DATS2300 / ITPE2300

Algoritmer og Datastrukturer Høst 2020

Mappeeksamen

NB! Dere må følge alle kravene til innlevering! Les nøye krav til innlevering før dere begynner arbeidet

I denne mappeeksamenen 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.

Det er et krav at arbeid med oppgaven dokumenteres ved bruk av git/github. Oppgaven leveres ved å lage en såkalt "git bundle" og levere denne inn. Dette gjøres ved å høyre-klikke på prosjektet i Intellij, og velge "open in terminal". Skriv så inn kommandoen

```
git bundle create mappeeksamen.bundle --all
```

Dette lager filen mappeeksamen.bundle som skal leveres inn. **Test at dette fungerer med en gang du har laget git repository for eksamenen.**

Krav til innlevering:

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. Dvs. at EksamenSBinTre.java ligger i stien src/no/oslomet/cs/algdat/Eksamen/EksamenSBinTre.java
   Veldig viktig at den ligger her i github-repositoriet.
3. Git bundle er laget og lastet opp til Inspera
4. Beskrivelse av hvordan oppgaven er løst (4-8 linjer/setninger per oppgave) står i Readme.md.
5. Klassen `EksamenSBinTre` skal ikke ha noen main-metode eller debug-utskrifter (system.out.println) når den leveres inn.
6. Warnings er enten beskrevet i readme.md eller fjernet.
7. Alle oppgavene består den utleverte testen.

`EksamenSBinTre` er et binært søketre av samme type som beskrevet i *Delkapittel* 5.2. Men `EksamenSBinTre` 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 ferdigkodet og de andre kaster en `UnsupportedOperationException`.

Hvis du i din løsning bruker noen av de strukturene som er laget i undervisningen (f.eks. en liste, en stakk, en kø eller noe annet), skal ***ikke*** koden for dem følge med løsningen.

Obs: De metodene i `EksamenSBinTre` 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.

`EksamenSBinTre` har variabelen *endringer* og `BladnodeIterator` har *iteratorendringer*. De skal fungere på samme måte her som i klassen `DobbeltLenketListe` fra Oblig 2.

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

0. Innledning: Legg klassen `EksamenSBinTre` inn i ditt Java-prosjekt. I git og Intellij-prosektet skal filen ligge som følger:
src/no/oslomet/cs/algdat/Eksamen/EksamenSBinTre.java
Det er svært viktig at filen ligger på nettopp denne stien. For at det skal virke må grensesnittet `Beholder` være tilgjengelige. Lag så noen instanser av klassen `EksamenSBinTre`. Sjekk at det ikke gir noen syntaksfeil (eller kjørefeil). Bruk f.eks. både `Integer`, `Character` og `String` som datatyper. Da kan du bruke en «naturlig» komparator i konstruktøren. Dvs. slik for datatypen `String`:

```
EksamenSBinTre<String> tre = new EksamenSBinTre<>(Comparator.naturalOrder());
System.out.println(tre.antall());  // Utskrift: 0
```

1. En `Node` i `EksamenSBinTre` 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};
EksamenSBinTre<Integer> tre = new EksamenSBinTre<>(Comparator.naturalOrder());
for (int verdi : a) tre.leggInn(verdi);
System.out.println(tre.antall());  // Utskrift: 10
```

2. Metodene *inneholder()*, *antall()* og *tom()* er ferdigkodet. 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};
EksamenSBinTre<Integer> tre = new EksamenSBinTre<>(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
```

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).

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.

5. Lag hjelpemetoden `public ArrayList<T> serialize()` og `static <K> EksamenSBinTre<K> deserialize(ArrayList<K> data, Comparator<? super K> c)`. Metodene skal henholdsvis serialisere (lage et kompakt format egnet for lagring til f.eks. fil - array) og deserialisere (lage et nytt tre ut ifra et array). Selve 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.

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};
EksamenSBinTre<Integer> tre = new EksamenSBinTre<>(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]
// OBS: Hvis du ikke har gjort oppgave 4 kan du her bruke toString()
```