Institutionen för
Datavetenskap
Chalmers, Göteborgs universitet

VT09
DIT960
18/5-09

T E N T A M E N för

Datastrukturer DV, 7.5p, DIT960
även DAT416 och DIT721

DAG : 25 maj 2009 Tid : 8.30-13.30 SAL : V

Ansvarig	:	Bror Bjerner, tel 772 10 29, 55 54 40
Resultat	:	senast den 2/6 -09
Hjälpmedel	:	Häftet med Fakta-rutor, papper och penna.
Betygsgränser GU	:	Godkänt = 28 p, Väl godkänt = 48 p
Betygsgränser CTH	:	3 = 28 p, 4 = 49 p, 5 = 50 p

OBS ! För att få betyg Godkänt eller bättre krävs minst 10 poäng på både A- och B-delen
(Om du endast tentar en del måste du lämna in senast **11.30**)

OBSERVERA NEDANSTÅENDE PUNKTER

- Börja varje ny uppgift på nytt blad.
- Skriv personlig kod på varje blad.
- Använd bara ena sidan på varje blad.
- Klumpiga, komplicerade och/eller oläsliga delar ger poängavdrag.
- **L y c k a t i l l ! ! !**

Del A

Datastrukturer på abstrakt nivå.

Uppg 1: **a)** Stoppa in 'värdena' 1, 6, 2, 3, 4 och 5 i ett binärt sökträd. Du skall stoppa in dem i **exakt** denna ordning (dvs först 1, sedan 6, sedan 2, sedan 3 osv) och utför en sökning med splay-balansering efter 5. Visa de balanseringar som utförs vid sökningen genom att visa trädet du har före, med noderna som ingår i balanseringen markerade, och efter varje rotation.

Obs! inga balanseringar vid insättningen, bara vid sökning.

(3 p)

b) Stoppa in samma noder i samma ordning i ett AVL-träd och redovisa när obalanserna uppstår och hur balanseringarna sker.

(3 p)

c) Stoppa in samma noder i samma ordning i ett rödsvart träd, där insättning sker enligt 'top-down med flip'. Markera svarta noder med en fyrkant och röda noder med en cirkel.Visa varje 'flip' och rita om trädet efter varje rotation.

(4 p)

Uppg 2: Vilken eller vilka av följande påstående är sann(a) och vilken eller vilka är falsk(a). Svara bara med Sant eller Falskt. För varje delfråga ger rätt svar 1 p och fel svar ger -0.6 p. Poängen för hela uppgiften kan däremot inte bli negativ.

a) Insättning i en kö och en prioritetskö har samma komplexitet.

b) I en enkellänkad lista tar det konstant tid att ta bort sista elementet, om vi har en referens (pekare) till det sista elementet i listan.

c) I en hashtabell tar det alltid konstant tid att hämta värdet till en nyckel.

d) Den bästa algoritmen som kan sortera n stycken tal som har värden mellan 1 och 1000 har bästa värstafallskomplexiteten $\mathbf{O}(\, n * {}^2log\, n\,)$

e) Alla träd kan implementeras som ett binärt träd.

f) Urvalssortering (selection sort) komplexitet beror inte på hur elementen är ordnade i fältet.

(6 p)

Uppg 3:

80	90	110	40	30	60	100	50	120	10	20	70

Visa hur 'quick sort' fungerar genom att sortera fältet ovan med den förenklade metoden, som gavs på en föreläsning. Pivotelementet skall vara **det mittersta elementet** i delsegmentet som skall sorteras. Markera med bågar vilka element som byter plats och med en liten pil pivotelementen. Rita ett nytt fält efter varje pivottering. Det är givetvis tillåtet (och lämpligt) att utföra pivotteringarna av delfälten 'samtidigt' på varje nivå. Notera att du inte skall byta sorteringsmetod utan göra 'quick sort' ända ner i botten. Vidare skall sorteringen göras i fältet.

(6 p)

Uppg 4: **a)** Givet en riktad viktad graf, en startnod och en maximal totalvikt. Ge en algoritm i någon form av pseudokod, som ger alla noder som kan nås från startnoden utan att totalvikten av vägen överskrider den givna totala maximala totalvikten.

(7 p)

b) Vad är värstafallskomplexiteten för din algoritm ?

(1 p)

Del B

Datastrukturer på implementeringsnivå.

Uppg 5: Antag att vi vill implementera en samling (collection) som en dubbelllänkad cikulär struktur där variabeln `last` refererar till det sista elementet i den länkade strukturen. vi startar med följande deklarationer:

```
public class DLinkedCollection<E>
       extends AbstractCollection<E>{

   protected Entry  last;

   protected class Entry {
      Entry next, previous;
      E     elem;
      public Entry( Entry n, Entry p, E e ) {
        next = n; previous = p; elem = e;
      }
}
```

Din uppgift är att implementera nedanstående 3 metoder. Du får inte lägga till någon instansvariabel. Du får heller inte använda dig av iteratorn.

a) Skriv en metod `public boolean add(E elem)`, som lägger till ett element till samlingen.

(2 p)

b) Skriv en metod `public int size()`, som ger antalet element i samlingen.

(2 p)

c) Skriv en metod `public boolean remove( E elem )`, som tar bort en instans av `elem` i samlingen.

(3 p)

d) Vad är komplexiteten för var och en av dessa 3 metoder, det krävs att alla 3 är rätt för att få poäng.

(1 p)

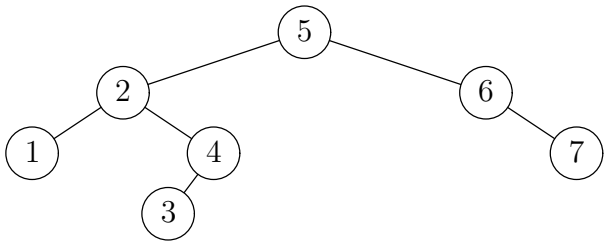
Uppg 6: Du skall implementera Prim's algoritm som ger ett minsta sammanhängande träd (minimum spanning tree) för en oriktad viktad graf, som implementerats med en matris (se nedan). Resultatet skall vara en iterator av bågar som deklarerats i grafen. Vikterna är alla positiva heltal och noderna är numrerade från 0 till antalet noder-1. (Jag har förenklat grafen en del för att det inte skall bli för många oväsentliga detaljer.)

Metoden ges ett nodnummer att starta med och generar iteratorn för minsta sammanhängade träd för grafen eller för den isolerade delgraf som noden med nodnumret ingår i.

(14 p)

```
public class UndirectedGraph {

 public static class Edge implements Comparable<Edge>{
   private int node1, node2, weight;
   public Edge( int n1, int n2, int w ) {
     node1  = n1; node2 = n2; weight = w;
   }
   public int compareTo( Edge e ) {
     return weight - e.weight;
   }
 } // Edge

 private int[][] graph;

 public UndirectedGraph( int noOfNodes ) {
   graph = new int[noOfNodes][noOfNodes];
   for( int i = 0; i < noOfNodes; i++ )
      Arrays.fill( graph[i], -1 )
 } // UndirectedGraph

 public void addEdge( int n1, int n2, int w ) {
   graph[n1][n2] = w; graph[n2][n1] = w;
 } // addEdge

 public Iterator<Edge>
        minSpanningTree( int startNode ) {
    ******     Här skriver du din kod      *****
 }
```

Uppg 7: Givet ett binärt träd vill du traversera trädet i **preorder**. Till exempel, för trädet:



skall elementen ges i följande ordning: 5, 2, 1, 4, 3, 6 och 7

a)

Detta alternativ är för DIT960
TDA416 och DIT721 skall endast göra alternativ b

Givet är en modul `BinTree`:

```
module BinTree (
  BT,        -- type of binary trees
  emptyTree, -- BT a
  isEmpty,   -- BT a -> Bool
  leftSub,   -- BT a -> BT a
  rightSub,  -- BT a -> BT a
  rootVal,   -- BT a -> a
  insert,    -- Eq a => a -> BT a -> BT a
  remove,    -- Eq a => a -> BT a -> BT a
  inorder,   -- BT a -> [a]
  contains   -- Eq a => a -> BT -> Bool
)
```

Du skall nu definiera funktionen

```
preorder :: BT a -> [a]
```

som ger elementen i trädet i preorder. För full poäng krävs att funktionen är av $\mathbf{O}(n)$

Notera: du skall **inte** göra denna funktion inuti modulen !!

(Du har även tillgång till köer och/eller stackar, se sista sidan.)

(8 p)

b)

Detta alternativ är för TDA416 och DIT721
DIT960 skall endast göra alternativ a

Du skall deklarera en iterator för trädet som ger elementen i preorder. För full poäng krävs att alla metoderna (även konstruktorn) är av $\mathbf{O}(1)$

Om `next` anropas och det ej finns fler element, så skall `NoSuchElementException` kastas.

Du skall ej implementera `remove`-metoden. (Men den måste deklareras).

(8 p)

```
public class BinTree<E> implements Iterable<E> {

    protected TreeNode<E> root;

    public static class TreeNode<E> {
       public E           element;
       public TreeNode<E> left, right;
       public TreeNode(E element) {
          this( null, element, null );
       } // constructor TreeNode

       public TreeNode( TreeNode<E> left,
                        E           element,
                       TreeNode<E>  right   ) {
           ... } // constructor TreeNode
   }  // class TreeNode
   ...
} // class BinTree
```

```
module Stack (
 Stack,      -- type of stacks
 emptyStack, -- Stack a
 isEmpty,    -- Stack a -> Bool
 push,       --  a -> Stack a -> Stack a
 pop,        -- Stack a -> Stack a
 top         -- Stack a -> a
)

module Queue (
  Queue,      -- type of queues
  isEmpty,    -- Queue a -> Bool
  emptyQueue, -- Queue a
  enqueue,    -- a -> Queue a -> Queue a
  dequeue,    -- Queue a -> Queue a
  front       -- Queue a -> a
)
```