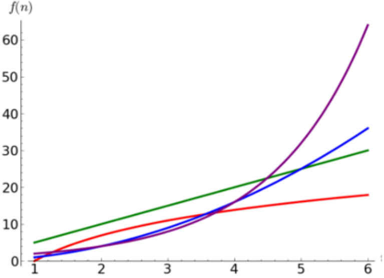
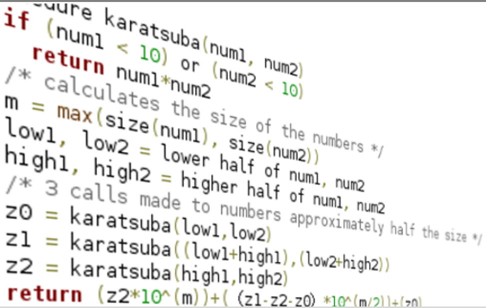
Karlsruher Institut für Technologie

Tutorium Algorithmen 1

Simon Bischof (simon.bischof2@student.kit.edu) | 29. April 2013

INSTITUT FÜR THEORETISCHE INFORMATIK, PROF. SANDERS





Zum Übungsblatt

- Achtet bitte auf formale Korrektheit
  - Besonders bei O-Notations-Beweisen ...
  - ... und vollständiger Induktion (aktuelles Blatt)
- Schaut euch nochmal den Pseudocode an
- Beachtet die Aufgabenstellung!

Zum Übungsblatt

- Achtet bitte auf formale Korrektheit
  - Besonders bei O-Notations-Beweisen ...
  - ... und vollständiger Induktion (aktuelles Blatt)
- Schaut euch nochmal den Pseudocode an
- Beachtet die Aufgabenstellung!

Zum Übungsblatt

- Achtet bitte auf formale Korrektheit
  - Besonders bei O-Notations-Beweisen ...
  - ... und vollständiger Induktion (aktuelles Blatt)
- Schaut euch nochmal den Pseudocode an
- Beachtet die Aufgabenstellung!

Wiederholung: Vollständige Induktion

- Induktionsanfang (meist $n = 0$ oder $n = 1$)
- Induktionsschritt ($n \rightsquigarrow n + 1$)

Wiederholung: Schleifeninvarianten

Beweis ähnlich zu vollständiger Induktion

- Invariante gilt am Anfang
- Invariante bleibt bei einem Durchlauf (auch dem letzten!) erhalten
- Aus Invariante und Abbruchbedingung folgt die Nachbedingung

Schleifeninvarianten beweisen nur die Korrektheit bei Terminierung!

Wiederholung: Schleifeninvarianten

Beweis ähnlich zu vollständiger Induktion

- Invariante gilt am Anfang
- Invariante bleibt bei einem Durchlauf (auch dem letzten!) erhalten
- Aus Invariante und Abbruchbedingung folgt die Nachbedingung

Schleifeninvarianten beweisen nur die Korrektheit bei Terminierung!

Rekurrenzen

$$T(n) = \begin{cases} 1 & (n \leq n_0) \\ 3T(\lfloor \frac{n}{5} \rfloor + 7) + cn^2 & (n > n_0) \end{cases}$$

Rekurrenzen

$$T(n) = \begin{cases} 1 & (n \leq n_0) \\ 3T(\lfloor \frac{n}{5} \rfloor + 7) + cn^2 & (n > n_0) \end{cases}$$

Lösung: $\Theta(n^2)$

Rekurrenzen

- $T(n) = T(\lceil \frac{n}{4} \rceil) + T(\lfloor \frac{3n}{4} \rfloor) + n$, $T(1) = 1$
- Lösung durch Auffalten des Rekursionsbaums
- Und mit $T(n) = \begin{cases} 1 & (n \leq 32) \\ T(\lceil \frac{n}{4} \rceil) + T(\lfloor \frac{3n}{4} \rfloor + 5) + n & (n > 32) \end{cases}$?

Rekurrenzen

- $T(n) = T(\lceil \frac{n}{4} \rceil) + T(\lfloor \frac{3n}{4} \rfloor) + n$, $T(1) = 1$
- Lösung durch Auffalten des Rekursionsbaums
- Und mit $T(n) = \begin{cases} 1 & (n \leq 32) \\ T(\lceil \frac{n}{4} \rceil) + T(\lfloor \frac{3n}{4} \rfloor + 5) + n & (n > 32) \end{cases}$?

Rekurrenzen

- $T(n) = T(\lceil \frac{n}{4} \rceil) + T(\lfloor \frac{3n}{4} \rfloor) + n$, $T(1) = 1$
- Lösung durch Auffalten des Rekursionsbaums
- Und mit $T(n) = \begin{cases} 1 & (n \leq 32) \\ T(\lceil \frac{n}{4} \rceil) + T(\lfloor \frac{3n}{4} \rfloor + 5) + n & (n > 32) \end{cases}$?
- Lösung für beide Rekurrenzen ist: $\Theta(n \log n)$

Rekurrenz (eine letzte)

- $T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \log n,\ T(1) = 1$
- Substituieren hilft!

Rekurrenz (eine letzte)

- $T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \log n,\ T(1) = 1$
- Substituieren hilft!

Rekurrenz (eine letzte)

- $T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \log n,\ T(1) = 1$
- Substituieren hilft!
- $T(n) \in O(\log n \cdot \log\log n)$

Einschub: Graphen

- Wichtige Begriffe: Knoten, Kanten, Relationen, (un)gerichtete Graphen, Knotengrade, Kantengewichte, knoteninduzierte Teilgraphen, Schlingen, Pfade, Kreise
- Wichtige Graphen sind z.B. K_3, K_4, K_5, $K_{3,3}$.
- Spezialfall Bäume: Zusammenhang, Bäume, Wurzeln, Wälder, Kinder, Eltern, ...

Einschub: Graphen

- Wichtige Begriffe: Knoten, Kanten, Relationen, (un)gerichtete Graphen, Knotengrade, Kantengewichte, knoteninduzierte Teilgraphen, Schlingen, Pfade, Kreise
- Wichtige Graphen sind z.B. K_3, K_4, K_5, $K_{3,3}$.
- Spezialfall Bäume: Zusammenhang, Bäume, Wurzeln, Wälder, Kinder, Eltern, ...

Einschub: Graphen

- Wichtige Begriffe: Knoten, Kanten, Relationen, (un)gerichtete Graphen, Knotengrade, Kantengewichte, knoteninduzierte Teilgraphen, Schlingen, Pfade, Kreise
- Wichtige Graphen sind z.B. K_3, K_4, K_5, $K_{3,3}$.
- Spezialfall Bäume: Zusammenhang, Bäume, Wurzeln, Wälder, Kinder, Eltern, ...

DAGs

- "Directed acyclic graph": Gerichteter kreisfreier Graph
- Test: Iterativ Knoten mit Ausgangsgrad 0 entfernen;
  G ist DAG $\Leftrightarrow$ Graph am Ende leer
- Test läuft mit passender Datenstruktur in $O(|V| + |E|)$
- Liefert auch topologische Sortierung

Datenstrukturen (für Folgen)

Doppelt verkettete Listen

Karlsruher Institut für Technologie

```
Class Handle = Pointer to Item

//one cell in a doubly linked list
Class Item of Element
e : Element
next : Handle
prev : Handle
invariant next→prev = prev→next = this
```

Dummy-Header / Wächter-Element

Karlsruher Institut für Technologie

- Führe Element mit e=$\perp$ ein, benutze es als Listenkopf
- Zusätzlich: mache die Liste zyklisch
- Invariante erfüllt, Vermeidung von Sonderfällen;
  aber: mehr Speicherplatz

Dummy-Header / Wächter-Element

- Führe Element mit e=$\perp$ ein, benutze es als Listenkopf
- Zusätzlich: mache die Liste zyklisch
- Invariante erfüllt, Vermeidung von Sonderfällen;
  aber: mehr Speicherplatz

Die Listenklasse

```
Class List of Element
// Item h is the predecessor of the first element
// and the successor of the last element.
// Pos. before any proper element
Function head : Handle; return address of h
// init to empty sequence
h = (⊥, head, head) : Item

// Simple access functions
Function isEmpty : {0,1}; return h.next = head
Function first : Handle;
     assert ¬isEmpty; return h.next
Function last : Handle;
     assert ¬isEmpty; return h.prev
```

splice

//Cut out $\langle a, \dots, b \rangle$ and insert after t

Procedure splice(a,b,t:Handle)

  assert b is not before a $\wedge$ t $\notin \langle a, \dots, b \rangle$

  //Cut out $\langle a, \dots, b \rangle$

  $a' := a \rightarrow prev$

  $b' := b \rightarrow next$

  $a' \rightarrow next := b'$

  $b' \rightarrow prev := a'$

  //insert $\langle a, \dots, b \rangle$ after t

  $t' := t \rightarrow next$

  $b \rightarrow next := t'$

  $a \rightarrow prev := t$

  $t \rightarrow next := a$

  $t' \rightarrow prev := b$

Und nun?

Wir benutzen eine einmal vorhandene Variable (static in Java):

- "freeList" enthält nicht benötigte Elemente
- "checkFreeList" sichert, dass diese nicht leer ist

Was bringt uns das?

Und nun?

Wir benutzen eine einmal vorhandene Variable (static in Java):

- "freeList" enthält nicht benötigte Elemente
- "checkFreeList" sichert, dass diese nicht leer ist

Was bringt uns das?

Und nun?

Wir benutzen eine einmal vorhandene Variable (static in Java):

- "freeList" enthält nicht benötigte Elemente
- "checkFreeList" sichert, dass diese nicht leer ist

Was bringt uns das? Kann für splice benutzt werden!

Wichtige Listenfunktionen

```
Procedure moveAfter(b,a : Handle) splice(b,b,a)
Procedure moveToFront(b : Handle) moveAfter(b, head)
Procedure moveToBack(b : Handle) moveAfter(b, last)

Procedure remove(b : Handle)
    moveAfter(b, freeList.head)
Procedure popFront remove(first)
Procedure popBack remove(last)

//(⟨a, . . . , b⟩, ⟨c, . . . , d⟩) ⇝ (⟨a, . . . , b, c, . . . , d⟩, ⟨⟩)
Procedure concat(L : List)
    splice(L.first, L.last, last)
//⟨a, . . . , b⟩ ⇝ ⟨⟩
Procedure makeEmpty freeList.concat(this)
```

Nun noch das Einfügen

```
Function insertAfter(x:Element, a:Handle) : Handle
  //make sure freeList is nonempty.
  checkFreeList
  a' := freeList.first
  moveAfter(a', a)
  a' → e := x
  return a'

Function insertBefore(x:Element,b:Handle) : Handle
    return insertAfter(x, b → prev)
Procedure pushFront(x:Element) insertAfter(x, head)
Procedure pushBack(x:Element) insertAfter(x, last)
```

Kreativaufgabe

Gegeben sei ein Array A[1..n] mit n Zahlen und eine Zahl x. Nun soll ein Paar (A[i],A[j]) gefunden werden mit A[i]+A[j]=x.

- Geben Sie eine Lösung für x=33; A=(7,15,21,14,18,3,9) an.
- Geben Sie einen effizienten Algorithmus an, der ein solches Paar in $O(n \log n)$ findet (falls es nicht existiert, soll NIL zurückgegeben werden).
- Wie sieht es aus, wenn man alle Paare finden will?

Kreativaufgabe

Karlsruher Institut für Technologie

Gegeben sei ein Array A[1..n] mit n Zahlen und eine Zahl x. Nun soll ein Paar (A[i],A[j]) gefunden werden mit A[i]+A[j]=x.

- Geben Sie eine Lösung für x=33; A=(7,15,21,14,18,3,9) an.
  Lösung: A[2]=15, A[5]=18
- Geben Sie einen effizienten Algorithmus an, der ein solches Paar in $O(n \log n)$ findet (falls es nicht existiert, soll NIL zurückgegeben werden).
- Wie sieht es aus, wenn man alle Paare finden will?

Kreativaufgabe

Gegeben sei ein Array A[1..n] mit n Zahlen und eine Zahl x. Nun soll ein Paar (A[i],A[j]) gefunden werden mit A[i]+A[j]=x.

- Geben Sie eine Lösung für x=33; A=(7,15,21,14,18,3,9) an.
  Lösung: A[2]=15, A[5]=18
- Geben Sie einen effizienten Algorithmus an, der ein solches Paar in $O(n \log n)$ findet (falls es nicht existiert, soll NIL zurückgegeben werden).
- Wie sieht es aus, wenn man alle Paare finden will?