Lösningsförslag till tentamen

Kursnamn	**Algoritmer och datastrukturer**
Tentamensdatum	**2017-10-06**
Program	**DAI2+I2**
Läsår	**2016/2017, lp 4**
Examinator	**Uno Holmer**

Uppgift 1 (10 p)

Ingen lösning ges. Se kurslitteraturen.

Uppgift 2 (10 p)

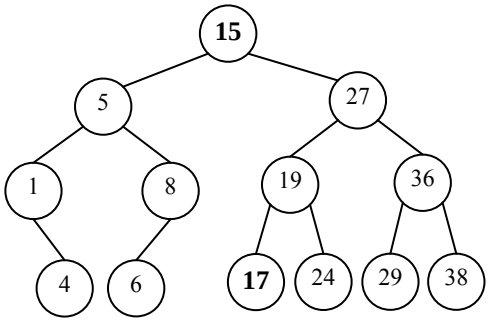
Utnyttja ett par användbara listfunktioner från rekursionslabben:

```
public static ListNode getLevel(TreeNode t,int i) {
    if ( t == null )
        return null;
    else if ( i == 0 )
       return cons(t.element,null);
    else
       return append(getLevel(t.left,i-1),
                     getLevel(t.right,i-1));
}
```

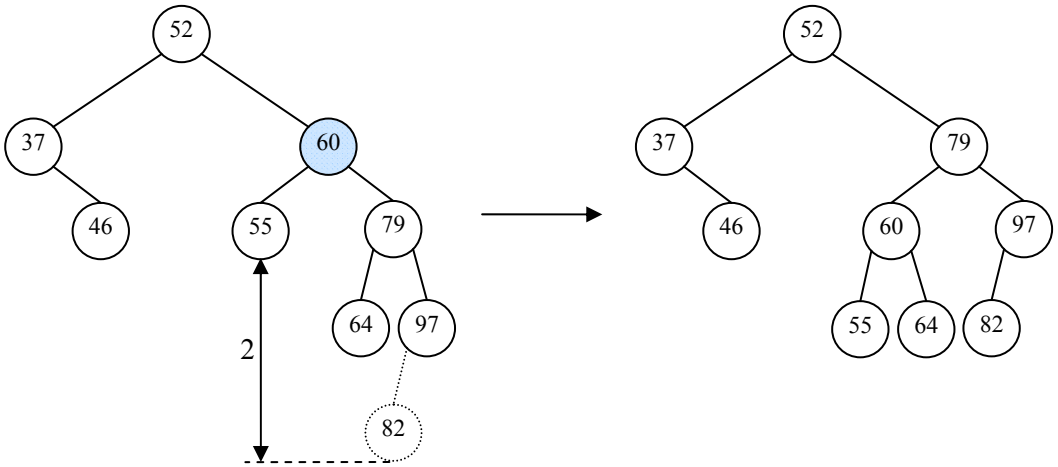
Uppgift 3

a) (2 p)

Ersätt roten med det minsta elementet i det högra delträdet (eller med det största elementet i det vänstra delträdet).



b) (3 p)
Detta motsvarar det fjärde AVL-rotationsfallet i Weiss.



Uppgift 4 (6 p)

a) (2 p)
Nej, belastningsfaktorn λ får ej överstiga 0.5, vilket skulle bli fallet om två element till sattes in i den befintliga tabellen.

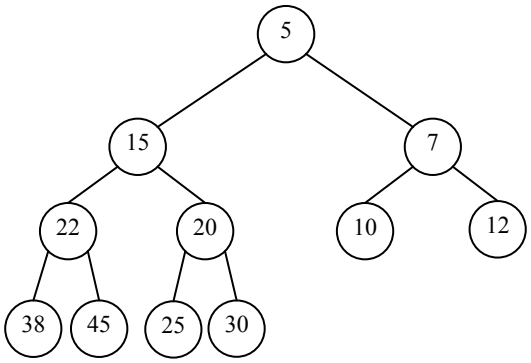
b) (4 p)
Tabellens storlek skall vara ett primtal. Minsta primtalet som är minst dubbelt så stort som 7 är 17. Den nya tabellen får utseendet

0		
1	18	$+2^2$
2		
3		
4		
5		
6	**48**	$+3^2$
7		
8		
9		
10		
11		
12		
13		
14	31	$+0^2$
15	49	$+1^2$
16		

Uppgift 5

a) (3 p)

Nej, trädet är inte komplett eftersom det finns luckor i den nedersta nivån. Placera 15 till vänster under 25 och utför operationen `percolateDown` på noden som innehåller 25 och därefter på noden som innehåller 20. Detta resulterar i högen:

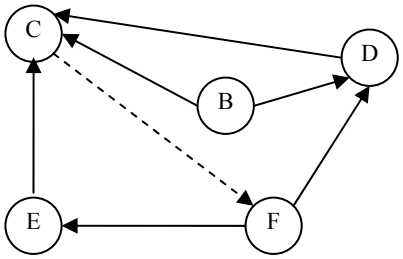


b) (2 p)

	5	15	7	22	20	10	12	38	45	25	30
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*

Uppgift 6 (7 p)

Bågen mellan noderna C och F ingår i två olika cykler: CFEC och CFDC. Tas bågen bort får vi en DAG.



De topologiska ordningarna är BFDEC, BFEDC, FBDEC, FBEDC, FEBDC.

Uppgift 7 (17 p)

a) (7 p)

```
public Graph computeSubGraph(String startNode,double maxDist) {
    clearAll();
    dijkstra(startNode);
    Graph g = new Graph();
    g.getVertex(startNode); // this adds the start node (!)
    for ( Vertex v : vertexMap.values())
        if ( v.dist <= maxDist ) // this test adds a little efficiency
            for ( Edge e : v.adj )
                if ( e.dest.dist <= maxDist )
                    g.addEdge(v.name,e.dest.name,e.cost);
    return g;
}
```

b) (10 p)

```
public List<String> nodesAtUWD(String startNode,int d) {
    // FIFO queue of <vertex,distance> pairs
    LinkedList<Pair<Vertex,Integer>> q =
        new LinkedList<Pair<Vertex,Integer>>();
    clearAll(); // the scratch field is used here
    q.addLast(new Pair<Vertex,Integer>(vertexMap.get(startNode),0));
    while ( !q.isEmpty() && q.peek().second < d ) {
        Pair<Vertex,Integer> p = q.poll();
        for ( Edge e : p.first.adj )
            if ( e.dest.scratch == 0 ) {
                e.dest.scratch = 1;
                q.addLast(
                     new Pair<Vertex,Integer>(e.dest,p.second+1));
            }
    }
    // Now the vertexes that are still left in the queue
    // are all at distance d from the startnode.
    // Extract their names into a list.
    List<String> result = new LinkedList<String>();
    while ( ! q.isEmpty() )
        result.add(q.poll().first.name);
    return result;
}
```