Lösningsförslag till tentamen

Kursnamn	**Algoritmer och datastrukturer**
Tentamensdatum	**2010-04-08**
Program	**DAI2+I2**
Läsår	**2008/2009, lp 4**
Examinator	**Uno Holmer**

Uppgift 1 (10 p)

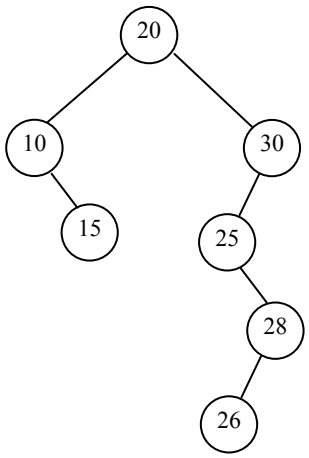
Ingen lösning ges. Se kurslitteraturen.

Uppgift 2 (8 p)

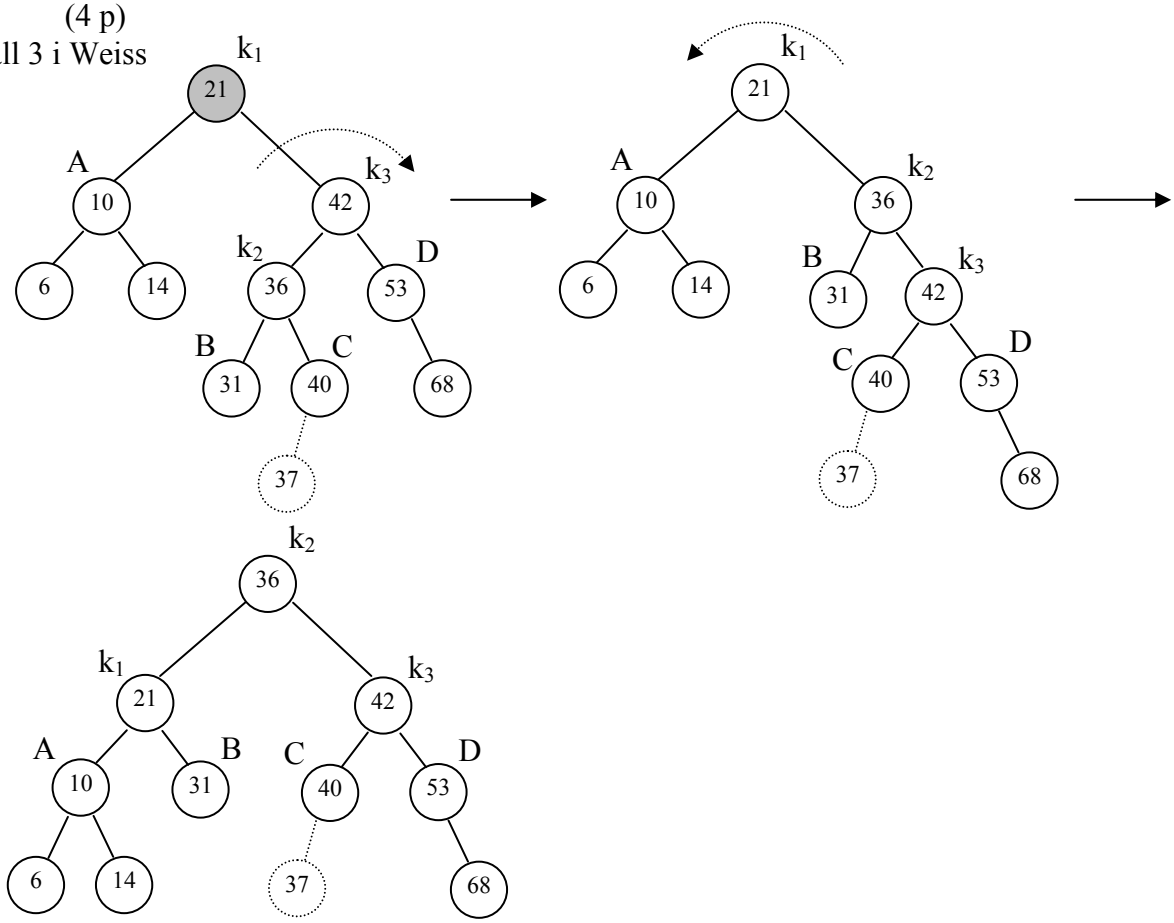
```
public static <E extends Comparable<? super E>>
TreeNode<E> insert( E x, TreeNode<E> t ) {
    if ( t == null )
        return new TreeNode<E>( x );
    else if ( x.compareTo(t.element) < 0 ) {
        t.left = insert( x, t.left );
        return t;
    } else if ( x.compareTo(t.element) > 0 ) {
        t.right = insert( x, t.right );
        return t;
    } else
        throw new IllegalArgumentException();
}
```

Uppgift 3

a) (2 p)



b) (4 p)
Fall 3 i Weiss



Uppgift 4

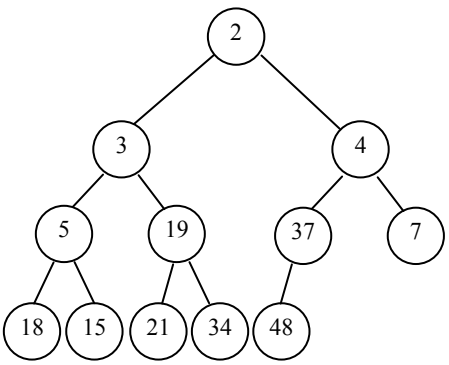
a) (3 p)
Ett primärt kluster är en ansamling av närliggande element med samma hashvärde. Primära kluster uppkommer då linjär sondering används som kollisionshanteringsmetod. Primär klusterbildning undviks genom att använda kvadratisk sondering.

b) (2 p)
Belastningsfaktorn lambda får ej överstiga 0.5 och tabellens storlek skall vara ett primtal.

Uppgift 5

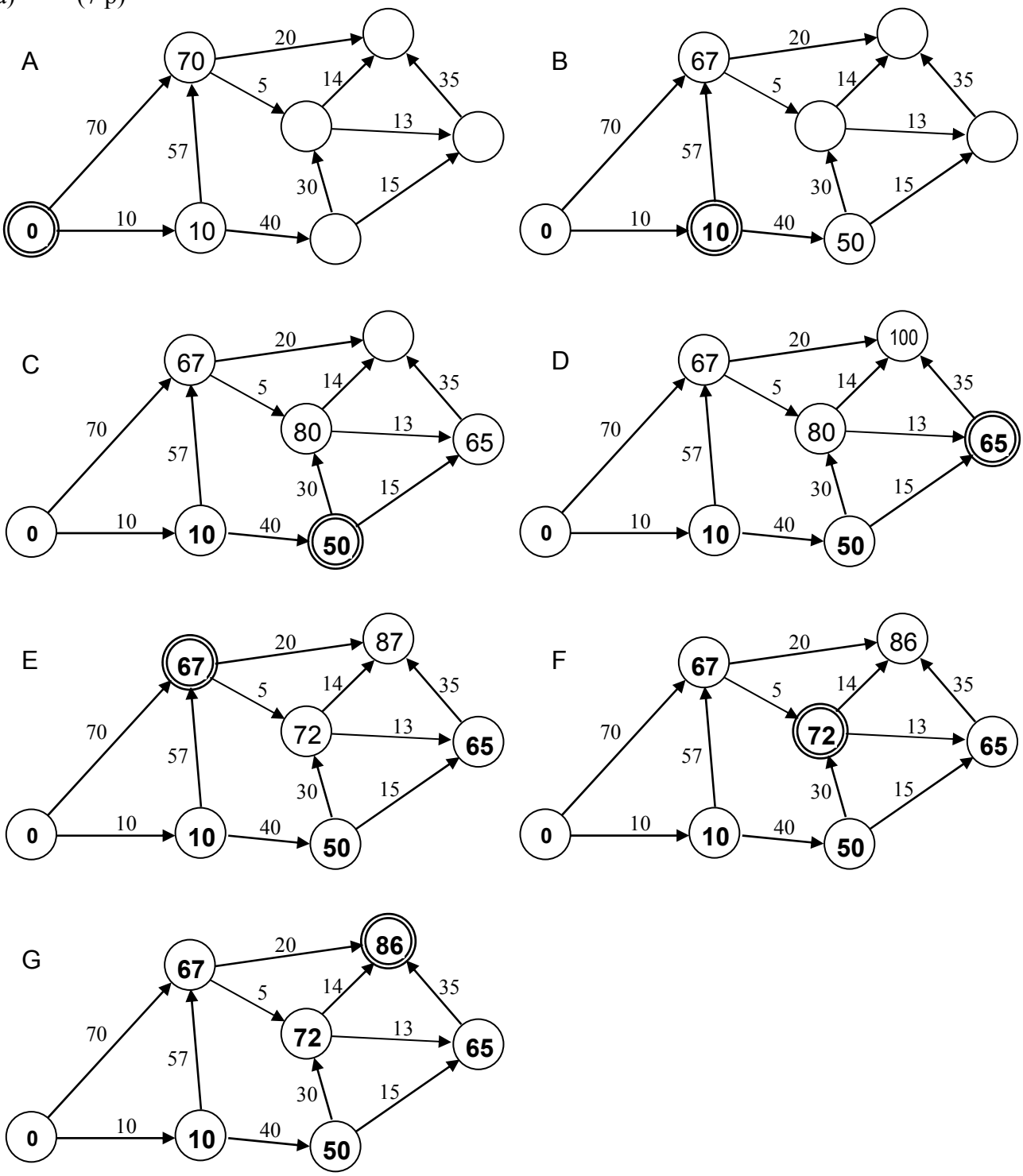
a) (2 p)
Nej, trädet är inte komplett eftersom det finns en lucka i det högra delträdet under 13. Dessutom är inte ordningsvillkoret uppfyllt mellan elementen 15 och 16.

b) (4 p)



Uppgift 6

a) (7 p)



b) (6 p)

ABCDE, ABDCE, ACBDE, ACDBE, ADBCE, ADCBE, DABCE, DACBE, ACDEB, ADCEB, DACEB.

Uppgift 7 (12 p)

```
public class Graph {
      ...

    public boolean isDAG() {
        Set<String> visited = new HashSet<String>();
        for ( String node : graphTable.keySet() )
            if ( hasCycle( node, visited ) )
                return false;
        return true;
    }

    private boolean hasCycle( String node, Set<String> visited ) {
        if ( visited.contains( node ) )
            return true;
        else {
            visited.add( node );
            for ( String neighbour : graphTable.get( node ) )
                if ( hasCycle( neighbour, visited ) )
                    return true;

            visited.remove( node );
            return false;
        }
    }
}
```