Lösningsförslag till tentamen *P r e l i m i n ä r*

Kursnamn	**Algoritmer och datastrukturer**
Tentamensdatum	**2010-05-29**
Program	**DAI2+I2**
Läsår	**2009/2010, lp IV**
Examinator	**Uno Holmer**

Uppgift 1 (10 p)

Ingen lösning ges. Se kurslitteraturen.

Uppgift 2 (4+8 p)

a)

```
public TreeNode(TreeNode left,TreeNode right,int element) {
    this.element = element;
    this.left = left;
    this.right = right;
    size = 1;
    if ( left != null )
        size += left.size;
    if ( right != null )
        size += right.size;
}
```

b)

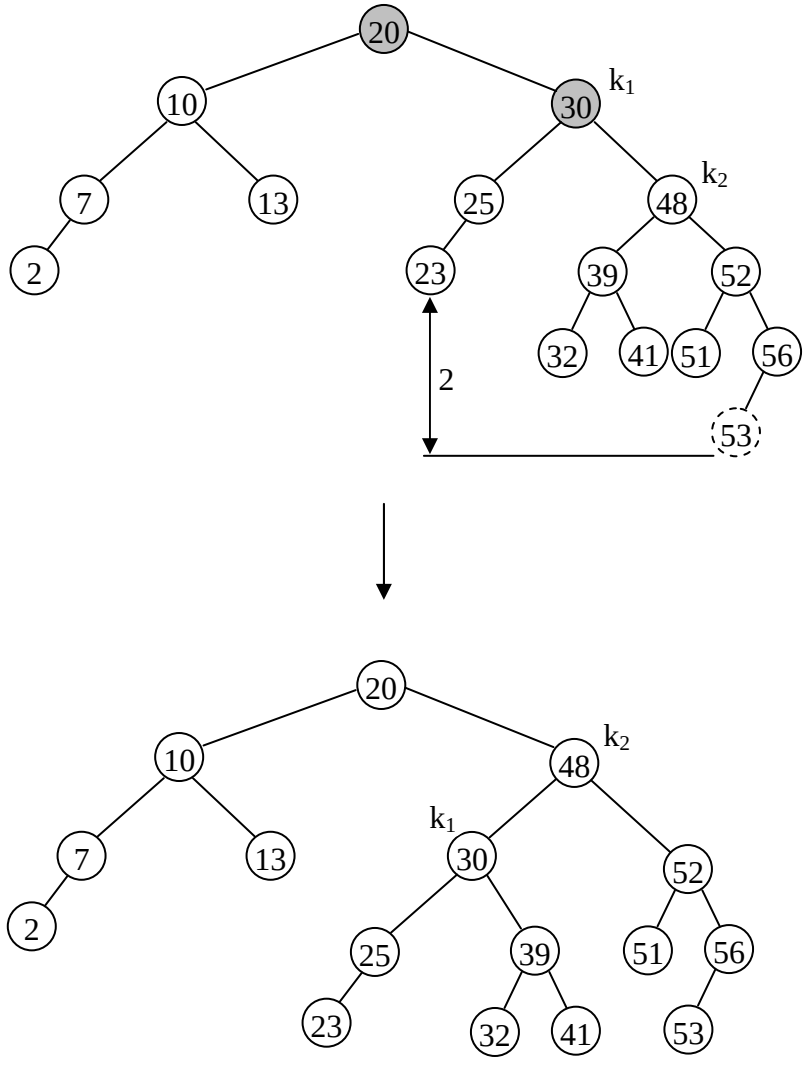
```
public static int findNth(int n,TreeNode t) {
    if ( t == null )
        throw new IllegalArgumentException();

    int leftSize = t.left != null ? t.left.size : 0;
    if ( n <= leftSize )
        return findNth(n,t.left);
    else if ( n == leftSize + 1)
        return t.element;
    else
        return findNth(n - t.left.size - 1,t.right);
}
```

Uppgift 3 (6+4 p)

a)
Preorder: ABDEGCFHI
Postorder: DGEBHIFCA
Inorder: DBGEAHFIC

b)
När 53 sätts in bryts AVL-villkoret. Den djupaste obalanserad noden är k_1. En enkelrotation enligt nedan upprättar balansen i trädet. Detta motsvarar fall 4 i Weiss:s figur 19.26.



Uppgift 4 (3+2 p)

a)
Så här ser tabellen ut efter anropssekvensen i uppgiften

0	10
1	
2	
3	43
4	
5	
6	
7	
8	
9	
10	21

hash(21) = 10
hash(10) = 10
hash(43) = 10

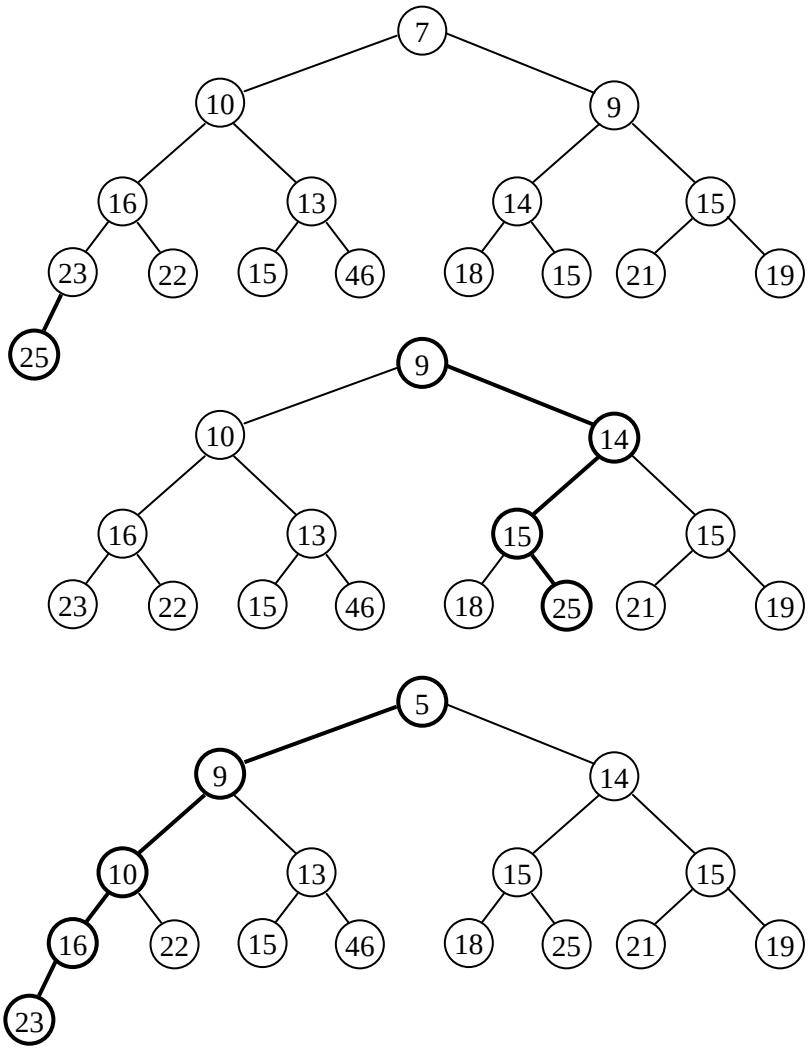
`add(21);`	Elementet sätts in på hashpositionen.
`add(10);`	10 kolliderar med 21 och hamnar i alternativ position $(10 + 1^2) \bmod 11 = 0$.
`add( 43);`	43 kolliderar först med 21 och sen med 10 och hamnar i position $(10 + 2^2) \bmod 11 = 3$.
`remove(10);`	Cell nr 0 markeras som struken
`add(43);`	`43` sätts ej in igen eftersom duplikat ej tillåts i en hashtabell. Observera att 43 ej sätts in i position 0 där 10 ströks tidigare. Alla insättningar måste i princip föregås av en misslyckad sökning. Därför är strukna celler ”frysta” tills omhashning sker.

b)
Belastningsfaktorn får ej överskida 0.5 vid kvadratisk sondering. Med 11 platser i tabellen görs omhashning då det sjätte elementet (inklusive strukna) sätts in. Tabellen måste alltid ha primtalsstorlek och det minsta primtalet större eller lika med 2*11 är 23.

Uppgift 5 (1+3+2 p)

a)
Strukturvillkor: Trädets skall vara komplett (så att det kan lagras i ett fält), samt ordningsvillkor: Inget barn är större än sin förälder.

b)



c)
Worst case för `insert` i en binär hög är O(log n), vilket ger O(n log n) för insättning av n element. En bättre metod är att först kopiera elementen till fältet i O(n) tid, och därefter använda metoden `buildHeap`. Vi visade på föreläsning att WC för denna metod är O(n). Alltså kan hela operationen göras i O(n) tid.

Uppgift 6 (3+1+1+1+1 p)

a)
g1 kan uteslutas eftersom det finns en båge från A till B och då kan inte en topologisk ordning inledas med BA. Av analogt skäl kan vi utesluta g3 som har en båge från B till A och då kan inte en topologisk ordning inledas med AB. I g4 finns en båge från D till C, vilket utesluter ABCD. Återstår g2 för vilken ABCD och BADC är möjliga ordningar, tillkommer ADBC, ABDC och BACD.

b) Först A, sen BCD i valfri ordning, sist E.

c) ADCBE

d) A(0), B(14), C(12), D(7), E(22).

e) I Dijkstras algoritm används en prioritetskö.

Uppgift 7 (10 p)

Iterativ algoritm med FIFO-kö. Glöm inte att testa om trädet är tomt.

```
public static <T> List<T> getLevelOrder(TreeNode<T> t) {
    List<T> result = new ArrayList<T>();
    if ( t != null ) {
        Queue<TreeNode<T>> q = new LinkedList<TreeNode<T>>();
        q.add(t);
        while ( ! q.isEmpty() ) {
            TreeNode<T> node = q.remove();
            result.add(node.element);
            if ( node.left != null )
                q.add(node.left);
            if ( node.right != null )
                q.add(node.right);
        }
    }
    return result;
}
```