Eksamen 2019 – Kontinuasjon-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, og at man kan få god uttelling dersom man viser at man behersker den delen av pensum som oppgaven spør etter selv om man har avvik fra dette løsningsforslaget. 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: Rekursjon og binærtrær

I denne oppgaven handler om rekursjon og traversering av binærtrær

1. Forklar hva en rekursiv funksjon er.
    1. Hvilke til krav stiller vi så en rekursiv funksjon skal virke etter hensikten?
       Funksjonen må kalle seg selv, forenkle parametere, og ende i et basistilfelle.
    2. Hva vil skje dersom disse kravene ikke fylles av funksjonen?
       Uendelig rekursjon (stack overflow), en ikke rekursiv funksjon (kaller ikke seg selv).
2. Skriv ut verdien på nodene i treet i vedlegget ved å traversere det
    1. Preorden,
       FACBDHGQ
    2. Inorden,
       ABCDFGHQ
    3. Postorden, og
       BDCAGQHF
    4. Nivå-orden.
       FAHCGQBD
3. Hvordan regner man ut ID'en til hver node i binærtreet?
    1. Skriv opp formelen for å regne ut ID til hver node.
       root = 1
       left_child = parent*2
       right_child = parent*2+1
       id er da nodenummeret beregnet her minus en.

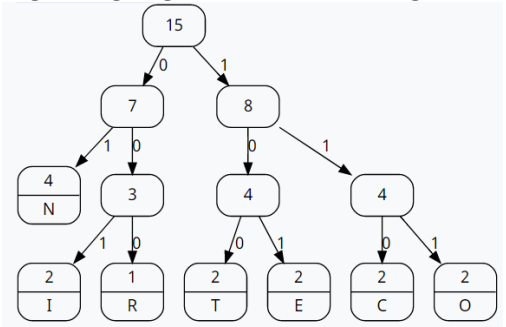
2. Skriv opp ID'en til hver node.
   F: 0, A:1, H:2, C: 4, G: 5, Q: 6, B: 9, D: 10
3. Hva brukes denne ID'en til?
   Lagring i array i heap-sort f.eks.

4. Forklar hvordan man traverserer treet
   1. Preorden ved å bruke rekursjon eller iterasjon, og
      Pseudokode rekursjon:
      ```
      void traverse (Node n) {
       if (n == null) { return }
       traverse(n.left)
       print(n.value)
       traverse(n.right)
      }
      ```
   2. Nivå-orden ved å bruke rekursjon eller iterasjon
      Pseudokode iterasjon:
      ```
      void traverse(Node n) {
       Queue q;
       q.add_back(n)
       while(!q.empty()) {
        Node p = q.remove_front()
        if (p == null) { continue }
        print(p.value)
        q.push_back(p.left)
        q.push_back(p.right)
       }
      }
      ```

Oppgave 2: Huffmantrær

I denne oppgaven skal du bruke et Huffmantre til å kode ordet «INTERCONNECTION».

1. Lag Huffmantreet basert på ordet.
   1. Lag en tabell med frekvenser til hver bokstav,
      I: 2, N: 4, T: 2, E:2, R: 1, C: 2, O: 2
   2. Lag en tegning av Huffmantreet, og



   3. Skriv opp Huffmankoden for hver bokstav.
      N: 01, I: 001, R: 000, T: 100, E: 101, C: 110, O: 111
2. Bruk så Huffmankodene til å komprimere ordet.
   1. Skriv opp den kodede binære meldingen.
      001 01 100 101 000 110 111 01 01 101 110 100 001 111 01

2. Hvor mange bit bruker den opprinnelige (ukodede) meldingen?
   15 * 8 = 120 bit
3. Hvor mange bit bruker du i den kodede meldingen?
   4*2 + 2*3 + 1*3 + 2*3 + 2*3 + 2*3 + 2*3 = 41

Oppgave 3: Quicksort

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

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

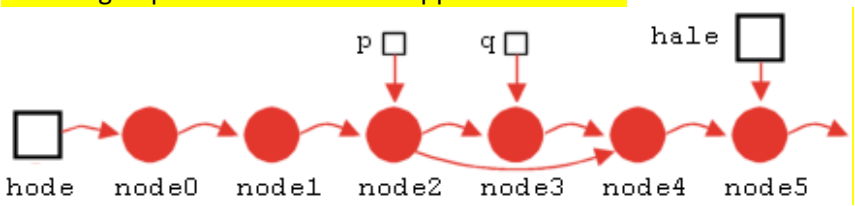
1. Forklar hvordan quicksort fungerer
   1. Forklar hva partisjonering er.
      Partisjonering deler et sett med verdier i alt som er større enn eller lik og mindre enn en skilleverdi (pivot)
   2. Forklar hvordan Quicksort bruker partisjonering til å sortere.
      Quicksort velger en skilleverdi blant verdiene i arrayet, og partisjonerer arrayet basert på denne skilleverdien. Etter partisjonering er skilleverdien på korrekt sortert plass. Quicksort gjør dette så rekursivt for høyre og venstre subliste.
   3. Lag en tegning som stegvis viser hvordan quicksort sorterer arrayet over. Når du skal partisjonere skal du bruke den midterste verdien.
      {'B', 'K', 'C', **'A'**, 'L', 'F', 'T', 'Q'} //A pivot
      {'A'**,** 'B', 'K', 'C', **'L'**, 'F', 'T', 'Q'} //L pivot, A sortert
      {'A'**,** 'B', **'K'**, 'C', 'F', 'L', **'T'**, 'Q'} //K og T pivot, AL sortert
      {'A'**,** 'B', **'C'**, 'F', 'K', 'L', **'Q'**, 'T'} //C og Q pivot, ALKT sortert
      {'A'**,** **'B'**, 'C', **'F'**, 'K', 'L', 'Q', 'T'} //B og F pivot, ACKLQT sortert
      {'A'**,** 'B', 'C', 'F', 'K', 'L', 'Q', 'T'} //Alle sortert
2. Algoritmeanalyse av quicksort
   1. Forklar hva beste, gjennomsnittlig og verste tilfelle betyr når man snakker om kompleksiteten til en algoritme.
      Beste tilfelle er når dataene algoritmen opererer på er på en slik måte at algoritmen bruker minst mulig tid. Tilsvarenede for verste tilfelle. Gjennomsnittstilfelle er en slags forventet tidsbruk til en algoritme, og kan beregnes ved å se på tiden det tar for alle permutasjoner av data til algoritmen, og ta gjennomsnittstiden.
   2. Forklar hvorfor Quicksort i gjennomsnittstilfellet har kompleksiteten O(n log(n)) og i verste tilfellet har kompleksiteten $O(n^2)$
      Dersom man ser på quicksort som et binærtre, hvor barna til hver node tilsvarer høyre og venstre subliste etter partisjonering vil man se at for en liste med n verdier vil dette binærtreet ha log(n) nivåer. For hvert nivå kan man tenke seg da at man må flytte på alle tallene i partisjonerings-biten, og man ender da med O(n log(n)).
      I verste tilfellet kan man se for seg at dette binærtreet er ekstremt høyre eller venstre-tungt. Da ender man opp med et binærtre med dybde n, og kompleksiteten blir da O(n^2)

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.

1. Lag en tegning som viser hvordan du fjerner en node fra en dobbelt lenket liste og beskriv med ord hvordan du går frem
   I figuren under vises det hvordan pekeren fra p «hopper over» q for å fjerne q fra den enkelt lenkede listen. For den dobbelt lenket liste gjør man tilsvarende med bakover-pekerene, slik at «forrige»-pekeren til node 4 «hopper over node3.



2. Kopier funksjonen void remove(int index), og skriv innholdet i funksjonen 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;
        }
    }
```

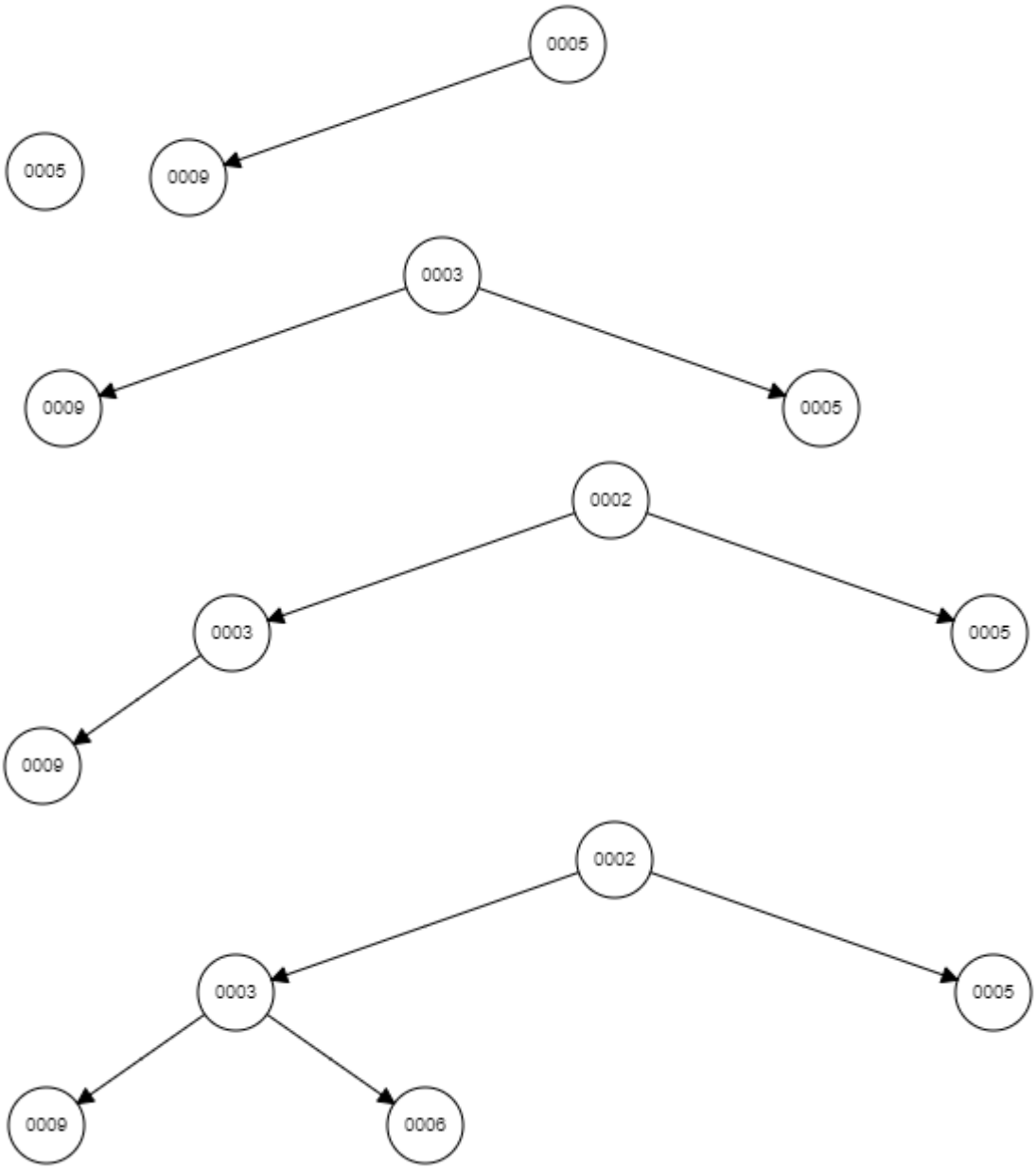
3. Kopier funksjonen void remove(char value), og skriv innholdet i funksjonen der det er markert. Funksjonen skal fjerne den første noden som har verdi «value».

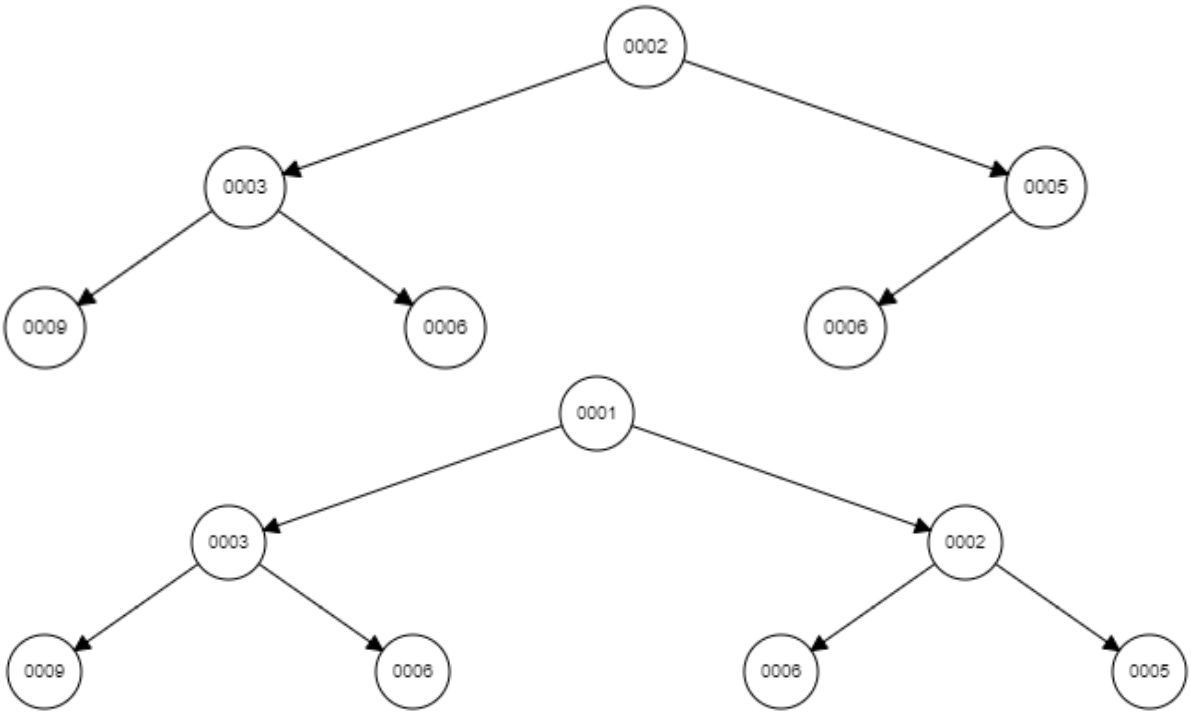
```
void remove(char value) {
    Node q = head;

    //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: Minimumsheap

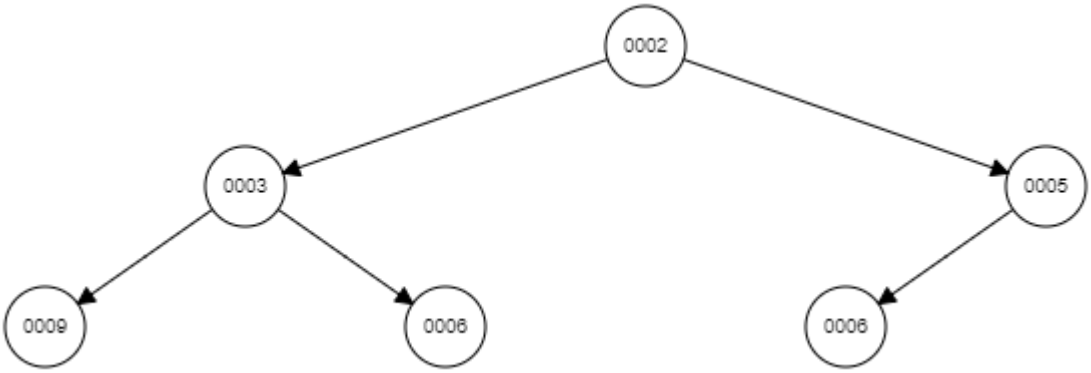
I denne oppgaven skal vi bruke en minimumsheap og se hvordan den kan brukes til sortering

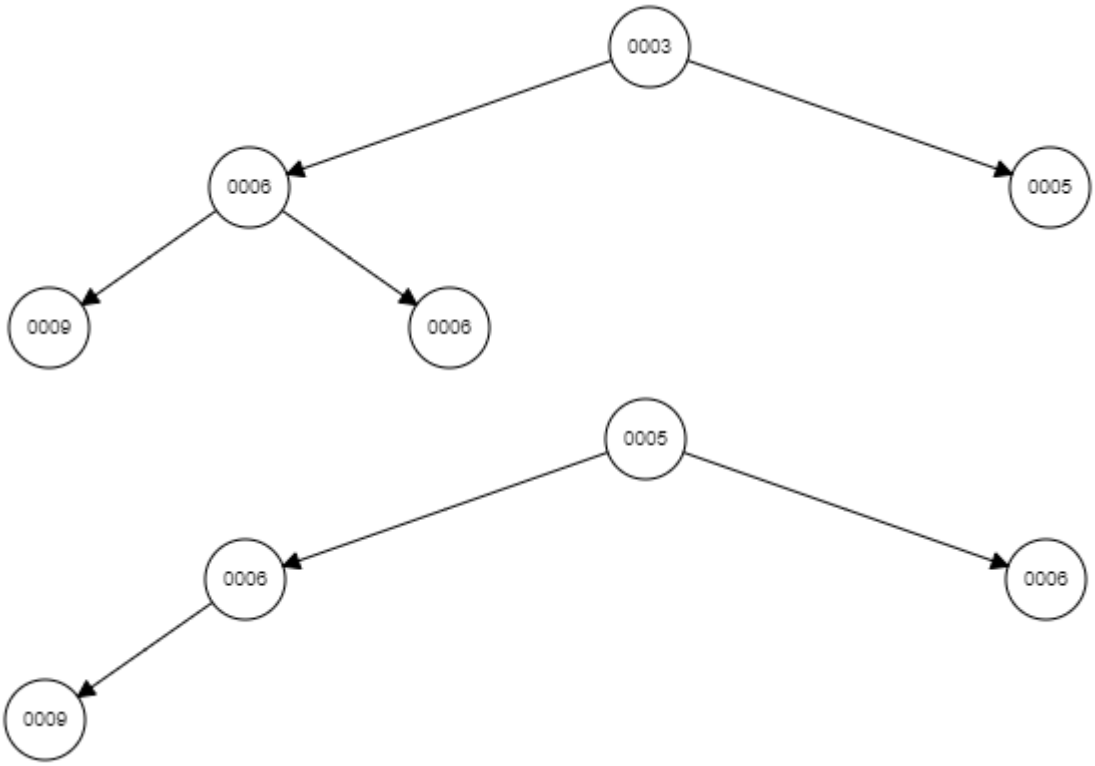
1. Hva er en minimumsheap, og hvilke krav stilles for at det skal kunne kalles en minimumsheap?
   En minimumsheap er et komplett binærtre hvor barnenoder er større eller lik foreldrenoden
2. Start med en tom minimumsheap. Legg tallene 5, 9, 3, 2, 6, 6, 1 og tegn heapen for hvert tall du legger inn.


0005
0005
0009
0003
0009
0005
0002
0003
0005
0009
0002
0003
0005
0009
0006



3. Vi skal nå ta ut **tre** tall fra heapen. Ta ut ett og ett tall og tegn heapen for hvert tall du tar ut.





4. Forklar hvordan en minimumsheap kan brukes til sortering uten å bruke ekstra lagringsplass. Bruke node-id'ene fra oppgave 1 til å lagre dem i et array. Bytter da bare plass internt i arrayet.