UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i	IN 115 — Algoritmer og datastrukturer
Eksamensdag:	14 . mai 1998
Tid for eksamen:	9.00–15.00
Oppgavesettet er på 8 sider.	
Vedlegg:	Ingen
Tillatte hjelpemidler:	Alle skrevne og trykte

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

Merk:

Oppgavesettet består av 4 deler som kan løses helt uavhengig av hverandre. Oppgavene innenfor hver del kan også stort sett løses i en vilkårlig rekkefølge, men du må ha lest de foregående deloppgavene nøye. Prosenten indikerer hvor stor vekt det blir lagt på denne delen under sensureringen.

Merk at det kan være flere enkeltspørsmål under hvert punkt, pass på å svare på alle.

Programmene 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 programmene. Du kan anta at leseren kjenner problemstillingen meget godt.

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

Les oppgavene nøye, og lykke til!

Stein Krogdahl og Anne Salvesen

(Fortsettes på side 2.)

Oppgave 1 15 %

Du skal skrive en prosedyre `lagAlle` som i en global `character array S(1:n)` genererer alle sekvenser av en bestemt type (se under). Tallet `n` er også gitt globalt, og for hver sekvens som er generert skal prosedyren `brukS` kalles.

De sekvensene du skal generere skal alle være av lengde `n`, og skal bygges opp av tegnene: `'('`, `')'` og `'*'`. I tillegg skal de være velformede parentetiske strukturer, men slik at tegnet `'*'` kan forekomme innimellom så mye det passer. For $n = 5$ vil altså f.eks. følgende være lovlige sekvenser:

```
* * * * *
* ( * * )
( ) ( * )
( * ( ) )
```

Følgende eksempler er derimot ulovlige:

```
* ( * * *
( ) * ) (
```

Velg parameterene til `lagAlle` selv, og angi også kallet som starter generer-ingen. Legg vekt på god avskjæring.

HINT: Du kan i en halvferdig sekvens holde greie på hvordan du er i forhold til parentesstrukturen med ett eneste tall.

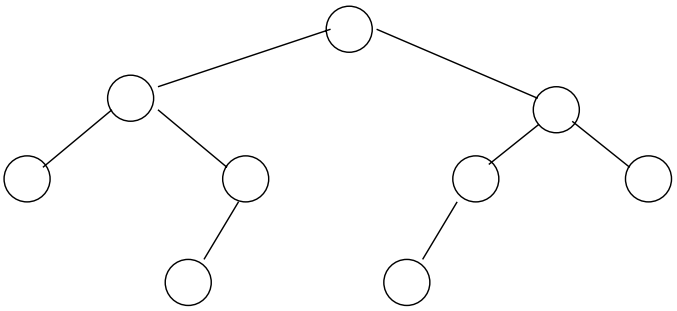
Oppgave 2 35 %

Vi skal se på binære trær, og er bare interessert i treets form, ikke i nodenes innhold. Nodene vil være representert med objekter av klassen

```
class node;
begin
  ref(node) vsub, hsub;      ! Venstre og høyre subtre ;
  ...;  ! Eventuelle andre data uten interesse for oss ;
end;
```

(Fortsettes på side 3.)

For et slikt binært tre skal vi se på den sekvensen som dannes når vi går gjennom nodene i innfiks rekkefølge, og skriver ut dybden for hver node. Denne sekvensen skal vi kalle treets "DI-sekvens" (for "dybde-innfiks-sekvens"). Eksempel: For følgende tre vil DI-sekvensen bli som angitt under:



DI−sekvensen: 2 1 3 2 0 3 2 1 2

2-a

Skriv en rekursiv prosedyre:

```
procedure DIsekvens(rot, ...); ref(node) rot; ... ;
```

Denne skal skrive ut DI-sekvensen for treet med angitt rot. (Du kan bruke `outint` på enkleste måte til utskriften.) Prosedyren skal virke også om treet er tomt.

2-b

Vi skal her gjøre det omvendte, nemlig å rekonstruere treet fra DI-sekvensen. Dette skal gjøres med en rekursiv prosedyre `rekBygg`, og for behagelighets skyld skal vi la det være en ytre prosedyre `byggTre` som brukeren skal kalle, og som skal levere roten til det ferdige treet som resultat. Det hele kan ha følgende skjelett:

```
ref(node) procedure byggTre(DI, n); integer array DI; integer n;
   ! DI er en integer array dimensjonert (1:n), og den inneholder
     DI-sekvensen til et tre med n noder ;
begin
   ref (node) procedure rekBygg(fra, til, ...); integer fra, til; ... ;
   ...
end;
```

Her skal parameterene `fra` og `til` til `rekBygg` angi det segmentet av arrayen DI som den skal bygge et (sub)tre av, og den skal levere roten til dette. Dette

(Fortsettes på side 4.)

skal den gjøre ved å søke (lineært) forfra i segmentet etter det tallet i DI-sekvensen som tilsvarer roten i subtreet, og så kalle `rekBygg` rekursivt for å bygge de to subtrærne til denne rotnoden.

Når du skriver prosedyren `byggTre` kan du anta at det som ligger i DI er en legal DI-sekvens til et tre. Prosedyren skal virke også for et tomt tre ($n = 0$).

Angi etterpå hvilke ekstra tester du må legge inn i programmet for at prosedyren også skal kunne gi feilmelding dersom innholdet av DI ikke er en legal DI-sekvens.

2-c

Vi er interessert i tiden prosedyren fra 2-b tar, uttrykt i O-notasjon som funksjon av lengden n av DI-sekvensen. Du skal angi dette for de to tilfellene nedenfor. Begrunn svaret, eventuelt ved å vise til liknende tilfeller i pensum.

- Dersom treet er rimelig balansert.
- Det verst mulige tilfellet for hver n.

2-d

MERK: Denne deloppgaven kan være litt vanskelig, så det kan lønne seg å ta den til slutt.

Problemstillingen er her den samme som i oppgave 2-b, men denne gangen skal du bygge treet ved å lese DI-sekvensen fra venstre mot høyre og danne treet etter hvert. Helst skal hele prosessen gå i tid $O(n)$.

- Du skal skrive en prosedyre `byggTre2` (med samme parametere etc. som `byggTre`) som utfører denne bygge-prosessen, og leverer roten av treet som svar. Du kan anta at DI-sekvensen er riktig. Begrunn at prosedyren virker.
- Du skal også angi med O-notasjon hvor mye tid din algoritme vil bruke, og begrunne dette kort.

HINT: Flere metoder kan brukes. Man kan benytte en rekursiv prosedyre eller man kan feks. ha en `ref(node)` hjelpearray, som er indeksert med dybdene 0, 1, 2, ... Grovt sett kan denne arrayen brukes til å holde tak i de nodene som ikke har fått på plass alle sine "naboer" (altså `vsub`, `hsub` og foreldren), og kanskje noen fler.

(Fortsettes på side 5.)

Oppgave 3 40 %

Vi skal i denne oppgaven ta for oss en spesiell form for urettede grafer og noen av deres subgrafer. I tillegg til informasjon om hvilke to noder en kant forbinder vil kantene også ha en *farge*. En kant kan enten ha farge blå, gul eller rosa. For hvert par av noder `u` og `v` kan det da finnes tre kanter, én for hver farge.

Som vanlig vil en subgraf G_i av en graf `G` inneholde et subsett av nodene i `G` og et subsett av kantene i `G`. En *fargesubgraf* G_i med farge `f` av en graf `G` er en subgraf G_i der:

- Alle kantene i G_i har farge `f`.
- G_i er forbundet (sammenhengende). Det vil si at for hvert par av noder `u` og `v` i G_i så finnes det en vei fra `u` til `v` i G_i.
- G_i er komplett. Det vil si at hvis det finnes en kant i `G` med farge `f` der den ene endenoden er med i G_i så vil også både denne kanten og den andre endenoden være med i G_i.

Merk at en node kan inngå i flere fargesubgrafer hvis disse grafene har forskjellig farge, mens en kant bare vil inngå i én. Merk også at en singel node er en fargesubgraf med farge `f` hvis det ikke finnes noen kanter med farge `f` i `G` som forbinder denne noden med andre noder.

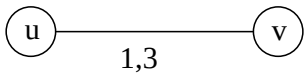
Vi skal anta at fargen i kantene er kodet som: `blå = 1, gul = 2, rosa = 3`.

Videre skal vi anta at nodene i grafen er nummerert fra `1` til `N`. Grafen skal være på naboliste-representasjon, og vi kan benytte oss av en `ref(Node) array G[1:N]` til å holde alle nodene. Med denne representasjonen er det naturlig at en kant mellom node `u` og `v` i `G` vil være representert av to kantobjekter, ett objekt i nabolisten til `u` og ett objekt i nabolisten til `v`.

I de programmene som du blir bedt om å skrive i de påfølgende deloppgaver må du selv angi de attributtene du trenger i nodene og kantene.

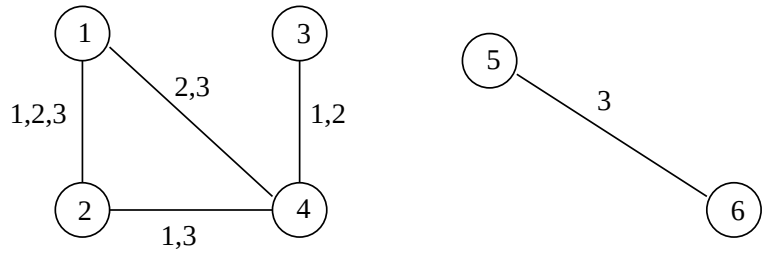
3-a

Merk først at hvis det går flere kanter mellom to noder, feks. både en rosa og en blå, kan vi tegne opp dette på følgende måte:



(Fortsettes på side 6.)

Finn alle rosa fargesubgrafer i grafen nedenfor:



Svaret skal du angi ved at du for hver rosa fargesubgraf skriver ut alle *kantene*. Kantene kan du angi som et trippel `(u,v,f)`. Hvis du finner en rosa fargesubgraf med bare en singel node skal du skrive ut *nodenummeret*.

3-b

Du skal nå programmere en:

```
procedure fargesubgrafer(G,f,N); ref(node) array G;
                                              integer f,N;
```

som finner alle fargesubgrafer i `G` med farge `f`, der antall noder i `G` er gitt av `N`. Resultatet av et kall på `fargesubgrafer` skal være en utskrift som for hver av disse fargesubgrafene skriver ut en passende ledetekst og deretter alle *kantene* i denne fargesubgrafen. Hvis du finner en fargesubgraf med farge `f` som bare har en singel node skal du skrive ut *nodenummeret*. Du kan anta at du allerede har en prosedyre:

```
skrivkant(k);ref(kant) k;
```

som skriver ut en kant`k`. Andre (lokale) prosedyrer som du eventuelt vil trenge må du programmere selv.

3-c

Vi skal så se på følgende problem for `G`: Gitt en farge `f`, en node `u` og en node `v` så skal man finne ut om det finnes en fargesubgraf med farge `f` som inneholder både `u` og `v`.

Vi skal anta at det blir svært mange slike forespørsler for `G` (med forskjellige farger `f` og noder `u` og `v`) så det er viktig å kunne avgjøre dette raskt og effektivt. Videre skal vi anta at man innimellom slike forespørsler stadig utvider `G` med nye kanter (men ikke nye noder).

(Fortsettes på side 7.)

- Gi et forslag til en datastruktur som gjør det lett å finne ut om to noder er i samme fargesubgraf med farge `f` og som også er enkel og effektiv å vedlikeholde med hensyn til at `G` stadig blir utvidet med nye kanter. Legg vekt på både plass og tidsforbruk. Begrunn svaret ditt kort.
- Du skal nå anta at du har datastrukturen du foreslo. Lag en boolsk prosedyre som tar to noder og en farge som parameter, og som returnerer med `true` hvis det finnes en fargesubgraf med den oppgitte farge-parameteren der de begge er med, ellers returneres `false`.
- Lag en prosedyre som oppretter datastrukturen du foreslo for `G`. Hva blir tidsforbruket for et kall på denne prosedyren ?

3-d

Vi skal nå se på noen spesielle veier mellom noder i `G`. En *polyfarget* vei er en vei der det ikke finnes to påfølgende kanter med samme farge. For eksempel vil en vei på tre kanter med først to blå kanter og så en rosa ikke være en polyfarget vei, men en vei med først en rosa, så en gul og så en rosa vil være det.

- Skriv en

  ```
  boolean procedure polyvei(u,v,...); ref(node) u,v;....;
  ```

  som returnerer med `true` hvis det finnes en *enkel* polyfarget vei mellom node `u` og `v`, ellers returneres `false`. Veien skal altså ikke ha løkker.
- Gi en kort begrunnelse for at hvis det finnes en enkel polyfarget vei fra `u` og `v` så vil algoritmen din faktisk finne den.

3-e

Vi skal nå se på tilsvarende problem som i oppgave 3-d men for rettede grafer med fargede kanter, der grafene ikke kan ha løkker.

- Skriv en

  ```
  boolean procedure rpolyvei(u,v,....);ref(node) u,v;....;
  ```

  som returnerer med `true` hvis det finnes en polyfarget vei fra `u` til `v`, ellers returneres `false`. Prosedyren skal ikke ha større orden enn $O(E)$, der E er antall kanter i grafen.
- Begrunn at denne prosedyren virkelig virker.

(Fortsettes på side 8.)

Oppgave 4 10 %

Et firma som leverer TV-kabler til boligfelt har to typer kabler, A og B. Kablen av type A er litt billigere, men til gjengjeld kan den ikke kuttes opp, men må legges i EN sammenhengende sløyfe som er innom alle husene, og som til slutt går tilbake til det punket der den startet.

Kablen av type B kan derimot deles opp og legges i stykker mellom husene. Kabel-stykkene må legges fra hus til hus slik at alle husene er forbundet med hverandre (direkte eller indirekte).

For å kunne gi gode tilbud vil firmaet bruke så lite kabel som mulig, og også beregne om det lønner seg å bruke kabeltype A eller B. Som en forberedelse til en leveranse vil de alltid dra ut på det aktuelle hus-feltet og måle opp for hvert par av hus hvor mye kabel det vil gå med for å strekke kabel direkte fra det ene huset til det andre. Vi lar n angi antall hus på feltet.

Ola og Kari jobber i firmaet. Ola påstår at han har en algoritme med tidsforbruk $O(n^3)$ som kan finne hvordan man kan legge kabel av type A slik at man bruker kortest mulig kabel. Kari påstår tilsvarende at hun har en algoritme med tidsforbruk $O(n^2)$ som finner hvordan man kan legge kabel B for å bruke minst mulig av denne.

Vurder påstandene til Ola og Kari. Begrunn svaret.