DM507 Algoritmer og datastrukturer

Forår 2019

Projekt, del I

Institut for matematik og datalogi
Syddansk Universitet

27. februar, 2019

Dette projekt udleveres i tre dele. Hver del har sin deadline, således at arbejdet strækkes over hele semesteret. Deadline for del I er søndag den 17. marts. 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*, og derefter testen den ved at bruge den til at sortere tal. 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.

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

For nemheds skyld skal du antage, at `extractMin()` kun kaldes på en ikke-tom prioritetskø og at `insert(e)` kun kaldes på en prioritetskø med plads til endnu et element - 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 et array af `Element`s, og 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 én constructor-metode `PQHeap(int maxElms)`, som returnerer en ny, tom prioritetskø. Argumentet `maxElms` angiver det maksimale antal elementer, der skal være plads til i køen—array'et i en `PQHeap` skal naturligvis oprettes med denne længde. 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 vendt.

2. Bogens pseudo-kode indekserer arrays startende med 1, mens Java starter med 0. En simpel måde at anvende bogens pseudo-kode på i Java, er at lægge én til den ønskede længde på array'et, og så ikke bruge pladsen med index 0 til noget. Hvis man i stedet vil skifte til start med 0, skal formlerne for venstre barn, højre barn og forælder justeres passende.

3. 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.

4. Bemærk (se side 151) forskellen på `A.length` og `A.heap-size`. Den første er array'ets længde (et fast tal), mens den sidste er, hvor mange celler i array'et som den nuværende heap bruger (nemlig de første `A.heap-size` celler). Du skal lave `heap-size` som en (privat) variabel i `PQHeap`.

5. I pseudo-koden på side 163 kan første **if** udelades pga. antagelsen om, at prioritetskøen ikke er tom.

6. På side 164 kan de to stykker pseudo-kode på siden bygges sammen til ét, da vi ikke skal lave en `increaseKey()`. Dette giver følgende variant (som skal bruges i dette projekt) af pseudo-koden for `insert`:

```
MAX-HEAP-INSERT(A, key)
  A.heap-size = A.heap-size + 1
  i = A.heap-size
  A[i] = key
  while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

Test

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

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

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

```
java Heapsort
34 645 -45 1 34 0
Ctrl-D
```

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

```
-45
0
1
34
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 Heapsort  < 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.).

[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.

[2] Læs evt. om redirection på Unix Power Tools eller Wikipedia.

Formalia

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

Filen skal afleveres elektronisk i Blackboard med værktøjet "SDU Assignment", som findes i menuen til venstre på kursussiden i Blackboard. Der behøves kun afleveres under én persons navn i gruppen.

Filen skal *også* afleveres udprintet på papir i instruktorens boks på Imada (vælg én af instruktorerne, hvis personerne i gruppen går på forskellige hold). På kursets hjemmeside under linket "Instruktorer" er der links, som angiver, hvor disse bokse er placeret på IMADA.

Der skal blot afleveres én kopi per gruppe. Afleveringens sider skal sættes sammen med hæfteklamme.

Aflever senest:

Søndag den 17. marts, 2019, kl. 24:00.

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.