Universitetet i Bergen
Det matematisk-naturvitenskaplege fakultetet

Nynorsk

Eksamen i	:	I-120 Algoritmer, Datastrukturer og Programmering
Dato	:	25 Mai 1999
Tid	:	9:00 – 15:00
Antal sider	:	4
Tillatne hjelpemiddel	:	Ingen

- Du kan bruke alle `interface` og data typer som vart gjennomgått i løpet av kurset – bruk i størst mulig grad `interface`'r og unngå unødvendig bruk av konkrete data typer.
- "Programmer" betyr 'skriv fullstendig Java kode'. (Klarer du ikkje det, skriv i det minste presis pseudo-kode.) Dersom programmet ditt treng ein *implementasjon* av ein gitt `interface`, kan du anta at ein slik implementasjon er tilgjengeleg utan å implementere den sjølv. Indiker stader der du brukar ein slik antakelse.
- Dokumenter all kode (beskriv forutsetningar om parametrar samt resultat av metodar, spesifiser invariantar for viktige løkker og rekursjon, etc.).
- Les heile oppgavesettet før du begynner å løyse enkelte oppgaver.
- Prosentsatsane ved dei enkelte oppgavene angir estimert tidsforbruk ved løysing samt omtrentleg vekt ved sensur.

1 Kompleksitet (10%)

For kvar funksjon under i oppgava, bestem funksjonens arbeidsmengde i $\mathcal{O}$-notasjon, uttrykt ved parameteren n. Anta at $n > 1$. Du må også gi ein kort begrunnelse for dine svar.

1. ```
   void f1 (int a[ ], int n) {
      int i = n;
      while (i>1) {
         for (int j = 0; j < n; j++) { a[j] = a[j]/i; }
         i = i/2;
   }  }
   ```

2. ```
   int f2 (int n) {
      int sum = 0;
      for (int i=0; i<n; i++) {
         for (int j=0; j< n*n; j++) { sum++; }
      }
      return sum; }
   ```

3. ```
   int f3 (int n) {
      if (n==1) return 1;
      return n*n * f3(n-1); }
   ```

4. ```
   void f4 (int n) {
      if (n>0) {
        for (int i=0; i<n; i++) {
           System.out.print("Beep");
           f4(n-1);
   } } }
   ```

2 Rekursjon (30%)

Vi betrakter eit hierarki av endelege mengder av heiltal. Ei slik mengde er definert rekursivt:

- for kvart tal x, er $\mathbf{x}$ ei mengde med eit element x (egentleg er $\mathbf{x} = \{x\}$ men vi skriver $\mathbf{x}$ for å forenkle notasjon)
- hvis $m_1, m_2, ..., m_k$ er forskjellige mengder, så er $\{m_1, m_2, ..., m_k\}$ ei mengde med k elementer: $m_1, m_2, ..., m_k$

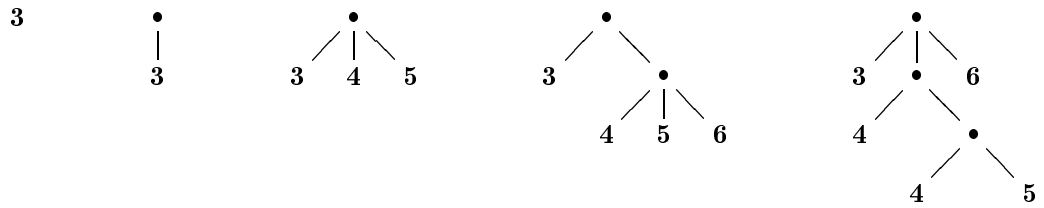
Det siste betyr spesielt at hvis m er ei mengde, så er $\{m\}$ ei mengde med eit element $m \neq \{m\}$! Det krever også at mengder skal være forskjellige – hvis $m_1 = m_2$, f.eks. $m_1 = m$ og $m_2 = m$, kan ikke regelen anvendast for å lage ei mengde $\{m, m\}$ (ei slik mengde underforstås til å vere det same som $\{m\}$).

Uttrykket "hierarki av mengder" henviser til det faktum at alle elementa i ei mengde $\{m_1, ..., m_k\}$ er sjølv mengder (det einaste unntaket er at elementet x av ei mengde $\mathbf{x}$ sjølv ikkje er ei mengde men eit tal); også mengdene $\mathbf{x}, \{\mathbf{x}\}, \{\{\mathbf{x}\}\}$ er alle inbyrdes forskjellige.

Fylgjande gir eit eksempel på inbyrdes forskjellige mengder

$$\mathbf{3} \qquad \{\mathbf{3}\} \qquad \{\mathbf{3}, \mathbf{4}, \mathbf{5}\} \qquad \{\mathbf{3}, \{\mathbf{4}, \mathbf{5}, \mathbf{6}\}\} \qquad \{\mathbf{3}, \{\mathbf{4}, \{\mathbf{4}, \mathbf{5}\}\}, \mathbf{6}\} \tag{1}$$

Vi kan representere mengder som trær på fylgjande måte: ein intern node er ei mengde og alle mengda sine element er representert av nodens barn. Løvnodar blir da mengder av type $\mathbf{x}$, der x er eit tal. Fylgjande tegning viser mengdene ovanfor representert på denne måten:



Du har gitt ein klasse `Mengde` som

- implementerer `interface Tree` (sjå vedlegg)
- eksterne nodar lagrer objekt av: `public class EtTallMengde{ public int er();}`

Metoden `er()` returnerer eit heiltal som er elementet i mengda, dvs. 3 for mengden **3**. Kvart objekt `m` av klassen `Mengde` tilfredstiller invarianten at for kvar node `v`:

- hvis `m.isExternal(v)` så vil `v.element()` være eit objekt av klassen `EtTallMengde`
- hvis `m.isInternal(v)` så er objektet lagret ved `v.element()` uvesentlig

Oppgaven er å programmere noen metodar for klassen `Mengde`. Nevn eksplisitt eventuelle tilleggsforutsetninga du gjer om måten `Mengde` representerer mengder.

2.1 toString (15%)

Programmer metode `String toString()` i klassen `Mengde` som skal skrive ut mengda med passande parantesar; kalla fra mengdene (trærne) fra tegninga, skal den returnere strengar som angitt tidlegare i (1).

2.2 basis (15%)

Programmer ein metode `String basis()` i klassen `Mengde` som skal "flate ut" mengda, dvs. finne alle forskjellige tal som finns i mengden på eit eller annat nivå. Kalla fra mengdene fra tegningen ovanfor, skal den returnere respektive strengar: "3", "3", "3, 4, 5", "3, 4, 5, 6" og "3, 4, 5, 6".

Lag denne metoden mest mulig effektiv og beskriv kompleksiteten den har.

3 Programmering og abstraksjon (60%)

Vi har ein retta graf G med N nodar. Vi antar at nodar er nummerert med heiltal k der $0 \leq k < N$. Ein retta kant fra node u til node z skriv vi (u, z). Fylgjande algoritme kan brukast for å finne lengda av kortaste sti fra noden x til alle andre nodar:

```
void kortest(int x) {
  int[] D= new int[N];
  for hver node u: D[u]= MaxValue;
  D[x]=0;
  for (int i=1; i<N; i++) {
    for hver kant (u,z) {
       if (D[u]+1 < D[z]) D[z]= D[u]+1;
  } }
}
```

Når algoritmen terminerer, vil `D[z]` inneholde lengda av kortaste sti fra `x` til `z` (eller `MaxValue` hvis det ikkje finns noen sti).

3.1 Kortest sti (10%)

Utvid algoritmen over slik at den tar som parametrar to nodar, x og y, og finn den kortaste stien fra x til y (dvs. alle nodar $x, v_1, v_2 ... v_k, y$ som identifiserer den kortaste stien fra x til y). Algoritmen skal gi ei passande melding dersom det ikkje finns ein sti, og dersom ein sti finnast, skal den kortaste skrives ut. Det er nok med presis pseudo-kode.

[Du vil ha bruk for noen ekstra variabler/datastruktur for å ta vare på nødvendig tilleggsinformasjon.]

3.2 Golf (20%)

Ein variasjon over golf vert spela på eit felt med N hull (numerert med $1 \leq h \leq N$). Målet er å spele ballen raskest mulig fra eit gitt hull x til eit gitt hull y – men på ein bestemt måte. Hvis ballen er i hullet i, er det lov å spele ballen til hullet j hvis og bare hvis:

i) $j \in \{i+1, i-1, 2i, i/2\}$ der

ii) $0 \leq j < N$ og vidare

iii) å spele til hullet $i/2$ er tillatt berre når i er eit partal.

Programmer ein algoritme `spill(int N, int x, int y)` som, gitt N, x og y beregner den raskeste (kortaste) måten å spele ballen fra x til y. (x treng ikkje å vere mindre enn y.) Output skal vere ruten ballen skal spelast gjennom, dvs. ein sekvens $x, v_1, v_2 ... v_k, y$. F.eks.

`spill(50,11,30)` = 11–12–13–14–15–30
`spill(50,11,32)` = 11–10–9–8–16–32
`spill(50,11,19)` = 11–10–20–19

3.3 Abstraksjon (30%)

I spillet fra 3.2 hadde vi faste regler i)–iii) for korleis ballen kan spelast mellom hullene. Vi vil nå lage ein generisk algoritme som kan brukast uansett hvilke reglar som definerer lovlege bevegelsar fra eit hull til neste. F.eks. vil ein kanskje spele etter regler der ballen kan spelast fra hullet i til hullet j hvis og berre hvis:

i*) $j \in \{i-1, 2i+1, i/3\}$ der

ii*) $0 \leq j < N$ og videre

iii*) å spele til hullet $i/3$ er tillatt berre når i er delelig med 3, dvs. når $i = 3k$ for ein eller annen $k > 0$.

Her ville vi fått

`spill(50,11,30)` = 11–23–22–45–15–31–30
`spill(50,11,32)` = 11–10–9–8–17–16–33-32
`spill(50,11,19)` = 11–10–9–19

3.3.1

Identifiser stadane i algoritmen fra 3.2 som må forandrast og design ein `interface Regler` som vil kunne brukast i ein genersik algoritme – forklar kort intensjonar med alle metodane. (Reglane definerast alltid som i eksemplane over, dvs. som aritmetiske uttrykk over heiltalane som identifiserer hulla.)

3.3.2

Programmer ein genersik algoritme `spill(int N, int x, int y, Regler r)` som kan brukast til å spele etter forskjellige reglar avhengig av implementasjon av `interface Regler`.

3.3.3

Programmer to implementasjonar av `interface Regler` fra 3.3.1, `ra` og `rb`, slik at eit kall til `spill(N,x,y,ra)` skal gi same resultat som eit kall til `spill(N,x,y)` fra 3.2, mens eit kall til `spill(N,x,y,rb)` skal gi eit resultat tilsvarande reglane i*)–iii*) over.

Lykke til!

Michał Walicki