SI-möte #8, Programkonstruktion och Datastrukturer

Elias Castegren
elca7381@student.uu.se

25 januari 2011

Begrepp

i) Vad är en *stack*? Vilka är de två grundläggande stackoperationerna?

ii) Vad menas med en *FIFO-kö*? Hur skiljer sig en kö från en stack?

iii) Vad är komplexiteten för lägga till eller ta bort ett element från en stack respektive en kö om man representerar dem med enkelt länkade listor (vanliga listor i ML)?

iv) Vad menas med ett *binärt sökträd*? Vad är höjden av ett binärt sökträd i sämsta och bästa fall? Hur ser trädet ut i de olika fallen?

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

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

Övningar

1.
Utför följande insättningar och borttagningar till en (tom) stack. Skriv ut hela stackens utseende i varje steg.

i) Pusha $9, 5, 1, 2$ och 3 ii) Poppa två element iii) Pusha $4, 1, 3, 7$ iv) Poppa ett element

Utför samma operationer fast till en (FIFO-)kö, alltså med enqueue och dequeue istället för push och pop. Skriv ut hela köns utseende i varje steg.

2.
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.

3.
Utgå från AVL-trädet i uppgift 2 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)

4.
Skriv en funktion `treeHeight(t)` som returnerar höjden av ett binärt träd `t` (se definitionen nedan), dvs den längsta vägen 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 2 (innan elementen togs bort i uppgift 3). 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!