

1 Datenstrukturen

- 1.1 Abstrakte Datentypen
- 1.2 Lineare Strukturen
- 1.3 Bäume
- 1.4 Prioritätsschlangen
- 1.5 Graphen




Lineare Strukturen

- Sequenz $\{x_1, \dots, x_n\}$ von beliebigen Datenobjekten x_i
- Typische Operationen
 - Füge y am Anfang / am Ende / hinter x_i ein
 - Ersetze x_i durch y
 - Entferne x_i
 - Lese das i -te Element



Lineare Strukturen

- Typische Operationen
 - Verknüpfe zwei Sequenzen
 - Zerlege eine Sequenz
 - Bestimme die Länge der Sequenz
 - Teste, ob ein Element y vorhanden ist
 - Sortiere die Elemente x_i
 - ...

3  Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Wie kann man diese Operationen in Form von Axiomen festlegen?

1.2 Lineare Strukturen

- 1.2.1 Listen
- 1.2.2 Warteschlangen
- 1.2.3 Stacks

4  Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Die (linearen) Datenstrukturen, die wir in dieser Vorlesung betrachten werden.

1.2.1 Listen

- $L = \{x_1, \dots, x_n\}$
- Zugriff auf beliebige Elemente x_i
 - Per Index (random access)
 - Get(i)
 - Per Marker (sequential access)
 - GetFirst()
 - GetNext()
 - GetPrevious()



Random Access

- Implementierung durch Arrays $L[]$
- $\text{Get}(i) = L[i]$
- Nachteile
 - Elemente löschen erzeugt Lücken oder alle Elemente mit höherem Index müssen verschoben werden (garbage collection)
 - Statische Obergrenze für Listenlänge



Was ist, wenn Daten nicht mehr in Speicher passen? Mit der Frage werden wir uns später auch beschäftigen. Random Access bedeutet, dass man auf jedes beliebige Element zu jeder Zeit zugreifen kann.

Sequential Access

- Implementierung durch Pointer oder Container
- Marker zeigt auf aktuelle Position
- Nachteil: Elementzugriff erfordert lineare Suche (kann jedoch meistens vermieden werden, z.B. „for each“).
- Vorteil: beliebiges Erweitern und Löschen

7

Datenstrukturen und Algorithmen

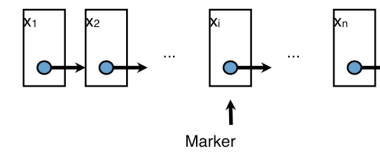
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

RWTH AACHEN
UNIVERSITY

Sequentieller Zugriff bedeutet, dass man nicht auf jedes Element zu jeder Zeit, sondern auf die Daten in linearer Reihenfolge zugreifen kann.

Sequential Access

- Pointer



8

Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

RWTH AACHEN
UNIVERSITY

Beispiel einer einfach verketteten Liste. Die blauen Punkte repräsentieren Pointer, die auf einen Speicherbereich, an dem Daten liegen, zeigen. Entweder werden direkt Werte gespeichert...

Sequential Access

- Container

X_1 X_2 ... X_i ... X_n
 ↑
 Marker

9 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

...oder Container-Klassen zum Kapseln von Daten benutzt.

Listen

- Wertebereich : $L = \{ \} \cup W \cup W^2 \cup W^3 \cup \dots$
- Create : $\rightarrow L$
- Get : $L \rightarrow W$
- Reset : $L \rightarrow L$
- Next : $L \rightarrow L$
- Insert : $W \times L \rightarrow L$
- Delete : $L \rightarrow L$
- Empty : $L \rightarrow \text{Bool}$
- IsLast : $L \rightarrow \text{Bool}$

10 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Listen

- Achtung: der Marker beschreibt einen Zustand, was sich mit funktionalen Axiomen schlecht formulieren lässt.
- Standard-Trick: führe ein zusätzliches Prädikat Insert^* ein, das aber keine weitere Listen-Funktion darstellt.

11

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Sternchen dient als „Marker“.

Listen

• $\text{Empty}(\text{Create}()) = \text{true}$
$\text{Empty}(\text{Insert}(x,z)) = \text{false}$
$\text{Empty}(\text{Insert}^*(x,z)) = \text{false}$
• $\text{IsLast}(\text{Create}()) = \text{true}$
$\text{IsLast}(\text{Insert}(x,z)) = \text{false}$
$\text{IsLast}(\text{Insert}^*(x,z)) = \text{IsLast}(z)$
• $\text{Get}(\text{Insert}(x,z)) = x$
$\text{Get}(\text{Insert}^*(x,z)) = \text{Get}(z)$
$\text{Insert}(x, \text{Insert}^*(y,z)) = \text{Insert}^*(y, \text{Insert}(x,z))$

12

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Erst Inserts mit Stern, dann Inserts ohne Stern. $\text{IsLast}(\text{Insert}^*(x,z)) = \text{IsLast}(z)$ wird solange rekursiv aufgerufen, bis Insert ohne Stern erreicht wurde (Markerposition). Der Marker gibt die aktuelle „Leseposition“ an. Das letzte Axiom wird benutzt, um neue Elemente einzufügen (und die alte Markerposition beizubehalten).

Listen

- $Delete(Create()) = Create()$
- $Delete(Insert(x,z)) = z$
- $Delete(Insert^*(x,z)) = Insert^*(x,Delete(z))$
- $Next(Create()) = Create()$
- $Next(Insert(x,z)) = Insert^*(x,z)$
- $Next(Insert^*(x,z)) = Insert^*(x,Next(z))$
- $Reset(Create()) = Create()$
- $Reset(Insert(x,z)) = Insert(x,z)$
- $Reset(Insert^*(x,z)) = Insert(x,Reset(z))$

13

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Reset() setzt Markerposition wieder zurück auf Ausgangsposition (äußerstes Element).

Listen

- Satz: Jede Liste hat die Form
 $Insert^*(x_1, \dots, Insert^*(x_i, Insert(x_{i+1}, \dots, Insert(x_n, Create()))))$
- Beweis durch vollständige Induktion über die Anzahl der verwendeten Operationen
 - Create() ... n=0
 - Jede andere Operation erhält die Form

14

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Get()

- `Get(Insert*(x1, ... Insert*(xi, Insert(xi+1, ... Insert(xn, Create()))))`
- `Get(... Insert*(xi, Insert(xi+1, ... Insert(xn, Create())))`
- `Get(Insert*(xi, Insert(xi+1, ... Insert(xn, Create())))`
- `Get(Insert(xi+1, ... Insert(xn, Create())))`
- `xi+1`

15



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Next()

- `Next(Insert*(x1, ... Insert*(xi, Insert(xi+1, ... Insert(xn, Create()))))`
- `Insert*(x1, Next(... Insert*(xi, Insert(xi+1, ... Insert(xn, Create()))))`
- `Insert*(x1, ... Next(Insert*(xi, Insert(xi+1, ... Insert(xn, Create()))))`
- `Insert*(x1, ... Insert*(xi, Next(Insert(xi+1, ... Insert(xn, Create()))))`
- `Insert*(x1, ... Insert*(xi, Insert*(xi+1, ... Insert(xn, Create())))`

16



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Insert()

- `Insert(y, Insert*(x1, ... Insert*(xi, Insert(xi+1, ... Insert(xn, Create()))))`
- `Insert*(x1, Insert(y, ... Insert*(xi, Insert(xi+1, ... Insert(xn, Create()))))`
- `Insert*(x1, ... Insert(y, Insert*(xi, Insert(xi+1, ... Insert(xn, Create()))))`
- `Insert*(x1, ... Insert*(xi, Insert(y, Insert(xi+1, ... Insert(xn, Create()))))`

17

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Hier wird ein neues Element bei bestehender Markerposition eingefügt.

Delete()

- `Delete(Insert*(x1, ... Insert*(xi, Insert(xi+1, ... Insert(xn, Create()))))`
- `Insert*(x1, Delete(... Insert*(xi, Insert(xi+1, ... Insert(xn, Create()))))`
- `Insert*(x1, ... Delete(Insert*(xi, Insert(xi+1, ... Insert(xn, Create()))))`
- `Insert*(x1, ... Insert*(xi, Delete(Insert(xi+1, ... Insert(xn, Create()))))`
- `Insert*(x1, ... Insert*(xi, ... Insert(xn, Create()))`

18

Datenstrukturen und Algorithmen
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Weitere Funktionen

- $\text{Overwrite}(y, \text{Insert}(x, z)) = \text{Insert}(y, z)$
- $\text{Overwrite}(y, \text{Insert}^*(x, z)) = \text{Insert}^*(x, \text{Overwrite}(y, z))$
- $\text{Join}(\text{Create}(), z) = z$
- $\text{Join}(\text{Insert}(x, y), z) = \text{Insert}^*(x, \text{Join}(y, z))$
- $\text{Join}(\text{Insert}^*(x, y), z) = \text{Insert}^*(x, \text{Join}(y, z))$
- ...

19



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

RWTH AACHEN
UNIVERSITY

Implementierung

- Finden, Lesen, Überschreiben
- Löschen (Sonderfälle bei leerer Liste)
- Einfügen (Sonderfälle an den Enden)
- Anchor, Sentinel
- Einfach / Doppelt verkettete Listen

20



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

RWTH AACHEN
UNIVERSITY

Einfach verkettete Liste

- class Element {
 Datentyp X
 Element next
}

21

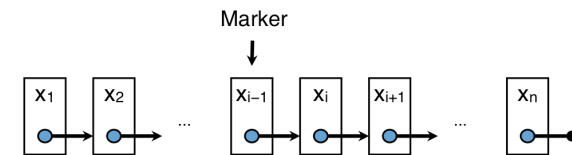
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Sequential Access

- Delete



22

Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Sequential Access

- Delete

Delete()

23 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Sequential Access

- Delete

Delete()

24 **Datenstrukturen und Algorithmen**
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Einfach verkettete Listen

- Delete()
Marker.next ← Marker.next.next

25



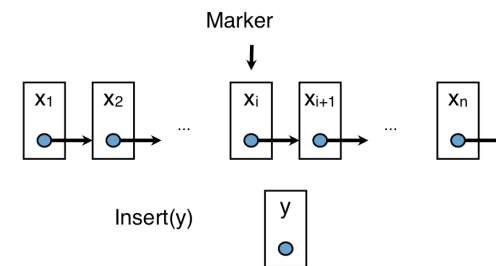
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Sequential Access

- Insert



26



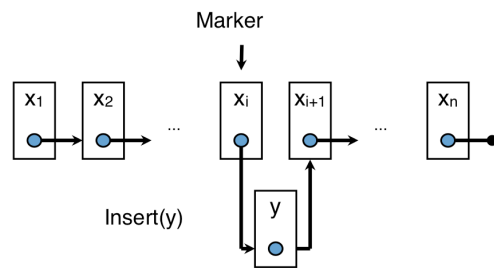
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Sequential Access

- Insert



27

Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Einfach verkettete Listen

- Insert(Y)
 $Y.next \leftarrow Marker.next$
 $Marker.next \leftarrow Y$

28

Datenstrukturen und Algorithmen


Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Modifikation am Listenanfang

Marker

The diagram shows a linked list with nodes $X_1, X_2, \dots, X_i, X_{i+1}, \dots, X_n$. Each node is a rectangle containing a blue circle and an arrow pointing to the next node. A 'Marker' points to the first node X_1 .


29  Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Modifikation am Listenanfang

Marker

The diagram shows a linked list with nodes $A, X_2, \dots, X_i, X_{i+1}, \dots, X_n$. Each node is a rectangle containing a blue circle and an arrow pointing to the next node. A 'Marker' points to the first node A .

Anchor
Erstes Element enthält keine Daten

30  Datenstrukturen und Algorithmen
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Doppelt verkettete Listen

- Flexiblerer Zugriff (upstream/downstream)
 - Next(), Prev()
- Bisher: Einfügen/Löschen nach dem Marker (wg. Pointer update)
- Jetzt: Marker zeigt auf aktuelles Element

31



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Doppelt verkettete Liste

- ```
class Element {
 Datentyp X
 Element prev, next
}
```

32

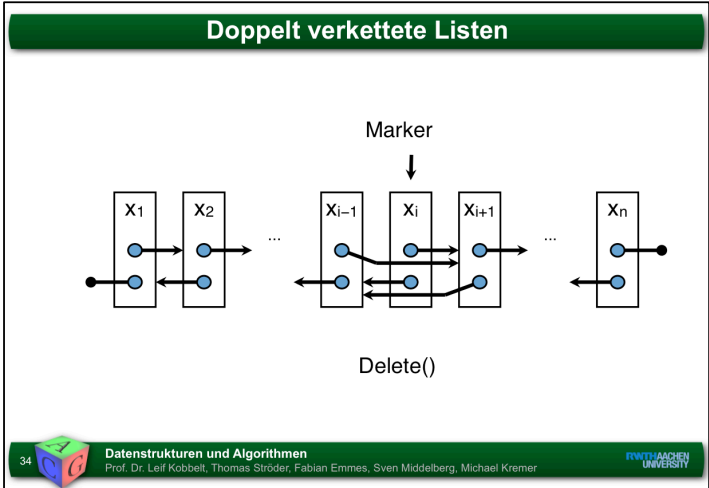
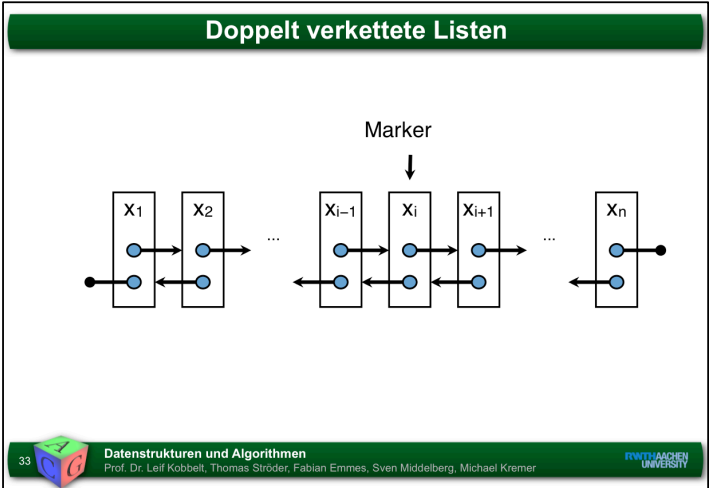


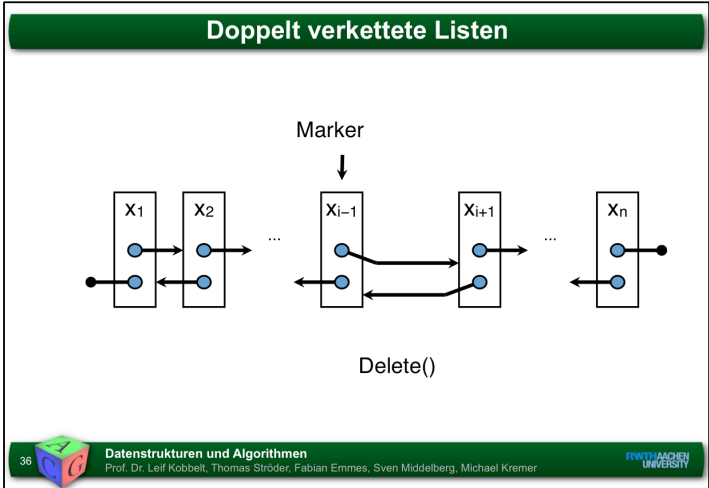
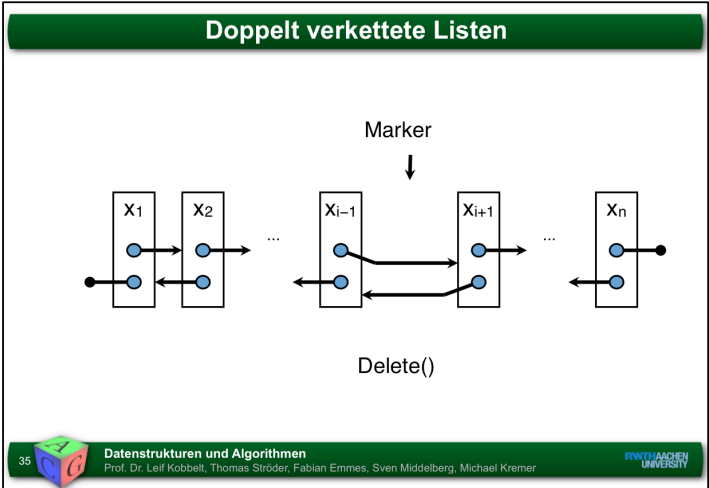
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer









## Doppelt verkettete Listen

- Delete()  
Marker.prev.next ← Marker.next  
Marker.next.prev ← Marker.prev  
Marker ← Marker.prev

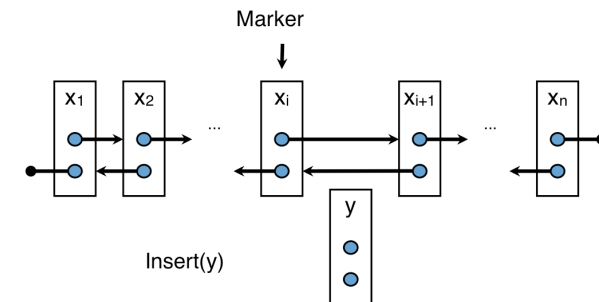
37

Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Doppelt verkettete Listen

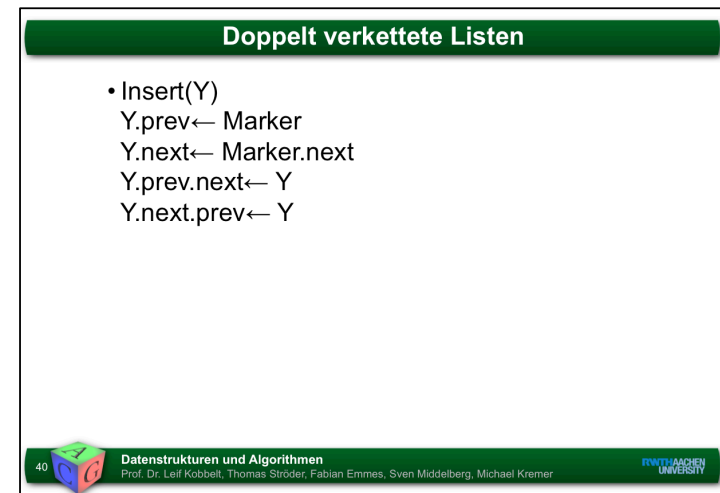
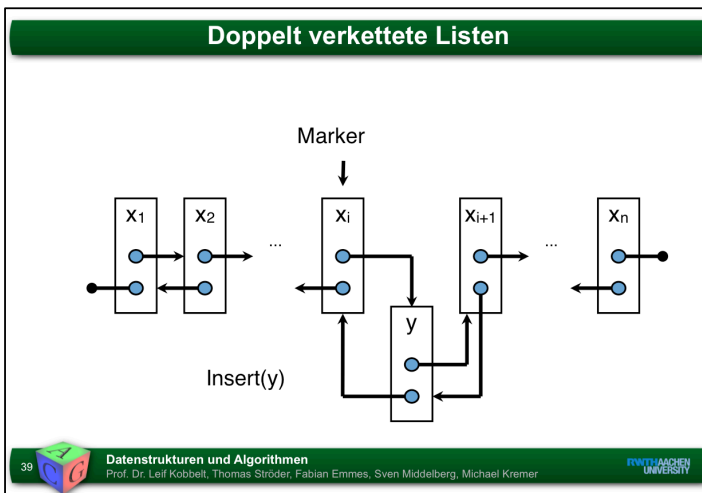


38

Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer





Fazit: Listen klingen erst mal unhandlich. Aber alle Operationen haben IMMER konstante Anzahl an Pointer-Operationen, unabhängig von der Datengröße.

### Modifikation an den Enden

Marker

41

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

### Modifikation an den Enden

Marker

Anchor/Sentinel (keine Daten)

42

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

## 1.2.2 Warteschlange

- Liste mit eingeschränkter Funktionalität
- Einfügen nur am Ende
- Auslesen/Entfernen nur am Anfang
- „First in, first out“ (FIFO)

43



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Warteschlange

- Wertebereich :  $L = \{ \} \cup W \cup W^2 \cup W^3 \cup \dots$
- Create :  $\rightarrow L$
- Enq :  $W \times L \rightarrow L$
- Deq :  $L \rightarrow L$
- Get :  $L \rightarrow W$
- Empty :  $L \rightarrow \text{Bool}$

44



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Enq = Enqueue (einfügen)  
Deq = Dequeue (ausfügen)

## Warteschlange

- $\text{Empty}(\text{Create}()) = \text{true}$
- $\text{Empty}(\text{Enq}(x,z)) = \text{false}$
- $\text{Deq}(\text{Enq}(x,\text{Create}())) = \text{Create}()$
- $\text{Deq}(\text{Enq}(x,z)) = \text{Enq}(x,\text{Deq}(z))$  if  $z \neq \{ \}$
- $\text{Get}(\text{Enq}(x,\text{Create}())) = x$
- $\text{Get}(\text{Enq}(x,z)) = \text{Get}(z)$  if  $z \neq \{ \}$

45



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Warteschlange

- Satz: Jede Warteschlange hat die Form  
 $\text{Enq}(x_n, \text{Enq}(x_{n-1}, \dots \text{Enq}(x_1, \text{Create}()))))$
- Beweis durch vollständige Induktion über die Anzahl der verwendeten Operationen
  - $\text{Create}()$  ...  $n=0$
  - Jede andere Operation erhält die Form

46



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Warteschlange

- `Get(Enq(xn, Enq(xn-1, ... Enq(x1, Create()))))`
- `Enq(xn, Get(Enq(xn-1, ... Enq(x1, Create()))))`
- `Enq(xn, Enq(xn-1, Get( ... Enq(x1, Create()))))`
- `Enq(xn, Enq(xn-1, ... Get(Enq(x1, Create()))))`
- `x1`

47



Datenstrukturen und Algorithmen

Prof. Dr. Lief Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Warteschlange

- `Deq(Enq(xn, Enq(xn-1, ... Enq(x1, Create()))))`
- `Enq(xn, Deq(Enq(xn-1, ... Enq(x1, Create()))))`
- `Enq(xn, Enq(xn-1, Deq( ... Enq(x1, Create()))))`
- `Enq(xn, Enq(xn-1, ... Deq(Enq(x1, Create()))))`
- `Enq(xn, Enq(xn-1, ... Create()))`

48



Datenstrukturen und Algorithmen

Prof. Dr. Lief Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer





## Array-Implementierung

- Da nur am Anfang eingefügt und am Ende gelöscht wird, ist keine garbage collection notwendig.
- Maximale Länge wird als bekannt vorausgesetzt.

49



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Array-Implementierung

- ```
class Schlange {  
    Datentyp S[Länge]  
    int front, back  
}
```

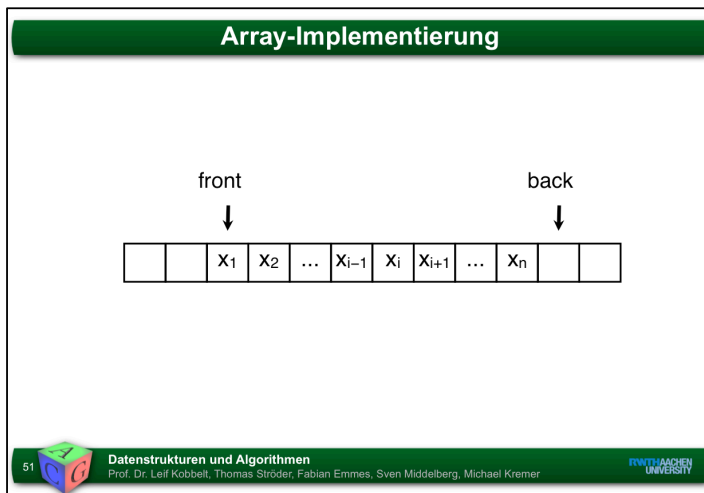
50



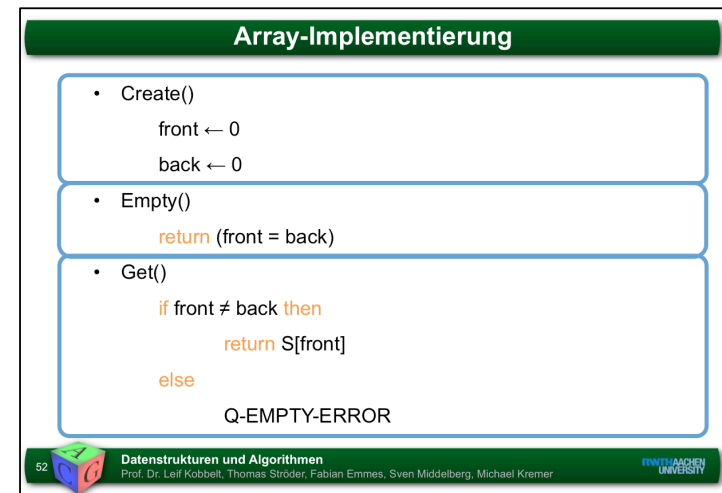
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer





Front- und Back-Pointer werden benutzt, um sich zu jeder Zeit die Position des ersten und letzten Elements zu „merken“. Diese müssen bei den Operationen auf dem Array natürlich auch angepasst werden (zusätzlich zu den Daten).



Array-Implementierung

- Deq()

```
if front ≠ back then  
  front ← (front + 1) % Länge
```

```
else
```

```
  Q-EMPTY-ERROR
```

- Enq(x)

```
S[back] ← x
```

```
back ← (back + 1) % Länge
```

```
if front = back then
```

```
  Q-FULL-ERROR
```

53

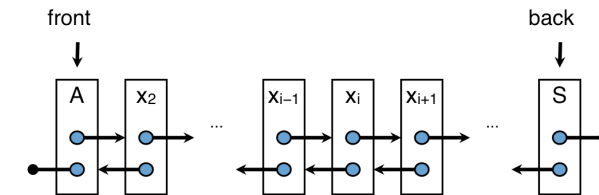


Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Pointer-Implementierung



Interner Marker entfällt!

54



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



1.2.3 Stack

- Liste mit eingeschränkter Funktionalität
- Einfügen nur am Anfang
- Auslesen/Entfernen nur am Anfang
- „Last in, first out“ (LIFO)

55



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Stack

- Wertebereich : $L = \{ \} \cup W \cup W^2 \cup W^3 \cup \dots$
- Create: $\rightarrow L$
- Push : $W \times L \rightarrow L$
- Pop : $L \rightarrow L$
- Top : $L \rightarrow W$
- Empty : $L \rightarrow \text{Bool}$

56



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Stack

- `Empty(Create())` = true
- `Empty(Push(x,z))` = false
- `Pop(Push(x,z))` = z
- `Top(Push(x,z))` = x

57



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Stack

- Satz: Jeder Stack hat die Form
 $\text{Push}(x_1, \text{Push}(x_2, \dots \text{Push}(x_n, \text{Create()})))$
- Beweis durch vollständige Induktion über die Anzahl der verwendeten Operationen
 - `Create()` ... $n=0$
 - Jede andere Operation erhält die Form

58



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Array-Implementierung

- Da nur am Anfang eingefügt und gelöscht wird, ist keine garbage collection notwendig.
- Maximale Größe wird als bekannt vorausgesetzt.

59



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



Array-Implementierung

- ```
class Stack {
 Datentyp S[Größe]
 int top
}
```

60

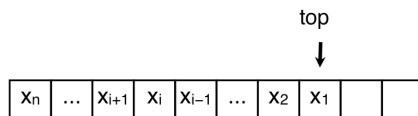


Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Array-Implementierung



61

Datenstrukturen und Algorithmen

Prof. Dr. Lief Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Array-Implementierung

- Create()

```
top ← -1
```

- Empty()

```
return (top = -1)
```

- Top()

```
if top ≠ -1 then
```

```
 return S[top]
```

```
else
```

```
 STACK-EMPTY-ERROR
```

62

Datenstrukturen und Algorithmen

Prof. Dr. Lief Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Array-Implementierung

- Push(x)

```
top ← top+1
```

```
if top < Größe then
```

```
 S[top] ← x
```

```
else
```

```
 STACK-FULL-ERROR
```

- Pop()

```
if top ≠ -1 then
```

```
 top ← top-1
```

```
else
```

```
 STACK-EMPTY-ERROR
```

63



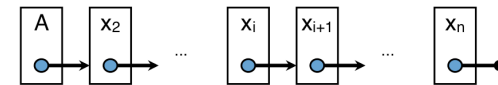
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Pointer-Implementierung

top  
↓



64



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer





## Suche im Labyrinth

- Geg:
  - Spielfläche  $W = [0..n] \times [0..m]$
  - mögliche Positionen  $P = (x,y) \in W$
  - Bewegungsrichtungen:  $R = \{N,S,W,O\}$
  - Labyrinth ...  $L : W \times R \rightarrow \text{Bool}$
  - Startposition  $P_{\text{begin}}$
  - Zielposition  $P_{\text{end}}$
- Ges: Pfad  $S \in R^k$  von  $P_{\text{begin}}$  nach  $P_{\text{end}}$

65



Datenstrukturen und Algorithmen

Prof. Dr. Lief Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

RWTH AACHEN  
UNIVERSITY

## Suche im Labyrinth

- Funktion  $Go: W \times R \rightarrow W$  beschreibt einen Schritt  $Go(P,r)=Q$  von  $P$  in die Richtung  $r$  nach  $Q$ .
- Wenn  $L(P_{\text{begin}},r) = \text{true}$  und  $Go(P_{\text{begin}},r) = Q$  und es existiert ein Pfad  $r_1, \dots, r_n$  von  $Q$  nach  $P_{\text{end}}$  dann ist  $r, r_1, \dots, r_n$  ein Pfad von  $P_{\text{begin}}$  nach  $P_{\text{end}}$ .

66



Datenstrukturen und Algorithmen

Prof. Dr. Lief Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

RWTH AACHEN  
UNIVERSITY

## Suche im Labyrinth

- Annahme: Labyrinth hat keine Zyklen
- Pfad = Liste
- Erweiterung nur am Anfang der Liste  
→ Pfad = Stack
- Rekursive Formulierung

67



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Rekursiver Algorithmus

- Bool Suche(P,S,Q)

```
if P = Q then
```

```
 S ← Create()
```

```
 return true
```

```
else
```

```
 for r ∈ {N,S,W,O} do
```

```
 if L(P,r) and Suche(Go(P,r),S,Q) then
```

```
 S ← Push(r,S)
```

```
 return true
```

```
 return false
```

68



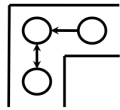
Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Rekursiver Algorithmus

- Problem: unendliche Suche
- Lösung: verbiete Schritte zurück.
- „Negative“ Richtung:  
-N = S, -S = N, -W = O, -O = W



69

Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Rekursiver Algorithmus

- Bool Suche(P,S,Q,r')

if P = Q then

S ← Create()

return true

else

for r ∈ {N,S,W,O} \ r' do

if L(P,r) & Suche(Go(P,r),S,Q,-r) then

S ← Push(r,S)

return true

return false

70

Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Iterativer Algorithmus

- Rekursion steuert die Suche
- Direkte Lösung unter Verwendung eines Stack
- Ordnung der Richtungen  $N < O < S < W$

(next(N) = O, next(O) = S, ...)

- Sortiere Pfade lexikographisch

71



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Lexikographische Reihenfolge

- |          |          |
|----------|----------|
| • N      | • NNNNOS |
| • NN     | • ..     |
| • NNN    | • NNNNOW |
| • ...    | • ...    |
| • NNNNO  | • NNNNS  |
| • NNNNON | • ...    |
| • ...    | • NNNO   |
| • NNNNOO | • ...    |
| • ...    | • WW     |

72



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Iterativer Algorithmus

```
• P ← Pbegin
 S ← Create()
 r ← 'N'
 while P ≠ Pend and (L(P,r) or r < 'W' or not Empty(S)) do
 while P ≠ Pend and L(P,r) do
 S = Push(r,S)
 P ← Go(P,r)
 r ← 'N'
 if P ≠ Pend then
 while r = 'W' and not Empty(S) do
 r = Top(S)
 P ← Go(P,-r)
 S ← Pop(S)
 if r < 'W' then
 r ← next(r)
```

73



Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## Iterativer Algorithmus

```
• P ← Pbegin
 S ← Create()
 r ← 'N'
 while P ≠ Pend and (L(P,r) or r < 'W' or not Empty(S)) do
 while P ≠ Pend and L(P,r) and r ≠ -Top(S) do
 S = Push(r,S)
 P ← Go(P,r)
 r ← 'N'
 if P ≠ Pend then
 while r = 'W' and not Empty(S) do
 r = Top(S)
 P ← Go(P,-r)
 S ← Pop(S)
 if r < 'W' then
 r ← next(r)
```

74




Datenstrukturen und Algorithmen

Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




**3 x 3 grid ...**



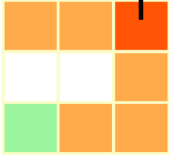
- N
- O
- S
- SO
- SS
- SSO
- SSS
- SSW
- SSWN
- SSWS
- SSWW

75

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




**3 x 3 grid ...**



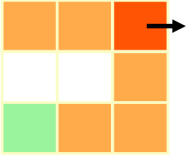
- N
- O
- S
- SO
- SS
- SSO
- SSS
- SSW
- SSWN
- SSWS
- SSWW

76

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




**3 x 3 grid ...**




- N
- O
- S
- SO
- SS
- SSO
- SSS
- SSW
- SSWN
- SSWS
- SSWW

77

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




**3 x 3 grid ...**



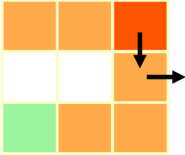
- N
- O
- S
- SO
- SS
- SSO
- SSS
- SSW
- SSWN
- SSWS
- SSWW

78

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




**3 x 3 grid ...**



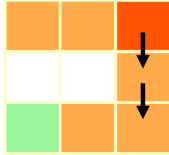
- N
- O
- S
- SO
- SS
- SSO
- SSS
- SSW
- SSWN
- SSWS
- SSWW

79

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




**3 x 3 grid ...**



- N
- O
- S
- SO
- SS
- SSO
- SSS
- SSW
- SSWN
- SSWS
- SSWW

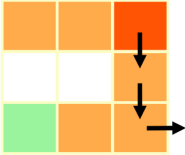
80

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer






**3 x 3 grid ...**



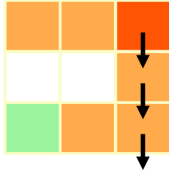
- N
- O
- S
- SO
- SS
- SSO
- SSS
- SSW
- SSWN
- SSWS
- SSWW

81

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




**3 x 3 grid ...**



- N
- O
- S
- SO
- SS
- SSO
- SSS
- SSW
- SSWN
- SSWS
- SSWW

82

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



**3 x 3 grid ...**

N  
 O  
 S  
 SO  
 SS  
 SSO  
 SSS  
 SSW  
 SSWN  
 SSWS  
 SSWW

83 **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

**3 x 3 grid ...**

N  
 O  
 S  
 SO  
 SS  
 SSO  
 SSS  
 SSW  
 SSWN  
 SSWS  
 SSWW

84 **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

**3 x 3 grid ...**

N  
 O  
 S  
 SO  
 SS  
 SSO  
 SSS  
 SSW  
 SSWN  
 SSWS  
 SSWW


85 **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

**3 x 3 grid ...**

N  
 O  
 S  
 SO  
 SS  
 SSO  
 SSS  
 SSW  
 SSWN  
 SSWS  
 SSWW

86 **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

**3 x 3 grid ...**




```

N
NO
SSO
SS
SSO
SSS
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS

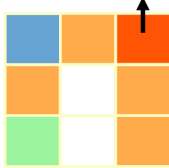
```

87

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



**3 x 3 grid ...**




```

N
NO
SSO
SS
SSO
SSS
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS


```

88

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer





**3 x 3 grid ...**



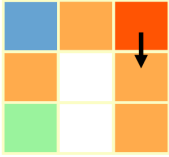
```

N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS

```

89  **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



**3 x 3 grid ...**



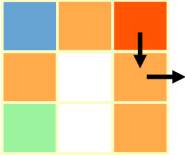
```

N
O
S
SO
SS
SSO
SSS
SSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS

```

90  **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 


**3 x 3 grid ...**




```

N
NO
SSO
SSSO
SSSS
SSSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS

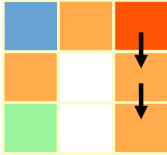
```

91


Datenstrukturen und Algorithmen  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




**3 x 3 grid ...**




```

N
NO
SSO
SSSO
SSSS
SSSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS


```

92


Datenstrukturen und Algorithmen  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




**3 x 3 grid ...**




N  
NO  
SO  
SS  
SSO  
SSS  
SSW  
SW  
W  
WN  
WS  
WW  
WWN  
WWS  
WWSO  
WWSS

93

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




**3 x 3 grid ...**



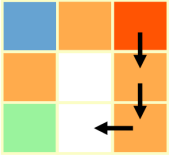
N  
NO  
SO  
SS  
SSO  
SSS  
SSW  
SW  
W  
WN  
WS  
WW  
WWN  
WWS  
WWSO  
WWSS

94

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer





**3 x 3 grid ...**

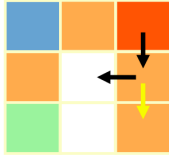


A 3x3 grid with colored cells: (1,1) blue, (1,2) orange, (1,3) red, (2,1) orange, (2,2) white, (2,3) orange, (3,1) green, (3,2) white, (3,3) orange. Arrows indicate moves: a left arrow from (3,3) to (3,2), a down arrow from (2,3) to (3,3), and another down arrow from (1,3) to (2,3).

N  
 SO  
 SS  
 SSO  
 SSS  
 SSW  
 SW  
 W  
 WN  
 WS  
 WW  
 WWN  
 WWS  
 WWSO  
 WWSS



95  **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

**3 x 3 grid ...**



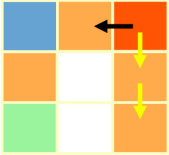
A 3x3 grid with colored cells: (1,1) blue, (1,2) orange, (1,3) red, (2,1) orange, (2,2) white, (2,3) orange, (3,1) green, (3,2) white, (3,3) orange. Arrows indicate moves: a left arrow from (2,3) to (2,2), a down arrow from (1,3) to (2,3), and a yellow arrow pointing down from (2,3) to (3,3).

N  
 SO  
 SS  
 SSO  
 SSS  
 SSW  
 SW  
 W  
 WN  
 WS  
 WW  
 WWN  
 WWS  
 WWSO  
 WWSS

96  **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



**3 x 3 grid ...**




```

N
NO
SSO
SSS
SSSO
SSSS
SSSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS

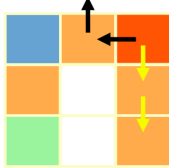
```

97

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



**3 x 3 grid ...**




```

N
NO
SSO
SSS
SSSO
SSSS
SSSW
SW
W
WN
WS
WW
WWN
WWS
WWSO
WWSS

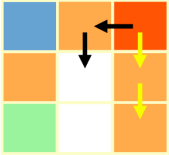
```

98

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer





**3 x 3 grid ...**

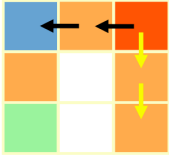


A 3x3 grid with colored cells: (1,1) blue, (1,2) orange, (1,3) red, (2,1) orange, (2,2) white, (2,3) orange, (3,1) green, (3,2) white, (3,3) orange. Arrows show moves: a black arrow from (1,2) to (1,1), a black arrow from (1,3) to (1,2), a black arrow from (2,3) to (2,2), and a yellow arrow from (2,3) to (3,3).

N  
 NO  
 SO  
 SS  
 SSO  
 SSS  
 SSW  
 SW  
 W  
 WN  
 WS  
 WW  
 WWN  
 WWS  
 WWSO  
 WWSS



99  **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

**3 x 3 grid ...**



A 3x3 grid with colored cells: (1,1) blue, (1,2) orange, (1,3) red, (2,1) orange, (2,2) white, (2,3) orange, (3,1) green, (3,2) white, (3,3) orange. Arrows show moves: a black arrow from (1,2) to (1,1), a black arrow from (1,3) to (1,2), a black arrow from (2,3) to (2,2), and a yellow arrow from (2,3) to (3,3).

N  
 NO  
 SO  
 SS  
 SSO  
 SSS  
 SSW  
 SW  
 W  
 WN  
 WS  
 WW  
 WWN  
 WWS  
 WWSO  
 WWSS

100  **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

**3 x 3 grid ...**

N  
 NO  
 SO  
 SS  
 SSO  
 SSS  
 SSSO  
 SSW  
 SW  
 W  
 WN  
 WS  
 WW  
 WWN  
 WWS  
 WWSO  
 WWSS

101 **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

**3 x 3 grid ...**

N  
 NO  
 SO  
 SS  
 SSO  
 SSS  
 SSSO  
 SSW  
 SW  
 W  
 WN  
 WS  
 WW  
 WWN  
 WWS  
 WWSO  
 WWSS

102 **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

**3 x 3 grid ...**

N  
 SO  
 SS  
 SSO  
 SSS  
 SSW  
 SW  
 W  
 WN  
 WS  
 WW  
 WWN  
 WWS  
 WWSO  
 WWSS

103 **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

**3 x 3 grid ...**

N  
 SO  
 SS  
 SSO  
 SSS  
 SSW  
 SW  
 W  
 WN  
 WS  
 WW  
 WWN  
 WWS  
 WWSO  
 WWSS

104 **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

## Stack

- Merke: Rekursive Algorithmen lassen sich in der Regel mit einer Stack-Datenstruktur auch iterativ formulieren.
- Das LIFO-Prinzip entspricht der Abarbeitungsreihenfolge der geschachtelten Prozeduren (was zuletzt aufgerufen wird, wird als erstes bearbeitet).
- Beispiel: systematische Suche in einem Labyrinth (bei Sackgassen zurückgehen).

