Algoritmer och Datastrukturer

—

Visualisering av träd-rotationer

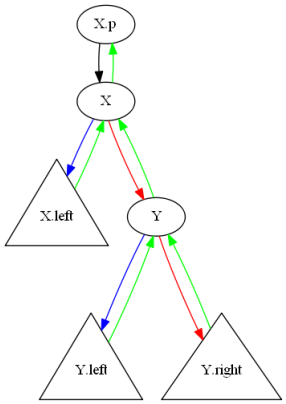
Joakim Ståhle-Nilsson
Blekinge Tekniska Högskola

2021vt, lp4

Syftet med detta dokument är att ge en visuell demonstration av hur rotationer i träd går till steg för steg. Detta för att vissa tycker det är lättare att förstå processen om det går att visualisera förloppet, och det blir lätt svårhittade fel i rotationer om man inte har förståelse för processen. Pseudokoden från boken ser ut på följande sätt.

```
Left-Rotate(T, x)
1.    y = x.right
2.    x.right = y.left
3.    if y.left != T.nil
4.       y.left.p = x
5.    y.p = x.p
6.    if x.p == T.nil
7.       T.root = y
8.    elseif x == x.p.left
9.       x.p.left = y
10.   else x.p.right = y
11.   y.left = x
12.   x.p = y
```

Vårt basfall ser ut som följande.



I grafen så ser ni ett antal noder, subträd, och länkar. De runda är noder, och trianglarna är godtyckliga subträd. Alla är markerade med namn baserat på vad de är innan vi börjar rotationsprocessen. Noden `X` är noden som vi ska rotera och i detta fallet ska vi göra en vänster rotation där vi har ett krav på att dess högerbarn, Y i detta fallet, inte ska vara en nilnod. Sedan har vi tre godtyckliga subträd, markerade som trianglar i grafen, som kan vara allt från en nilnod till ett stort träd i sig.

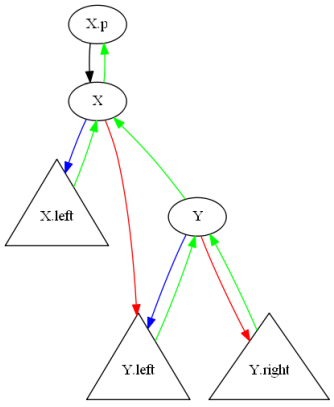
Blåa pilar indikerar ett vänsterbarn, röda pilar indikerar ett högerbarn, och gröna pilar indikerar förälder. `X.p` som är förälder till `X` har en svart pil då det från början är okänt och oviktigt om `X` är ett höger- eller vänsterbarn, eller om det är rot noden. Vi kommer nu gå igenom de olika operationerna som finns i rotationsfunktionen och titta på vad som händer i trädet.

1. `y = x.right`

I det första steget så sätter vi bara att vi har `Y` som är `X` originella högerbarn.

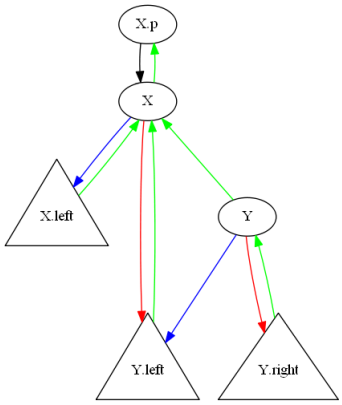
2. `x.right = y.left`

Operation två sätter att `X` tar över vänsterbarnet från `Y` som sitt högerbarn. Här är det viktigt att vi har `Y` så att vi inte tappar delar av trädet när vi ändrar högerbarnet till `X`.



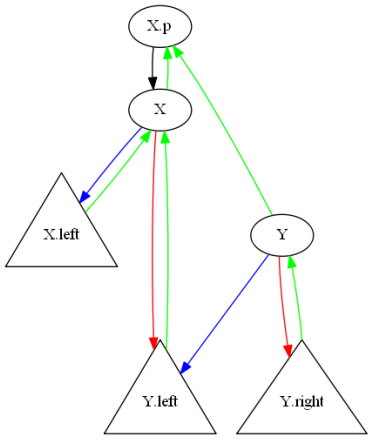
```
3.     if y.left != T.nil
4.         y.left.p = x
```

Den tredje operationen sätter föräldern för det originella vänsterbarnet till `Y` till `X` om barnet inte var nilnoden.



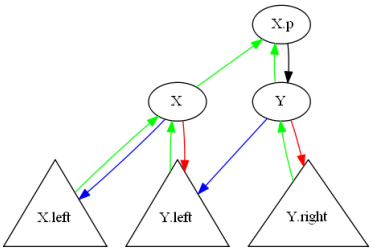

```
5.     y.p = x.p
```

Sedan så sätter den fjärde operationen `X.p` som förälder till `Y` så att det senare korrekt går att gå uppåt i trädet härifrån.



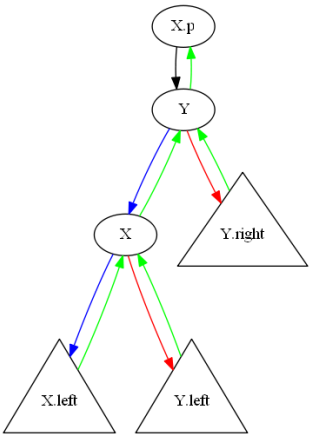
```
6.    if x.p == T.nil
7.       T.root = y
8.    elseif x == x.p.left
9.       x.p.left = y
10.   else x.p.right = y
```

Den femte operationen ser sedan till att `X.p` får `Y` som korrekt barn baserat på om `X` var ett höger- eller vänsterbarn. Om `X` var rotnoden så ser vi istället till så att roten pekar på `Y` korrekt så vi inte tappar hela trädet.




```
11.   y.left = x
12.   x.p = y
```

De två sista operationerna ser sedan till att koppla så att `X` blir ett korrekt vänsterbarn till `Y` så att vi inte tappar den delen av trädet. Detta gör den genom att sätta `X` som vänsterbarn till `Y`, och `Y` som förälder till `X`.



En högerrotation följer samma logik som den vi sett nu, fast speglad så att säga, så vi roterar åt andra hållet.