Algoritmer og datastrukturer
Eksamen − 01.03.2017

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. Hvis du har idéer om hvordan ting skal løses, men likevel ikke klarer å få det til, kan du demonstrere idéene dine med ord, tegninger o.l.

1A. Lag metoden **public static void *snu*(int**[] a). Den skal snu (reversere) innholdet i tabellen a. Målet er en metode som er mest mulig effektiv og med minst mulig bruk av hjelpestrukturer. Du skal lage all kode selv.

Et eksempel på hvordan metoden skal virke:

```
int[] a = {4, 2, 5, 1, 3, 6}, b = {};
snu(a); snu(b);
System.out.println(Arrays.toString(a) + " " + Arrays.toString(b));
// Utskrift: [6, 3, 1, 5, 2, 4] []
```

1B. Metoden **public static int *finn*(int**[] a, **int** verdi) har en stigende sortert tabell a og en *verdi* som parametere. Hvis *verdi* ligger i tabellen a, skal metoden returnere dens indeks. Hvis a inneholder flere forekomster av *verdi*, så skal indeksen til den første av dem returneres. Hvis *verdi* ikke ligger i tabellen, så skal metoden returnere −(indeks + 1) der indeks er innsettingspunktet til *verdi*, dvs. der *verdi* ville ha ligget hvis den hadde vært der. Lag metoden! Du skal lage all kode selv. Det optimale er å bruke en binærsøk-teknikk, men du vil få poeng (men færre) også om du bruker en annen teknikk.

Et eksempel på hvordan metoden skal virke:

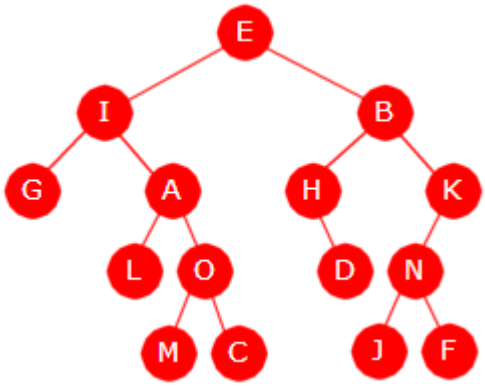
```
int[] a = {2, 3, 5, 7, 10, 12, 12, 15, 18, 20};
System.out.println( finn(a, 1) );   // utskrift -1
System.out.println( finn(a, 12) );  // utskrift 5
System.out.println( finn(a, 16) );  // utskrift -9
System.out.println( finn(a, 21) );  // utskrift -11
```

2A.

Hver node i et binærtre har en posisjon. Rotnoden har posisjon 1. Videre gjelder at hvis en node har posisjon k, så vil nodens venstre barn (hvis den har et venstre barn) ha posisjon $2k$ og høyre barn (hvis den har et høyre barn) ha posisjon $2k + 1$.

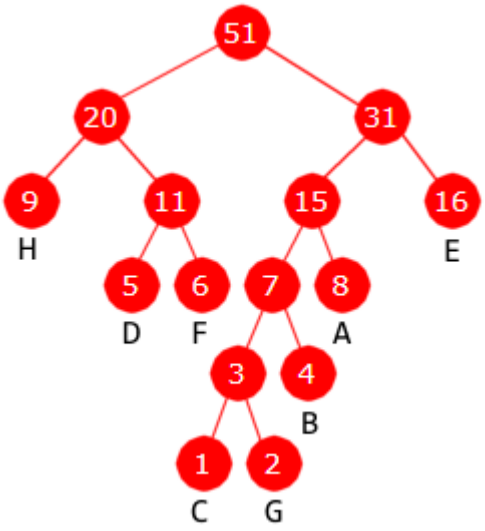
- Lag en kopi av treet til venstre i din besvarelse og skriv ved siden av hver node dens posisjon.

- Legg inn noder med bokstavene P og Q i treet slik at de får posisjoner 26 (for P) og 45 (for Q).



- Skriv ut (etter at du har lagt inn P og Q) treets verdier (dvs. bokstavene) i nivåorden og i inorden.

2B. Bokstavene A, B, C, D, E, F, G og H har frekvenser 8, 4, 1, 5, 16, 6, 2 og 9. Huffmans algoritme gir treet under der frekvensene er nodeverdier. Under hver bladnode står den tilhørende bokstaven.



i) Skriv ut treets verdier (frekvenser) i ***speilvendt*** nivåorden (dvs. nivå for nivå fra rotnoden og nedover og for hvert nivå fra ***høyre*** mot venstre). Hva betyr det at et binærtre er fullt?

ii) Vi oppdager at A egentlig skal ha frekvens 10 i stedenfor 8, og E frekvens 14 istedenfor 16. Tegn det Huffmantreet som dette gir. Hint: En mulighet er å bruke Huffmans algoritme til å bygge opp treet på nytt. En annen er å justere det eksisterende treet ved å bruke at det må være fullt, frekvensen i hver indre node må være lik summen av frekvensene til barna og at nodeverdiene (frekvensene) må komme i avtagende rekkefølge i speilvendt nivåorden.

3A. Lag metoden **public static** <T> **int** ***fjernBakerst***(Kø<T> kø, **int** antall). Den skal fjerne så mange verdier bakerst i køen *kø* som det *antall* sier. (Grensesnittet Kø står i vedlegget.) Metoden skal returnere antallet som ble fjernet. Hvis *antall* er større enn eller lik køens lengde (antall i køen), skal alle fjernes (dermed blir køen tom). Metoden skal virke for enhver klasse som implementerer grensenittet Kø. Du får best score hvis du løser dette uten bruk av hjelpestrukturer.

Et eksempel på hvordan metoden skal virke:

```
Kø<Character> kø = new LenketKø<>();
char[] c = "ABCDEFGHI".toCharArray();
for (char d : c) kø.leggInn(d);
System.out.println(kø);              // [A, B, C, D, E, F, G, H, I]
int antall = fjernBakerst(kø,5);
System.out.println(antall + " " + kø);   // 5 [A, B, C, D]
antall = fjernBakerst(kø,5);
System.out.println(antall + " " + kø);   // 4 []
```

3B. Hva blir utskriften til slutt i flg. programbit. Begrunn svaret ditt! Grensesnittet Stakk står i vedlegget.

```
Stakk<Character> stakk = new TabellStakk<>();
char[] bokstaver = "EDCBA".toCharArray();
for (char b : bokstaver) stakk.leggInn(b);
System.out.println(stakk);  // Utskrift: [A, B, C, D, E]
```

```
Stakk<Character> hjelp = new LenketStakk<>();

int n = stakk.antall();
for (int i = 0; i < n - 1; i++) hjelp.leggInn(stakk.taUt());
Character temp = stakk.taUt();
while (!hjelp.tom()) stakk.leggInn(hjelp.taUt());
stakk.leggInn(temp);
System.out.println(stakk);
```

4A. Gitt tallene 10, 5, 9, 1, 15, 13, 4, 6, 8, 2, 14, 11, 12, 3. Legg dem inn, i den gitte rekkefølgen, i et på forhånd tomt *binært søketre*. Tegn treet! Hvilken høyde har treet? Det er en bestemt algoritme for å fjerne verdier fra et binært søketre. Bruk den til å fjerne verdiene 1 og 10. Tegn treet på nytt (etter at de to er fjernet). Hvilken høyde har treet nå?

Vedlegget har et «skjelett» av klassen SBinTre - et binært søketre der (som vanlig) like verdier er tillatt.

4B. Beskriv med ord hvor den minste verdien i et binært søketre ligger. Lag så metoden **public T** min() i SBinTre. Den skal returnere treets minste verdi. Hvis treet er tomt, returneres null.

4C. SBinTre har instansvariabelen *høyde* og metoden *høyde*(). Se vedlegget. Variabelen *høyde* skal til enhver tid ha som verdi den høyden treet har.

i) Metoden *leggInn*() (se vedlegget) er nesten ferdigkodet, men *høyde* blir ikke oppdatert. Lag tilleggskode slik at instansvariabelen *høyde* får korrekt verdi etter en innlegging.

ii) Lag metoden **private int** *dybde*(Node<T> p). Den skal returnere dybden til noden *p*, dvs. avstanden mellom *p* og rotnoden. Her kan det tas som gitt at *p* ikke er *null*. Obs: Siden det er tillatt med like verdier i treet, kan to forskjellige noder ha like verdier. Kan forskjellige noder med like verdier ligge på samme nivå? Begrunn svaret!

4D. *i)* Skriv opp verdiene på nederste nivå (i sortert rekkefølge) i de to trærne du tegnet i Oppgave 4A.
ii) Lag metoden **public** T[] *nedersteNivå*() i SBinTre. Den skal returnere en tabell som inneholder (i sortert rekkefølge) de verdiene som ligger på treets nederste nivå. Hvis treet er tomt, returneres en tom tabell. Hvilken orden har metoden din? Begrunn svaret!

Vedlegg

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

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

public interface Stakk<T>
{
  public void leggInn(T verdi);     // legger verdi på toppen
  public T kikk();                  // ser på den øverste
  public T taUt();                  // tar ut den øverste
  public int antall();              // antall på stakken
  public boolean tom();             // er stakken tom?
  public void nullstill();          // tømmer stakken
```

```
  public String toString();          // fra toppen og nedover
}
```

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

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

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

  } // class Node

  private Node<T> rot;
  private int antall;
  private int høyde;
  private final Comparator<? super T> comp;

  public SBinTre(Comparator<? super T> c)  // konstruktør - lager et tomt tre
  {
    rot = null;    // rot er null i et tomt tre
    antall = 0;    // et tomt tre har ingen noder
    høyde = -1;    // et tomt tre har høyde lik -1
    comp = c;      // komparatoren får verdi
  }

  public int antall() { return antall; }

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

  public int høyde() { return høyde; }

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

    while (p != null)
    {
      q = p;
      cmp = comp.compare(verdi,p.verdi);
      p = cmp < 0 ? p.venstre : 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 min()
  {
    // Skal kodes
```

```
  }

  private int dybde(Node<T> p)
  {
    // Skal kodes
  }

  public T[] nedersteNivå()
  {
    // Skal kodes
  }

} // class SBinTre
```