28. November 2013

HAW Hamburg

Dokumentation

Zu Aufgabenblatt 05 aus der Vorlesungsreihe „Algorithmen und Datenstrukturen“

Dokumentation

Zu Aufgabenblatt 05 aus der Vorlesungsreihe „Algorithmen und Datenstrukturen“

# ÜbungsAufgabe 5.1

*Um ein Element y nach n Elementen in einer Liste xs einzufügen (Methode ins(y,n,xs)), können wir rekursiv vorgehen:*

* *Wenn wir y an der Stelle n=0 in die Liste einfügen sollen, dann fügen wir y vorne an. Dies geschieht mit der Verkettungsfunktion, notiert al (y : xs). Fertig.*
* *Wenn wir y ander Stelle n>0 in die Liste xs einfügen, dann fügen wir y an der Stelle m:=n-1 in den Listenrest tl (für engl. tail) von xs ein und erhalten eine neue Liste zs. Danach fügen wir den Listenkopf hd (für engl. head) von xs vorne an diese neue Liste zs an.*

*Mit Pattern-Matching können wir diesem Algorithmus durch die folgende Definition ausdrücken:*

## Teilaufgabe 1

*Implementieren Sie dir Einfüge-Methode mit Hilfe dieses rekursiven Ansatzes!*

**public** **boolean** insert(Object n, **int** i) {

**if**(i == 0) {

head(n);

**return** **true**;

} **else** **if** (tail == **null**) {

**return** **false**;

} **else** {

*count*.increment();

tail.insert(n,i-1);

**return** **true**;

}

}

## Teilaufgabe 2

*Vergleichen Sie Ihre iterative Formulierung mit Ihrer rekursiven Variante in Bezug auf die Zeitkomplexität!*

Da unsere bisherige Variante auch rekursiv ist, ist der Zeitaufwand der gleiche. Bei einer Implementationsgröße von 100 beträgt in beiden Implementationen der Zeitaufwand 4950 Zeiteinheiten.

## Teilaufgabe 3

*Testen Sie Ihre Methode, indem Sie die Testmethoden aus Blatt 1, Aufgabe 1.2.6 erneut verwenden: Erzeugen Sie also wieder zufällig Listen, die Sie an zufälligen, aber existierenden Positionen befüllen.  
Führen Sie die Einfügeoperationen diesmal doppelt aus: an einer Liste einmal mit der iterativen und an einer zweiten Liste mit der rekursiven Methode.  
Testen Sie diesmal, ob bei beiden Implementationsvarianten am Ende die gleiche Liste erzeugt wurde.*

Es wird die gleiche Liste erzeugt. Es sind lediglich zwei verschiedene Implementationsverfahren für das *insert*.   
Hier die gemessenen Laufzeiten als Diagramm dargestellt:

Wie sich leicht erkennen lässt ist die rekursive Variante marginal schneller (ca. 5 Zeiteinheiten) und ist im Rahmen der Schätzungen durch O-Klassen zu vernachlässigen.

# Übungsaufgabe 5.2

*Gegeben sei die Funktion:*

## Teilaufgabe 1

*Bestimmen Sie f(n) für n=0,...,9*

f(0)=1  
f(1)=1  
f(2)=1  
f(3)=6  
f(4)=11  
f(5)=26  
f(6)=66  
f(7)=151  
f(8)=361  
f(9)=861

## Teilaufgabe 2

*Implementieren Sie eine rekursive Prozedur, die f(n) berechnet.*

**public** **static** **int** fancyFunction(**int** n) {

//arrayNormal.set(n, arrayNormal.get(n)+1);

**int** accu = 0;

**if** (n < 3) {

accu = 1;

} **else** {

accu = *fancyFunction*(n-1) + 2\**fancyFunction*(n-2) + 3\**fancyFunction*(n-3);

}

**return** accu;

}

## Teilaufgbe 3

*Bestimmen Sie den Wert von n, bis zu dem Sie f(n) berechnen können (ohne Überlauf o.ä.).*

Ab einem n von 27 erschien ein wilder Integer-Overflow und verwandelte eine große Zahl in eine negative Zahl mit auch vielen Stellen.

## Teilaufgabe 4

*Legen Sie ein Array A an, in dem Sie in A[i] mitzählen, wie oft die Methode mit dem Argument i aufgerufen wurde.*  
Beispiel: Für n = 19

**int** result = Methods.*fancyFunction*(n-1);

**int** n = 20;

**for** (**int** i = 0; i < n; i++) {

Methods.*array*.add(i, 0);

}

[10609, 16377, 19513, 10609, 5768, 3136, 1705, 927, 504, 274, 149, 81, 44, 24, 13, 7, 4, 2, 1, 1]

## Teilaufgabe 5

*Entwerfen Sie eine alternative Methode, bei der Sie die mehrfachen Aufrufe umgehen, z.B. durch Zwischenspeicherung oder durch eine nicht-rekursive Berechnungsreihenfolge.*

Wir haben uns für die Zwischenspeicherung der berechneten Zwischenergebnisse entschieden:

**public** **static** **int** fancyFunctionOpt(**int** n) {

*fancyArray* = **new** ArrayList<Integer>();

**for**(**int** i = 0; i < n+1; i++) {

*fancyArray*.add(-1);

}

**return** *fancyHelper*(n);

}

**private** **static** **int** fancyHelper(**int** n) {

//arrayFancy.set(n, arrayFancy.get(n)+1);

**if**(*fancyArray*.get(n) != -1) {

**return** *fancyArray*.get(n);

} **else** **if** (n < 3) {

*fancyArray*.set(n, 1);

**return** 1;

} **else** {

*fancyArray*.set(n, 3\**fancyHelper*(n-3) + 2\**fancyHelper*(n-2) + *fancyHelper*(n-1));

**return** *fancyArray*.get(n);

}

}

## Teilaufgabe 6

*Kann man die Methode auch weiterhin (in irgendeiner Form) rekursiv formulieren, ohne aber das Problem der mehrfachen Aufrufe zu haben?*

Die Variante in Teilaufgabe 5 ist bereits rekursiv formuliert.  
Und selbst wenn sie iterativ wäre: JA! Natürlich könnte man es dann auch rekursiv implementieren. Denn alles was man iterativ implementieren kann,... kann man auch rekursiv implementieren!