Script **generated by TTT**

Title: Einf_HF (24.06.2013)

Date: Mon Jun 24 14:15:11 CEST 2013

Duration: 97:06 min

Pages: 42

Rekursion

Spezielle Form der Modularisierung. Zu definierendes Modul wird in seiner Definition selbst benutzt.

"natürlichere" Darstellung bei bestimmten Algorithmen und Datenstrukturen.

Beispiel Summe

Definiertes Ende einer rekursiven Schachtelung.

```
if (n > 0)
   summe = summe(n-1) + n;
else summe = 0; /* definiertes Ende, wenn n <= 0 */
```

Beispiel 'Fibonacci-Zahlen'

Beispiel 'Größter gemeinsamer Teiler'

Beispiel 'Türme von Hanoi'

Generated by Targeteam

Beispiel Summe

Die Summe einer Folge von aufeinanderfolgenden Zahlen kann iterativ und rekursiv berechnet werden.

Summe (n) =

iterativ: 1 + 2 + ... + (n-1) + n

rekursiv: Summe(n-1) + n

Iterative Berechnung

```
int summe = 0;
for (int i = 1; i < n+1; i++)
   summe = summe + i;
```

Rekursive Berechnung

```
If (n == 0)
   summe = 0;
else
   summe = summe(n-1) + n;
```

Generated by Targeteam

Beispiel 'Fibonacci-Zahlen'

Übliche mathematische Problemstellung rekursiv formuliert, direkt durch rekursiven Algorithmus lösbar.

Definition

fib(n) = fib(n-1) + fib(n-2) für n>2

fib(1) = fib(2) = 1

Wertebereich: n >= 1, mit n natürliche Zahl

Algorithmus

```
Modul fib(n)
Falls n kleiner 3
   dann Ergebnis = 1
   sonst Ergebnis = fib(n-1) + fib(n-2)
```

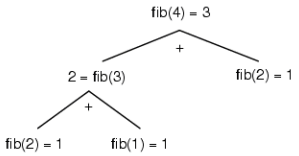
Implementierung in Java

```
static int fib(int n) {
   if (n < 3) return 1;
   else return fib(n-1) + fib(n-2);
}
```

Alternative Implementierung

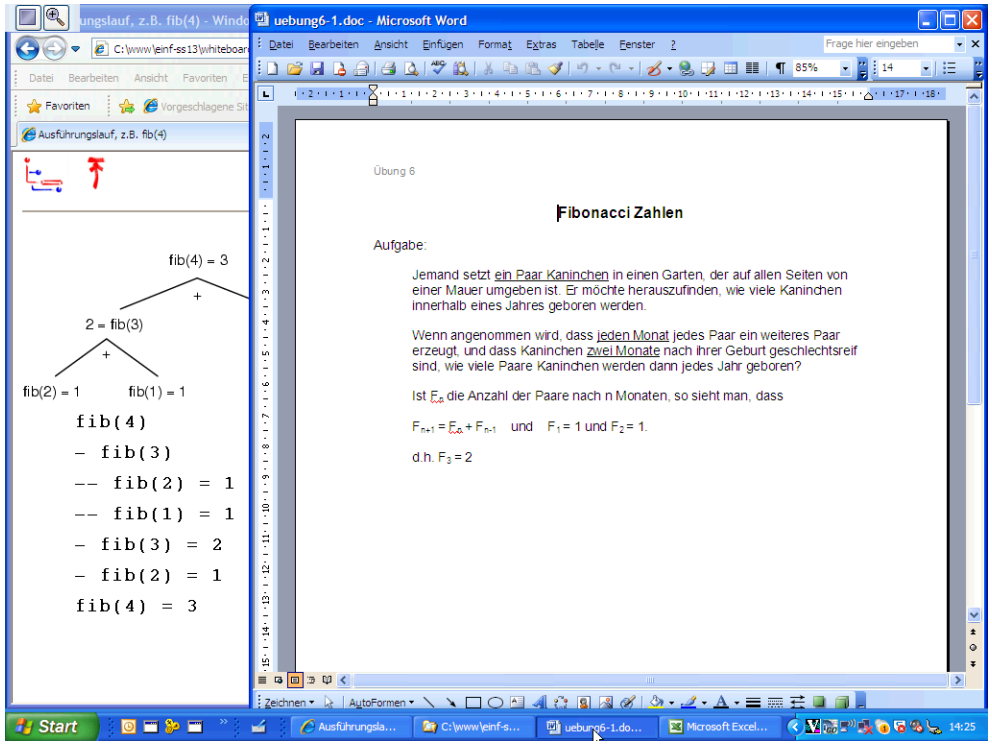
```
static int fib(int n) {
   return (n < 3) ? 1 : fib(n-1) + fib(n-2);
}
```

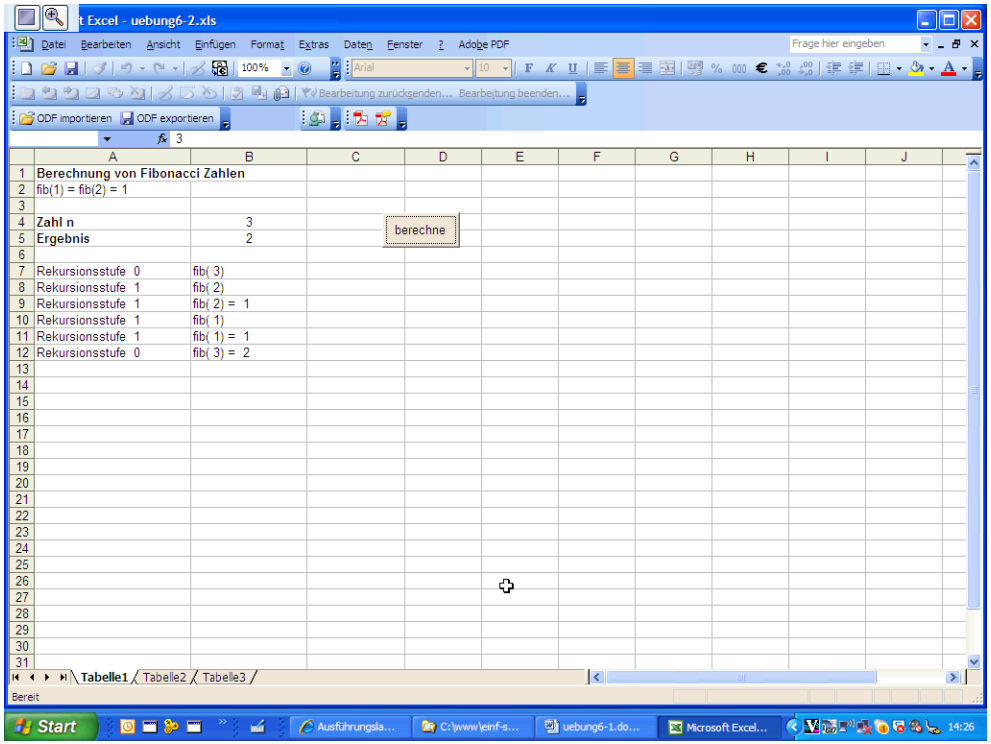
Ausführungslauf, z.B. fib(4)

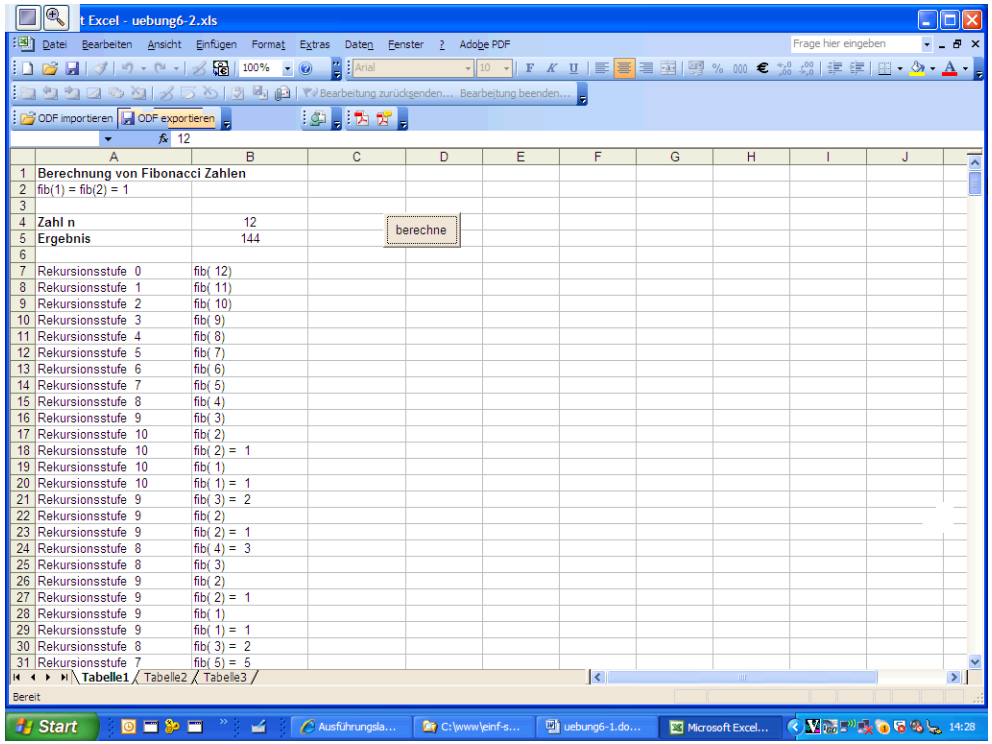


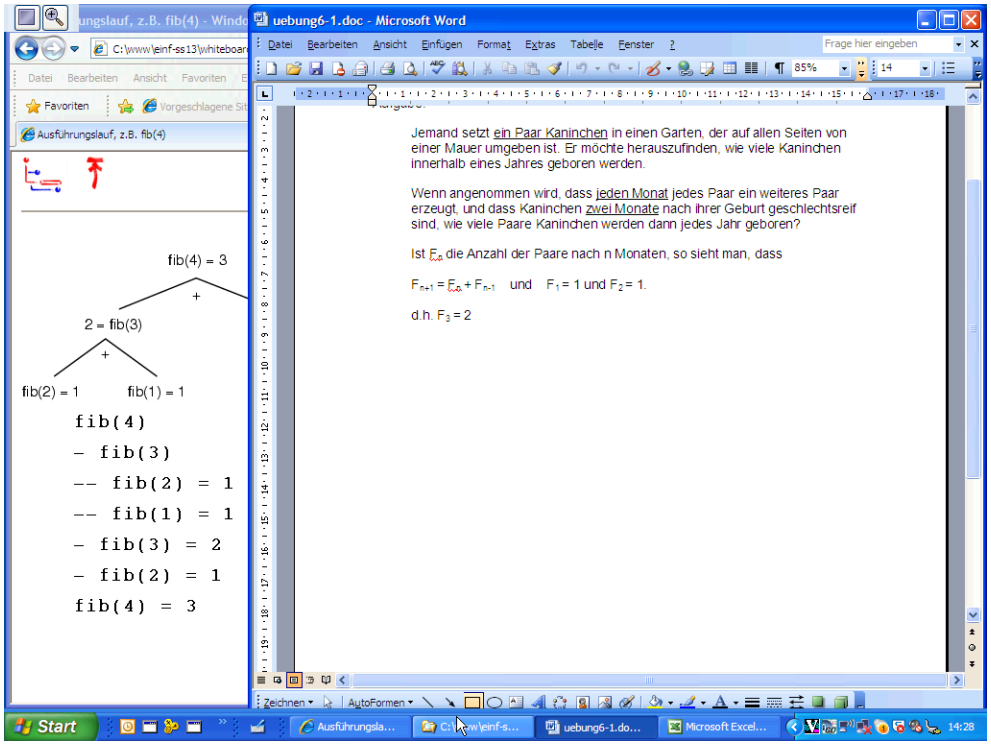

```
fib(4)
- fib(3)
-- fib(2) = 1
-- fib(1) = 1
- fib(3) = 2
- fib(2) = 1
fib(4) = 3
```

Generated by Targeteam









Spezielle Form der Modularisierung. Zu definierendes Modul wird in seiner Definition selbst benutzt.

"natürlichere" Darstellung bei bestimmten Algorithmen und Datenstrukturen.

Beispiel Summe

Definiertes Ende einer rekursiven Schachtelung.

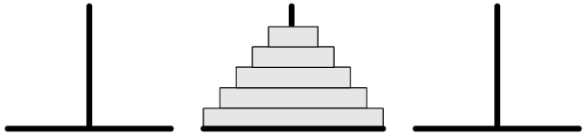
```
if (n > 0)
   summe = summe(n-1) + n;
else summe = 0; /* definiertes Ende, wenn n <= 0 */
```

Beispiel 'Fibonacci-Zahlen'

Beispiel 'Größter gemeinsamer Teiler'

Beispiel 'Türme von Hanoi'

Gegeben: drei senkrechte Stäbe, n Scheiben verschiedener Größe. Aufgabe: Turm von Scheiben von Stab auf anderen übertragen, jeweils nur eine Scheibe bewegen, nie größere Scheibe auf kleinerer.



Rekursive Lösung für n Scheiben

Aufrufschachtelung für n=3

Zahl der Schritte: $2^n - 1$; n ist die Anzahl der zu bewegenden Scheiben.

Anwendung in der Praxis

Lagerhaltungssystem des Hamburger Container-Hafens: Stapelung der Container entsprechend ihrer Abholung.

Animation Türme von Hanoi

übertrage n Scheiben von b nach a:

übertrage n-1 Scheiben von b nach c (d.h. in das Zwischenlager)

letzte Scheibe von b nach a

übertrage n-1 Scheiben von c nach a (d.h. vom Zwischenlager zum Ziel)

Algorithmus

Die Definition von solveTvH beinhaltet einen Aufruf von solveTvH (mit anderen Parametern)

```
Modul solveTvH(n, Quelle, Ziel, Zw)
  /* Zw ist das Zwischenlager */
Falls n = 1
dann bewege Scheibe von Quelle nach Ziel
sonst
  solveTvH(n-1, Quelle, Zw, Ziel);

  bewege Scheibe von Quelle nach Ziel;
  solveTvH(n-1, Zw, Ziel, Quelle);
```

Aufrufschachtelung für n=3

Aufruf solveTvH (3, b, a, c) führt zu den nachfolgenden Rekursionen:

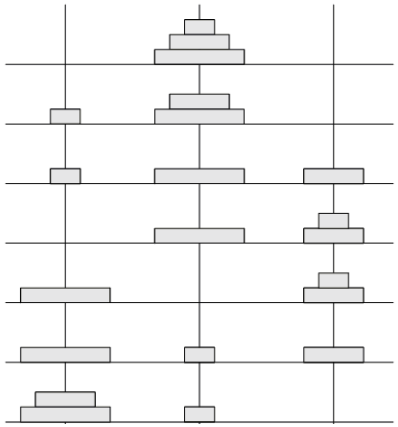
3bac (2bca (1bac (b → a); b → c; 1acb (a → c));

b → a;

2cab (1cba (c → b); c → a; 1bac (b → a)))

3bac entspricht dem Aufruf solveTvH (3, b, a, c)

b → a entspricht: bewege Scheibe von b nach a

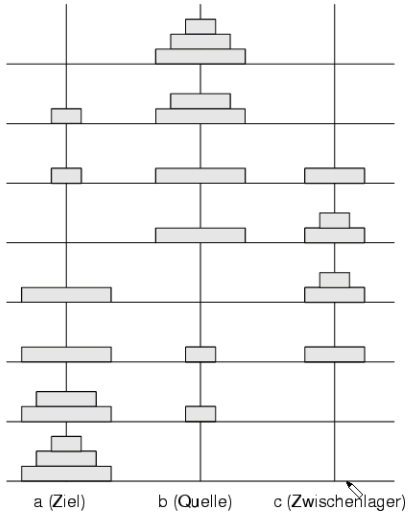


Aufrufschachtelung für n=3

2cab (1cba (c → b); c → a; 1bac (b → a)))

3bac entspricht dem Aufruf solveTvH (3, b, a, c)

b → a entspricht: bewege Scheibe von b nach a



Rekursive Lösung für n Scheiben

übertrage n Scheiben von b nach a:

übertrage n-1 Scheiben von b nach c (d.h. in das Zwischenlager)

letzte Scheibe von b nach a

übertrage n-1 Scheiben von c nach a (d.h. vom Zwischenlager zum Ziel)

Algorithmus

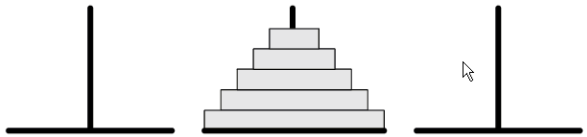
Die Definition von solveTvH beinhaltet einen Aufruf von solveTvH (mit anderen Parametern)

```
Modul solveTvH(n, Quelle, Ziel, Zw)
  /* Zw ist das Zwischenlager */
Falls n = 1
dann bewege Scheibe von Quelle nach Ziel
sonst
  solveTvH(n-1, Quelle, Zw, Ziel);

  bewege Scheibe von Quelle nach Ziel;
  solveTvH(n-1, Zw, Ziel, Quelle);
```


Beispiel 'Türme von Hanoi'

Gegeben: drei senkrechte Stäbe, n Scheiben verschiedener Größe. Aufgabe: Turm von Scheiben von Stab auf anderen übertragen, jeweils nur eine Scheibe bewegen, nie größere Scheibe auf kleinerer.



Rekursive Lösung für n Scheiben

Aufrufschachtelung für n=3

Zahl der Schritte: $2^n - 1$; n ist die Anzahl der zu bewegenden Scheiben.

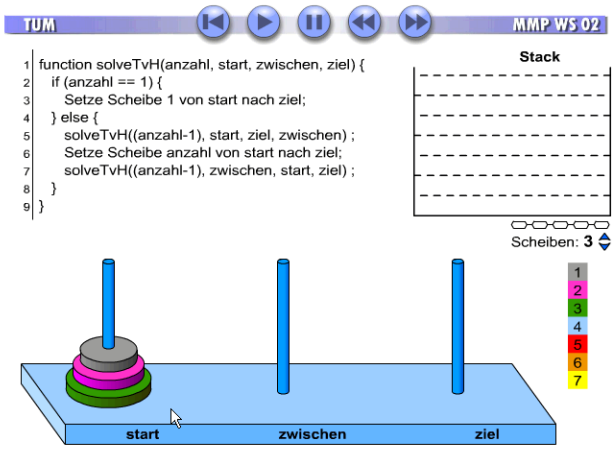
Anwendung in der Praxis

Lagerhaltungssystem des Hamburger Container-Hafens: Stapelung der Container entsprechend ihrer Abholung.

Animation Türme von Hanoi

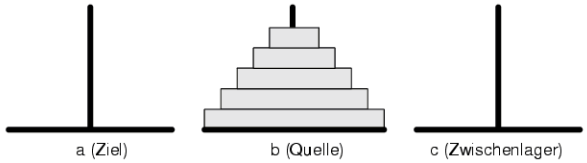
Generated by Targeteam

Animation Türme von Hanoi



Beispiel 'Türme von Hanoi'

Gegeben: drei senkrechte Stäbe, n Scheiben verschiedener Größe. Aufgabe: Turm von Scheiben von Stab auf anderen übertragen, jeweils nur eine Scheibe bewegen, nie größere Scheibe auf kleinerer.



Rekursive Lösung für n Scheiben

Aufrufschachtelung für n=3

Zahl der Schritte: 2^n -1; n ist die Anzahl der zu bewegenden Scheiben.

Anwendung in der Praxis

Lagerhaltungssystem des Hamburger Container-Hafens: Stapelung der Container entsprechend ihrer Abholung.

Animation Türme von Hanoi

Grundlagen der Programmierung

"Kunst des Programmierens". Grundlagen zu Datenstrukturen, Programmkonstrukte, Strukturierung von Programmen, objekt-orientierte Programmierung.

- Fragestellungen des Abschnitts:
  - Was ist ein Algorithmus?
  - Welche elementaren Datenstrukturen gibt es?
  - Was sind die grundlegenden Konstrukte einer Programmiersprache?
  - Was ist unter Objekt-orientierter Programmierung zu verstehen?
  - Was versteht man unter Modularisierung und Rekursion?

Einführung

Algorithmus

Datentypen und Ausdrücke

Programmkonstrukte

Objektorientierte Programmierung

Modularisierung von Programmen

Rekursion

Datenstrukturen und Algorithmen

In diesem Kapitel werden einige Klassen von Algorithmen vorgestellt, insbesondere Suchverfahren und Sortierverfahren.

- Fragestellungen des Abschnitts:
  - Welche Möglichkeiten gibt es, Datenmengen im System darzustellen?
  - Welche Möglichkeiten gibt es, in Datenmengen zu suchen?
  - Welche Möglichkeiten gibt es, Datenmengen zu sortieren?
  - Was versteht man unter der Komplexität eines Algorithmus?

Datenstrukturen

Suchverfahren

Sortierverfahren

Komplexität

Datenstruktur = Menge von Daten eines bestimmten Typs zusammen mit den auf der Menge ausführbaren Zugriffsoperationen.

Beispiel: natürliche Zahlen zusammen mit Grundrechenoperationen

Bei Programmierung wichtiges Hilfsmittel zur Organisation der Daten für die maschinelle Verarbeitung.

Listen

Queue - Warteschlange

Eine weitere grundlegende Datenstruktur sind Queues, die mit Warteschlangen vergleichbar sind (FIFO-Prinzip, first-in-first-out).

`put()` : fügt ein Element am Ende der Warteschlange hinzu

`get()` : entnimmt ein Element am Anfang der Warteschlange und liefert es zurück

Realisierung als verkettete Liste oder als Array (falls maximale Größe bekannt).

Graphen

Datenstrukturen

Datenstruktur = Menge von Daten eines bestimmten Typs zusammen mit den auf der Menge ausführbaren Zugriffsoperationen.

Beispiel: natürliche Zahlen zusammen mit Grundrechenoperationen

Bei Programmierung wichtiges Hilfsmittel zur Organisation der Daten für die maschinelle Verarbeitung.

Listen

Queue - Warteschlange

Eine weitere grundlegende Datenstruktur sind Queues, die mit Warteschlangen vergleichbar sind (FIFO-Prinzip, first-in-first-out).

`put()` : fügt ein Element am Ende der Warteschlange hinzu

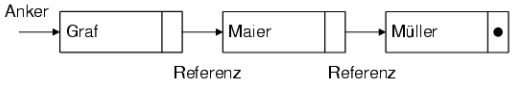
`get()` : entnimmt ein Element am Anfang der Warteschlange und liefert es zurück

Realisierung als verkettete Liste oder als Array (falls maximale Größe bekannt).

Graphen

Generated by Targeteam

Folge von Elementen mit Reihenfolge.



Anker definiert Verweis auf 1. Element.

Referenz definiert Verweis auf nächstes Element.

Beispiele

Wort: Liste von Zeichen.

Personaldaten: Liste von Datensätzen, wobei jeder Datensatz eine Person beschreibt.

Operationen auf Listen

Erzeugen einer Liste, Einfügen eines Listenelements, Suchen eines Listenelements mit bestimmten Eigenschaften, Löschen eines Listenelements.

Reihungen (Array)

Allgemeine Listen

Liste mit fester Zahl von Elementen gleichen Typs, gemäß Reihenfolge numeriert.

Reihungen meist in Programmiersprachen direkt unterstützt.

Deklaration durch:

```
boolean buffer[500]
```

einfach verkettete Listen

Listenelement: Inhalt und Referenz auf nächstes Listenelement.

doppelt verkettete Listen

Listenelement: Inhalt und Referenzen auf nächstes und vorhergehendes Listenelement.

Beispiel

```
class elem {
    elem previous;
    char content;
    elem next;
}
```


Datenstruktur = Menge von Daten eines bestimmten Typs zusammen mit den auf der Menge ausführbaren Zugriffsoperationen.

Beispiel: natürliche Zahlen zusammen mit Grundrechenoperationen

Bei Programmierung wichtiges Hilfsmittel zur Organisation der Daten für die maschinelle Verarbeitung.

Listen

Queue - Warteschlange

Eine weitere grundlegende Datenstruktur sind Queues, die mit Warteschlangen vergleichbar sind (FIFO-Prinzip, first-in-first-out).

`put()` : fügt ein Element am Ende der Warteschlange hinzu

`get()` : entnimmt ein Element am Anfang der Warteschlange und liefert es zurück

Realisierung als verkettete Liste oder als Array (falls maximale Größe bekannt).

Graphen

Modell für Beschreibung von Objekten, die in Beziehung zueinander stehen. Menge K von Knoten (Ecken, Punkten) und Menge V von Verbindungen (Kanten). Ggf. Namen und Werte. Graph G = (K, V).

gerichteter Graph

jede Kante hat Richtung.

Weg

Folge x = K_1 , K_2 , ... K_n = y, wobei jeweils zwischen den K_1 nach K_2 , K_2 nach K_3 , ... eine Kante besteht.

Beispiel

Straßenkarte wird durch einen Graph dargestellt:

Knoten = Städte

Kanten = Straßen zwischen den Städten

Nutzung des Graphen zur Bestimmung des kürzesten Weges zwischen 2 Städten.

Baum

spezieller Graph mit den folgenden Eigenschaften

Wurzel : ausgezeichneter Knoten, der den Ursprung des Baums kennzeichnet.

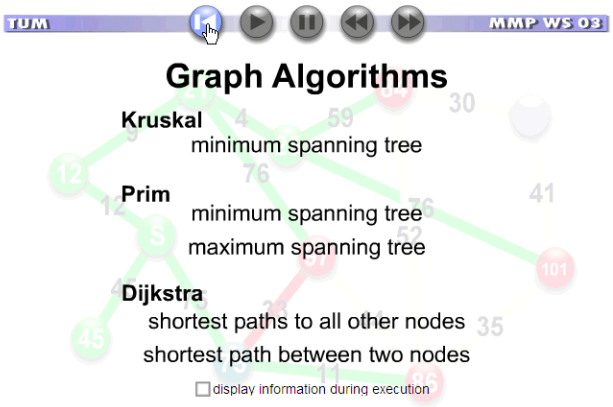
keine Zyklen.

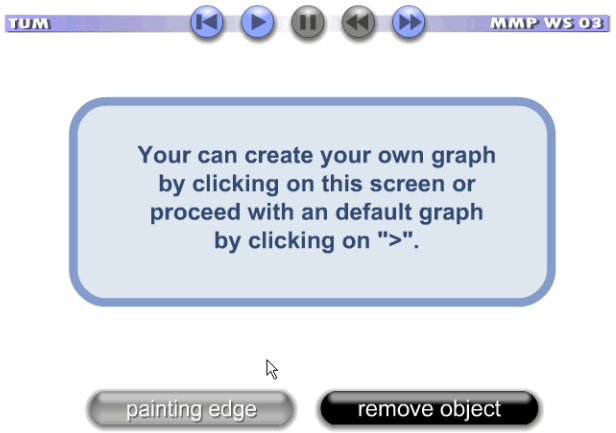
jeder Knoten (außer der Wurzel) hat genau einen Vorgänger.

Animation Graph Algorithmen

Your can create your own graph by clicking on this screen or proceed with an default graph by clicking on ">".

painting edge | remove object





In diesem Kapitel werden einige Klassen von Algorithmen vorgestellt, insbesondere Suchverfahren und Sortierverfahren.

- Fragestellungen des Abschnitts:
  - Welche Möglichkeiten gibt es, Datenmengen im System darzustellen?
  - Welche Möglichkeiten gibt es, in Datenmengen zu suchen?
  - Welche Möglichkeiten gibt es, Datenmengen zu sortieren?
  - Was versteht man unter der Komplexität eines Algorithmus?

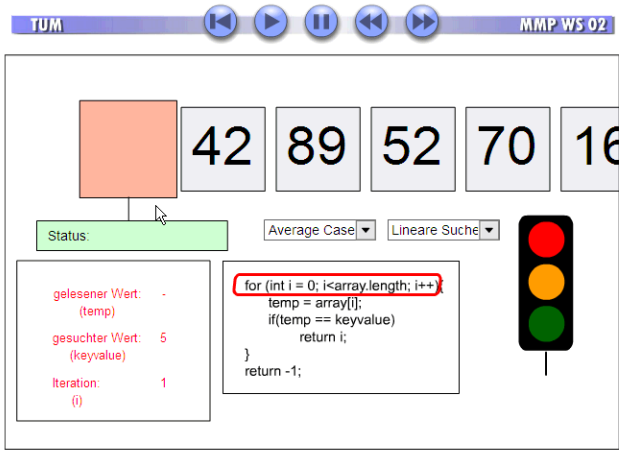
Voraussetzung: Ordnung auf Datenelementen. Entsprechend der Ordnung in der Reihung gespeichert (aufsteigend oder absteigend).

Sei int A[n] eine Reihung mit n Elementen, das aufsteigend sortiert ist.

Gesucht wird ein Element mit dem Wert x; gesucht wird x im Bereich A[0] bis A[n-1].

1. wähle m zwischen 0 und n-1; man wird m ungefähr in der Mitte zwischen 0 und n-1 wählen.
2. wenn A[m] == x, dann sind wir fertig, gib m als Ergebnis aus.
3. wenn x < A[m], dann suche weiter im Bereich A[0] bis A[m-1];
4. wenn x > A[m], dann suche weiter im Bereich A[m+1] bis A[n-1]

Doppelte Zahl von Elementen: ein zusätzlicher Vergleich. Bei linearer Suche: Verdopplung des Aufwandes.



Gegeben: Menge von Datensätzen. Gesucht: Datensatz mit bestimmter Eigenschaft.

Mengen von Datensätzen

Üblicherweise in Reihung oder Liste gespeichert.

Annahme für die folgenden Verfahren: Daten in einer Reihung gespeichert.

Lineare Suche

Binäre Suche

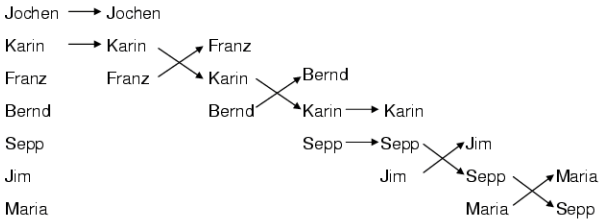
Suchverfahren Animation

Sortieren durch Vertauschen von benachbarten Elementpaaren.

Mehrere Durchgänge. Bei n Elementen maximal n Durchgänge.

Beispiel: alphabetisches Sortieren einer Menge von Namen

1. Sortierdurchgang mit jeweils paarweisen Vergleich

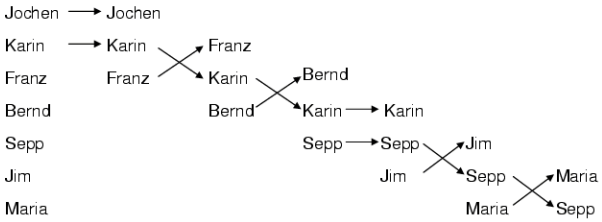


Original	nach 1	nach 2	nach 3	nach 4 Schritten
Jochen	Jochen	Franz	Bernd	Bernd
Karin	Franz	Bernd	Franz	Franz
Franz	Bernd	Jochen	Jim	Jim
Bernd	Karin	Jim	Jochen	Jochen
Sepp	Jim	Karin	Karin	Karin
Jim	Maria	Maria	Maria	Maria
Maria	Sepp	Sepp	Sepp	Sepp

Mehrere Durchgänge. Bei n Elementen maximal n Durchgänge.

Beispiel: alphabetisches Sortieren einer Menge von Namen

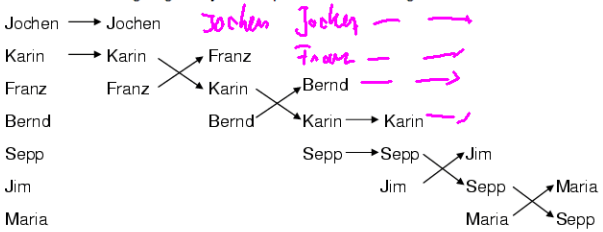
1. Sortierdurchgang mit jeweils paarweisen Vergleich



Original	nach 1	nach 2	nach 3	nach 4 Schritten
Jochen	Jochen	Franz	Bernd	Bernd
Karin	Franz	Bernd	Franz	Franz
Franz	Bernd	Jochen	Jim	Jim
Bernd	Karin	Jim	Jochen	Jochen
Sepp	Jim	Karin	Karin	Karin
Jim	Maria	Maria	Maria	Maria
Maria	Sepp	Sepp	Sepp	Sepp

Beispiel: alphabetisches Sortieren einer Menge von Namen

1. Sortierdurchgang mit jeweils paarweisen Vergleich

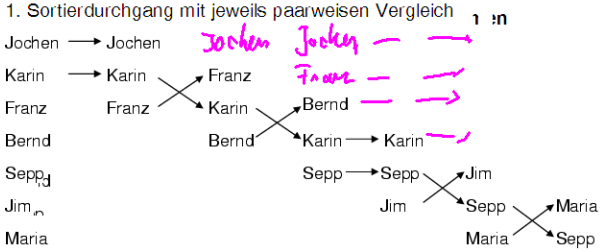


Original	nach 1	nach 2	nach 3	nach 4 Schritten
Jochen	Jochen	Franz	Bernd	Bernd
Karin	Franz	Bernd	Franz	Franz
Franz	Bernd	Jochen	Jim	Jim
Bernd	Karin	Jim	Jochen	Jochen
Sepp	Jim	Karin	Karin	Karin
Jim	Maria	Maria	Maria	Maria
Maria	Sepp	Sepp	Sepp	Sepp

Mehrere Durchgänge. Bei n Elementen maximal n Durchgänge.

Beispiel: alphabetisches Sortieren einer Menge von Namen

1. Sortierdurchgang mit jeweils paarweisen Vergleich



Original	nach 1	nach 2	nach 3	nach 4 Schritten
Jochen	Jochen	Franz	Bernd	Bernd
Karin	Franz	Bernd	Franz	Franz
Franz	Bernd	Jochen	Jim	Jim
Bernd	Karin	Jim	Jochen	Jochen
Sepp	Jim	Karin	Karin	Karin
Jim	Maria	Maria	Maria	Maria
Maria	Sepp	Sepp	Sepp	Sepp

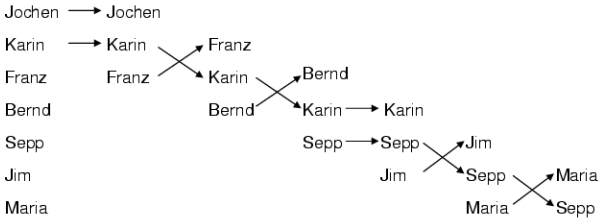
Ziel: Datenmenge entsprechend vorgegebenen Ordnungskriterium sortieren.

Sortieren durch Vertauschen von benachbarten Elementpaaren.

Mehrere Durchgänge. Bei n Elementen maximal n Durchgänge.

Beispiel: alphabetisches Sortieren einer Menge von Namen

1. Sortierdurchgang mit jeweils paarweisen Vergleich



Original	nach 1	nach 2	nach 3	nach 4 Schritten
Jochen	Jochen	Franz	Bernd	Bernd
Karin	Franz	Bernd	Franz	Franz
Franz	Bernd	Jochen	Jim	Jim
Bernd	Karin	Jim	Jochen	Jochen
Sepp	Jim	Karin	Karin	Karin
Jim	Maria	Maria	Maria	Maria
Maria	Sepp	Sepp	Sepp	Sepp

Sortierverfahren

Ziel: Datenmenge entsprechend vorgegebenen Ordnungskriterium sortieren.

Sortierverfahren - Beispiel Bubble Sort

Sortierverfahren - Beispiel Insert-Sort

Rekursives Sortierverfahren - Beispiel Quicksort

Sortierverfahren Animation

Beispiel

Karin	Jim	Bernd	Bernd	Bernd
Franz	Franz	Franz	Franz	Franz
Bernd	Bernd	Jim	Jim	Jim
Jochen	Jochen	Jochen	Jochen	Jochen
Sepp	Sepp	Sepp	Karin	Karin
Jim	Karin	Karin	Sepp	Maria
Maria	Maria	Maria	Maria	Sepp

Sortierverfahren Animation