DATS2300 / ITPE2300

Algoritmer og Datastrukturer Høst 2021

Oblig 3

Frist: Onsdag 3. november 2021 klokken 12:00

I denne obligen skal dere implementere et binært søketre med forskjellig funksjonalitet. Oppgaven skal løses individuelt og arbeidet skal dokumenteres ved bruk av git (se krav under). Det er lov å samarbeide om hvordan man kommer frem til en løsning, men all kildekode, readme, etc. må lages individuelt.

Ta utgangspunkt i utlevert kildekode, og fullfør oppgavene i SBinTre.java. Skriv i tillegg navn og studentnummer i filen som heter «README.md», samt beskrivelse av obligen. Beskriv (veldig kort) hvordan hver deloppgave er løst der.

Det er et krav at du bruker github for å lagre koden og dokumentere ditt arbeid. Oppgavene leveres via github classroom (se lenke under).

Invitasjon til prosjekt på Github:
https://classroom.github.com/a/e6FcFqFp

Krav til innlevering (hvis dette ikke oppfylles blir den ikke godkjent):

1. Git er brukt for å dokumentere arbeid med obligen. Du må ha
    a. Minst to commits per oppgave spredt over tid (ikke lov å committe alt rett før innleveringsfristen!)
    b. Beskrivende commit-meldinger, f.eks.
        i. "Laget første prototype med kildekodekommentarer over hvordan jeg har tenkt til å løse oppgave 1."
        ii. "Implementerte oppgave 1 slik at den passerer test 1a og 1b. Test 1c feiler fortsatt"
        iii. "Implementerte oppgave 1 slik at den passerer test 1c"
2. Alle filene ligger som levert ut - ingen av filene skal flyttes på.
3. Beskrivelse av hvordan oppgaven er løst (4-8 linjer/setninger per oppgave) står i Readme.md.
4. Ingen main-metode eller debug-utskrifter (system.out.println) når den leveres inn.
5. Warnings er enten beskrevet i readme.md eller fjernet.
6. For å bestå obligen må fem av de seks testene passere (du kan ha feil i en oppgave).

Hint: Oppgavene er veldig mye lettere å løse om man lager tegning på papir før man begynner implementere.

Hint: Bruk veldig gjerne discord og gruppene fra de forrige obligene til å diskutere løsninger på oppgavene og hjelp hverandre. Bruk penn og papir og del gjerne skisser. **MEN: all kode og readme mm. som leveres inn må du skrive selv. Ikke fall for fristelsen å kopiere noen andre, men bruk lab-timene til hjelp.**

SBinTre er et binært søketre av samme type som beskrevet i *Delkapittel 5.2*. Men SBinTre har litt mer struktur. En node har i tillegg til referanser til venstre og høyre barn, en referanse til dens forelder. I skjelettet er en del metoder ferdig kodet og de andre kaster en UnsupportedOperationException.

Obs: De metodene i SBinTre som skal kodes, kan kodes på mange forskjellige måter. I flere av oppgavene blir det bedt om at en metode skal kodes på en bestemt måte. Det er ikke fordi det nødvendigvis er den beste måten. Men poenget er å lære kodeteknikk, dvs. den teknikken som det bes om.

Oppgave 1

En Node i SBinTre har referanser til venstre barn, høyre barn, samt nodens forelder. Forelder må få riktig verdi ved hver innlegging, men forelder skal være null i rotnoden. Lag metoden *public boolean leggInn(T verdi)*. Der kan du kopiere *Programkode 5.2.3 a)*, men i tillegg må du gjøre de endringene som trengs for at referansen *forelder* får korrekt verdi i hver node. Teknikken med en forelder-referanse brukes f.eks. i klassen TreeSet i java.util. Sjekk at følgende kode er feilfri (ikke kaster noen unntak):

```
Integer[] a = {4,7,2,9,5,10,8,1,3,6};
SBinTre<Integer> tre = new SBinTre<>(Comparator.naturalOrder());
for (int verdi : a) {tre.leggInn(verdi); }
System.out.println(tre.antall());  // Utskrift: 10
```

Oppgave 2

Metodene *inneholder()*, *antall()* og *tom()* er ferdig kodet. Den første avgjør om en verdi ligger i treet eller ikke. De to andre fungerer på vanlig måte. Lag kode for metoden *public int antall(T verdi)*. Den skal returnere antall forekomster av verdi i treet. Det er tillatt med duplikater og det betyr at en verdi kan forekomme flere ganger. Hvis verdi ikke er i treet (*null* er ikke i treet), skal metoden returnere 0. Test koden din ved å lage trær der du legger inn flere like verdier. Sjekk at metoden din da gir rett svar. Her er ett eksempel:

```
Integer[] a = {4,7,2,9,4,10,8,7,4,6};
SBinTre<Integer> tre = new SBinTre<>(Comparator.naturalOrder());
for (int verdi : a) { tre.leggInn(verdi); }

System.out.println(tre.antall());      // Utskrift: 10
System.out.println(tre.antall(5));     // Utskrift: 0
System.out.println(tre.antall(4));     // Utskrift: 3
System.out.println(tre.antall(7));     // Utskrift: 2
System.out.println(tre.antall(10));    // Utskrift: 1
```

Oppgave 3

Lag hjelpemetodene *private static <T> Node<T> førstePostorden(Node<T> p)* og *private private static <T> Node<T> nestePostorden(Node<T> p)*. Siden dette er private metode, tas det som gitt at parameteren *p* ikke er *null*. Det er når metoden brukes at en må sikre seg at det ikke går inn en nullreferanse. Førstepostorden skal returnere første node post orden med p som rot, og nestePostorden skal returnere den noden som kommer etter *p* i ***postorden***. Hvis *p* er den siste i postorden, skal metoden returnere *null* (Se

seksjon "5.1.7 Preorden, inorden og postorden" for detaljer om postorden og hvordan man kan finne neste post orden).

Oppgave 4

Lag hjelpemetodene `public void postorden(Oppgave <? super T> oppgave)` og `private void postordenRecursive(Node<T> p, Oppgave<? super T> oppgave)` som brukes til å utføre en oppgave. Oppgave kan for eksempel være skriv til skjerm, og da vil denne metoden skrive ut treet i post orden. Du skal implementere den første funksjonen uten bruk av rekursjon og uten bruk av hjelpevariabler som stack / queue. Du skal bruke funksjonen nestePostorden fra forrige oppgave. Start med å finne den første noden *p* i postorden. Deretter vil (f.eks. i en while-løkke) setningen: *p = nestePostorden(p);* gi den neste. Osv. til *p* blir *null*. For den rekursive metoden skal du lage et rekursivt kall som traverserer treet i postorden rekkefølge.

Oppgave 5

For å lagre et binært søketre i en fil må vi legge treet inn i en datastruktur som egner seg for fil-skriving. I denne oppgaven skal du lage serialize som gjør om binærtreet til et array, og tilsvarende deserialize som tar et array og gjør om til et binært søketre. Lag hjelpemetoden **`public ArrayList<T> serialize()`** og **`static <K> SBinTre<K> deserialize(ArrayList<K> data, Comparator<? super K> c)`**. Metoden serialize skal være iterativ og må bruke en kø til å traversere treet i nivå orden. Arrayet som returneres av serialize skal inneholde verdiene i alle nodene i nivå orden. Deserialize skal da ta dette arrayet, og legge inn alle verdiene (igjen i nivå orden), og dermed gjenskape treet.

Et eksempel på hvordan det skal virke:

```
//Lag et nytt binærtre
SBinTre<Integer> tre =
       new SBinTre<>(Comparator.naturalOrder());
int[] a = {10, 14, 6, 8, 1, 12, 7, 3, 11, 9, 13, 5, 2, 4};
for (int verdi : a) { tre.leggInn(verdi); }

//Gjør om treet til et array
ArrayList<Integer> data = tre.serialize();

//Lag nytt tre fra arrayet over
SBinTre<Integer> tre2 = SBinTre.deserialize(data, Comparator.naturalOrder());

//Utskriften av tre og tre2 skal være identiske
System.out.println(tre.toStringPostOrder());
System.out.println(tre2.toStringPostOrder());
```

Oppgave 6

Lag metoden *`public boolean fjern(T verdi)`*. Der kan du kopiere *Programkode* 5.2.8 d), men i tillegg må du gjøre de endringene som trengs for at pekeren *`forelder`* får korrekt verdi i alle noder etter en fjerning. Lag så metoden *`public int fjernAlle(T verdi)`*. Den skal fjerne alle forekomstene av *`verdi`* i treet. Husk at duplikater er tillatt. Dermed kan en og samme verdi ligge flere steder i treet. Metoden skal returnere antallet som ble fjernet. Hvis treet er tomt, skal 0 returneres. Lag så metoden *`public void nullstill()`*. Den skal traversere (rekursivt eller iterativt) treet i **en** eller annen rekkefølge og sørge for at samtlige

pekere og nodeverdier i treet blir nullet. Det er med andre ord ikke tilstrekkelig å sette *rot* til *null* og *antall* til 0.

Et eksempel på hvordan det skal virke:

```
int[] a = {4,7,2,9,4,10,8,7,4,6,1};
SBinTre<Integer> tre = new SBinTre<>(Comparator.naturalOrder());
for (int verdi : a) tre.leggInn(verdi);

System.out.println(tre.fjernAlle(4));  // 3
tre.fjernAlle(7); tre.fjern(8);

System.out.println(tre.antall());  // 5

System.out.println(tre + " " + tre.omvendtString());
// [1, 2, 6, 9, 10] [10, 9, 6, 2, 1]
```