CS200 Functional Data Structures
Assignment 2

Beep Beep: Anas-am00634 and Affan-sa00310

March 4, 2016

Problem 1. Red Black Tree

Please see the ass2.hs file in the same folder.

Problem 2. Insertion

1. 2-4 Tree

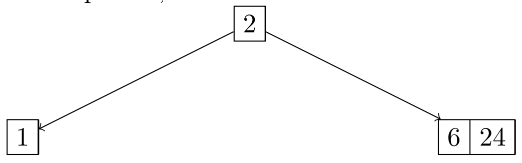
   (a) We begin with an empty tree. $\boxed{\text{NULL}}$

   (b) Insert(1) $\boxed{1}$

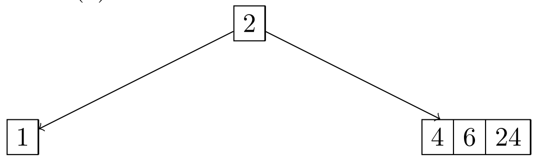
   (c) Insert(24) $\boxed{1 \mid 24}$

   (d) Insert(2) *(after rearrangement)* $\boxed{1 \mid 2 \mid 24}$

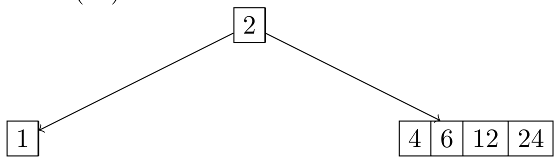
   (e) Insert(6) $\boxed{1 \mid 2 \mid 6 \mid 24}$
   Break parent;



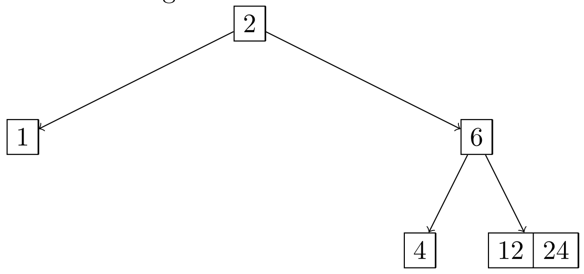
   (f) Insert(4)



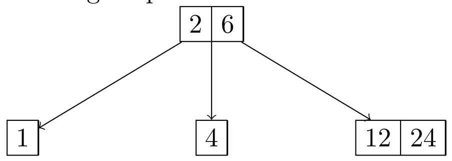
(g) Insert(12)



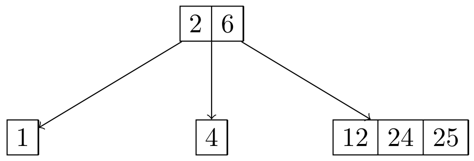
After breaking node 6



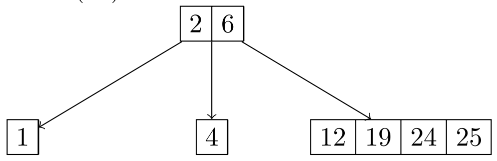
Pushing 6 up.



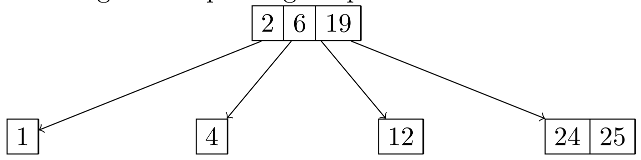
(h) Insert(25) *trivial*



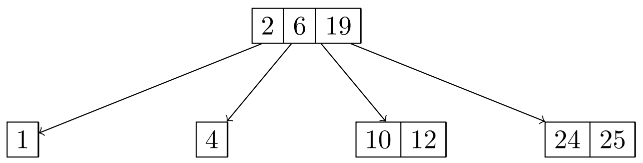
(i) Insert(19)



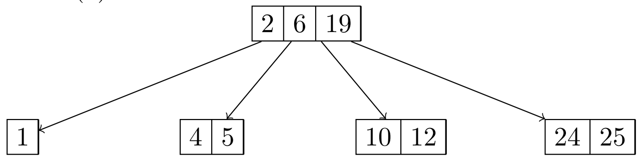
Breaking 19 and pushing it up



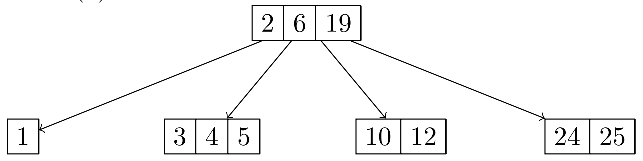
(j) Insert (10) *trivial*



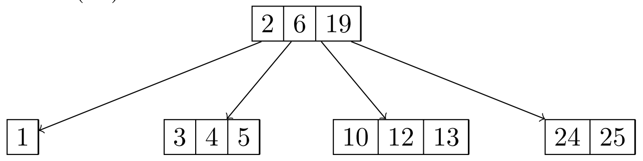
(k) Insert(5) *trivial*



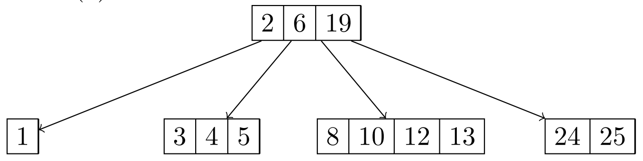
(l) Insert(3) *trivial*



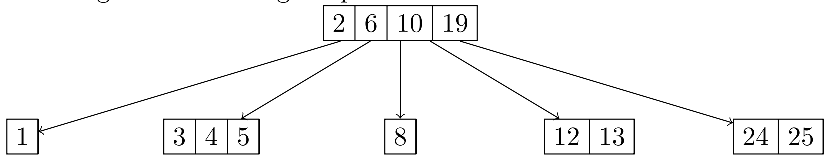
(m) Insert(13) *trivial*



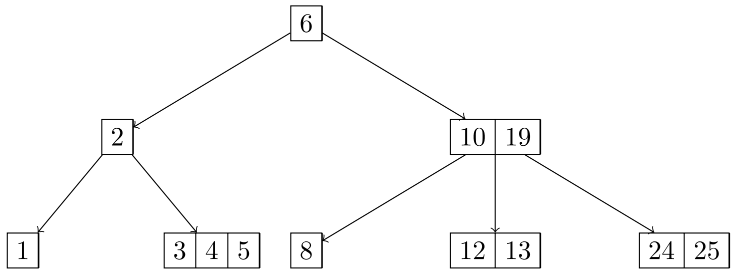
(n) Insert(8)



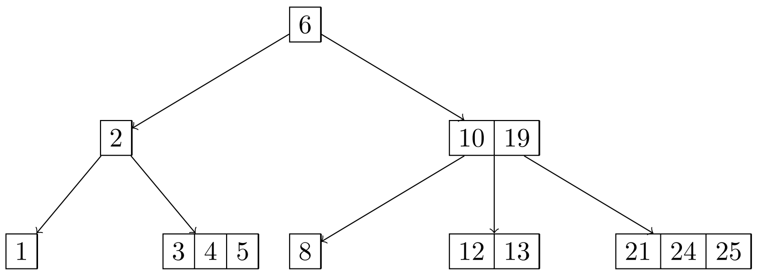
Breaking 10 and moving it up.



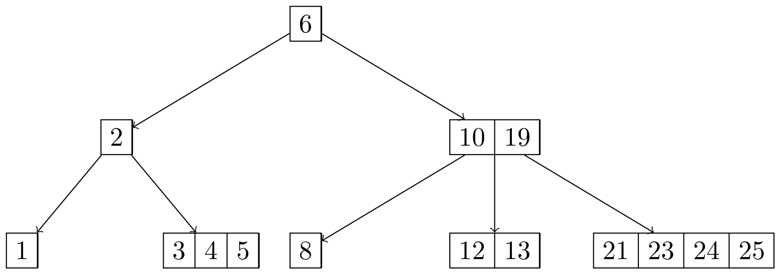
Breaking 6 and moving it up.



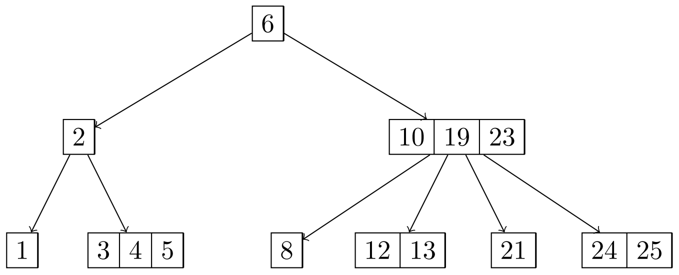
(o) Insert(21) *trivial*



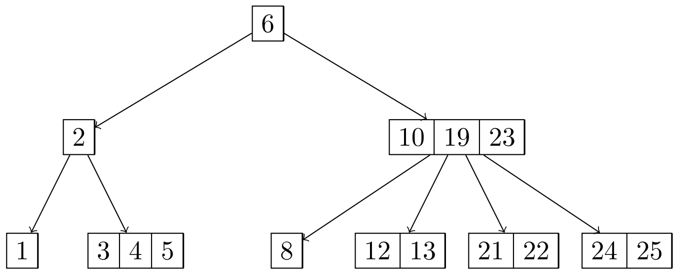
(p) Insert(23)



Breaking 23 and moving it up.

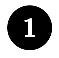


(q) Insert(22) *trivial*

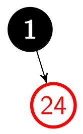


2. Red-Black Tree

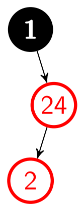
(a) Insert (1)
Node is added as red but if node is root, change color to black and done.



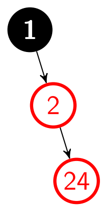
(b) Insert(24)
24 is added as red, and its parent is black therefore done.



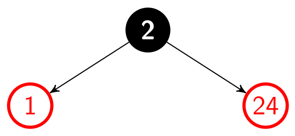
(c) Insert(2)
2 is added as red, and there is a conflict.



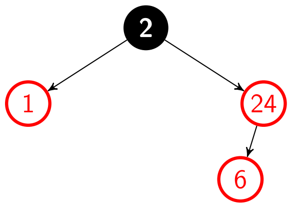
Since no uncle, we assume uncle is black. Case is right-left, therefore we RotateRight(24)



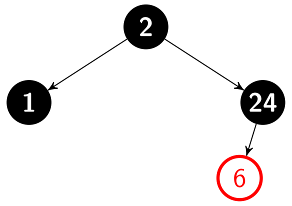
Fix (24) and case is right right. Therefore color 2 black, 1 red and left-rotate(1)



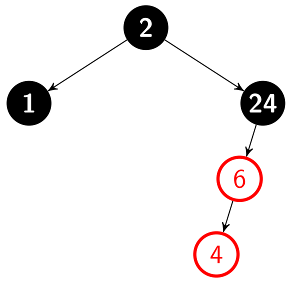
(d) Insert(6)
6 is inserted to the left of 24, with red as default. We get a conflict.



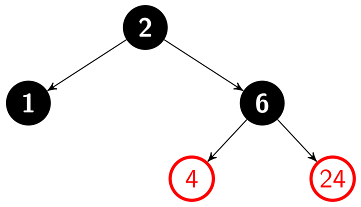
Since uncle is red, we just color parent and uncle black, and grandparent red. Since grandparent is root, fix(grandparent) would make grandparent black.



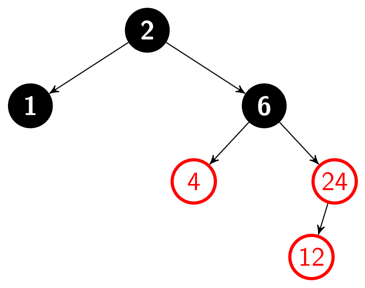
(e) Insert(4)
Since inserted 4 is red, we get a conflict.



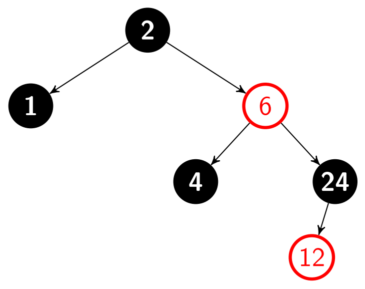
Since uncle is black and the inserted case is left left, we change parent to black, grandparent to red, and rotateRight(g)



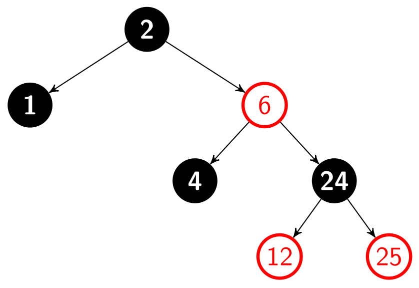
(f) Insert(12) Inserted 12 is red and to the left of 24.



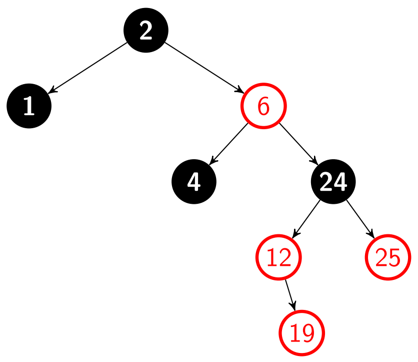
Inserted case is left-right and uncle is red. Therefore we color p and u black, g red, and fix(g). g has a black parent, therefore we stop there.



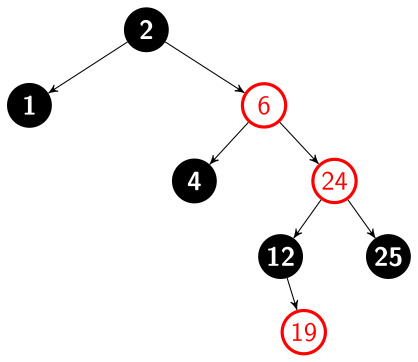
(g) Insert(25)
25 is red and is inserted to the right of 24. Trivial.



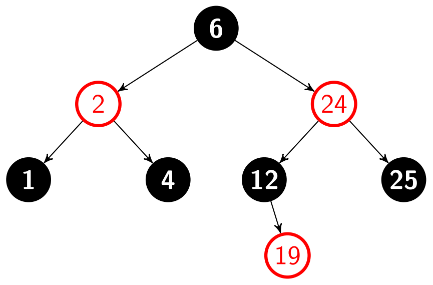
(h) Insert(19)
Inserted to the right of 12 which results in a conflic.



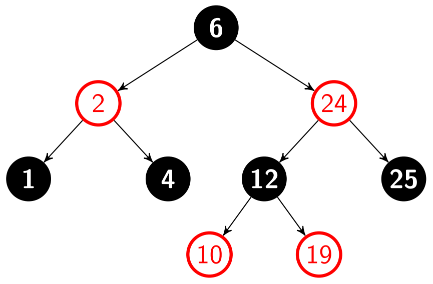
Since the uncle is red, we color p and u black, g red and fix(g).



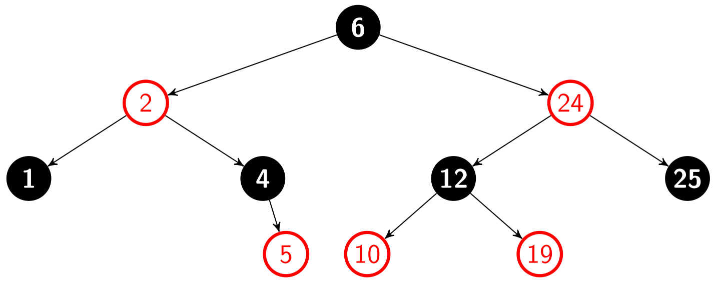
24 has black uncle and is right right case. Therefore we color p (6) black, g (2) red and rotateLeft(2)



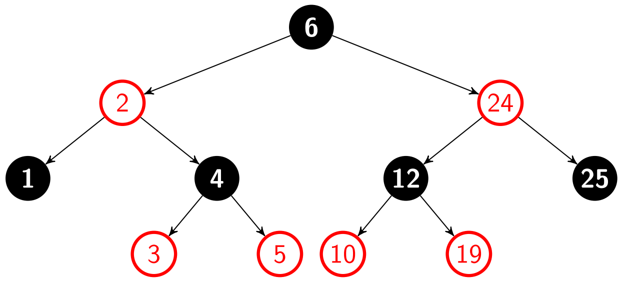
(i) Insert(10)
10 is inserted to the left of 12. Since the parent is black, we are done. This case is trivial.



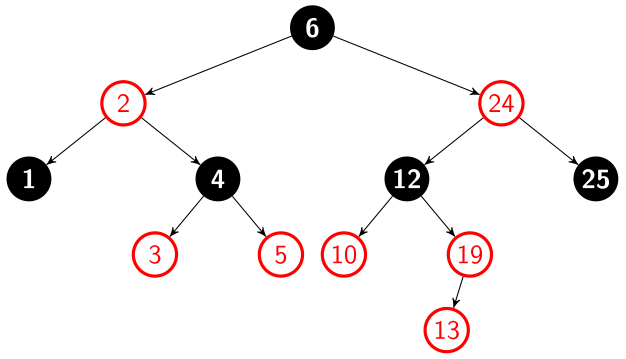
(j) Insert(5)
5 is inserted to the right of 4. Since the parent is black, we are done. This case is trivial.



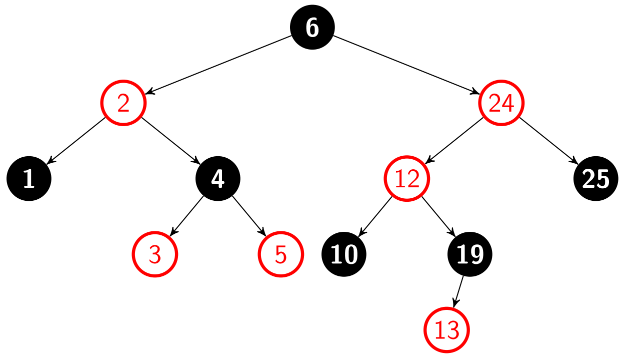
(k) Insert(3)
3 is inserted to the left of 4. Since the parent is black, we are done. This case is trivial.



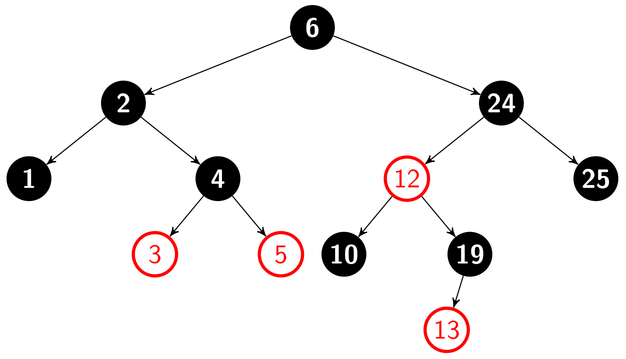
(l) Insert(13)
13 would be inserted to the left of 19. There will be a conflict.



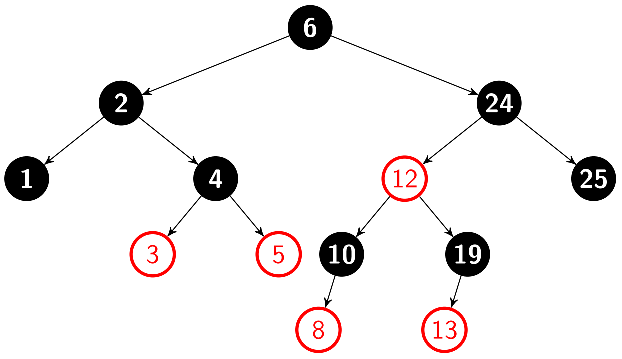
13's uncle is red, therefore, we color 19 and 10 black, and 12 red and call fix(12)



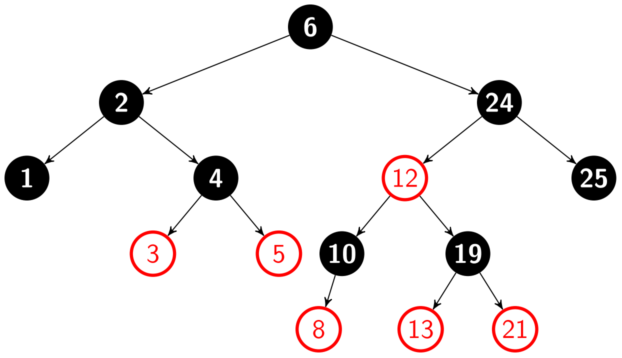
12's uncle is red and therefore, we color 24 and 2 black, and 6 red, and call fix(6). Since 6 is root, it is changed to black and we are done. (We have not shown one step here.)



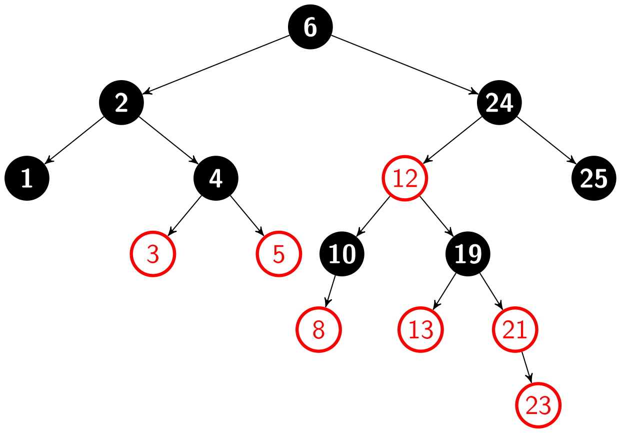
(m) Insert(8)
8 will go to the left of 10. Since the parent is black, this case is trivial.



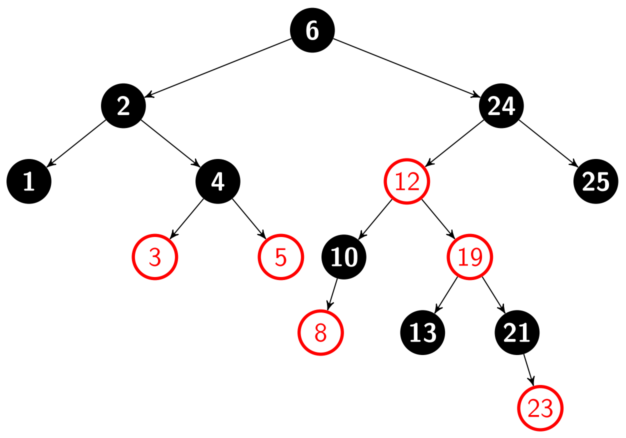
(n) Insert(21)
21 will be inserted to the right of 19. Since its parent is black, this case is trivial.



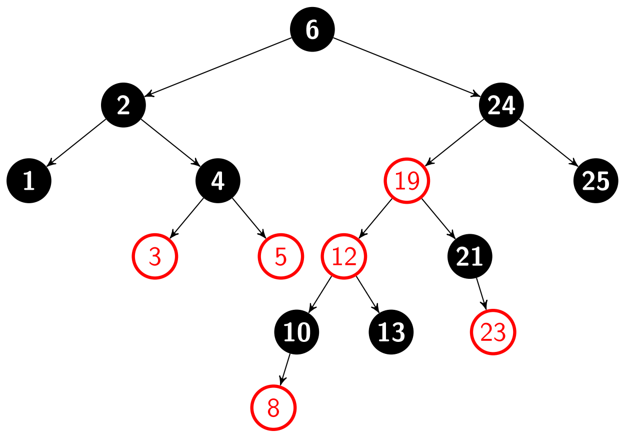
(o) Insert(23)
23 will be inserted to the right of 21. There will be a conflict.



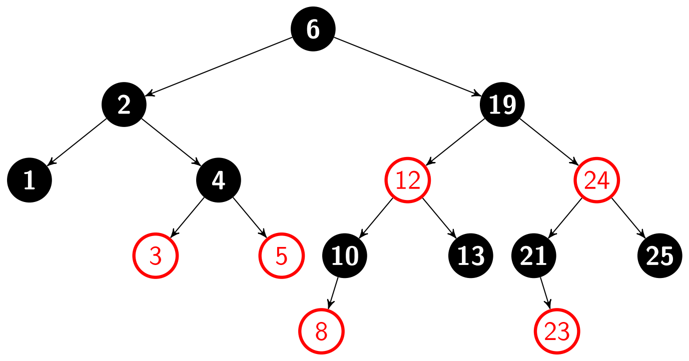
23 has red uncle, therefore we change 21 to black, 13 to black, 19 to red and fix(12).



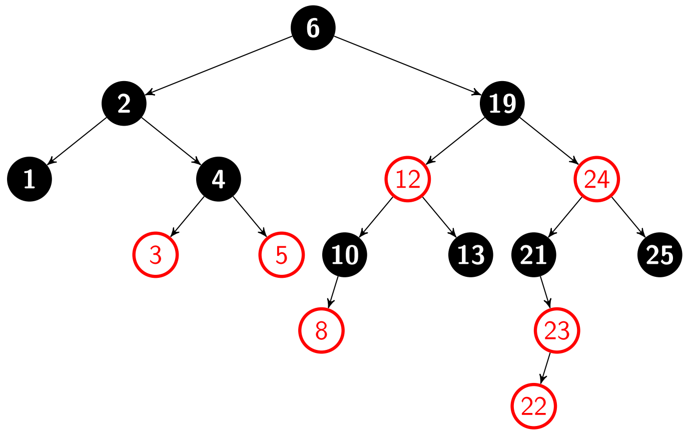
19 has black uncle, and its case is left-right. Therefore we rotateLeft(12) (12 is its parent)



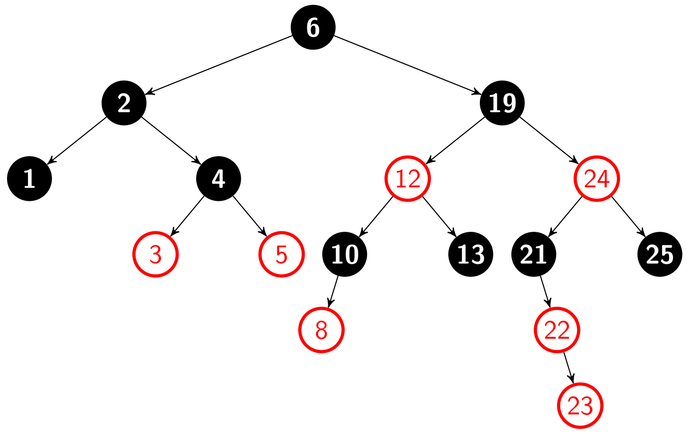
12 has black uncle and is left left case. Therefore we color 19 black, 24 red and rotateRight(24)



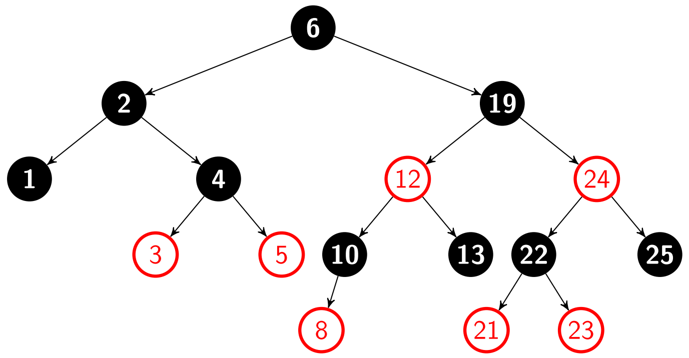
(p) Insert(22)
22 would be inserted to the left of 23. There will be a conflict.



Since uncle is (non-existent) black, we see that case is right-left. We rotateRight(23) (23 is parent) and fix(23).



23 case is right right, and therefore, we change color of 22 to black, 21 to red, and rotateLeft(21) (21 is grandparent)

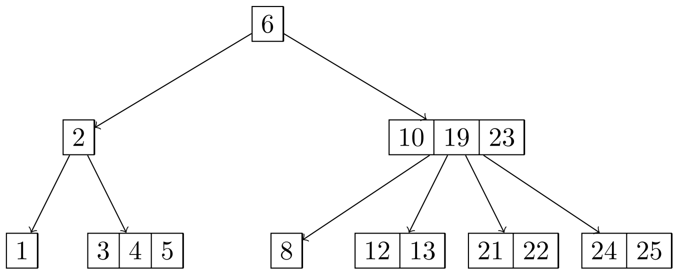


At this point we see that all 4 properties of the rb tree are balanced.

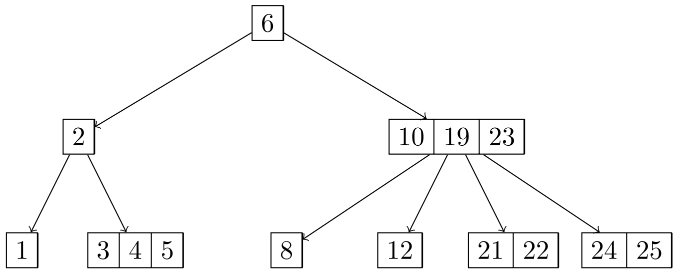
Each node has black height 4, root node is black, red has black children, and each node is either red or black.

Problem 3. Deletion

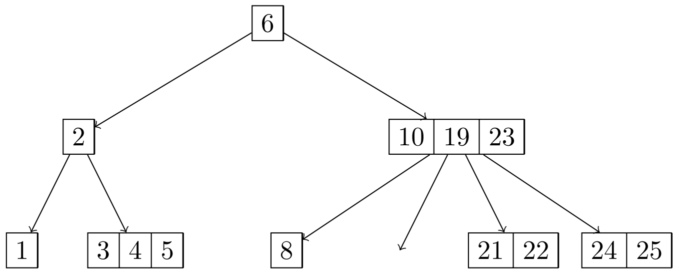
1. Original Tree (for reference purposes)



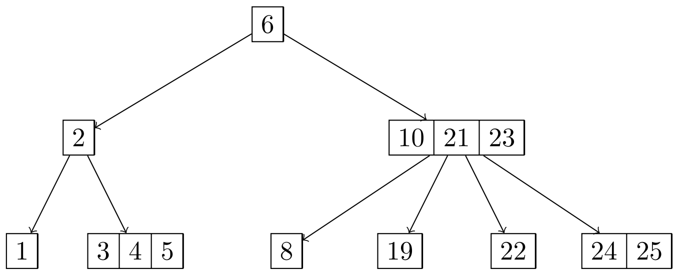
2. Remove(13) *trivial*



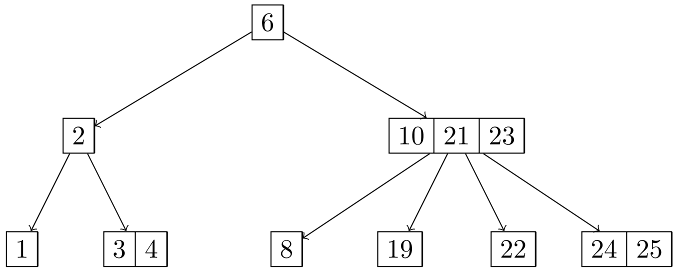
3. Remove(12)



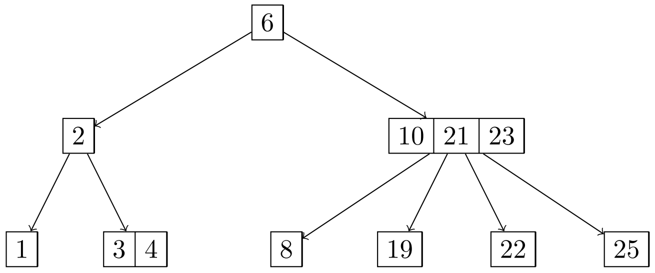
Pushing 19 down and 21 up,



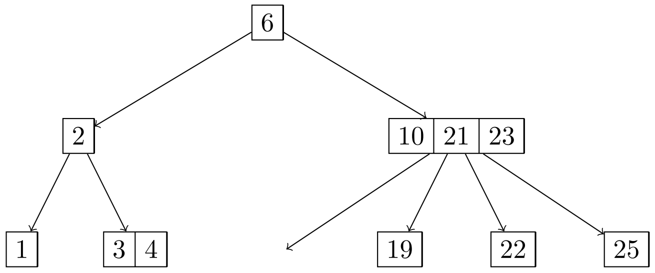
4. Remove(5) *trivial*



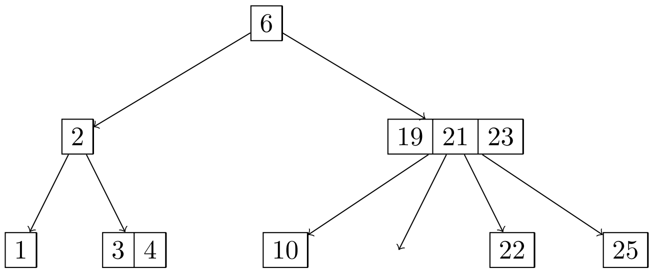
5. Remove(24) *trivial*



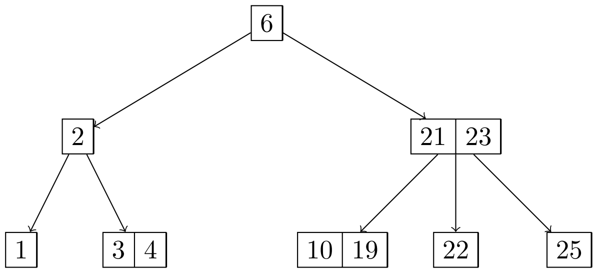
6. Remove(8)



Pushing 10 down and bringing 19 up,



Pulling 19 down will give us our final tree while 2-3 property is not violated.


6
2
21 23
1
3 4
10 19
22
25