Insert 30

jika root masih kosong buat node value 30, warna merah

jika node itu tidak punya parent (base root) warna = hitam.

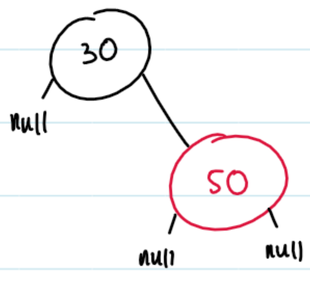
Insert 50

cek tree, jika ada cek value,

jika lebih besar traverse ke subtree kanan,

jika sudah null, root = newroot (50)

warnai merah



Insert 80

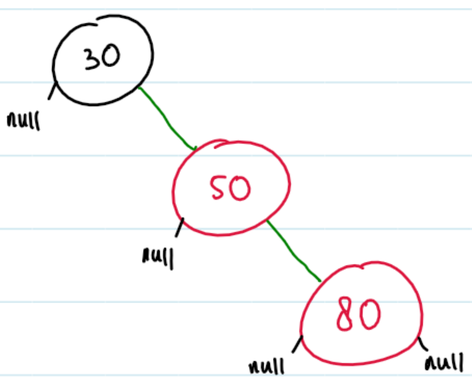
cek tree, jika ada cek value,

jika lebih besar traverse subtree kanan

else traverse subtree kiri.

jika ketemu null, root = newroot (80)

warna merah.



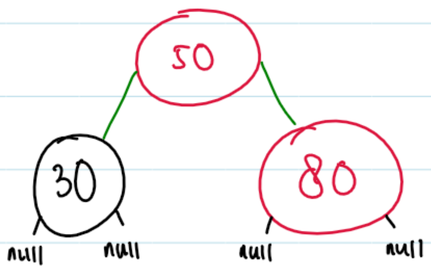
jika beda tinggi sudah lebih dari 1,

rotate,

node adalah subtree kanan parent,

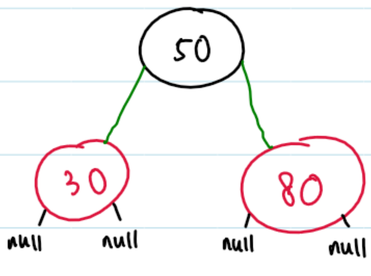
parent adalah subtree kanan grandparent,

maka single rotate left.



Cek warna baseroot. warnai hitam.

warnai 30 merah



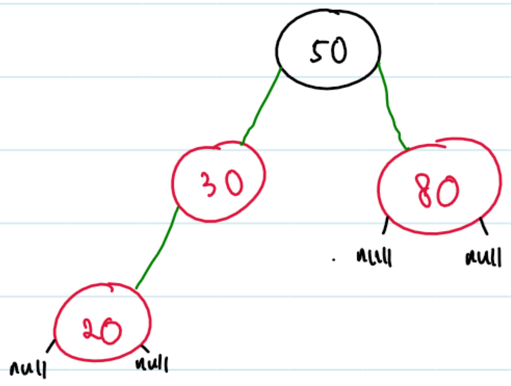
Insert 20

cek root, jika ada cek value,

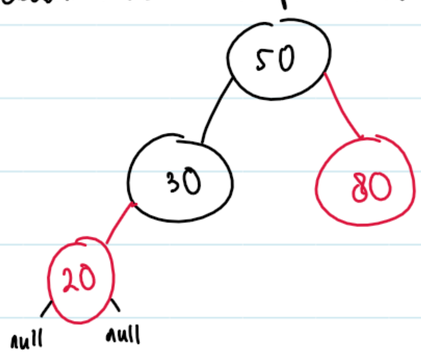
jika lebih besar traverse ke subtree kanan,

jika sudah null, root = newroot (50)

warnai merah



- 2 node warna merah berurutan melanggar aturan red-black.
- sebuah node merah parent dan child bewarna hitam.

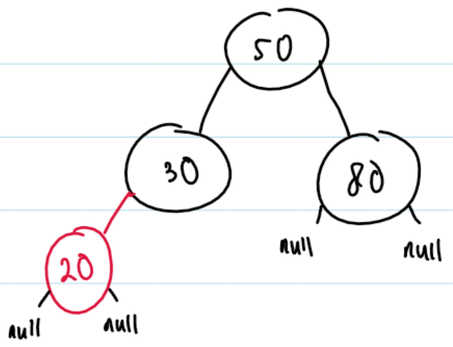


dari root ke semua null harus melewati jumlah node hitam yang sama.

root → left → left → left (2 node hitam)

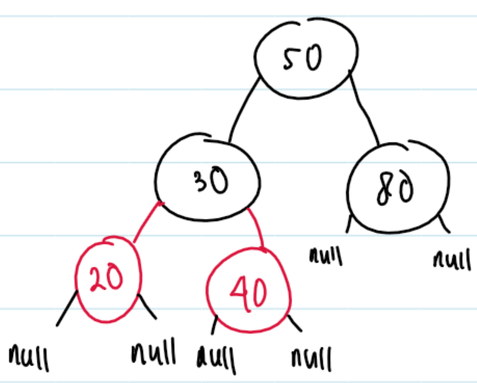
root → right → right (1 node hitam)

maka child kanan harus hitam juga.



Insert 40.

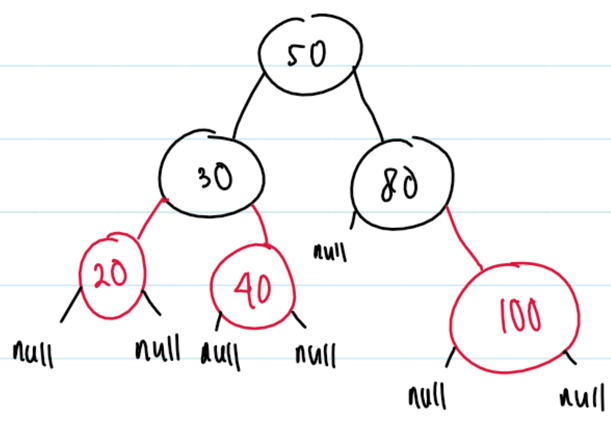
traverse ke kiri warnai merah.



pewarnaan masih belum ada masalah

insert 100.

traverse ke kanan. warnai merah.

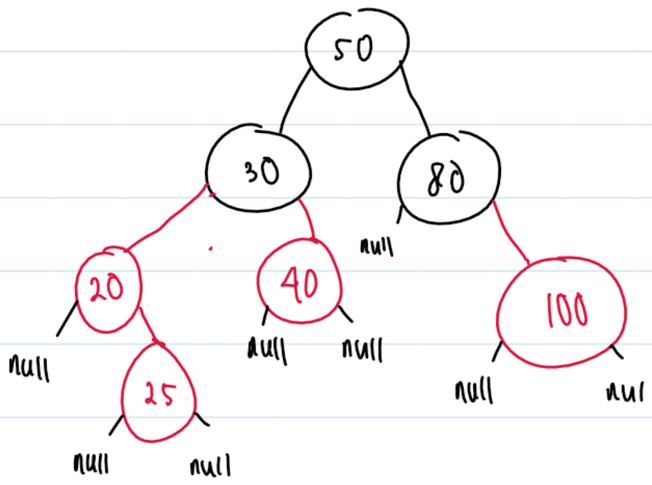


pewarnaan tidak ada masalah karena

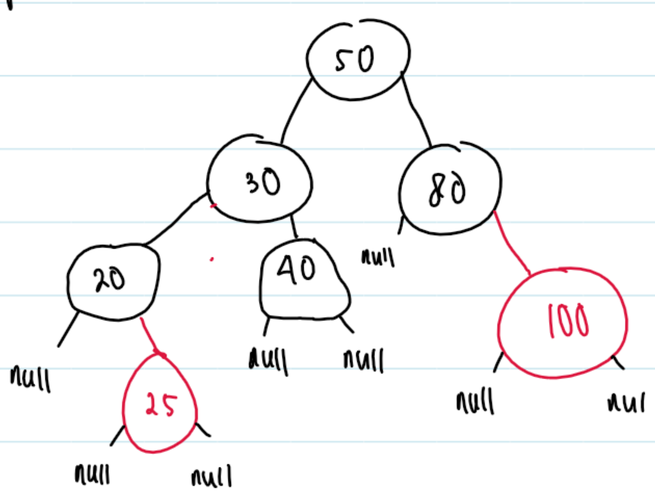
child dan parent dari node merah masih hitam.

Insert 25

traverse left if value is smaller else traverse right.

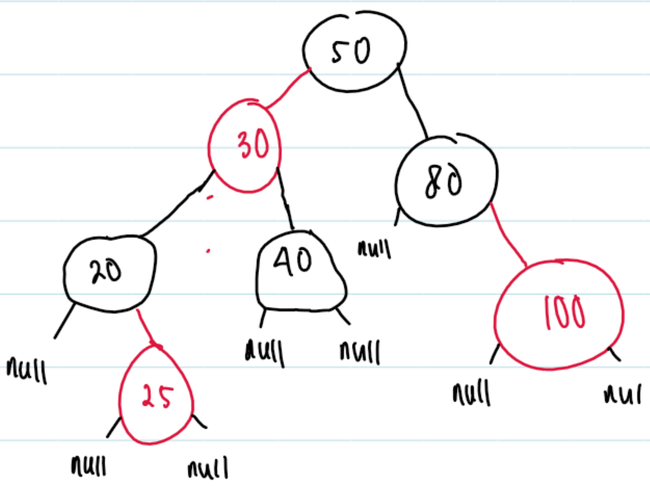


parent and uncle dari node merah harus hitam.



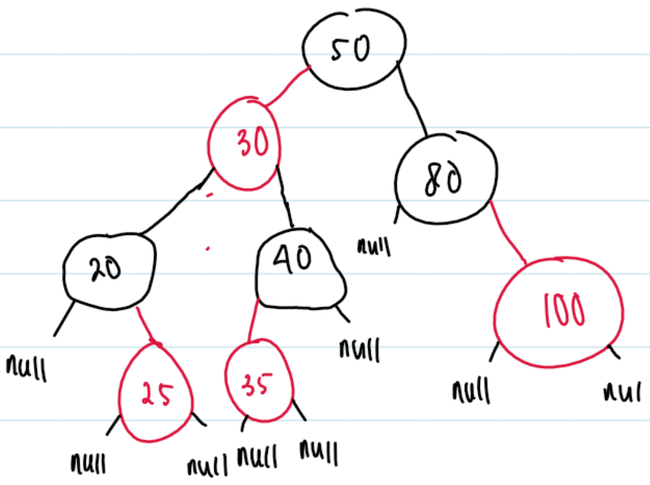
agar root ke semua null menemui jumlah node hitam yang sama,

node 30 kita warnai merah



Insert 35

traverse ke kiri jika value lebih kecil else traverse kanan.



Search

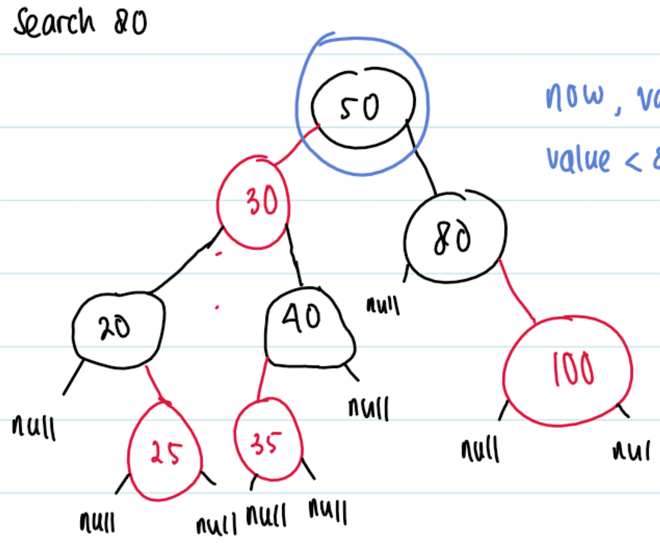
jika value root itu 80 return true.

traverse child kiri jika 80 lebih kecil dari value

else traverse child kanan.

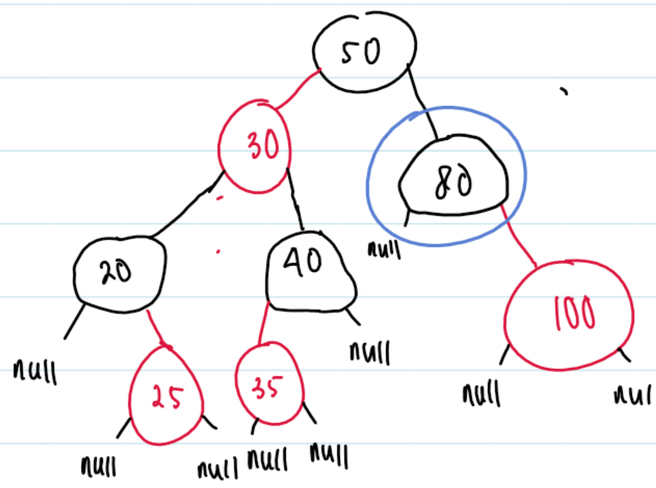
jika sampai null return false // ga ketemu.

Search 80



now, value != 80.

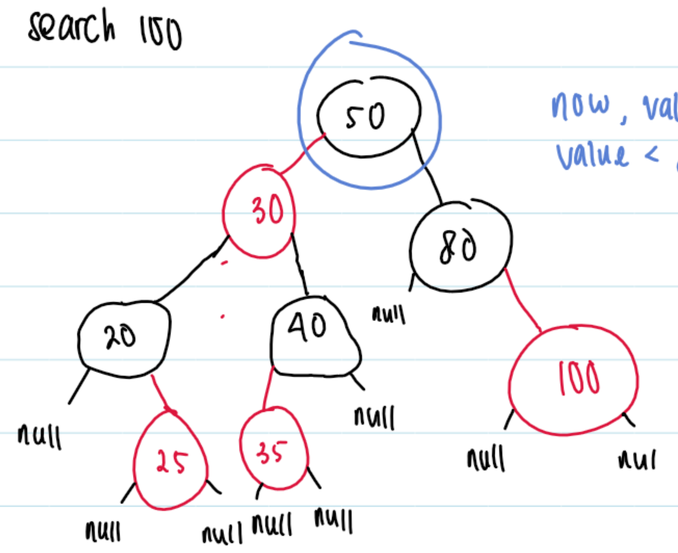
value < 80, traverse child kanan



now, value = 80

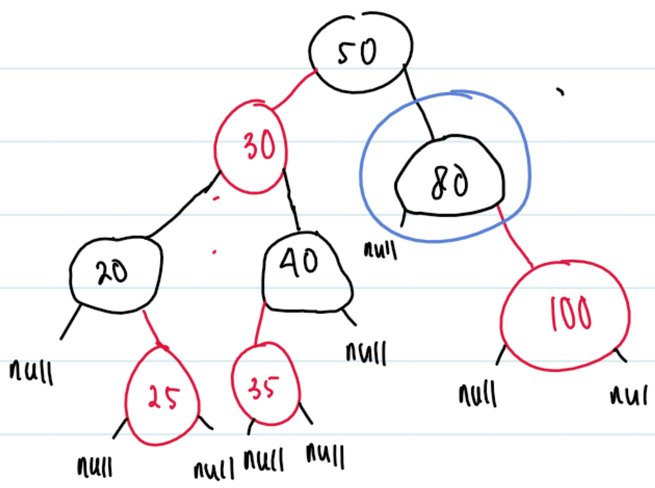
return true.

search 100



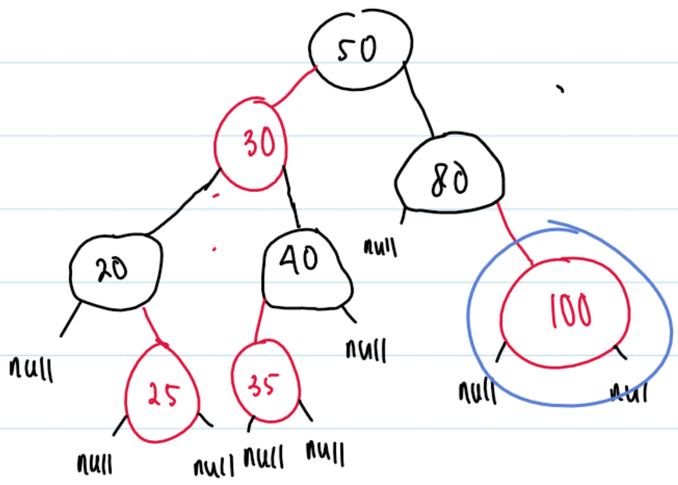
now, value != 100

value < 80, traverse child kanan



now, value != 100

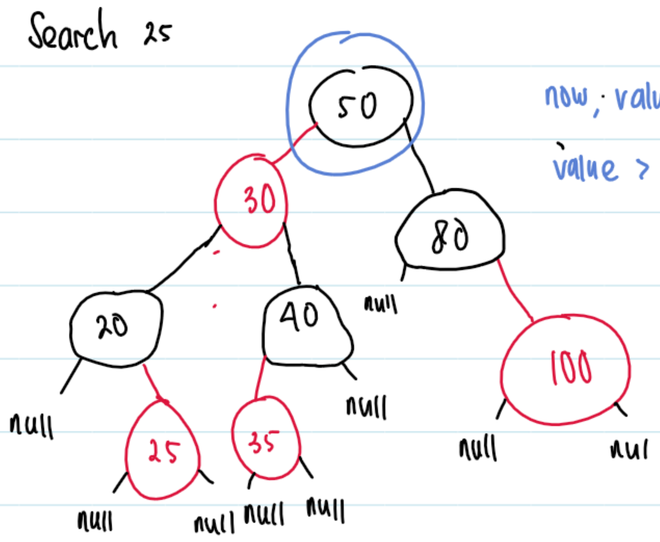
value < 100, traverse child kanan.



now, value = 100

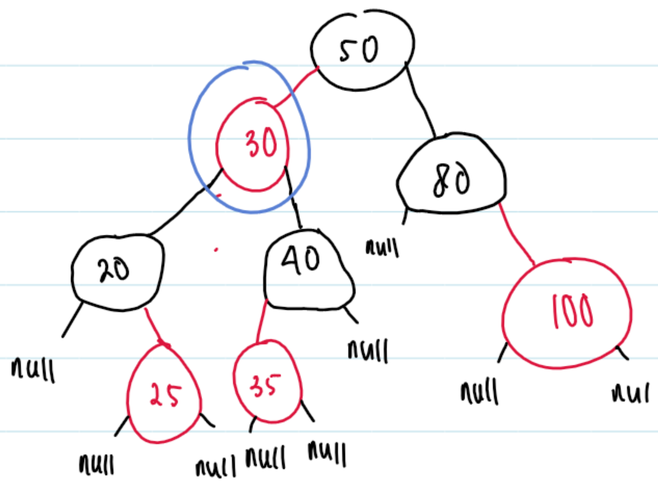
return true.

Search 25



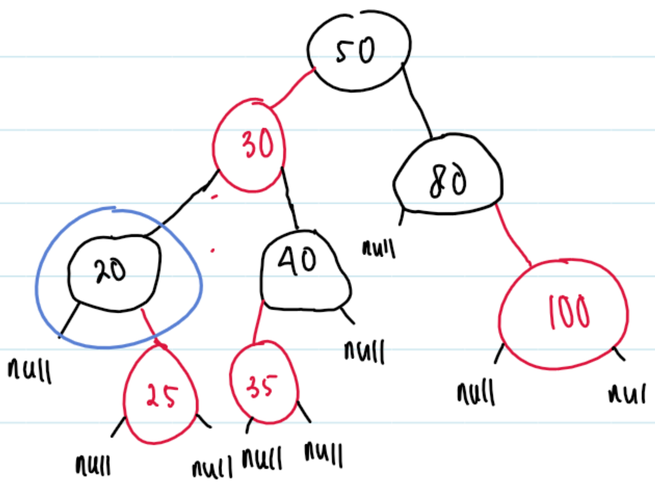
now, value != 25.

value > 25, traverse left child.



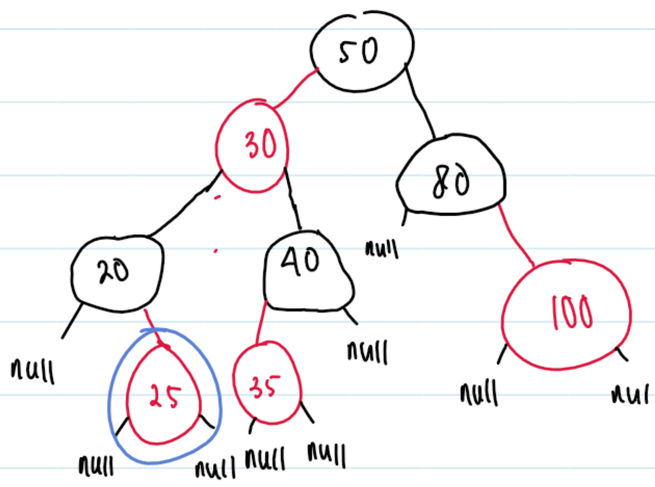
now, value != 25.

value > 25, traverse left child.



now, value != 25.

value < 25, traverse right child.



now, value = 25.

return true

Delete

first step : find x

if root != x :

if x < root.value : traverse child kiri.

else if x > root.value : traverse child kanan.

} algo. search

else :

kalo ga punya child : delete node x.

kalo punya child :

cari node value terbesar di child kiri :

dari child kiri traverse kekanan sampai ketemu null

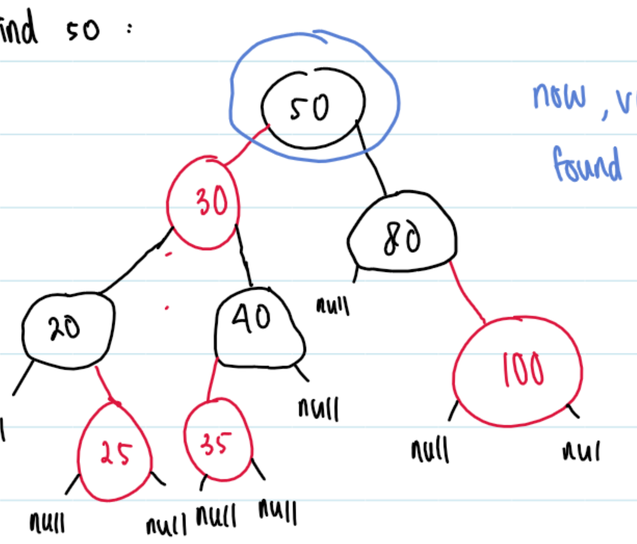
ganti value root dengan value terbesar di child kiri

Delete node value terbesar di child kiri

fix tree agar tidak melanggar aturan red black tree

Delete 50

find 50 :

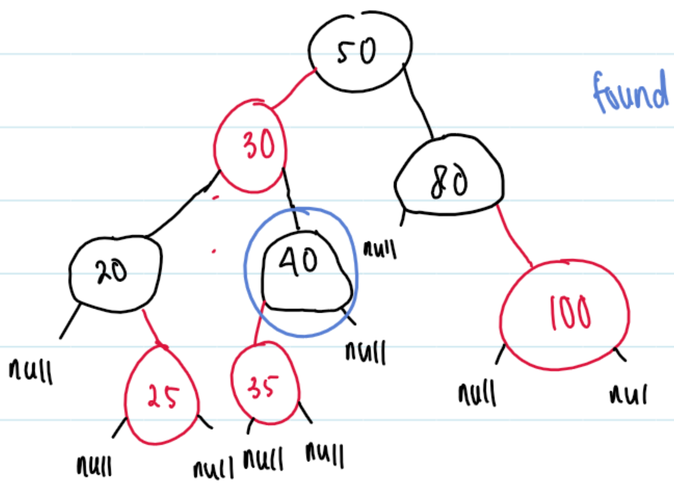


now, value = 50

found 50

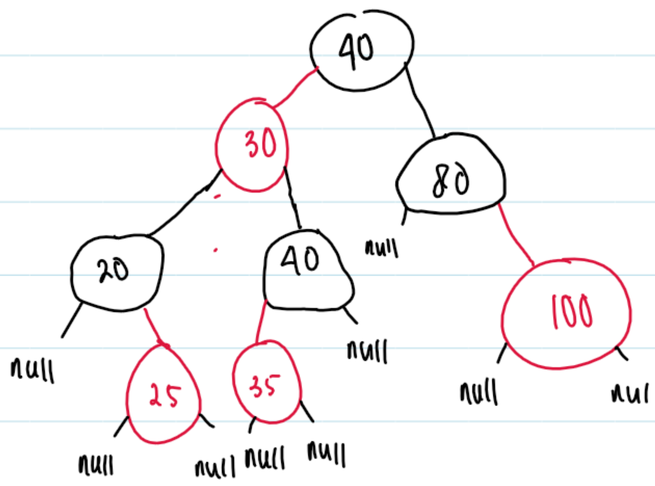
Cari node value terbesar di child kiri 50

dari child kiri 50 traverse kanan sampai null

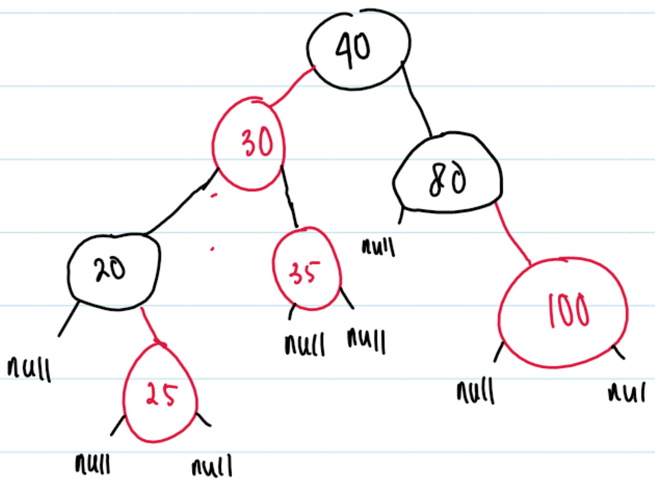


found largest element

ganti nilai root dengan 40

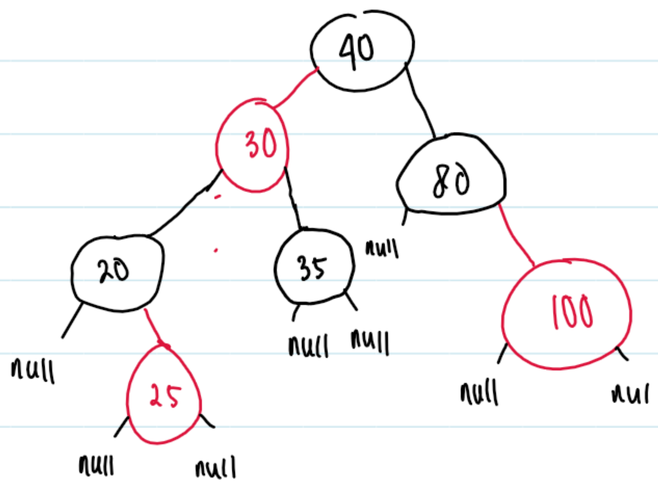


Delete node 40



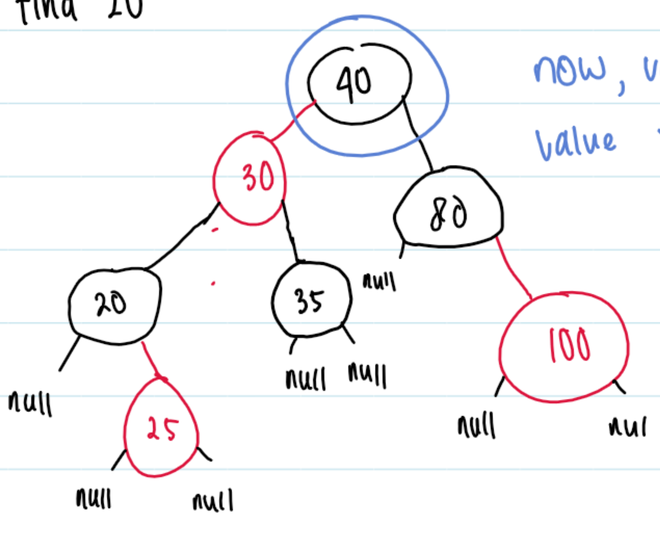
frx tree, recolor node

child node 30 harus hitam



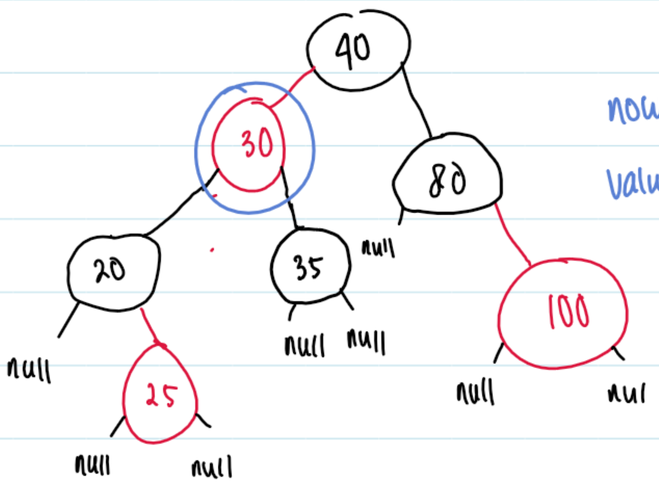
Delete 20

find 20



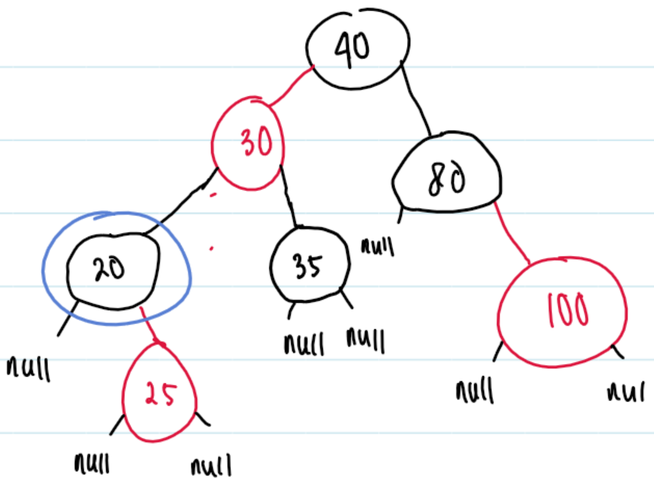
now, value != 20

value > 20, traverse child Kiri



now, value != 20

value > 20, traverse child Kiri

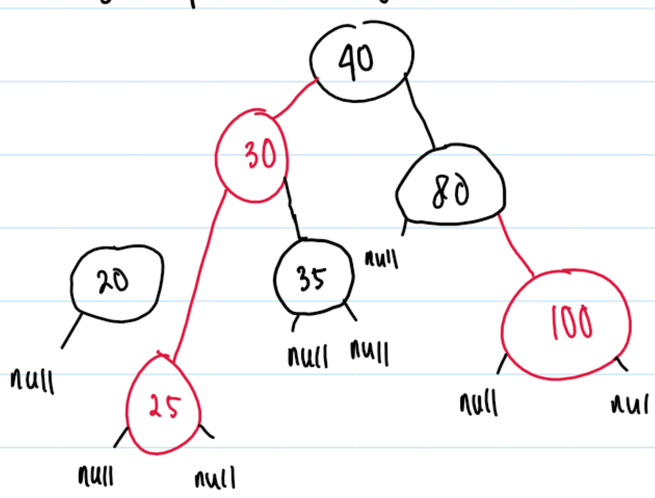


now, value = 20

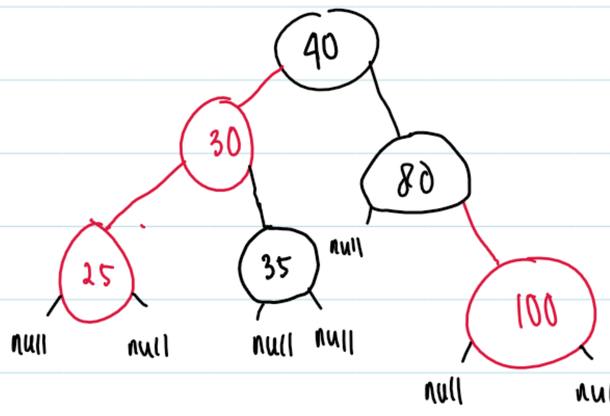
found

if node 20 cuma punya 1 child :

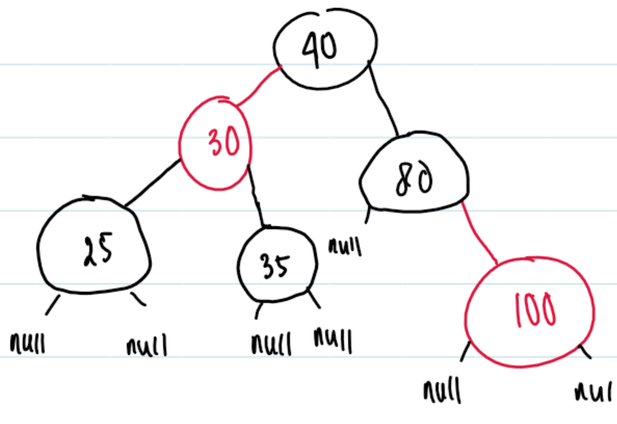
hubungkan parent 20 dengan child dari 20



Delete node 20

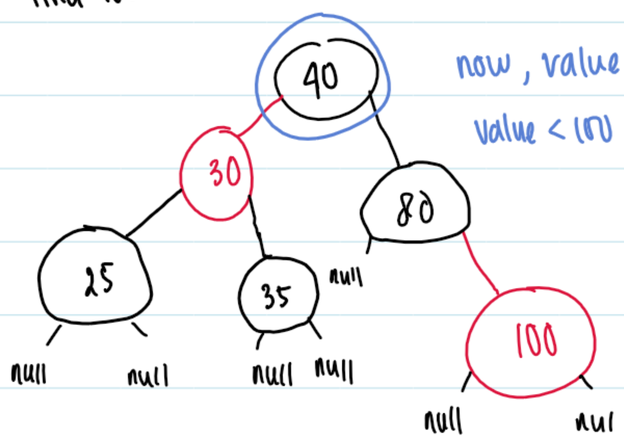


frx : recolor karena node merah childnya harus hitam



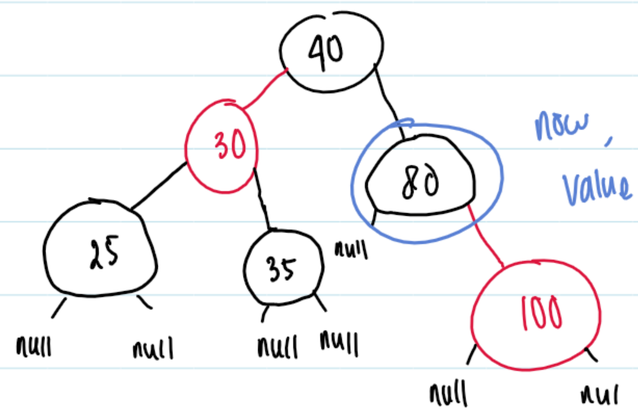
Delete 100

find 100 :



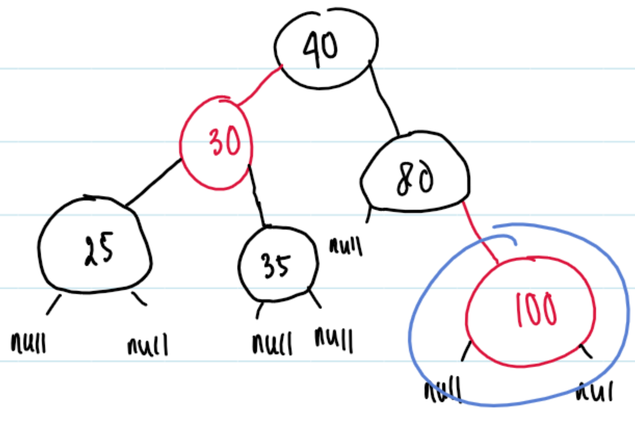
now, value != 100

value < 100, traverse child kanan



now, value != 100

value < 100, traverse child kanan



now, value = 100

found

Cek node 100 punya berapa child.

jika tidak ada child :

Delete node 100