

DM507/DS814 Algoritmer og datastrukturer

Forår 2023

Projekt, del II

Institut for matematik og datalogi
Syddansk Universitet

28. marts, 2023

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 24. april kl. 23:59. 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 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.

Opgaver

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, og det skal indeholde

følgende funktioner:

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

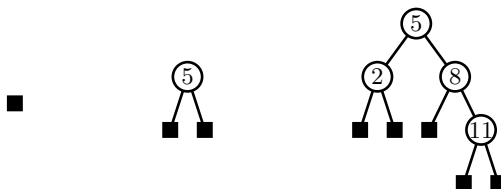
Implementationen skal følge beskrivelsen og pseudo-koden i Cormen et al. kapitel 12. Som det fremgår af ovenstående, skal der kun implementeres indsættelse (pseudo-kode side 321), søgning (pseudo-kode side 316), og in-order gennemløb (pseudo-kode side 314) [i 3. udgave er de tilsvarende sidetal 294,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

I programmet vil der være brug for at repræsentere knuder, som hver består af en nøgle, et venstre barn og et højre barn.¹ I dette projekt skal en knude blot repræsenteres ved en liste² af længde tre, hvor det første element er nøglen, det andet element er venstre barn, og det tredje element er højre barn. Tomme undertræer skal repræsenteres som `None`.

Her er tre eksempler:

- En “knode” repræsenterende et tomt træ: `None`
- En knude med nøglen 5 og to tomme undertræer: `[5, None, None]`
- Et knode som er rod i et træ med i alt fire knuder (med nøglerne 2,5,8,11): `[5, [2, None, None], [8, None, [11, None, None]]]`

Disse tre eksempler er illustreret her nedenfor:



¹I bogen indeholder knuder også en parent `p`, men den har vi ikke brug for her.

²Hvis du er fortrolig med klasser og objekter i Python, kan du i stedet for den her beskrevne liste-repræsentation af knuder og træer lave to klasser `BinNode` og `DictBinTree` til at repræsentere dem. Se figur 10.6 side 266 [figur 10.9 side 247] for en illustration.

For en knude `v` kan dens nøgle, venstre barn og højre barn altså tilgås som henholdsvis `v[0]`, `v[1]` og `v[2]`. Vi vil i denne opgave ikke have brug for at tilgå en knudes forælder.

For at få en pendant til T og $T.root$ fra Cormen et al. (jvf. figur 10.6 side 266 [figur 10.9 side 247] samt pseudokode i kapitel 12) skal et helt træ T repræsenteres som en liste af længde én, der blot indeholder roden. Derved kan pseudo-koden fra bogen bruges mest direkte.

De tre eksempler ovenfor bliver derved til:

- Et tomt træ: `[None]`
- En træ som indeholder en knude med nøglen 5: `[[5, None, None]]`
- Et træ som indeholder fire knuder med nøglerne 2,5,8,11:
`[[5, [2, None, None], [8, None, [11, None, None]]]]`

For et træ T kan dets rod (kaldet `T.root` i pseudokoden) altså tilgås som `T[0]`.

Bemærk at du *ikke* skal repræsentere træet i én stor liste på den måde, som vi gjorde for en heap i del I af projektet (hvor navigering i træet kunne gøres ved beregninger på listeindekser). Den metode virker kun for meget balancerede træer.

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 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 listen 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 eksisterende 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 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*³ 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 skal kunne testes automatisk efter aflevering.

³Læs evt. om redirection på William Shotts' website (med flere detaljer her), Wikipedia eller Unix Power Tools.

Formalia

Du skal kun aflevere dine 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).

Filen skal afleveres elektronisk i *itslearning* i mappen Projekt under faneblad 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 24. april kl. 23:59

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.