SI-möte #4, Algoritmer och datastrukturer

Elias Castegren
elca7381@student.uu.se

3 maj 2010

Begrepp

i) *(Favorit i repris)* Vad menas med ett *binärt sökträd*? Vad är höjden hos ett (obalanserat) binärt sökträd i sämsta och genomsnittliga fall? Hur ser trädet ut i de olika fallen?

ii) Vad innebär det att rotera ett träd?

iii) Vad är ett *AVL-träd*? Hur fungerar balanseringen av ett sådant träd? När är det användbart med AVL-träd (eller andra balanserade sökträd)?

iv) Vad menas med *traversering* när man pratar om träd? Vad blir resultatet av en inorder-traversering av ett binärt sökträd?

1.
Sätt in följande element (i given ordning) i ett tomt AVL-träd:

$$5, 3, 8, 2, 4, 1, 7, 9, 6$$

Rita upp hur hela trädet ser ut efter varje insättning. Skriv också ut varje nods balansfaktor.

2.
Utgå från AVL-trädet i uppgift 1 och ta bort nedanstående element (i given ordning):

$$1, 7, 9, 5, 3, 2$$

Rita upp hur hela trädet ser ut efter varje borttagning. Skriv också ut varje nods balansfaktor.

(Fortsätt gärna att sätta in och ta bort element om du vill öva mer på AVL-träd)

3.
Skriv en implementation av AVL-träd i ML-kod. Du kan utgå från följande datatypsdeklaration

```
datatype 'a tree = Leaf |
                   Node of int * int * 'a * 'a tree * 'a tree;
```

där `Node(key, h, d, L, R)` är en nod i ett binärt sökträd med nyckeln `key`, höjden `h`, data `d` och vänster och höger underträd `L` och `R`.

Skriv en funktion `insert(k, d, t)` som med logaritmisk komplexitet skapar (eller ersätter) noden med nyckel `k` och data `d` i AVL-trädet `t` så att `t` förblir ett AVL-träd. Fundera på vilka hjälpfunktioner du kan komma att behöva och skriv funktionsspecifikationer innan du skriver koden. Fundera också på hur man kan använda datafältet `h` för att beräkna en nods balansfaktor.

4.
Skriv en funktion `treeHeight(t)` som returnerar höjden av ett binärt träd `t` (se definitionen nedan), dvs det högsta antalet noder i en väg från roten till ett löv. Bestäm sedan tidskomplexiteten, t.ex. genom att hitta en rekursiv formel och bestämma dess slutna form.

```
datatype 'a tree = Void | Node of 'a * 'a tree * 'a tree
```

5.
Gör en preorder-, inorder- och postorder-traversering av AVL-trädet i uppgift 1 (innan elementen togs bort i uppgift 2). Skriv sedan en ML-funktion `inOrder(t)` som returnerar en lista med noderna i ett binärt träd `t` (se ovan) i den ordning som de besöks vid en inorder-traversering. Vad är tidskomplexiteten för `inOrder`? Hur kan man ändra i koden för att få en pre- eller postorder-traversering? Påverkas komplexiteten?

Lycka till!