Algoritmer og datastrukturer
Eksamen – 10.12.2015

Eksamensoppgaver

Råd og tips: Bruk ikke for lang tid på et punkt. Gå isteden videre til neste punkt og eventuelt tilbake hvis du får god tid. Resultatet fra et punkt som du ikke har løst, kan brukes senere i en oppgave som om det var løst. Prøv alle punktene. De 10 bokstavpunktene teller likt, men hvis et bokstavpunkt er delt opp, kan en krevende del telle mer enn en enkel del. Hvis du skulle trenge en hjelpestruktur (liste, stakk, kø o.l.) fra *java.util* eller fra kompendiet, kan du fritt bruke den uten å måtte kode den selv. Men den må brukes på en korrekt måte. Men du bør si fra om dette i en kommentar.

Oppgave 1

1A.

i) Klassen TabellKø er en implementasjon av grensesnittet Kø (se vedlegget). Hvis *kø* er en instans av TabellKø, vil System.***out***.println(kø); skrive ut køens innhold på vanlig form (uten at køen endres). Det er metoden *toString*() som implisitt blir kalt.

Sett opp den ***utskriften*** som flg. programsetninger vil gi og begrunn svaret ditt:

```
String[] person = {"Per","Kari","Elin","Ali"};

Kø<String> kø = new TabellKø<>();
for (String p : person) kø.leggInn(p);  // legger inn i køen

System.out.println(kø);  // skriver ut køen

kø.leggInn(kø.taUt());   // tar ut og legger inn
String p = kø.taUt();    // tar ut
kø.leggInn(kø.taUt());   // tar ut og legger inn

kø.leggInn(p);           // legger inn
kø.leggInn(kø.taUt());   // tar ut og legger inn

System.out.println(kø);  // skriver ut køen
```

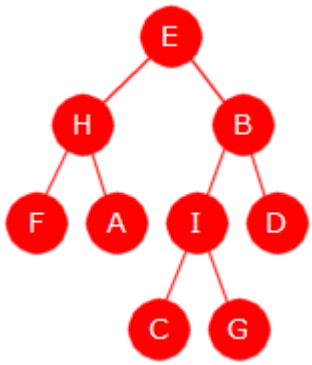
ii) Det hender, når vi står i en kø, at personen rett bak oss spør om å få bytte plass med oss. Du skal nå lage en generisk metode for dette. Den første i køen har indeks 0, den neste indeks 1, osv. Hvis det er n stykker i køen, er indeks til den siste lik $n - 1$. Lag metoden ***public static <T> void byttPlass(Kø<T> kø, int indeks)***. Metoden skal gjøre at den på plass *indeks* i køen bytter plass med den som står rett bak. Alle andre i køen skal ha samme indeks som før. Her må du teste om *indeks* er lovlig. Hvis ikke, skal det kastes et unntak med en passelig tekst. Du kan bruke hjelpevariabler i koden og eventuelt en hjelpestruktur, men du vil få best score hvis du kun bruker hjelpevariabler. Siden parametertypen er Kø er det kun metodene i grensesnittet Kø (se vedlegget) som kan benyttes. Antallet i køen kan variere fra 0 til så mange vi bare måtte ønske. Her er et eksempel med en LenketKø:

```
String[] person = {"Per","Kari","Elin","Ali","Jens","Siri"};
Kø<String> kø = new LenketKø<>();
for (String p : person) kø.leggInn(p);  // legger inn i køen

System.out.println(kø);  // [Per, Kari, Elin, Ali, Jens, Siri]
byttPlass(kø, 4);  // den på indeks 4 bytter plass med den rett bak (indeks 5)
System.out.println(kø);  // [Per, Kari, Elin, Ali, Siri, Jens]
```

Slutt på Oppgave 1A

1B.



Figur 1

i) Figur 1 til venstre inneholder et binærtre. Skriv ut treets verdier først i *preorden* og så i *inorden*.

ii) I et bestemt binærtre ble verdiene skrevet ut både i *preorden* og i *inorden*. Resultatet ble:

preorden: A, B, D, H, E, C, F, G, I
inorden: H, D, B, E, A, F, C, G, I

Deretter ble treet fjernet. Treet kan gjenskapes ved hjelp av de to utskriftene. Gjør det, dvs. tegn treet! Det er kun én løsning. Skriv så ut, fra det treet du har tegnet, verdiene i nivåorden.

Oppgave 2

2A. Grensesnittet *Liste* står i vedlegget. *DobbeltLenketListe* er fra Oblig 2. Metoden *fjernHvis*() hører til Beholder og var bl.a. tema i Oblig 2 - 10b). Hva blir utskriften:

```
Liste<String> liste = new DobbeltLenketListe<>();
liste.leggInn("Erik"); liste.leggInn("Kari");
liste.leggInn(1, "Reidar"); liste.leggInn(1, "Jasmin");
liste.fjern(2);
System.out.println(liste);
liste.fjernHvis(navn -> navn.contains("i"));
System.out.println(liste);
```

2B. Lag metoden ***public static int fjernDuplikater(int[] a)***. Her skal du ta som gitt at tabellen *a* er sortert stigende og inneholder positive tall. Metoden skal fjerne eventuelle duplikater (like verdier) og sørge for at de som er forskjellige ligger i første del av tabellen (og er sortert). Siste delen av tabellen skal, hvis det var duplikater, inneholde 0-er. Metoden skal returnere antallet forskjellige verdier. Målet er å kode dette med så få ekstra ressurser (hjelpetabeller og strukturer) som mulig. *Hint*: Lag først en løsning som virker og hvis du får god tid, se om du kan forbedre den (med bruk av færre ekstra ressurser). Et eksempel:

```
int[] a = {1,3,5,5,6,8,8,8,9,10,10};    // en sortert tabell med duplikater
int antall = fjernDuplikater(a);        // kaller metoden
System.out.println(antall + ": " + Arrays.toString(a));  // antallet og tabellen
// Utskrift: 7: [1, 3, 5, 6, 8, 9, 10, 0, 0, 0, 0]
```

Oppgave 3

3A. *i*) AAAEEEEEAACCBAAAFFDDDDEEEEAAAAEEEBBFFDDDEEE er en «melding» som inneholder 43 bokstaver. Finn frekvensfordelingen, dvs. antallet av hver bokstav A - F.
ii) Bruk Huffmans metode til å finne Huffmantreet. Tegn treet! *iii*) Sett opp bokstavenes bitkoder! Vis at bitsummen blir lik 100. Dekomprimer dette: 011001111010

3B. *i*) Huffmans metode gir et tre ved hjelp av frekvensfordelingen til tegnene i en melding (som i 3A). Treet gir oss bitkodene. Omvendt: Har vi bitkodene, kan vi konstruere det tilhørende Huffmantreet (kun formen på treet, ikke frekvensene). Gitt at de seks tegnene i en melding fikk flg. bitkoder: A = 00, B = 01, C = 1000, D = 1001, E = 101 og F = 11. Tegn treet og sett bokstaver ved bladnodene! *Hint*: Det blir et fullt tre med seks bladnoder.

ii) Klassen Huffman (se vedlegget) har en indre klasse Node med variabelen *tegn* og to pekere. Lag konstruktøren: ***public* Huffman(String[] *bitkoder*)**. Den skal konstruere et Huffmantre. String-tabellen *bitkoder* inneholder bitkoder i form av tegnstrenger med '0' eller '1' som tegn. Tegnstrengen *bitkoder*[*i*] er

(hvis den ikke er *null*) bitkoden til tegnet med *i* som ascii-verdi. F.eks. vil *bitkoder*[65] (hvis den ikke er null) være bitkoden til A. Tegn kan hentes ut ved metoden *charAt*() eller ved å lage en tabell ved hjelp av *toCharArray*(). Det kan tas som gitt at tabellen *bitkoder* kommer fra et Huffmantre. *Hint*: Lag først rotnoden. En bitkode hører til en bladnode og bitene viser veien dit. Hvis det mangler noder på veien, må de opprettes underveis. I en bladnode skal det tilhørende tegnet legges inn. I de øvrige nodene skal tegnet være det som nodekonstruktøren gir.

Oppgave 4

4A. Høyden til en node i et binærtre er høyden til det *subtreet* som har denne noden som rotnode. En bladnode har høyde 0. Spesielt vil høyden til et ikke-tomt tre være lik høyden til treets rotnode. Gitt tallene: 12, 4, 10, 22, 2, 6, 17, 8, 20, 14, 15, 5. Sett dem inn, i den gitte rekkefølgen, i et på forhånd tomt *binært søketre*. Skriv nodens høyde ved siden av hver node. Hvis du har gjort dette riktig, vil f.eks. noden med verdi 4 få høyde 3.

Vedlegget har et skjelett for klassen SBinTre − et binært søketre der like verdier er tillatt. Klassen Node har også variabelen *høyde*. Den skal ha som verdi høyden til det subtreet som har denne noden som rotnode. Det skal ikke legges inn andre instansvariabler i de to klassene. Hvis du skulle trenge ytterligere metoder i SBinTre, må du kode dem selv.

4B. Lag metoden ***public T nestMinst()***. Den skal returnere den nest minste verdien i treet (dvs. verdien i den andre noden i inorden). Hvis treet har færre enn to verdier, skal det kastes et unntak med en passelig tekst. *Hint*: Pass på de to tilfellene for den første noden i inorden. 1) Den har et ikke-tomt høyre subtre og 2) den har et tomt høyre subtre.

4C. *i*) Sett inn verdien 9 på rett sortert plass i det binære søketreet du laget i *Oppgave* 4A. Start så i den nye noden og gå oppover mot roten. På veien vil du se hvilke noder som må få endret sin høyde. Sett så inn verdien 13 og gjør på samme måte. Tegn det treet du nå har fått med korrekte høydetall ved siden av hver node.

ii) Lag metoden ***public boolean leggInn(T verdi)***. Den skal legge inn *verdi* på rett sortert plass i treet og samtidig sørge for at de nodene som blir påvirket etter innleggingen, får oppdatert verdien på sin *høyde*-variabel.

4D. Klassen SBinTre har metoden ***public T avstand(T verdi, int d)***. La *p* være noden som inneholder *verdi*. Hver node i subtreet med *p* som rotnode, har en bestemt avstand opp til *p* (antall kanter på veien). Metoden skal returnere verdien i den noden i subtreet (med *p* som rotnode) som har avstand *d* opp til *p*. Hvis subtreet har flere noder med avstand *d* opp til p, skal verdien i den av dem med størst verdi returneres. Hvis *d* er negativ, skal det kastes et unntak. Hvis *verdi* ikke finnes eller finnes flere ganger eller det ikke finnes noen node i subtreet med avstand *d* opp til *p*, returneres *null*. Lag metoden! Flg. eksempel viser hvordan det skal virke for treet fra Oppgave 4A med tilleggsverdiene 9 og 13 fra 4C:

```
int[] a = {12,4,10,22,2,6,17,8,20,14,15,5,9,13};  // verdiene fra Oppgave 4A/C
SBinTre<Integer> tre = new SBinTre<>(Comparator.naturalOrder());
for (int k : a) tre.leggInn(k);  // legger inn
System.out.print(tre.avstand(12,4) + " " + tre.avstand(4,4)); // Utskrift: 15 9
```

Vedlegg

```
public interface Kø<T>
{
  public void leggInn(T t);   // legger inn bakerst i køen
  public T kikk();            // ser på den første i køen
  public T taUt();            // tar ut den første i køen
  public int antall();        // antall i køen
  public boolean tom();       // er køen tom?
  public void nullstill();    // tømmer køen
  public String toString();   // køens innhold
}

////////////////////////////////////////////////

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 plass indeks
  public boolean inneholder(T verdi);       // Er verdi i listen?
  public T hent(int indeks);                // Hent verdi på plass indeks
  public int indeksTil(T verdi);            // Hvor ligger verdi?
  public T oppdater(int indeks, T verdi);   // Oppdater på plass indeks
  public boolean fjern(T verdi);            // Fjern verdi
  public T fjern(int indeks);               // Fjern verdien på plass indeks
  public int antall();                      // Antallet i listen
  public boolean tom();                     // Er listen tom?
  public void nullstill();                  // Listen nullstilles (og tømmes)
  public Iterator<T> iterator();            // En iterator
}

boolean fjernHvis(Predicate<? super T> p)    // hører til grensesnittet Beholder

////////////////////////////////////////////////

public class Huffman
{
  private static final class Node
  {
    private char tegn;     // et tegn
    private Node venstre;  // peker til venstre barn
    private Node høyre;    // peker til høyre barn

    private Node()         // konstruktør
    {
      tegn = 0; venstre = høyre = null;
    }
  } // slutt på class Node

  private Node rot;        // instansvariabel

  public Huffman(String[] bitkoder)
  {
    // kode mangler - skal lages
  }

} // slutt på class Huffman

////////////////////////////////////////////////
```

```
public class SBinTre<T>
{
  private static final class Node<T> // en indre nodeklasse
  {
    private T verdi;               // nodens verdi
    private Node<T> venstre, høyre;  // venstre og høyre barn
    private int høyde;               // nodens høyde

    private Node(T verdi)           // nodekonstruktør
    {
      this.verdi = verdi;
      venstre = null;
      høyre =   null;
      høyde = 0;
    }

  } // slutt på 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 int høyde() { return tom() ? -1 : rot.høyde; }

  public boolean leggInn(T verdi)
  {
    // kode mangler - skal kodes
  }

  public T nestMinst()
  {
    // kode mangler - skal kodes
  }

  public T avstand(T verdi, int d)
  {
    // kode mangler - skal kodes
  }

} // slutt på class SBinTre
```