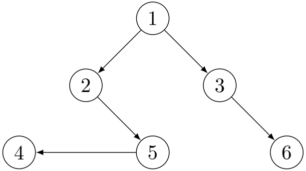
Tiefenbaum (depth-first tree)



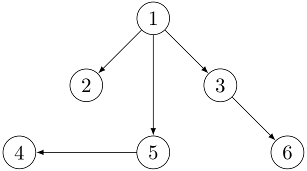
Breiten- und Tiefensuche in Graphen



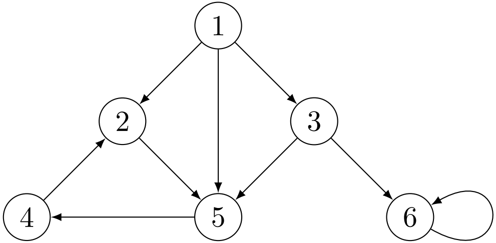
Tiefenbaum (depth-first tree)



Breitenbaum (breadth-first tree)

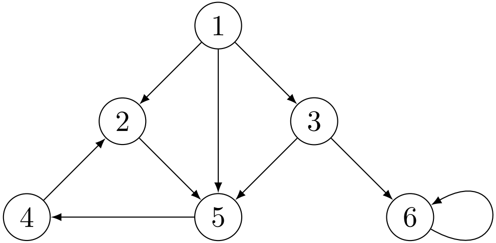


Tiefensuche



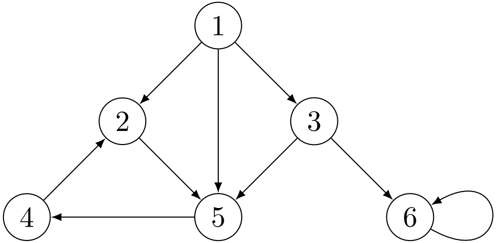
Tiefensuche

Iteration	Entdeckt	Depth-First Tree
0.	$[(0, 1)]$	

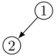


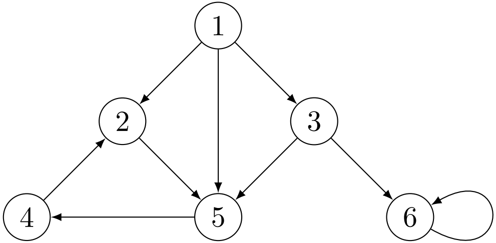
Tiefensuche

Iteration	Entdeckt	Depth-First Tree
0.	$[(0,1)]$	
1.	$[(1,2),(1,5),(1,3)]$	1

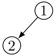
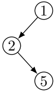


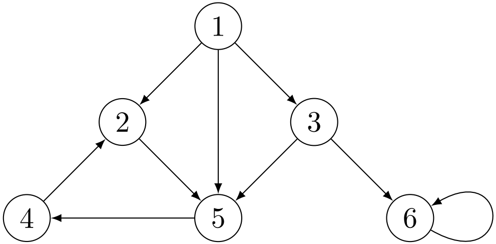
Tiefensuche

Iteration	Entdeckt	Depth-First Tree
0.	$[(0,1)]$	
1.	$[(1,2),(1,5),(1,3)]$	
2.	$[(2,5),(1,5),(1,3)]$	

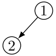
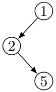
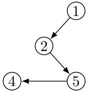


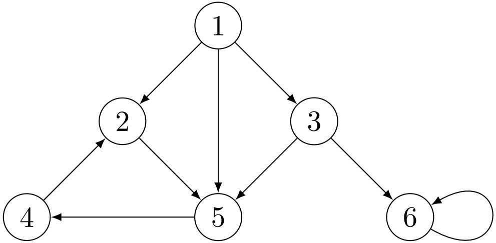
Tiefensuche

Iteration	Entdeckt	Depth-First Tree
0.	$[(0,1)]$	
1.	$[(1,2),(1,5),(1,3)]$	1
2.	$[(2,5),(1,5),(1,3)]$	
3.	$[(5,4),(1,5),(1,3)]$	

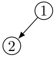
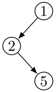
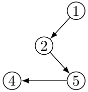
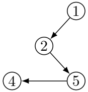


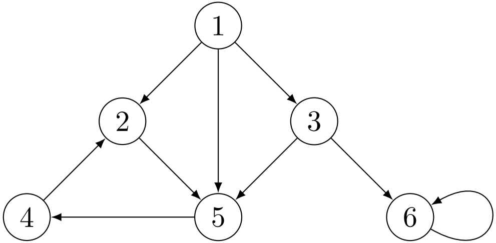
Tiefensuche

Iteration	Entdeckt	Depth-First Tree
0.	$[(0,1)]$	
1.	$[(1,2),(1,5),(1,3)]$	
2.	$[(2,5),(1,5),(1,3)]$	
3.	$[(5,4),(1,5),(1,3)]$	
4.	$[(1,5),(1,3)]$	

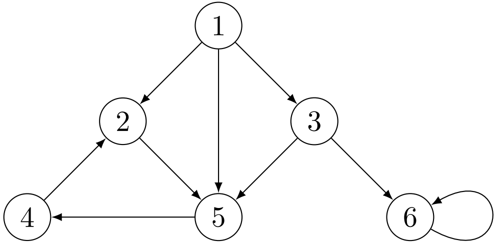


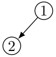
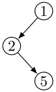
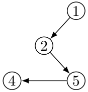
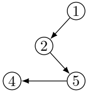
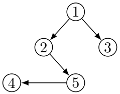
Tiefensuche

Iteration	Entdeckt	Depth-First Tree
0.	$[(0,1)]$	
1.	$[(1,2),(1,5),(1,3)]$	
2.	$[(2,5),(1,5),(1,3)]$	
3.	$[(5,4),(1,5),(1,3)]$	
4.	$[(1,5),(1,3)]$	
5.	$[(1,3)]$	

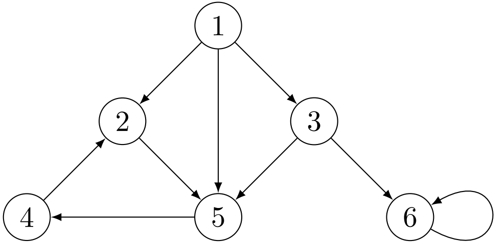


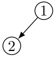
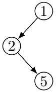
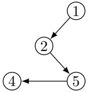
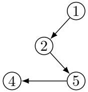
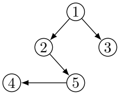
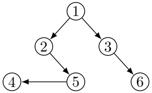
Tiefensuche



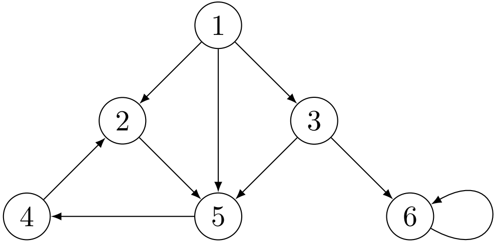
Iteration	Entdeckt	Depth-First Tree
0.	$[(0,1)]$	
1.	$[(1,2),(1,5),(1,3)]$	1
2.	$[(2,5),(1,5),(1,3)]$	
3.	$[(5,4),(1,5),(1,3)]$	
4.	$[(1,5),(1,3)]$	
5.	$[(1,3)]$	
6.	$[(3,6)]$	

Tiefensuche



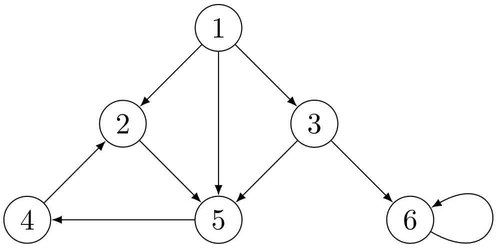
Iteration	Entdeckt	Depth-First Tree
0.	$[(0,1)]$	
1.	$[(1,2),(1,5),(1,3)]$	
2.	$[(2,5),(1,5),(1,3)]$	
3.	$[(5,4),(1,5),(1,3)]$	
4.	$[(1,5),(1,3)]$	
5.	$[(1,3)]$	
6.	$[(3,6)]$	
7.	$[]$	

Breitensuche



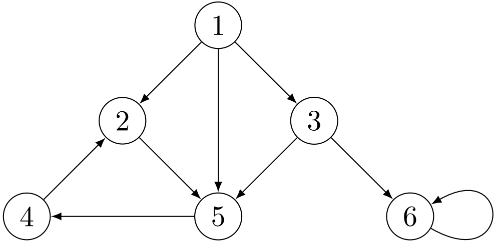
Breitensuche

Iteration	Entdeckt	Breadth-First Tree
0.	$[(0,1)]$	

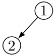


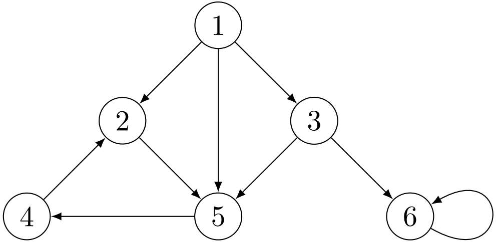
Breitensuche

Iteration	Entdeckt	Breadth-First Tree
0.	$[(0,1)]$	
1.	$[(1,2),(1,5),(1,3)]$	1

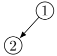
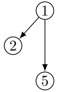


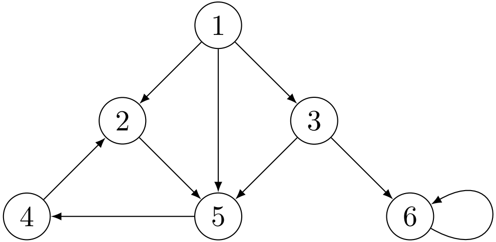
Breitensuche

Iteration	Entdeckt	Breadth-First Tree
0.	$[(0,1)]$	
1.	$[(1,2),(1,5),(1,3)]$	1
2.	$[(1,5),(1,3),(2,5)]$	

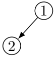
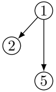
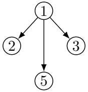


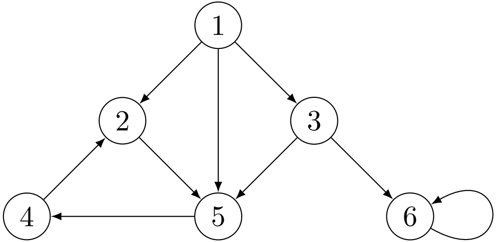
Breitensuche

Iteration	Entdeckt	Breadth-First Tree
0.	$[(0,1)]$	
1.	$[(1,2),(1,5),(1,3)]$	1
2.	$[(1,5),(1,3),(2,5)]$	
3.	$[(1,3),(2,5),(5,4)]$	

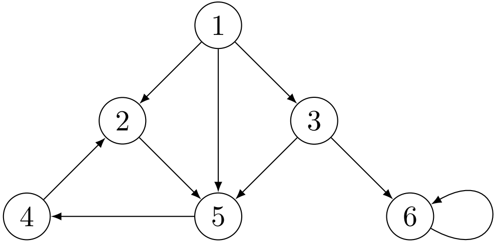


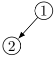
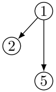
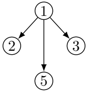
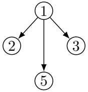
Breitensuche

Iteration	Entdeckt	Breadth-First Tree
0.	$[(0,1)]$	
1.	$[(1,2),(1,5),(1,3)]$	
2.	$[(1,5),(1,3),(2,5)]$	
3.	$[(1,3),(2,5),(5,4)]$	
4.	$[(2,5),(5,4),(3,6)]$	

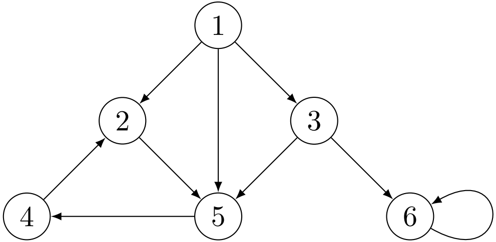


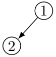
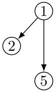
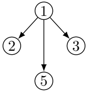
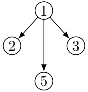
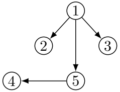
Breitensuche



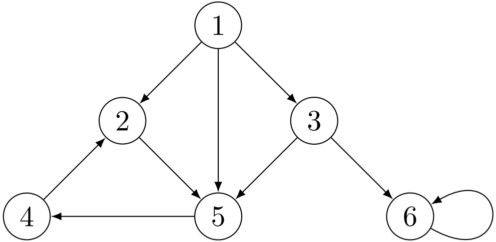
Iteration	Entdeckt	Breadth-First Tree
0.	$[(0,1)]$	
1.	$[(1,2),(1,5),(1,3)]$	
2.	$[(1,5),(1,3),(2,5)]$	
3.	$[(1,3),(2,5),(5,4)]$	
4.	$[(2,5),(5,4),(3,6)]$	
5.	$[(5,4),(3,6)]$	

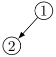
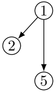
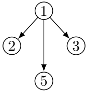
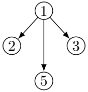
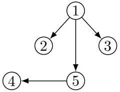
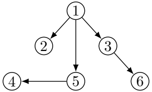
Breitensuche



Iteration	Entdeckt	Breadth-First Tree
0.	$[(0,1)]$	
1.	$[(1,2),(1,5),(1,3)]$	
2.	$[(1,5),(1,3),(2,5)]$	
3.	$[(1,3),(2,5),(5,4)]$	
4.	$[(2,5),(5,4),(3,6)]$	
5.	$[(5,4),(3,6)]$	
6.	$[(3,6)]$	

Breitensuche



Iteration	Entdeckt	Breadth-First Tree
0.	$[(0,1)]$	
1.	$[(1,2),(1,5),(1,3)]$	
2.	$[(1,5),(1,3),(2,5)]$	
3.	$[(1,3),(2,5),(5,4)]$	
4.	$[(2,5),(5,4),(3,6)]$	
5.	$[(5,4),(3,6)]$	
6.	$[(3,6)]$	
7.	$[]$	

Datentyp *Pushdown*

$$
\begin{aligned}
push\colon \quad & A^* \times A \to A^* \\
& (a_1 \cdots a_n, a) \mapsto a a_1 \cdots a_n \\
& \qquad\qquad\qquad \text{(für alle } n \in \mathbb{N}\text{)} \\
pop\colon \quad & A^* \to A^* \\
& a_1 a_2 \cdots a_n \mapsto a_2 \cdots a_n \\
& \qquad\qquad\qquad \text{(falls } n \geq 1\text{)} \\
top\colon \quad & A^* \to A \\
& a_1 \cdots a_n \mapsto a_1 \qquad \text{(falls } n \geq 1\text{)} \\
empty = \quad & \varepsilon \in A^*
\end{aligned}
$$

Datentyp *Pushdown*

$$
\begin{aligned}
\mathit{push}\colon \quad & A^* \times A \to A^* \\
& (a_1 \cdots a_n, a) \mapsto a a_1 \cdots a_n \\
& \qquad \text{(für alle } n \in \mathbb{N}) \\
\mathit{pop}\colon \quad & A^* \to A^* \\
& a_1 a_2 \cdots a_n \mapsto a_2 \cdots a_n \\
& \qquad \text{(falls } n \geq 1) \\
\mathit{top}\colon \quad & A^* \to A \\
& a_1 \cdots a_n \mapsto a_1 \qquad \text{(falls } n \geq 1) \\
\mathit{empty} = \quad & \varepsilon \in A^*
\end{aligned}
$$

Datentyp *Queue*

$$
\begin{aligned}
\mathit{enqueue}\colon \quad & A^* \times A \to A^* \\
& (a_1 \cdots a_n, a) \mapsto a_1 \cdots a_n a \\
& \qquad \text{(für alle } n \in \mathbb{N}) \\
\mathit{dequeue}\colon \quad & A^* \to A^* \\
& a_1 a_2 \cdots a_n \mapsto a_2 \cdots a_n \\
& \qquad \text{(falls } n \geq 1) \\
\mathit{head}\colon \quad & A^* \to A \\
& a_1 \cdots a_n \mapsto a_1 \qquad \text{(falls } n \geq 1) \\
\mathit{nil} = \quad & \varepsilon \in A^*
\end{aligned}
$$

Breiten- und Tiefensuche in Graphen

Datentyp *Pushdown*

$$\begin{aligned} push: \quad & A^* \times A \to A^* \\ & (a_1 \cdots a_n, a) \mapsto a a_1 \cdots a_n \\ & \qquad \text{(für alle } n \in \mathbb{N}) \\ pop: \quad & A^* \to A^* \\ & a_1 a_2 \cdots a_n \mapsto a_2 \cdots a_n \\ & \qquad \text{(falls } n \geq 1) \\ top: \quad & A^* \to A \\ & a_1 \cdots a_n \mapsto a_1 \quad \text{(falls } n \geq 1) \\ empty = \quad & \varepsilon \in A^* \end{aligned}$$

Datentyp *Queue*

$$\begin{aligned} enqueue: \quad & A^* \times A \to A^* \\ & (a_1 \cdots a_n, a) \mapsto a_1 \cdots a_n a \\ & \qquad \text{(für alle } n \in \mathbb{N}) \\ dequeue: \quad & A^* \to A^* \\ & a_1 a_2 \cdots a_n \mapsto a_2 \cdots a_n \\ & \qquad \text{(falls } n \geq 1) \\ head: \quad & A^* \to A \\ & a_1 \cdots a_n \mapsto a_1 \quad \text{(falls } n \geq 1) \\ nil = \quad & \varepsilon \in A^* \end{aligned}$$

Abstrakter Typ *STORAGE*

	`STORAGE`	`EMPTY`	`INSERT`	`REMOVE`	`READ`
Tiefensuche	$Pushdown$(`edge`)	$empty$	$push$	pop	top
Breitensuche	$Queue$(`edge`)	nil	$enqueue$	$dequeue$	$head$

```c
/* Datentyp zum Abspeichern von Kanten, d.h. Tupeln von Knoten */
typedef struct edge {
  int fst;
  int snd;
} edge;

/* Datentyp zum Abspeichern von gerichteten Graphen */
typedef struct graph { ... } graph;

/* add_edge(&G, e) fuegt dem Graphen G die Kante e, und, falls nicht enthalten, die
 * Knoten e.fst und e.snd hinzu. Ist e.fst == 0, wird ein neuer Graph erzeugt, der
 * als einzigen Knoten e.snd enthaelt. */
void add_edge(graph* G, edge e);

/* empty_graph() erstellt einen neuen, leeren Graphen */
graph empty_graph();

/* contains(G, v) gibt 1 zurueck, falls der Graph G den Knoten v enthaelt, sonst 0 */
int contains(graph G, int v);
```

graph.h

```c
typedef struct Pushdown { ... } STORAGE;
const STORAGE EMPTY = ... ;     /* empty: leerer Pushdown */

/* push: INSERT(S, e) fügt Kante e als oberstes Element in S ein */
STORAGE INSERT(STORAGE S, edge e);

/* pop: REMOVE(S) entfernt oberstes Element eines nichtleeren Pushdowns S */
STORAGE REMOVE(STORAGE S);

/* top: READ(S) liest oberstes Element eines nichtleeren Pushdowns S */
edge READ(STORAGE S);
```

pushdown.h

```c
typedef struct Queue { ... } STORAGE;
const STORAGE EMPTY = ...;      /* nil: leere Queue */

/* enqueue: INSERT(S, e) fügt Kante e als letztes Element in S ein */
STORAGE INSERT(STORAGE S, edge e);

/* dequeue: REMOVE(S) entfernt erstes Element einer nichtleeren Queue S */
STORAGE REMOVE(STORAGE S);

/* head: READ(S) liest erstes Element einer nichtleeren Queue S */
edge READ(STORAGE S);
```

queue.h

```c
/* Funktion zur generalisierten Graphensuche */
graph graphsearch(graph G, int s)
{ graph t = empty_graph();              /* Aufzubauender Suchbaum */
  STORAGE S = EMPTY;                    /* Datentyp von Kanten zu entdeckten Knoten */
  edge e = {0, s};                      /* Kein Vorgaengerknoten: e.fst == 0 */
  S = INSERT(S, e);

  while (S != EMPTY) {
    e = READ(S);
    S = REMOVE(S);
    if (!contains(t, e.snd)) {          /* e.snd noch nicht besucht */
      add_edge(&t, e);                  /* e.snd ist besucht */
      for (all successors v of e.snd in G)
        if (!contains(t, v)) {          /* Nur unbesuchte Knoten hinzufuegen */
          edge f = {e.snd, v};
          S = INSERT(S, f);             /* v ist entdeckt */
        }
    }
  }
  return t;
}
```

graphsearch.c

```c
/* Implementierung der Tiefensuche */
#include "graph.h"
#include "pushdown.h"     /* Instanziierung von STORAGE mit Pushdown */
#include "graphsearch.c"

graph dfs(graph G, int s) { return graphsearch(G, s); }
```

dfs.c

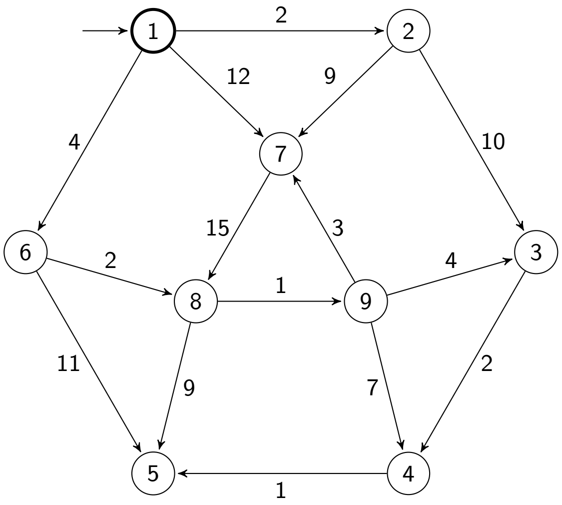
```c
/* Implementierung der Breitensuche */
#include "graph.h"
#include "queue.h"        /* Instanziierung von STORAGE mit Queue */
#include "graphsearch.c"

graph bfs(graph G, int s) { return graphsearch(G, s); }
```

bfs.c

Kürzeste Wege

Distanzgraph $G = (V, E, c)$
besteht aus:

- gerichteter Graph (V, E)
  mit $V = \{1, ..., n\}$
- Abbildung $c \colon E \to \mathbb{R}_{\geq 0}$
  mit $\mathbb{R}_{\geq 0} = \{r \in \mathbb{R} \mid r \geq 0\}$



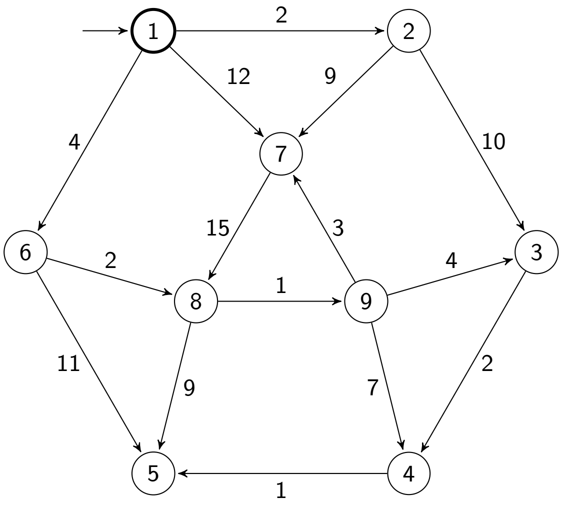
Kürzeste Wege

Distanzgraph $G = (V, E, c)$
besteht aus:

- gerichteter Graph (V, E)
  mit $V = \{1, ..., n\}$
- Abbildung $c\colon E \to \mathbb{R}_{\geq 0}$
  mit $\mathbb{R}_{\geq 0} = \{r \in \mathbb{R} \mid r \geq 0\}$

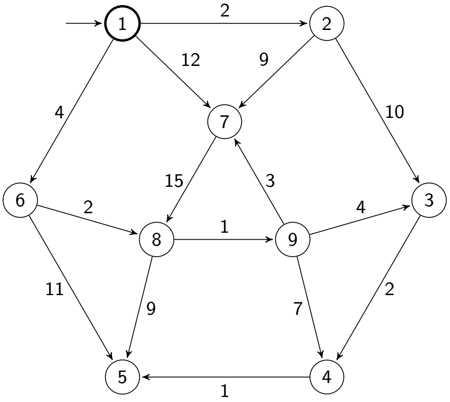
Adjazenzmatrix A_G von G:
$n \times n$-Matrix über $\mathbb{R}^{\infty}_{\geq 0} = \mathbb{R}_{\geq 0} \cup \{\infty\}$ mit

$$A_G(u, v) = \begin{cases} c(u, v) & \text{wenn } (u, v) \in E \\ \infty & \text{sonst} \end{cases}$$



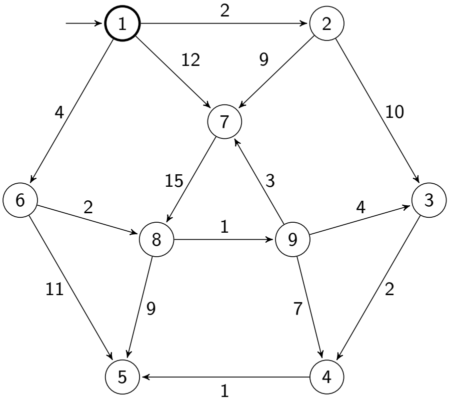
Kürzeste Wege

gewählt	Menge der Randknoten



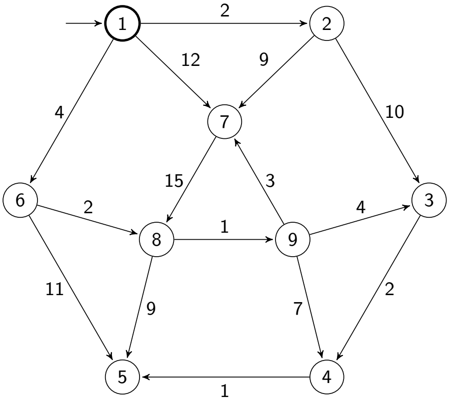
Kürzeste Wege

gewählt	Menge der Randknoten
$(1, 0, -)$	



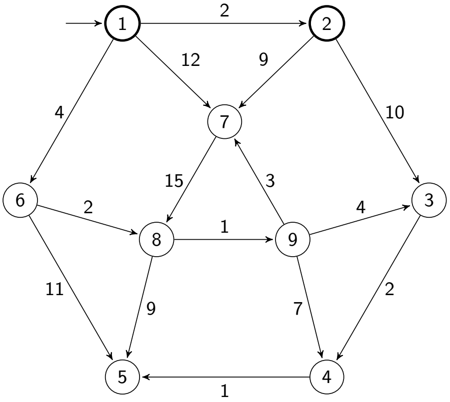
Kürzeste Wege

gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$



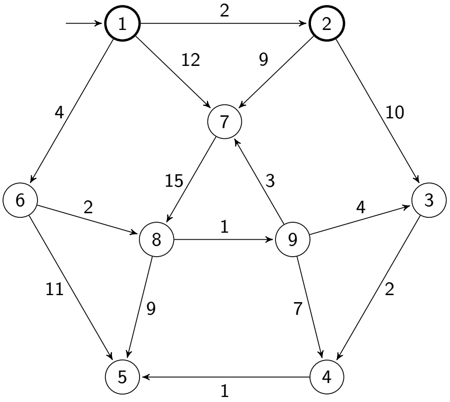
Kürzeste Wege

gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	



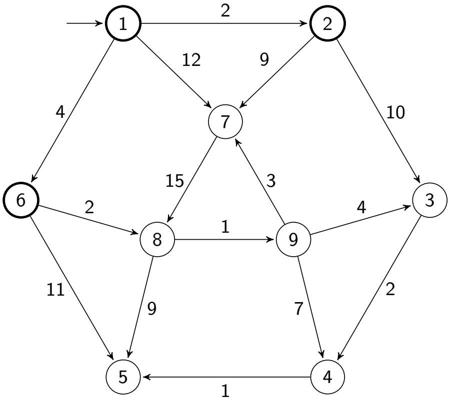
Kürzeste Wege

gewählt	Menge der Randknoten
$(1,0,-)$	$\{(2,2,1),(6,4,1),(7,12,1)\}$
$(2,2,1)$	$\{(3,12,2),(6,4,1),(7,11,2)\}$



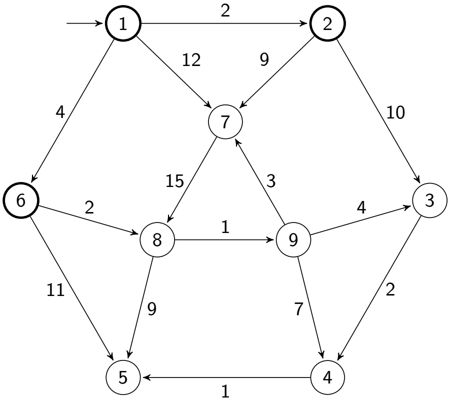
Kürzeste Wege

gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	$\{(3, 12, 2), (6, 4, 1), (7, 11, 2)\}$
$(6, 4, 1)$	



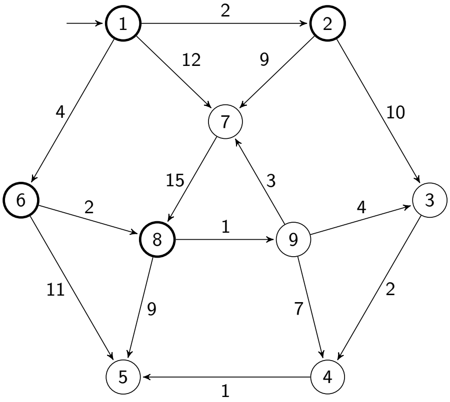
Kürzeste Wege

gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	$\{(3, 12, 2), (6, 4, 1), (7, 11, 2)\}$
$(6, 4, 1)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (8, 6, 6)\}$



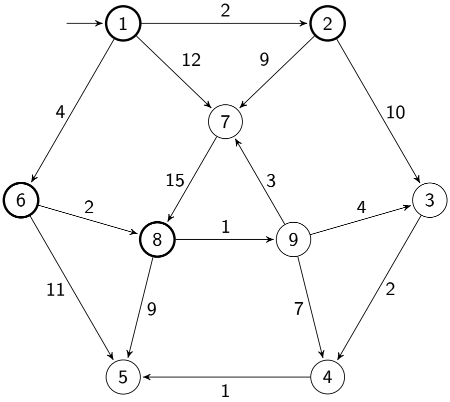
Kürzeste Wege

gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	$\{(3, 12, 2), (6, 4, 1), (7, 11, 2)\}$
$(6, 4, 1)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (8, 6, 6)\}$
$(8, 6, 6)$	



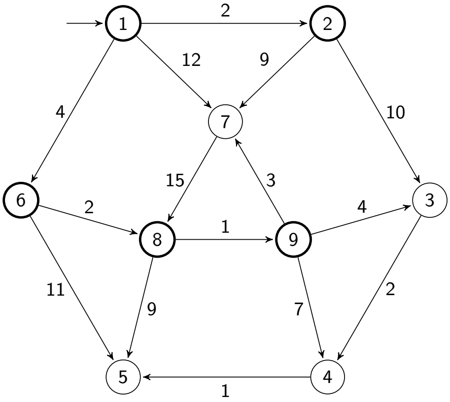
Kürzeste Wege

gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	$\{(3, 12, 2), (6, 4, 1), (7, 11, 2)\}$
$(6, 4, 1)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (8, 6, 6)\}$
$(8, 6, 6)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (9, 7, 8)\}$



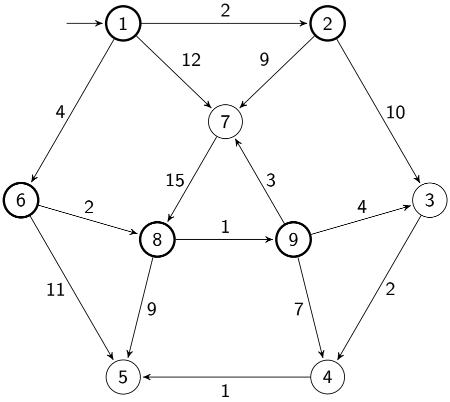
Kürzeste Wege

gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	$\{(3, 12, 2), (6, 4, 1), (7, 11, 2)\}$
$(6, 4, 1)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (8, 6, 6)\}$
$(8, 6, 6)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (9, 7, 8)\}$
$(9, 7, 8)$	



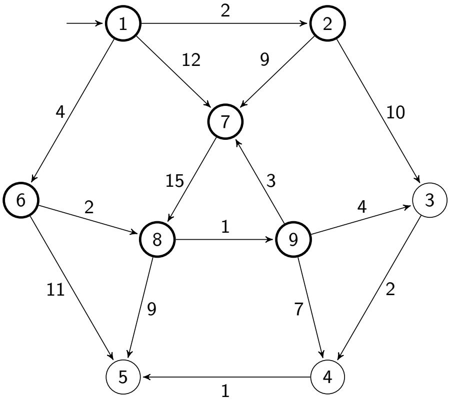
Kürzeste Wege

gewählt	Menge der Randknoten
$(1,0,-)$	$\{(2,2,1),(6,4,1),(7,12,1)\}$
$(2,2,1)$	$\{(3,12,2),(6,4,1),(7,11,2)\}$
$(6,4,1)$	$\{(3,12,2),(5,15,6),(7,11,2),(8,6,6)\}$
$(8,6,6)$	$\{(3,12,2),(5,15,6),(7,11,2),(9,7,8)\}$
$(9,7,8)$	$\{(3,11,9),(4,14,9),(5,15,6),(7,10,9)\}$



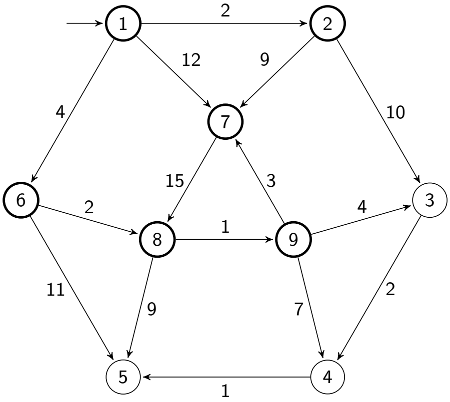
Kürzeste Wege

gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	$\{(3, 12, 2), (6, 4, 1), (7, 11, 2)\}$
$(6, 4, 1)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (8, 6, 6)\}$
$(8, 6, 6)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (9, 7, 8)\}$
$(9, 7, 8)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6), (7, 10, 9)\}$
$(7, 10, 9)$	



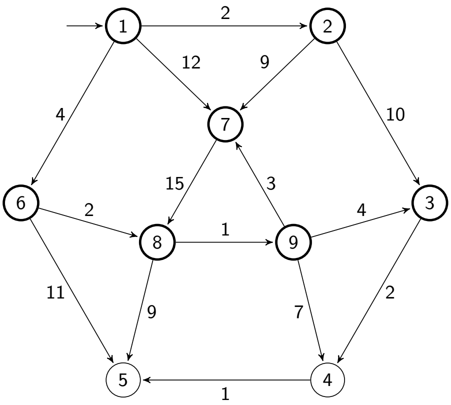
Kürzeste Wege

gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	$\{(3, 12, 2), (6, 4, 1), (7, 11, 2)\}$
$(6, 4, 1)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (8, 6, 6)\}$
$(8, 6, 6)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (9, 7, 8)\}$
$(9, 7, 8)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6), (7, 10, 9)\}$
$(7, 10, 9)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6)\}$



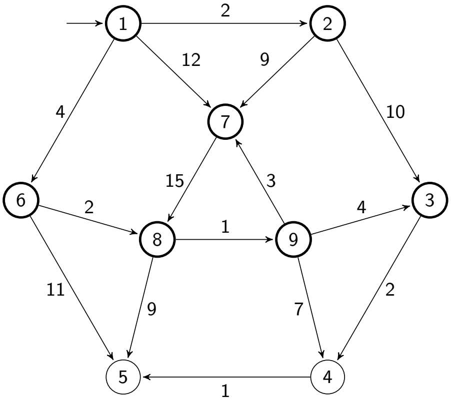
Kürzeste Wege

gewählt	Menge der Randknoten
$(1,0,-)$	$\{(2,2,1),(6,4,1),(7,12,1)\}$
$(2,2,1)$	$\{(3,12,2),(6,4,1),(7,11,2)\}$
$(6,4,1)$	$\{(3,12,2),(5,15,6),(7,11,2),(8,6,6)\}$
$(8,6,6)$	$\{(3,12,2),(5,15,6),(7,11,2),(9,7,8)\}$
$(9,7,8)$	$\{(3,11,9),(4,14,9),(5,15,6),(7,10,9)\}$
$(7,10,9)$	$\{(3,11,9),(4,14,9),(5,15,6)\}$
$(3,11,9)$	



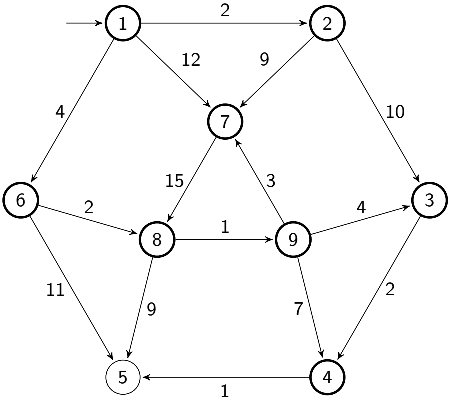
Kürzeste Wege

gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	$\{(3, 12, 2), (6, 4, 1), (7, 11, 2)\}$
$(6, 4, 1)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (8, 6, 6)\}$
$(8, 6, 6)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (9, 7, 8)\}$
$(9, 7, 8)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6), (7, 10, 9)\}$
$(7, 10, 9)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6)\}$
$(3, 11, 9)$	$\{(4, 13, 3), (5, 15, 6)\}$



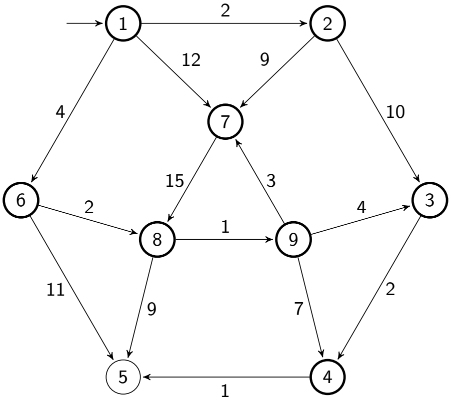
Kürzeste Wege

gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	$\{(3, 12, 2), (6, 4, 1), (7, 11, 2)\}$
$(6, 4, 1)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (8, 6, 6)\}$
$(8, 6, 6)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (9, 7, 8)\}$
$(9, 7, 8)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6), (7, 10, 9)\}$
$(7, 10, 9)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6)\}$
$(3, 11, 9)$	$\{(4, 13, 3), (5, 15, 6)\}$
$(4, 13, 3)$	



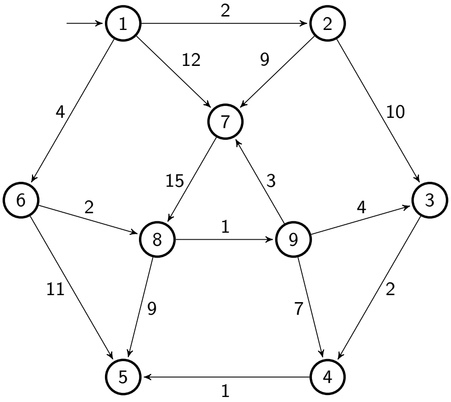
Kürzeste Wege

gewählt	Menge der Randknoten
$(1,0,-)$	$\{(2,2,1),(6,4,1),(7,12,1)\}$
$(2,2,1)$	$\{(3,12,2),(6,4,1),(7,11,2)\}$
$(6,4,1)$	$\{(3,12,2),(5,15,6),(7,11,2),(8,6,6)\}$
$(8,6,6)$	$\{(3,12,2),(5,15,6),(7,11,2),(9,7,8)\}$
$(9,7,8)$	$\{(3,11,9),(4,14,9),(5,15,6),(7,10,9)\}$
$(7,10,9)$	$\{(3,11,9),(4,14,9),(5,15,6)\}$
$(3,11,9)$	$\{(4,13,3),(5,15,6)\}$
$(4,13,3)$	$\{(5,14,4)\}$



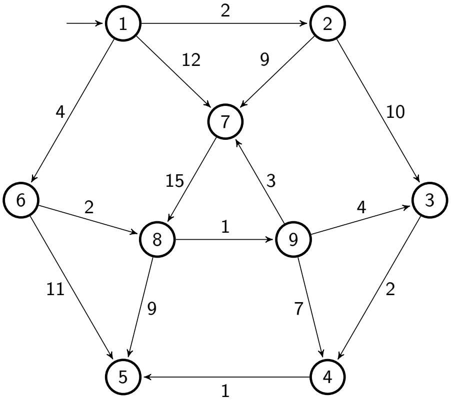
Kürzeste Wege

gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	$\{(3, 12, 2), (6, 4, 1), (7, 11, 2)\}$
$(6, 4, 1)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (8, 6, 6)\}$
$(8, 6, 6)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (9, 7, 8)\}$
$(9, 7, 8)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6), (7, 10, 9)\}$
$(7, 10, 9)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6)\}$
$(3, 11, 9)$	$\{(4, 13, 3), (5, 15, 6)\}$
$(4, 13, 3)$	$\{(5, 14, 4)\}$
$(5, 14, 4)$	



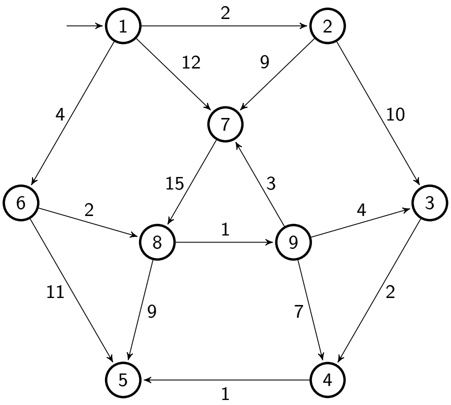
Kürzeste Wege

gewählt	Menge der Randknoten
$(1,0,-)$	$\{(2,2,1),(6,4,1),(7,12,1)\}$
$(2,2,1)$	$\{(3,12,2),(6,4,1),(7,11,2)\}$
$(6,4,1)$	$\{(3,12,2),(5,15,6),(7,11,2),(8,6,6)\}$
$(8,6,6)$	$\{(3,12,2),(5,15,6),(7,11,2),(9,7,8)\}$
$(9,7,8)$	$\{(3,11,9),(4,14,9),(5,15,6),(7,10,9)\}$
$(7,10,9)$	$\{(3,11,9),(4,14,9),(5,15,6)\}$
$(3,11,9)$	$\{(4,13,3),(5,15,6)\}$
$(4,13,3)$	$\{(5,14,4)\}$
$(5,14,4)$	$\emptyset$



Kürzeste Wege

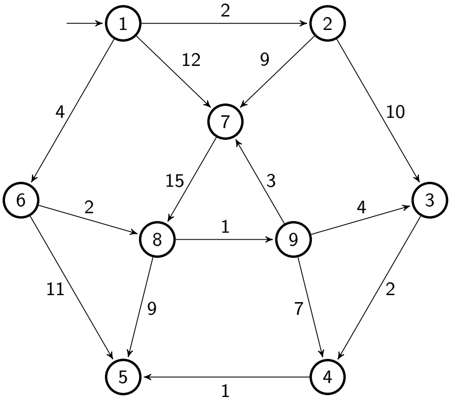
gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	$\{(3, 12, 2), (6, 4, 1), (7, 11, 2)\}$
$(6, 4, 1)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (8, 6, 6)\}$
$(8, 6, 6)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (9, 7, 8)\}$
$(9, 7, 8)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6), (7, 10, 9)\}$
$(7, 10, 9)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6)\}$
$(3, 11, 9)$	$\{(4, 13, 3), (5, 15, 6)\}$
$(4, 13, 3)$	$\{(5, 14, 4)\}$
$(5, 14, 4)$	$\emptyset$



zum Knoten v	kürzester Weg	Länge des Weges

Kürzeste Wege

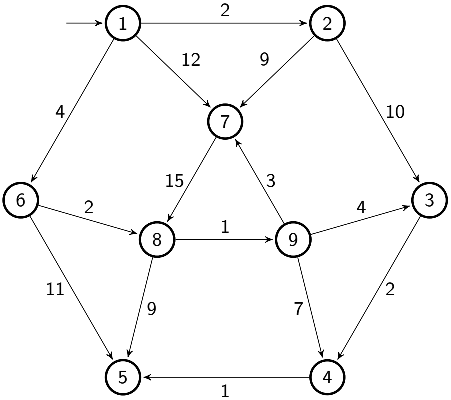
gewählt	Menge der Randknoten
$(1,0,-)$	$\{(2,2,1),(6,4,1),(7,12,1)\}$
$(2,2,1)$	$\{(3,12,2),(6,4,1),(7,11,2)\}$
$(6,4,1)$	$\{(3,12,2),(5,15,6),(7,11,2),(8,6,6)\}$
$(8,6,6)$	$\{(3,12,2),(5,15,6),(7,11,2),(9,7,8)\}$
$(9,7,8)$	$\{(3,11,9),(4,14,9),(5,15,6),(7,10,9)\}$
$(7,10,9)$	$\{(3,11,9),(4,14,9),(5,15,6)\}$
$(3,11,9)$	$\{(4,13,3),(5,15,6)\}$
$(4,13,3)$	$\{(5,14,4)\}$
$(5,14,4)$	$\emptyset$



zum Knoten v	kürzester Weg	Länge des Weges
2	$(1,2)$	2

Kürzeste Wege

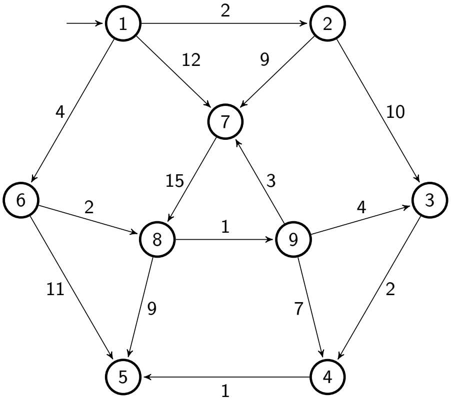
gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	$\{(3, 12, 2), (6, 4, 1), (7, 11, 2)\}$
$(6, 4, 1)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (8, 6, 6)\}$
$(8, 6, 6)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (9, 7, 8)\}$
$(9, 7, 8)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6), (7, 10, 9)\}$
$(7, 10, 9)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6)\}$
$(3, 11, 9)$	$\{(4, 13, 3), (5, 15, 6)\}$
$(4, 13, 3)$	$\{(5, 14, 4)\}$
$(5, 14, 4)$	$\emptyset$



zum Knoten v	kürzester Weg	Länge des Weges
2	$(1, 2)$	2
3	$(1, 6, 8, 9, 3)$	11

Kürzeste Wege

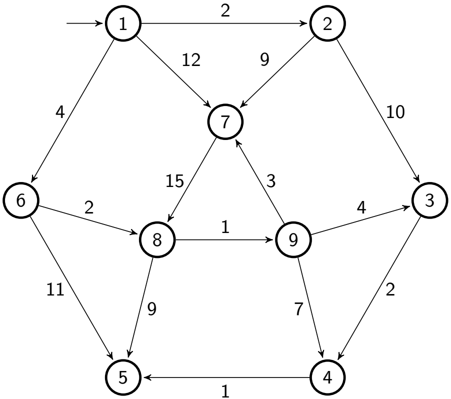
gewählt	Menge der Randknoten
$(1,0,-)$	$\{(2,2,1),(6,4,1),(7,12,1)\}$
$(2,2,1)$	$\{(3,12,2),(6,4,1),(7,11,2)\}$
$(6,4,1)$	$\{(3,12,2),(5,15,6),(7,11,2),(8,6,6)\}$
$(8,6,6)$	$\{(3,12,2),(5,15,6),(7,11,2),(9,7,8)\}$
$(9,7,8)$	$\{(3,11,9),(4,14,9),(5,15,6),(7,10,9)\}$
$(7,10,9)$	$\{(3,11,9),(4,14,9),(5,15,6)\}$
$(3,11,9)$	$\{(4,13,3),(5,15,6)\}$
$(4,13,3)$	$\{(5,14,4)\}$
$(5,14,4)$	$\emptyset$



zum Knoten v	kürzester Weg	Länge des Weges
2	$(1,2)$	2
3	$(1,6,8,9,3)$	11
4	$(1,6,8,9,3,4)$	13

Kürzeste Wege

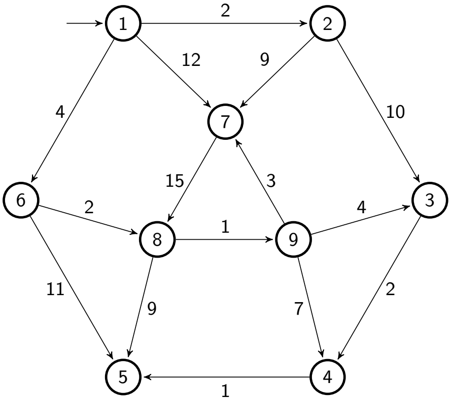
gewählt	Menge der Randknoten
$(1,0,-)$	$\{(2,2,1),(6,4,1),(7,12,1)\}$
$(2,2,1)$	$\{(3,12,2),(6,4,1),(7,11,2)\}$
$(6,4,1)$	$\{(3,12,2),(5,15,6),(7,11,2),(8,6,6)\}$
$(8,6,6)$	$\{(3,12,2),(5,15,6),(7,11,2),(9,7,8)\}$
$(9,7,8)$	$\{(3,11,9),(4,14,9),(5,15,6),(7,10,9)\}$
$(7,10,9)$	$\{(3,11,9),(4,14,9),(5,15,6)\}$
$(3,11,9)$	$\{(4,13,3),(5,15,6)\}$
$(4,13,3)$	$\{(5,14,4)\}$
$(5,14,4)$	$\emptyset$



zum Knoten v	kürzester Weg	Länge des Weges
2	$(1,2)$	2
3	$(1,6,8,9,3)$	11
4	$(1,6,8,9,3,4)$	13
5	$(1,6,8,9,3,4,5)$	14

Kürzeste Wege

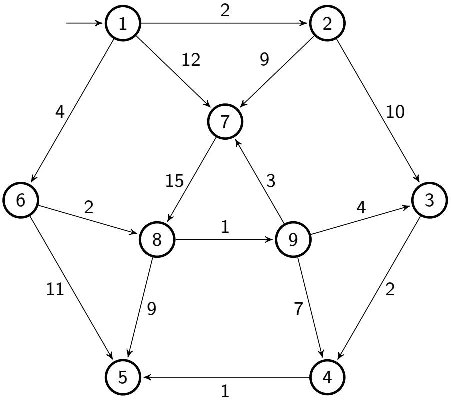
gewählt	Menge der Randknoten
$(1,0,-)$	$\{(2,2,1),(6,4,1),(7,12,1)\}$
$(2,2,1)$	$\{(3,12,2),(6,4,1),(7,11,2)\}$
$(6,4,1)$	$\{(3,12,2),(5,15,6),(7,11,2),(8,6,6)\}$
$(8,6,6)$	$\{(3,12,2),(5,15,6),(7,11,2),(9,7,8)\}$
$(9,7,8)$	$\{(3,11,9),(4,14,9),(5,15,6),(7,10,9)\}$
$(7,10,9)$	$\{(3,11,9),(4,14,9),(5,15,6)\}$
$(3,11,9)$	$\{(4,13,3),(5,15,6)\}$
$(4,13,3)$	$\{(5,14,4)\}$
$(5,14,4)$	$\emptyset$



zum Knoten v	kürzester Weg	Länge des Weges
2	$(1,2)$	2
3	$(1,6,8,9,3)$	11
4	$(1,6,8,9,3,4)$	13
5	$(1,6,8,9,3,4,5)$	14
6	$(1,6)$	4

Kürzeste Wege

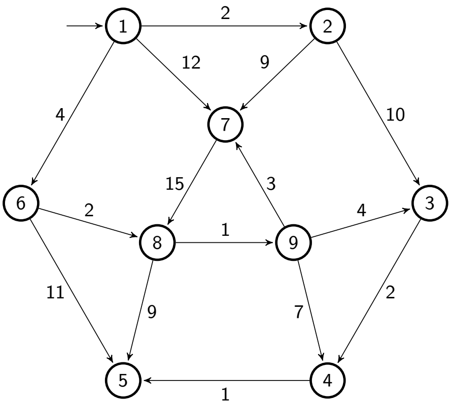
gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	$\{(3, 12, 2), (6, 4, 1), (7, 11, 2)\}$
$(6, 4, 1)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (8, 6, 6)\}$
$(8, 6, 6)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (9, 7, 8)\}$
$(9, 7, 8)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6), (7, 10, 9)\}$
$(7, 10, 9)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6)\}$
$(3, 11, 9)$	$\{(4, 13, 3), (5, 15, 6)\}$
$(4, 13, 3)$	$\{(5, 14, 4)\}$
$(5, 14, 4)$	$\emptyset$



zum Knoten v	kürzester Weg	Länge des Weges
2	$(1, 2)$	2
3	$(1, 6, 8, 9, 3)$	11
4	$(1, 6, 8, 9, 3, 4)$	13
5	$(1, 6, 8, 9, 3, 4, 5)$	14
6	$(1, 6)$	4
7	$(1, 6, 8, 9, 7)$	10

Kürzeste Wege

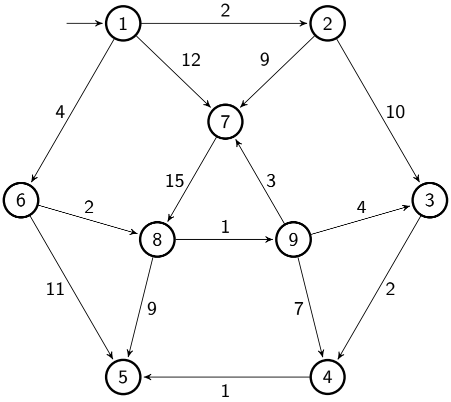
gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2, 2, 1), (6, 4, 1), (7, 12, 1)\}$
$(2, 2, 1)$	$\{(3, 12, 2), (6, 4, 1), (7, 11, 2)\}$
$(6, 4, 1)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (8, 6, 6)\}$
$(8, 6, 6)$	$\{(3, 12, 2), (5, 15, 6), (7, 11, 2), (9, 7, 8)\}$
$(9, 7, 8)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6), (7, 10, 9)\}$
$(7, 10, 9)$	$\{(3, 11, 9), (4, 14, 9), (5, 15, 6)\}$
$(3, 11, 9)$	$\{(4, 13, 3), (5, 15, 6)\}$
$(4, 13, 3)$	$\{(5, 14, 4)\}$
$(5, 14, 4)$	$\emptyset$



zum Knoten v	kürzester Weg	Länge des Weges
2	$(1, 2)$	2
3	$(1, 6, 8, 9, 3)$	11
4	$(1, 6, 8, 9, 3, 4)$	13
5	$(1, 6, 8, 9, 3, 4, 5)$	14
6	$(1, 6)$	4
7	$(1, 6, 8, 9, 7)$	10
8	$(1, 6, 8)$	6

Kürzeste Wege

gewählt	Menge der Randknoten
$(1, 0, -)$	$\{(2,2,1),(6,4,1),(7,12,1)\}$
$(2, 2, 1)$	$\{(3,12,2),(6,4,1),(7,11,2)\}$
$(6, 4, 1)$	$\{(3,12,2),(5,15,6),(7,11,2),(8,6,6)\}$
$(8, 6, 6)$	$\{(3,12,2),(5,15,6),(7,11,2),(9,7,8)\}$
$(9, 7, 8)$	$\{(3,11,9),(4,14,9),(5,15,6),(7,10,9)\}$
$(7, 10, 9)$	$\{(3,11,9),(4,14,9),(5,15,6)\}$
$(3, 11, 9)$	$\{(4,13,3),(5,15,6)\}$
$(4, 13, 3)$	$\{(5,14,4)\}$
$(5, 14, 4)$	$\emptyset$



zum Knoten v	kürzester Weg	Länge des Weges
2	$(1,2)$	2
3	$(1,6,8,9,3)$	11
4	$(1,6,8,9,3,4)$	13
5	$(1,6,8,9,3,4,5)$	14
6	$(1,6)$	4
7	$(1,6,8,9,7)$	10
8	$(1,6,8)$	6
9	$(1,6,8,9)$	7

Algorithmus Dijkstra-Algorithmus

Eingabe: gerichteter Distanzgraph $G = (V, E, c)$ und ein Knoten $s \in V$

Ausgabe: für jeden Knoten $v \in V$ ist der kürzeste Weg von s nach v der Weg: $(s, ..., p(p(v)), p(v), v)$

Verfahren:

```
Set R;                 // Menge der Randknoten
Node u, v;             // Knoten aus V
PredVector p;          // ordnet jedem v in V einen Vorgängerknoten zu
LengthVector d;        // ordnet jedem v in V einen Abstand (aus ℕ oder ∞) zur Quelle zu

for (alle v in V)      // Initialisierung
{ d(v) = ∞;    p(v) = undefiniert; }
d(s) = 0;
p(s) = s;
U = V;
R = {s};

while (R nicht leer)
{ wähle u in R, so dass d(u) = min{ d(v) | v in U }
  entferne u aus U und aus R;

  for (jedes v in U mit (u,v) in E)
    if (d(u) + c(u,v) < d(v))
    { d(v) = d(u) + c(u,v);
      p(v) = u;
      füge v zu R hinzu; }
}
```

Das algebraische Pfadproblem

D_G: *Für beliebige $u, v \in V$: Wie lang ist der kürzeste Weg p von u nach v?*

Das algebraische Pfadproblem

D_G: *Für beliebige $u, v \in V$: Wie lang ist der kürzeste Weg p von u nach v?*

Kürzeste-Wege-Matrix

$$D_G(u,v) = \begin{cases} \min\{c(p) \mid p \in P_{u,v}\} & \text{wenn } P_{u,v} \neq \emptyset \\ \infty & \text{sonst;} \end{cases}$$

für jeden Weg $p = (v_0, ..., v_r)$ (mit $r \geq 0$ und $v_0, ..., v_r \in V$) ist dessen Länge gleich

$$c(p) = \sum_{l=0}^{r-1} c(v_l, v_{l+1});$$

Das algebraische Pfadproblem

$D_G^{(k)}$: *Für beliebige* $u, v \in V$*: Wie lang ist der kürzeste Weg* p *von* u *nach* v*, so dass* $p \in P_{u,v}^{(k)}$*?*

Dabei ist $P_{u,v}^{(k)}$ die Menge aller der Wege in $P_{u,v}$, deren innere Knoten in der Menge $\{l \mid 1 \leq l \leq k\}$ liegen.

Das algebraische Pfadproblem

$D_G^{(k)}$: *Für beliebige $u, v \in V$: Wie lang ist der kürzeste Weg p von u nach v, so dass $p \in P_{u,v}^{(k)}$?*

Dabei ist $P_{u,v}^{(k)}$ die Menge aller der Wege in $P_{u,v}$, deren innere Knoten in der Menge $\{l \mid 1 \leq l \leq k\}$ liegen.

$$D_G^{(k)}(u,v) = \begin{cases} \min\{c(p) \mid p \in P_{u,v}^{(k)}\}, & \text{falls } P_{u,v}^{(k)} \neq \emptyset \\ \infty, & \text{sonst.} \end{cases}$$

Das algebraische Pfadproblem

$$D_G^{(0)}(u,v) = \begin{cases} c(u,v) & \text{wenn } u \neq v \text{ und } (u,v) \in E \\ 0 & u = v \\ \infty, & \text{sonst.} \end{cases}$$

$$D_G^{(0)}(u,v) = \begin{cases} A_G(u,v) & \text{wenn } u \neq v \\ 0 & u = v, \end{cases}$$

Das algebraische Pfadproblem

$$D_G^{(0)}(u,v) = \begin{cases} c(u,v) & \text{wenn } u \neq v \text{ und } (u,v) \in E \\ 0 & u = v \\ \infty, & \text{sonst.} \end{cases}$$

$$D_G^{(0)}(u,v) = \begin{cases} A_G(u,v) & \text{wenn } u \neq v \\ 0 & u = v, \end{cases}$$

$$D_G^{(k+1)}(u,v) = \min\{D_G^{(k)}(u,v)\ ,\ D_G^{(k)}(u,k+1) + D_G^{(k)}(k+1,v)\}.$$

Algorithmus Floyd-Warshall-Algorithmus

Eingabe: Distanzgraph $G = (V, E, c)$ mit $V = \{1, \dots, n\}$ und $c\colon E \to \mathbb{R}_{\geq 0}$

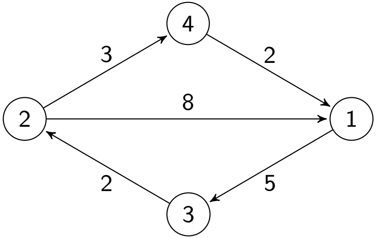
Ausgabe: $n \times n$-Matrix D_G über $\mathbb{R}^{\infty}_{\geq 0}$

Verfahren:

begin
    $D_G^{(0)} := mA_G$;
    seien $D_G^{(1)}, \dots, D_G^{(n)}$ $n \times n$-Matrizen
    for $k := 1$ **to** n **do**
        for $u, v \in \{1, \dots, n\}$ **do**
            $D_G^{(k)}(u,v) := \min\left\{D_G^{(k-1)}(u,v),\ \ D_G^{(k-1)}(u,k) + D_G^{(k-1)}(k,v)\right\}$;
    $D_G := D_G^{(n)}$
end

Floyd-Warshall-Algorithmus

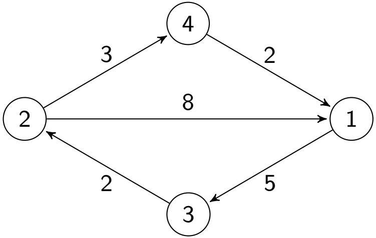
Gegeben sei der folgende Distanzgraph G:



Floyd-Warshall-Algorithmus

Gegeben sei der folgende Distanzgraph G:

$$mA_G = \begin{pmatrix} 0 & \infty & 5 & \infty \\ 8 & 0 & \infty & 3 \\ \infty & 2 & 0 & \infty \\ 2 & \infty & \infty & 0 \end{pmatrix}$$

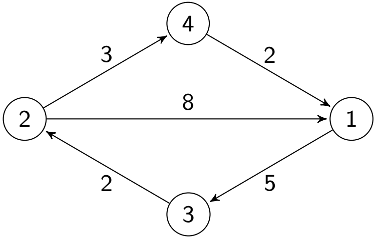


Floyd-Warshall-Algorithmus

Gegeben sei der folgende Distanzgraph G:

$$mA_G = \begin{pmatrix} 0 & \infty & 5 & \infty \\ 8 & 0 & \infty & 3 \\ \infty & 2 & 0 & \infty \\ 2 & \infty & \infty & 0 \end{pmatrix}$$

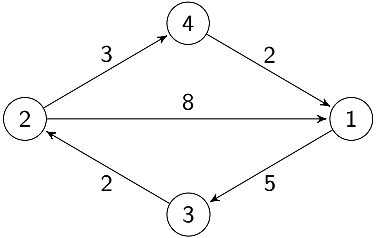
$$D_G^{(1)} = \begin{pmatrix} 0 & \infty & 5 & \infty \\ 8 & 0 & \mathit{13} & 3 \\ \infty & 2 & 0 & \infty \\ 2 & \infty & \mathit{7} & 0 \end{pmatrix}$$



Floyd-Warshall-Algorithmus

Gegeben sei der folgende Distanzgraph G:

$$mA_G = \begin{pmatrix} 0 & \infty & 5 & \infty \\ 8 & 0 & \infty & 3 \\ \infty & 2 & 0 & \infty \\ 2 & \infty & \infty & 0 \end{pmatrix}$$



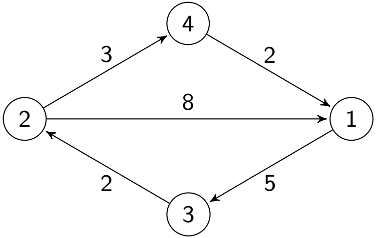
$$D_G^{(1)} = \begin{pmatrix} 0 & \infty & 5 & \infty \\ 8 & 0 & \mathit{13} & 3 \\ \infty & 2 & 0 & \infty \\ 2 & \infty & \mathit{7} & 0 \end{pmatrix}$$

$$D_G^{(2)} = \begin{pmatrix} 0 & \infty & 5 & \infty \\ 8 & 0 & 13 & 3 \\ \mathit{10} & 2 & 0 & \mathit{5} \\ 2 & \infty & 7 & 0 \end{pmatrix}$$

Floyd-Warshall-Algorithmus

Gegeben sei der folgende Distanzgraph G:

$$mA_G = \begin{pmatrix} 0 & \infty & 5 & \infty \\ 8 & 0 & \infty & 3 \\ \infty & 2 & 0 & \infty \\ 2 & \infty & \infty & 0 \end{pmatrix}$$



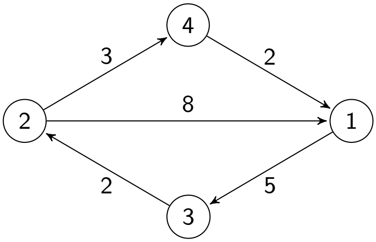
$$D_G^{(1)} = \begin{pmatrix} 0 & \infty & 5 & \infty \\ 8 & 0 & \mathit{13} & 3 \\ \infty & 2 & 0 & \infty \\ 2 & \infty & \mathit{7} & 0 \end{pmatrix}$$

$$D_G^{(2)} = \begin{pmatrix} 0 & \infty & 5 & \infty \\ 8 & 0 & 13 & 3 \\ \mathit{10} & 2 & 0 & \mathit{5} \\ 2 & \infty & 7 & 0 \end{pmatrix} \qquad D_G^{(3)} = \begin{pmatrix} 0 & \mathit{7} & 5 & \mathit{10} \\ 8 & 0 & 13 & 3 \\ 10 & 2 & 0 & 5 \\ 2 & \mathit{9} & 7 & 0 \end{pmatrix}$$

Floyd-Warshall-Algorithmus

Gegeben sei der folgende Distanzgraph G:

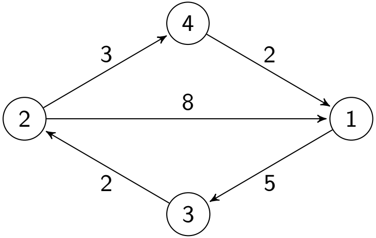
$$mA_G = \begin{pmatrix} 0 & \infty & 5 & \infty \\ 8 & 0 & \infty & 3 \\ \infty & 2 & 0 & \infty \\ 2 & \infty & \infty & 0 \end{pmatrix}$$



$$D_G^{(1)} = \begin{pmatrix} 0 & \infty & 5 & \infty \\ 8 & 0 & \mathit{13} & 3 \\ \infty & 2 & 0 & \infty \\ 2 & \infty & \mathit{7} & 0 \end{pmatrix}$$

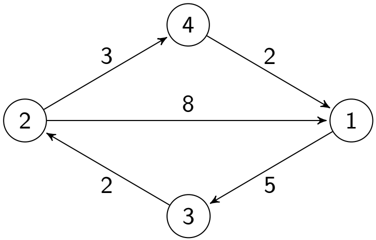
$$D_G^{(2)} = \begin{pmatrix} 0 & \infty & 5 & \infty \\ 8 & 0 & 13 & 3 \\ \mathit{10} & 2 & 0 & \mathit{5} \\ 2 & \infty & 7 & 0 \end{pmatrix} \qquad D_G^{(3)} = \begin{pmatrix} 0 & \mathit{7} & 5 & \mathit{10} \\ 8 & 0 & 13 & 3 \\ 10 & 2 & 0 & 5 \\ 2 & \mathit{9} & 7 & 0 \end{pmatrix} \qquad D_G^{(4)} = \begin{pmatrix} 0 & 7 & 5 & 10 \\ \mathit{5} & 0 & \mathit{10} & 3 \\ \mathit{7} & 2 & 0 & 5 \\ 2 & 9 & 7 & 0 \end{pmatrix}$$

Kürzestes Wegeproblem



Wie lang ist der kürzeste Weg von 2 nach 3?

Kürzestes Wegeproblem

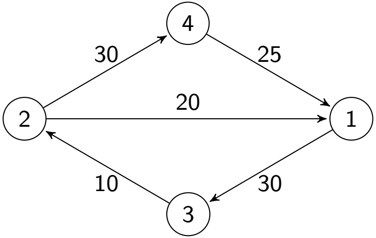


Wie lang ist der kürzeste Weg von 2 nach 3?

Weg p_1: $3 + 2 + 5 = 10$
Weg p_2: $8 + 5 = 13$

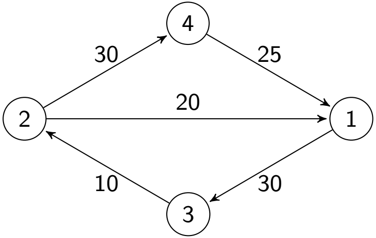
also: $\min\{10, 13\} = 10$

Kapazitätsproblem



Mit welcher maximalen Tonnage kann man von 2 nach 3 fahren?

Kapazitätsproblem

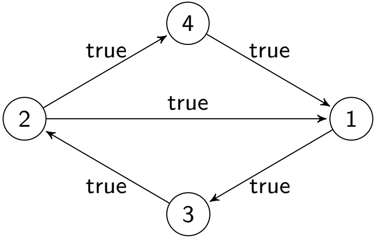


Mit welcher maximalen Tonnage kann man von 2 nach 3 fahren?

Weg p_1: $\min\{30, 25, 30\} = 25$
Weg p_2: $\min\{20, 30\} = 20$

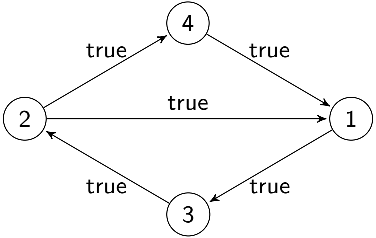
also: $\max\{25, 20\} = 25$

Erreichbarkeitsproblem



Gibt es eine Verbindung von 2 nach 3?

Erreichbarkeitsproblem

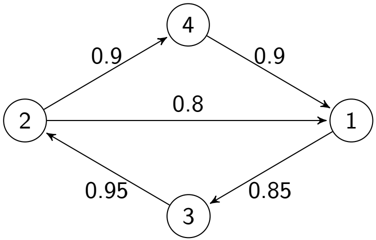


Gibt es eine Verbindung von 2 nach 3?

Weg p_1: $\text{true} \wedge \text{true} \wedge \text{true} = \text{true}$
Weg p_2: $\text{true} \wedge \text{true} = \text{true}$

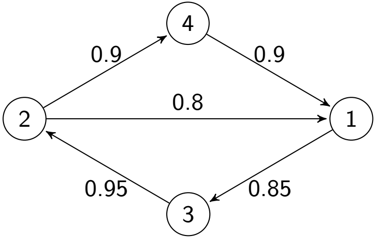
also: $\text{true} \vee \text{true} = \text{true}$

Zuverlässigkeitsproblem



Wie zuverlässig kann die Information von Station 2 zu Station 3 übertragen werden?

Zuverlässigkeitsproblem

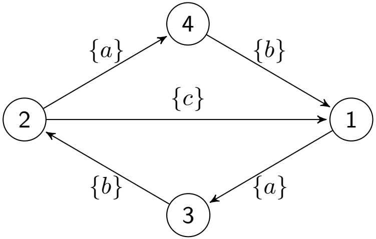


Wie zuverlässig kann die Information von Station 2 zu Station 3 übertragen werden?

Weg p_1: $0.9 \cdot 0.9 \cdot 0.85 = 0.6885$
Weg p_2: $0.8 \cdot 0.85 = 0.68$

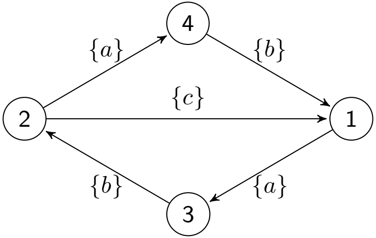
also: $\max\{0.6885, 0.68\} = 0.6885$

Prozessproblem



Wie lautet die Menge aller Prozesse, die die Anlage vom Zustand 2 in den Zustand 3 überführen?

Prozessproblem



Wie lautet die Menge aller Prozesse, die die Anlage vom Zustand 2 in den Zustand 3 überführen?

Weg p_1: $\{a\} \circ \{b\} \circ \{a\} = \{aba\}$
Weg p_2: $\{c\} \circ \{a\} = \{ca\}$

also: $\{aba\} \cup \{ca\} = \{aba, ca\}$

Das algebraische Pfadproblem

	Menge S der Werte	$\oplus$	$\odot$	**0**	**1**
kürzestes Wegeproblem:	$\mathbb{R}_{\geq 0}^{\infty}$	$\min$	$+$	∞	0
Kapazitätsproblem:	$\mathbb{N}_{\infty}$	$\max$	$\min$	0	∞
Erreichbarkeitsproblem:	{true,false}	$\vee$	$\wedge$	false	true
Zuverlässigkeitsproblem:	$[0,1]$	$\max$	$\cdot$	0	1
Prozessproblem:	$P(\Sigma^*)$	$\cup$	$\circ$	$\emptyset$	$\{\varepsilon\}$

Ein *Semiring* ist eine algebraische Struktur $(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$ wobei

- $\oplus$ eine binäre, assoziative und kommutative Operation über S ist (Addition),
- $\odot$ eine binäre, assoziative Operation über S ist (Multiplikation),
- $\mathbf{0}$ ist neutrales Element bzgl. $\oplus$, d. h. $s \oplus \mathbf{0} = s$ für jedes $s \in S$,
- $\mathbf{1}$ ist neutrales Element bzgl. $\odot$, d. h. $s \odot \mathbf{1} = \mathbf{1} \odot s = s$ für jedes $s \in S$,
- $\odot$ ist *distributiv* über $\oplus$, d. h., $s \odot (t \oplus r) = (s \odot t) \oplus (s \odot r)$ und $(s \oplus t) \odot r = (s \odot r) \oplus (t \odot r)$ für jedes $s, t, r \in S$ und
- $\mathbf{0}$ ist ein *Annihilator* für $\odot$, d. h. $\mathbf{0} \odot s = s \odot \mathbf{0} = \mathbf{0}$ für jedes $s \in S$.

Ein *Semiring* ist eine algebraische Struktur $(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$ wobei

- $\oplus$ eine binäre, assoziative und kommutative Operation über S ist (Addition),
- $\odot$ eine binäre, assoziative Operation über S ist (Multiplikation),
- $\mathbf{0}$ ist neutrales Element bzgl. $\oplus$, d. h. $s \oplus \mathbf{0} = s$ für jedes $s \in S$,
- $\mathbf{1}$ ist neutrales Element bzgl. $\odot$, d. h. $s \odot \mathbf{1} = \mathbf{1} \odot s = s$ für jedes $s \in S$,
- $\odot$ ist *distributiv* über $\oplus$, d. h., $s \odot (t \oplus r) = (s \odot t) \oplus (s \odot r)$ und $(s \oplus t) \odot r = (s \odot r) \oplus (t \odot r)$ für jedes $s, t, r \in S$ und
- $\mathbf{0}$ ist ein *Annihilator* für $\odot$, d. h. $\mathbf{0} \odot s = s \odot \mathbf{0} = \mathbf{0}$ für jedes $s \in S$.

	$(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$	Semiring
kürzestes Wegeproblem:	$(\mathbb{R}_{\geq 0}^{\infty}, \min, +, \infty, 0)$	tropischer Semiring
Kapazitätsproblem:	$(\mathbb{N}_{\infty}, \max, \min, 0, \infty)$	
Erreichbarkeitsproblem:	$(\{\text{true,false}\}, \vee, \wedge, \text{false,true})$	Boolescher Semiring
Zuverlässigkeitsproblem:	$([0,1], \max, \cdot, 0, 1)$	Viterbi-Semiring
Prozessproblem:	$(P(\Sigma^*), \cup, \circ, \emptyset, \{\varepsilon\})$	Semiring der formalen Σ-Sprachen
	$(\mathbb{R}_{\geq 0}^{-\infty}, \max, +, -\infty, 0)$	arktischer Semiring
	$(\mathbb{N}, +, \cdot, 0, 1)$	Semiring der natürlichen Zahlen

Ein *Semiring* ist eine algebraische Struktur $(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$ wobei

- $\oplus$ eine binäre, assoziative und kommutative Operation über S ist (Addition),
- $\odot$ eine binäre, assoziative Operation über S ist (Multiplikation),
- $\mathbf{0}$ ist neutrales Element bzgl. $\oplus$, d. h. $s \oplus \mathbf{0} = s$ für jedes $s \in S$,
- $\mathbf{1}$ ist neutrales Element bzgl. $\odot$, d. h. $s \odot \mathbf{1} = \mathbf{1} \odot s = s$ für jedes $s \in S$,
- $\odot$ ist *distributiv* über $\oplus$, d. h., $s \odot (t \oplus r) = (s \odot t) \oplus (s \odot r)$ und $(s \oplus t) \odot r = (s \odot r) \oplus (t \odot r)$ für jedes $s, t, r \in S$ und
- $\mathbf{0}$ ist ein *Annihilator* für $\odot$, d. h. $\mathbf{0} \odot s = s \odot \mathbf{0} = \mathbf{0}$ für jedes $s \in S$.

	$(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$	Semiring
kürzestes Wegeproblem:	$(\mathbb{R}_{\geq 0}^{\infty}, \min, +, \infty, 0)$	tropischer Semiring
Kapazitätsproblem:	$(\mathbb{N}_{\infty}, \max, \min, 0, \infty)$	
Erreichbarkeitsproblem:	$(\{\text{true,false}\}, \vee, \wedge, \text{false,true})$	Boolescher Semiring
Zuverlässigkeitsproblem:	$([0,1], \max, \cdot, 0, 1)$	Viterbi-Semiring
Prozessproblem:	$(P(\Sigma^*), \cup, \circ, \emptyset, \{\varepsilon\})$	Semiring der formalen Σ-Sprachen
	$(\mathbb{R}_{\geq 0}^{-\infty}, \max, +, -\infty, 0)$	arktischer Semiring
	$(\mathbb{N}, +, \cdot, 0, 1)$	Semiring der natürlichen Zahlen

Ein Semiring $(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$ ist *idempotent* wenn $s \oplus s = s$ für jedes $s \in S$.

kürzestes Wegeproblem:	$\sum_{i\in I}^{\min} s_i = \inf\{s_i \mid i \in I\}$
Kapazitätsproblem:	$\sum_{i\in I}^{\max} s_i = \sup\{s_i \mid i \in I\}$
Erreichbarkeitsproblem:	$\sum_{i\in I}^{\vee} s_i =$ false wenn alle $s_i =$ false, sonst true
Zuverlässigkeitsproblem:	$\sum_{i\in I}^{\max} s_i = \sup\{s_i \mid i \in I\}$
Prozessproblem:	$\sum_{i\in I}^{\cup} s_i = \bigcup_{i\in I} s_i$

kürzestes Wegeproblem:	$\sum_{i \in I}^{\min} s_i = \inf\{s_i \mid i \in I\}$
Kapazitätsproblem:	$\sum_{i \in I}^{\max} s_i = \sup\{s_i \mid i \in I\}$
Erreichbarkeitsproblem:	$\sum_{i \in I}^{\vee} s_i =$ false wenn alle $s_i =$ false, sonst true
Zuverlässigkeitsproblem:	$\sum_{i \in I}^{\max} s_i = \sup\{s_i \mid i \in I\}$
Prozessproblem:	$\sum_{i \in I}^{\cup} s_i = \bigcup_{i \in I} s_i$

Sei $(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$ ein Semiring und $\sum^{\oplus}$ eine Abbildung, die jeder Familie $(s_i \mid i \in I)$ ein Element in S zuordnet. Dann heißt der Semiring $\sum^{\oplus}$-*vollständig*, wenn

- $\sum^{\oplus}$ eine Fortsetzung von $\oplus$ ist, d.h. $\sum_{i \in \emptyset}^{\oplus} s_i = \mathbf{0}$, $\sum_{i \in \{j\}}^{\oplus} s_i = s_j$, $\sum_{i \in \{j,k\}}^{\oplus} s_i = s_j \oplus s_k$,
- $\sum^{\oplus}$ ist assoziativ und kommutativ, d.h. wenn die Indexmenge I partitioniert werden kann durch $(I_j \mid j \in J)$, also $I = \bigcup_{j \in J} I_j$ und $I_l \cap I_k = \emptyset$ für $l \neq k$, dann gilt $\sum_{j \in J}^{\oplus}(\sum_{i \in I_j} s_i) = \sum_{i \in I}^{\oplus} s_i$ und
- $\odot$ ist distributiv über $\sum^{\oplus}$, d. h., $\sum_{i \in I}^{\oplus}(a \odot s_i) = a \odot (\sum_{i \in I}^{\oplus} s_i)$ und $\sum_{i \in I}(s_i \odot a) = (\sum_{i \in I}^{\oplus} s_i) \odot a$ für jedes $a \in S$.

Das algebraische Pfadproblem

Ein *gewichteter Graph über einem Semiring* $(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$ ist ein Tupel $G = (V, E, c)$ mit $c : E \to S \setminus \{\mathbf{0}\}$. Die Adjazenzmatrix von G ist die $n \times n$-Matrix A_G über S mit

$$A_G(u, v) = \begin{cases} c(u, v) & \text{wenn } (u, v) \in E \\ \mathbf{0} & \text{sonst.} \end{cases}$$

Das algebraische Pfadproblem

Ein *gewichteter Graph über einem Semiring* $(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$ ist ein Tupel $G = (V, E, c)$ mit $c : E \to S \setminus \{\mathbf{0}\}$. Die Adjazenzmatrix von G ist die $n \times n$-Matrix A_G über S mit

$$A_G(u, v) = \begin{cases} c(u, v) & \text{wenn } (u, v) \in E \\ \mathbf{0} & \text{sonst.} \end{cases}$$

Algebraisches Pfadproblem Sei jetzt $G = (V, E, c)$ ein gewichteter Graph über einem $\sum^{\oplus}$-vollständigen, idempotenten Semiring $S = (S, \oplus, \odot, \mathbf{0}, \mathbf{1})$.

$$D_G(u, v) = \sum\nolimits^{\oplus}_{p \in P_{u,v}} c(p),$$

wobei für jeden Weg $p = (v_0, ..., v_r)$ mit $r \geq 0$ gilt $c(p) = c(v_0, v_1) \odot c(v_1, v_2) \odot ... \odot c(v_{r-1}, v_r)$

Das algebraische Pfadproblem

Stern von $s \in S$. Wir definieren $s^* = \sum_{n \in \mathbb{N}}^{\oplus} s^n$, wobei $s^0 = \mathbf{1}$ und $s^{n+1} = s \odot s^n$.

Das algebraische Pfadproblem

Stern von $s \in S$. Wir definieren $s^* = \sum_{n\in\mathbb{N}}^{\oplus} s^n$, wobei $s^0 = \mathbf{1}$ und $s^{n+1} = s \odot s^n$.

$(\mathbb{R}^{\infty}_{\geq 0}, \min, +, \infty, 0)$	$r^* = \sum^{\min}\{0,\ r,\ r+r,\ r+r+r, ...\} = 0$
$(\mathbb{N}_{\infty}, \max, \min, 0, \infty)$	$k^* = \sum^{\max}\{\infty,\ k,\ \min\{k,k\},\ \min\{k,k,k\}, ...,\} = \infty$
$(\{\text{true}, \text{false}\}, \vee, \wedge, \text{false}, \text{true})$	$b^* = \sum^{\vee}\{\text{true}, b, b \wedge b, b \wedge b \wedge b, ...\} = \text{true}$
$([0,1], \max, \cdot, 0, 1)$	$s^* = \sum^{max}\{1,\ s,\ s \cdot s,\ s \cdot s \cdot s, ...\} = 1$
$(P(\Sigma^*), \cup, \circ, \emptyset, \{\varepsilon\})$	$L^* = \sum^{\cup}\{\{\varepsilon\}, L, L \circ L, L \circ L \circ L, ...\} = L^*$ (der Stern von L wie bereits definiert)

$$mA_G(u,v) = \begin{cases} A_G(u,v) & \text{wenn } u \neq v \\ A_G(u,v) \oplus \mathbf{1} & \text{sonst.} \end{cases}$$

$$mA_G(u,v) = \begin{cases} A_G(u,v) & \text{wenn } u \neq v \\ A_G(u,v) \oplus \mathbf{1} & \text{sonst.} \end{cases}$$

Algorithmus Aho-Hopcroft-Ullman-Algorithmus

Eingabe: gewichteter Graph $G = (V, E, c)$ mit $V = \{1, \dots, n\}$ über einem $\sum^{\oplus}$-vollst., idempotenten Semiring S

Ausgabe: $n \times n$-Matrix D_G über S

Verfahren:

```
begin
    D_G^(0) := mA_G;
    seien D_G^(1), ..., D_G^(n) n × n-Matrizen
    for k := 1 to n do
        for u, v ∈ {1, ..., n} do
            D_G^(k)(u,v) := D_G^(k-1)(u,v) ⊕ ( D_G^(k-1)(u,k) ⊙ (D_G^(k-1)(k,k))* ⊙ D_G^(k-1)(k,v) );
    D_G := D_G^(n)
end
```

$D_G^{(0)} := mA_G$; $D_G^{(k)}(u,v) := D_G^{(k-1)}(u,v) \oplus \left(D_G^{(k-1)}(u,k) \odot (D_G^{(k-1)}(k,k))^* \odot D_G^{(k-1)}(k,v)\right)$; $D_G := D_G^{(n)}$

$$mA_G(u,v) = \begin{cases} A_G(u,v) & \text{wenn } u \neq v \\ A_G(u,v) \oplus \mathbf{1} & \text{sonst.} \end{cases}$$

Algorithmus Aho-Hopcroft-Ullman-Algorithmus

Eingabe: gewichteter Graph $G = (V, E, c)$ mit $V = \{1, \dots, n\}$
über einem $\sum^{\oplus}$-vollst., idempotenten Semiring S

Ausgabe: $n \times n$-Matrix D_G über S

Verfahren:

```
begin
    D_G^(0) := mA_G;
    seien D_G^(1), ..., D_G^(n) n × n-Matritzen
    for k := 1 to n do
        for u, v ∈ {1, ..., n} do
            D_G^(k)(u,v) := D_G^(k-1)(u,v) ⊕ (D_G^(k-1)(u,k) ⊙ (D_G^(k-1)(k,k))* ⊙ D_G^(k-1)(k,v));
    D_G := D_G^(n)
end
```

$$D_G^{(k-1)}(u,v) \oplus \left(D_G^{(k-1)}(u,k) \odot (D_G^{(k-1)}(k,k))^* \odot D_G^{(k-1)}(k,v) \right)$$

$$mA_G(u,v) = \begin{cases} A_G(u,v) & \text{wenn } u \neq v \\ A_G(u,v) \oplus \mathbf{1} & \text{sonst.} \end{cases}$$

Algorithmus Aho-Hopcroft-Ullman-Algorithmus

Eingabe: gewichteter Graph $G = (V, E, c)$ mit $V = \{1, \dots, n\}$
über einem $\sum^{\oplus}$-vollst., idempotenten Semiring S

Ausgabe: $n \times n$-Matrix D_G über S

Verfahren:

```
begin
    D_G^(0) := mA_G;
    seien D_G^(1), ..., D_G^(n) n × n-Matritzen
    for k := 1 to n do
        for u, v ∈ {1, ..., n} do
            D_G^(k)(u,v) := D_G^(k-1)(u,v) ⊕ (D_G^(k-1)(u,k) ⊙ (D_G^(k-1)(k,k))* ⊙ D_G^(k-1)(k,v));
    D_G := D_G^(n)
end
```

$$D_G^{(k-1)}(u,v) \oplus \left(D_G^{(k-1)}(u,k) \odot (D_G^{(k-1)}(k,k))^* \odot D_G^{(k-1)}(k,v)\right)$$

$$= \min\left\{D_G^{(k-1)}(u,v),\ \ D_G^{(k-1)}(u,k) + (D_G^{(k-1)}(k,k))^* + D_G^{(k-1)}(k,v)\right\}$$

$$mA_G(u,v) = \begin{cases} A_G(u,v) & \text{wenn } u \neq v \\ A_G(u,v) \oplus \mathbf{1} & \text{sonst.} \end{cases}$$

Algorithmus Aho-Hopcroft-Ullman-Algorithmus

Eingabe: gewichteter Graph $G = (V, E, c)$ mit $V = \{1, \ldots, n\}$
über einem $\sum^{\oplus}$-vollst., idempotenten Semiring S

Ausgabe: $n \times n$-Matrix D_G über S

Verfahren:

```
begin
    D_G^(0) := mA_G;
    seien D_G^(1), ..., D_G^(n) n × n-Matritzen
    for k := 1 to n do
        for u, v ∈ {1, ..., n} do
            D_G^(k)(u,v) := D_G^(k-1)(u,v) ⊕ (D_G^(k-1)(u,k) ⊙ (D_G^(k-1)(k,k))* ⊙ D_G^(k-1)(k,v));
    D_G := D_G^(n)
end
```

$$\begin{aligned} & D_G^{(k-1)}(u,v) \oplus \left(D_G^{(k-1)}(u,k) \odot (D_G^{(k-1)}(k,k))^* \odot D_G^{(k-1)}(k,v)\right) \\ &= \min\left\{D_G^{(k-1)}(u,v),\ \ D_G^{(k-1)}(u,k) + (D_G^{(k-1)}(k,k))^* + D_G^{(k-1)}(k,v)\right\} \\ &= \min\left\{D_G^{(k-1)}(u,v),\ \ D_G^{(k-1)}(u,k) + 0 + D_G^{(k-1)}(k,v)\right\} \end{aligned}$$

$$mA_G(u,v) = \begin{cases} A_G(u,v) & \text{wenn } u \neq v \\ A_G(u,v) \oplus \mathbf{1} & \text{sonst.} \end{cases}$$

Algorithmus Aho-Hopcroft-Ullman-Algorithmus

Eingabe: gewichteter Graph $G = (V, E, c)$ mit $V = \{1, \ldots, n\}$
über einem $\sum^{\oplus}$-vollst., idempotenten Semiring S

Ausgabe: $n \times n$-Matrix D_G über S

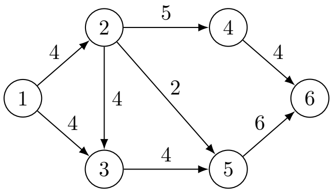
Verfahren:

```
begin
    D_G^(0) := mA_G;
    seien D_G^(1), ..., D_G^(n) n × n-Matritzen
    for k := 1 to n do
        for u, v ∈ {1, ..., n} do
            D_G^(k)(u,v) := D_G^(k-1)(u,v) ⊕ (D_G^(k-1)(u,k) ⊙ (D_G^(k-1)(k,k))* ⊙ D_G^(k-1)(k,v));
    D_G := D_G^(n)
end
```

$$\begin{aligned}
& D_G^{(k-1)}(u,v) \oplus \left(D_G^{(k-1)}(u,k) \odot (D_G^{(k-1)}(k,k))^* \odot D_G^{(k-1)}(k,v)\right) \\
&= \min\left\{D_G^{(k-1)}(u,v),\ \ D_G^{(k-1)}(u,k) + (D_G^{(k-1)}(k,k))^* + D_G^{(k-1)}(k,v)\right\} \\
&= \min\left\{D_G^{(k-1)}(u,v),\ \ D_G^{(k-1)}(u,k) + 0 + D_G^{(k-1)}(k,v)\right\} \\
&= \min\left\{D_G^{(k-1)}(u,v),\ \ D_G^{(k-1)}(u,k) + D_G^{(k-1)}(k,v)\right\}
\end{aligned}$$

Beispiel

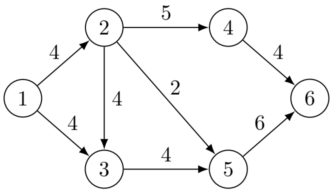
Kapazitätsproblem $(\mathbb{N}_\infty, \max, \min, 0, \infty)$



Beispiel

Kapazitätsproblem $(\mathbb{N}_\infty, \max, \min, 0, \infty)$

$$D_G^{(0)} = \begin{pmatrix} \infty & 4 & 4 & 0 & 0 & 0 \\ 0 & \infty & 4 & 5 & 2 & 0 \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix}$$

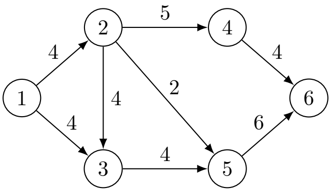


Beispiel

Kapazitätsproblem $(\mathbb{N}_\infty, \max, \min, 0, \infty)$

$$D_G^{(0)} = \begin{pmatrix} \infty & 4 & 4 & 0 & 0 & 0 \\ 0 & \infty & 4 & 5 & 2 & 0 \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix}$$

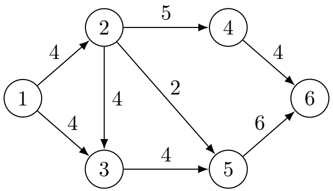
$$D_G^{(1)} = D_G^{(0)}$$



Beispiel

Kapazitätsproblem $(\mathbb{N}_\infty, \max, \min, 0, \infty)$

$$D_G^{(0)} = \begin{pmatrix} \infty & 4 & 4 & 0 & 0 & 0 \\ 0 & \infty & 4 & 5 & 2 & 0 \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix} \qquad D_G^{(1)} = D_G^{(0)}$$

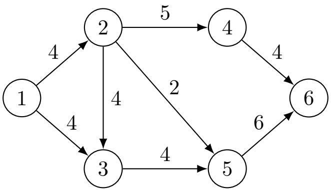


$$D_G^{(2)} = \begin{pmatrix} \infty & 4 & 4 & \underline{4} & \underline{2} & 0 \\ 0 & \infty & 4 & 5 & 2 & 0 \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix}$$

Beispiel

Kapazitätsproblem $(\mathbb{N}_\infty, \max, \min, 0, \infty)$

$$D_G^{(0)} = \begin{pmatrix} \infty & 4 & 4 & 0 & 0 & 0 \\ 0 & \infty & 4 & 5 & 2 & 0 \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix} \qquad D_G^{(1)} = D_G^{(0)}$$

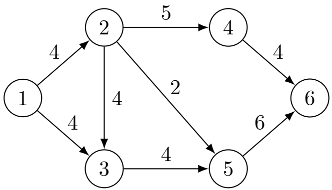


$$D_G^{(2)} = \begin{pmatrix} \infty & 4 & 4 & \underline{4} & \underline{2} & 0 \\ 0 & \infty & 4 & 5 & 2 & 0 \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix} \qquad D_G^{(3)} = \begin{pmatrix} \infty & 4 & 4 & 4 & \underline{4} & 0 \\ 0 & \infty & 4 & 5 & \underline{4} & 0 \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix}$$

Beispiel

Kapazitätsproblem $(\mathbb{N}_\infty, \max, \min, 0, \infty)$

$$D_G^{(0)} = \begin{pmatrix} \infty & 4 & 4 & 0 & 0 & 0 \\ 0 & \infty & 4 & 5 & 2 & 0 \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix} \qquad D_G^{(1)} = D_G^{(0)}$$



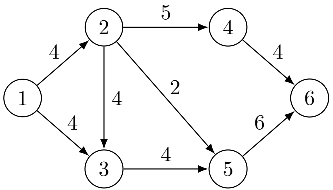
$$D_G^{(2)} = \begin{pmatrix} \infty & 4 & 4 & \underline{4} & \underline{2} & 0 \\ 0 & \infty & 4 & 5 & 2 & 0 \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix} \qquad D_G^{(3)} = \begin{pmatrix} \infty & 4 & 4 & 4 & \underline{4} & 0 \\ 0 & \infty & 4 & 5 & \underline{4} & 0 \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix}$$

$$D_G^{(4)} = \begin{pmatrix} \infty & 4 & 4 & 4 & 4 & \underline{4} \\ 0 & \infty & 4 & 5 & 4 & \underline{4} \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix}$$

Beispiel

Kapazitätsproblem $(\mathbb{N}_\infty, \max, \min, 0, \infty)$

$$D_G^{(0)} = \begin{pmatrix} \infty & 4 & 4 & 0 & 0 & 0 \\ 0 & \infty & 4 & 5 & 2 & 0 \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix} \qquad D_G^{(1)} = D_G^{(0)}$$

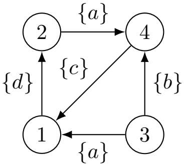


$$D_G^{(2)} = \begin{pmatrix} \infty & 4 & 4 & \underline{4} & \underline{2} & 0 \\ 0 & \infty & 4 & 5 & 2 & 0 \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix} \qquad D_G^{(3)} = \begin{pmatrix} \infty & 4 & 4 & 4 & \underline{4} & 0 \\ 0 & \infty & 4 & 5 & \underline{4} & 0 \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix}$$

$$D_G^{(4)} = \begin{pmatrix} \infty & 4 & 4 & 4 & 4 & \underline{4} \\ 0 & \infty & 4 & 5 & 4 & \underline{4} \\ 0 & 0 & \infty & 0 & 4 & 0 \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix} \qquad D_G^{(5)} = \begin{pmatrix} \infty & 4 & 4 & 4 & 4 & 4 \\ 0 & \infty & 4 & 5 & 4 & 4 \\ 0 & 0 & \infty & 0 & 4 & \underline{4} \\ 0 & 0 & 0 & \infty & 0 & 4 \\ 0 & 0 & 0 & 0 & \infty & 6 \\ 0 & 0 & 0 & 0 & 0 & \infty \end{pmatrix} = D_G^{(6)} = D_G$$

Beispiel

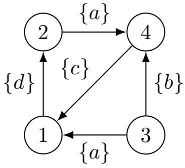
Prozessproblem $(P(\Sigma^*), \cup, \circ, \emptyset, \{\varepsilon\})$



Beispiel

Prozessproblem $(P(\Sigma^*), \cup, \circ, \emptyset, \{\varepsilon\})$

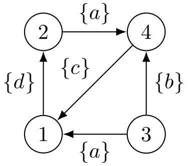
$$D_G^{(0)} = \begin{pmatrix} \{\varepsilon\} & \{d\} & \emptyset & \emptyset \\ \emptyset & \{\varepsilon\} & \emptyset & \{a\} \\ \{a\} & \emptyset & \{\varepsilon\} & \{b\} \\ \{c\} & \emptyset & \emptyset & \{\varepsilon\} \end{pmatrix}$$



Beispiel

Prozessproblem $(P(\Sigma^*), \cup, \circ, \emptyset, \{\varepsilon\})$

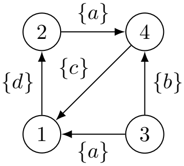
$$D_G^{(0)} = \begin{pmatrix} \{\varepsilon\} & \{d\} & \emptyset & \emptyset \\ \emptyset & \{\varepsilon\} & \emptyset & \{a\} \\ \{a\} & \emptyset & \{\varepsilon\} & \{b\} \\ \{c\} & \emptyset & \emptyset & \{\varepsilon\} \end{pmatrix}$$



$$D_G^{(1)} = \begin{pmatrix} \{\varepsilon\} & \{d\} & \emptyset & \emptyset \\ \emptyset & \{\varepsilon\} & \emptyset & \{a\} \\ \{a\} & \{ad\} & \{\varepsilon\} & \{b\} \\ \{c\} & \{cd\} & \emptyset & \{\varepsilon\} \end{pmatrix}$$

Beispiel

Prozessproblem $(P(\Sigma^*), \cup, \circ, \emptyset, \{\varepsilon\})$



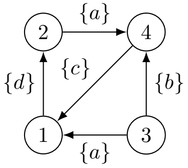
$$D_G^{(0)} = \begin{pmatrix} \{\varepsilon\} & \{d\} & \emptyset & \emptyset \\ \emptyset & \{\varepsilon\} & \emptyset & \{a\} \\ \{a\} & \emptyset & \{\varepsilon\} & \{b\} \\ \{c\} & \emptyset & \emptyset & \{\varepsilon\} \end{pmatrix}$$

$$D_G^{(1)} = \begin{pmatrix} \{\varepsilon\} & \{d\} & \emptyset & \emptyset \\ \emptyset & \{\varepsilon\} & \emptyset & \{a\} \\ \{a\} & \{ad\} & \{\varepsilon\} & \{b\} \\ \{c\} & \{cd\} & \emptyset & \{\varepsilon\} \end{pmatrix}$$

$$D_G^{(2)} = \begin{pmatrix} \{\varepsilon\} & \{d\} & \emptyset & \{da\} \\ \emptyset & \{\varepsilon\} & \emptyset & \{a\} \\ \{a\} & \{ad\} & \{\varepsilon\} & \{b, ada\} \\ \{c\} & \{cd\} & \emptyset & \{\varepsilon, cda\} \end{pmatrix}$$

Beispiel

Prozessproblem $(P(\Sigma^*), \cup, \circ, \emptyset, \{\varepsilon\})$



$$D_G^{(0)} = \begin{pmatrix} \{\varepsilon\} & \{d\} & \emptyset & \emptyset \\ \emptyset & \{\varepsilon\} & \emptyset & \{a\} \\ \{a\} & \emptyset & \{\varepsilon\} & \{b\} \\ \{c\} & \emptyset & \emptyset & \{\varepsilon\} \end{pmatrix}$$

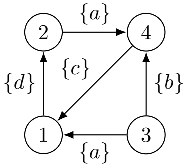
$$D_G^{(1)} = \begin{pmatrix} \{\varepsilon\} & \{d\} & \emptyset & \emptyset \\ \emptyset & \{\varepsilon\} & \emptyset & \{a\} \\ \{a\} & \{ad\} & \{\varepsilon\} & \{b\} \\ \{c\} & \{cd\} & \emptyset & \{\varepsilon\} \end{pmatrix}$$

$$D_G^{(2)} = \begin{pmatrix} \{\varepsilon\} & \{d\} & \emptyset & \{da\} \\ \emptyset & \{\varepsilon\} & \emptyset & \{a\} \\ \{a\} & \{ad\} & \{\varepsilon\} & \{b, ada\} \\ \{c\} & \{cd\} & \emptyset & \{\varepsilon, cda\} \end{pmatrix}$$

$$D_G^{(3)} = D_G^{(2)}$$

Beispiel

Prozessproblem $(P(\Sigma^*), \cup, \circ, \emptyset, \{\varepsilon\})$



$$D_G^{(0)} = \begin{pmatrix} \{\varepsilon\} & \{d\} & \emptyset & \emptyset \\ \emptyset & \{\varepsilon\} & \emptyset & \{a\} \\ \{a\} & \emptyset & \{\varepsilon\} & \{b\} \\ \{c\} & \emptyset & \emptyset & \{\varepsilon\} \end{pmatrix}$$

$$D_G^{(1)} = \begin{pmatrix} \{\varepsilon\} & \{d\} & \emptyset & \emptyset \\ \emptyset & \{\varepsilon\} & \emptyset & \{a\} \\ \{a\} & \{ad\} & \{\varepsilon\} & \{b\} \\ \{c\} & \{cd\} & \emptyset & \{\varepsilon\} \end{pmatrix}$$

$$D_G^{(2)} = \begin{pmatrix} \{\varepsilon\} & \{d\} & \emptyset & \{da\} \\ \emptyset & \{\varepsilon\} & \emptyset & \{a\} \\ \{a\} & \{ad\} & \{\varepsilon\} & \{b, ada\} \\ \{c\} & \{cd\} & \emptyset & \{\varepsilon, cda\} \end{pmatrix}$$

$$D_G^{(3)} = D_G^{(2)}$$

$$D_G^{(4)} = \begin{pmatrix} \{dac\}^* & \{dac\}^* \circ \{d\} & \emptyset & \{dac\}^* \circ \{da\} \\ \{acd\}^* \circ \{ac\} & \{acd\}^* & \emptyset & \{acd\}^* \circ \{a\} \\ \{a, bc\} \circ \{dac\}^* & \{a, bc\} \circ \{dac\}^* \circ \{d\} & \{\varepsilon\} & \{b, ada\} \circ \{cda\}^* \\ \{cda\}^* \circ \{c\} & \{cda\}^* \circ \{cd\} & \emptyset & \{cda\}^* \end{pmatrix}$$

Inhaltsverzeichnis I

Inhaltsverzeichnis II

Lernverfahren

Lernen durch Aufnahme von Fakten.

- Lernen auf Prüfung
- Lesen von Büchern/Besuchen einer Vorlesung

$\rightarrow$ *klares* Wissen

Lernverfahren

Lernen durch Aufnahme von Fakten.

- Lernen auf Prüfung
- Lesen von Büchern/Besuchen einer Vorlesung

$\rightarrow$ *klares* Wissen

Empirisches Lernen.

- wiederholte Betrachtung der Umwelt
- Analyse wiederholter Verhaltensmuster

$\rightarrow$ *unscharfes* Wissen

Zwölf Satzpaare außerirdischer Sprachen

1a. ok-voon ororok sprok .
1b. at-voon bichat dat .

2a. ok-drubel ok-voon anok plok sprok .
2b. at-drubel at-voon pippat rrat dat .

3a. erok sprok izok hihok ghirok .
3b. totat dat arrat vat hilat .

4a. ok-voon anok drok brok jok .
4b. at-voon krat pippat sat lat .

5a. wiwok farok izok stok .
5b. totat jjat quat cat .

6a. lalok sprok izok jok stok .
6b. wat dat krat quat cat .

7a. lalok farok ororok lalok sprok izok enemok .
7b. wat jjat bichat wat dat vat eneat .

8a. lalok brok anok plok nok .
8b. iat lat pippat rrat nnat .

9a. wiwok nok izok kantok ok-yurp .
9b. totat nnat quat oloat at-yurp .

10a. lalok mok nok yorok ghirok clok .
10b. wat nnat gat mat bat hilat .

11a. lalok nok crrrok hihok yorok zanzanok .
11b. wat nnat arrat mat zanzanat .

12a. lalok rarok nok izok hihok mok .
12b. wat nnat forat arrat vat gat .

Zwölf Satzpaare außerirdischer Sprachen

anok	–	pippat
brok	–	lat
clok	–	bat
crrrok	–	*keines* (?)
drok	–	sat
enemok	–	eneat
erok	–	totat
farok	–	jjat
ghirok	–	hilat
hihok	–	arrat
izok	–	vat / quat
jok	–	krat
kantok	–	oloat
lalok	–	wat / iat
mok	-	gat
nok	-	nnat
ok-drubel	-	at-drubel
ok-voon	-	at-voon
ok-yurp	-	at-yurp
ororok	-	bichat
plok	-	rrat
rarok	-	forat
sprok	-	dat
stok	-	cat
wiwok	-	totat
yorok	-	mat
zanzanok	-	zanzanat

Zufallsexperimente

Zufallsexperiment: (X, p)

X endliche Menge (Ergebnismenge),

$p \colon X \to [0,1]$ mit $\sum_{x \in X} p(x) = 1$ (Wahrscheinlichkeitsverteilung)

Zufallsexperimente

Zufallsexperiment: (X, p)

X endliche Menge (Ergebnismenge),

$p\colon X \to [0,1]$ mit $\sum_{x \in X} p(x) = 1$ (Wahrscheinlichkeitsverteilung)

Menge aller W-verteilungen über X: $\mathcal{M}(X)$

Zufallsexperimente

Zufallsexperiment: (X, p)

X endliche Menge (Ergebnismenge),

$p\colon X \to [0,1]$ mit $\sum_{x \in X} p(x) = 1$ (Wahrscheinlichkeitsverteilung)

Menge aller W-verteilungen über X: $\mathcal{M}(X)$

Wahrscheinlichkeitsmodell: $\mathcal{M} \subseteq \mathcal{M}(X)$

Zufallsexperimente

Zufallsexperiment: (X, p)

X endliche Menge (Ergebnismenge),

$p\colon X \to [0,1]$ mit $\sum_{x \in X} p(x) = 1$ (Wahrscheinlichkeitsverteilung)

Menge aller W-verteilungen über X: $\mathcal{M}(X)$

Wahrscheinlichkeitsmodell: $\mathcal{M} \subseteq \mathcal{M}(X)$

Seien $p^1 \in \mathcal{M}(X_1)$ und $p^2 \in \mathcal{M}(X_2)$. Das unabhängige Produkt von p^1 und p^2 ist die W-verteilung $p^1 \times p^2$ über $X_1 \times X_2$ definiert durch

$$(p^1 \times p^2)(a, b) = p^1(a) \cdot p^2(b)$$

für jedes $a \in X_1$ und $b \in X_2$.

Korpora und Korpuswahrscheinlichkeiten

Korpus $h\colon X \to \mathbb{R}^{\geq 0}$ $\quad$ $\{x \mid h(x) > 0\}$ ist endlich und $\sum_{x \in X} h(x)$ ist nicht gleich 0

Korpora und Korpuswahrscheinlichkeiten

Korpus $h \colon X \to \mathbb{R}^{\geq 0}$ $\quad$ $\{x \mid h(x) > 0\}$ ist endlich und $\sum_{x \in X} h(x)$ ist nicht gleich 0

Likelihood $L(h, p) = \prod_{x \in X} p(x)^{h(x)}$

Korpora und Korpuswahrscheinlichkeiten

Korpus $h\colon X \to \mathbb{R}^{\geq 0}$ $\quad \{x \mid h(x) > 0\}$ ist endlich und $\sum_{x \in X} h(x)$ ist nicht gleich 0

Likelihood $L(h, p) = \prod_{x \in X} p(x)^{h(x)}$

Maximum-Likelihood Schätzer $\mathrm{mle}(h, \mathcal{M}) = \mathrm{argmax}_{p \in \mathcal{M}} L(h, p)$

Korpora und Korpuswahrscheinlichkeiten

Korpus $h\colon X \to \mathbb{R}^{\geq 0}$ $\{x \mid h(x) > 0\}$ ist endlich und $\sum_{x \in X} h(x)$ ist nicht gleich 0

Likelihood $L(h,p) = \prod_{x \in X} p(x)^{h(x)}$

Maximum-Likelihood Schätzer $\mathrm{mle}(h, \mathcal{M}) = \mathrm{argmax}_{p \in \mathcal{M}} L(h,p)$

relative Häufigkeit $\mathrm{rfe}(h)(x) = \frac{h(x)}{|h|}$ für jedes $x \in X$

Korpora und Korpuswahrscheinlichkeiten

Korpus $h\colon X \to \mathbb{R}^{\geq 0}$ $\{x \mid h(x) > 0\}$ ist endlich und $\sum_{x \in X} h(x)$ ist nicht gleich 0

Likelihood $L(h, p) = \prod_{x \in X} p(x)^{h(x)}$

Maximum-Likelihood Schätzer $\mathrm{mle}(h, \mathcal{M}) = \mathrm{argmax}_{p \in \mathcal{M}} L(h, p)$

relative Häufigkeit $\mathrm{rfe}(h)(x) = \frac{h(x)}{|h|}$ für jedes $x \in X$

Satz: Sei X eine Ergebnismenge und h ein X-Korpus.

1. $\mathrm{rfe}(h) \in \mathcal{M}(X)$
2. $\mathrm{rfe}(h) = \mathrm{mle}(h, \mathcal{M}(X))$

Korpora und Korpuswahrscheinlichkeiten

Beispiel
Eine Münze mit unbekannter W-verteilung $p\colon \{K, Z\} \to [0, 1]$

30 Würfe, davon 12 Mal Kopf und 18 Mal Zahl.

Korpora und Korpuswahrscheinlichkeiten

Beispiel
Eine Münze mit unbekannter W-verteilung $p\colon \{K, Z\} \to [0, 1]$

30 Würfe, davon 12 Mal Kopf und 18 Mal Zahl.

$\{K, Z\}$-Korpus $h\colon \{K, Z\} \to \mathbb{R}^{\geq 0}$, $h(K) = 12$, $h(Z) = 18$

Korpora und Korpuswahrscheinlichkeiten

Beispiel
Eine Münze mit unbekannter W-verteilung $p\colon \{K, Z\} \to [0, 1]$

30 Würfe, davon 12 Mal Kopf und 18 Mal Zahl.

$\{K, Z\}$-Korpus $h\colon \{K, Z\} \to \mathbb{R}^{\geq 0}$, $h(K) = 12$, $h(Z) = 18$

relative Häufigkeit von h; $\quad \mathrm{rfe}(h)(K) = \frac{12}{30} = \frac{2}{5} \qquad \mathrm{rfe}(h)(Z) = \frac{18}{30} = \frac{3}{5}$

Beispiel

Eine Münze mit unbekannter W-verteilung $p\colon \{K, Z\} \to [0, 1]$

30 Würfe, davon 12 Mal Kopf und 18 Mal Zahl.

$\{K, Z\}$-Korpus $h\colon \{K, Z\} \to \mathbb{R}^{\geq 0}$, $h(K) = 12$, $h(Z) = 18$

relative Häufigkeit von h; $\mathrm{rfe}(h)(K) = \frac{12}{30} = \frac{2}{5}$ $\mathrm{rfe}(h)(Z) = \frac{18}{30} = \frac{3}{5}$

$$\mathrm{mle}(h, \mathcal{M}(\{K, Z\})) = \mathrm{rfe}(h)$$

Was passiert, wenn $\mathcal{M} \neq \mathcal{M}(X)$?

Sei $\mathcal{M} = \{p^1 \times p^2 \mid p^1, p^2 \in \mathcal{M}(\{K, Z\})\}$.

Dann gilt: $\mathcal{M}(X_1 \times X_2) \setminus \mathcal{M} \neq \emptyset$ denn $p \notin \mathcal{M}$ wenn

$$p(K, K) = p(Z, Z) = 0 \text{ und } p(K, Z) = p(Z, K) = 0.5$$

Was passiert, wenn $\mathcal{M} \neq \mathcal{M}(X)$?

Sei $\mathcal{M} = \{p^1 \times p^2 \mid p^1, p^2 \in \mathcal{M}(\{K, Z\})\}$.

Dann gilt: $\mathcal{M}(X_1 \times X_2) \setminus \mathcal{M} \neq \emptyset$ denn $p \notin \mathcal{M}$ wenn

$$p(K, K) = p(Z, Z) = 0 \text{ und } p(K, Z) = p(Z, K) = 0.5$$

Satz: Seien X_1 und X_2 Ergebnismengen und $\mathcal{M} = \{p^1 \times p^2 \mid p^1 \in \mathcal{M}(X_1), p^2 \in \mathcal{M}(X_2)\}$ ein Wahrscheinlichkeitsmodell über $X_1 \times X_2$. Weiterhin sei h ein $(X_1 \times X_2)$-Korpus. Dann ist

$$\mathrm{mle}(h, \mathcal{M}) = \mathrm{rfe}(h^1) \times \mathrm{rfe}(h^2) \ ,$$

wobei h^1 der X_1-Korpus und h^2 der X_2-Korpus ist, die wie folgt definiert sind:

$$h^1(x_1) = \sum_{x_2 \in X_2} h(x_1, x_2) \ , \qquad \text{(für jedes } x_1 \in X_1\text{)}$$

$$h^2(x_2) = \sum_{x_1 \in X_1} h(x_1, x_2) \ . \qquad \text{(für jedes } x_2 \in X_2\text{)}$$

Den Übergang von h nach h^1 oder h^2 nennt man *Marginalisieren*.

30-maliger Münzwurf

Ziel: Finden der Wahrscheinlichkeitsverteilungen der Münzen

Korpus:

$$h(K,K) = 5 \qquad h(K,Z) = 10 \qquad h(Z,K) = 5 \qquad h(Z,Z) = 10$$

30-maliger Münzwurf

Ziel: Finden der Wahrscheinlichkeitsverteilungen der Münzen

Korpus:

$$h(K,K) = 5 \qquad h(K,Z) = 10 \qquad h(Z,K) = 5 \qquad h(Z,Z) = 10$$

Marginalisierung liefert die Teilkorpora:

$$h^1(K) = h(K,K) + h(K,Z) = 15 \qquad h^2(K) = h(K,K) + h(Z,K) = 10$$
$$h^1(Z) = h(Z,K) + h(Z,Z) = 15 \qquad h^2(Z) = h(K,Z) + h(Z,Z) = 20$$

30-maliger Münzwurf

Ziel: Finden der Wahrscheinlichkeitsverteilungen der Münzen

Korpus:

$$h(K,K) = 5 \qquad h(K,Z) = 10 \qquad h(Z,K) = 5 \qquad h(Z,Z) = 10$$

Marginalisierung liefert die Teilkorpora:

$$h^1(K) = h(K,K) + h(K,Z) = 15 \qquad h^2(K) = h(K,K) + h(Z,K) = 10$$
$$h^1(Z) = h(Z,K) + h(Z,Z) = 15 \qquad h^2(Z) = h(K,Z) + h(Z,Z) = 20$$

Schätzung nach relativer Häufigkeit:

$$\mathrm{rfe}(h^1)(K) = \frac{h^1(K)}{|h^1|} = 1/2 \qquad \mathrm{rfe}(h^2)(K) = \frac{h^2(K)}{|h^2|} = 1/3$$
$$\mathrm{rfe}(h^1)(Z) = \frac{h^1(Z)}{|h^1|} = 1/2 \qquad \mathrm{rfe}(h^2)(Z) = \frac{h^2(Z)}{|h^2|} = 2/3$$

Korpora mit unvollständigen Daten

Beispiel
Person A wirft zwei Münzen.

Ergebnismenge $X = \{(K, K), (K, Z), (Z, K), (Z, Z)\}$

Korpora mit unvollständigen Daten

Beispiel
Person A wirft zwei Münzen.

Ergebnismenge $X = \{(K,K),(K,Z),(Z,K),(Z,Z)\}$

A teilt Person B nach jedem Wurf mit, wie oft die Kopfseite zu sehen ist.

Korpora mit unvollständigen Daten

Beispiel
Person A wirft zwei Münzen.

Ergebnismenge $X = \{(K,K),(K,Z),(Z,K),(Z,Z)\}$

A teilt Person B nach jedem Wurf mit, wie oft die Kopfseite zu sehen ist.

B beobachtet 0, 1 oder 2

Beobachtungsmenge $Y = \{0,1,2\}$

Korpora mit unvollständigen Daten

Beispiel
Person A wirft zwei Münzen.

Ergebnismenge $X = \{(K, K), (K, Z), (Z, K), (Z, Z)\}$

A teilt Person B nach jedem Wurf mit, wie oft die Kopfseite zu sehen ist.

B beobachtet 0, 1 oder 2

Beobachtungsmenge $Y = \{0, 1, 2\}$

Beobachtung	Ergebnis
0	(Z, Z)
1	(K, Z) oder (Z, K)
2	(K, K)

Korpora mit unvollständigen Daten

Ergebnismenge X, **Beobachtungsmenge** Y.

Beobachtungsfunktion: $\mathrm{yield}\colon X \to Y$

Korpora mit unvollständigen Daten

Ergebnismenge X, **Beobachtungsmenge** Y.

Beobachtungsfunktion: $\mathrm{yield}\colon X \to Y$

Analysator (Umkehrfunktion von yield): $A\colon Y \to \mathcal{P}(X)$

Korpora mit unvollständigen Daten

Ergebnismenge X, **Beobachtungsmenge** Y.

Beobachtungsfunktion: $\mathrm{yield}\colon X \to Y$

Analysator (Umkehrfunktion von yield): $A\colon Y \to \mathcal{P}(X)$

$$A(y) = \{x \in X \mid \mathrm{yield}(x) = y\} \qquad \text{(für jedes } x \in X\text{)}$$

Korpora mit unvollständigen Daten

Ergebnismenge X, **Beobachtungsmenge** Y.

Beobachtungsfunktion: $\mathrm{yield} \colon X \to Y$

Analysator (Umkehrfunktion von yield): $A \colon Y \to \mathcal{P}(X)$

$$A(y) = \{x \in X \mid \mathrm{yield}(x) = y\} \qquad \text{(für jedes } x \in X\text{)}$$

Beispiel

$$\mathrm{yield}(K, K) = 2 , \qquad \mathrm{yield}(K, Z) = 1 , \qquad \mathrm{yield}(Z, K) = 1 , \qquad \mathrm{yield}(Z, Z) = 0 .$$

$$A(0) = \{(Z, Z)\} , \qquad A(1) = \{(K, Z), (Z, K)\} , \qquad A(2) = \{(K, K)\} .$$

Korpora mit unvollständigen Daten

Sei h ein Y-Korpus und $p \in \mathcal{M}(X)$. Die *Korpuswahrscheinlichkeit von* h *unter* p ist

$$L(h,p) = \prod_{y \in Y} \Big(\sum\nolimits_{x \in A(y)} p(x) \Big)^{h(y)} .$$

Sei $\mathcal{M} \subseteq \mathcal{M}(X)$. Der *Maximum-Likelihood-Schätzer* von h und $\mathcal{M}$ ist dann definiert wie im Fall mit vollständigen Daten, also

$$\mathrm{mle}(h, \mathcal{M}) = \mathrm{argmax}_{p \in \mathcal{M}} L(h,p)$$

Korpora mit unvollständigen Daten

Sei h ein Y-Korpus und $p \in \mathcal{M}(X)$. Die *Korpuswahrscheinlichkeit von* h *unter* p ist

$$L(h,p) = \prod_{y \in Y} \Big(\sum\nolimits_{x \in A(y)} p(x) \Big)^{h(y)} .$$

Sei $\mathcal{M} \subseteq \mathcal{M}(X)$. Der *Maximum-Likelihood-Schätzer* von h und $\mathcal{M}$ ist dann definiert wie im Fall mit vollständigen Daten, also

$$\mathrm{mle}(h, \mathcal{M}) = \mathrm{argmax}_{p \in \mathcal{M}} L(h,p)$$

Satz: Sei $q_1, q_2, q_3, \ldots$ die durch den EM-Algorithmus berechnete Sequenz von Wahrscheinlichkeitsverteilungen über X. Dann gilt

$$L(h, q_0) \leq L(h, q_1) \leq L(h, q_2) \leq L(h, q_3) \leq \cdots \leq L(h, \mathrm{mle}(h, \mathcal{M})) \ .$$

Algorithmus EM-Algorithmus

Eingabe ein Y-Korpus h;
ein Analysator $A\colon Y \to \mathcal{P}(X)$;
ein Wahrscheinlichkeitsmodell $\mathcal{M} \subseteq \mathcal{M}(X)$ über X;
ein $q_0 \in \mathcal{M}$, so dass $q_0(x) > 0$ für jedes $x \in X$.

Ausgabe eine Sequenz $q_1, q_2, q_3, \dots$ von Elementen aus $\mathcal{M}$.

für jedes $i = 1, 2, 3, \dots$

  E-Schritt berechne den X-Korpus h_i:

$$h_i(x) = h(\mathrm{yield}(x)) \cdot \frac{q_{i-1}(x)}{\sum_{x' \in A(\mathrm{yield}(x))} q_{i-1}(x')}$$

  M-Schritt berechne den Maximum-Likelihood-Schätzer von h_i und $\mathcal{M}$:

$$q_i = \mathrm{argmax}_{p \in \mathcal{M}} L(h_i, p)$$

  print q_i

A wirft zwei Münzen 15 mal und teilt B mit, dass bei 4 Würfen 0 mal Kopf gefallen ist, bei 9 Würfen 1 mal Kopf und bei 2 Würfen 2 mal Kopf.

$$h(0) = 4\,, \qquad h(1) = 9\,, \qquad h(2) = 2\,.$$

Wahrscheinlichkeitsmodell: $\mathcal{M} = \{p^1 \times p^2 \mid p^1, p^2 \in \mathcal{M}(\{K, Z\})\}$

A wirft zwei Münzen 15 mal und teilt B mit, dass bei 4 Würfen 0 mal Kopf gefallen ist, bei 9 Würfen 1 mal Kopf und bei 2 Würfen 2 mal Kopf.

$$h(0) = 4\,, \qquad h(1) = 9\,, \qquad h(2) = 2\,.$$

Wahrscheinlichkeitsmodell: $\mathcal{M} = \{p^1 \times p^2 \mid p^1, p^2 \in \mathcal{M}(\{K, Z\})\}$

Sei $p^1(K) = a,\ p^1(Z) = 1 - a$ und
$p^2(K) = b,\ p^2(Z) = 1 - b$

	Ablauf 1	Ablauf 2	Ablauf 3	Ablauf 4
(a_0, b_0)	$(0.200, 0.500)$	$(0.900, 0.600)$	$(0.000, 1.000)$	$(0.400, 0.400)$
(a_1, b_1)	$(0.253, 0.613)$	$(0.648, 0.219)$	$(0.133, 0.733)$	$(0.433, 0.433)$
(a_2, b_2)	$(0.239, 0.628)$	$(0.654, 0.213)$	$(0.165, 0.687)$	$(0.433, 0.433)$
(a_3, b_3)	$(0.228, 0.639)$	$(0.658, 0.208)$	$(0.180, 0.679)$	$(0.433, 0.433)$
(a_4, b_4)	$(0.219, 0.648)$	$(0.661, 0.205)$	$(0.188, 0.674)$	$(0.433, 0.433)$
(a_5, b_5)	$(0.213, 0.654)$	$(0.663, 0.204)$	$(0.193, 0.671)$	$(0.433, 0.433)$
(a_{20}, b_{20})	$(0.200, 0.667)$	$(0.667, 0.200)$	$(0.200, 0.667)$	$(0.433, 0.433)$

	$\widehat{p}_1$	$\widehat{p}_2$	$\widehat{p}_3$
(K,K)	$\frac{1}{5} \cdot \frac{2}{3} = \frac{2}{15}$	$\frac{2}{3} \cdot \frac{1}{5} = \frac{2}{15}$	$\frac{13}{30} \cdot \frac{13}{30} = \frac{169}{900}$
(K,Z)	$\frac{1}{5} \cdot \frac{1}{3} = \frac{1}{15}$	$\frac{2}{3} \cdot \frac{4}{5} = \frac{8}{15}$	$\frac{13}{30} \cdot \frac{17}{30} = \frac{221}{900}$
(Z,K)	$\frac{4}{5} \cdot \frac{2}{3} = \frac{8}{15}$	$\frac{1}{3} \cdot \frac{1}{5} = \frac{1}{15}$	$\frac{17}{30} \cdot \frac{13}{30} = \frac{221}{900}$
(Z,Z)	$\frac{4}{5} \cdot \frac{1}{3} = \frac{4}{15}$	$\frac{1}{3} \cdot \frac{4}{5} = \frac{4}{15}$	$\frac{17}{30} \cdot \frac{17}{30} = \frac{289}{900}$

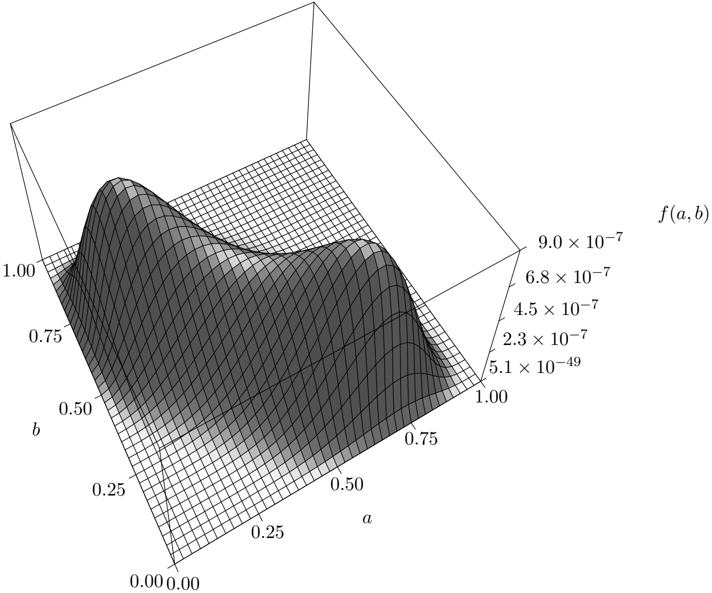
	$\widehat{p}_1$	$\widehat{p}_2$	$\widehat{p}_3$
(K,K)	$\frac{1}{5}\cdot\frac{2}{3}=\frac{2}{15}$	$\frac{2}{3}\cdot\frac{1}{5}=\frac{2}{15}$	$\frac{13}{30}\cdot\frac{13}{30}=\frac{169}{900}$
(K,Z)	$\frac{1}{5}\cdot\frac{1}{3}=\frac{1}{15}$	$\frac{2}{3}\cdot\frac{4}{5}=\frac{8}{15}$	$\frac{13}{30}\cdot\frac{17}{30}=\frac{221}{900}$
(Z,K)	$\frac{4}{5}\cdot\frac{2}{3}=\frac{8}{15}$	$\frac{1}{3}\cdot\frac{1}{5}=\frac{1}{15}$	$\frac{17}{30}\cdot\frac{13}{30}=\frac{221}{900}$
(Z,Z)	$\frac{4}{5}\cdot\frac{1}{3}=\frac{4}{15}$	$\frac{1}{3}\cdot\frac{4}{5}=\frac{4}{15}$	$\frac{17}{30}\cdot\frac{17}{30}=\frac{289}{900}$

$$L(h,\hat{p}_1) = 0.90596 \cdot 10^{-6}$$

$$L(h,\hat{p}_2) = 0.90596 \cdot 10^{-6}$$

$$L(h,\hat{p}_3) = 0.62305 \cdot 10^{-6}$$

Sei $f : [0,1]^2 \to \mathbb{R}$ mit $f(a,b) = \big((1-a)(1-b)\big)^4 \cdot \big(a(1-b) + (1-a)b\big)^9 \cdot \big(ab\big)^2$.



Inhaltsverzeichnis I

Inhaltsverzeichnis II

Teil II – Algorithmische Problemstellungen

Fibonacci

1. Im ersten Jahr gibt es ein Kaninchenpaar (KP).
2. Jedes KP hat erstmals nach zwei Jahren *ein* KP als Nachwuchs und gebiert dann jährlich *ein* KP.
3. Alle Kaninchen sind unsterblich.

Fibonacci

1. Im ersten Jahr gibt es ein Kaninchenpaar (KP).
2. Jedes KP hat erstmals nach zwei Jahren *ein* KP als Nachwuchs und gebiert dann jährlich *ein* KP.
3. Alle Kaninchen sind unsterblich.

```
int fib_rek(int n)
{ if (n <= 1) return n;
  else return (fib_rek(n-1) + fib_rek(n-2));
}
```

Fibonacci

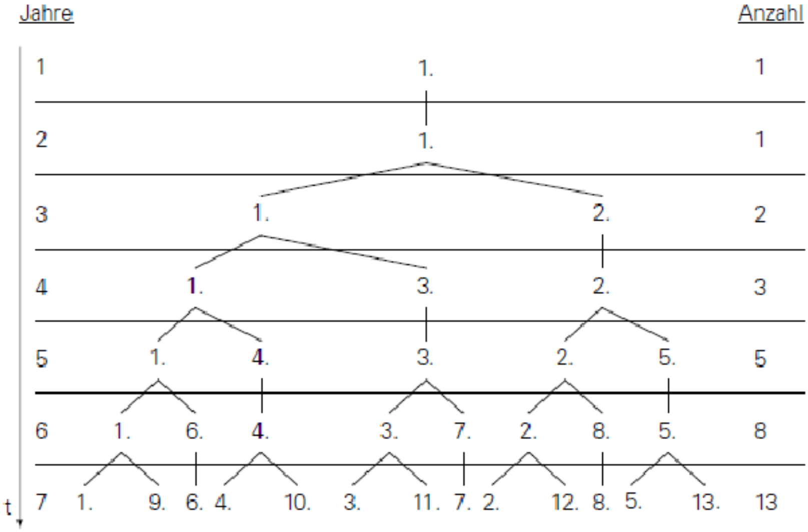


Abbildung: Entwicklung der Kaninchenpopulation

Towers of Hanoi

```
if (n ungerade)
  {RICHTUNG = (A -> B -> C -> A)  /* nach rechts */}
else
  {RICHTUNG = (A -> C -> B -> A)  /* nach links */}

loop = 1;
while (loop)
{ verschiebe kleinste Scheibe um einen Platz in RICHTUNG;
  if (Aufgabe erfüllt)
    break;
  führe den einzig möglichen Schritt durch, der sich nicht
    auf die kleinste Scheibe bezieht
}
```

$$
\begin{aligned}
iter(n) &= it(n, 0, 1) && (n \geq 1)\\
it(1, x, y) &= y\\
it(n+1, x, y) &= it(n, y, x+y)
\end{aligned}
$$

$$
\begin{aligned}
iter(n) &= it(n,0,1) && (n \geq 1)\\
it(1,x,y) &= y\\
it(n+1,x,y) &= it(n,y,x+y)
\end{aligned}
$$

Es gilt für alle $0 \leq k \leq n-1$:

$$it(n-k, KP(k), KP(k+1)) = it(n,0,1) \qquad \text{(Behauptung)}$$

$$
\begin{aligned}
iter(n) &= it(n, 0, 1) && (n \geq 1)\\
it(1, x, y) &= y\\
it(n+1, x, y) &= it(n, y, x+y)
\end{aligned}
$$

Es gilt für alle $0 \leq k \leq n-1$:

$$it(n-k, KP(k), KP(k+1)) = it(n, 0, 1) \qquad \text{(Behauptung)}$$

$k = 0$ (Induktionsanfang)

$$
\begin{aligned}
it(n-k, KP(k), KP(k+1)) &= it(n-0, KP(0), KP(1))\\
&= it(n, 0, 1)
\end{aligned}
$$

$$
\begin{aligned}
iter(n) &= it(n, 0, 1) && (n \geq 1)\\
it(1, x, y) &= y\\
it(n+1, x, y) &= it(n, y, x+y)
\end{aligned}
$$

Es gilt für alle $0 \leq k \leq n-1$:

$$it(n-k, KP(k), KP(k+1)) = it(n, 0, 1) \qquad \text{(Behauptung)}$$

$k = 0$ (Induktionsanfang)

$$
\begin{aligned}
it(n-k, KP(k), KP(k+1)) &= it(n-0, KP(0), KP(1))\\
&= it(n, 0, 1)
\end{aligned}
$$

$k \rightarrow k+1$ (Induktionsschritt)

$$
\begin{aligned}
&it(n-(k+1), KP(k+1), KP(k+2))\\
&= it(n-k-1, KP(k+1), KP(k+1) + KP(k)) && \text{(nach Def. } KP\text{)}\\
&= it(n-k, KP(k), KP(k+1)) && \text{(nach Definition } it \text{ rückwärts)}\\
&= it(n, 0, 1) && \text{(nach Induktionsvoraussetzung)}
\end{aligned}
$$

$$iter(n) = it(n,0,1) \qquad (n \geq 1)$$
$$it(1,x,y) = y$$
$$it(n+1,x,y) = it(n,y,x+y)$$

Es gilt für alle $0 \leq k \leq n-1$:

$$it(n-k, KP(k), KP(k+1)) = it(n,0,1) \qquad \text{(Behauptung)}$$

$k = 0$ (Induktionsanfang)

$$\begin{aligned} it(n-k, KP(k), KP(k+1)) &= it(n-0, KP(0), KP(1)) \\ &= it(n,0,1) \end{aligned}$$

$k \to k+1$ (Induktionsschritt)

$$\begin{aligned} & it(n-(k+1), KP(k+1), KP(k+2)) \\ &= it(n-k-1, KP(k+1), KP(k+1)+KP(k)) && \text{(nach Def. } KP\text{)} \\ &= it(n-k, KP(k), KP(k+1)) && \text{(nach Definition } it \text{ rückwärts)} \\ &= it(n,0,1) && \text{(nach Induktionsvoraussetzung)} \end{aligned}$$

Somit gilt für jedes $n \geq 1$:

$$\begin{aligned} iter(n) &= it(n,0,1) \\ &= it(1, KP(n-1), KP(n)) && \text{(für } k = n-1\text{)} \\ &= KP(n) \end{aligned}$$

Fibonacci

```c
int x, y, n, swap;

scanf("%d", &n);
x = 0;
y = 1;
while (n > 1)
{ swap = y;
  y = x + y;
  x = swap;
  n = n-1;
}
printf("%d", y);
```

Matrix-Kettenmultiplikation

$$M_1 * (M_2 * (M_3 * M_4))$$
$$M_1 * ((M_2 * M_3) * M_4)$$
$$(M_1 * (M_2 * M_3)) * M_4$$
$$((M_1 * M_2) * M_3) * M_4$$
$$(M_1 * M_2) * (M_3 * M_4)$$

Klammerung	Aufwand
$(M_1 * M_2) * M_3$	$5 \cdot 11 \cdot 6 + 5 \cdot 6 \cdot 20 = 930$
$M_1 * (M_2 * M_3)$	$5 \cdot 11 \cdot 20 + 11 \cdot 6 \cdot 20 = 2420$

Matrix-Kettenmultiplikation

bei Klammerung von $M_i, ..., M_j$ hinter der k-ten Matrix:

$$(M_i * \cdots * M_k) * (M_{k+1} * \cdots M_j)$$

minimaler Aufwand bei optimaler Klammerung:

$$m(i,j) = m(i,k) + m(k+1,j) + p_{i-1} \cdot p_k \cdot p_j$$

Matrix-Kettenmultiplikation

bei Klammerung von $M_i, \ldots, M_j$ hinter der k-ten Matrix:

$$(M_i * \cdots * M_k) * (M_{k+1} * \cdots M_j)$$

minimaler Aufwand bei optimaler Klammerung:

$$m(i,j) = m(i,k) + m(k+1,j) + p_{i-1} \cdot p_k \cdot p_j$$

bei unbekannter optimaler Klammerung von $M_i, \ldots, M_j$:

$$m(i,j) = \begin{cases} 0 & \text{wenn } i = j \\ \min_{i \leq k < j}\{m(i,k) + m(k+1,j) + p_{i-1} \cdot p_k \cdot p_j\} & \text{wenn } i < j \end{cases}$$

Matrix-Kettenmultiplikation

$$m(i,j) = \begin{cases} 0 & \text{wenn } i = j \\ \min_{i \leq k < j}\{m(i,k) + m(k+1,j) + p_{i-1} \cdot p_k \cdot p_j\} & \text{wenn } i < j \end{cases}$$

Matrix-Kettenmultiplikation

$$m(i,j) = \begin{cases} 0 & \text{wenn } i = j \\ \min_{i \le k < j} \{ m(i,k) + m(k+1,j) + p_{i-1} \cdot p_k \cdot p_j \} & \text{wenn } i < j \end{cases}$$

Sei `P` das Feld mit den Dimensionen $p_0, p_1, \dots, p_n$ der Matrizen $M_1 \dots M_n$.

```
for (i = 1; i <= n; i = i+1) { m[i][i] = 0 };

for (l = 1; l <= n-1; l = l+1)  /* gehe die Diagonalen der Reihe nach durch */
  for (i = 1; i <= n; i = i+1)
    for (j = 1; j <= n; j = j+1)
      if (j-i == l)
      { m[i][j] = MaxInteger;
        for (k = i; k <= j-1; k = k+1)  /* berechne in m[i][j] das Minimum */
        { q = m[i][k] + m[k+1][j] + P[i-1]* P[k] * P[j];
          if (q < m[i][j])  m[i][j]=q;
        }
      }
```

Matrix-Kettenmultiplikation

Matrix	Dimension
M_1	(5,11)
M_2	(11,6)
M_3	(6,20)
M_4	(20,8)
M_5	(8,9)

m:

$i \diagdown j$	1	2	3	4	5
1	0	330^1	930^5	1530^8	1890^{10}
2	-	0	1320^2	1488^6	1986^9
3	-	-	0	960^3	1392^7
4	-	-	-	0	1440^4
5	-	-	-	-	0

$$m[1][3] = \min\{0 + 1320 + 5 \cdot 11 \cdot 20,\ \ 330 + 0 + 5 \cdot 6 \cdot 20\} = \min\{2420, 930\} = 930$$
$$k = 2$$

$$m[2][4] = \min\{1320 + 0 + 11 \cdot 20 \cdot 8,\ \ 0 + 960 + 11 \cdot 6 \cdot 8\} = \min\{3080, 1488\} = 1488$$
$$k = 2$$

$$m[3][5] = \min\{960 + 0 + 6 \cdot 8 \cdot 9,\ \ 0 + 1440 + 6 \cdot 20 \cdot 9\} = \min\{1392, 2520\} = 1392$$
$$k = 4$$

$$\begin{aligned} m[1][4] &= \min\{0 + 1488 + 5 \cdot 11 \cdot 8,\ \ 330 + 960 + 5 \cdot 6 \cdot 8,\ \ 930 + 0 + 5 \cdot 20 \cdot 8\} \\ &= \min\{1928, 1530, 1730\} = 1530 \\ k &= 2 \end{aligned}$$

$$\begin{aligned} m[2][5] &= \min\{0 + 1392 + 11 \cdot 6 \cdot 9,\ \ 1320 + 1440 + 11 \cdot 20 \cdot 9,\ \ 1488 + 0 + 11 \cdot 8 \cdot 9\} \\ &= \min\{1986, 4740, 2280\} = 1986 \\ k &= 2 \end{aligned}$$

$$\begin{aligned} m[1][5] &= \min\{0 + 1986 + 5 \cdot 11 \cdot 9,\ \ 330 + 1392 + 5 \cdot 6 \cdot 9,\ \ 930 + 1440 + 5 \cdot 20 \cdot 9,\ \ 1530 + 0 + 5 \cdot 8 \cdot 9\} \\ &= \min\{2481, 1992, 3270, 1890\} = 1890 \\ k &= 4 \end{aligned}$$

Matrix-Kettenmultiplikation

$$M_1 * M_2 * M_3 * M_4 * M_5$$

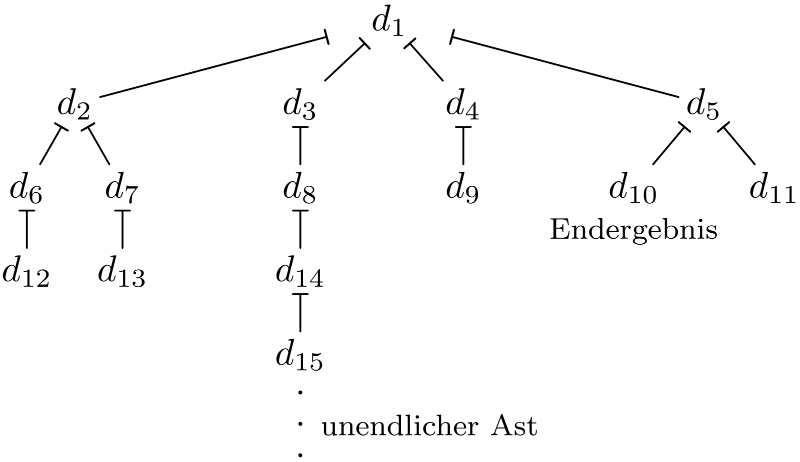
bei $m[1][5]$ ist $k = 4$ (d. h. Trennung nach der Matrix M_4) $\rightarrow$

$$(M_1 * M_2 * M_3 * M_4) * M_5$$

bei $m[1][4]$ ist $k = 2$ $\rightarrow$

$$((M_1 * M_2) * (M_3 * M_4)) * M_5$$

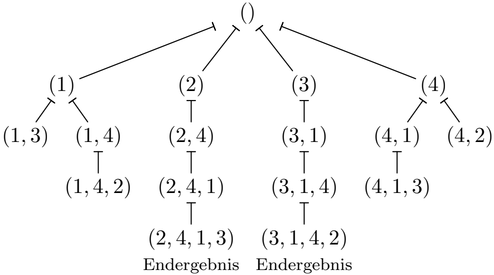
Backtracking

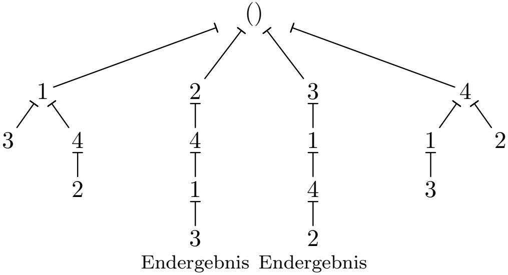


Algorithmus Backtracking

```
void  backtrack (Teillösung)
{ if (Teillösung == Gesamtlösung)
    gib Teillösung aus
  else
    for (jede Erweiterung der Teillösung)
      if (Erweiterung zulässig)
        backtrack (erweiterte Teillösung)
}
```

Backtracking





```c
/* acht-Damen-Problem */
#include <stdio.h>
#include <math.h>

short a[8];

void drucke()
{ short i;
  for (i=0; i<=7;i=i+1) printf("%2d", a[i]);
  printf("\n");
}

int konsistent(int t)
{ short j;

  for (j = 0; j < t; j = j+1)
  { if (a[j] == a[t]) return 0;          /* prüft Bedrohung auf der Zeile a[t] */
    if (abs(a[j] - a[t]) == (t-j)) return 0; /* prüft Bedrohung auf den beiden */
  }                                          /* Diagonalen durch a[t]         */
  return 1;
}

void suche(int t)
{ short i;

  if (t == 8) drucke();
  else
    for (i = 0; i <= 7; i = i+1)
    { a[t] = i;
      if (konsistent(t)) suche(t+1);
    }
}

int main ()
{ suche(0);
  return 0;
}
```

Inhaltsverzeichnis I

Inhaltsverzeichnis II

Literaturverzeichnis I

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman.
The Design and Analysis of Computer Algorithms.
Addison-Wesley, 1974.

[AO91] K. Apt and E.-R. Olderog.
Programmverifikation – Sequentielle, parallele und verteilte Programme.
Springer-Verlag, 1991.

[AO97] K. Apt and E.-R. Olderog.
Verification of Sequential and Concurrent Programs.
Springer-Verlag, 1997.
2nd edition.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest.
Introduction to algorithms.
The MIT Press, 1990.

[CLRS04] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein.
Algorithmen - Eine Einführung.
Oldenbourg Verlag, 2004.

Literaturverzeichnis II

[Hoa69] C. A. R. Hoare.
An Axiomatic Basis for Computer Programming.
Comm. of the ACM, 12(10):576–583, 1969.

[HSAF94] E. Horowitz, S. Sahni, and S. Anderson-Freed.
Grundlagen von Datenstrukturen in C.
International Thomson Publishing, 1994.

[Hut07] G. Hutton.
Programming in Haskell.
Cambridge University Press, 2007.

[Kni97] K. Knight.
Automating knowledge acquisition for machine translation.
AI Magazine, 18(4), 1997.

[Kow74] R. Kowalski.
Predicate logic as a programming language.
Information Processing, 74:569–574, 1974.

[Llo87] J.W. Lloyd.
Foundations of Logic Programming.
Springer-Verlag, 1987.

Literaturverzeichnis III

[McC60] J. McCarthy.
Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I.
Comm. of the ACM, 3:184–195184–195, 1960.

[MHR80] N. Metropolis, J. Howlett, and G. Rota.
A History of Computing in the 20th Century.
Academic Press, 1980.

[OGS08] B. O'Sullivan, J. Goerzen, and D. Stewart.
Real World Haskell.
O'Reilly Media, Inc., 1st edition, 2008.

[OW02] T. Ottmann and P. Widmayer.
Algorithmen und Datenstrukturen.
Sprektrum - Akademischer Verlag, 4 edition, 2002.

[Rob65] J.A. Robinson.
A machine-oriented logic based on the resolution principle.
Journal of the Association for Computer Machinery, 12:23–41, 1965.

[Sch93] U. Schöning.
Vorlesungsskript Informatik I, 5. Auflage.
Universität Ulm, 1993.

Literaturverzeichnis IV

[SGJ86] I.S. Sominskij, L.I. Golovina, and I.M. Jaglom.
Die vollständige Induktion.
Deutscher Verlag der Wissenschaften, 1986.

[SS94] L. Sterling and E. Shapiro.
The Art of Prolog.
MIT Press, 1994.

[Wex81] R. Wexelblatt.
History of programming Languages.
Academic Press, 1981.