Red-Black Trees

- A red-black tree is a height balanced binary search trees.
- Maintains height balance property of the tree differently from AVL.
- No repeated computation of balance factor.
- As there is no need to maintain a balance factor at every node.
- The nodes are placed logically into two different bins, namely, (i) red, and (ii) black.

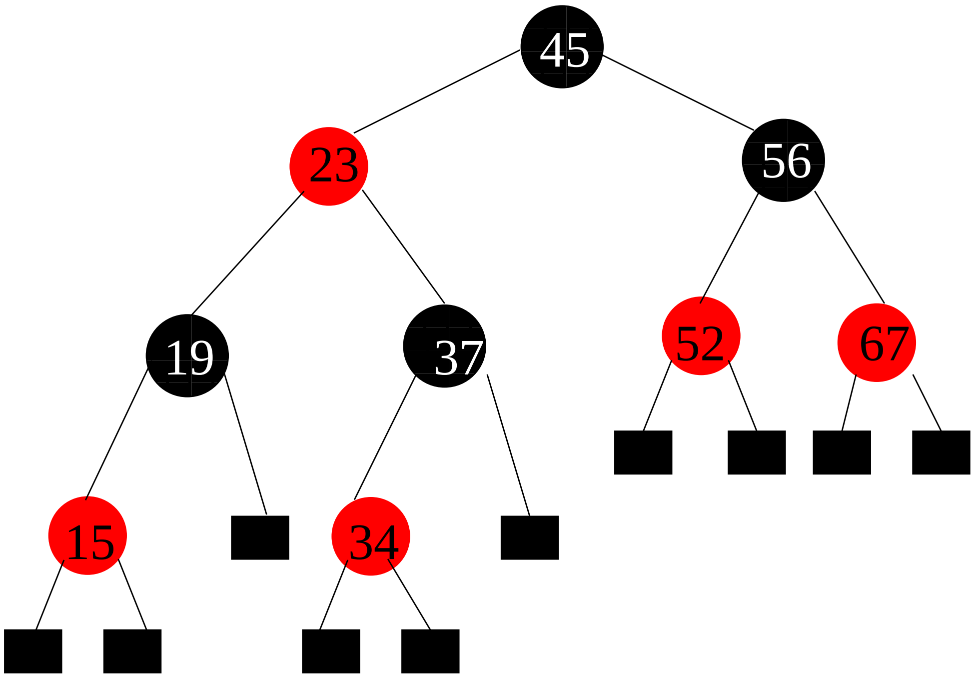
Red black tree

- A bit is maintained at every node to remember its current color.
- Basic transformation for preserving height balance property is to apply rotation bottom up as was done for restoring height balance property as in case of AVL trees.
- But the rotation is applied on the basis of colors nodes on the tree path up the node to root.
- Since, rotation operation is done more sparingly than in an AVL tree, red-black trees may result in better performance in some cases.
- In fact, for an insertion into a red-black tree only at most 2 rotations may be performed to update the the tree.

Definition of red-black tree

- An extended binary search tree having the following properties:
  1. every node is colored by either red or black color,
  2. every leaf node (an external node) is colored black,
  3. if a node is red its both children must be colored black,
  4. every simple path from any node to a descendant leaf contains equal number of black nodes.
- Root of a red-black tree is always colored black.
- By property three at least half the nodes in a red-black tree are black.
- A red black tree of with n internal nodes has height at most $2\log(n+1)$.

Example of a red-black tree



Insertion

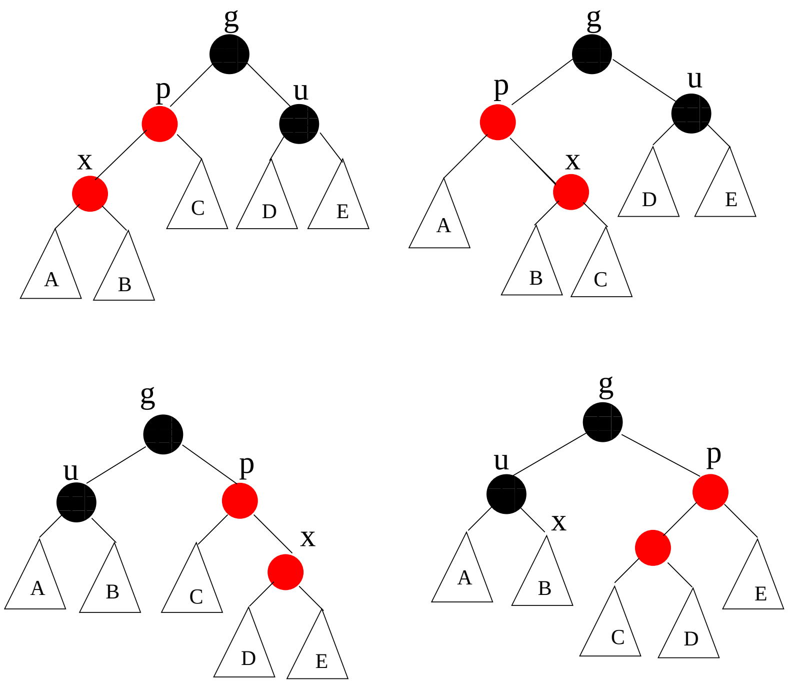
- A newly inserted node is always colored red.
- Insertion proceeds as binary search tree insertion.
- A newly inserted node thus replaces an external node by one internal node (containing the new element) and two new external nodes.
- If the newly inserted node is child of a black node then no need to perform recoloring operation.
- If the newly inserted node is the root red-black tree, then color the node as black.

Recoloring requirement

- If the newly inserted node has the red parent which can be described as a *double red* configuration, property 3 is violated. This requires a recoloring.
- Since root is always black the double red configuration can occur only when parent of inserted node is also an internal node.
- Furthermore, grand parent of the inserted node must be black.
- Which means double red configuration occurs only internal to the tree.
- But recoloring must be done in such a way that property 4 shall not be disturbed.
- Looking at various possibilities of double red configuration two distinct cases are identifiable.

Recoloring requirement

- The four possibe double red configurations may occur with black colored uncle as follows.
  1. the color of grand parent g is black, and
  2. the color of parent p is red, and
  3. the color of uncle u is black.

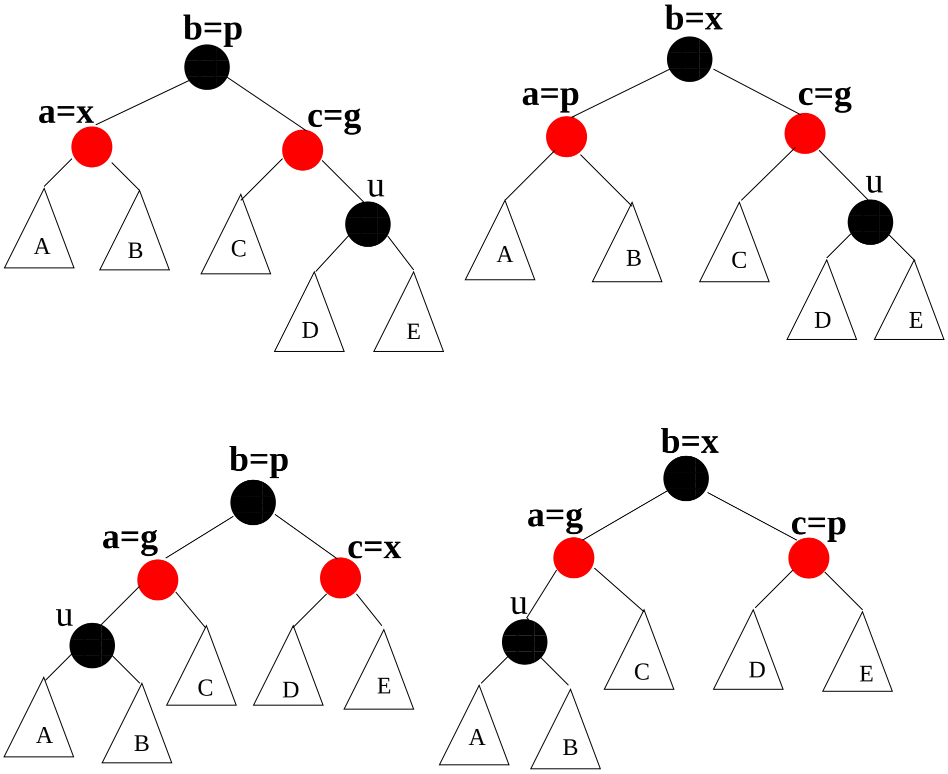


Handling case 1 recoloring

- Restructuring the tree at the tri-node structure x, p, and g as follows.

  1. Relabel x, p, g according to their respect in-order ranks. Let the new labels be a, b, and c.
  2. Replace previous g with the node that is labeled b, set a and c respectively as left and right children of b.
  3. Note that g will be relabled as either c or a. It can only be child of the root when trinode structure restructured.
  4. Exchange (black) color of g with the node (always red colored) that becomes root of the subtree

Case 1 Recoloring

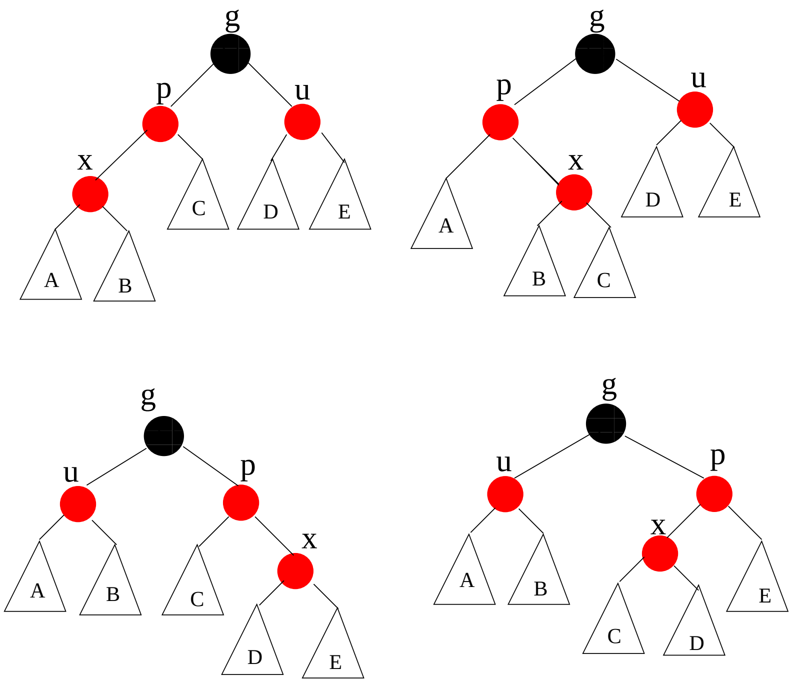


Insertion: Case 1 Analysis

- There is no change in the of subtrees of x, u and right subtree of p.
- Note that all properties of red-black tree were valid with subtrees before recoloring. As there is no change in those subtrees after recoloring the properties are preserved.
- So the black height of those trees are unchanged.
- Moreover, color of g transferred to p or x depending on whether the node replace g as root of the restructured subtree.
- Therefore, the black height of the restructured subtree does not change but the double red configuration is eliminated.

Insertion: case 2

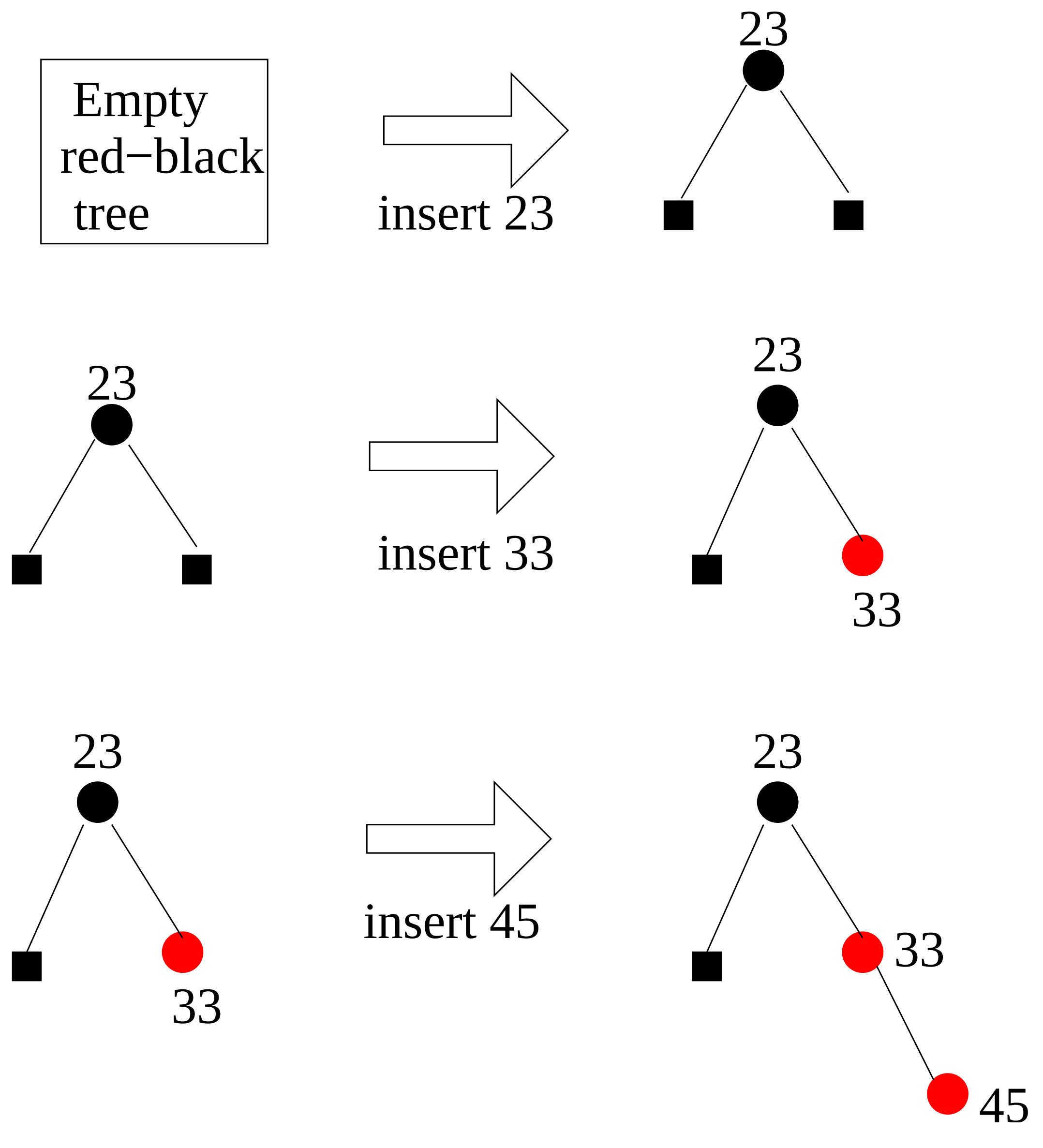
1. The color of grand parent g is black.
2. The color of parent node p is red,
3. The color of uncle node u is red.



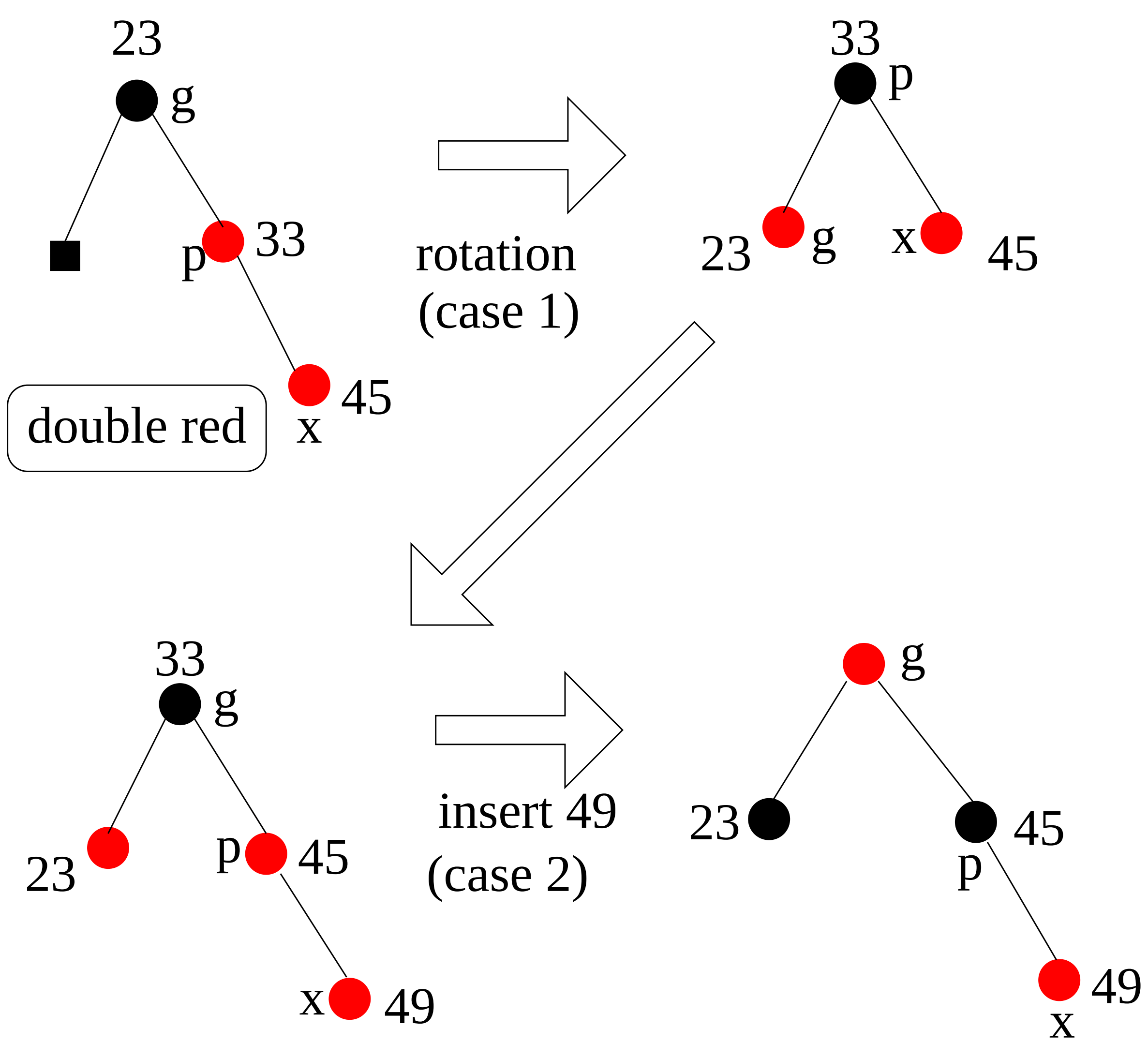
Analysis of case 2

- The color of g is replaced with red and both p, u are colored as black.
- The color changes suggested above may lead to reappearance of the double red at g and its parent – case 1 or repetition of case 2.
- Case 1 recoloring discussed already.
- For a repeat of case 2, steps outlined above applied repeatedly.
- Finally, if case 2 appears with g as root, then color it black and terminate.
- Recoloring, therefore, may be applied O($\log n$) times in worst case.
- But if case 1 appears, then one restructuring is need.
- Usually not more than 2 rotations are needed.

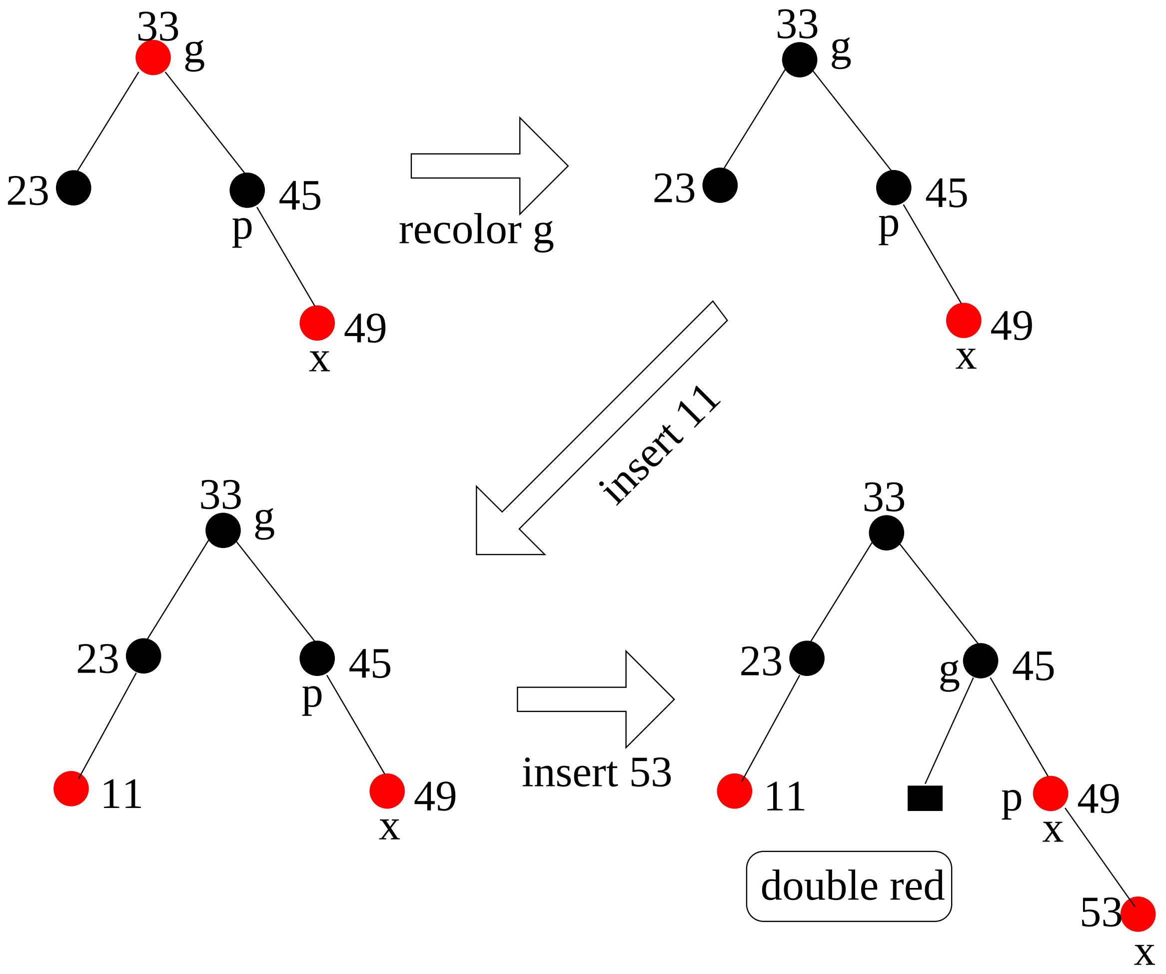
Insertion: Example



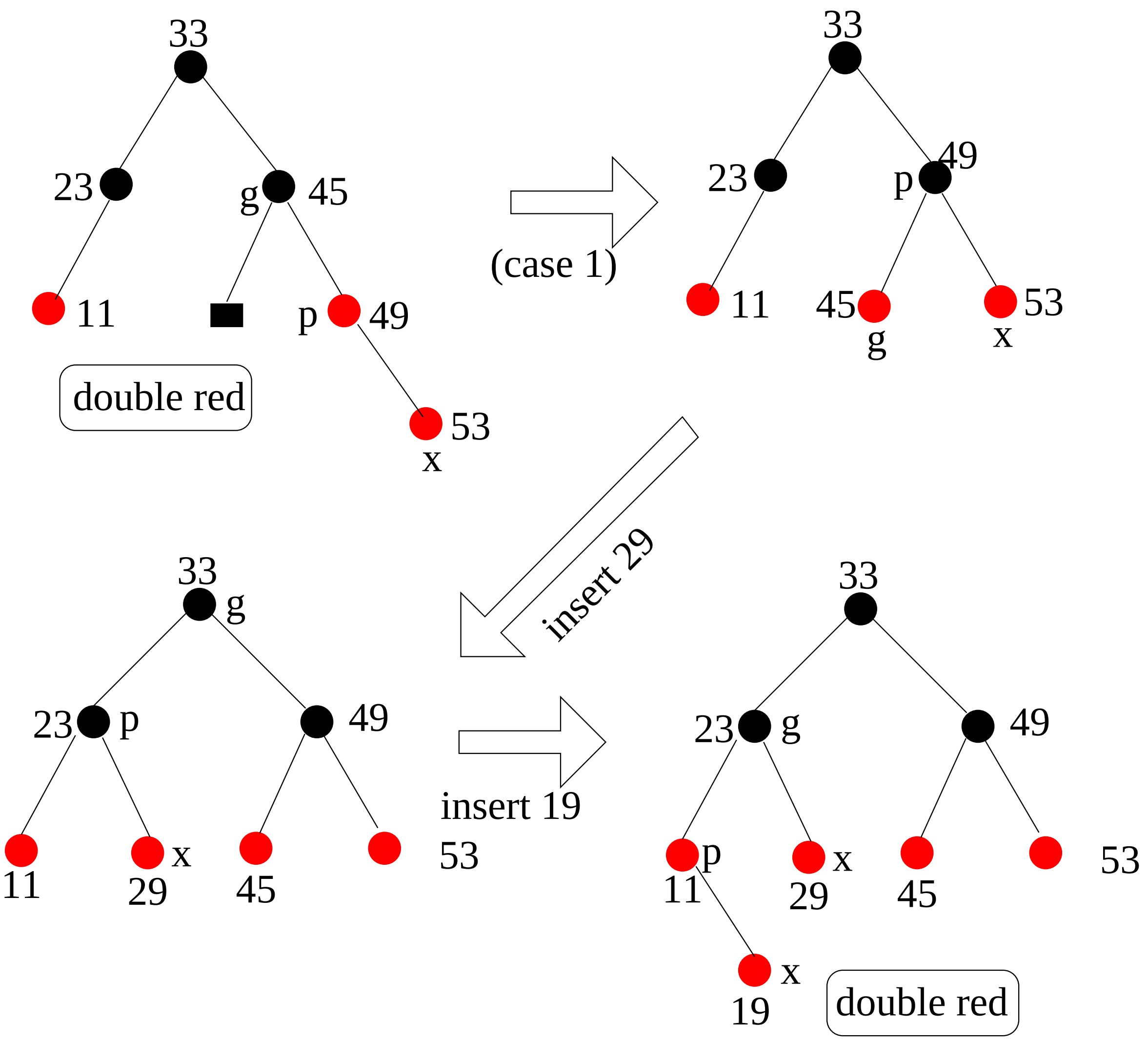
Insertion: Example



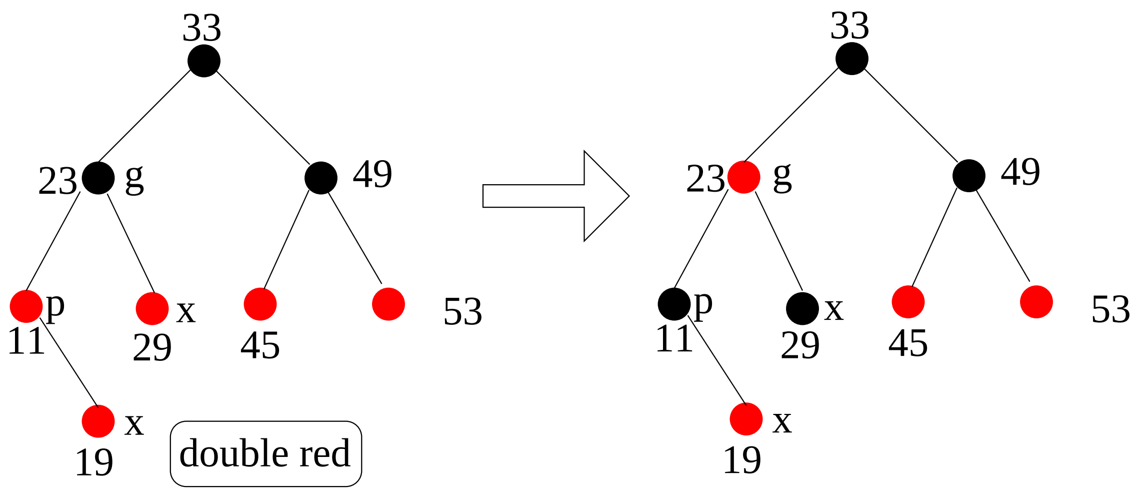
Insertion: Example



Insertions: Example



Insertions: Example



Deletion on red black tree

- For removal, proceed as a BST deletion.
- Either a node with an external child is removed, or
- An internal node which is the inorder successor of the node to removed. That inorder successor must have an external child.
- So, just consider the case of removing an internal node v with an external child say w, and color w has to be black.

Simple deletion

- If removed node v is red then no recoloring required.
  - Tree originally a valid red-black tree. So, a red node cannot have a red parent – ruling out a double red configuration.
  - The remaining child r of v must also be black and is also an external node.
  - Therefore, the black height is undisturbed.

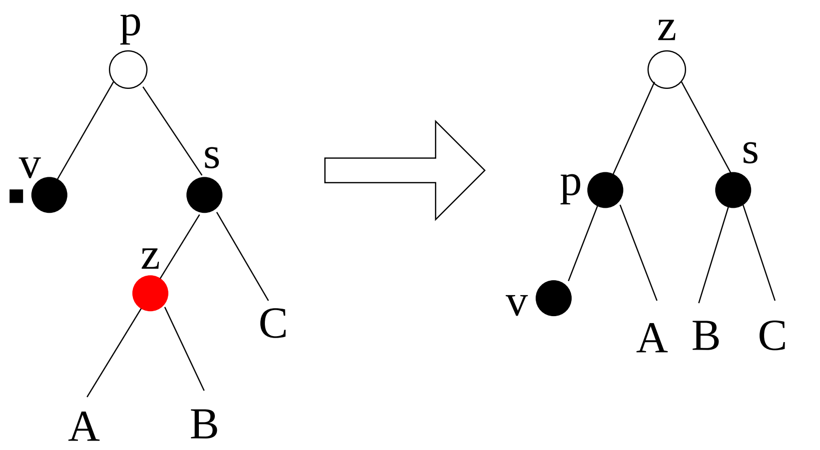
The double black configuration

- But when a black node is removed, its descendants must be provided with additional black ancestor to maintain the black height.
- We can either find an appropriate red node and color it black or we must restructure and recolor the nodes of the tree.
- This configuration arises when a black node with an external child is removed.
- The cases we consider for deletion as general cases reproducible from the attempt to eliminated double black configuration.

Deletion: double black cases

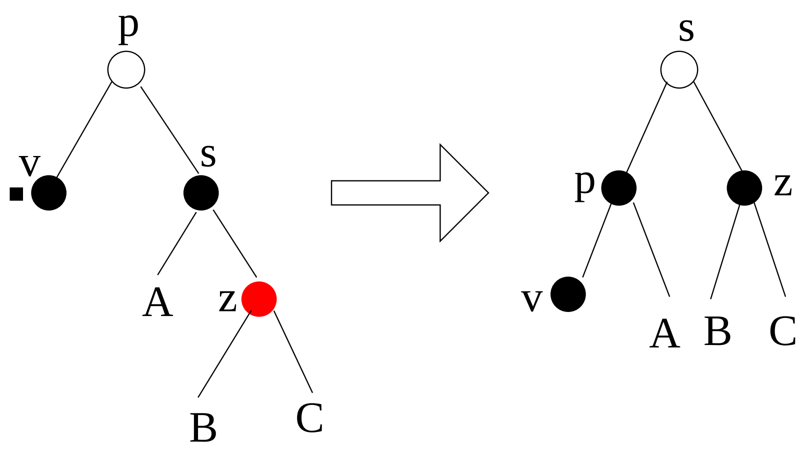
- We consider the cases based on the color combinations of the parent of the removed node, the sibling of the removed node, and the children nodes of the sibling.
  - Case 1: sibling are black, but sibling has a red child – parent can be black or red.
  - Case 2: sibling and both the children of sibling are black – parent can be black or red.
  - Case 3: sibling is red – parent must be black.
- As the node carries two black color, the configuration is termed as a *double black* configuration.

Deletion: double black case 1a



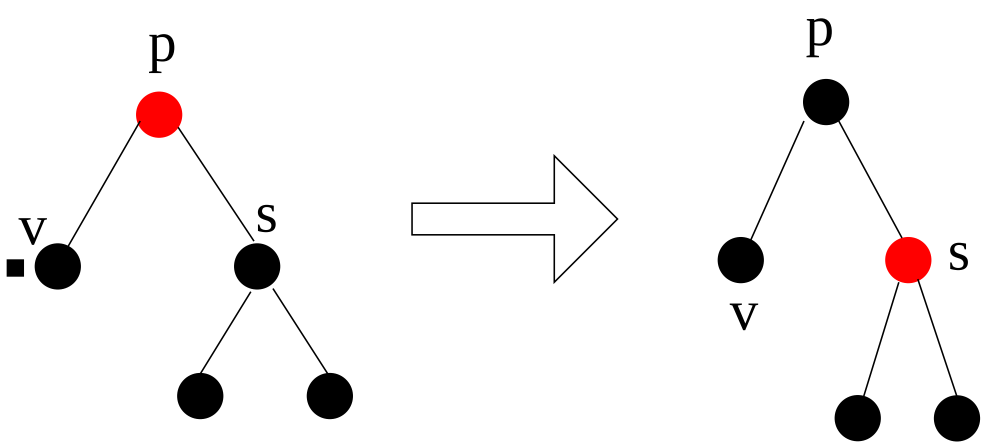
- The color of p is transferred to z.
- p is colored black.
- The nodes in subtree A and B lose a red ancestor each.
- But A gets black p as ancestor in stead of black s.
- B remains under s.
- Thus subtrees A, B do not lose any black ancestor.
- There is no change in colors of any of C's ancestor.

Deletion: double black case 1a



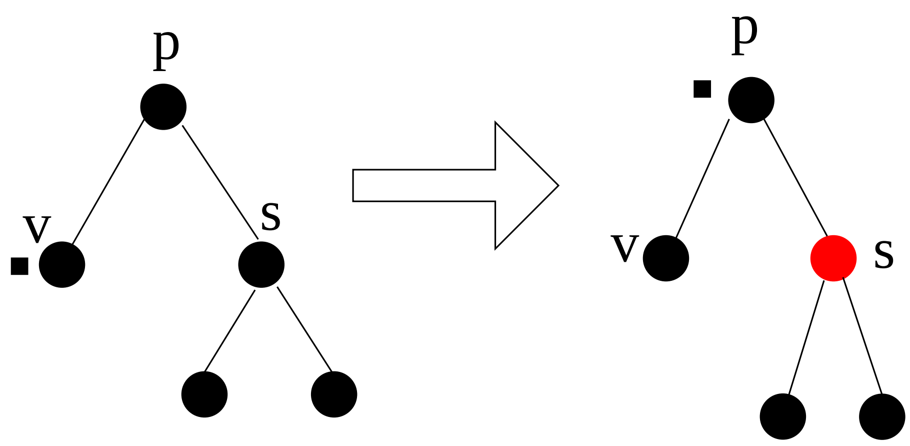
- In case 1b the color of p is transferred to s.
- p gets black color and z's color turns black.
- The nodes in subtree A and B lose a red ancestor each.
- But A becomes a subtree under p instead of s. Since s was black before restructuring there is no disturbance in number of black ancestor of nodes in A.
- B and C become children of z which turned black.

Deletion: case 2a



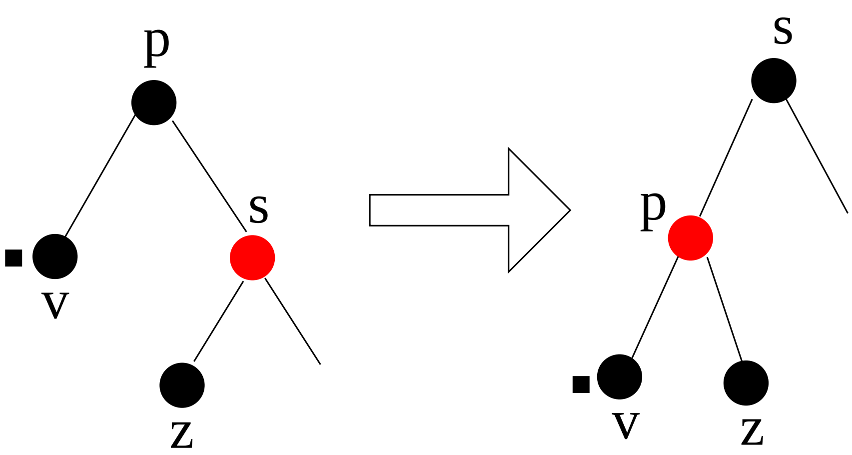
- Parent of v is red, sibling s and its children are black
- In this case absorb the extra black at p and re-color s as red.
- It ensures number of black ancestors for nodes below s remain unchanged.

Deletion: Case 2b



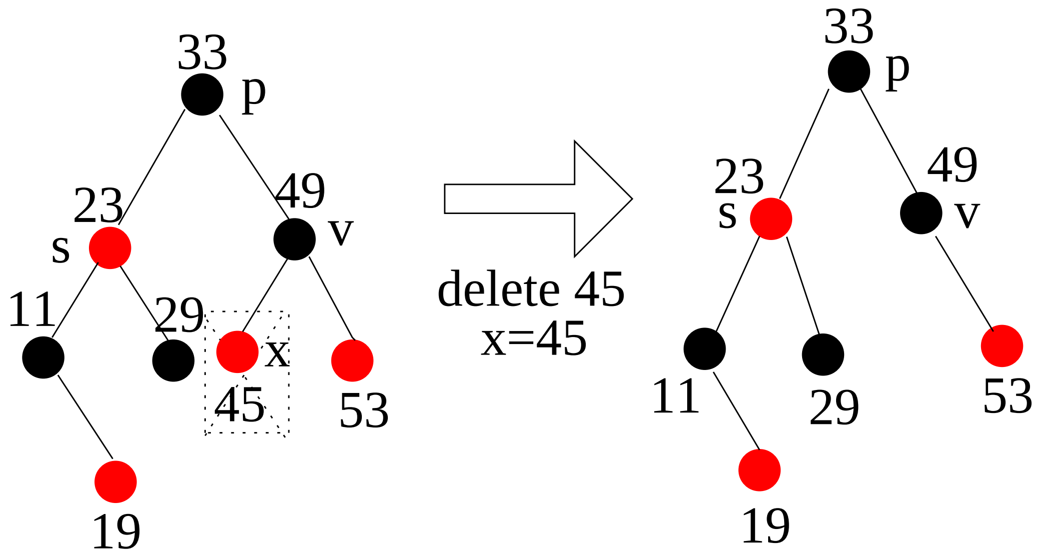
- Sibling s, its both children and parent p are black.
- s gets red color and extra black token pushed up to p.
- Recoloring shifts the problem of double black to p.
- So the refixing can generate one of the three cases at p.
- If p is root simply eliminate the extra token.

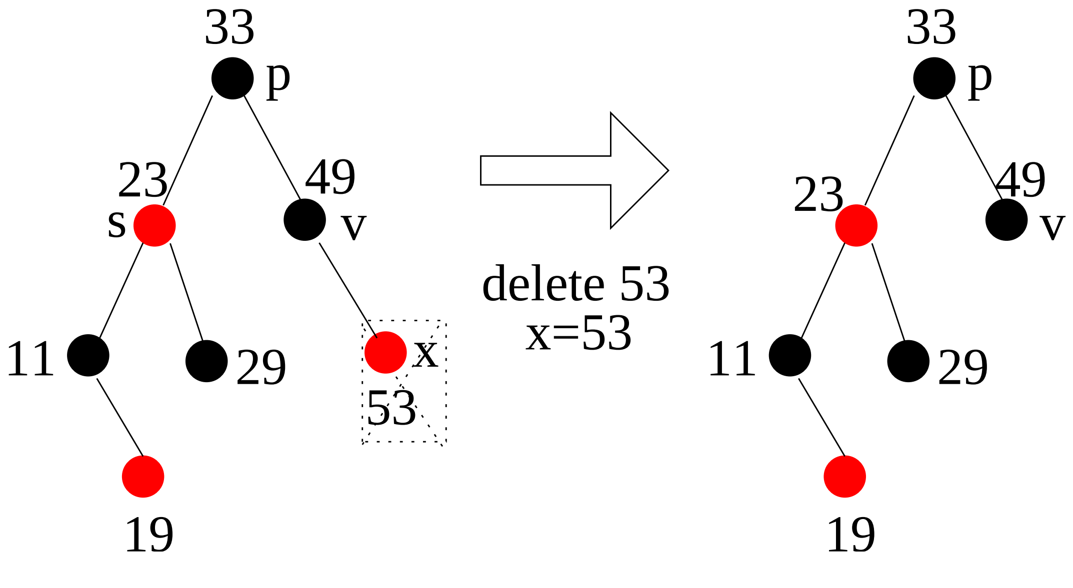
Deletion: Case 3



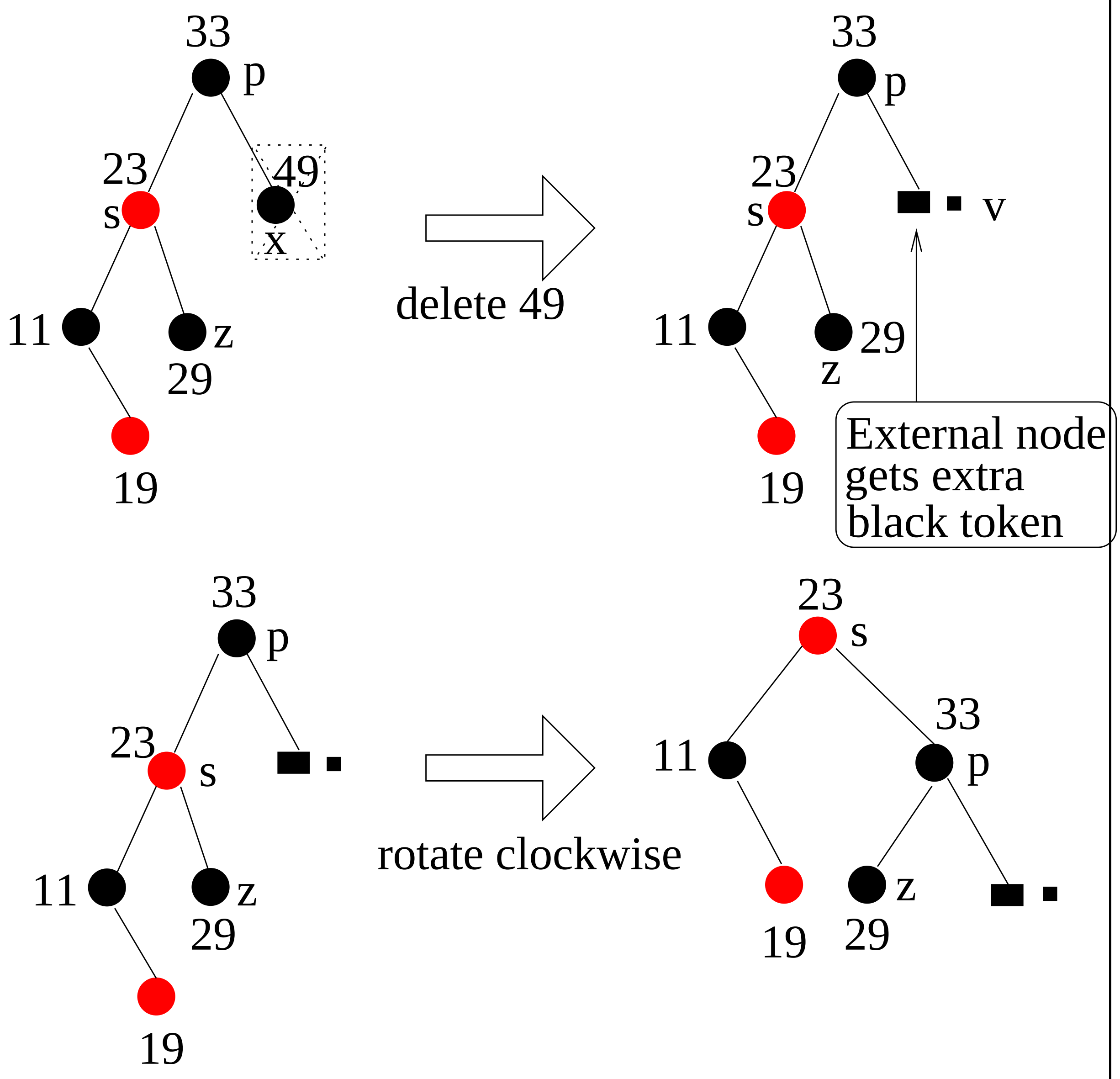
- Sibling s of v is red. It means the parent p is black.
- In this case apply a rotation to bring s to the position of p.
- Swap colors of p and s. So p is now colored red.
- But as both children of s were black and one of them, say z, becomes a new child of p, it gets both black children such that v has extra black token.
- Recolor p as black remove extra black from v and recolor z as red.

Deletion: Example





Deletion: Example



Deletion: Example