DM507/DS814 Algoritmer og datastrukturer

Forår 2026

Projekt, del II

Institut for matematik og datalogi
Syddansk Universitet

9. april, 2026

Dette projekt udleveres i tre dele. Hver del har sin deadline, således at arbejdet strækkes over hele semesteret. Deadline for del II er mandag den 27. april kl. 23:59. De tre dele I/II/III er ikke lige store, men har omfang omtrent fordelt i forholdet 15/25/60. 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 II af projektet er først at implementere datastrukturen *ordered dictionary* (ordnet ordbog), og derefter bruge den til at sortere tal. Arbejdet vil også virke som forberedelse til del III af projektet.

Opgave 1

Kort sagt er opgaven at overføre bogens pseudo-kode for ubalancerede søgetræer til et Python-program.

Krav i opgave 1

Dit program skal hedde `DictBinTree.py`. Programmet skal implementere datastrukturen *binært søgetræ* med tal som nøgler. Det skal du gøre ved at lave en Python klasse `DictBinTree`, som tilbyder følgende metodekald på et objekt `T` af typen `DictBinTree`:

- `T.search(k)`, der returnerer en Boolean, der angiver om nøglen `k` er i træet `T`.
- `T.insert(k)`, der indsætter nøglen `k` i træet `T`.
- `T.orderedTraversal()`, der returnerer en liste med nøglerne i træet `T` i sorteret orden (fremfor at printe dem på skærmen som i bogens pseudo-kode).

Nye træer oprettes som sædvanligt med et kald `DictBinTree()`, der skal returnere et tomt træ.

Implementationen skal følge beskrivelsen og pseudo-koden i Cormen et al. kapitel 12. Som det fremgår af ovenstående, skal der kun implementeres indsættelse (pseudo-kode side 321), søgning (pseudo-kode side 316), og inorder gennemløb (pseudo-kode side 314) [i 3. udgave er de tilsvarende sidetal 294, 290/291 og 288]. Implementationen *skal* tage udgangspunkt i denne pseudo-kode. Træet skal ikke holdes balanceret (der skal ikke bruges metoder fra kapitel 13).

Du skal definere en separat klasse `BinNode` til at repræsentere knuder i træer (så du skal ikke bruge en array-struktur til at repræsentere træet, i modsætning til heapen i del I af projektet). Et objekt af typen `BinNode` skal indeholde to andre `BinNode`'s (knudens venste barn og højre børn), med værdi `None` hvis pågældende barn ikke findes. Det skal også indeholde en nøgle `k`, men behøver i dette projekt ikke at indeholde forælderen.

Et objekt af typen `DictBinTree` skal indeholder den `BinNode`, som er rod i træet (hvis træet er tomt, er denne rod `None`). Det vil sige, at et `DictBinTree` objekt og dets rod svarer til T og $T.root$ fra Cormen et al.. Se figur 10.6 side 266 [figur 10.9 side 247] for en illustration.

For hvert træ er der således præcis ét objekt af typen `DictBinTree` og nul eller flere objekter af typen `BinNode`. Under `insert` skal ét nyt objekt af typen `BinNode` oprettes.

Fra pseudo-kode til implementation

Der vil være behov for at implementere funktioner udover ovennævnte, til internt brug i programmet. F.eks. vil der for funktioner baseret på rekursion skulle laves *to* udgaver: den "offentlige" funktion beskrevet ovenfor i starten af dette afsnit, samt en funktion som gør det virkelige arbejde (og som svarer til pseudo-koden). Den første er ikke rekursiv, men kalder blot den anden og tilføjer i kaldet parametre med relevante værdier (f.eks. at knuden, som der kaldes på, er træets rod). For funktioner baseret på en løkke, kan disse parameter blot oprettes, inden løkken går i gang. I inorder gennemløb vil der være brug for at sende en liste med som argument. Når pseudo-koden for

inorder gennemløb vil udskrive en knudes element, skal man i stedet tilføje elementet til denne list med `append()`.

Husk at teste dit program grundigt (herunder test på tomme træer, test af indsættelse af ens nøgler, samt test af søgning efter både eksistende og ikke-eksisterende nøgler), inden du går videre til næste opgave.

Opgave 2

Du skal implementere en sorteringsalgoritme kaldet Treesort baseret på funktionerne i programmet `DictBinTree.py`. Algoritmen består i at lave gentagne `insert`'s i en dictionary, efterfulgt af et kald til `orderedTraversal`. Tallene i listen returneret fra dette kald skal så blot skrives ud.

Krav i opgave 2

Algoritmen skal implementeres i et program kaldet `Treesort.py`. Dette program skal bruge funktionerne fra dit program `DictBinTree.py` udviklet ovenfor.

Præcis som det udleverede program `PQSort.py` fra del I af projektet skal `Treesort.py` via filobjektet `sys.stdin` læse fra standard input (der som default er tastaturet), og via `print` skrive til standard output (der som default er skærmen). Input til `Treesort.py` er en sekvens af `char`'s bestående af heltal adskilt af newlines, og programmet skriver som output tallene i sorteret orden, adskilt af newlines. Som eksempel skal `Treesort.py` kunne kaldes således i en kommandoprompt:

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

(Control-D angiver 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
```

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

```
python Treesort.py   < inputfile   > outputfile
```

Som test af `Treesort.py` kan man køre det på testfilerne fra del I.

En vigtig grund til, at du skal afprøve ovenstående metode (med redirection i en kommandoprompt), er, at programmerne vil blive testet på denne måde efter aflevering.

Formalia

Du skal kun aflevere dine to filer med Python kildekode. Disse skal indeholde en fornuftig mængde kommentarer om, hvad koden gør. De skal også indeholde navnene og SDU-logins på gruppens medlemmer.

Filerne skal enten afleveres som individuelle filer eller som ét `zip`-arkiv (med alle filer på topniveau, dvs. uden nogen directory struktur).

Hvis nogen dele af koden er skrevet af generativ AI, eller på anden måde via kopiering fra andet værk, skal dette erklæres i source-filerne. Koden vil så *ikke* blive læst og vil ikke få feedback, men vil kun blive udsat for tests. Dette er fordi, at formålet med projektet er, at de studerende opnår kompetencer til *selv at udføre opgaven*. Hvis en gruppe sætter læringsambitionen lavere end dette, ønsker jeg også at sænke brugen af ressourcer under retning.

Filerme skal afleveres elektronisk i itslearning i mappen Projekt under fanebladet Ressourcer. Afleveringsmodulet er også sat ind i en itslearning plan i kurset.

Under afleveringen skal man erklære gruppen ved at angive alle medlemmernes navne. Man skal kun aflevere én gang per gruppe. Bemærk, at man under aflevering kan oprette midlertidige "drafts", men man kan kun aflevere *én gang*.

Aflever materialet senest:

Mandag den 27. april kl. 23:59.

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