II.3.2 Rekursive (dynamische) Datenstrukturen

Freitag, 25. November 2016 17:00

Rekursion kann nicht nur für Algorithmen, sondern auch für Datenstrukturen benutzt.

Bsp: "Ausdruck" in Java-Programmen ist durch rekursives Syntaxdiagramm definiert.

- Rek. def. Datenstrukturen können beliebig groß werden: <u>dynamische</u> Datenstrukturen.
- Eignen sich für Objekte, deren Größe nicht feststeht bzw. deren Größe sich im Lauf der Zeit verändert.
- Bsp: lineare Listen, doppelt verkettete Listen, Binärbäume, Mehrwegbäume, Graphen, ...
- Typische Algorithmen auf rekursiven Datenstrukturen sind auch oft rekursiv:
  - Initialisierg. v. Datenstrukturen
  - Suchen, Sortieren
  - Löschen
  - Einfügen
  - ...
- Dynamische Datenstrukturen werden meist mit Referenzen (Zeigern) realisiert: ermöglicht einfaches Einfügen + Löschen an bel. Positionen der Datenstruktur, ermöglicht bel. Wachsen der Struktur

<u>Bsp</u>: Listen

lässt sich als EBNF definieren:

Liste = Element Liste | "leer"

Bsp-Liste: (4, 17, 25)

Jedes Listenelement hat 2 Attribute:

- <u>wert</u>: des aktuellen Elements

jedes Listenelement hat 2 Attribute:
- wert: des aktuellen Elements
- next: Zeiger auf das nächste Listenelement

Klasse Element ist "rekursiv", da sie eine Attribut vom Typ Element hat.

- Wenn man nur die Klasse Element benutzt, dann wird die leere Liste durch null dargestellt.
  Nachteil: so kann man keine nicht-statischen Methoden schreiben, die auch auf leeren Listen arbeiten.

  l.fuegeEin(2) wäre nicht möglich für die leere Liste, da null keine nicht-statischen Methoden hat (NullPointerException)

- Lösung: Verwende eine weitere Klasse Liste mit einem Attribut kopf, das auf das erste Element der Liste zeigt.
  Jetzt ist die leere Liste ein Objekt l vom Typ Liste mit l.kopf == null.

Typische (rekursive) Algorithmen auf Listen

Entwerfe zuerst Schnittstellendokumentation und danach Implementierung.

Suche-Methode ist rekursiv

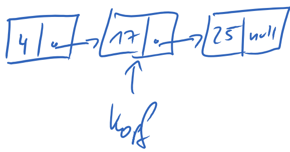
- behandelt die rekursive Datenstruktur Element
- Methodenrumpf untersucht nicht-rekursive Attribute (kopf.wert)
- Rekursive Attribute werden im rekursiven Aufruf behandelt (kopf.next)

• Rekursive Attribute werden im rekursiven Aufruf behandelt (kopf.next)

⇒ strukturelle Rekursion

• Falls kopf==null: Liste ist leer, wert kommt nicht in der Liste vor

• Auf kopf.wert darf man nur zugreifen, wenn man sichergestellt hat, dass kopf nicht null ist.



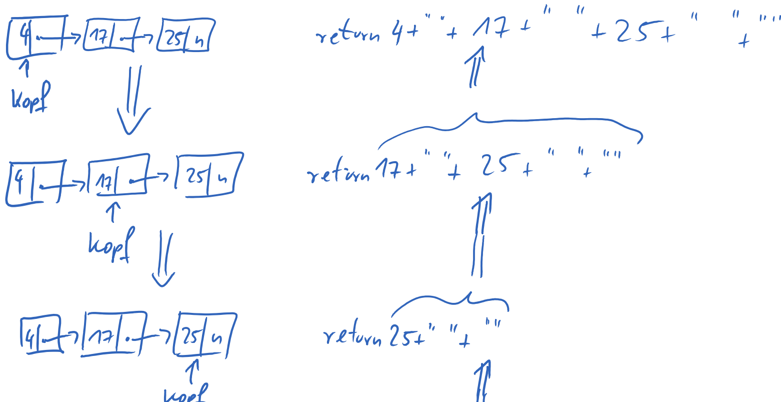
Aufruf von suche(17, kopf)

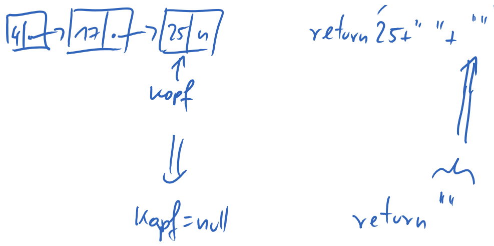
toString: erzeugt String der Werte in der Liste, in (...) eingeschlossen

durchlaufe(e): erzeugt String aller Werte in der Liste, die mit e beginnt

l.drucke(): ruft System.out.println(l) auf. Dieses führt l.toString() aus.

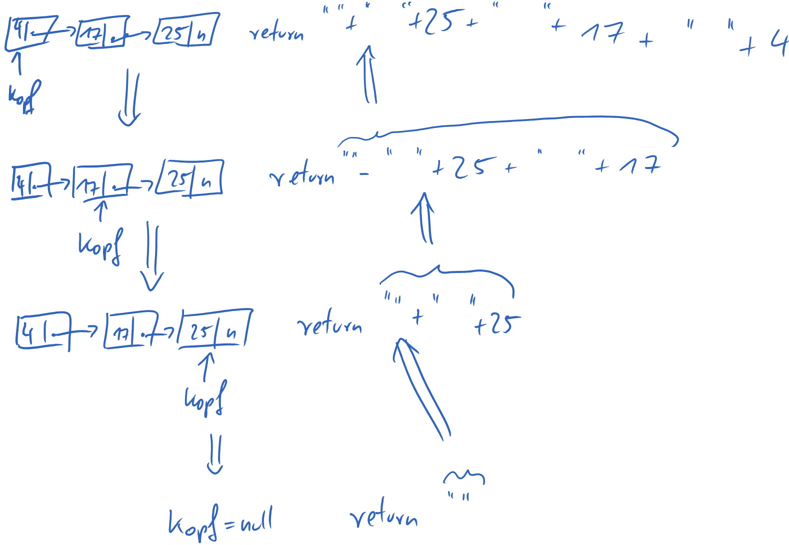
Methode "durchlaufe" ist rekursiv, ähnlich zu "suche".





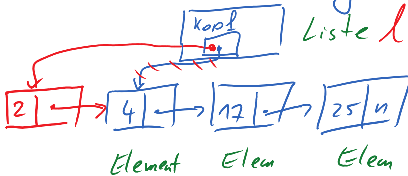
toStringRueckwaerts, druckeRueckwaerts

Wie arbeitet durchlaufeRueckwaerts?



Einfügen von Werten in Listen:

- fuegeVorneEin:
  Falls Liste bislang leer ist, erzeuge neue 1-elementige Liste (mit 1-stelligem Element-Konstruktor)
  Falls Liste bislang nicht leer:



l.fuegeVorneEin(2)

"kopf = ..." ist nötig, damit das kopf-Attribut der Liste auf das neue Element zeigt

l. fuegeSortiertEin(x):

fügt den Wert x vor dem ersten Element in der Liste l ein, das größer als x ist (sonst wird x am Ende der Liste eingefügt).
Wenn l vorher aufsteigend sortiert war, dann ist die Liste hinterher immer noch aufsteigend sortiert.

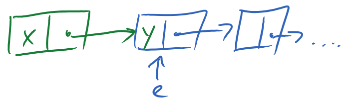
Statische Hilfsmethode:

fuegeSortiertEin(x, e)

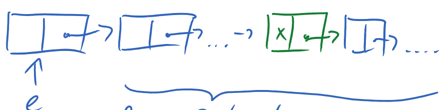
fügt x vor dem ersten größeren Wert ein in der Liste, die mit Element e beginnt. Diese Methode liefert als Ergebnis das erste Element der Liste zurück, die durch dieses Einfügen von x entsteht.

1. Fall: Wenn e leer ist, dann wird als Ergebnis zurückgeliefert: | x | null |

2. Fall Wenn der Wert von e (d.h. y) (d.h. der erste Wert in der Liste ab e) größer ist als der einzufügende Wert x, dann wird als Ergebnis zurückgeliefert:

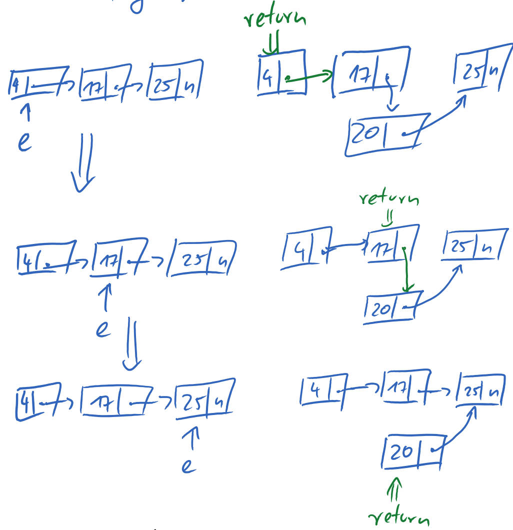


3. Fall: Der einzufügende Wert x wird hinter dem ersten Element e eingefügt.



fuegeSortiertEin(x, e.next)

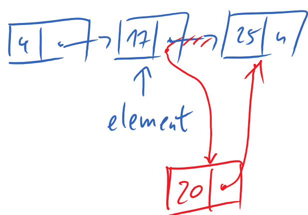
Bsp: Rufe fuegeSortiertEin auf, um 20 in die Liste (4, 17, 25) einzufügen.



Bislang haben wir immer rekursive Algorithmen geschrieben, um auf rek. Datenstrukturen zu arbeiten. Ist oftmals elegant, geht aber auch mit Schleifen.

Bsp: iterative Version von fuegeSortiertEin

- nutzt aus, dass bei der Disjunktion || das 2. Argument nur ausgewertet wird, wenn das 1. Argument false ist
- Wenn wert nicht vorne eingefügt wird, dann verwende while-Schleife, bis "element" auf das Element zeigt, <u>hinter</u> dem der neue Wert eingefügt werden soll.

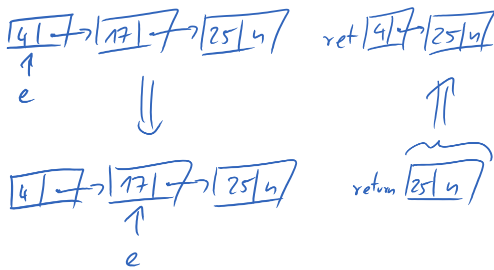


Methoden zum Löschen von Werten:

l.loesche() : löscht komplette Liste l

l.loesche(x) : löscht erstes Vorkommen von x aus der Liste l.
Hierzu wird wieder eine statische rekursive Hilfsmethode benutzt:
loesche(x, e) : löscht das erste Vorkommen von x in der Liste, die mit e beginnt.
Als Resultat wird das erste Element der dadurch entstandenen Liste geliefert.

Bsp: loesche 17 aus (4, 17, 25)



Neben Listen gibt es in der Informatik viele weitere rekursive Datenstrukturen.
z.B. Binärbaum
(Jedes Element kann nun 2 Nachfolger haben)