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 sänka vissa algoritmers tidskomplexitet med hjälp av extra minne?
   a. Divide & Conquer
   b. dynamisk programmering
   c. snåla algoritmer
   d. rekursion

2. Vid implementering av stackar och köer med fält eftersträvas s.k. kostnadsamortering vid omallokering av fältet, vilket kräver att det nya fältet dimensioneras så att
   a. storleken är en tvåpotens
   b. det är ett konstant antal element längre än det föregående
   c. storleken är ett primtal
   d. det är dubbelt så stort som det föregående

3. Vilket begrepp är speciellt användbart vid analys av sortering?
   a. iteration
   b. klusterbildning
   c. rekursion
   d. inversion
   e. rotation

4. Vid nivåvis genomlöpning av träd används lämpligen
   a. en prioritetskö och iteration
   b. en FIFO-kö och iteration
   c. en hashtabell och rekursion
   d. en stack och rekursion
   e. en stack och iteration

5. Ange minsta övre begränsningar för funktionernas tidskomplexiteter:

```
public static int f1(int n) {
   if ( n == 0 )
      return 0;
   else {
      int sum = 0;
      for (int i = 0; i < n; i++)
         sum++;

      return sum + f1(n/2);
   }
}
```

```
public static int f2(int n) {
   if ( n == 0 )
      return 0;
   else {
      int sum = 0;
      for (int i = 0; i < n/2; i++)
         sum++;

      return sum + f2(n-1);
   }
}
```

   a. `f1` är $O(n)$, `f2` är $O(n^2)$
   b. `f1` är $O(n)$, `f2` är $O(n \cdot \log n)$
   c. `f1` är $O(n \cdot \log n)$, `f2` är $O(n^2)$
   d. Båda är $O(n^2)$
   e. Båda är $O(n)$
   f. Båda är $O(n \cdot \log n)$

(10 p)

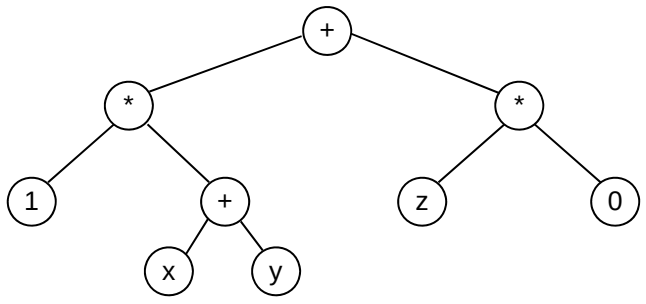
Uppgift 2

Denna uppgift kan lösas i två svårighetsgrader. Lös antingen a (5 p) eller b (12 p). Du får bara poäng för en av dem.

Följande typ kan användas för att representera aritmetiska uttryck som binära träd. Uttrycken kan innehålla variabler, heltalskonstanter, samt operatorerna + och *:

```
public class Expression {
    private char variable;
    private int constant;
    private char operator;
    private enum Type { VARIABLE,CONSTANT,PLUS,MULT }
    private Type t;
    private Expression left,right;        // The operands
    public Expression(char variable) {    // Variable leaf
        this.variable = variable;
        t = Type.VARIABLE;
        this.left = this.right = null;
    }
    public Expression(int constant) {     // Constant leaf
        this.constant = constant;
        t = Type.CONSTANT;
        this.left = this.right = null;
    }
                                          // Operator node
    public Expression(char operator,Expression left,Expression right) {
        this.operator = operator;
        if ( operator == '+' ) t = Type.PLUS;
        else if ( operator == '*' ) t = Type.MULT;
        else throw
          new IllegalArgumentException("Unsupported operator: " + operator);
        this.left = left; this.right = right;
    }
    public boolean isVar() { return t == Type.VARIABLE; }
    public boolean isConst() { return t == Type.CONSTANT; }
    public boolean isPlus() { return t == Type.PLUS; }
    public boolean isMult() { return t == Type.MULT; }
    public char getVar() { return variable; }
    public int getConst() { return constant ; }
    public char getOp() { return operator; }
    public Expression getLeft() { return left; }
    public Expression getRight() { return right; }
}
```

Exempel: Uttrycket `((1*(x+y))+(z*0))` kan representeras med trädet



forts.

och med typen `Expression`:

```
Expression e =
      new Expression('+',
              new Expression('*',
                      new Expression(1),
                      new Expression('+',
                              new Expression('x'),
                              new Expression('y')
                          )
                  ),
              new Expression('*',
                      new Expression('z'),
                      new Expression(0)
                  )
          )
```

Lös antingen a eller b. Du får bara poäng för en av dem.

a) Konstruera en rekursiv `toString`-metod för klassen `Expression`.
T.ex. skall `e.toString()` returnera strängen `"((1*(x+y))+(z*0))"`

(5 p)

b) Uttrycket i a kan förenklas till `(x+y)` genom att använda de algebraiska lagarna

x + 0 = x
0 + x = x
x * 1 = x
1 * x = x
x * 0 = 0
0 * x = 0

Konstruera metoden

```
public static Expression simplify(Expression e)
```

som rekursivt förenklar uttryckskträd genom att tillämpa lagarna ovan. Variabler, konstanter samt `null` förenklas förstås till sig själva. Om båda operanderna är konstanter skall uttrycket förenklas till sitt numeriska värde. Ex. förenklas 3+2 till 5 och 7*4 till 28.

Ex. `(simplify(e)).toString()` skall returnera strängen `"(x+y)"`.

(12 p)

Uppgift 3

a) Rita den binära högen både som träd och som fält efter insättningssekvensen 10,20,15,5,8, 22,9,6,18 i en tom hög. Operationen `insert` avses.

(3 p)

b) Om man istället placerar talen 10,20,15,5,8,22,9,6,18 (i denna ordning) i fältet i a och sedan utför operationen `buildHeap`, kommer då den resulterande högen att se exakt likadan ut? Om du anser det så ge ett övertygande argument, om inte visa hur högen kommer att se ut.

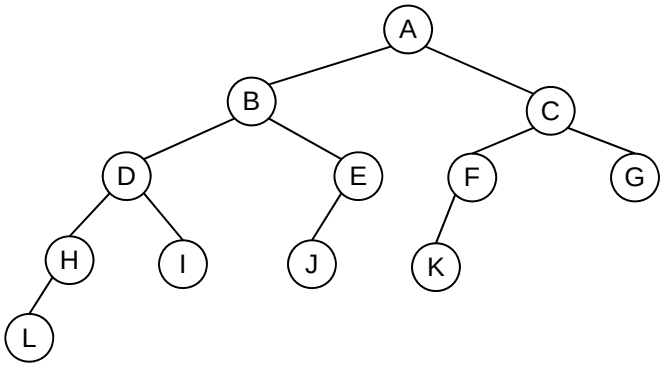
(3 p)

Uppgift 4

a) Rita trädet som fås efter insättningssekvensen 30,60,40,70,20,25,15,26 i ett tomt AVL-träd. (Slutresultatet efterfrågas, men det kan vara bra att ta med ev. rotationssteg för att underlätta rättningen.)

(3 p)

b) Ange inorder, preorder samt postorder genomlöpningar av trädet:



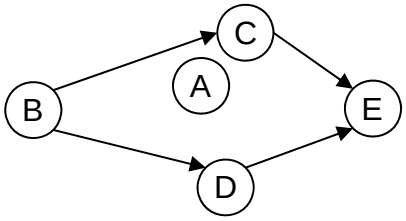
(3 p)

Uppgift 5

a) Vilken datastruktur spelar en avgörande roll i en effektiv implementering av Dijkstras algoritm? Rita dessutom en graf för vilken Dijkstras algoritm inte går att tillämpa.

(2 p)

b) Ange alla möjliga topologiska ordningar av noderna i grafen (som är korrekt ritad):



(5 p)

c) Ange med ett ordouttryck en minsta övre begränsning för hur många topologiska ordningar det kan finnas för en graf med N noder. *Tips*: Grafen i b innehåller en ledtråd.

(1 p)

Uppgift 6

I hashtabellen nedan tillämpas kvadratisk sondering för kollisionshantering. Hashfunktionen är *hash*(*x*) = *x* mod *M*, där *M* är tabellstorleken.

a) Utför sekvensen

```
insert(9);
insert(24);
remove(9);
insert(21);
remove(24);
insert(31);
insert(10);
remove(10);
```

i en hashtabell med 11 platser och visa med en figur hur resultatet blir.

(4 p)

b) Med utgångspunkt från resultatet i a, visa tabellens utseende efter `insert(13);` och omhashning. Den nya tabellen skall dimensioneras i enlighet med principen för fältdubblering, men tänk efter noga vad denna storlek skall vara i det här fallet.

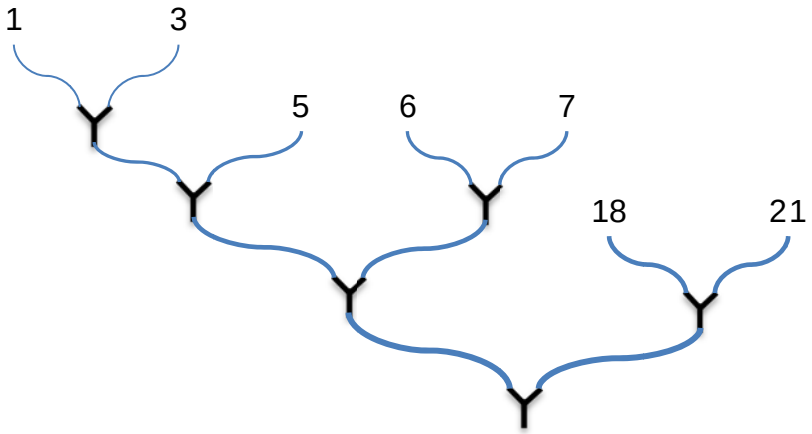
(4 p)

Uppgift 7

Ett bevattningssystem för växter skall byggas av slangar av olika dimensioner, samt förgreningskopplingar, s.k. Y-rör (eng. split). Precis som vid brandsläckning bör systemet dimensioneras så att de tunnare slangarna leds till växterna med låg vattenförbrukning och de grövre slangarna till de med hög förbrukning. Dessutom är det fördelaktigt att placera storförbrukarna nära vattenkällan och småförbrukarna längre bort. Nedan presenteras några växter med sina förväntade vattenförbrukningar i liter per vecka:

1 3 5 6 7 18 21

Om vi tänker oss en liten anläggning med ett exemplar av varje av varje växt så finns ett optimalt sätt att konfigurera slangar och kopplingar:



Detta liknar principen naturen använder för att dimensionera och förgrena kärl i växter och djur.

forts.

Uppgiften är att konstruera en algoritm som givet vattenbehoven för ett antal växter kan skapa ett nätverk av ovanstående typ. Till vår hjälp finns klasserna:

```
public abstract class Consumer implements Comparable<Consumer> {

    public abstract int getDemand();

    public int compareTo(Consumer other) {
        if ( getDemand() < other.getDemand() )
            return -1;
        else if ( getDemand() > other.getDemand() )
            return 1;
        else
            return 0;
    }
}

public class Plant extends Consumer {

    private int demand;

    public Plant(int demand) { this.demand = demand; }

    public int getDemand() { return demand; }
}

public class Split extends Consumer {  // Y-rör

    private Consumer left,right;

    public Split(Consumer left,Consumer right) {
        this.left = left; this.right = right;
    }

    public int getDemand() {
        return left.getDemand() + right.getDemand();
    }
}
```

Uppgift: Konstruera metoden

```
public static Consumer computeNetwork(Collection<Integer> demands)
```

som givet en lista av vattenförbrukningar för ett antal växter skapar och returnerar ett förgreningsnät av samma typ som visas i exemplet ovan. Om listan är `null` eller tom skall undantaget `IllegalArgumentException` kastas. Använd en lämplig datastruktur i algoritmen. Metoden skall alltså returnera roten till trädet, vilket motsvarar ingången på det största Y-röret längst ner i figuren på förra sidan.

(10 p)