DMA Opgave 8

Hold 1

Aditya Fadhillah (hjg708)

17. december 2021

1

Vis at operationerne kan gives følgende tidsgrænser:
* $Initialize(F)$ tager amortiseret tid $O(nlog(n))$
* $Remove(F, x, y)$ tager amortiseret tid $O(1)$

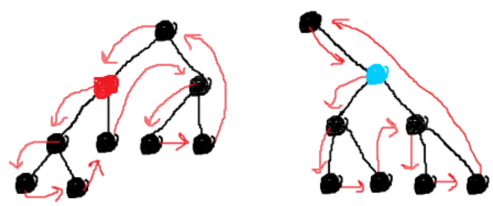
Ved brug af regnskabsmetoden kan jeg vise at $Initialize(F)$ tager amortiseret tid $O(nlog(n))$. Jeg antager at ved initialiseringen ligges der $log(n)$ dollars for hver knude, da der er n knuder, vil $Initialize(F)$ tager amortiseret tid $O(nlog(n))$. Jeg antager også at hver knude x betaler 1 dollar hver gang der bliver fjernet en kant fra det træ som indeholder x og hvor x er i det mindste af de to resulterende træer, det vil sige at $Remove(F, x, y)$ tager amortiseret tid $O(1)$. Jeg kan argumenterer for at en knude x i et træ Tx altid har mindst $log(|Tx|)$ dollars, ved at kigge tilbage på hvor mange dollars jeg lagde for hver knude. Da jeg antog at der lægges $log(n)$ dollars for hver knude, vil der naturligvis mindst være $log(|Tx|)$ dollars i et træ Tx, da $|Tx|$ betegner antallet af knuder i et træ Tx.

Som et eksempel har jeg taget et træ T med 16 knuder, idet med at der er tale om binær logaritme, vil hele træet have 4 dollars, da $log(16) = 4$. Træet indeholder knude x. Når man kalder $Remove$ på træet vil den del med knude x har halvdelen af knuderne af træet T, dvs. 8 knuder, igen idet med at der tale om binært logaritmem, vil træet Tx have 3 dollars, da $log(8) = 3$. Her ser vi at $Remove$ koster 1 dollars.

2

Lav en figur som illustrerer hvordan funktionen virker.

Man kan lave SmallestTree(x,y) ved at køre begge træer sammetidig med Binary Tree Traversal. Med Preorder Binary Tree Traversal vil de to træer Tx og Ty starter på deres rod, hvor de derefter kører hele deres knuder igennem, indtil de igen rammer deres rod. Idet med at de to træer T_x og T_y gennemløber parallelt kan det tænkes at det vil returnerer x hvis T_x når dens rod først eller hvis de to træer når deres rod på sammetid, og det vil returnerer y hvis T_y når dens rod først.



Til venstre Ty med knude y (rød), til højre Tx
med knude x (blå)

På figuren oven på ser man at T_x vil komme hurtigere tilbage til dens rod, dvs. SmallestTree(x,y) vil returnerer x.

3

Skriv pseudokode til funktionerne $Remove(F, x, y)$ og $SameTree(F, x, y)$, så $SameTree$ tager konstant tid i værste fald og de amortiserede grænser ovenfor gælder.

```
SameTree(F, x, y)
    if x.treeID = y.treeID
        return TRUE
    else
        return FALSE
```

```
Remove(F, x, y)
    if x == x.parent OR y.parent
        if x == NIL
            node <- x
        else
            if y == NIL
                node <- y
    else
        if y == NIL
            node <- y
        else
            if x == NIL
                node <- x
    for node in SmallestTree(x,y)
        node.treeID <- new.node.treeID
```

Jeg antager at knude x og knude y er på samme træ, og at en af knuderne altid være parent af den anden. Det tager konstant tid til at finde ud om x er parent af y og omvendt. Dvs. det vil tage $O(1)$ for at sammeligne x og y. Det vil derefter tage $O(min|Tx|, |Ty|)$ at ændre treeID af de elementer ind i de mindste træ. Så i worst case vil min algoritme tage $O(min|Tx|, |Ty|)$, som stemmer med de tidligere grænser.