Datenstrukturen

Programmierung in R

Referent: Dominik Scheinert

Betreuer: Julian Kunkel

Agenda

- Motivation
- Übersicht: Datenstrukturen
- Datenstrukturen in R
- Zusammenfassung

Daten

„Daten sind in erkennungsfähiger Form dargestellte Elemente einer Information, die in Systemen verarbeitet werden können.“

- ITWissen [Zi01]

- Zentrales Thema in der Informatik
- Zeichenketten, für die Abmachungen gelten
- Ziel: Bearbeitungseffizienz, Speicher- / Recheneffizienz

Abbildung 01
Quelle: „Jason Schneider“ [Ab01]

Datenstrukturen

„Ein abstrakter Datentyp (ADT) ist ein Verbund von Daten zusammen mit der Definition aller zulässigen Operationen, die auf sie zugreifen. […] Ein ADT beschreibt, was die Operationen tun, aber noch nicht, wie sie es tun sollen.“

- Wikipedia [Zi02]

Als Datenstruktur bezeichnet man das Konstrukt in einem Programm (bzw. Im Speicher […]), das Daten auf eine gewisse Weise speichert.

- Computerlexikon [Zi03]

Abbildung 02
Quelle: „Brigitte Schneider“ [Ab02]

Komplexitätskennzahlen

Datenstruktur	Einfügen	Löschen	Suchen
Array	O(n)	O(n)	O(n)
Liste	O(1)	O(1)	O(n)
Stack	O(1)	O(1)	O(n)
Baum	O(log n)	O(log n)	O(log n)

Welche Datenstrukturen können wir in R nutzen?

- *Integriert*
  - Vektoren
  - Matrizen
  - Faktoren
  - Data Frames
  - Arrays*
  - Listen*
- *Pakete*
  - Stacks*
  - Queues*
  - Bäume*
  - ...

* *Auch außerhalb von R bekannte Datenstrukturen, die deshalb nochmal im Allgemeinen kurz vorgestellt werden.*

Vektoren

- Array von Elementen eines Datentyps
- Eindimensional
- Zugriff auf einzelne Elemente durch Index

```
> a = vector()
> b = vector(length = 4)
> length(b)
[1] 4
> c = vector("numeric", length = 10)
> length(c)
[1] 10
> class(c)
[1] "numeric"
> d = c(1,2,3)
> d
[1] 1 2 3
```

Vektoren

```
> # Zugriff auf Elemente
> d[2]
[1] 2
> d[1:3]
[1] 1 2 3
```

```
> a = c(1L,2.2,TRUE,'h')
> a
[1] "1"    "2.2"  "TRUE" "h"
> a = c(1L,2.2,TRUE)
> a
[1] 1.0 2.2 1.0
> a = c(1L,TRUE)
> a
[1] 1 1
```

Bei mehreren Datentypen in einem Vektor werden alle Elemente in den flexibelsten Datentyp umgewandelt.
logical → integer → numeric → complex → character

Vektoren

- Datentyp: Numeric
  - Geordnete Sammlung von Zahlen
  - Erlauben arithmetische Operationen

```
> num = c(1,4,2,9,7)
> num
[1] 1 4 2 9 7
```

```
> c(1,2,3) * c(4,5,6)
[1]  4 10 18
> max(c(1,10,100))
[1] 100
```

Vektoren

- Datentyp: Logical
  - Werte: TRUE, FALSE
  - Generierung u.a. durch Konditionen

```
> logical = c(TRUE,FALSE)
> class(logical)
[1] "logical"
```

```
> values = c(40:45)
> logical <- values > 41 & values < 44
> logical
[1] FALSE FALSE  TRUE  TRUE FALSE FALSE
```

Vektoren – weitere Datentypen

- integer
- character
- complex

```
> character = c("hallo","test")
> character
[1] "hallo" "test"
> class(character)
[1] "character"
```

```
> integer = c(1L,2L,3L)
> integer
[1] 1 2 3
> class(integer)
[1] "integer"
```

```
> complex = c(2+3i,4-2i)
> complex
[1] 2+3i 4-2i
> class(complex)
[1] "complex"
```

Matrizen

- Spezielle Vektoren
- Elemente besitzen den gleichen Datentyp
- Zweidimensional
- Zugriff auf einzelne Elemente durch Index

```
> m = matrix(nrow = 2, ncol = 2)
> dim(m)
[1] 2 2
> m
     [,1] [,2]
[1,]   NA   NA
[2,]   NA   NA
> m[2,2] = 4
> m
     [,1] [,2]
[1,]   NA   NA
[2,]   NA    4
```

Matrizen

```
> n = matrix(1:9,nrow = 3 , ncol = 3)
> n
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Spaltenbefüllung

```
> t(n)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Transponieren

```
> s = matrix(10:12, nrow = 3, ncol = 1)
> s
     [,1]
[1,]   10
[2,]   11
[3,]   12
> cbind(n,s)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Spaltenkombination

```
> s = matrix(10:12, nrow = 1, ncol = 3)
> rbind(n,s)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
[4,]   10   11   12
```

Zeilenkombination

Faktoren

- Geeignet, um Daten zu kategorisieren
- Eingabevektor vom Typ *character* oder *numeric*
- Kategorien werden intern effizient zu Integern abgebildet

```
> fac = factor(c("blue","green","yellow","green","blue"))
> fac
[1] blue   green  yellow green  blue
Levels: blue green yellow
```

Faktoren

```
> fac = factor(sample(letters, size = 5, replace = TRUE))
> fac
[1] h t i n z
Levels: h i n t z
> fac = factor(sample(letters, size = 5, replace = TRUE), ordered = TRUE)
> fac
[1] f a y q r
Levels: a < f < q < r < y
```

ordered vs. unordered factors

```
> fac = factor(sample(5, size = 5, replace = TRUE),levels = c(1), ordered = TRUE)
> fac
[1] <NA> 1    <NA> <NA> 1
Levels: 1
```

Levelmodifikation

```
> fac = factor(sample(10, size = 10000, replace = TRUE), ordered = TRUE)
> table(fac)
fac
   1    2    3    4    5    6    7    8    9   10
1029 1035 1007 1038  990  974  930 1002  987 1008
```

Table-Funktion

Faktoren

Der Nutzen von Faktoren am Beispiel der tapply-Funktion

Tapply-Funktion: Wendet eine Funktion auf alle durch einen bzw. mehrere Faktoren definierten Untergruppen an

```
> city = factor(c("hh","nrw","hh","mv","hh","nrw","hh","th","sh"))
> dept = c(20000,5000,10000,2000,37000,250,7500,3000,150)
> deptsum <- tapply(dept, city, sum)
> deptsum
   hh    mv   nrw    sh    th
74500  2000  5250   150  3000
```

Data Frames

- Sammlung von Vektoren gleicher Länge
- zweidimensional
- Wie Matrix, erlaubt jedoch verschiedene Datentypen
- Vergleichbar zu Tabelle mit benannten Spalten und Zeilen
- Geeignet, um einen ganzen Datensatz darzustellen

```
> integer = c(1L,2L,3L,4L)
> character = c("a","b","c","d")
> logical = c(TRUE,FALSE,TRUE,FALSE)
> df = data.frame(integer,character,logical)
> df
  integer character logical
1       1         a    TRUE
2       2         b   FALSE
3       3         c    TRUE
4       4         d   FALSE
```

Erzeugung

Data Frames

```
> colnames(df)
[1] "integer"   "character" "logical"
> colnames(df) <- c("ich","bin","test")
> df
  ich bin  test
1   1   a  TRUE
2   2   b FALSE
3   3   c  TRUE
4   4   d FALSE
```

```
> rownames(df)
[1] "1" "2" "3" "4"
> rownames(df) <- c("w","x","y","z")
> df
  ich bin  test
w   1   a  TRUE
x   2   b FALSE
y   3   c  TRUE
z   4   d FALSE
```

```
> cbind(df, test2 = c(5,6))
  ich bin  test test2
1   1   a  TRUE     5
2   2   b FALSE     6
3   3   c  TRUE     5
4   4   d FALSE     6
```

Data Frames

```
> df[,1]
[1] 1 2 3
> df[,1]= c(7,8,9)
> df
  integer character logical complex
1       7         a    TRUE    2+2i
2       8         b   FALSE    4-3i
3       9         c    TRUE    1+1i
```

Spalten adressieren und modifizieren

```
> df[1,]
  integer character logical complex
1      10         a   FALSE    3-3i
> df[1,]= list(10,"a",FALSE,3-3i)
> df
  integer character logical complex
1      10         a   FALSE    3-3i
2       8         b   FALSE    4-3i
3       9         c    TRUE    1+1i
```

Zeilen adressieren und modifizieren

Data Frames - *Anwendungsbeispiel*

```
> cars
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Arrays

- Zusammenhängende Sequenz von Elementen
- Zugriff über Indizes
- Elemente haben gleichen Datentyp

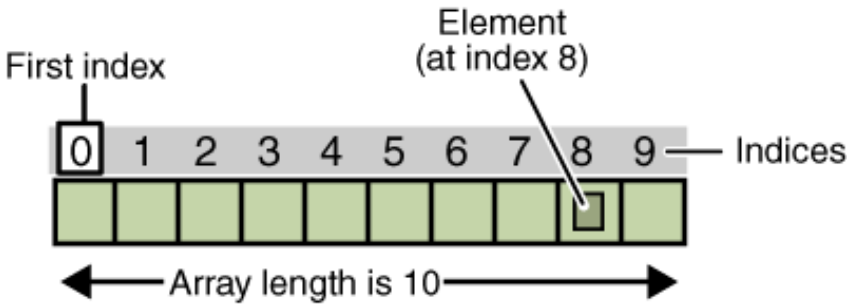


Abbildung 03
Quelle: „Javin Paul“ [Ab03]

Arrays

```
> array = array(1:9, dim = c(3,3,1))
> array
, , 1

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Matrizen als zweidimensionale Arrays

```
> array = array(1:27, dim = c(3,3,3))
> array
, , 1

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

, , 2

     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18

, , 3

     [,1] [,2] [,3]
[1,]   19   22   25
[2,]   20   23   26
[3,]   21   24   27
```

dreidimensionales Array

Listen

- Lineare Datenstruktur
- Zeiger auf Nachbarobjekt
- Vorteil beim Löschen und Einfügen neuer Elemente

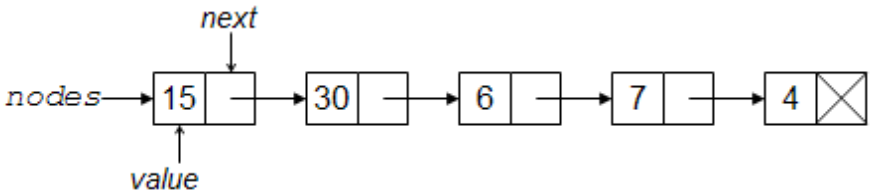


Abbildung 04
Quelle: „Wikibooks: Data Structures“ [Ab04]

Listen

```
> list[[1]]
[1] 1 2 3 4
> list[[2]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Listenzugriff

```
> list[[1]][1] = 4
> list[[1]]
[1] 4 2 3 4
> list[[2]][1,1] = 10
> list[[2]]
     [,1] [,2]
[1,]   10    3
[2,]    2    4
```

Modifikation von Inhalten

```
> a = c(1,2,3,4)
> b = matrix(1:4 , nrow = 2 , ncol = 2)
> c = array(1:8, dim = c(2,2,2))
> list = list(a,b,c,10)
> list
[[1]]
[1] 1 2 3 4

[[2]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4

[[3]]
, , 1

     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8


[[4]]
[1] 10
```

Listenerzeugung

Übersicht

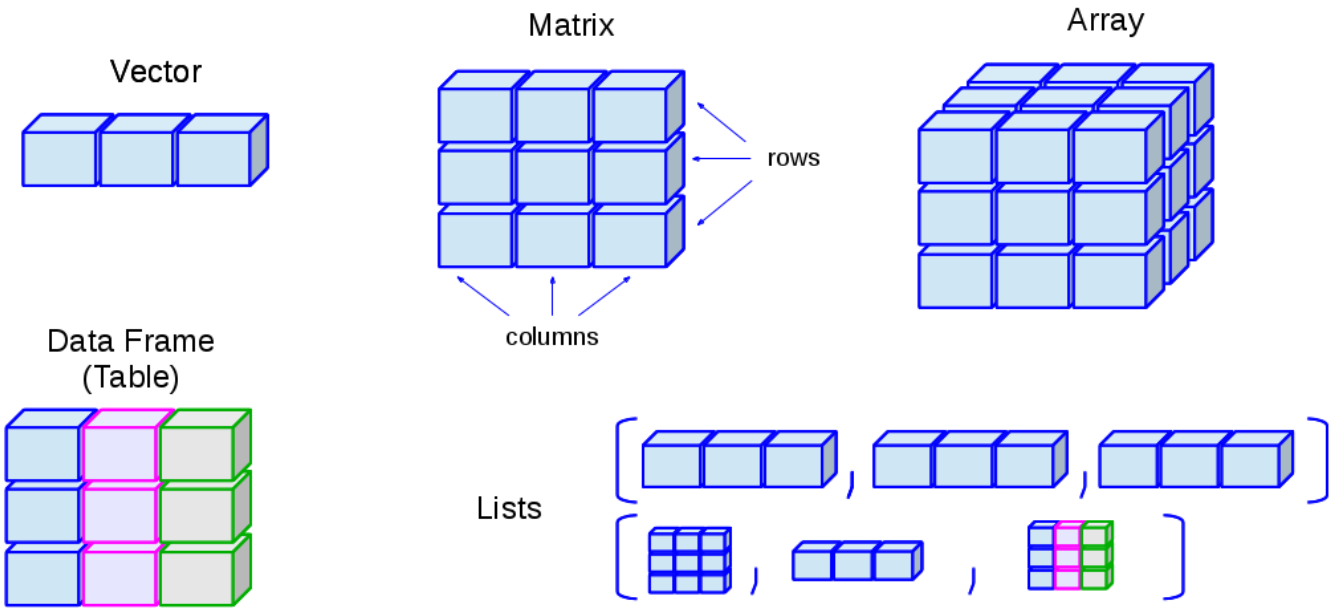


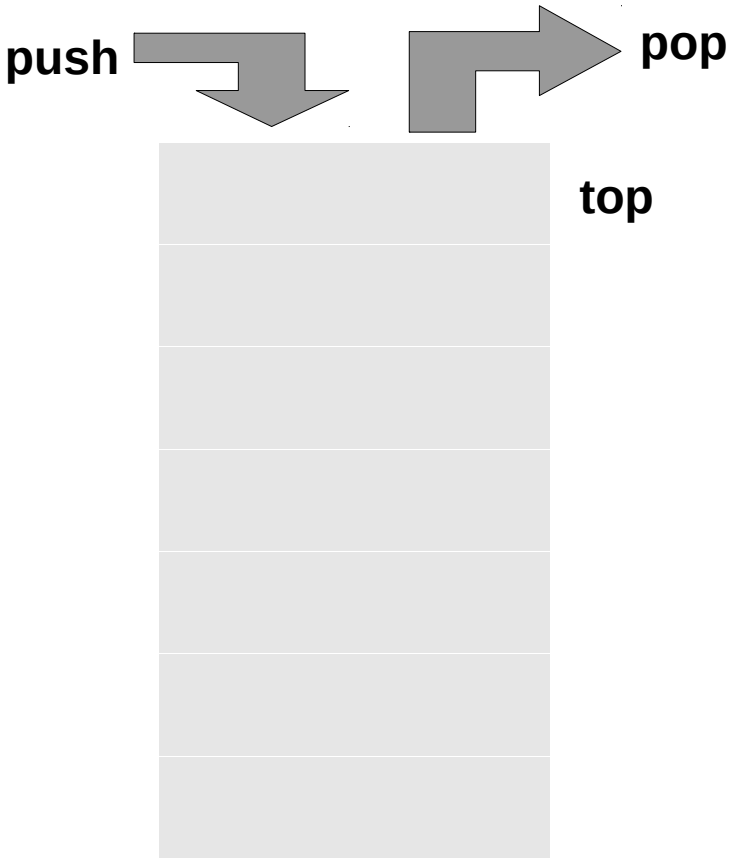
Abbildung 05
Quelle: „Maite Ceballos & Nicolas Cardiel“ [Ab05]

Pakete

Stacks

- LIFO-Prinzip
- Implementation z.B. mit

  Liste



Queues

- FIFO-Prinzip
- Implementation z.B. mit verketteter Liste

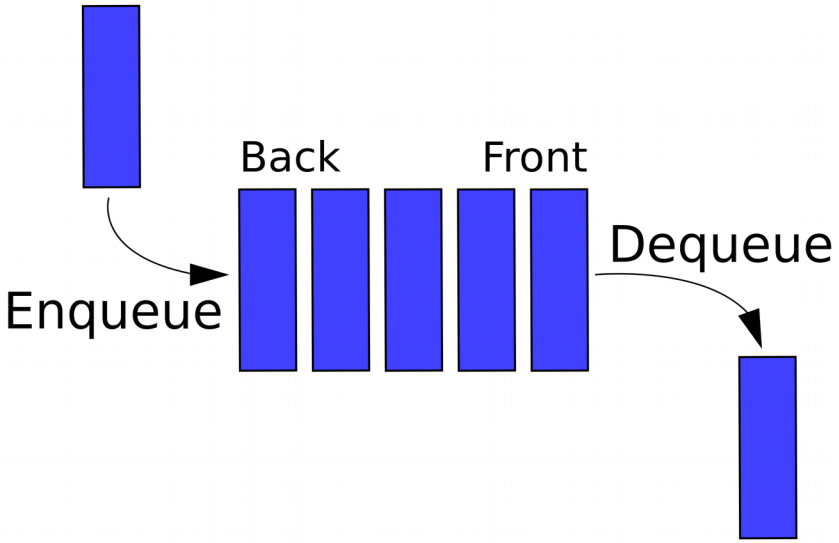


Abbildung 06
Quelle: „Wikipedia: Queue“ [Ab06]

Queues und Stacks – Paket *rstackdeque*

```
> stack <- rstack()
> stack <- insert_top(stack , 1)
> stack <- insert_top(stack , 2)
> stack <- insert_top(stack , "test")
> stack
An rstack with  3  elements.
 : chr "test"
 : num 2
 : num 1
> stack <- without_top(stack)
> stack
An rstack with  2  elements.
 : num 2
 : num 1
> top <- peek_top(stack)
> stack
An rstack with  2  elements.
 : num 2
 : num 1
> top
[1] 2
```

Stack

```
> queue = rpqueue()
> queue <- insert_back(queue, "a")
> queue <- insert_back(queue, "b")
> queue <- insert_back(queue, "c")
> queue
A queue with  3  elements.
Front:
 $: chr "a"
 $: chr "b"
 $: chr "c"
> queue <- without_front(queue)
> queue
A queue with  2  elements.
Front:
 $: chr "b"
 $: chr "c"
> front <- peek_front(queue)
> queue
A queue with  2  elements.
Front:
 $: chr "b"
 $: chr "c"
> front
[1] "b"
```

Queue

Bäume

- Geeignet, um hierarchische Strukturen abzubilden
- Wichtige Datenstruktur
- Verschiedene Ausprägungen
- Implementation als verkettete Liste

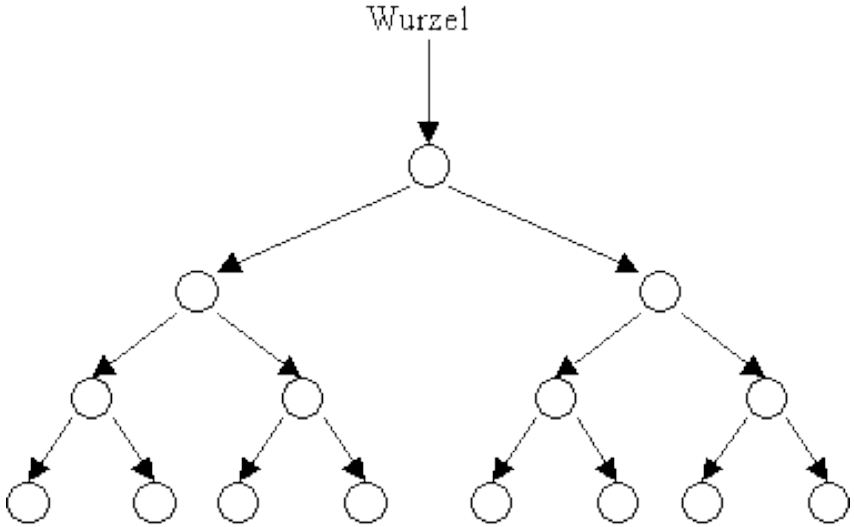


Abbildung 07
Quelle: „Jürgen Dehmer“ [Ab07]

Bäume – Paket *data.tree*

```
> fbinformatik <- Node$new("Fachbereich Informatik")
> bachelor <- fbinformatik$AddChild("Bachelorstudiengänge")
> master <- fbinformatik$AddChild("Masterstudiengänge")
> sse <- bachelor$AddChild("Software-System-Entwicklung")
> informatik <- bachelor$AddChild("Informatik")
> mci <- bachelor$AddChild("Mensch-Computer-Interaktion")
> informatik2 <- master$AddChild("Informatik")
> print(fbinformatik)
                            levelName
1 Fachbereich Informatik
2  ¦--Bachelorstudiengänge
3  ¦   ¦--Software-System-Entwicklung
4  ¦   ¦--Informatik
5  ¦   °--Mensch-Computer-Interaktion
6  °--Masterstudiengänge
7      °--Informatik
```

Baum aufbauen

Bäume – Paket *data.tree*

Attribute

```
> sse$students <- 300
> informatik$students <- 400
> mci$students <- 100
>
> print(fbinformatik, "students")
                                levelName students
1 Fachbereich Informatik                       NA
2  ¦--Bachelorstudiengänge                     NA
3  ¦   ¦--Software-System-Entwicklung         300
4  ¦   ¦--Informatik                          400
5  ¦   °--Mensch-Computer-Interaktion         100
6  °--Masterstudiengänge                       NA
7      °--Informatik                           NA
```

Bäume – Paket *data.tree*

```
> bachelor$students <- Aggregate(node = bachelor, attribute = "students", aggFun = sum)
> print(fbinformatik, "students")
                            levelName students
1 Fachbereich Informatik                   NA
2  ¦--Bachelorstudiengänge                800
3  ¦   ¦--Software-System-Entwicklung     300
4  ¦   ¦--Informatik                      400
5  ¦   °--Mensch-Computer-Interaktion     100
6  °--Masterstudiengänge                   NA
7      °--Informatik                       NA
```

Beispiel: Sum-Funktion

Zusammenfassung

- Wichtige Datentypen
  - Vektoren
  - Data Frames
- Benutzung
  - Funktionsargumente setzen oft bestimmten Datentyp voraus
- Zusätzliche Pakete stellen weitere Datentypen bereit

Quellenverzeichnis

- **„CRAN: An Introduction to R“** https://cran.r-project.org/doc/manuals/r-release/R-intro.html , 19.04.2016
- „Christoph Glur: Data.tree“ https://cran.r-project.org/web/packages/data.tree/vignettes/data.tree.html , 19.04.2016
- „Shawn T. O'Neil: rstackdeque“ https://cran.r-project.org/web/packages/rstackdeque/rstackdeque.pdf , 19.04.2016
- „Mike Malecki: Stack“ https://cran.r-project.org/web/packages/Stack/Stack.pdf , 19.04.2016
- „Kelly Black: R Tutorial“ http://www.cyclismo.org/tutorial/R/basicOps.html , 19.04.2016
- „Chi Yau: Matrix Construction“ http://www.r-tutor.com/r-introduction/matrix/matrix-construction , 19.04.2016
- „Chi Yau: Data Frame“ http://www.r-tutor.com/r-introduction/data-frame , 19.04.2016
- „IDRE: Factor variables“ http://www.ats.ucla.edu/stat/r/modules/factor_variables.htm , 19.04.2016
- „Software Carpentry: Understanding basic data types in R“ http://nicercode.github.io/2014-02-13-UNSW/lessons/01-intro_r/data-structures.html , 19.04.2016

Zitate

- Zi01: „ITWissen: Daten“ http://www.itwissen.info/definition/lexikon/Daten-data.html , 20.04.2016
- Zi02: „Wikipedia: Abstrakter Datentyp“ https://de.wikipedia.org/wiki/Abstrakter_Datentyp , 19.04.2016
- Zi03: „Computerlexikon: Datenstruktur“ http://www.computerlexikon.com/was-ist-datenstruktur , 19.04.2016

Abbildungsverzeichnis

- Ab01: „clarabridge.com , Jason Schneider“ http://www.clarabridge.com/wp-content/uploads/2014/04/DATA.jpg, 19.04.2016
- Ab02: „business-netz.com , Brigitte Miller“ http://www.business-netz.com/Kommunikation/PowerPoint-Gliederung, 19.04.2016
- Ab03: „java67.blogspot.de , Javin Paul“ http://java67.blogspot.de/2014/05/how-to-compare-two-arrays-in-java-string-int-example.html, 19.04.2016
- Ab04: „Wikibooks.org: Data Structures“ https://en.wikibooks.org/wiki/Data_Structures/All_Chapters , 19.04.2016
- Ab05: „venus.ifca.unican.es , Maite Ceballos & Nicolás Cardiel“ http://venus.ifca.unican.es/Rintro/_images/dataStructuresNew.png , 19.04.2016
- Ab06: „Wikimedia.org: Data Queue“ https://upload.wikimedia.org/wikipedia/commons/thumb/5/52/Data_Queue.svg/2000px-Data_Queue.svg.png , 19.04.2016
- Ab07: „lehrer.uni-karlsruhe.de , Jürgen Dehmer“ http://www.lehrer.uni-karlsruhe.de/~za714/informatik/infkurs/img/zbaum2.gif , 19.04.2016