UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i	IN 110 — Algoritmer og datastrukturer
Eksamensdag:	16. mai 1994
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:

Oppgavesettet består av tre deler som kan løses helt uavhengig av hverandre. Oppgavene innen hver del kan også løses i vilkårlig rekkefølge, men du må da ha lest de foregående oppgavene nøye. Prosenten angitt på hver del antyder hvor mye vekt det vil bli lagt på denne delen under sensureringen.

Programmer skal skrives i Simula. Du behøver ikke gi noen fullstendig dokumentasjon av programmene, men du skal skrive noen få linjer som gir leseren nøkkelen til forståelse av programmet. Du kan anta at leseren kjenner problemstillingen i oppgaven meget godt.

Vi har en del steder brukt ordet "attributt", og om det skulle være tvil er dette altså Simula-terminologi for en lokal variabel i et klasse-objekt.

Når det i teksten under refereres til "læreboka" menes boka "Data Structures and Algorithm Analysis" av Mark Allen Weiss.

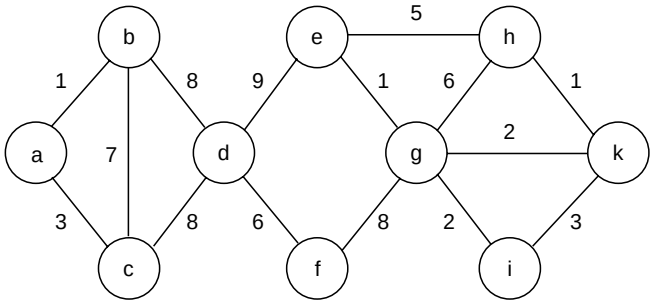
Alle steder der det er spørsmål etter et program (eller en programbit) skal du skrive dette helt ut, og *ikke* bare henvise til liknende programmer f.eks. i læreboka.

Les oppgavene nøye, og lykke til! Stein Krogdahl og Ellen Munthe-Kaas

(Fortsettes side 2.)

Del 1 (8%)

Vi skal finne et minimalt spenntre i en urettet graf ved Kruskals algoritme. Den aktuelle grafen er som følger, med kantvekter angitt for hver kant:



Oppgave 1

Angi hvilke kanter som vil være plukket ut til treet (i læreboka: "akseptert") etter at kanten *b-c* er behandlet av Kruskal-algoritmen. Angi en kant som et par av noder, og gi svaret som en liste av slike. Alternativt kan du tegne opp grafen, og utheve (tydelig!) de kanter som er plukket ut.

Del 2 (42%)

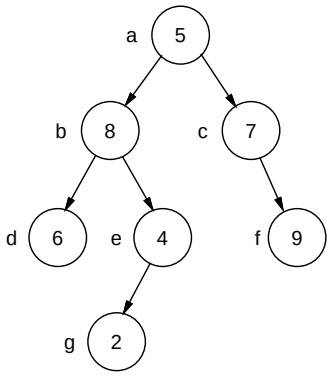
I denne oppgaven skal vi betrakte binære trær, der nodene har pekere til sine sub-noder (kalt `vsub` og `hsub`) og en heltallsverdi (kalt `verdi`), samt noen flere attributter som vi skal se på senere. Merk at i de trærne vi skal arbeide med her vil det ikke være noen åpenbar sammenheng mellom verdiene og strukturen av treet. Det blir f.eks. ikke noe søketre. Vi skal bare tenke oss at vi er blitt bedt om å programmere operasjonene slik de er beskrevet, uten at vi kjenner den anvendelsen som ligger bak.

Når nye noder skal inn i treet er vi interessert i at disse plasseres så nær roten som mulig, og vi skal derfor se på noen strukturer der vi fra hver node lett kan finne nærmeste ledige plass i det subtreet den er rot i.

I det videre skal vi kalle noder som har ett eller ingen barn, for **åpne noder**, i og med at de har plass til en ny node under seg. Resten av nodene kalles **indre noder**, og for hver indre node skal vi definere dens **nærmeste åpninger** som de åpne noder i dens subtrær som er så nær den som mulig.

(Fortsettes side 3.)

I figuren under er a og b indre noder, og c, d, e, f og g er åpne noder. a's nærmeste åpninger er c, mens b's nærmeste åpninger er d og e.



Vi er interessert i lett å kunne komme fra en indre node til en av dens nærmeste åpninger. Vi skal derfor bruke følgende nodestruktur:

```
class node(verdi); integer verdi;
 begin
    ref(node) vsub, hsub, sti;
    integer avst;
 end
```

I hver indre node skal attributtet `sti` angi retningen til en nærmeste åpning (og derved altså peke enten til nodens `hsub` eller `vsub`), og `avst` skal angi avstanden (antall mellomliggende kanter) til den nærmeste åpning. Dersom det er en nærmeste åpning både i venstre og høyre subtre skal `sti` være lik `vsub`. For åpne noder skal `sti==none` og `avst=0`.

Oppgave 2-a

Vi skal skrive en prosedyre som går ut fra at attributtene `sti` og `avst` ikke har verdier som stemmer med kravet angitt over. Den skal gå rekursivt gjennom hele treet og sørge for at `sti` og `avst` blir riktig satt, og den skal ha følgende form.

```
procedure stifinner(t); ref(node) t;..
```

Prosedyren kommer til å bli kalt utenfra med treets rot som parameter.
(Slutt oppgave 2-a)

Når vi skal sette en ny node inn i treet skal den plasseres under den nærmeste åpning vi kommer til fra roten ved å følge `sti`-pekerne. Om denne åpne noden ikke har noen barn skal den settes inn som ny `vsub`, ellers skal den plasseres der det er plass. Etter innsettingen må vanligvis attributtene `sti` og `avst` justeres i en del noder for igjen å stemme med kravet angitt over.

(Fortsettes side 4.)

Oppgave 2-b

Anta at vi skal sette en node h med verdi 3 inn i treet i figuren.

Tegn hvordan treet ser ut, inklusive attributtene `sti` og `avst` både før og etter innsettingen.

Oppgave 2-c

Skriv en prosedyre

```
procedure settinn(t,w); ref(node) t; integer w; ..
```

som lager en ny node med verdi `w` og setter den inn slik som angitt over. Prosedyren skal deretter gjøre den nødvendige oppdateringen av attributtene `sti` og `avst` i nodene i treet. Prosedyren vil naturlig inneholde rekursive kall, men du kan anta at prosedyren alltid blir kalt på ytterste nivå med `t=rot`, og at `rot=/=none`.
(Slutt oppgave 2-c)

Vi skal nå gå over til en datastruktur der vi mer direkte peker ut en nærmeste åpning fra hver indre node, og der nodene også har forelderpeker. Klassen node ser nå dermed slik ut:

```
class node(verdi); integer verdi;
 begin
    ref(node) vsub, hsub, foreld, næråpen;
    integer avst;
    ref(node) procedure enebarn;
       enebarn:- if vsub=/=none then vsub else hsub;
 end
```

Attributtet `avst` skal ha samme verdier som før, men `næråpen` skal i indre noder direkte angi den noden vi fant ved å følge `sti`-pekerne i forrige struktur, nemlig en nærmeste åpning. I åpne noder skal `næråpen` være `none` og i rotnoden skal `foreld` være `none`.

Prosedyren `enebarn` er en hjelpeprosedyre som du kan bruke om du finner det passelig. Vi kunne nå programmert om igjen tilsvarende prosedyrer som under 2-a og 2-b, men vi skal i stedet se på hvordan fjerning skal gjøres, og din oppgave blir å programmere dette.

Når vi skal fjerne en åpen node, skal det gjøres ved bare å lappe treet sammen lokalt på enkleste måte. Når vi skal fjerne en indre node, skal dette gjøres ved å la den utpekte nærmeste åpning ta dens plass. Mer konkret kopierer vi verdien fra nærmeste åpning inn i den som skulle fjernes, og fjerner deretter den åpne noden i stedet. (Det er dermed riktigere å si at vi fjerner en *verdi* enn at vi fjerner en node).

(Fortsettes side 5.)

Oppgave 2-d

Skriv en prosedyre

```
procedure fjern(t,rot); name rot; ref(node) t,rot; ..
```

som fjerner noden `t` fra treet `rot` i henhold til beskrivelsen over, og også gjør de nødvendige oppdateringer slik at alle krav angitt over igjen blir tilfredstilt. Du kan anta at `t` alltid vil være en node som faktisk befinner seg i treet (dvs. den er aldri `none` og roten i dens tilhørende tre er `rot`).

Merk: Det kan her bli litt fikleprogrammering, så sørg for å få sett på 2-e og del 3 heller enn å bruke masse tid på denne i første omgang!

Oppgave 2-e

Vi tenker oss nå at vi holder oss til den siste datastrukturen diskutert over, og at treet på et visst tidspunkt er helt balansert. Altså: Alle åpne noder er blader i treet, og de ligger alle like langt fra roten. Vi kaller så prosedyren `fjern` gjentatte ganger, med begge parameterne lik roten i treet, og holder på med dette til treet er tomt.

Spørsmål A: Hvor mange ganger blir prosedyren kalt før vi får et nytt node-objekt som rotnode (altså, at rotnoden ikke bare får erstattet sin verdi)?

Spørsmål B: Anta at vi, som angitt over, har gjort et antall kall på `fjern`, og at høyden i treet (antall kanter mellom roten og fjerneste node) nå er h. Hvor mange noder er det da minst i treet?

Del 3 (50%)

Vi skal arbeide med rettede grafer, der kantene er "farget", ved at hver kant er tilordnet et tall mellom 1 og *af* ("antall farger"). Generelt er vi ute etter en bestemt type veier i grafen, og vi skal si at en vei er "F-begrenset" dersom den benytter kanter med maksimalt F forskjellige farger.

Vi skal anta at grafen er gitt, og at nodene i utgangspunktet er representert med objekter av klassen:

```
class node(ak); integer ak;  ! antall (utgående) kanter i denne node;
begin
   integer nr; ! Nodene er nummerert i tilfeldig rekkefølge fra 1 til an ;
   ref(node) array kant (1:ak); ! Endenodene for hver utgående kant ;
   integer   array farge(1:ak); ! Farge mellom 1 og af for tilsv. kant;
   boolean merke;   ! Står til disp. som merke ved forskjellige søk etc. ;
   boolean aktuell; ! Se teksten under ;
end;
```

(Fortsettes side 6.)

Videre finnes en array "`ref(node) array graf(1:an);`" som har en peker til alle grafens noder, slik at `graf(i).nr = i`. Vi vet ikke noe spesielt om grafens struktur, den kan f.eks. godt ha løkker.

Vår generelle problemstilling vil være at vi har gitt to forskjellige noder a og b, samt et heltall F, og at vi ønsker å finne en F-begrenset enkel vei fra a til b (eller avgjøre at ingen slik finnes). Merk imidlertid at for en del av de innledende oppgavene under så kommer ikke fargen på kantene inn i problemstillingen.

Oppgave 3-a

Skriv en prosedyre:

```
procedure fyllnabomatrise(graf, an, G);
   ref(node)array graf; ! Angir grafen slik som beskrevet over ;
   integer an;
   boolean array G; ! Dimensjonert G(1:an, 1:an), skal fylles ;
begin
   ...
end;
```

Denne prosedyren skal fylle matrisen G slik at den blir en nabomatrise for den grafen som er angitt av arrayen `graf` slik som forklart over (og med samme nodenummerering som er brukt der). Du kan anta at G er fylt av bare `false` når prosedyren kalles.

Oppgave 3-b

Som forberedelse til å lete etter veier slik som antydet i innledningen, skal vi ut fra gitte noder a og b finne alle noder som kan ligge på veier (ikke nødvendigvis enkle veier) fra a til b. For å finne disse nodene blir grafen først lest inn i en nabomatrise G som angitt i 3-a, og deretter blir Warshall-algoritmen (angitt i "Tilleggsstoff til IN 110") utført på denne matrisen. (Dette skal ikke programmeres!).

Oppgaven er å skrive en prosedyre:

```
procedure settakt1(G, graf, an, a, b);
   boolean array G; ! Denne er ferdig behandlet av Warshall-algoritmen ;
   ref(node)array graf; integer an;
   ref(node) a, b;
begin
   ...
end;
```

Denne skal, ut fra det som da står i matrisen G, sette attributtet `aktuell` i hvert node-objekt til `true` om det går en vei fra a til b gjennom denne noden, og ellers sette den til

(Fortsettes side 7.)

`false`. Hvilken verdi `aktuell` får i a og b er ikke viktig. Vi vet ikke noe om verdien av attributtet `aktuell` når prosedyren kalles.

Oppgave 3-c

Vi skal her se på samme problemstilling som i 3-b, nemlig å få satt attributtet `aktuell` slik som beskrevet. Forklar først kort hvordan man kan løse denne oppgaven ved å foreta to søk: Ett fra a der vi følger kantene framover og ett fra b der vi følger kantene bakover.

Diskuter så kort hvordan dette kan implementeres om vi først lager oss en nabomatrise som angitt i 3-a, og bruker denne til å orientere oss i grafen (og forøvrig står fritt til å bruke attributtet `merke` i nodene). Angi ved O-notasjon hvor lang tid algoritmen da vil ta, og sammenlikn dette med tiden som går med til den løsningen som er foreslått i 3-b (inklusive Warshall-algoritmen).

Oppgave 3-d

Vi skal her nok en gang løse den samme oppgaven som i 3-b, men vi skal nå anta at grafen *ikke har løkker*, og bare bruke den opprinnelige graf-representasjonen. Vi skal nå løse den ved å gjøre et dybde-først-søk fra a i grafen. Dette skal gjøres av en prosedyre:

```
boolean procedure settakt2(graf, an, a, b);
   ref(node)array graf; integer an;
   ref(node) a, b;
begin
   procedure rekakt(...); ... ! Kanskje skal den også levere et svar? ;
   ...
end;
```

Selve søket skal gjøres av den rekursive prosedyren `rekakt`. Denne skal du selv bestemme parameterene til (samt om den skal levere et svar av passelig type), og du skal programmere den fullt ut, inklusive det kall som setter den i gang. Du kan anta at attributtet `merke` er `false` i alle noder når prosedyren kalles.

Det viktigste er her å få prosedyren riktig programmert, men gi også en kort forklaring til hvorfor den virker.

Oppgave 3-e

Du skal her skrive en prosedyre som implementerer et kombinatorisk søk etter en F-begrenset *enkel* vei fra en gitt node a til en annen gitt node b. Dette skal gjøres ved følgende prosedyre:

(Fortsettes side 8.)

```
procedure F_vei_let(svar, an, a, b, F, af);
ref(node) array svar; ! Dimensjonert "svar(0:an)". Her skal
             nodesekvensen langs den veien som er funnet ligge,
             inklusive a (i posisjon 0) og b, og med 'none' i den
             delen av arrayen som ikke er i bruk ;
integer an; ref(node) a, b; ! Nodene vi skal finne vei mellom ;
integer F;  ! Maksimalt antall forskjellige farger veien kan ha ;
integer af; ! Det totale antall farger, nummerert 1, 2, ..., af ;
begin
   ! Følgende globale data kan passelig brukes i søket ;
   integer array brukt(1:af); ! Antall kanter brukt med hver av fargene ;
   integer fargant; ! antall forskjellige farger brukt langs veien;

   procedure reklet(...); ... ; ! Prosedyren som gjennomfører selve søket;
   ...
end;
```

Grafen kan godt ha løkker, og du skal anta at når prosedyren `F_vei_let` kalles så er attributtet `aktuell` satt i hver node slik som angitt i 3-b. Attributtet `merke` vil da være `false` i alle noder. Forøvrig skal du gjøre den avskjæring at du ikke bygger videre på veier som har flere enn F farger.

Søket skal avsluttes så fort en vei er funnet (og du må gjerne til dette bruke en velplassert `goto`-setning). Om ingen vei finnes skal arrayen `svar` inneholde bare `none` når prosedyren terminerer. Skriv prosedyren `reklet`, og setninger i `F_vei_let` slik at søket starter og avsluttes riktig. Studer nøye kommentarene til parameterne og de lokale deklarasjoner i `F_vei_let`.

Det er helt OK å bruke arrayen `svar` som arbeidsarray under søket. Du kan anta at denne inneholder bare `none` når `F_vei_let` kalles.

Oppgave 3-f (Kan puffes)

I en passelig forstand finner Dijkstras algoritme den 'best mulige vei' (i betydningen 'korteste') fra en gitt node til alle andre noder i grafen.

Ved å la 'best mulig' i stedet bety 'med så få forskjellige farger som mulig' kunne man forsøke å lage en variant av Dijkstras algoritme som finner veier med så få farger som mulig fra en node til alle andre noder i grafen. Gi en kort vurdering, gjerne med et eksempel, av om denne filosofien vil fungere eller ikke.

(Slutt på oppgavesettet)