

# DMA Opgave 8

Hold 1

Aditya Fadhillah (hjl708)

8. december 2021

# 1

**Vis at operationerne kan gives følgende tidsgrænser:**

\* *Initialize*(*F*) tager amortiseret tid  $O(n \log(n))$

\* *Remove*(*F*, *x*, *y*) tager amortiseret tid  $O(1)$

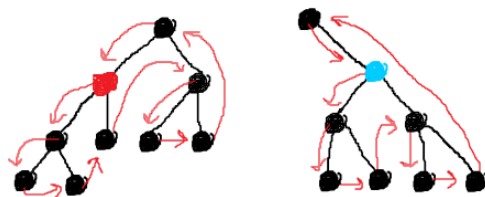
Ved brug af regnskabsmetoden kan jeg vise at *Initialize*(*F*) tager amortiseret tid  $O(n \log(n))$ . Jeg antager at ved initialiseringen liggess der  $\log(n)$  dollars for hver knude, da der er  $n$  knuder, vil *Initialize*(*F*) tage amortiseret tid  $O(n \log(n))$ . Jeg antager også at hver knude *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 argumentere for at en knude *x* i et træ  $T_x$  altid har mindst  $\log(|T_x|)$  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  $\log(|T_x|)$  dollars i et træ  $T_x$ , da  $|T_x|$  betegner antallet af knuder i et træ  $T_x$ .

Som et eksempel har jeg taget et træ  $T$  med 16 knuder, idet med at der er tale om binær ligaritme, 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  $T_x$  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 samtidigt med Binary Tree Traversal. Med Preorder Binary Tree Traversal vil de to træer  $T_x$  og  $T_y$  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 returnere *x* hvis  $T_x$  når dens rod først eller hvis de to træer når deres rod på sammetid, og det vil returnere *y* hvis  $T_y$  når dens rod først.



Til venstre  $T_y$  med knude *y* (rød), til højre  $T_x$  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 returnere *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.

---

```
1 SameTree(F, x, y)
2   if x.treeID = y.treeID
3     return TRUE
4   else
5     return FALSE
```

---

```
1 Remove(F, x, y)
2   for SameTree(F, x, y) = TRUE
3     if x.pointerParent = y
4       x = x.parent
5       y.pointerToX == NULL
6       for nodes in SmallestTree(x,y)
7         nodes.treeID1 <- nodes.treeID2
8     else y.pointerParent = x
9       y = y.parent
10      x.pointerToY == NULL
11      for nodes in SmallestTree(x,y)
12        nodes.treeID1 <- nodes.treeID2
13  for SameTree(F, x, y) = FALSE
14    x = x.parent
15    nodes.pointerToX == NULL
16    for nodes in SmallestTree(x,y)
17      nodes.treeID1 <- nodes.treeID2
18    y = y.parent
19    nodes.pointerToY == NULL
20    for nodes in SmallestTree(x,y)
21      nodes.treeID3 <- nodes.treeID4
```

---

Jeg antager hvis knude x og knude y er på samme træ vil en af knuderne altid være parent af den anden. Og hvis knude x og knude y ikke er på samme træ, vil x og y være parent af det nyt træ, når man kalder Remove på det oprindeligt træ.