Fachrichtung 6.2 — Informatik
Universität des Saarlandes
Tutorenteam der Vorlesung Programmierung 1

Programmierung 1 (Wintersemester 2012/13)

Lösungsblatt 7

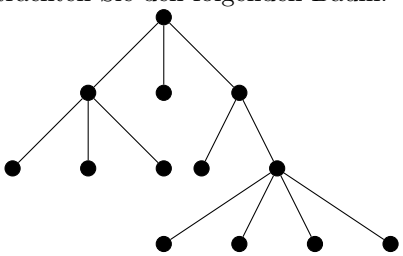
(Kapitel 7)

Hinweis: Dieses Übungsblatt enthält von den Tutoren für die Übungsgruppe erstellte Aufgaben.

Die Aufgaben und die damit abgedeckten Themenbereiche sind für die Klausur weder relevant, noch irrelevant.

1 Baumschule

Aufgabe 7.1 (*So viele Begriffe!*)
Betrachten Sie den folgenden Baum:



(a) Welche Größe hat dieser Baum? Welchen Grad, welche Tiefe, welche Stelligkeit, welche Breite? Wie viele innere Knoten besitzt dieser Baum?

(b) Malen Sie den gespiegelten Baum auf!

(c) Markieren Sie den zweiten Nachfolger der Wurzel! Welche Knoten sind der Wurzel übergeordnet?

(d) Wie viele Teilbäume besitzt dieser Baum? Wie viele Unterbäume?

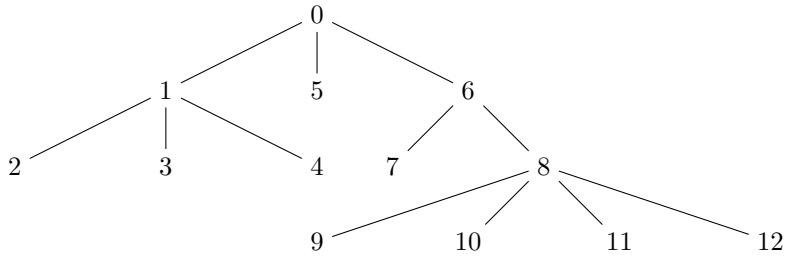
(e) Geben Sie jeweils die Prä- und Postnummerierung des obigen Baumes an!

(f) Fast geschafft!! Wie lauten Prä- und Postlinearisierung?

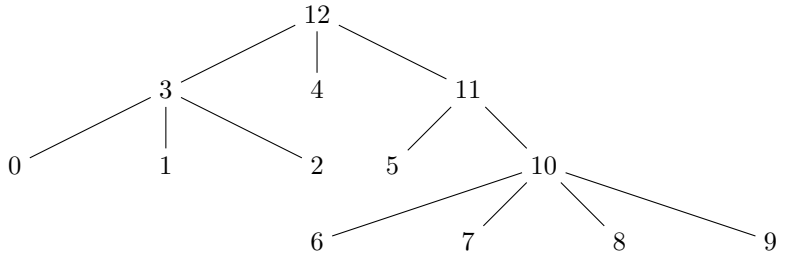
Lösung 7.1:

- Größe: 13
- Grad: 4
- Tiefe: 3
- Stelligkeit: 3
- Breite: 9
- innere Knoten: 4
- Zweiter Nachfolger: Mittlerer Knoten.
- Die Wurzel selbst ist der Wurzel übergeordnet (Definition: Es existiert eine Liste *ns* mit *a@ns = a'*).

- Unterbäume: 3
- Teilbäume: 5 (der Baum selbst, die Unterbäume, der Baum mit vier direkten Kindern)
- Präordnung:

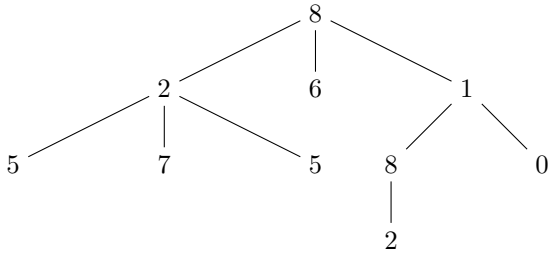


- Postordnung:



- Prälinearisierung: [3,3,0,0,0,0,2,0,4,0,0,0,0]
- Postlinearisierung: [0,0,0,3,0,0,0,0,0,0,4,2,3]

Aufgabe 7.2 (*Markierte Bäume*)
Wir betrachten nun markierte Bäume.



(a) Was ist die Gestalt dieses markierten Baums?

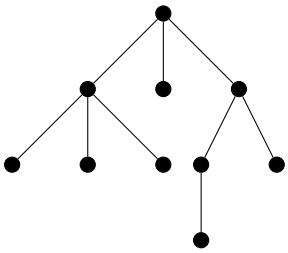
(b) Wie lautet der Kopf des Baums?

(c) Geben Sie Prä- und Postprojektion an!

(d) Wie lautet die Grenze dieses Baums?

(e) Geben Sie die erste Ebene an!

Lösung 7.2:

(a) 

(b) Der Kopf ist die Marke der Wurzel, in diesem Fall also die 8.

(c) Präprojektion: [8,2,5,7,5,6,1,8,2,0]

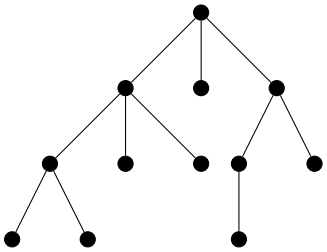
(d) Postprojektion: [5,7,5,2,6,2,8,0,1,8]

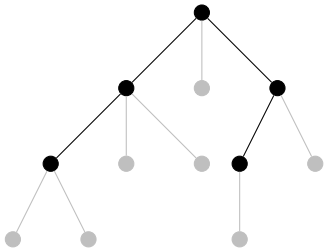
(e) Grenze: [5,7,5,6,2,0]

(f) erste Ebene: [2,6,1]

Aufgabe 7.3 (*Der Wald brennt*)

(a) Der Baum brennt. Nach dem Feuer sind alle Blätter des Baumes verschwunden, nur noch die verkohlten Äste sind übrig. Schreiben sie eine Prozedur $feuer : tree \to tree$, die alle Blätter eines Baumes entfernt. Lediglich der atomare Baum darf unverändert bleiben. Beispiel:





(b) Hat der Baum nach einer Anwendung von $feuer$ noch Blätter? Ist es möglich, dass der Baum nach einer Anwendung mehr Blätter als vorher hat?

(c) Wie oft müssen sie $feuer$ anwenden, um den Baum komplett zu zerstören? Der Baum ist komplett zerstört, wenn nur noch die Wurzel übrig ist, also $T[]$. Welche wichtige Größe spielt hier mit?

(d) Schreiben sie eine Prozedur, die mithilfe von $feuer$ diese Größe bestimmt!

Lösung 7.3:

(a)
```
fun feuer (T ts) = T (foldr (fn (T [],a) => a | (t,a) => (feuer t)::a) nil ts)
```

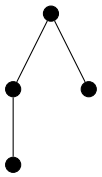
(b) Der Baum hat noch mindestens ein Blatt, maximal so viele, wie er vorher hatte (durch jedes zerstörte Blatt wird maximal ein Knoten (der Vater) zu einem neuen Blatt).

(c) *depth t* mal. Jeder Aufruf von $feuer$ verringert die Tiefe des Baumes um 1.

(d)
```
fun d (T []) = 0
  | d t      = d (feuer t) + 1
```

Aufgabe 7.4 (*Pre- und Postordnung*)

(a) Schreiben sie zwei Prozeduren $preorder : tree \to treelist$ und $postorder : tree \to treelist$, die alle Teilbäume eines Baumes in Pre- bzw. Postordnung ausgeben. Das Beispiel wird zu [T[T[T[]], T[]] , T[T[]] , T[] , T[]] in Postordnung. Sie können sich an der Aufgabe zur Pre/Postlinearisierung orientieren.



(b) Schreiben sie mithilfe dieser Prozeduren zwei Prozeduren $pre : tree \to int \to tree$ und $post : tree \to int \to tree$, die den n-ten Teilbaum eines Baumes in Pre- bzw. Postordnung liefern. Die Prozeduren sollen äquivalent zu denen aus Kapitel 7.6.1 (Teilbaumzugriff) sein.

Lösung 7.4:

(a)
```
fun preorder (T []) = [T []]
  | preorder (T ts) = (T ts) :: (foldr op@ nil (map preorder ts))
fun postorder (T []) = [T []]
  | postorder (T ts) = (foldr op@ nil (map postorder ts)) @ [T ts]
```

(b)
```
fun pre  t n = List.nth(preorder  t , n)
fun post t n = List.nth(postorder t , n)
```

2 Knobelecke

Aufgabe 7.5 (*Bäume bauen*)
Dieter hat gelernt, dass ein Baum durch die Menge seiner möglichen Adressen beschrieben werden kann. Da ihm diese Darstellung besser gefllt, sind alle seine Bäume vom Typ *int list list*. Damit sie mit Dieters Bäumen etwas anfangen können, müssen sie einen Baum aus einer Menge von Adressen konstruieren. Da Dieter ordentlich ist können sie davon ausgehen, dass sie nur gültige Bäume bekommen und dass die Adressen strikt sortiert sind. Beispiel für eine solche Liste:

```
[ [] , [1] , [1,1] , [1,2] , [2] ]
```

Schreiben sie eine Prozedur $buildTree : int\ list\ list \to tree$, mit der sie Dieters Baumdarstellung in die ihnen vertraute Darstellung konvertieren können!

Lösung 7.5:
replaceNthxsny ersetzt das n-te Listenelement durch y. $insertInTree(a, t)$ fügt einen neuen Knoten, der als Adresse a übergeben wird, in den Baum t ein. Wir bauen den Baum, indem wir alle Adressen einfügen.

```
fun buildTree adrlist = let
      fun replaceNth nil n y = raise Subscript
        | replaceNth (x::xr) 0 y = y::xr
        | replaceNth (x::xr) n y = x::(replaceNth xr (n-1) y)
      fun insertInTree(nil, t) = t
        | insertInTree([a], T ts) = T (ts @ [T[]])
        | insertInTree(a::ar, T ts) = T (replaceNth ts (a-1) (insertInTree(ar, List.nth(ts, a-1))))
in foldl insertInTree (T[]) adrlist end
```

Aufgabe 7.6 (*Baumschüler Dieter Schlau, heute mal ganz linear*)

(a) Schreiben Sie eine Prozedur $postlin : tree \to int\ list$, die die Postlinearisierung eines Baumes bestimmt.

(b) Und nun zurück: Schreiben Sie eine Prozedur $parsePostLin : int\ list \to tree$, die aus einer Postlinearisierung den Baum rekonstruiert.

(c) Für Wahnsinnige:

Schreiben Sie Prozeduren $encode : tree \to int$ und $decode : int \to tree$, so dass für alle Bäume t gilt: $decode\ (encode\ t) = t$

Nutzen Sie dabei, dass es in SML keine obere Grenze für *int*s gibt.

Lösung 7.6:

(a)
```
fun postlin (T ts) = List.concat (map postlin ts) @ [length ts]
```

(b)
```
fun popStack n stack = iter n (stack, nil) (fn (stack, res) => (tl stack, hd stack :: res))

fun parsePostLinToStack stack  nil      = stack
  | parsePostLinToStack stack (x::rlin) = let
                                            val (rstack, children) = popStack x stack
                                          in
                                            parsePostLinToStack (T children :: rstack) rlin
                                          end

fun parsePostLin lin = case parsePostLinToStack nil lin of
                         [t] => t
                       | _  => raise Overflow
```

(c)
```
fun max xs = foldl Int.max (hd xs) xs

fun encListBase base  nil    = 0
  | encListBase base (x::xr) = x+1 + base * encListBase base xr

fun decListBase base 0 = nil
  | decListBase base n = let val (r,x) = (n div base, n mod base)
                         in  (x-1) :: decListBase base r
                         end

fun encList xs = let val base = max xs + 2
                 in  (base, encListBase base xs)
                 end

fun decList (base, enc) = decListBase base enc

fun decTup' (a,b) 0 = (a,b)
  | decTup' (a,b) x = if a = 0 then decTup' (b+1, a ) (x-1)
                               else decTup' (a-1,b+1) (x-1)

fun decTup x = decTup' (0,0) x

fun encTup (a,b) = ((a+b) * (a+b+1) div 2) + b

fun encode t    = encTup (encList (postlin t))
fun decode code = parsePostLin (decList (decTup code))
```

Aufgabe 7.7 (*Für Experten mit viel zu viel Zeit*)
Die Baumdarstellung des Interpreters ist für Menschen nur sehr schlecht lesbar. Schreiben sie eine Prozedur $printTree : tree \to unit$, die einen Baum lesbar im Interpreter ausgibt. Verwenden sie dazu die Prozedur $print : string \to unit$, die einen String ausgibt.

Lösung 7.7:
Aufgrund des Umfangs (knapp über einer Seite) wird die Lösung hier nicht abgedruckt.