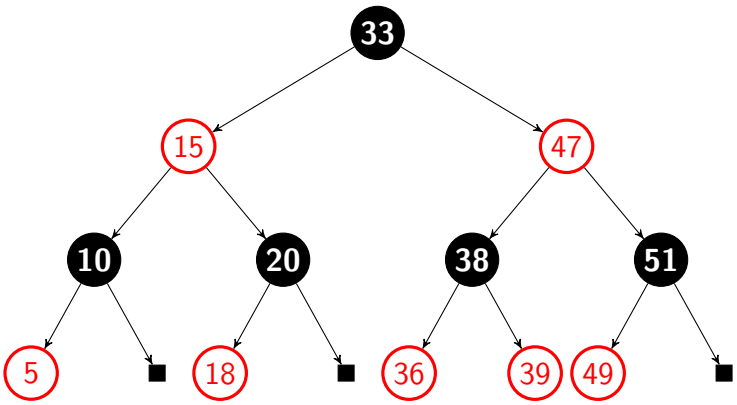
فهرست مطالب

۱ درخت قرمز-مشکی (Red-Black)



۱.۱ ویژگی های درخت قرمز-مشکی (Red-Black)

۱. یک درخت جستجوی دودویی متوازن از نظر ارتفاع می باشد .

۲. هر نود به رنگ قرمز یا مشکی می باشد .

۳. عنصر root در درخت به رنگ مشکی می باشد .

۴. عنصر null یا تهی نیز به رنگ مشکی می باشد .

۵. تعداد عناصر مشکی از هر مسیری از root به سمت برگ ها برابر هست

۶. Parent و Children نمی توانند هر دو به رنگ قرمز باشند .

۷. نود تازه اضافه شده به رنگ قرمز می باشد .

۸. ارتفاع درخت قرمز-مشکی برابر است با :

$$\log(n) \leq h \leq 2\log(n)$$

۲ نحوه ی ساخت درخت قرمز-مشکی

فرض کنید عناصر

$$\text{keys}: 10, 20, 30, 50, 40, 60, 70, 80, 4, 8$$

را به ترتیب برای ساخت درخت قرمز-مشکی استفاده کنیم .

درج در درخت قرمز-مشکی همانند درخت جستوی دودویی می باشد .
وقتی یک نود جدید به درخت قرمز-مشکی اضافه می شود ، آن نود به رنگ قرمز باشد باشد

در ابتدا عنصر 10 را اضافه می کنیم و این عنصر به رنگ قرمز می باشد .

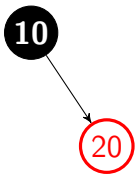
insert 10 ⟹ 10

از آنجایی که ریشه (root) به رنگ مشکی می باشد عنصر 10 را به 10 تغییر می دهیم .

Change root to Black ⟹ 10

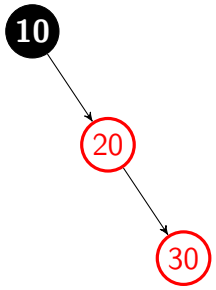
سپس عنصر 20 را اضافه می کنیم ، به خاطر اینکه این عنصر تازه وارد است به رنگ قرمز می باشد .

insert 20 ⟹



سپس عنصر 30 را اضافه می کنیم ، به خاطر اینکه این عنصر تازه وارد است به رنگ قرمز می باشد ، درج این عنصر مشکل **برخورد قرمزها** (**red-red conflict**) را به وجود می آورد .

insert 30 ⟹



برخورد قرمزها (red-red conflict)
هرگاه برخورد قرمزها پیش بیاید به معنای عدم توازن در درخت قرمز-مشکی می باشد و شما برای برقراری دوباره ی توازن نیاز به تغییراتی در ساختار درخت قرمز-مشکی دارید .
دو روش برای انجام تغییرات وجود دارد :

۱. تغییر رنگ (Re-Coloring)

۲. چرخش (Rotation)

دایی (Uncle)
برای ادامه ی بحث ما نیاز به آشنایی با یک مفهوم جدید به نام دایی یا Uncle داریم که به معنای نودِ همسایه یِ پدر می باشد
از این به بعد برای اشاره به این مفهوم از واژه ی Uncle استفاده می کنیم .

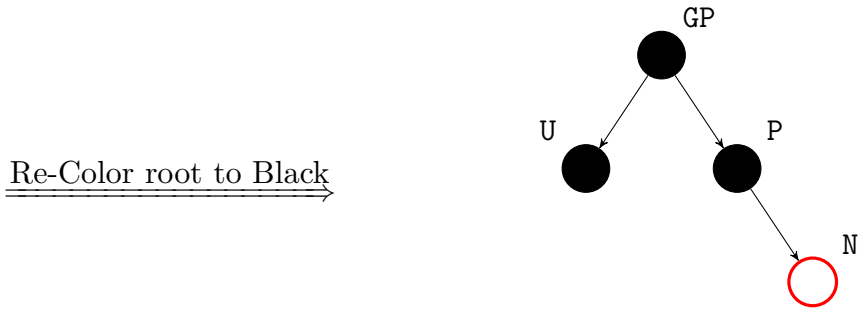
وقتی برخورد قرمزها به وجود می آید می توانیم ۲ حالت نسبت به Uncle نود تازه اضافه شده داشته باشیم .

۱.۲ Uncle قرمز است

وقتی Uncle نود تازه اضافه شده قرمز باشد ما از روش تغییر رنگ (Re-Coloring) استفاده می کنیم .

در صورتی که بعد از تغییر رنگ ، نود (Grand Parent) همان ریشه ی درخت بود آن را به رنگ مشکی تغییر می دهیم .

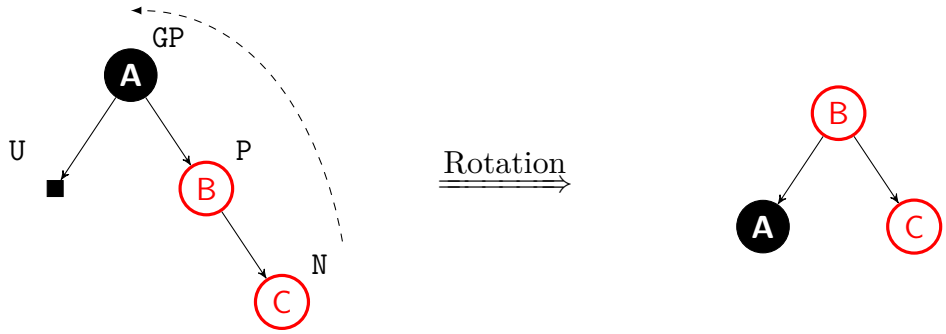
```
if ( GP == root ) {
   Re-Color GP to Black
}
```



۲.۲ Uncle مشکی است

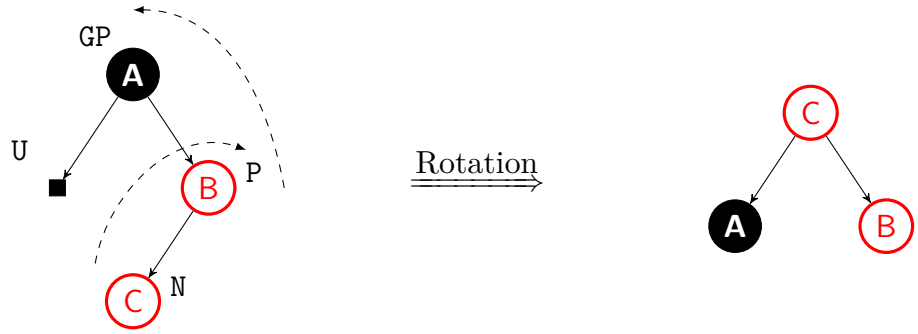
۱.۲.۲ حالت اول Zig-Zig

وقتی Uncle نود تازه اضافه شده مشکی باشد ما از روش چرخش (Rotation) و در صورت نیاز از تغییر رنگ (Re-Coloring) استفاده می کنیم .

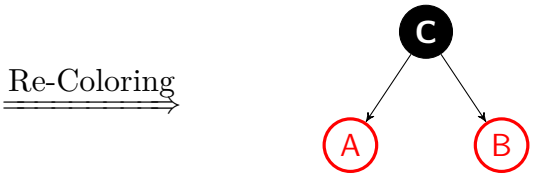


به خاطر اینکه نود ریشه ی درخت قرمز است آن را به رنگ مشکی تغییر می دهیم .

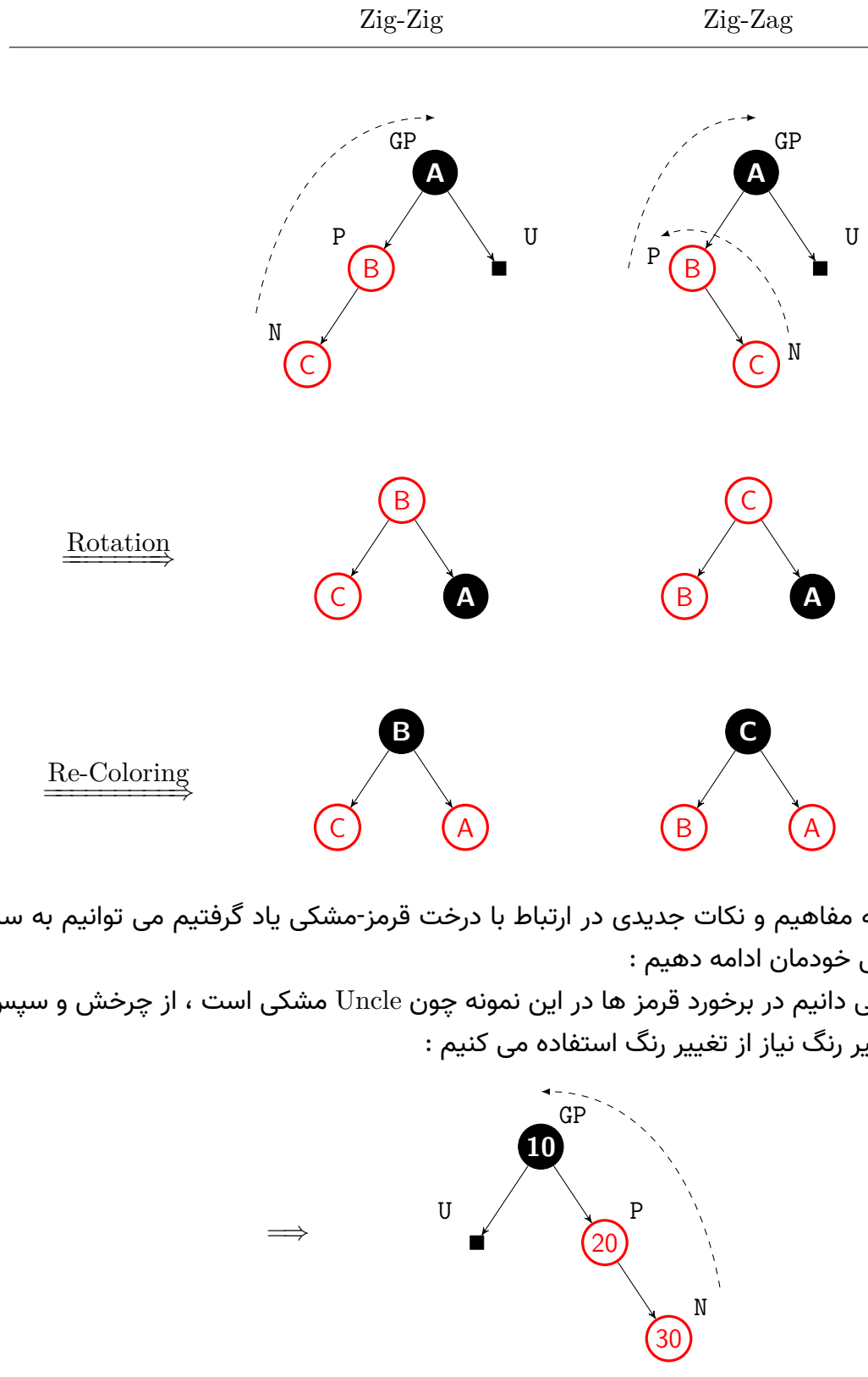
۲.۲.۲ حالت دوم Zig-Zag



به خاطر اینکه نود ریشه ی درخت قرمز است آن را به رنگ مشکی تغییر می دهیم .

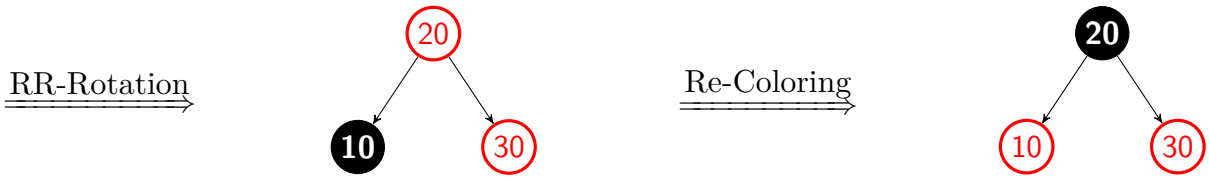


حالت های زیر نیز می توانند رخ دهند :

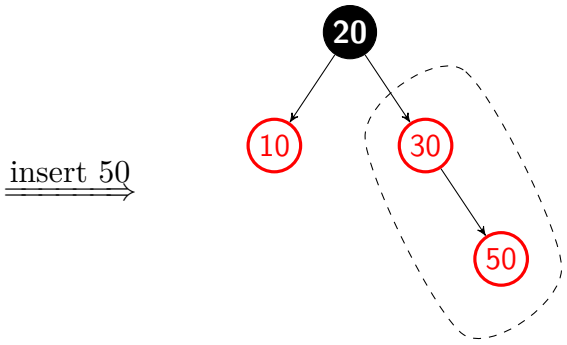


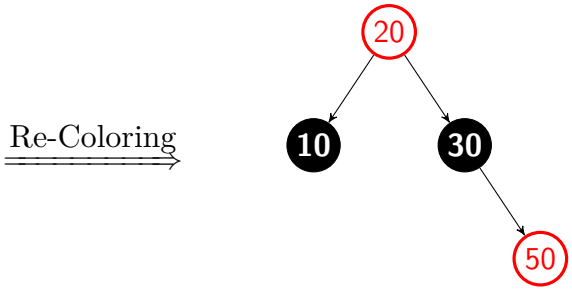
حالا که مفاهیم و نکات جدیدی در ارتباط با درخت قرمز-مشکی یاد گرفتیم می توانیم به ساخت درخت قبلی خودمان ادامه دهیم :

حال می دانیم در برخورد قرمز ها در این نمونه چون Uncle مشکی است ، از چرخش و سپس در صورت تغییر رنگ نیاز از تغییر رنگ استفاده می کنیم :



سپس عنصر 50 را اضافه می کنیم ، به خاطر اینکه این عنصر تازه وارد است به رنگ قرمز می باشد ، درج این عنصر مشکل **برخورد قرمزها** را به وجود می آورد ، از آنجایی که Uncle به رنگ قرمز است از روش تغییر رنگ استفاده می کنیم .

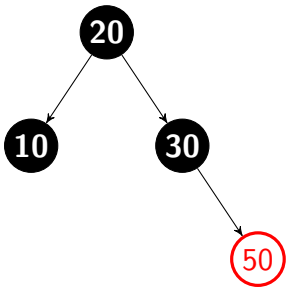




چون بعد از تغییر رنگ ، نود (Grand Parent) همان ریشه ی درخت بود آن را به رنگ مشکی تغییر می دهیم .

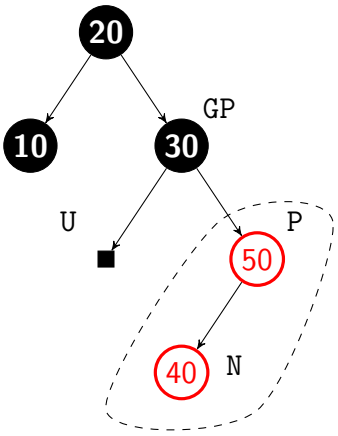
```
if ( GP == root ) {
   Re-Color GP to Black
}
```

Re-Color root to Black

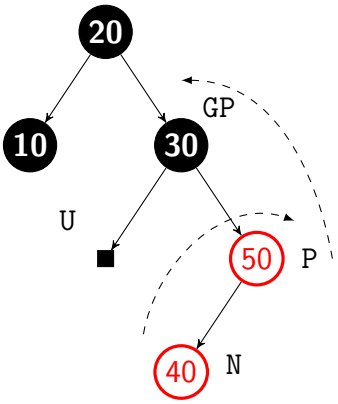


سپس عنصر 40 را اضافه می کنیم ، به خاطر اینکه این عنصر تازه وارد است به رنگ قرمز می باشد ، درج این عنصر مشکل **برخورد قرمزها** را به وجود می آورد ، از آنجایی که Uncle به رنگ مشکی است از روش چرخش استفاده می کنیم .

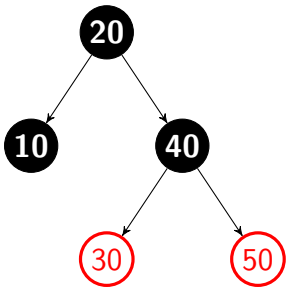
insert 40



RL-Rotation

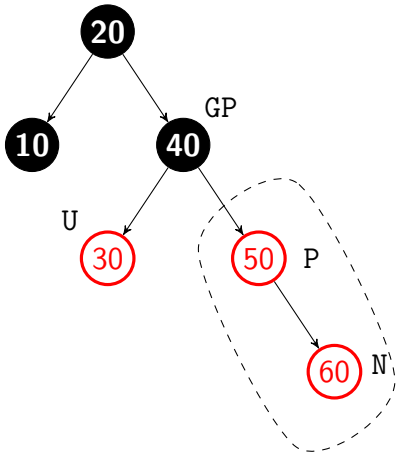


after RL-Rotation

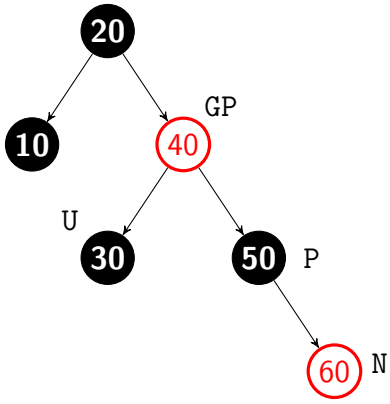


سپس عنصر 60 را اضافه می کنیم ، به خاطر اینکه این عنصر تازه وارد است به رنگ قرمز می باشد ، درج این عنصر مشکل **برخورد قرمزها** را به وجود می آورد ، از آنجایی که Uncle به رنگ قرمز است از روش تغییر رنگ استفاده می کنیم .

insert 60

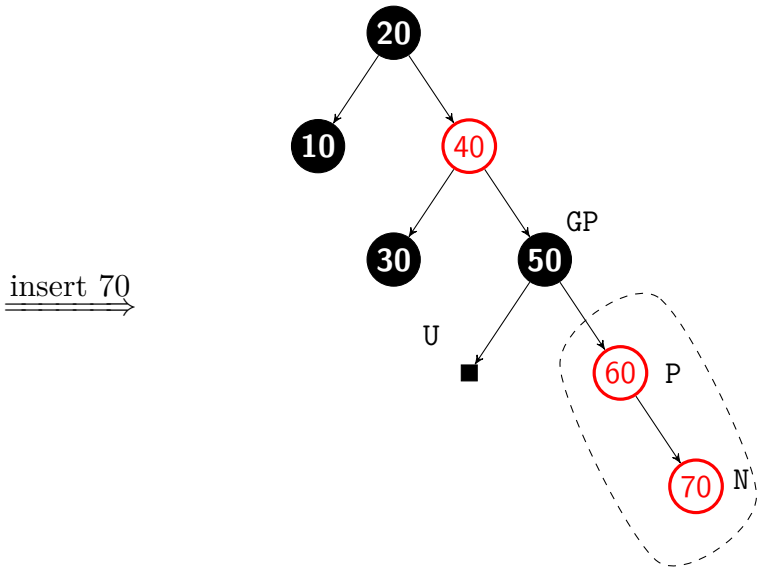


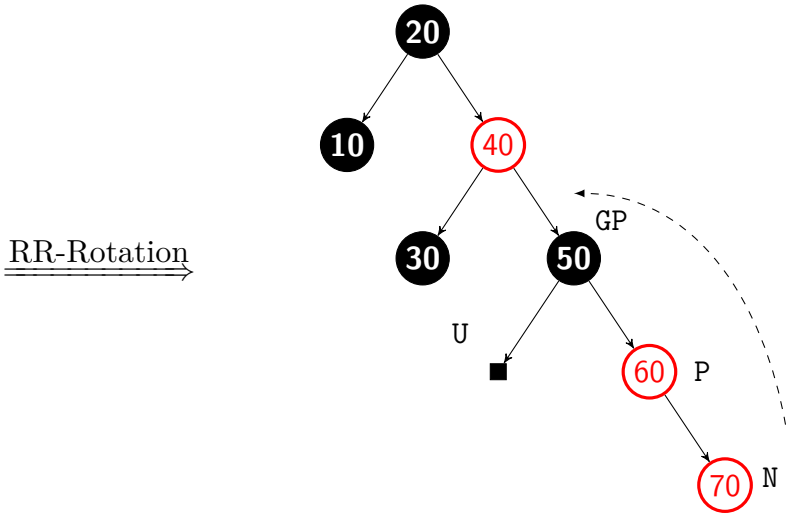
Re-Coloring



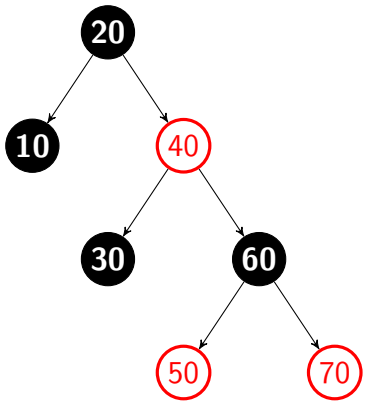
حال برای نود Grand Parent چک می کنیم که بعد از تغییر رنگ مشکل برخورد قرمزها به وجود آمده است یا خیر و این کار را تا زمانی که ببینیم برخورد قرمز وجود ندارد ادامه می دهیم .
در این نمونه Grand Parent مشکی می باشد و بنابراین برخورد قرمزها به وجود نمی آید .

سپس عنصر 70 را اضافه می کنیم ، به خاطر اینکه این عنصر تازه وارد است به رنگ قرمز می باشد ، درج این عنصر مشکل **برخورد قرمزها** را به وجود می آورد ، از آنجایی که Uncle به رنگ مشکی است از روش چرخش استفاده می کنیم .



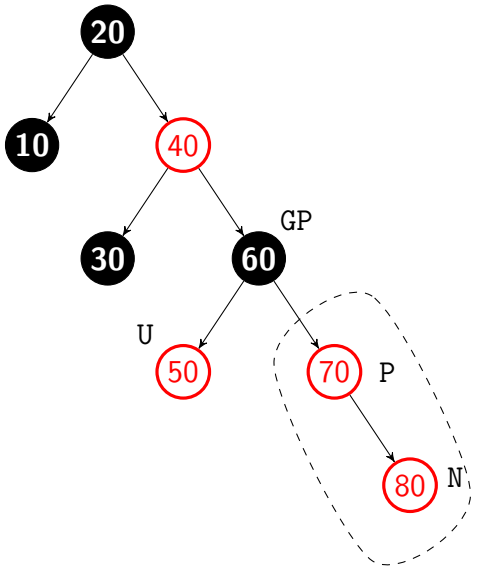


after RR-Rotation

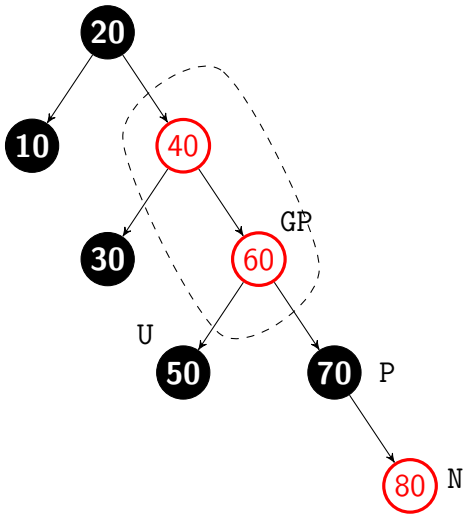


سپس عنصر 80 را اضافه می کنیم ، به خاطر اینکه این عنصر تازه وارد است به رنگ قرمز می باشد ، درج این عنصر مشکل **برخورد قرمزها** را به وجود می آورد ، از آنجایی که Uncle به رنگ قرمز است از روش تغییر رنگ استفاده می کنیم .

insert 80

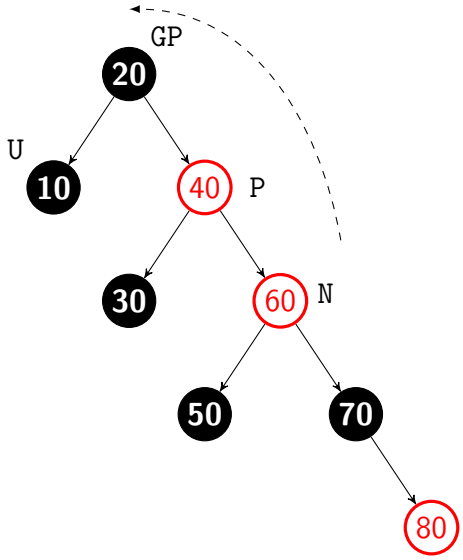


Re-Coloring

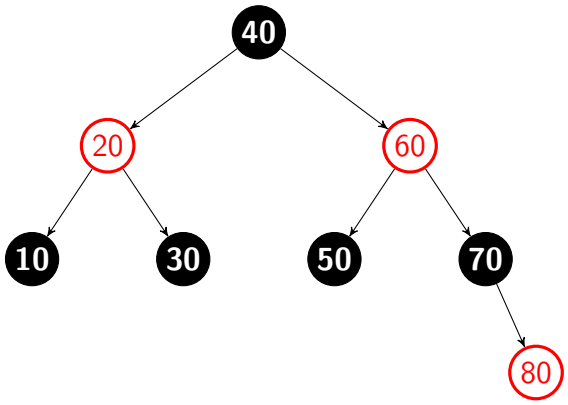


حال برای نود Grand Parent چک می کنیم که بعد از تغییر رنگ مشکل برخورد قرمزها به وجود آمده است یا خیر و این کار را تا زمانی که ببینیم برخورد قرمز وجود ندارد ادامه می دهیم .
در این نمونه Grand Parent قرمز می باشد و بنابراین برخورد قرمزها به وجود می آید ، از آنجایی که Uncle جدید به رنگ مشکی است از روش چرخش استفاده می کنیم .

Re-Coloring

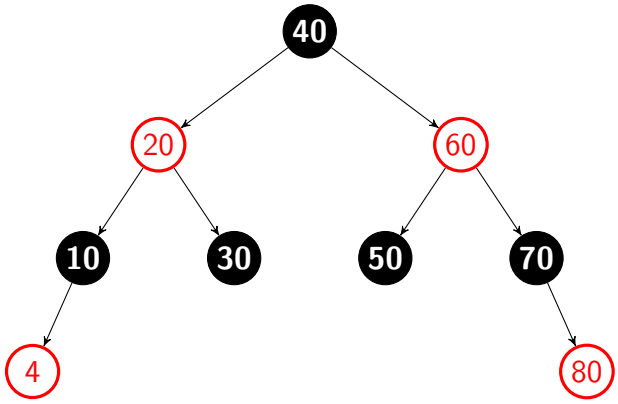


RR-Rotation



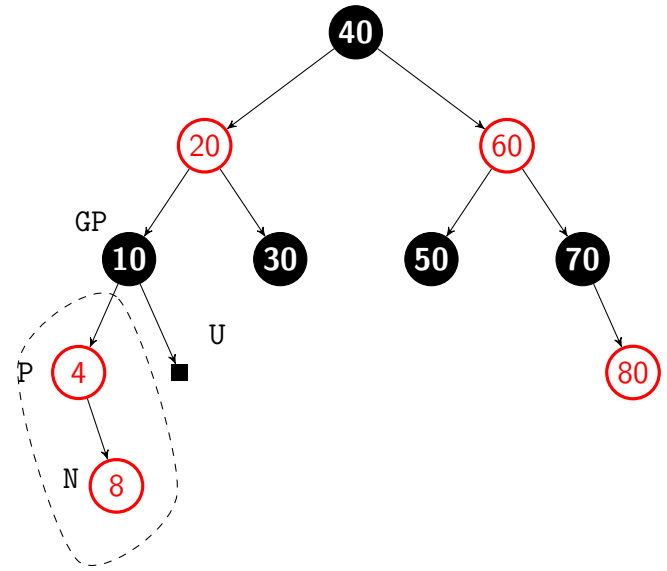
سپس عنصر 4 را اضافه می کنیم ، به خاطر اینکه این عنصر تازه وارد است به رنگ قرمز می باشد ، اضافه کردن این عنصر هیچ مشکلی در درخت ما به وجود نمی آورد .

insert 4

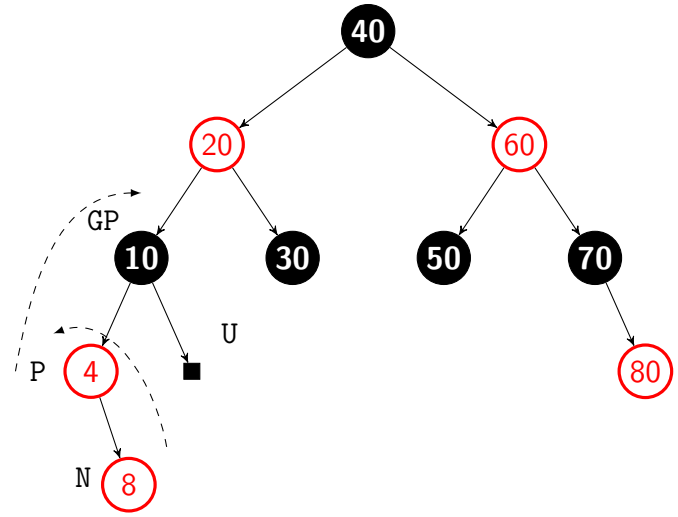


سپس عنصر 8 را اضافه می کنیم ، به خاطر اینکه این عنصر تازه وارد است به رنگ قرمز می باشد ، درج این عنصر مشکل **برخورد قرمزها** را به وجود می آورد ، از آنجایی که Uncle به رنگ مشکی است از روش چرخش استفاده می کنیم .

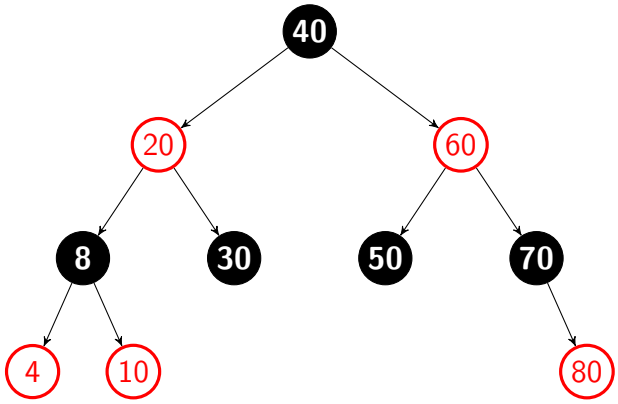
insert 8 ⟹



LR-Rotation ⟹



after LR-Rotation ⟹



۳ حالت های مختلف حذف از درخت قرمز-مشکی

- حذف از درخت قرمز-مشکی همانند حذف از درخت جستجوی دودویی است با این تفاوت که ممکن است شامل عملیات چرخش یا تغییر رنگ باشد .

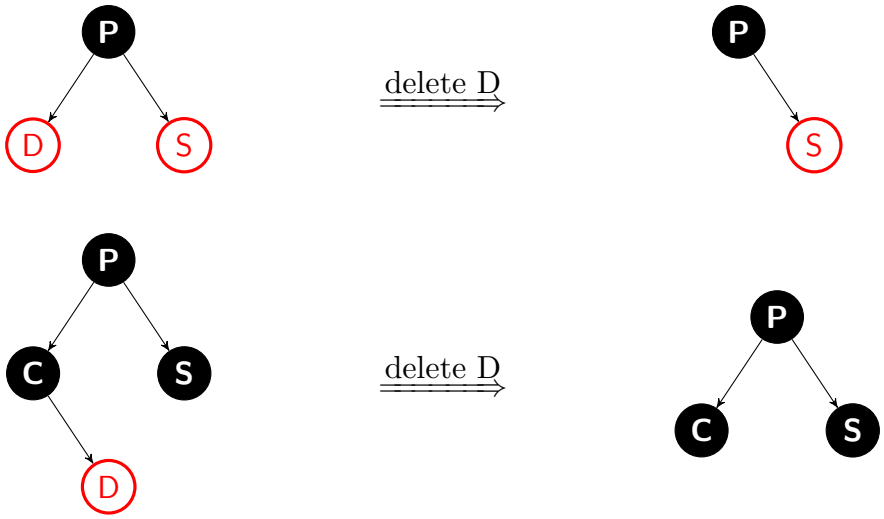
۱.۳ حذف در درخت جستجوی دودویی چطور رخ می دهد ؟

در درخت جستجوی دودویی ما کل نود را حذف نمی کنیم بلکه فقط مقدار نود را جایگزین می کنیم و در واقع نودی که حذف می شود کمترین مقدار بزرگتر یا بزرگترین مقدار کمتر می باشد

- با تعریفی که در بالا از حذف از درخت جستجوی دودویی ارائه دادیم ، نود حذف شده از درخت یا برگ می باشد و فرزندی ندارد و یا اینکه شامل تنها یک فرزند می باشد .

۲.۳ حالت اول

- وقتی در نظر داریم که یک نود قرمز را از درخت حذف کنیم ، خیلی ساده آن را حذف می کنیم و در صورتی که فرزندی داشت (که قطعاً مشکی است) آن را با نود حذف شده جایگزین می کنیم .



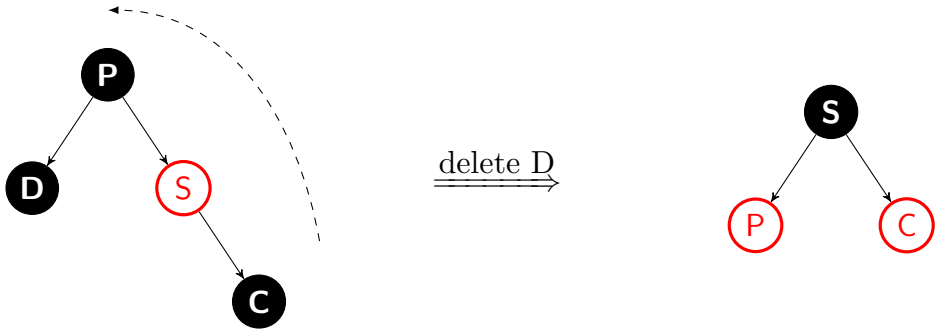
- چرا حذف نود های قرمز از درخت قرمز-مشکی مشکلی ندارد ؟

جواب : زیرا در درخت قرمز-مشکی تعداد نود های مشکی در طور یک مسیر از هر نود مهم است و باید یکسان باشد ، بنابراین اگر یک نود قرمز از درخت حذف شود ، تاثیری بر روی درخت ندارد .

- قسمت مشکل در حذف نود از درخت قرمز-مشکی کجاست؟ جواب : وقتی نود مشکی است .

۳.۳ حالت دوم

وقتی می خواهید که یک نود مشکی را از درخت حذف کنید ، همسایه ی آن نود را چک کنید ، اگر همسایه اش قرمز بود آنگاه نود را حذف کنید و باید چرخش مناسب را انجام دهید .



۴.۳ حالت سوم

وقتی نودی که میخواهیم حذف کنیم مشکی باشد و همسایه ی آن نود هم مشکی باشد ، چند گزینه پیش رو داریم :

۱. اگر هر دو فرزند همسایه مشکی باشد ، از روش تغییر رنگ استفاده می کنیم .

- بستگی به اینکه همسایه ، چند فرزند دارد ، حالت های مختلفی داریم

۱. اگر فرزندهای همسایه قرمز باشد ، آنگاه روش چرخش را انجام می دهیم

- بستگی به اینکه همسایه چند فرزند دارد ، حالت های مختلفی داریم

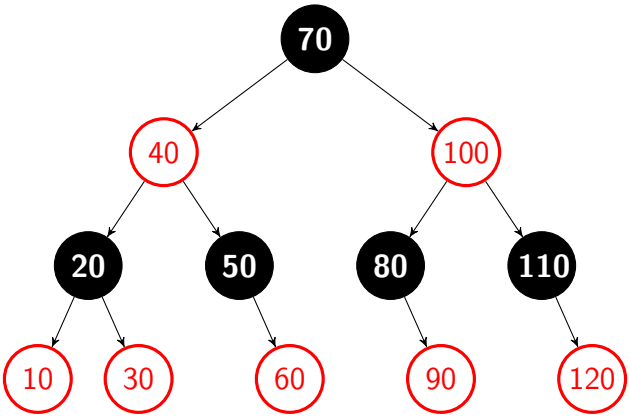
۵.۳ خلاصه ای از حالت های حذف از درخت قرمز -مشکی

Sibling is Red $\Longrightarrow$ Rotate

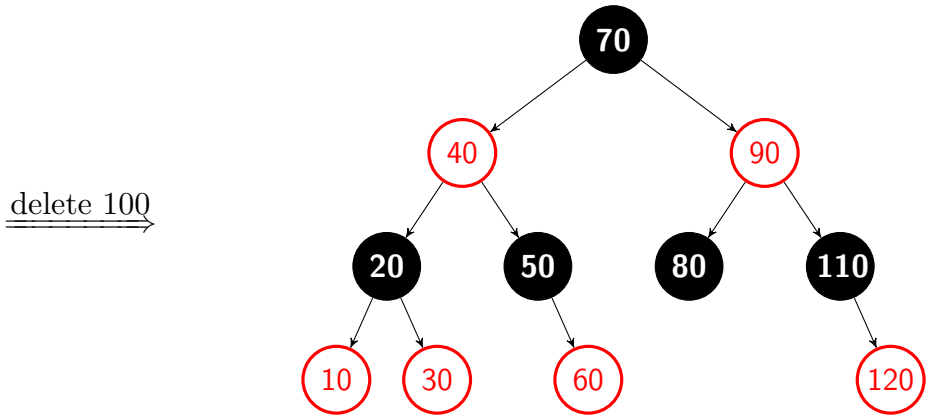
Sibling is Black
Children are Red
$\Longrightarrow$ Rotate

Sibling is Black
Children are Black
$\Longrightarrow$ Re-Color

۴ نمونه هایی از حذف از درخت قرمز-مشکی

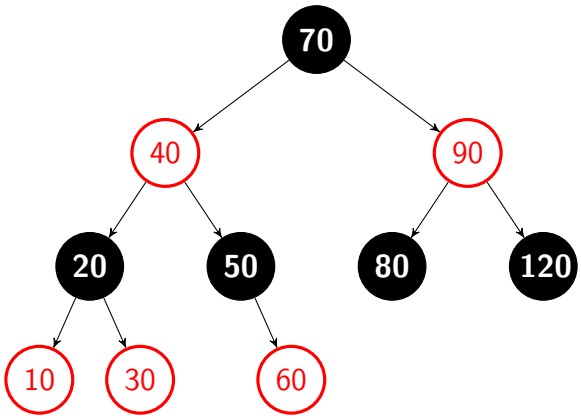


حذف 100 منجر به حذف 90 که بیشترین مقدار کمتر می باشد می شود و چون 90 قرمز است بنابراین حذف آن مشکلی به وجود نمی آورد .



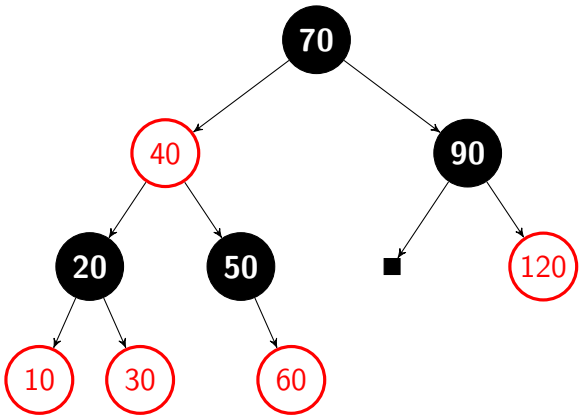
حذف 110 منجر به حذف 120 که بیشترین مقدار کمتر می باشد می شود و چون 120 قرمز است بنابراین حذف آن مشکلی به وجود نمی آورد .

delete 110 ⟹



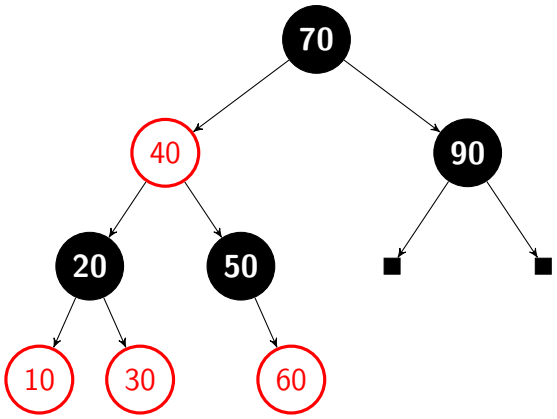
وقتی می خواهیم 80 را حذف کنیم ، همسایه اش مشکی است و فرزند های همسایه نیز مشکی است (چون تهی است) بنابراین تغییر رنگ می دهیم

delete 80 ⟹

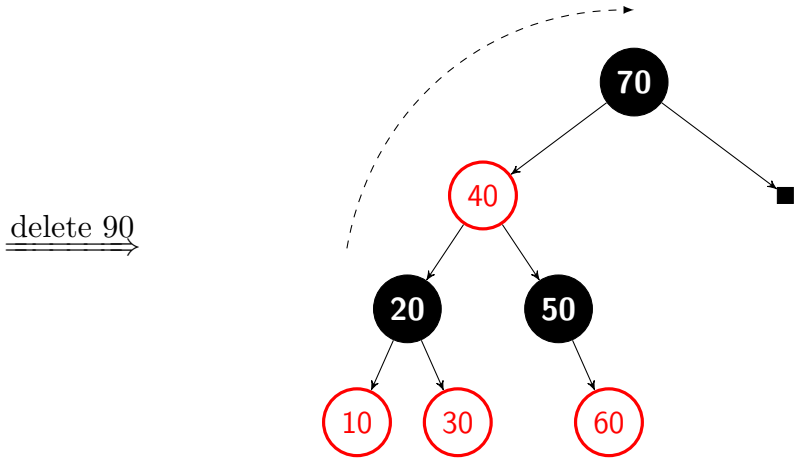


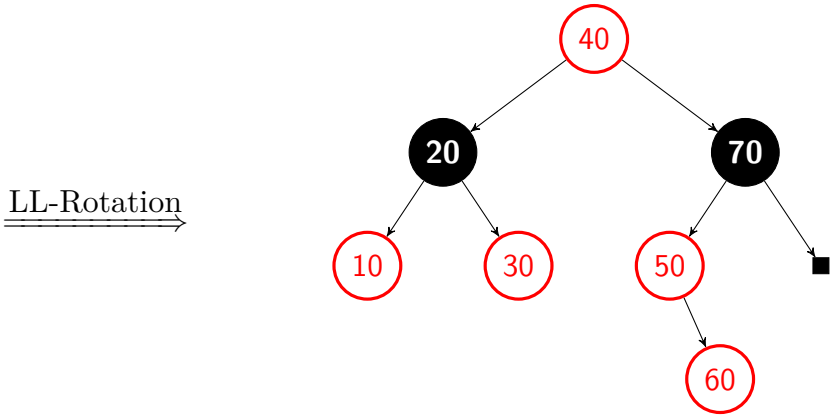
نود 120 قرمز است بنابراین حذف آن مشکلی به وجود نمی آورد .

delete 120 ⟹



وقتی می خواهیم 90 را حذف کنیم ، چون 90 مشکی است به همسایه ی آن نگاه می کنیم و چون همسایه ی آن که 40 می باشد قرمز است بنابراین روش چرخش را انجام می دهیم .

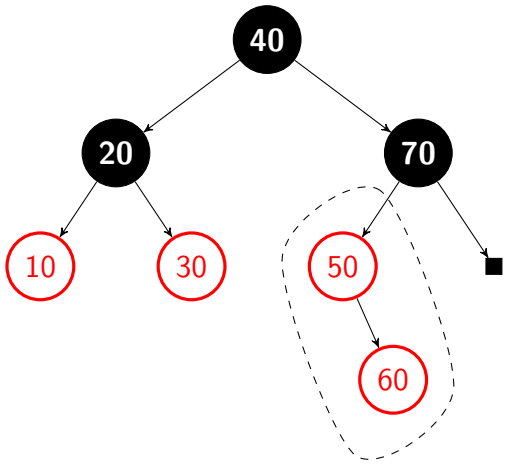




بعد از چرخش می بینیم که نود ریشه ی درخت که 40 می باشد به رنگ قرمز درآمده است ، بنابراین باید رنگ آن به رنگ مشکی تغییر یابد .

```
if ( GP == root ) {
   Re-Color GP to Black
}
```

Re-Color root to Black



بعد از چرخش میبینیم که برخورد قرمز ها بین نودهای 50 و 60 به وجود آمده است و Uncle نیز مشکی می باشد بنابراین روش چرخش را انجام می دهیم .

LL-Rotation