Eksamen i

Algoritmer og Datastrukturer

IAI 21899

Høgskolen i Østfold
Avdeling for informatikk
og automatisering

Lørdag 15. desember 2001, kl. 09.00 - 14.00

Hjelpemidler:
Alle trykte og skrevne hjelpemidler. Kalkulator.

Oppgavesettet består av i alt 10 sider, inkludert forside og vedlegg, og er delt i fem 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, skal programkode skrives i Java, som pseudokode, eller som en kombinasjon. 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. Les oppgavetekstene nøye, planlegg arbeidet og disponer tiden før du begynner på besvarelsen.

Skriv tydelig!

Lykke til!

Del 1: Algoritmeanalyse – Totalt 20%

Oppgave 1.1 - 20%

Under er det beskrevet noen operasjoner på ulike datastrukturer (en tabell er ekvivalent med en Java array). *For hver av dem skal du angi arbeidsmengden i O-notasjon, forutsatt at det ikke brukes ekstra hjelpestrukturer. Gjør også rede for eventuelle antagelser og presiseringer*.

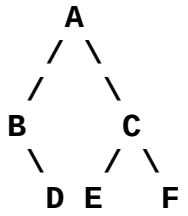
1. Finne et bestemt element, for eksempel det 7. elementet, i en tabell.
2. Finne et bestemt element, for eksempel nr. 7, i en lenket liste.
3. Skrive ut alle elementene i en enkeltlenket liste i omvendt orden (det siste elementet skrives ut først etc.).
4. Finne et element med en gitt nøkkel i en usortert tabell.
5. Finne et element med en gitt nøkkel i en dobbeltlenket liste.
6. Finne et element med en gitt nøkkel i et binært søketre.
7. Sortere en enkeltlenket liste med reelle tall (float/double).
8. Sortere en tabell med 4242 heltall som alle ligger i intervallet [0 – 42].
9. Sortere en tabell med heltall der du vet at 50% av tallene er like.
10. Finne en record med en bestemt nøkkel i en hashtabell.

Hvert av delspørsmålene teller likt.

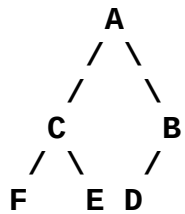
Slutt på del 1

Del 2: Trær I – Totalt 20%

Vi har følgende binærtre:



Etter en såkalt *speiling* ser treet ut slik:



En speiling medfører altså at for enhver node i treet vil venstre og høyre subtre bytte plass. Du skal nå implementere to metoder som håndterer slike speilinger.

Oppgave 2.1 - 10%

Her skal du implementere en metode som rekursivt bytter om venstre og høyre subtre for alle nodene i treet angitt med referansen `rot`.

Implementer metoden `void` **`speile`**`(BinNode rot)` *i klassen* **`BinTre`** *i vedlegget. Eksakt Javakode gir best uttelling.*

Oppgave 2.2 - 10%

Nå skal du implementere en metode som lager en kopi av et tre som i sin helhet blir en speiling av originalen.

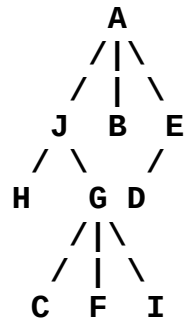
Implementer metoden `BinNode` **`speiletKopi`**`(BinNode rot)` *i klassen* **`BinTre`** *i vedlegget. Eksakt Javakode gir best uttelling.*

Slutt på del 2

Del 3: Trær II – Totalt 15%

Oppgave 3.1 - 5%

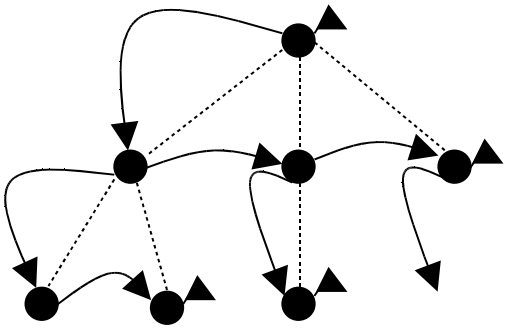
Vi har følgende generelle tre:



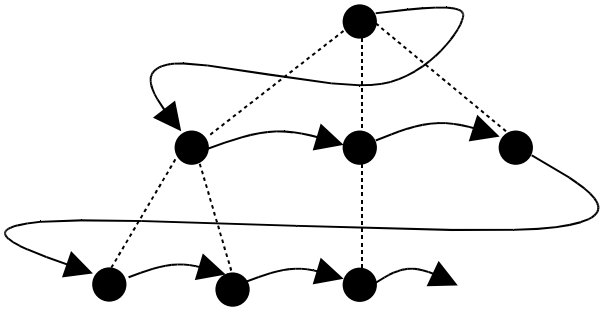
Utfør en såkalt levelorder traversering av dette treet. Angi kun rekkefølgen av nodene. Forklar kort og skissemessig hvordan en slik traversering vanligvis implementeres.

Oppgave 3.2 - 10%

I vedlegget er det beskrevet en nodeklasse, **`GenNode`**, og en treklasse, **`GenTre`**, som til sammen representerer et generelt tre, dvs. et tre der hver node kan et vilkårlig antall barn. Merk at det brukes såkalte søskenreferanser, dvs. referanser fra en nodes første barn, til dets andre barn osv. For å angi barna til en node er det da nok å ha en referanse til det første barnet. Fra dette barnet kan vi så gå til det andre, og så til det tredje osv. Søskenreferansen til "siste" barnet er lik null. Strukturen kan skisseres slik:



En annen måte å foreta en levelorder traversering er å la hver node ha en referanse til den neste noden på sitt nivå. Hvis noden er den siste i et nivået, peker referansen på den første noden i neste nivå. Når disse referansene er korrekt satt, gjennomføres traversering ved å følge pekerne til vi kommer til den "siste" noden. Denne noden sin nestereferanse er lik null. Se figuren under.



Nå skal du lage en metode som setter nestereferansene til å peke på riktige noder.

Implementer metoden `void` **`settNesteReferanser`**`()` *i klassen* **`GenTre`** *i vedlegget. Du skal anta at alle søskenreferansene er korrekt satt. Presis pseudokode gir like god uttelling som eksakt Javakode.*

Slutt på del 3

Del 4: Rekursjon – Totalt 20%

Et rektangulært brett med ruter er representert ved en boolsk matrise `boolean brett[M][N]`. En rute `brett[i][j]` er sort hvis verdien er lik `true`, og hvit ellers. Antall kolonner er lik `M`, og antall rader er lik `N`.

Du skal nå studere en algoritme arbeider med et slikt brett som input. Hovedrutinen ser slik ut:

```
int hokus ( boolean brett[][] ) {
  int antall = 0;
  for (int i = 0; i < brett.length; i++ ) {
    for (int j = 0; j < brett[i].length; j++ ) {
      if ( brett[i][j] ) {
        antall += pokus ( brett, i, j );
      }
    }
  }
  return antall;
}
```

Den rekursive funksjonen **`pokus`** er definert som følger:

```
int pokus ( boolean brett[][], int i, int j ) {

  brett[i][j] = false;

  if ( ok (brett, i-1, j  ) ) pokus ( brett, i-1, j   );
  if ( ok (brett, i+1, j  ) ) pokus ( brett, i+1, j   );
  if ( ok (brett, i,   j-1) ) pokus ( brett, i,   j-1 );
  if ( ok (brett, i,   j+1) ) pokus ( brett, i,   j+1 );

  return 1;
}
```

Vi har også en hjelpefunksjon **`ok`**:

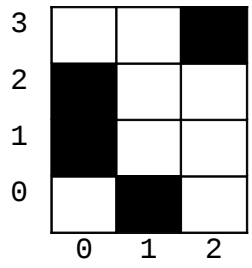
```
boolean ok ( boolean brett[][], int i, int j ) {
  if ( i <    0               ||
       j <    0               ||
       i >=   brett.length    ||
       j >=   brett[i].length ||
       !brett[i][j]           ) {

    return false;
  }

  return true;
}
```

Oppgave 4.1 – 10%

Gitt følgende 3x4 brett:



Håndeksekver metoden for med dette brettet som input. Du kan stoppe når telleren **`i`** *blir lik 2. Bruk figurer som du synes illustrerer tilstander og utvikling på en god måte.*

Oppgave 4.2 – 10%

Hva gjør denne algoritmen? Ta utgangspunkt i håndeksekveringen i 4.1, og forklar så presist som mulig hvordan algoritmen virker.

Hva er algoritmens orden? Gi en kort begrunnelse.

Slutt på del 4

Del 5: Nettverk – Totalt 25%

Et rettet nettverk kan representeres ved en naboliste, for eksempel slik:

Node:	**Naboer:**
A	B, C
B	E, C, A
C	D
D	
E	D

Oppgave 5.1 – 5%

Tegn nettverket som er representert i lista over, slik at relasjonene mellom nodene kommer tydelig fram.

Oppgave 5.2 – 5%

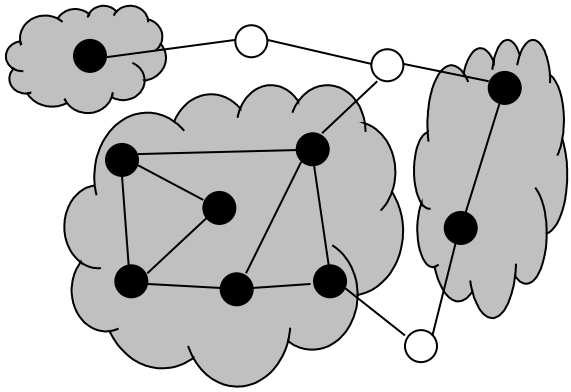
Foreta en dybde først traversering med start i A. Angi både rekkefølgen av nodene og traverseringstreet[1].

Oppgave 5.3 – 5%

Foreta en bredde først traversering med start i A. Angi både rekkefølgen av nodene og traverseringstreet.

Oppgave 5.4 – 10%

Vi har et urettet nettverk der nodene representerer henholdsvis sort og hvitt. En såkalt *sky* er en samling ”sorte noder”, der vi fra alle nodene i skyen kan finne en vei med bare sorte noder til en hvilken som helst av de andre nodene i skyen. Nedenfor ser du et eksempel på et nettverk med tre skyer.



Beskriv en metode som finner ut hvor mange skyer det er i et gitt nettverk. Anta at hver node har en boolsk variabel `farge` *som er* `true` *for sorte noder og* `false` *for hvite. Det er tilstrekkelig å bruke klartekst eller pseudokode. Angi metodens kompleksitet inkludert en kort begrunnelse.*

Slutt på del 5

[1] Et traverseringstre består av nodene som inngår i traverseringen. Barna til en gitt node i treet er de av denne nodens naboer man ”går til” under traverseringen.

Vedlegg

De følgende klassene og metodene brukes i enkelte oppgaver. Du står fritt i å legge til ekstra funksjonalitet i form av datamedlemmer og metoder.

Oppgave 2: `class BinNode`

Klassen **`BinNode`** representerer en node i et binærtre.

```
class BinNode {

  // Referanse til barna
  BinNode   venstre;
  BinNode   høyre;
  char      tegn;

  // Konstruktør, bør brukes i oppgave 2.2
  // Ferdig implementert
  BinNode(char t, BinNode v, BinNode h) {
    tegn     = t;
    venstre  = v;
    høyre    = h;
  }
  ...
}
```

Klassen ***`BinTre`*** inneholder metoder som arbeider på binære trær.

```
class BinTre {

  // Skal implementeres i oppgave 2.1
  // Metoden bytter om på venstre og høyre subtre
  // i denne noden og i alle etterfølgende noder
  // Metoden skal være rekursi
  void speile(BinNode rot)         {...}

  // Skal implementeres i oppgave 2.2
  // Metoden lager en speilet kopi av dette treet
  // og returnerer rota til kopien
  // Metoden skal være rekursiv
  BinNode speiletKopi(BinNode rot) {...}
  ...
}
```

Oppgave 3: `class GenNode`

Klassen **`GenNode`** representerer en node i et generelt tre**.**

```
class GenNode {

// Referanse til første barn
  GenNode barn;

  // Referanse til evt. neste søsken
  // Hvis denne noden er den siste i rekken av søsken,
  // er referansen lik null
  GenNode søsken;

  // Referanse til neste node på samme nivå i treet
  // Hvis denne noden er den siste på sitt nivå,
  // vil neste være første node på etterfølgende nivå
  // For siste node på siste nivå er neste lik null
  GenNode neste;
  ...
}
```

Klassen **`GenTre`** representerer et generelt tre**.**

```
class GenTre {

  // Referanse til rota i treet
  GenNode rot;

  // Skal implementeres i oppgave 3.2
  // Du skal anta at alle søskenreferansene er korrekt satt
  // og at rot ikke er lik null
  void settNesteReferanser() {...}

  ...
}
```

Generelle datastrukturer

```
  // Følgende datastrukturer kan anses som ferdig implementert
  // og kan brukes ved eventuelt behov


  // Generelt interface for lister

public interface   Liste  {
  boolean   erTom       ();
  void      nullstill   ();
}

  // Vanlig kø
public class       Fifo implements Liste {
  ...
  void      settInn     (Object x)  {...}  // Setter inn
  Object    taUt        ()          {...}  // Fjerner
  Object    sjekkFront  ()          {...}  // Se på frontelementet
}

  // Stack
public class       Lifo implements Liste {
  ...
  void      push        (Object x)  {...}  // Setter inn
  Object    pop         ()          {...}  // Fjerner
  Object    sjekkFront  ()          {...}  // Se på frontelementet
}
```

Slutt på oppgavesettet