درخت جستجوی دودویی متوازن

فرشته دهقانی

سرفصل مطالب

- اهمیت درخت دودویی جستجوی متوازن
- درخت قرمز سیاه
  - چرخش
  - درج
  - حذف

اهمیت متوازن بودن

- می‌دانیم که اکثر عملیات روی د.د.ج‌ها (درج، حذف، جستجو، پیدا کردن عنصر قبلی و بعدی و پیدا کردن عنصر کمینه و بیشینه) از مرتبه‌ی زمانی ارتفاع درخت
- ارتفاع بین n و logn
- د.د.ج‌های متوازن با انجام تغییرات و چرخش‌هایی در حین انجام عملیات مورد نیاز، تضمین میکنند ارتفاع درخت از O(logn) بماند

انواع BST متوازن

- AVL tree
- Red-black tree
- Splay tree
- Treap

خواص درخت قرمز-سیاه

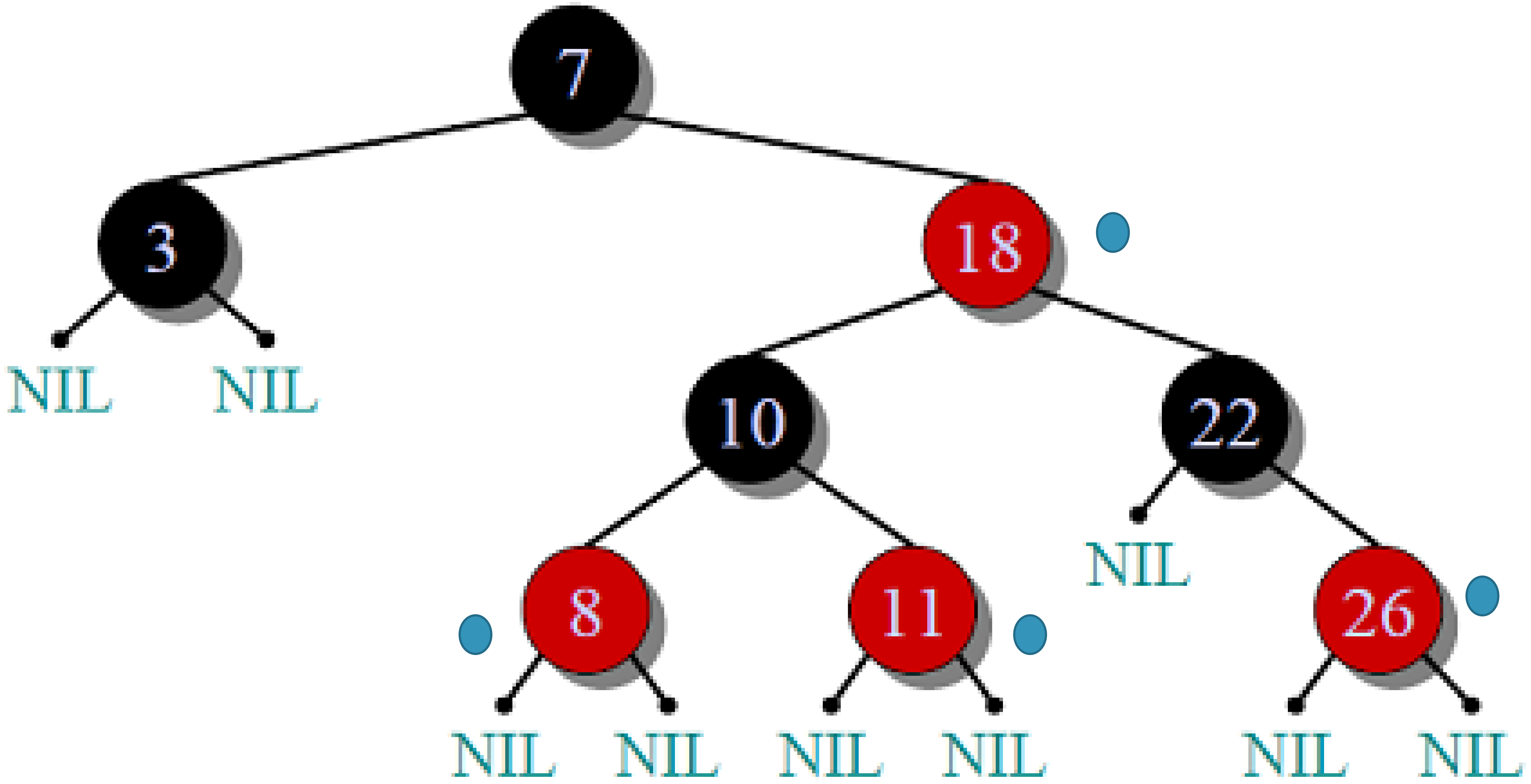
- هر راس یک رنگ دارد که یا سیاه است یا قرمز.

۱- راس **ریشه** همیشه **سیاه** است.

۲- هر راس غیر NIL دقیقا ۲ فرزند دارد (یعنی تمامی برگ ها NIL یا پوچ هستند)

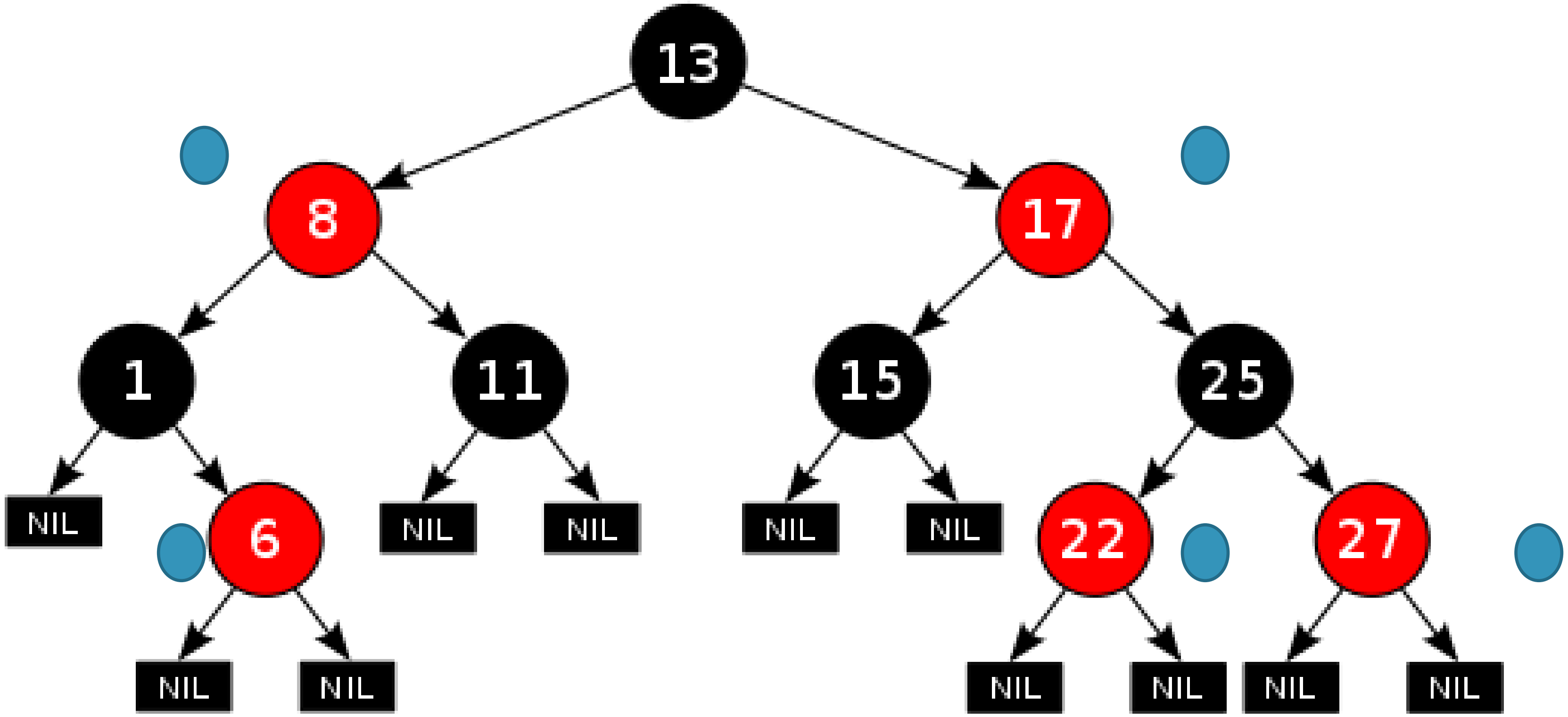
۳- اگر راسی قرمز باشد، حتما هر دو فرزند آن سیاه هستند. (**دو گره قرمز مجاور نیستند**)

۴- برای هر راس، هر مسیر ساده از آن به یکی از برگ‌های زیر درختش **دارای تعداد مساوی از راس‌های سیاه** است.



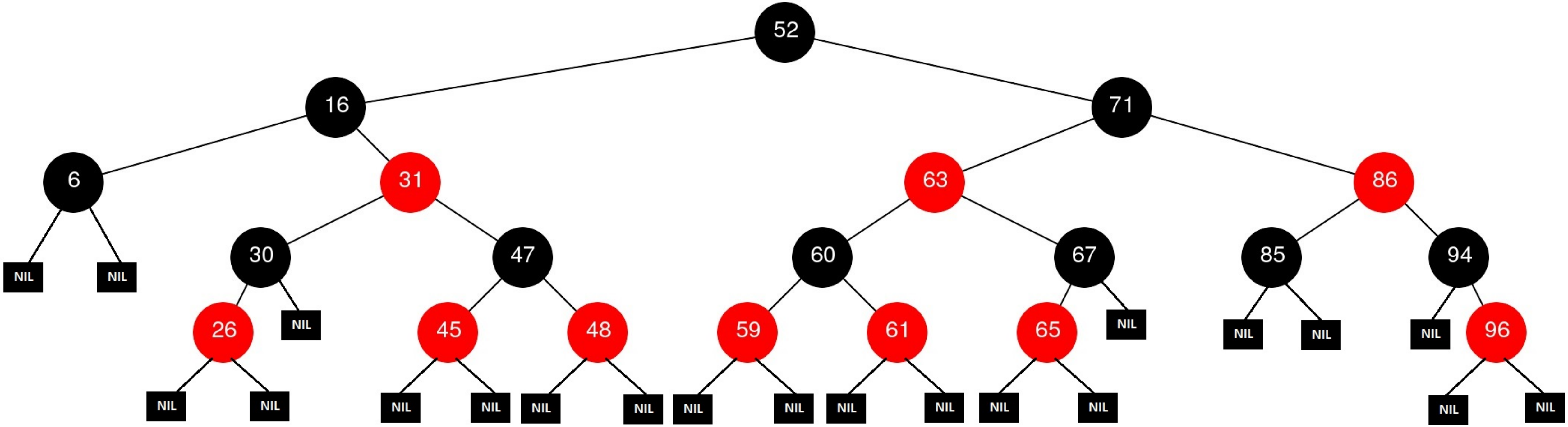
7
3
18
NIL
NIL
10
22
NIL
8
11
26
NIL
NIL
NIL
NIL
NIL
NIL

مثال



13
8
17
1
11
15
25
NIL
6
NIL
NIL
NIL
NIL
22
27
NIL
NIL
NIL
NIL
NIL
NIL

مثال



```
A chain of 3 nodes is nodes is not possible in Red-Black Trees.
Following are NOT Red-Black Trees
        30             30              30
       / \            /  \            /  \
      20  NIL       20    NIL       20    NIL
     / \           /  \            /  \
    10  NIL      10   NIL        10   NIL
Violates         Violates         Violates
Property 4.     Property 4       Property 3

Following are different possible Red-Black Trees with above 3 keys
         20                           20
        /   \                        /  \
      10     30                    10     30
     /  \   /  \                  /  \   /  \
   NIL  NIL NIL NIL             NIL  NIL NIL NIL
```

نتیجه

- این خواص باعث می‌شوند طول بلندترین مسیر از ریشه به یکی از برگ‌ها حداکثر دو برابر طول کوتاه‌ترین مسیر از ریشه به یکی از برگ‌ها باشد که توازن خوبی را در درخت ایجاد می‌کند. (حداکثر ارتفاع $\geq 2\log(n+1)$)

- تعداد رئوس سیاه مسیرها که مساوی است، از آنجایی که فرزندان رئوس قرمز حتما سیاه هستند، تعداد رئوس قرمز حداقل ۰ و حداکثر به تعداد رئوس سیاه است که خاصیت بالا را نتیجه می‌دهد

چرخش

- برای حفظ خواص درخت قرمز سیاه (در هنگام درج و حذف) از دو عمل تغییر رنگ راس‌ها و چرخش به راست یا چپ استفاده می‌شود.
- خواص گفته شده برای درخت قرمز-سیاه باید پس از هر عمل بر روی این درخت باز هم برقرار باشند.
- چرخش به راست در واقع تغییر جهت یالی از درخت که به سمت چپ بوده است به سمت راست با یک چرخش و سپس انجام تغییرات لازم است تا منطق درخت جست و جو بودن در درخت حفظ شود.
- دقت کنید که در یک چرخش خاصیت‌های دیگر درخت قرمزسیاه الزاما حفظ نمی‌شوند. (چرخش در واقع برای درخت‌های دودویی جست و جو تعریف می‌شود و منطق درخت جست‌وجو بودن آن‌ها را بر هم نمی‌زند)

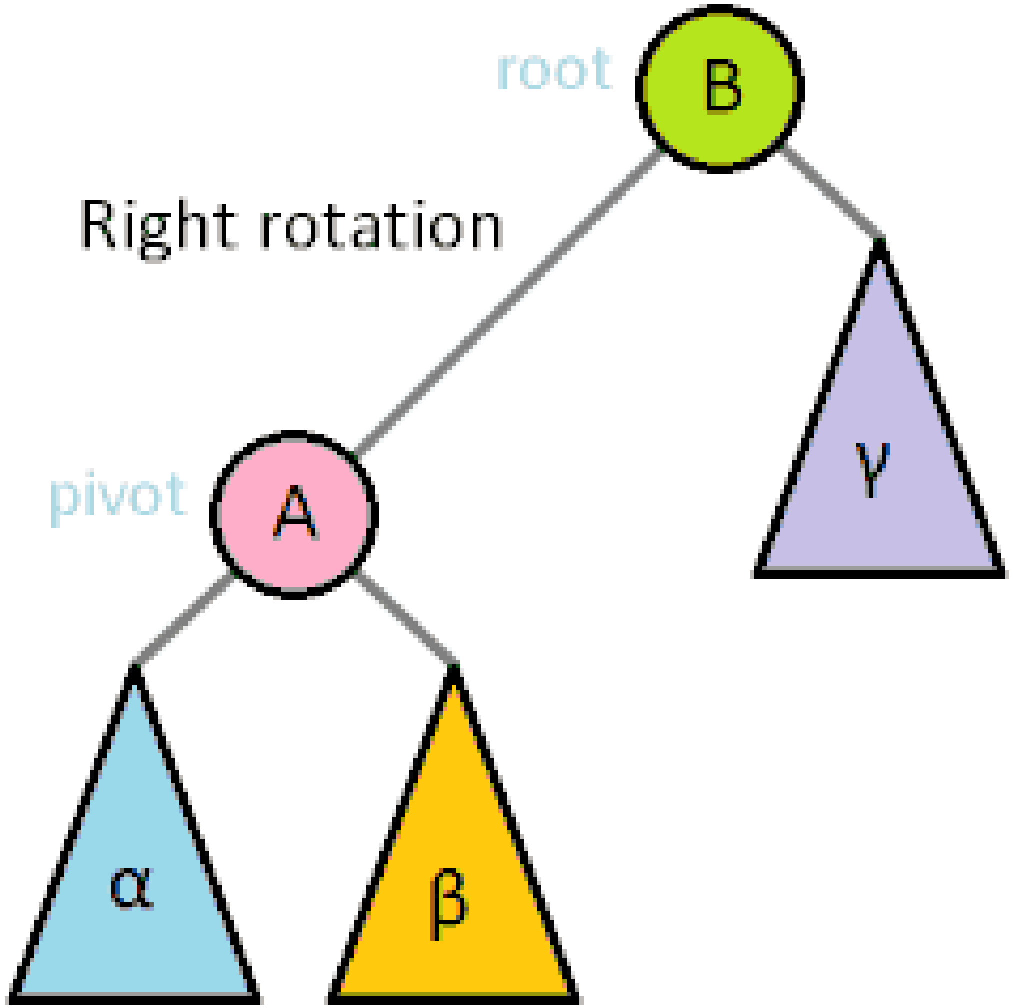
چرخش

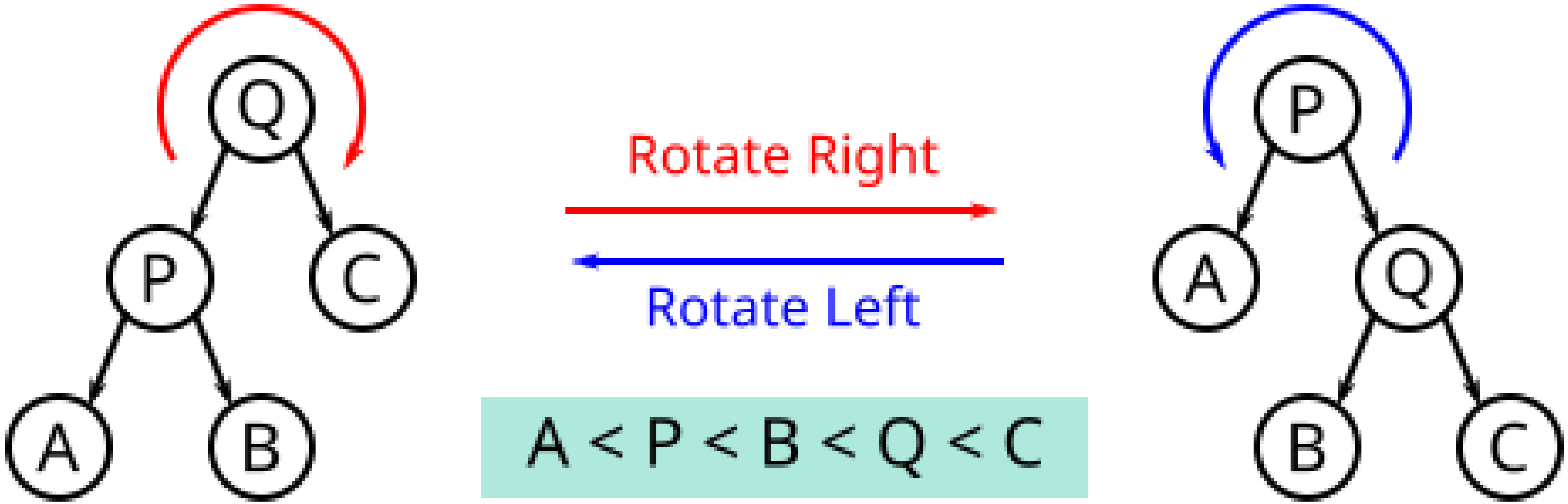
در چرخش به راست روی A فرض می‌کنیم فرزند چپ آن (مثلا α) پوچ (NIL) نیست، اگر فرزند سمت راست A را β بنامیم، و و فرزند راست B را γ بنامیم، با چرخش به راست:

1. A ریشه‌ی این زیردرخت می‌شود
2. B فرزند راست A می‌شود.
3. β (و زیردرختش) فرزند چپ B می‌شود.

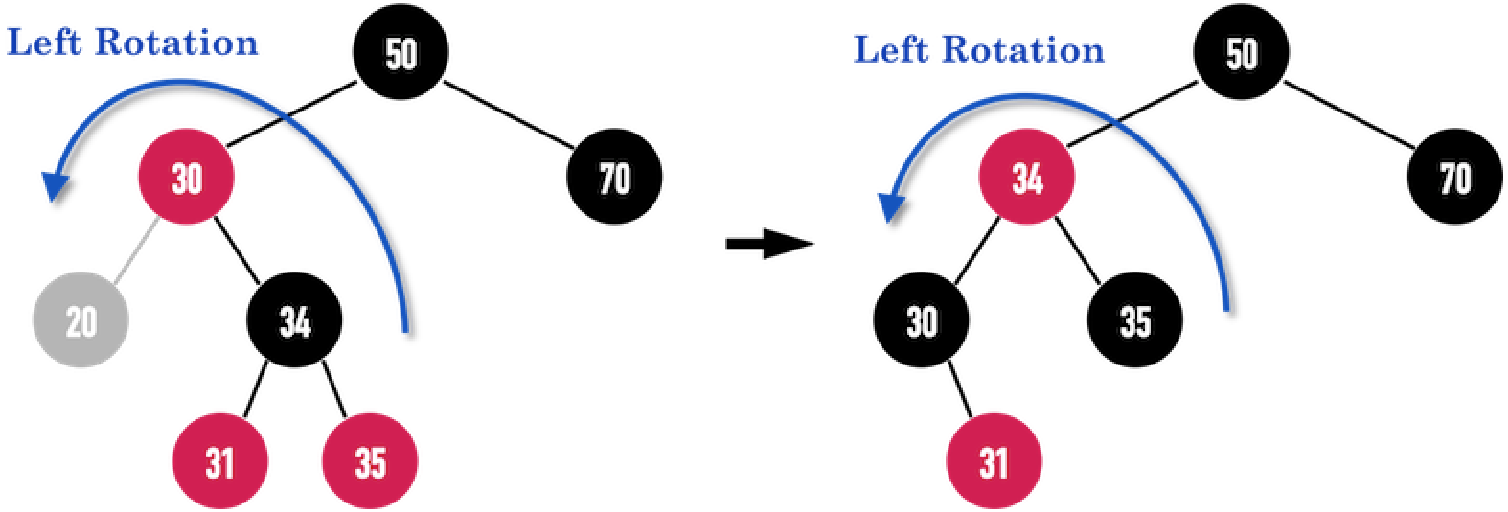
چرخش به چپ مشابه است.

چرخش

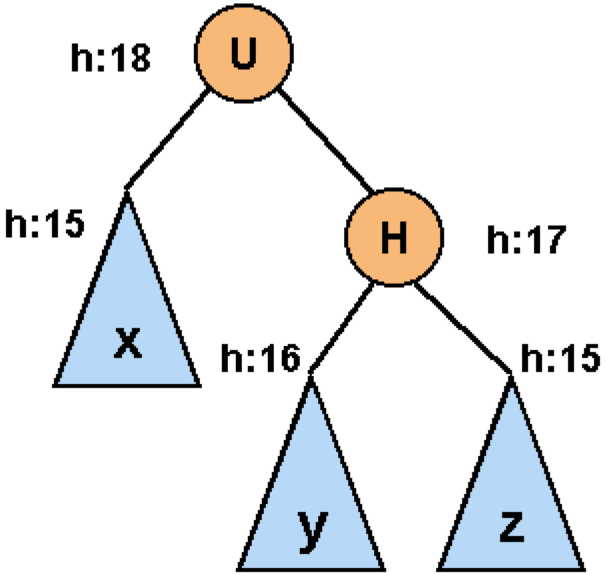



Q
P
C
A
B
Rotate Right
Rotate Left
A < P < B < Q < C
P
A
Q
B
C

مثال چرخش چپ



Left Rotation
50
30
70
20
34
31
35
Left Rotation
50
34
70
30
35
31



h:18
U
h:15
H
h:17
x
h:16
h:15
y
z

درج

برای درج عنصر **x** در یک درخت قرمز-سیاه، ابتدا آن را به همان روشی که در د.د.ج عادی درج می‌کردیم درج می‌کنیم و رنگش را **قرمز** می‌کنیم و **۲ فرزند سیاه پوچ** (NIL) برای آن در نظر می‌گیریم.

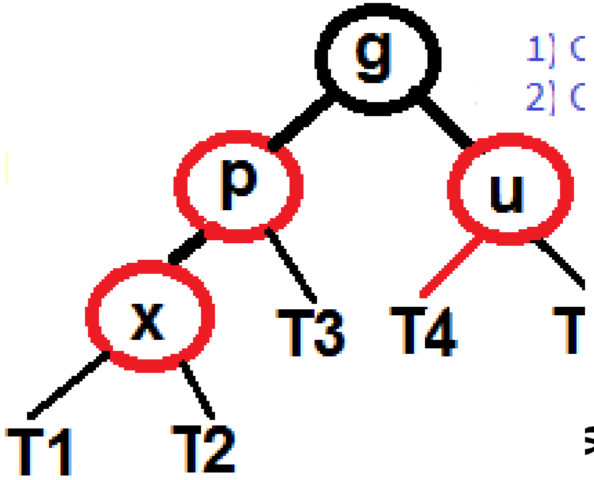
.

درج

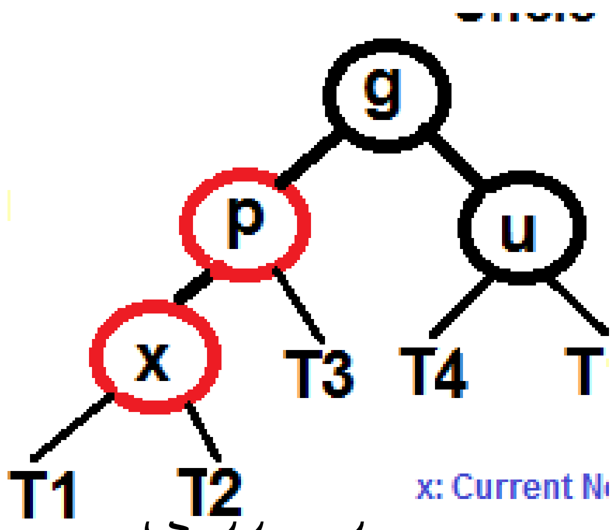
دو تا از قواعد درخت قرمز-سیاه ممکن است نقض شده باشند:

۱- ممکن است راس پدر x ، (P[x]) قرمز باشد

۲- ممکن است فرزندهای سیاه اضافه شده باعث شده باشند که برای یک راس، دو مسیر ساده از آن به دوتا از برگ‌های زیردرختش دارای تعداد متفاوتی از راس‌های سیاه باشند



درج



ر ...رتی ...ره اضافه شده، ریشه باشد، رنگ آن مشکی م...

در صورتی که گره اضافه شده (x) ریشه نباشد و رنگ گره پدر(p[x]) آن قرمز باشد، دو حالت پیش می آید:

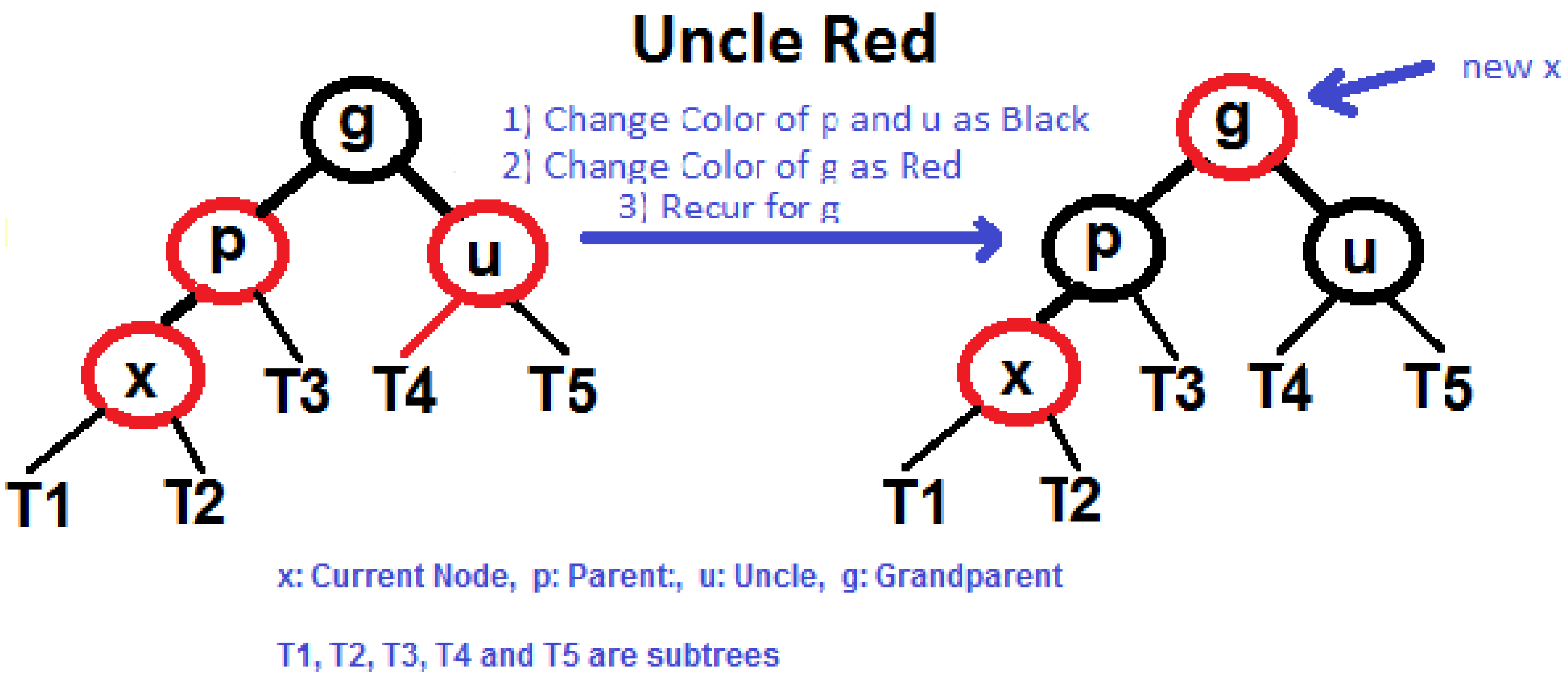
درج –۱ : رنگ گره عمو قرمز باشد

درج –۲ رنگ گره عمو مشکی باشد

درج – ۱ : رنگ گره عمو قرمز باشد

- رنگ گره پدر و عمو مشکی می شود
- رنگ گره پدربزرگ (p[p[x]]) قرمز میشود
- گره مورد نظر به گره پدریزرگ x=p[p[x]] تغییر یافته و دو عمل بالا برای آن تکرار می شود

درج - ١


Uncle Red
1) Change Color of p and u as Black
2) Change Color of g as Red
3) Recur for g
new x
g
p
u
x
T1
T2
T3
T4
T5
x: Current Node, p: Parent:, u: Uncle, g: Grandparent
T1, T2, T3, T4 and T5 are subtrees

درج۲- رنگ گره عمو سیاه باشد

درج ۲- رنگ گره عمو مشکی باشد

چهار حالت :

۱- x فرزند چپ p[x] و p[x] فرزند چپ p[p[x]]

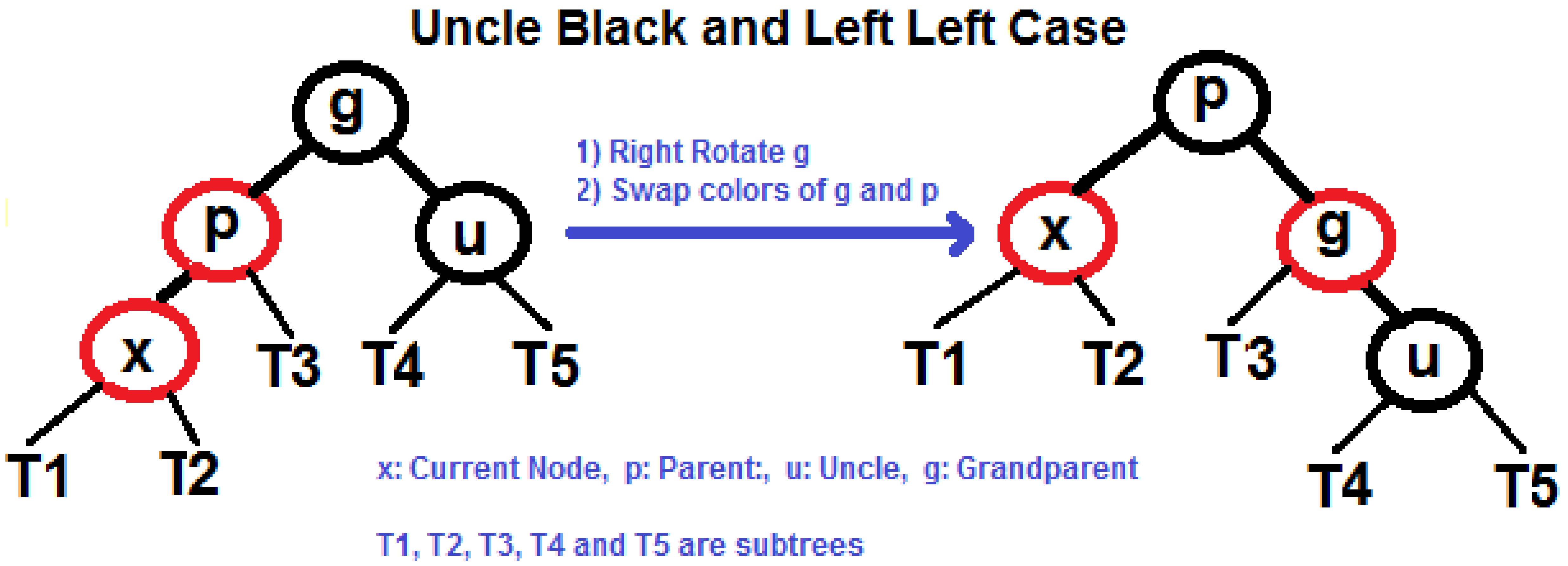
۲- x فرزند راست p[x] و p[x] فرزند چپ p[p[x]]

۳- عکس حالت یک

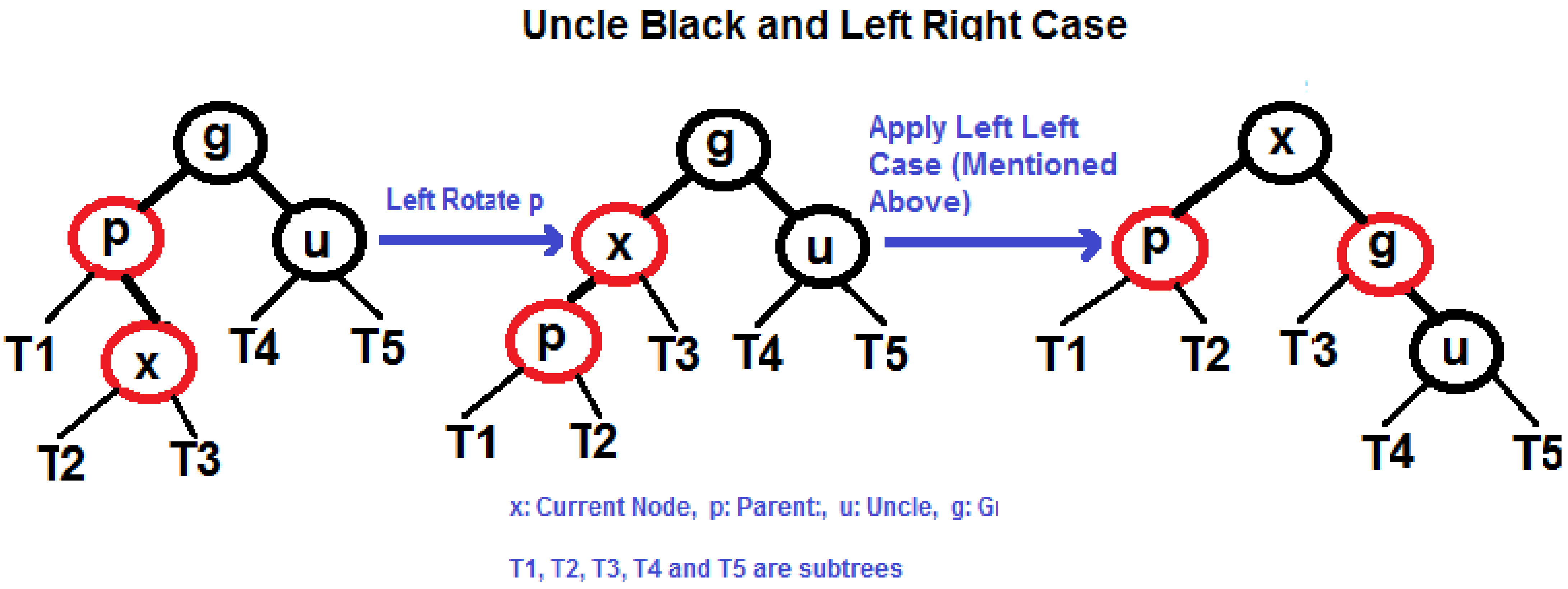
۴- عکس حالت دو

درج ۲-۱ (LEFT-LEFT)

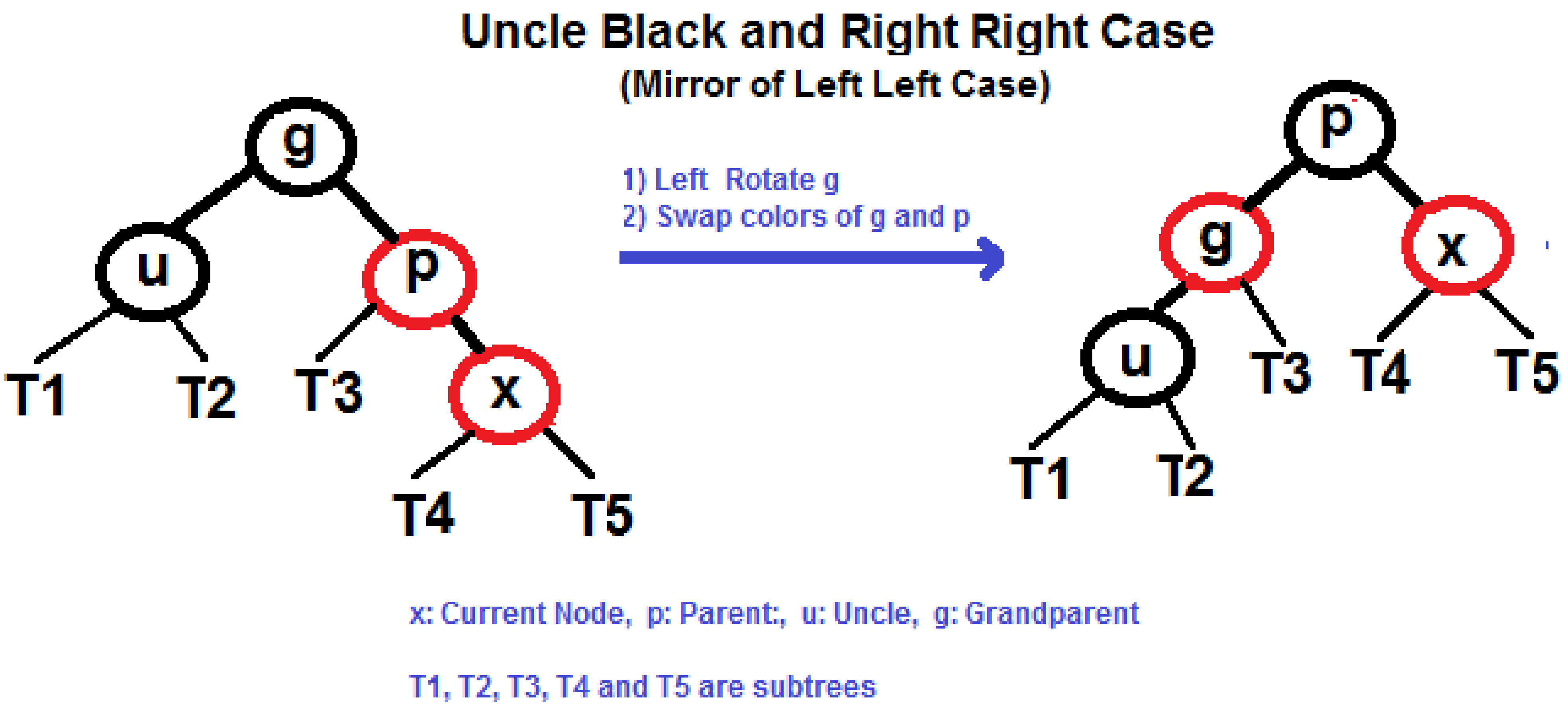
X فرزند چپ P[X] و P[X] فرزند چپ P[P[X]]



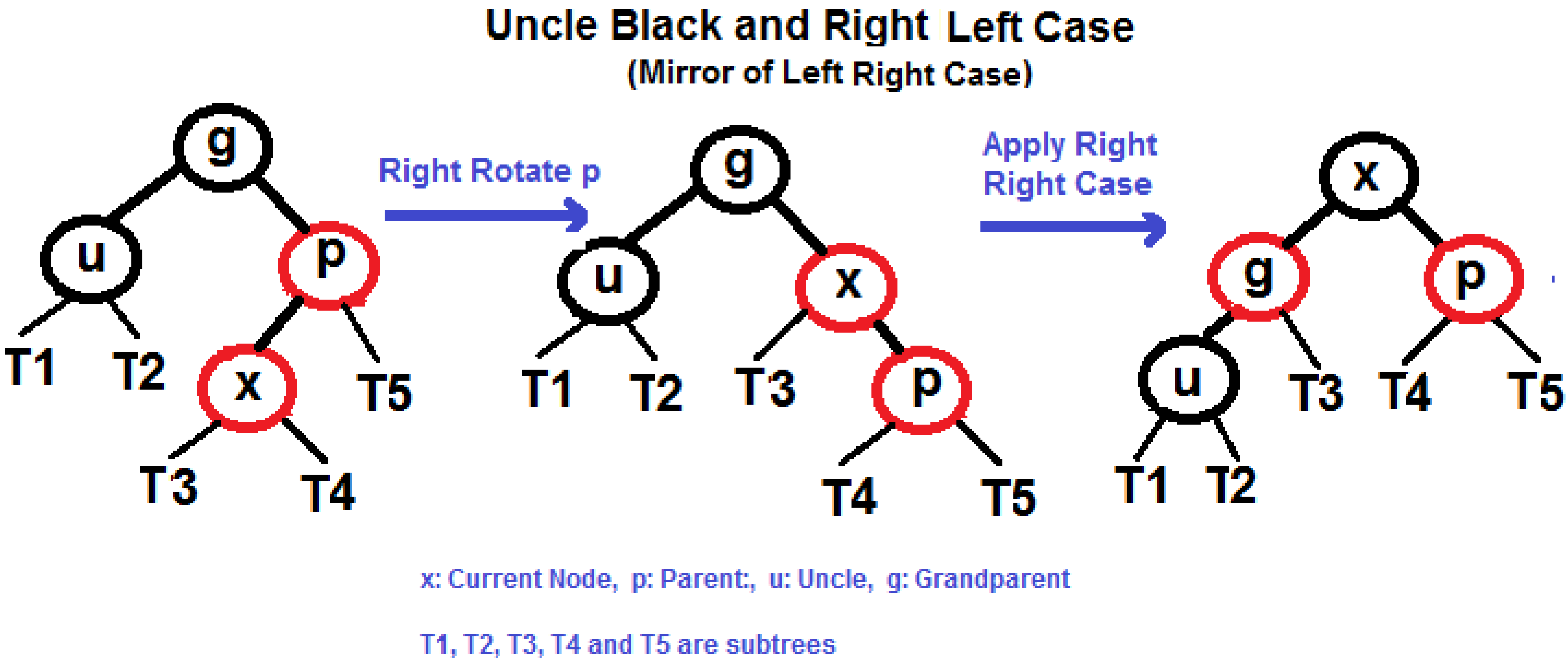
درج ۲-۲ (LEFT-RIGHT)
X فرزند راست P[X] و P[X] فرزند چپ P[P[X]]



درج ۲-۳ (RIGHT-RIGHT)
X فرزند راست P[X] و P[X] فرزند راست P[P[X]]



درج ۲–۴ (RIGHT-LEFT)
X فرزند چپ P[X] و P[X] فرزند راست P[P[X]]



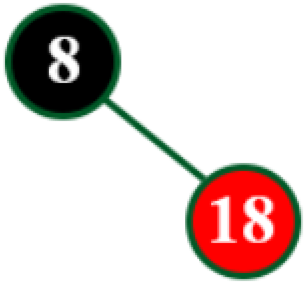
مثال

insert (8)

Tree is Empty. So insert newNode as Root node with black color.

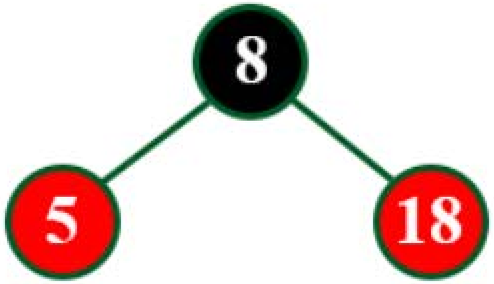
insert (18)

Tree is not Empty. So insert newNode with red color.



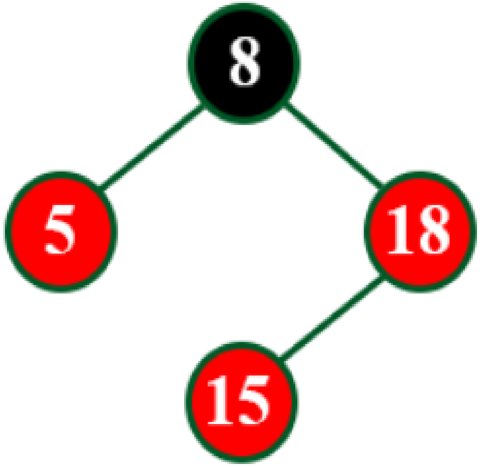
insert (5)

Tree is not Empty. So insert newNode with red color.



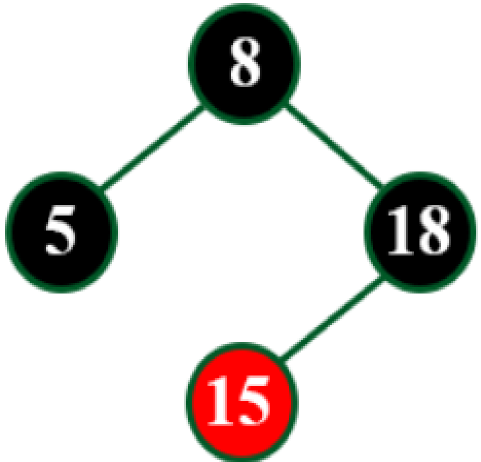
insert (15)

Tree is not Empty. So insert newNode with red color.



Here there are two consecutive Red nodes (18 & 15).
The newnode's parent sibling color is Red
and parent's parent is root node.
So we use RECOLOR to make it Red Black Tree.

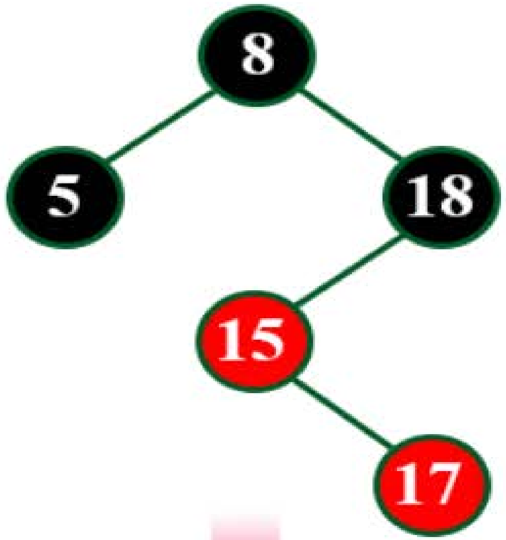
After RECOLOR



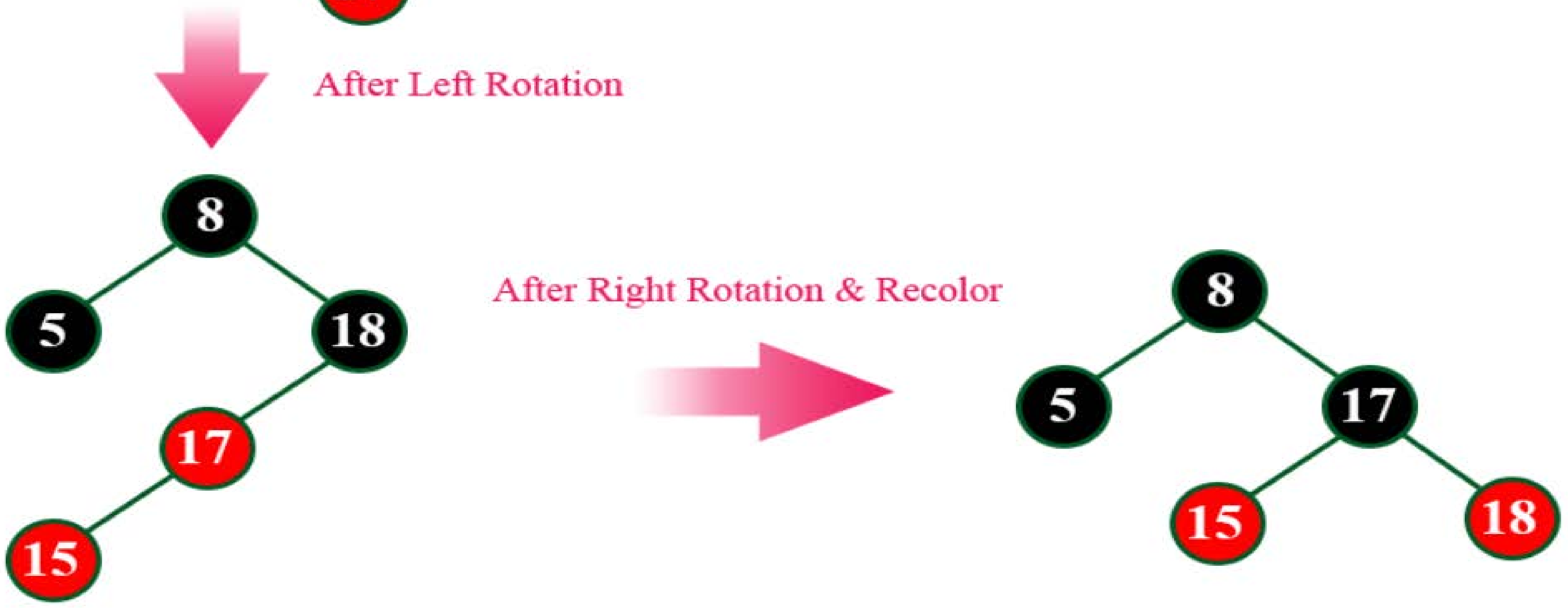
After Recolor operation, the tree is satisfying all Red Black Tree properties.

insert (17)

Tree is not Empty. So insert newNode with red color.

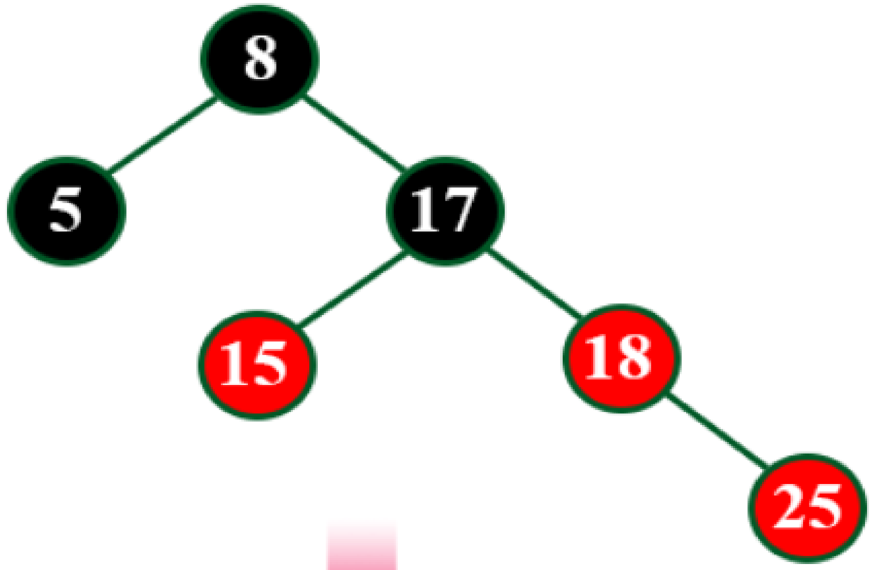


Here there are two consecutive Red nodes (15 & 17).
The newnode's parent sibling is NULL. So we need rotation.
Here, we need LR Rotation & Recolor.



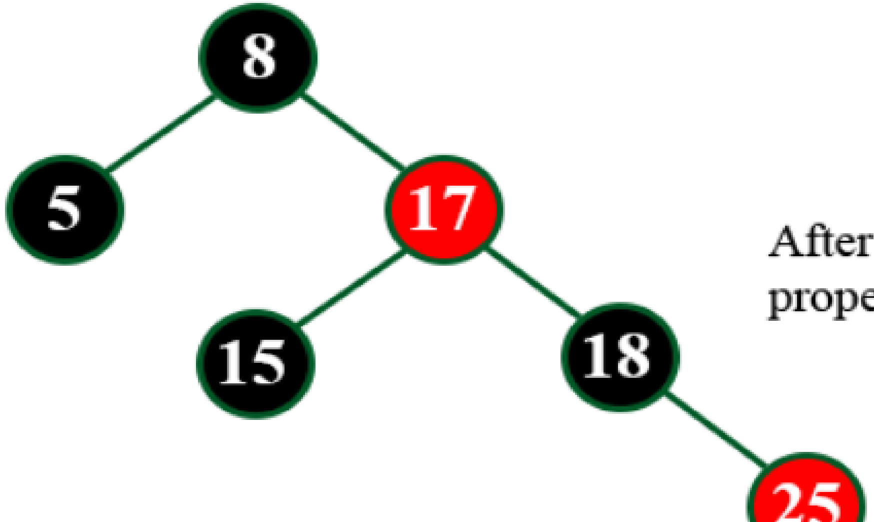
insert (25)

Tree is not Empty. So insert newNode with red color.



Here there are two consecutive Red nodes (18 & 25).
The newnode's parent sibling color is Red
and parent's parent is not root node.
So we use RECOLOR and Recheck.

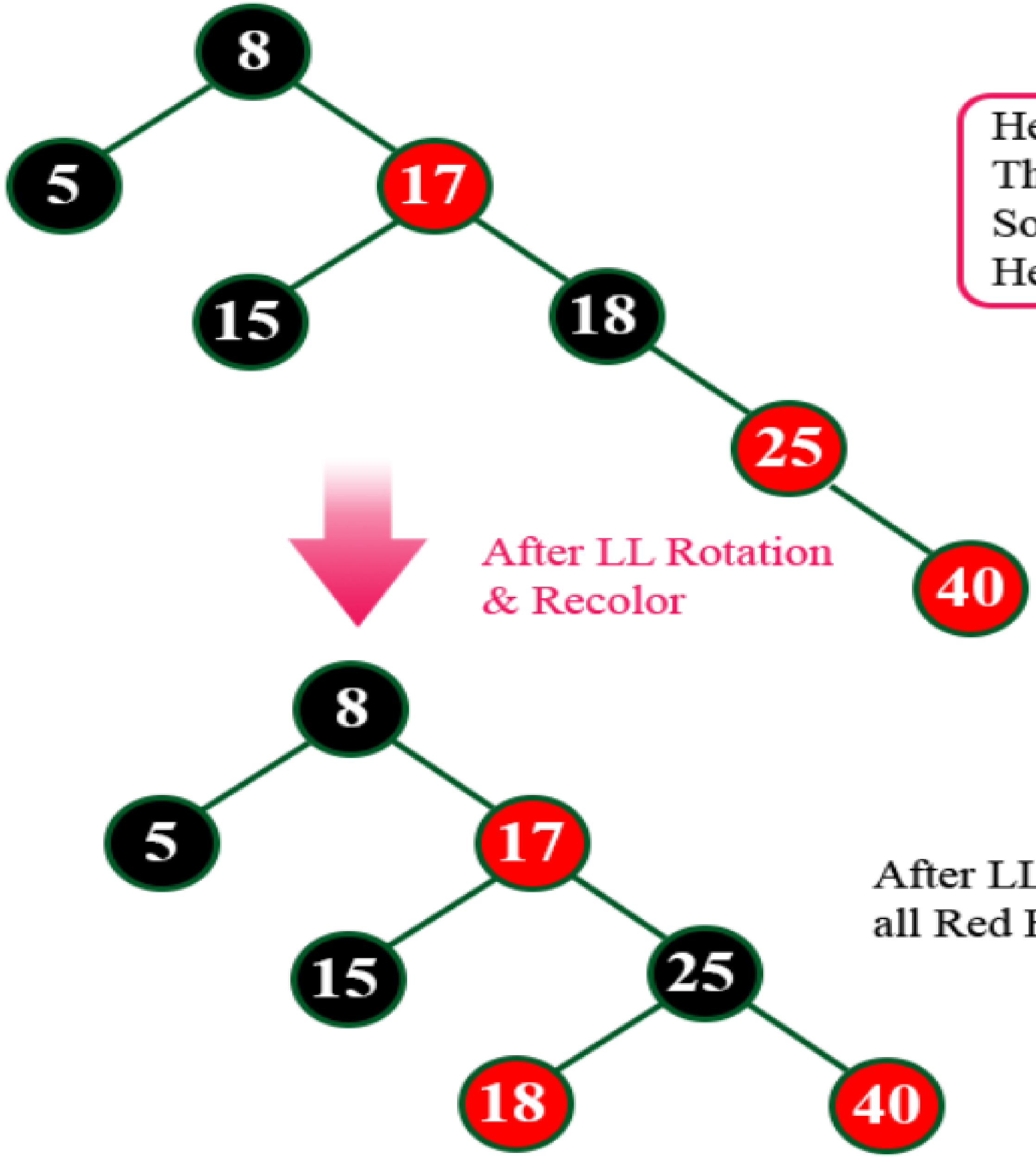
After Recolor



After Recolor operation, the tree is satisfying all Red Black Tree properties.

insert (40)

Tree is not Empty. So insert newNode with red color.

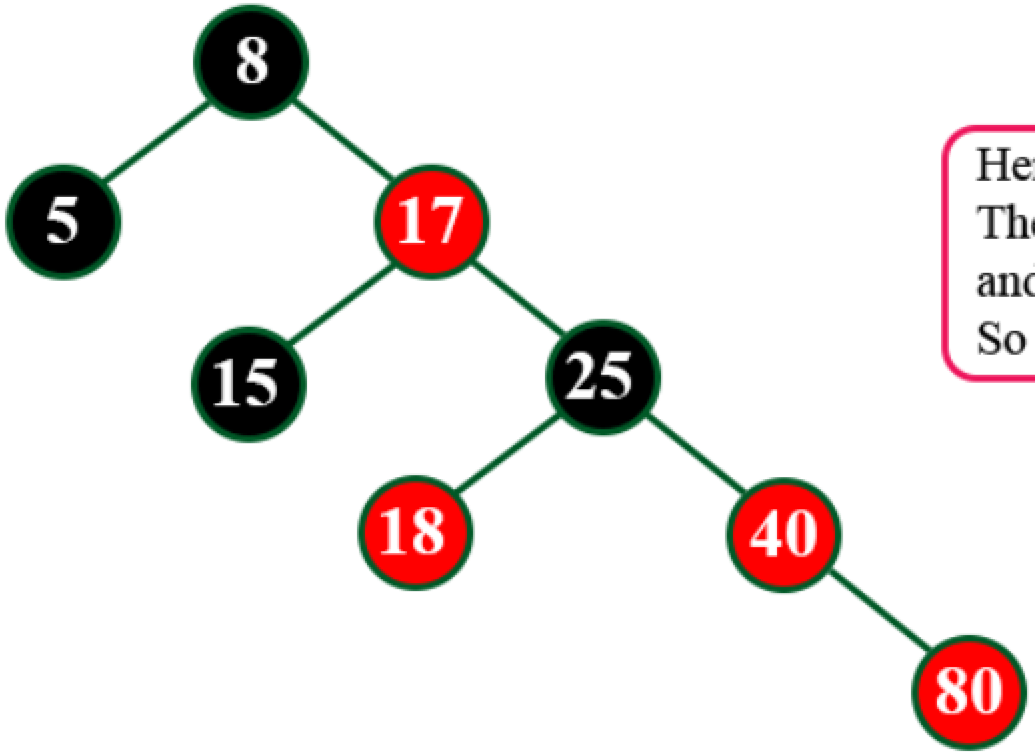


Here there are two consecutive Red nodes (25 & 40).
The newnode's parent sibling is NULL
So we need a Rotation & Recolor.
Here, we use LL Rotation and Recheck.

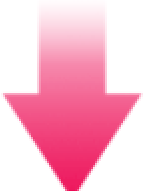
After LL Rotation & Recolor operation, the tree is satisfying all Red Black Tree properties.

insert (80)

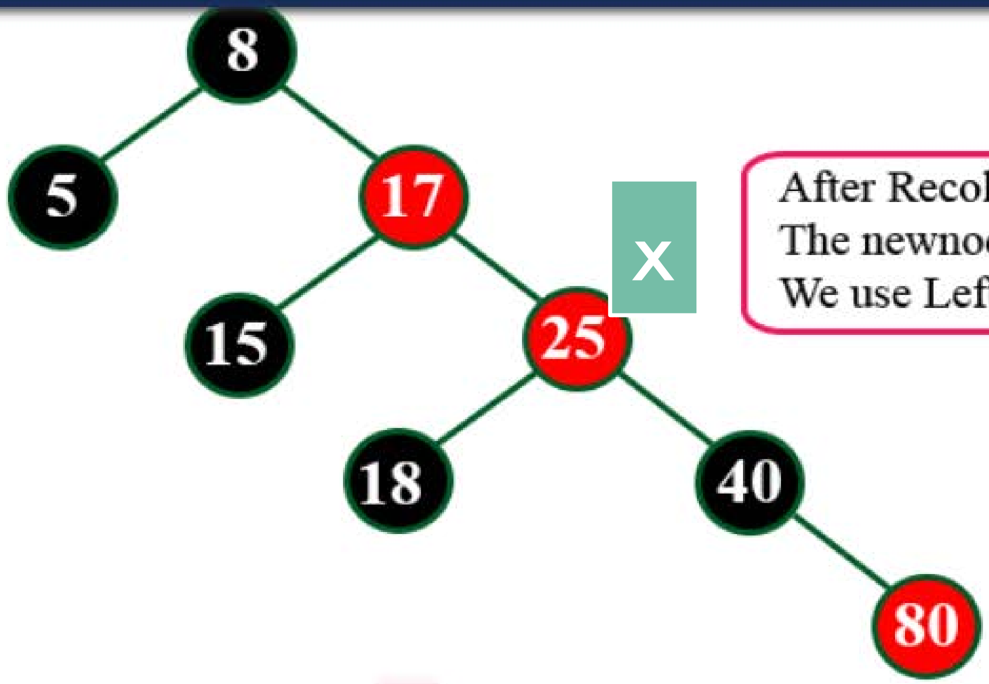
Tree is not Empty. So insert newNode with red color.



Here there are two consecutive Red nodes (40 & 80).
The newnode's parent sibling color is Red
and parent's parent is not root node.
So we use RECOLOR and Recheck.

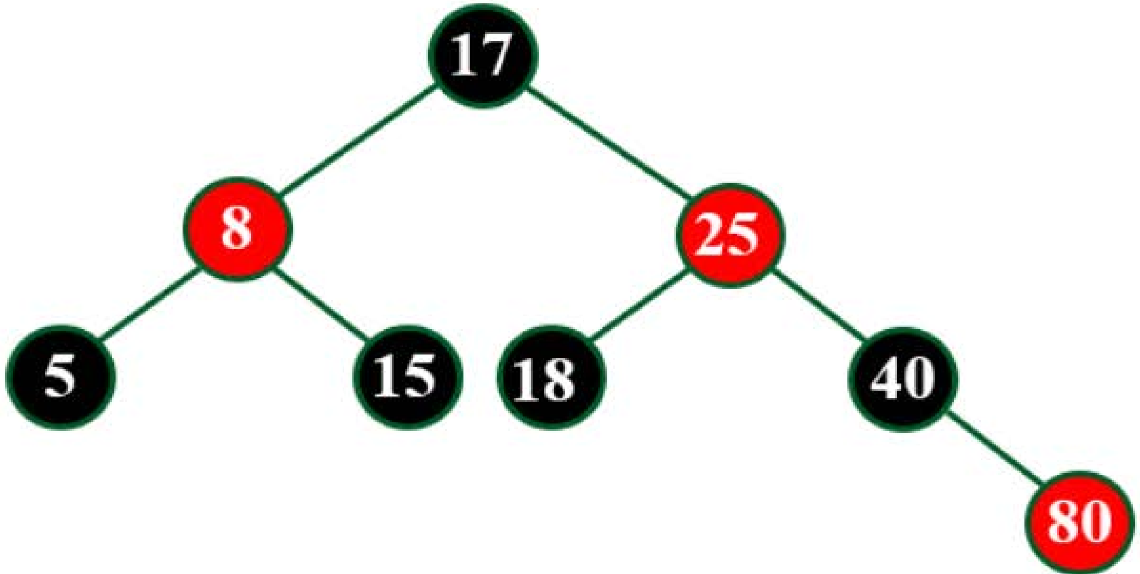


After Recolor



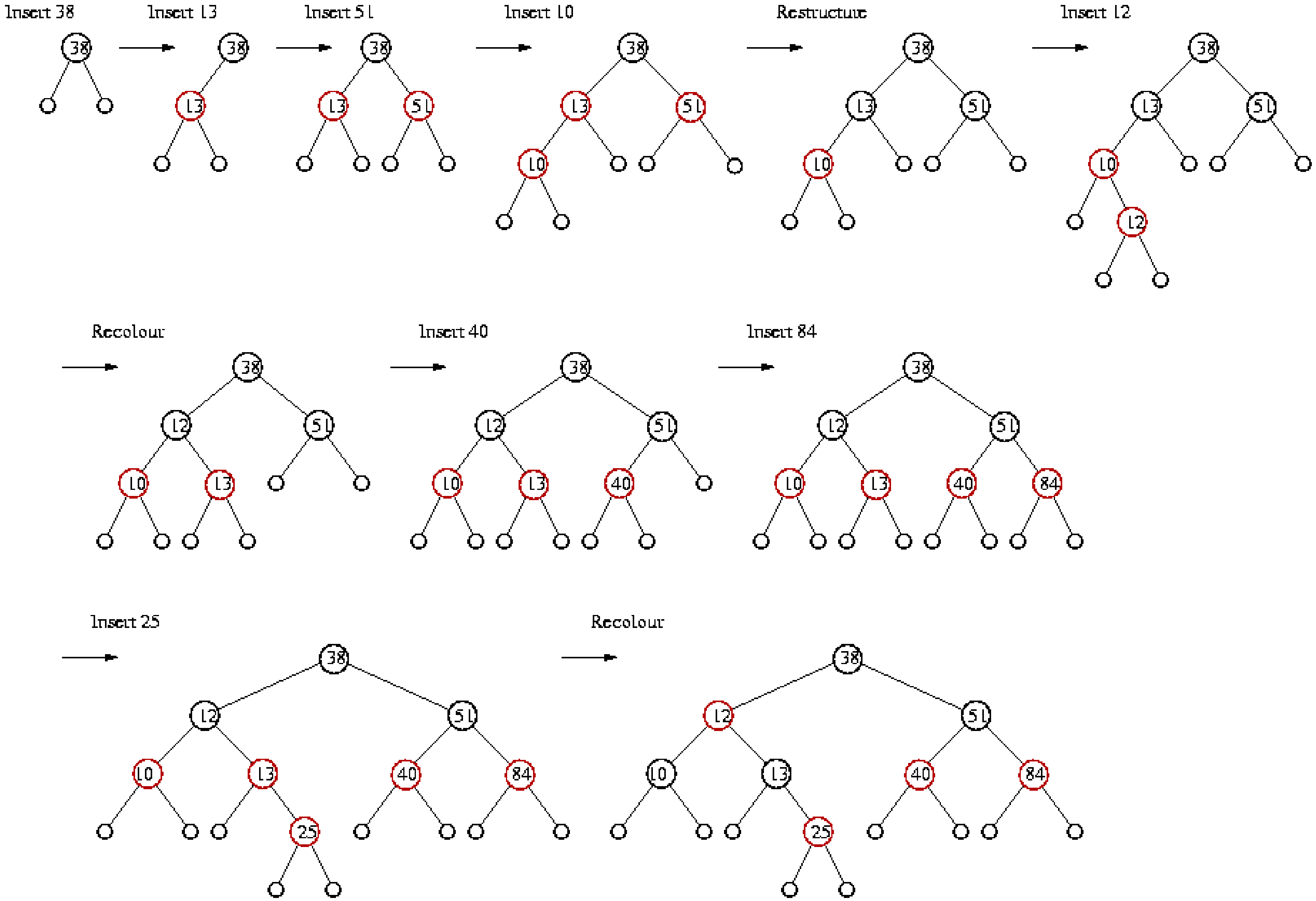
After Recolor again there are two consecutive Red nodes (17 & 25).
The newnode's parent sibling color is Black. So we need Rotation.
We use Left Rotation & Recolor.

After Left Rotation & Recolor



Red–Black Tree Example

Insertions: 38, 13, 51, 10, 12, 40, 84, 25



حذف

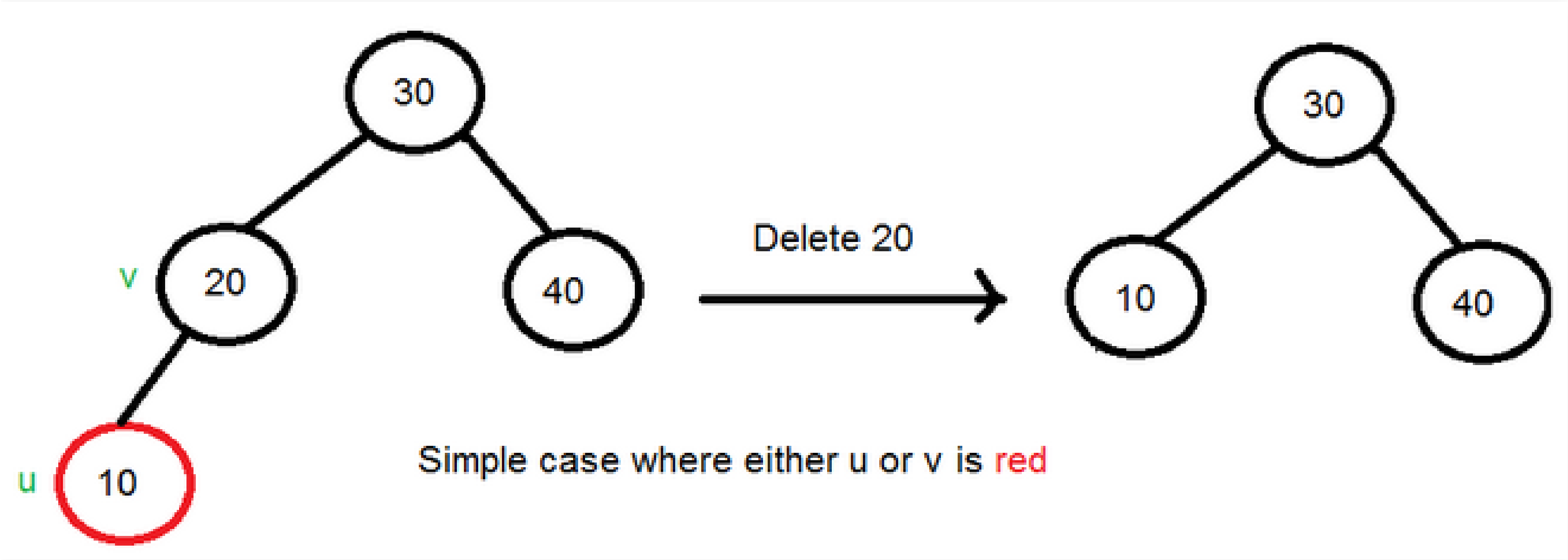
- در عملیات حذف هم ابتدا مانند عمل حذف در یک د.د.ج عادی عمل می‌کنیم (یادآوری: در حذف در درخت قرمز-سیاه همیشه گره مورد بحث یک فرزند داشت (و یا با گره ای با حداکثر یک فرزند جایگزین میشد))
- اگر راسی که حذف کردیم قرمز باشد، هیچ کدام از خواص درخت نقض نخواهند شد.
- ولی اگر راس حذف شده سیاه باشد خاصیت چهارم نقض خواهد شد.

حذف

- چند حالت بررسی میشود
- تمام حالت ها هر گره یک فرزند دارد
- **u** گره فرزند و **v** گره مورد نظر ما برای حذف است

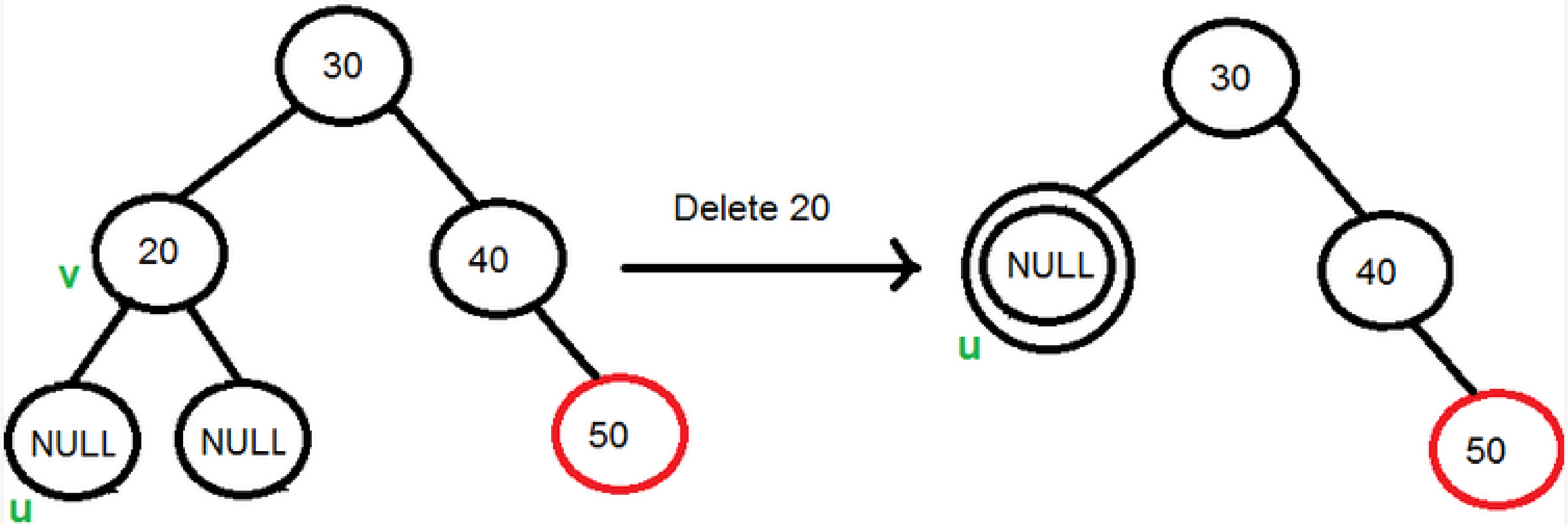
حذف ۱

- گره u و یا v قرمز باشد



حذف ۲

اگر **u** و **v** سیاه باشند



When 20 is deleted, it is replaced by a NULL, so the NULL becomes double black.

Note that deletion is not done yet, this double black must become single black

حذف ۲

در صورتی که **u** و **v** هر دو سیاه باشند، سه حالت داریم:

۱- اگر گره **sibling** گره **u**، سیاه باشد و حداقل یک فرزند قرمز داشته باشد (شامل چهار حالت)

۲- اگر گره **sibling** گره **u**، سیاه باشد و دو فرزند سیاه هم داشته باشد

۳-اگر گره **sibling** گره **u**، قرمز باشد

حذف ۲ – ۱

گره sibling گره u (گرهs)، سیاه باشد و حداقل یک فرزند قرمز (r) داشته باشد (شامل چهار حالت)

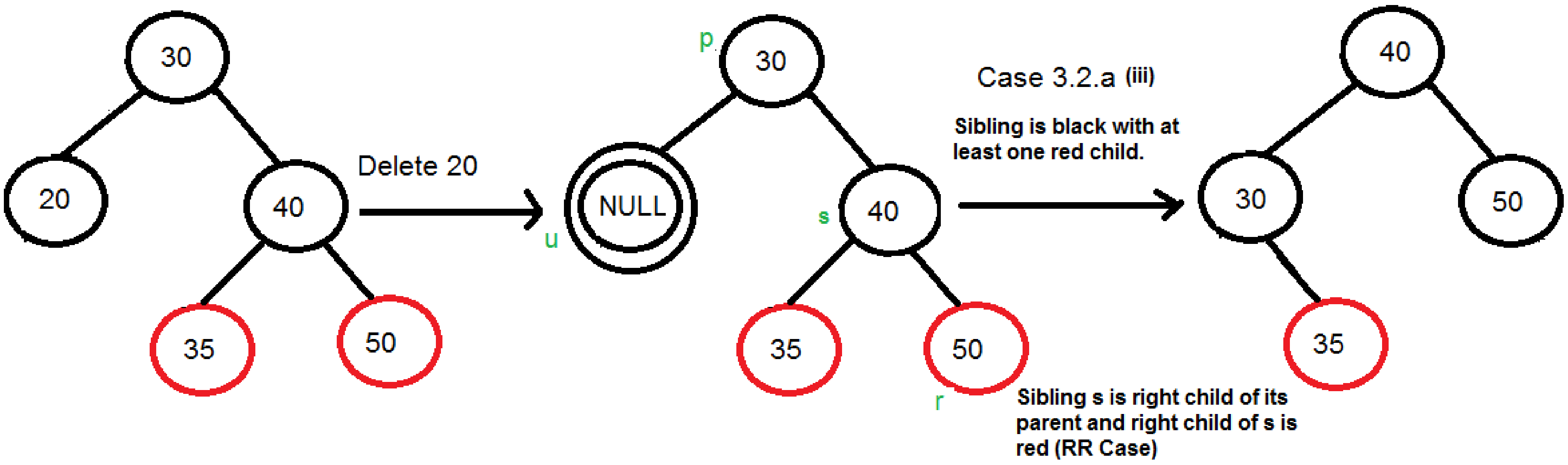
a) گره s و r هر دو فرزند چپ باشند (left left)

b) گره s فرزند چپ و گره r فرزند راست باشد (left right)

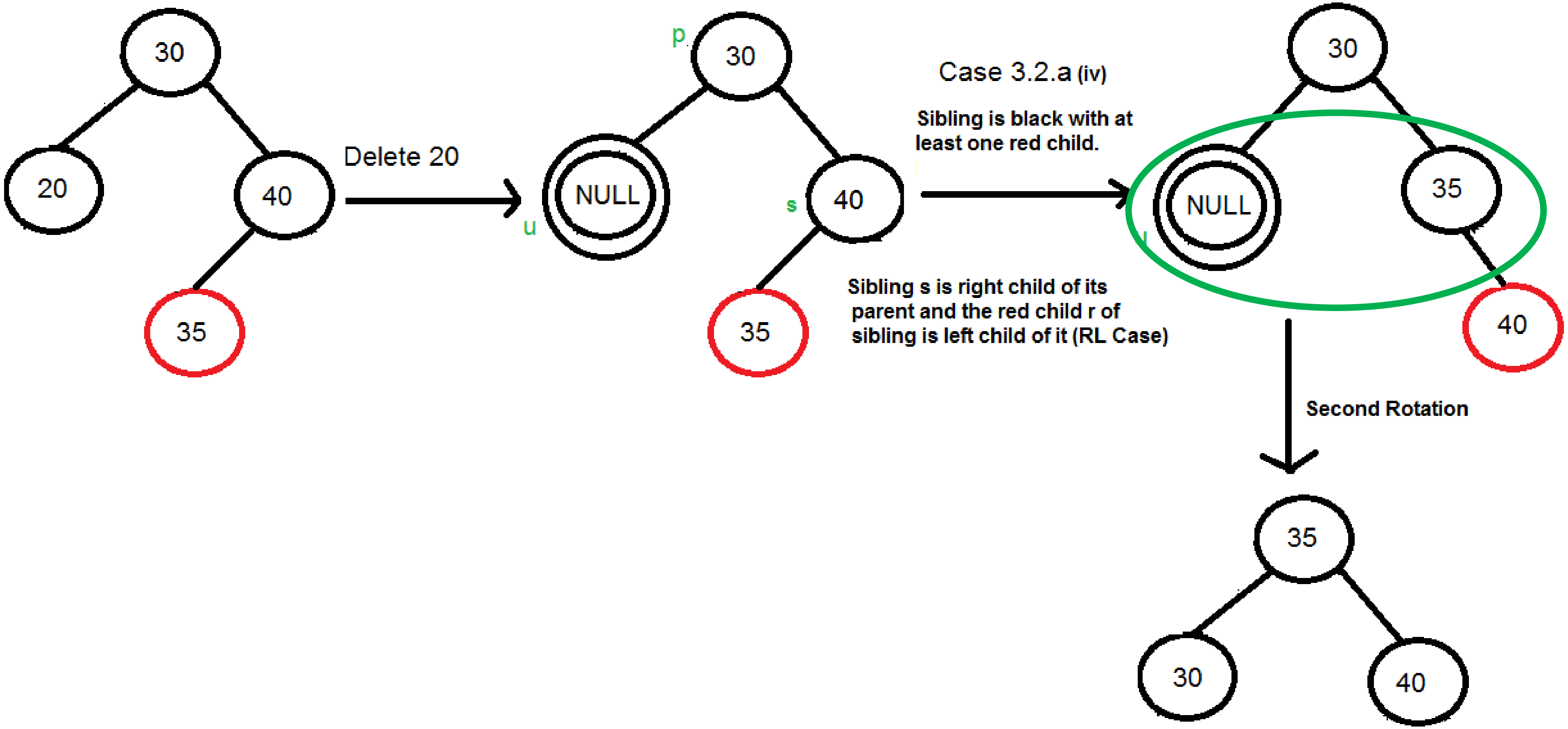
c) گره s و r هر دو فرزند راست باشند (right right)

d) گره s فرزند راست و گره r فرزند چپ باشد (right left)

مثال حذف ٢ – ١ (RIGHT-RIGHT)



مثال حذف ٢ –١– (RIGHT-LEFT)



حذف ۲-۱

```
       x                            y (x's color)
      / \                          /     \
  rdb     y        =====>         x       z
           \                     /
            z                   r
```

```
       x                            z (x's color)
      / \                          /     \
  rdb     y        =====>         x       y
         /                       /
        z                       r
```

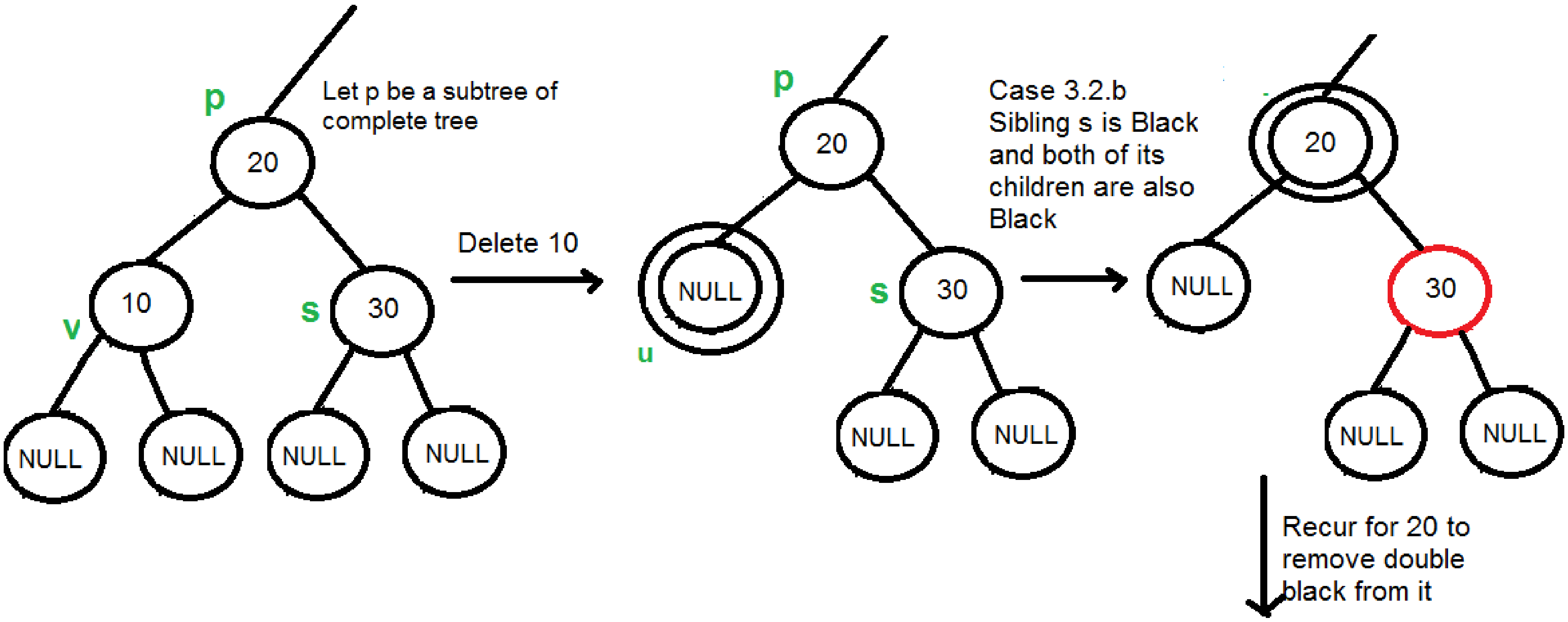
حذف ۲-۱

```
      x                              z (x's color)
     / \                             /    \
    y    rdb       =====>           y      x
     \                                      \
      z                                      r
```

```
      x                              y (x's color)
     / \                             /    \
    y    rdb       =====>           z      x
   /                                        \
  z                                          r
```

حذف ۲-۲

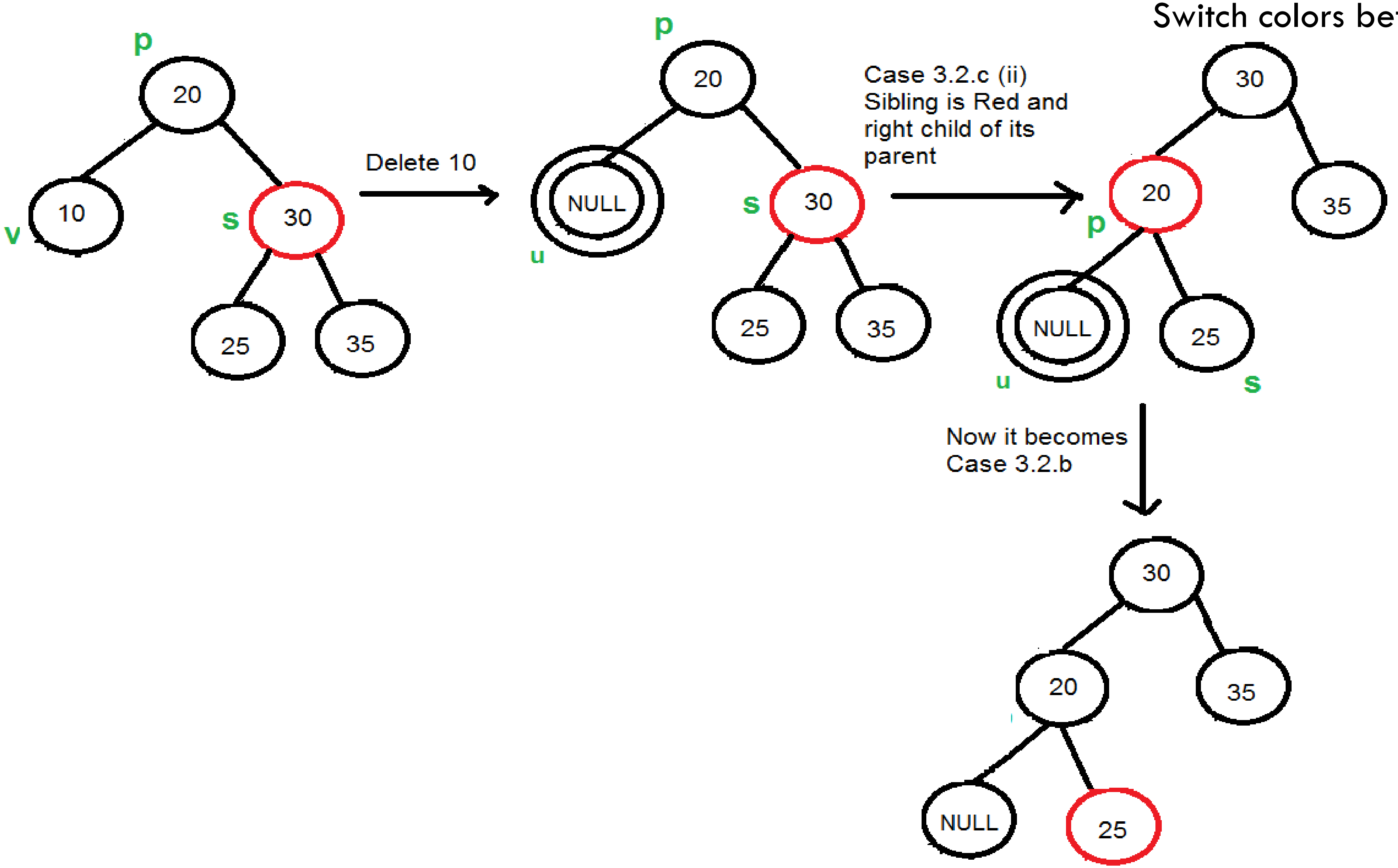
اگر گره SIBLING گره U، سیاه باشد و دو فرزند سیاه هم داشته باشد
(تغییر رنگ S)



x / \\ $\mathbf{r}_{db}$ $\underline{\mathbf{y}}$	=====>	$\underline{\mathbf{x}}$ / \\ $\underline{\mathbf{r}}$ y
$\underline{\mathbf{x}}$ / \\ $\mathbf{r}_{db}$ $\underline{\mathbf{y}}$	=====>	$\mathbf{x}_{db}$ / \\ $\underline{\mathbf{r}}$ y
x / \\ $\underline{\mathbf{y}}$ $\mathbf{r}_{db}$	=====>	$\underline{\mathbf{x}}$ / \\ y $\underline{\mathbf{r}}$
$\underline{\mathbf{x}}$ / \\ $\underline{\mathbf{y}}$ $\mathbf{r}_{db}$	=====>	$\mathbf{x}_{db}$ / \\ y $\underline{\mathbf{r}}$

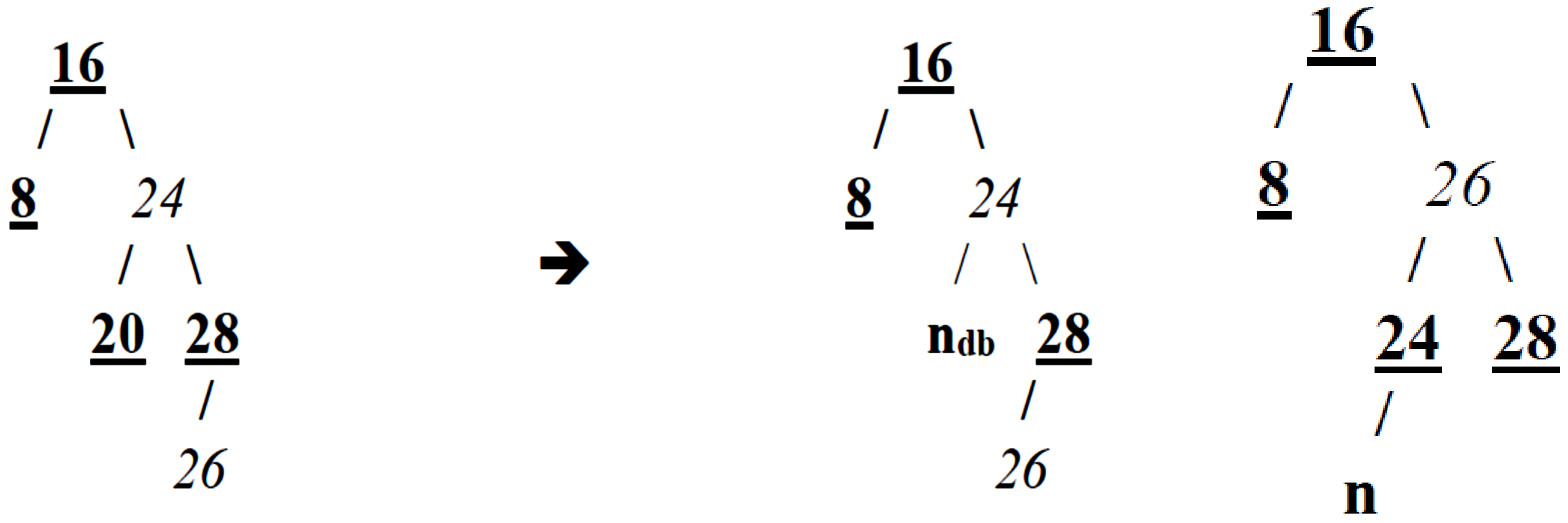
حذف ۲-۳

Rotate left around a’s father and
Switch colors between u’s father (p) and grandfather (s)



گره sibling گره
u قرمز باشد

مثال



مثال

```
      11
     /  \
    6    15
   / \   / \
  5  10 14  17
```

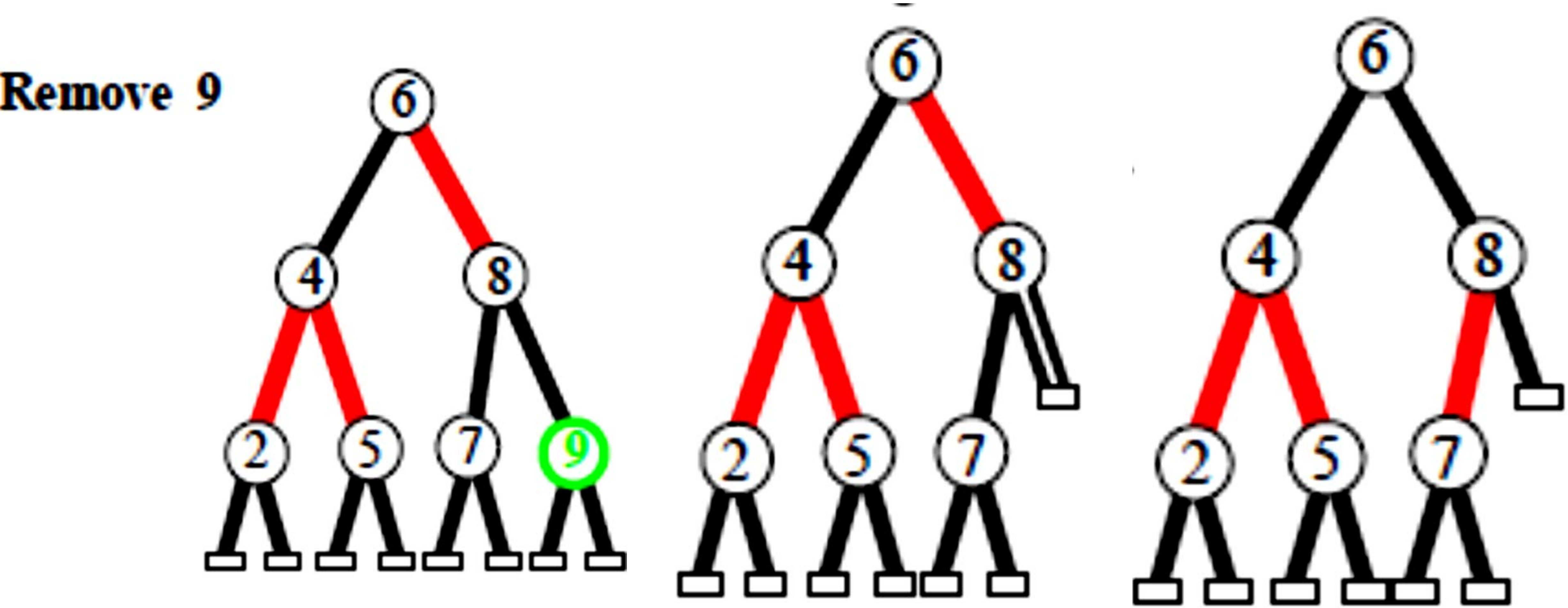
```
      11
     /  \
    6    15
   / \   / \
  5  10 ndb 17
```

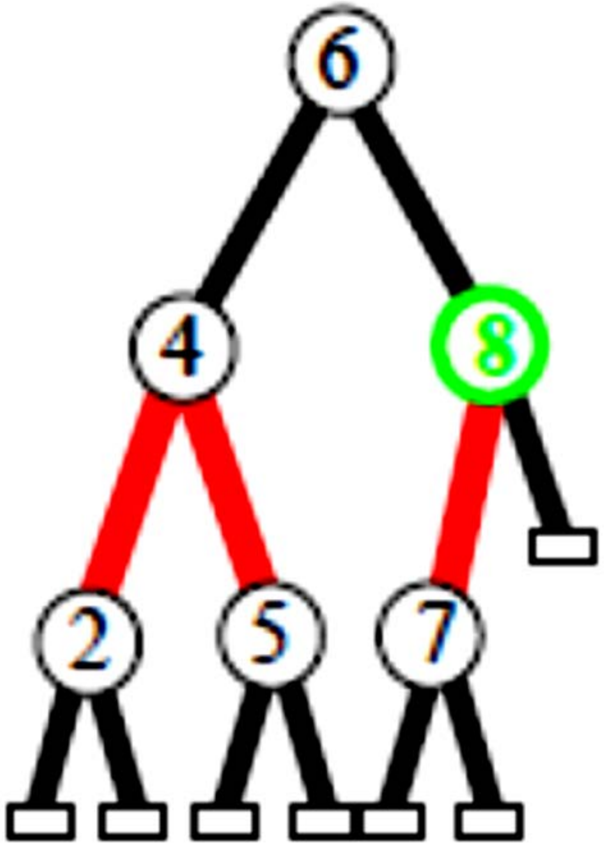
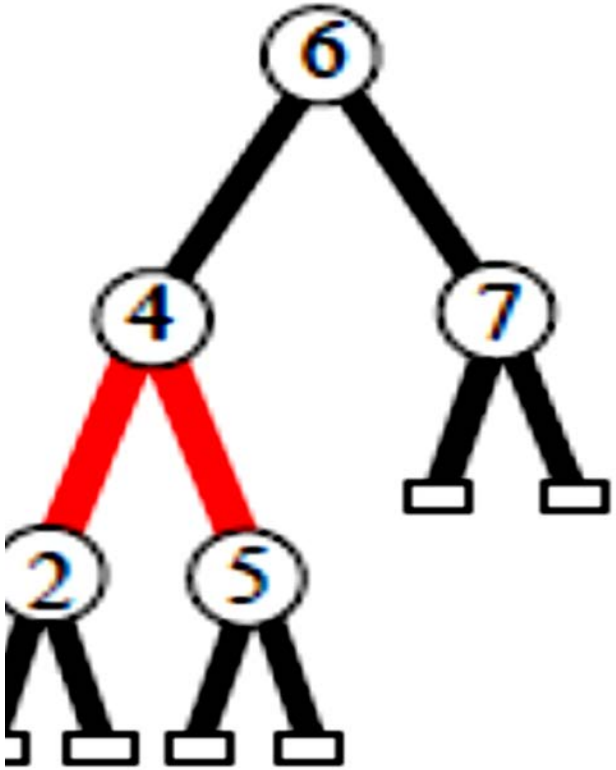
```
      11
     /  \
    6    15db
   / \   / \
  5  10 n   17
```

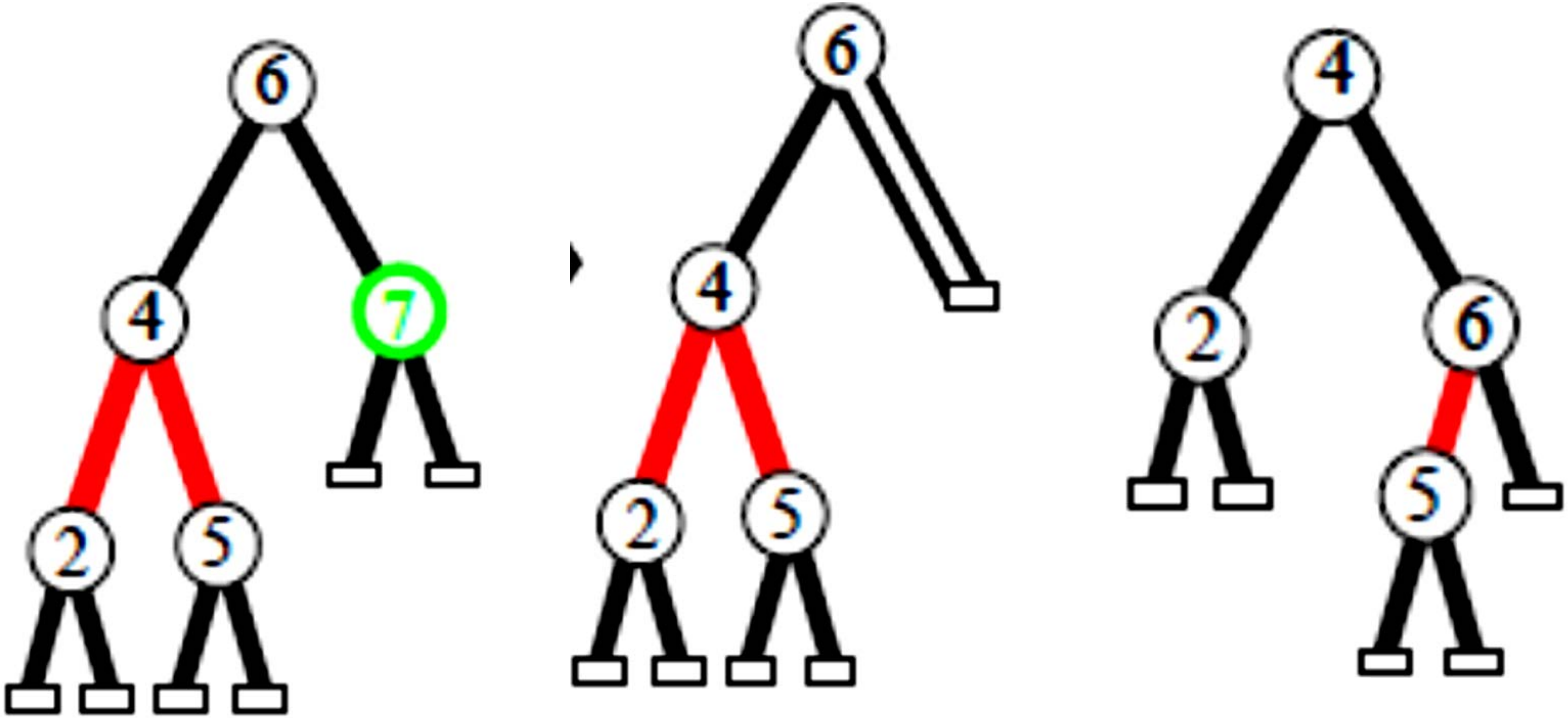
```
     11db
     /  \
    6    15
   / \   / \
  5  10 n   17
```

```
      11
     /  \
    6    15
   / \   / \
  5  10 n   17
```

مثال




6
4
8
2
5
7

6
4
7
2
5


6
4
7
2
5
6
4
2
5
4
2
6
5

https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

مرتبه تنظیم مجدد ارتفاع در RB

در صورت چرخش درخت مشکل حل میشود و به سطح بالاتر ادامه پیدا نمیکند ((O(1)

در صورت تغیر رنگ گره ها ممکن است الگوریتم مجددا برای گره پدر اجرا شود((O(logn)

بنابراین از (O(logn است

هزینه درج و حذف و جستجو در RB

Operation	Time
Search	O(log N)
Insert	O(log N)
Delete	O(log N)