TENTAMEN: Algoritmer och datastrukturer

Läs detta!

- *Uppgifterna är inte avsiktligt ordnade efter svårighetsgrad.*
- Börja varje uppgift på ett nytt blad. Skriv inga lösningar i tesen.
- Skriv ditt idnummer på varje blad (så att vi inte slarvar bort dem).
- Programmen skall skrivas i Java 5 eller senare, vara indenterade och renskrivna, och i övrigt vara utformade enligt de principer som lärts ut i kursen.
- **Skriv rent dina svar. Oläsliga svar r ä t t a s e j!**
- 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 på varje fråga. Motivering krävs ej. Varje korrekt svar ger två poäng. Garderingar ger noll poäng.

1. Vilka av metoderna 1-3 måste vara implementerade för typen T vid användning av resp. standardklass A-C i Java? Välj det alternativ som anger ett nödvändigt och tillräckligt urval av metoder.

A. `TreeSet<T>`	1. `equals`
B. `HashSet<T>`	2. `compareTo` eller `compare`
C. `ArrayList<T>`	3. `hashCode`

a. A:1, B:2, C:2
b. A:1+2, B:1+2, C:1
c. A:3, B:2, C:1
d. A:1+2, B:1+3, C:1
e. A:2, B:3, C:1

2. Om fältdubblering används vid implementering av en hashtabell och kvadratisk sondering används, vilket genomsnittligt minnesutnyttjande får man ungefär i tabellen?
   a. 10%
   b. 25%
   c. 38%
   d. 50%
   e. 75%

3. Ange en minsta övre begränsning för f(n), definierad av ekvationerna

$$f(0) = 0$$
$$f(n) = 2f(n-1) + 1 \qquad (n > 0)$$

   a. $O(2^n)$
   b. $O(n \log n)$
   c. $O(n^2)$
   d. $O(\log n)$

4. Sorteringsalgoritmer som baseras på successiva byten av närliggande element har för genomsnittliga indata tidskomplexiteten
   a. $O(n \log n)$
   b. $\Omega(n^2)$
   c. $O(n^2)$

forts.

5. Metoden `getReverse` tar en enkellänkad lista som argument och returnerar innehållet i omvänd ordning i en standardlista. Vilken av de fyra metoderna är korrekt implementerad?

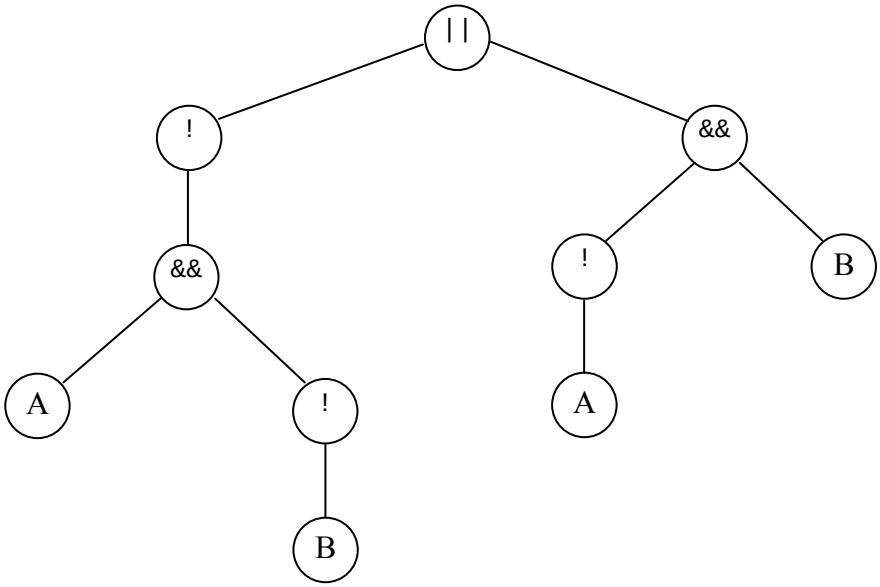
```
public class ListNode {
    int element;
    ListNode next;
}

public class Lists {
    private static ArrayList<Integer> result = new ArrayList<Integer>();

    public static List<Integer> getReverse1(ListNode l) {
        if ( l == null )
            return result;
        else {
            getReverse1(l.next);
            result.add(l.element);
            return result;
        }
    }

    public static List<Integer> getReverse2(ListNode l) {
        List<Integer> result = new ArrayList<Integer>();
        if ( l == null )
            return result;
        else {
            getReverse2(l.next);
            result.add(l.element);
            return result;
        }
    }

    public static List<Integer> getReverse3(ListNode l) {
        if ( l == null )
            return new ArrayList<Integer>();
        else {
            List<Integer> result = getReverse3(l.next);
            result.add(l.element);
            return result;
        }
    }

    public static List<Integer> getReverse4(ListNode l) {
        if ( l == null )
            return null;
        else {
            List<Integer> result = getReverse4(l.next);
            result.add(l.element);
            return result;
        }
    }
}
```

a. `getReverse1`
b. `getReverse2`
c. `getReverse3`
d. `getReverse4`

(10 p)

Uppgift 2

Logiska uttryck med variabler och operatorerna `&&`, `||` och `!` kan representeras som träd, t.ex.



Konstruera en rekursiv javametod som skriver ut ett godtyckligt sådant uttryckträd med parenteser runt alla deluttryck, utom runt variabler.

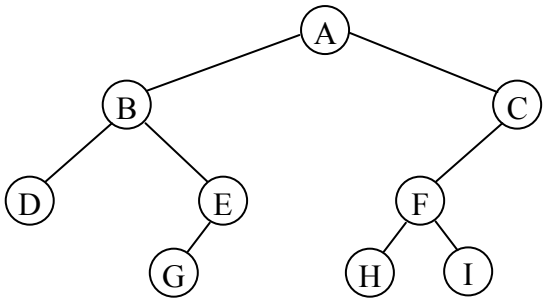
```
public static void prettyPrint( TreeNode t )
```

Trädet ovan skall t.ex. skrivas ut som `((!(A&&(!B)))||((!A)&&B))`. Klassen `TreeNode` för representation av uttryckträd finns i bilaga.

(10 p)

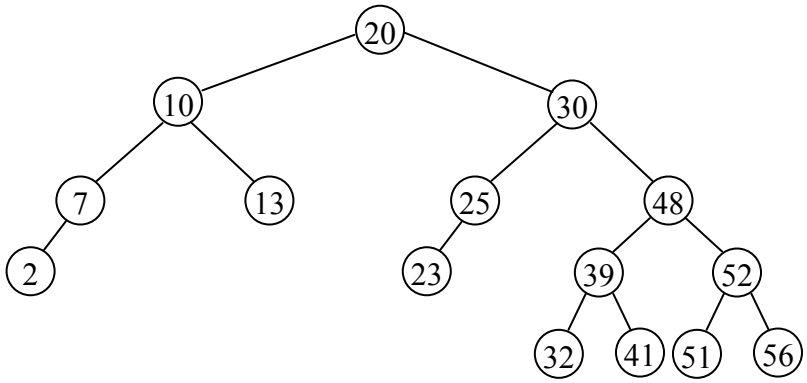
Uppgift 3

a) Ange i vilken ordning noderna besöks vid preorder, postorder, resp. inorder genomlöpning av trädet



(6 p)

b) Visa hur AVL-trädet nedan ser ut efter insättning av 53 och lämplig AVL-rotation



(4 p)

Uppgift 4

a) Visa med en figur hur en initialt tom hashtabell med 11 platser ser ut efter sekvensen

```
add(21); add(10); add( 43); remove(10); add(43);
```

Kvadratisk sondering används vid insättning och hashfunktionen definieras hash(x) = x mod 11. Förklara vad som händer vid varje anrop ovan.

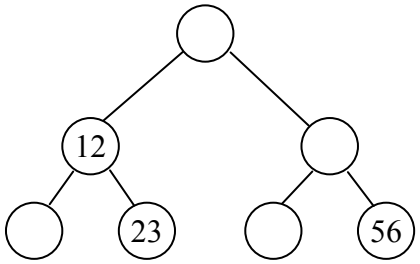
(4 p)

b) Hur många platser kan ockuperas i tabellen i a innan omhashning till en större tabell måste ske? Vilken är den minsta storlek man då bör välja om vi antar att det nya fältet skall vara minst dubbelt så stort?

(3 p)

Uppgift 5

Stoppa in talen 7, 27, 34 och 48 i trädet



så att det blir

a) en binär hög.

(3 p)

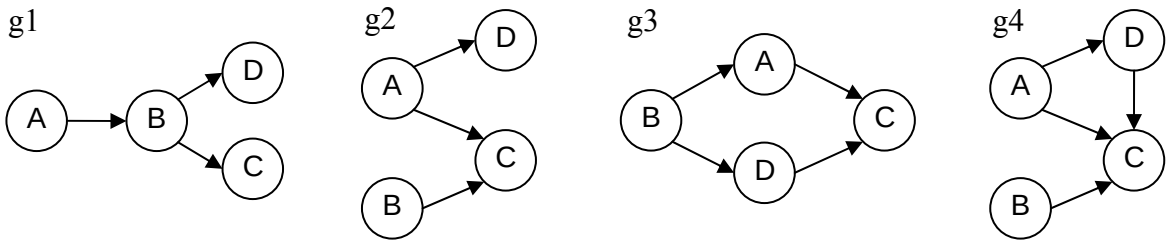
b) ett binärt sökträd.

(3 p)

Du skall alltså rita två träd.

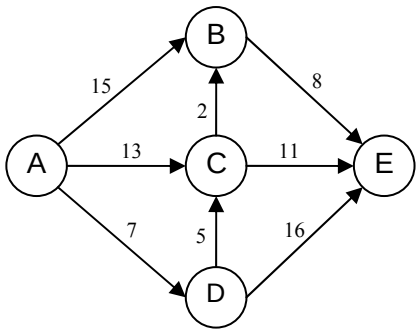
Uppgift 6

a) För en av graferna g1 – g4 är sekvenserna ABCD och BADC möjliga topologiska ordningar av noderna. Vilken är grafen? Motivera svaret! Lista övriga topologiska ordningar för grafen i fråga.



(3 p)

Betrakta grafen



b) I vilken ordning besöks noderna vid bestämning av de kortaste oviktade avstånden från A till samtliga övriga noder?

(1 p)

c) I vilken ordning besöks noderna i Dijkstras algoritm vid bestämning av de kortaste viktade avstånden från A till samtliga övriga noder?

(1 p)

d) Ange de kortaste viktade avstånden från A till samtliga övriga noder.

(1 p)

e) Vilken datastruktur, förutom graftabellen, är vital i Dijkstras algoritm?

(1 p)

Uppgift 7

Ett naturligt tal *N* är ett *Hammingtal* om det kan skrivas som en produkt av potenser av talen 2, 3 och 5, d.v.s. om det finns $a, b, c \geq 0$, så att $N = 2^a \cdot 3^b \cdot 5^c$. De minsta Hammingtalen är 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, ... Implementera metoden

```
public static void hamming( int n )
```

som skriver ut de *N* första Hammingtalen ($N \geq 0$). Talen skall skrivas ut i växande följd och inget tal får skrivas ut två gånger. För poäng på uppgiften krävs att lämplig datastruktur används, algoritmen får inte baseras på "bortdividerande" av faktorer.

(10 p)

Bilaga till uppgift 2

```
public class TreeNode {

      // Variable
      public TreeNode( String variable ) {
            nodeType = NodeType.VARIABLE;
            this.value = variable;
            left = right = null;
      }

      // Unary expressions
      public TreeNode( String operator, TreeNode right ) {
            nodeType = NodeType.OPERATOR;
            this.value = operator;
            this.left = null;
            this.right = right;           // just one operand
      }

      // Binary expressions
      public TreeNode( String operator, TreeNode left, TreeNode right ) {
            nodeType = NodeType.OPERATOR;
            this.value = operator;
            this.left = left;
            this.right = right;
      }

      public enum NodeType { VARIABLE, OPERATOR }
      public NodeType nodeType;
      public String value;               // A variable or an operator
      public TreeNode left, right;       // The (left) and right operand(s)
```