Balanserte søketrær

IN2010 – Algoritmer og Datastrukturer

Lars Tveito og Daniel Lupp

Høsten 2020

Institutt for informatikk, Universitetet i Oslo
larstvei@ifi.uio.no
danielup@ifi.uio.no

Oversikt uke 36

Oversikt uke 36

- Denne uken konsentrerer vi oss om binære søketrær, som har
  - $O(h)$ tid på *innsetting, sletting og oppslag*
  - der h er høyden på treet
- Hvis vi klarer å holde h tilstrekkelig liten
  - så får vi $O(log(n))$ på samtlige operasjoner
  - der n er antall noder i treet
- Vi skal se på to teknikker
  - AVL-trær
  - Rød-svarte trær

AVL-træer

AVL-trær

- AVL-trær oppfyller de samme egenskapene som ordinære binære søketrær
- I *tillegg* må de oppfyle følgende egenskap:
  - for hver node i et AVL-tre, så må høydeforskjellen på venstre og høyre subtre være mindre eller lik 1
- Denne invarianten må opprettholdes ved innsetting og sletting
  - (oppslag er helt uforandret)

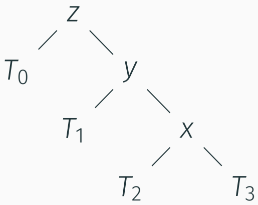
Høyde

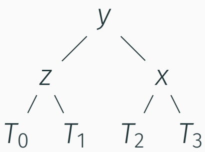
- Fordi vi ofte vil referere til høyden i treet, utvider vi nodene i treet
- Hvis v er en node i et AVL-tre, så gir
  - v.**element** dataen som er lagret i noden
  - v.**left** venstre barn av v
  - v.**right** høyre barn av v
  - v.**height** høyden til v
- Husk at høyden til et tomt tre er definert som -1
  - og at høyden til en node er én mer enn sitt høyeste subtre
- Ved innsetting og sletting må vi vedlikeholde høydene
- Vi antar at vi har en prosedyre **Height** som:
  - returnerer -1 dersom den får **null** som input
  - returnerer v.**height** for alle noder v

Overordnet idé

- Vi bruker metodene for sletting og innsetting fra ordinære binære søketrær
- Etter operasjonen er utført, balanserer vi hver node lokalt fra der operasjonen ble utført og opp til roten (hvis det er nødvendig)
    - Vi balanserer når høydeforskjellen mellom venstre og høyre subtre er mer enn 1
- For å balansere en node, vil vi anvende *rotasjoner*
- Underveis må vi passe på å oppdatere høydene
- Husk at AVL innsetting og sletting bare fungerer på AVL-trær!
    - Altså kan vi anta at treet ikke har høydeforskjeller større enn 1
    - Ved én innsetting eller sletting i et AVL-tre vil vi bare forårsake en midlertidig høydeforskjell på 2

Rotasjoner – venstrerotasjon





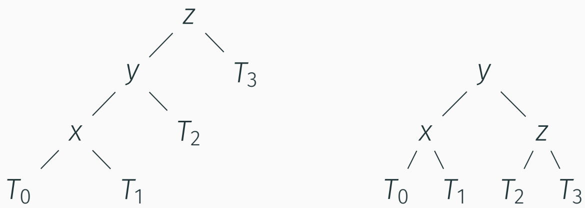
Algorithm 1: Venstrerotasjon av et Binærtre

Input: En node z

Output: Roter treet til venstre, slik at y blir den nye roten

Procedure `LeftRotate`(z)

- $y \leftarrow z.\text{right}$
- $T_1 \leftarrow y.\text{left}$
- $y.\text{left} \leftarrow z$
- $z.\text{right} \leftarrow T_1$
- $z.\text{height} \leftarrow 1 + max(\text{Height}(z.\text{left}), \text{Height}(z.\text{right}))$
- $y.\text{height} \leftarrow 1 + max(\text{Height}(y.\text{left}), \text{Height}(y.\text{right}))$
- **return** y



Algorithm 2: Høyrerotasjon av et Binærtre

Input: En node z

Output: Roter treet til høyre, slik at y blir den nye roten

Procedure `RightRotate`(z)

$y \leftarrow z$.**left**
$T_2 \leftarrow y$.**right**

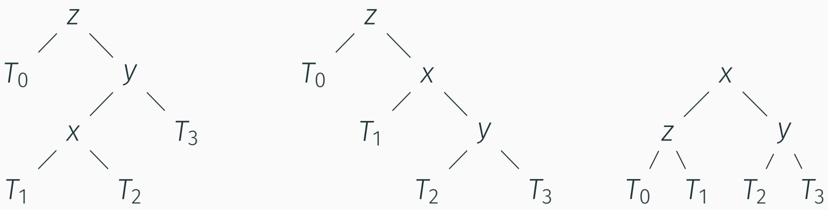
y.**right** $\leftarrow z$
z.**left** $\leftarrow T_2$

z.**height** $\leftarrow 1 + max($**Height**$(z$.**left**$)$, **Height**$(z$.**right**$))$
y.**height** $\leftarrow 1 + max($**Height**$(y$.**left**$)$, **Height**$(y$.**right**$))$

return y

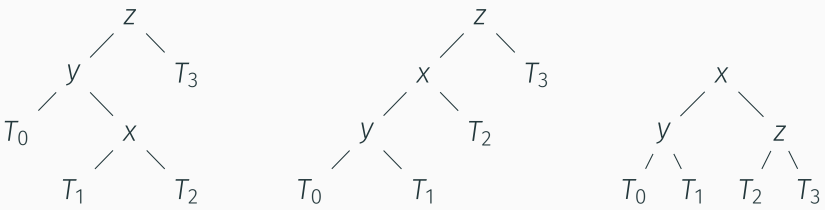
Rotasjoner – doble rotasjoner

`RightRotate`(y) `LeftRotate`(z)



Rotasjoner – doble rotasjoner

`LeftRotate`(y) `RightRotate`(z)



Balansefaktor

Algorithm 3: Balansefaktoren av en node

Input: En node v

Output: Returner høydeforskjellen på v sitt venstre- og høyrebarn

```
Procedure BalanceFactor(v)
    if v == null then
        return 0
    return height(v.left) − height(v.right)
```

- En hjelpeprosedyre som sier hvor vestre- eller høyretungt v er
- 0 betyr at v er balansert
- Et positivt tall betyr at treet er venstretungt
- Et negativt tall betyr at treet er høyretungt

Balansering

Algorithm 4: Balansering av et AVL-tre

Input: En node *v*

Output: En balansert node

```
Procedure Balance(v)
    if BalanceFactor(v) < −1 then
        if BalanceFactor(v.right) > 0 then
            v.right = RightRotate(v.right)
        return LeftRotate(v)
    if BalanceFactor(v) > 1 then
        if BalanceFactor(v.left) < 0 then
            v.left = LeftRotate(v.left)
        return RightRotate(v)
    return v
```

Innsetting

Algorithm 5: Innsetting i et AVL-tre

Input: En node v og et element x

Output: En oppdatert node v der en node som inneholder x er en etterkommer av v

```
Procedure Insert(v, x)
    if v = null then
        v ← new Node(x)
    else if x < v.element then
        v.left ← Insert(v.left, x)
    else if x > v.element then
        v.right ← Insert(v.right, x)
    v.height ← 1 + max(Height(v.left), Height(v.right))
    return Balance(v)
```

Sletting

Algorithm 6: Sletting i et AVL-tre

Input: En node v og et element x

Output: Dersom x forekommer i en node u som en etterkommer av v, fjern u.

```
Procedure Remove(v, x)
    if v = null then
        return null
    if x < v.element then
        v.left ← Remove(v.left, x)
    else if x > v.element then
        v.right ← Remove(v.right, x)
    else if v.left = null then
        v ← v.right
    else if v.right = null then
        v ← v.left
    else
        u ← FindMin(v.right)
        v.element ← u.element
        v.right ← Remove(v.right, u.element)
    end
    v.height ← 1 + max(Height(v.left), Height(v.right))
    return Balance(v)
```

Rød-svarte trær

Rød-svarte trær

- Rød-svarte trær er, i likhet med AVL-trær, *balanserte* binære søketrær
- Likhetene mellom rød-svarte- og AVL-trær er:
  - De er begge selvbalanserende binære søketrær
  - De har begge $O(log(n))$ på innsetting, sletting og oppslag
  - De anvender begge rotasjoner for å bevare et krav om balanse
- De store forskjellene mellom rød-svarte- og AVL-trær er:
  - Rød-svarte trær har *svakere* krav om balanse enn AVL-trær
  - Rød-svarte trær bruker mindre minne, siden vi ikke trenger å lagre høydene
  - Rød-svarte trær bruker færre rotasjoner

HVEM ER BEST!?

- Rød-svarte trær er raskere enn AVL-trær når innsetting og sletting forekommer ofte, til sammenligning med oppslag
  - Dette er fordi rød-svarte trær trenger færre rotasjoner
- AVL-trær er raskere enn rød-svarte trær når oppslag forekommer ofte, til sammenligning med innsetting og sletting
  - Dette er fordi AVL-trær er mer balanserte

Invarianter for rød-svarte trær

1. Hver node fargelegges *rød* eller *svart* (derav navnet)
2. Roten til treet er svart
3. En rød node kan ikke ha et rødt barn
4. Hver gren fra roten til et tomt tre (eller **null**) inneholder *like mange svarte noder*

Intuisjon

- Verste tilfellet (med hensyn til balanse) for et rødsvart tre er at vi har
  - en gren med bare svarte noder
  - en annen gren med annenhver svarte og røde noder
- Da har vi en gren som er dobbelt så lang som en annen!
  - Men dobbelt så langt er ikke så mye lenger
  - Så vi bevarer $O(log(n))$ på innsetting, sletting og oppslag