Høgskolen i Østfold

EKSAMEN

Emnekode:	**Emne:**	
ITF20006	Algoritmer og datastrukturer	
Dato: 20. mai 2008	**Eksamenstid:** kl 09.00 til kl 13.00	
Hjelpemidler: 4 A4-sider (2 ark) med valgfritt innhold Kalkulator		**Faglærer:** Mari-Ann Akerjord og Jan Høiberg

Eksamensoppgaven:
Oppgavesettet består av 11 sider inklusiv vedlegg og denne forsiden.

Oppgavesettet består av 6 oppgaver, med tilsammen 12 deloppgaver. Alle oppgavene skal besvares. Ved sensur teller hver deloppgave omtrent like mye. Karakter fastsettes dog på basis av en helhetsvurdering av besvarelsen.

Kontroller at oppgaven er komplett før du begynner å besvare spørsmålene!

Sensurdato: 10. juni 2007

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

Oppgave 1

Vi har sett på forskjellige implementasjoner av lister. I javabiblioteket definerer grensesnittet List operasjonene som kan utføres på en liste. Utdrag av dette grensesnittet finnes i Vedlegg 1. To implementasjoner av dette grensesnittet er ArrayList og LinkedList, som implementerer List med henholdsvis tabell og dobbeltlenket liste som underliggende datastruktur.

Nedenfor ser du tre metoder som alle har grensesnitt-typen List som parameter. Det vil si at de kan kalles med lister av type ArrayList og LinkedList (og andre typer som implementerer grensesnittet List).

```
public static void byggListe(List<Integer> liste, int n) {
      liste.clear();
      for (int i = 0; i < n; i++) {
            liste.add(i);
      }
}
```

```
public static void skrivListe(List<Integer> liste) {
      for (int i = 0; i < liste.size(); i++) {
            System.out.print(liste.get(i) + " ");
     }
}
```

```
public static void fjernOddetall(List<Integer> liste) {
      int i = 0;
      while (i < liste.size()) {
            if (liste.get(i) % 2 != 0) {
                  liste.remove(i);
            }else {
                  i++;
            }
      }
}
```

a) Angi arbeidsmengde i O-notasjon som funksjon av listens lengde N for hver av de tre metodene når de kalles med lister av type:

1) ArrayList
2) LinkedList

Svarene må begrunnes!

b) På bakgrunn av svarene du ga under punkt a): Kunne noen av metodene vært implementert mer effektivt? Lag i så fall en alternativ implementasjon for hver metode du mener kan forbedres, og forklar hvorfor din versjon er mer effektiv enn den opprinnelige.

Slutt, oppgave 1

Oppgave 2

a) Forklar hvordan man bestemmer posisjonen til elementene i en hashtabell. Hva menes med en kollisjon? Forklar begrepene åpen og lukket adressering.

b) Sett inn verdiene

19, 28, 39, 38, 10, 49

i den rekkefølgen de står i en hashtabell med lengde 10 som i utgangspunktet er tom. Bruk hashfunksjonen h(x) = x mod 10 , åpen adressering med lineær prøving. Vis hvordan tabellen ser ut etter at alle verdiene er satt inn.

Slutt, oppgave 2

Oppgave 3

Nedenfor ser du grensesnittene StackADT og QueueADT:

```
public interface StackADT<T>
{
  public void push(T el);//Adds one element to the top of this stack
  public T pop(); //Removes and returns the top element
  public T peek();//Returns without removing the top element
  public boolean isEmpty();//Returns true if this stack is empty
  public int size();//Returns the number of elements
}

public interface QueueADT<T>
{
   public void enqueue (T el);//Adds one element to the rear of this queue
   public T dequeue();   //Removes and returns the front-element
   public T first();   //Returns  without removing the front-element
   public boolean isEmpty();   //Returns true if this queue is empty
   public int size();   // Returns the number of elements in this queue
}
```

a) ArrayStack og ArrayQueue er implementasjoner av henholdsvis StackADT og QueueADT.

Hva skrives til skjerm når kodelinjene nedenfor kjøres?

```
ArrayStack<Integer> s = new ArrayStack<Integer>();
ArrayQueue<Integer> q = new ArrayQueue<Integer>();
for (int i=1; i< 11;i++)
    s.push(i);
while(!s.isEmpty())
    q.enqueue(s.pop());
while(!q.isEmpty())
    s.push(q.dequeue());
while(!s.isEmpty())
    System.out.print(s.pop() + " ");
```

b) Lag en egen implementasjon av grensesnittet StackADT hvor du benytter en *kø* som intern datastruktur. Et utkast til en slik implementasjon er vist nedenfor:

```
public class QueueStack<T>  implements StackADT<T>{

    // Kø som lagrer elementene på stakken:
    private ArrayQueue<T> element_q;


    // Konstruktør og implementasjon av metodene fra
    // grensesnittet StackADT kommer her.
}
```

Slutt, oppgave 3

Oppgave 4

a) Følgende verdier skal settes inn, i gitt rekkefølge, i et vanlig binært søketre som til å begynne med er tomt:

```
342  206  444  523  607  301  142  183  102  157  149
```

Tegn en figur som viser hvordan søketreet ser ut etter at alle verdiene er satt inn.

b) De samme verdiene som i oppgave a) skal nå settes inn i gitt rekkefølge i et AVL-tre som til å begynne med er tomt. Tegn figurer som viser hvordan AVL-treet ser ut etter at:

i) Verdien `607` er satt inn.

ii) Verdien `157` er satt inn.

iii) Alle verdiene er satt inn.

Tegn også figurer som viser de rotasjonene som evt. må gjøres for å holde AVL-treet balansert.

c) Verdiene nedenfor skal settes inn, i gitt rekkefølge, i et B-tre av orden 4. Hver node, med mulig unntak for rotnoden, har høyst 4 verdier og minst 2 verdier. Til å begynne med er B-treet tomt.

```
659  767  702  157  728  102  461  899  920  44  774  264  384
344  973  905  999
```

Tegn figurer som viser hvordan B-treet ser ut etter at:

i) Verdien `728` er satt inn.

ii) Verdien `344` er satt inn.

iii) Alle verdiene er satt inn.

Slutt, oppgave 4

Oppgave 5

Vi har gitt følgende klasse, som implementerer innsetting av verdier og inorder traversering i AVL-trær, der dataene i hver node er et *positivt* heltall:

```
public class AVLTree
{
    public AVLTree()
    {
        // Constructor
        ...
    }

    public void insert(int value)
    {
        // Inserts new node with given value in AVL tree
        ...
    }

    public int firstInorder()
    {
        // Returns first value in inorder traversal
        ...
    }

    public int nextInorder()
    {
        // Returns next value in inorder traversal
        // Returns -1 when all nodes have been visited
        ...
    }
}
```

a) Programmér en metode:

```
void treeSort(int a[])
```

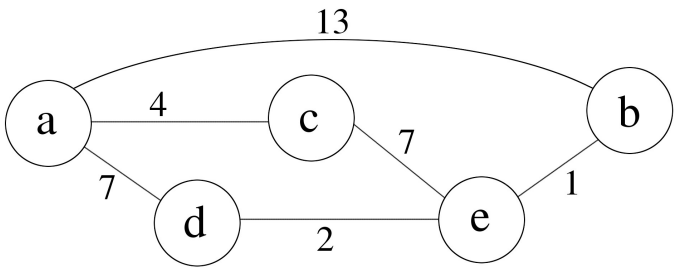
som sorterer tabellen `a`. Metoden `treeSort` skal utføre sorteringen ved først å flytte dataene i tabellen `a` over i et AVL-tre og deretter legge dem sortert tilbake. Metoden skal bruke metoder fra den gitte klassen `AVLTree`.

b) Hva er arbeidsmengden til metoden `treeSort`? Svaret må begrunnes. Bruk *O*-notasjon til å angi arbeidsmengden som en funksjon av tabellens lengde *n*.

Slutt, oppgave 5

Oppgave 6

Følgende urettede og vektede graf er gitt:



Dijkstra's algoritme skal brukes for å finne korteste vei fra noden **a** til alle de andre nodene.

Sett opp en tabell som viser forløpet av Dijkstra's algoritme på denne grafen. Hver rad i tabellen skal vise tilstanden etter en iterasjon av algoritmen, ved å angi:

- Hvilke noder som vi nå har funnet korteste vei til.
- Hvilke noder som vi har funnet en vei til, men ikke nødvendigvis den korteste.

Legg vekt på at tabellen du lager er enkel og oversiktlig.

Slutt, oppgave 6