Løsningsforslag til INF110 – h2001

Eksamen i :	INF 110 — Algoritmer og datastrukturer
Eksamensdag :	Lørdag 8. desember 2001
Tid for eksamen :	09.00 - 15.00
Oppgavesettet er på :	5 sider inkludert vedlegget
Vedlegg :	Koden til permutasjon fra forelesningene
Tillatte hjelpemidler :	Alle trykte og skrevne

Oppgave 1 (20 %)

Kommentar: *Mange studenter hadde problemer med å forstå denne (De hadde vansker med å huske at det bare var ett rør, og de 'ikke-norsk-etniske' forsto ofte ikke ordet 'avkapp' – selv om det var definert. Mange spørsmål på denne oppgaven.)*

Et rørleggerfirma har et langt rør med lengde L som det selger i biter. Bestillingene på rørbiter (egentlig lengden på bestilte rørbiter) ligger lagret i en array a[n] (der altså n er antall bestillinger). Vi antar at summen av bestillingene er større enn L, og oppgaven er å minimalisere avkappet, det vil si lengden av den delen av røret som ikke blir solgt.

I programmeringen skal du gå ut fra at både L og a[ ] er heltall.

Oppgave 1 A

Bruk en enklest mulig (orden O(n)) grådig algoritme til å skrive en metode til å foreslå hvordan røret bør kappes opp. Skriv ut hvilke lengder fra a[ ] du kapper røret opp i og lengden av avkappet. Begrunn hvorfor algoritmen din er O(n).

```
void grådigKapp(){
  // Grådig-alg: Første som passer, L = len
     int j = 1, kaplen =0;
       for(int i =0; i < n ; i++)
         if ( a[i] + kappLen <= len ){
               kappLen += a[i];
               System.out.println(" Kappno:" + (j++) + " = " + a[i]);
         }
       System.out.println(" Avkappet:" + kaplen);
}
Som opplagt er O(n) – en enkel forløkke.
```

Oppgave 1 B

Bruk permutasjoner og skriv en metode som finner en optimal løsning (permutasjons-prosedyrene fra forelesningene er vedlagt).
Skriv ut som i 1A.

Bruker prumtasjonsmetodene fra vedlegget – ideen er at før eller siden vil optimal løsning ligger i starten av a[] når den permuterers. N.B dette er meget inneffektivt – fordi bl.a vi

permuterer rundt på denne begynnelsen av a[]og resten av a[] uten at det gir noen annen løsning.

```
void brukPerm (int [] ordre) {
      // kalles fra perm-metode, sjekker verdien a
      kappNum =0; kappLen =0;
      while( kappLen+ a[kappNum] <= len) {
            kappLen += a[kappNum];
            nå[kappNum] = a[kappNum];
            kappNum++;
      }
      if (kappLen > maxKappLen)
                  lagreNyBeste();  // lagre unna 'nå' og 'kappNum –1'
      som
                                   // hittil beste, skrives ut til sist
  } // end optimalKapp

Etterpå må det som lagres av 'lagreNyBeste()' skrives ut –trivielt.
```

Oppgave 1 C

Forklar hvordan du, etter å ha sortert a[ ], kan finne minste antall og største antall kapp du kan lage. Bruk dette til å foreslå forbedringer i metoden du skrev i 1 B. Her skal du ikke programmere, men bare beskrive med ord hva du ville ha gjort.

Ved å sortere, vil vi finne det minste antall (minLen) kapp vi kan ha (i enhver løsning , også da i en optimal løsning) ved å ta for oss alle de største ordrene inntil lengden er brukt opp. Tilsvarende kan vi finne det største antall kapp (maxLen)vi kan ha (i enhver løsning , også da i en optimal løsning) ved å ta for oss alle de minste ordrene inntil lengden er brukt opp.

Dvs.: Vi får delt vår array a[]i tre deler:

1) *Den delen inntil minLen som er med i enhver løsning (her er vi **ikke** interessert i permutere denne, bare å få alle mulig utplukk av elementer fra a[]etter tur inn her).*

2) *Så har vi området mellom minLen og maxLen, hvor vi også må permutere ethvert innhold på alle måter og få inn alle utplukk*

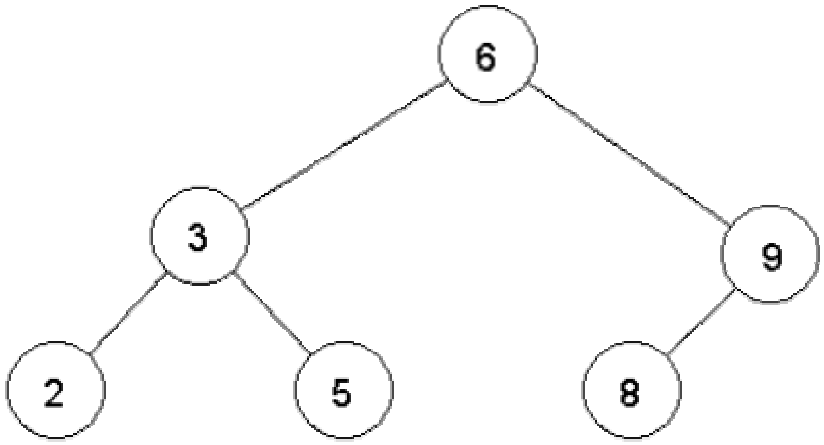
3) *og så har vi området utenfor maxLen, hvor vi ikke skal gjøre noen permutasjoner (faktisk ingen ting hvis vi har 'gjordt' 1) og 2)).*

Brukperm må altså omprogrammeres til å følge denne 'oppskriften'.

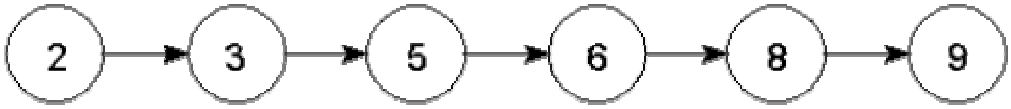
Oppgave 2 (30 %)

Vi skal i denne oppgaven se på hvordan vi kan omforme binære søketrær til sorterte lister.

Her er et eksempel på hvordan dette søketreet:



kan omformes til denne sorterte enveislisten:



Nodene i treet er objekter av følgende klasse:

```
class Node {
    int verdi;      // sorteringskriterium
    Node vsub;      // (peker til) venstre subtre
    Node hsub;      // (peker til) høyre subtre
    - - - - - - - - // eventuelle metoder
}
```

I denne oppgaven har vi et program hvor vi har deklarert en **`Node rota`**; som peker på rot-noden i det binære søketreet.

Oppgave 2 A

Skriv en metode **`void lagEnveisListe(Node hode)`** i **`class Node`** som lager en sortert enveis liste av binærtreet med **`vsub`** i nodene som nestepeker i lista. Programbiten som kaller metoden er:

```
Node liste,
     hode = new Node();
rota.lagEnveisListe(hode);
liste = hode.vsub;
```

Etter kallet skal **`vsub`** i noden som bringes med som parameter, peke på første node i lista, og **`hsub`** på siste node i lista. I den siste noden i lista skal **`vsub`** peke på **`null`**.

Kommentar: Her, og på 2B er det to alternative tenkemøter som gir ulike, riktige løsninger som begge kan honoreres til 1.0:

a) *Her tar vi oppgaveteksten som er rekursiv definisjon, dvs. at et rekusivt kall på metoden, returnerer en liste som definert i oppgaveteksten, og ikke bryr seg om (ødelegger) hva vsub og hsub i parameteren 'hode' innholdt ved kallet:*

```
void lagEnveisListe (Node hode) {
                  // return
                 Node f;
                  if (vsub != null) {
                        vsub.lagEnveisListe (hode);
                        f = hode.vsub;
                       hode.hsub.hsub = this;
                  } else{ f = this;}
                  if (hsub != null) {
                        hsub.lagEnveisListe(hode);
                        hsub = hode.vsub;
                 } else { hode.hsub = this; }
               hode.vsub = f;
       }// end lagEnveisListe
```

b) *Her tenker vi oss bare at vi bygger opp lista etter hvert som vi travereserer treet (dette er enklere)- dvs. ren infix innsetting i lista:*

```
void lagEnveisListe (Node hode) {
                 if (vsub != null) {
                        vsub.lagEnveisListe (hode);
                  }
                 if (hode.vsub == null) {
                   // første elementet i lista
                   hode.vsub = hode.hsub = this;
                   vsub = null;
                  } else{
                    hode.hsub.vsub = this;
                    hode.hsub = this;
                   }
                  if (hsub != null) {
                        hsub.lagEnveisListe(hode);
                  }
               hsub = null;
       }// end lagEnveisListe
```

c) *Da jeg gikk rundt i eksamenslokalet fikk jeg inntrykk at flere la fra seg nodene midlertidig i en 'stakk' før de lager lista, og det er vel OK- (men neppe en 1.0-løsning, men heller 2.0) hvis det blir riktig.*

Oppgave 2 B

Skriv en metode **`void lagToveisListe(Node hode)`** i **`class Node`** som lager en sortert toveis liste av binærtreet med **`vsub`** som nestepeker og **`hsub`** som forrigepeker i lista.

Metoden i 2B kalles med helt tilsvarende kode som i 2A. Nå skal **`hsub`** i den første noden i lista og **`vsub`** i siste node begge peke på **`null`**.

Her gjelder samme kommentarer om to tankemåter for løsning, og bare den første vises her:

```
void lagToveisListe (Node hode) {
            // return toveis liste av treet hode.vsub, minste elem
            // hode.hsub = største
            Node f;

            if (vsub != null) {
                  vsub.lagToveisListe (hode);
```

```
            f = hode.vsub;
            hode.hsub.hsub = this;
            vsub = hode.hsub;      // inn i liste
        } else{ f = this;}

        if (hsub != null) {

              hsub.lagToveisListe(hode);
              hsub = hode.vsub;
              hode.vsub.vsub = this;

      } else { hode.hsub = this; }

      hode.vsub = f;

    }// end lagToveisListe
```

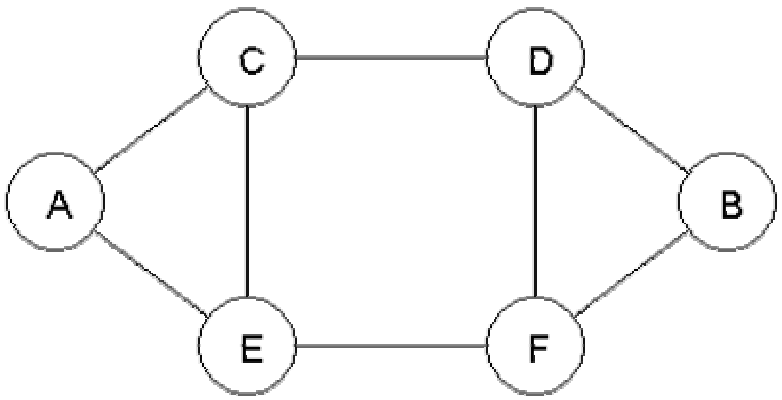
Oppgave 2 C

Forklar hvorfor det ikke er lurt å prøve det omvendte: å lage et binært søketre av en sortert liste.

Her er poenget at hvis vi bare starter med å sette inn i et binært søketre fra starten av en enveis-sortertert liste får vi et totalt ubalansert tre. Det er vanskelig/umulig å lage en enkel O(n) algoritme for å lage et balansert binært søketre ut fra en slik sortert liste.

Oppgave 3 (50 %)

I denne oppgaven skal vi se hvordan vi kan finne flest mulig ulike (disjunkte) stier mellom to noder i en graf. Vi ser av grafen nedenfor at det opplagt er maksimalt to disjunkte stier mellom A og B: A-E-F-B og A-C-D-B. Hvis vi derimot først velger f.eks. stien A-C-E-F-B, greier vi ikke å trekke flere stier mellom A og B som er disjunkt med den første.



Mer presist, la G = (V,E) være en urettet graf, der V er mengden av noder, E er mengden av kanter i G, og anta at det mellom to noder går høyst en kant, og at ingen kant går fra en node til seg selv.

La A og B være to forskjellige noder i V. Nedenfor skal du også anta at det ikke går noen kant mellom A og B (dvs at (A,B) $\notin$ E).

En sti fra A til B er en mengde av noder $\{V_0, V_1, ... , V_k\}$ slik at $V_0 = A$, $V_k = B$, og $(V_{i-1},V_i) \in$ E for $1 \leq i \leq k$. To stier fra A til B kalles disjunkte hvis A og B er de eneste nodene som er med i begge stiene.

Nodene i programdelene du skal skrive, er representert med objekter av klassen Gnode:

```
class GNode {
   String navn;
   GNode [ ] kanter;
   boolean medISti;
}
```

Når programdelene du skal skrive starter, er grafen initiert, og
GNode [ ] kanter inneholder pekere til de andre nodene som det er kanter til fra denne noden i G. En kant i G er følgelig representert med en peker i begge nodene kanten går mellom.

*Her fikk jeg de desidert fleste spørsmålene om hvordan oppgave 3A skulle løses – dvs. hvordan man tenkte seg en array som lagret stiene. Vårt svar var at vi ville for eksempelgrafen ha svar-arrayen som en enkel oppramsing av pekerne til nodene i hver sti (da blir pekerne til A og B repetert for hver sti). Da blir i verste fall plassbehovet =(|V|-2)*3 i arrayen (vi har da en graf hvor alle noder, med unntak av A og B, har kant til både A og B).*

Her har noen trodd at vi ut fra arrayen av pekere skulle ha en lenket liste for hver sti. Hvis de greier å fine riktig svar med dette, er det OK, særlig hvis de ut fra dette da har 3B omlag riktig med en slik datastruktur.

Oppgave 3 A

Hvis vi bruker en enkel (endimensjonal) array av nodepekere til å holde oversikt over de disjunkte stiene vi har funnet fra A til B (og vi registrerer dem slik det er gjort i første avsnitt i oppgaven), hvor lang er da denne arrayen maksimalt? Begrunn svaret.

Oppgave 3 B (35 %)

Skriv en metode: **void maxAntallStier(GNode a, GNode b)** og andre metoder og data du evt. trenger til å finne det maksimalt mulige antall disjunkte stier fra A til B i G. Vi antar at algoritmen du bruker gjør *rekursiv, dybde-først søk* – men andre riktige løsninger vil selvagt også bli akseptert.
(*Hint: Hva gjør du når du finner B ?*)
Du skal også lage kode som tar vare på de ulike stiene du finner. Til sist skal du skrive ut stiene i (en av de) løsningen(e) med maksimalt antall stier som du har funnet. Bruk samme format på utskriften som i innledningen og skriv ut Stringen **navn** for hver node i hver sti, samt max antall stier funnet.

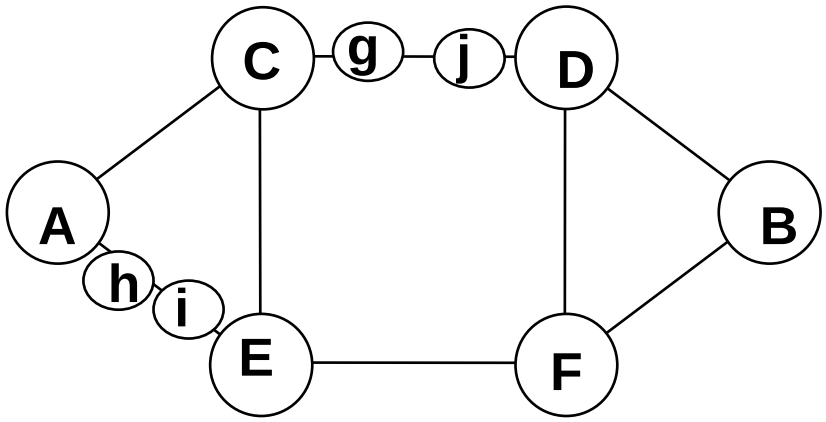
```
class Node {
      String navn;
      Node [] kanter;
      boolean besøkt = false;
```

```
      Node (String navn) {
      this.navn = navn;
    }

} // Node

public class MaxVeier {
  Node [ ] nodene ;
  static int antNoder = 0;
  static String filNavn;
  Node  [] beste, nå;
  int antVeier =0,
      maxVeier = 0 ;
  int skritt=0;
  Node start, slutt;

void lagreNyBeste() {
       for (int i = 0; i< skritt; i++)
         beste[i] = nå[i];
       maxVeier = antVeier;
 }


  MaxVeier (int n) {
        nodene = new Node [n];
        antNoder = n;
        lesNoder();
        beste = new Node[antNoder *3-6];
        nå = new Node[antNoder*3 -6];
  }

 void finnMaxVeier(Node start, Node slutt) {
       // antar ingen direkte vei fra start til slutt

       this.start = start;
       this.slutt = slutt;

       finnVei (start);

 }

 void finnVei (Node her) {
       // antar ingen direkte vei fra start til slutt
       // System.out.println(" FinnVei: " + her.navn);

       if (! her.besøkt) {
             // Ikke brukt node
             if (her != start && her != slutt)
                  her.besøkt = true;
              nå[skritt++] = her;

              if (her == slutt) {
                    antVeier++;
                    if (antVeier > maxVeier) lagreNyBeste();

                    finnVei (start);  // finn nok en vei fra start
                    antVeier --;      // tilbaketrekking

              } else {
                   // let videre
                   for (int i = 0; i< her.kanter.length; i++)
                        if( her.kanter[i] != start)
                              finnVei(her.kanter[i]);
```

```
                }

                skritt--;
                her.besøkt = false;
          }

} // end finnVei



void skrivBeste() {
      System.out.println("Antall veier funnet :" + maxVeier+":");

      for (int i =0; beste[i] != null; i++) {
            if ( beste[i]== start )
                  System.out.println("");
      if (beste[i] != slutt) System.out.print(beste[i].navn + "-");
            else System.out.print(beste[i].navn );
  }
} // end skrivBeste
```

Oppgave 3 C

Forklar hvorfor vi ikke kan bruke en algoritme som finner korteste vei fra A til B som subalgoritme for å løse problemet i oppgave 3B.
(*Hint: Legg inn nye noder i eksempelgrafen ovenfor og lag et moteksempel*)



Vi ser at de ekstra nodene gjør at A-C-E-F-B blir korteste sti fra A-B, men som før, sperrer dette for at vi finner to stier.

Vedlegg – permutasjon:

```
void bytt (int [] a, int k, int m)
  { int temp = a[k];
     a[k] = a[m];
     a[m] = temp;
  }
void roterVenstre(int [] a, int i)
   { int x,k;
     x = a[i];
     for (k= i+1; k < n; k++)
       a[k-1] = a[k];
     a[n-1] = x;
   }

void permuter (int [] a, int i)
   { // finn neste permutasjon og kall "brukPerm(a)
     // N.B. Permutasjonene startes ved kallet:  permuter(a,0);

    if ( i == n-1) brukPerm(a) ;
    else {
        permuter(a,i+1);
        for (int t = i+1 ; t < n; t++)
        {  bytt (a,i,t);
           permuter(a,i+1);
        }
        roterVenstre(a,i);
     }
   }
```

Slutt på oppgavesettet, Lykke til !

Arne Maus og Ragnar Normann