UNIVERSITETET I BERGEN

KANDIDAT

352

PRØVE

INF102 0 Algoritmer, datastrukturer og programmering

Emnekode	INF102
Vurderingsform	Skriftlig eksamen
Starttid	22.11.2022 14:00
Sluttid	22.11.2022 17:00
Sensurfrist	--
PDF opprettet	03.05.2024 11:01

Informasjon

Oppgave	Tittel	Oppgavetype
ℹ	Egenerklæring	Informasjon eller ressurser
ℹ	Info om eksamen	Informasjon eller ressurser
ℹ	Kontakt under eksamen	Informasjon eller ressurser

Oppgaver

Oppgave	Tittel	Oppgavetype
1	Simplify bigO	Flervalg
2	Search order	Paring
3	Climate Emergency responce	Tekstfelt
4	SplitList Runtime	Flervalg
5	SplitListImprove	Programmering
6	SolarLane	Programmering

Semesteroppgaver

Oppgave	Tittel	Oppgavetype
7	Poeng fra Semesteroppgavene H22	Tekstfelt

1 Simplify bigO

I denne oppgaven blir du gitt flere funksjoner. Du skal velge det bigO uttrykket som best beskriver hver funksjon, altså det minste og enkleste bigO uttrykket som inneholder den gitte funksjonen.
For hver oppgave får 2 poeng for rett svar og 0 poeng for blank/feil svar.

a) $n^2 + n \log n$

Velg ett alternativ:

- $O(n^2 + n \log n)$
- **$O(n^2)$**
- $O(n^2 \log n)$
- $O(n \log n)$

b) $m \log n + n \log m$
Her kan du anta at $n < m < n^2$

Velg ett alternativ

- $O(mn)$
- $O(n \log m)$
- **$O(m \log n)$**
- $O(m \log n + n \log m)$

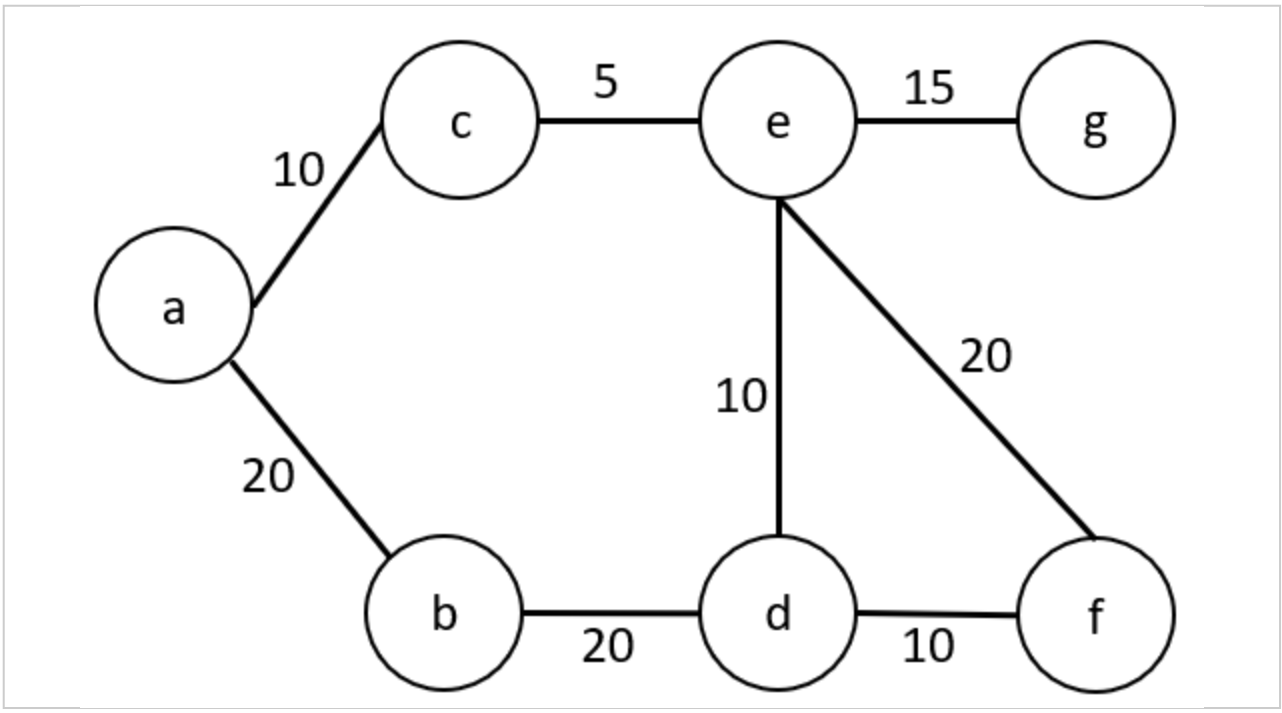
c) $n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} \ldots + 4 + 2 + 1$

Velg ett alternativ

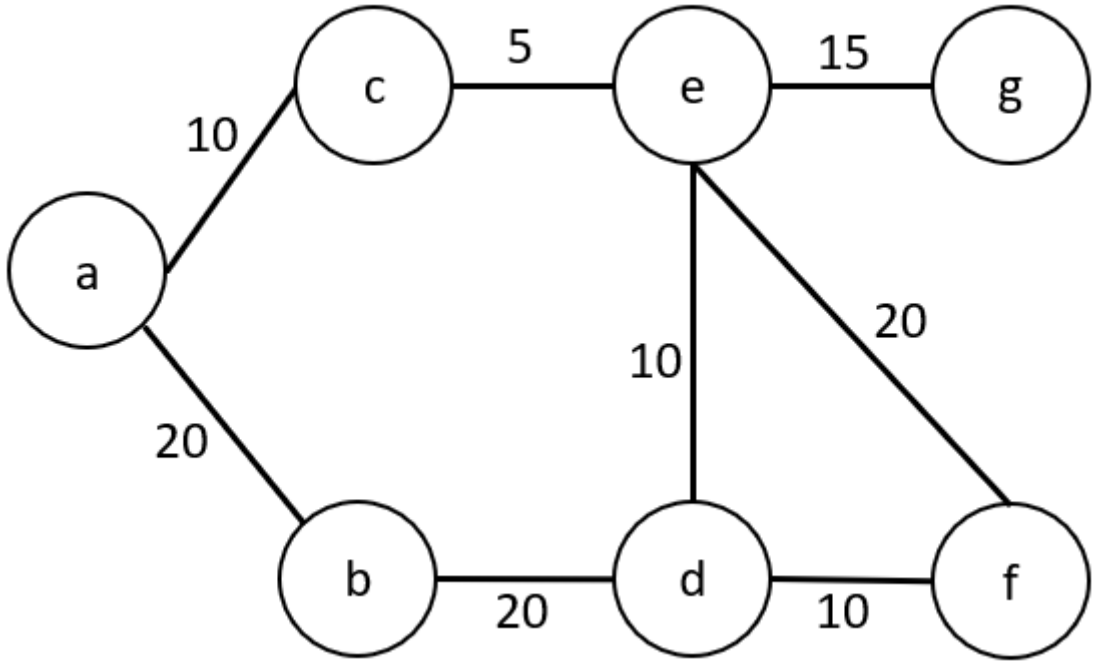
- $O(n^2)$
- **$O(n)$**
- $O(n(\log n)^2)$
- $O(n \log n)$

Maks poeng: 6

2 Search order



I denne oppgaven er du gitt en graf.
Det finnes mange algoritmer som søker igjennom grafer, en type av slike algoritmer har det til felles at de kan implementeres med 2 datastrukturer, en Collection **toSearch** og et Set **visited**.
Man velger en node fra **toSearch** (etter forskjelllige regler), hvis denne ikke finnes i **visited** så legger vi den til i **visited** og alle naboene dens legges til i **toSearch**. I denne oppgaven skal vi se på hvilken rekkefølge nodene legges til i **visited**.



For hver rad skal du velge en ordning. På hver rad får du 2 poeng for rett svar, 0 poeng for blank og -0.5 poeng for feil svar.

Finn de som passer sammen:

	a,c,e,d,f,g,b	a,c,e,b,d,g,f	a,b,c,d,e,f,g	a,b,c,d,f,e,g	a,b,d,e,c,f,g
BFS	○	○	◉	○	○
Prim	◉	○	○	○	○
DFS	○	○	○	○	◉
Dijkstra	○	◉	○	○	○

Maks poeng: 8

3 Climate Emergency responce

Forskning har vist at rask responstid er viktig for å redde liv. Brannstasjoner, sykehus og politistasjoner er bygget på strategiske lokasjoner. Politikerne har fått for seg at vi trenger en nødetat for klima for å slå ned på de som forsøpler og forurenser.

Nå ber de om din hjelp til å velge lokasjon for denne nye nødetaten slik at gjennomsnittlig responstid blir minst mulig. Nødetaten for klima skal selvsagt utstyres med sykler slik at de raskt kommer frem. Følgende data er blitt samlet inn:

- En liste av **n** punkter i byen der forurensning kan skje
- For hvert punkt er det estimert hvor ofte utrykning forventes å skje (ganger/per år)
- For hver gate er det målt hvor lang tid det tar å sykle

Vi antar at alle mulige gatekryss i byen er en potensiell lokasjon for Nødetat for Klima.

Oppgave a) teller 4 poeng, b) 8 poeng og c) 8 poeng, totalt er 20 poeng mulig.

a) Hvilken datastruktur passer for å representere denne dataen?
Her må du også kunne lagre mulige potensielle lokasjoner for senteret.

Skriv ditt svar for a) her

jeg ville brukt et Minimum spanning tree for å lagre vertex og edges.
jeg ville brukt prims algo for å lage MST
jeg ville lagret estimatet for hvor ofte utrykningen forventes i et hashmap hvor vertex er key og estimatet er value.

prims:
jeg bruker et priorityQueue for å holde styr på edges som jeg kan velge mellom
jeg bruker et hashset til å holde styr på vertexes jeg har sett.
jeg bruker en linkedlist til å holde styr på stien jeg har valgt

jeg starter med en random node.

for en gitt node:
- jeg legger til noden i hashsettet mitt.
- jeg legger til alle edges i priorityQueue min
- jeg velger en ny edge fra priorityQueue, så lenge den vertexen på andre siden av edgen ikke er i hashsettet så legger jeg edgen til i linkedlisten min.
- jeg setter så vertexen på andre siden av edgen som den nye vertexen å se på.

b) Beskriv en algoritme for å finne den lokasjonen som gir minst gjennomsnittlig responstid.
Oppgi kjøretid for din algoritme.

Skriv ditt svar for b) her

jeg ville gitt hver node en verdi (gjennomsnittlig responstid) som blir kalkulert på denne måten:

-bruk dijkstra på den gitte noden til å finne shortest path til alle andre noder.
-jeg summer lengden til alle andre noder og deler på antall noder for å få gjennomsnitts responstid, og setter verdien til noden lik dette.

jeg velger til slutt den noden med den minste verdi

---

dijkstra:

-start med alle noder i et unvisited set.
-alle noder starter med verdi uendelig bortsett fra start noden som har verdi 0
-for den gitte noden kalkuler lengden til naboene ved å plusse lengden fra starten til den gitte noden med lengden fra den gitte noden til naboen. hvis naboen har en verdi høyere en den vi nettop kalkulerte, så oppdater verdien til naboen.
-når alle naboer har blitt sjekket, så fjern den gitte noden fra unvisited settet.
-
* gikk tom for tid

c) Politikerne ønsker å vite hvor mye lavere responstid det blir om de velger 2 lokasjoner der nødetat for klima holder til. Beskriv en algoritme som finner de to lokasjonene slik at gjennomsnittlig responstid blir minst mulig. Oppgi kjøretid for din algoritme.

Skriv ditt svar for c) her

jeg går gjennom hver edge i treet og ser hvor store de to subtreene blir hvis jeg kutter treet på den gitte edgen. jeg velger så til slutt den edgen som gjør at de to subtreene har like mange vertexer (eller så nerme som mulig hvis det er odd antall vertexer)

jeg kjører så samme algo som i oppgave b på hver av de to subtreene for å finne de to vertexene som har minst responstid i hvert sitt subtree.

Maks poeng: 20

4 SplitList Runtime

I denne oppgaven skal dere bli kjent med en datastruktur som er kalt SplitList.
Dere skal analysere kjøretiden på en del av metodene, men det kan være lurt å samtidig prøve å forstå hva metodene gjør for dere skal jobbe videre med SplitList i neste oppgave.
SplitList

n er antall elementer i listen som blir gitt som input til konstruktøren
altså har vi at **n** = low.size()+hi.size()+1. Kjøretiden kan variere avhengig av hvor langt i iterasjonen vi er kommet, da skal dere anta at vi er på det verste mulige steget i iterasjonen.

På denne oppgaven vil du for hver av de 6 deloppgavene få:
+1 poeng for rett svar, 0 poeng for ubesvart og -1 poeng for feil svar.

a) Hva er kjøretiden til Konstruktøren?
Velg ett alternativ:

- O(1)
- O(n)

b) Hva er kjøretiden til removeMin()?
Velg ett alternativ

- O(n)
- O(1)

c) Hva er kjøretiden til sumBelow(), sumAbove() og sum()?
Velg ett alternativ

- O(1)
- O(n)

d) Hva er kjøretiden til numBelow() og numAbove()?
Velg ett alternativ

- O(n)
- O(1)

e) Hva er kjøretiden til next()?

Velg ett alternativ

- O(1)
- O(n)

f) Hva er kjøretiden til removeSmallest()?

Velg ett alternativ

- O(n)
- O(1)

Maks poeng: 6

5 SplitListImprove

I forrige oppgave ble dere kjent med datastrukturen SplitList.
Klarer du å forbedre kjøretiden på noen av metodene i SplitList?
SplitList

Når du skal vurdere hva som er bedre kan du anta at konstruktøren skal bli kallt 1 gang mens alle andre metoder skal blir kallt O(n) ganger.

Du skal ikke endre metodesignaturen eller funksjonalitet på de metodene som er public.
Du kan endre på private metoder, felt variable og alt av kode.

Følgende fil er en JUnit test du kan bruke til å sjekke at du ikke har endret funksjonaliteten på SplitList. Alle testene skal passere når du har endret koden.
SplitListTest

Skriv ditt svar her

```java
package INF102.examH22.solar;

import java.util.*;

/**
 * WHAT I HAVE IMPROVED:
 *  sumAbove() and sumBelow() now run at O(1)
 *  removeMin() now runs at O(logn) becuase i use a priorityqueue
 *  next() now runs at O(logn)
 *  removeSmallest() now runs at O(logn)
 *
 * WHAT HAS BECOME SLOWER:
 *  the constructor now runs at O(nlogn) instead of O(n)
 */
public class SplitList implements Iterator<Integer> {

    private PriorityQueue<Integer> low; //the low numbers
    private PriorityQueue<Integer> hi; //the high numbers
    private Integer current;
    private int sumAbove;
    private int sumBelow;

    public SplitList(List<Integer> list) {
        low = new PriorityQueue<>();
        hi = new PriorityQueue<>(list); // O(nlogn)
        current = removeMin(hi);
        sumAbove = sum(hi);
    }

    private int removeMin(PriorityQueue<Integer> c) {
        int min = c.remove();
        sumAbove -= min;
        return min; // O(log n)
    }

    public int sumBelow() {
        return sumBelow;
    }


    public int sumAbove() {
        return sumAbove;
```

```
43      }
44
45
46      private int sum(Collection<Integer> coll) {
47          int sum = 0;
48          for(int val : coll)
49              sum += val;
50          return sum;
51      }
52
53
54      public int numBelow() {
55          return low.size();
56      }
57
58      //returns the number of elements in hi
59      public int numAbove() {
60          return hi.size();
61      }
62
63      //returns the current element
64      public int getCurrent() {
65          return current;
66      }
67
68      @Override
69      public boolean hasNext() {
70          return !hi.isEmpty();
71      }
72
73      //next() changes current to the smallest element in hi and returns the new curr
74      @Override
75      public Integer next() {
76          low.add(current); // log(n)
77          sumBelow += current;
78          current = removeMin(hi); // log(n)
79          return current;
80      }
81
82
83      public void add(int num) {
84          if(num>=current) {
85              hi.add(current);
86              sumAbove += num;
87          }
88          else {
89              low.add(current);
90              sumBelow += num;
91          }
92      }
93
94
95      public int removeSmallest() {
96          if(low.isEmpty()) {
97              int smallest = current;
98              current = removeMin(hi); // log n
99              return smallest;
100         }
101         int min = low.remove(); // log n
102         sumBelow -= min;
103         return min;
104     }
105 }
106
```

Maks poeng: 10

6 SolarLane

I denne oppgaven skal vi jobbe med en enkel versjon av problemet i semesteroppgave 1.
Langs en lang rett vei er det installert mange solsellepaneler. Disse trenger vedlikehold/fiksing som skal gjøres av roboter, det trengs en robot for å fikse et solcellepanel.
Disse solsellepanelene er ganske driftssikre så vi antar at det aldri vil skje at 2 solsellepaneler trenger reparasjon samtidig.
Når et solsellepanel trenger reparasjon får nærmeste robot beskjed og å dra for å fikse solcellepanelet, problemet med robotene er at de er ganske trege så det er viktig at roboten står på en strategisk posisjon og venter på neste jobb.

Du er gitt listen av tidligere jobber (alle tidligere jobber er beskrevet med en x koordinat som sier avstanden fra starten av veien). Disse jobbene gir deg statistikk som du skal bruke til å estimere optimal posisjon for neste jobb. Anta at en hver x fra tidligere jobber har like stor sannsynlighet for å bli x posisjon for neste jobb. Det vil si at du skal plassere roboten slik at gjennomsnittlig avstand til de tidligere jobbene blir minst mulig (vi antar at dette også er beste posisjon for fremtidige jobber.

Det er flere måter å løse denne på, du kan bruke kode fra labber eller semesteroppgavene.
Hint: Det kan også være mulig å bruke koden fra SplitList.

Du skal implementere interfacet ISolarLane.java
ISolarLane
Du kan bruke følgende jUnit test til å sjekke at din implementasjon virker:
SolarLaneTest
Skriv ditt svar her

```java
package INF102.examH22.solar;

import java.util.Collections;
import java.util.List;

public class SolarLane implements ISolarLane {
    SplitList splitList;
    @Override
    public Integer place1Robot(List<Integer> jobs) {
        // runtime: nlogn + n/2 * logn + 1   = nlogn + nlogn + 1       = O(nlogn)

        splitList = new SplitList(jobs); // nlogn
        while (splitList.numAbove() > splitList.numBelow()){ // n/2 = n
            splitList.next(); // logn
        }
        return splitList.getCurrent(); // 1
    }

    @Override
    public Pair<Integer> place2Robots(List<Integer> jobs) {
        // runtime: nlogn + n + n + n/2 * logn + 1 + 1      = 2nlogn + 2n    = nlogn

        splitList = new SplitList(jobs); // nlogn
        int smallest = Collections.min(jobs); // n
        int largest = Collections.max(jobs); // n
        while (splitList.getCurrent() < ((smallest + largest) / 2)) { // n/2 = n
            splitList.next(); // logn
        }
        Integer one = splitList.sumBelow() / splitList.numBelow(); // 1
        Integer two = (splitList.sumAbove() + splitList.getCurrent()) / (splitList.r
        return new Pair<>(one, two);
    }
}
```

```
33  }
34
```

Maks poeng: 20

7 Poeng fra Semesteroppgavene H22

Denne oppgaven trenger du ikke gjøre noe på. Her får du poeng du for det du har gjort på semesteroppgavene.

Du er nå ferdig med eksamen :-)
God Jul!

Vennlig hilsen
Martin

Skriv en hyggelig melding til foreleser ;-)

god jul

Maks poeng: 30