# UNIVERSITETET I OSLO

## Det matematisk-naturvitenskapelige fakultet

| | |
|---|---|
| Eksamen i | IN 110 — Algoritmer og datastrukturer |
| Eksamensdag: | 18. mai 1993 |
| 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:

Oppgavesettet består av fire deler som kan løses helt uavhengig av hverandre. Oppgavene innen hver del er tenkt løst i den rekkefølge de står, men dette er selvsagt intet krav til din besvarelse. Prosenten angitt på hver del antyder hvor mye vekt det vil bli lagt på denne delen under sensureringen.

Programmer skal skrives enten i Simula eller Pascal. Oppgavene er formulert ut fra at de skal skrives i Simula. De som vil bruke Pascal skal erstatte de angitte klasse-deklarasjonene med tilsvarende record-deklarasjoner, samt gjøre andre opplagte småjusteringer. Du behøver ikke gi noen fullstendig dokumentasjon av programmene, men du skal 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 boka.

*Les oppgavene nøye, og lykke til!* Stein Krogdahl og Arne Maus

*(Fortsettes side 2.)*

# Del 1 (15%)

Vi skal se på prioritetskøer av heltall, der vi hele tiden er interessert i å få ut den *minste* verdien. Prioritetskøene skal være implementert ved en "heap"-struktur lagret i en array, og det er da som kjent visse "heap"-betingelser som må oppfylles mellom verdiene i arrayen.

### Oppgave 1-a

Et antall heltallsverdier ligger i arrayen H, fra indeks 1 til indeks n. De ligger i tilfeldig rekkefølge, og vi skal ordne dem slik at de oppfyller "heap"-betingelsen. Kari har nettopp skrevet en effektiv sorteringsalgoritme, og foreslår derfor at man rett og slett sorterer tallene i stigende rekkefølge i arrayen. Hennes medarbeider Ola mener imidlertid at dette ikke alltid vil bli riktig. Hvem har rett? Begrunn svaret.

### Oppgave 1-b

Fra indeks 1 og oppover i arrayen H ligger følgende tall:

3, 12, 4, 33, 13, 5, 6, 34, 35, 17

Disse representerer en prioritetskø med 10 elementer, organisert som en heap. Vi skal fra denne situasjonen utføre følgende operasjoner (implementert som vanlige heap-operasjoner).

- Sett inn verdien 8

- Ta ut den minste verdien

NB: Også i siste tilfellet skal du starte med de opprinnelige 10 verdiene. Angi svaret som den sekvens av 11 tall og den sekvens av 9 tall som blir liggende i arrayen H etter operasjonene.

# Del 2 (30%)

Vi skal her se på urettede grafer, og for en gitt slik graf skal vi si at en ikketom undermengde (et utplukk) av nodene er "uavhengig" dersom det ikke går noen kant mellom noder i utplukket. Se f.eks. på følgende graf:

Her utgjør f.eks. både $\{1, 3, 5, 6\}$ og $\{3, 4\}$ uavhengige nodemengder, mens f.eks. mengden $\{4, 5, 6\}$ *ikke* er uavhengig (p.g.a. kanten mellom 4 og 6). Merk at alle mengder med bare én node er uavhengige, mens den tomme nodemengden altså ikke regnes som uavhengig.

*(Fortsettes side 3.)*

**Oppgave 2-a**

For en gitt (urettet) graf vil vi ha skrevet ut alle mulige uavhengige nodemengder (selv om dette kan bli nokså mange!). Vi antar at nodene er identifisert med tallene fra 1 til $n$, og at grafen er angitt ved en (symmetrisk) nabomatrise G, som er globalt deklarert. Du skal lage en prosedyre "skriv_alle" som gjør den riktige utskriften, ved å fylle ut innmaten i prosedyren "genresten(k)" angitt under. Studér kommentaren inne i prosedyren nøye før du begynner.

Gi også en kort forklaring på hvorfor din prosedyre vil skrive ut alle uavhengige nodemengder.

```
...
boolean array G(1:n,1:n);  ! Denne er ferdig fylt med verdier ;
...
procedure skriv_alle;  ! Denne skal ikke være rekursiv;
begin
   integer array utplukk(1:n);
   integer ant;

   procedure genresten(k); integer k; ! Denne skal være rekursiv ;
   begin
      ! Når denne kalles er det gjort et uavhengig utplukk blant
        nodene 1,2, ... , k-1, og antallet i dette utplukket ligger i
        "ant", og de plukkede nodene er angitt i arrayen "utplukk" fra
        indeks 1 til "ant".  Denne prosedyren skal (gjerne med hjelp fra
        nye rekursive kall) sørge for å få laget alle mulige (lovlige)
        utvidelser av dette utplukket (også den tomme utvidelse!), og
        å få skrevet ut hver av de resulterende uavhengige node-mengder.;
   end;

   genresten(1);
end;
```

Hint: Én mulig organisering er at hvert rekursivt kall bare ser på alle muligheter for å utvide utplukket med én node, og å overlate videre utplukk etc. til nye rekursive kall. Tenk nøye over hvor i programmet det skal gis utskrift.

— — —

Vi skal nå lage en ny variant "skriv_en(M)" av prosedyren "skriv_alle", der vi gjør to forandringer i forhold til 2-a. For det første angir vi en heltallsparameter 'M', og er nå bare interessert i uavhengige mengder med minst 'M' noder. For det andre er vi ikke interessert i *alle* slike uavhengige mengder, men bare ett eneste slikt utplukk. Om det ikke finnes noe slikt utplukk med minst 'M' elementer skal prosedyren skrive ut: "INGEN FINNES!". Dette

*(Fortsettes side 4.)*

kan selvfølgelig gjøres med en enkel tilleggstest i programmet over, men vi skal forsøke å bruke 'M' til litt ekstra avskjæring, og filosofien skal være som følger:

Prosedyren "skriv_en(M)" skal lages etter samme mønster som "skriv_alle", men vi skal i tillegg ha en "integer array ant_kanter(1:n)". Når "genresten(k)" kalles skal vi i "ant_kanter(i)", for $i = k, k+1, \ldots, n$, ha registrert hvor mange kanter det går mellom node $i$ og noder som allerede er plukket ut.

### Oppgave 2-b

Angi kort hvordan arrayen "ant_kanter" kan brukes til (mest mulig) avskjæring ved programmering av "skriv_en(M)".

### Oppgave 2-c

Skriv prosedyren "skriv_en(M)" med den nye versjonen av "genresten(k)".

# Del 3 (30%)

Vi skal her arbeide med et binært søketre, der vi stadig skal sette inn noder, men aldri ta noen ut. Innimellom innsettingene skal vi gjøre vanlige oppslag. Forholdet her er imidlertid at når en ny node skal settes inn vil den alltid ha større verdi enn alle tidligere innsatte noder. Om vi her hadde satt noder inn på vanlig måte ville dette gitt et helt skjevt tre, men siden vi vet at vi nå vil sette inn nodene i stigende rekkefølge kan vi utnytte dette til å lage et ganske godt balansert tre. Men merk at treet hele tiden (mellom innsettinger) må være et søketre, som vi kan gjøre vanlige oppslag i.

Om vi tenker oss at verdiene i de nodene vi skal sette inn er 1, 2, 3, 4, 5, og 6, så vil vi organisere oss slik at trærne vi etter hvert får er som følger:

Som en generell regel skal en innsetting gjøres som følger: Vi sier at en node er "ubalansert" dersom høyden av dens to subtrær ikke er like. Ved innsetting av en ny node N (som altså har verdi større enn alle nåværende noder i treet) følger vi først den veien vi ville gått ned gjennom treet om vi lette etter en node med denne verdien, altså hele tiden til høyre. På veien registrerer vi de noder som er ubalanserte, og velger til slutt den siste ubalanserte noden U vi fant. Vi setter så N inn som ny høyre subnode til U, og lar det gamle høyre

*(Fortsettes side 5.)*

subtreet til U bli nytt venstre subtre til N. Hva som skal gjøres dersom man ikke finner *noen* ubalanserte noder må du selv finne ut av.

**Merk**: For å ha en felles terminologi fastslår vi her at høyden av en node er definert som veilengden fra noden ned til fjerneste bladnode, og høyden av et (sub)tre er høyden til roten av (sub)treet. Om man her definerer høyden av en bladnode til 0 eller 1 spiller liten rolle (bare man er konsekvent), og høyden av et "tomt subtre" defineres som én lavere enn høyden av en bladnode.

### Oppgave 3-a

Anta at du fortsetter innsettingen angitt over ved å sette inn noder med verdiene 7, 8, 9, 10, 11 og 12. Tegn opp hvordan treet vil se ut etter at verdien 8, etter at verdien 10 og etter at verdien 12 er satt inn. Du skal altså tegne opp tre fullstendige trær.

### Oppgave 3-b

I en implementasjon ønsker vi i hver node å ha registrert dens høyde i det nåværende treet. Når vi setter inn en ny node slik som angitt over må vi selvfølgelig gi den nye noden riktig høyde. Hvilke andre noder i treet kan få forandret sin høyde? Forklar kort.

— — —

Nodene skal ha følgende klasse-deklarasjon

```
class node;
begin
 integer verdi;
 integer h; ! nodens høyde ;
 ref(node) vsub, hsub; ! Pekere til venstre og høyre subtre, om de finnes ;
end;
```

### Oppgave 3-c

Skriv en prosedyre:

```
procedure settinn(rot, v); name rot; ref(node) rot; integer v;
begin
   ! Vi antar altså her at 'v' er større enn alle tidligere verdier
     i treet. Lag en ny node med verdi 'v' og sett den inn i treet
     med den angitte rot, på den måten som er beskrevet over;
end;
```

*(Fortsettes side 6.)*

### Oppgave 3-d

I den sammenhengen disse trærne skal brukes viser det seg å være behov for en opersjon "forandre_verdi(rot, v)" som forandrer én av verdiene i treet til 'v'. Treet er som vanlig angitt ved sin rot. Den verdien som skal forandres er den som ligger nærmest (sett som vanlige heltall) til 'v'. Dersom 'v' ligger *midt* mellom to verdier i treet skal vi velge å forandre den som er lavere enn 'v', og dersom 'v' er mindre enn alle verdier i treet skal den laveste verdien forandres (og tilsvarende om 'v' er større enn alle verdier). Om 'v' allerede er i treet skal det ikke gjøres noen forandring. Skriv en prosedyre som gjør denne operasjonen:

```
procedure forandre_verdi(rot, v); ref(node) rot; integer v;
```

NB: Selv om prosedyren primært er beregnet på trær av den typen vi har diskutert over skal den skrives slik at den også virker for binære søketrær som ikke er spesielt balanserte. Du kan fremdeles anta at alle verdiene i treet er forskjellige. Legg vekt på at operasjonen blir rask.

## Del 4 (25%)

Petter Smart fulgte forelesningene i IN-110 for noen år siden, og kom da til foreleserene med den sorteringsalgoritmen "PS_sort" som er gjengitt på neste side.

### Oppgave 4-a

Anta at arrayen "A(1:8)" i rekkefølge inneholder verdiene (8, 6, 7, 4, 5, 1, 2, 3), og at vi kaller "PS_sort(A,8)". Skriv ut innholdet av arrayen A ved terminering av de forskjellige rekursive kall av PS, i den rekkefølge de terminerer. Du kan hoppe over de kallene som ikke har '`fra { til`' (og som dermed ikke gjør noe). Understrek hver gang det intervallet (fra, til) som det terminerende prosedyrekallet arbeidet på.

### Oppgave 4-b

Angi den invarianten som gjelder ved programpunktet 'INV', for den omkringliggende forløkka. Argumenter kort (hvertfall ikke mer enn en side) for at algoritmen faktisk sorterer arrayen. Invarianten kan gjerne angis ved skisser og tekst.

### Oppgave 4-c Kan puffes (dårligste karakter er 4.0)

Petter Smart hevder at dette er en effektiv sorteringsalgoritme. For å avgjøre om han har rett skal du foreta en "worst case" analyse av algoritmen, og angi hvilken orden (angitt ved O-notasjon) denne sorterings-algoritmen har. Er du enig med Petter Smart?

*(Fortsettes side 7.)*

**Programmet som Petter Smart leverte til foreleserene var som følger:**

```
procedure PS_sort(A,n); integer array A; integer n;
begin
   procedure PS(fra,til); integer fra, til;
   begin
      integer i,j,m,v;
      if fra { til then
      begin
         m:= (fra + til) // 2;  ! '//' er heltallsdivisjon (runder ned!) ;
         PS(fra, m);
         PS(m+1, til);

         for i:= fra step 1 until m do
         begin
    INV:
            if A(i) } A(m+1) then
            begin
               v:= A(i);
               A(i):= A(m+1);

               for j:= m+2 step 1 until til do
               begin
                  if v } A(j) then A(j-1):= A(j)
                  else goto UT;
               end;
          UT:
               A(j-1):= v;
            end;
         end;
      end;
   end;

   PS(1,n);
end;
```

**(Slutt på oppgavesettet)**