SI-möte #8, Programkonstruktion och datastrukturer
Lösningsförslag

Elias Castegren
elca7381@student.uu.se

25 januari 2010

Övningar

1.
Stack

$i) < 9 > \Longrightarrow < 9, 5 > \Longrightarrow < 9, 5, 1 > \Longrightarrow < 9, 5, 1, 2 > \Longrightarrow < 9, 5, 1, 2, 3 >$

$ii) < 9, 5, 1, 2 > \Longrightarrow < 9, 5, 1 >$

$iii) < 9, 5, 1, 4 > \Longrightarrow < 9, 5, 1, 4, 1 > \Longrightarrow < 9, 5, 1, 4, 1, 3 > \Longrightarrow < 9, 5, 1, 4, 1, 3, 7 >$

$iv) < 9, 5, 1, 4, 1, 3 >$

FIFO-kö

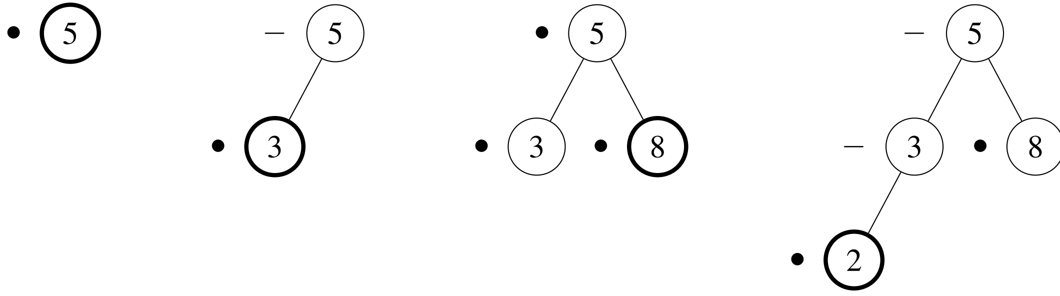
$i) < 9 > \Longrightarrow < 9, 5 > \Longrightarrow < 9, 5, 1 > \Longrightarrow < 9, 5, 1, 2 > \Longrightarrow < 9, 5, 1, 2, 3 >$

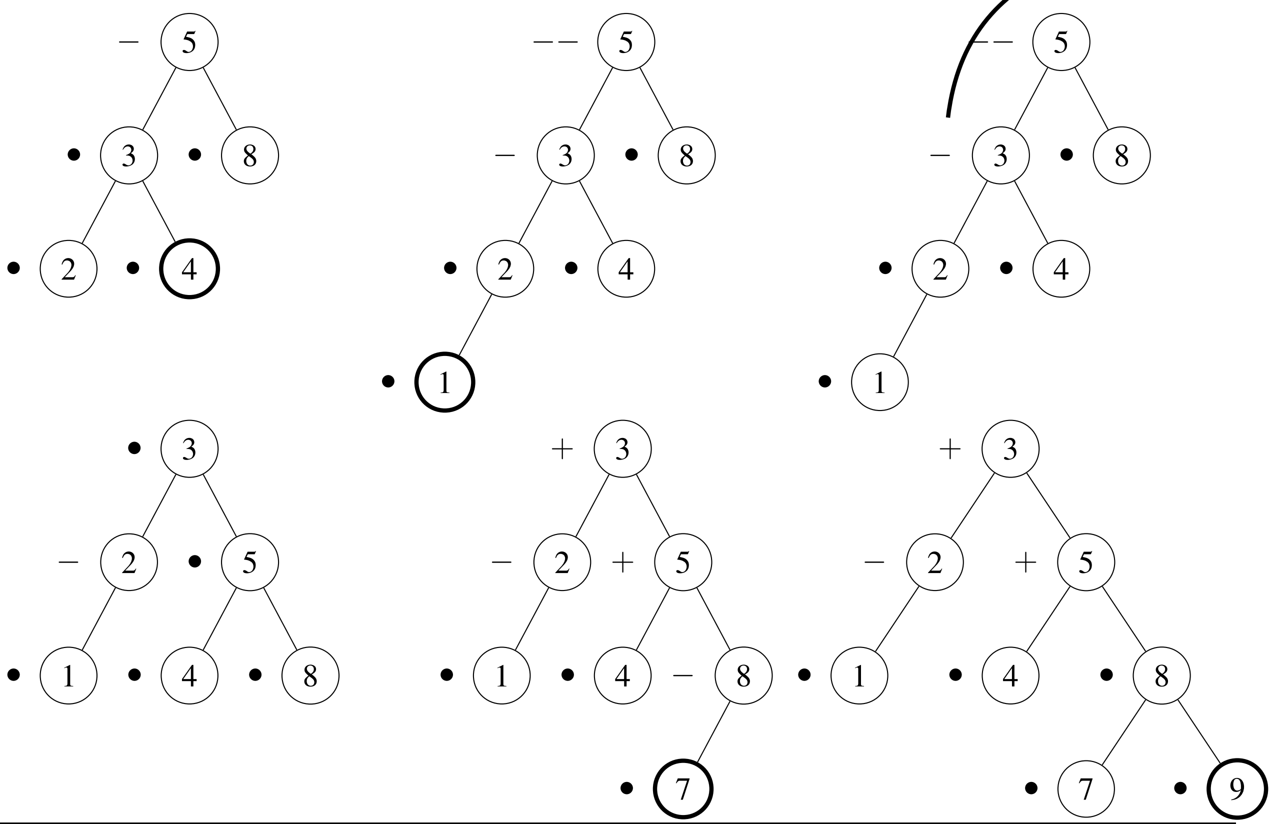
$ii) < 5, 1, 2, 3 > \Longrightarrow < 1, 2, 3 >$

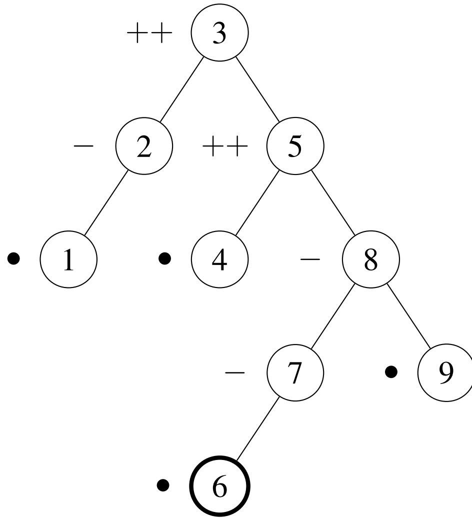
$iii) < 1, 2, 3, 4 > \Longrightarrow < 1, 2, 3, 4, 1 > \Longrightarrow < 1, 2, 3, 4, 1, 3 > \Longrightarrow < 1, 2, 3, 4, 1, 3, 7 >$

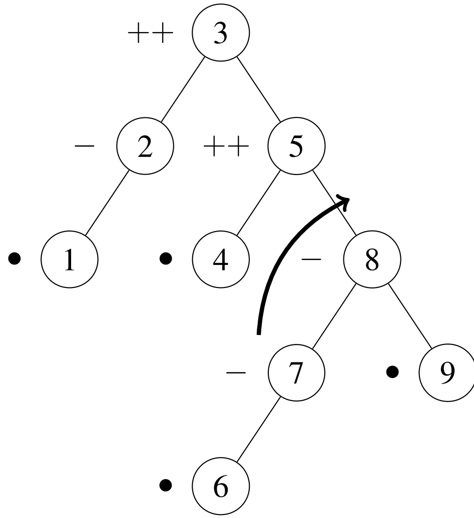
$iv) < 2, 3, 4, 1, 3, 7 >$

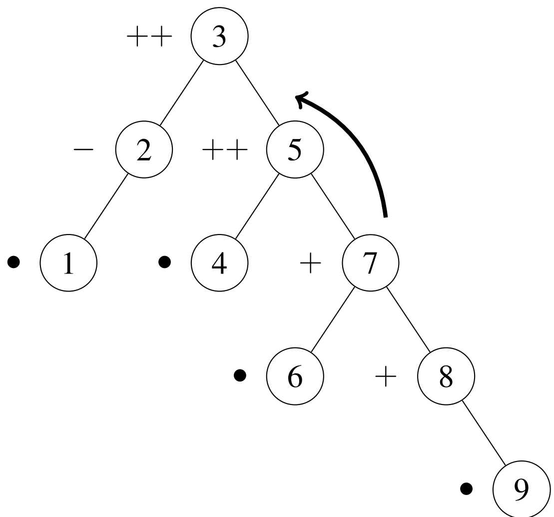
2.

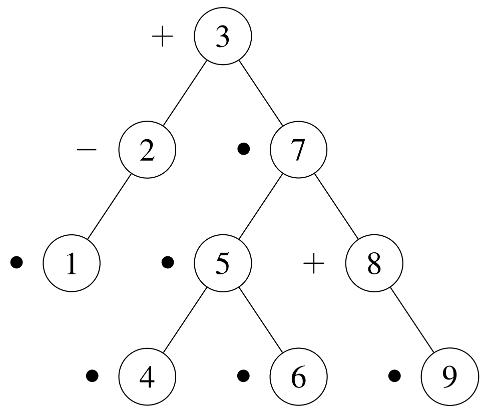




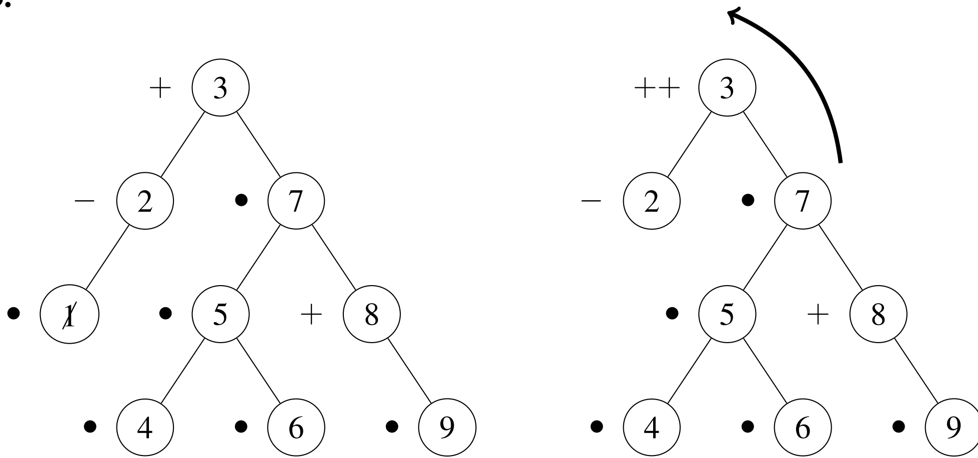

++ 3
− 2
++ 5
• 1
• 4
− 8
− 7
• 9
• 6

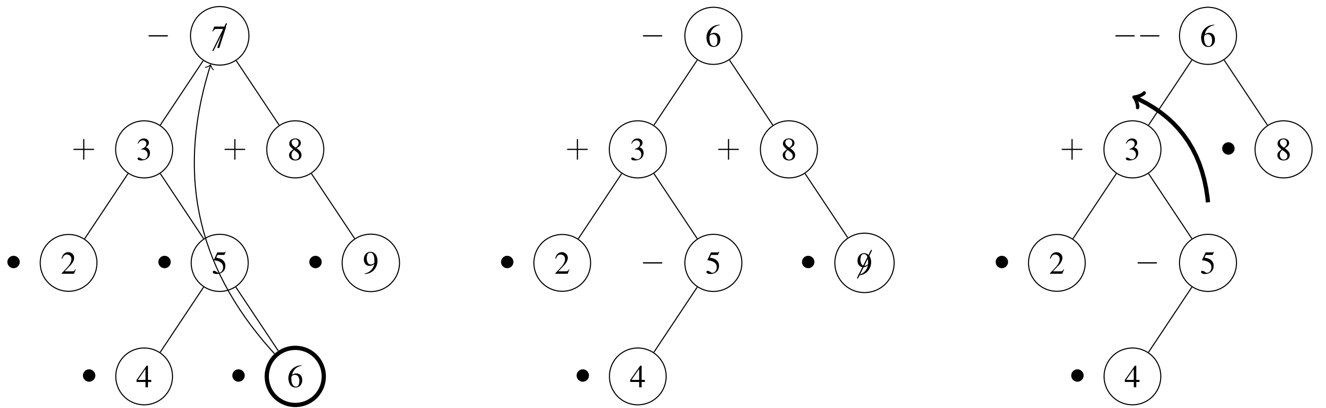

++ 3
− 2
++ 5
• 1
• 4
− 8
− 7
• 9
• 6

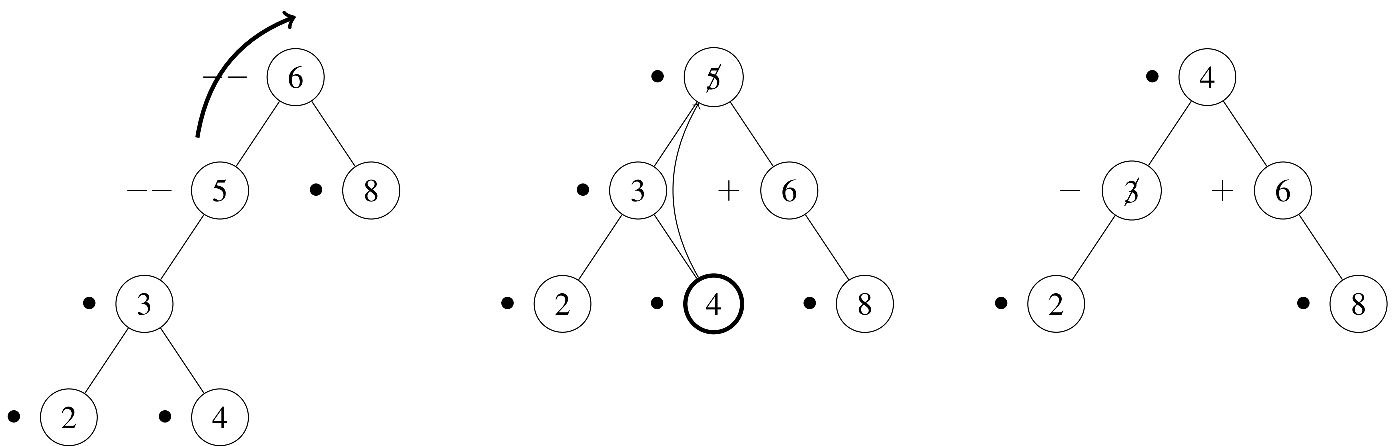

++ 3
− 2
++ 5
• 1
• 4
+ 7
• 6
+ 8
• 9

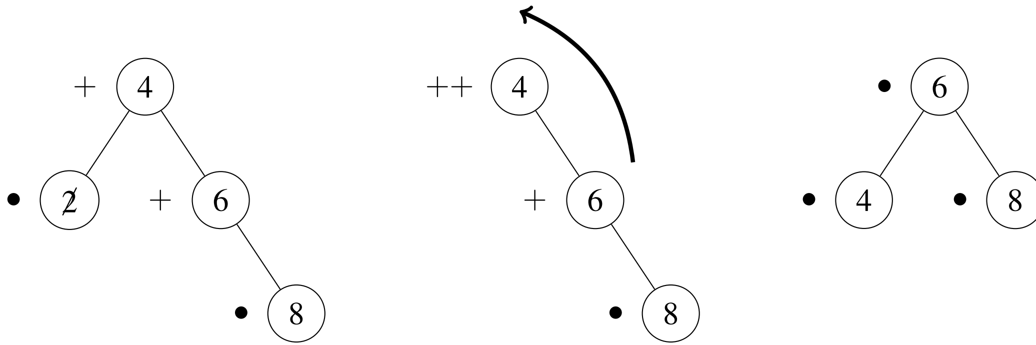

+ 3
− 2
• 7
• 1
• 5
+ 8
• 4
• 6
• 9

3.









4.

```
datatype 'a tree = Void | Node of 'a * 'a tree * 'a tree;

(* treeHeight(t)
   TYPE: 'a tree -> int
   PRE: ()
   POST: höjden av t
   EXAMPLES: treeHeight(Node(1, Node(2, Void, Void) Void)) = 2
*)
(*VARIANT: antalet noder i t*)
fun treeHeight(Void) = 0
  | treeHeight(Node(_,L,R)) = 1 + Int.max(treeHeight(L), treeHeight(R));
```

Om man tänker lite på hur funktionen fungerar så inser man att varje nod i trädet kommer besökas exakt en gång, vilket tyder på att tidskomplexiteten borde vara $\Theta(n)$, där n är antalet noder i `t`. Om man antar att trädet är balanserat kan man ställa upp följande rekursion:

$$T(n) = \begin{cases} \Theta(1) & \text{om } n = 0 \\ 2T(\frac{n}{2}) + \Theta(1) & \text{om } n > 0 \end{cases}$$

Applicerar man sedan Master Theorem får man att $n^{\log_2 2} = n^1 = n$, och eftersom det finns $\epsilon > 0$ så att $\frac{n}{k} > n^\epsilon$ för konstanta k (fall 1 av Master Theorem) så är tidskomplexiteten $\Theta(n)$.

5.

```
(* inOrder(t)
   TYPE: 'a tree -> 'a list
   PRE: ()
   POST: en lista med noderna i t i den ordning de besöks med en
         inorder-traversering
   EXAMPLES: inOrder(t) = [1,2,3,4,5,6,7,8,9]
             (där t är AVL-trädet från uppgift 2)

(*VARIANT: antalet noder i t*)
fun inOrder(Void) = []
  | inOrder(Node(e, L, R)) = inOrder(L) @ (e::inOrder(R));
```

Ifall det blir en pre-, in- eller postorder-traversering beror på var man sätter in elementet `e` i varje steg. För pre- och postorder-traverseringar skulle funktionskroppen bli
`e::inOrder(L) @ inOrder(R)`
respektive
`inOrder(L) @ inOrder(R) @ [e]`

I varje steg utförs en (eller två) listsammanslagning (append) med linjär tidskomplexitet. Om man antar att trädet är balanserat kan man ställa upp följande rekursion:

$$T(n) = \begin{cases} \Theta(1) & \text{om } n = 0 \\ 2T(\frac{n}{2}) + \Theta(n) & \text{om } n > 0 \end{cases}$$

Applicerar man Master Theorem får man att $n^{\log_2 2} = n = \Theta(n)$ (fall 2 av Master Theorem), alltså att tidskomplexiteten är $T(n) = \Theta(n \log n)$. Eftersom varje nod besöks exakt en gång skulle man vilja att komplexiteten var linjär som i uppgift 4. Problemet ligger i listsammanslagningen som ger den irriterande $\Theta(n)$-termen i rekursionen. Ett sätt att bli av med den för en inorder-traversering är att besöka alla noder i omvänd ordning och i varje steg lägga in noden först i en ackumulatorlista:

```
fun inOrder'(Void, ack) = ack
  | inOrder'(Node(e, L, R), ack) = inOrder(L, e::inOrder(R, ack));

fun inOrder(t) = inOrder'(t, []);
```

Den resulterande rekursionen blir då istället

$$T(n) = \begin{cases} \Theta(1) & \text{om } n = 0 \\ 2T(\frac{n}{2}) + \Theta(1) & \text{om } n > 0 \end{cases}$$

Vilket ger $T(n) = \Theta(n)$ enligt samma resonemang som i uppgift 4.