Om eksamensoppgavene

Høgskolen i Østfold

EKSAMEN

Emnekode og -navn: ITF20006 Algoritmer og datastrukturer

Dato og tid: 14.05.2019, varighet 4 timer

Fagansvarlig: Jan Høiberg

Tillatte hjelpemidler: Alle skriftlige hjelpemidler er tillatt

Sensurfrist: 04.06.2019**,** 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.
- To vedlegg. Vedlegg A skal brukes i oppgave 3, vedlegg B skal brukes i oppgave 4.
- 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. 20 %
2. 30 %
3. 20 %
4. 30 %

Innen hver oppgave vektes alle deloppgaver likt.

Oppgave 1: Flervalgsoppgave

Oppgave 1 er en flervalgsoppgave med ulike delspørsmål:

- Hvert delspørsmål har fire svaralternativer, men bare ett av dem er riktig.
- Du skal for hvert av spørsmålene angi hvilket av de fire alternativene du mener er det riktige.

Merk at et riktig svar på et delspørsmål gir 1 poeng, mens et feil svar gir 0.5 minuspoeng. Hvis du lar være å svare på et delspørsmål, gir dette null poeng:

- Det vil derfor ikke lønne seg å "gjette" på et alternativ hvis du ikke vet svaret på et delspørsmål, da dette sannsynligvis vil gi deg minuspoeng.
- Det er da bedre å la delspørsmålet være ubesvart.

Oppgaven har 20 nummererte delspørsmål. Du kan totalt score maksimalt 20 poeng, hvis alle svarene dine er riktige. Dårligste totale poengscore for oppgave 1 er null poeng.

1. Vi skal bruke algoritmen som anvender en stack til å sjekke om paranteser i input er balanserte. Hva er det største antallet paranteser som lagres på stacken, når algoritmen får følgende input:

   ```
   ((a)(b(c))(d(e))f)
   ```

- 1
- 3
- 2
- 4 eller flere

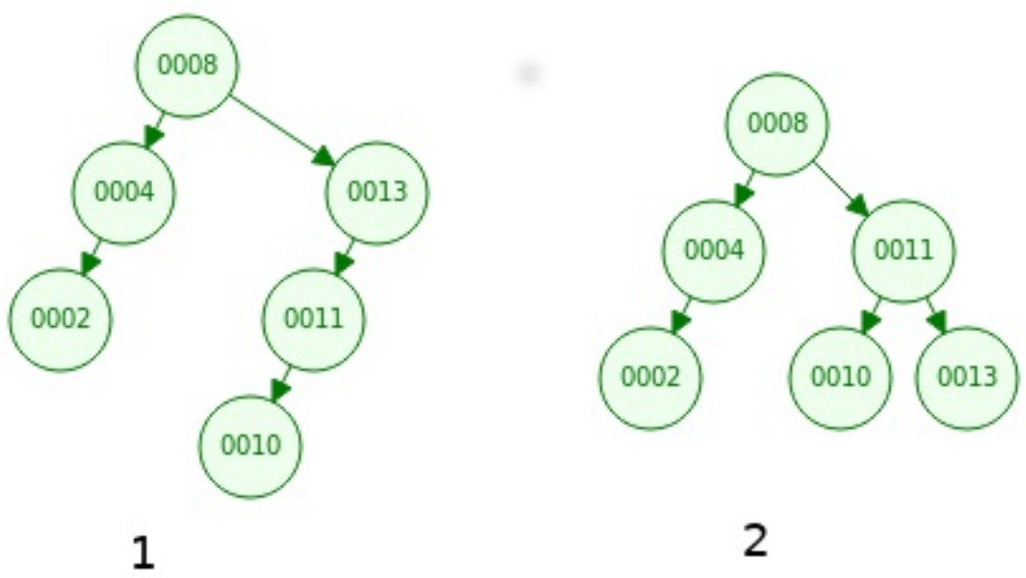
2. Hva er verdien av postfixuttrykket: `6 6 2 1 + - *`

- 18
- 24
- 9
- 36

3. Verdiene A, B, C og D legges inn i en vanlig kø, i alfabetisk rekkefølge. De tas deretter ut igjen av køen. I hvilken rekkefølge vil verdiene bli tatt ut?

- DCBA
- DCAB
- ABCD
- ABDC

4. Hvilke av disse to trærne er et binært søketre?



- Bare 2
- Ingen av dem
- Både 1 og 2
- Bare 1

5. Hvilke av de to trærne i forrige oppgave er et AVL-tre?

- Bare 1
- Både 1 og 2
- Bare 2
- Ingen av dem

6. Hvilke av de to trærne i oppgave 4 ovenfor er en heap?

- Bare 1
- Både 1 og 2
- Ingen av dem
- Bare 2

7. En array med n tall kan sorteres med en arbeidsmengde som i verste tilfelle ("worst case") er $O(n)$ hvis følgende er oppfylt:

- Alle tallene er heltall med maksimalt m siffer, der m er mye mindre enn n
- Sammenligningen av tallene kan gjøres i konstant tid, dvs. som en $O(1)$ operasjon
- Antall tall som er ulike er mindre enn $n/2$
- Dette kan aldri gjøres fordi sortering alltid er minst $O(n \log n)$

8. En array med n tall kan sorteres med en arbeidsmengde som i verste tilfelle er $O(n)$ hvis følgende er oppfylt:

- Dette kan aldri gjøres fordi sortering alltid er minst $O(n \log n)$
- Alle tallene er mindre enn $n^2/2$
- Arrayen er nesten sortert, det er bare noen få tall som står feil
- Tallene er jevnt (uniformt) fordelt mellom største og minste verdi

9. Hva er arbeidsmengden, i verste tilfelle, for å sette inn ett nytt datalelement i et binært søketre som allerede inneholder n elementer?

- $O(n)$
- $O(n \log n)$
- $O(n^2)$
- $O(\log n)$

10. Hva er arbeidsmengden, i verste tilfelle, for å sette inn n datalelementer i et binært søketre som til å begynne med er tomt?

- $O(n^2)$
- $O(n \log n)$
- $O(n\sqrt{n})$
- $O(n^3)$

11. Hva er arbeidsmengden for å sette inn n datalelementer i et AVL-tre som til å begynne med er tomt?

- $O(n^2)$
- $O(n)$
- $O(n\sqrt{n})$
- $O(n \log n)$

12. Et B-tre av orden m er et flerveis søketre der vi krever at antall nøkkelverdier i alle noder (unntatt roten) skal være:

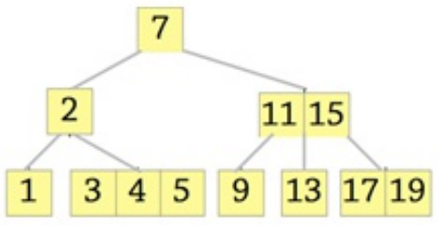
- Minst $2m$
- Eksakt lik $(m-1)/2$
- Minst $(m-1)/2$
- Høyst $(m-1)/2$

13. Det maksimale antall nøkkelverdier som kan lagres i et B-tre av orden 4 som har 3 nivåer, er:

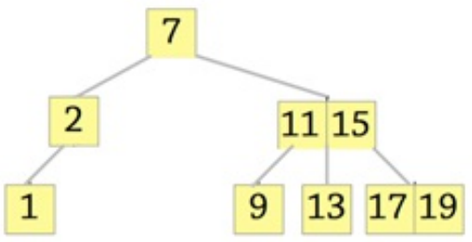
- 63
- 32
- 255
- 127

14. Hvilken av de fire figurene nedenfor er et B-tre av orden 3 (et "2-3 tre")?

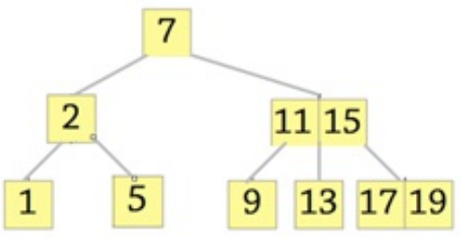
a



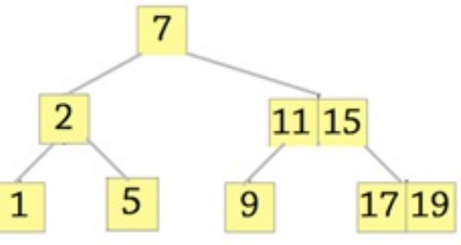
b



c



d



- a
- b
- c
- d

15. Hashtabellen nedenfor har hashlengde lik 10. Hashindekser beregnes som siste siffer i tallet som skal settes inn. Det brukes lineær probing. På hvilken indeks vil verdien 8237 bli satt inn?

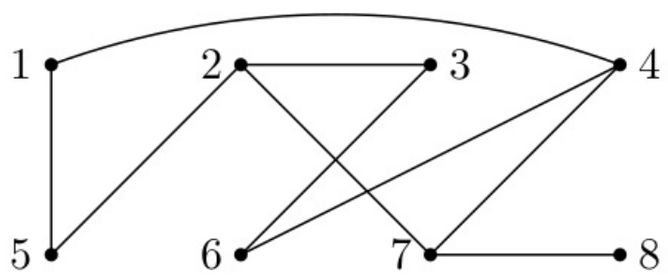
0	9050
1	1001
2	
3	
4	
5	
6	
7	9877
8	2037
9	1059

- 2
- 3
- 0
- 6

16. Hvilken strategi for insetting/søking brukes i hashing med kjeding, der kollisjoner løses ved å legge elementer først i en en lenket liste?

- "Cashmere Cat"-hashing
- "Last come, first served"-hashing
- "Robin Hood"-hashing
- "First come, first served"-hashing

I de fire neste spørsmålene skal følgende urettede graf brukes:



Grafen har 8 noder som er nummeret fra 1 til 8. I algoritmene som skal brukes på denne grafen, skal alltid naboene til en node oppsøkes i stigende nummerrekkefølge.

17. I hvilken rekkefølge oppsøkes nodene i en dybde-først traversering med start i node 1?

- 1, 4, 5, 8, 7, 6, 2, 3
- 1, 5, 2, 3, 6, 4, 7, 8
- 1, 4, 6, 3, 2, 5, 7, 8
- 1, 2, 3, 4, 5, 6, 7, 8

18. I hvilken rekkefølge oppsøkes nodene i en bredde-først traversering med start i node 1?

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 4, 5, 6, 7, 2, 3, 8
- 1, 4, 6, 3, 2, 5, 7, 8
- 1, 5, 4, 2, 3, 6, 7, 8

Grafen som er gitt i figuren ovenfor er også en vektet graf, Kantlengdene er ikke satt inn på figuren, men kantlengdematrisen for grafen ser slik ut:

$$\begin{bmatrix} 0 & \infty & \infty & 2 & 3 & \infty & \infty & \infty \\ \infty & 0 & 1 & \infty & 4 & \infty & 7 & \infty \\ \infty & 1 & 0 & \infty & \infty & 7 & \infty & \infty \\ 2 & \infty & \infty & 0 & \infty & 6 & 13 & \infty \\ 3 & 4 & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & 7 & 6 & \infty & 0 & \infty & \infty \\ \infty & 7 & \infty & 13 & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Her er f.eks. lengden på de to kantene fra node 1 til nodene 4 (lengde lik 2) og 5 (lengde lik 3) gitt i første linje i matrisen.

19. Floyds algoritme skal brukes på den gitte grafen, slik at kantlengdematrisen transformeres til en matrise med korteste veilengder mellom alle par av noder. Hvordan ser første rad i denne matrisen ut etter at

Floyds algoritme er ferdig?

- 0 7 8 2 1 9 11 11
- 0 8 7 2 3 6 12 11
- 0 7 8 2 3 8 14 15
- 0 8 8 2 3 8 8 10

20. Dijkstras algoritme skal brukes på den gitte grafen, med node 1 som start-node. Algoritmen er programmert slik at den alltid velger noden med lavest nummer, hvis det i et steg er to eller flere noder som har samme korteste avstand til start-noden. I hvilken av følgende rekkefølger vil nodene få beregnet sin korrekte korteste avstand fra node 1?

- 1, 4, 5, 2, 3, 6, 7, 8
- 1, 4, 5, 6, 3, 2, 7, 8
- 1, 4, 5, 2, 6, 7, 3, 8
- 1, 4, 5, 2, 3, 7, 6, 8

2 Oppgave 2: Søking

I en array av lengde n er det lagret en sekvens med n heltall i stigende rekkefølge. Alle tallene er forskjellige.
Sekvensen kan enten begynne i første element i arrayen, eller den kan begynne et sted inne i arrayen. Hvis sekvensen begynner inne i arrayen, gjøres det en "wrap-around" slik at de stigende tallene fortsetter fra starten av arrayen (indeks 0) etter elementet på indeks $n-1$.

0	1	2	3	4	5
2	3	5	7	8	9

0	1	2	3	4	5
7	8	9	2	3	5

I figuren ovenfor er $n = 6$. I arrayen til venstre starter sekvensen med stigende tall i indeks 0 og går helt ut til indeks 5. I arrayen til høyre starter sekvensen i indeks 3 og slutter i indeks 2.
I deloppgavene a og b nedenfor skal du programmere to funksjoner som finner ut hvor i arrayen en slik sekvens med stigende tall starter. Besvarelsen på begge deloppgavene skrives inn i det samme tekstfeltet for Java-kode nedenfor.

Oppgave a)

Skriv en ikke-rekursiv funksjon `int startSekvens1(int A[])`, der parameteren `A` er en array som inneholder en sekvens med stigende tall som beskrevet ovenfor. Funksjonen skal returnere indeksen der sekvensen starter. Den skal bruke en enkel `for`-løkke til å finne startindeksen. Arbeidsmengden til funksjonen skal være $O(n)$.

Oppgave b)

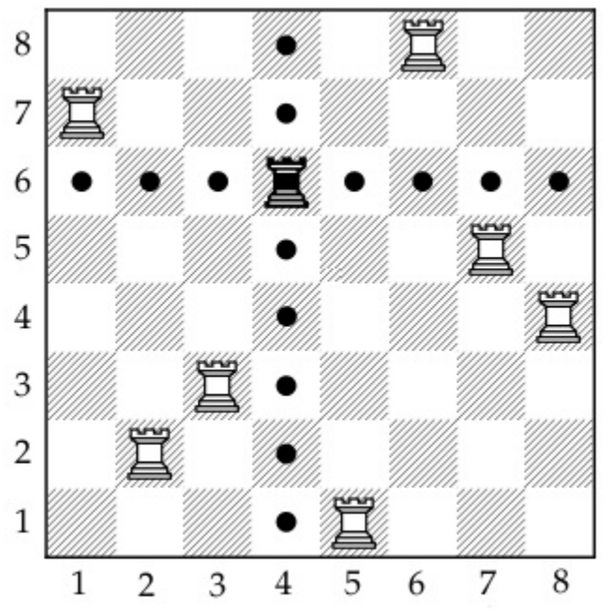
Skriv en ikke-rekursiv funksjon `int startSekvens2(int A[])`, der parameteren `A` er en array som inneholder en sekvens med stigende tall som beskrevet ovenfor. Funksjonen skal returnere indeksen der sekvensen starter. Den skal bruke en enkel `while`-løkke til å finne startindeksen. Arbeidsmengden til funksjonen skal være $O(\log n)$.

3 Oppgave 3: Et sjakkproblem

I det såkalte tårnproblemet har vi et $n \times n$ sjakkbrett og n tårn som skal plasseres på dette brettet, slik at ingen av dem kan slå hverandre.

Sjakkbrikken som kalles tårn kan slå alle brikker i samme rad og i samme kolonne som den selv står i. Med andre ord går problemet ut på å plassere nøyaktig ett tårn i hver rad og nøyaktig ett tårn i hver kolonne. Det er flere ulike løsninger på dette problemet for en gitt verdi av $n > 1$.

Figuren nedenfor viser en løsning av tårnproblemet for $n = 8$. Det er også angitt (med sorte prikker) hvilke felter som det svarte tårnet kan slå.



En løsning av tårnproblemet kan representeres som en heltallsarray, der tårnenes plassering lagres rad for rad. Hvis vi nummererer rader og kolonner som angitt i figuren ovenfor, kan denne løsningen lagres i en array `p` som:

```
p = { 5, 2, 3, 8, 7, 4, 1, 6 }
```

Vedlegg A inneholder Java-klassen `tårn`. Dette er et ufullstendig Java-program for å finne alle løsninger av tårnproblemet for en gitt verdi av n. Det mangler her en del kode som du skal skrive:

- Programmer den rekursive funksjonen `lagLøsning` i klassen `tårn`.
- Legg også inn evt. ekstra variabler/arrayer og annen kode i klassen `tårn` og i `main`, som trengs for å få løsningen din til å fungere.

4 Oppgave 4: Binært søketre

Klassen `BST` som er gitt i vedlegg B inneholder en ufullstendig implementasjon av et binært søketre som lagrer heltallsverdier. I tillegg til pekere til venstre og høyre barn, inneholder hver node tre heltall:

- Variabelen `data` som lagrer selve dataverdien som søketreet er sortert på. Alle nodene i treet skal ha forskjellige dataverdier.
- Variabelen `antall` som lagrer antall forekomster av verdien lagret i `data`.
- Variabelen `høyde` som skal brukes i deloppgavene b og c nedenfor.

I hver av deloppgavene nedenfor skal du programmere én metode i klassen `BST`. Alle metodene skrives inn i det samme tekstfeltet for Java-kode som er tilgjengelig nederst.

Oppgave a)

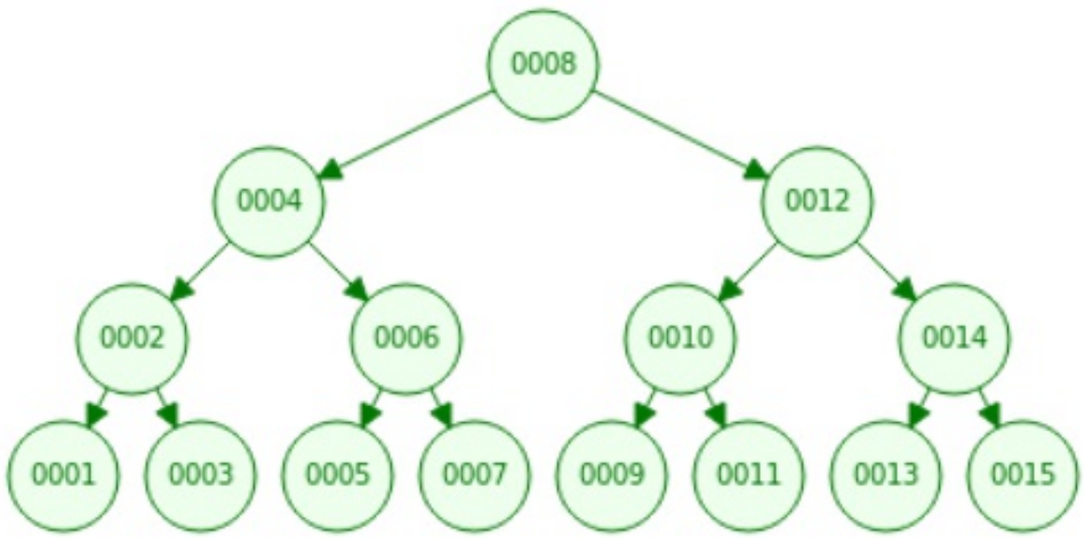
Programmer metoden `settInn` i klassen `BST`, som setter inn en ny forekomst av en dataverdi i søketreet. Metoden skal være ikke-rekursiv og så effektiv som mulig.

Høyden til et binært tre defineres som lengden av den lengste veien fra roten til en bladnode. Et tre med bare 1 node har høyde lik 0, mens et tre med 2 nivåer har høyde lik 1. Generelt har et tre med k nivåer en høyde lik $k-1$. Høyden til et tomt tre kan defineres som -1.

Oppgave b)

Nodene i søketreet definert i klassen `BST` inneholder variabelen `høyde`, som for hver node skal lagre høyden til subtreet med rot i denne noden. Programmer den rekursive metoden `settHøyde`. Metoden skal kunne brukes til å sette `høyde` til riktig verdi for alle nodene i treet.

I et Perfekt Binært Tre - et PBT - er alle nivåene i treet helt fylt opp med noder. Et eksempel på et PBT med fire nivåer, som også er et søketre, er dette:



Et PBT vil alltid være fullkomment balansert. Mer presist kan vi definere et perfekt binært tre rekursivt på denne måten:

I et PBT med høyde $h = 0$ har rotnoden to tomme subtrær. For et PBT med høyde $h > 0$, er de to subtrærne til rotnoden begge et PBT med høyde $h - 1$.

Oppgave c)

Programmer metoden `erPerfekt` i klassen `BST` , som skal returnere `true` hvis treet er et PBT, og `false` ellers. Du kan anta at høyden er satt til riktig verdi for alle nodene i treet. Metoden skal være rekursiv.

Vedlegg A

```
// Finner og skriver ut alle løsninger av tårnproblemet
public class tårn
{
   public static int n;   // Størrelse på brettet
   public static int p[]; // Lagrer en løsning på tårnproblemet

   <Evt. andre variabler/arrayer som trengs>

   public static void lagLøsning(int rad)
   {
   // Skriver ut alle mulige løsninger på tårnproblemet rekursivt
   // ved å sette ut tårn fra og med denne raden og til og med rad n.
   // Det er allerede satt ut tårn på radene 1, 2, ..., rad-1

   <Skal programmeres i oppgave 2>
}

public static void skrivLøsning()
{
   for (int i = 1; i <= n; i++)
      System.out.print(p[i] + " ");
   System.out.println();
}

public static void main(String args[])
{
      System.out.print("n?: ");
      n = Integer.parseInt(System.console().readLine());
      p = new int[n + 1];

      <Evt. annen kode som er nødvendig>

      lagLøsning(1);
   }
}
```

Vedlegg B

```
// Binært søketre med heltallsverdier
public class BST
{
   // Indre klasse for nodene i treet
   class treNode
   {
      int data;         // Verdi/data som lagres i noden
      int antall;       // Antall forekomster av denne verdien
      int høyde;        // Høyden av subtreet med rot i denne noden
      treNode venstre; // Venstre barn
      treNode høyre;   // Høyre barn

      // Konstruktør
      public treNode(int verdi)
      {
```

```
            data = verdi;
            antall = 1;
            venstre = høyre = null;
        }
    }

    // Roten for hele søketreeet
    treNode rot;

    // Konstruktør, lager et tomt søketre
    public void BST()
    {
        rot = null;
    }

    // Sjekk for tomt tre
    boolean erTomt()
    {
        return (rot == null);
    }

    void settInn(int verdi)
    {
        // Setter inn en ny forekomst av en verdi i søketreet

        < Skal programmeres i oppgave 4 a >
    }

    int settHøyde(treNode t)
    {
        // Setter høyden riktig for alle noder i treet med rot i t
        // Returnerer verdien som blir satt for høyden i t

        < Skal programmeres i oppgave 4 b >
    }

    boolean erPerfekt(treNode t)
    {
            // Returnerer true hvis treet med rot i t er et perfekt binært tre
            // Forutsetter at høyden er satt riktig i alle noder

            < Skal programmeres i oppgave 4 c >
    }
}
```

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.