UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i	IN 115 og IN110 — Algoritmer og datastrukturer
Eksamensdag:	15 . mai 1997
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:

Dette oppgavesettet gjelder både In110 og In115. Alle oppgavene inngår for begge emnene. Oppgavesettet består av 2 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 somn gir leseren nøkkelen til forståelse av programmene. Du kan anta at leseren kjenner problemstillingen meget godt.

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

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.

(Fortsettes på side 2.)

Oppgave 1 Trær 55 %

Vi skal i denne oppgaven se på en form for søkestrukturer som er spesielt godt egnet til å håndtere alfabetiske nøkler bestående av strenger av bokstaver, altså ord. Typiske anvendelser av en slik søkestruktur vil være store ordbøker og programmer som sjekker ord for stavefeil.

For at opptegningen av disse trærne ikke skal ta for stor plass på arket vil vi i denne oppgaven bare benytte et lite utsnitt av alfabetet, vi benytter oss bare av bokstavene `d,e,r,t`. Eksempler på ord over dette alfabetet er: `et, ett, etter, det, der` etc.

For å ikke gjøre programmeringen under denne oppgaven altfor "fiklete" skal vi benytte oss av en matrise

```
character array Ord(1:maxo,1:maxb)
```

der `maxo` er en øvre grense for hvor mange ord vi kan ha, og `maxb` er en øvre grense for hvor mange bokstaver et ord kan inneholde. Ordene i ordlisten eller ord-databasen ligger da lagret i radene i denne matrisen. Matrisen vil være usortert. Den behøver ikke være fylt opp.

I tillegg skal vi ha en

```
integer array lengde(1:maxo)
```

der denne angir for hvert ord hvor mange bokstaver ordet inneholder.

Vi skal nå bygge opp en søkestruktur for å kunne lete etter ord i denne matrisen. Søkestrukuren vi skal bygge opp er et tre der hver interne node vil ha følgende form:

```
class node;
begin
    ref(node) array nesteb(1:4);
    intger index;
end;
```

I tillegg har vi en (svært rask) integer prosedyre `finnnr(c);char c;` der:

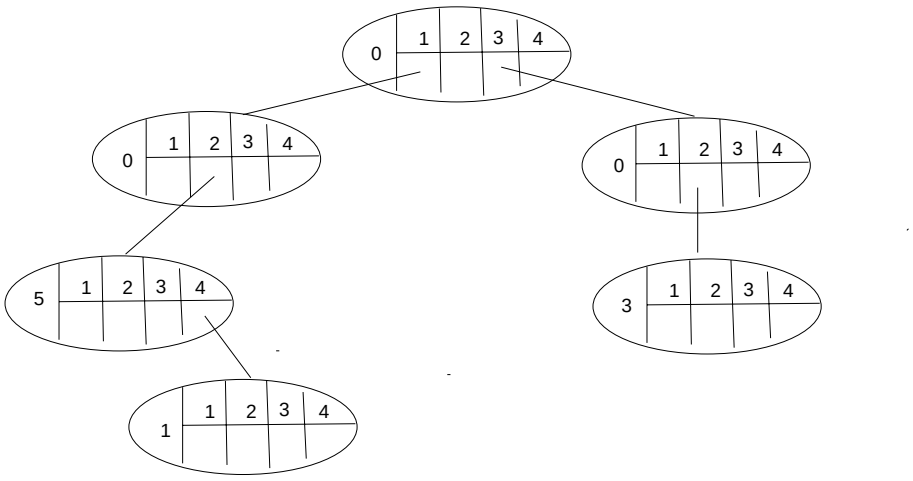
```
finnnr('d') = 1
finnnr('e') = 2
finnnr('r') = 3
finnnr('t') = 4
```

(Fortsettes på side 3.)

Gitt at matrisen inneholdt følgende ord på de seks første indeksene:

d	e	t			
e	t				
r	e				
d	e	t	t	e	
d	e				
e	t	t			

Da skulle vi etter å ha bygget opp søketreet med hensyn på ordene: `re, det` og `de` ha følgende tre:



Vi ser altså at ord med samme innledende sekvens av bokstaver (samme prefiks) deler en sti nedover i treet. Videre markeres slutten på et ord ved at `index` i noden angir hvor i matrisen `Ord` vi kan finne ordet. Ellers er `index = 0`. Vi ser også at siden et ord i seg selv kan være et prefiks til et annet så behøver ikke slutten på et ord bety at vi har en løvnode i treet.

Vi kan si at en bokstav *finnes* i en node hvis den tilhørende pekeren i arrayen `nesteb` ikke er `none`. For å søke etter et ord starter man i roten av treet, tar bokstav for bokstav i ordet, ser om bokstaven finnes og følger bokstavens neste-peker nedover i treet.

1-a

Gitt treet over skal du tegne inn situasjonen etter at ordene: `et, dette,ett` også er lagt inn i treet. Indeksene til ordene finner du fra matrisen gitt over.

(Fortsettes på side 4.)

1-b

Skriv en prosedyre:

```
integer procedure insert(nr,rot);
integer nr;
ref(node) rot;
begin .... end;
```

som setter ordet angitt av matrisen `Ord` på rad `nr` inn i treet angitt av `rot`. Hvis ordet finnes fra før returneres indeksen til den allerede eksisterende forekomsten i matrisen `Ord`, ellers returneres `0`.

1-c

Vi antar at vi har bygget opp datastrukturene som beskrevet over, I tillegg har vi en peker til starten av søketreet: `rot`. Skriv en prosedyre:

```
procedure alfaliste(mb,ro); integer mb; ref(node) rot;
```

slik at `alfaliste(maxb,rot` skriver ut alle ordene i matrisen `Ord` i alfabetisk rekkefølge. Du skal *ikke* ikke benytte deg av matrisen. Prosedyren kan derimot naturlig benytte seg av en `character array ford(1:maxb)`, en `integer plengde` og en indre rekursiv prosedyre `traverser(n); ref(node) n;`.

Du kan anta at det finnes en prosedyre

```
skrivord(tekst,le); character array tekst; integer le;
```

som skriver ut innholdet av arrayen `tekst` fra posisjon `1` til `le` som en tekst. Du skal *ikke* programmere `skrivord`.

1-d

Du skal anta her at *enhver* sekvens av bokstaver utgjør et ord.

Anta vi har lagt inn 20 ord i søketreet, alle med lengde mindre eller lik 4.

Hva blir det maksimale antallet noder vi kan få, og hva blir det minimale ?

Hvis du benyttet et vanlig binært søketre for å lagre de `20` ordene, hva ville den maksimale og minimale høyden bli på treet ?

Kan du si noe om hvorfor denne søkestrukturen egner seg spesielt godt til *søking* når man har svært mange ord i ordlisten eller ord-databasen ?

(Fortsettes på side 5.)

1-e

Legg merke til at det kan finnes flere noder i treet som bare har *en* etterfølger, og der denne igjen bare har *en* etterfølger etc. inntil man støter på en løvnode. For å optimalisere treet med hensyn på antall noder kunne man tenke seg å kvitte seg med flest mulige noder på slike stier.

Kan du gi ett forslag til hvordan dette kan gjøres ?

Skriv en prosedyre `insert` som beskrevet i oppgave 1-b, men der treet er optimalisert (slik du har foreslått) både før og etter innsettingen.

Det kan fort bli mye "småplukk" i denne prosedyren, så angi gjerne en skisse eller forfiningsvariant av prosedyren først. Selve programmeringen kan det kanskje lønne seg å vente med til du har løst deloppgaven under, og eventuelt del 2.

1-f

Hvis hashing ble brukt som et alternativ til den ovenforstående datastrukturen, hva slags hashing ville du ha brukt ? Hva ville vært en passende tabell størrelse ? Presiser hvilke forutsetninger du har gjort for valget ditt.

Gi et eksempel på en god hashfunksjon for en ordbok.

Oppgave 2 Grafer (45 %)

Vi skal i denne oppgaven se på aktivitetsgrafer.

Vi tenker oss at vi har aktivitetene representert ved en rettet graf der hver node tilsvarer en aktivitet og hver kant `(u,v)` representerer at aktivitet `v` ikke kan starte før aktiviteten `u` er avsluttet. Vi skal anta at grafen er cycelfri.

Hver aktivitet er gitt et unikt nummer og grafen er representert ved nabolister. For å representere utgående kanter fra en node skal vi bruke klassen:

```
class kant;
begin
   integer til;
   ref(kant) neste
end;
```

(Fortsettes på side 6.)

Grafen kan således representeres som en `ref(kant) array Node(1:antalln)` der `Node(i)` inneholder en peker til en liste som angir alle etterfølgerene til aktivitet med nummer `i`.

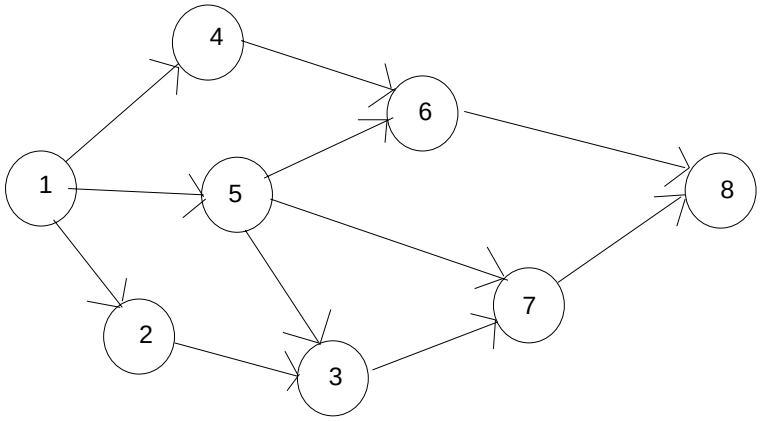
Vi har også en `integer array Tid(1:antalln)` som angir tiden hver aktivitet tar.

2-a

Den første oppgaven er å ordne aktivitetene i *faser*. Fasene er grupper av aktiviteter som skal startes opp samtidig, og der fasen avsluttes når alle de respektive aktivitetene er avsluttet. Fase-inndelingen har følgende definisjon:

Fase `1` består av *alle* aktiviteter som kan startes umiddelbart. Når fase `1` er avsluttet kan fase `2` starte opp. Fase `2` består da av *alle* aktiviteter som nå kan startes. Generelt vil fase `i+1` bestå av alle aktiviteter som kan startes etter at fase `i` er avsluttet.

Gitt grafen grafen under skal du skrive opp hvilke aktiviteter som hører til hvilke faser.



2-b

Din oppgave er nå å programmere en prosedyre

```
procedure fase(Node,Tid,antalln);
ref(kant) array Node;
integer array Tid;
integer antalln;
begin .....   end;
```

`antalln` skal inneholde lengden på arrayen `Node` og `Tid`.

(Fortsettes på side 7.)

Når prosedyren blir kalt med `Node,Tid,antalln` som parametere skal den sørge for at hver fase blir skrevet ut sammen med sine aktiviteter og tidligste avslutningspunkt.

Tidligste avslutningstid for enhver aktivitet er gitt ved følgende forutsetning: En aktivitet i fase `i` skal aldri begynne før alle aktivitetene i faser før `i` er avsluttet.

Ut av programmet skal du altså ha en utskrift:

```
Fase 1: <alle noder og deres tidligste avslutningstid i fase 1>
Fase 2: <alle noder og deres tidligste avslutningstid i fase 2>
.
.
Fase n: <alle noder og deres tidligste avslutningstid i fase n>
```

Du må selv tenke ut hvilke datastrukturer og variable du trenger i tillegg til de som er oppgitt.

Det er viktig at programmet ditt løser fase-inndelingen og du skal gi en kort begrunnelse for hvorfor og hvordan programmet ditt virker med hensyn på dette.

Vi kan også anta at det er mange noder i grafen, men den er ikke tett: Antall kanter er langt færre enn N^2. Forsøk derfor å ta hensyn til tidskompleksiteten i denne oppgaven.

Hva blir tidskompleksiteten for programmet ditt ?

2-c

Hva skjer med programmet ditt under oppgave 2-b hvis det likevel skulle finnes cycler i grafen ?

2-d

Vi skal nå arbeide med en nabomatrise for grafen fra oppgave 2-a. Grafen kan også her antaes å være uten cycler. Matrisen kan vi kalle `Mnode`.

Vi påstår at vi kan modifisere Floyd-algoritmen slik at hvis vi kjører denne på `Mnode` kan vi for alle noder `u` og `v` finne den lengste veien fra `u` til `v`. Den lengste veien er her den veien som har *flest antall kanter*. Se ellers oppgave 2-f om å gi en begrunnelse for dette.

(Fortsettes på side 8.)

Din oppgave blir å programmere denne varianten av Floyd. Du kan anta at det finnes en prosedyre `transform(..)` der `transform(Node,Mnode,antalln)` lager en representasjon av grafen gitt av `Node` i matrisen `Mnode` der `antalln` angir lengden på `Node` og lengdene på `Mnode`. Du skal *ikke* programmere `transform`, men du må presisere hvordan matrisen `Mnode` bør se ut etter at `transform` er ferdig for at din variant av Floyd skal virke.

2-e

Du kan nå anta at du har den modifiserte Floyd algoritmen fra oppgave 2-d. Vi skal videre anta at det er svært mange noder i grafen, og at vi ønsker å redusere antall kanter så langt som mulig uten at avhengigheter mellom aktiviteter forsvinner. Vi sier at:

- En aktivitet `v` er avhengig av en aktivitet `u` hvis det går en vei fra `u` til `v` i grafen.

Vi ønsker altså å fjerne kanter på en slik måte at hvis `v` var avhengig av `u` før fjerningen så skal `v` fremdeles være avhengig av `u` etter fjerningen.

Kan du se en måte å bruke resultatene fra den modifiserte Floyd algoritmen slik at vi får identifisert alle kanter vi kan fjerne uten at avhengigheter blir borte ?

Hvis vi hadde fjernet alle overflødige kanter som beskrevet over vil vi da få samme fase-inndeling som før vi fjernet kantene ?

Gi en kort begrunnelse for svarene dine.

2-f Kan puffes

Formuler invarianten for den ytterset løkka i den modifiserte Floyd algoritmen fra 2-d og gi en begrunnelse for at : Utføringen av programsetningene inne i den innerste løkken gjør at invarianten blir bevart *og* invarianten medfører at når programmet terminerer vil vi ha funnet lengste vei fra-alle-til-alle, gitt at grafen er cycelfri.

Lykke til!

Almira Karabeg Anne Salvesen