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!**
- Programmen skall skrivas i Java 5 eller senare version, vara indenterade och renskrivna, och i övrigt vara utformade 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. Om vi implementerar en FIFO-kö med en enkellänkad lista så får operationerna enqueue, front och dequeue alla tidskomplexiteten
   a. $O(1)$ i genomsnitt och $O(N)$ i värsta fall
   b. $O(N)$ i genomsnitt och $O(N)$ i värsta fall
   c. $O(1)$ i genomsnitt och $O(1)$ i värsta fall
   d. $O(N)$ i genomsnitt och $O(1)$ i värsta fall

2. Avsikten med följande algoritm är att beräkna största delsegmentsumman i ett heltalsfält genom att utnyttja tekniken "divide and conquer"

```
private static int maxSum(int[] a,int lo,int hi) {
    if ( lo > hi )
        return 0;
    if ( lo == hi )
        return a[lo];
    else {
        int mid = (lo+hi)/2;
        return Math.max(maxSum(a,lo,mid),maxSum(a,mid+1,hi));
    }
}
public static int maxSum(int[] a) { return maxSum(a,0,a.length-1); }
```

   Vilket påstående är rätt?
   a. implementeringen är korrekt
   b. den rekursiva `maxSum` terminerar ej i vissa fall
   c. metoden löser ej problemet

3. Ange en undre begränsning för den genomsnittliga tidskomplexiteten för sortering av n slumpmässigt valda heltal med en maskin som kan göra högst en jämförelse i taget?
   a. $O(\log n)$
   b. $\Omega(n \cdot \log n)$
   c. $O(n \cdot \log n)$
   d. $O(n)$
   e. $\Omega(n)$
   f. $\Omega(n^2)$

4. Ineffektiva rekursiva algoritmer med många överlappande fall kan ofta göras mer effektiva genom att utnyttja
   a. en snål algoritm
   b. backtracking
   c. divide and conquer
   d. dynamisk programmering

5. Ett av sambanden 1-3 beskriver tidskomplexiteten hos f.

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

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

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

```
int f(int n) {
    if ( n == 0 )
        return 0;
    else {
        int s = f(n-1);
        for ( int i = n; i > 0; i /= 2 )
            s += 1;

        return s;
    }
}
```

Vilket av a-d är en minsta övre begränsning för lösningen till sambandet i fråga?

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

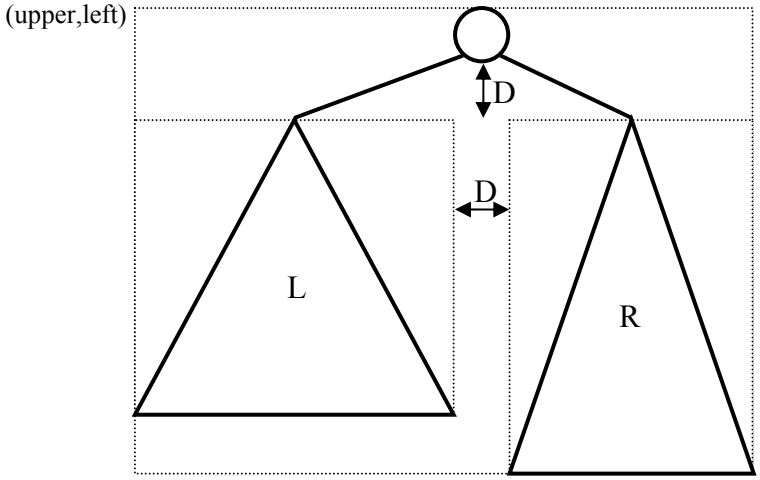
(10 p)

Uppgift 2

Följande typ kan användas för att representera noder i ett binärt träd.

```
public class TreeNode {
    int element;
    TreeNode left,right;

    public boolean isLeaf() {
       return left == null && right == null;
    }
}
```

Om man vill rita upp ett träd i ett fönster behöver först trädets utsträckning i planet beräknas. Som grund för beräkningen kan man ta storleken hos en nod. I den här uppgiften antar vi att noderna skall ritas som runda cirklar med diametern D. Inbördes avstånd i trädet framgår av följande generella figur.



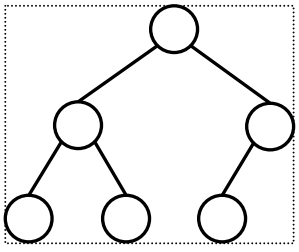
Skriv en rekursiv funktion som returnerar en begränsande rektangel till ett binärt träd.

```
public static Rect boundingRect(TreeNode t,int upper,int left)
```

I figuren ovan har begränsande rektanglar ritats ut för delträden, samt en för hela trädet. Trädet och rektangelns övre vänstra begränsningspunkt skall ges som argument till funktionen. För ett tomt träd skall den punktformiga rektangeln (upper,left,upper,left) returneras och för ett löv (upper,left,upper+D,left+D). Rektanglar representeras med typen

```
public class Rect {
    public int upper,left,lower,right;
}
```

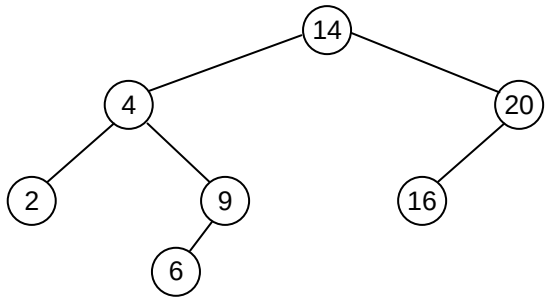
Exempel: Om noderna skall ritas med diametern 10 enheter och trädet placeras i ritfönstrets övre vänstra hörn så får nedanstående träd den begränsande rektangeln (0,0,50,60).



(10 p)

Uppgift 3

a) Antag att vi skall sätta in *ett* av talen 1, 3, 5, 7, 11, 15, 18, 23 i det binära sökträdet nedan. I vilka fall skulle en sådan insättning bryta mot AVL-villkoret?



(2 p)

b) Låt $\min_{AVL}(h)$ = minimala antalet noder i ett AVL-träd av höjd h. Ex. Trädet i a är ett minimalt AVL-träd av höjd 3 och $\min_{AVL}(3) = 7$. Definiera $\min_{AVL}$ med rekursionsekvationer (analogt med exempel som förekommit i kursen).

(4 p)

Uppgift 4

a) Vad är tidskomplexiteterna för operationerna insert, resp. deleteMin i en binär hög i det genomsnittliga fallet?

(2 p)

b) Kan ett binärt träd med minst två noder vara både ett binärt sökträd och en binär hög? Ge i så fall exempel, annars en motivering till varför det skulle vara omöjligt.

(4 p)

Uppgift 5

En graf har följande bågar:
{(A,B,2), (A,G,1), (B,D,7), (B,E,3), (B,A,19), (C,A,4), (D,A,10), (D,C,6), (D,F,2), (E,A,35), (E,B,15), (E,G,21), (F,C,3), (F,G,0), (G,A,12), (G,C,8), (G,F,4)}

a) Rita grafen så att inga bågar korsar varandra.

(4 p)

b) Bestäm kortaste viktade avstånden från nod E till samtliga övriga noder och ange i vilken ordning noderna besöks med Dijkstras algoritm.

(6 p)

Uppgift 6

I en tillämpning bestämde man sig för att lagra information om städer i en hashtabell. I tabellen tillämpas kvadratisk sondering för kollisionshantering. Objekten som lagras i tabellen har typen:

```
public class City {
    private String position;
    private String name;
    public City(String position,String name) { ... }
    public boolean equals(Object o);
    public int hashCode();
    // other members omitted
}
```

Följande objekt skall sättas in i en tabell `tab` med 11 platser:

```
City gbg        = new City(”57,42’N 11,57’Ö”,”Göteborg”),
     alingsås   = new City(”57,47'N 13,24’Ö”,”Alingsås”),
     borås      = new City(”57,43’N 12,56’Ö”,”Borås”),
     kungsbacka = new City(”57,29’N 12,4’Ö”, ”Kungsbacka”),
     ulricehamn = new City(”57,47'N 13,24’Ö”,”Ulricehamn”);
```

Metoderna `equals` och `hashCode` beräknas från objektens positioner. Två objekt är lika om deras positionssträngar är lika. `hashCode` returnerar följande värden:

```
gbg.hashCode()            4
borås.hashCode()          4
kungsbacka.hashCode()     6
alingsås.hashCode()       5
ulricehamn.hashCode()     5
```

Betrakta sekvensen:

```
tab.insert(gbg);
tab.insert(kungsbacka);
tab.insert(borås);
tab.insert(alingsås);
tab.remove(borås);
tab.insert(ulricehamn);
tab.find(alingsås);
```

Vilket objekt returneras av `find`? Motivera! Rita upp tabellen och visa var elementen hamnar.

(*Notera att ovanstående delvis får betraktas som pseudokod. Metodnamnen avviker t.ex. från namnen i javas API.)*

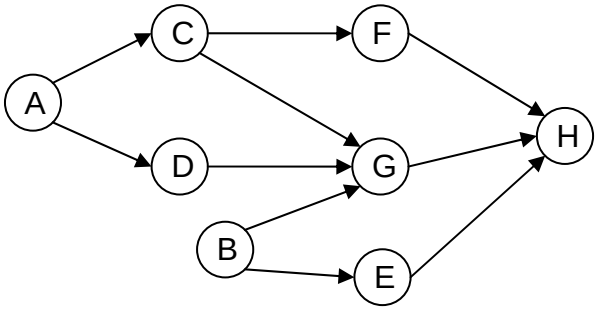
(8 p)

Uppgift 7

Följande klasser kan användas för att representera riktade grafer (utan kostnader på bågarna):

```
class GraphException extends RuntimeException {
    public GraphException(String name) {
        super(name);
    }
}

// Represents a vertex in the graph.
class Vertex {
    public String       name;   // Vertex name
    public List<Vertex> adj;    // Adjacent vertices
    public int          indegree;

    public Vertex(String name) {
        this.name = name;
        adj = new LinkedList<Vertex>();
        indegree = 0;
    }
}

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

    ... // other methods omitted

   // This method is finished and may be used in your code.
    public boolean hasCycle() { ... }

    // Print all nodes in an arbitrary topological order.
    // Implement this method!
    public void topologicalSort() { ... }
}
```

Metoden `topologicalSort` skall, om möjligt, beräkna och skriva ut alla nodernas namn i en godtycklig topologisk ordning. Om ingen sådan ordning finns skall undantaget `GraphException` kastas.
Exempel: För grafen nedan är B A C E D F G H en av flera möjliga utskrifter.



(10 p)