Algoritmer og datastrukturer
Eksamen – 22. februar 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. I en undervisningstime ble Huffmans metode brukt til å lage et Huffmantre for tegnene *A, B, D, E, H, K, M, N, S* og og ved hjelp av treet var det mulig å sette opp bitkodene som *Tabell* 1 under viser. Men en student som skrev av dette var litt uoppmerksom. Studenten hadde glemt å notere bitkodene for både *K* og *M*. Studenten hadde imidlertid fått med seg at det var 3 biter i bitkoden til *K*. Det er mulig å rekonstruere Huffmantreet ved hjelp av det som nå er kjent. Gjør det (tegn treet) og bruk treet til å dekomprimere flg. del av en komprimert melding: 011101011100001101111.

Tegn	A	B	D	E	H	K	M	N	S
Bitkode	100	0010	1010	01	000	*	*	111	1011

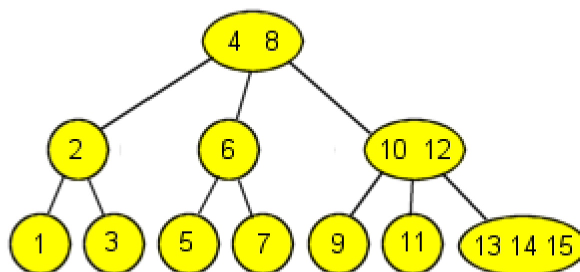
Tabell 1 : Bitkoder fra et Huffmantre

1B. Hva blir utskriften fra flg. programbit? Gi forklaring!

```
Integer[] a = {2,5,3,1,4};
Stakk<Integer> s = new TabellStakk<Integer>();
for (int i = 0; i < a.length; i++) s.leggInn(a[i]);
while (!s.tom()) System.out.print(s.taUt() + " ");
```

1C. Det er gitt et 2-3-4 tre. Se *Figur* 1 under. Gjør om dette treet til et rød/svart tre og tegn treet. Hvis en node i et 2-3-4 tre har to verdier, kan den gjøres om til to noder på to måter. Her skal du velge flg. måte: Den første (minste) verdien går over i en svart node og den andre i en rød node. Den røde noden skal være høyre barn til den svarte. Sett så inn verdien 16

i det rød/svarte treet du har kommet frem til. Bruk her reglene for å sette inn i et rød/svart tre. Tegn treet! Bruk fargepenn eller skriv R ved siden av en rød og S ved siden av en svart node.



Figur 1 : Et 2-3-4 tre med 11 noder

1D. 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.

Oppgave 2

2A. Det kan være aktuelt å «forskyve» elementene i en tabell. En forskyvning på én enhet gjøres ved at det siste elementet blir det første og alle de andre forskyves én enhet mot høyre.

A	B	C	D	E	F	G	H	I	J

Tabell 2 : Bokstavene fra A til I

J	A	B	C	D	E	F	G	H	I

Tabell 3 : Elementene i Tabell 2 forskjøvet én enhet

På figuren over har elementene i den første tabellen blitt forskjøvet én enhet. Lag metoden `public static <T> void forskyv(T[] a)`. Den skal forskyve innholdet i tabellen *a* én enhet.

2B. Hvis vi tenker oss at tabellen er «bøyd til en sirkel», kan en forskyvning ses på som en rotasjon. En forskyvning på *k* enheter blir derfor en rotasjon på *k* enheter. Hvis *k* er negativ, ser vi på det som en forskyvning eller rotasjon motsatt vei. Lag den generiske metoden `public static <T> void forskyv(T[] a, int k)` der k er et vilkårlig heltall. Hvis *k* = 1, skal metoden ha samme effekt som metoden i *Oppgave* 2*A*. Målet er å gjøre metoden så effektiv som mulig. Følgende programbit viser hvordan metoden skal virke:

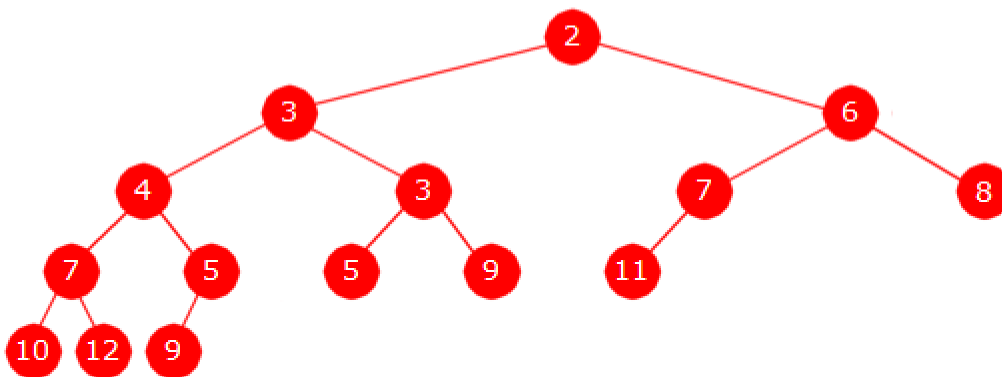
```
Character[] a = {'A','B','C','D','E','F','G','H','I','J'};
System.out.println(Arrays.toString(a));

forskyv(a,3); System.out.println(Arrays.toString(a));
forskyv(a,-2); System.out.println(Arrays.toString(a));

// Utskrift:
[A, B, C, D, E, F, G, H, I, J]   // originaltabellen
[H, I, J, A, B, C, D, E, F, G]   // forskyvning tre enheter mot høyre
[J, A, B, C, D, E, F, G, H, I]   // forskyvning to enheter mot venstre
```

Oppgave 3

En node *p* i et binærtre sies å ha *venstretyngde* hvis antall noder i det venstre subtreet til *p* er større enn eller lik antall noder i det høyre subtreet. Et tomt subtre har 0 noder. Et binærtre kalles *venstretungt* hvis hver node i treet har venstretyngde. Et tomt tre er venstretungt.

Et binærtre kalles et *minimumstre* hvis hver node (unntatt rotnoden) har en verdi som er større enn eller lik foreldreverdien. En gren i et binærtre består av alle nodene på veien fra rotnoden og ned til en bladnode. I et *minimumstre* er alle grenene sortert stigende.

Et binærtre kalles et *venstretungt minimumstre* hvis det både er *venstretungt* og er et *minimumstre*. På neste side er det satt opp et venstretungt minimumstre med 15 noder:



Figur 2 : Et venstretungt minimumstre med 15 noder

3A. 1) Tegn treet i *Figur* 2 i din besvarelse og skriv opp ved siden av hver node det antallet noder det er i treet/subtreet som har denne noden som rotnode.
2) Skriv opp verdiene i hver gren i treet i *Figur* 2. En linje for hver gren.
3) Skriv opp treets verdier i inorden.

Vedlegget inneholder et «skjelett» for klassen *VMBinTre* - et venstretungt minimumstre. Klassen har en privat indre klasse *Node* med ferdige konstruktører. I **nodeklassen** er det, i tillegg til de vanlige variablene, en *antall*-variabel. Verdien til den skal alltid være lik antallet noder i det treet/subtreet som har denne noden som rotnode. Det skal ikke legges inn flere variabler (instans- eller klassevariabler) i klassen *VMBinTre* og klassen *Node* enn de som allerede er der.

3B. Metodene `public T kikk()`, `public int antall()` og `public boolean tom()` er satt opp i vedlegget. Metoden `kikk()` skal returnere (og ikke fjerne) den minste verdien i treet, `antall()` skal returnere antallet noder i treet og `tom()` skal returnere *true* hvis treet er tomt og *false* ellers. Lag kode for disse tre metodene.

3C. Metoden `public boolean erVenstretungt()` (se vedlegget) skal returnere *true* hvis treet er venstretungt og *false* ellers. Lag metoden.

3D. Ved innlegging av verdier må treet opprettholdes som et venstretungt minimumstre. En ny verdi legges på rett sortert plass i treets høyre kant. I *Figur* 2 består høyre kant av de tre nodene 2, 6 og 8. Generelt består høyre kant i et binærtre av de nodene vi får ved å starte i rotnoden og så gå videre nedover til høyre så langt det går. I en ny node skal alltid venstre barn være null. Hvis den nye verdien er mindre enn rotnodens verdi, legges den i en ny rotnode og

den gamle rotnoden blir høyre barn til den nye rotnoden. Hvis en ny verdi hører hjemme mellom to noder i høyre kant (f.eks. mellom 2 og 6), legges den mellom de to aktuelle nodene. Den nye noden blir dermed høyre barn til den første (her 2) og den andre (her 6) blir høyre barn til den nye noden. Hvis ny verdi er større enn den siste på høyre kant, blir den nye noden høyre barn til denne. Dette fører til at treet bevares som et minimumstre. Men fra og med den nye noden og oppover til roten, kan det ha skjedd at en eller flere noder ikke lenger er venstretunge. Dette må repareres ved at venstre og høyre subtre bytter plass i alle slike noder.

1. Legg inn verdiene 10, 12 og 5 (i den gitte rekkefølgen) i treet i *Figur* 2. Tegn resultatet.
2. Metoden `public void leggInn(T verdi)` (se vedlegget) skal legge inn en verdi i treet slik at det opprettholdes som et venstretungt minimumstre. Lag kode for metoden. Hjelpemetoden `private static <T> void byttSubtrær(Node<T> p)` (se vedlegget) som bytter om subtrærne til noden *p*, er ferdigkodet og kan benyttes der det er aktuelt.

Vedlegg - Algoritmer og datastrukturer - februar 2011

```
/// Oppgave 1D ////////////////////

public interface Iterable<T>
{
  Iterator<T> iterator();
}

/// Oppgave 3 ////////////////////

public class VMBinTre<T>
{
  private static final class Node<T>  // en indre nodeklasse
  {
    private T verdi;            // nodens verdi
    private int antall;         // antall noder
    private Node<T> venstre;    // peker til venstre barn/subtre
    private Node<T> høyre;      // peker til høyre barn/subtre

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

    private Node(T verdi, int antall)  // konstruktør
    {
      this(verdi,antall,null,null);
    }
  } // class Node<T>

  private Node<T> rot;        // peker til rotnoden
  private Comparator<? super T> comp;
```

```
  private static <T> void byttSubtrær(Node<T> p)
  {
    Node<T> q = p.venstre;
    p.venstre = p.høyre;
    p.høyre = q;
  }

  public VMBinTre(Comparator<? super T> c)  // konstruktør
  {
    if (c == null) throw
      new NullPointerException("Komparatoren c er null!");
    rot = null;
    comp = c;
  }

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

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

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

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

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

}  // class VMBinTre<T>
```