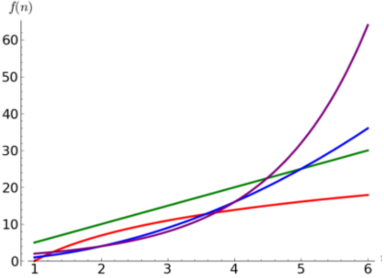
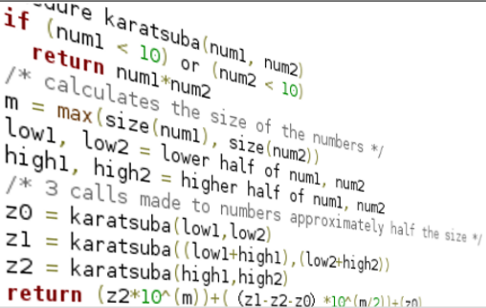
Karlsruher Institut für Technologie

Tutorium Algorithmen 1

Simon Bischof (simon.bischof2@student.kit.edu) | 11. Juni 2013

Institut für theoretische Informatik, Prof. Sanders





Wahlen der Studierendenschaft

Geht wählen!

- Wahl des Studierendenparlaments und der Fachschaftsvorstände
- Keine Bindung an bestimmte Wahlurnen
- Noch bis Freitag, 14.6., 15:00 Uhr an den meisten Urnen; bis 16:00 Uhr in der Mensa

Probeklausur

Gut:

- Mastertheorem
- Hashtabellen
- DAG

Weniger gut:

- unbeschränkte Felder
- rectMult
- Datenstrukturinvariante doppelt-verkettete Liste / Heap-Eigenschaft

Sortierte Folgen

- Folge $\langle e_1, e_2, \ldots, e_n \rangle$ mit $e_1 \leq e_2 \leq \ldots \leq e_n$
- wichtigste Funktion: M.locate(k) = address of min $\{e \in M \mid e \geq k\}$
- hier: sortierte, zyklische Liste mit Dummy ∞ + Navigationsdatenstruktur
- statische Variante: sortiertes Array

Sortierte Folgen

- Folge $\langle e_1, e_2, \ldots, e_n \rangle$ mit $e_1 \leq e_2 \leq \ldots \leq e_n$
- wichtigste Funktion: M.locate(k) = address of min $\{e \in M \mid e \geq k\}$
- hier: sortierte, zyklische Liste mit Dummy ∞ + Navigationsdatenstruktur
- statische Variante: sortiertes Array

Sortierte Folgen

- Folge $\langle e_1, e_2, \ldots, e_n \rangle$ mit $e_1 \leq e_2 \leq \ldots \leq e_n$
- wichtigste Funktion: M.locate(k) = address of min $\{e \in M \mid e \geq k\}$
- hier: sortierte, zyklische Liste mit Dummy ∞ + Navigationsdatenstruktur
- statische Variante: sortiertes Array

Sortierte Folgen

- Folge $\langle e_1, e_2, \ldots, e_n \rangle$ mit $e_1 \leq e_2 \leq \ldots \leq e_n$
- wichtigste Funktion: M.locate(k) = address of min $\{e \in M \mid e \geq k\}$
- hier: sortierte, zyklische Liste mit Dummy ∞ + Navigationsdatenstruktur
- statische Variante: sortiertes Array

Operationen

- O(log n): insert, remove, update, locate
- O(1): min, max
- O(log n+|result|): rangeSearch
- O(n): (re)build
- Weitere Möglichkeiten: concat, split, rank, select, rangeSize, …

Operationen

Karlsruher Institut für Technologie

- O(log n): insert, remove, update, locate
- O(1): min, max
- O(log n+|result|): rangeSearch
- O(n): (re)build
- Weitere Möglichkeiten: concat, split, rank, select, rangeSize, ...

Binärer Suchbaum

- Blätter: Elemente der sortierten Folge
- innere Knoten: (k:Element, l:Teilbaum, r:Teilbaum)
- über l erreichbare Blätter haben Werte $\leq$ k
- über r erreichbare Blätter haben Werte > k
- locate einfach zu implementieren

Binärer Suchbaum

- Blätter: Elemente der sortierten Folge
- innere Knoten: (k:Element, l:Teilbaum, r:Teilbaum)
- über l erreichbare Blätter haben Werte $\leq$ k
- über r erreichbare Blätter haben Werte > k
- locate einfach zu implementieren

Binärer Suchbaum

Karlsruher Institut für Technologie

- Blätter: Elemente der sortierten Folge
- innere Knoten: (k:Element, l:Teilbaum, r:Teilbaum)
- über l erreichbare Blätter haben Werte $\leq$ k
- über r erreichbare Blätter haben Werte > k
- locate einfach zu implementieren

Kreativaufgabe

Gegeben sei ein binärer Suchbaum, der auch in den inneren Knoten Elemente speichert. Jeder Knoten habe drei Zeiger, zwei Kindzeiger und einen Zeiger auf den Elternknoten (u.U. sind einige davon Nullzeiger). Die Elemente sind jedoch nicht zusätzlich in einer Liste enthalten, d.h. das nächstgrößere Element kann nicht direkt gefunden werden. Es sei ein ∞-Dummy enthalten.

- Implementiere find(x), also eine Funktion die einen Schlüssel x nimmt und entweder das passende Element aus der Datenstruktur zurückgibt, oder $\perp$ falls kein passendes Element enthalten ist. Die Laufzeit sollte in O(Baumtiefe) liegen.
- Implementiere locate(x) wie aus der Vorlesung. Die Laufzeit sollte in O(Baumtiefe) liegen.
- Implementiere locate(x) wie aus der Vorlesung. Die Baumknoten haben in diesem Fall aber keine Zeiger mehr auf ihre Elternknoten, und es soll nur O(1) zusätzlicher Speicher verwendet werden. Die Laufzeit sollte in O(Baumtiefe) liegen.

Laufzeit von locate

Karlsruher Institut für Technologie

- O(Höhe)
- Balancierter Suchbaum: worst case O(log n)
- Für entartete Bäume aber O(n)
- Sortieren mit binären Suchbäumen?

Laufzeit von locate

- O(Höhe)
- Balancierter Suchbaum: worst case O(log n)
- Für entartete Bäume aber O(n)
- Sortieren mit binären Suchbäumen?

Laufzeit von locate

- O(Höhe)
- Balancierter Suchbaum: worst case O(log n)
- Für entartete Bäume aber O(n)
- Sortieren mit binären Suchbäumen?

(a,b)-Bäume

- $a \geq 2$, $b \geq 2a-1$
- Blätter wie vorher; allerdings gleiche Tiefe
- innere Knoten mit Ausgangsgrad a..b
- Wurzel hat Ausgangsgrad 2..b (1 für $\langle\rangle$)
- bei Ausgangsgrad d gibt es d-1 "Splitter"

(a,b)-Bäume

- $a \geq 2$, $b \geq 2a-1$
- Blätter wie vorher; allerdings gleiche Tiefe
- innere Knoten mit Ausgangsgrad a..b
- Wurzel hat Ausgangsgrad 2..b (1 für $\langle\rangle$)
- bei Ausgangsgrad d gibt es d-1 "Splitter"

Locate

- Code siehe VL
- height=$h \leq 1 + \lfloor \log_a \frac{n+1}{2} \rfloor$
- Laufzeit O(b·height)
- Für $\{a, b\} \subseteq O(1)$ ist dies in O(log n)

Locate

- Code siehe VL
- height=$h \leq 1 + \lfloor \log_a \frac{n+1}{2} \rfloor$
- Laufzeit O(b·height)
- Für $\{a, b\} \subseteq O(1)$ ist dies in O(log n)

Insert

- Code siehe VL
- an der richtigen Stelle einfügen
- falls Knoten voll: spalten, Trennelement nach oben durchreichen
- evtl. rekursiv weiter

Insert

- Code siehe VL
- an der richtigen Stelle einfügen
- falls Knoten voll: spalten, Trennelement nach oben durchreichen
- evtl. rekursiv weiter

Delete

- Code siehe VL
- Element aus Blattliste löschen
- Splitter entfernen
- bei Unterlauf mit Nachbarknoten fusionieren (falls möglich)
- sonst balancieren

Delete

- Code siehe VL
- Element aus Blattliste löschen
- Splitter entfernen
- bei Unterlauf mit Nachbarknoten fusionieren (falls möglich)
- sonst balancieren

Delete

- Code siehe VL
- Element aus Blattliste löschen
- Splitter entfernen
- bei Unterlauf mit Nachbarknoten fusionieren (falls möglich)
- sonst balancieren