DM507 Algoritmer og datastrukturer

Forår 2020

Projekt, del II

Python version

Institut for matematik og datalogi
Syddansk Universitet

17. marts, 2020

Dette projekt udleveres i tre dele. Hver del har sin deadline, således at arbejdet strækkes over hele semesteret. Deadline for del II er tirsdag den 7. april kl. 12:00. 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 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: Funktionen `search(T,k)`, som returnerer en boolean, der angiver om nøglen `k` er i træet `T`. Funktionen `insert(T,k)`, som indsætter nøglen `k` i træet `T`. Funktionen `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).

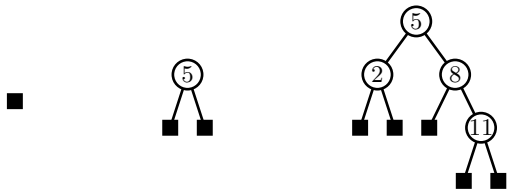
Implementationen skal følge beskrivelsen og pseudo-koden i Cormen et al. kapitel 12. Som det fremgår af ovenstående skal der kun implementeres indsættelse (pseudo-kode side 294), søgning (pseudo-kode side 290 eller 291), og inorder gennemløb (pseudo-kode side 288). Træet skal ikke holdes balanceret (der skal ikke bruges metoder fra kapitel 13). Der vil være behov for at implementere funktioner udover ovennævnte, til internt brug i programmet.

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

Her er tre eksempler:

- En "knude" repræsenterende et tomt træ: `None`
- En knude med nøglen 5 og to tomme undertræer: `[5,None,None]`
- Et knude 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:



[1]Hvis 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 evt. beskrivelsen af sådanne klasser i Java-versionen af projektbeskrivelsen.

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. (figur 10.9 side 247 samt 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 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.

For rekursive funktioner i din implementation vil der være tale om *to* udgaver: den beskrevet ovenfor, samt en som gør det virkelige arbejde. 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.

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 den returnerede liste 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*[2] 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.

[2]Læs 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 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.

Hvis SDU ikke er lukket ned på tidspunktet for deadline, skal filerne *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 papirkopi per gruppe. Afleveringens sider skal sættes sammen med hæfteklamme.

Aflever materialet senest:

Tirsdag den 7. april, 2020, kl. 12:00.

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.