Utkast løsningsforslag Inf 1020 H06

Oppgave 2

```
Count check( RedBlack t){
        Count c = new Count();
        if( t==null ) return c;
        Count left = check( t.getVsub());
        Count right = check( t.getHsub());
        c.valid = ( left.valid && right.valid &&
                                left.black == right.black &&
                                !( t.red() && left.lastRed ) &&
                                !( t.red() && right.lastRed ) );
        c.lastRed = t.red();
        c.black = max( left.black, right.black );
        if( !t.red()) c.black++;
        return c;
}
```

Oppgave 3

Finn først minste spenntre. Finn deretter avstanden til nabonoden som ligger lengst unna.

Oppgave 5

Anta at vi skal sette inn en kant *fra* => *til*.

Ide: Hvis *til* ligger etter *fra* i den topologiske ordningen pos, er det bare å legge til kanten: grafen forblir asyklisk.

I motsatt fall må vi finne ut om det går en vei *til* ~> *fra* i grafen. Vi kan da gå dybde-først med start i *til* og søke oss mot *fra*, men stoppe når søket kommer til noder som ligger etter *fra* i den topologiske ordningen pos.

Hvis det ikke finnes en slik vei, kan vi legge til kanten. Vi må endre pos slik at *fra* kommer før *til*. Algoritme:

1. Fjern *til*-noden fra pos og alle nodene som (a) vi kan nå med start i *til* og (b) som ligger foran *fra* i pos.
2. Legg nodene tilbake i pos etter fra, og i den innebyrdes rekkefølge som de lå i før. Dette har vi orden på gjennom nodenes nr-variabel.

I løsningen under har jeg implementert dette. Koden er litt flikkete fordi nr-feltet i nodene må oppdateres etter innsetting (pkt 2 over). *NB: I dybde-først-søket i koden har jeg glemt en boolean array som holder orden på hvilke noder som har blitt besøkt i søket!*

Den beste løsningen tror jeg vil være at vi samtidig med dybde-først søket legger nodene inn i en heap ordnet etter pos direkte. Da har vi enten en sykel eller en heap som inneholder de nodene som må få endret posisjon i pos.

Kode:

```
import java.util.*;

class InkGrafBygger2 {

   List<Node> pos = new ArrayList<Node>();
   Graf g;


   InkGrafBygger2( Graf g) {
      this.g = g;

      for( int i=0; i<g.size(); i++ ){
         pos.add( i, g.node(i) );
         pos.get(i).nr = i;
      }
   }

   boolean sti( Node fra, Node til ){
      if( fra.nr == til.nr ) return true;
      if( fra.nr > til.nr ) return false;
      for( Node n : fra.etterf )
         if( sti(n,til) ) return true;
      return false;
   }


   boolean adderKant( int idFra, int idTil ){
      return adderKant( g.node(idFra), g.node(idTil) );
   }


  public String toString(){
      String s = "";
      int i=0;
      for( Node n : pos )
         s += "[pos" + (i++) + "] " + n + "\n";
      return s;
```

```
    }

    Heap<Node> byggHeap( Heap<Node> h, Node fra, Node til ){
        if( fra.nr > til.nr ){
            h.insert( til );
            for( Node n : til.etterf )
                h = byggHeap( h, fra, n );
        }
        return h;
    }
// MERK: IKKE LEGG INN I HEAPEN MER ENN EN GANG: MÅ BRUKE boolean
array for å holde orden på besøkte noder.

    boolean adderKant( Node fra, Node til ){
        if( sti(til,fra) ) return false;
        g.adderKant(fra, til);
        if( fra.nr > til.nr ){
            int slutt = fra.nr;
            Heap<Node> heap = byggHeap( new Heap<Node>(), fra, til ) ; // komparator på
n.nr
            int start = slutt - heap.size();
            List<Node> temp = new ArrayList<Node>();
            int i = 0;
            while( !heap.isEmpty() ){
                pos.remove( heap.findMin().nr - i );
                temp.add( i++, heap.findMin() );
                heap.deleteMin();
            }
            for( i=start; i<slutt; i++ ){
                pos.add(i, temp.get(i-start) );
                pos.get(i).nr = i;
            }
        }
        return true;
    }


}
```