UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i	IN 115 — Algoritmer og datastrukturer
Eksamensdag:	26. mai 2001
Tid for eksamen:	9.00–15.00
Oppgavesettet er på 7 sider.	
Vedlegg:	Ingen
Tillatte hjelpemidler:	Alle trykte og skrevne

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

Merk:

Alle programmer skal skrives i Java. Hvis du har godkjente obligatoriske oppgaver fra tidligere, kan du besvare oppgaven i Simula, selv om vi anbefaler at du også da benytter Java. Du skal ikke gi noen fullstendig dokumentasjon av programmene, men du kan skrive noen få linjer som gir leseren nøkkelen til forståelse av programmet. Tegn gjerne figurer hvis dette gjør det lettere å forstå programmet. Du kan anta at leseren kjenner problemstillingen i oppgaven meget godt.

Alle steder der det er spørsmål etter et program (eller en programbit) skal du skrive dette helt ut, og *ikke* bare henvise til liknende programmer f.eks. i læreboka. Imidlertid både kan og bør du bruke abstrakte datatyper fra pensum i programmene dine uten å implementere disse på nytt. Hvis du f.eks. bruker en kø i løsningen av et problem, trenger du ikke gi fullstendig kode for køen. Det er tilstrekkelig å kort definere køoperasjonene samt skissere datastrukturen som du ville brukt i en implementasjon.

Les oppgavene nøye, og lykke til!

Ragnhild Kobro Runde og Arild Waaler

(Fortsettes på side 2.)

Oppgave 1 Binære søketrær (25%)

Vi skal i denne oppgaven arbeide med binære søketrær der vi tillater at det er flere forekomster av samme verdi. Følgende klassedefinisjon skal brukes:

```
class BinNode {
  int data;
  BinNode vsub; // venstre subtre
  BinNode hsub; // høyre subtre

  // Konstruktører
  BinNode( int element ) {
    data = element; vsub = hsub = null;
  }

  BinNode( int element, BinNode v, BinNode h ) {
    data = element; vsub = v; hsub = h;
  }
}
```

I tillegg har vi en variabel `rot` av type `BinNode` som refererer til roten i et binærtre. Treet er et *binært søketre* dersom enten `rot` er `null` eller følgende betingelser holder:

- både `rot.vsub` og `rot.hsub` er binære søketrær,
- alle data-verdiene i `rot.vsub` er *mindre eller lik* `rot.data` og alle data-verdiene i `rot.hsub` er større enn `rot.data`.

1-a

Programmer en *ikke-rekursiv* metode

```
int antall( BinNode t, int n )
```

som returnerer antall forekomster av tallet `n` i et binært søketre med rot `t`.

1-b

Programmer en metode

```
boolean bst( BinNode t )
```

som returnerer `true` dersom treet med rot `t` er et binært søketre i henhold til definisjonen over og `false` ellers.

(Fortsettes på side 3.)

1-c

Programmer en metode

```
void skrivIntervall( BinNode t, int min, int max )
```

Metoden skal gi en sortert utskrift av alle verdiene i treet med rot `t` som er i intervallet fra og med `min` til og med `max`. Du skal imidlertid ikke skrive ut samme tall mer enn en gang. Legg vekt på at metoden skal være så rask som mulig, også når treet er stort og inneholder mange forekomster av samme tall. Hvilken kompleksitet har metoden din?

1-d

Et binærtre er *komplett* dersom hvert nivå i treet er fullt, med et mulig unntak for det laveste nivået, som skal være fylt fra venstre mot høyre. Programmer en boolsk metode

```
boolean komplett( BinNode t )
```

som returnerer `true` dersom treet med rot `t` er komplett og `false` ellers.

Oppgave 2 Sortering (25%)

Et mobiltelefonselskap har en stor mengde samtaler hver dag. I løpet av en dag skrives informasjon om disse fortløpende til en fil, slik at fila ved dagens slutt inneholder informasjon om alle dagens samtaler i kronologisk rekkefølge. En slik fil lagrer informasjon om T samtaler.

Oppgaven går ut på å lage metoder som skal bli brukt til å generere dagens samtalelogg. Dette er en fil der all samtaleinformasjon er ordnet primært etter kundenummer, sekundært etter start-tidspunkt for samtalene. Logg-fila skal altså begynne med data for den første samtalen til brukeren med lavest kundenummer og skal slutte med data for den siste samtalen til brukeren med høyest kundenummer.

Vi skal i denne oppgaven representere samtalene som objekter, og ikke bry oss om hvordan disse faktisk lagres på fil. Vi lagrer da samtaledata som instanser av klassen `SamtaleInfo`, som blant annet har metodene

```
int knr();    // kundenummer til kunden bak samtalen
Dato tid();   // starttidspunkt for samtalen.
```

Hver kunde har et entydig kundenummer. Samtaledata kan sorteres etter `knr` og etter `tid`. Du kan anta at `Dato` er en klasse som implementerer interfacet `Comparable`.

(Fortsettes på side 4.)

2-a

Vi antar først at kundenumrene ligger i intervallet 1 til K og at nesten alle kundene vil ha minst en samtale pr. dag. Anta videre at internminnet alltid vil være vesentlig større enn informasjonen om alle samtalene for en dag. Programmer en Java-metode

```
SamtaleInfo[] loggOrdner( SamtaleInfo[] samtale )
```

som tar inn dagens samtaler *som en array av lengde* T og returnerer en array av samtaler ordnet som en logg.

Din metode skal være lineær i tid, og du skal vise dette.

2-b

Vi skal nå (noe mer realistisk) anta at selskapet bruker et 8-sifret telefonnummer som kundenummer. Abonnentenes telefonnumre ligger spredt over et stort intervall. Det kan nå også være mange abonnenter som ikke har noen samtaler på en gitt dag, og at de dermed ikke bidrar til loggen på den dagen.

Du skal fremdeles anta at internminnet har rikelig plass til alle samtalene.
(i) Programmer en metode som gjør det samme som metoden i oppgave 2–a.
(ii) Gi en vurdering av kompleksiteten til metoden.

2-c

Anta nå at internminnet ikke lenger er stort nok til å lagre alle samtalene på en gang. Hvordan vil du endre datastrukturen og algoritmen du brukte i 2–b? Skriv kort og i stikkordsform.

Oppgave 3 Grafer (40%)

Et flyselskap lagrer informasjon om sine reisetilbud som en vektet, urettet graf. Du kan anta at grafen er sammenhengende. Nodene angir byer (flyplasser). En kant K mellom node A og node B angir at det går en direkte flyreise fra A til B og en direkte flyreise fra B til A. Vi skal i denne oppgaven lese vekten k til en kant mellom A og B som *kostnaden* til en direkte flyreise mellom A og B (uansett vei).

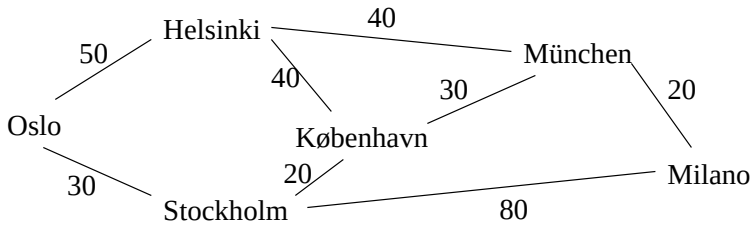
Grafen på neste side gir et bilde av hvordan en slik graf kan se ut. Den virkelige grafen er selvsagt mye større. I eksempelgrafen er kostnaden fra Oslo til Helsinki 50.

En *reiserute* fra A til B er en liste av kanter $K_1, \ldots, K_m$, $m \geq 0$, slik at man ved å følge kantene i angitt rekkefølge kommer fra A til B uten å besøke

(Fortsettes på side 5.)

noen noder mer enn en gang. *Kostnaden* til denne reiseruten defineres som $k_1 + \ldots + k_m$, der k_i er kostnaden til kant K_i. Kostnaden fra en node til seg selv (dvs. en reiserute uten kanter) er 0. *Prisen* fra A til B bestemmes som kostnaden til den reiseruten fra A til B som har minst kostnad. (Merk at pris og kostnad brukes forskjellig!)

Eksempelgraf:



3-a

Vi skal først se på grafen over. Bruk Dijkstras algoritme til å finne prisen fra Oslo til de andre byene. Du skal ikke forklare hvordan algoritmen virker i detalj, men vise noen situasjoner under utførelsen av algoritmen. Du må få med sluttresultatet og et par figurer til slik at mønsteret i algoritmen kommer tydelig frem. **(Slutt oppgave 3–a)**

Anta at prisen fra A til B er n. Det gis av og til flere mulige reiseruter fra A til B med pris n. Hvis N er en nabonode til B og prisen fra A til N er mindre eller lik n, tillater selskapet at man reiser via N uten tillegg i prisen. Man må da velge en reiserute fra A til N som har minimal kostnad, og etterpå reise direkte fra N til B. Vi sier da at ruten fra A til B via N er en *valgbar* reiserute fra A til B.

Ut fra grafen ser vi at det er to valgbare reiseruter fra Oslo til København, siden vi også kan reise om Helsinki.

Flyselskapet tillater at man gjør et opphold underveis på en reise. Hvis C er på en valgbar reiserute fra A til B, kan man gå av i C og fortsette reisen senere uten tillegg i prisen. Vi sier da at man reiser fra A til B *med stopp i C*. Hvis C ikke besøkes i en valgbar reiserute fra A til B, kan man ikke reise fra A til B med stopp i C uten å løse ny billett.

(Fortsettes på side 6.)

3-b

(i) Hvorfor er en reiserute med minimal kostnad alltid valgbar?
(ii) Hva er prisen fra Oslo til München med stopp i Helsinki?
(iii) Kan man reise fra Oslo til München med stopp i Milano?

3-c

Representasjonen av flyselskapets graf skal bruke nabolister (i en eller annen form). Definer en datastruktur for flyrute-grafen i form av Java-klasser. Du skal bruke denne datastrukturen i resten av oppgaven. Tegn så med "piler og bokser" hvordan en del av eksempelgrafen representeres i din datastruktur.

3-d

Programmer Dijkstras algoritme for å finne prisen fra en by til alle andre byer. Bruk datastrukturen du definerte i 3–c.

3-e

Du skal nå skissere en algoritme for en metode som tar inn tre byer som argumenter: A, B og C. Metoden skal skrive ut alle valgbare reiseruter fra A til B med stopp i C, samt prisen. Vi tillater at C kan være lik `null`. Metoden skal da skrive ut alle valgbare reiseruter fra A til B, samt prisen. Angi hvordan du vil tilpasse metoden du skrev i 3–d slik at den kan brukes til dette problemet. Det er tilstrekkelig med pseudokode og forklaringer (gjerne med figurer).

3-f

La K være en kant fra A til B med kostnad k. Kanten er *feilpriset* dersom det finnes en annen reiserute fra A til B med kostnad mindre eller lik k.
(i) Finn først alle feilprisede kanter i eksempelgrafen.
(ii) Programmer deretter en metode som finner alle feilprisede kanter i en graf. Husk at grafen skal bruke den representasjonen du har definert i punkt 3–c. Legg vekt på effektivitet.

(Fortsettes på side 7.)

Oppgave 4 Mønstre i brett (10%)

Anta at vi har et $N \times N$-brett der hver rute kan være enten hvit eller sort. Vi angir koordinatene til et felt med en x-verdi og en y-verdi, begge i området fra 0 til $N-1$. Vi sier at to felt er *naboer* dersom de enten har samme x-verdi og y-verdiene kun er forskjellig med 1, eller de har samme y-verdi og x-verdiene er kun forskjellig med 1.

En *sort sky* er en mengde av sorte felter slik at

(1) en sky inneholder minst ett sort felt,

(2) hvis en sky har mer enn ett felt, er ethvert felt i skyen nabo til et annet felt i skyen,

(3) alle sorte felt på brettet som er nabo til et felt i skyen, er selv med i skyen.

Vi representerer brettet som en 2-dimensjonal `boolean` array:

```
boolean[][] brett = new boolean[N][N];
```

(i) Tegn en figur som illustrerer begrepet om en sort sky.
(ii) Programmer deretter en metode som finner ut hvor mange sorte skyer det finnes på et gitt brett. Metoden har lov til å "ødelegge" representasjonen av brettet underveis.