Lösningsförslag till tentamen

Kursnamn	**Algoritmer och datastrukturer**
Tentamensdatum	**2013-05-31**
Program	**DAI2+I2**
Läsår	**2012/2013, lp 4**
Examinator	**Uno Holmer**

Uppgift 1 (10 p)

Ingen lösning ges. Se kurslitteraturen.

Uppgift 2 (10 p)

```
public static Rect boundingRect(TreeNode t,int upper,int left) {
    Rect r = new Rect();
    r.upper = upper;
    r.left = left;
    if ( t == null ) {
        r.lower = upper;
        r.right = left;
    } else if ( t.isLeaf() ) {
        r.lower = upper + D;
        r.right = left + D;
    } else {
        Rect2 brl = boundingRect(t.left,upper + 2*D,left);
        Rect2 brr = boundingRect(t.right,upper +
                                 2*D,brl.right+D);
        r.lower = Math.max(brl.lower,brr.lower);
        r.right = brr.right;
    }
    return r;
}
```

Uppgift 3 (2+4 p)

a) Insättning av 5, 7, 15 eller 18 ger höjdskillnaden två i trädet.

b) Ett minimalt AVL-träd av höjd -1 har 0 noder, ett träd av höjd 0 har 1 nod (höjd enl. Weiss:s definition).

$min_{AVL}(-1) = 0$
$min_{AVL}(0) = 1$
$min_{AVL}(h) = min_{AVL}(h-1) + min_{AVL}(h-2) + 1 \qquad (h > 0)$

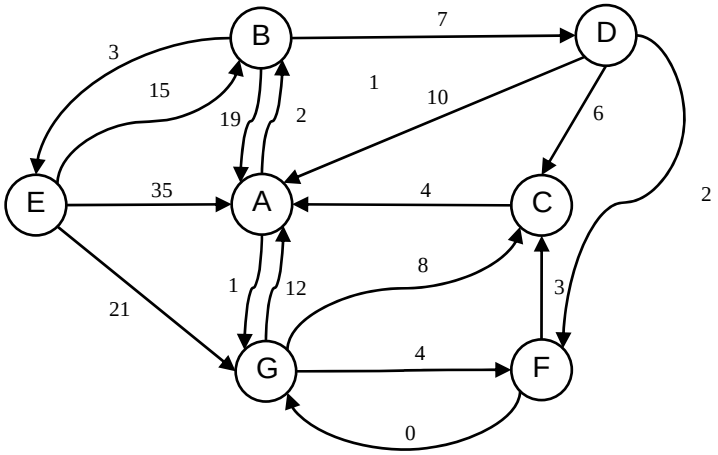
Uppgift 4 (2+4 p)

a) AC för insert är *O*(1) och för deleteMin *O*(log N)

b) En binär (min)hög är ett komplett träd där inget barn är mindre än sin förälder. I ett binärt sökträd är alla elementen unika. Dessutom gäller, rekursivt, att alla element i vänster delträd är mindre än roten och alla element i höger delträd är större än roten. En binär hög med minst två element måste ha ett icketomt delträd till vänster om roten, annars vore inte högen ett komplett träd. Men roten i det delträdet får ju inte vara strikt mindre än roten till hela trädet. Alltså kan högen inte vara ett sökträd. Omvänt: I ett sökträd med minst två noder som är komplett (vilket det givetvis får vara) har ett icketomt delträd till vänster om roten. Roten i det delträdet måste vara strikt mindre än hela trädets rot. Då kan trädet omöjligen vara en binär hög.

Uppgift 5 (4+6 p)

a)



b) Kortaste viktade avstånden är E-A:31, E-B:15, E-C:27, E-D:22, E-F:24, E-G:21.
Dijkstras algoritm besöker noderna i ordningen: EBGDFCA.

Uppgift 6 (8 p)

Bör vara Alingsås, men om Ulricehamn lagras i Borås gamla plats fås Ulricehamn istället. Om lazy deletion tillämpas misslyckas insättningen av Ulricehamn eftersom Alingsås och Ulricehamn har samma söknyckel. Ulricehamn betraktas då som ett duplikat eftersom den fått samma söknyckel som Alingsås. Om insättningen skulle göras i Borås gamla position får vi en situation där två olika städer felaktigt lagras med samma geografiska position i tabellen. Dessutom skulle Alingsås ej längre kunna hittas eftersom den maskeras av Ulricehamn.

0	
1	
2	
3	
4	gbg
5	borås
6	kungsbacka
7	
8	
9	alingsås
10	

Uppgift 7 (10 p)

Här används en FIFO-kö, men det går lika bra med vilken behållare som helst.

```
public void topologicalSort() {
    if ( hasCycle() )
        throw new GraphException("Graph has a cycle");

    // Compute indegrees
    for ( Vertex v : vertexMap.values() )
        for ( Vertex w : v.adj )
            w.indegree++;

    // Enqueue all indegree zero nodes
    Queue<Vertex> q = new LinkedList<>();
    for ( Vertex v : vertexMap.values() )
        if ( v.indegree == 0 )
            q.add(v);

    // Sort topologically
    while ( ! q.isEmpty() ) {
        Vertex v = q.poll();
        System.out.print(v.name + " ");
        for ( Vertex w : v.adj )
            if ( --w.indegree == 0 )
                q.add(w);
    }
}
```