Algoritmer og datastrukturer
Eksamen – 30.11.2010

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. Det er mulig å finne det største tallet i en samling heltall ved å gjennomføre en utslagsturnering. Gjør dette for tallene 6, 5, 8, 7, 9, 4, 9, 5, 4, 10, 5 og 8. Tegn turneringstreet. Hvilke tall ble «slått ut» av «vinneren» i løpet av turneringen?

1B. En kø har de metodene som er satt opp i grensnittet `Kø` (se vedlegget). Hva blir utskriften etter at flg. kodebit er utført? Gi en begrunnelse for svaret ditt!

```
Kø<Integer> kø = new TabellKø<Integer>();
kø.leggInn(3); kø.leggInn(1); kø.leggInn(4); kø.leggInn(2);

for (int i = 0; i < 10; i++) kø.leggInn(kø.taUt());
while (!kø.tom()) System.out.print(kø.taUt() + " ");
```

1C. Alle klasser arver instansmetoden `toString`. Hvis den ikke kodes i klassen, vil den versjonen som ligger i basisklassen `Object` gjelde. Den returnerer kun navnet på klassen og en heksadesimal kode. Her skal vi lage en metode som skal kunne gi oss innholdet i køen som en tegnstreng selv om køens instansmetode `toString` ikke er kodet, dvs. vi skal lage metoden `public static <T> String toString(Kø<T> kø)`. Den skal returnere en tegnstreng som inneholder elementene i køen med komma og mellomrom mellom hvert element. Etter et kall på metoden skal køen inneholde de samme elementene og i samme rekkefølge som før kallet. For å få til dette må køen gjennomgås på en eller annen måte. Det er et mål å bruke så lite med ekstra «resurser» (tabeller og andre datastrukturer) som mulig. Husk at det er helt ukjent hvilken klasse som parameteren `kø` hører til, men den har de metodene som står i grenesnittet `Kø`. Instansmetoden `toString` kan ikke benyttes siden vi ikke vet om den er kodet i denne

klassen. Flg. programbit viser hvordan metoden skal virke:

```
Kø<Integer> kø = new TabellKø<Integer>();
System.out.println(toString(kø));   // Utskrift: []

kø.leggInn(3);
System.out.println(toString(kø));   // Utskrift: [3]

kø.leggInn(1);
System.out.println(toString(kø));   // Utskrift: [3, 1]

kø.leggInn(4);
System.out.println(toString(kø));   // Utskrift: [3, 1, 4]
```

Oppgave 2

2A. Lag metoden `public static int[] utenDuplikater(int[] a)`. Den skal returnere en tabell som inneholder det samme som `a`, men i sortert rekkefølge og uten duplikater. Flg. programbit viser hvordan den skal virke:

```
int[] a = {};
int[] b = {5,2,3,8,2,5,6};
int[] c = {3,3,3,3,3,3};
int[] d = {3,2,1};

String x = Arrays.toString(utenDuplikater(a));
String y = Arrays.toString(utenDuplikater(b));
String z = Arrays.toString(utenDuplikater(c));
String u = Arrays.toString(utenDuplikater(d));

System.out.println(x + " " + y + " " + z + " " + u);

// Utskrift: [] [2, 3, 5, 6, 8] [3] [1, 2, 3]
```

2B. Sett inn tallene 8, 7, 6, 5, 4, 3, 2 og 1 i den oppgitte rekkefølgen i et på forhånd tomt rød/svart tre. Tegn treet etter hver innsetting. Skriv en S ved siden av en svart node og en R ved siden av en rød node hvis du ikke bruker fargeblyant eller fargepenn.

Oppgave 3

3A. Legg tallene 7, 2, 5, 11, 10, 4, 6, 5, 15, 12, 8, 7 og 9 i den oppgitte rekkefølgen, inn i et på forhånd tomt **binært søketre**. Tegn treet. Skriv ut verdiene i preorden.

I vedlegget er det satt opp et «skjelett» for klassen `SBinTre` – et vanlig binært søketre (duplikater er tillatt). Det skal **ikke** legges inn flere instansvariabler eller statiske variabler hverken i den indre `Node`-klassen eller i `SBinTre`-klassen. Noen av metodene i «skjelettet» er ferdigkodet. Klassen skal ikke ha flere offentlige metoder enn de som allerede er satt opp. Men en kan lage nye private hjelpemetoder.

3B. Lag metoden `public T rotverdi()`. Den skal returnere verdien i treets rot. Hvis roten ikke finnes, dvs. at treet er tomt, skal det kastes en `NoSuchElementException` med feilmeldingen *Treet er tomt!*.

3C. Dybden til en node er lik avstanden mellom noden og rotnoden, dvs. antall kanter på veien mellom dem. Lag metoden `private int dybde(Node<T> p)`. Den skal returnere dybden til noden `p`. Det kan antas at `p` ikke er `null`.

3D. ***i)*** Ta utgangspunkt i tegningen du laget i punkt **3A)**. Hva er den første noden i preorden. Hva er den neste i preorden for noden med verdien 11? Hva er den neste i preorden for noden med verdien 2? Hva er den neste i preorden for noden med verdien 9?

ii) Lag metoden `private Node<T> nestePreorden(Node<T> p)`. Den skal returnere den noden som kommer rett etter `p` i preorden. Hvis `p` er den siste i preorden skal den returnere `null`. Det kan antas at `p` ikke er `null`.

iii) Lag så metoden `public void traverserPreorden(Oppgave<? super T> oppgave)`. Den skal ved hjelp av metoden `nestePreorden` traversere treet i preorden og «utføre oppgaven» for hver nodeverdi. Du kan benytte `nestePreorden` selv om du ikke har kodet den.

3E. En node er en forgjenger til en annen node hvis det går en vei fra den første ned til den andre. Hvis `x` og `y` er to noder slik at `y` er forgjenger til `x` eller `x` er forgjenger til `y`, så er deres avstand, dvs. `avstand(x,y)`, lik lengden på veien mellom dem. Hvis ikke, vil `x` og `y` ha en **nærmeste** felles forgjenger `z`. Da er deres avstand lik summen av avstandene mellom `x` og `z` og mellom `y` og `z`, dvs. `avstand(x,y) = avstand(z,x) + avstand(z,y)`.

i) Ta utgangspunkt i tegningen du laget i punkt **3A)**. La `x` være noden med verdien 2 og `y` noden med verdien 6. Hva er avstanden mellom dem? La isteden `x` være noden med verdien 9 og `y` noden med verdien 12. Hvilken node er deres nærmeste felles forgjenger og hva er nå avstanden mellom `x` og `y`?

ii) Lag metoden `private int avstand(Node<T> x, Node<T> y)`. Den skal returnere avstanden mellom `x` og `y`. Det kan antas at ingen av dem er `null`.

Vedlegg - Algoritmer og datastrukturer - 30.11.2010

```
import java.util.*;

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

  } // grensesnittet Kø
```

```
public static <T> String toString(Kø<T> kø)
{
  return null;  // foreløpig kode - skal kodes
}


public static int[] utenDuplikater(int[] a)
{
  return null;  // foreløpig kode - skal kodes
}


public class SBinTre<T> // binært søketre
{
  private static final class Node<T>    // en indre nodeklasse
  {
    private T verdi;                    // nodens verdi
    private Node<T> venstre;            // peker til venstre barn/subtre
    private Node<T> høyre;              // peker til høyre barn/subtre

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

    private Node(T verdi)  // nodekonstruktør
    {
      this.verdi = verdi;
      venstre = høyre = null;
    }
  } // class Node

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

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

  public void leggInn(T verdi)
  {
    Node<T> p = rot, q = null;  // p starter i roten
    int cmp = 0;  // hjelpevariabel

    while (p != null)  // fortsetter til p er "ute av" treet
    {
      q = p;  // q skal være forelder til p
      cmp = comp.compare(verdi,p.verdi);  // bruker komparatoren
      p = cmp < 0 ? p.venstre : p.høyre;  // flytter p
    }
```

```
    p = new Node<T>(verdi);  // lager en ny node med gitt verdi

    if (rot == null) rot = p;    // treet var opprinnelig tomt
    else if (cmp < 0) q.venstre = p;
    else q.høyre = p;

    antall++;  // én verdi mer i treet
  }

  public int antall()
  {
    return antall;  // returnerer antallet
  }

  public boolean tom()
  {
    return antall == 0;  // tomt tre?
  }

  public T rotverdi()
  {
    return null;  // foreløpig kode - skal kodes
  }

  private int dybde(Node<T> p)
  {
    return 0;  // foreløpig kode - skal kodes
  }

  private Node<T> nestePreorden(Node<T> p)
  {
    return null;  // foreløpig kode - skal kodes
  }

  public void traverserPreoden(Oppgave<? super T> oppgave)
  {
    return;  // foreløpig kode - skal kodes
  }

  private int avstand(Node<T> x, Node<T> y)
  {
    return 0;  // foreløpig kode - skal kodes
  }

} // class SBinTre
```