DM507 Algoritmer og datastrukturer

Forår 2016

Projekt, del I

Institut for matematik og datalogi
Syddansk Universitet

29. februar, 2016

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 18. marts. Projektet skal besvares i grupper af størrelse to (evt. tre, efter aftale med underviser).

Mål

Målet for del I af projektet er først at implementere datastrukturen *prioritetskø*, og derefter bruge den til at sortere tal. Prioritetskøen vil blive brugt igen senere i del III af projektet.

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

Opgaver

Opgave 1

Du skal i Java implementere en datastruktur, som tilbyder (i Java: `implements`) følgende interface:

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

Her er `Element` en type, der implementerer et (nøgle,data)-par. Denne type er defineret ved følgende klasse:

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

De to dele af et objekt `e` af typen `Element` skal blot tilgås som `e.key` og `e.data`. Vi vil omtale `e.key` som elementets prioritet. Elementers prioriteter er altså af typen `int`, og deres associerede data er af typen `Object`.

Metoden `extractMin()` returner det element i prioritetskøen som har mindst prioritet. Det må gerne antages at metoden kun kaldes på en ikke-tom prioritetskø. Metoden `insert(e)` indsætter elementet `e` i prioritetskøen. Det må gerne antages at metoden kun kaldes på en prioritetskø med plads til endnu et element. Dette betyder, at det overlades til brugeren af prioritetskøen at sikre at ovenstående antagelser er opfyldt under brug, f.eks. ved at holde styr på antal elementer i prioritetskøen.

> Implementationen skal laves ved hjælp af strukturen *heap* i et array af `Element`s, og *skal* basere sig på pseudo-koden i Cormen et al. kapitel 6 på siderne 163, 154, 152 og 164. Dog kan følgende simplificeringer gøres: på side 163 kan første **if** udelades pga. antagelsen om at prioritetskøen ikke er tom, og på side 164 kan de to stykker pseudo-kode på siden bygges sammen til én (pga. at vi ikke skal lave en eksplicit `increaseKey()` (eller rettere, for min-heaps, en `decreaseKey()`)), hvorved man kan spare brugen af ∞ samt **if** i det første stykke pseudo-kode.

Implementationen skal være i form af en Java-klasse, som kan bruges af andre programmer. Klassen skal hedde `PQHeap`, og skal implementere interfacet `PQ`. Klassen skal have én constructor-metode `PQHeap(int maxElms)`, som returnerer en ny, tom prioritetskø. Argumentet `maxElms` angiver det maksimale antal elementer i køen. Der vil være behov for at implementere metoder udover dem i interfacet, til internt brug i klassen.

Bemærk følgende detaljer:

- 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.
- 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 at bruge pladsen med index 0 til noget.

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

Opgave 2

Du skal implementere en sorteringsalgoritme baseret på at bruge metoderne i interfacet `PQ`. Algoritmen skal bestå i at lave gentagne `insert`'s i en prioritetskø, efterfulgt af at lave gentagne `extractMin`'s. Da vi kun skal sortere tal kan data-delen af elementerne sættes til at være `null`.[1]

Algoritmen skal implementeres i et program kaldet `Heapsort`. Dette program skal bruge din ovenfor udviklede klasse `PQHeap` som blackbox/biblioteksfunktion.

Programmet `Heapsort` skal læse fra standard input (der som default er tastaturet), og skrive til standard output (der som default er skærmen). Input skal læses via klassen `Scanner` fra biblioteket `java.util` og skal bruge dens metode `nextInt()` til at indlæse tallene. Mere præcist er input en sekvens af `char`'s bestående af heltal adskilt af whitespace. Programmet skal som output skrive tallene i sorteret orden, adskilt af whitespace.

Derved skal `Heapsort` kunne kaldes således i en kommandoprompt:

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

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

```
-45
0
1
34
34
645
```

[1] Senere i del III af projektet skal data-delen faktisk bruges til noget.

Bemærk at ved hjælp af redirection[2] af standard input og output kan man i en kommandoprompt anvende programmerne (helt uden at ændre i dem) på filer også:

```
java Heapsort   < inputfile   > outputfile
```

Det er vigtigt at I afprøver ovenstående i en kommandoprompt, da programmerne skal kunne testes automatisk. Man må af samme grund heller ikke i sin kildekode have `package` statements, eller organisere sin kode i en folderstruktur[3].

En detaljer er, at man under scan af input ikke ved, hvor mange tal der er. Dette er et problem for Heapsort, som skal kende det maksimale antal elementer i prioritetskøen. Én løsning er at indlæse i en `ArrayList`, og derfra (når scan af input stopper, og man kender antallet af elementer) lave `insert`'s i en prioritetskø.

Formalia

Du skal kun aflevere dine to Java source-filer `PQHeap.java` og `Heapsort.java`. Disse skal indeholde grundige kommentarer. De skal også indeholde navnene og SDU-logins på gruppens medlemmer.

Filerne skal afleveres elektronisk i Blackboard med værktøjet "SDU Assignment", som findes i menuen til venstre på kursussiden i Blackboard. De skal enten afleveres som individuelle filer eller som ét `zip`-arkiv (med alle filer på topniveau, dvs. uden nogen directory struktur). Der behøves kun afleveres under én persons navn i gruppen.

Filerne skal *også* afleveres udprintet på papir i instruktorens boks på Imada (vælg én af instruktorerne, hvis personerne i gruppen går på forskellige hold). Spørg Imadas sekretær hvis man ikke ved, hvor instruktorboksene er. Der skal blot afleveres én kopi per gruppe.

Det er *vigtigt* at overholde alle syntaktiske regler ovenfor (navngivning af klasser, kald af programmer, `zip`-arkiv uden directory struktur, ingen `package` statements), da programmerne som sagt vil blive testet på automatiseret måde.

Aflever materialet senest:

Fredag den 18. marts, 2015, kl. 12:00.

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

[3] NB: Dette sker ofte automatisk hvis man bruger en IDE som Eclipse eller NetBeans under udvikling af koden. I så fald må man fjerne `package` statements og folderstruktur inden aflevering, (og derefter igen teste funktionaliteten, herunder redirection.)

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 som sådan. Man lærer desuden heller ikke noget.