TENTAMEN: Algoritmer och datastrukturer

Läs detta!

- *Uppgifterna är inte avsiktligt ordnade efter svårighetsgrad.*
- Börja varje uppgift på ett nytt blad.
- Skriv ditt idnummer på varje blad (så att vi inte slarvar bort dem).
- **Skriv rent dina svar. Oläsliga svar r ä t t a s e j!**
- Programkod skall skrivas i Java 5 eller senare version, vara indenterad och renskriven, och i övrigt vara utformad enligt de principer som lärts ut i kursen.
- Onödigt komplicerade lösningar ger poängavdrag.
- Programkod som finns i tentamenstesen behöver ej upprepas.
- Givna deklarationer, parameterlistor, etc. får ej ändras, såvida inte annat sägs i uppgiften.
- Läs igenom tentamenstesen och förbered ev. frågor.

I en uppgift som består av flera delar får du använda dig av funktioner klasser etc. från tidigare deluppgifter, även om du inte löst dessa.

Lycka till!

Uppgift 1

Välj **ett** svarsalternativ. Motivering krävs ej. Varje korrekt svar ger två poäng. Garderingar ger noll poäng.

1. Vilken teknik kan effektivisera rekursiva algoritmer med många överlappande fall?
   a. divide & conquer
   b. dynamisk programmering
   c. snål algoritm
   d. backtracking

2. Antag att man vill konstruera en algoritm som läser strängar av parenteser och undersöker om varje vänsterparentes har en matchande högerparentes. Ex. "([]){()()[]{()()()}}" är korrekt matchad, men inte "([]){([]{()()()}". Vilken ADT lämpar sig för att lösa problemet?
   a. en stack
   b. en FIFO-kö
   c. en tabell (map)
   d. det behövs ingen ADT, en heltalsvariabel för varje parentestyp räcker

3. Vilket av påståendena stämmer om sortering?
   a. samsortering (merge sort) kan, liksom Quicksort, sortera utan extra minne
   b. tidskomplexiteten för insättningssortering är i bästa fall $\Theta(n)$
   c. Shellsort är en vidareutveckling av Quicksort
   d. tidskomplexiteten för Quicksort är i värsta fall $O(n \cdot \log n)$

4. I de sökalgoritmer för riktade grafer som tagits upp i kursen används ADT:er, vilken kombination är korrekt?
   a. kortaste oviktade avståndet: prioritetskö
   b. kortaste oviktade avståndet: stack
   c. kortaste viktade avståndet: prioritetskö
   d. kortaste viktade avståndet: disjoint sets
   e. kortaste viktade avståndet: FIFO-kö

5. En av rekurensekvationerna1-3 beskriver tidskomplexiteten i värsta fall hos f.

   1.
   $$\begin{cases} T(0) = 0 \\ T(1) = 1 \\ T(N) = 2T(N/2) \end{cases}$$

   2.
   $$\begin{cases} T(0) = 0 \\ T(1) = 1 \\ T(N) = T(N/2) + 1 \end{cases}$$

   3.
   $$\begin{cases} T(0) = 0 \\ T(1) = 0 \\ T(N) = 2T(N/2) + N \end{cases}$$

```
boolean f(int[] a,int i,int j,int x) {
   if (i > j )
      return false;
   else if ( i == j )
       return a[i] == x;
   else {
      int mid = (i + j)/2;
      return f(a,i,mid,x) || f(a,mid+1,j,x);
   }
}
```

   Vilket av a-d är en minsta övre begränsning för lösningen till ekvationen i fråga?
   a. $O(\log N)$
   b. $O(N^2)$
   c. $O(N \cdot \log N)$
   d. $O(N)$

(10 p)

Uppgift 2

Denna uppgift behandlar symbolisk förenkling av aritmetiska uttryck bestående av konstanter (tal), variabler, addition och multiplikation. Exempel:

((0 + x) + (y + 0))
((1*(0 + x)) + ((y + 0)*0))

Normalt skriver vi inte som ovan utan förenklar genom att tillämpa de aritmetiska lagarna (*E* betecknar ett godtyckligt uttryck):

$$\begin{aligned} 0 + E &= E \\ E + 0 &= E \\ 1*E &= E \\ E*1 &= E \\ 0*E &= 0 \\ E*0 &= 0 \end{aligned}$$

Genom att tillämpa lagarna kan exemplen ovan förenklas till (x + y) respektive x. I bilagan sist i tesen finns klasser för att representera uttryck som Java-objekt. Uppgiften är att komplettera klasserna med metoder för konvertering av uttryck till textform samt förenkling enligt ovan.

a) Implementera metoden `toString` i en av klasserna `Addition` eller `Multiplikation`. Metoden skall vara rekursiv.

(2 p)

b) Implementera metoden `simplify` i en av klasserna `Addition` eller `Multiplikation`. Metoden skall vara rekursiv. Du får max 6 p för `simplify` i `Addition` och max 8 p för `simplify` i `Multiplikation`.

(6 p eller 8 p)

Exempel:

```
Expression zero = new Constant("0");
Expression one = new Constant("1");
Expression x = new Variable("x");
Expression y = new Variable("y");
Expression e1 = new Addition(zero,x);
Expression e2 = new Addition(y,zero);
Expression e3 = new Addition(e1,e2);
System.out.println(e3);                Utskrift: ((0 + x) + (y + 0))
System.out.println(e3.simplify());     Utskrift: (x + y)

Expression e4 = new Multiplication(one,e1);
Expression e5 = new Multiplication(e2,zero);
Expression e6 = new Addition(e4,e5);
System.out.println(e6);         Utskrift: ((1*(0 + x)) + ((y + 0)*0))
System.out.println(e6.simplify());  Utskrift: x
```

Uppgift 3

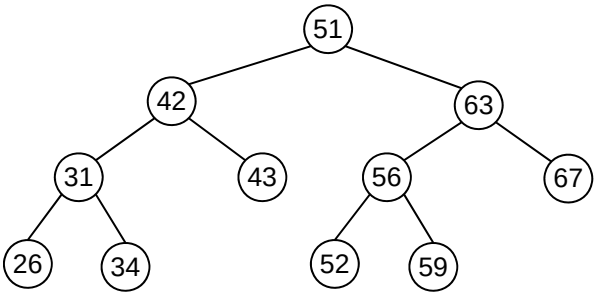
a) Ange fem olika insättningsordningar av talen 7, 2, 6, 3, 5, 1 och 4 som ger ett perfekt balanserat sökträd (om det är tomt från början). Inga rotationer skall göras. För poäng krävs att alla svaren är rätt.

(2 p)

b) Ge en formel för det maximala antalet noder i ett fullt binärt träd som en funktion av höjden (höjd enligt Weiss).

(1 p)

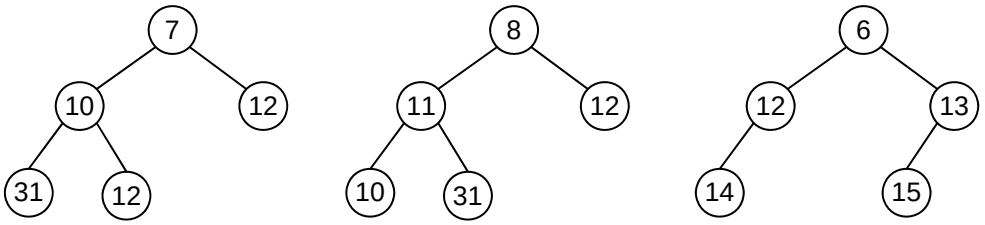
c) Rita det binära sökträdet nedan efter anropen `insert(28);insert(57)`. Genomför tillämpliga AVL-rotationer. Du behöver inte visa mellanresultaten, men det kan underlätta rättningen om de är med.



(4 p)

Uppgift 4

a) Vilket av träden är en binär hög? Motivera!



(3 p)

b) Ange hur fältet nedan ser ut efter ett anrop av metoden `buildHeap` (enl. kursboken) som arrangerar om elementen så att de utgör en binär hög. Rita även resultatet som ett träd.

-∞	13	35	10	25	20	12	14	27	22	30
0	1	2	3	4	5	6	7	8	9	10

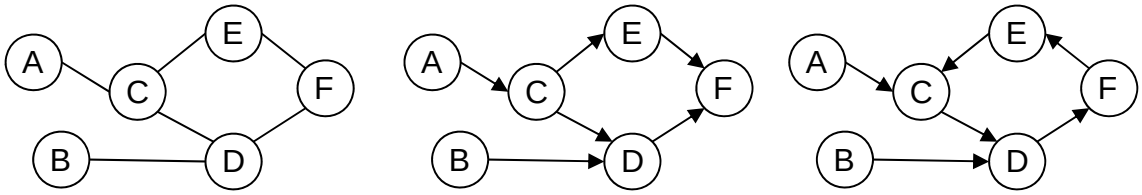
(4 p)

Uppgift 5

a) Ge ett exempel på en sammanhängande graf med minst tre noder där det kortaste avståndet från en startnod till alla övriga noder inte kan beräknas med Dijkstras algoritm. Motivera!

(3 p)

b) Ange alla möjliga topologiska ordningar av noderna i den av graferna nedan som uppfyller de nödvändiga kriterierna. Vilka är kriterierna?



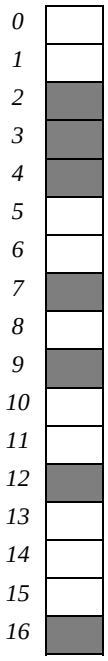
(6 p)

Uppgift 6

a) Vilka krav måste vara uppfyllda då kvadratisk sondering används i en hashtabell?

(2 p)

b) Om objektet X får hashvärdet 3, var sätts det in med kvadratisk sondering i hashtabellen nedan? Gråmarkerade positioner är upptagna.



(3 p)

Uppgift 7

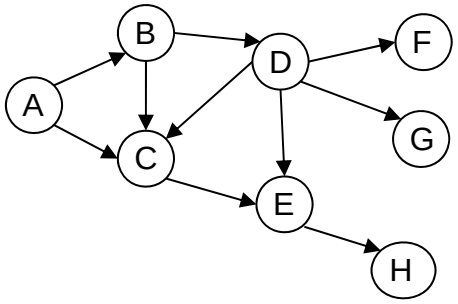
Riktade grafer kan representeras med följande klasser:

```
public class Vertex {
   String name;
   boolean visited = false;
   int dist = 0;
   List<Vertex> neighbours;
   public Vertex(String name) {
      this.name = name;
      neighbours = new LinkedList<>();
   }
   public String toString() {
      return name;
   }
}
```

```
public class Graph
   private Map<String,Vertex>
      graphTable = new HashMap<>();

   public void addVertex(String name) {
      if ( ! graphTable.containsKey(name) )
          graphTable.put(name,new Vertex(name));
   }
   public void addEdge(String from,String to) {
      addVertex(from);
      addVertex(to);
      graphTable.get(from).neighbours.add(graphTable.get(to));
   }
   public void levelOrder(String startVertex) { /* TODO */ }
}
```

Metoden `levelOrder` skall skriva ut noder i grafen i bredden-först-ordning (BFS).



I exemplet ovan skall anropet `levelOrder("B")` ge utskriften:

```
0: B
1: C D
2: E F G
3: H
```

Det är alltså noderna i subgrafen som kan nås från den valda startnoden som skall skrivas ut. Som exemplet visar skall alla noder på samma avstånd från startnoden skrivas på samma rad och avståndet stå i början av raden. Den inbördes ordningen mellan namnen i raderna spelar ingen roll. Använd en lämplig ADT i lösningen.

(12 p)

BILAGA till uppgift 2

```
public interface Expression {
    boolean isZero();
    boolean isOne();
    String toString();
    Expression simplify();
}

public abstract class AbstractExpression implements Expression {
    public boolean isZero()        { return false; }
    public boolean isOne()         { return false; }
}

public class Constant extends AbstractExpression  {
    private String digits;
    public Constant(String digits) { this.digits = digits; }
    public String toString()       { return digits;}
    public boolean isZero()        { return digits.equals("0"); }
    public boolean isOne()         { return digits.equals("1");}
    public Expression simplify()   { return new Constant(digits); }
}

public class Variable extends AbstractExpression {
    private String name;
    public Variable(String name)   { this.name = name; }
    public String toString()       { return name; }
    public Expression simplify()   { return new Variable(name); }
}

public abstract class AbstractBinaryExpression extends AbstractExpression {
    protected Expression leftOperand,rightOperand;
    public AbstractBinaryExpression(Expression leftOperand,
                                    Expression rightOperand) {
        if ( leftOperand == null || rightOperand == null )
            throw new IllegalStateException();
        this.leftOperand = leftOperand;
        this.rightOperand = rightOperand;
    }
}

public class Addition extends AbstractBinaryExpression {
    public Addition(Expression leftOperand,Expression rightOperand) {
        super(leftOperand,rightOperand);
    }
    public String toString() { /* TODO */ }
    public Expression simplify() { /* TODO */ }
}

public class Multiplication extends AbstractBinaryExpression {
    public Multiplication(Expression leftOperand,Expression rightOperand) {
        super(leftOperand,rightOperand);
    }
    public String toString() { /* TODO */ }
    public Expression simplify() { /* TODO */ }
}
```