EKSAMEN

5610 ALGORITMAR OG DATASTRUKTURAR

15.05.2009

Tid:	9-14 (5 timar)
Målform:	Bokmål/Nynorsk
Sidetal:	11 (forside + 5 + 5)
Hjelpemiddel:	Alle trykte og skrivne
Merknader:	Ingen
Vedlegg:	Ingen

Eksamensresultata blir offentliggjort på nettet, via Arena høgskole. I tillegg finn du eksamensresultatlister på utsida av eksamenskontoret, men da treng du kandidatnummeret ditt, så du bør notere dette på ein lapp og legge den i lommeboka.

Avdeling for allmennvitskaplege fag.

Råd og retningslinjer. Les oppgaveteksten godt før du går i gang med å løse oppgava. Deloppgavene er uavhengige av hverandre i den forstand at om du ikke får til en oppgave, kan du likevel gjøre neste, som om den første var løst. Fordél tida godt på alle oppgavene, alle 10 deloppgaver teller likt. Om du mener en oppgave er upresis, så skriv din egen presisering.

Oppgave 1 Algoritmeanalyse (10%)

Oppgi hvilken orden de følgende algoritmer har (i tid) som funksjon av n, med en kort begrunnelse/forklaring.

Algoritme 1:
```
public int it1(int n){
  int sum=0;
  for (int i=0; i<n; i+=2) sum++;
  return sum;
}
```

Algoritme 2:
```
public int it2(int n){
  int sum=0;
  for (int i=0; i<n; i++)
    for (int j=-i; j<i; j++) sum++;
  return sum;
}
```

Algoritme 3:
```
public int it3(int n){
  int sum=0;
  for (int i=0; i<n*n; i++)
    for (int j=1; j<i; j*=2) sum++;
  return sum;
}
```

Algoritme 4:
```
public int rek1(int n){
  if (n<=1) return 1;
  else return rek1(n-1)+1;
}
```

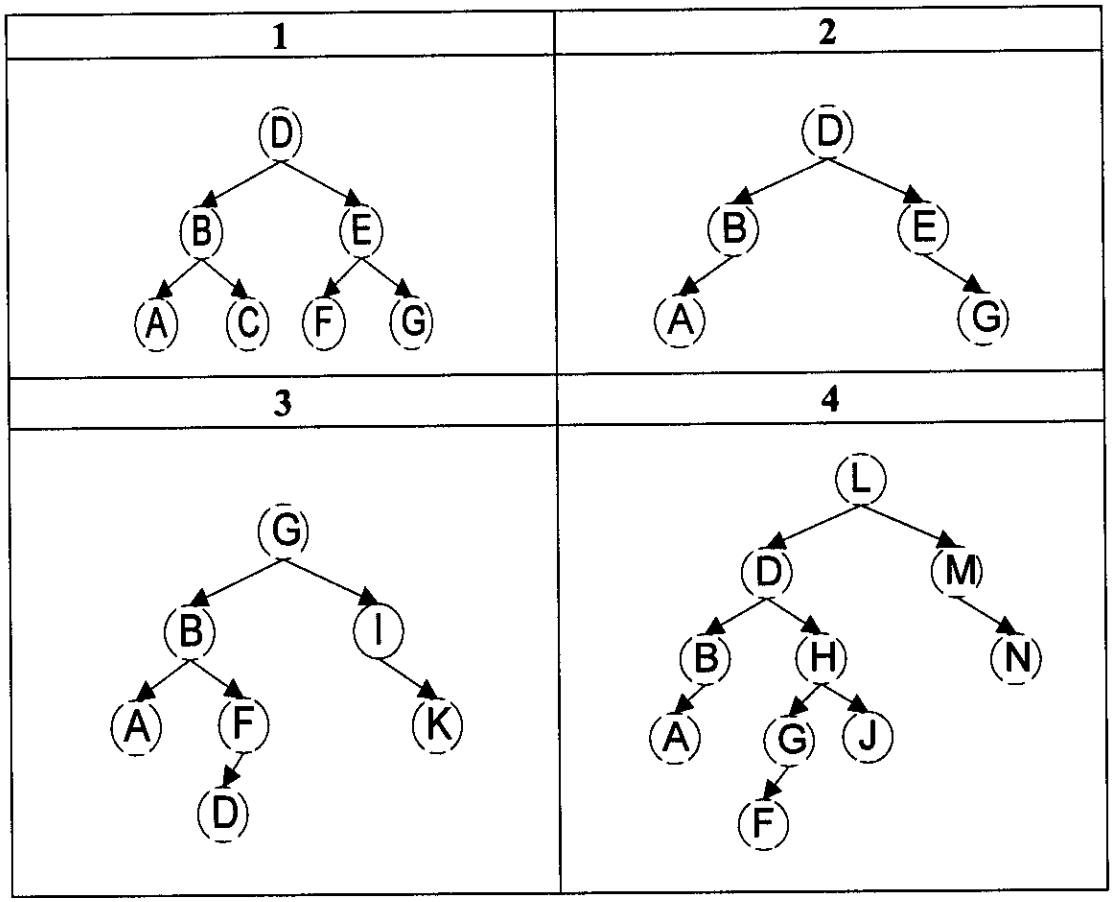
Algoritme 5:
```
public int rek2(int n){
  if (n<=1) return 1;
  else return rek2(n/2)+1;
}
```

Algoritme 6:
```
public int rek3(int n){
  if (n<=1) return 1;
  else return rek3(n/2) + rek3(n/2);
}
```

Oppgave 2 Binærtre (40 %)

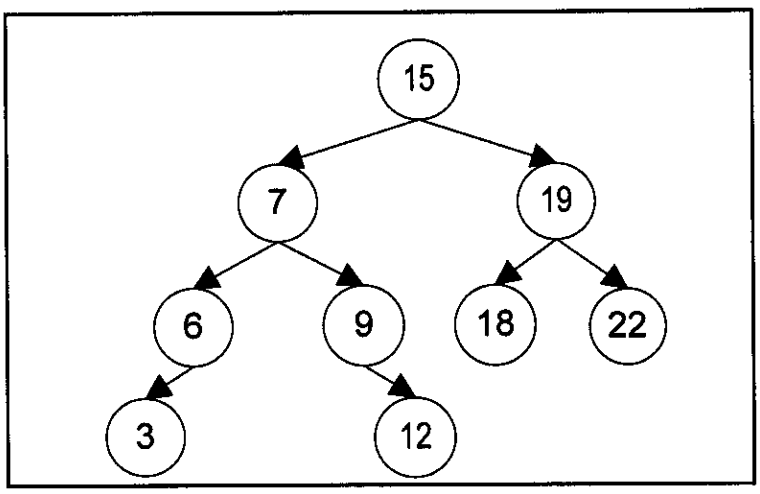
2a)

Se på følgende fire binærtre. Hvilke av disse er søketrær? Hvilke av disse er AVL-trær? Begrunn svaret.



2b)

Se på følgende AVL-tre. Sett inn verdiene 14, 4 og 10 (i den rekkefølge) i treet, og tegn treet for hver innsetting. Du skal altså tegne tre figurer, og siste figur skal ha 12 noder.



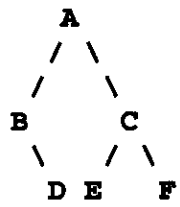
2c)

Ta nå utgangspunkt i klassene BinærTre og BinærNode:

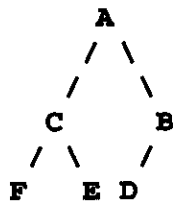
```
public class BinærTre<E>{
  protected BinærNode<E> rot;
  // evt. diverse metoder
}

class BinærNode<E>{
  BinærNode<E> venstre, høyre;
  E element;
  // evt. diverse metoder
}
```

Vi har følgende binærtre:



Etter en *speiling* ser treet ut slik:



En speiling medfører altså at for enhver node i treet vil venstre og høyre subtre bytte plass.

Lag metoden
`public void speil()`
i class BinærTre. Et kall på speil skal resultere i at treet speiles. Du skal bruke rekursjon, lag gjerne hjelpemetoder i class BinærTre og/eller i class BinærNode.

2d)

Også her skal vi speile et binærtre, men det skal skje i en *kopikonstruktør*, og du *skal ikke bruke rekursjon.* Lag metoden

```
public Binærtre<E> speilKopi()
```

i class Binærtre. Den skal opprette og returnere et nytt tre der alle nodeobjektene er speilede kopier av de tilsvarende nodene i originalen. Elementpekerne skal peke til de opprinnelige objektene, de skal altså ikke kopieres. Hint: traverser i nivåorden rekkefølge og bruk en kø. Lag gjerne hjelpemetoder i class BinærTre og/eller class BinærNode.

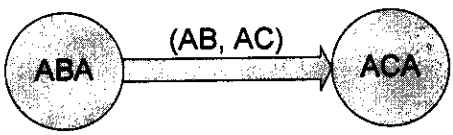
Oppgave 3 (50%) Grafer

Vi skal i denne oppgaven se på et *substitusjonssystem*. Slike systemer kan bl.a. brukes til å gjennomføre bevis, og påvise at en setning er en del av et språk.

Produksjoner er en sentral del av systemet. En produksjon er representert ved to tekster, og betyr at dersom en setning inneholder den første teksten, *venstresiden*, kan den biten byttes ut med den andre, *høyresiden*. Eksempelvis kan produksjonen (AB, AC) omforme setningen "ABA" til "ACA".

Dersom vi vil vise at en setning kan omformes til en annen setning, må vi vise at det finnes en sekvens av omforminger (anvendelser av produksjoner) som gjør at vi ender opp med det resultatet vi ønsker. Det kan tenkes å være flere slike sekvenser som gir samme resultat, vi ønsker da den korteste.

Man kan illustrere substitusjonssystemet som en graf. De produserbare setningene blir til nodene i grafen, og hver produksjon som kan anvendes på en setning vil bli til en kant fra tilsvarende node. Eksempelet over vil bli til følgende graf:



3a)

Ta nå utgangspunkt i setningen "ABA", og følgende produksjoner:

1. (AB, A)
2. (AB, BA)
3. (BA, B)

Tegn denne grafen. Den vil være endelig (ha et endelig antall noder og kanter), det er ikke generelt tilfelle. Merk nodene med setning og merk kantene med nummeret på produksjonen, 1.-3. Nodene skal være unike, dvs. det skal ikke finnes to noder med samme setning, men det kan være flere kanter til en node. Dette medfører at vi generelt ikke vil få en trestruktur.

3b)

Klassen Produksjon skal holde på de to tekstene i produksjonen. Den skal ha equals-metode og hashCode-metode. For at to produksjoner skal defineres som like må de ha like venstresider og like høyresider. Produksjoner skal også være *sammenlignbare* (Comparable) med hensyn på venstresidene. Lag klassen Produksjon.

3c)

Du skal her lage en metode som prøver å anvende en bestemt produksjon på en setning. Resultatet skal være en liste av alle setninger som kan produseres. Merk at generelt kan en produksjon anvendes flere steder i en setning, eksempelvis kan produksjonen (AB, B) omforme "ABAB" både til "BAB" og til "ABB". Metodens signatur:

```
public static List<String> produser(String setn, Produksjon p)
```

Du kan ha nytte av denne metoden i class String:
int indexOf(String str, int fromIndex)

> Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. If no such index exists, then -1 is returned.

3d)

Her skal vi endelig undersøke om vi kan komme fra en bestemt startsetning til en sluttsetning ved hjelp av en mengde produksjoner. Vi antar her at den tenkte grafen (jmf oppgave 3a) er endelig. Metoden skal ha følgende signatur

```
public static boolean kanProduseres(String start, String slutt,
  List<Produksjon> produksjoner)
```

Den skal altså svare true dersom sluttsetningen kan produseres, false ellers. Gjør helst et bredde-først-søk slik at metoden blir effektiv.

3e)

I tillegg til å få vite *om* en setning lar seg produsere, kan vi være interessert i hvordan den *enklest* lar seg produsere, dvs. med kortest "produksjonslinje". Vis de endringer som må til i koden for at metoden i oppgave 3d skal klare å rapportere sekvensen av setninger man er innom og produksjonene som anvendes for å komme fram. I denne oppgaven vil også en god forklaring/skisse gi god – men ikke full – uttelling.

Lykke til!

Råd og retningsliner. Les oppgåveteksten godt før du går i gang med å løyse oppgåva. Deloppgåvene er uavhengige av kvarandre i den meining at om du ikkje får til ei oppgåve, kan du likevel gjere neste, som om den fyrste var løyst. Fordél tida godt på alle oppgåvene, alle 10 deloppgåver tel likt. Om du meiner ei oppgåve er upresis, så skriv din eigen presisering.

Oppgåve 1 Algoritmeanalyse (10%)

Oppgje kva for orden dei fylgjande algoritmar har (i tid) som funksjon av n, med ei kort grunngjeving/forklaring.

Algoritme 1:
```
public int it1(int n){
  int sum=0;
  for (int i=0; i<n; i+=2) sum++;
  return sum;
}
```

Algoritme 2:
```
public int it2(int n){
  int sum=0;
  for (int i=0; i<n; i++)
    for (int j=-i; j<i; j++) sum++;
  return sum;
}
```

Algoritme 3:
```
public int it3(int n){
  int sum=0;
  for (int i=0; i<n*n; i++)
    for (int j=1; j<i; j*=2) sum++;
  return sum;
}
```

Algoritme 4:
```
public int rek1(int n){
  if (n<=1) return 1;
  else return rek1(n-1)+1;
}
```

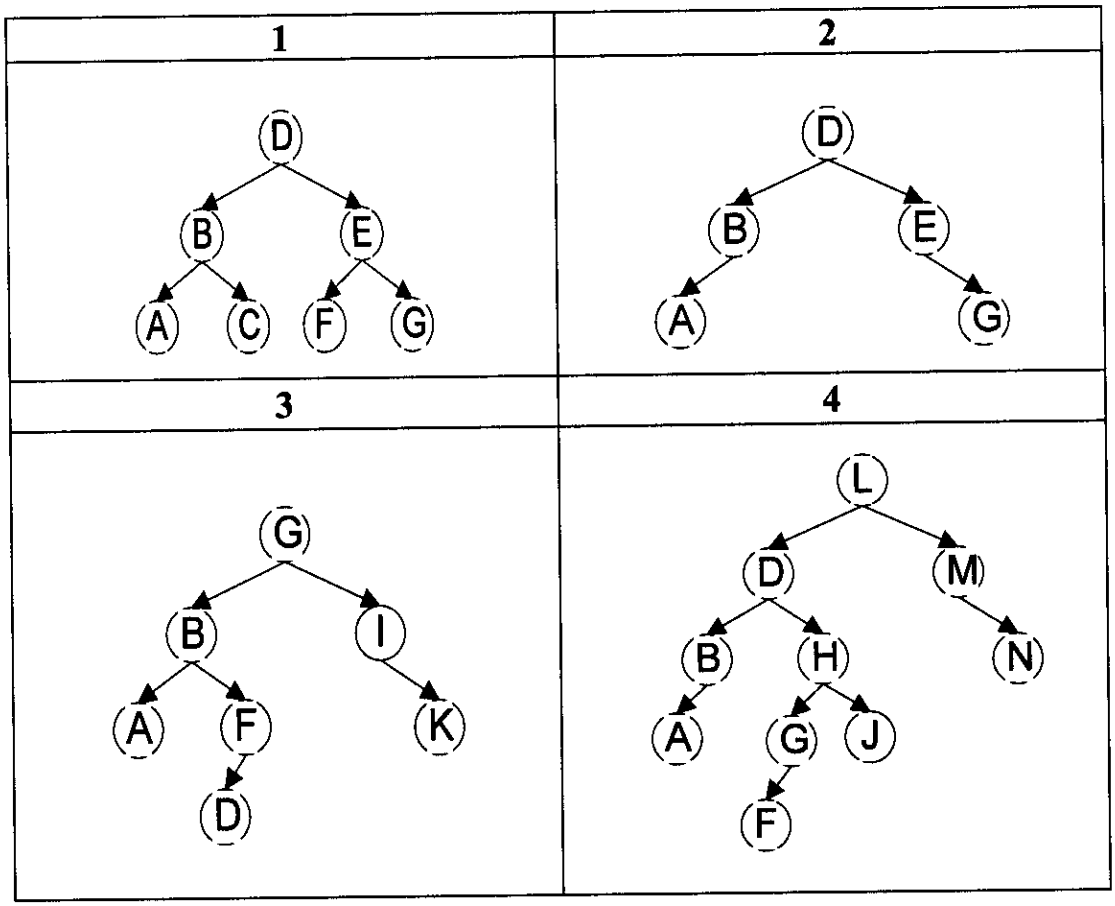
Algoritme 5:
```
public int rek2(int n){
  if (n<=1) return 1;
  else return rek2(n/2)+1;
}
```

Algoritme 6:
```
public int rek3(int n){
  if (n<=1) return 1;
  else return rek3(n/2) + rek3(n/2);
}
```

Oppgåve 2 Binærtre (40 %)

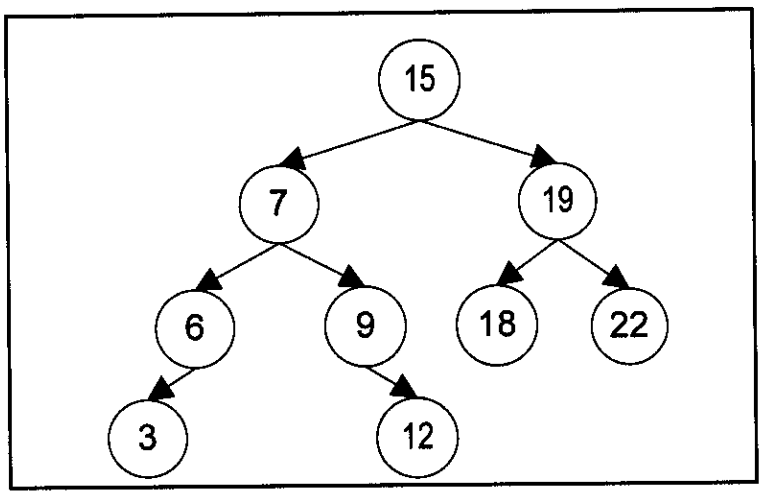
2a)

Sjå på fylgjande fire binærtre. Kva for nokre av desse er søketre? Kva for nokre av desse er AVL-tre? Grunngje svaret.



2b)

Sjå på fylgjande AVL-tre. Set inn verdiane 14, 4 og 10 (i den rekkefylgje) i treet, og teikn treet for kvar innsetting. Du skal altså teikne tre figurar, og siste figur skal ha 12 noder.



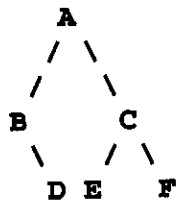
2c)

Ta nå utgangspunkt i klassene BinærTre og BinærNode:

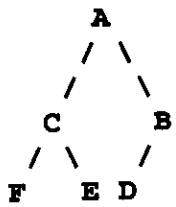
```
public class BinærTre<E>{
  protected BinærNode<E> rot;
  // evt. diverse metodar
}

class BinærNode<E>{
  BinærNode<E> venstre, høgre;
  E element;
  // evt. diverse metodar
}
```

Vi har fylgjande binærtre:



Etter ei *spegling* ser treet ut slik:



Ei spegling medfører altså at for kvar node i treet vil venstre og høgre subtre bytte plass.

Lag metoden
`public void speg1()`
i class BinærTre. Eit kall på spegl skal resultere i at treet speglast. Du skal bruke rekursjon, lag gjerne hjelpemetodar i class BinærTre og/eller i class BinærNode.

2d)

Også her skal vi spegle eit binærtre, men det skal skje i en *kopikonstruktør*, og du *skal ikkje bruke rekursjon.* Lag metoden

```
public Binærtre<E> speglKopi()
```

i class Binærtre. Den skal opprette og returnere eit nytt tre der alle nodeobjekta er spegla kopiar av dei tilsvarande nodane i originalen. Elementpeikarane skal peike til dei opphavelege objekta, dei skal altså ikkje kopierast. Hint: traverser i nivåorden rekkefylgje og bruk ein kø. Lag gjerne hjelpemetodar i class BinærTre og/eller class BinærNode.

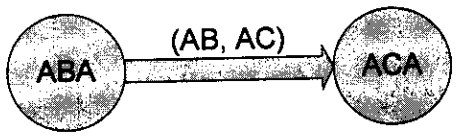
Oppgåve 3 (50%) Grafar

Vi skal i denne oppgåva sjå på eit *substitusjonssystem*. Slike system kan bl.a. nyttast til å gjennomføre bevis, og påvise at ei setning er ein del av eit språk.

Produksjonar er ein sentral del av systemet. Ein produksjon er representert ved to tekstar, og tyder at dersom ei setning inneheld den fyrste teksten, *venstresida*, kan den biten bytast ut med den andre, *høgresida*. Til dømes kan produksjonen (AB, AC) omforme setninga "ABA" til "ACA".

Dersom vi vil vise at ei setning kan omformast til ei anna setning, må vi vise at det finnast ein sekvens av omformingar (anvendingar av produksjonar) som gjer at vi ender opp med det resultatet vi ynskjer. Det kan tenkast å vere fleire slike sekvensar som gjev same resultat, vi ynskjer då den kortaste.

Ein kan illustrere substitusjonssystemet som ein graf. Dei produserbare setningane blir til nodane i grafen, og kvar produksjon som kan anvendast på ei setning vert til ein kant frå tilsvarande node. Dømet over vert til fylgjande graf:



3a)

Ta nå utgangspunkt i setninga "ABA", og fylgjande produksjonar:

1. (AB, A)
2. (AB, BA)
3. (BA, B)

Teikn denne grafen. Den vil vere endelig (ha eit endelig tal på nodar og kantar), det er ikkje generelt tilfelle. Merk nodane med setning og merk kantane med nummeret på produksjonen, 1..3. Nodane skal vere unike, dvs. det skal ikkje finnast to nodar med same setning, men det kan vere fleire kantar til ein node. Dette medfører at vi generelt ikkje vil få ein trestruktur.

3b)

Klassa Produksjon skal halde på dei to tekstane i produksjonen. Den skal ha equals-metode og hashCode-metode. For at to produksjonar skal definerast som like må dei ha like venstresider og like høgresider. Produksjonar skal også vere *samanliknbare* (Comparable) med omsyn på venstresidene. Lag klassa Produksjon.

3c)

Du skal her lage ein metode som prøver å anvende ein bestemt produksjon på ei setning. Resultatet skal vere ei liste av alle setningar som kan produserast. Merk at generelt kan ein produksjon anvendast fleire stader i ei setning, til dømes kan produksjonen (AB, B) omforme "ABAB" både til "BAB" og til "ABB". Signaturen til metoden:

```
public static List<String> produser(String setn, Produksjon p)
```

Du kan ha nytte av denne metoden i class String:
int indexOf(String str, int fromIndex)

Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. If no such index exists, then -1 is returned.

3d)

Her skal vi endeleg undersøke om vi kan kome frå ei bestemt startsetning til ei sluttsetning ved hjelp av ei mengde produksjonar. Vi antek her at den tenkte grafen (jmf oppgåve 3a) er endeleg. Metoden skal ha fylgjande signatur

```
public static boolean kanProduserast(String start,
  String slutt, List<Produksjon> produksjonar)
```

Den skal altså svare true dersom sluttsetninga kan produserast, false elles. Gjer helst eit bredde–først–søk slik at metoden blir effektiv.

3e)

I tillegg til å få vite *om* ei setning lar seg produsere, kan vi vere interessert i korleis den *enklast* lar seg produsere, dvs. med kortast "produksjonsline". Vis dei endringar som må til i koden for at metoden i oppgåve 3d skal klare å rapportere sekvensen av setningar ein er innom og produksjonane som anvendast for å kome fram. I denne oppgåva vil også ei god forklaring/skisse gje god – men ikkje full – uttelling.

Lykke til!