Algoritmer og datastrukturer
Eksamen – 29.11.2011

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. Metoden `public static int min(char[] a)` står i vedlegget. Hva blir verdien til variabelen `m` når flg. kodebit utføres? Gi en begrunnelse for svaret ditt.

```
char[] a = {'D','H','B','C','G','J','A','E','I','F'};
int m = Tabell.min(a);
```

1B. En liste har de metodene som er satt opp i grensnittet `Liste` (se vedlegget). I flg. kodebit brukes en `DobbeltLenketListe` (den klassen som ble laget i oblig 2). Hva blir utskriften? Begrunn svaret! Bruk tegninger eller tekst.

```
Liste<Character> liste = new DobbeltLenketListe<>();
liste.leggInn('A');
liste.leggInn('B');
liste.leggInn(1,'C');
liste.leggInn(1,'D');
liste.fjern(2);
liste.leggInn(0,'E');
liste.fjern(1);
System.out.println(liste);
```

1C. Lag metoden `public static <T> void snu(Liste<T> liste)`. Den skal snu rekkefølgen i en liste. Det er ikke kjent hvilken listeimplementasjon parameteren `liste`hører til. Det kan være `TabellListe`, `DobbeltLenketListe`eller noe annet. Det betyr at i kodingen er det kun metodene i grensesnittet `Liste`(se vedlegget) som kan brukes mot listen. Det tas som gitt at de virker som beskrevet. Hvor effektiv blir metoden hvis den brukes på en `TabellListe`? Hvor effektiv blir den hvis den brukes på en `DobbeltLenketListe`? Flg. eksempel viser hvordan metoden skal virke:

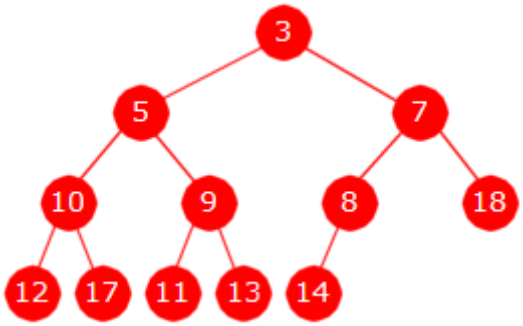
```
Liste<Integer> a = new TabellListe<>();
a.leggInn(1); a.leggInn(2); a.leggInn(3);

Liste<Character> b = new DobbeltLenketListe<>();
b.leggInn('A'); b.leggInn('B'); b.leggInn('C');

System.out.println(a + "  " + b);
snu(a); snu(b);
System.out.println(a + "  " + b);

// Utskrift:
// [1, 2, 3]  [A, B, C]
// [3, 2, 1]  [C, B, A]
```

1D. I *Figur* 1 under er det satt opp en minimumsheap:



Figur 1 : En minimumsheap

Sett først inn verdien 6 i minimumsheapen i *Figur* 1. Tegn treet! Sett så inn verdien 15 i det treet du nå har. Tegn treet!. Ta så ut den minste fra minimumsheapen. Tegn treet. Avslutt med igjen å ta ut den minste. Tegn treet!

Oppgave 2

I denne oppgaven skal vi lage klassen **`EnkelPrioritetskø`** ved å bruke en vanlig kø som intern datastruktur. Se vedlegget. Den interne køen skal holdes sortert slik at minst verdi ligger først. Dermed er det lett å finne den minste verdien. Men det krever at køen holdes sortert. Det skal ikke legges inn noen instansvariabler eller ekstra hjelpestrukturer i klassen. Flg. eksempel viser hvordan det skal virke:

```
Character[] a = {'D','H','B','C','G','J','A','E','I','F'};
Comparator<Character> c = Komparator.naturlig();
Kø<Character> kø = new TabellKø<>();

PrioritetsKø<Character> prkø = new EnkelPrioritetskø<>(kø,c);

for (char d : a) prkø.leggInn(d);
while (!prkø.tom()) System.out.print(prkø.taUt() + " ");

// Utskrift: A B C D E F G H I J
```

2A. Lag kode for metodene **`public T kikk()`** og **`public T taUt()`**i klassen **`EnkelPrioritetskø`**. Se vedlegget. Det skal i begge kastes en **`NoSuchElementException`** hvis det er tomt. Den første metoden skal returnere den minste verdien uten at den fjernes. Den andre skal returnere (og fjerne fra køen) den minste verdien. Se beskrivelsen i innledningen til *Oppgave* 2.

2B. Lag kode for metodene **`void leggInn(T verdi)`**. Den skal legge **`verdi`** inn på rett sortert plass i den interne køen.

Oppgave 3

3A. Gitt følgende heltall: 12, 16, 9, 2, 19, 7, 5, 10, 13, 17, 8, 6, 14. Legg dem inn, i den oppgitte rekkefølgen, i et på forhånd tomt binært søketre av vanlig type. Tegn treet. Skriv opp treets verdier i postorden. En bladnode i et binærtre er en node som ikke har barn. Hvor mange bladnoder er det i treet du tegnet? Skriv opp bladnodenes verdier i sortert rekkefølge.

Vedlegget inneholder et «skjelett» for klassen **`SBinTre`** - et binært søketre av vanlig type. Den private klassen **`Node`** har tre instansvariabler - nodens verdi og pekere til venstre og høyre barn. Det skal ikke legges inn flere variabler i denne klassen. Klassen **`SBinTre`** har også tre instansvariabler - en rotpeker, antall og en komparator. Det skal ikke legges inn flere variabler i klassen. Noen av metodene er ferdigkodet. Du bestemmer selv om du vil lage private hjelpemetoder.

3B. Lag kode for metoden **`public T førsteBladverdi()`**. Den skal returnere verdien til første bladnode (den av dem som har minst verdi). Hvis treet er tomt, returneres **`null`**.

3C. Lag kode for metoden **`public int antallBladnoder()`**. Den skal returnere antallet bladnoder i treet. Et tomt tre har ingen noder og dermed ingen (dvs. 0) bladnoder.

3D. Lag det som mangler i den private klassen **`BladnodeIterator`** (se vedlegget) for at den skal fungere som en iterator. Iteratoren skal «besøke» bladnodene (i inorden). Kall på **`next()`** skal derfor kun gi bladnodeverdier. Se flg. eksempel:

```
int[] a = {12, 16, 9, 2, 19, 7, 5, 10, 13, 17, 8, 6, 14};
SBinTre<Integer> tre = new SBinTre<>(Komparator.<Integer>naturlig());
for (int k : a) tre.leggInn(k);  // bygger opp treet

for (Iterator<Integer> i = tre.bladiterator(); i.hasNext(); )
{
  System.out.print(i.next() + " ");
}
// Utskrift: 6 8 10 14 17
```

Vedlegg - Eksamen i Algoritmer og datastrukturer

```
public static int min(char[] a)
{
  int m = 0;
  char minverdi = a[0];

  for (int i = 1; i < a.length; i++)
  {
    if (a[i] < minverdi)
    {
      minverdi = a[i];
      m = i;
    }
  }
  return m;
}
```

```java
public interface Liste<T> extends Beholder<T>
{
  public boolean leggInn(T verdi);           // Ny verdi bakerst
  public void leggInn(int indeks, T verdi);  // Ny verdi på indeks
  public boolean inneholder(T verdi);        // Er verdi i listen?
  public T hent(int indeks);                 // Hent verdi på indeks
  public int indeksTil(T verdi);             // Hvor ligger verdi?
  public T oppdater(int indeks, T verdi);    // Oppdater verdi på indeks
  public boolean fjern(T verdi);             // Fjern verdi
  public T fjern(int indeks);                // Fjern verdi på indeks
  public int antall();                       // Antallet i listen
  public boolean tom();                      // Er listen tom?
  public void nullstill();                   // Nullstilles (tømmes)
  public Iterator<T> iterator();             // En iterator
}

public interface Kø<T>        // eng: Queue
{
  public void leggInn(T t);   // legger inn bakerst
  public T kikk();            // ser på det som er først
  public T taUt();            // tar ut det som er først
  public int antall();        // antall i køen
  public boolean tom();       // er køen tom?
  public void nullstill();    // tømmer køen
}

public class EnkelPrioritetskø<T> implements PrioritetsKø<T>
{
  private Kø<T> kø;
  private Comparator<? super T> c;

  public EnkelPrioritetskø(Kø<T> kø, Comparator<? super T> c)
  {
    this.kø = kø; this.c = c;
  }

  public void leggInn(T verdi)
  {
    // kode mangler - skal lages
  }

  public T kikk()
  {
    // kode mangler - skal lages
  }

  public T taUt()
  {
    // kode mangler - skal lages
  }

  public int antall()
  {
    return kø.antall();
  }

  public boolean tom()
  {
    return kø.tom();
  }
```

```
  public void nullstill()
  {
    kø.nullstill();
  }
}


public class SBinTre<T>  // et standard binært søketre
{
  private static final class Node<T>      // en indre nodeklasse
  {
    private T verdi;                      // nodens verdi
    private Node<T> venstre, høyre;       // venstre og høyre barn


    private Node(T verdi) // konstruktør
    {
      this.verdi = verdi;
    }
  } // class Node

  private Node<T> rot;                          // peker til rotnoden
  private int antall;                           // antall noder
  private final Comparator<? super T> comp;     // komparator

  public SBinTre(Comparator<? super T> c)       // konstruktør
  {
    rot = null; antall = 0; comp = c;
  }

  public int antall()
  {
    return antall;
  }

  public boolean tom()
  {
    return antall == 0;
  }

  public void leggInn(T verdi)
  {
    if (verdi == null)
      throw new NullPointerException("Nullverdier er ulovlig!");

    Node<T> p = rot, q = null;
    int cmp = 0;

    while (p != null)
    {
      q = p;
      cmp = comp.compare(verdi, p.verdi);
      if (cmp < 0) p = p.venstre;
      else  p = p.høyre;
    }

    p = new Node<>(verdi);

    if (q == null) rot = p;
```

```
      else if (cmp < 0) q.venstre = p;
      else  q.høyre = p;

      antall++;
    }

    public T førsteBladverdi()
    {
      // kode mangler - skal lages
    }

    public int antallBladnoder()
    {
      // kode mangler - skal lages
    }

    private class BladnodeIterator implements Iterator<T>
    {
      // eventulle variabler, eventuelle hjelpestrukturer
      // og eventuelle hjelpemetoder skal inn her

      private BladnodeIterator()
      {
        // kode mangler - skal lages
      }

      public boolean hasNext()
      {
        // kode mangler - skal lages
      }

      public T next()
      {
        // kode mangler - skal lages
      }

      public void remove()
      {
        throw new UnsupportedOperationException();
      }
    }

    public Iterator<T> bladiterator()
    {
      return new BladnodeIterator();
    }
  } // class SBinTre
```