

# Tentamen

## Datastrukturer DIT960

**Datum** Tisdag 29 maj 2012, kl 8.30–12.30  
**Lokal** V-huset  
**Kursansvarig** Peter Ljunglöf, tel. 0736–24 24 76

**OBS!**

G-gränsen är  
ändrad till minst  
4 rätta svar!

Tentamen består av sex enkla frågor (nr 1–6) och tre svåra frågor (nr 7–9).

För *Godkänd* krävs att du har svarat korrekt på minst 5 enkla frågor.

För *Väl Godkänd* krävs *dessutom* att du har svarat korrekt på minst 2 svåra frågor.

**Tillbehör** Papper, penna, färgkritor, sax, lim

En (1) fuskklapp:

- en enkelsidig A4-sida med egna anteckningar
- anteckningarna får vara handskriva eller utskrivna med skrivare
- endast ena sidan får vara använd!

*Men inga övriga anteckningar!*

**Observera** Börja varje ny uppgift på ett nytt blad

Skriv personlig kod på varje blad

Onödigt komplicerade lösningar godkänns inte

*Skriv läsligt!*



2. En heap kan lagras i ett fält. Vilket av fälten A, B eller C representerar en korrekt heap?

	0	1	2	3	4	5	6	7	8	9	10	11
A =	3	5	15	7	18	22	35	30	9	17		

	0	1	2	3	4	5	6	7	8	9	10	11
B =	3	5	18	15	7	22	35	30	9	17		

	0	1	2	3	4	5	6	7	8	9	10	11
C =	3	5	18	7	15	22	35	30	9	17		

a) Rita den korrekta heapen som ett binärt träd.

b) Tag ut det minsta elementet ur heapen, och balansera om.  
Hur ser fältet ut nu?

[illegible]

3. Antag att vi har en klass Queue som är implementerad som ett cirkulärt fält med följande privata variabler och publika metoder:

```
class ArrayQueue<E> implements Queue<E> {  
    private E[] data;  
    private int front, rear;  
    public ArrayQueue();  
    public boolean offer(E item);  
    public E poll();  
}
```

Om vi skapar en kö *q*, stoppar in fyra element och tar bort ett, så ser instansvariablerna ut såhär efteråt:

	0	1	2	3	4
data =	a	b	c	d	

front = 1

rear = 3

Antag nu att vi utför följande sekvens:

```
q.offer("e");  
q.offer("f");  
q.poll();  
q.poll();  
q.offer("g");  
q.poll();
```

Hur ser *q*:s instansvariabler ut efter detta?

	0	1	2	3	4
data =					

front =

rear =

- 
4. Här ska du definiera om de binära sökträden i Haskellkompendiet så att de implementerar en *avbildning* (mapping) från *nycklar* (keys) till *värden* (values).

Definiera följande datatyp, och funktioner för att slå upp en nyckel och för att lägga till/uppdatera ett nyckel-värde-par:

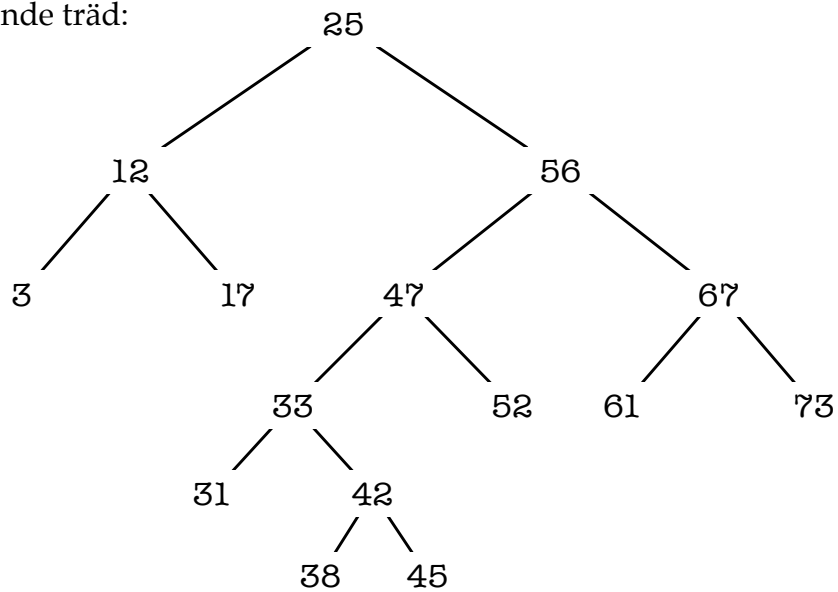
```
data BSTMap k v = ...  
  
lookup :: Ord k => k -> BSTMap k v -> Maybe v  
  
update :: Ord k => k -> v -> BSTMap k v -> BSTMap k v
```

*Tips:* Du kan inspireras av den här koden för "vanliga" binära sökträd:

```
data BST a = Empty  
           | Node a (BST a) (BST a)  
  
member :: Ord a => a -> BST a -> Bool  
  
member x Empty = False  
member x (Node a left right)  
  | x < a = member x left  
  | x > a = member x right  
  | otherwise = True  
  
insert :: Ord a => a -> BST a -> BST a  
  
insert x Empty = Node x Empty Empty  
insert x (Node a left right)  
  | x < a = Node a (insert x left) right  
  | x > a = Node a left (insert x right)  
  | otherwise = Node x left right
```

Svara på ett löst papper.

5. Givet följande träd:



- a) Färga noderna röda och svarta på ovanstående träd så att det blir ett korrekt rödsvart träd. Om du inte har färgpennor, så kan du använda fyrkanter för svarta noder och cirklar för röda.
- b) Stoppa in 40 i det rödsvarta trädet, och balansera om. Rita det slutliga trädet.

6. Utför en quicksort-partitionering av följande fält:

0	1	2	3	4	5	6	7	8
53	88	32	44	23	90	67	71	42

Visa hur det partitionerade fältet blir med nedanstående val av pivot.

Markera dessutom vilka delfält som behöver sorteras vidare i ett rekursivt anrop. (T.ex. genom att rita en klammer under varje delfält, eller genom att kryssa över de celler som *inte* behöver sorteras).

a) Pivot = första elementet

0	1	2	3	4	5	6	7	8

b) Pivot = mittersta elementet

0	1	2	3	4	5	6	7	8

c) Pivot = sista elementet

0	1	2	3	4	5	6	7	8

.....

7. (svår) Begrunda följande funktion:

```
public void f(int[] a, int m) {  
    int[] b = new int[m];  
    for (int i = 0; i < a.length; i++) {  
        int j = a[i];  
        b[j]++;  
    }  
    int i = 0;  
    for (int j = 0; j < m; j++) {  
        for (int k = 0; k < b[j]; k++) {  
            a[i] = j;  
            i++;  
        }  
    }  
}
```

a) Vad är resultatet av följande sekvens?

```
int[] a = new int[10] {5,3,7,1,0,3,5,4,3,6};  
f(a, 8);  
System.out.println(Arrays.toString(a));
```

b) Vad gör funktionen?

c) Vilken är komplexiteten för funktionen?



- 
8. (svår) Definiera en Haskell-datatyp för riktade, viktade grafer, samt följande funktioner för att skapa och använda grafer.

```
-- the type of graphs depends on the type v of vertices, and the type w of weights
type/data/newtype Graph v w = ...

-- create an empty graph
empty :: Ord v => Graph v w

-- insert an edge (from, to, weight) into a graph
insert :: Ord v => (v, v, w) -> Graph v w -> Graph v w

-- return a list of all edges in the graph
edges :: Ord v => Graph v w -> [(v, v, w)]

-- return a list of the adjacent vertices of an edge
neighbors :: Ord v => Graph v w -> v -> [v]

-- return the weight of an edge, if the edge exists
weight :: Ord v => Graph v w -> (v, v) -> Maybe w
```

Med hjälp av ovanstående funktioner kan vi t.ex. skapa en graf från en lista med bågar:

```
-- create a graph from a list of edges (from, to, weight)
graph :: Ord v => [(v, v, w)] -> Graph v w
graph edges = foldr insert empty
```

Funktionerna `edges`, `neighbors` och `weight` ska alla vara effektiva. Det betyder att `edges` och `neighbors` ska vara  $O(N \log n)$  och att `weight` ska vara  $O(\log n)$ , där  $N$  är antal element i resultatlistan, och  $n = |V|$  är antalet noder i grafen. Förklara också varför funktionerna har den komplexiteten.

Du får använda dig av Haskell's inbyggda datatyp `Data.Map`, vilken är en implementation av ett självbalanserande träd. För mer information om `Data.Map`, se det bifogade utdraget ur GHC's biblioteksdokumentation. Där står också de olika funktionernas komplexitet.

Svara på ett löst papper.

- 
9. (svår) Implementera Javaklassen `HeapTree<E>` som implementerar nedanstående delmängd av gränssnittet `Queue<E>`.

```
public interface Queue<E> {  
    // Inserts the specified element into this queue  
    // if it is possible to do so immediately without  
    // violating capacity restrictions.  
    boolean offer(E e);  
    // Retrieves, but does not remove, the head of this  
    // queue, or returns null if this queue is empty.  
    E peek();  
    // Optional: Retrieves and removes the head of this  
    // queue, or returns null if this queue is empty.  
    E poll();  
}
```

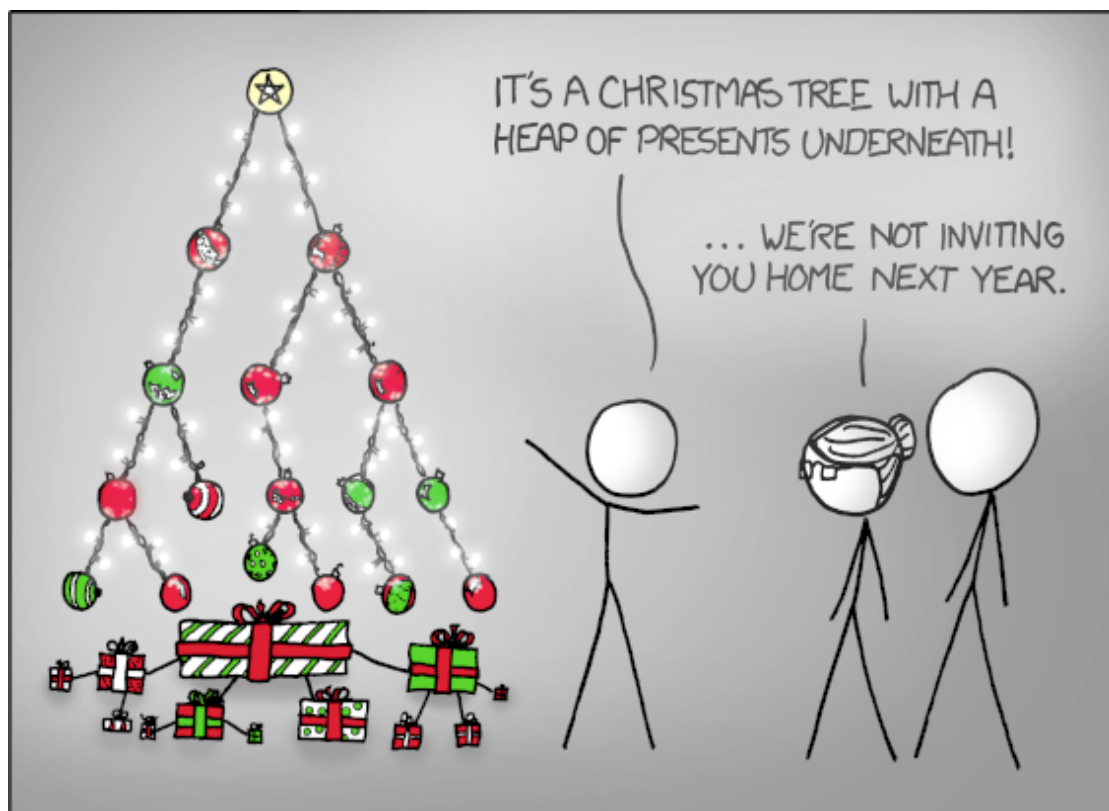
Klassen ska implementera samma datastruktur och samma algoritmer som beskrivs i Haskellkompendiet, avsnitt 6. Det avsnittet är bifogat tentan.

**OBS!** Du behöver *inte* implementera funktionen `poll()`. Däremot ska funktionen vara möjlig att implementera, vilket den är om instansvariablerna, konstrueraren, `offer()` och `poll()` är implementerade korrekt.

Glöm inte konstrueraren, som bör vara någon av dessa två:

```
// Creates a HeapTree that orders its elements  
// according to their Comparable natural ordering.  
public HeapTree();  
  
// Creates a HeapTree with the specified initial  
// capacity that orders its elements according to  
// the specified comparator.  
public HeapTree(Comparator<? super E> comparator);
```

Svara på ett löst papper.



<http://xkcd.com/835/>