Høgskolen i Østfold

EKSAMEN

Emnekode: ITF 20006	**Emne:** Algoritmer og datastrukturer
Dato: 16. mai 2006	**Eksamenstid:** kl. 0900 til kl. 1300
Hjelpemidler: 8 A4-sider (4 ark) med valgfritt innhold. Det er ikke tillatt å dele arkene inn i flere rader/kolonner.	**Faglærer:** Mari-Ann Akerjord

Eksamensoppgaven:

Oppgavesettet består av 10 sider inklusiv denne forsiden. I tillegg kommer vedleggene.

Oppgavesettet består av 3 oppgaver med henholdsvis 6, 4 og 3 deloppgaver. Alle oppgavene skal besvares. Ved sensur teller hver deloppgave omtrent like mye. Karakter fastsettes dog på basis av en helhetsvurdering av besvarelsen.

Vedlegg:

Oppgavesettet har to vedlegg:

- Vedlegg A, 3 sider
- Vedlegg B, 14 sider

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

Sensurdato: 6. juni 2006

Karakterene er tilgjengelige for studenter på studentweb senest dagen etter oppgitt sensurfrist. Følg instruksjoner gitt på:
http://www.hiof.no/index.php?ID=7027

Oppgave 1

Lønningskontoret i en bedrift vil lage et register over sine lønnstagere. Følgende data skal registreres for hver person:

- ID
- Etternavn
- Fornavn
- Fødselsår
- Inntekt

De bestemmer seg for å legge inn data for hver person i et binært søketre. Søketreet skal være sortert på personenes ID, som er unik.

Oppgave 1.1

Nedenfor ser du en liste med personer. Tegn treet du får når du legger disse personene inn - i den rekkefølgen de står - i et på forhånd tomt binært søketre. Søketreet skal være sortert etter personenes ID. Når du tegner treet trenger du kun å angi personenes ID i treets noder.

ID	***Etternavn***	***Fornavn***	***Fødselsår***	***Inntekt (kr)***
7	Tommysen	Nils	1978	153300
4	Arildsen	Trond	1969	189200
10	Franksen	Dag	1972	182800
2	Grodatter	Laila	1947	336100
12	Ivarsen	Øyvind	1979	970002
9	Trinedatter	Grete	1940	603600
8	Catrinedatter	Sissel	1982	1473000
14	Tonedatter	Eva	1946	241000
13	Jennydatter	Sigrid	1973	159100
1	Frodesen	Frank	1970	144000
3	Johannesen	Kåre	1963	208500
5	Torilldatter	Grete	1944	283500
6	Thomassen	Geir	1952	133500
11	Alfsen	Oddvar	1986	1001300

Oppgave 1.2

Skriv nodene i treet fra forrige deloppgave i inorden rekkefølge. Du trenger kun å skrive personenes ID. Denne oppgaven skal altså utføres manuelt, det vil si du skal ikke skrive noen programkode.

De tre neste deloppgavene går ut på å implementere metoder for å utføre operasjoner på søketreet. I vedlegg A finner du klassen *Person,* samt utkast til en klasse *PersonSoketre*. Klassen *Person* representerer en person. Klassen *PersonSoketre* representer selve treet, og inneholder blant annet en privat klasse *Node* som representerer en node i søketreet. En node har i tillegg til data om en person referanser til nodens barn i treet.

Metodene som skal lages i de neste tre deloppgavene er plassert i klassen *PersonSoketre.*

Oppgave 1.3

Skriv metoden

public Node settInn(Person ny, Node rot)

som setter inn person-objektet *ny* på riktig plass i det binære søketreet som har rot i noden *rot*. Metoden skal returnere roten i treet. Du kan selv velge om du vil implementere metoden *settInn* rekursivt eller iterativt.

Oppgave 1.4

Skriv metoden

public Node finnPerson(int id, Node rot)

som finner og returnerer noden som inneholder person-objektet med ID = *id* i det binære søketreet som har rot i noden *rot*. Hvis det ikke finnes en person med angitt ID i treet, returneres *null*. Velg selv om du vil implementere metoden *finnPerson* rekursivt eller iterativt.

Oppgave 1.5

Skriv metoden

public void skrivMillionInntekter(Node rot)

som skriver til skjerm alle personer som har inntekt større enn en million. Utskriften skal være sortert på ID. Parameteren *rot* er roten i søketreet.

Oppgave 1.6

Anta at det er N noder i søketreet. Hva er arbeidsmengden, uttrykt med O-notasjon, til hver av metodene fra oppgavene 1.3, 1.4 og 1.5?

Hvis du ikke har fått til en eller flere av disse deloppgavene, kan du likevel besvare denne oppgaven ut fra din kjennskap til binære søketrær.

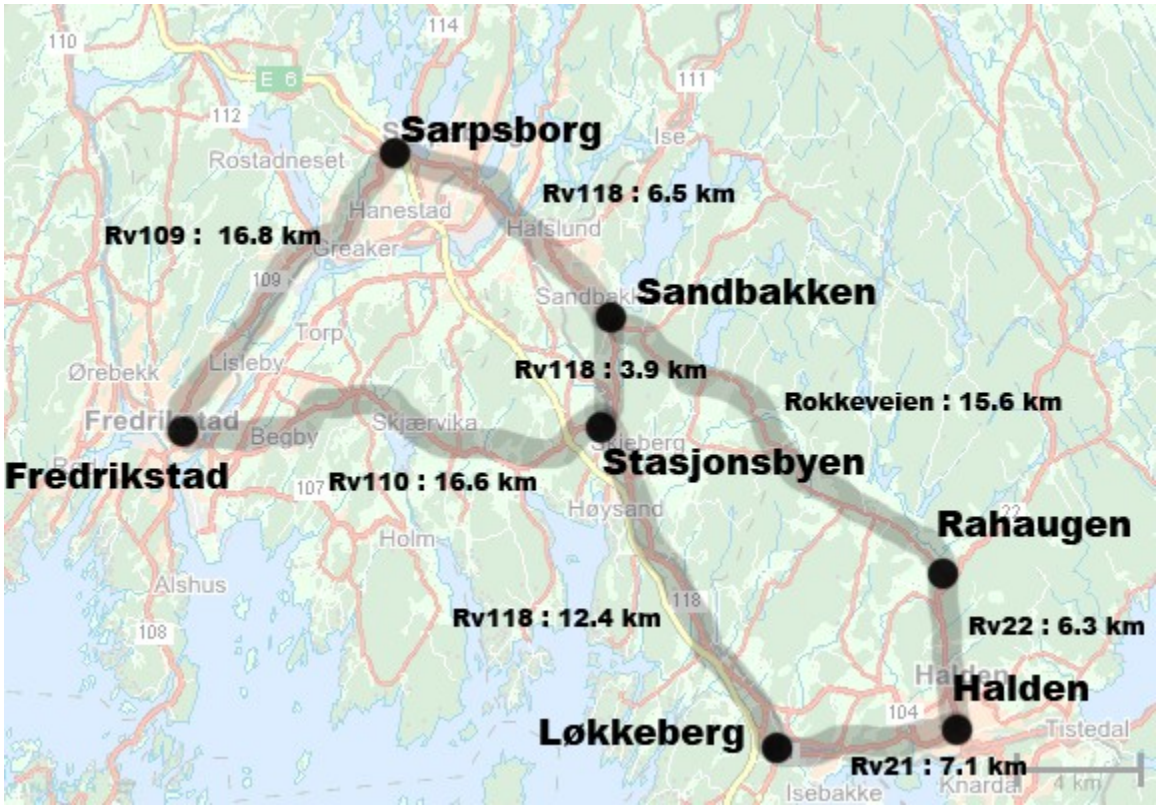
Slutt oppgave 1

Oppgave 2

Et vanlig bruksområde for grafer er å finne ruter mellom posisjoner. I denne oppgaven er et system for å angi veier og knutepunkter angitt. Et knutepunkt er ett sted der en vei deler seg i flere veier (eller flere veier møtes). En vei angis av veinavn, lengde, start og slutt. Merk at en vei her egnetlig er en etappe på en vei. Det vil si at for eksempel E6 vil bli modellert av flere veier, der alle har navn E6, og går mellom ulike knutepunkter.

Oppgave 2.1

Nedenfor ser du et utsnitt av et kart hvor markerte veier og knutepunkter tilsammen danner en graf. Bruk Dijkstras algoritme på denne grafen til å finne korteste vei mellom Sarpsborg og Halden med hensyn på avstand. Denne oppgaven skal ikke kodes, men du skal manuelt vise hvordan algoritmen virker.



Studer klassene *Vei* og *Knutepunkt* som ligger i vedlegg B. Under brukes disse klassene til å sette opp grafen fra veikartet over.

```
Knutepunkt sarpsborg = new Knutepunkt(”Sarpsborg”);
Knutepunkt lokkeberg = new Knutepunkt("Løkkeberg");
Knutepunkt sandbakken = new Knutepunkt("Sandbakken");
Knutepunkt rahaugen = new Knutepunkt("Rahaugen");
Knutepunkt halden = new Knutepunkt("Halden");
Knutepunkt fredrikstad = new Knutepunkt("Fredrikstad");
Knutepunkt stasjonsbyen = new Knutepunkt("Stasjonsbyen");

sarpsborg.settOppForbindelse("rv118",6.5,sandbakken);
sandbakken.settOppForbindelse ("rv118",3.9,stasjonsbyen);
sarpsborg.settOppForbindelse ("rv109",16.8,fredrikstad);
fredrikstad.settOppForbindelse ("rv110",16.6,stasjonsbyen);
stasjonsbyen.settOppForbindelse ("rv118",12.4,lokkeberg);
lokkeberg.settOppForbindelse ("rv21",7.1,halden);
sandbakken.settOppForbindelse ("rokkeveien",15.6,rahaugen);
rahaugen.settOppForbindelse ("rv22",6.3,halden);
```

Oppgave 2.2

Metoden *hentVeier()* i klassen *Knutepunkt* returnerer en *VeiIterator,* som skal kunne brukes til å iterere over veiene fra et knutepunkt. Så langt finnes imidlertid ikke klassen *VeiIterator*. Lag klassen *VeiIterator* slik at den implementerer interfacet *Iterator.* Javadoc til interfacet *Iterator* og til klassen *ArrayList* ligger i vedlegg B. Du skal **ikke** bruke *ArrayList* sin innebyggede iterator.

Du trenger ikke implementere metoden *remove(),* men kan la metoden kaste et unntak av typen *UnsupportedOperationException*.

De to neste deloppgavene går ut på å implementere metoder som bruker en ferdig oppsatt graf til å finne bestemte relasjoner mellom knutepunkter. Du kan anta at oppsett av graf og metodene som er beskrevet nedenfor er samlet i en egen klasse.

Oppgave 2.3

Skriv en metode naboKnutepunkt,

public static void naboKnutepunkt(Knutepunkt k)

som finner og skriver ut alle naboer til ett knutepunkt. For hver nabo skal knutepunktets navn, samt navn på vei frem til dette knutepunktet og avstand i kilometer skrives til skjerm.

Brukt på grafen i eksemplet over, skal metodekallet

naboKnutepunkt(sarpsborg) ;

gi følgende utskrift til skjerm:

Sandbakken: rv118, 6.5 km
Fredrikstad: rv109, 16.8 km

Bruk iteratoren fra oppgave 2.2. Hvis du ikke fikk til den deloppgaven, kan du likevel anta at du har en fungerende iterator. Legg ellers merke til metoden *motsattEnde(Knutepunkt k),* som befinner seg i klassen *Vei*.

Oppgave 2.4

Lag en metode som skriver ut alle mulige destinasjoner fra et knutepunkt, det vil si hvilke andre knutepunkter det er mulig å komme til fra et gitt knutepunkt.

public static void muligeDestinasjoner(Knutepunkt k)

Du trenger kun å skrive ut navnet på hver destinasjon. Brukt på grafen i eksemplet over, kan metodekallet

muligeDestinasjoner(sarpsborg);

gi følgende utskrift til skjerm:

Sandbakken
Stasjonsbyen
Fredrikstad
Løkkeberg
Halden
Rahaugen

Velg selv om du vil løse oppgaven rekursivt eller ikke-rekursivt. Velger du en ikke-rekursiv løsning kan du for eksempel benytte en kø. Du finner javadoc til en kø-implementasjon i vedlegg B.

For en rekursiv løsning er et tips å lage en annen metode til bruk i drivermetoden over.

private static void muligeDestinasjoner(Knutepunkt k, ArrayList<Knutepunkt> besokt)

Slutt oppgave 2

Oppgave 3

Oppgave 3.1

Under er det beskrevet noen operasjoner på ulike datastrukturer. For hver av dem skal du angi arbeidsmengden i O-notasjon. Begrunn svarene og gjør rede for eventuelle antakelser og presiseringer.

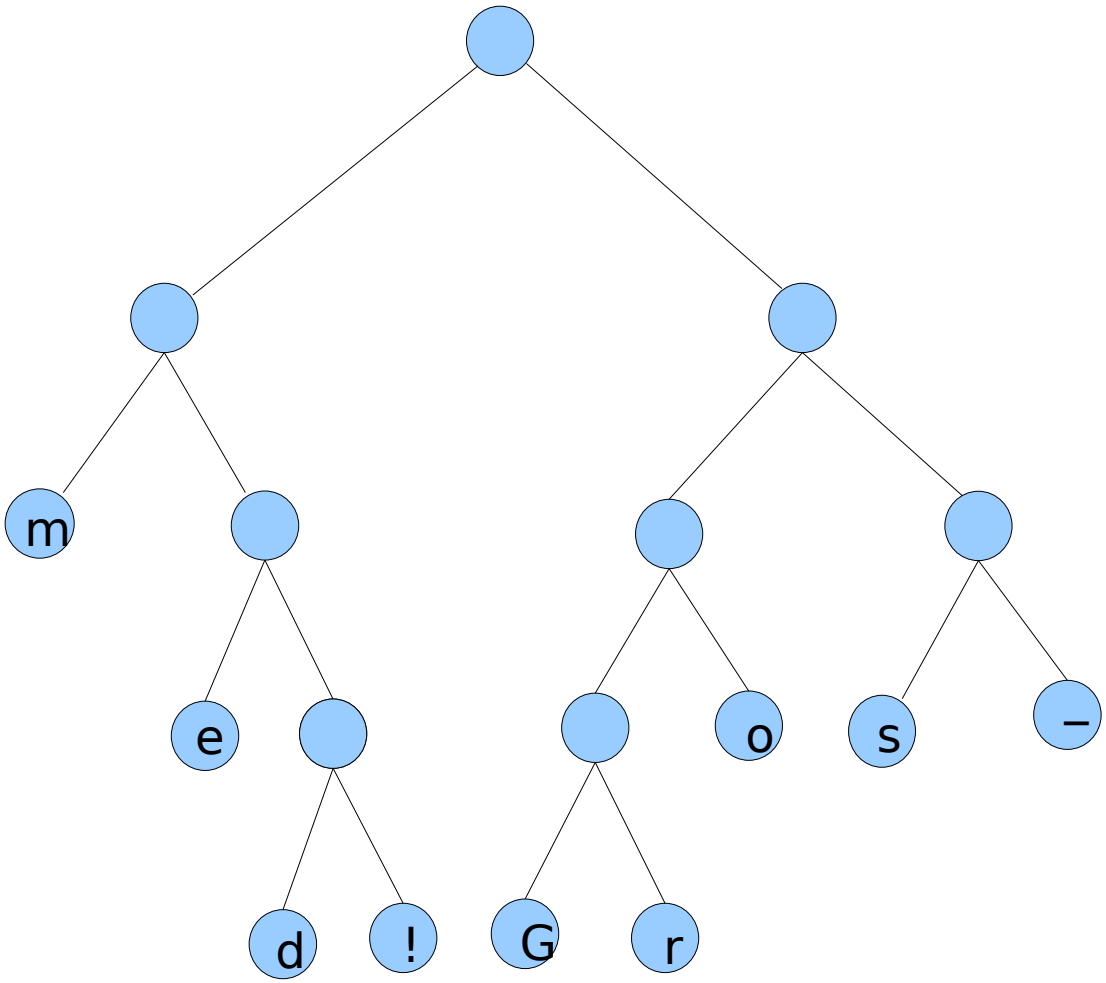
A) Fjerne øverste elementet på en stakk.
B) Skrive ut alle elementene som ligger på en stakk slik at det øverste elementet kommer først.
C) Skrive ut alle elementene som ligger på en stakk slik at det nederste elementet kommer først.
D) Skrive ut det femte siste elementet i en tabell.
E) Skrive ut det femte siste elementet i en enkeltlenket liste.
F) Finne det minste elementet i et binært søketre.
G) Sette inn et nytt element på begynnelsen av en lenket liste.
H) Fjerne elementet som ligger i begynnelsen (indeks 0) i en tabell.

Oppgave 3.2

A) Forklar kort hva en prioritetskø er, og gi ett eksempel på anvendelse.
B) Gitt at du skal implementere en prioritetskø. Hvilke fordeler og ulemper ser du ved å benytte følgende datastrukturer som intern struktur:
    1. Lenket liste
    2. Sortert lenket liste
    3. Binært søketre
C) Ofte benyttes en såkalt minimumsheap til å implementere prioritetskøer. Forklar kort hva en minimumsheap er, og hvorfor nettopp denne strukturen er gunstig med hensyn på implementasjon av prioritetskøer.

Oppgave 3.3

Nedenfor ser du et eksempel på et Huffman-tre. Bruk dette til å dekode følgende bit-sekvens:
10001010110111110101000001010010111
Forklar kort hvordan du gikk frem.



Slutt oppgave 3

Vedlegg A

Vedlegg til oppgave 1:

1. Klassen Person
2. Klassen PersonSoketre

```
public class Person implements Comparable<Person>
{
	private int id;
	private String etternavn;
	private String fornavn;
	private int fodselsaar;
	private int inntekt;

	public Person(	int id,
				String etternavn,
				String fornavn,
				int fodselsaar,
				int inntekt)
	{
		this.id = id;
		this.etternavn = etternavn;
		this.fornavn = fornavn;
		this.fodselsaar = fodselsaar;
		this.inntekt = inntekt;
	}

	public int compareTo(Person p)
	{
		return id-p.id;
	}

	public int hentId() { return id; }
```

```
	public int hentInntekt() { return inntekt;}

	public String toString()
	{
		return
			id +", " + etternavn +", " + fornavn + ", " +
			fodselsaar + ", " + inntekt;
	}
}// Slutt class Person

public class PersonSokeTre
{
	private Node rot;

	// Oppgave 1.3
	public Node settInn(Person ny, Node rot)
	{
		// Din kode her.
	}

	// Oppgave 1.4
	public Node finnPerson(int id, Node rot)
	{
		// Din kode her.
	}

	// Oppgave 1.5
	public void skrivMillionInntekter(Node rot)
	{
		// Din kode her
	}

	private  class Node
	{
		private  Person data; // Nodens data
		private  Node venstre; // Venstre barn
```

```
            private  Node hoyre; // Hoyre barn

            public Node(Person p, Node v, Node h)
            {
                  data = p;
                  venstre = v;
                  hoyre = h;
            }

      } // Slutt class Node


} // Slutt class PersonSoketre
```

Vedlegg B

Vedlegg til oppgave 2:

1. Klassen Vei
2. Klassen Knutepunkt
3. Javadoc til interfacet Iterator
4. Javadoc til klassen ArrayList
5. Javadoc til klassen ArrayQueue (kø-implementasjon)

```
public class Vei
{
	private Knutepunkt k1 = null;
	private Knutepunkt k2 = null;
	private String veinavn = "";
	private double avstand = 0.0;

	public Vei(String veinavn,double avstand,Knutepunkt k1,Knutepunkt
k2)
	{
		this.veinavn=veinavn;
		this.avstand=avstand;
		this.k1=k1;
		this.k2=k2;
	}

	public Knutepunkt motsattEnde(Knutepunkt k)
	{
		return (k==k1?k2:k1);
	}

	public double hentAvstand()
	{
		return avstand;
	}
```

```
	public String hentNavn()
	{
		return veinavn;
	}

} // Slutt class Vei
```

```
public class Knutepunkt
{
	private ArrayList<Vei> veier = new ArrayList<Vei>();
	private String knutepunktnavn = "";

	public Knutepunkt(String navn)
	{
		knutepunktnavn = navn;
	}

	private void leggTilVei(Vei v)
	{
		veier.add(v);
	}

	public void settOppForbindelse(String veinavn,double km,
Knutepunkt maal)
	{
		Vei v = new Vei(veinavn,km,this,maal);
		this.leggTilVei(v);
		maal.leggTilVei(v);
	}

	public Iterator<Vei> hentVeier()
	{
```

```
            return new VeiIterator();
      }



      public String hentNavn()
      {
            return knutepunktnavn;
      }

} // Slutt class Knutepunkt
```