Institutionen för
Datavetenskap
Göteborgs universitet

VT11
DIT960
29/5 -11

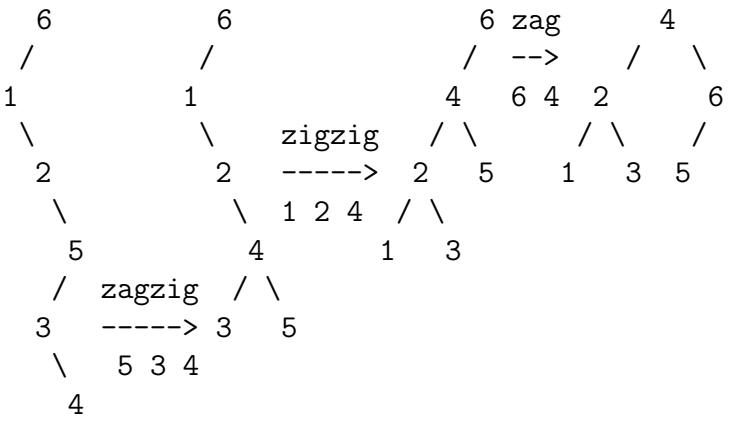
# Lösningsförslag till tentamen för Datastrukturer DV

DAG : 30 maj 2011

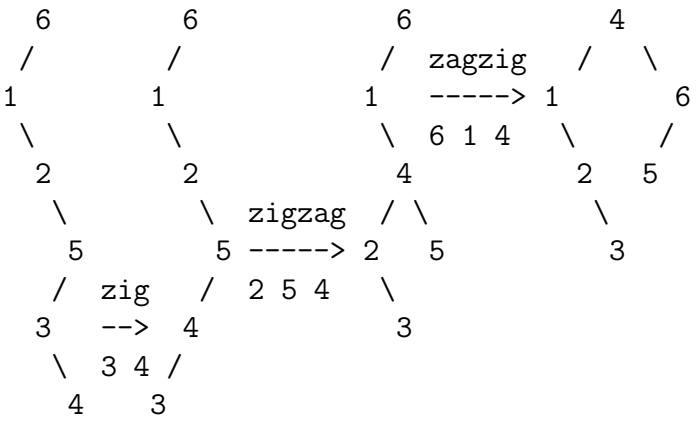
# Del A

Datastrukturer på abstrakt nivå.

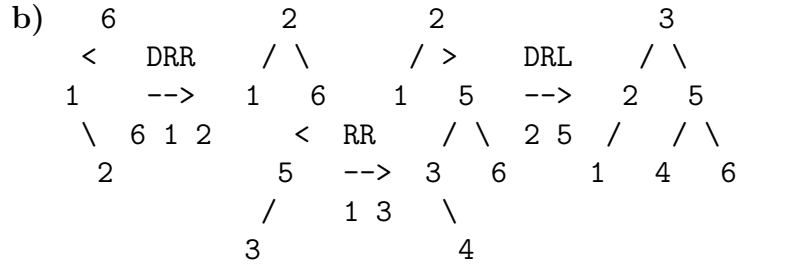
**Uppg 1:** **a)**



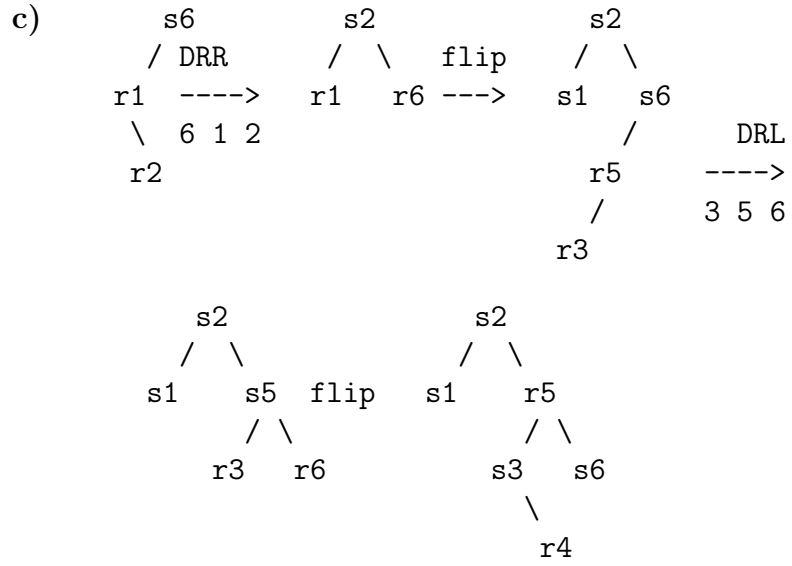

alternativt



**b)**


6
< DRR
1 -->
\ 6 1 2
2
2
/ \
1 6
< RR
5 -->
/
3
2
/ >
1 5
/ \
3 6
1 3
\
4
3
DRL
/ \
2 5
-->
2 5
/ \
1 4 6

**c)**


s6
/ DRR
r1 ----> r1 r6 --->
\ 6 1 2
r2
s2
/ \ flip
s2
/ \
s1 s6
/ DRL
r5 ---->
/ 3 5 6
r3
s2
/ \
s1 s5 flip
/ \
r3 r6
s2
/ \
s1 r5
/ \
s3 s6
\
r4

**Uppg 2:** **a)** Falskt, eftersom ett splayträd kan vara hur skevt som helst.

**b)** Sant, en heap är ju nivåbalanserad. (Fast det är tyvärr inte riktigt sant, som någon upptäckte. Om $n$ är $k^i$ så är höjden $1 + \lceil {}^2log\, n \rceil$. Därför får även de tre som svarat falskt 1 p).

**c)** Falskt, endast om fältet är sorterat.

**d)** Sant, (däremot inte i en riktad graf.)

**e)** Falskt, det räcker med en pekare till första elementet.

**f)** Falskt, looparna går alltid fullt ut vid urvalssortering.

**Uppg 3:** På raden under elementen anges med ett p vilka element som är pivot-element, och index anger med vilket index som swap har gjorts.

Maximala rekursionsdjupet blir 6, där den sista ej ritas ut eftersom inget händer,

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 80 | 90 | 110 | 40 | 30 | 60 | 100 | 50 | 120 | 10 | 20 | 70 |
| 10 | 9 | 7 | | | | 11 | 2 | | 1 | 0 | p6 |
| 20 | 10 | 50 | 40 | 30 | 60 | 70 | 110 | 120 | 90 | 80 | 100 |
| | | | | | p5 | - | 10 | 9 | 8,11 | 7 | p9 |
| 20 | 10 | 50 | 40 | 30 | 60 | 70 | 80 | 90 | 100 | 110 | 120 |
| | | 4 | | p2 | - | - | | p8 | - | | p11 |
| 20 | 10 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 |
| 1 | p0 | - | | p4 | - | - | | - | - | | - |
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 |
| - | | - | | - | - | - | - | - | - | - | - |

Notera att det är onödigt att flytta pivot-elementet längst till höger, eftersom det redan 'är ur vägen'. Däremot har jag inte dragit av poäng för de som flyttade pivot-telementet längst till vänster i delsegmentet, även om det är onödigt.

**Uppg 4:** **a)** Följande fält av listor fås:

```
under index   fås listan
    0            []
    1            [20]
    2            []
    3            []
    4            [16, 5]
    5            [44, 88, 11]
    6            [94, 39]
    7            [12, 23]
    8            []
    9            [13]
   10            []
```

**b)** Följande sluttabell fås:

| 11 | 39 | 20 | 5 | 16 | 44 | 88 | 12 | 23 | 13 | 94 |
|---|---|---|---|---|---|---|---|---|---|---|

Då elementet 11 skall sättas in så har vi följande tabell:

| -1 | -1 | -1 | -1 | -1 | 44 | 88 | 12 | 23 | 13 | 94 |
|---|---|---|---|---|---|---|---|---|---|---|

där -1 anger att det är en ledig plats.

Hash-koden för 11 är 5, och eftersom positioner med index som är 5 eller större alla är upptagna blir första lediga plats 0 !

**Uppg 5:** **a)** Som vanligt vid bredden först använder jag en kö `nextNode` för att få noderna i rätt ordning. För att sortera grannarna till en använder jag en prioritetskö. som sorteras via comparatorn `toComp`, sedan läggs noderna in i kön `nextNode`. Jag behöver fältet `visited` för att få loopen att terminera, så därför gör det inget om en nod råkar förekomma flera gånger i kön `nextNode`.

```
private class toComp
        implements Comparator<E> {
   public int compare( E e1, E e2 )
     { return e1.to - e2.to; }
} // toComp

public List<Integer> breadthFirst( int startNode ) {

  boolean[]       visited  = new boolean[ neighbours.length ];
  List<Integer>   res      = new LinkedList<Integer>();
  Queues<Integer> nextNode = new LinkedQueue<Integer>();
  PriorityQueues<E> sortNeighbours =
    new PriQueHeapComparator<E>( new toComp() );

  nextNode.enqueue( startNode );
  while( ! nextNode.isEmpty() ) {
    int next = nextNode.dequeue();
    if ( ! visited[ next ] ) {
      res.add( next );
      visited[ next ] = true;
      for( E e : neighbours[ next ] )
      sortNeighbours.add( e );
      while ( ! sortNeighbours.isEmpty() ) {
        int newToNode = sortNeighbours.removeMin().to;
        if ( ! visited[ newToNode ] )
          nextNode.enqueue( newToNode );
      }
    }
  }
  return res;
} // breadthFirst
```

**b)** Varje båge behandlas en gång och läggs in i och plockas ut ur prioritetskön högst en gång, dvs vi får värsta komplexiteten till $\mathbf{O}(\,|E| * {}^2log\,|E|\,)$.

Om parallella bågar ej förekommer, så blir det högst $\mathbf{O}(\,|E| * {}^2log\,|V|\,)$, där $|E|$ är antalet bågar och $|V|$ antal noder.

Ett litet testexempel: (ingick ej i uppgiften)

```
public static void main(String[] args ) {

  DirectedGraph<IntEdge> g = new DirectedGraph<IntEdge>(7);
  g.addEdge( new IntEdge(1,2,1));
  g.addEdge( new IntEdge(2,1,1));
  g.addEdge( new IntEdge(1,4,1));
  g.addEdge( new IntEdge(4,1,1));
  g.addEdge( new IntEdge(1,6,1));
  g.addEdge( new IntEdge(6,1,1));
  g.addEdge( new IntEdge(3,2,1));
  g.addEdge( new IntEdge(2,3,1));
  g.addEdge( new IntEdge(3,4,1));
  g.addEdge( new IntEdge(4,3,1));
  g.addEdge( new IntEdge(3,6,1));
  g.addEdge( new IntEdge(6,3,1));
  g.addEdge( new IntEdge(5,2,1));
  g.addEdge( new IntEdge(2,5,1));
  g.addEdge( new IntEdge(5,4,1));
  g.addEdge( new IntEdge(4,5,1));
  g.addEdge( new IntEdge(5,6,1));
  g.addEdge( new IntEdge(6,5,1));

  System.out.println( g.breadthFirst( 1 ));
}

-----------------------------------

sed javac DirectedGraph.java
Note: DirectedGraph.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
sed: java DirectedGraph
[1, 2, 4, 6, 3, 5]
```

**Uppg 6:**

```
import Queue
import Stack

reverseQueue :: Queue a -> Queue a
reverseQueue que = sToQ ( qToS que )
  where
   sToQ s | Stack.isEmpty s = emptyQueue
          | otherwise       = enqueue (top s) $
                                        sToQ $ pop s
   qToS q | Queue.isEmpty q = emptyStack
          | otherwise       = push (front q)  $
                                     qToS $ dequeue q

-----------------------------------------------------
*Main> enqueue 4 emptyQueue
([4],[])
*Main> enqueue 1 $ enqueue 2 $ enqueue 3 it
([4],[1,2,3])
*Main> reverseQueue it
([1],[4,3,2])
```

**Uppg 7:**

```
private class BinTreeIterator
        implements Iterator<E>{

    TreeNode<E>         lastNext  = null;
    Stacks<TreeNode<E>> nextOnTop
              = new LinkedStack<TreeNode<E>>();

    public BinTreeIterator() {
       if ( root != null )
           pushLeftChains( root );
    } // constructor

    private void pushLeftChains(TreeNode<E> t ) {
       while ( t != null ) {
          nextOnTop.push( t );
          if (t.left == null)
             t = t.right;
          else
             t = t.left;
       }
    }

    public boolean hasNext() {
       return ! nextOnTop.isEmpty();
    }

    public E next() { // The stack throws right
      // kind of exception

      TreeNode<E> t = nextOnTop.top();

      if ( t.right != lastNext )
         pushLeftChains( t.right );
      lastNext = nextOnTop.pop();
      return lastNext.element;
    }

    public void remove() {
       throw new UnsupportedOperationException();
    }
} // class BinTreeIterator
```

Ett litet testprogram, ingick ej i uppgiften. För att lättare kunna göra detta, det saknas ju en add-metod, gjorde jag om nodetypen till:

```
protected static class TreeNode<E>
```

Jag bygger upp exemplet i tentan och skriver ut elementen via iteratorn.

```
public static void main( String[] args ) {

   BinTree<Integer> bt = new BinTree<Integer>();
   bt.root =
   new TreeNode<Integer>(
       5,
       new TreeNode<Integer>(
           2,
           new TreeNode<Integer>(1,null,null ),
           new TreeNode<Integer>(
               4,
               new TreeNode<Integer>(3,null,null),
               null
           )
       ),
       new TreeNode<Integer>(
           6,
           null,
           new TreeNode<Integer>(7,null,null)
       )
    );

   for ( Integer n : bt )
      System.out.print( n + " ");
   System.out.println();

}
```

```
-----------------------------------------------

sed:bjerner$ javac BinTree.java
sed:bjerner$ java BinTree
1 3 4 2 7 6 5
```