Eksamen 2019 – Ordinær- Løsningsforslag

Sensureringen av eksamensbesvarelser i DATS2300 har tatt utgangspunkt i den generelle kvalitative beskrivelsen for karakterene er som følger (Universitet og høyskolerådet, 2004):

A - fremragende - Fremragende prestasjon som klart utmerker seg. Kandidaten viser svært god vurderingsevne og stor grad av selvstendighet.

B - meget god - Meget god prestasjon. Kandidaten viser meget god vurderingsevne og selvstendighet.

C - god - Jevnt god prestasjon som er tilfredsstillende på de fleste områder. Kandidaten viser god vurderingsevne og selvstendighet på de viktigste områdene.

D - nokså god - En akseptabel prestasjon med noen vesentlige mangler. Kandidaten viser en viss grad av vurderingsevne og selvstendighet.

E - tilstrekkelig - Prestasjonen tilfredsstiller minimumskravene, men heller ikke mer. Kandidaten viser liten vurderingsevne og selvstendighet.

F - ikke bestått - Prestasjon som ikke tilfredsstiller de faglige minimumskravene. Kandidaten viser både manglende vurderingsevne og selvstendighet.

Eksamen besto av fem oppgaver som har blitt vektet likt. Ved sensurering har det blitt lagt vekt på hva kandidaten viser av forståelse av kursets pensum opp mot læringsutbyttet. Det vil si at det vektlegges hvordan kandidaten kommer frem til et svar ved å bruke pensum. Under følger en gjennomgang av oppgavene og mulig svar som vil gi god uttelling mot karakterbeskrivelsene over.

Pensum

Pensum i kurset er dekket av online-kompendiet til Ulf Uttersrud, https://www.cs.hioa.no/~ulfu/appolonius

Alle tema som har blitt tatt opp i ukeoppgaver, forelesninger, og obligatoriske oppgaver er en del av pensum.

Læringsutbytte

Etter å ha gjennomført dette emnet har studenten følgende læringsutbytte, definert i kunnskap, ferdigheter og generell kompetanse.

Kunnskap

Studenten kan:

- forklare oppbyggingen og hensikten med datastrukturer som tabeller, lister, stakker, køer av ulike typer, heaper, hashtabeller, trær av ulike typer, grafer og filer
- gjøre rede for virkemåten og effektiviteten til ulike varianter av algoritmer for opptelling, innlegging, søking, sletting, traversering, sortering, optimalisering og komprimering

Ferdigheter

Studenten kan:

- designe, implementere og anvende datastrukturer for ulike behov
- analysere, designe, implementere og anvende de algoritmene som trengs for å løse konkrete oppgaver
- bruke både egenutviklede og standardiserte algoritmer og datastrukturer til å løse sammensatte og kompliserte problemer

Generell kompetanse

Studenten kan:

- delta i diskusjoner og gi råd om hvilke datastrukturer og algoritmer det er mest hensiktsmessig å bruke i ulike situasjoner
- formidle viktigheten og nødvendigheten av å bruke gode strukturer og effektive algoritmer i programmeringsprosjekter

Oppgave 1: Generelle køer

I denne oppgaven handler om forskjellige typer køer

a) Beskriv kort hva de forskjellige følgende køene er:
   i. Kø: en first in first out kø (se FIFO-kø)
   ii. FIFO-kø: en kø hvor det du legger inn først er det som tas ut først
   iii. LIFO-kø: en kø hvor det du legger inn sist er det som tas ut først
   iv. Stack: en LIFO-kø
   v. Deque: En double ended queue. Kan fungere som både FIFO og LIFO-kø
   vi. Prioritetskø: En kø hvor elementet med høyest prioritet blir tatt ut først

b) Se på kildekoden i vedlegget. Beskriv kort hva kildekoden gjør
   Kildekoden bruker en Queue, Stack, og PriorityQueue for å legge inn og ta ut verdier. Først legges «ALFABET» inn i køen. Så tas fem verdier ut. Så legges «FISK» inn i køen. Så tas alle resterende verdier i køen ut.

c) Hva blir utskrift av kildekoden i vedlegget?
   Queue first: A, L, F, A, B
   Queue second: E, T, F, I, S, K
   Stack first: T, E, B, A, F
   Stack second: K, S, I, F, L, A
   Priority queue first: A, A, B, E, F
   Priority queue second: F, I, K, L, S, T

Oppgave 2: Quicksort

I denne oppgaven skal du sortere et sett med verdier med Quicksort. I denne oppgaven bruker vi arrayet

```
char[] values = {'B', 'C', 'K', 'A', 'F', 'L', 'T'};
```

a) Ta utgangspunkt arrayet:
   i. Forklar hva partisjonering er. Å «dele» et sett med verdier slik at alt som er større enn eller lik en gitt skilleverdi ligger til høyre for verdien, og alt mindre ligger til venstre.
   ii. Partisjoner arrayet over ved å bruke 'K' som skilleverdi.
   `{'B', 'C', 'A', 'F', 'K', 'L', 'T'};`
   iii. Hvilken indeks vil skilleverdien ligge på etter partisjonering? 4

b) Forklar hvordan quicksort fungerer
   i. Beskriv, med ord, hvordan quicksort fungerer. Quicksort bruker partisjonering som sorteringselement rekursivt. Først partisjoneres hele vektoren med en gitt skilleverdi, og rekursivt partisjoneres både høyre og venstre subliste. Partisjonering setter skilleverdien på riktig sortert plass.
   ii. Lag en tegning som stegvis viser hvordan quicksort sorterer arrayet over.

```
{'B', 'C', 'K', 'A', 'F', 'L', 'T'};
{'B', 'C', 'A', 'F'}, 'K', {'L', 'T'}; // skilleverdi k
{'B', 'A'}, 'C', {'F'}, 'K', 'L', {'T'}; // skilleverdi c, l
{'B', 'A'}, 'C', 'F', 'K', 'L', 'T'; // skilleverdi b
{'A'}, 'B', 'C', 'F', 'K', 'L', 'T'; // ferdig sortert
```

c) Algoritmeanalyse av quicksort

i. Vil du bruke iterasjon eller rekursjon for å implementere quicksort? Rekursjon

ii. Hva slags kompleksitet har quicksort i gjennomsnittstilfellet? Forklar kort hvordan du kommer frem til svaret ditt. Quicksort har kompleksiten O(n log n) i gjennomsnittstilfellet. Dette kan vi komme frem til ved å se på basisoperasjonene vi utfører. For hver iterasjon (nivå) av quicksort utfører vi partisjonering som er en O(n) operasjon (må røre alle n elementer). Antall nivåer vi må gjennom for å ha partisjonert alle elementer er O(log2 n): i nivå 0 sorterer vi ett tall, så 2 tall i nivå 1, så 4 tall i nivå 3, osv. slik at vi har sortert 2^n tall etter n nivåer. Når vi da gjør O(n) operasjoner O(log2 n) nivåer får vi O(n log2 n) eller O(n log n).

Oppgave 3: Minimumsheap

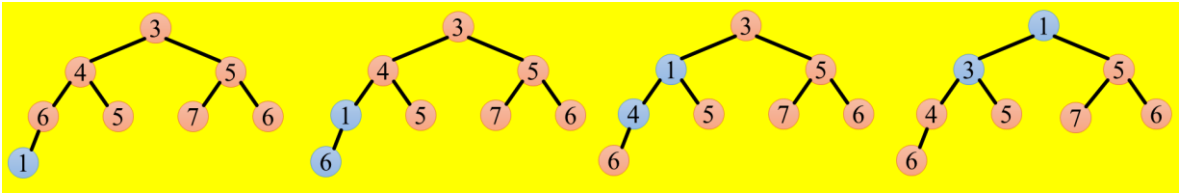
I denne oppgaven skal vi bruke en minimumsheap og se hvordan den kan brukes til sortering

a) Hva er en minimumsheap, og hvilke krav stilles for at det skal kunne kalles en minimumsheap? En effektiv datastruktur for en dynamisk prioritetskø. Er et komplett binærtre hvor alle foreledrenoder er mindre eller lik sine to barn.

b) Start med minimumsheapen i vedlegget

i. Forklar hvordan man legger inn ett tall i en minimumsheap. Plasserer først tallet nederst og til venstre i det komplette binærtreet. Bytter så rekursivt med foreldrenoden så lenge forelderens verdi er større enn den insatte verdien.

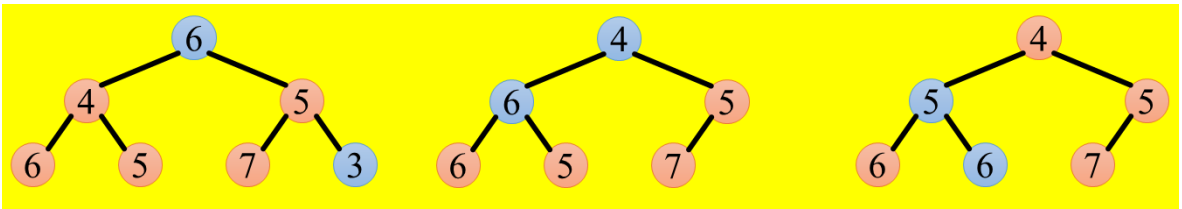
ii. Legg inn tallet 1 og lag en tegning av svaret ditt for hvert steg i algoritmen



c) Start med minimumsheapen i vedlegget

i. Forklar hvordan man tar ut ett tall av en minimumsheap. Man bytter plass med rot og siste element. Så lar man rotnoden rekursivt bytte verdi med sitt minste barn så lenge barnet har mindre verdi.

ii. Ta ut ett tall og lag en tegning av svaret ditt. Tegn hvert steg av algoritmen



d) Start med minimumsheapen i vedlegget

i. Forklar hvordan en minimumsheap kan lagres i et array. Man bruker node-id-1 som posisjon i arrayet. Nodene er da sorter i nivå orden i arrayet

ii. Skriv opp minimumsheapen i vedlegget som et array. {3, 4, 5, 6, 5, 7, 6}

Oppgave 4: Dobbelt lenket liste

I denne oppgaven skal vi operere med en dobbelt lenket liste (se vedlegget).
I denne oppgaven er følgende viktig:

- Kildekoden skal være kort, oversiktlig, og lett leselig.
- Skriv funksjonen så effektiv som mulig.
- Du trenger ikke ta hensyn til spesialtilfeller og indekskontroll, dvs du kan anta at noden du skal fjerne finnes i listen, og at du ikke skal fjerne første eller siste node.

a) Skriv innholdet i funksjonen void remove(int index) der det er markert.
Funksjonen skal fjerne noden på plass index.

```
void remove(int index) {
    if (index == 0) {
        removeFirst();
    }
    else if (index == size-1) {
        removeLast();
    }
    else {
        Node q = null;

        //Søk fra starten hvis index før halvveis
        if (index < size / 2) {
            q = head;
            for (int i = 0; i < index; ++i) {
                q = q.next;
            }
        }
        //Søk fra slutten hvis index over halvveis
        else {
            q = tail;
            for (int i = size - 1; i > index; --i) {
                q = q.prev;
            }
        }

        //Finn de to nabonodene
        Node p = q.prev;
        Node r = q.next;

        //Sett pekere til naboer
        p.next = r;
        r.prev = p;

        //Oppdater størrelse
        --size;
    }
}
```

b) Skriv innholdet i funksjonen void remove(char value).
Funksjonen skal fjerne den første noden som har verdi «value».

```
void remove(char value) {
    Node q = head;
```

```
    //Søk fra starten, vi skal fjerne første "value"
    for (int i = 0; i < size; ++i) {
        if (q.value == value) {
            break;
        }
        else {
            q = q.next;
        }
    }

    //Finn nabonoder
    Node p = q.prev;
    Node r = q.next;

    //Sett pekere til naboer
    p.next = r;
    r.prev = p;

    //Oppdater størrelse
    --size;
}
```

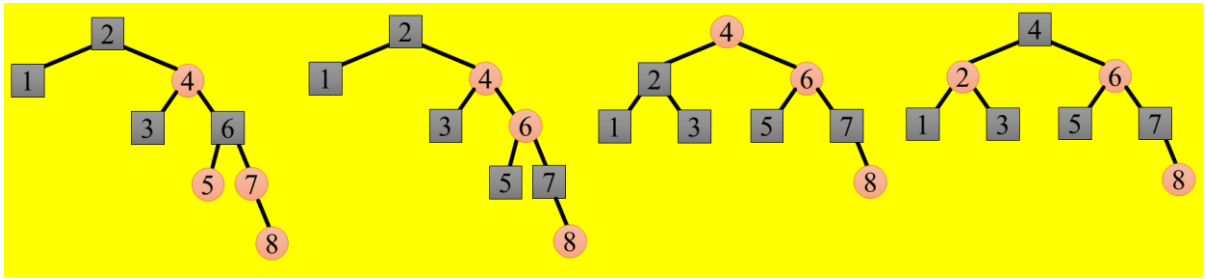
Oppgave 5: Binært søketre

I denne oppgaven skal du lage et balansert binært søketre

a) Hva er et balansert binært søketre? Et søketre som automatisk minsker høyden til treet ved innlegging av nye noder. Det unngår da ekstremt dype trær med få noder.

b) Ta utgangspunkt i det rød-sorte treet i vedlegget, og legg inn tallet 8.

   a. Tegn hvert skritt i algoritmen for rød-sorte trær når du legger inn tallet, og beskriv med ord hva du gjør. 1) Legger inn 8, rød. 2) Onkel er rød, fargeskifte. 3 og 4) Oldeforelder til 8 er rød og besteforelder er rød => venstre rotasjon med fargeskifte.



c) Tegn B-treet av orden 4 som tilsvarer treet i vedlegget.