# SI-möte #7, Programkonstruktion och datastrukturer
*Lösningsförslag*

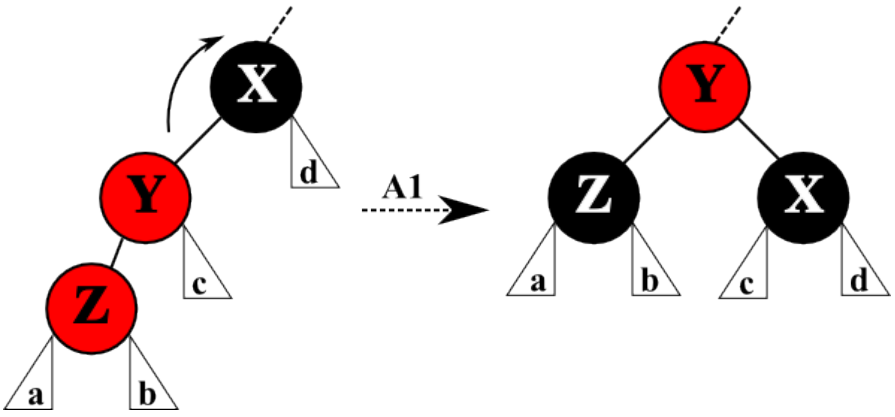
Elias Castegren & Kristiina Ausmees
elca7381@student.uu.se ‖ krau6498@student.uu.se

16 December 2011

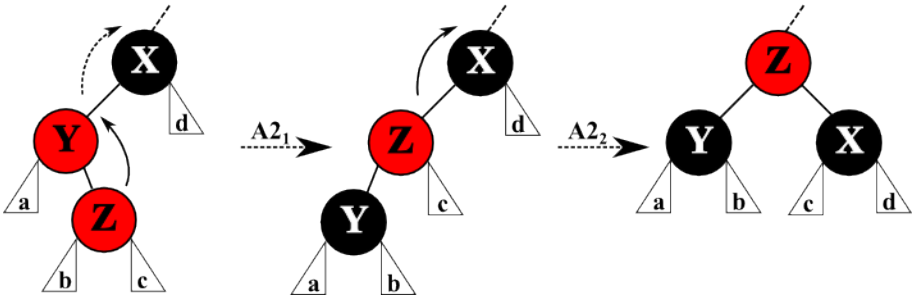
## Övningar

**1.**
Invarianterna för ett red-black tree är att varje väg från roten till ett löv har lika många svarta noder, och att ingen röd nod har en röd förälder. När den senare invarianten bryts löser man det med hjälp av rotationer. Lägg märke till att de nya barnen efter en rotation alltid färgas svarta. Det finns fyra olika fall som kan dyka upp:

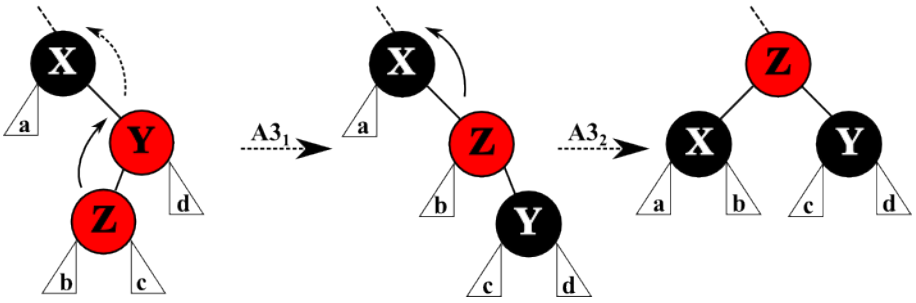
**A1**
När de två röda noderna båda är vänsterbarn så räcker det att rotera mot förföräldern (**X**).
a, b, c och d är godtyckliga underträd.



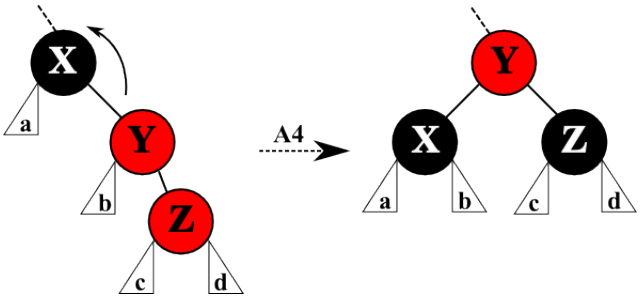
**A2**
Om den lägsta noden är ett högerbarn och dess förälder är ett vänsterbarn måste man först rotera mot föräldern (**Y**) och sen mot förföräldern (**X**):




**A3**
Speglingen av fall **A2**.




**A4**
Speglingen av fall **A1**



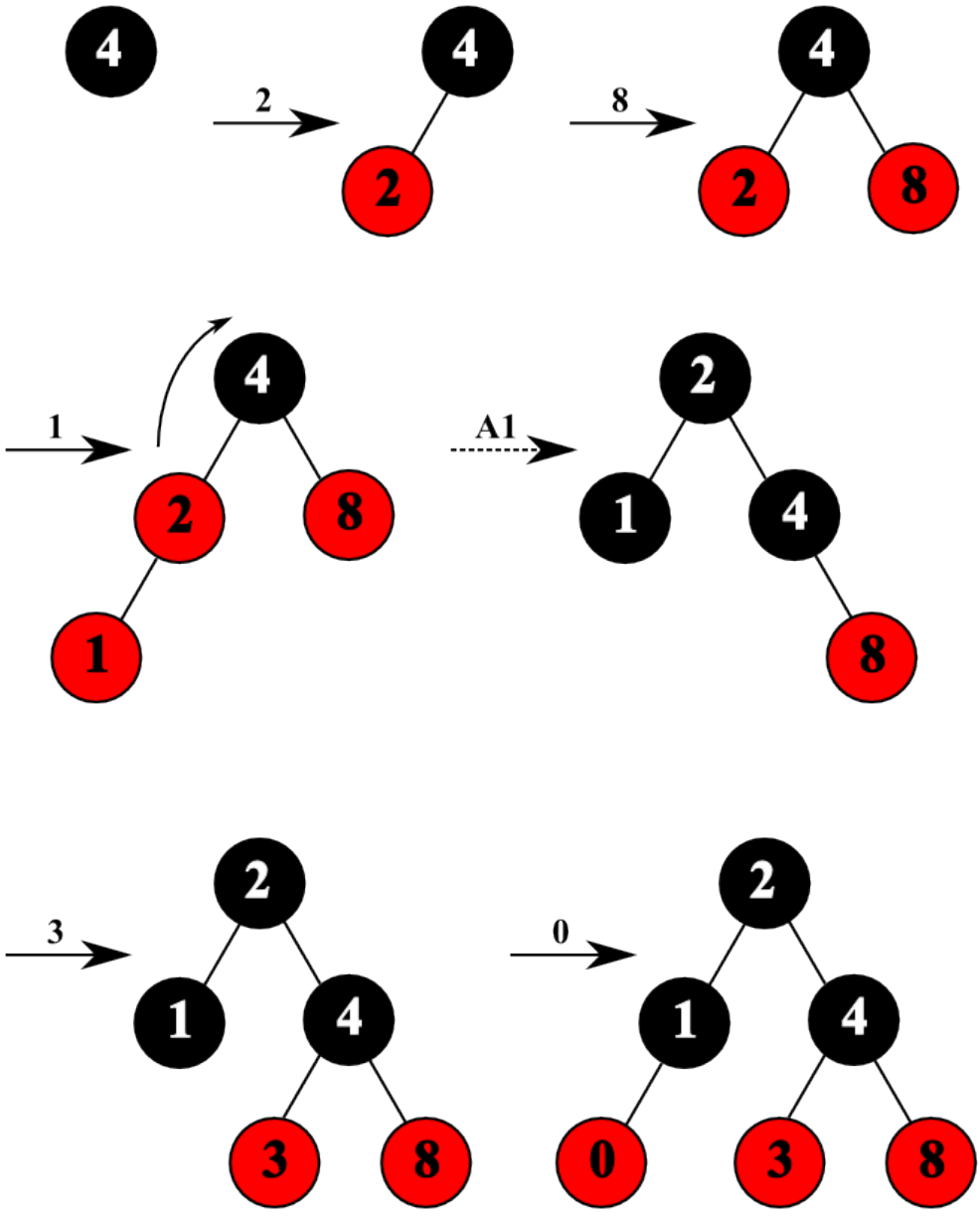

Kom ihåg att man sätter in nya noder med följande algoritm:

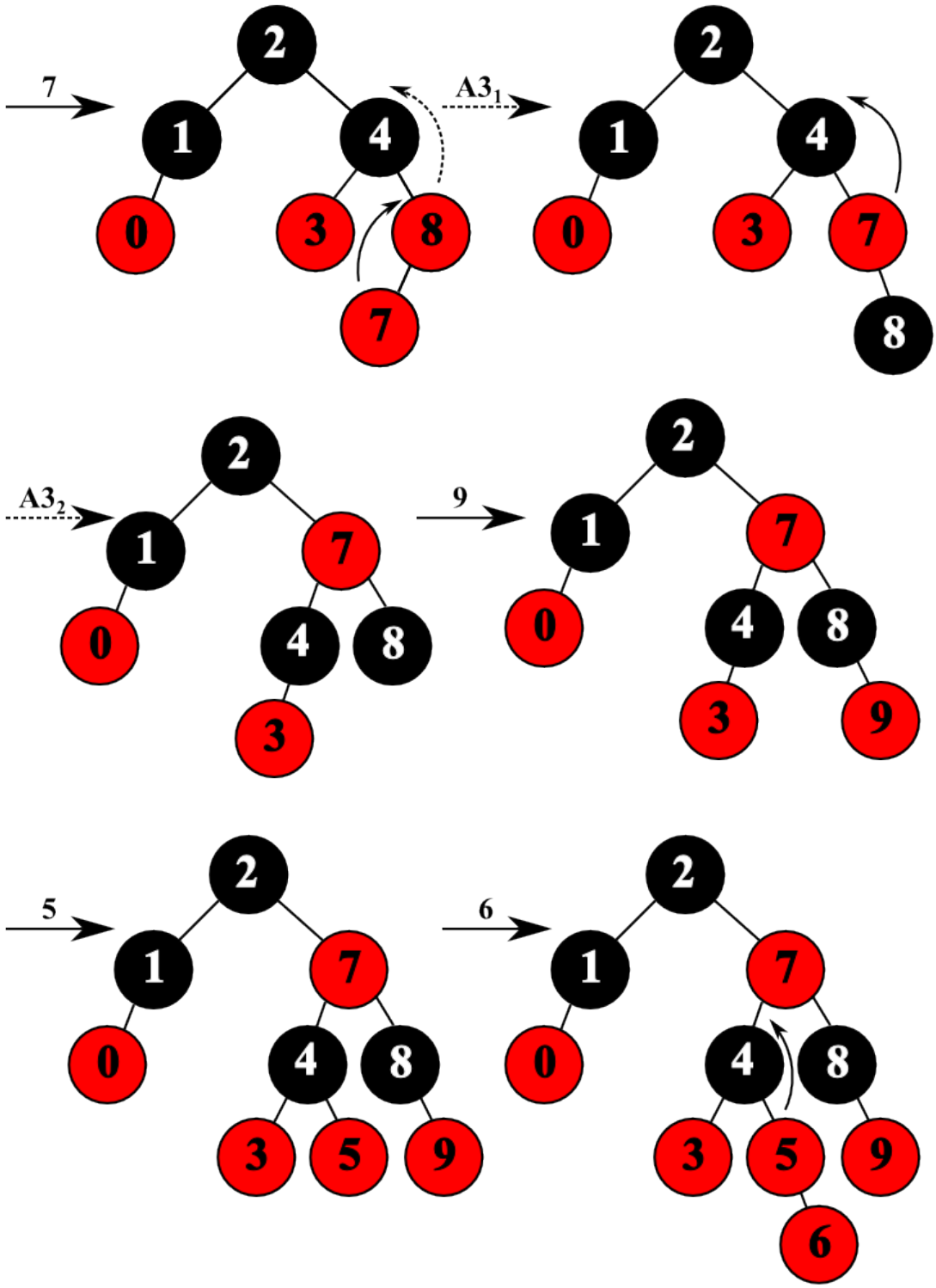
$i)$ Sätt in den nya noden på rätt plats

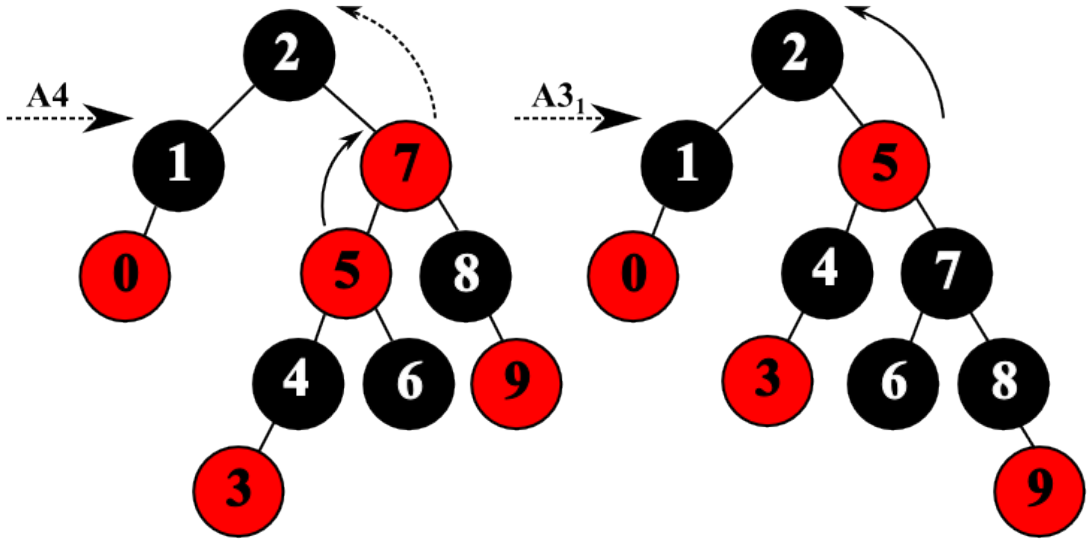
$ii)$ Färga den nya noden röd

$iii)$ Om någon röd nod har en röd förälder, rotera enligt ovan tills problemet är löst

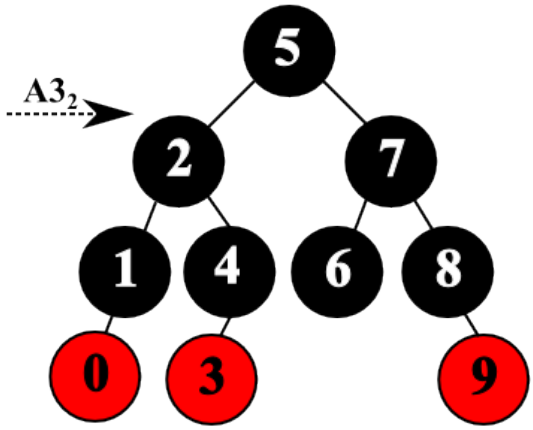
$iv)$ Färga rotnoden svart


4
2
4
2
8
4
2
8
1
4
2
8
1
A1
2
1
4
8
3
2
1
4
3
8
0
2
1
4
0
3
8


7
A3$_1$
A3$_2$
9
5
6

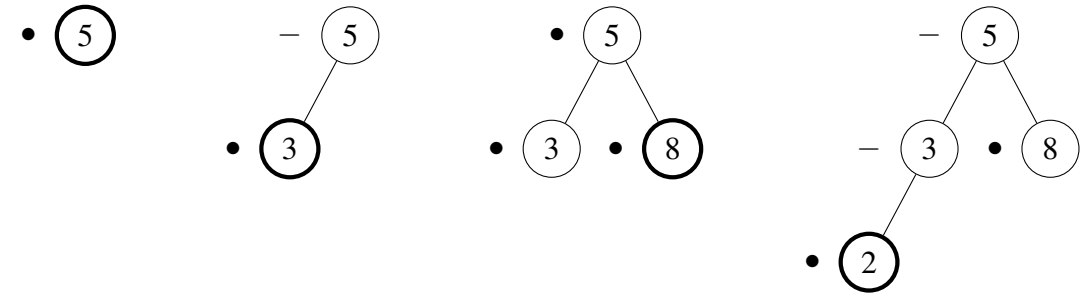

A4
2
1
7
0
5
8
4
6
9
3

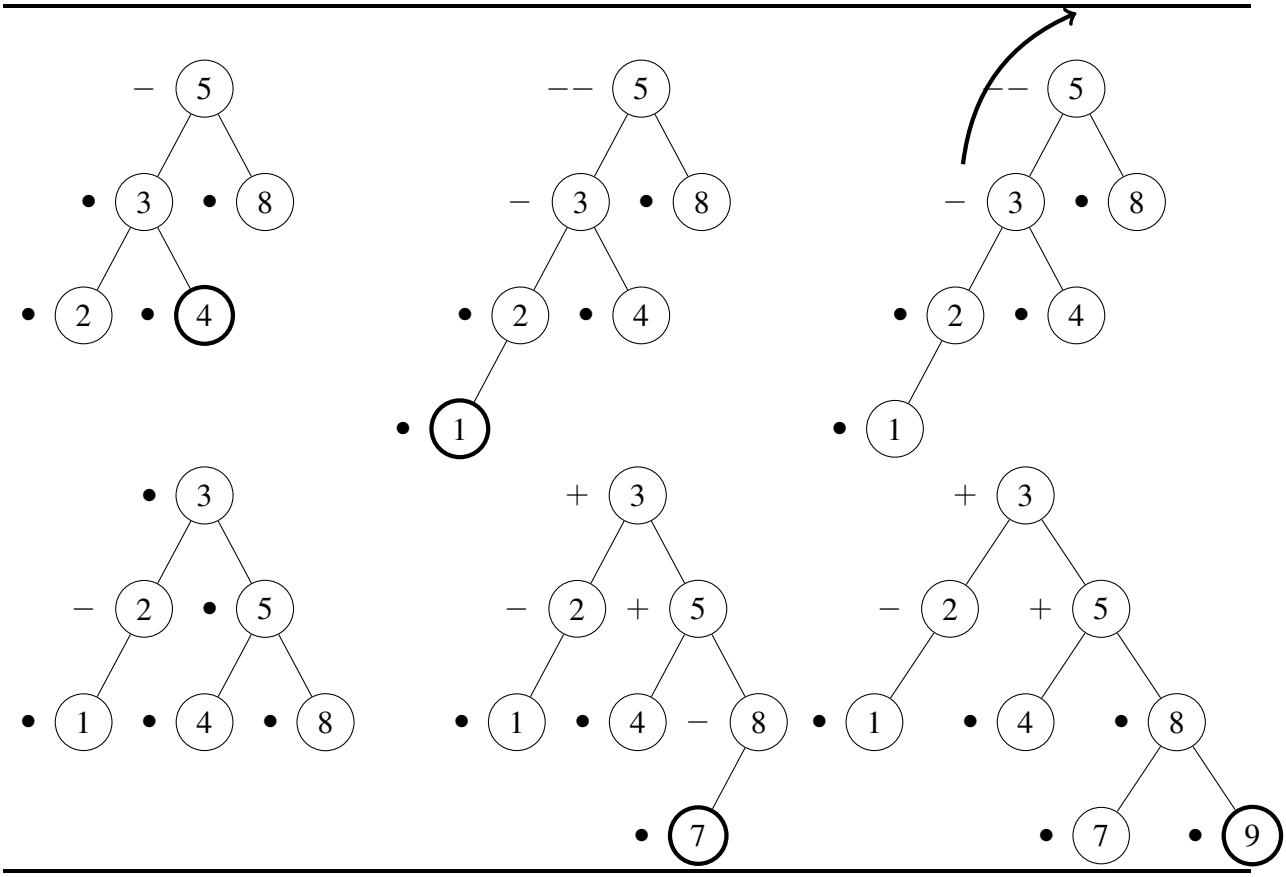

A3₁
2
1
5
0
4
7
3
6
8
9

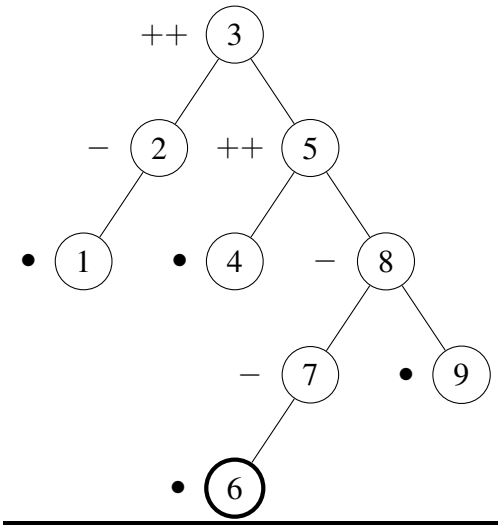

A3₂
5
2
7
1
4
6
8
0
3
9

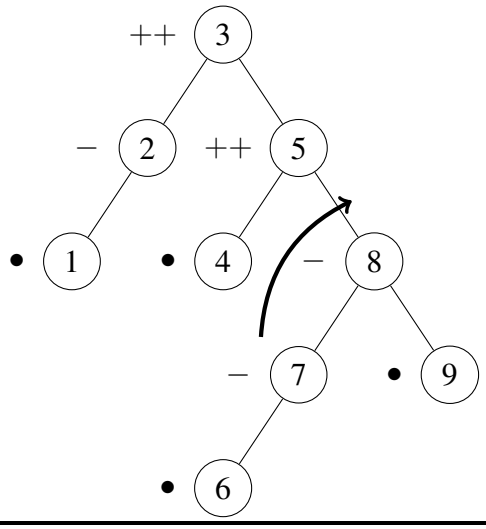
**2.**

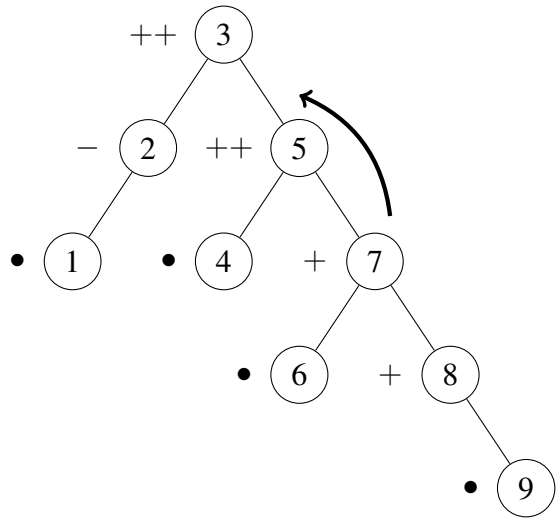
AVL-träd har ingen färg men varje nod har en balansfaktor som är höjden av det högra delträdet minus höjden av det vänstra. Om en nod blir obalanserad (**++** eller **- -**) så roterar man för att balansera trädet. Även här finns det fyra fall som kan dyka upp. Om en nod **X** är vänsterobalanserad (**- -**) och dess vänstra barn är vänstertungt (**-**) så roterar man mot **X** (som i A1). Om en nod **X** är vänsterobalanserad (**- -**) och dess vänstra barn **Y** är högertungt (**-**) så roterar man först mot **Y** och sen mot **X** (som i A2). De andra två fallen är bara speglingar av de två första.

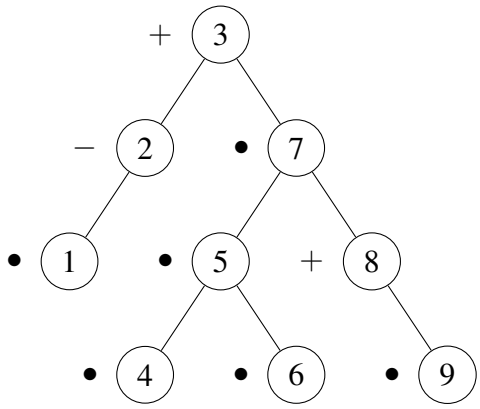



++ 3
− 2 ++ 5
• 1 • 4 − 8
− 7 • 9
• 6


++ 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.**

```
fun swedish(0,b) = 0
  | swedish(a,b) = b + swedish(a-1,b);
```

$$T(a) = \begin{cases} k_0 & \text{om } n = 0 \\ T(a-1) + k_1 & \text{om } n > 0 \end{cases}$$

$T(a) = \Theta(a)$ *(enligt Theorem 1)*

```
fun russian(1,b) = b
  | russian(a,b) =
        if a mod 2 = 1 then
            b + russian(a div 2, b*2)
        else
            russian(a div 2, b*2);
```

$$T(a) = \begin{cases} k_0 & \text{om } n = 0 \\ T(\frac{a}{2}) + k_1 & \text{om } n > 0 \end{cases}$$

$T(a) = \Theta(\log a)$ *(enligt Fall 2 av Master theorem)*

```
fun russianTail(1, b, ack) = b + ack
  | russianTail(a, b, ack) =
        russianTail(a div 2, b*2, ack+b*(a mod 2))
```

Tidskomplexiteten är densamma (med samma rekursion som ovan), däremot har funktionen nu konstant minneskomplexitet eftersom alla värden beräknas direkt.

**4.**
Antagandet att träden är balanserade gör att vi alltid halverar trädet när vi rekurserar.

```
datatype 'a tree = Leaf | Node of int * 'a * 'a tree * 'a tree

fun dataFromKey(_, Leaf) = NONE
  | dataFromKey(k, Node(k', d, L, R)) =
        if k>k' then
            dataFromKey(k, R)
        else if k<k' then
           dataFromKey(k, L)
        else
            SOME d;
```

$$T(n) = \begin{cases} k_0 & \text{om } n = 0 \\ T(\frac{n}{2}) + k_1 & \text{om } n > 0 \end{cases}$$

$T(n) = \Theta(\log n)$ *(enligt fall 2 av Master theorem)*

där $n$ är antalet noder i trädet.

```
fun keyFromData(d, Leaf) = NONE
  | keyFromData(d, Node(k, d', L, R)) =
        if d = d' then
            SOME k
        else
            let
                val k' = keyFromData(d, L)
            in
                if isSome k' then
                    k'
                else
                    keyFromData(d, R)
            end;
```

I värsta fall ligger noden vi söker längst ned till höger i trädet. Då kommer vi alltid rekursera två gånger:

$$T(n) = \begin{cases} k_0 & \text{om } n = 0 \\ 2T(\frac{n}{2}) + k_1 & \text{om } n > 0 \end{cases}$$

$T(n) = \Theta(n)$ *(enligt fall 1 av Master theorem)*

där $n$ är antalet noder i trädet.