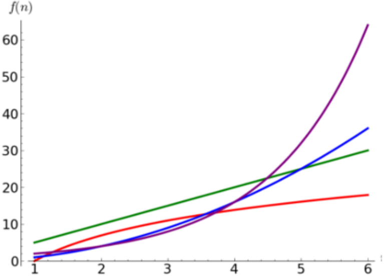
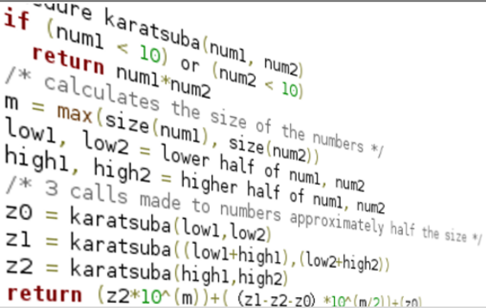
Tutorium Algorithmen 1

Simon Bischof (simon.bischof2@student.kit.edu) | 6. Mai 2013

INSTITUT FÜR THEORETISCHE INFORMATIK, PROF. SANDERS





Zum Übungsblatt

Karlsruher Institut für Technologie

- Vorsicht beim Mastertheorem
  - Immer hinschreiben, welcher Fall
  - Immer ε hinschreiben
  - Vorsicht beim Runden von $\log_a b$
  - Regularität ($\exists d \in (0, 1) : af(\frac{n}{b}) \leq df(n)$) immer zeigen (und d angeben)
  - Es gibt noch eine einfache Version des Mastertheorems (siehe VL)
- Beweise für Invarianten etwas ausführlicher
- Bei Aufgabe 3 kam nicht $\frac{1}{0}$, sondern die leere Summe (definiert als 0) raus

Zum Übungsblatt

- Vorsicht beim Mastertheorem
  - Immer hinschreiben, welcher Fall
  - Immer ε hinschreiben
  - Vorsicht beim Runden von $\log_a b$
  - Regularität ($\exists d \in (0,1) : af(\frac{n}{b}) \leq df(n)$) immer zeigen (und d angeben)
  - Es gibt noch eine einfache Version des Mastertheorems (siehe VL)
- Beweise für Invarianten etwas ausführlicher
- Bei Aufgabe 3 kam nicht $\frac{1}{0}$, sondern die leere Summe (definiert als 0) raus

Zum Übungsblatt

- Vorsicht beim Mastertheorem
  - Immer hinschreiben, welcher Fall
  - Immer ε hinschreiben
  - Vorsicht beim Runden von $\log_a b$
  - Regularität ($\exists d \in (0,1) : af(\frac{n}{b}) \leq df(n)$) immer zeigen (und d angeben)
  - Es gibt noch eine einfache Version des Mastertheorems (siehe VL)
- Beweise für Invarianten etwas ausführlicher
- Bei Aufgabe 3 kam nicht $\frac{1}{0}$, sondern die leere Summe (definiert als 0) raus

einfach verkettete Listen

- Unterschied zur doppelt verketteten Liste?

einfach verkettete Listen

- Unterschied zur doppelt verketteten Liste?
- weniger Platzverbrauch, schneller

einfach verkettete Listen

- Unterschied zur doppelt verketteten Liste?
- weniger Platzverbrauch, schneller
- eingeschränkte Schnittstelle, z.B. kein "remove"

einfach verkettete Listen

- Unterschied zur doppelt verketteten Liste?
- weniger Platzverbrauch, schneller
- eingeschränkte Schnittstelle, z.B. kein "remove"
- Invariante: Eingangsgrad = 1

wichtige Operationen

Karlsruher Institut für Technologie

- splice
  - Beachtet die Schnittstelle!
  - Function splice(a′,b,t:Handle)

$$\begin{pmatrix} a' \rightarrow next \\ t \rightarrow next \\ b \rightarrow next \end{pmatrix} := \begin{pmatrix} b \rightarrow next \\ a' \rightarrow next \\ t \rightarrow next \end{pmatrix}$$

- pushBack

wichtige Operationen

- splice
  - Beachtet die Schnittstelle!
  - Function splice(a',b,t:Handle)

$$\begin{pmatrix} a' \rightarrow next \\ t \rightarrow next \\ b \rightarrow next \end{pmatrix} := \begin{pmatrix} b \rightarrow next \\ a' \rightarrow next \\ t \rightarrow next \end{pmatrix}$$

- pushBack

wichtige Operationen

- splice
  - Beachtet die Schnittstelle!
  - Function splice(a′,b,t:Handle)

$$\begin{pmatrix} a' \rightarrow next \\ t \rightarrow next \\ b \rightarrow next \end{pmatrix} := \begin{pmatrix} b \rightarrow next \\ a' \rightarrow next \\ t \rightarrow next \end{pmatrix}$$

- pushBack
  - braucht Zeiger aufs letzte Element

Felder (Arrays)

- $A[i] = a_i$ falls $A = \langle a_0, \ldots, a_{n-1} \rangle$
- Beschränkte Felder (Bounded Arrays): bekannt
- Unbeschränkte Felder (Unbounded Arrays): pushBack, popBack, size

Unbeschränkte Felder

- Hinzufügen: size++, evtl. umkopieren (falls zu voll)
- Löschen: size--, evtl. umkopieren (falls zu leer)
- Schlechte Implementierungen brauchen für n Operationen bis zu $\Theta(n^2)$ Zeit

Unbeschränke Felder mit teilweise ungenutztem Speicher

```
UArray of Element
  w:=1 : ℕ //allocated
  n:=0 : ℕ //actual
  invariant n ≤ w < αn or (n = 0 and w ≤ 2)
  b : Array[0..w-1] of Element

  Operator [i:ℕ]:Element
    assert 0 ≤ i < n
    return b[i]

  Function size:ℕ return n
```

hinzufügen und löschen

```
Procedure pushBack(e:Element)
  if n=w
    //copy to an array of size 2n
    reallocate(2n)
  b[n]:=e
  n++

Procedure popBack()
  assert n>0
  n--
  if 4n≤w and n>0
    reallocate(2n)
```

Amortisierte Analyse

- Durchschnittliche Dauer einer Operation in einer Folge von Operationen (im worst case!)
- Sinnvoll, falls worst case garantiert selten auftritt
- KEINE average-case-Betrachtung
- Beweis z.B. mit Kontomethode (siehe VL)
- pushBack und popBack haben eine amortisierte Laufzeit von O(1)

Amortisierte Analyse

- Durchschnittliche Dauer einer Operation in einer Folge von Operationen (im worst case!)
- Sinnvoll, falls worst case garantiert selten auftritt
- KEINE average-case-Betrachtung
- Beweis z.B. mit Kontomethode (siehe VL)
- pushBack und popBack haben eine amortisierte Laufzeit von O(1)

Amortisierte Analyse

- Durchschnittliche Dauer einer Operation in einer Folge von Operationen (im worst case!)
- Sinnvoll, falls worst case garantiert selten auftritt
- KEINE average-case-Betrachtung
- Beweis z.B. mit Kontomethode (siehe VL)
- pushBack und popBack haben eine amortisierte Laufzeit von O(1)

Amortisierte Analyse

- Durchschnittliche Dauer einer Operation in einer Folge von Operationen (im worst case!)
- Sinnvoll, falls worst case garantiert selten auftritt
- KEINE average-case-Betrachtung
- Beweis z.B. mit Kontomethode (siehe VL)
- pushBack und popBack haben eine amortisierte Laufzeit von O(1)

Amortisierte Analyse

- Durchschnittliche Dauer einer Operation in einer Folge von Operationen (im worst case!)
- Sinnvoll, falls worst case garantiert selten auftritt
- KEINE average-case-Betrachtung
- Beweis z.B. mit Kontomethode (siehe VL)
- pushBack und popBack haben eine amortisierte Laufzeit von O(1)

Amortisierte Analyse

- Unbounded Arrays
- Binärzähler: Ein Binärzähler unterstütze als einzige Operation eine Inkrementierung um 1. Der Binärzähler sei am Anfang mit 0 initialisiert. Für die Implementation der Inkrementierungen darf bloß die Operation "ändere das i-te Bit", welche konstanten Aufwand besitzt, verwendet werden. Berechnen Sie die amortisierten Kosten der Inkrementierung.

Amortisierte Analyse

Karlsruher Institut für Technologie

- Unbounded Arrays
- Binärzähler: Ein Binärzähler unterstütze als einzige Operation eine Inkrementierung um 1. Der Binärzähler sei am Anfang mit 0 initialisiert. Für die Implementation der Inkrementierungen darf bloß die Operation "ändere das i-te Bit", welche konstanten Aufwand besitzt, verwendet werden. Berechnen Sie die amortisierten Kosten der Inkrementierung.

Amortisierte Analyse

- Unbounded Arrays
- Binärzähler: Ein Binärzähler unterstütze als einzige Operation eine Inkrementierung um 1. Der Binärzähler sei am Anfang mit 0 initialisiert. Für die Implementation der Inkrementierungen darf bloß die Operation "ändere das i-te Bit", welche konstanten Aufwand besitzt, verwendet werden. Berechnen Sie die amortisierten Kosten der Inkrementierung.
  Lösung: O(1)

Stapel und Queues

- effizient und einfach für bestimmte Operationen
- wenig fehleranfällig
- Stack (LIFO), Queue (FIFO) und Deque (beides)

Stapel und Queues

- effizient und einfach für bestimmte Operationen
- wenig fehleranfällig
- Stack (LIFO), Queue (FIFO) und Deque (beides)

Aufgabe zu Datenstrukturen

Beschreiben Sie eine Datenstruktur die folgendes kann:

- pushBack und popBack in O(1) im Worst-Case nicht nur amortisiert.
- Zugriff auf das k-te Element in O(log n) im Worst-Case nicht nur amortisiert.

Nehmen Sie an, dass eine Speicherallokation beliebiger Größe in O(1) geht.

Aufgabe zu Datenstrukturen 2

Beschreiben Sie eine Datenstruktur die folgendes kann:

- pushBack und popBack in O(log n) im Worst-Case nicht nur amortisiert.
- Zugriff auf das k-te Element in O(1) im Worst-Case nicht nur amortisiert.

Nehmen Sie an, dass eine Speicherallokation beliebiger Größe in O(1) geht.