Lösningsförslag till tentamen

Kursnamn	**Algoritmer och datastrukturer**
Tentamensdatum	**2014-01-14**
Program	**DAI2+I2**
Läsår	**2012/2013, lp 4**
Examinator	**Uno Holmer**

Uppgift 1 (10 p)

Ingen lösning ges. Se kurslitteraturen.

Uppgift 2 (5+5 p)

a)

```
public static boolean matches(TreeNode t1,TreeNode t2) {
    if ( t1 == null )
        return true;
    else if ( t2 == null )
        return false;
    else
        return
            t1.element == t2.element &&
            matches(t1.left,t2.left) &&
            matches(t1.right,t2.right);
}
```

eller enklare

```
public static boolean matches(TreeNode t1,TreeNode t2) {
    return
        t1 == null || t2 != null &&
        t1.element == t2.element &&
        matches(t1.left,t2.left) &&
        matches(t1.right,t2.right);
}
```

b)

```
public static boolean contains(TreeNode t1,TreeNode t2) {
    return
        t2 == null ||
        t1 != null &&
        (matches(t2,t1) ||
         contains(t1.left,t2) ||
         contains(t1.right,t2));
}
```

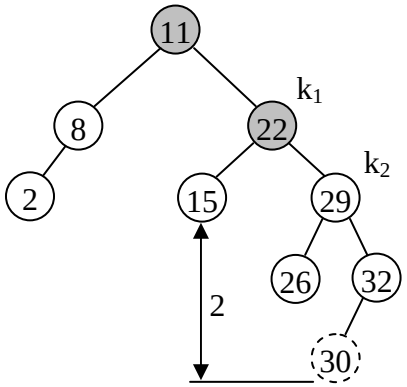
Uppgift 3 (3+3 p)

a) (3 p)

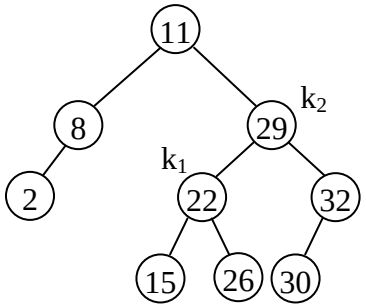
$x_1 < 21$, $21 < x_2 < 33$, $40 < x_3 < 48$, $53 < x_4$.

b) (3 p)

När 30 sätts in bryts AVL-villkoret. Den djupaste obalanserad noden är k_1. Detta motsvarar fall 4 i Weiss:s figur 19.26.



En enkelrotation upprättar balansen i trädet.



Uppgift 4 (3+3+2 p)

a) (3 p)

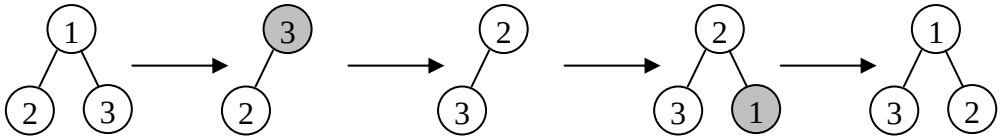
Nej, i en minhög med minst två element måste minst en nod ha ett vänsterbarn. Eftersom alla barn är minst lika stora som sina föräldrar kan trädet inte vara ett binärt sökträd. I ett sökträd är ju alla vänsterbarn strikt mindre än sina föräldrar.

b) (3 p)

I värsta fall är `insert` *O*(log N) så N insättningar är *O*(N log N), men `buildHeap` är bara *O*(N).

c) (2 p)

Elementen kan hamna på andra ställen, vilket framgår följande exempel. De tre första träden visar `deleteMin` och de två sista `insert`.



Uppgift 5 (5 p)

Så här ser tabellen ut efter anropssekvensen i uppgiften

0	
1	
2	Malin
3	Lisa
4	
5	
6	Kalle
7	
8	
9	
10	
11	Sven
12	

Malin och Kalle har nu båda medlemsnummer 15, vilket tabellen borde upptäcka vid försöket att sätta in Malin. Om man söker Kalle så hittas Malin. Vi hashar Kalles medlemsnummer 15 och får position 2, och där ligger ju ett element med nyckeln 15. Bingo!? Problemet är att vi tillåtit flera element med samma söknyckel i samma tabell. Alla element i en hashtabell måste vara unika (i vårt exempel: ha unika medlemsnummer). För att garantera det måste varje insättning föregås av en misslyckad sökning. Då kan man inte "återanvända" celler där element strukits, utan dessa måste markeras som strukna, men upptagna (lazy deletion). Om detta skett på rätt sätt skulle krocken med Kalle upptäckts och insättningen av Malin misslyckats. 1-0 till pessimisterna denna gång alltså!

Uppgift 6 (4+2+2+1 p)

a)
g1: ABC, ACB, BAC, BCA, CAB, CBA.
g2: Inga, grafen är ej riktad.
g3: inga: grafen har en cykel och är alltså ingen DAG.
g4: ABCDE, ABDCE, ABDEC, ADBCE, ADBEC, DABCE, DABEC.

b) D;A;E;F;G;B;C. Alt. D;{AEFG};{BC}, där {xyz} betecknar en godtycklig permutation av xyz. Alltså: Först D, därefter A,E,F,G i valfri ordning, och sist B och C i valfri ordning.

c) D;E;G;A;F;B;C

d)
I en graf som innehåller en negativ kostnadscykel riskerar Dijsktras algoritm att inte terminera eftersom avståndet i en sådan cykel inte har något minimum.

Uppgift 7 (4+8 p)

Detta är laboration 6 i miniformat. Stickorna bildar ett uppspännande träd som förbinder alla punkter.

a)
```
private static List<Stick> getRandomSticks(int size) {
    List<Stick> sticks = new ArrayList<Stick>(size*size);
    for ( int i = 0; i < size; i++ )
        for ( int j = 0; j < size; j++ ) {
            if ( j < size - 1 )
                // Horizontal stick
                sticks.add(new Stick(i*size + j,i*size + j+1));
            if ( i < size - 1 )
                // Vertical stick
                sticks.add(new Stick(i*size + j,(i+1)*size + j));
        }
    Collections.shuffle(sticks);
    return sticks;
}
```

b)
```
public static List<Stick> solve(int size) {
    DisjointSets ds = new DisjointSets(size*size);
    List<Stick> sticks = getRandomSticks(size);
    Iterator<Stick> it = sticks.iterator();
    while ( it.hasNext() ) {
        Stick stick = it.next();
        int set1 = ds.find(stick.oneEnd);
        int set2 = ds.find(stick.theOtherEnd);
        if ( set1 != set2 )
            ds.union(set1,set2);
        else
            it.remove();
    }
    return sticks;
}
```