Daten, Algorithmen und Strategien

Algorithmus versus Implementierung

Beispiel: summiere $1 + 2 + \cdots + n$

```
int sumupLoop (int n) {
   int sum = 0;
   while (n > 0) {
      sum += n;
      n--;
   }
   return sum;
}
```

```
int sumupRec (int n) {
   if (n > 0) {
      return n + sumupRec(n-1);
   } else {
      return 0;
   }
}
```

gleicher Algorithmus – unterschiedliche Implementierungen

Algorithmus versus Problem

es gilt: $1 + 2 + \cdots + n = \frac{n \cdot (n+1)}{2}$

```
int triangleNum (int n) {
   if (n > 0) {
      return n * (n + 1) / 2;
   } else {
      return (0);
   }
}
```

gleiches Problem – unterschiedliche Algorithmen

Beispiel für Datenstruktur (Stack)

```
public class IntStack {
   private int[] elems;
   private int top = 0;

   public IntStack (int max) { elems = new int[max]; }

   public void push (int elem) { elems[top++] = elem; }

   public int pop() { return (elems[--top]); }
}
```

Typische Datenstrukturen

- Array
- verkettete Liste
- binärer Baum
- Hashtabelle
- Stack

Datenstrukturen und Algorithmen

- Datenstruktur $\neq$ Implementierung der Datenstruktur
- Datenstrukturen verwenden andere Datenstrukturen
- Zugriffsoperationen bestimmen Datenstruktur
- Zugriffsoperationen durch Algorithmen festgelegt
- Datenstrukturen und Algorithmen hängen eng zusammen (wie Objektvariablen und Methoden einer Klasse)

Strategiesches Ziel: Einfachheit

Ursachen für Vernachlässigung dieses Ziels:

- Konzentration auf einzelne Teile statt dem Ganzen
- falsche Einschätzung der Komplexität
- offensichtlich $\neq$ einfach
  (einfache Lösungen erfordern mehr Wissen)

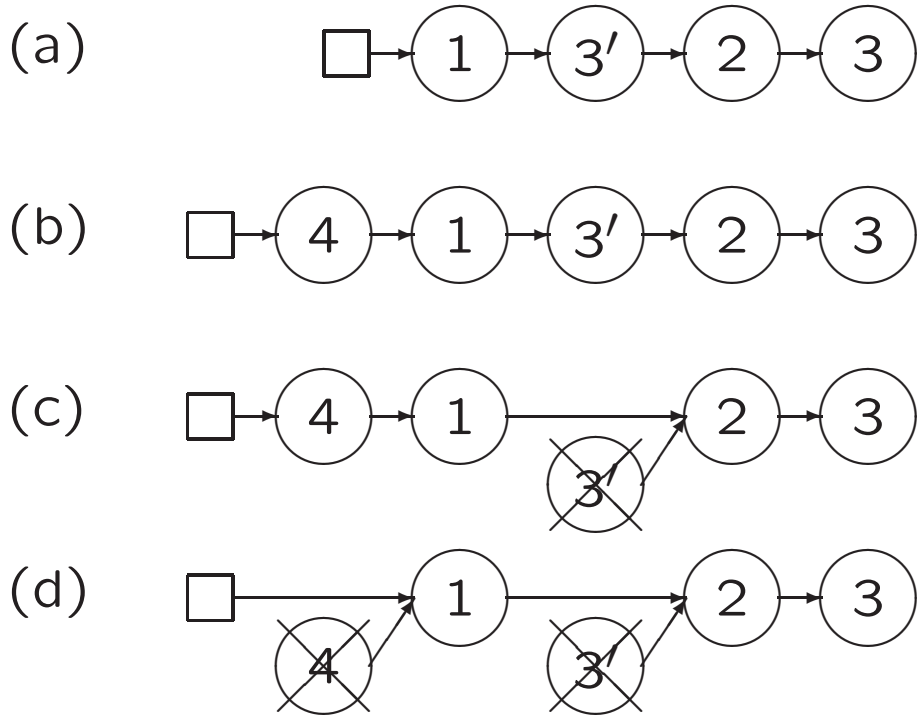
$\Rightarrow$ bewährte Strategien statt übertriebenem Effizienzdenken

Lösungsstrategien

- Teile und Herrsche
- Top Down
- Bottom Up
- Schrittweise Verfeinerung
- Verwendung vorgefertigter Teile

Rekursive Datenstrukturen

Verkettete Liste



Kopf der verketteten Liste

```
public class IntList {
    private IntListNode head = null;
    public void add (int elem) {
        head = new IntListNode (elem, head);
    }
    public boolean contains (int elem) {
        return head != null && head.contains(elem);
    }
    public void remove (int elem) {
        if (head != null) {
            head = head.remove(elem);
        }
    }
}
```

Knoten der verketteten Liste

```
class IntListNode {
    private int elem;
    private IntListNode next;

    IntListNode (int e, IntListNode n) {
        elem = e;
        next = n;
    }

    boolean contains (int e)   {/*suche e in Restliste  */}
    IntListNode remove (int e) {/*lösche e aus Restliste*/}
}
```

Rekursives Suchen und Löschen (Liste)

```
boolean contains (int e) {
    return elem == e || (next != null && next.contains(e));
}

IntListNode remove (int e) {
    if (elem == e) {
        return next;
    } else if (next != null) {
        next = next.remove(e);
    }
    return this;
}
```

Iteratives Suchen (Liste)

```
boolean contains (int e) {
    IntListNode node = this;
    do {
        if (node.elem == e) {
            return true;
        }
        node = node.next;
    } while (node != null);
    return false;
}
```

Iteratives Löschen (Liste)

```
IntListNode remove (int e) {
    if (elem == e) {
        return next;
    }
    IntListNode node = this;
    while (node.next != null) {
        if (node.next.elem == e) {
            node.next = node.next.next;
            return this;
        }
        node = node.next;
    }
    return this;
}
```

Vollständige Induktion (Beispiel)

zu zeigen: $\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$

Induktionsanfang: $\sum_{i=1}^{1} i = 1 = \frac{1 \cdot (1+1)}{2}$

Induktionsschritt: wenn für n gültig, dann auch für $n+1$

zu zeigen: $\sum_{i=1}^{n+1} i = \frac{(n+1) \cdot (n+2)}{2}$

$$\sum_{i=1}^{n+1} i = (\sum_{i=1}^{n} i) + n + 1 = \frac{n \cdot (n+1)}{2} + n + 1$$

Induktion auf Datenstruktur (Liste)

Listeneigenschaften: Datenstruktur ist Liste wenn

- zusammenhängender Graph mit genau einem Kopf
- jeder Knoten höchstens einen Nachfolger
- Knoten in umgekehrter Reihenfolge des Einfügens

Induktionsanfang: neu erzeugte Datenstruktur ist Liste

Induktionsschritt: Wenn x eine Liste ist, dann ist x auch nach Ausführung jeder Listenoperation eine Liste