Eksamen i

Algoritmer og Datastrukturer

IAI 20102

Høgskolen i Østfold
Avdeling for informatikk
og automatisering

Tirsdag 3. desember 2002, kl. 09.00 - 14.00

Hjelpemidler:
Alle trykte og skrevne hjelpemidler. Kalkulator.

Oppgavesettet består av i alt 7 sider, inkludert forside og vedlegg, og er delt i tre hoveddeler som kan løses uavhengig av hverandre.

Hver deloppgave er prosentvis vektet. Dette gjenspeiler ikke vanskelighetsgraden, men hvor mye hver enkelt oppgave kommer til å bidra til sluttkarakteren.

Hvis ikke annet er angitt, betyr ”implementering” å skrive programkode i Java. Koden bør være oversiktlig og godt kommentert. Du behøver ikke legge vekt på generelle prinsipper for tilgang til og beskyttelse av data, men kan anta at du har ubegrenset tilgang (friendly access) til alle data og metoder i alle klasseobjekter. Du behøver heller ikke legge vekt på å gjøre koden særskilt robust. Som et alternativ/kombinasjon til Java kan du bruke godt strukturert pseudokode, men dette vil gi noe dårligere uttelling.

Les oppgavetekstene nøye, planlegg arbeidet og disponer tiden før du begynner på besvarelsen.

Skriv tydelig!

Lykke til!

Del 1 – Totalt 25%

Her finner du 5 ulike algoritmer.

For hver av disse skal du angi kompleksiteten (arbeidsmengden) i O-notasjon, inkludert en kort begrunnelse.

```
int alg1(int n) {
  int a = 0;
  for(int i = n; i > 0; i--) {
       a++;
  }
  return a;
}


int alg2(int m) {
  int n = 42 + m;
  int a = 0;
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      a++;
    }
  }
  return a;
}


void alg3(int[] arr, int a, int b) {
  if(a < 0 || b >= arr.length || a >= b) return;
  System.out.println(arr[a]);
  alg3(arr, (a+b)/2, b);
}


void alg4(int[] arr) {
  int i = 0;
  while(i < arr.length) {
    i++;
  }

  for(int j = 0; j < i; j++) {
    System.out.println(j);
  }
}


void alg5(int m, int n) {
  int a = 0;
  for(int i = 0; i < m; i++) {
    for(int j = m-i; j > 0; j--) {
      a = 42;
      for(int k = a - j; k > 0; k--) {
        System.out.println(k);
      }
    }
  }
}
```

Slutt på del 1

Del 2 - 40%

I vedlegget er det beskrevet en klasse **`BST`** med metoder som er knyttet til binærtre. Binærtreet er bygget opp av noder av type **`Node.`** I denne delen skal du implementere endel av denne klassens metoder.

Oppgave 2.1 - 5%

Forklar kort hvordan vi definerer et binærtre.

Hvilke betingelser må være oppfylt for at et binærtre skal være et binært søketre?

Oppgave 2.2 - 5%

Implementer metoden **`BST.antallNoder`** *i vedlegget.*

Oppgave 2.3 - 5%

Høyden i et tre er definert som antall nivåer minus én.

Implementer metoden **`BST.høyde`** *i vedlegget.*

Oppgave 2.4 - 5%

Her skal du sjekke om et tre er et binært søketre. Husk at det lovlige intervallet for verdien til en gitt node er avhengig av verdiene i de foregående foreldrenodene.

Implementer metoden **`BST.sjekk`** *i vedlegget.*

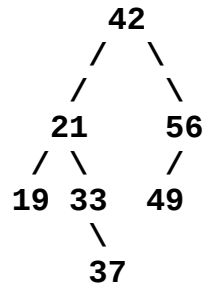
Oppgave 2.5 - 10%

Et binærtre kan representeres som en tabell av verdiene til nodene. Legg verdien i rotnoden i indeks *0*, dvs. første element i tabellen. For en vilkårlig node *k*, la verdien til venstre barn tilsvare indeks *2k+1*, og verdien til høyre barn tilsvare indeks *2k+2*. Hvis en slik indeks enten ligger utenfor tabellen, eller inneholder verdien 'N', vil det si at dette barnet ikke eksisterer.

Tabellen

42–21–56 –19 – 33 – 49 – N – N – N – N – 37

vil da representere følgende tre:



Hvis vi har et rimelig balansert binært søketre på tabellform, kan vi også foreta et effektivt søk, dvs. av logaritmisk orden, etter en bestemt verdi.

Implementer metoden **`BST.finn`** *i vedlegget.*

Oppgave 2.6 - 10%

Ved å bruke opplysningene i oppgave 2.5 kan vi konvertere et allerede eksisterende binærtre på tradisjonell referanseform til en tabell av verdiene til nodene.

Implementer metoden **`BST.konverterTilTabell`** *i vedlegget.*

Slutt på del 2

Del 3 - 35%

Et *magisk kvadrat* er en *n*x*n* firkant, som består av tallene 1, 2, ..., *n*n* (hvert tall forekommer en og kun en gang) og som har følgende egenskap: Det finnes et *magisk tall* slik at summen av tallene på hver rad er lik dette tallet, og det er også summen av tallene i hver kolonne. I tillegg er summen av tallene på hver hoveddiagonal (de to lengste diagonalene) lik det magiske tallet, f.eks. slik:

4	3	8
9	5	1
2	7	6

I dette 3x3 kvadratet er det magiske tallet 15.

I de følgende oppgavene skal du implementere ulike metoder som arbeider med magiske kvadrater. Vi bruker en heltallstabell for å representere kvadratet, der elementene i kvadratet er lagt ut radvis. Kvadratet over blir da representert som sekvensen 4-3-8-9-5-1-2-7-6.

Oppgave 3.1 - 5%

Implementer metoden **`MagiskKvadrat.sumRad`** *i vedlegget.*

Hva er ordenen i O-notasjon på denne metoden? Gi en kort begrunnelse.

Oppgave 3.2 - 5%

Implementer metoden **`MagiskKvadrat.sjekkKvadrat`** *i vedlegget.*

Hva er ordenen i O-notasjon på denne metoden? Gi en kort begrunnelse.

Oppgave 3.3 - 15%

*Forklar kort hvorfor et magisk kvadrat av størrelse n kan betraktes som en permutasjon av tallene 1, 2, ..., n*n.*

Implementer metoden **`MagiskKvadrat.skrivAlleMagiskeKvadrater`** *i vedlegget. Metoden skal generere alle mulige kvadrater av den gitte størrelsen, og sjekke hvert kvadrat om det er magisk.*

Hva er ordenen i O-notasjon på denne metoden? Gi en kort begrunnelse.

Oppgave 3.4 - 10%

Forklar hvordan metoden i oppgave 3.3 kan gjøres mer effektiv.

Implementer metoden **`MagiskKvadrat.finnEtMagiskKvadrat`** *i vedlegget med utgangspunkt i denne effektiviseringen.*

Slutt på del 3

Vedlegg

De følgende klassene og metodene skal brukes i Oppgave 2. En del metoder er angitt som allerede implementerte, *disse skal altså ikke implementeres*, men kan brukes i andre metoder. Du står fritt i å legge til ekstra funksjonalitet i form av datamedlemmer og metoder. Husk at metodene som skal implementeres kan brukes i andre metoder selv om du ikke har klart å implementere de!

Oppgave 2: `class BST`

Klassen **`BST`** inneholder metoder knyttet til binære søketrær.

```
class BST {



    // Returnerer antall noder i binærtreet som er dannet av rot
  int antallNoder(Node rot) { /* Skal implementeres i 2.2! */ }


    // Returnerer høyden (dvs. antall nivåer-1) i binærtreet
    // som er dannet av rot
  int høyde(Node rot) { /* Skal implementeres i 2.3! */ }


    // Returnerer true om binærtreet som er dannet av rot
    // er et binært søketre, false ellers.
  boolean sjekk(Node rot) { /* Skal implementeres i 2.4! */  }


    // Vi antar at tre er et binært søketre på tabellform
    // Metoden returnerer indeksen som tilsvarer tallet verdi.
    // Hvis en slik verdi ikke finnes, returnes konstanten N.
  int finn(int[] tre, int verdi) { /* Skal implementeres i 2.5! */ }


    // Metoden konverterer et binærtre til tabellform
    // Tabellen returneres
  int[] konverterTilTabell(Node rot) { /* Skal implementeres i 2.6! */ }


  ...

}


  // Binærtrenode
class      Node {
  ...
  Node     venstre; // venstre barn
  Node     høyre;   // høyre barn

  int      verdi;
}
```

Oppgave 3: `class MagiskKvadrat`

Klassen **`MagiskKvadrat`** inneholder metoder knyttet til magiske kvadrater.

```
class MagiskKvadrat {


    // Returnerer summen av elementene i rad nr. r.
    // Du kan regne med at r ligger i intervallet [0,n-1],
    // der n er størrelsen på kvadratet.
  int sumRad(int[] kvadrat, int r) { /* Skal implementeres i 3.1! */  }


    // Returnerer summen av kolonne nr. k
    // Kan med fordel benyttes i oppgave 3.2.
    // Metoden er å anse for ferdig implementert og klar til bruk.
  int sumKolonne(int[] kvadrat, int k) { /* Ferdig implementert */  }


    // Returnerer summen av diagonalen
    // øverste venstre hjørne til nederste høyre.
    // Kan med fordel benyttes i oppgave 3.2.
  int sumDiag1(int[] kvadrat) { /* Ferdig implementert */  }


    // Returnerer summen av diagonalen
    // nederste venstre hjørne til øverste høyre.
    // Kan med fordel benyttes i oppgave 3.2.
  int sumDiag2(int[] kvadrat) { /* Ferdig implementert */  }


    // Returnerer true hvis kvadratet er magisk, ellers false.
  boolean sjekkKvadrat(int[] kvadrat) { /* Skal implementeres i 3.2! */ }


    // Skriver ut et kvadrat
    // Kan med fordel benyttes i oppgave 3.3.
  void skrivKvadrat(int[] kvadrat) { /* Ferdig implementert */ }


    // Returnerer antallet magiske kvadrater av størrelse n*n.
    // Hvert magisk kvadrat skrives ut.
    // Metoden skal generere alle mulige kvadrater av denne størrelse,
    // og sjekke hvert enkelt om det er magisk.
  int skrivAlleMagiskeKvadrater(int n) { /* Skal implementeres i 3.3! */ }


    // Returnerer true hvis et magisk kvadrat
    // av størrelse n*n finnes, false ellers.
    // Metoden skal være så effektiv som mulig.
  boolean finnEtMagiskKvadrat(int n) { /* Skal implementeres i 3.4! */ }

  ...
}
```

Slutt på oppgavesettet