DM507 Algoritmer og datastrukturer

Forår 2016

Projekt, del II

Institut for matematik og datalogi
Syddansk Universitet

27. marts, 2016

Dette projekt udleveres i tre dele. Hver del har sin deadline, således at arbejdet strækkes over hele semesteret. Deadline for del II er fredag den 15. april. Projektet skal besvares i grupper af størrelse to (evt. tre, efter aftale med underviser).

Mål

Målet for del II af projektet er først at implementere datastrukturen *ordnet dictionary* (ordbog), og derefter bruge den til at sortere tal. Arbejdet kan ses som en forberedelse til elementer af 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

Der skal i Java implementeres en datastruktur, som tilbyder følgende interface:

```
public interface Dict {
    public void insert(int k);
    public int[] orderedTraversal();
    public boolean search(int k);
}
```

Nøgler er af typen `int`, og elementer består kun af nøgler (der er ikke yderligere data tilknyttet en nøgle). Metoden `search(k)` returnerer blot en boolean, som angiver om nøglen `k` er i træet. Metoden `insert(k)` indsætter nøglen `k` i træet. Metoden `orderedTraversal()` returnerer en kopi af træets elementer i et array i sorteret orden (fremfor at printe dem på skærmen som i bogens pseudo-kode).

> Implementationen skal laves ved hjælp af strukturen *binært søgetræ*, som beskrevet i Cormen et al. kapitel 12. Som det fremgår af interfacet `Dict` skal der kun implementeres indsættelse (pseudo-kode side 294), søgning (pseudo-kode side 290 eller 291), og in-order gennemløb (pseudo-kode side 288). Implementationen *skal* basere sig på denne pseudo-kode. Træet skal ikke holdes balanceret (der skal ikke bruges metoder fra kapitel 13).

Implementationen skal være i form af en Java-klasse, som kan bruges af andre programmer. Klassen skal hedde `DictBinTree`, og skal implementere interfacet `Dict`. Klassen skal have én constructor-metode ved navn `DictBinTree()`, som returnerer en ny, tom dictionary. Der vil være behov for at implementere metoder udover dem i interfacet, til internt brug i klassen.

Man skal definere en separat klasse til at repræsentere knuder i træer (man skal *ikke* bruge et array til at repræsentere træet, sådan som for en heap i del I af projektet). Et knudeobjekt skal indeholde referencer til to andre knudeobjekter (dens venste barn og højre børn), med værdi `null` hvis et barn ikke findes. Det skal også indeholde en nøgle. Objekter af typen `DictBinTree` skal være et header-objekt, som indeholder en reference til knuden, der er rod i træet, samt anden relevant global information on træet, f.eks. dets størrelse (så man kan oprette et array af den rigtige størrelse i starten af `orderedTraversal`) samt en tæller (til at holde styr på, hvor langt man er kommet i dette array under `orderedTraversal`). Dvs. objekter af typen `DictBinTree` og knudeobjekter er to forskellige slags objekter og defineres i to forskellige klasser. For hvert træ er der ét headerobjekt og mange knudeobjekter.

Bemærk at parametrene i bogens pseudo-kode er anderledes end i interfacet ovenfor. Dette skyldes at implementationsdetaljer (såsom at der findes knuder i dictionary'en) ikke skal være synlige for brugere af datastrukturen. For rekursive metoder vil der være tale om to udgaver, den officielle fra interfacet, og en intern, som gør det virkelige arbejde. Den officielle er ikke rekursiv og kalder blot den interne, og tilføjer i kaldet yderligere parametre med relevante værdier. For metoder baseret på en løkke kan den ekstra parameter blot oprettes inden løkken går i gang.

Opgave 2

Du skal implementere en sorteringsalgoritme kaldet Treesort baseret på metoderne i interfacet `Dict`. Algoritmen består i at lave gentagne `insert`'s i en dictionary, efterfulgt af et kald til `orderedTraversal`. Elementerne i det returnerede array skal så blot skrives ud.

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

Som i del I af projektet skal programmet `Treesort` 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 `Treesort` kunne kaldes således i en kommandoprompt:

```
java Treesort
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
```

Bemærk at ved hjælp af redirection[1] af standard input og output kan man i en kommandoprompt anvende programmerne (helt uden at ændre i dem) på filer også:

```
java Treesort   < inputfile   > outputfile
```

Det er vigtigt at I afprøver ovenstående i en kommandoprompt, da programmerne skal kunne testes automatisk. Man må af samme grund heller

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

ikke i sin kildekode have `package` statements, eller organisere sin kode i en folderstruktur[2].

Formalia

Du skal kun aflevere dine Java source-filer. 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 15. april, 2015, 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 som sådan. Man lærer desuden heller ikke noget.

[2]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.)