Om eksamensoppgavene

Høgskolen i Østfold

EKSAMEN

Emnekode og -navn: ITF20006 Algoritmer og datastrukturer

Dato og tid: 03.01.2020, varighet 4 timer

Fagansvarlig: Jan Høiberg

Tillatte hjelpemidler: Alle skriftlige hjelpemidler er tillatt

Sensurfrist: 24.01.2020**,** resultater blir publisert i Studentweb.

OM OPPGAVESETTET

Dette oppgavesettet består av:

- En innledning med informasjon om oppgavesettet (siden som du nå leser).
- 4 eksamensoppgaver, nummerert fra 1 til 4, med flere mindre deloppgaver.
- En avsluttende side med et kommentarfelt, der du kan legge inn evt. kommentarer til oppgavene og til din egen besvarelse, f.eks. hvis du trenger å utdype noen svar, eller du har hatt tekniske problemer med programvaren for digital eksamen.

De 4 eksamensoppgavene er vektet slik:

1. 25 %
2. 20 %
3. 30 %
4. 25 %

Innen hver oppgave vektes alle deloppgaver likt.

1 Oppgave 1: Algoritmeanalyse

Nedenfor er det gitt 6 algoritmer (A-F), implementert som enkle metoder i Java. Hver algoritme har en arbeidsmengde som avhenger av parameteren `n`.

Hva er arbeidsmengden for hver av disse algoritmene, uttrykt med *O*-notasjon? Gi en kort begrunnelse for hvert av svarene dine.

Algoritme A:

```
public int it1(int n)
{
   int sum=0;
   for (int i=0; i<n; i+=2)
```

```
        sum++;
    return sum;
}
```

Svar A:

Algoritme B:

```
public int it2(int n)
{
    int sum=0;
    for (int i=0; i<n; i++)
        for (int j=-i; j<i; j++)
            sum++;
    return sum;
}
```

Svar B:

Algoritme C:

```
public int it3(int n)
{
    int sum=0;
    for (int i=0; i<n*n; i++)
        for (int j=1; j<i; j*=2)
            sum++;
    return sum;
}
```

Svar C:

Algoritme D:

```
public int rek1(int n)
{
    if (n<=1)
        return 1;
    else
        return rek1(n-1)+1;
}
```

Svar D:

Algoritme E:

```
public int rek2(int n)
{
    if (n<=1)
        return 1;
    else
        return rek2(n/2)+1;
}
```

Svar E:

Algoritme F:

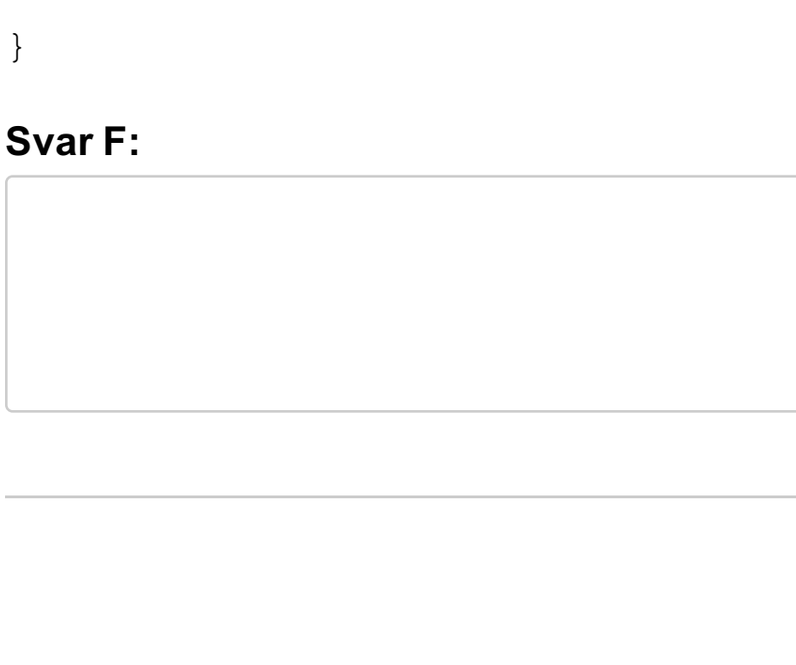
```
public int rek3(int n)
{
   if (n<=1)
      return 1;
   else
      return rek3(n/2) + rek3(n/2);
}
```

Svar F:

Maks poeng: 25

2 Oppgave 2: Binære trær - Teori

Følgende binære søketre er gitt:



Hvis vi gjør en nivå-for-nivå ("level order") traversering av dette treet, vil nodene oppsøkes i denne rekkefølgen:

15 7 19 6 9 18 22 3 12

Oppgave A
For treet gitt ovenfor, angi rekkefølgen som nodene oppsøkes i for hver av disse tre traverseringsmetodene:

1. Preorder
2. Inorder
3. Postorder

Svar A:

Oppgave B
Er søketreet gitt ovenfor et AVL-tre? Begrunn svaret ditt.

Svar B:

Oppgave C
Algoritmen for innsetting av nye verdier i et *vanlig* binært søketre skal brukes på treet gitt i figuren ovenfor. Følgende tre verdier settes inn i denne rekkefølgen:

8 5 21

Hva er nivå-for-nivå rekkefølgen av nodene i treet etter innsetting av disse tre verdiene?

Svar C:

Oppgave D
Algoritmen for fjerning av en verdi i et *vanlig* binært søketre skal brukes på treet gitt i figuren ovenfor. Hva er nivå-for-nivå rekkefølgen av nodene etter fjerning av roten i treet?

Svar D:

Maks poeng: 20

3 Oppgave 3: Binære trær - Programmering

I denne oppgaven skal du programmere tre Java-metoder. All Java-koden fra hver deloppgave skal skrives inn i det samme tekstfeltet som ligger nederst i oppgaven.

I et binært tre (ikke et søketre) er dataene enkle tegn. Nodene i treet er implementert slik i Java:

```
public class binærNode
{
   binærNode venstre, høyre;
   char data;
}
```

I tillegg til koden ovenfor, inneholder klassen en konstruktør.

Oppgave A

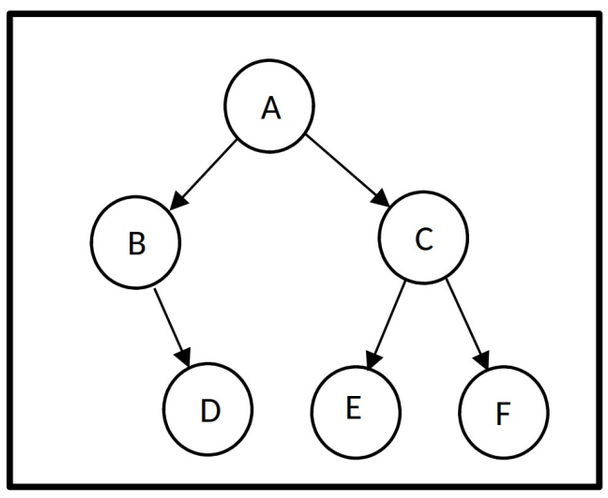
Skriv en *rekursiv* metode:

```
int antallBlader(binærNode rot);
```

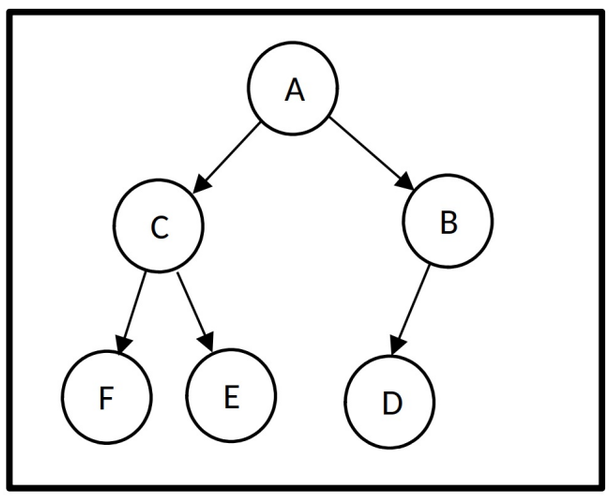
som returnerer antall bladnoder i treet med roten `rot`.

Oppgave B

Følgende binære tre er gitt:



Etter en *speiling* av treet, ser det slik ut:



En speiling medfører altså at: For enhver node i treet vil venstre og høyre subtre bytte plass.
Skriv en *rekursiv* metode:

```
void speil(binærNode rot);
```

som gjør en speiling av treet med roten `rot`.

Oppgave C

Skriv en *ikke-rekursiv* versjon av metoden `speil` fra oppgave B.

Denne metoden skal bruke en vanlig kø til å lagre noder underveis i speilingen. Du kan anta at du har tilgjengelig en klasse `Queue` som kan brukes til lagre objekter av klassen `binærNode` i en kø.
Klassen `Queue` inneholder bl.a. metodene `enqueue`, `dequeue` og `isEmpty`.

Skriv svarene på alle de tre deloppgavene A, B og C her:

```
1
```

Maks poeng: 30

4 Oppgave 4 - Hashing

Hvis vi bruker hashing med åpen adressering, kan f.eks en hashtabell med lengde 10 som inneholder 7 heltall fremstilles slik:

```
0: 19
1: 31
2:
3: 133
4: 63
5: 93
6:
7:
8: 8
9: 59
```

Hvis vi bruker kjeding (med lenkede lister) til håndtere kollisjoner, kan hashtabellen se slik ut:

```
0:
1: 31
2:
3: 93 63 133
4:
5:
6:
7:
8: 8
9: 19 59
```

Oppgave A

En hashtabell med lengde 10 som lagrer heltall er i utgangspunktet tom. Følgende 7 verdier skal settes inn i tabellen i denne rekkefølgen:

```
4371 1323 6173 4199 4344 9679 1989
```

Hashfunksjonen som brukes er denne:

```
int hash(int verdi)
{
  return verdi % 10;
}
```

Hashindeksen som beregnes for en verdi er altså lik resten ved heltallsdivisjon med hashlengden.

Vis hvordan hashtabellen ser ut etter innsetting av de 7 verdiene ovenfor, for hver av de følgende 5 metodene for håndtering av kollisjoner:

1. Lineær probing:

2. Kvadratisk probing:

3. Hashing med kjeding (med lenkede lister):

4. "Last come, first served" (med lineær probing):

5. "Robin Hood" hashing (med lineær probing):

Oppgave B

Forklar kort hvordan *fjerning* av verdier i en hashtabell gjøres for hver av følgende to metoder for hashing:

1. Hashing med åpen adressering (probing):

2. Hashing med kjeding (med lenkede lister):

Oppgave C

I oppgave A ovenfor er det to av metodene for hashing med åpen adressering -- "Last come, first served" og "Robin Hood" -- som kan flytte verdier som allerede er lagt inn i hashtabellen til nye posisjoner. Begge metodene er implementert som varianter av lineær probing i pensum.

Vil disse metodene fungere også hvis vi bruker kvadratisk probing til å beregne neste indeks i hashtabellen ved innsetting/flytting av verdier? Gi en kort begrunnelse for svaret for hver av de to metodene:

1. "Last come, first served" hashing

2. "Robin Hood" hashing

Maks poeng: 25

Egne kommentarer

Her kan legge inn evt. kommentarer til oppgavene og til din egen besvarelse, f.eks. hvis du trenger å utdype noen svar, eller du har hatt tekniske problemer med programvaren for digital eksamen.