TENTAMEN: Algoritmer och datastrukturer

Läs detta!

- *Uppgifterna är inte avsiktligt ordnade efter svårighetsgrad.*
- Börja varje uppgift på ett nytt blad.
- Ordna bladen i uppgiftsordning.
- 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 på frågorna 2-5. Motivering krävs ej. Varje korrekt svar ger två poäng. Garderingar ger noll poäng.

1. Visa implementeringsrelationer mellan ADT:er och datastrukturer genom att dra streck mellan begreppen nedan. För poäng krävs att alla relevanta streck ritas ut, men inga irrelevanta. Det bör bli nio eller tio streck, beroende på hur en av strukturerna implementeras. (*Svara på separat blad, ej i tesen.*)

oordnad mängd prioritetskö

ordnad mängd

hashtabell binär hög

fält

FIFO-kö stack

länkad lista binärt sökträd

2. Antag att en referensvariabel (pekare) kan lagras med 32 bitar, och ett ASCII-tecken med 8 bitar. Vilket genomsnittligt minnesutnyttjande får man då följande lagringsformer används för att lagra sekvenser av ASCII-tecken (1 = 100%)?
   A. fält med fältdubblering
   B. enkellänkad lista
   C. binärt träd.

   a. A – 7/8, B – 1/8, C – 1/10
   b. A – 3/4, B – 1/6, C – 1/8
   c. A – 1/2, B – 1/4, C – 1/6
   d. A – 3/4, B – 1/5, C – 1/9

3. Sorteringsalgoritmer som endast kan jämföra två element i taget har för genomsnittliga indata tidskomplexiteten
   a. O(n log n)
   b. Ω(n log n)
   c. $O(n^2)$
   d. Ω (n^2)

forts. på nästa sida ->

4. Vi söker minsta övre begränsningar för lösningarna till ekvationerna A-E. Kombinera ordouttrycken 1-5 med respektive ekvation.

A
$T(0) = 0$
$T(n) = 2T(n-1) + 1 \quad (n > 0)$

B
$T(0) = 0$
$T(n) = T(n-1) + n \quad (n > 0)$

C
$T(1) = 1$
$T(n) = 2T(n/2) + n \quad (n > 1)$

D
$T(1) = 1$
$T(n) = 2T(n/2) + 1 \quad (n > 1)$

E
$T(1) = 1$
$T(n) = T(n/2) + n \quad (n > 1)$

1: $O(\log N)$
2: $O(N)$
3: $O(N \log N)$
4: $O(N^2)$
5: $O(2^N)$

a. A-2, B-5, C-4, D-3, E-3
b. A-5, B-5, C-3, D-3, E-2
c. A-5, B-4, C-3, D-2, E-2
d. A-5, B-4, C-3, D-3, E-1

forts. på nästa sida ->

5. Metoden `countLeafs` skall räkna antalet löv i ett binärt träd. En av metoderna är korrekt implementerad, vilken?

```
public class TreeNode {
    int element;
    TreeNode left,right;
}
public static int countLeafs1(TreeNode t) {
    if ( t.left == null && t.right == null ){
        return 1;
    else
        return countLeafs1(t.left) + countLeafs1(t.right);
}
public static int countLeafs2(TreeNode t) {
    if ( t == null )
        return 0;
    else if ( t.left == null && t.right == null ){
        return 1;
    else if (t.left != null )
        return countLeafs2(t.left);
    else
        return countLeafs2(t.right);
}
public static int countLeafs3(TreeNode t) {
    if ( t == null )
        return 0;
    else if ( t.left == null && t.right == null ){
        return 1;
    else
        return countLeafs3(t.left) + countLeafs3(t.right);
}
public static int countLeafs4(TreeNode t) {
    if ( t.left == null && t.right == null ){
        return 1;
    else if (t.left != null )
        return countLeafs4(t.left);
    else if (t.right != null )
        return countLeafs4(t.right);
    else
        return countLeafs4(t.left) + countLeafs4(t.right);
}
public static int countLeafs5(TreeNode t) {
    if ( t.left == null && t.right == null ){
        return 1;
    else if (t.left != null )
        return countLeafs5(t.left);
    else if (t.right != null )
        return countLeafs5(t.right);
}
```

a. `countLeafs1`
b. `countLeafs2`
c. `countLeafs3`
d. `countLeafs4`
e. `countLeafs5`

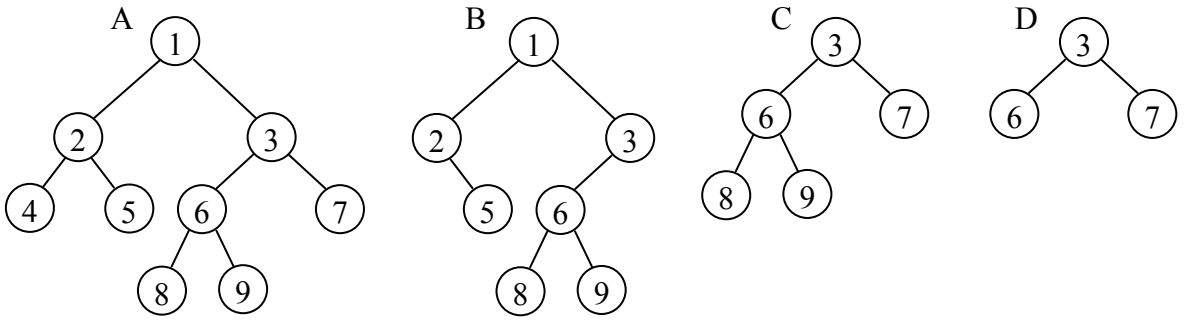
(10 p)

Uppgift 2

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

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

Här följer några sådana träd



a) Skriv en rekursiv funktion som avgör om ett träd matchar ett annat träd. Ett tomt träd matchar alla träd men inga icketomma träd matchar ett tomt träd. En nod matchar en annan nod om de har samma innehåll. För att t1 skall matcha t2 krävs att varje nod i t1 matchar en nod i motsvarande position i t2.

```
public static boolean matches(TreeNode t1,TreeNode t2)
```

Ex.

```
matches(null,A) -> true
matches(null,null) -> true
matches(A,null) -> false
matches(A,A) -> true
matches(B,A) -> true
matches(A,B) -> false
matches(C,A) -> false
matches(D,A) -> false
matches(D,C) -> true
```

(5 p)

b) Skriv en rekursiv funktion som avgör om ett träd innehåller ett annat träd. Alla träd innehåller det tomma trädet och ett tomt träd innehåller inga icketomma träd. t1 innehåller t2 om t2 matchar t1, eller något delträd i t1.

```
public static boolean contains(TreeNode t1,TreeNode t2)
```

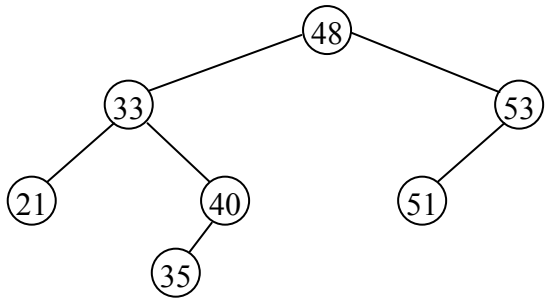
Ex.

```
contains(A,null) -> true
contains(null,null) -> true
contains(null,A) -> false
contains(A,A) -> true
contains(A,B) -> true
contains(B,A) -> false
contains(A,C) -> true
contains(A,D) -> true
contains(B,C) -> false
contains(B,D) -> false
contains(C,D) -> true
```

(5 p)

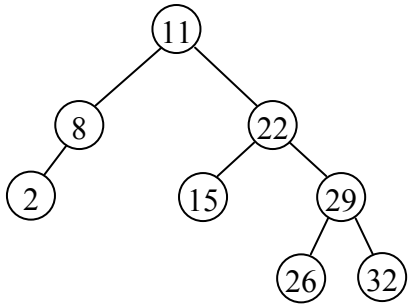
Uppgift 3

a) På vilka platser kan *ett* element sättas in i sökträdet nedan utan att bryta mot AVL-villkoret? Ange i samtliga fall inom vilket intervall detta element måste ligga.



(3 p)

b) Visa hur AVL-trädet nedan ser ut efter insättning av 30 och lämplig AVL-rotation.



(3 p)

Uppgift 4

a) Kan ett och samma binära träd med minst två noder vara både en binär hög (minhög) och ett binärt sökträd? Ge ett exempel på ett sådant träd, eller om det är omöjligt – förklara varför?

(3 p)

b) Vilket har garanterat lägst tidskomplexitet vid insättning av N element i en binär hög? Att sätta in elementen ett och ett med operationen `insert`, eller att lägga in elementen i fältet och därefter anropa `buildHeap`? Motivera!

(3 p)

c) Kommer elementen i en icketom binär hög h alltid att finnas på samma platser före och efter följande operationssekvens? Varför i så fall? Om inte, ge ett motexempel!

```
x = h.deleteMin();
h.add(x);
```

(2 p)

Uppgift 5

En idrottsförening har konstruerat ett javaprogram för att hantera sitt medlemsregister. I programmet används följande klass för att representera medlemmar (här något förenklad):

```
public class Member {
    private int id;        // medlemsnumret
    private String name;
    private String email;

    public Member(int id,String name,String email) {
        this.id = id;
        this.name = name;
        this.email = email;
    }
    public boolean equals(Object other) {
        if ( this == other )
            return true;
        if ( other instanceof Member ) {
            Member m = (Member)other;
            return id == m.id;
        }
        return false;
    }
    public int hashCode() { return id; }
}
```

Medlemsobjekten lagras i en hashtabell av samma typ som i kursboken. Kollisionshanteringen i tabellen sker med kvadratisk sondering, men klubbmedlemmarna är lite oeniga om hur man skall implementera operationen `remove`. Optimisterna tycker att tabellpositioner för borttagna element kan återanvändas för nya insättningar, medan pessimisterna menar att det är säkrast att låta dessa positioner vara spärrade för insättningar tills tabellen hashas om till dubbel storlek. Hashfunktionen definieras

```
public int hash(Member m,int tableSize) {
    return m.hashCode() % tableSize;
}
```

Antag att följande medlemsobjekt skapats:

```
Member nisse = new Member(28,"Nisse Hult","nisse@hult.se");
Member lisa  = new Member(3,"Lisa Nilsson",lisa@nilsson.no");
Member kalle = new Member(15,"Kalle Modin","kalle@modin.dk");
Member sven = new Member(2,"Sven Eklund",sven@eklund.se");
Member malin = new Member(15,"Malin Brandin","malin@brandin.fi");
```

Visa med en figur hur en initialt tom hashtabell med 13 platser ser ut efter sekvensen nedan om man tillåter insättning av nya element på platser där element tagits bort.

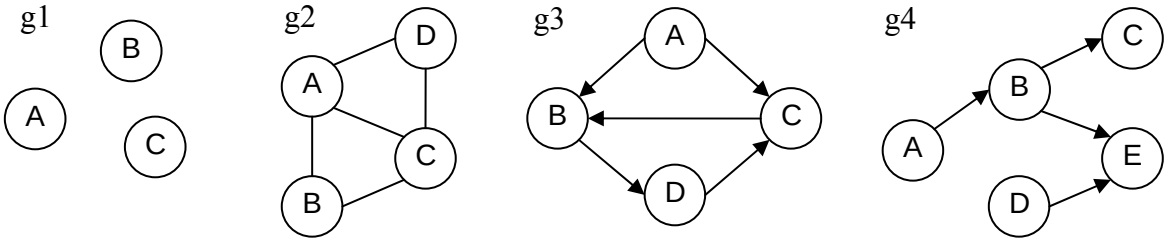
```
add(nisse); add(lisa); add(kalle); add(sven); remove(nisse); add(malin);
```

Fungerar tabellen som den skall eller kan problem uppstå? Motivera!

(5 p)

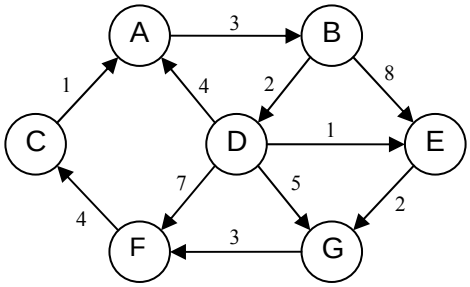
Uppgift 6

a) Ange för var och en av graferna g1 – g4 alla möjliga topologiska ordningar av noderna.



(4 p)

Betrakta grafen



b) Ange en möjlig besöksordning till noderna vid bestämning av de kortaste oviktade avstånden från D till alla övriga noder.

(2 p)

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

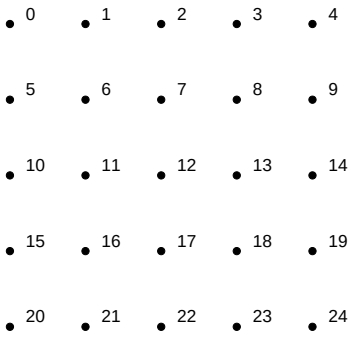
(2 p)

d) Ge ett exempel på en graf där Dijkstras algoritm ej kan användas och förklara varför.

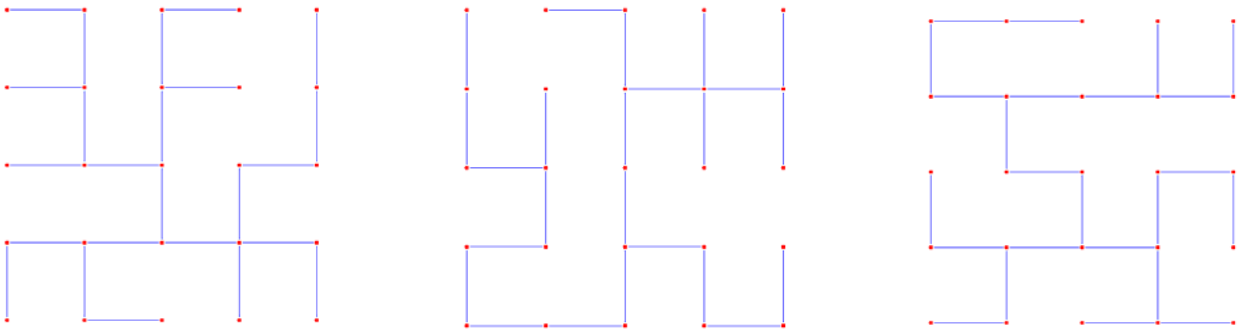
(1 p)

Uppgift 7

Ett tändstickspussel har en spelplan bestående av numrerade punkter ordnade i ett kvadratiskt rutnät. Exempel:



De horisontella och vertikala avstånden mellan två närliggande punkter motsvarar längden hos en sticka. Varje sticka är märkt med numren på de två punkter den får ansluta. Det finns en sticka för varje par av horisontella respektive vertikala grannpunkter, alltså 40 stickor ovan, eller 2N(N-1) för ett pussel av storlek N. Pusslet läggs på följande sätt: Stickorna blandas i en burk. Spelaren tar upp en sticka i taget och kan välja att antingen placera den på sin plats i spelplanen, eller kasta bort den. Pusslet är löst när alla punkter är förbundna, men det får bara finnas en väg mellan två punkter. Nedan följer några lösningar för pussel av storlek 5:



Uppgiften går ut på att implementera datastrukturer och algoritmer till ett program som kan lösa tändstickspussel. Den grafiska presentationen ingår ej i uppgiften. Stickor representeras av klassen

```
public class Stick {
    public int oneEnd,theOtherEnd;

    public Stick(int oneEnd,int theOtherEnd) {
        this.oneEnd = oneEnd;
        this.theOtherEnd = theOtherEnd;
    }
}
```

De tre stickorna längst upp till vänster i den vänstra figuren ovan motsvaras av `Stick(0,1)`, `Stick(1,6)`, samt `Stick(5,6)`.

forts.

Pusslet implementeras av klassen

```
public class SticksGame {
    private static List<Stick> getRandomSticks(int size)
    public static List<Stick> solve(int size)
}
```

a) Implementera metoden `getRandomSticks`. Parametern `size` anger pusslets storlek. Metoden skall returnera en lista med 2*`size`*(`size`-1) objekt av typen `Stick`, motsvarande alla horisontella och vertikala grannpunkter enligt föregående sida. Objekten i listan skall vara slumpmässigt ordnade.

(4 p)

b) Implementera metoden `solve`. Parametern `size` anger pusslets storlek. Metoden skall returnera en lista av stickobjekt som utgör en lösning till pusslet. Utnyttja metoden i a samt använd lämpliga datastrukturer.

(8 p)