Chapter 12
Advanced Data Structures

Red-Black Trees

- add the attribute of color (red or black) to links/nodes
- red-black trees used in
  - C++ Standard Template Library (STL)
  - Java to implement maps (or dictionaries, as in Python)

Red-Black Trees

- a red-black tree is a BST with the following properties:
  - every node is either red or black
  - the root is black
  - if a node is red, its children must be black
  - every path from the root to a null link contains the same number of black nodes
    - perfect black balance
  - the height of an N-node red-black BST is at most $2\lg(N+1)$, so
    - search, insertion, and deletion are $\lg N$ operations

Red-Black Trees

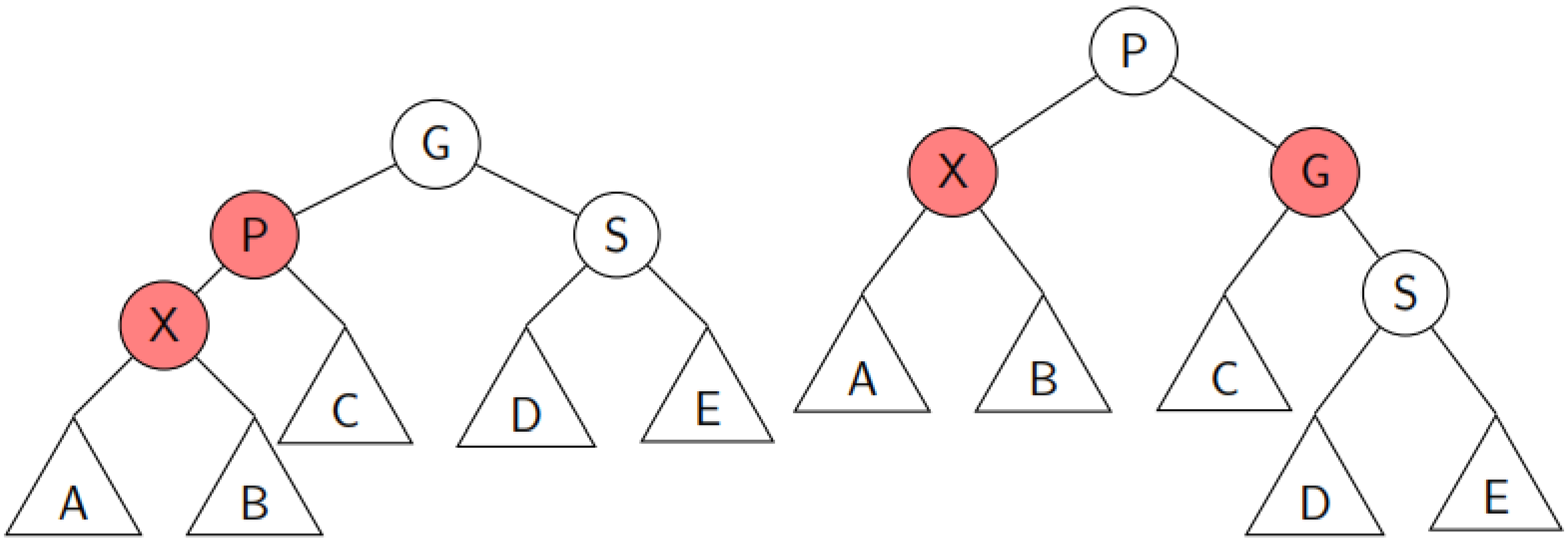
- building a red-black tree
  - in order to maintain perfect black balance, any new node added to the tree must be <u>red</u>
  - if the parent of the new node is <u>black</u>, all is well
  - if the parent of the new node is <u>red</u>, this violates the condition that red nodes have only black children
    - fix with <u>rotations</u> similar to those for AVL and splay trees
    - can be used to maintain the red-black structure at any point in the tree, not just at insertion of a new node

Red-Black Trees

- top-down insertion: color flips
  - to preserve perfect black balance, a newly inserted node must be red
  - in top-down insertion, we change the tree as we move down the tree to the point of <u>insertion</u>
    - the changes we make ensure that when we insert the new node, the parent is <u>black</u>
    - if we encounter a node X with two red children, we make X red and its children black
      - if X is the root, we change the color back to black
    - a color flip can cause a red-black violation (a red child with a red parent) only if X's parent P is <u>red</u>

Red-Black Trees

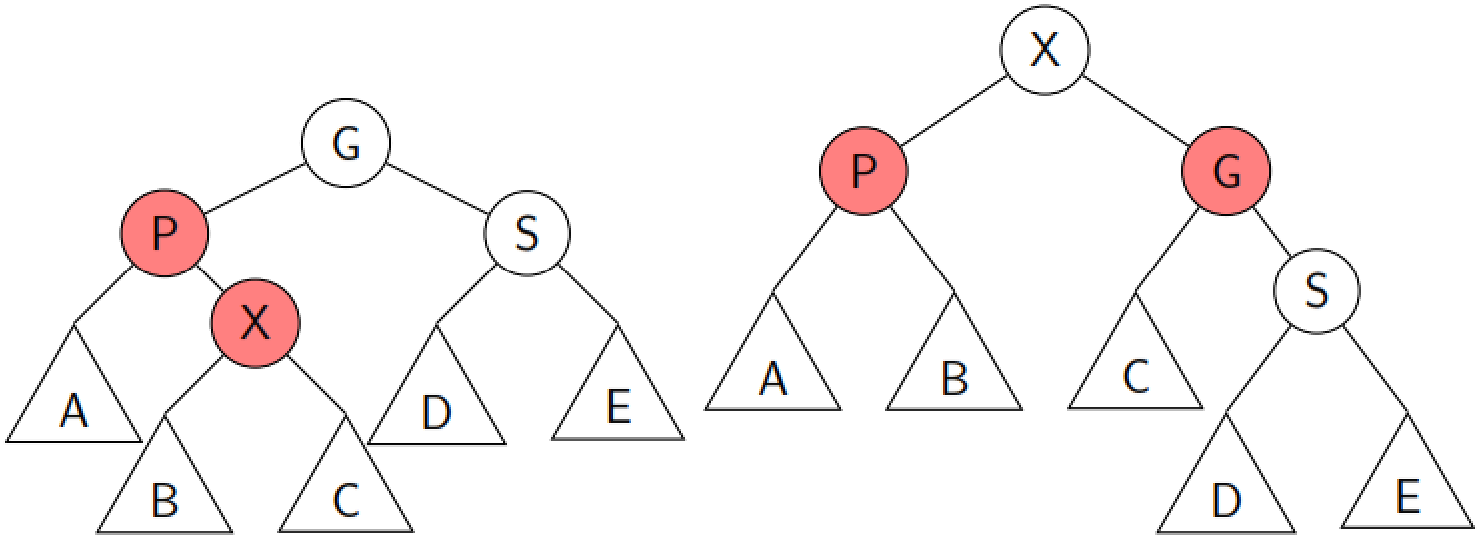
- red-black tree rotations
  - case 1: the parent is red and the parent's sibling is black (or missing)



  - this is a single rotation and a color swap for P and G

Red-Black Trees

- red-black tree rotations
  - case 2: the parent is red and the parent's sibling is black (or missing)



  - this is a double rotation and a color swap for X and G

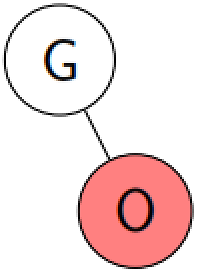
Red-Black Trees

- red-black tree rotations
  - the parent is red and the parent's sibling is <u>red</u>
    - this can't happen since it would mean that the parent and its sibling are both red
      - we changed all such pairs to <u>black</u> on the way down

- example: GOTCHA

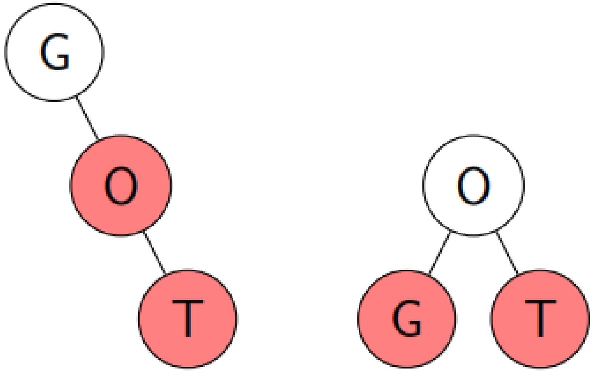
Insert G:

Insert O:

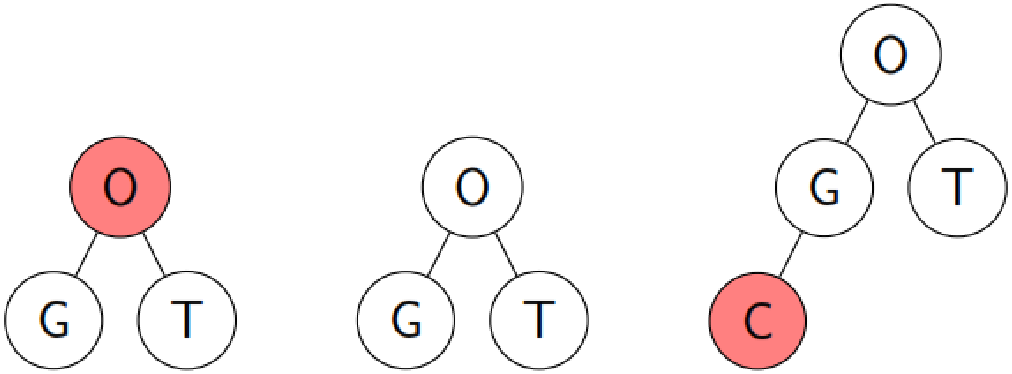


- example: GOTCHA (cont.)

Insert T:



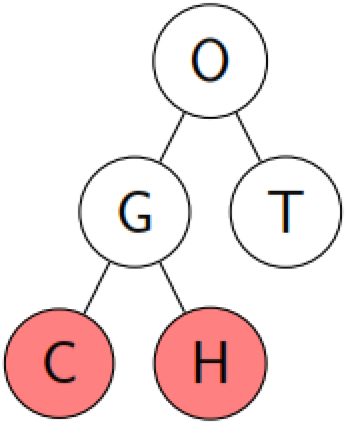
Insert C:



Red-Black Trees

– example: GOTCHA (cont.)

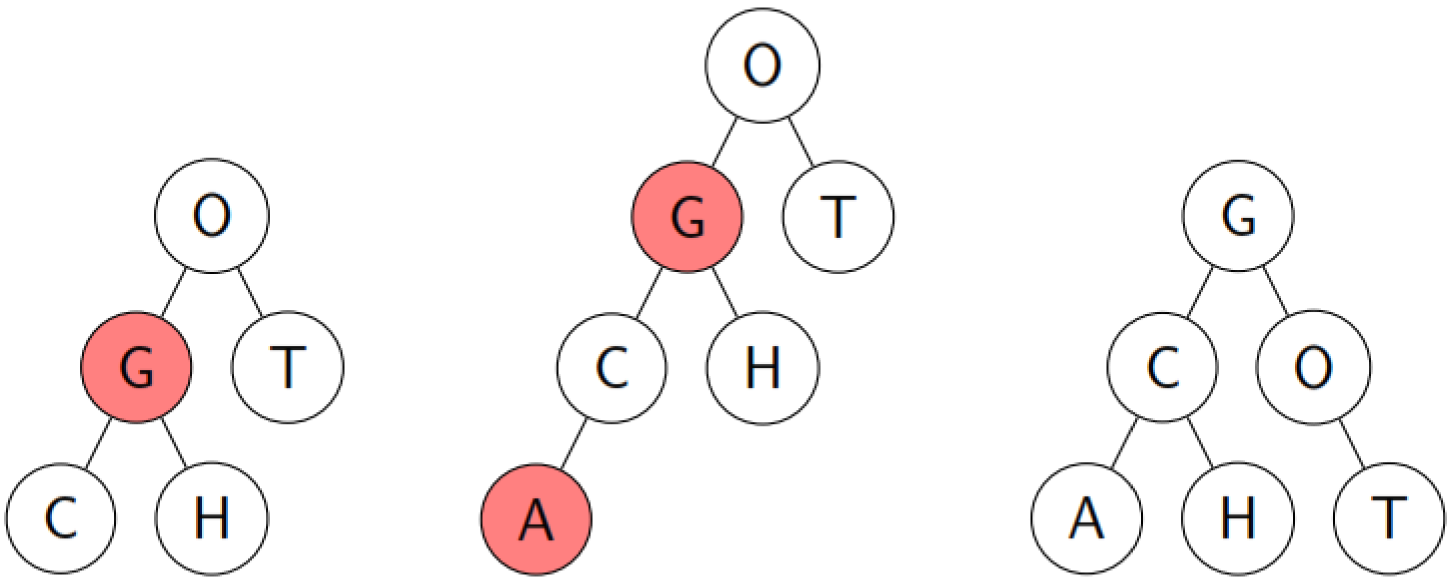
Insert H:



Red-Black Trees

- example: GOTCHA (cont.)

Insert A:



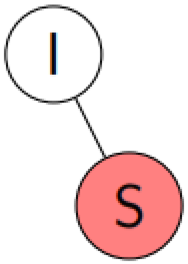
- the standard BST (rightmost) for GOTCHA is slightly shorter

Red-Black Trees

– example: ISOGRAM

Insert I:

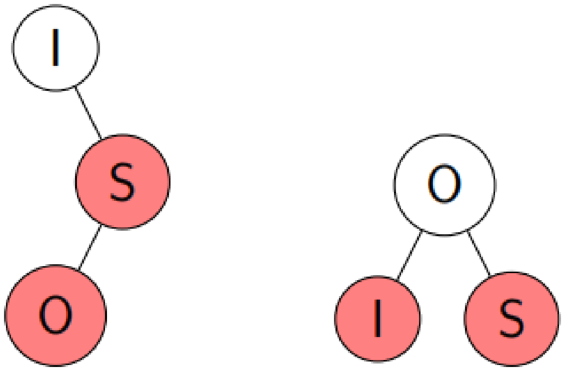
Insert S:



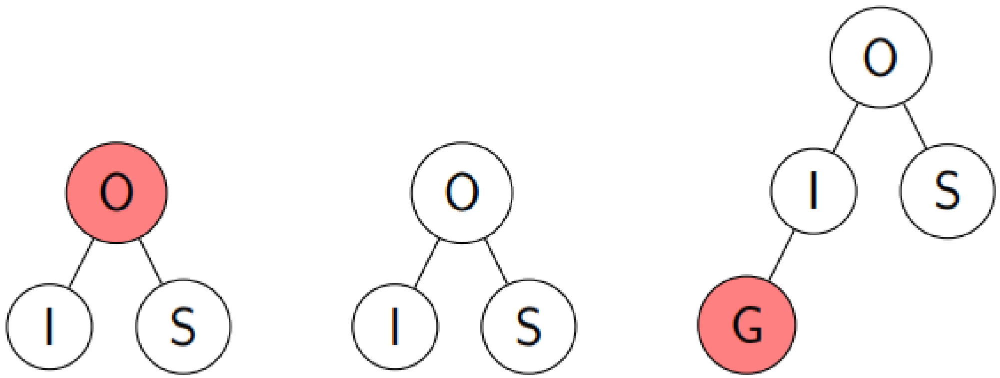
Red-Black Trees

- example: ISOGRAM (cont.)

Insert O:

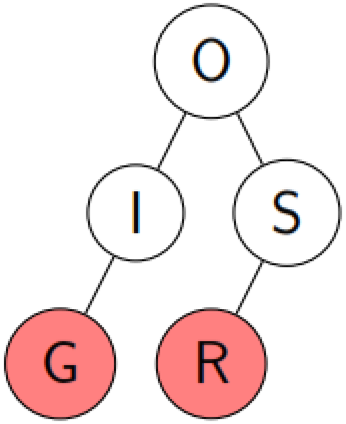


Insert G:



– example: ISOGRAM (cont.)

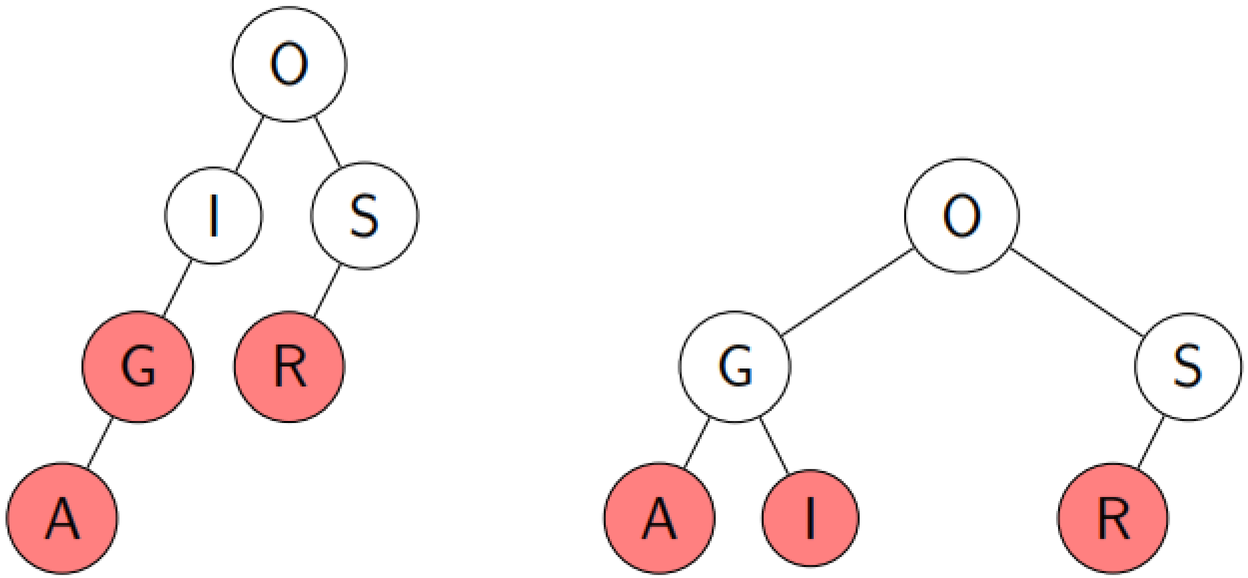
Insert R:



Red-Black Trees

– example: ISOGRAM (cont.)

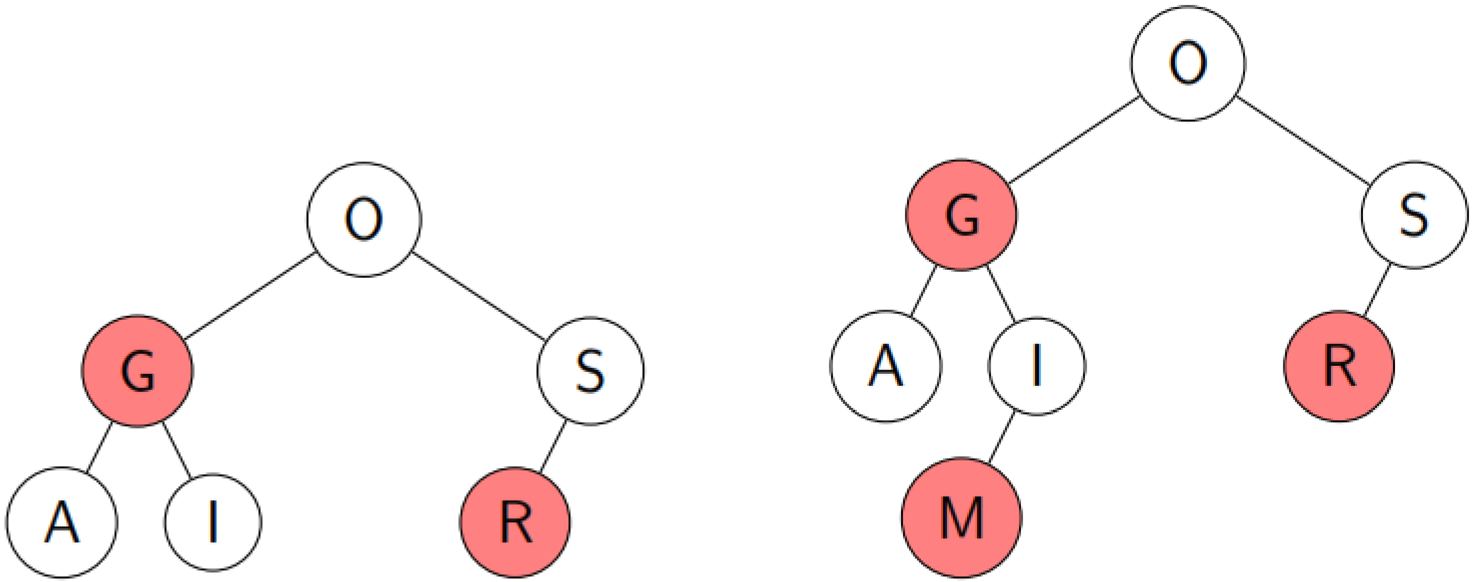
Insert A:



Red-Black Trees

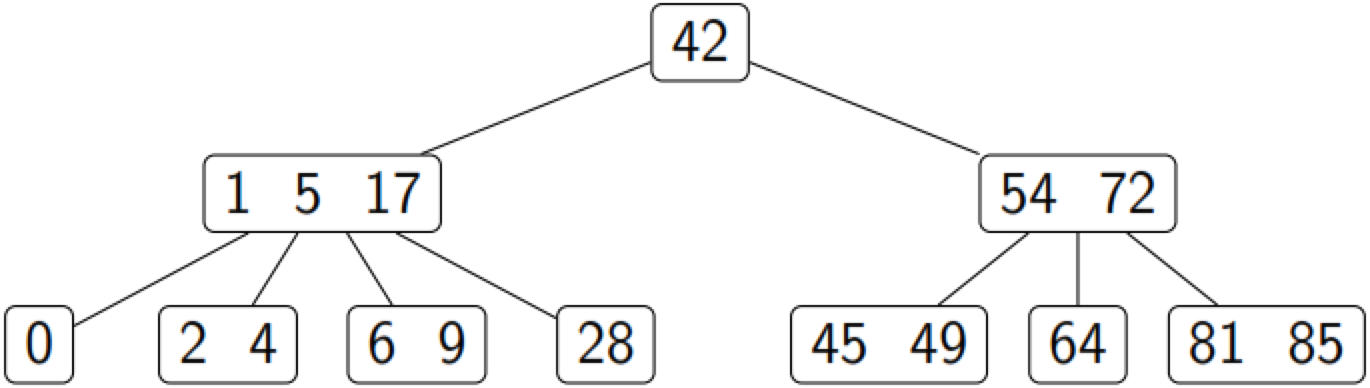
– example: ISOGRAM (cont.)

Insert M:



2-3-4 Trees

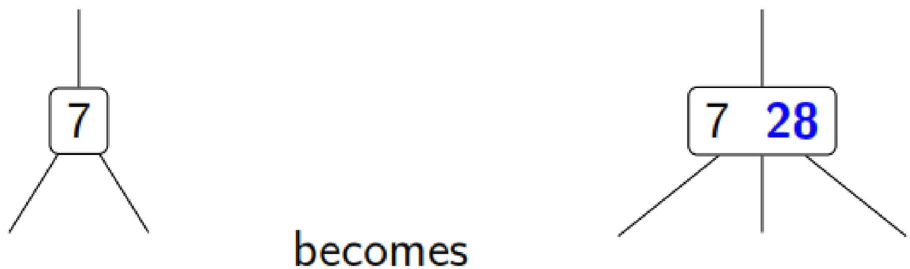
- 2-3-4 trees
  - useful because we can insert new items while maintaining perfect balance
  - a 2-3-4 tree consists of
    - 2-nodes: one key, two children
    - 3-nodes: two keys, three children
    - 4-nodes: three keys, four children



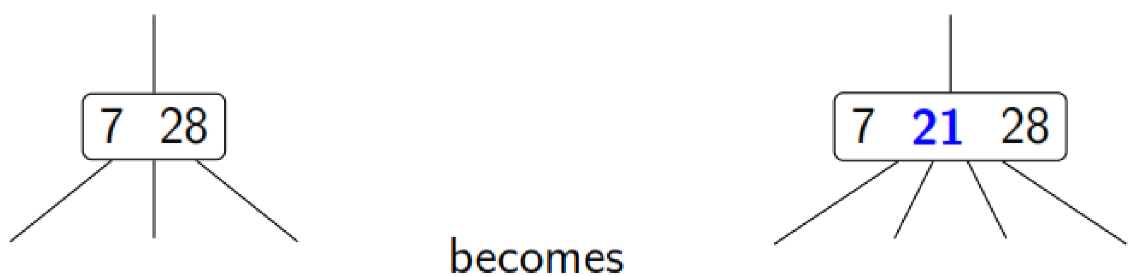
2-3-4 Trees

- insertion into 2-3-4 trees
  - insert the new key into the lowest existing node reached in the search

A 2-node becomes a 3-node:

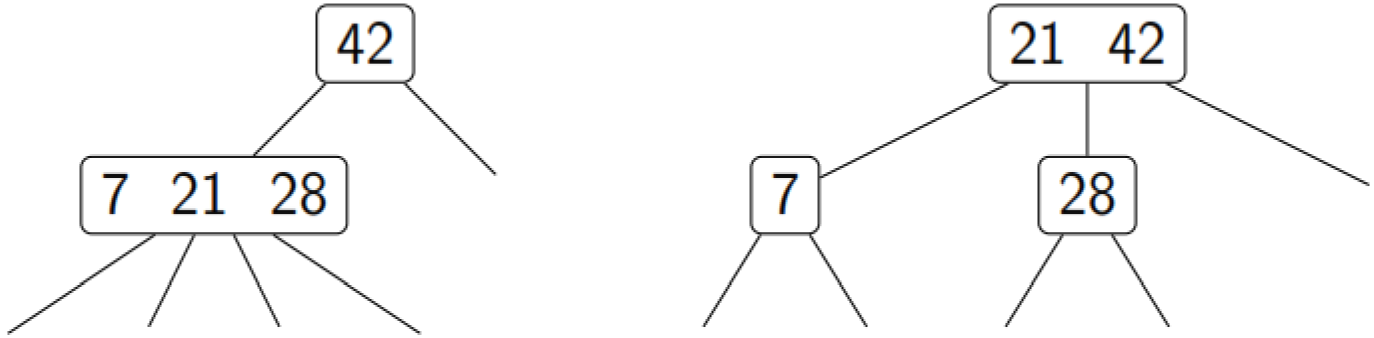


A 3-node becomes a 4-node:



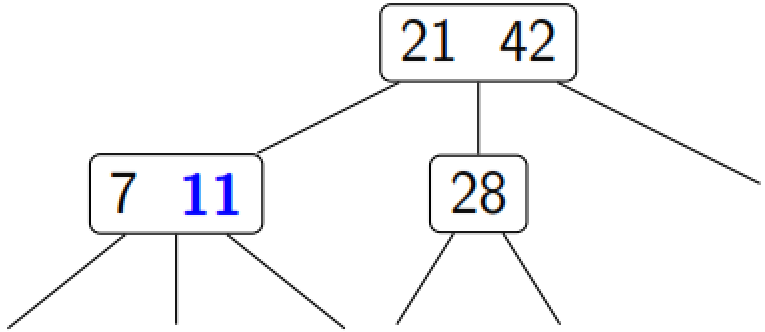
2-3-4 Trees

- what about a 4-node?
  - top-down insertion
    - as we move down the tree, whenever we encounter a 4-node, we move the middle element up into the parent node and break up the remainder into two 2-nodes



2-3-4 Trees

- what about a 4-node?
  - top-down insertion (cont.)
    - insertion, if done here, now reduces to the case of a 2-node or 3-node



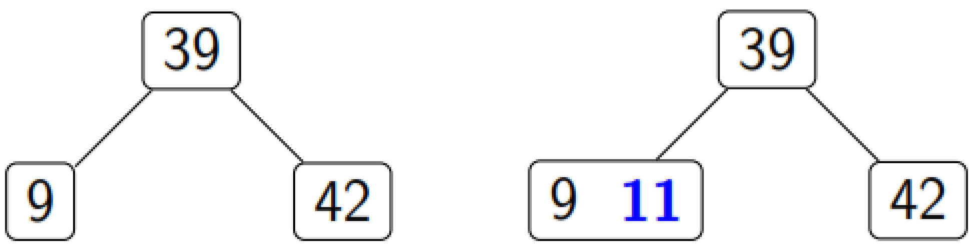
2-3-4 Trees

- top-down insertion
  - as we move down the tree, we split up 4-nodes as we encounter them through the following process
    - move the middle key up to the parent
    - split the remaining keys into 2-nodes
  - this action guarantees that the parent of any 4-node we encounter is a 2-node or 3-node
    - therefore, the tree will always have room to accept the middle element of the 4-node

2-3-4 Trees

- example: insert 42, 9, 39, 11, 27, 13, 33, 16, 28
  - the first three insertions are straightforward

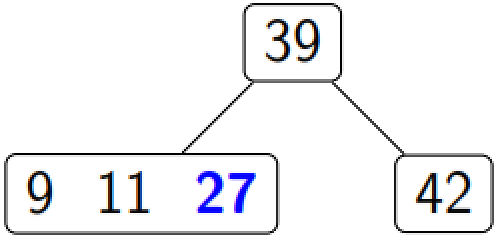
  - when inserting 11, we encounter a 4-node, which we split



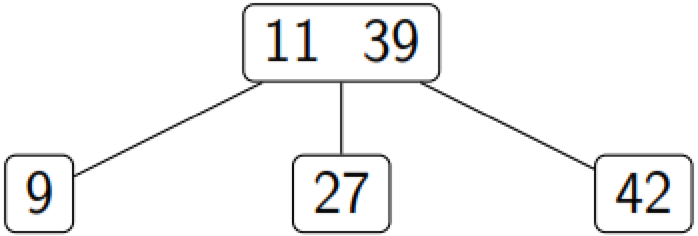
  - 39 is first promoted as a new root node
  - perfect balance is maintained in 2-3-4 trees by growing at the root

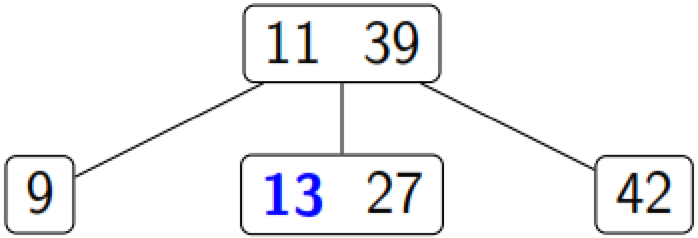
2-3-4 Trees

- example: insert 42, 9, 39, 11, 27, 13, 33, 16, 28 (cont.)
  - insert 27



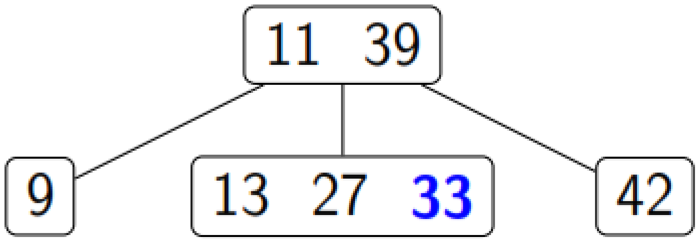
  - insert 13: first split 4-node



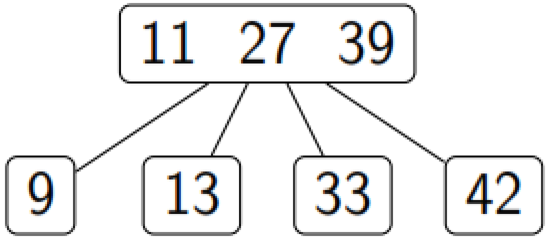


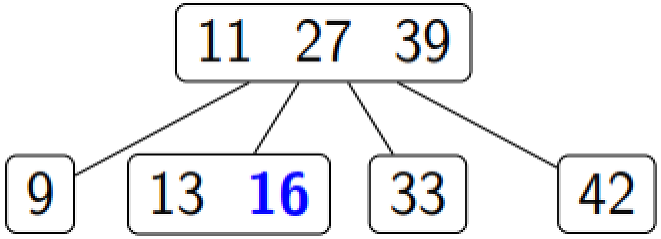
2-3-4 Trees

- example: insert 42, 9, 39, 11, 27, 13, 33, 16, 28 (cont.)
  - insert 33



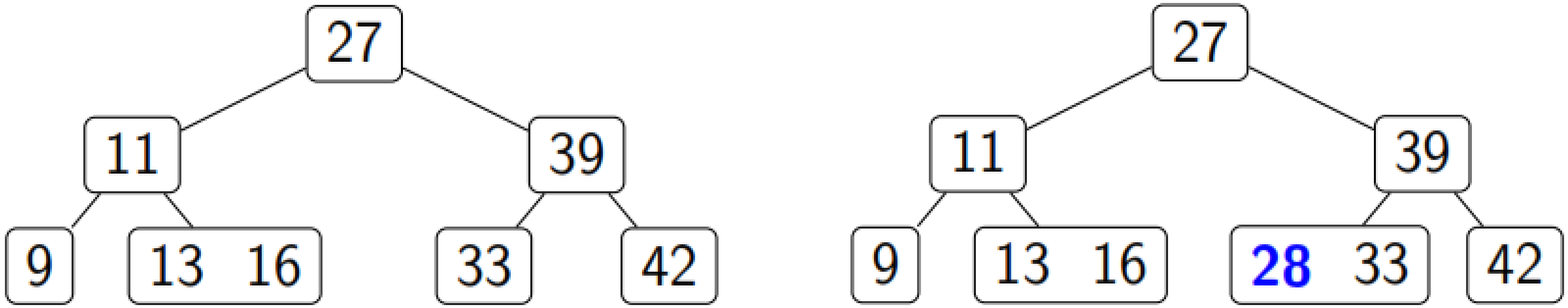
  - insert 16: split 4-node





2-3-4 Trees

- example: insert 42, 9, 39, 11, 27, 13, 33, 16, 28 (cont.)
  - insert 28: split 4-node at root



  - once again, growth at the root maintains perfect balance

2-3-4 Trees

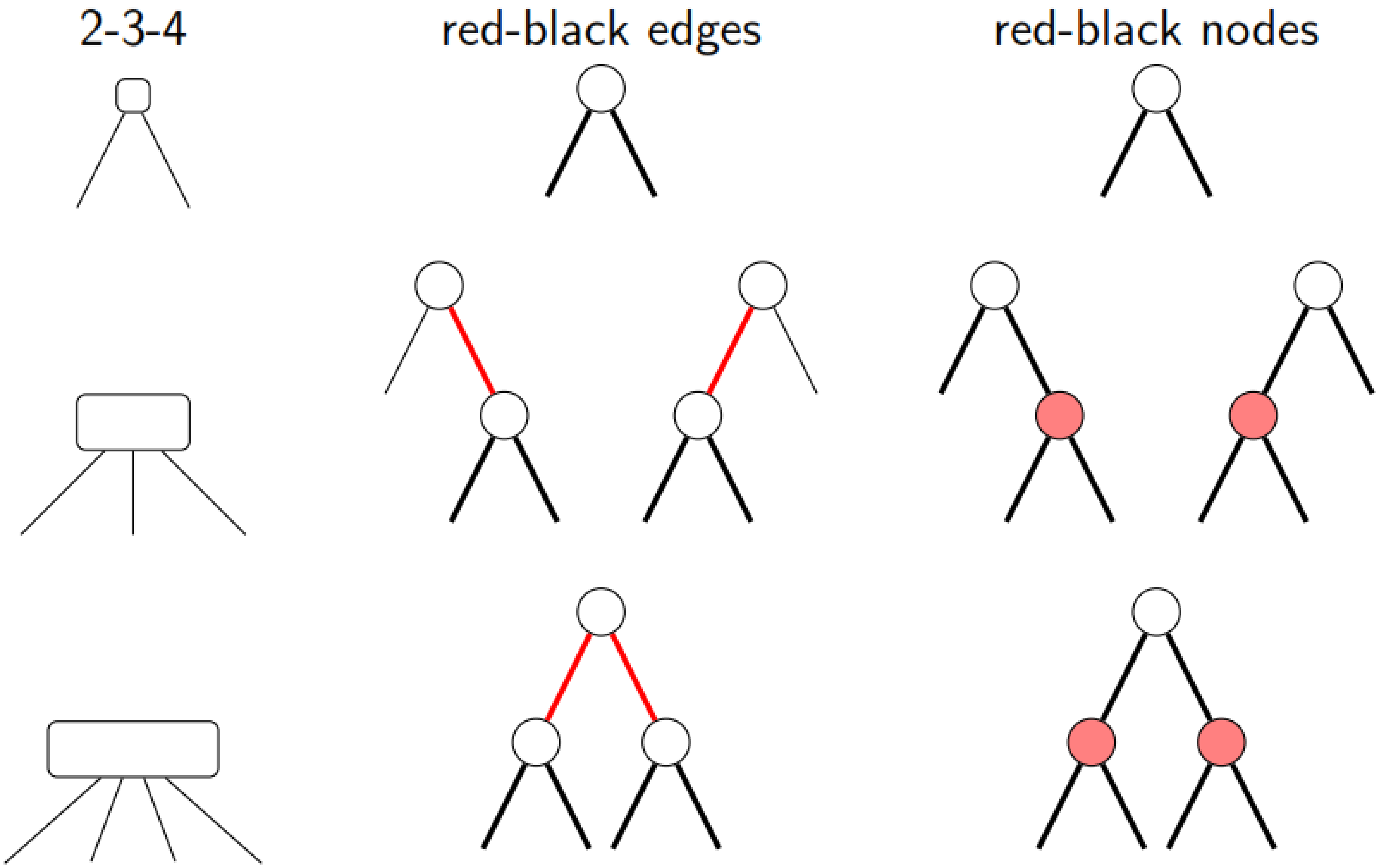
- complexity of 2-3-4 tree operations
  - the height of an N-node 2-3-4 tree is between $\log_4 N = \frac{1}{2} \lg N$ and $\lg N$
  - searching and inserting are both $\lg N$ operations
  - rather than splitting 4-nodes on the way down, we could also perform bottom-up insertion, starting at the insertion node and moving upwards
  - deletion involves fusing nodes (and is also $\lg N$)

2-3-4 Trees as Red-Black Trees

- red-black trees are a way of realizing 2-3-4 trees as binary search trees
  - allows us to re-use an implementation of a BST, and simplifies <u>deletion</u>
  - add the attribute of <u>color</u> (red or black) to links/nodes

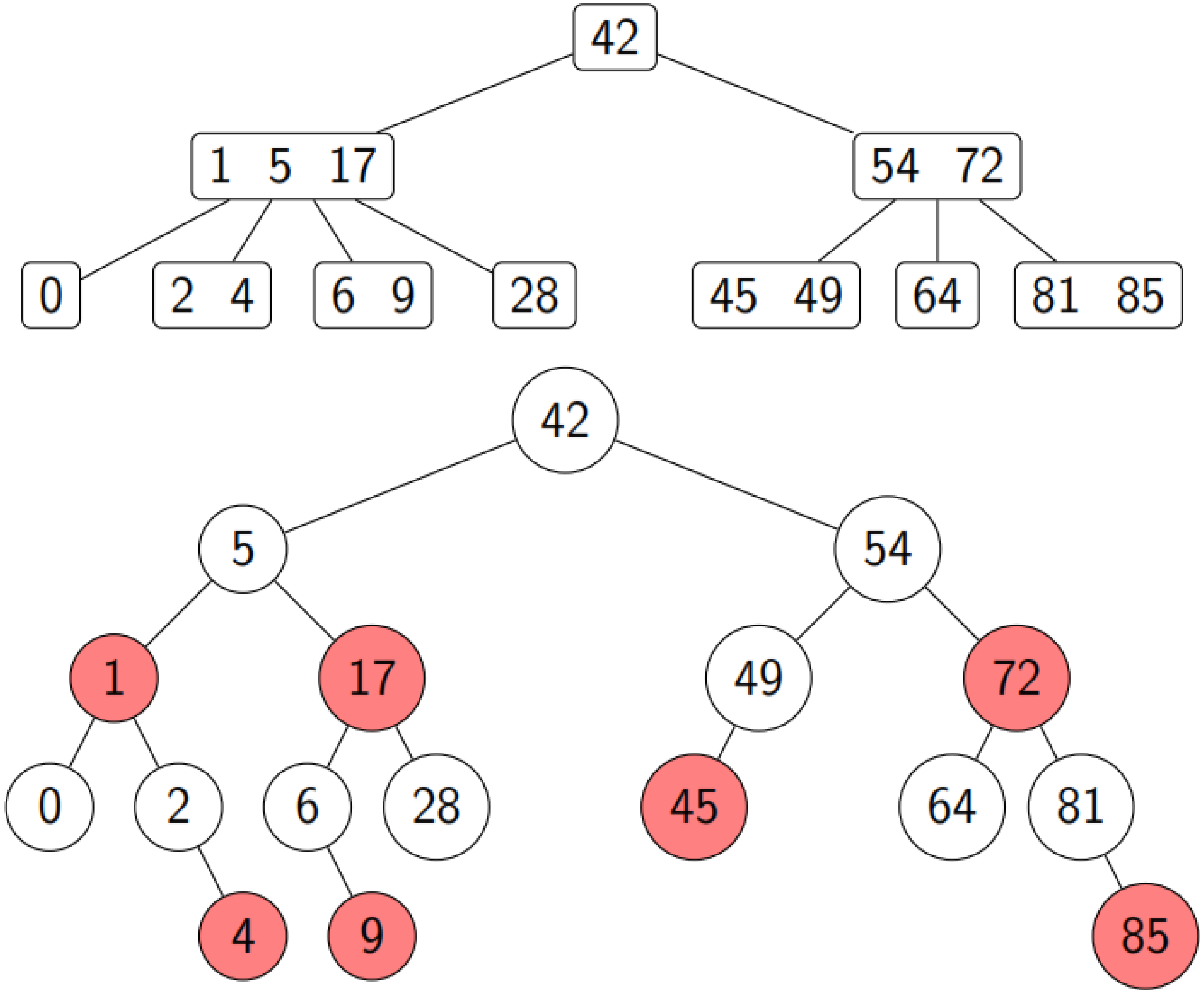
2-3-4 Trees as Red-Black Trees

– encoding 2-3-4 trees as red-black trees



2-3-4 Trees as Red-Black Trees

- encoding 2-3-4 trees as red-black trees (red = group with parent)



2-3-4 Trees as Red-Black Trees

- a red-black tree is a BST with the following properties:
  - every node is either red or black
  - the root is black
  - if a node is red, its children must be black
  - every path from the root to a null link contains the same number of black nodes
- in the encoding of 2-3-4 trees from red-black trees, the <u>black</u> links in the red-black tree correspond to the links in the 2-3-4 tree, while the <u>red</u> links denote a split of a 2-node or 3-node
- condition 4 corresponds to the perfect balance of 2-3-4 trees

- the height of an N-node red-black BST is at most $2\lg(N+1)$, so search, insertion, and deletion are $\lg N$ operations

2-3-4 Trees as Red-Black Trees

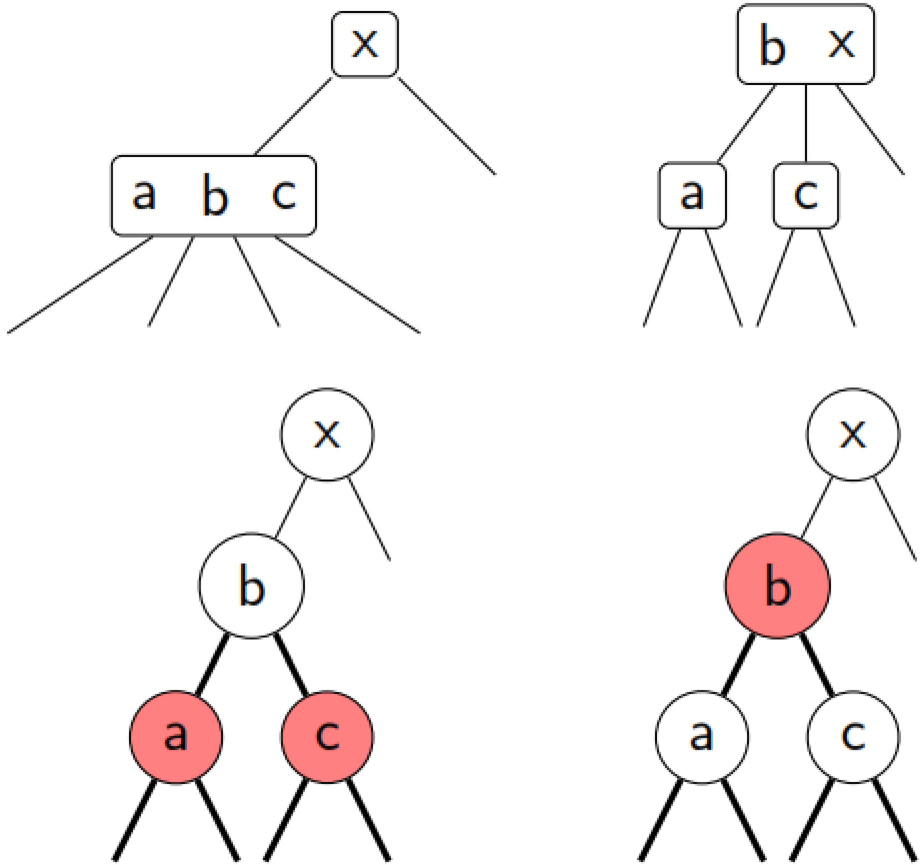
- building a red-black tree
  - in order to maintain perfect black balance, any new node added to the tree must be red
  - if the parent of the new node is black, all is well
  - if the parent of the new node is red, this violates the condition that red nodes have only black children
    - fix with rotations similar to those for AVL and splay trees
    - can be used to maintain the red-black structure at any point in the tree, not just at insertion of a new node

2-3-4 Trees as Red-Black Trees

- top-down insertion: color flips
  - we will follow a top-down insertion scheme as we did with 2-3-4 trees
  - as we move down the tree to insert a node, if we encounter a node X with two red children, we make X red and its children black
    - if X is the root, we change the color back to black
  - a color flip can cause a red-black violation (a red child with a red parent) only if X's parent P is red

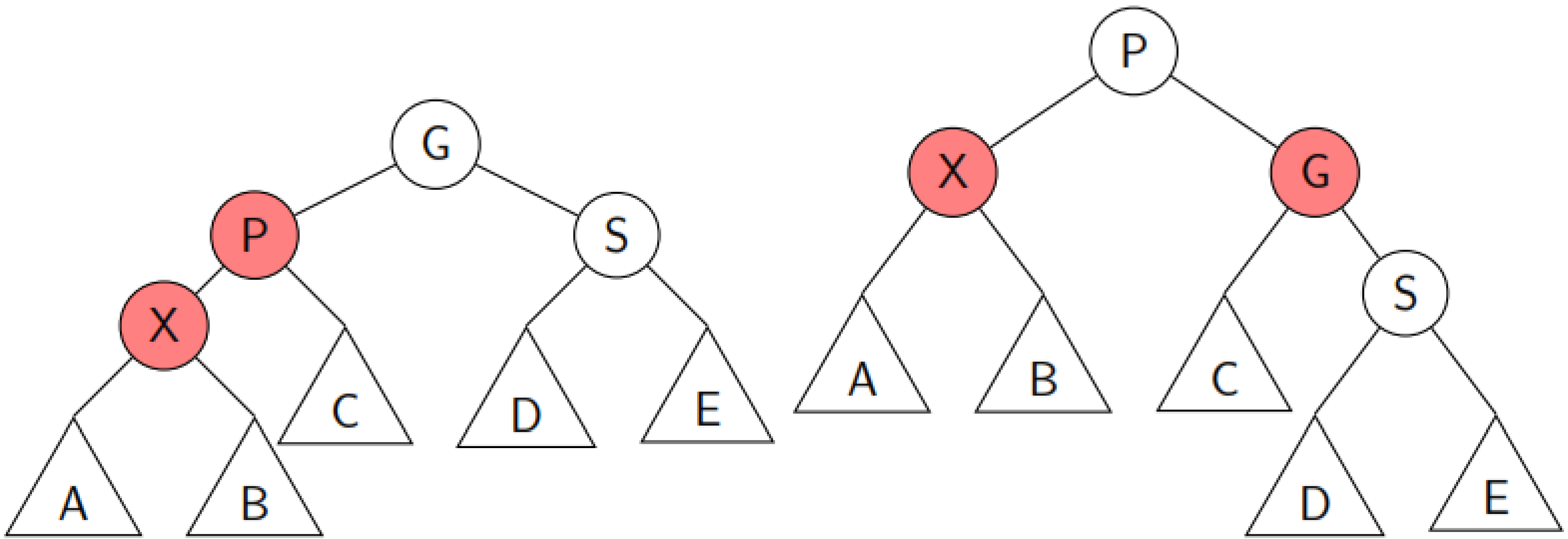
2-3-4 Trees as Red-Black Trees

- color flips correspond to splitting 4-nodes



2-3-4 Trees as Red-Black Trees

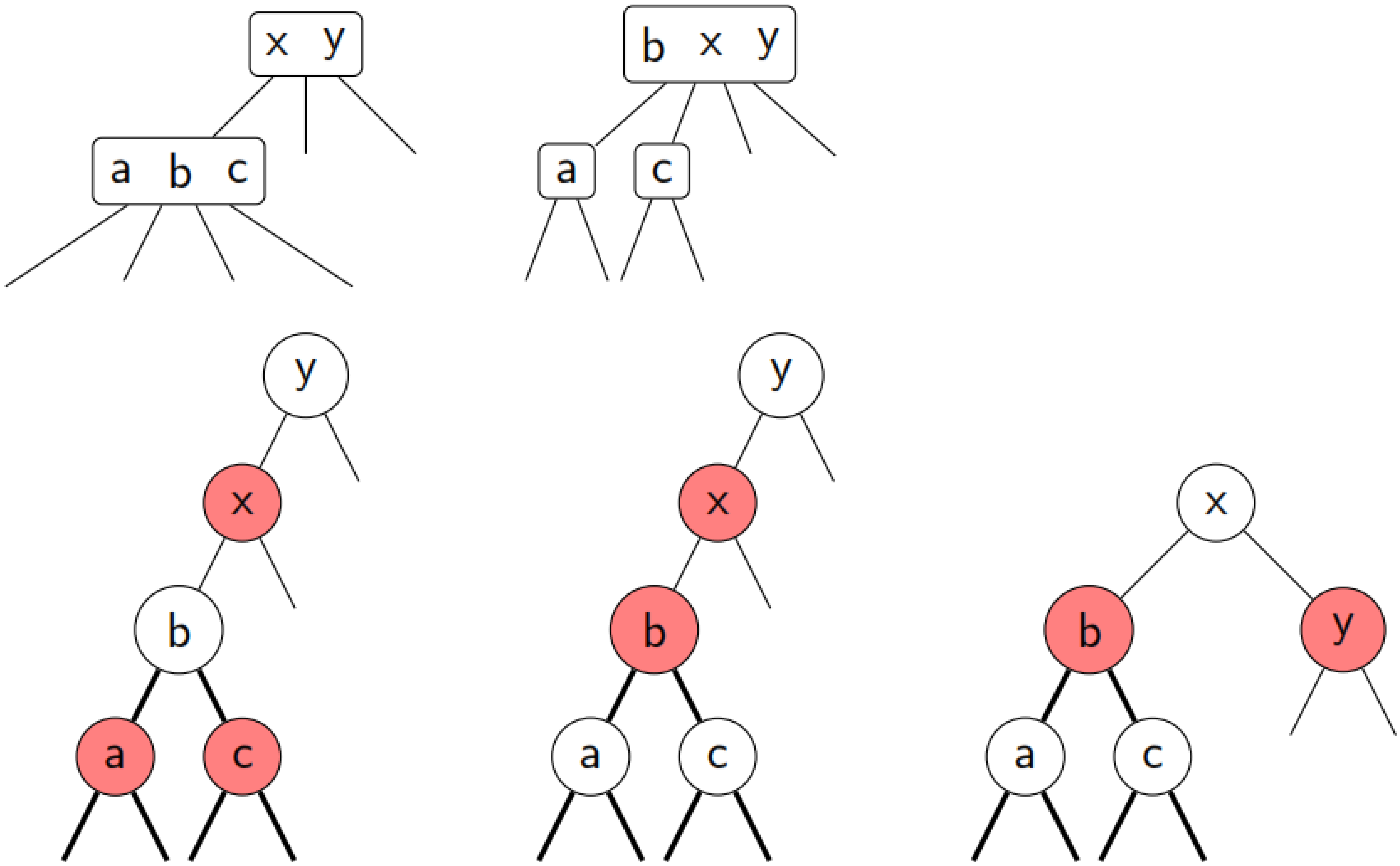
- red-black tree rotations
  - case 1: the parent is red and the parent's sibling is black (or missing)



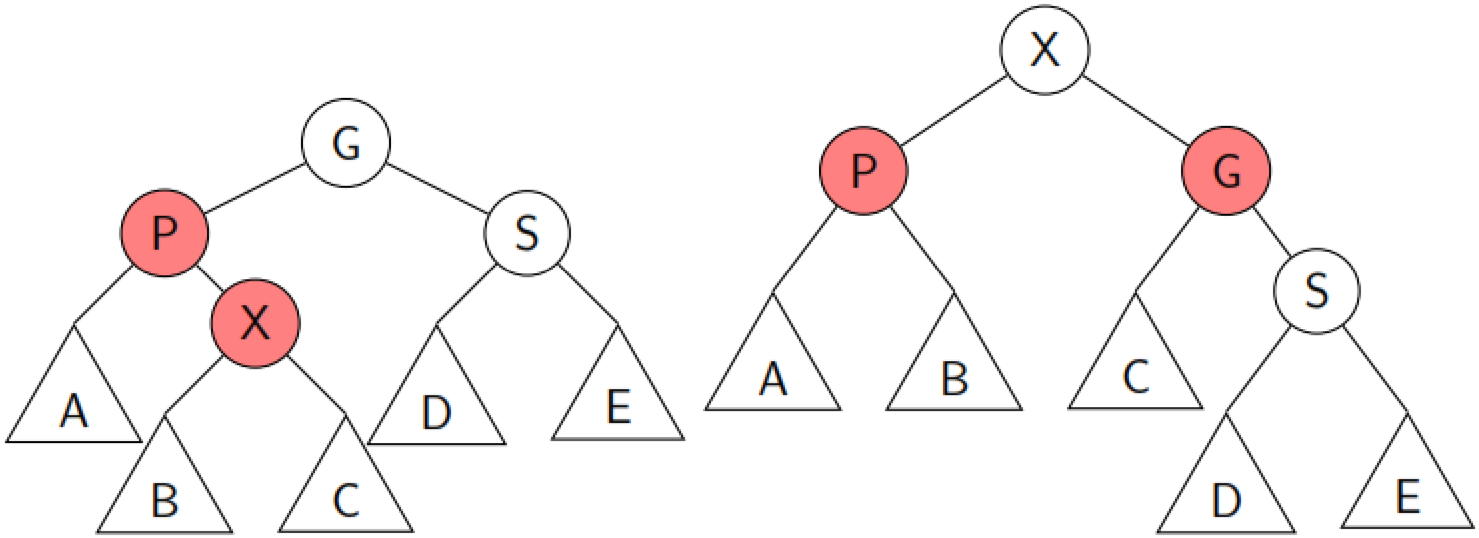
  - this is a single rotation and a color swap for P and G

2-3-4 Trees as Red-Black Trees

- rotations correspond to splitting 4-nodes



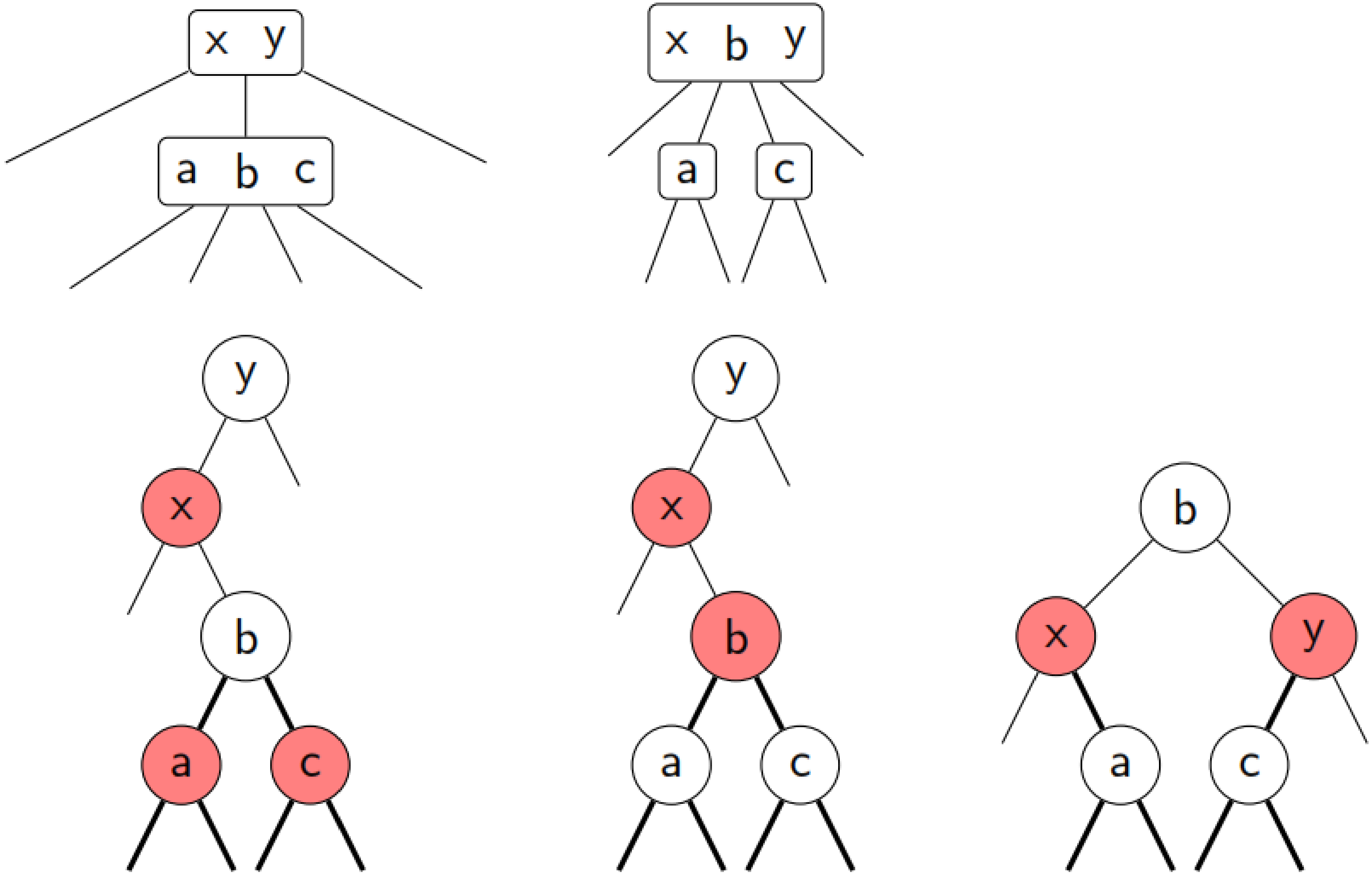
- red-black tree rotations
  - case 2: the parent is red and the parent's sibling is black (or missing)



  - this is a double rotation and a color swap for X and G

2-3-4 Trees as Red-Black Trees

- rotations correspond to splitting 4-nodes



2-3-4 Trees as Red-Black Trees

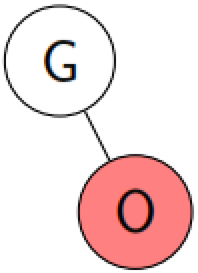
- red-black tree rotations
  - the parent is red and the parent's sibling is red
    - this can't happen since it would mean that the parent and its sibling are part of a <u>4-node</u>
      - we split all the 4-nodes we encountered on the way down

2-3-4 Trees as Red-Black Trees

– example: GOTCHA

Insert G:

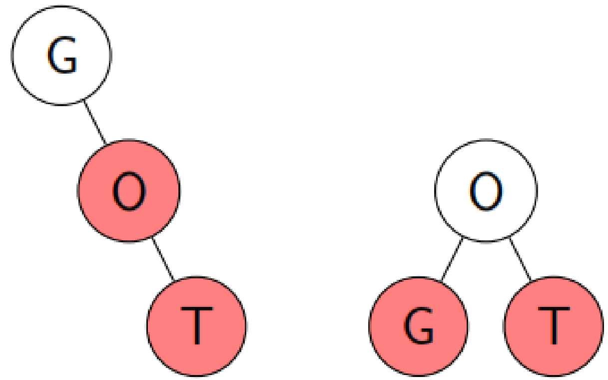
Insert O:



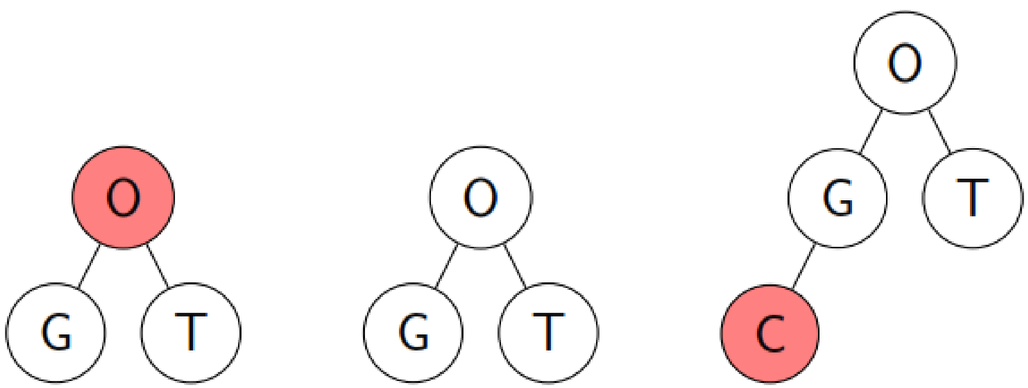
2-3-4 Trees as Red-Black Trees

– example: GOTCHA (cont.)

Insert T:



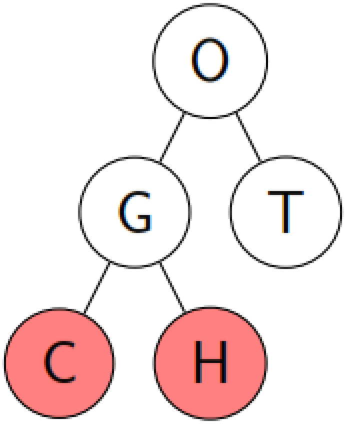
Insert C:



2-3-4 Trees as Red-Black Trees

– example: GOTCHA (cont.)

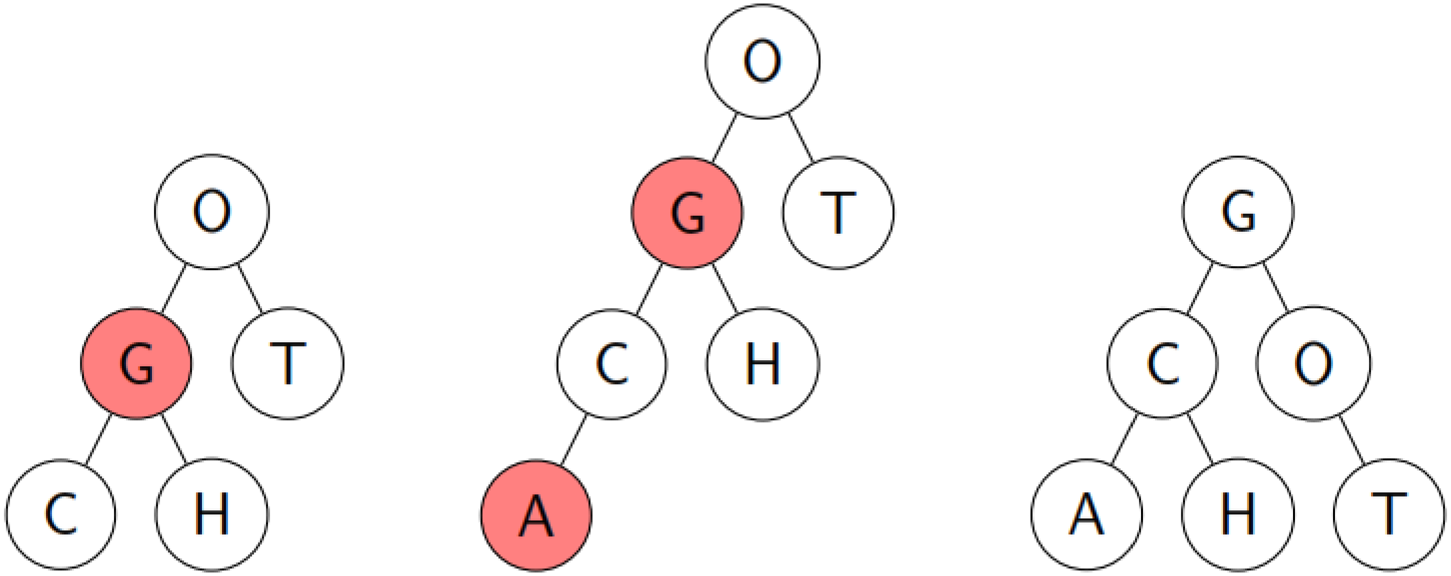
Insert H:



2-3-4 Trees as Red-Black Trees

- example: GOTCHA (cont.)

Insert A:



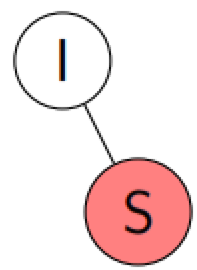
- the standard BST (rightmost) for GOTCHA is slightly shorter

2-3-4 Trees as Red-Black Trees

– example: ISOGRAM

Insert I:

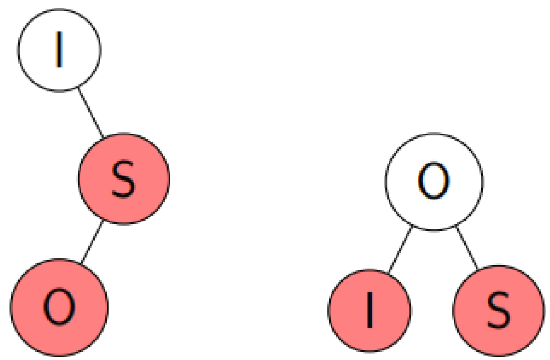
Insert S:



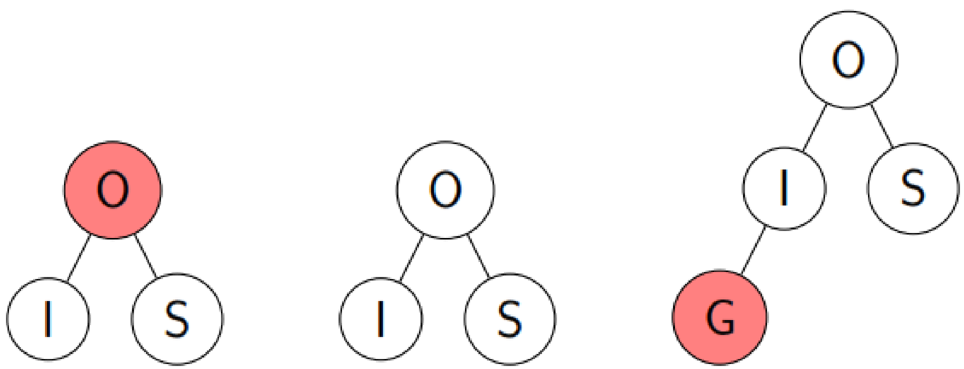
2-3-4 Trees as Red-Black Trees

– example: ISOGRAM

Insert O:



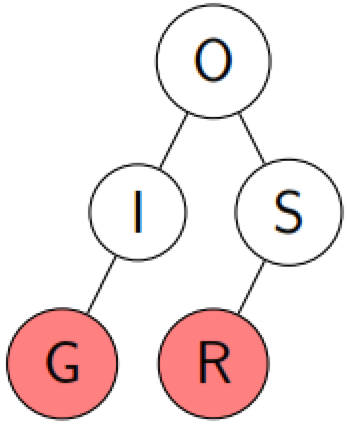
Insert G:



2-3-4 Trees as Red-Black Trees

- example: ISOGRAM

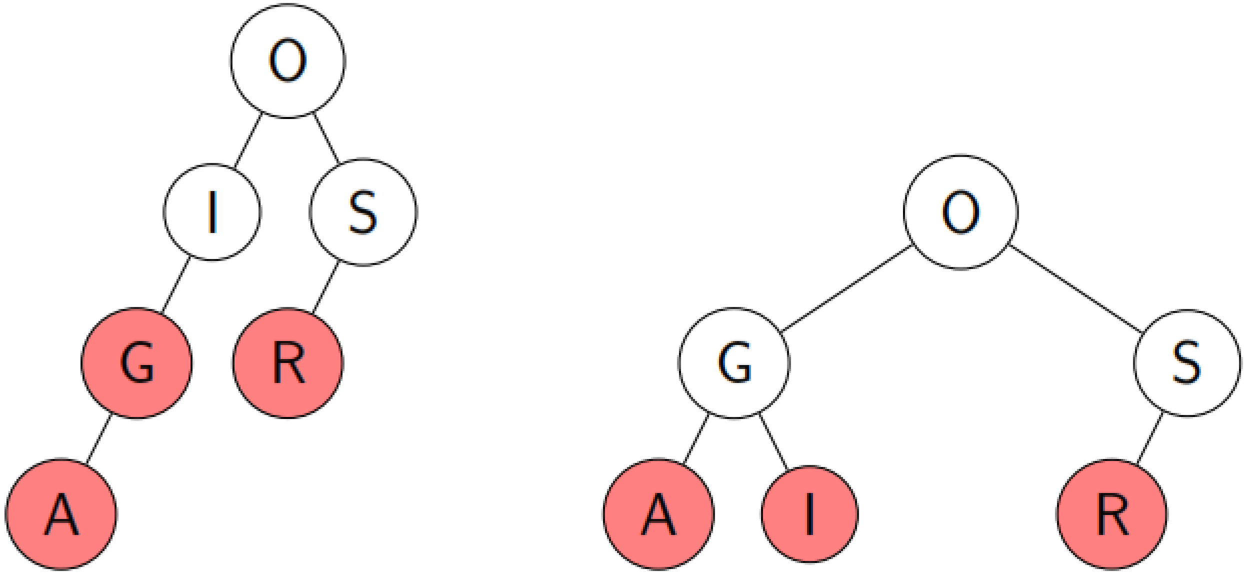
  Insert R:



2-3-4 Trees as Red-Black Trees

- example: ISOGRAM

Insert A:



2-3-4 Trees as Red-Black Trees

– example: ISOGRAM

Insert M: