UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i	IN 115 og IN 110 — Algoritmer og datastrukturer
Eksamensdag:	14. mai 1996
Tid for eksamen:	9.00–15.00
Oppgavesettet er på 8 sider.	
Vedlegg:	ingen
Tillatte hjelpemidler:	Alle trykte og skrevne

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

Merk:

Dette oppgavesettet gjelder både for IN 115 og IN 110. De som tar IN 115 skal ikke løse del 4 og de som tar IN 110 skal løse del 4, men ikke 1-d og 1-e. Dette er også angitt ved selve oppgavene. Prosentene angitt på delene 1, 2 og 3 er angitt for IN 115 og summerer seg derfor til 100%, mens prosenten på del 4 angir hvor mye denne teller for de som tar IN 110.

Programmer skal skrives i Simula. Du behøver ikke gi noen fullstendig dokumentasjon av programmene, men du kan skrive noen få linjer som gir leseren nøklen til forståelse av 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.

Les oppgavene nøye, og lykke til! Almira Karabeg og Stein Krogdahl

(Fortsettes på side 2.)

Del 1 (35%)

I en rettet graf definerer vi "til-alle-kjernen" (TA-kjernen) og "fra-alle-kjernen" (FA-kjernen) som følgende nodemengder:

- En node A er med i TA-kjernen til en rettet graf hvis og bare hvis det går rettede veier fra A til alle andre noder i grafen.
- En node A er med i FA-kjernen til en rettet graf hvis og bare hvis det går rettede veier til A fra alle andre noder i grafen.

Oppgave 1-a

Skisser hvordan man kan finne TA-kjernen og FA-kjernen i en rettet graf ved å utnytte Warshall-algoritmen. Du behøver ikke programmere den algoritmen du vil bruke, men du skal angi dens tidsforbruk ved O-notasjon i forhold til antall noder N og antall kanter K i grafen. Forklar kort.

Oppgave 1-b

Vi antar nå at den rettede grafen er representert ved noder av formen:

```
class node(ae); integer ae; ! antall etterfølgere ;
begin
   text navn;
   ref(node) array etterf(1:ae);
   boolean merke;
end;
```

Dessuten finnes en

```
ref(node)array G(1:N);  ! N er antall noder i grafen ;
```

Denne inneholder pekere til alle nodene i grafen i tilfeldig rekkefølge.

Din oppgave er å skrive en

```
procedure skriv_TAK(G); ref(node) array G; ... ;
```

Denne skal, ved hjelp av en indre (lokal) rekursiv prosedyre 'reksok' gjøre et dybde-først-søk fra hver av nodene i grafen for å avgjøre hvilke noder som er med i TA-kjernen i grafen. Prosedyren skal skrive ut navnet på nøyaktig de nodene som er med i TA-kjernen. Prosedyren får bare røre på (forandre) variablen 'merke' i nodene, og du vet ikke noe om verdien i denne ved starten.

Angi tidsforbruket til prosedyren ved O-notasjon i forhold til antall noder N og antall kanter K i grafen. Forklar kort.

(Fortsettes på side 3.)

Oppgave 1-c

Anta at en node A er med i både TA-kjernen og FA-kjernen. Hva kan du da si mer om de to kjernene? Forklar kort.

Oppgave 1-d. Bare for de som tar IN 115

Vi skal her anta at grafen er løkkefri. Du skal skrive en:

```
boolean procedure er_i_FAK(G,nd);
    ref(node) array G; ref(node) nd;
begin ... end;
```

Denne skal undersøke om noden 'nd' er med i FA-kjernen til G, og i så fall gi svaret TRUE, ellers FALSE. Prosedyren skal fungere ved at det gjøres et dybde-først-søk gjennom grafen, og dette skal gjøres ved en indre (lokal) rekursiv prosedyre 'reksok'. Prosedyren skal avslutte søket (gjerne ved en velplassert 'goto') så fort det eventuelt er klart at svaret er negativt.

Prosedyren får bare røre på (forandre) variablene 'merke' i nodene, og du vet ikke noe om verdien i denne ved starten.

Oppgave 1-e. Bare for de som tar IN 115

Hva kan man si om størrelsen på TA-kjernen og FA-kjernen når grafen er løkkefri? Forklar kort.

Del 2 (30%)

Vi skal se på binære trær, representert ved noder med pekere 'vsub' og 'hsub' som peker til hhv. venstre og høyre subtre om disse ikke er tomme. Ved tomme subtrær skal 'vsub' være NONE som vanlig, mens 'hsub' da skal brukes på en spesiell måte.

Vi er interessert i å se på nodene i prefiks rekkefølge, og vi tenker oss at vi stadig vil få angitt en node og skal finne neste node i prefiks rekkefølge. For å lette denne oppgaven skal vi bruke 'hsub'-variablen til å angi neste node i prefiks rekkefølge i de tilfellene den normalt ville være NONE. Vi legger derfor inn en 'boolean hpref' i nodene. Om denne er FALSE er høyre subtre ikke-tomt, og 'hsub' peker til dette. Om 'hpref' er TRUE angir 'hsub' neste node i prefiks rekkefølge. Dersom en node A både har tomt høyre subtre og er den siste i prefiks rekkefølge vil 'A.hpref' være TRUE og 'A.hsub' være NONE.

Nodene er representert ved følgende klasse:

(Fortsettes på side 4.)

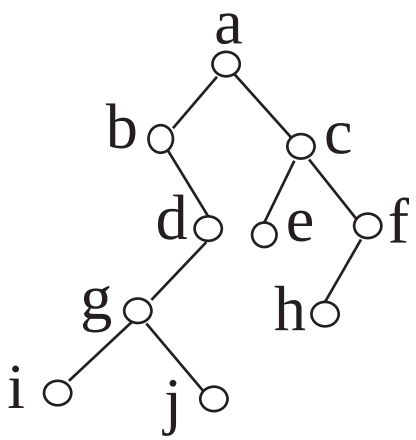
```
class node;
begin
   text navn;
   ref(node) vsub, hsub;
   boolean hpref;
end;
```

Oppgave 2-a

Vi antar nå at vi har et tre der 'hsub' brukes som beskrevet over. Ut fra treet i figuren under skal du gjøre følgende:

A: Skriv ut nodene (ved navn) i prefiks rekkefølge

B: Lag en tegning av treet der du også tegner inn hvordan 'hsub' vil peke i de nodene som har tomt høyre subtre. Sett et kryss i de nodene som har 'hpref' lik TRUE.



Oppgave 2-b

Vi antar igjen at vi har et tre der 'hsub' og 'hpref' er satt som beskrevet over. Du skal skrive en prosedyre som for en gitt node 'x' skal levere en peker til den neste noden i prefiks rekkefølge. Dersom 'x' er siste noden i prefiks rekkefølge skal prosedyren levere NONE. Prosedyren skal ha formen:

```
ref(node) procedure neste(x); ref(node)x; ! Kan anta x=/=NONE;
```

Prosedyren bør være så effektiv som mulig.

Oppgave 2-c

Med den angitte bruk av 'hsub' og 'hpref' er det altså alltid mulig, ut fra en gitt node, å finne neste node i prefiks rekkefølge. Dette gjelder selv om

(Fortsettes på side 5.)

nodene ikke har foreldre-pekere og selv om vi ikke kjenner roten av treet.

Vi vurderer nå å gjøre noe tilsvarende for *postfiks* rekkefølge. Ved tomt høyre subtre vil vi altså la 'hsub' peke til neste node i *postfiks* rekkefølge, og vi vil ikke ha foreldrepekere i nodene. Vi får så stadig gitt en node (men ikke roten) i treet, og får spørsmål om å finne neste node i *postfiks* rekkefølge. Vurder om dette vil fungere. Forklar kort.

Oppgave 2-d

Vi har gitt et binært tre representert ved noder av klassen angitt over. Treet er imidlertid representert på helt vanlig måte (med NONE i 'hsub' ved tomt høyre subtre, og med en tilfeldig verdi i 'hpref'). Du skal skrive en:

```
procedure setprefiks(rot); ref(node)rot; ... ;
```

Denne skal, ved en indre (lokal) rekursiv prosedyre 'rekpref', gå gjennom treet og sette verdi til 'hpref' og prefikspekere i 'hsub' slik som angitt over.

HINT: Det kan være behagelig å ha en peker (lokalt i den ytterste prosedyren) som i en passelig forstand angir "forrige node".

Del 3 (35%)

Vi skal her se på sortering. Oppgave 3-a er helt uavhengig av de andre.

Oppgave 3-a

Sekvensen på 7 tall angitt under skal sorteres ved Heap-sort i stigende rekkefølge.

5 3 7 2 1 4 6

Angi hvordan situasjonen vil være:

A: Etter at første fase (opprydningen = "buildheap") er ferdig.

B: Etter at selve sorteringsløkka er gått tre ganger (og de tre største verdiene altså er funnet fram).

Oppgave 3-b

Bit-strenger er sekvenser av 0'er og 1'ere, og vi skal se på bit-strenger av lengde K. En sekvens av slike bit-strenger skal sorteres. Om vi har to ulike bit-strenger avgjør vi hvilken som er størst slik: Vi leser dem begge forfra (fra venstre) og stopper ved den første bitposisjonen der de er forskjellige.

(Fortsettes på side 6.)

Den som har 0 i denne posisjonen er da mindre enn den som har 1 (og den med 0 skal derfor komme først i sorteringen).

For å forenkle programmeringen lar vi bit-strengene framstå i programmet som objekter av klassen:

```
class bitstreng;
begin
   integer array bs(1:K); ! Inneholder bare 1'ere og 0'er ;
end;
```

Det som skal sorteres er en 'ref(bitstreng) array A(1:N)', hvor alle 'A(i)' angir separate bitstreng-objekter. Sorteringen skal gjøres ved å flytte rundt på pekerne i A.

Til sorteringen skal vi bruke en metode som henter noen ideer fra Quicksort. Vi skal gjøre rekursive kall omtrent som i Quicksort, og hvert kall skal få oppgitt et intervall av arrayen den skal sortere, samt en bit-posisjon 'p' den skal arbeide på. Prosedyren skal så gjøre en 'partitioning' (omtrent som i Quicksort) slik at de bitstrengene med '0' i posisjon p kommer til venstre og de med '1' kommer til høyre. Deretter skal den gjøre rekursive kall på disse to segmentene, med en litt forandret p-verdi (om den da ikke er ferdig).

Du skal programmere dette slik at du får en riktig sortering i A. Du kan passelig bruke et skjema med en rekursiv prosedyre inne i en ikke-rekursiv, slik:

```
procedure SLU_sort(A, N); ref(bitstreng) array A; integer N;
begin
    procedure bytt(i,j); ...; ! Kan være behagelig, men du skal
                                programmere den om den brukes.;

    procedure reksort(i,j,p); integer i, j, p;
    begin ... end;

    < Kall som starter rekursjonen >; ! Skal også programmeres ;
end;
```

Oppgave 3-c

Angi ut fra N og K hvor mange kall det totalt kan bli på 'reksort' i 3-b. Forklar kort.

Oppgave 3-d

Angi ved O-notasjon ut fra N og K det totale tidsforbruket til prosedyren i 3-b. Forklar kort.

(Fortsettes på side 7.)

Del 4. Bare for de som tar IN 110 (15%)

Et antall objekter av klassen 'elem' (gitt under) er kjedet sammen til en liste ved 'neste'-pekeren, og slik at siste element har 'neste==NONE'. Det første elementet i listen er et "hode", og er altså ikke med i den egentlige listen. Listen er ikke tom, dvs. at den inneholder hodet og minst ett vanlig element. Klassen 'elem' er deklarert som følger:

```
class elem;
begin
   ref(elem) neste;
   < Andre attributter uten interesse for oss >;
end;
```

Du skal skrive en:

(Fortsettes på side 8.)

```
procedure permliste(H,S);
     ref(elem) H,S;  ! Pekere til hodet og til SISTE element i listen;
begin
   procedure rekperm(E); ref(elem) E; ... ;

   rekperm(H); ! Forslag til kall som setter genereringen i gang ;
end;
```

Oppgaven er å skrive en prosedyre som permuterer elementene i listen på alle måter, ved å flytte rundt på elementene i listen. For hver permutasjon skal man kalle prosedyren 'bruklist(H)' (Der H er listens hode), og etter at det hele er over skal listen se ut som da 'permliste' ble kalt.

Det kan være greit å løse oppgaven med en rekursiv prosedyre inne i en ikke-rekursiv prosedyre, slik som antydet over, og det er best om du løser oppgaven slik at du i prosedyrekallene ikke gjør noe søk i lista.

NB: Du skal løse oppgaven ved å arbeide med selve listen, IKKE ved å definere en array med pekere til elementene i listen e.l. Definer (og programmer) gjerne som egne prosedyrer de liste-operasjonene du får behov for.

HINT: Parameteren E til 'rekperm' skal altså definere den delen av lista som dette kallet skal arbeide på. Denne parametern bør angis slik at du friest mulig kan arbeide med elementene i denne delen av lista. Du står fritt til å generere permutasjonene i den rekkefølge det passer deg.