Übung Algorithmen und Datenstrukturen

Sommersemester 2016

Patrick Schäfer, Humboldt-Universität zu Berlin

Agenda

1. Organisatorisches
2. Fragen zum ersten Übungsblatt
3. Landau-Notation (Zusammenfassung)
4. Infix- und Postfix-Notation
5. Stacks und Queues
6. Vorstellung des zweiten Übungsblatts

Organisatorisches

- Literaturempfehlung:
  Ottmann, Thomas, and Peter Widmayer. *Algorithmen und Datenstrukturen*. 5. Auflage, Springer-Verlag, 2011.
- Programmieraufgaben (Java 1.7) sind auf gruenau2 zu testen und in Goya abzugeben (gleicher Termin wie schriftliche Aufgaben)
- Wer sucht noch Anschluss an eine Übungsgruppe?

Landau Notation

- $O(g) = \left\{ f: \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0} \middle| \begin{matrix} \exists c \in \mathbb{R} > 0 \quad \exists n_0 \in \mathbb{R} \geq 0) \\ \forall n \geq n_0\colon\; f(n) \leq c \cdot g(n)) \end{matrix} \right\}$
- Zusammenhänge zwischen O, Ω, Θ, o und ω
  - $f \in o(g) \Longrightarrow f \in O(g)$ $\qquad f \in \omega(g) \Longrightarrow f \in \Omega(g)$
  - $f \in O(g) \Longleftrightarrow g \in \Omega(f)$ $\qquad f \in o(g) \Longleftrightarrow g \in \omega(f)$
  - $f \in o(g) \Longrightarrow f \notin \Omega(g)$ $\qquad f \in \omega(g) \Longrightarrow f \notin O(g)$
- Grenzwert als hinreichendes Kriterium
  - $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} < \infty \Rightarrow f \in O(g)$
  - $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f \in o(g)$
- Satz von L'Hôpital: $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \lim\limits_{n\to\infty} \frac{f'(n)}{g'(n)}$, falls f und g differenzierbar sind und ihre Grenzwerte beide gegen 0 oder beide gegen ∞ gehen

Aufgabe: Funktionen ordnen

- Beweisen Sie, dass $f \epsilon O(g)$

	f(n)	g(n)
(a)	$ln\, n$	$log\, n$
(b)	2^n	$n!$

Infix- und Postfix-Notation

- Infix-Ausdrücke sind wie folgt rekursiv definiert:
  - Eine Ziffer $d \in \{0,1,2,3,4,5,6,7,8,9\}$ ist ein Infix-Ausdruck
  - Für alle Infix-Ausdrücke a_1, a_2 sind $(a_1 + a_2)$, $(a_1 - a_2)$, $(a_1 * a_2)$ und (a_1/a_2) Infix-Ausdrücke
  - Ungültig: leerer String, $\left((4-(4-4))\right)$, $(3+(4-7)-2)$
- Postfix-Ausdrücke sind wie folgt rekursiv definiert:
  - Eine Ziffer $d \in \{0,1,2,3,4,5,6,7,8,9\}$ ist ein Postfix-Ausdruck
  - Für alle Postfix-Ausdrücke a_1, a_2 sind $a_1 a_2 +$, $a_1 a_2 -$, $a_1 a_2 *$ und $a_1 a_2 /$ Postfix-Ausdrücke
  - ungültig: leerer String, $44+-$, $4-4$, $(44-)$

Infix-Ausdruck	Postfix-Ausdruck	Zu lesen als
$(2-5)$	$25-$	
$(5-2)$	$52-$	
$\left((2-5)*6\right)$	$25-6*$	$\left((25-)6*\right)$
$\left(2-(5*6)\right)$	256*-	$\left(2(56*)-\ \right)$

Infix- und Postfix-Notation

Infix-Ausdruck	Postfix-Ausdruck	Eval
$\big(((2-5)+6\big)*2\big)$	?	?
?	$3729*+-$	?
$\big((5*5)+(2-5)\big)$	?	?
?	$442/-94+*$	?

Infix- und Postfix-Notation

$$\Big(\big((2-5)+6\big)*2\Big)$$

$$= \underbrace{\Big(\underbrace{\big(\underbrace{(2-5)}_{-3}+6\big)}_{3}*2\Big)}_{6}$$

$$=6$$

$$25-6+2\ *$$

$$=\underbrace{\underbrace{\underbrace{25\ -}_{-3}\ 6\ +}_{3}\ 2\ *}_{6}$$

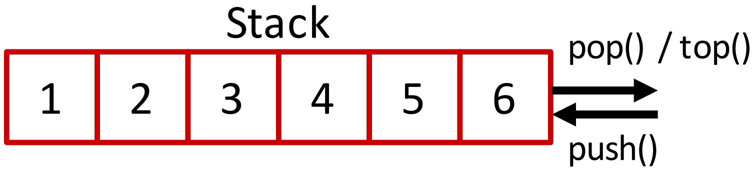
$$=\ 6$$

Infix- und Postfix-Notation

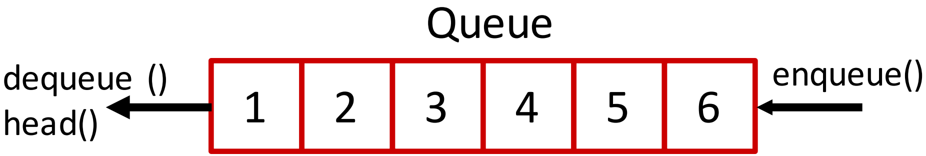
Infix-Ausdruck	Postfix-Ausdruck	Eval
$\Big(\big((2-5)+6\big)*2\Big)$	$25-6+2*$	6
$\Big(3-\big(7+(2*9)\big)\Big)$	$3729*+-$	-22
$\big((5*5)+(2-5)\big)$	$55*25-+$	22
$\Big(\big(4-(4/2)\big)*(9+4)\Big)$	$442/-94+*$	26

Stacks und Queues

- Stack / Stapel: Abstrakter Datentyp mit last-in, first-out (LIFO) Semantik
  - push(value): legt das Element oben auf den Stack.
  - pop(): entfernt das oberste Element vom Stack und gibt es zurück.
  - top() / peek(): gibt das oberste Element zurück (ohne es zu entfernen).
  - isEmpty(): gibt „true“ zurück, falls der Stack leer ist.



- Queue / Warteschlange: Abstrakter Datentyp mit first-in, first-out (FIFO) Semantik
  - enqueue(value): hängt das Element an das Ende der Queue an.
  - dequeue(): entfernt das erste Element vom Anfang der Queue und gibt es zurück.
  - head(): liefert das erste Element vom Anfang der Queue ohne es zu entfernen.
  - isEmpty(): gibt „true“ zurück, falls die Queue leer ist.



Aufgabe: Rangieren

- Die Abbildungen zeigen Eisenbahngleise, welche einen Stack bzw. eine Queue mit Überholspur darstellen.

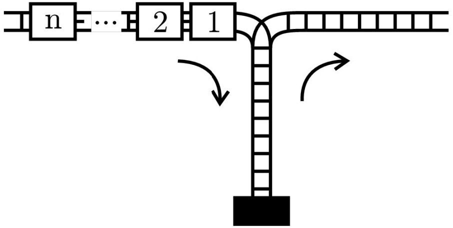


Abbildung 1: Stack

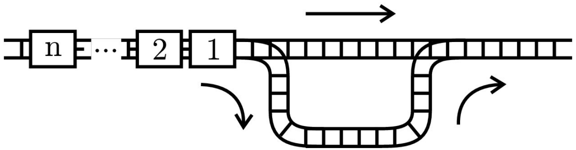


Abbildung 2: Queue

- Auf der linken Seite stehen die absteigend nummerierten Waggons n bis 1, die auf dem Rangierwerk so umgestellt werden müssen, dass sie rechts in einer neuen Zusammenstellung herausfahren können.
- Gibt es für einen Zug mit n = 5 Waggons (5, 4, 3, 2, 1) Rangiermöglichkeiten, so dass die folgenden Waggon-Zusammenstellungen auf der rechten Seite entstehen?
  1) 5,4,3,2,1
  2) 5,3,1,2,4
  3) 4,2,5,3,1
  4) 1,4,5,3,2

Aufgabe: Rangieren

1) 5,4,3,2,1 ?
   a) Stack: Ja.
   b) Queue: Ja.
2) 5,3,1,2,4 ?
   a) Stack: Nein.
   b) Queue: Nein.
3) 4,2,5,3,1 ?
   a) Stack: Nein.
   b) Queue: Ja.
4) 1,4,5,3,2 ?
   a) Stack: Ja.
   b) Queue: Nein.

Aufgabe: Rangieren

Algorithmus $rangiere\ (n)$

Input: Int-Array c der Länge |c|=n
Output: Boolean

```
stack := Stack();
max:= 0;
for i := n-1 to 1 do
  if (c[i]>max) then
    for m := max+1 to c[i]-1 do
      stack.push(m);
    end for
    max := c[i];
  else if (c[i]<max) then
    if (stack.pop() != c[i]) then
      return false;
    end if
  end if
end for
return true;
```

Balancierte Klammerausdrücke

- Bestimme, ob die Klammern in einem String balanciert sind.

String	Balanciert
([ ])	Ja
(([ ] [ ]))	Ja
( ]) [	Nein
) (	Nein

Balancierte Klammerausdrücke

Algorithmus $balanced(n)$

Input: Character-Array c der Länge |c|=n
Output: Boolean

```
for i := 1 to n do
  if (c[i] == '(') then
    stack.push(')')
  else if (c[i] == '[') then
    stack.push(']')
  else if (stack.isEmpty() or stack.pop() != c[i]) then
    return false;
  end if
 end for
 return stack.isEmpty();
```