Daniel Kårtveit(600884), Lars Jevnaker(600861), Øyvind Holter(599033)

## Oppgave 1

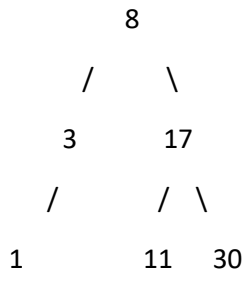
a)

Et balansert BS-tre er når alle noder utenom den nederste har to barn og det er like mange blander på hver side.

b)

Dette treet er ikke balansert fordi alle nodene ikke har like mange barn, og fordi de er ikke like lange på hver side.

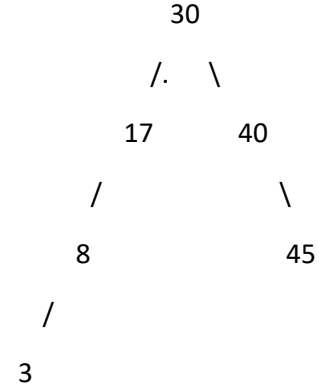
Etter høyrerotasjon:




c)

Dette treet er ikke balansert fordi alle nodene ikke har 2 barn, og fordi venstre side or kortere enn høyre side.

Etter venstrerotasjon:




d)

Grunnen for at det er viktig at et BS-tre er balansert er fordi dette fører til en mer effektiv søking, og dette hjelper til med mer effektiv innsetting og sletting.

## Oppgave 2

a) Starter på C og går gjennom barna alfabetisk. Da blir barna til C, e og f. deretter finner vi barna til første alfabetiske barn, altså e. Det blir b og d. Siden B har ingen barn som ikke er nevnt går vi til d som har barnet a.

**c-e-f-b-d-a**

b) Starter på C og går så ned til venstre som er e. deretter går vi opp til b. Repeterer dette slik at vi får d så a, legger så til manglende barn f.

**c-e-b-d-a-f**

c)

Sette opp grafen:

V = {a,b,c,d,e,f}

Siden denne grafen har uordna par, vil (v,w)=(w,v)

E = {a,d}.{d,b}.{d,e}.{b,f}.{b.e}.{e,c}.{e,f}.{c,f}

nabomatrise

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | o | o | o | x | o | o |
| B | o | o | o | x | x | x |
| C | o | o | o | o | x | x |
| D | x | x | o | o | x | o |
| E | o | x | x | x | o | x |
| F | o | x | x | o | x | o |

nabolister

A:[E]

B:[D,E,F]

C:[E,F]

D:[A,B,E]

E:[B,C,D,F]

F:[B,C,E]

**Oppgave 3**

a) Hashing lagrer og finner data ved å bruke en søkenøkkel som bestemmer indeksen i en hashtabell. Eks alfabetet.

Hashfunksjoner brukes for å bestemme hvilken posisjon i tabellen ett element har, eller for å finne ett element i tabellen. En perfekt hashfunksjon vil sørge for at ingen element ender opp med samme nøkkel, men hvis dette skjer kalles det en kollisjon. Det er mulig å få kollisjon med mange elementer, dette kalles en opphoping.

**b)** Lineær probing, vil flytte elementet ned ett steg (i dette tilfellet) hvis indexen element skal plasseres i er brukt. Kjedet lister plasserer bare elementet i indexen der den skal, så den ender opp med å erstatte elementet som var der fra før. Så tabellene blir slik:

| Lineær probing. | kjeda lister |
|---|---|
| 0 VV50000 | 0 VV50000, |
| 1 EL65431 | 1 ZX87181 |
| 2 ZX87181 | 2 |
| 3 | 3 |
| 4 TA14374 | 4 EL32944 |
| 5 UV14544 | 5 |
| 6 EL32944 | 6 |
| 7 EL47007 | 7 EL47007 |
| 8 | 8 |
| 9 | 9 |

**c)** Svart på i java.

d) Default hashcoden java bruker har en equals metode som sørger for at alle element får sin egen verdi. Med å override denne metoden og bruke en basic hash metode i stedet vil den kun bruke standard måten å regne ut en hashverdi, som vil gi like element lik verdi.

e) Java code.

Ved å bruke en binær søke metode, ender koden opp med å bruke ca 3,7sek. Med å bruke hashset brukte koden ca 0.01 sek på å fullføre søket.