DM507 Algoritmer og datastrukturer

Forår 2021

Projekt, del I

Java version

Institut for matematik og datalogi
Syddansk Universitet

1. marts, 2021

Dette projekt udleveres i tre dele. Hver del har sin deadline, således at arbejdet strækkes over hele semesteret. Deadline for del I er fredag den 19. marts kl. 18:00. De tre dele I/II/III er ikke lige store, men har omfang omtrent fordelt i forholdet 15/30/55. Projektet skal besvares i grupper af størrelse to eller tre.

Mål

Det overordnede mål for projektet i DM507 er træning i at overføre kursets viden om algoritmer og datastrukturer til programmering. Projektet og den skriftlige eksamen komplementerer hinanden, og projektet er ikke ment som en forberedelse til den skriftlige eksamen.

Det konkrete mål for del I af projektet er at implementere datastrukturen *prioritetskø*. Prioritetskøen vil blive brugt igen senere i del III af projektet.

Opgaven

Kort sagt er opgaven at overføre bogens pseudo-kode for prioritetskøer til en Java-klasse og og derefter testen den, bl.a. ved at bruge den til at sortere tal.

I detaljer: Du skal i Java lave en klasse `PQHeap`, som tilbyder (`implements`) følgende interface (koden udleveres):

```
public interface PQ {
    public Element extractMin();
    public void insert(Element e);
```

```
}
```

Metoden `extractMin()` skal fjerne det element i prioritetskøen, som har mindst prioritet, og returnere det. Metoden `insert(e)` skal indsætte elementet `e` i prioritetskøen.

For nemheds skyld skal du antage, at `extractMin()` kun kaldes på en ikke-tom prioritetskø. Det overlades til brugeren af prioritetskøen at sikre dette, f.eks. ved at holde styr på antal elementer i prioritetskøen.

Krav

Du skal implementere `PQHeap.java` ved hjælp af strukturen *Heap* i en `ArrayList` af `Element`s. Du skal basere den på pseudo-koden i Cormen et al. kapitel 6 på siderne 163, 154, 152, samt på den sammenskrivning af pseudo-koden fra side 164, som findes længere nede i denne tekst.

Klassen `PQHeap` skal have en constructor-metode `PQHeap()`, som opretter en ny, tom prioritetskø. Der vil være brug for at implementere metoder udover constructor-metode og metoderne i interfacet, til internt brug i klassen.

`Element` er en type, der implementerer et (nøgle, data)-par. Denne type er givet i følgende klasse (koden udleveres):

```
public class Element {

    private int key;
    private Object data;

    public Element(int i, Object o){
        this.key = i;
        this.data = o;
    }
    public int getKey(){
        return this.key;
    }
    public Object getData(){
        return this.data;
    }
}
```

Vi kalder `e.key` for elementets prioritet. Elementers prioriteter er altså af typen `int`, og deres associerede data er af typen `Object`.

Detaljer mht. implementationen:

1. Du skal i dette projekt lave en *min*-heap struktur, mens bogen formulerer sin pseudo-kode for en *max*-heap struktur. Pseudo-koden skal derfor have alle uligheder mellem elementer vendt (men ikke uligheder mellem indekser).

2. Da vi bruger en `Arraylist`, som automatisk gøre sig større og mindre, skal værdien *heap-size* ikke vedligeholdes, men kan findes via metoden `size()` fra `Arraylist`.

3. Bogens pseudo-kode indekserer arrays startende med 1, mens Java starter med 0. Det betyder, at formlerne for venstre barn, højre barn og forælder skal justeres fra dem i bogen til følgende:

   - $\textsc{Left}(i) = 2i + 1$
   - $\textsc{Right}(i) = 2i + 2$
   - $\textsc{Parent}(i) = \lfloor (i-1)/2 \rfloor$

4. Det betyder også, at første index (roden) er 0 (ikke 1), og at sidste index (sidste blad) er `size()-1` (ikke *heap-size*). I pseudo-koden skal anvendelser af indekser og sammenligninger mellem indekser justeres tilsvarende.

5. I `Arraylist`'en skal man tilgå første element via `get()` og `set()` (disse tager $\Theta(1)$ tid), ikke `add()` og `remove()` (disse tager $\Theta(n)$ tid for første element). For at tilgå sidste element skal man bruge `add()` og `remove()` (disse tager $\Theta(1)$ tid for sidste element, og sikrer at `size()` passer med antal elementer i heapen).

6. Parametrene i metoderne i interfacet `PQ` er ikke præcis de samme som i bogens pseudo-kode. Dette skyldes at i objektorienteret programmering kaldes metoder på et objekt `Q` via syntaksen `Q.metode()` fremfor `metode(Q)`, samt at bogen kun opererer med prioriteter og ikke elementer med ekstra data. Derudover er `A` i bogens pseudo-kode et array indeholdende heapen, som *ikke* bør kunne tilgås direkte af brugere af et prioritetskø-objekt `Q` på anden måde end gennem metoderne fra interfacet.

7. I pseudo-koden på side 163 kan **if** i linie 1–2 udelades pga. antagelsen om, at prioritetskøen ikke er tom.

8. På side 164 kan de to stykker pseudo-kode på siden bygges sammen til ét, da vi ikke skal lave en `increaseKey()`. Dette giver nedenstående variant af pseudo-koden for `insert`. Denne variant skal bruges i projektet.

   ```
   MAX-HEAP-INSERT(A, key)
     A.heap-size = A.heap-size + 1
   ```

$i = A.\textit{heap-size}$
$A[i] = \textit{key}$
while $i > 1$ and $A[\textsc{Parent}(i)] < A[i]$
  exchange $A[i]$ with $A[\textsc{Parent}(i)]$
  $i = \textsc{Parent}(i)$

Bemærk at ovenstående pseudo-kode bruger samme konventioner som den i bogen, og derfor skal justeres ligesom denne (dvs. ud fra punkterne 1–6 ovenfor).

Test

Du skal naturligvis afprøve din klasse grundigt. Herunder skal du teste den med det udleverede program `PQSort`, der sorterer ved at lave gentagne `insert`'s i en prioritetskø, efterfulgt af gentagne `extractMin`'s.[1]

Programmet `PQSort` læser (via klassen `Scanner`) fra standard input, der som default er tastaturet, og skriver til standard output, der som default er skærmen. Input til `PQSort` er en sekvens af `char`'s bestående af heltal adskilt af newlines, og programmet skriver som output tallene i sorteret orden, adskilt af whitespace.

Som eksempel kan `PQSort` kaldes således i en kommandoprompt:

```
java PQSort
34
645
-45
1
34
0
Control-D
```

(Control-D angiver slut på data under Linux og Mac, under Windows brug Control-D og derefter Enter) og giver så flg. output på skærmen:

```
-45
0
1
34
```

[1] Da programmet kun sorterer `int`s, sætter det data-delen af elementerne til at være `null`. Senere i del III af projektet skal data-delen faktisk bruges til noget.

```
34
645
```

Ved hjælp af *redirection*[2] af standard input og output kan man i en kommandoprompt anvende *samme* program også på filer således:

```
java PQSort  < inputfile  > outputfile
```

Du skal inden aflevering på denne måde teste sortering af alle de udleverede datafiler. En vigtig grund til, at du skal afprøve ovenstående metode (med redirection i en kommandoprompt), er at programmerne skal kunne testes automatisk efter aflevering. Man må af samme grund heller ikke i sin kildekode have `package` statements, eller organisere sin kode i en folderstruktur. Dette sker ofte automatisk hvis man bruger en IDE som Eclipse, NetBeans eller IntelliJ under udvikling af koden. I så fald må man fjerne `package` statements og folderstruktur inden aflevering, (og derefter igen teste funktionaliteten, herunder redirection.).

Formalia

Du skal kun aflevere din Java source-fil `PQHeap.java`. Den skal indeholde navnene og SDU-logins på gruppens medlemmer.

Filen skal afleveres elektronisk i itslearning i mappen Afleveringsopgaver under faneblad Ressourcer. Der behøves kun afleveres under én persons navn i gruppen. Bemærk at man kan oprette midlertidige "drafts", men man kan kun aflevere *én gang*.

Aflever senest:

Fredag den 19. marts, 2021, kl. 18:00.

Bemærk at aflevering af andres kode eller tekst, hvad enten kopieret fra medstuderende, fra nettet, eller på andre måder, er eksamenssnyd, og vil blive behandlet særdeles alvorligt efter gældende SDU regler. Man lærer desuden heller ikke noget. Kort sagt: kun personer, hvis navne er nævnt i den afleverede fil, må have bidraget til koden.

[2] Læs evt. om redirection på William Shotts' website (med flere detaljer her), Wikipedia eller Unix Power Tools.