Løsningsforslag DAT200 eksamen kont 2019

Oppgave 1

Worst-case O(n)

Best-case O(1)

Oppgave 2

$O(n^2)$

Oppgave 3

O(log(n)). Dette er en formulering av algoritmen Binærsøk. Den fungerer med gjentatt halvering av problemstørrelsen, derfor O(log(n))

Oppgave 4

Sett inn på slutten: Like rask på begge

Fjern på slutten: Like rask på begge

Sett inn på starten: LinkedList

Fjern fra starten: LinkedList

Hent ut element med oppgitt indeks: ArrayList

Gi element med oppgitt indeks en ny verdi: ArrayList

Hent ut neste element i en for each loop: Like rask på begge

Oppgave 5

For å sette inn på en valgfri indeks i en arraylist:

- Sjekk om arrayen er full. Hvis arrayen er full
    - Lag en ny array med dobbelt så mange elementer
    - Kopier inn elementene fra den gamle arrayen til den nye
- For alle elementer fra og med siste gyldige element i lista og til og med indeksen som man skal sette inn i:
    - Flytt elementet ett hakk bakover i lista
- Nå er det en ledig plass på den oppgitte indeksen, så da overskriver man gammel verdi med den nye verdien som skal settes inn.

Oppgave 6

Dette er et AVL tre.

Beskrivelse av AVL tre

- Binærtre

- Binært søketre. Det vil si at venstre barn er mindre enn noden selv, mens høyrebarn er større enn noden selv.
- Selvbalanserende. AVL-trær har et krav om at for hver interne node så må høyden på venstre og høyre subtre være maksimalt én forskjellig. Hvis man ette ren innsetting finner at det ikke stemmer så må man gjøre rotasjoner for å gjøre treet til et lovlig AVL tre igjen.

Begrunnelse for at det er et AVL tre

- Det er en rettet graf uten sykler hvor hver node har maksimalt 1 innkommende kant og maksimalt to utgående kanter. Det er derfor et binærtre. Forskjellen i dybde på venstre og høyre subtre er maksimalt 1 for alle nodene i treet. Hvis du ordner navnene alfabetisk så er alltid venstre barn lavere enn noden selv og høyrebarn høyere enn noden selv.

Poeng for mindre nøyaktige svar

- Binært søketre: 8 poeng hvis korrekt beskrevet
- Binærtre: 6 poeng
- DAG: 5 poeng
- Graf: 4 poeng

Merk at oppgaven har 3 deler, og du må ha med alle tre for å få full score.

- Identifiser datastrukturen
- Begrunn svaret
- Beskriv datastrukturen

Oppgave 7

Pseudokode:

Rekursiv metode, starter med å kalle den på rota:

Formelberegner(trenode):

Hvis noden inneholder et tall, er den en bladnode. Returner tallet.

Hvis noden inneholder «+», returner «formelberegner(venstrebarn) + formelberegner(høyrebarn)», to rekursive kall til formelberegner, ett for hvert barn

Hvis noden inneholder «*», returner «formelberegner(venstrebarn) * formelberegner(høyrebarn)»

Og tilsvarende for andre operatorer

Java-kode:

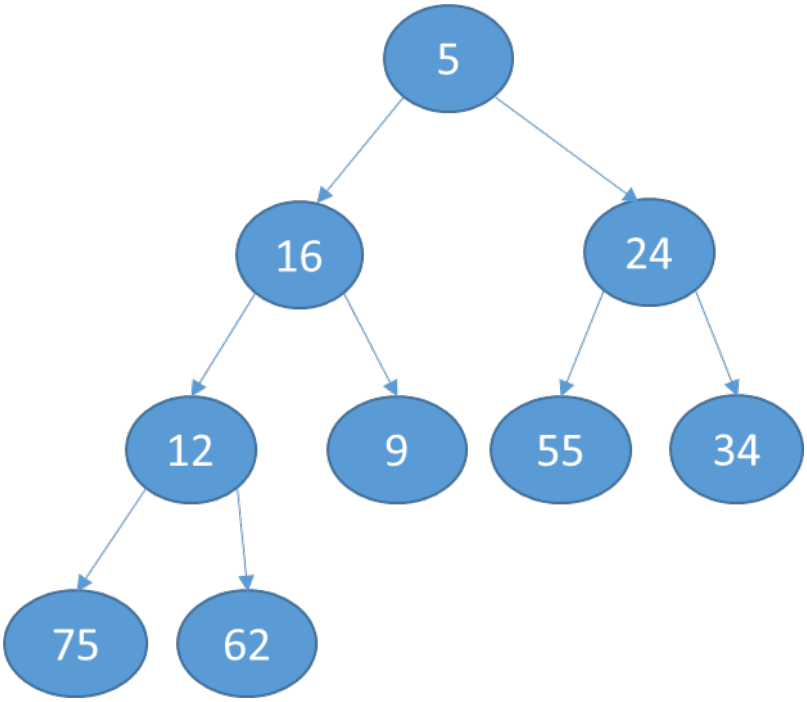
```
public class Formeltre extends Binaertre<String> {
	public double formelberegner() {
		return formelberegner(rot);
	}

	private double formelberegner(Binaertre<String>.Binaertrenode noden) {
		double tall = 0;
		try {
			tall = Double.parseDouble(noden.objekt);
		} catch (NumberFormatException e) {
			if (noden.objekt.equals("+")) {
```

```
                        tall = formelberegner(noden.venstrebarn) +
formelberegner(noden.hoyrebarn);
                    }
                    if (noden.objekt.equals("*")) {
                        tall = formelberegner(noden.venstrebarn) *
formelberegner(noden.hoyrebarn);
                    }
            }
            return tall;
        }
}
```

Merk: Et alternativ til denne rekursive formuleringen er å bruke en postorder iterator og en eksplisitt stabel med tall. Når iteratoren gir ut et tall så legger man (push) tallet på stabelen. Når iteratoren gir ut en operator så tar man (pop) to tall fra stabelen, utfører operasjonen på dem, og legger resultatet tilbake på stabelen.

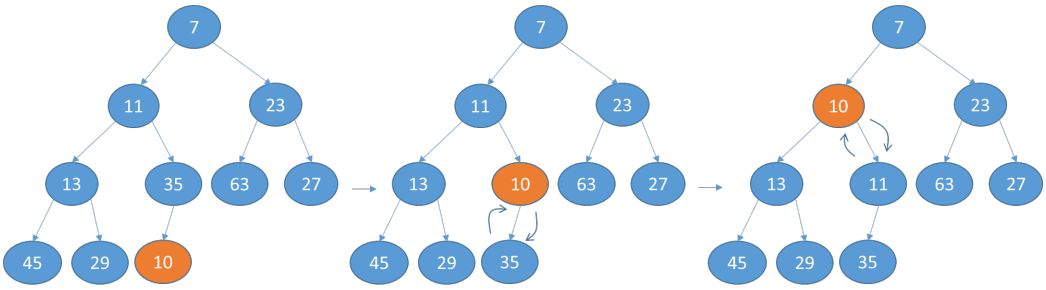
Oppgave 8



Oppgave 9

I en minimumhaug skal alle noder være mindre enn eller lik barna sine. I en maksimumhaug skal alle noder være større enn eller lik barna sine. I denne haugen så er de fleste nodene mindre enn barna sine, men det er ett unntak: noden 16 er større enn barna sine. Derfor er dette hverken en minimumhaug eller en maksimumhaug og derfor ikke en lovlig haug.

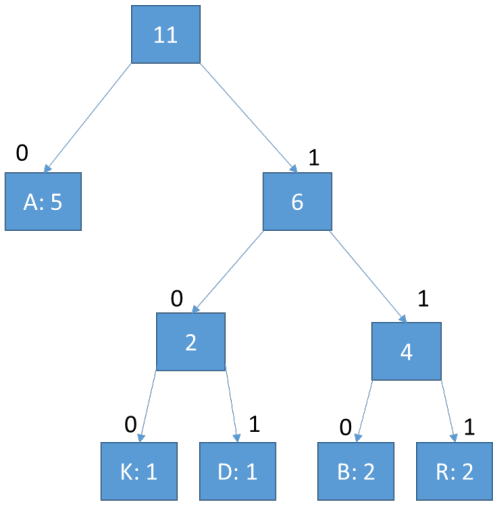
Oppgave 10



Steg:

- Setter inn elementet 10 på første ledige sted
- Sjekker elementet 10 mot forelder 35. Finner ut at det er mindre. Bytter elementet med forelder
- Sjekker elementet 10 mot den nye forelderen, 11. Finner ut at det er mindre. Bytter elementet med forelder.
- Sjekker elementet 10 mot den nye forelderen, 7. Finner ut at det er større. Da ligger elementet 10 i riktig posisjon i haugen og algoritmen er ferdig.

Oppgave 11



Selve treet er løsning på oppgaven. Kodene som blir produsert fra treet er:

A: 0

B: 110

D: 101

K: 100

R: 111

Oppgave 12

Ut-graden til en node i denne representasjonen er antall elementer i «Edges» hashmap-et. Så «edges.size()» gir ut-graden.

Kjøretid O(1) med antakelse om at HashMap lagrer antall elementer. O(h) hvor h er størrelsen på hashmap-et hvis det ikke lagrer antall elementer men man må telle opp. Hvis man antar at hashmappet har en fornuftig størrelse så blir det O(antall naboer)

Oppgave 13

For å finne inn-graden til en node A så må man gå gjennom alle de andre nodene og finne alle gangene node A fins i edges hashmap.

Sett inngrad=0

La noden det spørres om hete A

For hver node N i nodes:

> Hvis N=A, gå til neste
>
> Hvis N.edges inneholder A, øk inngrad med 1

Returner inngrad.

Kjøretid O(antall noder i grafen) siden å spørre et hashmap om det inneholder et element er O(1)

Oppgave 14

Enkel formulering:

Sett kostnaden til startnoden til 0 og alle andre kostnader til uendelig.

Gå gjennom alle nodene V ganger.

For hver node N for hver gang:

> Sjekk alle naboene til N om kostnad(N) + kostnad(kant fra N til node) < kostnad(node)
>
> Hvis ja. Sett kostnad(node) = kostnad(N) + kostnad(kant fra N til node)

Gå gjennom alle nodene en ekstra gang

For hver node

> Sjekk alle naboene til N om kostnad(N) + kostnad(kant fra N til node) < kostnad(node)
>
> Hvis ja er det en sykel i grafen og det eksisterer ingen kosteste vei
>
> Hvis nei, er algoritmen ferdig med gyldige korteste veier

Kjøretid O(V*E)

Mer effektiv men mer komplisert formulering

Sett kostnaden til startnoden til 0 og alle andre kostnader til uendelig.

Lag et kø Q

Lag en teller i alle noder som teller antall ganger den er lagt til og tatt ut av Q. Telleren starter på 0 for alle noder

Sett inn startnoden i Q. Øk dens teller til 1

Så lenge det er noder igjen i Q:

Ta ut første node N fra Q. Øk dens teller med 1

Hvis telleren til N > 2**antall noder* er det en sykel i grafen. Avbryt algoritmen og rapporter at det ikke eksisterer en gyldig korteste vei

For hver nabo Na av N:

Nykostnad = kostnad for N + kostnad for kanten fra N til Na

Hvis kostnaden for Na > nykostnad:

Sett kostnaden for Na = nykostnad

Hvis telleren til Na er et partall (Na er ikke i Q):

Sett Na inn i Q og øk telleren til Na med 1

Kjøretid: Har samme kjøretid som den enkle formuleringen i O-notasjon men kjører i praksis en del fortere siden den unngår en del unødvendig arbeid.