3.2 Punamustat puut

- Kaikki binäärihakupuun operaatiot siis pystytään toteuttamaan ajassa $\mathcal{O}(h)$, missä h puun korkeus
- jos voimme taata että korkeus on kohtuullinen, eli logaritminen suhteessa puun solmujen määrään, on kaikkien operaatioiden aikavaativuus $\mathcal{O}(log\ n)$, eli todella hyvä

  (huom: esim $log_2\ 16777216 = 24$, $log_2\ 4294967296 = 32$, logaritmi siis kasvaa todella hitaasti)

- *punamusta puu* on binäärihakupuu missä jokaiseen solmuun on lisätty yhden bitin verran tietoa, *solmun väri*, joka voi olla punainen tai musta
- rajoittamalla solmujen väritystä poluilla sisäsolmuista lehtiin voidaan taata että mikään polku ei ole yli kaksi kertaa pitempi kuin joku toinen
- näin varmistutaan siitä että puu on suurin piirtein tasapainossa
- puunamusta puu rakentuu RB-puusolmu-tietueista, joilla on seuraavat kentät:

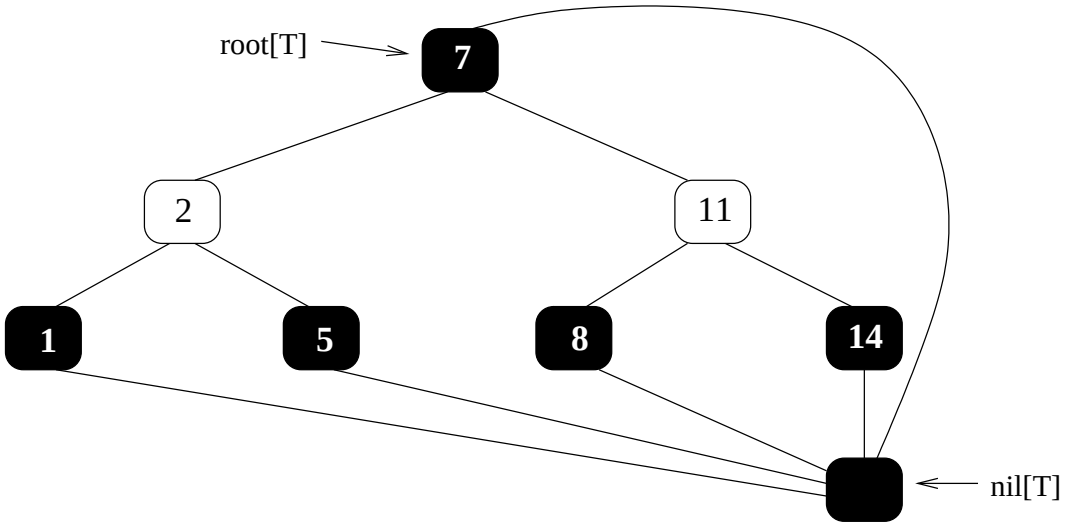
key	talletettu tietoalkio
left	viite vasempaan lapseen
right	viite oikeaan lapseen
p	viite vanhempaan
color	solmun väri

- puulla T attribuutti $root[T]$ joka osoittaa juurisolmuun

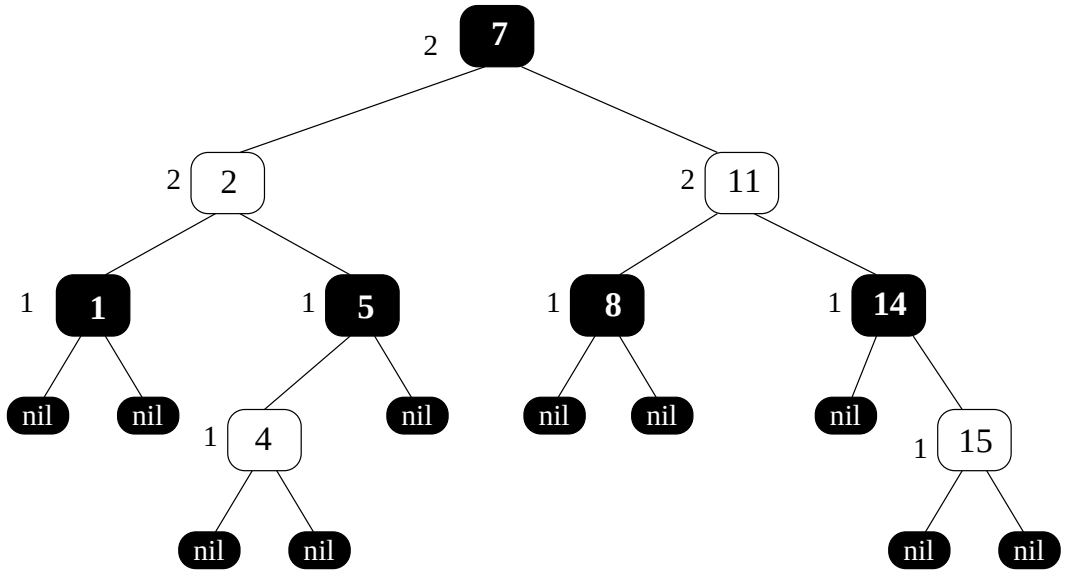
- puuhin liittyy erityinen nil-solmu $nil[T]$, jolla on samat kentät kuin normaaleilla puusolmuilla, *nil-solmun väri on musta*, muiden kenttien arvoilla ei ole väliä
- jos solmulla x ei ole vasenta lasta, $left[x] = nil[T]$, vastaavasti oikealle lapselle
- puun juurisolmulla parent-kentän arvo $nil[T]$
- nil-solmu siis korvaa tavallisessa binäärihakupuussa käyttämämme NIL viitteen, syyn tälle käytännölle näemme poisto-operaation yhteydessä
- punamustan puun lehdet siis ovat nil-solmuja ja kaikki tieto talletetaan sisäsolmuihin

- binäärihakupuu on punamusta puu, jos seuraavat ehdot ovat voimassa
  1. jokainen solmu on joko punainen tai musta
  2. juurisolmu on musta
  3. jokainen lehti (nil-solmu) on musta
  4. jos solmu on punainen, sen molemmat lapset ovat mustia
  5. jokainen polku tietystä solmusta sen lehtiin sisältää saman määrän mustia solmuja

- seuraavassa eräs punamusta puu



- yleensä emme piirrä kuviin nil[T] solmua, tai piirrämme kaikille solmuille oman nil-solmun
- määritellään solmun x *punamustakorkeudeksi* mustien solmujen lukumäärä jollain polulla solmusta lehteen, solmua x ei lasketa mukaan. itseään solmun x punamustakorkeudesta käytetään merkintää $bh(x)$
- huom: punamustaehdon 5 mukaan solmun jokaisen lehteen johtavan polun mustien lukumäärä on sama, siis solmun punamustakorkeus on käsitteenä hyvin määrittely
- puun punamustakorkeus on sen juurisolmun punamustakorkeus
- seuraavassa puu, missä kunkin solmun punamustakorkeus merkitty solmun viereen merkitty



- todistamme nyt tuloksen mikä kertoo punamustan puun korkeuden suhteessa sen sisäsolmujen eli alkioita sisältävien solmujen lukumäärään
- **Lause 3**:

  Jos punamustalla puulla on n sisäsolmua, sen korkeus on korkeintaan $2 \times log(n+1)$

  todistus:

  > Osoitetaan aluksi että *minkä tahansa solmun x alipuussa on vähintään $2^{bh(x)}-1$ sisäsolmua.* Todistetaan tämä aputulos induktiolla solmun x (normaalin) korkeuden suhteen. Jos korkeus on 0, on kyseessä lehti, ja tällöin alipuu sisältää $2^0 - 1 = 0$ sisäsolmua.
  >
  > Oletetaan sitten että x:n korkeus on vähintään 1, ja että x on sisäsolmu, jolla on kaksi lasta. x:n lapsilla punamustakorkeus on nyt $bh(x)$ tai $bh(x) - 1$, riippuen lapsen väristä. Koska lapsen korkeus on pienempi kuin x:n

korkeus, voimme soveltaa induktio-oletusta ja päätellä että lapsella on ainakin $2^{bh(x)-1}-1$ sisäsolmua. Siten x:n alipuulla on vähintään $2 \times (2^{bh(x)-1} - 1) + 1 = 2^{bh(x)} - 1$ sisäsolmua, mikä todistaa väitteen.

Varsinaisen lauseen todistamiseksi oletetaan että puun korkeus on h. Punamustaehdon 4 perusteella tiedämme että vähintään joka toinen solmu polulla juuresta lehteen on musta, siis juuren *punamustakorkeus on vähintään* $h/2$.

Aluksi osoitetun aputuloksen perusteella tiedämme nyt puun sisäsolmujen lukumäärä on vähintään $2^{h/2} - 1$, eli $n \geq 2^{h/2} - 1$. Siirretään 1 yhtälön vasemmalle puolelle ja otetaan molemmilta puolilta 2 kantainen logaritmi, jolloin tuloksena $log(n+1) \geq h/2$ eli $h \leq 2 \times log(n+1)$

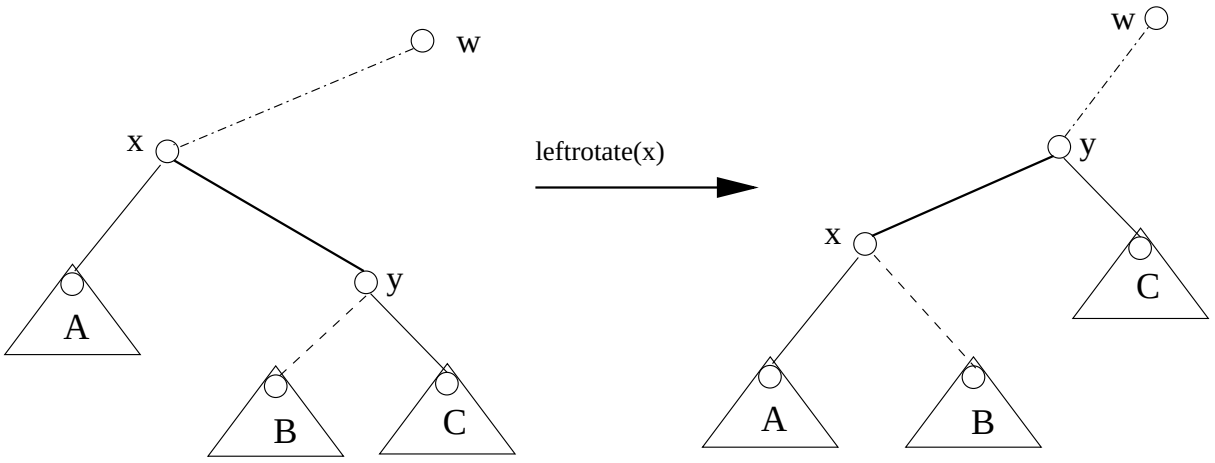
- Lauseesta seuraa heti että punamustassa puussa haku-operaation sekä pienimmän, suurimman, seuraavan ja edellisen palauttavien operaatioiden aikavaativuus on pahimmassa tapauksessa $\mathcal{O}(log\ n)$
- eli esim. 16777215 avainta sisältävän punamustan puun korkeus on enintään 48 ja 4294967295 avainta sisältävän enintään 64
- suhteellisen matalaan (eli todella nopeat operaatiot takaavaan) puuhun mahtuu siis hyvin paljon tavaraa
- nyt on vaan huolehdittava että lisäys ja poisto eivät tuhoa punamustaominaisuutta

Kierrot

- alkion lisäys tai poisto punamustaan puuhun siis saattaa aiheuttaa sen että punamustaehdot rikkoutuvat
- lisäys ja poisto punamustaan puuhun tapahtuu lähes samaan tyyliin kuin normaalin binäärihakupuun yhteydessä, tämän lisäksi kutsutaan erillistä operaatiota joka korjaa mahdollisesti rikkoutuneen punamustaominaisuuden
- korjausoperaatio muokkaa puussa olevien solmujen järjestystä sekä väritystä
- solmujen järjestystä muokataan seuraavassa esitettävien *kierto-operaatioiden* avulla
- ensin ns. *vasen kierto*

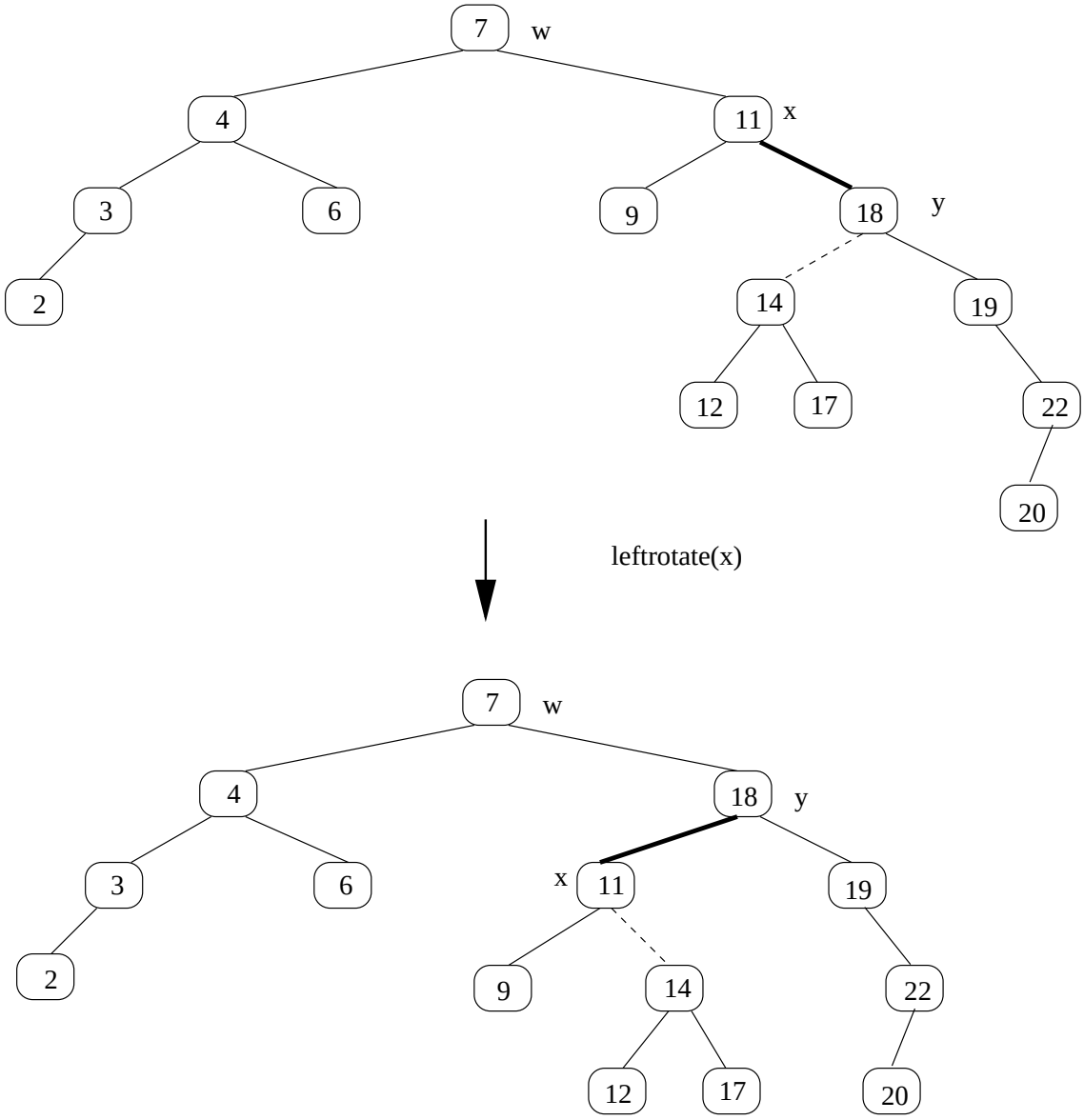
```
left-rotate(T,x)
1   y ←right[x]
2   right[x] ←left[y]
3   p[left[y]] ←x
4   w ←p[x]
5   p[y] ←w
6   if w = NIL
7       then root[T] ←y
8       else if left[w] = x
9                then left[w] ←y
10               else right[w] ←y
11  left[y] ←x
12  p[x] ←y
```

- seuraava kuva valaisee kierron vaikutusta puuhun:



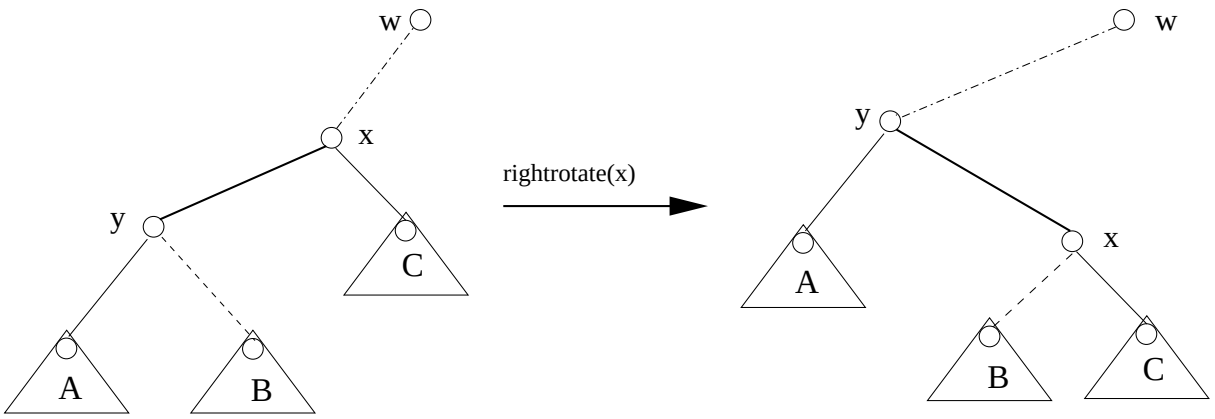
- solmun x vasemmassa kierrossa siis oikea lapsi y tulee alipuun uudeksi juureksi, x tulee y:n vasemmaksi lapseksi ja y:n vasen lapsi x:n oikeaksi lapseksi
- kierto on paikallinen operaatio joka säilyttää solmujen järjestyksen oikeanlaisena
- ennen kiertoa alipuun B kaikki solmut ovat binäärihakupuuehdon perusteella suurempia kuin key[x] ja pienempiä kuin key[y]

  selvästikin sama ominaisuus on voimassa myös kierron jälkeen
- seuraavalla sivulla konkreettinen esimerkki
  - solmujen värejä ei ole merkitty, eikä puu ole tasapainossa
  - huomaa miten kierto tasapainoittaa puun epätasapainossa olevaa kohtaa
  - hakupuuominaisuus säilyy kierrosta huolimatta


7
w
4
11
x
3
6
9
18
y
2
14
19
12
17
22
20
leftrotate(x)
7
w
4
18
y
3
6
x
11
19
2
9
14
22
12
17
20

- oikea kierto on symmetrinen edellisen kanssa

```
right-rotate(T,x)
1   y ←left[x]
2   left[x] ←right[y]
3   p[right[y]] ←x
4   w ←p[x]
5   p[y] ←w
6   if w = NIL
7       then root[T] ←y
8       else if left[w] = x
9                then left[w] ←y
10               else right[w] ←y
11  right[y] ←x
12  p[x] ←y
```



Alkion lisäys

- aluksi uusi solmu lisätään samoin kuin lisäys tehtäisiin normaaliin binäärihakupuuhun, tämän jälkeen kutsutaan apurutiinia rb-insert-fixup joka korjaa mahdollisen punamustaehdon rikkoutumisen
- lisäysoperaatio

```
insert(T, k)
1   z ←new RB-puusolmu
2   key[z] ←k
3   left[z] ←nil[T]
4   right[z] ←nil[T]
5   x ←root[T]
6   y ←nil[T]
7   while x ≠ nil[T] do
8       y ←x
9       if k < key[x]
10          then x ←left[x]
11          else x ←right[x]
12  p[z] ←y
13  if y = nil[T]
14      then root[T] ←z
15      else if k < key[y]
16          then left[y] ←z
17          else right[y] ←z
18  color[z] ←RED
19  rb-insert-fixup(T,z)
```

- erona binääripuun lisäys-operaatioon NIL korvattu nil[T]:llä,

lisättävän solmun väriksi asetetaan punainen ja lopussa kutsutaan korjausapurutiinia

- korjausapurutiini

```
rb-insert-fixup(T, z)
1   while color[p[z]] = RED do
2       if p[z] = left[p[p[z]]] then
3           y ←right[p[p[z]]]
4           if color[y] = RED then
5               color[p[z]] ←BLACK          ▷tapaus 1
6               color[y] ←BLACK             ▷tapaus 1
7               color[p[p[z]]] ←RED         ▷tapaus 1
8               z ←p[p[z]]                  ▷tapaus 1
9           else if z = right[p[z]] then
10              z ←p[z]                     ▷tapaus 2
11              left-rotate(T,z)            ▷tapaus 2
12          color[p[z]] ←BLACK              ▷tapaus 3
13          color[p[p[z]]] ←RED             ▷tapaus 3
14          right-rotate(T,p[p[z]])         ▷tapaus 3
15      else
16-27       ▷esitetty sivulla 82
28  color[root[T]] ←BLACK
```

- apurutiini näyttää aluksi todella vaikeaselkoiselta, mutta huolellisella tutkimisella saamme selville sen toimintaidean
  - ainoa punamustaehto mikä lisäyksessä voi rikkoutua on ehto 4 *punaisen solmun lapset ovat mustia*, jos siis lisätyn solmun vanhempi on punainen (rivin 1 ehto), on jotain tehtävä

– while-silmukan tavoitteena on siirtää kohtaa missä ehtoa 4 rikotaan ylöspäin säilyttäen samalla ominaisuus 5 *jokainen polku tietystä solmusta sen lehtiin sisältää saman määrän mustia solmuja* voimassa
– iteraation alussa z osoittaa solmuun missä ehto 4 on rikkoutunut, iteraatiokierroksella joko z liikkuu ylöspäin puussa tai suoritetaan joitakin kertoja ja iteraatio loppuu
– silmukassa on **kuusi erilaista tapausta**, kolme niistä on symmetristä kolmelle muulle erottavana tekijänä se onko z:n isä isoisän vasen vai oikea lapsi (testi rivillä 2), tapausten 4-6 koodi on esitetty sivulla 82
– tapauksessa 1 sedän (johon y osoittaa) väri on punainen (testaus rivillä 4), kun taas tapauksissa 2 ja 3 setä on musta
– **tapaus 1**: valutetaan isoisän musta väri isään ja setään, nyt solmu z ei ole enää ongelma, mutta koska isoisä muutetaan punaiseksi, se saattaa rikkoa ehdon 4, eli muutetaan z osoittamaan isoisää ja palataan silmukan alkuun
– tapauksissa 2 ja 3 setä on musta, ja tapaukset eroavat siinä onko käsiteltävä solmu z oikea vai vasen lapsi
– **tapaus 2**: z on oikea lapsi, sovelletaan vasenta kiertoa isään ja tilanne muuttuu tapaus 3:ksi (kierron yhteydessä entisestä isästä tulee ongelmasolmu z)
– **tapaus 3**: z on vasen lapsi, muutetaan isän ja isoisän väritystä ja kierretään isoisää oikealle
– huomion arvoista on se että jokainen tapauksista säilyttää punamustaehdon 5 voimassaolon, **miksi?**

– tapauksen 3 jälkeen vanhempi on aina musta joten ongelmat korjattu ja työ on tehty (punamustaehto 4 saatettu voimaan ja ehto 5 pidetty voimassa)

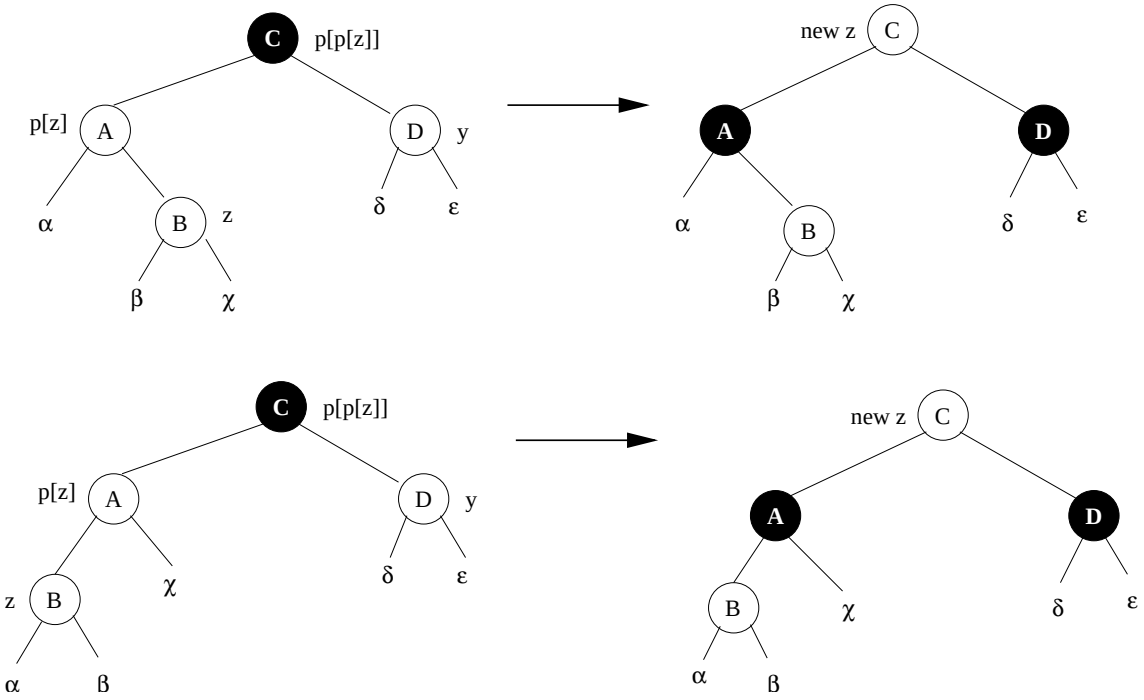
– huom: miksi lopussa asetetaan juuren väriksi musta?

– while-silmukka toistuu vain tapauksen 1 kohdalla, tällöin seuraavalla kerralla aloitetaan lähempänä juurta, rb-insert-fixup operaation aikavaativuus siis pahimmassa tapauksessa on sama kuin puun korkeus, eli $\mathcal{O}(log\ n)$, tilaa operaatio ei vie kuin vakiomäärän

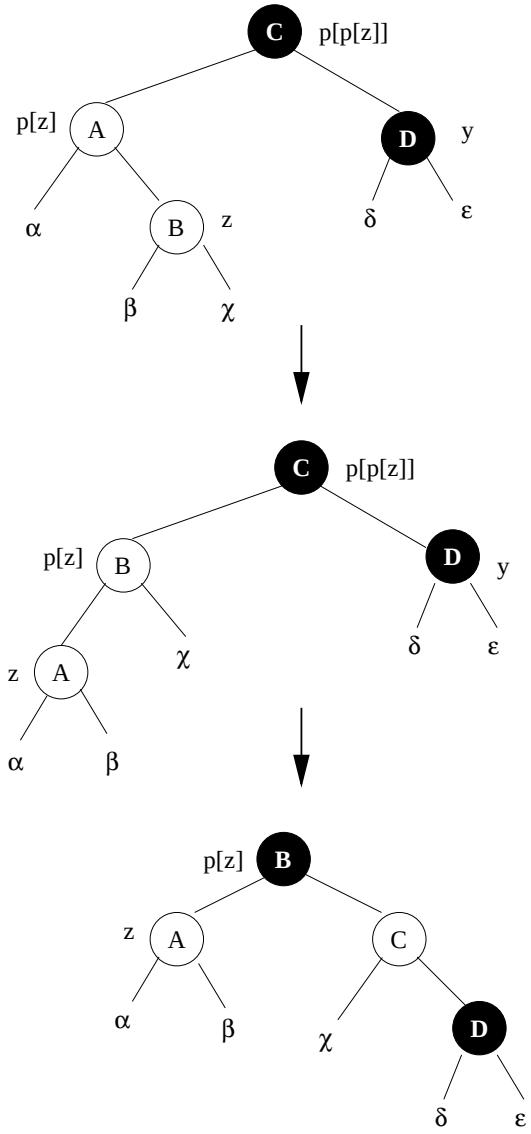
  korjausoperaatio ei siis huononna lisäysoperaation vaativuutta

– insert-operaation aikavaativuus punamustassa puussa on siis $\mathcal{O}(log\ n)$ ja tilavaativuus $\mathcal{O}(1)$

• tapausta 1 selittävä kuva



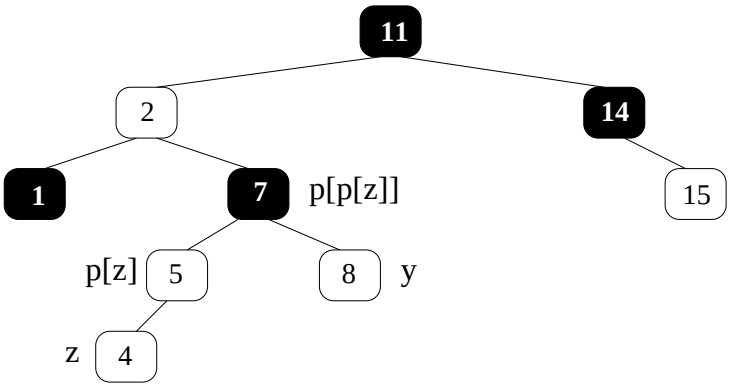
- tapauksia 2 ja 3 selittävä kuva



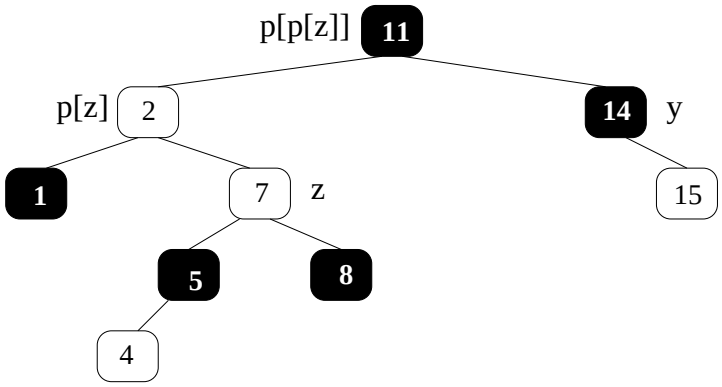
Tapaus 2

Tapaus 3

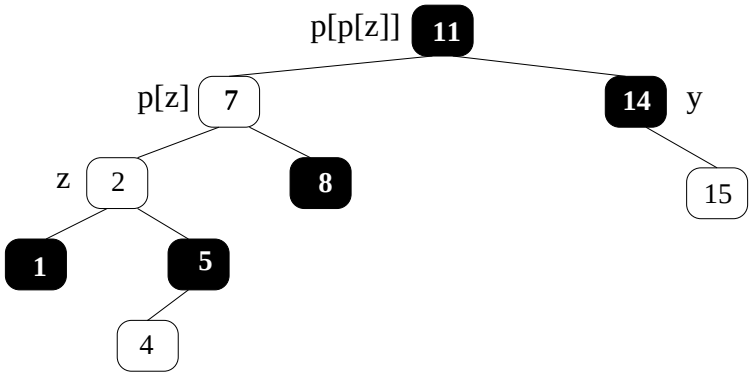
- seuraavalla sivulla vielä konkreettinen esimerkki missä solmun lisääminen puuhun aiheuttaa kaikkien korjaustapausten suorittamisen



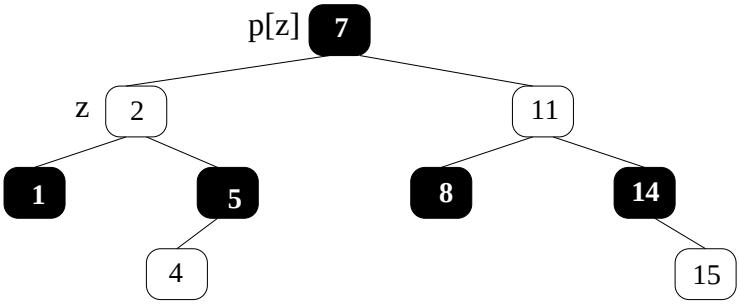
tapaus 1



tapaus 2



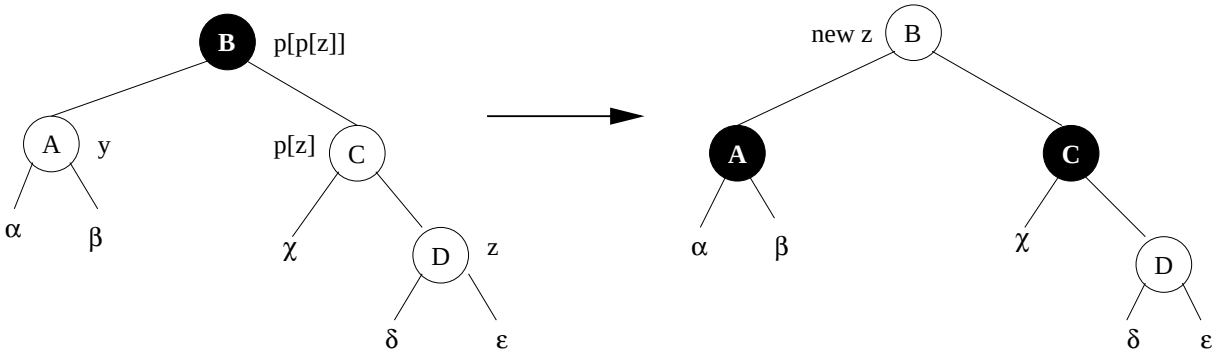
tapaus 3

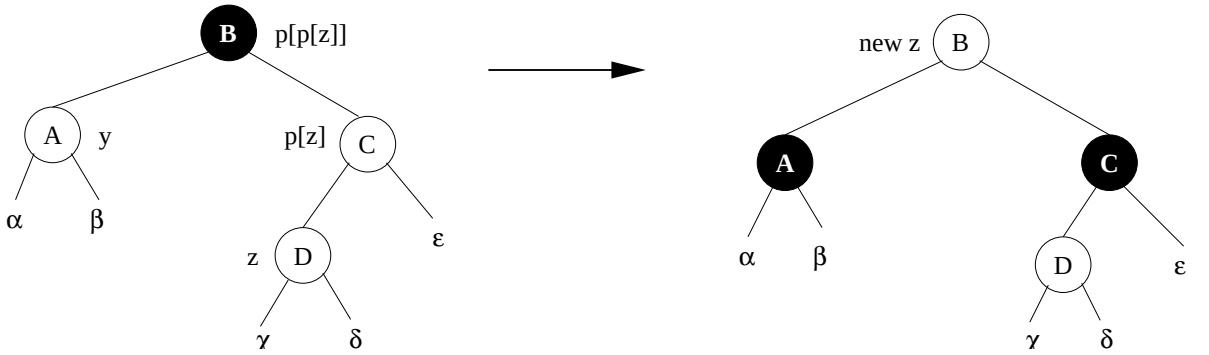


- esitetään vielä tapaukset 4, 5 ja 6 käsittelevä else-haara

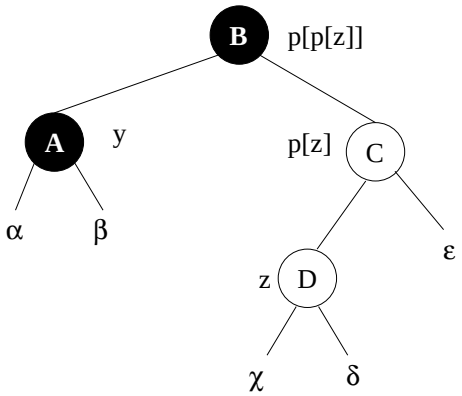
```
rb-insert-fixup(T, z)
1   while color[p[z]] = RED do
2       if p[z] = left[p[p[z]]] then
3-14        ▷esitetty sivulla 77
15      else
16          y ←left[p[p[z]]]
17          if color[y] = RED then
18              color[p[z]] ←BLACK          ▷tapaus 4
19              color[y] ←BLACK             ▷tapaus 4
20              color[p[p[z]]] ←RED         ▷tapaus 4
21              z ←p[p[z]]                  ▷tapaus 4
22          else if z = left[p[z]] then
23                  z ←p[z]                 ▷tapaus 5
24                  right-rotate(T,z)       ▷tapaus 5
25              color[p[z]] ←BLACK          ▷tapaus 6
26              color[p[p[z]]] ←RED         ▷tapaus 6
27              left-rotate(T,p[p[z]])      ▷tapaus 6
28  color[root[T]] ←BLACK
```

- tapauksissa 4-6 siis solmun z isä on isoisän oikea lapsi ja setä y on isoisän vasen lapsi
- **tapaus 4**: valutetaan isoisän musta väri isään ja setään, nyt solmu z ei ole enää ongelma, mutta koska isoisä muutetaan punaiseksi, se saattaa rikkoa ehdon 4, eli muutetaan z osoittamaan isoisää ja palataan silmukan alkuun
- tapausta 4 selittävä kuva

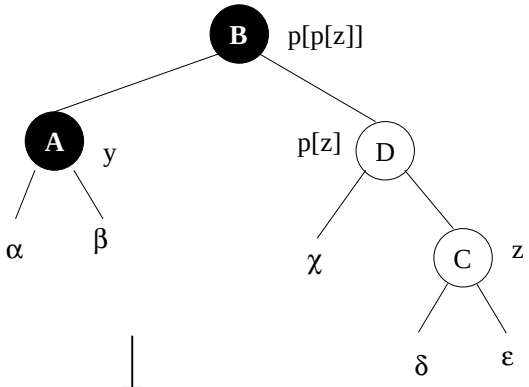




- tapauksissa 5 ja 6 setä on musta, ja tapaukset eroavat siinä onko käsiteltävä solmu z oikea vai vasen lapsi
- **tapaus 5**: z on vasen lapsi, sovelletaan oikeaa kiertoa isään ja tilanne muuttuu tapaus 5:ksi (kierron yhteydessä entisestä isästä tulee ongelmasolmu z)
- **tapaus 6**: z on oikea lapsi, muutetaan isän ja isoisän väritystä ja kierretään isoisää vasemmalle
- tapauksia 5 ja 6 selittävä kuva seuraavalla sivulla



Tapaus 5



Tapaus 6