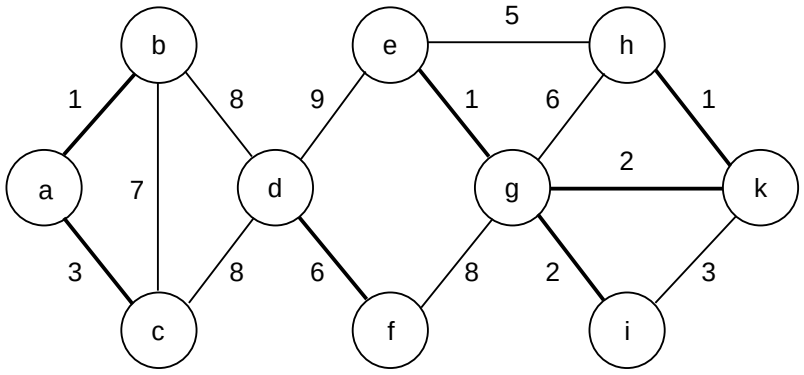
Svarforslag til eksamen IN 110, 16. mai 1994

Del 1

Oppgave 1

De kantene som vil være plukket når *b*–*c* er behandlet er markert på følgende figur:



Del 2

Oppgave 2-a

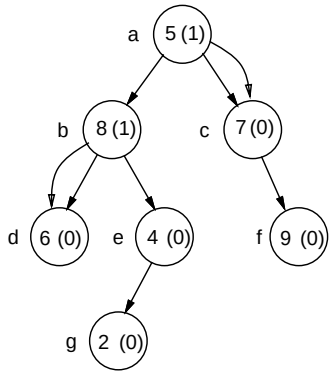
Legg denne prosedyren inn i klassen node:

```
procedure oppdater;
 if vsub==none or hsub==none then
  begin sti:- none; avst:= 0 end else
 if vsub.avst <= hsub.avst then
  begin sti:- vsub; avst:= vsub.avst+1 end
 else
  begin sti:- hsub; avst:= hsub.avst+1 end;


procedure stifinner(t); ref(node) t;
 if t=/=none then
  begin
     stifinner(t.vsub); stifinner(t.hsub);
     t.oppdater
  end;
```

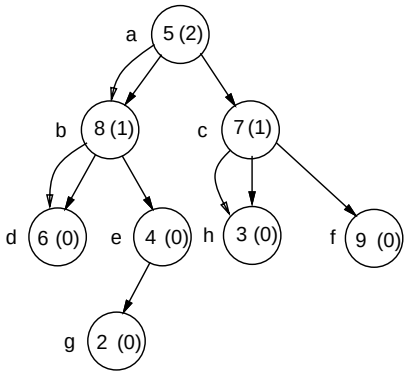
Oppgave 2-b

Attributtet `avst` er gjengitt i parentes i hver node. Der `avst`=0 er `sti`==`none`, `sti`-attributtet er utelatt for disse tilfellene. Forøvrig er `sti`-attributtet gjengitt med buet pil.

Før innsetting av h:



Etter innsetting av h:



Oppgave 2-c

```
procedure settinn(t,w); ref(node) t; integer w;
 if t.sti=/= none then
  begin
     settinn(t.sti,w);
     t.oppdater
  end
 else if t.vsub==none then
  begin
     t.vsub:- new node(w); t.oppdater
  end
 else
  begin
     t.hsub:- new node(w); t.oppdater
  end;
```

Faktisk kan oppdateringen stanse hvis vi på et tidspunkt kommer til en node der `avst` forblir uendret etter `t.oppdater`. Dette skjer hvis en eller flere av nodene langs stien fra roten og nedover, hadde andre nærmeste åpninger enn den som stien pekte ut.

Oppgave 2-d

Vi endrer prosedyren oppdater i klassen node slik:

```
procedure oppdater;
 if vsub==none or hsub==none then
  begin nåråpen:- none; avst:= 0 end else
 if vsub.avst <= hsub.avst then
  begin
    if vsub.avst=0 then nåråpen:- vsub
                   else nåråpen:- vsub.nåråpen;
    avst:= vsub.avst+1;
  end
 else
  begin
    if hsub.avst=0 then nåråpen:- hsub
                   else nåråpen:- hsub.nåråpen;
    avst:= hsub.avst+1;
 end;
```

Dermed kan `fjern`-prosedyren utformes som følger:

```
procedure fjern(t,rot); name rot; ref(node) t,rot;
 if t.nåråpen=/=none then
  begin
     ! Erstatt t.verdi med nærmeste åpne nodes verdi og fjern sistenevnte;
     t.verdi:= t.nåråpen.verdi; fjern(t.nåråpen)
  end
 else if t==rot then
  begin
     ! Roten skal fjernes; ny rot er dens (ene)barn;
     rot:- rot.enebarn;
     if rot =/= none then rot.foreld:- none;
  end
 else
  begin
     ref(node) u, e;

     ! En indre åpen node (som ikke er roten) skal fjernes.
       Treet lappes sammen og oppdateres langs veien mot roten;

     u:- t.foreld;  ! Vet: u =/= none ;
     e:- t.enebarn; ! Kan være none ;

     ! Fjerning og sammenlapping;
     if u.vsub==t then u.vsub:- e else u.hsub:- e;
     if e =/= none then e.foreld:- none;
```

```
    ! Oppdatering av nærråpen og avst lags veien mot roten:  ;
    if u.nærråpen=/=none then
     begin
        ! Spesialbehandling dersom t.foreld blir en åpen node
          etter fjerningen (fordi e == none): Da skal oppdateringen
          fortsette oppover i treet. Dersom t.foreld ikke blir en åpen
          node, kan t.foreld oppdateres på samme måte som de øvrige;

        if e == none then
         begin u.nærråpen:- none; u.avst:= 0; u:- u.foreld end;

        ! Oppdateringen kan avbrytes dersom en åpen node påtreffes ;

        while u=/=none and then u.nærråpen=/=none do
         begin
            u.oppdater; u:- u.foreld;
         end
     end
  end;
```

Oppgave 2-e

A. Anta at det fullstendig balanserte treet inneholder $n = 2^m - 1$ noder. Rotens `nærråpen` vil i utgangspunktet peke til det bladet som ligger lengst til venstre. Når roten "fjernes", blir derfor i stedet dette bladet fjernet og `rot.nærråpen` vil bli satt til å peke på dette bladets forelder. Ved neste fjerning blir så denne (lokale) foreldrenoden fjernet, treet lappes sammen, og neste `rot.nærråpen` blir den fjernede nodens gjenværende barn. Prosessen vil fortsette med å fysisk fjerne noder i rotens venstre subtre inntil roten selv blir en åpen node, først da vil roten selv fysisk bli fjernet. Prosedyren må følgelig kalles $2^{m-1} - 1$ ganger før roten blir en åpen node; ved det 2^{m-1}ste kallet fjernes roten fysisk.

B. Det fremgår fra diskusjonen under A at det tidspunktet hvor treet inneholder færrest noder i forhold til høyden, er når rotnoden er blitt en åpen node. Den har da et tomt venstre subtre og et høyre subtre som er perfekt balansert. Siden høyden i det totale treet er h, er antall noder i høyre subtre $2^h - 1$ og antall noder totalt 2^h. (Til sammenlikning inneholder altså det fullstendig balanserte treet $2^{h+1} - 1 = 2 \cdot 2^h - 1$ noder. Særlig ubalansert blir treet derfor aldri.)

Del 3

Oppgave 3-a

```
procedure fyllnabomatrise(graf, an, G);
   ref(node) array graf; integer an; boolean array G;
begin
   integer i, k;   ref(node) rn;

   for i:= 1 step 1 until an do
   begin
      rn:- graf(i);
      for k:= 1 step 1 until rn.ak do
```

```
         G(i, rn.kant(k).nr):= true;
   end;
end;
```

Oppgave 3-b

```
procedure settakt1(G, graf, an, a, b);
   boolean array G;
   ref(node) array graf; integer an;
   ref(node) a, b;
begin
   integer i, anr, bnr;   ref(node) rn;

   anr:= a.nr ; bnr:= b.nr;

   for i:= 1 step 1 until an do
   begin
      if G(anr,i) and G(i, bnr) then graf(i).aktuell:= true
                                else graf(i).aktuell:= false;
   end;
end;
```

Her kunne den innerste setningen eventuelt forenkles til:

```
    graf(i).aktuell:=  G(anr,i) and G(i, bnr);
```

Oppgave 3-c

Ved å gjøre et søk av enkleste type (f.eks. dfs eller bfs) fra noden a kommer vi til nøyaktig de noder som kan nåes ved en vei fra a. Om vi gjør et tilsvarende søk fra b der vi følger kantene bakover fra b (altså tenker oss at alle kanter er snudd og gjør et tilsvarende søk som fra a), kommer vi til nøyaktig de noder c slik at det finnes en vei fra c til b i den opprinnelige grafen. De noder der 'aktuell' skal settes er de som både har en vei fra a og en vei til b, og det blir de derved de vi kommer til i begge de to søkene.

Det er egentlig helt vilkårlig hva slags søk vi gjør, bare vi får satt et merke i de nodene vi kommer til i hvert av søkene. Vi har to boolske variable vi kan disponere, nemlig `merke` og `aktuell`. Vi kan derved bruke dem i h.h.v. søket forover fra a og søket bakover fra b, og til slutt gå gjennom hver node 'nd:- graf(i)' og gjøre noe i retning av:

```
    nd.aktuell:= nd.aktuell and nd.merke;
    nd.merke:= false;  ! Om man vil "nullsette" denne ;
```

Det er vel greiest å gjøre et rekusivt dybde-først-søk, der vi også kan bruke det merket vi setter til ikke å gå inn i samme node to ganger. Dersom vi bruker en nabomatrise til å orientere oss i grafen, tar det tid $O(an)$ å finne alle etterfølgere til en node og tid $O(an)$ å finne alle forgjengerne til en node. Dermed vil det å gjøre et dybde-først-søk ta tid $O(an^2)$. Vi får også to gjennomganger av nodene (initialiseringen og settingen av `aktuell`), men disse tar bare tid $O(an)$, så totalt tar det hele tid $O(an^2)$. Om man regner med oppsettingen av nabomatrisen eller ikke spiller ingen rolle, da den hvertfall ikke tar mer tid enn $O(an^2)$.

Ser vi på den løsningen som er foreslått i 3-b, så blir det tyngste leddet her å utføre Warshall-algoritmen. Dette tar nemlig tid $O(an^3)$, mens de andre tingene (oppsett av nabomatrise og settingen av `aktuell` i alle nodene) ikke tar mer tid enn $O(an^2)$.

Som konklusjon får vi derved at metoden under 3-c er bedre, hvertfall for store verdier av `an`, siden den tar tid $O(an^2)$ mens den under 3-b tar tid $O(an^3)$.

Oppgave 3-d

```
procedure settakt2(graf, an, a, b);
   ref(node) array graf; integer an;
   ref(node) a, b;
begin
   procedure rekakt(nd); ref(node) nd;
   begin
      integer i;

      nd.merke:= true; ! Vi har vært her ;
      nd.aktuell:= false ! Blir senere satt til true om b kan nåes ;

      if nd == b then nd.aktuell:= true else
      begin
        for i:= 1 step 1 until nd.ak do
        begin
          if not nd.kant(i).merke then rekakt(nd.kant(i));
          if nd.kant(i).aktuell then nd.aktuell:= true; ! b kan nåes ;
        end;
      end;
   end;

   rekakt(a);  ! Dette setter det hele i gang;
end;
```

Kommentar: I oppgaven var `settakt2` angitt som en boolsk prosedyre, noe som stammet fra en tidligere versjon av denne oppgaven, og som ikke gav helt mening slik den til slutt ble. Det har forhåpentligvis ikke virket for forvirrende.

En kort forklaring på hvorfor prosedyren over virker: Siden grafen ikke har løkker, vil den rekkefølgen vi "forlater" nodene i være en baklengs topologisk sortering. Dette gjør riktigheten av prosedyren "nesten opplagt", men for å få argumentet helt godt påstår vi at følgende invariant gjelder:

INVARIANT: De nodene der prosedyren `rekakt` er kalt og har terminert vil ha attributtet `aktuell` satt hvis og bare hvis det går en vei fra noden til b.

Grunnen til at denne invarianten forblir sann når prosedyren terminerer for nok en node er at dette prosedyrekallet da har sett på alle etterfølgere, og for disse var prosedyren enten allerede kalt (og dermed terminert), eller den ble kalt fra vårt prosedyrekall, og er da også nå terminert. Vi så på etterfølgernes `aktuell` etter at det tilsvarende prosedyrekallet er terminert (og dermed fulgte invarianten), og det må derfor være riktig å sette vår `aktuell` dersom monst en av etterfølgernes aktuell var satt.

Siden vi kommer til å kalle prosedyren nøyaktig i de nodene som kan nåes fra a (i de resterende nodene vil `aktuell` bli stående på false), vil aktuell da være riktig satt når alle

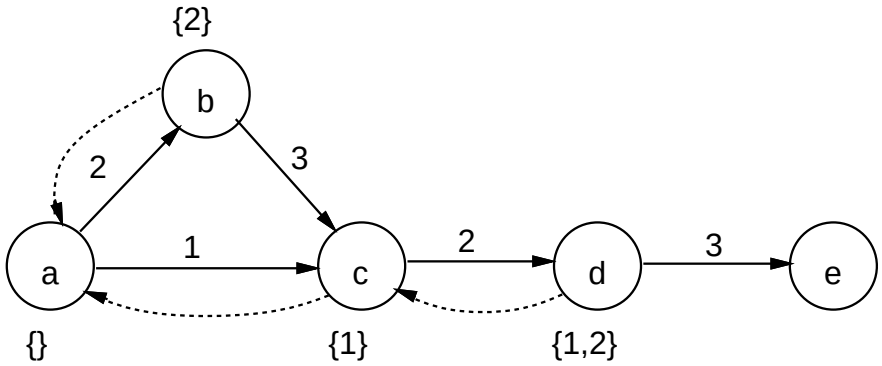
kall er terminert.

Oppgave 3-e

```
procedure F_vei_let(svar, an, a, b, F, af);
ref(node) array svar; ref(node) a,b; integer an, F, af;
begin
   integer array brukt(1:af);
   integer fargant;

   procedure reklet(nd, k); ref(node) nd; integer k;
   begin
      integer i;

      nd.merke:= true; ! Angir at denne noden er med i den foreslåtte vei ;
      svar(k):- nd; if nd == b then goto FUNNET;

      for i:= 1 step 1 until nd.ak do
      begin
         if nd.kant(i).aktuell  and    ! Test om b kan nåes ;
            not nd.kant(i).merke then  ! Test om denne er med i veien ;
           begin
            if brukt(nd.farge(i)) = 0 then fargant:= fargant+1;
            brukt(nd.farge(i)):= brukt(nd.farge(i)) + 1;

            if fargant <= F then reklet(nd.kant(i), k+1);

            brukt(nd.farge(i)):= brukt(nd.farge(i)) - 1;
            if brukt(nd.farge(i)) = 0 then fargant:= fargant-1;
           end;
      end;
      nd.merke:= false; svar(k):- none;
   end;

   b.aktuell:= true; ! Siden dette ikke var helt tydelig definert ;
   reklet(a, 0);
  FUNNET:
end;
```

Oppgave 3-f

Svaret her er at man hvertfall ikke uten videre kan bruke Dijkstras filosofi for å finne de veier som bruker færrest mulig farger. En måte å se dette på er å studere eksempelet angitt under, der tallene på kantene angir kantens farge.



Anta her at vår startnode er 'a', og at vi i henhold til en Dijkstra-aktig algoritme har gjort oss ferdig med nodene 'a', 'b', 'c' og 'd' (altså at 'known' er satt for disse). I stedet for variablen 'dist' ('avst') i nodene er det da rimelig å ha mengden av de farger som er brukt langs beste vei fram til denne node, og denne er angitt i forbindelse med hver av de ferdige nodene. Vi har også angitt (med stiplet pil) det rotrettede treet som angir de korteste veier til (eller fra) 'a'.

Etter filosofien i Dijkstras algoritme skulle vi som neste steg ta den 'beste' av de gjenværende nodene, og uansett må dette bli 'e'. Denne skal vi så gi verdier for 'dist' og 'path' ut fra å se på alle direkte kanter fra noder som er ferdige, og så velge det 'beste'. Den eneste kanten til 'e' er fra 'd', og vi får derfor 'path' i 'e' satt til 'd', og fargene brukt til 'e' er $\{1, 2, 3\}$. Men dette er jo feil, siden veien a–>b–>c–>d–>e jo faktisk bare bruker *to* farger. Det som ikke fanges opp i Dijkstras algoritme er altså det at man må gjøre forandringer i en allerede 'godkjent' vei (den beste mellom 'a' og 'd') når vi skal 'forlenge' den til 'e'.

Dijkstras algoritme bygger altså nettopp på at slike forandringer aldri er nødvendige, og dette tilsvarer også at de beste veiene faktisk kan representeres som et rotrettet tre. Vi ser at det ikke er mulig i eksempelet over, i og med at 'path' i 'c' må gå til 'a', og at 'path-veien' fra 'e' må gå innom 'b'.