DMA — Ugeopgave 8i

Helga Rykov Ibsen <mcv462>

4. januar 2022

1

Vi starter med at have følgende udsagn:

`Initialize(F)` kører i $O(n)$ tid
`Remove(F,x,y)` kører i $O(min\{|T_x|, |T_y|\})$ tid

Vi ønsker at vise at:

`Initialize(F)` kører i $O(n\text{lg}n)$ amortiseret tid
`Remove(F,x,y)` kører i $O(1)$ amortiseret tid

Dette kan vises på to måder:

(1) Vi starter med at lave den normale kørtidsanalyse.
Lad os definere at `Initialize` lægger lg n dollars på hver knude. Da der er n knuder i et træ, kan dens amortiserede køretid udtrykkes som den oprindelige kørtid plus det samlede antal af dollars lagt på et træ (dvs. $n \cdot \text{lg}n$) :

$$O(n) + n\text{lg}n = O(n\text{lg}n)$$

Lad os nu definere at `Remove` tager én dollar fra hver træknude i det mindste af de to træer der bliver dannet efter at man har kaldt `Remove`. Dermed har vi at dens amortiseret kørtid er dens oprindelige kørtid minus det samlede antal af dollars i det mindste træ ($1\text{dol} \cdot min(|T_x|, |T_y|)$). Vi får dermed som ønsket, at amortiseret køretid for `Remove` er konstant:

$$O(min\{|T_x|, |T_y|\}) - min(|T_x|, |T_y|) = O(1)$$

(2) Vi mangler nu at vise analysens ovenfor gyldighed, altså at der aldrig findes en træknude med et negativt antal dollars.
Dette kan bevises ved matematisk induktion.

1. Lad udsagnet være:

"en træknude x har mindst $\text{lg}(|T_x|)$ dollars"

Dette kan også formuleres som en funktion af x, $\$(x)$:

$$\$(x) \geq \lg(|T_x|)$$

Vi ønsker at vise at udsagnet 1. er sandt for vilkårlige x under gentagne kald til `Remove`.

2. **[Basistrinnet]** Vi vil gerne vise at udsagnet 1. er sand for et træ T ved initialiseringen, dvs. inden det første kald af `Remove`. Her er $T_x = T$ og dermed:

$$\$(x) = \lg(n) = \lg(|T|) = \lg(|T_x|)$$

3. **[Induktionstrinnet]** Vi antog at ved initialiseringen er der mindst $\lg(|T_x|)$ dollars på en vilkårlig knude i træet T før det bliver delt op i to træer. Vi vil vise at det samme gælder efter at der blev kaldt til `Remove`. Dvs. vi skal vise at udsagnet 1. er sandt for de to tilfælde: (1) hvor T_x er det største træ og (2) hvor T_x er det mindste træ efter opdelingen.

   a) I det første scenarie betaler træet T_x ikke noget da det ifølge difinitionen kun det mindste træ der betaler, så det gælder:

   $$\$(x') = \$(x) \geq \lg(|T|) \geq \lg(|T_x|)$$

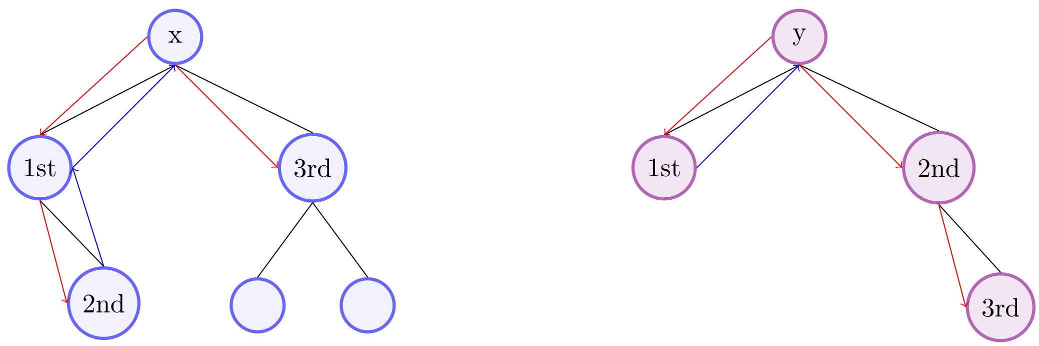
   b) I det andet scenarie er træet T_x mindst og betaler 1 dollar per hver knude, så det gælder:

   $$\begin{aligned} \$(x\prime) &= \$(x) - 1 \\ &\geq \lg(|T|) - 1 \\ &\geq \lg(2 \cdot |T_x|) - 1 \quad (\text{fordi} \quad |T_x| \geq |T/2|) \\ &\geq \lg(2) + \lg(|T_x|) - 1 \\ &\geq 1 + \lg(|T_x|) - 1 \\ &\geq \lg(|T_x|) \end{aligned}$$

4. **[Konklusion]** Vi har vist at basistrinnet hvor udsagnet 1. er sandt for et træ T ved initialiseringen. Vi har vist induktionstrinnet, hvor udsagnet 1. også er sand for T_x efter kaldet til `Remove`. Så er altså udsagnet 1. sand for alle træknuder x efter gentagne kald til `Remove`. Da et træ med n knuder har $n-1$ kanter, er det kun lovligt at kalde `Remove` $n-1$ gange.

2

Lad x og y være rodknuder i to forskellige træer T_x og T_y i F. Man kan lave en funktion `SmallestTree` som løber igennem de to træer og tæller deres knuder sideløbende. Når der ikke er flere knuder i et af træerne, returnerer den rodknuden af det træ der har færrest knuder. Betragt en visualisering af denne funktion nedenunder med to binære træer — et blåt træ og et lilla træ. `SmallestTree` besøger hvert af dem i den rækkefølge som er markeret inde i knuderne.



Figur 1: Binære træer, gennemløb `SmallestTree`. Hver knude indeholder den rækkefølge `SmallestTree` besøger den.

Som det kan ses på billedet, `SmallestTree` stopper med at tælle knuder når den når til den tredje knude og i og med at der ikke er flere knuder i det lilla træ, returnerer den rodknuden y af det mindste træ, i dette tilfælde T_y.

Køretiden for funktionen illustreret på billedet svarer til $O(|T_y|)$, eller sagt på en mere generel måde, $O(min\{|T_x|, |T_y|\}$. Antallet af skridt som algoritmen udfører for at løbe igennem træerne svarer til antallet af de røde pile, altså tre i alt. De blå pile viser blot at funktionen returnerer, hvilket koster ingen tid.

3

Præmissen siger at hver knude x indeholder et ID, $x.treeID$ på det træ som indeholder x, dvs. det mindste træ T_x, og at ved kaldet til `Remove` bliver der ændret knudernes ID'er i det mindste træ når en kant slettes i T.

Lad t være antallet af træer, hvor træID'et kan have en værdi fra mængden $\{1, ..., t\}$.

Funktionen ∀nodes gennemløber alle knuderne i træet:

Algorithm 1 Remove(F,x,y)

```
newID = F.size + 1
F.size = newID
SmallTree = SmallestTree(x,y)
y.x = NIL                          ▷ y's x-pointer
x.parent = NIL                     ▷ x's y-pointer
for SmallTree.∀nodes do
    node.treeID = newID
```

Hvis vi antager at enhver knude indeholder et $treeID$ som er unik for netop det træ den hører til, returnerer `SameTree` TRUE, hvis to knuder har det samme $treeID$ og ellers FALSE:

Algorithm 2 SameTree(F,x,y)

```
return x.treeID = y.treeID
```