Løsningsforslag til Eksamen i

DATS2300 / ITPE2300 Algoritmer og datastrukturer 19.12.2017

Oppgave 1 Tabell, sortering

Lag metoden `public static void frekvensFordeling(int[] a)`*. a skal være sortert stigende, og hvis a ikke er det, skal det kastes en* `IllegalStateException` *(med en passende tekst). Metoden skal skrive ut de forskjellige verdiene i a og hvor mange ganger de dukker opp. Hvis f.eks. a inneholder 3, 3, 4, 5, 5, 6, 7, 7, 7 og 8, skal metoden returnere*

3 2

4 1

5 2

6 1

7 3

8 1

Metoden skal ***ikke*** *endre noe på tabellens innhold. Hvis tabellen er tom (har lengde 0), skal metoden returnere*

0 0

Med andre ord er ikke en tom tabell en feilsituasjon.

Vedlegg: ingen

Løsning:

```
public static void frekvensFordeling(int[] a)
{
    if(a.length == 0)
    {
        System.out.println(0 + " " + 0);
    }
    int temp = a[0]; //verdien man måler frekvensen til
    int frekvens = 1; //frekvensen for verdien

    for(int i = 1; i < a.length; i++)
    {
        if(temp == a[i]) //sjekker om temp er lik som verdi i tabellen
        {
            frekvens++;
        } else if(temp < a[i])
        {
```

```
            System.out.println(temp + " " + frekvens);
            temp = a[i];
            frekvens = 1;
        } else  //tabellen er her ikke sortert
        {
            throw new IllegalStateException("Tabellen er ikke sortert");
        }
    }
    //må ha en ekstra utskrift for å få skrevet ut den siste verdien (som også evt. er den eneste)
    System.out.println(temp + " " + frekvens);
}
```

Oppgave 2 komparator

En datastruktur s av typen Iterable (se vedlegget) har garantert en iterator. Ved hjelp av den kan datastrukturen traverseres. Lag den generiske metoden
public static <T> T maks(Iterable<T> s, Comparator<? super T> c).
Den skal returnere største verdi i s. Sammenligningene gjøres ved hjelp av komparatoren c. Hvis s er tom (det kan avgjøres ved hjelp av iteratoren), skal det kastes et unntak.

Vedlegg

```
/// Oppgave 2 ////////////////////
public interface Iterable<T>
{
  Iterator<T> iterator();
}
```

Løsning:

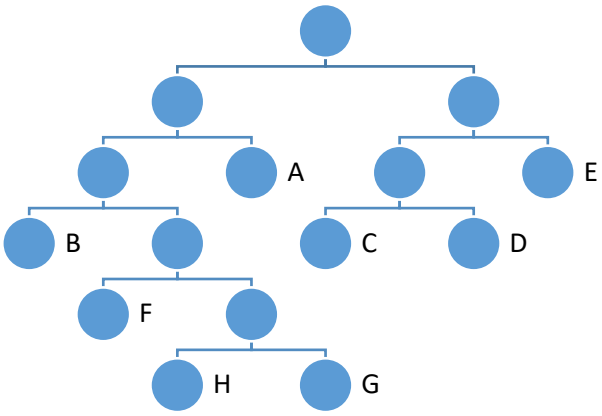
```
public static <T> T maks(Iterable<T> s, Comparator<? super T> c)
  {
    Iterator<T> i = s.iterator();
    if (!i.hasNext()) throw new NoSuchElementException("s er tom!");

    T maks = i.next();

    while (i.hasNext())
    {
      T verdi = i.next();
      if (c.compare(verdi,maks) > 0) maks = verdi;
    }
    return maks;
  }
```

Oppgave 3 Huffman tre

En melding inneholder kun tegnene A, B, C, D, E, F, G og H. Tegnenes frekvensfordeling gir følgende Huffmantre der hvert tegn står under tilhørende bladnode:



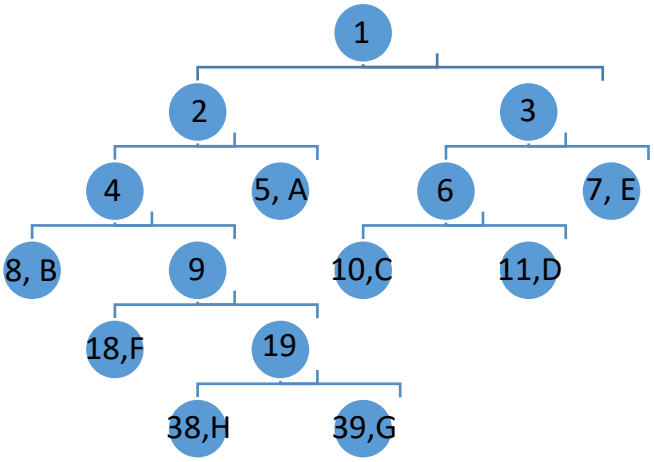
I et binærtre har hver node en posisjon. Rotnoden har posisjon 1. Hvis en node har posisjon k, har venstre barn posisjon 2k og høyre barn posisjon 2k + 1.

1. *Sett opp posisjonene til bladnodene i Huffmantreet.*
2. *Sett opp de forskjellige bitkodene som treet gir for de åtte tegnene.*
3. *En melding som ble komprimert ved hjelp av disse bitkodene, ble komprimert til 00001101111010100111. Dekomprimer dette.*

Vedlegg: ingen

Løsning:

1.



2.

A = 01

B = 000

C = 100

D = 101

E = 11

F = 0010

G = 00111

H = 00110

3. BADEDAG

Oppgave 4

I denne oppgaven skal du tegne to binære søketrær.

Gitt tallene 12, 7, 11, 3, 17, 15, 6, 8, 10, 4, 16, 13, 14, 5. Legg dem inn, i den gitte rekkefølgen, i et på forhånd tomt `binært søketre`.

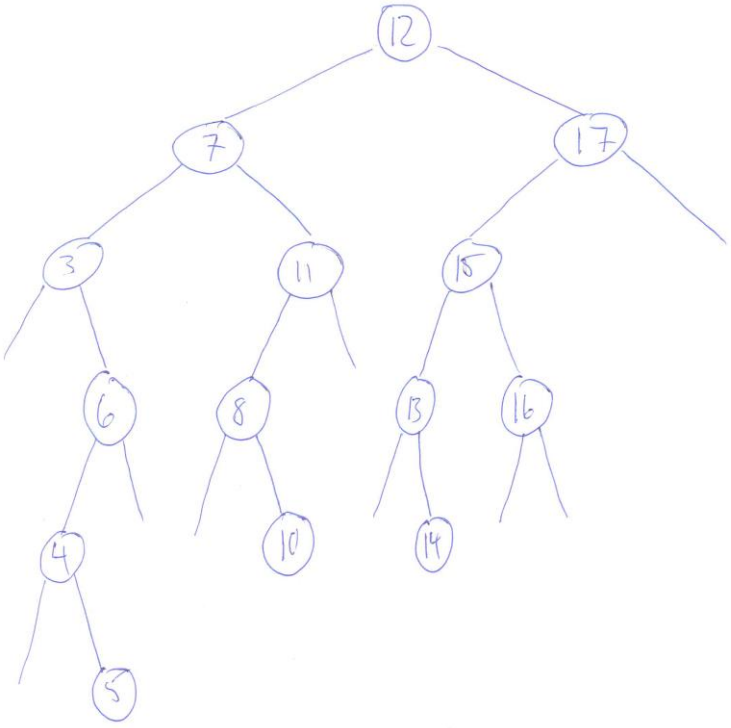
Tegn treet! Hvilken høyde har treet?

Du skal nå legge inn de samme tallene i et binært rød-svart søketre, marker røde noder som firkanter, og sorte noder som rundinger.

Tegn treet etter at du har lagt inn 12, 7, og 11. Hvilken høyde har dette treet?
Tegn treet etter at du i tillegg har lagt inn 3, 17, og 15. Hvilken høyde har dette treet?
Tegn det ferdige treet (etter at alle tallene er lagt inn). Hvilken høyde har dette treet?

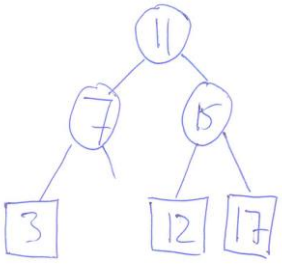
Løsning:

Binært tre etter å ha lagt inn 12, 7, 11, 3, 17, 15, 6, 8, 10, 4, 16, 13, 14, 5. Treet har høyde 5 (lengste veien i treet har lengde 5).

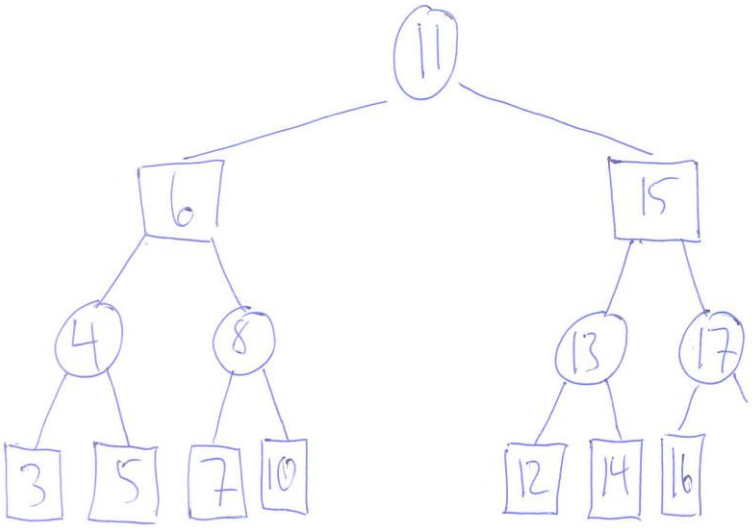


Rød-svart tre etter å ha lagt inn 12, 7, og 11 har høyde en.

Rød-svart tre etter å ha lagt inn 3, 17, 15 har høyde to.



Rød-svart tre etter å ha lagt inn alle tallene har høyde 3.



Oppgave 5

Klassen `LenketHashTabell` bruker «lukket adressering med separat lenking». Den inneholder en `tabell` med `nodereferanser` der alle i utgangspunktet er null. Et `objekt` legges inn på objektets `tabellindeks` (objektets hashverdi modulo tabellengden). Dvs. en node (med objektet) legges først i den (eventuelt tomme) lenkede nodelisten som hører til tabellindeksen.

En samling navn (tegnstrenger) skal legges inn. Det er en jobb å regne ut hashverdier og indekser for hånd. Dette er derfor allerede gjort for noen lengder/dimensjoner:

navn	*Espen*	*Bo*	*Ali*	*Petter*	*Karl*	*Siri*	*Muhammad*	*Mari*	*August*	*Åse*
hashverdi	*80088*	*2357*	*65968*	*89125*	*69562*	*257197*	*7934*	*23763*	*65085*	*1983*
hashverdi % 13	*8*	*4*	*6*	*10*	*12*	*5*	*4*	*12*	*7*	*7*
hashverdi % 17	*1*	*11*	*8*	*11*	*15*	*4*	*12*	*14*	*9*	*11*
hashverdi % 19	*3*	*1*	*0*	*15*	*3*	*13*	*11*	*13*	*10*	*7*

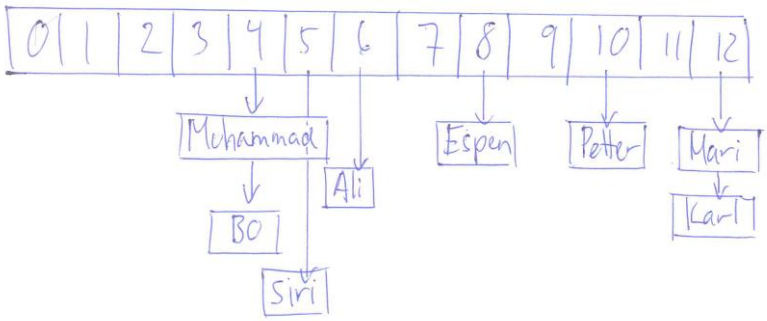
*Setningen `LenketHashTabell<String> hash = new LenketHashTabell<>(n);` oppretter en instans av klassen der den interne tabellen får dimensjon (lengde) lik **n**. Legg inn én og én verdi (`leggInn`-metoden med et navn som verdi) i den gitte rekkefølgen (dvs. Espen, Bo, Ali, osv). En node skal ha både `verdi` og `hashverdi`, men på en tegning holder det med `verdi`.*

Utfør følgende oppgave med en lenket hashtabell av lengde 13

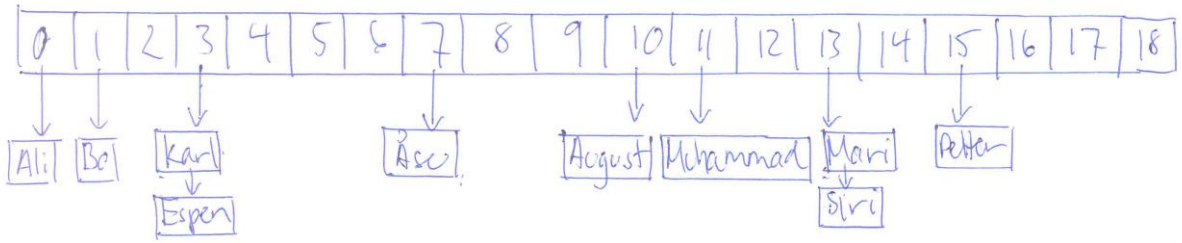
- *Lag en tegning av datastrukturen når de åtte første navnene er lagt inn*
- *Lag så en tegning som viser når alle navnene er lagt inn*

Løsning:

Lenket hashtabell med lukket addressering med separat lenking etter de 8 første navnene er lagt inn.



Lenket hashtabell med lukket addressering med separat lenking etter alle navnene er lagt inn (hvis man har tatt hensyn til tetthet (load factor) og økt kapasiteten til 19.



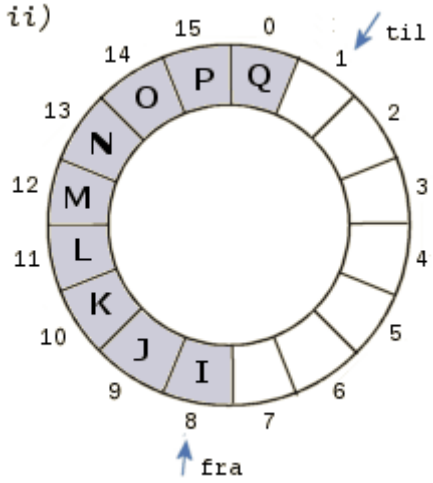
Oppgave 6

En TabellKø oppfører seg som en vanlig kø. Dvs. leggInnBak() legger en verdi bakerst i køen, taUtForan() tar ut den første verdien i køen, og tilsvarende for leggInnForan() og taUtBak(). Hva blir utskriften i følgende programbit:

```
Kø<Character> kø = new TabellKø<>();
char[] c1 = "ABCDEFGHI".toCharArray(), c2 = "JKLM".toCharArray();

for (char bokstav : c1) kø.leggInnForan(bokstav);
for (int i = 0; i < 8; i++) kø.taUtBak();
for (char bokstav : c2) kø.leggInnBak(bokstav);
System.out.println(kø);  // skriver ut køen
```

ii)



Klassen TabellKø bruker internt en såkalt sirkulær tabell. Se figuren. Der refererer indeks fra til den første i køen og til til den første ledige plassen (en bak den siste).

. Legg inn N, O, P, og Q i den sirkulære køen med leggInnForan() og ta så ut seks verdier med taUtBak(). Tegn så køen (du behøver ikke bruke grå bakgrunn)! Hvor mange verdier har køen?

Løsning:

Programsnutten legger først inn bokstavene A-I i tabellen, og vi får

Foran -> [I, H, G, F, E, D, C, B, A] <- bak

Vi tar så ut 8 bokstaver og står igjen med

Foran -> [I] <- bak

Legger så inn J-M bak og får

Foran -> [I, J, K, L, M] <- bak

Utskriften blir dermed

I, J, K, L, M

Køen går fra element 8 (fra-peker / foran) til element 1 (til-peker / bak). Ved å legge inn foran, så legges disse inn i elementer 7, 6, 5, og 4. Ved å ta ut seks verdier med taUtBak så flyttes til-pekeren til element 11. Køen har nå 7 verdier.