Institutionen för
Datavetenskap
Chalmers, Göteborgs universitet

VT10
DIT960
24/5-10

T E N T A M E N för

Datastrukturer DV, 7.5p, DIT960

DAG : 31 maj 2010 Tid : 8.30-13.30 SAL : V

Ansvarig	:	Bror Bjerner, tel 772 10 29, 55 54 40
Resultat	:	senast den 7/6 -10
Hjälpmedel	:	Häftet med Fakta-rutor, papper och penna.
Betygsgränser GU	:	Godkänt = 28 p, Väl godkänt = 48 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' 6, 5, 4, 1, 3 och 2 i ett binärt sökträd. Du skall stoppa in dem i **exakt** denna ordning (dvs först 6, sedan 5, sedan 4, sedan 1 osv) och utför en sökning med splay-balansering efter 2. 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) Värstafallskomplexiteten för sökning i ett splay-träd är $\mathbf{O}(\,^2log\, n\,)$

b) Om man tar bort färgerna från ett rödsvart träd får man ett AVL-träd utan höjdinformationen.

c) Binär sökning är en effektiv metod för att söka i en sorterad länkad lista.

d) I en sammanhängande oriktad graf kan en kortaste väg mellan två noder alltid hittas med hjälp av Dijkstra's metod.

e) I en dubbellänkad cirkulär lista tar det konstant tid att ta bort det sista eller det första elementet, om vi endast har en referens (pekare) till det sista elementet i listan.

f) Insättningssorterings (insertion 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 högersta 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.

(7 p)

Uppg 4: **a)** Givet en riktad viktad graf och en lista av noder som ingår i grafen (eller listan innehåller en dalmängd av de noder som ingår i grafen). Ge en algoritm i någon form av pseudokod, som ger den mest 'centrala' noden av de noder som ingår i den givna listan, dvs den nod som ligger närmast alla andra noder som ingår i den givna listan. Vidare gäller att 'shortest path' mellan två noder och 'minimum spanning tree' redan finns i grafen.

(Du skall sedan implementera denna algoritm i del B.)

(5 p)

b) Vad är värstafallskomplexiteten för din algoritm ?

(2 p)

Del B

Datastrukturer på implementeringsnivå.

Uppg 5: Du skall nu implementera algoritmen från **uppgift 4**, dvs fylla i metoden `centerNode` nedan.

(5 p)

```
public abstract class Edge {
  public final int from, to;
  public abstract double  weight();
} // Edge

public class DirectedGraph<E extends Edge> {

 private  List<E>[] neighbours;

 public DirectedGraph( int noOfNodes ) { .. }
 } // DirectedGraph

 public void addEdge( E e ) { .. }

 public Iterator<E>
        shortestPath( int startNode,
                      int slutNode  ) {
   /*** Redan implementerat ***/
 } // shortestPath

 public Iterator<E>
        minimumSpanningTree() {
   /*** Redan implementerat ***/
 } // minimumSpanningTree

 public int centerNode( List<Integer> li ) {
    /** HÄR SKRIVER DU DIN KOD  **/
 } // centerNode

} // DirectedGraph
```

Uppg 6: Givet en modul för binära sökträd:

```
module BinSearchTree (
  BST,       -- type of binary search trees
  emptyTree, -- BST a
  isEmpty,   -- BST a -> Bool
  leftSub,   -- BST a -> BST a
  rightSub,  -- BST a -> BST a
  rootVal,   -- BST a -> a
  insert,    -- Ord a => a -> BST a -> BST a
  remove,    -- Ord a => a -> BST a -> BST a
  inorder,   -- BST a -> [a]
  get        -- Ord a => a -> BST a -> Maybe a
) where
```

skall du definiera två funktioner.

a) `contains :: Ord a => a -> BST a -> Bool`
som avgör om det finns ett element i trädet som är lika stort som det givna.

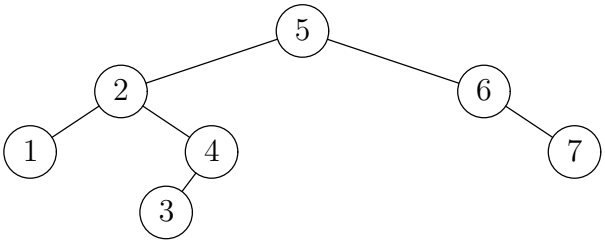
(3 p)

b) `postOrder :: BST a -> [a]`
som ger en lista av elementen i trädet i postorder, T.ex. för noderna i trädet i uppgift 7 blir listan:
`[1,3,4,2,7,6,5]`
För poäng krävs att funktionen är av $\mathbf{O}(n)$

(6 p)

Obs: du skall *inte* göra dessa funktioner inuti modulen !!

Uppg 7: Givet ett binärt träd, **obs:** detta är *inte* ett *binärt sökträd*, vill du traversera trädet i **preorder**. Till exempel, för trädet:



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

Du skall deklarera en iterator för trädet som ger elementen i preorder. För full poäng krävs att alla metoderna, inklusive konstruktorn, utom `remove` är av $\mathbf{O}(1)$. `remove` får högst ha en komplexitet av $\mathbf{O}(h)$, där h är trädets höjd.

Alla metoderna skall uppfylla beskrivningen som ges av API:n för `Iterator<E>` (se längst bak). Vidare måste `remove` implementeras så att ordningen för elementen ej förändras från den ursprungliga ordningen. Metoden `remove( E e )` är *inte* implementerad, dvs du måste implementera `remove` i iteratorn fullt ut. (Korrekt `remove` ger 10 p, övriga delar 6 p.)

(16 p)

```
public class BinTreeWithParent<E>
       implements Iterable<E> {
  protected TreeNode<E> root;
  public static class TreeNode<E> {
    public E           element;
    public TreeNode<E> left, right, parent;
    public TreeNode( E           element,
                     TreeNode<E> left,
                     TreeNode<E> right
                     TreeNode<E> parent   ) {
           ... } // constructor TreeNode
   }  // class TreeNode
 public Iterator<E> iterator()
 { return new BinTreeIterator() }

 /*** Här definierar du iteratorklassen  ***/
} // class BinTreeWithParent
```