# TOD063 Datastrukturer og algoritmer

| | |
|---|---|
| Øving | : 6 |
| Utlevert | : Veke 16-18 |
| Innleveringsfrist | : **fredag 7. mai 2010** |
| Klasse | : 1 Data og 1 Informasjonsteknologi |

Gruppearbeid: 2- 3 personar pr. gruppe som før.

## Formål

Forstå tre, haug, sortering med haug og hashing.

## Oppgåve 1

Du vil finna (deler av) ein klasse for binære søketre (BS-tre) og andre nødvendige klassar / interface i filen bstre_øving6.java. Denne filen kan du bruka til oppgåvene nedanfor, eller du kan laga dine eigne variantar av klassane.

a) Boka definerer høgda av eit binært til å vere lengda på den lengste stien frå rota til eit blad. Eit tomt tre får dermed høgde 0. Lag ein rekursiv metode i BSTre for å finne høgda av eit binært tre.

b) Lag ein metode som skriv ut alle verdiar i eit BS-tre mellom to gitte grenser nedre og øvre. Verdiane skal koma ut i sortert rekkefølgje.

c) Lag eit testprogram som genererer for eksempel 100 tilfeldige binære søketre med 1024 nodar. (Testkjør først med eit lite tre.) For å generere eit tilfeldig binært søketre må nøklane kome i tilfeldig ordning. Dette kan de få til ved å bruke Random-klassen og bruke heiltal som nøklar. Skisse nedanfor:

```
import java.util.*;
…
Random terning = new Random();
…
int tal = terning.nextInt();
…
```

Programmet skal skrive ut

i) antal nodar n
ii) den minimale teoretiske høgda (her vil det vere naturleg å lage ein metode som reknar ut denne verdien for ein gitt verdi av n).
iii) den maksimale teoretiske høgda
iv) minste høgde i løpet av køyringane
v) største høgde i løpet av køyringane
vi) gjennomsnittleg høgde av alle køyringane

d) Det kan visast at den gjennomsnittlege høgda av eit binært søketre når vi set inn n tilfeldige nøklar er $O(\log_2 n)$, dvs at høgda er tilnærma lik $c*\log_2 n$ der c er ein konstant. For å bestemma konstanten kan de måle den gjennomsnittlege høgda for 1 bestemt verdi av n, for eksempel n = 1024. Dermed får de ei likning med c som ukjent. Bruk formelen for å finna c og deretter gje eit overslag for den gjennomsnittlege høgda når n = 4096. Køyr programmet med n = 4096 og vurder resultatet. Stemmer det med den utrekna verdien for høgda?

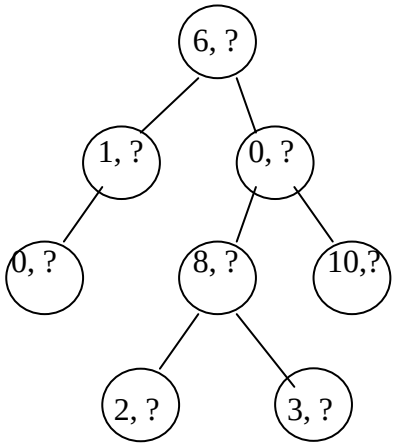
## Oppgave 2

a) I et binært tre har datadelen to felter: data1 og data2, begge heltall. data2-verdien for en vilkårlig node er summen av alle data1-verdiene i hele undertreet som denne noden er rot i.

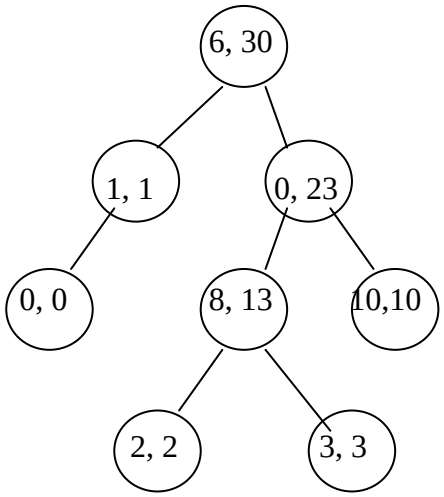
Vi har gitt et slikt binært tre T der alle data1-feltene har fått verdi. Lag en rekursiv metode i Java (eller en detaljert algoritme) som går gjennom dette treet og regner ut data2-verdiene i alle noder.

Eksempel nedenfor:

Før:




Etter:




NB! Det er ikkje meininga de skal lage main()-program og testkøyre metoden.

b)

Teikn 2-3 treet som vi får når vi startar med eit tomt tre og set inn 32, 43, 5, 85, 63, 30, 1, 10 og 15. Vis treet etter kvar innsetting.

Slett følgjande element frå treet ovanfor (i gitt rekkefølgje): 5, 85, 10, 15 og 43.

## Oppgave 3

a) En spesiell type binære tre er kalt en haug ("heap").
Hvilke to vesentlige egenskaper har en haug?

b) I minst en av de tre tabellene under er elementene ordnet slik at de danner en **maks-haug** (node >= v.b og h.b ).
Finn hvilke(n). Begrunn svaret.

A: [0] … [9]

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 9 | 15 | 12 | 7 | 4 | 2 | 1 | 6 | 5 | 3 |

B: [0] … [9]

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 15 | 12 | 10 | 11 | 2 | 6 | 3 | 4 | 8 | 1 |

C: [0] … [9]

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 15 | 10 | 14 | 8 | 7 | 13 | 6 | 2 | 5 | 4 |

c) Elementene i tabellen D under utgjør en maks-haug.:

D: [0] … [9]

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 14 | 12 | 6 | 11 | 10 | 2 | 5 | 1 | 7 | 4 |

i) Tegn haugen som tre og vis hvordan treet ser ut når vi først fjerner elementet med verdi 14 og deretter organiserer til en haug (tips: etter fjerning av 14, flyttes først siste element (4) til rotposisjon, osv…).

ii) Etter at vi har utført pkt i) setter vi nå et nytt element med verdi 13 inn sist i D og organiserer igjen til en haug. Vis hvordan treet nå ser ut.

d) Forklar kort hvordan prinsippet i pkt c) kan brukes til å sortere elementer.

e) Finn antall sammenligninger i verste tilfelle uttrykt i O-notasjon som går med til å organisere en tabell med n antall elementer som en haug. Forklar eventuelle utregninger/formler som du kommer frem til.

f) Det skal lages en minimumshaug (node <= v.b og h.b ). Elementene blir plassert i en tabell som på figuren under.

D: [0] … [9]

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Tegn først opp treet med de elementene som her er gitt.
Du skal nå lage en minimumshaug. Vis alle overgangene ved å tegne opp treet på nytt for hver ombytting inntil du har en minimumshaug.

## Oppgåve 4 (Hashing)

a) Bruk tabellstorleik 10 (vanlegvis eit dårleg val, men her gir det oss betre oversikt) . Tabellstorleiken bør som regel vere eit primtal eller eit produkt av to store primtal. La hashfunksjonen returnera siste siffer i bilnummeret. Vis korleis tabellen ser ut etter vi har sett inn følgjande bilnr (i gitt rekkefølgje): SU65431, TA14374, ZX87181, ST47007, VV50000, UV14544, SU32944, når vi bruker

1 - open adressering med lineær sondering (”linear probing”, steglengde alltid 1)
2 - kjeda lister – tabell med 10 posisjonar

b) De skal implementere deler av ein klasse HashTabell slik at vi er uavhengige av kva objekttype vi har i tabellen. Dette får vi til ved å definere ein **interface** HashNoekkel som har 3 metodar: ein *hash*-funksjon, ein *steglengde*-funksjon (”probe-function”) og ein metode *lik*. Altså:

```
interface HashNoekkel {
      public int hash(int tabLengde);
      public int steglengde(int tabLengde);
      public boolean lik(HashNoekkel hn);
}
```

Parameteren tabLengde er storleiken på hashtabellen. I klassen HashTabell vil vi ha:

**private** HashNoekkel[] tab;

No slepp vi å gjere endringar i HashTabell avhengig av kva type element vi skal ha i tabellen. Dersom vi ønskjer å definere objekt av ObjektType som skal leggjast inn i ein HashTabell, så skriv vi:

```
class ObjektType implements HashNoekkel{…}
```

og implementerer dei tre metodane i HashNoekkel. Når vi har bruk for hash-verdien til eit objekt i ein av metodane i klassen HashTabell reknar vi ut ved:

```
int i = obj.hash(lengde av hash-tabellen);
```

Variabelen *obj* er av HashNoekkel. Tilsvarande for probe-funksjonen.

I HashTabell skal de ha med ein konstruktør og metodar for å setja inn nytt element og søkja etter element med gitt nøkkelverdi. Sjå også utdelt skisse til HashTabell med type *int* som nøkkel.

Lag til eit enkelt testprogram der de set inn og søkjer etter eit lite antal bilar i ein hash-tabell. Bruk bilnummer som nøkkelfelt og la merke og årsmodell vere ekstra informasjon. De kan velja tabellstorleik 11 for hashtabellen og som hashfunksjon kan de bruke

(<Unicode-verdien til første bokstav>*<dei tre midtsiffera i bilnr>) % 11.

De kan sjølv velja steglengde-funksjon (”probe-function”, alltid 1 ved ”linear probing”, ny hash-funksjon ved ”double hashing”). De kan bruke nullpeikar som markering for tom posisjon i tabell (ville gitt litt problem viss de også skulle hatt med sletting).

Frivillig:
Lag også en metode for sletting eller forklar hvordan du vil lage en slik metode.