- Recursion and Applications
- Red Black Tree

OUR TEAM

Md Abdullah Al Forhad
2013-1-60-029

Red Black Tree

Tasneem Jannat Sultan
2013-1-60-027

Recursion

Md Nadim Hussain
2013-1-60-067

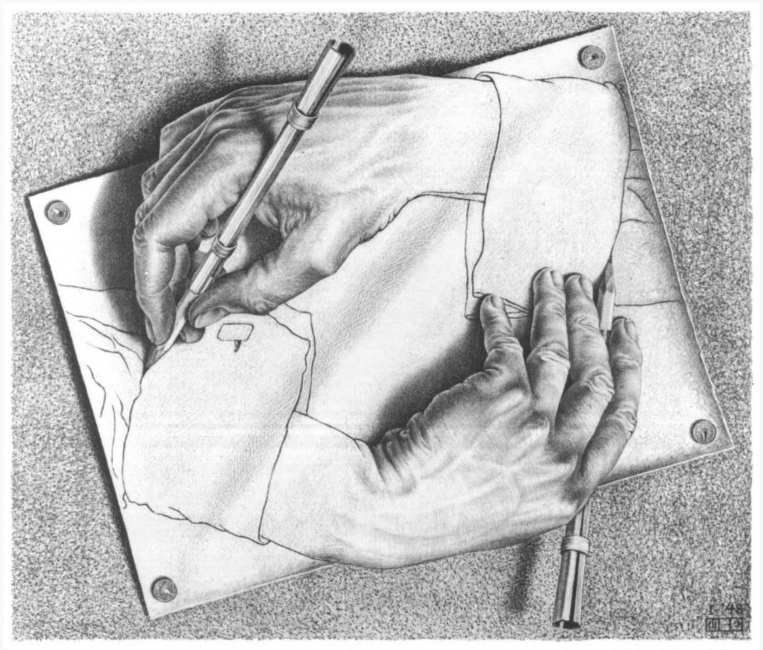
Introduction

Overview

- Introduction
    - What is recursion?
- Applications
    - How to think recursively?
    - Some common applications
- Red Black Tree
- Conclusions

What is Recursion?

- Recursion is a principle closely related to mathematical induction.
- In a recursive definition, an object is defined in terms of itself.
- We can recursively define sequences, functions and sets.
- Recursion is an extremely powerful problem-solving technique.
- It breaks a problem in smaller identical problems.
- It is an alternative to iteration, which involves loops.



A typical example of recursion

How to Think Recursively?

- When coding, do not concern yourself how recursion is unfolding during execution.

- Rather, think of yourself as a boss, and treat each recursive call as an order to one of your trusted subordinates to do something.

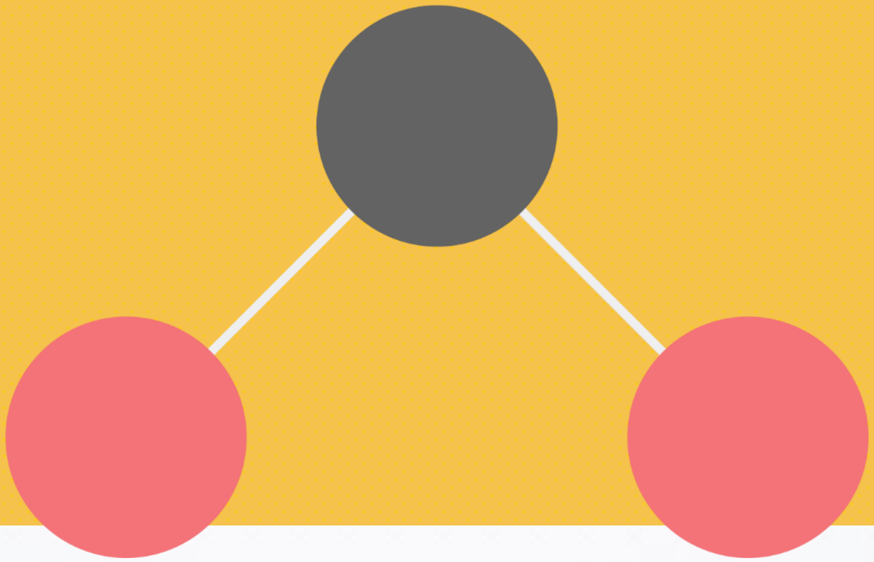
- As a boss, you need not worry how the subordinate does their work. Instead, take the outcome of their work.

- Finally, take the outcome of the subordinate’s work, and use it to compute by yourself the final result.

Some common applications

Some common recurrence relations are:

- Golden Ratio : $\phi = 1 + (1/\phi) = 1 + (1/(1 + (1/(1 + 1/...))))$
- Factorial : $n! = n(n-1)! = n(n-1)\cdots 1$
- Fibonacci Numbers : $f(n) = f(n-1) + f(n-2)$
- Catalan Numbers : $C_0 = 1 \quad C_{n+1} = (4n+2)C_n/(n+2)$
- Red Black Tree

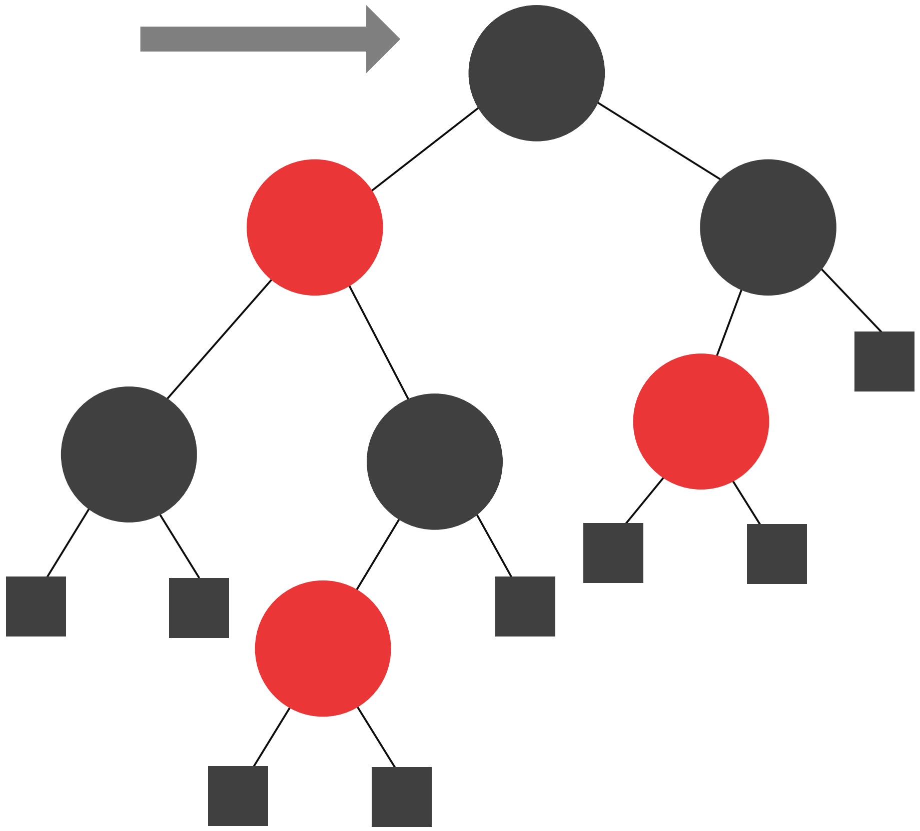


RED BLACK TREE

What is Red Black Tree?

Red Black Tree is a type of self balancing binary search tree. Its design ensured five specific properties are preserved.

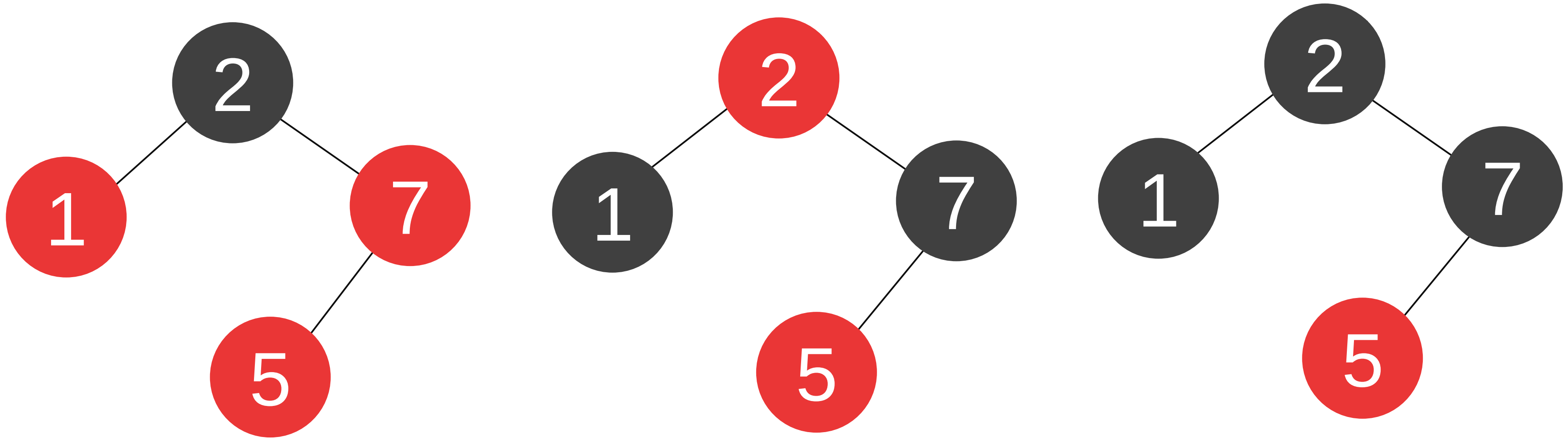
- Every node is either red or black
- The root is black.
- Every leaf (NULL pointer) is black
- If a node is red, both children are black
- Every path from root to descendent leaf contains the same number of black nodes



Insertion into a Red-Black Tree

Algorithm for insertion: Color changing

- Case 0: The color of a new leaf is always red. If parent of new leaf is black, then done.
- Case 1: We just change colors.



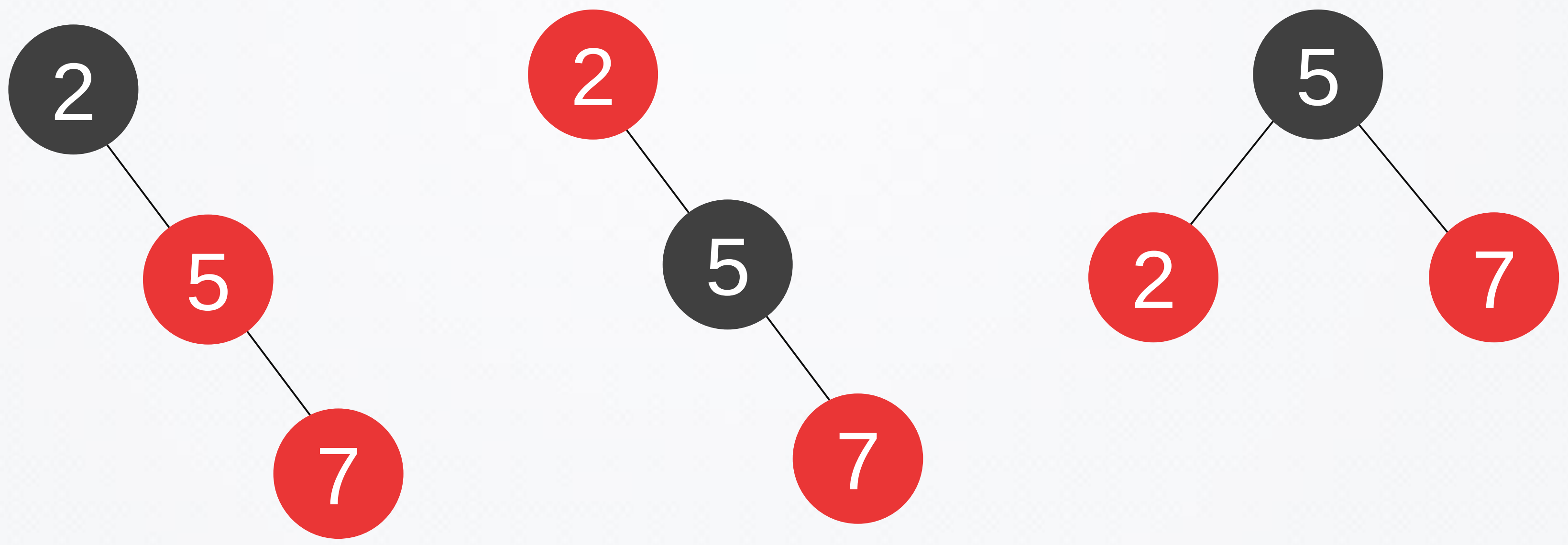
1. If sibling of parent also red, change color of parent and its sibling to black, and change color of the grandparent.
2. Ensure color of the root is black.

Insertion into a Red-Black Tree

Tree rotation

What if parent does not have a red sibling?

- Case 2: One rotation suffices



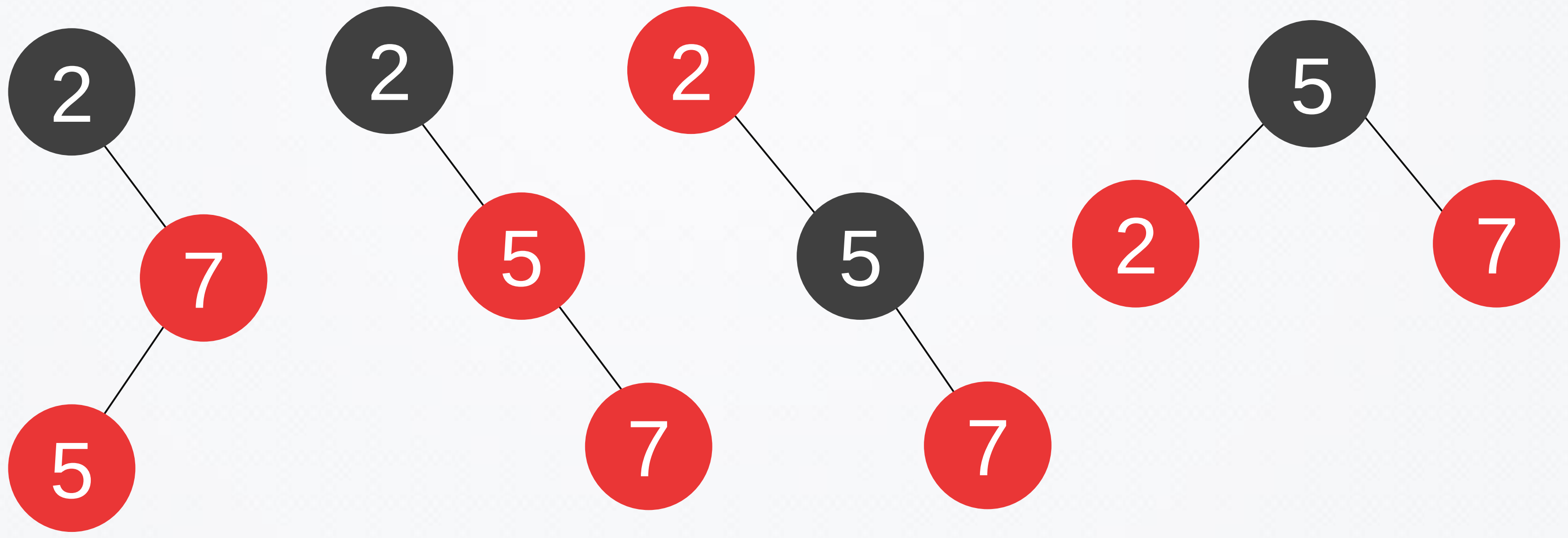
1. Change the color of the parent to black, change the color of the grandparent to red.
2. Rotate to restore the 4th invariant.

Insertion into a Red-Black Tree

Double Rotation

One rotation works for right-right or left-left tree.

Case 3: Tree is right-left (or left-right), then we need double rotation



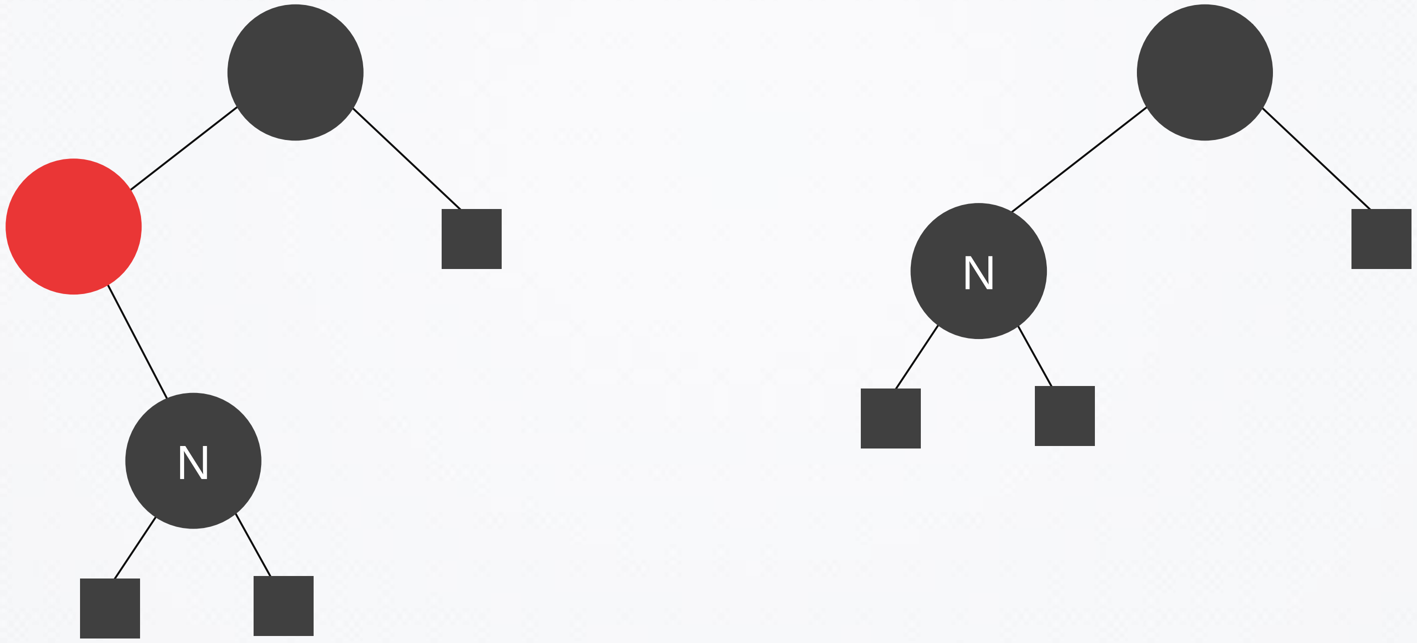
1. Rotate right-left tree to right-right tree.
2. Change color of parent and grandparent.
3. Rotate to restore the 4th invariant.

Pseudo Code of insertion

```
bool insert ( Tree &root,
Item_Type item )
if(root == NULL)
root = new black node;
return true;
else if(item == root->data)
return false;
else if (item < root->data)
if(left == NULL)
left = new red node;
return true;
else if((left == red) && (right
== red))
change left and right to black
and color the local root red;
if(insert(left,item))
if(left grandchild == red)
change left to black
and color the local root red;
rotate the local root right;
else if(right grandchild == red)
rotate the left child left;
change left to black
and color the local root to red;
rotate the local root right;
else // item > root->data:
exercise
if(local root is root of the
tree)
color the root black.
```

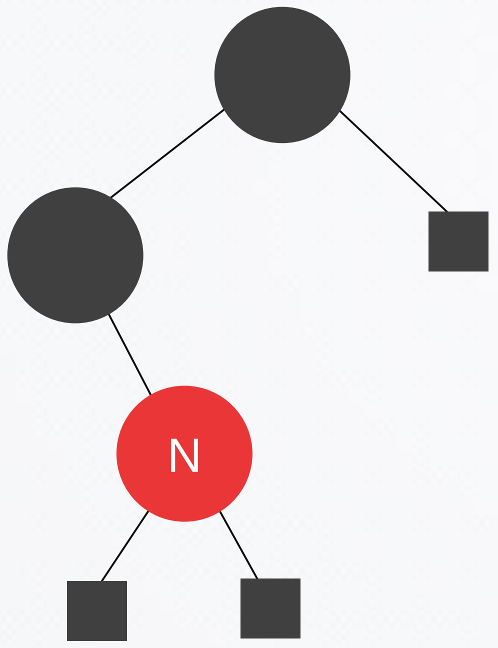
Deletion in Red Black Tree

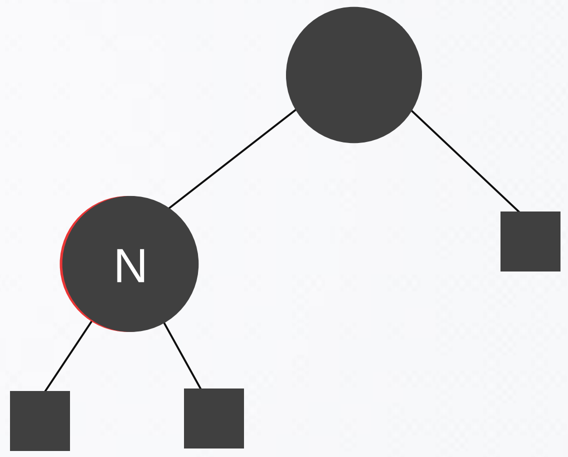
Case A: Delete red node



Deletion in Red Black Tree

Case B: Delete black node

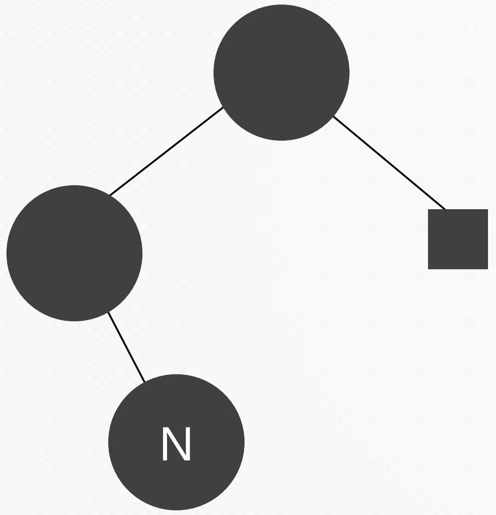


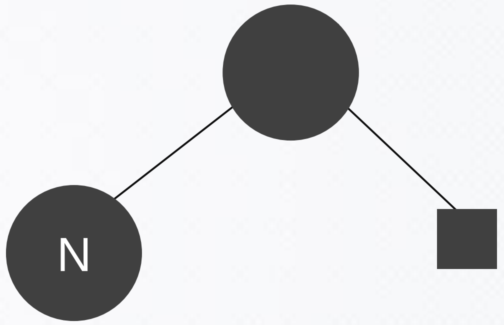


,

Deletion in Red Black Tree

Case C: Delete black node with black child

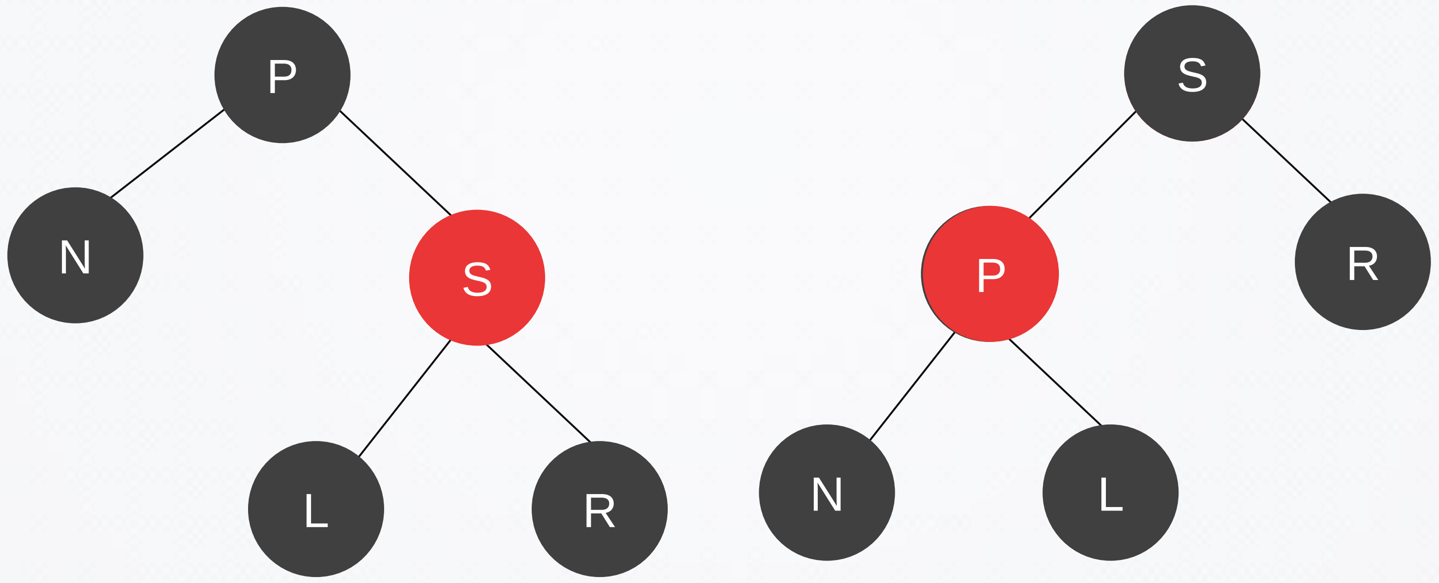




But It creates problem with red black tree property no 5
5. Every path from root to descendent leaf contains the same number of black nodes

Deletion in Red Black Tree

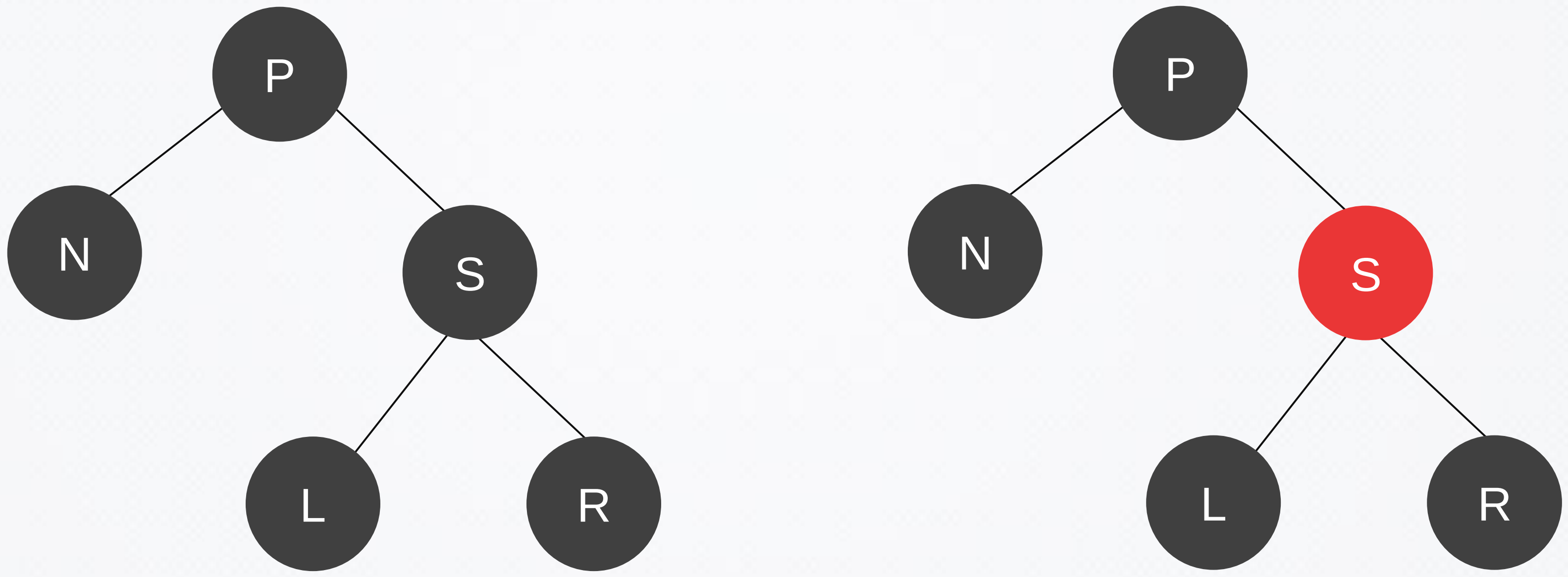
Case 1: If siblings of current node is red



1. First rotate the parent so that current nodes parent is not changed
2. Recolor the parent red and former sibling black

Deletion in Red Black Tree

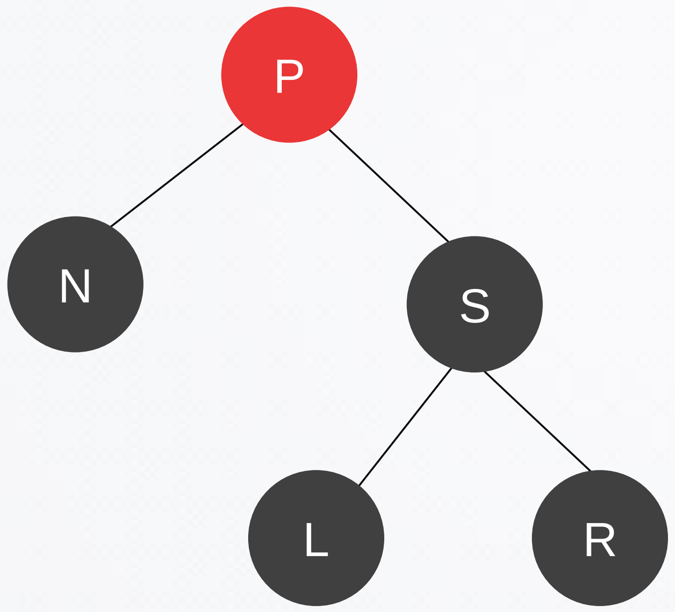
Case 2: If siblings of current node is black

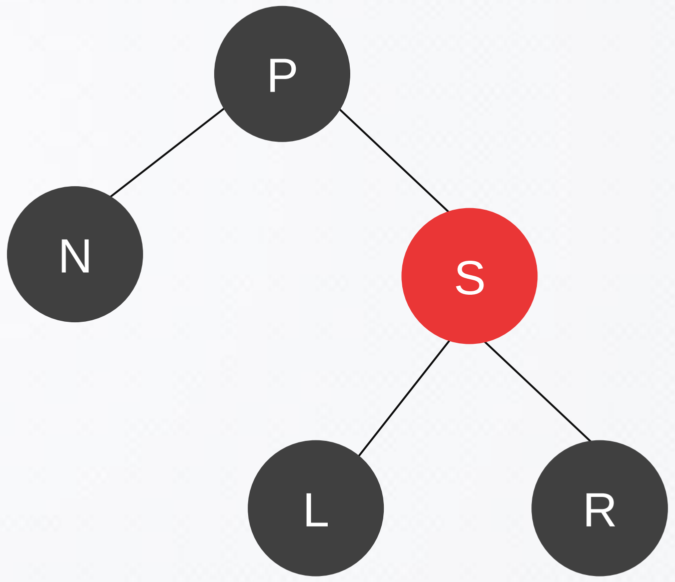


1. Change the color of sibling

Deletion in Red Black Tree

Case 3: Parent of current node red

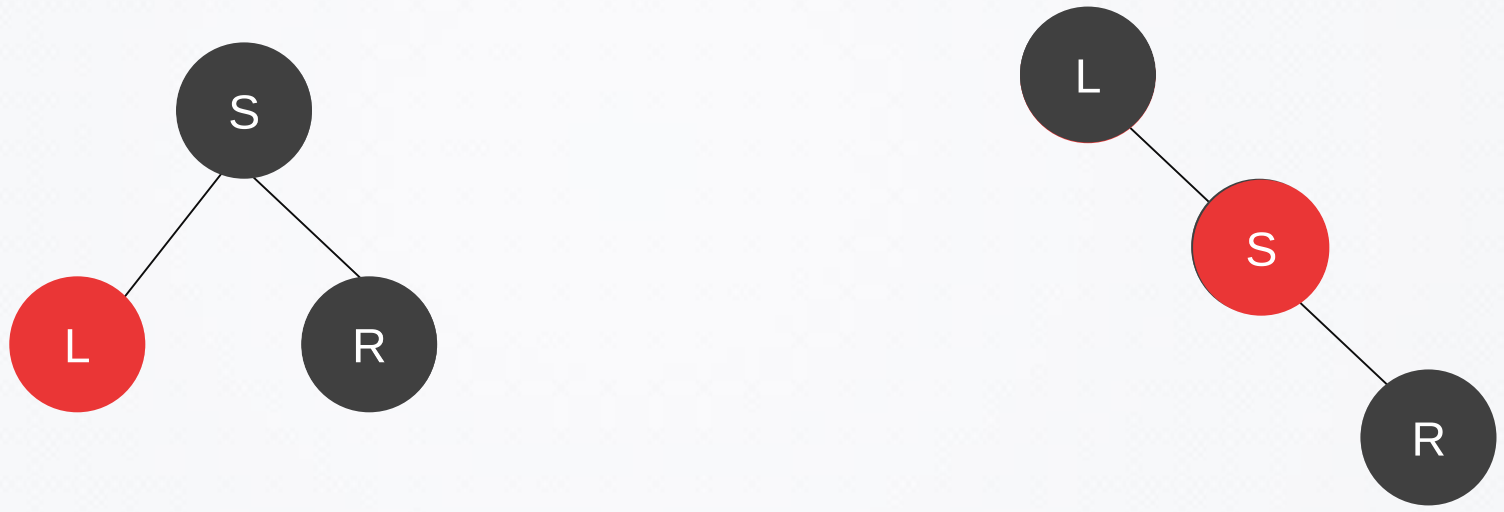




1. Change the color of parent and sibling

Deletion in Red Black Tree

Case 4: Siblings left child red and right child black



1. Rotate it
2. Recolor it

Conclusion

Red–black trees offer worst-case guarantees for insertion time, deletion time, and search time.

The AVL tree is another structure supporting O(log n) search, insertion, and removal. It is more rigidly balanced than red–black trees, leading to slower insertion and removal but faster retrieval.

,

Any Questions?

,

Thank You!

,