Algoritmer og datastrukturer
Eksamen – 24.02.2010

Eksamenstid: 5 timer

Hjelpemidler: Alle trykte og skrevne + håndholdt kalkulator som ikke kommuniserer.

Faglærer: Ulf Uttersrud

Råd og tips: Bruk ikke for lang tid på et punkt i en oppgave hvis du ikke får det til innen rimelig tid. Gå isteden videre til neste punkt. Hvis du i et senere punkt får bruk for det du skulle ha laget i et tidligere punkt, kan du fritt bruke resultatet som om det var løst og at løsningen virker slik som krevd i oppgaven. Prøv alle punktene. Det er ikke lurt å la noen punkter stå helt blanke. Til og med det å demonstrere i en eller annen form at du har forstått hva det spørres etter og/eller at du har en idé om hvordan det kunne løses, er bedre enn ingenting. Det er heller ikke slik at et senere punkt i en oppgave nødvendigvis er vanskeligere enn et tidlig punkt. **Alle de 10 bokstavpunktene teller likt!**

Hvis du skulle ha bruk for en datastruktur fra *java.util* eller fra kompendiet, kan du fritt bruke det uten å måtte kode det selv. Men du bør kommentere at du gjør det.

Oppgave 1

1A. Klassen **`SKomparator`** (**se vedlegget**) er laget for å sammenligne tegnstrenger, dvs. instanser av klassen **`String`**.

1. Hvilken verdi vil variabelen **k** få i følgende kodebit? Gi forklaring!!

```
SKomparator c = new SKomparator();
int k = c.compare("Per", "Kari");
```

2. Hva blir utskriften fra følgende kodebit der String-tabellen **navn** sorteres ved å bruke en instans av klassen **`SKomparator`** sammen med den vanlige sorteringsmetoden fra klassen **`Arrays`** i **`java.util`**? Gi forklaring!!

```
String[] navn = {"Kari","Jasmin","Per","Ali","Nils","Liv"};
Arrays.sort(navn,new SKomparator());
for (String s : navn) System.out.print(s + " ");
```

1B. En melding inneholder bokstavene fra *A* til *H* med en bestemt frekvensfordeling. Hvis Huffmans algoritme brukes på denne fordelingen, får vi et Huffmantre og treet gir oss flg. bitkoder: *A* = 011, *B* = 010, *C* = 1000, *D* = 110, *E* = 00, *F* = 101, *G* = 1001, *H* = 111. Tegn det Huffmantreet som gir disse bitkodene. Skriv hver av de åtte bokstavene ved siden av tilhørende node.

1C. Sett inn verdiene 10, 15, 5, 12, 7, 3, 1, 6, 2, 4 i oppgitt rekkefølge i et på forhånd tomt 2-3-4 tre. 1) Tegn treet etter at 12 er satt inn, 2) tegn det etter at 3 er satt inn og 3) tegn det når alle verdiene er satt inn.

1D. Lag metoden **`public static void omorganiser(char[] c)`**. Det skal tas som gitt at parametertabellen **`c`** kun inneholder bokstavene (tegnene) **`'U'`**, **`'L'`** eller **`'F'`** (bokstavene i faglærers fornavn). Det kan være alt fra ingen til mange av hver av de tre bokstavene. Metoden **`omorganiser`** skal omorganisere tabellen **`c`** slik at U-ene kommer først, så L-ene og til slutt F-ene. Se flg. programeksempel:

```
char[] bokstaver = {'L','F','U','F','U','U','L','F','L','L','U','U'};
omorganiser(bokstaver);
for (char bokstav : bokstaver) System.out.print(bokstav + " ");

// Utskrift: U U U U U L L L L F F F
```

Oppgave 2

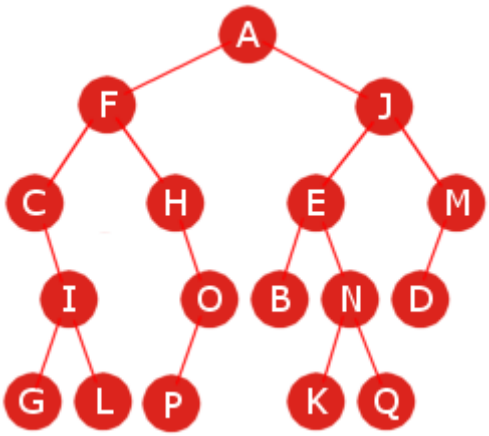
En toveiskø er en kø der en både kan ta ut og legge inn verdier i begge ender. Her skal vi lage en toveiskø ved hjelp en enveis pekerkjede som kun har en hodepeker:

I vedlegget er det satt opp et «skjelett» for klassen **`EnkelToveiskø`**. Det skal **ikke** legges inn flere instansvariabler eller statiske variabler i klassen **`EnkelToveiskø`** eller i den indre **`Node`**-klassen. En slik klasse skal normalt ha mange metoder, men her nøyer vi oss med de som er satt opp.

2A. Lag metodene **`public void leggInnFørst(T verdi)`** og **`public T taUtFørst()`**. Den første skal legge en ny node med oppgitt verdi som **`verdi`** først i pekerkjeden. Den andre skal fjerne den første noden og returnere nodens verdi. Hvis pekerkjeden er tom, skal det kastes et unntak.

2B. Lag metodene **`public void leggInnSist(T verdi)`** og **`public T taUtSist()`**. Den første skal legge en ny node med oppgitt verdi som **`verdi`** sist/bakerst i pekerkjeden. Den andre skal fjerne den siste/bakerste noden og returnere nodens verdi. Hvis pekerkjeden er tom, skal det kastes et unntak.

Oppgave 3



Figur 1: Et binærtre

3A. *Figur* 1 til venstre viser et generelt binærtre der nodeverdiene er bokstaver. I et binærtre har hver node en posisjon. Rotnoden har posisjon 1. Videre gjelder at hvis en node har posisjon k, vil et eventuelt venstre barn ha posisjon $2k$ og et eventuelt høyre barn ha posisjon $2k + 1$. Se f.eks. *Delkapittel* 5.1 i kompendiet.

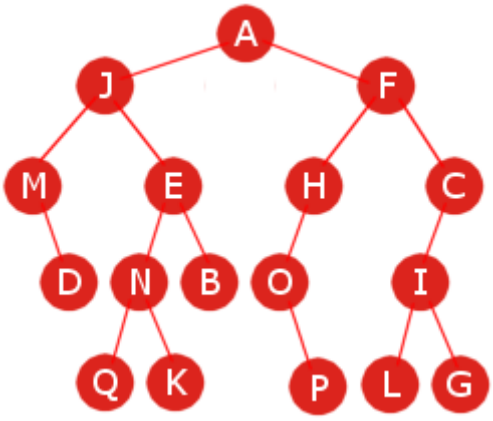
1. Lag en kopi av treet i *Figur* 1 i din besvarelse og skriv ved siden av hver node posisjonen til noden.

2. Legg inn to nye noder: Den første skal ha verdien R og posisjon 39 og den andre verdien S og posisjon 52. Skriv så treets verdier i inorden og i nivåorden.

I vedlegget er det satt opp et «skjelett» for den generiske klassen **`BinTre`**. Det står for et generelt binærtre slik som beskrevet i *Delkapittel* 5.1 i kompendiet. Den indre nodeklassen har imidlertid fått **`int posisjon`** som ekstra instansvariabel. For hver node skal den inneholde nodens posisjon. Metoden **`public void leggInn(T verdi, int posisjon)`** som er ferdigkodet, sørger for at den får rett verdi. Det skal **ikke** legges inn flere instansvariabler eller statiske variabler i **`BinTre`**-klassen eller i **`Node`**-klassen. Noen av metodene i «skjelettet» er ferdigkodet. Klassen skal ikke ha flere offentlige metoder enn de som allerede er satt opp.

3B. 1. Tegn et **komplett** binærtre som inneholder de 10 bokstavene fra *A* til *J* som verdier i nivåorden. Dvs. at rotnoden skal inneholde *A*, rotnodens venstre barn *B*, rotnodens høyre barn *C*, osv. Skriv nodeposisjonen ved siden av hver node.

2. Lag metoden **`public static <T> BinTre<T> komplett(T[] a)`**. Den skal returnere et **`BinTre`** som er komplett og som inneholder verdiene fra parametertabellen **`a`** i nivåorden. Hvis for eksempel den generiske typen **`T`** er lik **`Character`** og vi bruker tabellen **`Character[] a = {'A','B','C','D','E','F','G','H','I','J'}`**, skal treet bli som i 1.



Figur 2: Speilvendig av treet i Figur 1

3C. Binærtreet til venstre i *Figur* 2 er speilvendt i forhold til binærtreet i *Figur* 1. Et binærtre speilvendes ved at for hver node som har to barn, bytter barna side. Hvis en node har kun ett barn, flyttes det barnet til motsatt side. Treet i *Figur* 2 viser resultatet når treet *Figur* 1 speilvendes.

I vår datastruktur har nodene også en variabel med navn **`posisjon`**. Hvis treet speilvendes, vil nodene få nye posisjoner. Dermed må denne variabelen oppdateres. I *Figur* 1 har f.eks. noden med verdi *Q* posisjonen 27, men i det speilvendte treet i *Figur* 2 får den posisjonen 20.

Lag metoden **`public void speilvend()`** (se vedlegget). Den skal speilvende treet. Du bestemmer selv om du vil bruke rekursjon eller ikke, og om du vil lage hjelpemetoder.

3D. I klassen **`BinTre`** er den indre iteratorklassen **`InordenIterator`** ferdigkodet (se vedlegget). Den er imidlertid avhengig av den private hjelpemetoden i **`BinTre`** med navn **`private Node<T> nesteInorden(Node<T> p)`** (se vedlegget). Lag den. Metoden skal returnere den noden som kommer etter *p* i **inorden** eller **`null`** hvis *p* er den siste i inorden. Det kan tas som gitt at *p* ikke er **`null`** og at **`posisjon`** har korrekt verdi i hver node i treet.

Vedlegg - Algoritmer og datastrukturer

```
///// OPPGAVE 1A //////////////////////

public class SKomparator implements Comparator<String>
{
  public int compare(String a, String b)
  {
    if (a.length() < b.length()) return -1;
    else if (a.length() > b.length()) return 1;
    else if (a.compareTo(b) < 0) return -1;
```

```
    else if (a.compareTo(b) > 0) return 1;
    else return 0;
  }
}

///// OPPGAVE 1D //////////////////////

public static void omorganiser(char[] c)
{
  // Kode mangler. Skal kodes!
}

///// OPPGAVE 2 //////////////////////

public class EnkelToveiskø<T>
{
  private final static class Node<T>
  {
    private T verdi;
    private Node<T> neste;

    private Node(T verdi, Node<T> neste)
    {
      this.verdi = verdi;
      this.neste = neste;
    }
  }

  private Node<T> hode;
  private int antall;

  public EnkelToveiskø()
  {
    hode = null; antall = 0;
  }

  public void leggInnFørst(T verdi)
  {
    // Kode mangler. Skal kodes!
  }

  public T taUtFørst()
  {
    // Kode mangler. Skal kodes!
  }

  public void leggInnSist(T verdi)
  {
    // Kode mangler. Skal kodes!
  }

  public T taUtSist()
  {
    // Kode mangler. Skal kodes!
  }
} // slutt på EnkelToveiskø
```

```
///// OPPGAVE 3 ////////////////////

public class BinTre<T> implements Iterable<T>
{
  private final static class Node<T>
  {
    private T verdi;
    private Node<T> venstre, høyre;
    private int posisjon ;

    private Node(T verdi, int posisjon)
    {
      this.verdi = verdi;
      this.posisjon = posisjon;
    }

    private Node(Node<T> v, Node<T> h, T verdi, int posisjon)
    {
      this.verdi = verdi;
      this.posisjon = posisjon;
      venstre = v; høyre = h;
    }
  }

  private Node<T> rot;
  private int antall;

  public BinTre()
  {
    rot = null; antall = 0;
  }

  public void leggInn(T verdi, int posisjon)
  {
    if (posisjon < 1) throw new
      IllegalArgumentException("Posisjon(" + posisjon + ") < 1!");

    Node<T> p = rot, q = null;               // hjelpevariabler
    int i = 1;                               // hjelpevariabel

    String s = Integer.toBinaryString(posisjon);

    while (p != null && i < s.length())
    {
      q = p;
      p = s.charAt(i) == '0' ? p.venstre : p.høyre;
      i++;
    }

    if (i < s.length() || p != null) throw new
      IllegalArgumentException("Posisjon(" + posisjon + ") er ulovlig!");

    p = new Node<T>(verdi,posisjon);  // en ny node

    if (q == null)  rot = p;          // tomt tre
```

```
    else if (s.charAt(i-1) == '0')    // til venstre for q
      q.venstre = p;
    else                              // til høyre for q
      q.høyre = p;
    antall++;  // en ny verdi i treet
  }

  public static <T> BinTre<T> komplett(T[] a)
  {
    // Kode mangler. Skal kodes!
  }

  public void speilvend()
  {
    // Kode mangler. Skal kodes!
  }

  private Node<T> nesteInorden(Node<T> p)
  {
    // Kode mangler. Skal kodes!
  }

  private final class InordenIterator implements Iterator<T>
  {
    private Node<T> p;

    private InordenIterator()
    {
      p = rot;
      if (p != null)
      {
        while (p.venstre != null) p = p.venstre;
      }
    }

    public boolean hasNext() { return p != null; }

    public T next()
    {
      T temp = p.verdi;
      p = nesteInorden(p);
      return temp;
    }

    public void remove()
    {
      throw new UnsupportedOperationException();
    }
  }  // Slutt på class InordenIterator

  public Iterator<T> iterator()
  {
    return new InordenIterator();
  }

}  // Slutt på class BinTre
```