Institutionen för
Datavetenskap
Chalmers, Göteborgs universitet

VT09
TDA416, DIT721
9/3-09

# T E N T A M E N för

## Datastrukturer och algoritmer IT, 7.5p, TDA416
## Datastrukturer och algoritmer, 7.5 p, DIT721.

DAG : 13 mars 2009 Tid : 8.30-13.30 SAL : Hörsalarna

| | | |
|---|---|---|
| Ansvarig | : | Bror Bjerner, tel 772 10 29, 55 54 40 |
| Resultat | : | Senast 27/3 -09 |
| Hjälpmedel | : | Häftet med Fakta-rutor, papper och penna. |
| Betygsgränser CTH | : | 3 = 28 p, 4 = 39 p, 5 = 50 p |
| Betygsgränser GU | : | Godkänt = 28 p, Väl godkänt = 48 p |

**OBS ! För att få betyg 3/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 personnummer 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, 2, 6, 5, 4 och 3 i ett binärt sökträd. Du skall stoppa in dem i **exakt** denna ordning (dvs först 1, sedan 2, sedan 6, sedan 5 osv) och utför en sökning med splay-balansering efter 3. 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.

**(4 p)**

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

**(4 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 rotering.

**(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, fel svar ger -0.6 p och inget svar ger 0 p. Poängen för hela uppgiften kan däremot inte bli negativ.

**a)** Urvalssorterings (Selection Sort) komplexitet beror på hur elementen är ordnade i fältet.

**b)** Binär sökning är en effektiv metod att söka i en sorterad länkad lista.

**c)** Man kan använda en enkellänkad cirkulär lista för att implemtera en kö effektivt.

**d)** Att lägga in ett element i en hashtabell är alltid av $\mathbf{O}(1)$

**e)** Värstafallskomplexiteten för quicksort är $\mathbf{O}(n * {}^2log\ n)$, där $n$ är antalet element som skall sorteras.

**f)** I en sammanhängande riktad graf kan en kortaste väg mellan två noder alltid hittas med hjälp av Dijkstra's algoritm.

**g)** Prims algoritm fungerar för alla nätverk (network).

**(7 p)**

**Uppg 3:**

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

Visa hur merge sort fungerar genom att sortera fältet ovan. Du skall göra merge ända ner i botten och inte byta till någon annan sorteringsmetod vid någon viss storlek. Du får välja antingen rekursivt eller iterativt tankesätt, bara du är konsekvent. Givetvis får du utföra de olika merge-operationerna parallellt i varje steg.

**(5 p)**

**Uppg 4:** Heapar brukar lagras i fält. Avgör vilket av följande två fält, **a** eller **b**, som är en heap.

| **a:** | 1 | 3 | 7 | 4 | 5 | 8 | 6 | 9 |
|---|---|---|---|---|---|---|---|---|
| *index* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |

eller

| **b:** | 1 | 3 | 6 | 4 | 5 | 8 | 7 | 9 |
|---|---|---|---|---|---|---|---|---|
| *index* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |

**a)** Rita upp heapen som ett binärt träd. **(1 p)**

**b)** Motivera varför det andra fältet ej är en heap. **(1 p)**

**c)** Stoppa in 2 i heapen och rita upp fältet efter instoppningen. **(2 p)**

**d)** Ta sedan bort det minsta elementet och rita upp fältet efter borttagandet. **(2 p)**

# Del B

Datastrukturer på implementeringsnivå.

**Uppg 5:** **a)** Uppgiften är att skriva ett program som utgående från en given nod besöker alla noder **enligt bredden först** i en sammanhängande riktad graf. Resultatet skall vara en lista av nodnummer som visar i vilken ordning noderna besöks. Vid jämbördiga val, kan du välja vilken ordning du vill. Du skall använda den graf som vi hade i laboration 3, se nedan.

**(8 p)**

**b)** Vad är värstafalls-komplexiteten för metoden med avseende på antalet bågar $\mathbf{0}(|E|)$ och/eller antalet noder $\mathbf{0}(|V|)$, motivera.

**(1 p)**

```
public abstract class Edge {

  public final int from, to;

  public Edge( int from, int to ) {
    this.from   = from;
    this.to     = to;
  }  // constructor Edge

  public abstract double  weight();

}  // class Edge
```

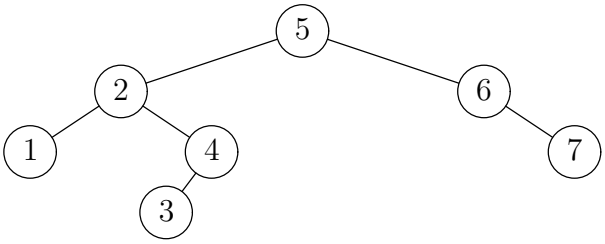
```
public class DirectedGraph<E extends Edge> {

  private List<E>[] neighbours;

  public DirectedGraph(int noOfNodes) {
    neighbours = (List<E>[]) new List[noOfNodes];
    for ( int i = 0; i < noOfNodes ; i++ )
      neighbours[i] = new LinkedList<E>();
  } // constructor DirectedGraph

  public void addEdge(E e) {
    try { neighbours[e.from].add(e); }
    catch (IndexOutOfBoundsException iobe)
      { throw new IndexOutOfBoundsException(
          "Wrong Node Number in Graph: "
          + e.from); }
  } // addEdge

  public List<Integer> traversBF(int startnode){

   ***   Här skriver du din kod ***

  } // traversBF

} // class DirectedGraph
```

**Uppg 6:** Givet ett binärt träd (*Obs: detta är ej ett sökträd!*), se nästa sida, skall du

**a)** deklarera en metod som ger dig antalet löv, dvs antalet noder som ej har några söner. Använd dig av rekursion !

**(4 p)**

**b)** deklarera en intern iterator i Java, se bilaga, som ger en lista med noderna i **preorder**. Till exempel, för trädet:




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

Du skall i denna version **inte** göra en implementering av `remove`.

Alla metoder och konstruktorn måste vara av komplexiteten **0**( 1 )

I övrigt skall metoderna bete sig enligt specifikationen i bilagan.

**(8 p)**

**c)** du skall nu deklarera en subklass till iteratorn i föregående uppgift. I subklassen skall du implementera metoden `remove`. `remove`-metoden får **inte** ändra på den ordning elementen kommer ut ur iteratorn och inte heller tappa något element.

I övrigt skall metoden(erna) bete sig enligt specifikationen i bilagan.

**(9 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   ) {
            this.left    = left;
            this.element = element;
            this.right   = right;
       } // constructor TreeNode

   }  // class TreeNode

  public Iterator<E> iterator() {
      return new BinTreeIterator();
  } // iterator

   public int noOfLeafs() {
     ...  deluppgift a

   private class BinTreeIterator
           implements Iterator<E> {
      ... deluppgift b

   // implementering av subklassen
      ...deluppgift c

} // class BinTree
```