Skriftlig eksamen i Datalogi

Modul 1

Vinter 1999/2000

Opgavesættet består af 6 opgaver, der ved bedømmelsen tillægges følgende vægte:

Opgave 1	5%
Opgave 2	10%
Opgave 3	15%
Opgave 4	20%
Opgave 5	20%
Opgave 6	30%

Alle sædvanlige hjælpemidler er tilladt. I tilfælde af unøjagtigheder i opgaveteksterne forventes det, at deltagerne selv præciserer besvarelsernes forudsætninger.

Opgavesættet består af en forside og 8 paginerede sider. Kontroller at din kopi er fuldstændig.

Opgave 1: Ombytning (5%)

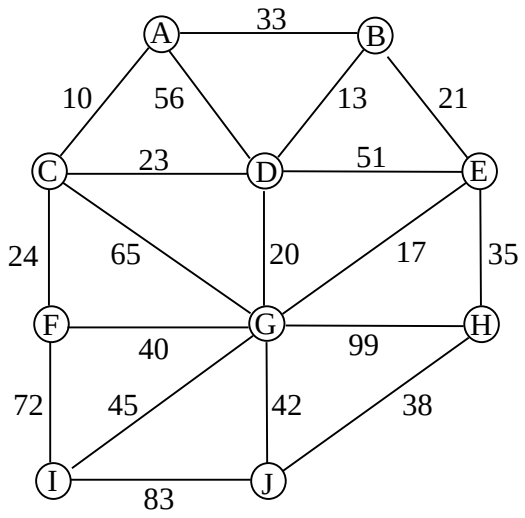
Det er velkendt, at værdierne for to heltalsvariable `a` og `b` kan ombyttes ved hjælp af koden

```
int temp = a;
a = b;
b = temp;
```

Spørgsmål 1.1 Vis at denne ombytning kan foretages uden brug af hjælpevariable.
(Vink: brug addition og/eller subtraktion).

Opgave 2: Grafer (10%)

Betragt nedenstående vægtede graf.

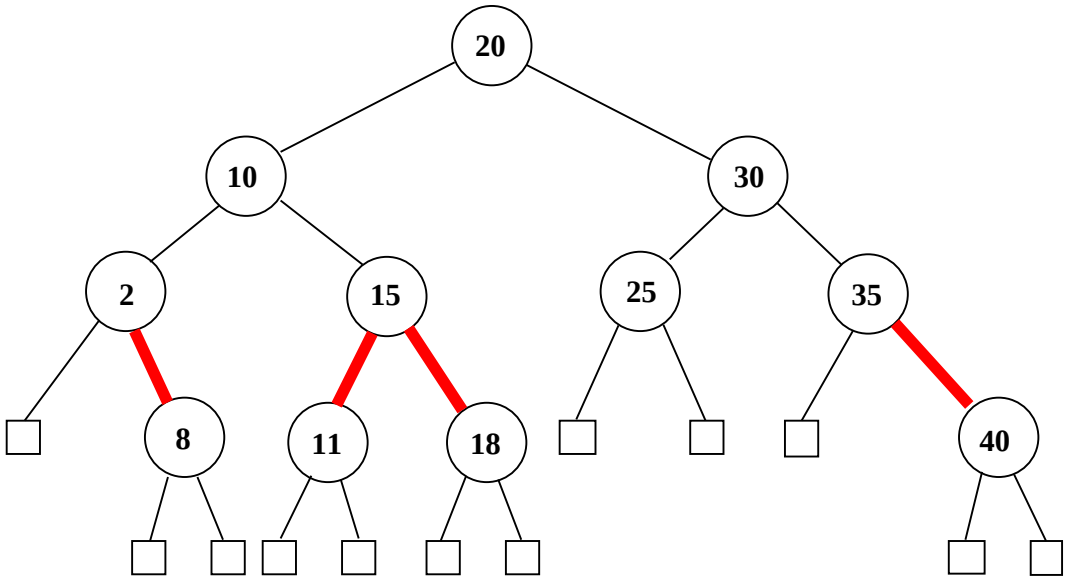


Spørgsmål 2.1 Angiv et mininimalt udspændende træ for grafen.

Spørgsmål 2.2 Angiv et korteste-vej-træ, der har knude A som rod.

Opgave 3: Rød-sort-træer (15%)

Betragt nedenstående rød-sort-træ.



Sorte grene er markeret med ——, røde med ▬.

Spørgsmål 3.1 Argumenter for at træet er et rød-sort-træ.

Spørgsmål 3.2 Tegn et tilsvarende 2-3-4-træ.

Spørgsmål 3.3 Tegn det rød-sort-træ, der fremkommer ved i rød-sort-træet ovenfor at indsætte nøglen 13.

Opgave 4: Sortering (20%)

En mekaniker har N møtrikker af forskellig størrelse og N bolte, der passer til møtrikkerne. Hans opgave er for hver møtrik at finde den bolt, der passer til møtrikken.

Han kan sammenligne en møtrik med en bolt (ved at forsøge at skrue dem sammen) og dermed fastslå, om bolten er for stor, for lille, eller passer præcist med møtrikken. Men han har ingen mulighed for hverken at sammenligne to bolte eller to møtrikker.

Vi kan løse mekanikerens opgave, hvis vi kan sortere såvel møtrikker som bolte efter deres størrelse. Efter sorteringen vil den første møtrik svare til den første bolt, den anden møtrik til den anden bolt, o.s.v.

Men hvorledes skal vi f.eks. sortere møtrikkerne, når vi ikke må sammenligne dem indbyrdes?

På næste side er skitseret en mulig løsning i Java.

I klassen `Nuts_and_Bolts` angiver tabellerne `bolt` og `nut` henholdsvis mængden af bolte (objekter af klassen `Bolt`) og mængden af møtrikker (objekter af klassen `Nut`).

Den centrale algoritme er programmeret i metoden `sort_nuts_and_bolts`. Metoden benytter sortering ved udvælgelse. I hvert skridt bestemmes enten den mindste møtrik eller den mindste bolt blandt de endnu ikke ordnede møtrikker og bolte. Hvis den mindste bolt blev bestemt, findes den tilsvarende møtrik. Hvis den mindste møtrik blev bestemt, findes den tilsvarende bolt. Dernæst placeres den fundne møtrik og bolt efter alle hidtil ordnede møtrikker og bolte. Således fortsættes, indtil begge tabeller er ordnet.

Sammenligninger foretages ved hjælp af metoden `compareTo`. Metoden returnerer –1, 0 eller 1, alt efter om en given bolt (møtrik) er mindre end, lig eller større end en given møtrik (bolt).

Spørgsmål 4.1 Færdigprogrammér metoden `compareTo` i klassen `Nut`.

Spørgsmål 4.2 Færdigprogrammér metoden `sort_nuts_and_bolts`.

```
abstract class Element {
    abstract int compareTo(Element e);
    int id;
    float size;
}

class Bolt extends Element {
    Bolt(int i, float s) { id = i; size = s; }

    public int compareTo(Element e) {
        Nut n = (Nut) e;
        return size < n.size ? -1 : (size > n.size ? 1 : 0);
    }
}

class Nut extends Element {
    Nut(int i, float s) { id = i; size = s; }

    public int compareTo(Element e) {
        /* A: indsæt kode her (spørgsmål 4.1) */
    }
}

class Nuts_and_Bolts {
    Bolt bolt[];
    Nut nut[];
    int N;

    void swap(Element a[], int i, int j)
      { Element t = a[i]; a[i] = a[j]; a[j] = t; }

    void sort_nuts_and_bolts() {
        for (int i = 0; i < N; i++) {
            int n = i, b = i;
            while (n < N && b < N)
                if (nut[n].compareTo(bolt[b]) <= 0)
                  { /* B: indsæt kode her (spørgsmål 4.2) */ }
                else
                  { /* C: indsæt kode her (spørgsmål 4.2) */ }
            if (n == N)
              { /* D: indsæt kode her (spørgsmål 4.2) */ }
            else
              { /* E: indsæt kode her (spørgsmål 4.2) */ }
            swap(bolt, i, b);
            swap(nut, i, n);
        }
    }
}
```

En hurtigere algoritme kan opnås ved at benytte følgende variant af quicksort:

Udtag en bolt, `b`, og sammenlign den med alle møtrikkerne. Find den møtrik, `n`, der passer til `b`. Opdel derefter problemet i to, hvor det ene omfatter alle møtrikker, der er mindre end `b`, og alle bolte, der er mindre end `n`, mens det andet omfatter alle møtrikker, der er større end `b`, og alle bolte, der er større end `n`. Løs derefter disse to problemer på samme måde (rekursivt).

Nedenfor er skitseret en mulig implementering af denne algoritme i form metoderne `partition` og `quicksort_nuts_and_bolts`.

```
int partition(Element e[], Element pivot, int left, int right) {
    int i = left, j = right;
    while (true) {
        /* F: indsæt kode her (spørgsmål 4.3) */
        if (i == j)
            return i;
        swap(e, i, j);
    }
}

void quicksort_nuts_and_bolts() {
    quicksort_nuts_and_bolts(0, N-1);
}

void quicksort_nuts_and_bolts(int left, int right) {
    if (left < right) {
        Bolt b = bolt[(left + right)/2];
        int i = left;
        while (nut[i].compareTo(b) != 0)
            i++;
        Nut n = nut[i];
        /* G: indsæt kode her (spørgsmål 4.4) */
    }
}
```

Spørgsmål 4.3 Færdigprogrammér metoden `partition`.

Spørgsmål 4.4 Færdigprogrammér metoden `quicksort_nuts_and_bolts`.

Opgave 5: Objektorienteret programmering (20%)

Til en studenterdatabase skal der bruges en repræsentation af eksamensresultater. I første omgang antager vi alle eksamener bedømt efter 13-skalaen. Den første skabelon til klassen er

```
class Karakter {
   int karakter;
}
```

Spørgsmål 5.1 Vi ønsker at gøre feltet "`karakter`" privat (`private`) og tildele det en værdi i en konstruktor. Skriv en sådan konstruktor, der kaldes med en heltalsværdi.

```
Karakter(int k){...}
```

Spørgsmål 5.2 Skriv en metode "`lovlig`", der returnerer sand (`true`) hvis tallet i feltet "`karakter`" er en lovlig karakter efter 13-skalaen.

Spørgsmål 5.3 Skriv en metode "`toString`" som returnerer karakteren som tekststreng. Der skal bruges foranstillet "0" for karakterne "00" og "03".

Spørgsmål 5.4 Skriv en konstruktor som kaldes med en karakter som teksstreng.

```
Karakter(String k){...}
```

Konstruktoren skal også forstå karakterne "00" og "03".

Spørgsmål 5.5 Skriv en konstruktor som kaldes med en karakter som "`Karakter`" objekt.

```
Karakter(Karakter k){...}
```

Et kald af konstruktoren laver altså en kopi af et "`Karakter`" objekt.

Spørgsmål 5.6 Skriv en metode "`haev`", som hæver karakteren en plads i tretten skalaen. Metoden skal ikke have nogen effekt ved et tretten tal. For 00, 03 og 11 er de respektive nye værdier 03, 5 og 13.

Spørgsmål 5.7 Tilføj en variant af metoden "`haev`" som har en ekstra parameter, der angiver hvor mange pladser karakteren skal hæves.

```
void haev(int k){...}
```

Skriv metoden så den ikke direkte ændrer i feltet "`karakter`" men i stedet benytter metoden skrevet i spørgsmål 5.6.

Opgave 6: Objektorienteret programmering (30%)

Studenterdatabasen skal nu udvides så der også kan gemmes resultater bedømt efter "bestået/ikke bestået".

```
interface Resultat{
}
class Karakter implements Resultat{
  int karakter;
}
class Bestaaet implements Resultat{
  boolean ok;
}
```

Det kan her antages at felterne er synlige uden for klasserne.

Spørgsmål 6.1 Tilføj en konstruktor til klassen "`Bestaaet`" og lad den initialisere feltet "`ok`".

```
Bestaaet(boolean ok){...}
```

Spørgsmål 6.2 Tilføj en metode "`toString`" til klassen "`Bestaaet`". Metoden skal returnere "Bestået" eller "Ikke bestået" afhængig af om feltet i klassen har en sand eller falsk værdi.

Spørgsmål 6.3 Objekter, der implementerer interface "`Resultat`" ønskes at implementere en metode "`erBestaaet`" der returnerer en logisk værdi. Tilføj en skabelon for metoden (en abstrakt erklæring af metoden) til interface "`Resultat`" og implementationer af metoden til klasserne "`Karakter`" og "`Bestaaet`". En karakter efter 13-skalaen er bestået hvis karakteren er mindst 6.

Spørgsmål 6.4 Lad der være defineret en tabel med resultater fra en eksamen.

```
Resultat[] tabel;
```

Det kan antages at hverken "`tabel`" eller noget element i "`tabel`" har værdien "`null`".
Skriv en kodestump der udregner hvor mange der er bestået.

Spørgsmål 6.5 Skriv en kodestump der udregner hvor mange fra eksamenen som er bedømt med karakter.

Spørgsmål 6.6 Skriv en kodestump der udregner gennemsnittet af karaktererne for de der er bedømt med karakter.

Spørgsmål 6.7 Der ønskes konstrueret en klasse til en studerendes eksamensbevis. Udgangspunktet er følgende abstrakte klasse.

```
abstract class Bevis{
   abstract void tilfoej(Resultat k);
}
```

Konstruer en konkret klasse `EksamensBevis` der er en udvidelse af `Bevis`. Klassen skal indeholde en metode `tilfoej` der gemmer resultater i en intern struktur. Det er tilladt at antage en øvre grænse for hvormange eksamensresultater der kan tilføjes.

Spørgsmål 6.8 Der ønskes tilføjet en metode `studentOK` til klassen `EksamensBevis`. Metoden skal returnerer sand (`true`) hvis bevisets eksamensresulteter med karakter opfylder følgende betingelse: Summen af de to laveste karakterer og gennemsnittet af de øvrige karakterer skal være mindst 13. Såfremt betingelsen ikke er opfyldt eller der ikke er mindst tre resultater med karakter skal metoden returnere falsk (`false`).