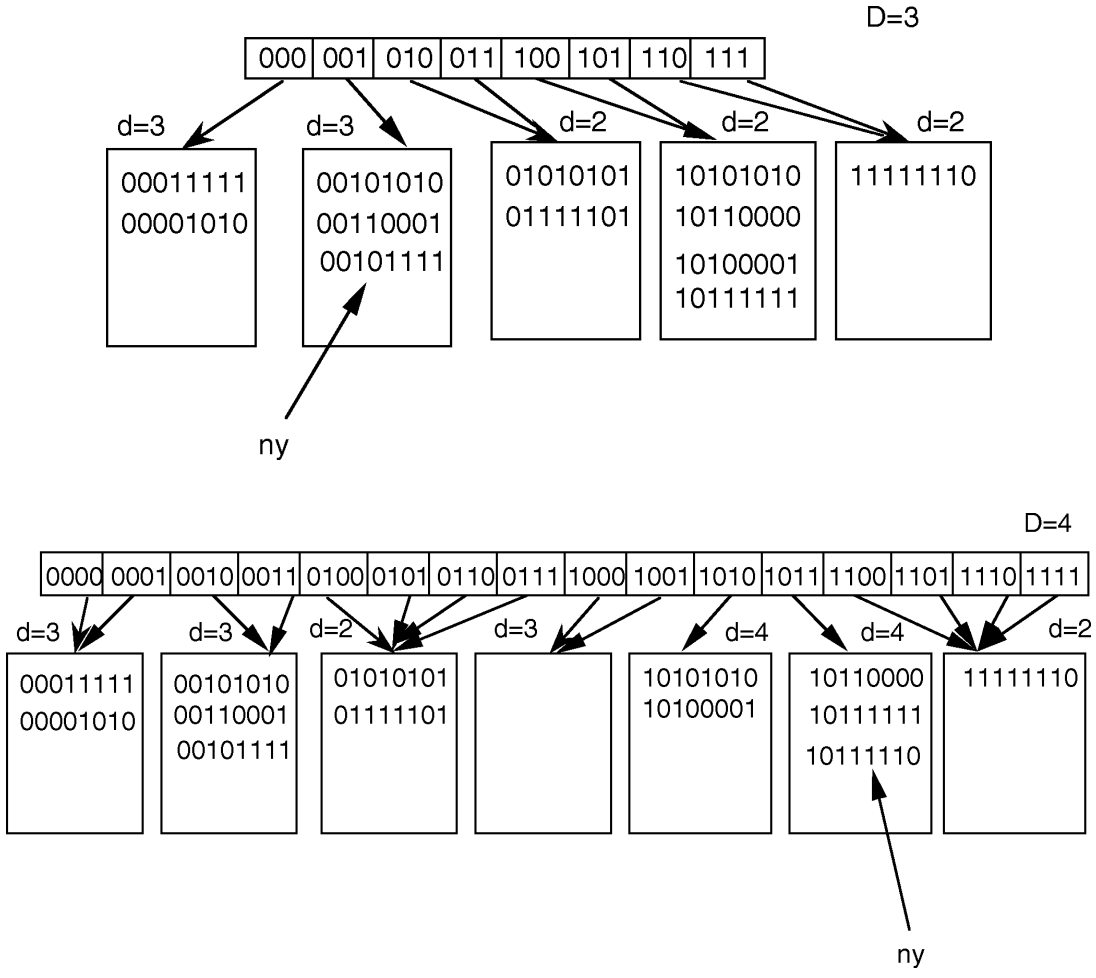
Eksamen IN 110, 15. mai 1995
Forslag til svar

Del 1

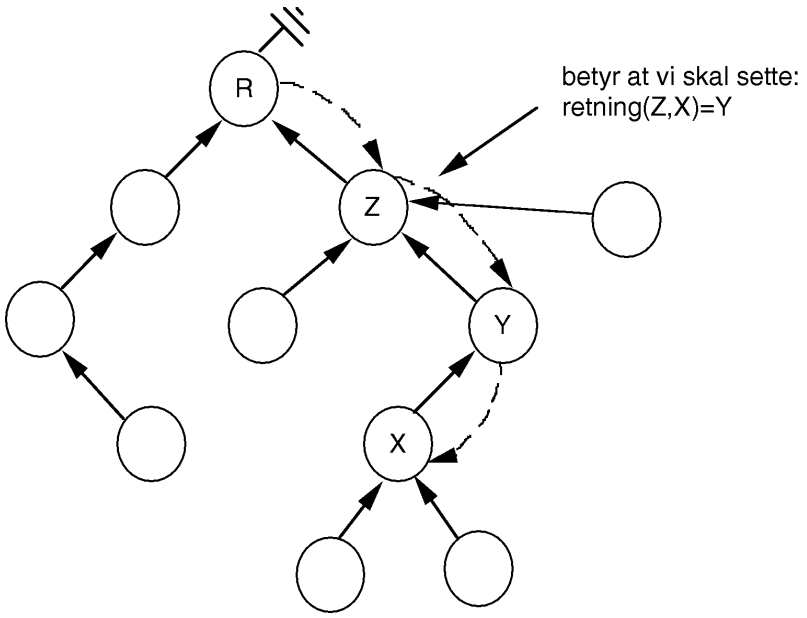
1-a



For å ha fullgodt svar bør man hvertfall ha satt på d-verdiene på boksene, men det skal vel ikke være dramatisk trekk i karakteren om dette ikke er gjort.

1-b

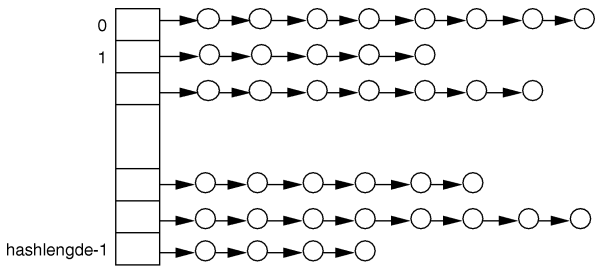
Fra hver node følger vi veien opp til roten, og setter inn forlengs-pekere. På figuren under er dette de stiplede pekerne, og ut fra det skulle programmet være greit.




```
procedure fyllretning(frd, retning);
      integer array frd;
      integer array retning;
begin
      integer x;
      for x:= 1 step 1 until N do
      begin
            y:= x;
            while frd(y)<>0 do    ! Eller: "y<>R" ;
            begin
                  retning(frd(y), x):= y;
                  y:= frd(y);
            end;
      end;
end;
```

1-c

Under forholdene gitt i oppgave vil vi få lange lister ut fra hver hash-indeks, og ved en god hash-algoritme vil disse bli omtrent like lange, og dermed av lengde ca. 'n/hashlengde'. I gjennomsnitt vil vi måtte lete halve listen (om elementet er der, ellers hele listen). Dette gir en søketid $O(n/hashlengde)$, og siden hashlengden er konstant, er riktig svar $O(n)$. Merk at separat kjeding også kalles "open hashing" i læreboka.



1-d

Det *er* håp om at Ole kan klare å sortere 5 elmenter med 7 sammenlikninger ut fra følgende: For å kunne gjøre de riktige ombyttinger i sekvensen slik at den blir sortert, må man finne ut hvilken permutasjon input-sekvensen er i forhold til den riktig sorterte sekvensen. Det er i alt 5!=120 mulige slike permutasjoner, og man kan se på disse som bladene i et desisjonstre (se side 242 i læreboka). Et lemma i boka sier da at et desisjonstre må ha høyde med minst $\lceil \log_2 120 \rceil$, som er 7. Så ut fra dette er det håp (og nærmere fikling viser at det også er mulig, men dette var det ikke meningen at man skulle gå inn på). Teorem 7.5 i læreboka kan også brukes.

Del 2

2-a

```
ref (node) procedure gentre(n,S);
integer n; character array S;
begin
    ref (node) array stakk(1:n+1);    ! Max n+1 elementer;
    integer topp;
    integer i;        ref (node) rn;
    for i:=1 step 1 until 2 * n + 1 do
    begin
        if S(i) = 'N' then
        begin
            rn:- new node;
            rn.vsub:- stakk(topp-1);
            rn.hsub:- stakk(topp);
            topp:= topp-1; stakk(topp):- rn;
        end else
        if S(i) = 'T' then
        begin
            topp:= topp+1; stakk(topp):- none;
        end else ERROR; ! Egentlig unødvendig, antar riktig input ;
    end;
    ! Nå skal 'topp' være 1;
    gentre:- stakk(1);
```

end;

Vi tillater altså at man antar at input-sekvensen er riktig. Når det gjelder stakken kan jo den med fordel programmeres separat, men for å få helt full uttelling må man da også programmere stakken, enten ved en array (max lengde $n + 1$) eller ved en liste-struktur. Sentrale deler av programmet over kan da ta seg slik ut:

```
if S(i) = 'N' then
begin
      rn:- new node;
      rn.hsub:- pop;
      rn.vsub:- pop;
      push(rn);
end else
if S(i) = 'T' then
      push(none)
else ...;
```

Alternativ løsning: Lest bakfra vil en utvidet postfiks form angi det speilvendte utvidede treet i prefiks form. Det gir mulighet til å lese sekvensen bakfra med "recursive descent"-aktig prosedyre. Om dette er gjennomført så er det en helt OK løsning.

2-b

```
procedure lagalle(n); integer n;
begin
      character array S(1:2 * n + 1);
      integer antN, antT;    ! Skal anta: antN+antT = k-1 og antN<antT og antT≤n+1;
      procedure lagrek(k); integer k;
      begin
            if k>2*n+1 then bruksekv(n,S) ! Mer avskjæring unødvendig ;
            else
            begin
                  if antT<n+1 then
                  begin
                        antT:= antT+1;
                        S(k):= 'T';
                        lagrek(k+1);
                        antT:= antT-1;
                  end;
                  if antN+1<antT and antN<n then
                  ! Siste del av testen er unødvendig, siden antT<=n+1 ;
                  begin
                        antN:= antN+1;
                        S(k):= 'N';
                        lagrek(k+1);
                        antN:= antN-1;
                  end;
            end;
      end;
```

```
        antN:= 0; antT:= 0; ! Egentlig unødvendig i Simula ;
        lagrek(1);
end;
```

Man kunne her kalle 'bruksekv' ett nivå tidligere. Dette ville redusere antall prosedyrekall, men ville grumse til programmet litt. Disse løsningene regnes som likeverdige. Man kunne også godt klare seg uten enten 'antT' eller 'antN' på grunn av invarianten angitt i toppen av programmet.

2-c

Dybden til hver av nodene i f.eks. t1 vil øke med 1 når vi ser på dem som noder i treet t. Deres bidrag til dybdesummen i treet t vil derfor være 'sumd(t1)+ant(t1)', og tilsvarende for t2. Roten i t har dybde null i t, og bidrar derfor ikke til sumd(t). Dette skulle vise formlen i oppgaven. For at dette skal stemme også om t1 eller t2 er tomme må 'sumd' av et tomt tre være null.

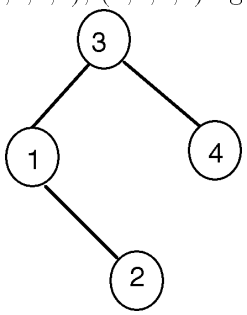
2-d

Vi løser oppgaven ved å beregne 'sumd' og 'ant' for hele treet, og dividerer til slutt for å få gjennomsnittet. For å finne disse bruker vi samme stakk-filosofi som i 2-a, og bruker formlen fra 2-c. Programmet kan da f.eks. bli som angitt under. Merk at vi nå har antatt at input-sekvensen er helt riktig, og at man selvfølgelig kan gjøre de samme variasjoner med programmet som omtalt for 2-a.

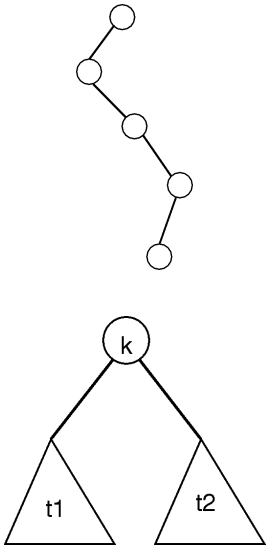
```
real procedure snittdybde(n, S);
        integer n; character array S;
begin
        integer array ant, sumd(1:n+1);
        integer topp;
        integer i;
        for i:=1 step 1 until 2 * n + 1 do
        begin
                if S(i)='N' then
                begin
                        sumd(topp-1):= sumd(topp-1)+ant(topp-1)+sumd(topp)+ant(topp);
                        ant(topp-1):= ant(topp-1)+ant(topp)+1;
                        topp:=topp-1;
                end else    ! Vi gjør ingen testing av input;
                begin
                        topp:= topp+1;
                        sumd(topp):= 0;
                        ant(topp):= 0;
                end;
        end;
        snittdybde:= sumd(1)/ant(1);    ! Antar n≥1;
end;
```

2-e

Kjernen ligger her i at det første gjennomsnittet er tatt over alle forskjellige trær (med hvert tre lik vekt), mens det andre er tatt over sekvenser som settes inn (med lik vekt på hver sekvens). Forskjellen oppstår fordi det i det andre tilfellet er mange sekvenser som gir samme søketreet. F.eks. vil både (3,1,2,4), (3,1,4,2) og (3,4,1,2 gi treet:



Dette at flere sekvenser gir samme treet gjelder kraftigere for de lavere (og balanserte) trærne enn for de høye. F.eks. vil det for trær av den første typen angitt under bare være én sekvens som gir akkurat dette søketreet. For den siste typen trær gjelder derimot at de vil framkomme av sekvensen (k, S1, S2) der S1 gir t1 og S2 gir t2, samt av enhver sekvens (k,S), der S er en fletting av S1 og S2.

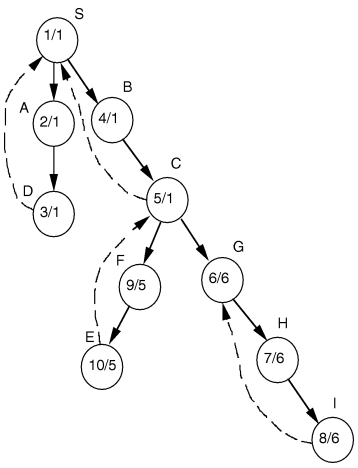


Totalt vil det bevirke at de lave trærne får større vekt når vi regner over sekvenser, og dette bevirker altså at gjennomsnitlig høyde her blir mindre, nettopp slik formelene i oppgaven påstår.

Del 3

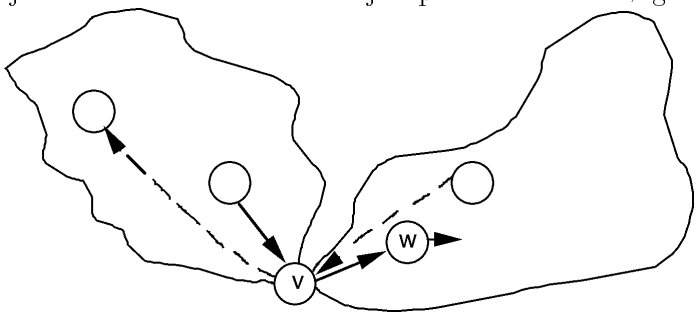
3-a

Under har vi angitt én løsning, men andre er selvfølgelig også mulige om man velger andre rekkefølger ut fra hver node. Man kunne selvfølgelig like gjerne lage en tegning som liknet mer på den opprinnelige grafen. Artikulasjonspunktene er altså nodene S, C og G.



3-b

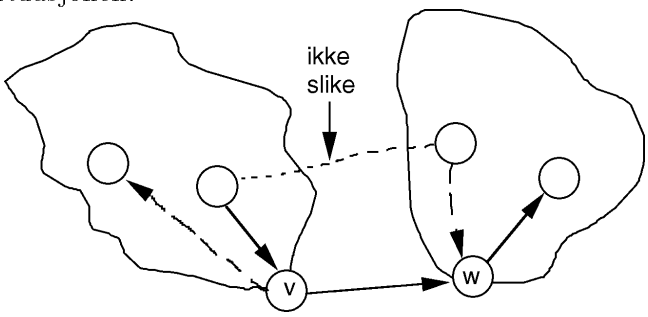
Den typiske situasjonen når vi finner artikulasjonspunkter er som følger:



Testen for å detektere denne situasjonen er da:

if w.low >= v.num then .... (samt litt spesialbehandling i start-noden).

Vi er nå ute etter situasjonen:



Denne kan vi detektere ved å forandre linje 16 til:

if w.low > v.num then <(v,w) er artikulasjonskant>

eller

if w.low >= w.num then <(v,w) er artikulasjonskant>

Når vi bare skal ha tak i artikulasjonskanter kan vi også fjerne spesial-behandlingen av start-noden, linje 10 – 14.

Merk at det *ikke* er nok å teste på om både v og w er artikulasjonspunkter. Det er lett å finne moteksempler til dette.