Lösningsförslag till tentamen

Kursnamn	**Algoritmer och datastrukturer**
Tentamensdatum	**2017-06-02**
Program	**DAI2+I2**
Läsår	**2016/2017, lp 4**
Examinator	**Uno Holmer**

Uppgift 1 (10 p) Ingen lösning ges. Se kurslitteraturen.

Uppgift 2 (9 p)

a)
```
public static <E> LinkedList<E> getPreorder(TreeNode<E> t) {
    if ( t == null )
        return new LinkedList<E>();
    else {
        LinkedList<E> result = getPreorder(t.left);
        result.addFirst(t.element);
        result.addAll(getPreorder(t.right));
        return result;
    }
}
```

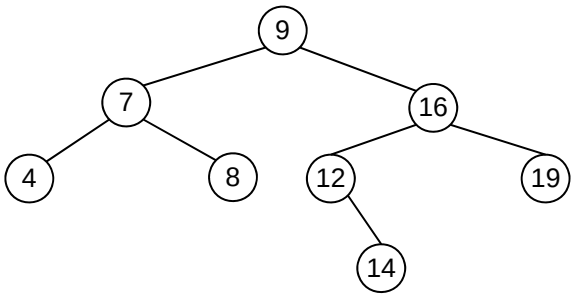
Uppgift 3 (1+2+3 p)

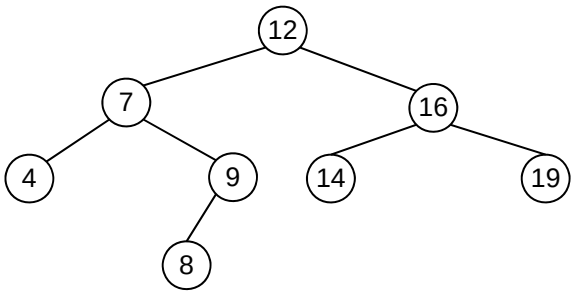
a) Ett fullt träd av höjd H har 2^{H+1} - 1 noder.

b) Vi kan skapa ett minimalt AVL-träd av höjd H genom att sätta samman ett minimalt AVL-träd av höjd H-2 och ett minimalt AVL-träd av höjd H-1 med en gemensam rot. Sambandet är alltså

antalNoder(0) = 1
antalNoder(1) = 2
antalNoder(H) = antalNoder (H-2) + antalNoder (H-1) + 1

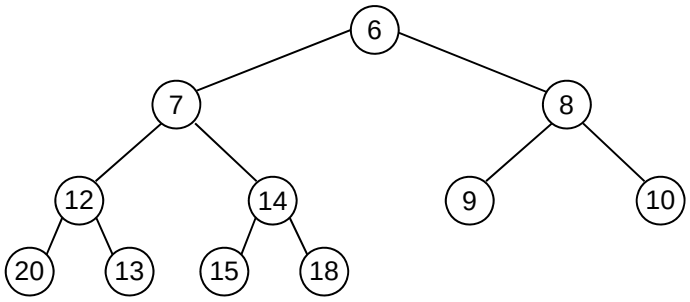
c) Det finns två möjliga resultat, till vänster placerades det största elementet i det vänstra delträdet i roten, i det högra det minsta elementet i det högra delträdet.



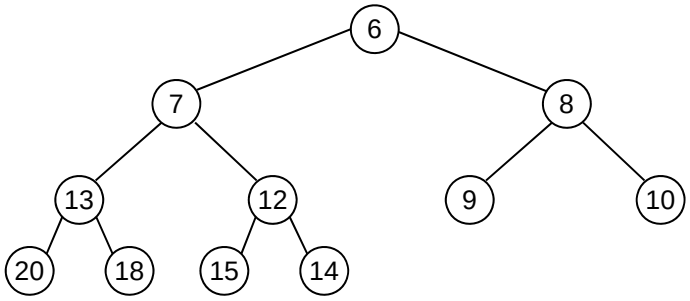


Uppgift 4 (3+3 p)

a) Efter sekvensen `insert(6);deleteMin();`



b) efter sekvensen `deleteMin();insert(6);`

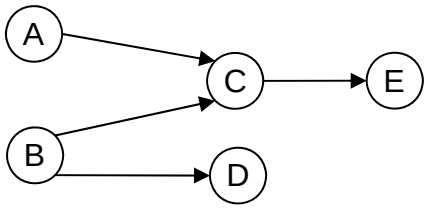


Uppgift 5 (2+1+4 p)

a) En prioritetskö krävs för att garantera minimalitet. Det preliminära avståndet till en nod beräknas alltid från nodens grannar som vi redan vet har fått sina minimala avstånd fastställda.

b) Grafen skall vara en DAG, alltså en riktad graf utan cykler.

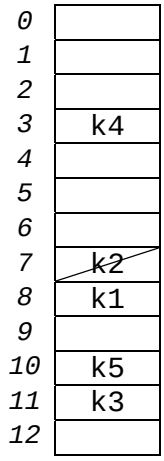
c) Följande DAG har de angivna topologiska ordningarna:



Uppgift 6 (4+2 p)

a)

Insättning av k1-k4 går bra, därefter stryks k2 (lazy deletion) och k5 sätts in. k6 sätts inte in eftersom objektet har samma kontonummer som k4. En insättning måste ju alltid föregås av en misslyckad sökning.



b)
find(k4) returnerar k4! Om inte lazy deletion tillämpats, skulle k6 satts in på plats 7. Det skulle då finnas två saldoobjekt för kontonumret 983619. Vid sökning efter k4 med saldot 198000 hittas istället k6 med saldot 0!

Uppgift 7 (2+6+8 p)

a)
```
public class MapAssignment implements Assignment {
    private Map<Character,Boolean> assignments = new HashMap<>();

    public void assign(char variable,boolean value) {
        assignments.put(variable,value);
    }
    public boolean getValue(char variable) {
        return assignments.get(variable);
    }
}
```

b)
```
public class Variable implements BooleanFormula {
    private char name;

    public Variable(char name) {
        this.name = name;
    }
    public boolean getValue(Assignment a) {
        return a.getValue(name);
    }
    public Set<Character> getVariables() {
        Set<Character> result = new HashSet<>();
        result.add(name);
        return result;
    }
}
```

```
public class Conjunction implements BooleanFormula {
    private BooleanFormula leftOperand,rightOperand;

    public Conjunction(BooleanFormula leftOperand,
                       BooleanFormula rightOperand)
    {
        this.leftOperand = leftOperand;
        this.rightOperand = rightOperand;
    }
    public boolean getValue(Assignment a) {
        return leftOperand.getValue(a) && rightOperand.getValue(a);
    }
    public Set<Character> getVariables() {
        Set<Character> result = leftOperand.getVariables();
        result.addAll(rightOperand.getVariables());
        return result;
    }
}

public class Disjunction implements BooleanFormula {
    private BooleanFormula leftOperand,rightOperand;

    public Disjunction(BooleanFormula leftOperand,
                       BooleanFormula rightOperand)
    {
        this.leftOperand = leftOperand;
        this.rightOperand = rightOperand;
    }
    public boolean getValue(Assignment a) {
        return leftOperand.getValue(a) || rightOperand.getValue(a);
    }
    public Set<Character> getVariables() {
        Set<Character> result = leftOperand.getVariables();
        result.addAll(rightOperand.getVariables());
        return result;
    }
}

public class Negation implements BooleanFormula {
    private BooleanFormula operand;

    public Negation(BooleanFormula operand) {
        this.operand = operand;
    }
    public boolean getValue(Assignment a) {
        return ! operand.getValue(a);
    }
    public Set<Character> getVariables() {
        return operand.getVariables();
    }
}
```

c)

```
public class Satisfiability {
    private static Assignment
    makeAssignment(Set<Character> variables,int bits)
    {
        Assignment assignment = new MapAssignment();
        for ( Character variable : variables ) {
            assignment.assign(variable,bits % 2 == 1);
            bits /= 2;
        }
        return assignment;
    }
    public static boolean
    isSatisfiable(BooleanFormula f)
    {
        Set<Character> variables = f.getVariables();
        for ( int i = 0; i < (int)Math.pow(2,variables.size()); i++ ){
            if ( f.getValue(makeAssignment(variables,i)) )
                return true;
        }
        return false;
    }
}
```