Algorithms and Datastructures assignment 3

Thomas Broby Nielsen (xlq119)
Tobias Overgaard (vqg954)
Christian Buchter (zvc154)

2. juni 2015

Indhold

Task 1

```
get-kth-key(x,k)
if k < 0 or k > x.max
   return NILL
else
       if k > x.left.size+1
          return get-kth-key(x.right,k-x.left.size)
       else if k < x.left.size
          return get-kth-key(x.left,k)
       else if k = x.left.size+1
          return x.key
```

task 2

base cases: size=0
size=1
hypotese: hvis x.size=i, så gælder koden virker for alle sizes < i

cases:
hvis k > x.left.size+1. Så bliver k mindre og size bliver mindre. k=k-x.left.size og x=x.right
hvis k < x.left.size. Så forbliver k den samme værdi, og size bliver mindre. x=x.left
hvis k = x.left.size+1. Så returneres x.key og algortimen er færdig.
Fordi at size altid bliver mindre, medmindre algoritmen er færdig, og da vores hypotese siger at algoritmen virker for alle sizes < i virker algoritmen.

task 3

```
sized-Left-Rotate(T,x)
1 y.size=x.size
...
14 x.size=x.right.size+x.left.size+1
------
everything inbetweeen is normal.
```

This way the size of x will not change when rotated.

Task 4

```
RB-INSERT(T, z)
1 y = T.nil
2 x = T.root
3 while x =! T.nil
4         y = x
6         x.size = x.size + 1
5         if z.key < x.key
7            x = x.left
8         else x = x.right
...
17 z.color = RED
18 z.size=1
---------
everything from here on is normal.
```

[1]

[1] Se side 315 på "RB-Insert(T,z)"pseudo code

Task 5

Siden at vi kun har tilført to linjer kode, til den originale RB-INSERT, som arbejder i konstant tid, på linje 6 med indførelsen af "$x.size = x.size + 1$"og igen på linje 8 med $x.size = x.size + 1$, vil det ikke påvirke algoritmens originale køretid på O(lg n), som står skrevet i bogen[2].
Ændringerne som vi har lavet til den originale version af RB_delete, ændrer ikke køretiden. Dette er fordi at vores ændringerne kun ændrer size når den aligevel var i noden.
Query/get-kth-key kører i O(lg n) tid, da den går et trin ned i træet hver gang den bliver kaldt. Der er $lg(n)$ niveauer. Der er lg(n) niveauer fordi at for er skal bruges 2*det sidste niveaus knuder for alt fylde et nyt niveau ud. Deri lg(n) niveauer for n knuder.

Task 6

siden insert tager O(lg n) tid og siden delete tager O(lg n) tid og query tager O(lg n) så tager en arbitrær sekvens af disse 3 funktionskald i værste tilfælde
O(y lg n + z lg n + x lg n)

[2]se side 322 "Analysis"