Høgskolen i Østfold

EKSAMEN

med løsningsforslag

Emnekode:	**Emne:**
ITF20006	Algoritmer og datastrukturer
Dato: 20. mai 2009	**Eksamenstid:** kl 09.00 til kl 13.00
Hjelpemidler: 8 A4-sider (4 ark) med egne notater Kalkulator	**Faglærer:** Gunnar Misund

Eksamensoppgaven:
Oppgavesettet består av 6 sider inkludert vedlegg og denne forsiden. I implementasjonsoppgaver gir Java-kode best uttelling, men godt dokumentert pseudo-kode kan også brukes.

Oppgavesettet består av 6 oppgaver, med tilsammen 10 deloppgaver. Alle oppgavene skal besvares. Ved sensur teller hver deloppgave like mye. Karakteren fastsettes dog på basis av en helhetsvurdering av besvarelsen.

Kontroller at oppgaven er komplett før du begynner!

SKRIV TYDELIG :-)

Sensurfrist: 16. juni 2009

Karakterene er tilgjengelige for studenter på studentweb senest 2 virkedager etter oppgitt sensurfrist. Følg instruksjoner gitt på: `www.hiof.no/studentweb`

Oppgave 1

A) *Her følger 5 funksjoner. For hver av de skal du angi kompleksiteten i O-notasjon, inkludert en kort begrunnelse.*

```
void alg1(int k) {
   for(int i = k; i > 0; i--) {
     int j = i+k;
   }
}
```

En løkke som går k ganger: O(k)

```
void alg2(int n) {
   int j = 1;
   while (j <= n) {
     System.out.println(j);
     j *= 2;
   }
}
```

Løkka starter på 1, og for hver iterasjon dobles verdien på j, helt til j blir større enn n. Sagt på en annen måte: antallet iterasjoner er lik hvor mange ganger vi kan dele n på 2 til vi kommer til 1...dvs ca $\log_2 n$, som gir O(log n).

```
void alg3(int[] a) {
   int m = a.length - 1;
   while (m >= 0) {
     for (int i = m; i < a.length; i++) {
       System.out.print(a[i]+" ");
     }
     System.out.println();
     m--;
   }
}
```

I hver hoved-iterasjon reduseres verdien på m med 1, fra lengden på a minus 1 til 0. I tillegg har vi en indre løkka som starter på m, og går helt til a.length – 1. I første hovediterasjon går indre løkke 1 ganger, neste gang 2 ganger, etc. Helt til siste iterasjon, som går a.length ganger. Totalt blir det 1 + 2 + 3 + ... + N, der N er lengden på a. Summen av de N første heltallene er som kjent $(N^2 + N) / 2$, som da gir $O(N^2)$.

```
void alg4(int[] b) {
   for(int i = 0; i < b.length; i++) {
     alg3(b);
     alg1(i);
   }
}
```

Hovedløkka går N ganger, der N er lengden på b. Hver gang kalles alg3 med b, og alg1 med økende verdi på i. Siden alg3 har høyere orden enn alg1, kan vi se bort fra alg1. Hvis vi sier at N er lengden på B, blir det da en ytre løkke som kjøre N ganger, og den indre løkka har kvadratisk orden, totalt blir det da $O(N^3)$.

```
double alg5(double[] c, int i) {
   if (i <= 0 || i >= c.length) {
      return 0;
   }
   return c[i] + alg5(c, i/2);
}
```

Rutinen kaller seg selv rekursivt med halve verdien av i, og går helt til i blir 0. Antallet iterasjoner er da lik hvor mange ganger man kan dele i før vi får 0...dvs ca $\log_2 i$, som gir O(log i).

Slutt, oppgave 1

Oppgave 2

Vi har følgende sorterte tabell (array) med heltall:

3 – 8 – 42 – 56 – 61 – 70 – 82

A) *Beskriv hvordan et binærsøk etter verdien 42 foregår, gjerne ved hjelp av passende figurer.*

Finn den midterste indeksen, og sjekk om verdien ligger der. I så fall returnerer vi indeksen. Hvis verdien av den midterste indeksen er større en søkeverdien, foretar vi et rekursivt binærsøk på den venstre delen, fra startindeksen til midtindeksen-1. Hvis verdien er mindre, foretart vi et tilsvarende søk på den høyre delen, fra midtindeksen+1 til sluttindeksen. Søket stopper og returner hvis søke(del)tabellener tom.

B) *Implementer en* rekursiv *metode som søker etter en verdi i en heltallstabell (array), og returnerer indeksen der verdien finnes, -1 ellers. Funksjonen skal ha følgende signatur:*

```
int binærSøk(int[] liste, int verdi, <evt. ekstra argumenter>) { ... }
```

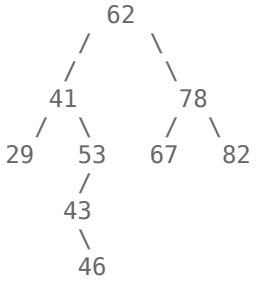
Du kan altså legge til flere argumenter hvis du har behov for det.

```
public int binærSøk(int[] liste, int verdi, int start, int slutt) {
    if (slutt-start < 0) {
        return -1;
    }

    int splitt = (start+slutt) / 2;
    if (liste[splitt] == verdi) {
        return splitt;
    }

    if (verdi < liste[splitt]) {
        return binærSøk(liste, verdi, start, splitt-1);
    }

    return binærSøk(liste, verdi, splitt+1, slutt);
}
```

Slutt, oppgave 2

Oppgave 3

Her er et binært søketre med heltallsverdier:



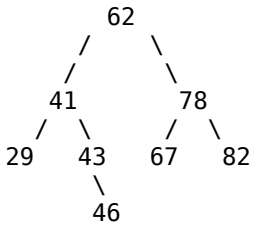
A) *Sett inn en node med verdi 42 i treet. Forklar framgangsmåten og tegn treet etter innsetting. Implementer deretter metoden* ***`settInn`****`(int v)`* *i klassen* ***`BinSøkeTre`*** *(se vedlegget), som setter inn en node med verdi v i treet. Metoden skal være* iterativ*.*

42 blir 43 sitt venstre barn. Man starter i rota, og sjekker om verdien er mindre eller lik, eller større. Hvis større, og rota ikke har noe høyre barn, settes noden inn der, og returnerer, og vise versa. Hvis større, og rota har ett høyre barn, gjentar man prosedyren med høyrebarnet som utgangspunkt, etc.

```
public void settInn(int v) {
 //...skal implementeres i 3A...
 if (rot == null) {
     rot = new Node(v);
     return;
 }

 Node node = rot;
 while (true) {
     if (v <= node.verdi) {
         if (node.venstre == null) {
             node.venstre = new Node(v);
             return;
         }
         node = node.venstre;
     } else {
         if (node.hoyre == null) {
             node.hoyre = new Node(v);
             return;
         }
         node = node.hoyre;
     }
 }
}
```

B) *Bruk standard regler for fjerning av noder i et binært søketre, og slett noden med verdi 53 fra treet slik det var før innsettingen. Forklar framgangsmåten, illustrer de ulike trinnene, og tegn treet etter sletting. Gjør det samme med node 41 (bruk treet slik det var opprinnelig før innsetting/sletting).*

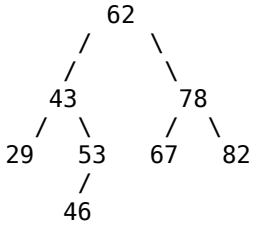
Slette 53: 53 har ett barn; vi sletter noden og setter inn subtreet med barnet som rot.



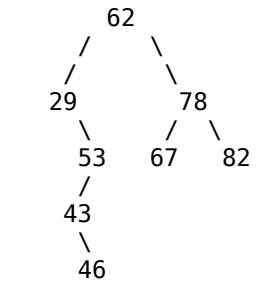
Slette 41: 41 har to barn; vi bytter med verdien av

1) noden i høyre subtre med minst verdi, dvs 43. Da må noden med 43 også slettes, den har ett barn, 46, som tar dens plass, og som til slutt slettes.
2) eller 2) noden med størst verdi i venstre subtre, dvs 29. Denne noden har ingen barn, og er bare å slette uten videre.

1)



2)



Slutt, oppgave 3

Oppgave 4

I et *perfekt* binærtre er hvert nivå helt fylt opp. Det medfører at alle bladnodene finnes på samme (nederste) nivå, og at alle interne noder har to barn. Et slikt tre kan representeres *implisitt* ved hjelp av en tabell som referer til nodene (eller deres verdier). Rotnoden plasseres i første posisjon, og tabellen fylles med resten av nodene i level-order rekkefølge. Med små modifikasjoner kan også *enhver type* binærtre representeres implisitt.

A) *Forklar kort hvordan et* generelt *binærtre kan implementeres* implisitt *ved hjelp av en tabell (array). Implementer metodene* ***`venstre`****`(int forelder)`* *og* `int` ***`hoyre`****`(int forelder)`* *i klassen* ***`GenBinTre`*** *gitt i vedlegget. Metodene returnerer indeksene til henholdsvis venstre og høyre barn gitt indeksen til foreldrenoden. Hvis ikke nodene finnes, returneres verdien* `TOM`*.*

Et generelt tre kan implementeres på samme måte som et komplett tre med tillegg av at man må markere (f.eks. med en spesiell verdi) at et element i tabellen peker på en «tom» plass, uten node.

```
// Returnerer indeksen til venstre barn (TOM hvis det ikke eksisterer)
int venstre(int forelder)  {
   int barn = (2*forelder) + 1;
   if (barn > 0 && barn < tre.length && tre[barn] != TOM) {
     return barn;
   }
  return TOM;
}

// Returnerer indeksen til venstre barn (TOM hvis det ikke eksisterer)
int hoyre(int forelder) { .
   int barn = (2*forelder) + 2;
   if (barn > 0 && barn < tre.length && tre[barn] != TOM) {
     return barn;
   }
  return TOM;
}
```

B) *Implementer den rekursive metoden* ***`preOrder`****`(int node)`* *i klassen* ***`GenBinTre`*** *i vedlegget. Metoden skriver ut verdiene i et implisitt binærtre i preorder rekkefølge.*

```
// Rekursiv traversering av treet med preorder utskrift av verdiene
void preOrder(int node) {
   if (node == TOM) {
     return;
   }
   System.out.println(tre[node]);
   preOrder(venstre(node));
   preOrder(hoyre(node));
}
```

Slutt, oppgave 4

Oppgave 5

Klassen **Nettverk** representerer et nettverk ved hjelp av en liste av alle nodene i nettverket. Hver node inneholder et navn, samt en liste av navnene på alle nabonodene (merk at vi ikke har noe eksplisitt representasjon av kantene). Klassen inneholder en utskriftsfunksjon, **toString**() (se vedlegget). Anta at vi har bygd opp et nettverk, og resultatet av å skrive ut nettverket er som følger:

```
Sarpsborg:   Halden  Moss
Rakke:   Halden  Askim
Askim:   Moss  Rakke
Halden:   Fredrikstad  Sarpsborg  Rakke
Fredrikstad:   Halden  Moss
Moss:   Fredrikstad  Sarpsborg  Askim
```

A) *Forklar kort hvordan man foretar en* dybde*traversering av et nettverk. Skriv opp resultatet av å foreta et dybdesøk der navnet på hver node blir skrevet ut en og bare en gang, i* pre-order *rekkefølge, med start i* Moss*. Implementer så den* rekursive *metoden* `void` ***dybde***`()` *i klassen* ***Nettverk*** *i vedlegget.*

For hver node som besøkes, markerer vi at vi har vært der. Traverseringen foregår rekursivt ved å kalle traverseringen for all naboene til en node som ikke er besøkt.

Moss - Fredrikstad - Halden - Sarpsborg - Rakke - Askim

```
void dybde(String n) {
    Node node = noder.get(n);
    if (node == null) {
        //System.out.println(n + " finnes ikke!");
        return;
    }
    if (node.besokt) {
        //System.out.println(n + " er allerede besøkt");
        return;
    }

    node.besokt = true;

    for (String nabo : node.naboer) {
        dybde(nabo);
    }
}
```

B) *Forklar kort hvordan man foretar en* bredde*traversering av et nettverk. Skriv opp resultatet av å foreta et breddesøk der navnet på hver node blir skrevet ut en og bare en gang, med start i* Moss*. Implementer så metoden* `void` ***bredde***`()` *i klassen* ***Nettverk*** *i vedlegget.*

Traverseringen foregår ved hjelp av en kø. Vi starter med å legge rota inn i køen. Deretter fortsetter vi i en løkke helt til køen er tom. I hver iterasjon, tar vi ut første node i køen (og f.eks. skriver den ut). Deretter legger vi inn alle naboene til denne noden som ikke er besøkt. Hver gang vi legger inn en node i køen markerer vi den som besøkt.

Moss - Fredrikstad - Sarpsborg - Askim - Halden - Rakke

```
 void bredde(String n) {
    Node node = noder.get(n);
    if (node == null) {
        //System.out.println(n + " finnes ikke!");
        return;
    }

    LinkedList<Node> q = new LinkedList<Node>();
    q.add(node);
    node.besokt = true;

    while (!q.isEmpty()) {
        Node curr = q.remove();
        System.out.println(curr.name);
        for (String nabo : curr.naboer) {
            Node nb = noder.get(nabo);
            if (nb == null) {
                //System.out.println(nabo + " finnes ikke!");
            }
            else if (nb.besokt) {
                //System.out.println(nb.name + " er allerede besøkt");
            }
            else {
                q.add(nb);
                nb.besokt = true;
            }
        }
    }
}
```

Slutt, oppgave 5

Oppgave 6

Gitt følgende rekursive funksjon:

```
void mystisk(double x1, double x2, double y, double h,
             int t, int nivå, int max) {
    if (nivå > max) return;

    linje(x1, y, x1, y+h);
    linje(x2, y, x2, y+h);

    double d = (x2-x1)/t;
    double x = x1;

    for (int i = 0; i < t; i++) {
        mystisk(x, x+d,  y, h/2, t, nivå+1, max);
        x += d;
    }
 }
```

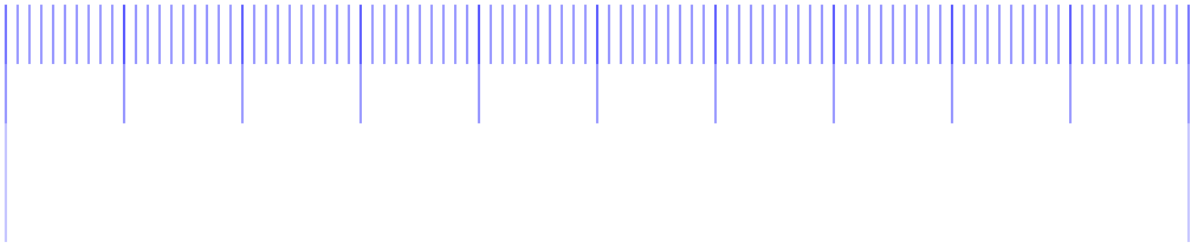
Metoden `linje(double x1, double y1, double x2, double y2)` tegner en linje på skjermen, fra koordinaten `(x1, y1)` til `(x2, y2)`.

A) Forklar hva metoden gjør, og tegn resultet av følgende kall:

```
mystisk(100, 1100, 100, 200, 10, 1, 3);
```

Ved repeterende mønster holder det med å antyde hva som skjer, og ikke tegne alle linjene.

I hvert rekursive kall tegnes to vertikale streker med høyde h, fra y-verdien y, og med x-verdier x1 og x1. Deretter kjøres et antall (i eksempelet: 10) rekursive kall som hver tegner nye streker med halve lengde...og avsluttes når vi har tegnet ønsket antall nivåer. Dette gir en tommestokk-liknende struktur:



Slutt, oppgave 6

Vedlegg

```
public class BinSøkeTre {
   // Rota i treet
    public Node rot;

    //Representasjon av en node
    public class Node {

        int verdi;
        Node venstre = null;
        Node hoyre = null;

        public Node(int v) {
            verdi = v;
        }
    }

   // Setter inn en node med verdi v
    // Metoden skal være iterativ
    public void settInn(int v) { ...skal implementeres i 3A... }
}
```

```
class GenBinTre {
    // Implisitt representasjon av et generelt binærtre der nodeverdiene er heltall
    int[] tre;

    int TOM = Integer.MAX_VALUE;

    void preOrder() {
        preOrder(0);
    }

    // Returnerer indeksen til venstre barn (TOM hvis det ikke eksisterer)
    int venstre(int forelder)  { ...skal implementeres i 4A...}

    // Returnerer indeksen til venstre barn (TOM hvis det ikke eksisterer)
    int hoyre(int forelder) { ...skal implementeres i 4A...}

    // Rekursiv traversering av treet med preorder utskrift av verdiene
    void preOrder(int node) { ...skal implementeres i 4B...}

}



class Nettverk {
    // Liste av alle nodene i nettverket
    HashMap<String, Node> noder = new HashMap<String, Node>();

    // Representasjon av en node
    class Node {
        // Liste av navnene til nabonodene
        LinkedList<String> naboer;
        String navn;
        boolean besokt = false;
    }

    public String toString() {
        String s = "";
        for (Node n : noder.values()) {
            s += n.navn + ": ";
            for (String nabo : n.naboer) {
                s += "  " +nabo;
            }
            s += "\n";
        }
        return s;
    }

    // Skriver ut navnene på alle nodene en og kun en gang
    // ved hjelp av en rekursiv dybdetraversering.
    // Nodene skrives ut i preorder rekkefølge.
    void dybde(String n) {...skal implementeres i 5A...}

    // Skriver ut navnene på alle nodene en og kun en gang
    // ved hjelp av en breddetraversering
    void bredde(String n) {...skal implementeres i 5B...}
}
```

Slutt, vedlegg
Slutt på oppgavesettet