DM507 Algoritmer og datastrukturer

Forår 2012

Projekt, del II

Institut for matematik og datalogi
Syddansk Universitet

15. marts, 2012

Dette projekt udleveres i tre dele. Hver del har sin deadline, således at afleveringerne, og dermed arbejdet, strækkes over hele semesteret. Deadline for del II er onsdag den 25. april. Projektet skal som udgangspunkt besvares i grupper af størrelse to. Individuelle besvarelser er tilladt, men tilskyndes ikke.

Mål

Målet for del II af projektet er at implementere rød-sorte træer, samt bevise en øvre grænse for mængden af rebalancering. Med hensyn til updates skal der kun ses på indsættelser, ikke sletninger.

Rød-sorte træer

Rød-sorte træer er grundigt beskrevet i Cormen et al., kapitel 13. Bemærk at pseudo-koden i dette kapitel er baseret på en implementation med en sentinel-knude T.nil til at repræsentere træets NIL-pointere (bladene samt rodens forælder), som illustreret i Figur 13.1 (b) på side 310. Denne knude er altid sort. Vi antager også denne implementation her.

Opgaver

Opgave 1

Der skal laves en Java-implementation, baseret på bogens pseudo-kode, af rød-sorte træer, som indeholder metoderne Search (pseudo-kode side 290 eller 291, skal justeres til at bruge T.nil i stedet for NIL), Insert (pseudo-kode side 315 (og siderne 316 og 313)), samt InorderTraversal (pseudo-

kode side 288, skal justeres til at bruge T.nil i stedet for NIL)). Der skal ikke implementeres DELETE eller yderligere metoder.

Det antages at nøgler er af typen `int` (så man ikke behøver bruge f.eks. generics i Java), og at elementer blot består af nøgler (der er ikke yderligere data tilknyttet en nøgle). Dette vil være tilstrækkeligt for den senere anvendelse i del III. Implementationen skal være i form af en Java-klasse, som kan bruges af andre programmer. Klassen skal hedde `RBT`, og skal implementere flg. interface:

```
public interface RBTree {
    public boolean search(int k);
    public void insert(int k);
    public int[] inorderTraversal();
    public boolean isRedBlack();
}
```

Metoden `search(k)` returnerer blot en boolean som angiver om nøglen `k` er i træet. Metoden `insert(k)` indsætter nøglen `k` i træet. Metoden `inorderTraversal()` returnerer en kopi af træets elementer i et array (i sorteret orden) fremfor at printe dem på skærmen som i bogens pseudo-kode. Metoden `isRedBlack()` checker om et givet `RBTree` overholder (de vigtigste af) kravene 1–5 på side 308. Denne metode er bla. anvendelig til fejlfinding under implementationsprocessen. Metoden skal baseres på algoritmen beskrevet i pseudo-kode i appendikset nedenfor. Man skal ikke bevise noget om korrekthed eller køretid for denne algoritme.

Opgave 2

En indsættelse består af en søgning og indsættelse af ny knude (linie 1–16 i pseudo-kode side 315), samt en efterfølgende rebalancering af træet (pseudo-koden side 316).

Vi ønsker i denne opgave at bevise, at hvis man starter med et tomt rød-sort træ og laver n indsættelser, da er den samlede mængde rebalanceringsarbejde (dvs. arbejde lavet af pseudo-koden side 316 (og 313)) i alt $O(n)$. Denne viden vil vise sig brugbar i del III af projektet. Bemærk at ovenstående øvre grænse er stærkere end den simple vurdering at n indsættelser hver højst kan lave rebalanceringsarbejde svarende til stien mod roden, hvilket blot giver en øvre grænse på $O(n \log n)$.

For et rød-sort træ T lader vi $\psi(T)$ være antallet af knuder i træet som er sorte og har to røde børn. For eksempel er $\psi(T) = 1$ for træet i Figur 13.1 i Cormen et al. (side 310). Vi skal se på hvordan $\psi(T)$ udvikler sig, når T ændrer form under indsættelser og efterfølgende rebalanceringer.

Som det fremgår af diskussionen i afsnit 13.3 af Cormen et al. vil while-løkken i pseudo-koden side 316 løbe nul eller flere gange gennem Case 1, og derefter højst een gang gennem Case 2 og højst een gang gennem Case 3, hvorefter den stopper.

Bemærk følgende observationer:

- Selve indsættelsen (at erstatte et tomt undertræ med en ny knude, uden rebalancering) kan højst øge $\psi(T)$ med een.
- Under rebalancering vil Case 2 ikke ændre $\psi(T)$ (følger af Figur 13.6 og den tilhørende figurtekst).
- Under rebalancering kan Case 3 højst øge $\psi(T)$ med een (følger af Figur 13.6 og den tilhørende figurtekst).

Opgaven består af nedenstående delopgaver i) til v).

i) Argumentér for at hvis en rebalancering starter med k gange Case 1, da vil disse sænke $\psi(T)$ med mindst $k-1$ (brug Figur 13.5 og den tilhørende figurtekst).

Vi ser nu på situationen hvor man starter med et rød-sort træ T_{start} og laver n indsættelser, resulterende i et træ T_{slut}, og ønsker at vurderere den samlede mængde rebalanceringsarbejde. Lad k_i betegne det antal gange Case 1 udføres ved rebalancering efter den i'te indsættelse.

ii) Argumentér for at der højst n gange i alt udføres Case 3.

iii) Argumentér for at

$$\psi(T_{\text{slut}}) \leq \psi(T_{\text{start}}) + 2n - \sum_{i=1}^{n}(k_i - 1)$$

ved at bruge ovenstående observationer og udsagn.

Vi ser nu på situationen hvor de n indsættelser starter med et tomt træ. Så gælder naturligvis $\psi(T_{\text{start}}) = 0$. For alle træer T er $0 \leq \psi(T)$, så vi har $0 \leq \psi(T_{\text{slut}})$. Heraf følger fra sidste ulighed ovenfor at

$$0 \leq 2n - \sum_{i=1}^{n}(k_i - 1).$$

iv) Argumentér for at det heraf følger at $\sum_{i=1}^{n} k_i \leq 3n$.

v) Argumentér for at den samlede mængde rebalanceringsarbejde under de n indsættelser er $O(n)$, hvis man starter med et tomt træ.

Formalia

Lav en rapport, som indeholder dine svar på opgave 1 og 2 ovenfor. Koden for opgave 1 skal være passende kommenteret, skal inkluderes i rapporten som bilag, og eventuelle ikke-trivielle aspekter af implementeringen skal diskuteres i rapportens hoveddel. Der skal afleveres rapporten i pdf-format, samt Java-implementationen som separate filer (dvs. udover deres inklusion på tryk i rapporten). Husk at skrive navnene på personerne i gruppen på forsiden af rapporten.

Materialet afleveres i Blackboard med værktøjet "SDU Assignment" (ikke at forveksle med "Assignment hand in", som er et andet afleveringsværktøj i Blackboard). Det kan findes under "Tools" i menuen i kursussiden i Blackboard. Menuen findes ved at klikke på det lille "dobbelt-firkant"-ikon i øverste halvdel af venstre kant af kursussiden i Blackboard (om nødvendigt maksimer det fremkomne vindue).

Aflever materialet senest:

Onsdag den 25. april, 2012, kl. 23:59.

Appendiks

Algoritmen IsRedBlack(T) nedenfor returnerer **true** hvis og kun hvis T opfylder kravene 2, 4, og 5 på side 308 i Cormen et al. (det antages at krav 1 *er* opfyldt, dvs. at der kun bruges farverne sort og rød, og krav 3 bliver automatisk opfyldt via brugen af en sentinel-knude med farven sort).

Man skal ikke bevise noget om korrekthed eller køretid for denne algoritme.

```
IsRedBlack(T)
    return {T.root.color == black and
                BlackHeight(T, T.root) ≥ 0 and
                (not TwoRedsInRow(T, T.root))}

BlackHeight(T, v)
    if v == T.nil
        return 0
    else
        h1 = BlackHeight(T, v.left)
        h2 = BlackHeight(T, v.right)
        if h1 ≠ h2 or h1 == −1
            return −1
        elseif v.color == black
            return h1 + 1
        else
            return h1


TwoRedsInRow(T, v)
    if v == T.nil
        return false
    elseif {v.color == red and
                (v.left.color == red or v.right.color == red)}
        return true
    else
        return TwoRedsInRow(T, v.left) or TwoRedsInRow(T, v.right)
```