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 misslyckas ibland med att lösa problemet?
   a. divide & conquer
   b. dynamisk programmering
   c. backtracking
   d. snål algoritm

2. Stackar och köer kan implementeras med fält eller med länkad lista. Vilket av lagringssätten har högst värstafallkomplexitet?
   a. fält
   b. länkad lista
   c. det är igen skillnad, alla operationer är $O(1)$
   d. det är igen skillnad i värstafallet, men i genomsnitt är fält sämre än listor

3. Dijkstras algoritm har tidskomplexiteten
   a. $O(|E|^2)$
   b. $O(|E|)$
   c. $O(|E|\cdot|V|^2)$
   d. $O(|E|\cdot\log|V|)$
   e. $O(|E|\cdot\log|E|)$

4. Vilken datastruktur är lämplig för att effektivt beräkna lägesmått som t.ex. median i en föränderlig datasamling
   a. länkad lista
   b. sorterat fält
   c. binärt sökträd
   d. disjoint sets

5. Ange en minsta övre begränsning för T(n) som definieras av rekurensekvationerna
   $T(1) = 1$
   $T(n) = 3T(n/3) + n$
   a. T(n) är $O(n \cdot \log_3 n)$
   b. T(n) är $O(n \cdot \log n)$
   c. T(n) är $O(n + \log_3 n)$
   d. T(n) är $O(n \cdot \log_2 n)$
   e. T(n) är $O(n + \log n)$

(10 p)

Uppgift 2

Binära träd kan representeras med typen

```
public class TreeNode<E> {
    E element;
    TreeNode<E> left,right;
}
```

Konstruera metoden

```
public static <E> LinkedList<E> getPreorder(TreeNode<E> t)
```

Som returnerar elementen i trädet `t` i en lista. Elementen i listan skall ordnas enligt preorder. Ett tomt träd (**`null`**) skall ge en tom lista (alltså inte **`null`**).

(9 p)

Uppgift 3

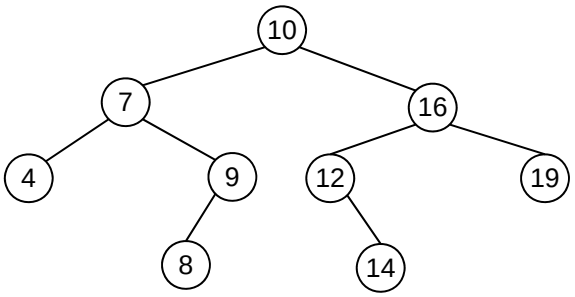
a) Ange antalet element i ett fullt binärt träd som en funktion av trädets höjd H, $H \geq 0$.

(1 p)

b) Ge ett rekursivt samband för det minimala antalet element i ett AVL-träd av höjd H, $H \geq 0$.

(2 p)

c) Rita det binära sökträdet nedan efter anropet `remove(10)`:



(3 p)

Uppgift 4

En binär hög har följande element

-∞	5	7	8	12	14	9	10	20	13	15	18
0	1	2	3	4	5	6	7	8	9	10	11

a) Rita den binära högen som träd efter sekvensen `insert(6);deleteMin();`

(3 p)

b) Rita den binära högen som träd efter sekvensen `deleteMin();insert(6);`

(3 p)

Uppgift 5

a) Varför används en prioritetskö och inte en FIFO-kö i Dijkstras algoritm? Motivera!

(2 p)

b) Vilka krav måste en graf uppfylla för att noderna skall kunna ordnas topologiskt?

(1 p)

c) Konstruera en graf med fem noder sådan att BADCE, ABCDE, BDACE, ABDCE, BACDE, BACED, ABCED är de enda möjliga topologiska ordningarna av grafens noder. *Tips:* Grafen skall ha fyra bågar.

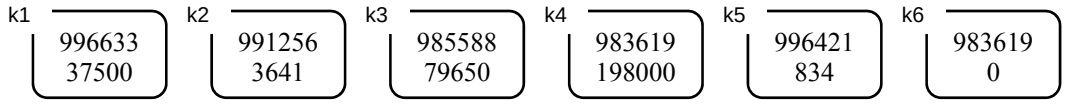
(4 p)

Uppgift 6

I en bankapplikation finns kontoobjekt av följande typ

```
public class Account {
    private String accountNo;
    private long balance;
    public Boolean equals(Object o);
    public int hashCode();
    // ... other code omitted
}
```

Metoden `equals` returnerar `true` om de jämförda kontoobjekten har likadana kontonummer. Metoden `hashCode` ger samma värde för två kontoobjekt som är lika när de jämförs med `equals`.



a) Utför sekvensen

```
insert(k1);
insert(k2);
insert(k3);
insert(k4);
remove(k2);
insert(k5);
insert(k6);
```

i en hashtabell med 13 platser och visa med en figur hur resultatet blir. Kvadratisk sondering skall tillämpas. Du kan anta att `hashCode()%13` ger följande värden: k1:8, k2:7, k3:11, k4:7, k5:10, k6:7.

(4 p)

b) Med utgångspunkt från resultatet i a, vad ger `find(k4)` för resultat? Motivera!

(2 p)

Uppgift 7

En satslogisk (boolesk) formel består av satsvariabler: A, B, C, … och logiska operatorer: & (konjunktion), | (disjunktion) samt ! (negation).
Ex.

f_1: A | !A
f_2: A & !A
f_3: A & B & !(A | C)
f_4: (A | B) & B & !C

En formel är *satisfierbar* om det finns en tilldelning av sanningsvärden till satsvariablerna så att formeln blir sann. T.ex. är f_1 satisfierbar både med tilldelningen {A=true} och med {A=false}. Däremot är inte f_2 satisfierbar eftersom en av konjunkterna blir falsk oavsett vilket sanningsvärde A har. Inte heller f_3 är satisfierbar, men tilldelningen {A=false,B=true,C=false} satisfierar f_4 och även {A=true,B=true,C=false}.

Uppgiften går ut på att konstruera en algoritm som avgör om en satslogisk formel är satisfierbar. För att avgöra problemet kan det krävas att alla möjliga sanningstilldelningar undersöks. Exempelvis kan tre variabler tilldelas sanningsvärden på åtta olika sätt.

Följande gränssnitt beskriver sanningstilldelningar och formler.

```
public interface Assignment {
    void assign(char variable,boolean value);
    boolean getValue(char variable);
}
public interface BooleanFormula {
    boolean getValue(Assignment a);
    Set<Character> getVariables();
}
```

Gränssnittet `BooleanFormula` implementeras av klasserna `Variable`, `Conjunction`, `Disjunction` samt `Negation`.

Exempel:

```
BooleanFormula a = new Variable('A');
BooleanFormula b = new Variable('B');
BooleanFormula c = new Variable('C');
BooleanFormula A_and_B = new Conjunction(a,b);
BooleanFormula A_or_B = new Disjunction(a,b);
BooleanFormula A_or_C = new Disjunction(a,c);
BooleanFormula B_and_not_C = new Conjunction(b,new Negation(c));
BooleanFormula notSatisfiable =
    new Conjunction(A_and_B,new Negation(A_or_C));
BooleanFormula isSatisfiable = new Conjunction(A_or_B,B_and_not_C);
```

a) Skriv en klass som implementerar `Assignment`. Använd lämplig datastruktur.
(2 p)

b) Skriv klassen `Variable` samt *en* av `Conjunction`, `Disjunction` eller `Negation`, välj själv vilken. Inför lämpliga instansvariabler och konstruktorer.
(6 p)

c) Skriv färdigt klassen

```
public class Satisfiability {
    public static boolean isSatisfiable(BooleanFormula f) { … }
}
```

Ange också ordokomplexiteten för metoden `isSatisfiable` som en funktion av antalet distinkta variabler i formeln.

Tips: Inför en privat hjälpmetod för att skapa sanningstilldelningar. Genom att utnyttja kvot och rest vid division med två kan bitarna i ett heltals binära representation översättas till sanningsvärden. Talet 11 skrivs t.ex. binärt som 1011, vilket motsvarar sanningsvärdena true,false,true,true. Utnyttja resultatet från a).

Exempel:

```
Satisfiability.isSatisfiable(notSatisfiable); // false
Satisfiability.isSatisfiable(isSatisfiable);  // true
Satisfiability.isSatisfiable(new Conjunction(a,
                                 new Negation(a))); // false
Satisfiability.isSatisfiable(new Disjunction(a,
                                 new Negation(a))); // true
```

(8 p)