DM507 Algoritmer og datastrukturer

Forår 2021

Projekt, del I

Python 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 et Python-program og og derefter testen det, bl.a. ved at bruge det til at sortere tal.

Krav

Dit program skal hedde `PQHeap.py` og det skal implementere strukturen *Heap*. Heapen skal bestå af en liste `A` af tal (ikke et array af tal, som i bo-

gen). Programmet skal indeholde funktionen `extractMin(A)`, som fjerner det element i prioritetskøen `A`, der har mindst prioritet, og returnere det. Programmet skal også indeholde funktionen `insert(A,e)` som indsætter elementet `e` i prioritetskøen `A`. Desuden skal programmet indeholde funktionen `createEmptyPQ()`, der returner en ny, tom prioritetskø (dvs. en tom liste).

Du skal bruge Python 3 (ikke Python 2). Du skal basere implementationen 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. Der vil være brug for at implementere funktioner udover `extractMin(A)` og `insert(A,e)`, til internt brug i programmet.

For nemheds skyld skal du antage, at `extractMin(A)` 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.

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 liste, som automatisk kan gøre sig større og mindre i listens ende (med metoderne `append(x)` og `pop()` (uden argument)) skal værdien *heap-size* ikke vedligeholdes, men kan findes via funktionen `len(A)`.

3. Bogens pseudo-kode indekserer arrays startende med 1, mens Python 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 `len(A)-1` (ikke *heap-size*). I pseudo-koden skal anvendelser af indekser og sammenligninger mellem indekser justeres tilsvarende.

5. I pseudo-koden på side 163 kan **if** i linie 1–2 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 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.heap-size
  A[i] = key
  while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

   Bemærk at ovenstående pseudo-kode bruger samme konventioner som den i bogen, og derfor skal justeres ligesom denne (dvs. ud fra punkterne 1-4 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.

Programmet `PQSort` læser (via filobjektet `sys.stdin`) 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 tegn, som repræsenterer heltal, adskilt af newlines (dvs. ét tal pr. linie). Programmet skriver som output tallene i sorteret orden, adskilt af newlines.

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

```
python PQSort.py
34
645
-45
1
34
0
Control-D
```

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

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

Ved hjælp af *redirection*[1] af standard input og output kan man i en kommandoprompt anvende *samme* program også på filer således:

```
python PQSort.py  < 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.

Formalia

Du skal kun aflevere din Python source-fil `PQHeap.py`. 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.

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