Alle Definitionen zusammen

!Nicht "Alle" nur die, die ich lerne!

O-Notation

$O(g) := \{ f \in \mathbb{R}_+^{\mathbb{N}} \mid \exists k \in \mathbb{N} : \exists c \in \mathbb{R}_+ : \forall n \in \mathbb{N} : (n \geq k \rightarrow f(n) \leq c \cdot g(n) \}$

$\Omega(g) := \{ f \in \mathbb{R}_+^{\mathbb{N}} \mid \exists k \in \quad \exists c \in \mathbb{R}_+ : \forall n \in \mathbb{N} (n \geq k \rightarrow f(n) \geq c \cdot g(n) \}$

$\Theta(g) := O(g) \cap \Omega(g)$

Master-Theorem

$\alpha, \beta \in \mathbb{N}$ und $\beta \leq 2$
$\delta \in \mathbb{R}$ und $\delta \leq 0$
$f : \mathbb{N} \rightarrow \mathbb{R}_+$ mit der Form

$$f(n) = \alpha \cdot f(n // \beta) + O(n^\delta)$$

1. Fall : $\alpha < \beta^\delta \rightarrow f(n) \in O(n^\delta)$
2. Fall : $\alpha = \beta^\delta \rightarrow f(n) \in O(n^\delta \cdot \log_\beta(n))$
3. Fall : $\alpha > \beta^\delta \rightarrow f(n) \in O(n^{\log_\beta(\alpha)})$

Sortierproblem

Partielle Ordnung

1. $\forall x \in S : x \leq x$ (Reflexiv)
2. $\forall x,y \in S : (x \leq y \wedge y \leq x \rightarrow x = y)$ (anti-Stimmetrie)
3. $\forall x,y,z \in S : (x \leq y \wedge y \leq z \rightarrow x \leq z)$ (transitiv)

Quasi Ordnung

1. $\forall x \in S : x \leq x$ (Reflexiv)
2. $\forall x,y,z \in S : (x \leq y \wedge y \leq z \rightarrow x \leq z)$ (transitiv)

Lineare Ordnung

1. Partiell geordnet
2. $\forall x,y \in S : (x \leq y \vee y \leq x)$ (Linear)

Totale Quasiordnung

1. Quasiordnung
2. $\forall x,y \in S : (x \leq y \vee y \leq x)$ (Linear)

Sortierproblem

gegeben sei $\langle M, \leq \rangle$ in totaler Quasiordnung.
L ist Liste aus M
gesucht ist S mit folgenden Bedingungen

1. $\forall i \in \{0, \dots, \mathrm{len}(S) - 2\} : S[i] \leq S[i+1]$
2. $\forall x \in M : \mathrm{count}(x, L) = \mathrm{count}(x, S)$

Sortier-Algorithmen:

Insertion Sort:

$\mathrm{sort}([]) = []$
$\mathrm{sort}([x]+R) = \mathrm{insert}(x, \mathrm{sort}(R))$
$\mathrm{insert}(x, []) = [x]$
$x \leq y \rightarrow \mathrm{insert}([x], [y]+R) = [x]+[y]+R$
$x > y \rightarrow \mathrm{insert}([x], [y]+R) = [y] + \mathrm{insert}(x, R)$

best case : $O(n)$
worst case : $\frac{1}{2} \cdot n^2 + O(n)$
average case : $\frac{1}{4} \cdot n^2 + O(n)$

Selection Sort

$\mathrm{sort}([]) = []$
$x := \min(L)$
$\mathrm{sort}(L) = [x] + \mathrm{sort}(\mathrm{delete}(x, L))$
$\mathrm{delete}(x, []) = []$
$\mathrm{delete}(x, [x]+R) = R$
$x \neq y \rightarrow \mathrm{delete}(x, [y]+R) = \mathrm{delete}(x, R)$
$\min([]) = \infty$
$\min([x]+R) = \min(x, \min(R))$
$\min(x,y) \begin{cases} x & \text{falls } x < y \\ y & \text{sonst} \end{cases}$

immer $\frac{1}{2} \cdot n^2 + O(n)$

Merge Sort

$n := \mathrm{len}(L)$
$n < 2 \rightarrow \mathrm{sort}(L) = L$
$n \geq 2 \rightarrow \mathrm{sort}(L) = \mathrm{merge}(\mathrm{sort}(L[:n//2]), \mathrm{sort}(L[n//2:]))$
$\mathrm{merge}(L_1, []) = L_1$
$\mathrm{merge}([], L_2) = L_2$
$L_1 \neq [] \wedge L_2 \neq [] \wedge x_1 \leq x_2$
$\rightarrow \mathrm{merge}([x_1|R_1], [x_2|R_2]) = [x_1] + \mathrm{merge}(R_1, [x_2|R_2])$
$L_1 \neq [] \wedge L_2 \neq [] \wedge \neg(x_1 \leq x_2)$
$\rightarrow \mathrm{merge}([x_1|R_1], [x_2|R_2]) = [x_2] + \mathrm{merge}([x_1|R_1], R_2)$

immer $O(n \cdot \log_2(n))$

oder $2 \cdot f(n//2) \cdot O(n)$

Quicksort

$\mathrm{sort}([]) = []$
$\mathrm{sort}([x]+R) = \mathrm{sort}([y \in R \mid y \leq x]) + [x] + \mathrm{sort}([y \in R \mid y > x])$

worst case $O(n^2)$
aver. case $\underbrace{2 \cdot \ln(2)}_{1{,}39} \cdot n + O(n \cdot \log_2(n))$

Counting sort

1. Zählphase
2. Indixing-Phase
3. Distribution-Phase

immer $O(n)$

Heapsort

$Nil.\mathrm{toList}() = []$
$h \neq Nil \wedge \langle p, \_ \rangle = h.\mathrm{top}()$
$\rightarrow h.\mathrm{toList}() = [p] + h.\mathrm{remove}().\mathrm{toList}()$

immer $O(n \cdot \log_2(n))$

Abstrakt Datentypen (ADT)

Formale Definition von ADT's

$D = \langle N, P, F_s, T_s, Ax \rangle$

N = Name des ADT's als String
P = Typparameter als Menge
F_s = Funktionsspezifikationen als Menge (für
T_s = Typspezifikationen als Menge
Ax = Axiome

Vorteile von ADT's:
1) Wiederverwendbar
2) Austauschbar
3) Abstrahieren von der Implementierung

Stack

N = "stack"
$P = \{ \text{Element} \}$
$F_s = \{ \text{stack, push, pop, top, isEmpty} \}$
T_s :
stack : Stack
push : Stack $\times$ Element $\rightarrow$ Stack
pop : Stack $\rightarrow$ Stack $\cup \{\Omega\}$
top : Stack $\rightarrow$ Element $\cup \{\Omega\}$
isEmpty : Stack $\rightarrow \mathbb{B}$

Ax:
$S.\mathrm{push}(x).\mathrm{isEmpty}() = \text{false}$
$S.\mathrm{push}(x).\mathrm{top}() = x$
$S.\mathrm{push}(x).\mathrm{pop}() = S$
$\mathrm{Stack}().\mathrm{isEmpty}() = \text{true}$
$\mathrm{Stack}().\mathrm{top}() = \Omega$
$\mathrm{Stack}().\mathrm{pop}() = \Omega$

Map

N = "Map"
$P = \{ \text{key, Value} \}$
$F_s = \{ \text{map, insert, delete, find} \}$
T_s:
map : Nil
insert : Map $\times$ key $\times$ value $\rightarrow$ Map
delete : Map $\times$ key $\rightarrow$ Map
find : Map $\times$ key $\rightarrow$ Value $\cup \{\Omega\}$

Ax :
$\mathrm{Map}().\mathrm{find}(k) = \Omega$
$\mathrm{Map}().\mathrm{insert}(k,v).\mathrm{find}(k) = v$
$k_1 \neq k_2 \rightarrow \mathrm{Map}().\mathrm{insert}(k_1,v).\mathrm{find}(k_2) = m.\mathrm{find}(k_2)$
$\mathrm{Map}().\mathrm{delete}(k).\mathrm{find}(k) = \Omega$
$k_1 \neq k_2 \rightarrow \mathrm{Map}.\mathrm{delete}(k_1).\mathrm{find}(k_2) = m.\mathrm{delete}(k_2)$

geordnete Binärbäume

Menge der B (Binärbäume)

$Nil \in B$
$\mathrm{Node}(k,v,l,r) \in B$ g.d.w.
$k \in$ keys
$v \in$ Values
$l, r \in B$
l, r sind Unterbäume von $\mathrm{Node}(\_,\_,\_)$
Alle Element aus $l < k$
" " " $r > k$

map : Nil
insert : $B \times k \times v \rightarrow B$
delete : $B \times k \rightarrow B$
find : $B \times k \rightarrow v \cup \{\Omega\}$
delMin: $B \rightarrow B \times k \times v$

INSERT

$Nil.\mathrm{insert}(k,v) = \mathrm{Node}(k, v, Nil, Nil)$
$\mathrm{Node}(k_1, v_1, l, r).\mathrm{insert}(k_1, v_2, l, r) = \mathrm{Node}(k_1, v_2, l, r)$
$k_1 < k_2 \rightarrow \mathrm{Node}(k_1, v_1, l, r).\mathrm{insert}(k_2, v_2, l, r) = r.\mathrm{insert}(k_2, v_2)$
$k_1 > k_2 \rightarrow$ $l.\mathrm{insert}(k_2, v_2)$

DELETE

$Nil.\mathrm{delete}(k) = Nil$
$\mathrm{Node}(k, v, Nil, r).\mathrm{delete}(k) = r$
$\mathrm{Node}(k, v, l, Nil).\mathrm{delete}(k) = r$
$l, r \neq Nil \wedge r.\mathrm{delMin}() < r', k_{min}, v_{min} >$
$\mathrm{Node}(k, v, l, r).\mathrm{delete}(k) = \mathrm{Node}(k_{min}, v_{min}, l, r')$
$k_1 < k_2 \rightarrow \mathrm{Node}(k_1, v_1, l, r).\mathrm{delete}(k_2) = \mathrm{Node}(k_1, v_1, l, r.\mathrm{delete}(k_2))$
$k_1 > k_2 \rightarrow$ " " " = " $(k_1, v_1, l.\mathrm{delete}(k_2), r)$

DELMIN

$l.\mathrm{delMin}() = \langle l', k_{min}, v_{min} \rangle$
$\mathrm{Node}(k, v, Nil, r).\mathrm{delMin}() = \langle r, k, v \rangle$
$\mathrm{Node}(k, v, l, r).\mathrm{delMin} = \langle \mathrm{Node}(k, v, l', r), k_{min}, v_{min} \rangle$

FIND

$Nil.\mathrm{find}(k) = Nil$
$\mathrm{Node}(k, v, l, r).\mathrm{find}(k) = v$
$k_1 < k_2 \rightarrow \mathrm{Node}(k_1, v_1, l, r).\mathrm{find}(k_2) = r.\mathrm{find}(k_2)$
$k_1 > k_2 \rightarrow$ " " $= l.\mathrm{find}(k_2)$

AVL-Bäume

Menge A der AVL Bäume

$Nil \in A$
$\mathrm{Node}(k, v, l, r) \in A$ g.d.w.

1. $\mathrm{Node}(k, v, l, r) \in B$
2. $l, r \in A$
3. $|l.\mathrm{height}() - r.\mathrm{height}()| \leq 1$

insert : $A \times k \times v \rightarrow A$
delete : $A \times k \rightarrow A$
find : $A \times k \rightarrow V \cup \{\Omega\}$
delMin: $A \rightarrow A \times k \times v$
Restore: $B \rightarrow A$

INSERT

$Nil.\mathrm{insert}(k,v) = \mathrm{Node}(k, v, Nil, Nil)$
$\mathrm{Node}(k_1, v_1, l, r).\mathrm{insert}(k_1, v_2, l, r) = \mathrm{Node}(k_1, v_2, l, r)$
$k_1 < k_2 \rightarrow \mathrm{Node}(k_1, v_1, l, r).\mathrm{insert}(k_2, v_2) = \mathrm{Node}(k_1, v_1, l, r.\mathrm{insert}(k_2, v_2)).\mathrm{restore}()$
$k_1 > k_2 \rightarrow$ " " $= \mathrm{Node}(k_1, v_1, l.\mathrm{insert}(k_2, v_2), r).\mathrm{restore}()$

FIND

$Nil.\mathrm{find}(k) = Nil$
$\mathrm{Node}(k, v, l, r).\mathrm{find}(k) = v$
$k_1 < k_2 \rightarrow \mathrm{Node}(k_1, v_1, l, r).\mathrm{find}(k_2) = r.\mathrm{find}(k_2)$
$k_1 > k_2 \rightarrow$ " " $= l.\mathrm{find}(k_2)$

DELETE

$Nil.\mathrm{delete}(k) = Nil$
$r.\mathrm{delMin}() := \langle r', k_{min}, v_{min} \rangle \wedge l, r \neq Nil$
$\mathrm{Node}(k, v, l, r).\mathrm{delete}(k) = \mathrm{Node}(k_{min}, v_{min}, l, r').\mathrm{restore}()$
$\mathrm{Node}(k, v, Nil, r).\mathrm{delete}(k) = r$
$\mathrm{Node}(k, v, l, Nil).\mathrm{delete}(k) = l$
$k_1 < k_2 \rightarrow \mathrm{Node}(k_1, v_1, l, r).\mathrm{delete}(k_2) = \mathrm{Node}(k_1, v_1, l, r.\mathrm{delete}(k_2)).\mathrm{restore}()$
$= \mathrm{Node}(k_1, v_1, l.\mathrm{delete}(k_2), r).\mathrm{restore}()$

DELMIN

$\mathrm{Node}(k, v, Nil, r).\mathrm{delMin}() = \langle r, k, v \rangle$
$l.\mathrm{delMin}() := \langle l', k_{min}, v_{min} \rangle \wedge l, r \neq Nil$
$\rightarrow \mathrm{Node}(k, v, l, r).\mathrm{delMin}() = \langle \mathrm{Node}(k, v, l', r).\mathrm{restore}(), k_{min}, v_{min} \rangle$