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 ADT:n stack med fält och tillämpar fältdubblering så får båda operationerna push och pop tidskomplexiteten
   a. $O(N)$ i genomsnitt och $O(N)$ i värsta fall
   b. $O(1)$ i genomsnitt och $O(1)$ i värsta fall
   c. $O(1)$ i genomsnitt och $O(N)$ i värsta fall
   d. $O(N)$ i genomsnitt och $O(1)$ i värsta fall

2. I en algoritm som analyserar källkoden för ett Java- eller C-program för att undersöka om varje vänsterparentes har en matchande högerparentes bör vi använda en
   a. hashtabell
   b. stack
   c. FIFO-kö
   d. DisjointSets

3. Insättningssortering har tidskomplexiteten
   a. $O(n)$ i bästa fall och $O(n^2)$ i genomsnitt
   b. $O(n \cdot \log n)$ i genomsnitt och $O(n^2)$ i värsta fall
   c. $O(\log n)$ i bästa fall och $O(n^2)$ i värsta fall
   d. $O(n)$ i bästa fall och $O(n \cdot \log n)$ i genomsnitt

4. Vilken ADT används i en effektiv implementering av Dijkstras algoritm?
   a. hashmap
   b. prioritetskö
   c. sökträd
   d. lista
   e. FIFO-kö
   f. stack

5. Ange en minsta övre begränsning för kodavsnittets tidskomplexitet:

```
int n = ...; // n > 0
int k = 2;
boolean p = true;
while ( p && n/k >= k ) {
    if ( n % k == 0 ) p = false;
    if ( k == 2 ) k++;
    else k = k + 2;
}
```

   a. $O(2^n)$
   b. $O(n^2)$
   c. $O(n)$
   d. $O(n \cdot \log n)$
   e. $O(\log n)$
   f. $O(\sqrt{n})$

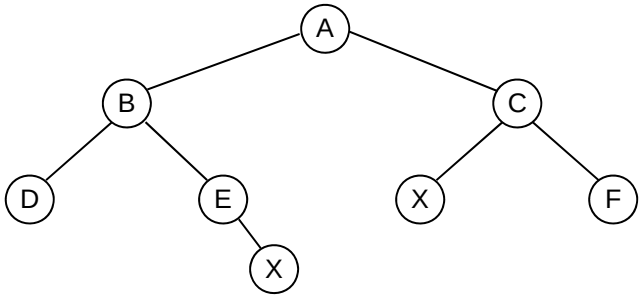
(10 p)

Uppgift 2

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

```
public class TreeNode {
    char element;
    TreeNode left,right;
}
```

Ett sätt att ange var ett visst element finns i ett träd är att ange ett spår från roten till elementet. Ett spår är en sträng av ettor och nollor, där 1 betyder höger och 0 vänster. Till exempel hittar man X i trädet nedan genom att följa spåren 011 och 10.



Skriv en rekursiv funktion som returnerar en lista av alla spår till ett sökt element i ett träd.

```
public static List<String> findPaths(TreeNode t,char x)
```

Funktionen skall returnera en lista av strängar av ettor och nollor. Det är lämpligt att skriva en hjälpfunktion som adderar ett tecken först i varje sträng i en lista av strängar.

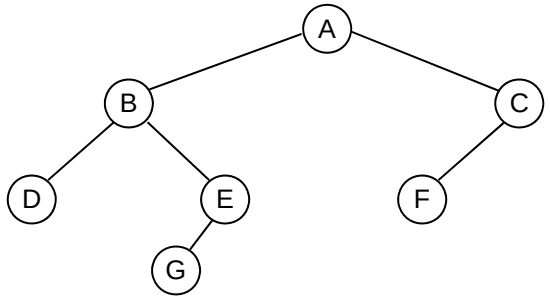
Exempel:

Anrop	Resultat
`findPaths(null,'E')`	`[]`
`findPaths(t,'A')`	`[""]`
`findPaths(t,'B')`	`["0"]`
`findPaths(t,'C')`	`["1"]`
`findPaths(t,'D')`	`["00"]`
`findPaths(t,'E')`	`["01"]`
`findPaths(t,'F')`	`["11"]`
`findPaths(t,'X')`	`["011","10"]`

(10 p)

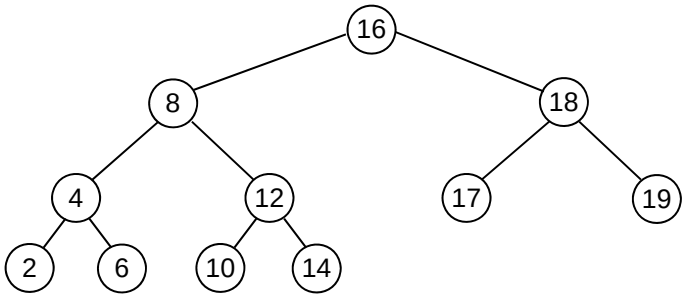
Uppgift 3

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



(3 p)

b) I AVL-trädet nedan finns åtta ställen där insättning av ett element förstör AVL-villkoret. För några av dessa ställen krävs en dubbelrotation för att återställa trädet så att AVL-villkoret åter uppfylls och för några ställen är en enkelrotation tillräcklig. Ange tal som kräver en dubbelrotation, samt för vilka en enkelrotation räcker. Du skall alltså ange åtta olika tal.



(4 p)

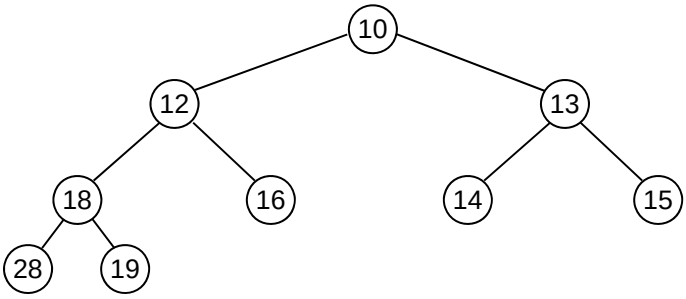
Uppgift 4

a) Rita den binära högen som ett träd:

-∞	1	7	3	9	12	5	4	20	11	14	17	8	11	127	32	35
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*

(2 p)

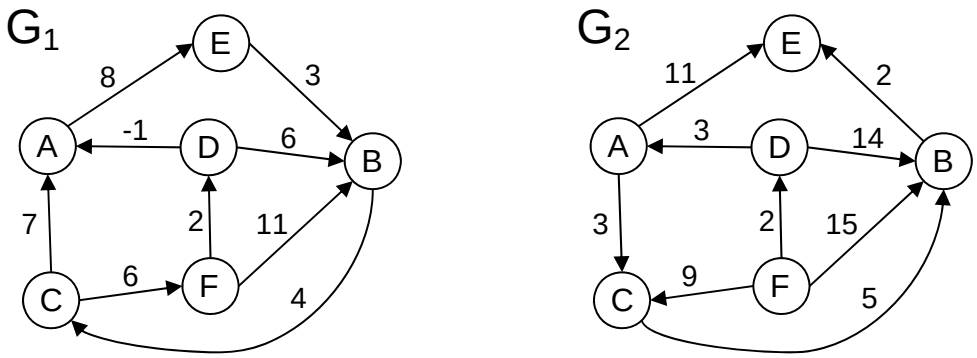
b) Rita den binära högen efter anropen `insert(11);insert(13);insert(9);`



(3 p)

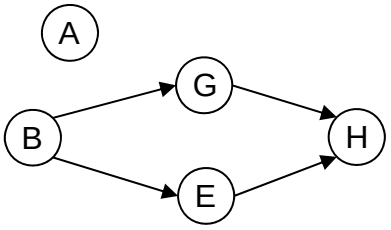
Uppgift 5

a) Beräkna samtliga kortaste viktade avstånd från nod F till alla övriga noder med Dijkstras algoritm i den av graferna nedan där algoritmen är tillämplig. Ange varför algoritmen inte kan användas i det andra fallet:



(5 p)

b) Ange alla möjliga topologiska ordningar av noderna i grafen:



(5 p)

Uppgift 6

I en viss hashtabell tillämpas kvadratisk sondering för kollisionshantering.
Hashfunktionen är *hash*(*x*) = *x* % 12.

Vad händer om insättningssekvensen

```
insert(44);insert(8);insert(32);insert(12);
```

utförs på tabellen:

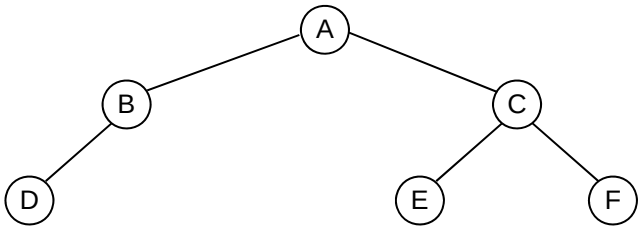
0	
1	13
2	
3	
4	40
5	
6	
7	
8	
9	57
10	
11	

Tabellstorleken skall ej ändras. Motivera ditt svar!

(8 p)

Uppgift 7

Konstruera en metod som skriver ut innehållet i noderna i ett binärt träd nivåvis. T.ex. skall utskriften för trädet nedan bli A B C D E F:



Trädnoder representeras med typen:

```
public class TreeNode {
    char element;
    TreeNode left,right;
}
```

Ett tomt träd representeras som vanligt med **`null`**. En viss datastruktur kan vara användbar i algoritmen. Metoden skall ha signaturen:

```
public static void printLevelOrder(TreeNode t);
```

(10 p)