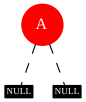
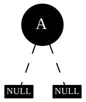
Insertion of 'A'

Insertion of 'A' - Tree empty, inserting as root

Insertion of 'A' - Fixup of root element: paint black

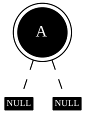


Insertion of 'A' - Outcome

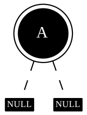


Insertion of 'L'

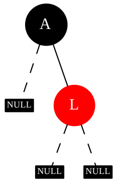
Insertion of 'L' - Finding the right position



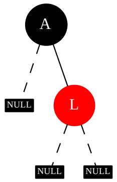
Insertion of 'L' - Inserting as right child of 'A' node



Insertion of 'L' - No property violated, no fixup needed

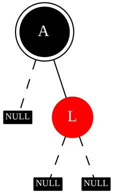


Insertion of 'L' - Outcome

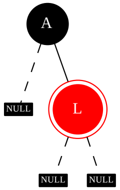


Insertion of 'G'

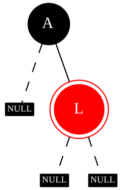
Insertion of 'G' - Finding the right position



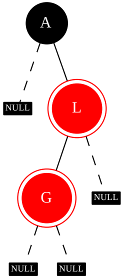
Insertion of 'G' - Finding the right position



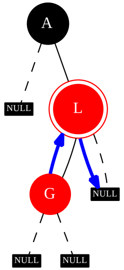
Insertion of 'G' - Inserting as left child of 'L' node



Insertion of 'G' - Property 4 violated, fixup of 'G' needed

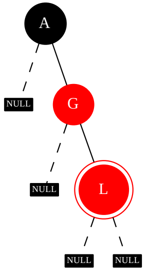


Fixup of 'G' - Case 2



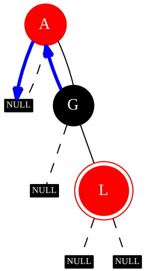
Case 2: the parent ('L', highlighted) is the right child of the grandparent, the uncle is black and the node being fixed up ('G') is the left child of the parent. Reduce to case 3 by right-rotating (blue arrows) the parent and then recursively fixing up the parent ('L').

Fixup of 'L' - Case 3



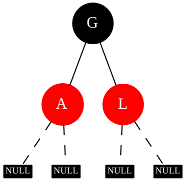
Case 3: the parent ('G') is the right child of the grandparent ('A'), the uncle is black and the node being fixed up ('L', highlighted) is the right child of the parent. Color the parent black, the grandparent red and left-rotate around the grandparent ('A').

Fixup of 'L' - Case 3

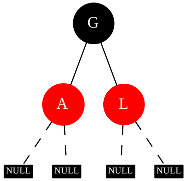


Case 3: the parent ('G') is the right child of the grandparent ('A'), the uncle is black and the node being fixed up ('L', highlighted) is the right child of the parent. Color the parent black, the grandparent red and left-rotate (blue arrows) around the grandparent ('A').

Fixup of 'G' - Done

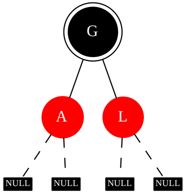


Insertion of 'G' - Outcome

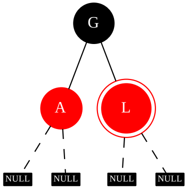


Insertion of 'O'

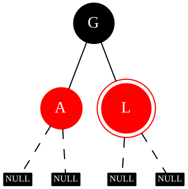
Insertion of 'O' - Finding the right position



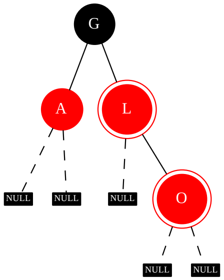
Insertion of 'O' - Finding the right position



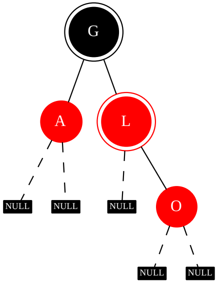
Insertion of ’O’ - Inserting as right child of ’L’ node



Insertion of 'O' - Property 4 violated, fixup of 'O' needed

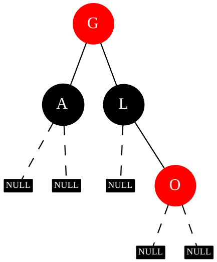


Fixup of 'O' - Case 1

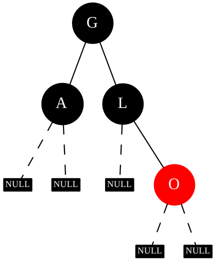


Case 1: the parent ('L', highlighted) is the right child of the grandparent ('G', also highlighted) and the uncle is red. Paint the parent and the uncle black and paint the grandparent red, then recursively fixup the grandparent ('G').

Insertion of 'O' - Fixup of root element: paint black

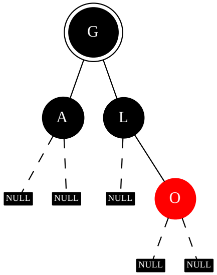


Insertion of 'O' - Outcome

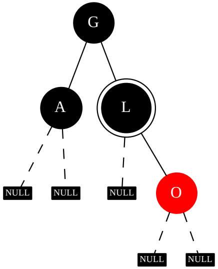


Insertion of ’R’

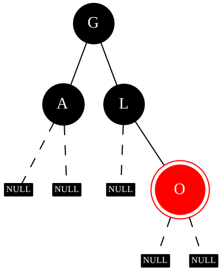
Insertion of 'R' - Finding the right position



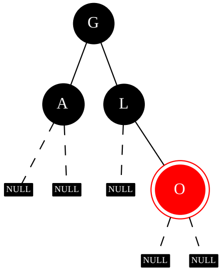
Insertion of 'R' - Finding the right position



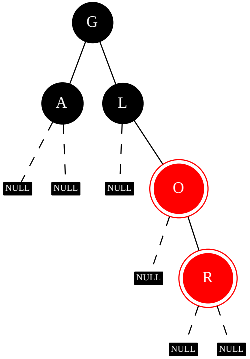
Insertion of 'R' - Finding the right position



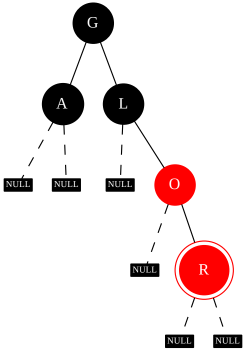
Insertion of 'R' - Inserting as right child of 'O' node



Insertion of 'R' - Property 4 violated, fixup of 'R' needed

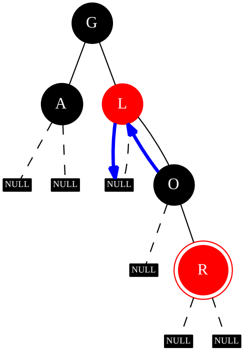


Fixup of 'R' - Case 3



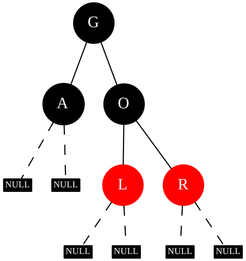
Case 3: the parent ('O') is the right child of the grandparent ('L'), the uncle is black and the node being fixed up ('R', highlighted) is the right child of the parent. Color the parent black, the grandparent red and left-rotate around the grandparent ('L').

Fixup of 'R' - Case 3

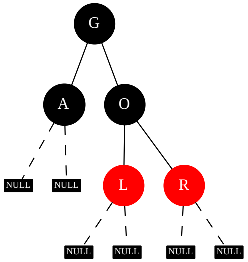


Case 3: the parent ('O') is the right child of the grandparent ('L'), the uncle is black and the node being fixed up ('R', highlighted) is the right child of the parent. Color the parent black, the grandparent red and left-rotate (blue arrows) around the grandparent ('L').

Fixup of 'R' - Done

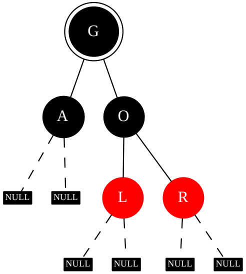


Insertion of 'R' - Outcome

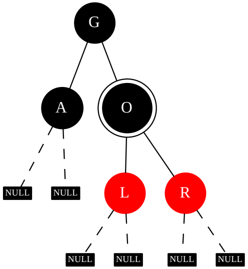


Insertion of 'I'

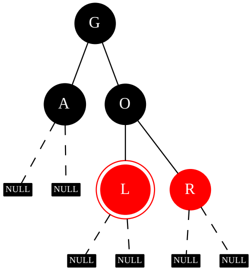
Insertion of 'I' - Finding the right position



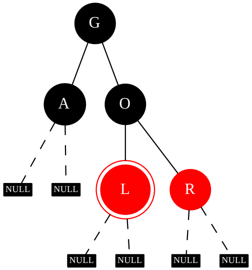
Insertion of 'I' - Finding the right position



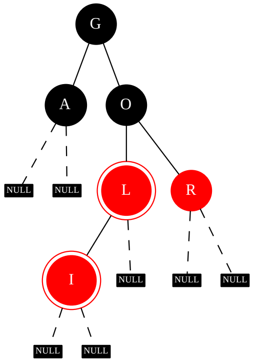
Insertion of 'I' - Finding the right position



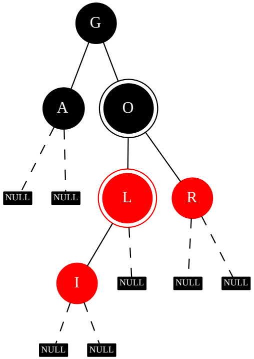
Insertion of 'I' - Inserting as left child of 'L' node



Insertion of ’I’ - Property 4 violated, fixup of ’I’ needed

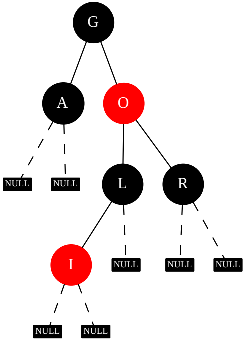


Fixup of 'I' - Case 1

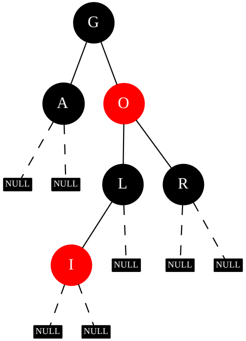


Case 1: the parent ('L', highlighted) is the left child of the grandparent ('O', also highlighted) and the uncle is red. Paint the parent and the uncle black and paint the grandparent red, then recursively fixup the grandparent ('O').

Fixup of 'I' - Done

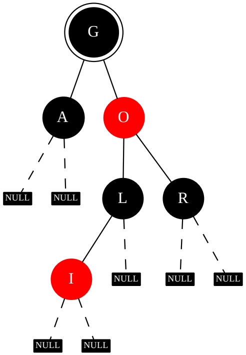


Insertion of 'I' - Outcome

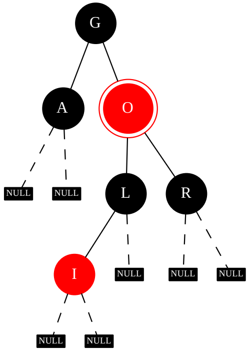


Insertion of 'T'

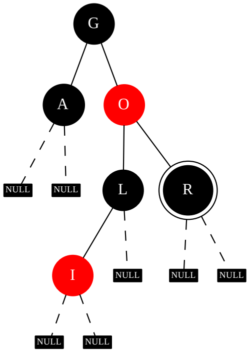
Insertion of 'T' - Finding the right position



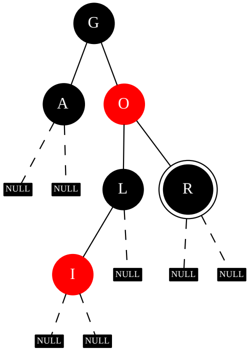
Insertion of 'T' - Finding the right position



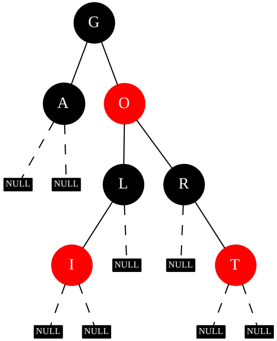
Insertion of 'T' - Finding the right position



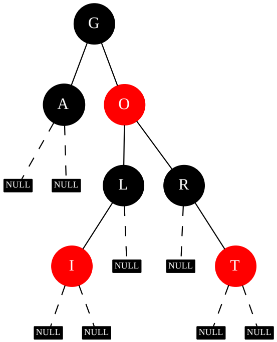
Insertion of 'T' - Inserting as right child of 'R' node



Insertion of 'T' - No property violated, no fixup needed

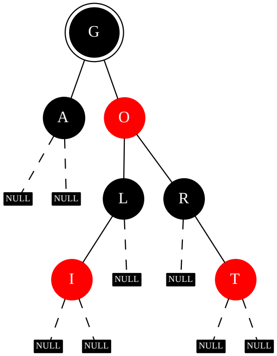


Insertion of 'T' - Outcome

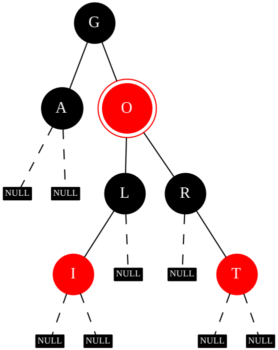


Insertion of 'H'

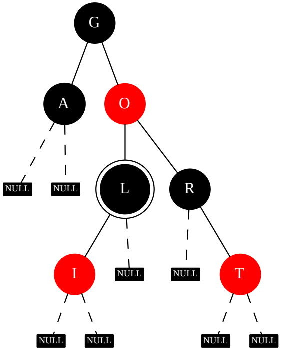
Insertion of 'H' - Finding the right position



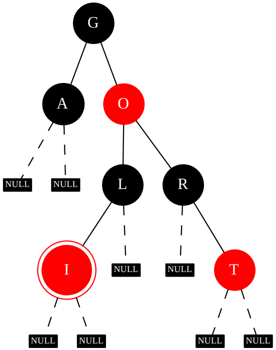
Insertion of 'H' - Finding the right position



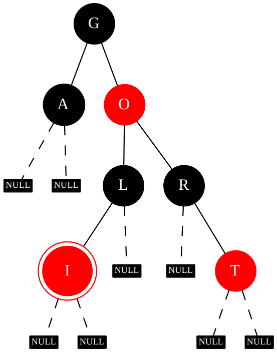
Insertion of 'H' - Finding the right position



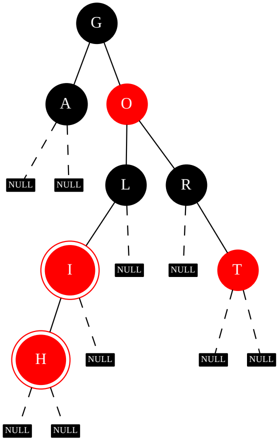
Insertion of 'H' - Finding the right position



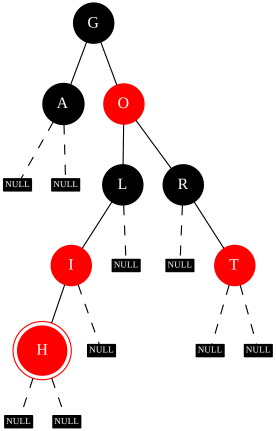
Insertion of 'H' - Inserting as left child of 'I' node



Insertion of 'H' - Property 4 violated, fixup of 'H' needed

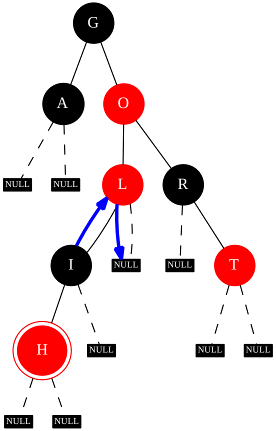


Fixup of 'H' - Case 3



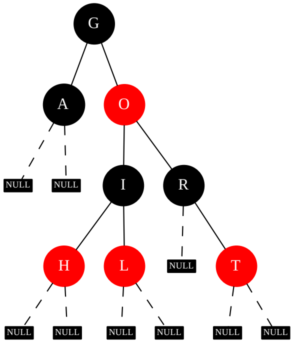
Case 3: the parent ('I') is the left child of the grandparent ('L'), the uncle is black and the node being fixed up ('H', highlighted) is the left child of the parent. Color the parent black, the grandparent red and right-rotate around the grandparent ('L').

Fixup of 'H' - Case 3

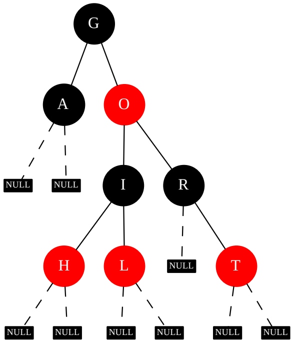


Case 3: the parent ('I') is the left child of the grandparent ('L'), the uncle is black and the node being fixed up ('H', highlighted) is the left child of the parent. Color the parent black, the grandparent red and right-rotate (blue arrows) around the grandparent ('L').

Fixup of 'H' - Done

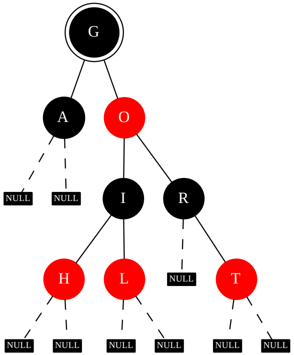


Insertion of 'H' - Outcome

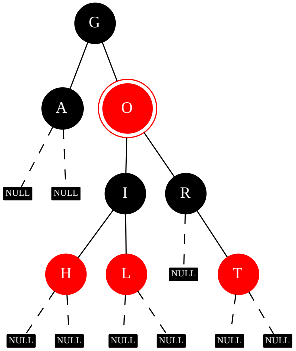


Insertion of 'M'

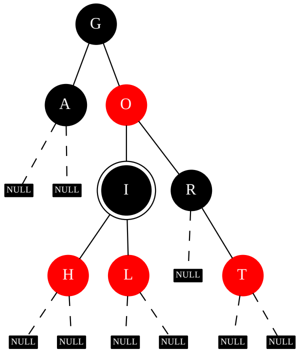
Insertion of 'M' - Finding the right position



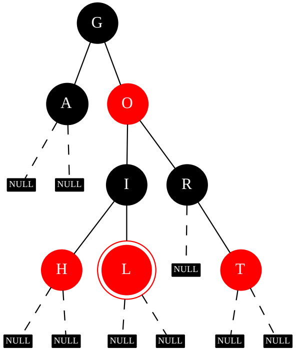
Insertion of 'M' - Finding the right position



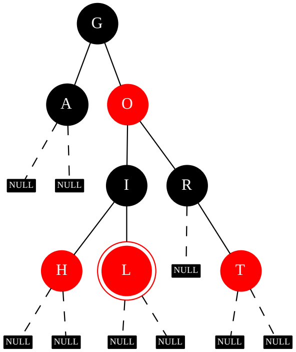
Insertion of 'M' - Finding the right position



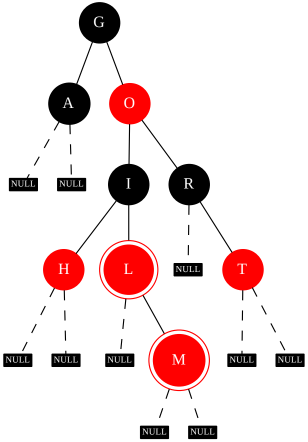
Insertion of ’M’ - Finding the right position



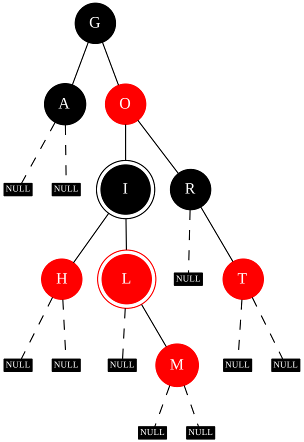
Insertion of ’M’ - Inserting as right child of ’L’ node



Insertion of 'M' - Property 4 violated, fixup of 'M' needed

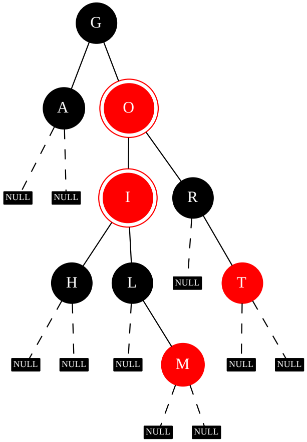


Fixup of 'M' - Case 1

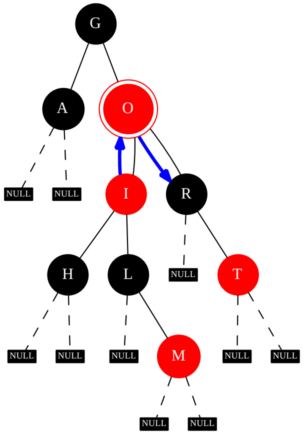


Case 1: the parent ('L', highlighted) is the right child of the grandparent ('I', also highlighted) and the uncle is red. Paint the parent and the uncle black and paint the grandparent red, then recursively fixup the grandparent ('I').

Insertion of 'M' - Property 4 violated, fixup of 'I' needed

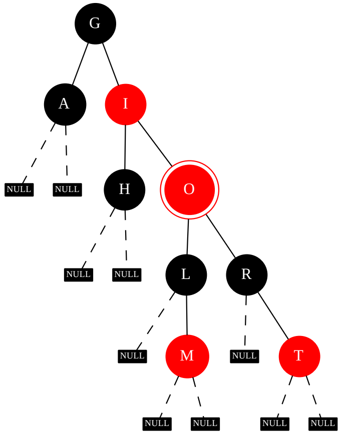


Fixup of 'I' - Case 2



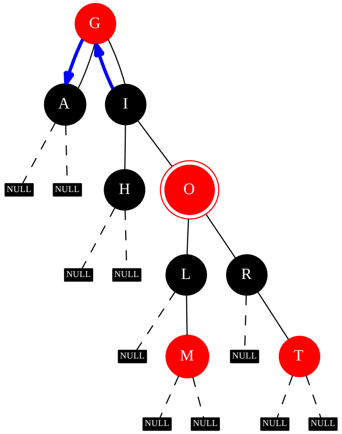
Case 2: the parent ('O', highlighted) is the right child of the grandparent, the uncle is black and the node being fixed up ('I') is the left child of the parent. Reduce to case 3 by right-rotating (blue arrows) the parent and then recursively fixing up the parent ('O').

Fixup of 'O' - Case 3



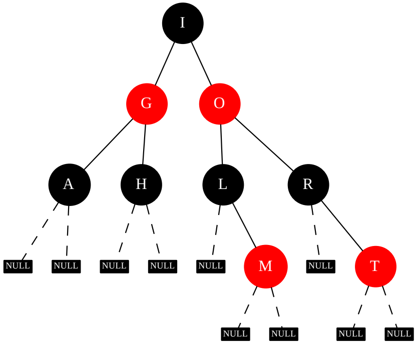
Case 3: the parent ('I') is the right child of the grandparent ('G'), the uncle is black and the node being fixed up ('O', highlighted) is the right child of the parent. Color the parent black, the grandparent red and left-rotate around the grandparent ('G').

Fixup of 'O' - Case 3

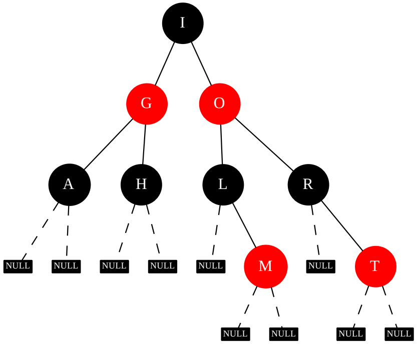


Case 3: the parent ('I') is the right child of the grandparent ('G'), the uncle is black and the node being fixed up ('O', highlighted) is the right child of the parent. Color the parent black, the grandparent red and left-rotate (blue arrows) around the grandparent ('G').

Fixup of 'M' - Done

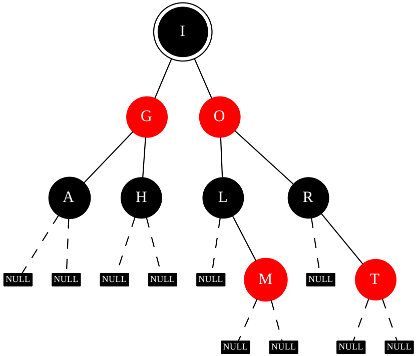


Insertion of 'M' - Outcome

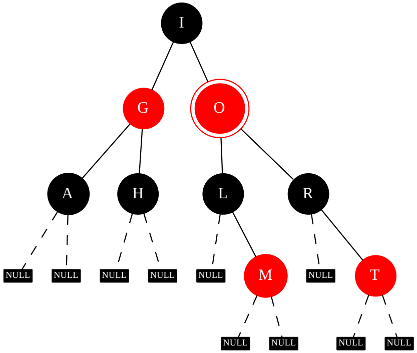


Insertion of ’S’

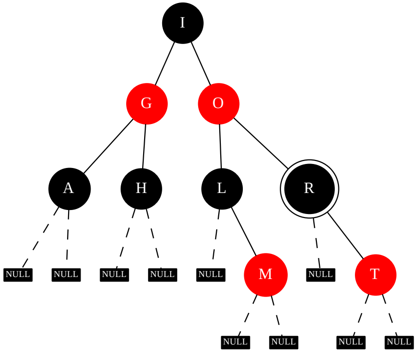
Insertion of 'S' - Finding the right position



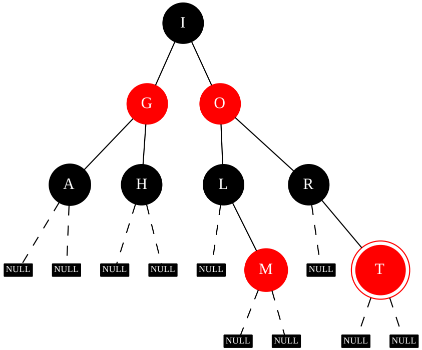
Insertion of 'S' - Finding the right position



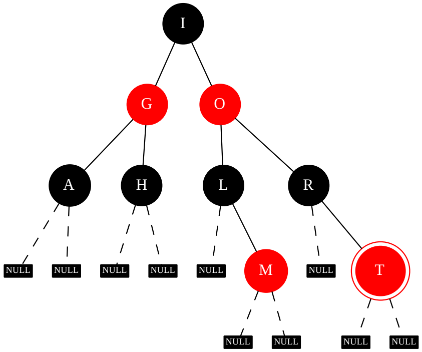
Insertion of 'S' - Finding the right position



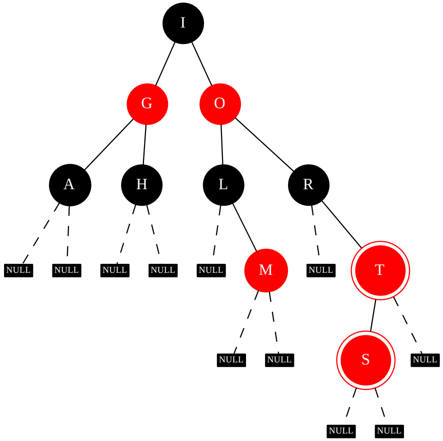
Insertion of 'S' - Finding the right position



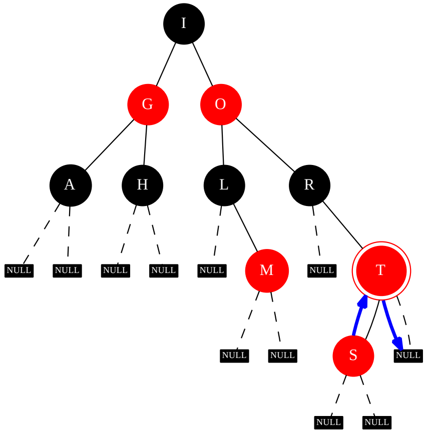
Insertion of 'S' - Inserting as left child of 'T' node



Insertion of 'S' - Property 4 violated, fixup of 'S' needed

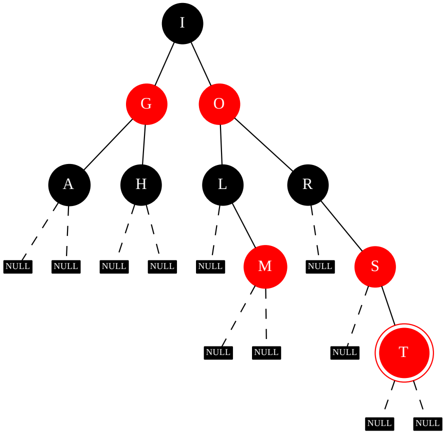


Fixup of 'S' - Case 2



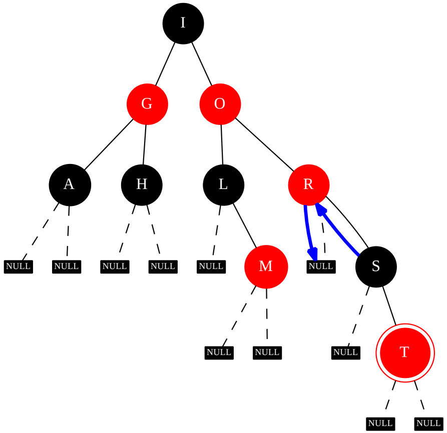
Case 2: the parent ('T', highlighted) is the right child of the grandparent, the uncle is black and the node being fixed up ('S') is the left child of the parent. Reduce to case 3 by right-rotating (blue arrows) the parent and then recursively fixing up the parent ('T').

Fixup of 'T' - Case 3



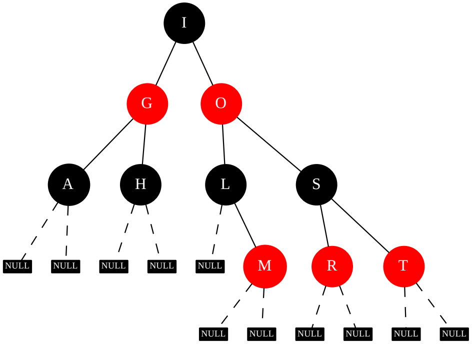
Case 3: the parent ('S') is the right child of the grandparent ('R'), the uncle is black and the node being fixed up ('T', highlighted) is the right child of the parent. Color the parent black, the grandparent red and left-rotate around the grandparent ('R').

Fixup of 'T' - Case 3

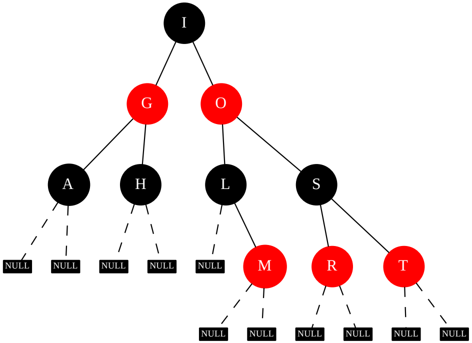


Case 3: the parent ('S') is the right child of the grandparent ('R'), the uncle is black and the node being fixed up ('T', highlighted) is the right child of the parent. Color the parent black, the grandparent red and left-rotate (blue arrows) around the grandparent ('R').

Fixup of 'S' - Done

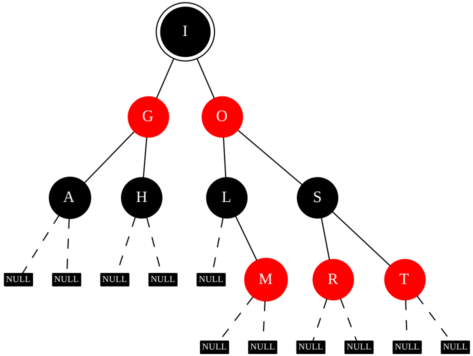


Insertion of 'S' - Outcome

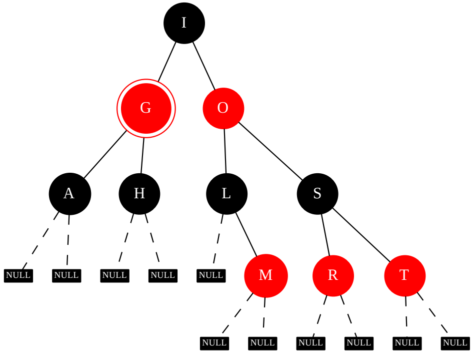


Insertion of 'C'

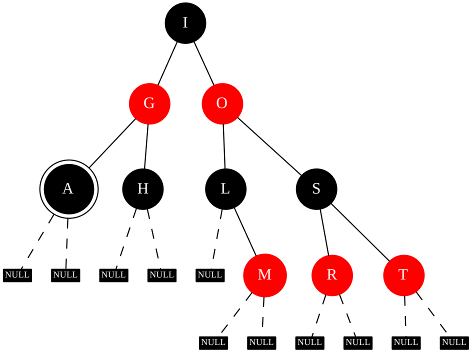
Insertion of 'C' - Finding the right position



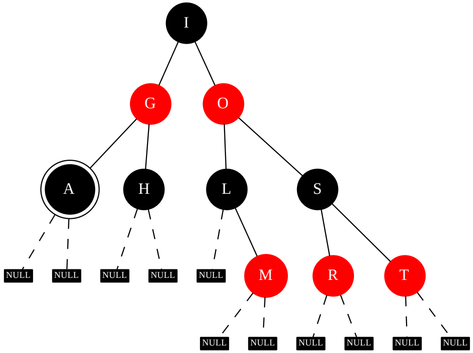
Insertion of 'C' - Finding the right position



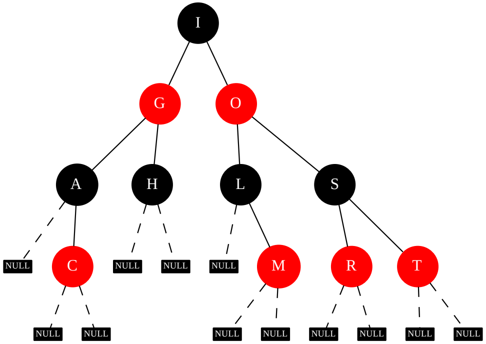
Insertion of 'C' - Finding the right position



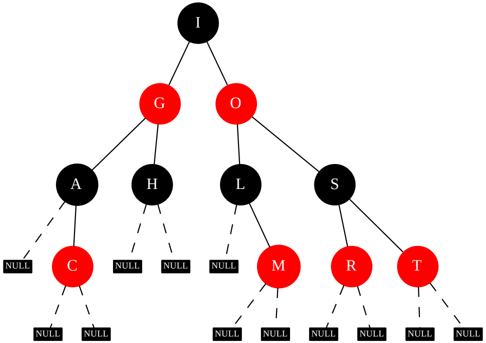
Insertion of 'C' - Inserting as right child of 'A' node



Insertion of 'C' - No property violated, no fixup needed

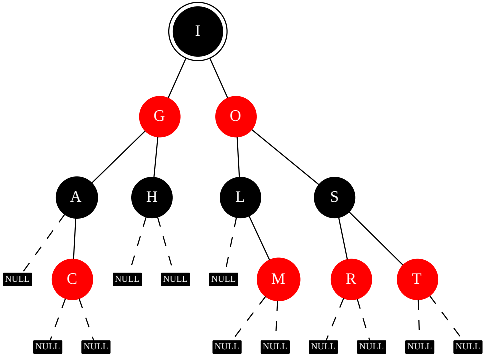


Insertion of 'C' - Outcome

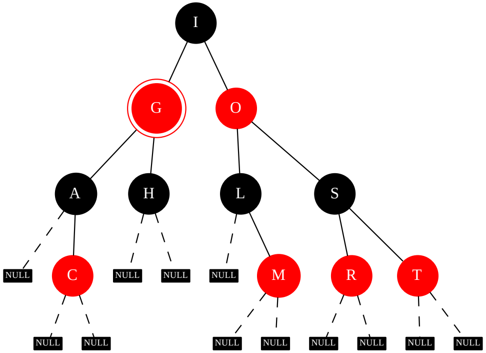


Insertion of ’A’

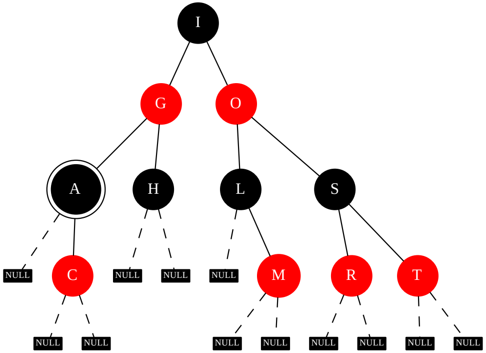
Insertion of 'A' - Finding the right position



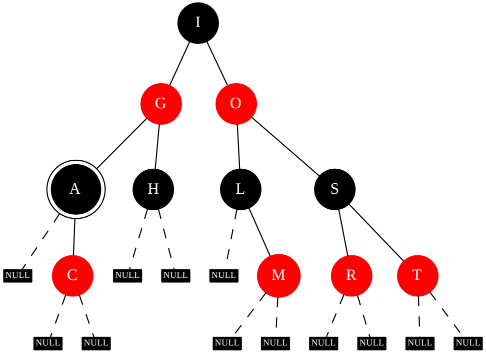
Insertion of 'A' - Finding the right position



Insertion of 'A' - Finding the right position

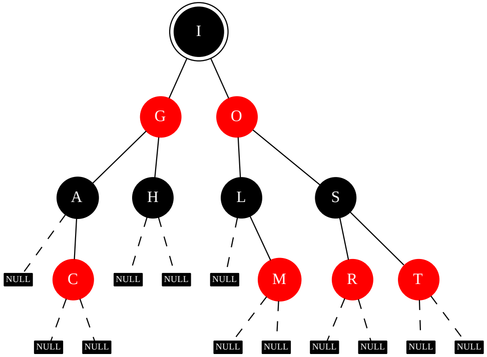


Insertion of 'A' - Key found, overwriting

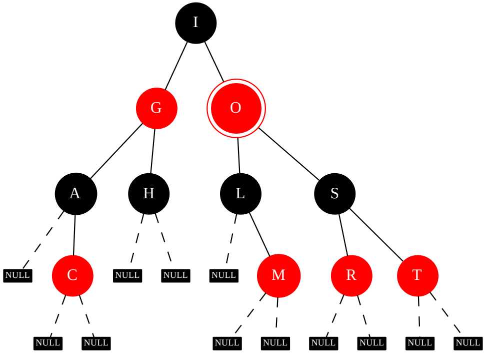


Insertion of 'M'

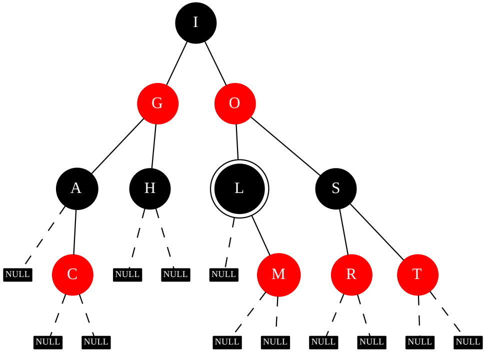
Insertion of 'M' - Finding the right position



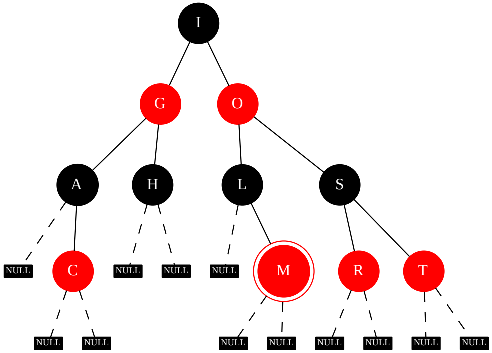
Insertion of 'M' - Finding the right position



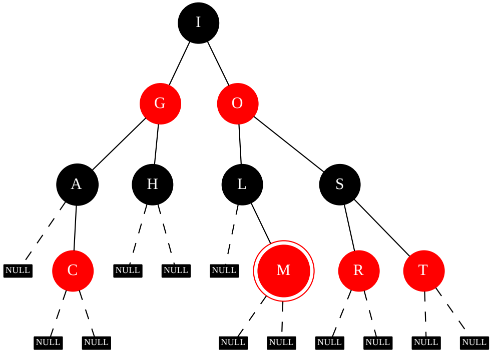
Insertion of 'M' - Finding the right position



Insertion of 'M' - Finding the right position

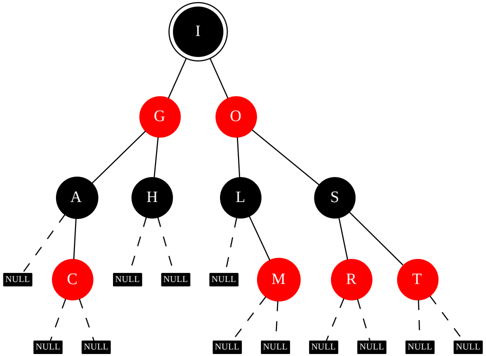


Insertion of 'M' - Key found, overwriting

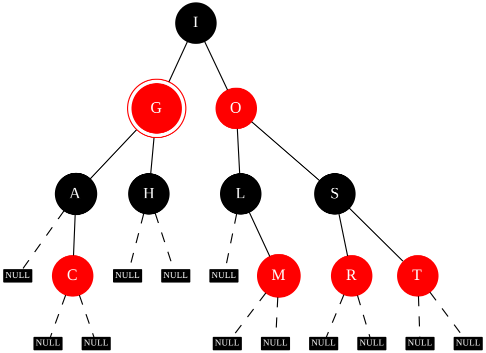


Insertion of 'B'

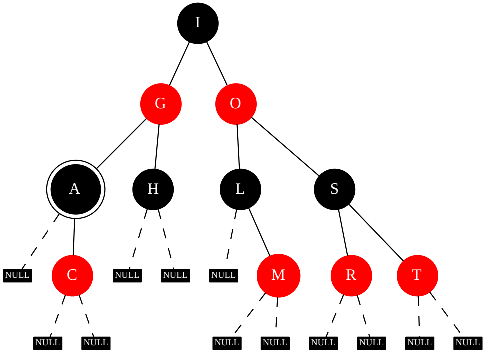
Insertion of 'B' - Finding the right position



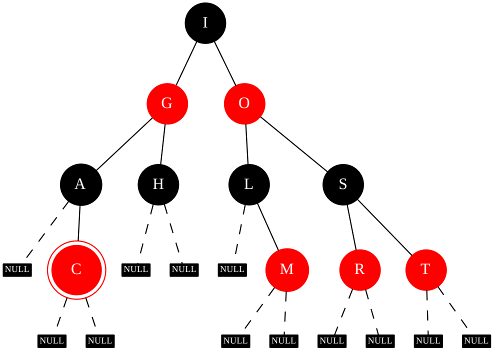
Insertion of 'B' - Finding the right position



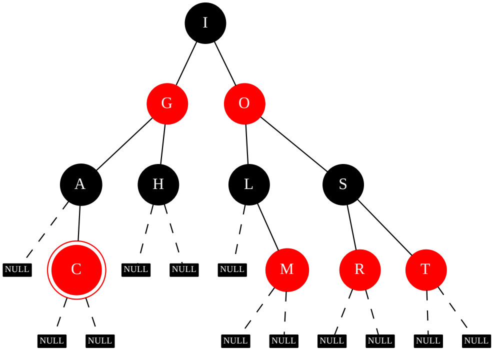
Insertion of 'B' - Finding the right position



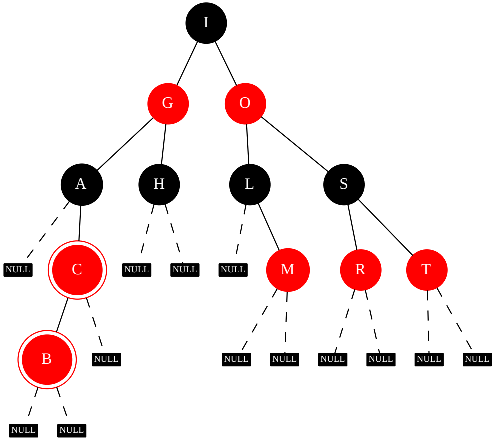
Insertion of 'B' - Finding the right position



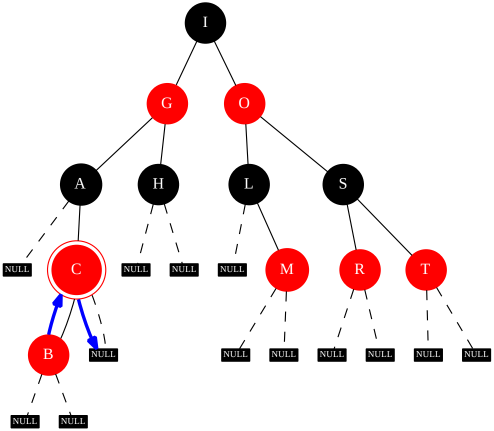
Insertion of 'B' - Inserting as left child of 'C' node



Insertion of 'B' - Property 4 violated, fixup of 'B' needed

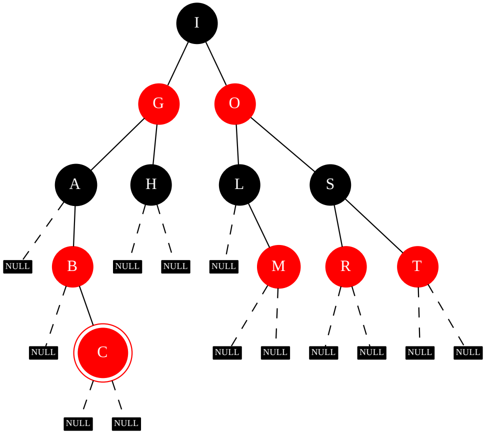


Fixup of 'B' - Case 2



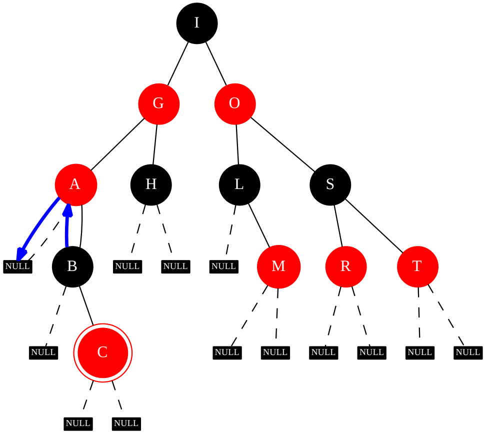
Case 2: the parent ('C', highlighted) is the right child of the grandparent, the uncle is black and the node being fixed up ('B') is the left child of the parent. Reduce to case 3 by right-rotating (blue arrows) the parent and then recursively fixing up the parent ('C').

Fixup of 'C' - Case 3



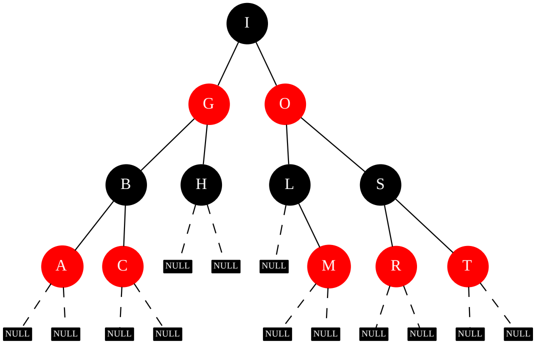
Case 3: the parent ('B') is the right child of the grandparent ('A'), the uncle is black and the node being fixed up ('C', highlighted) is the right child of the parent. Color the parent black, the grandparent red and left-rotate around the grandparent ('A').

Fixup of 'C' - Case 3

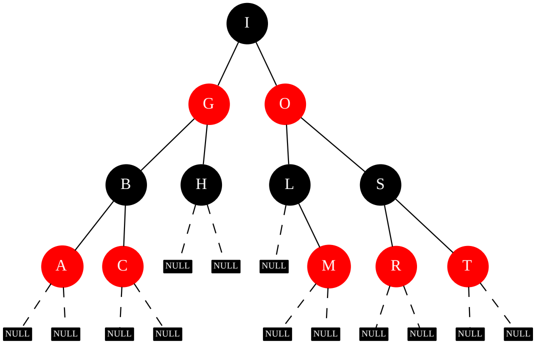


Case 3: the parent ('B') is the right child of the grandparent ('A'), the uncle is black and the node being fixed up ('C', highlighted) is the right child of the parent. Color the parent black, the grandparent red and left-rotate (blue arrows) around the grandparent ('A').

Fixup of 'B' - Done

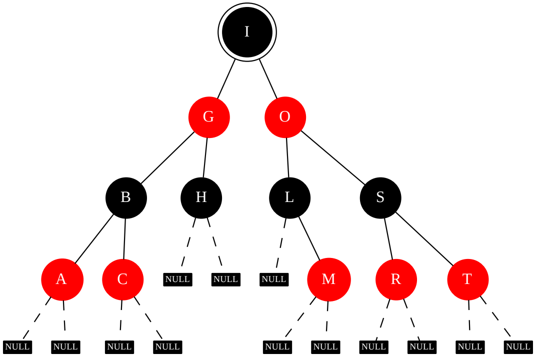


Insertion of 'B' - Outcome

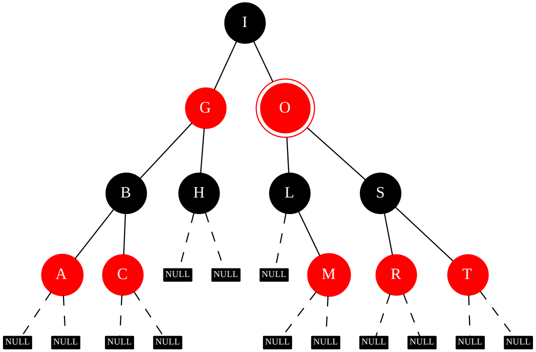


Insertion of 'R'

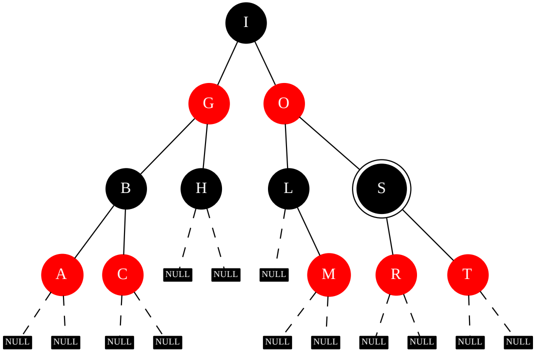
Insertion of 'R' - Finding the right position



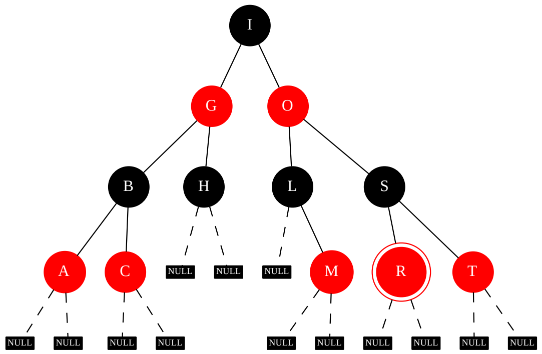
Insertion of 'R' - Finding the right position



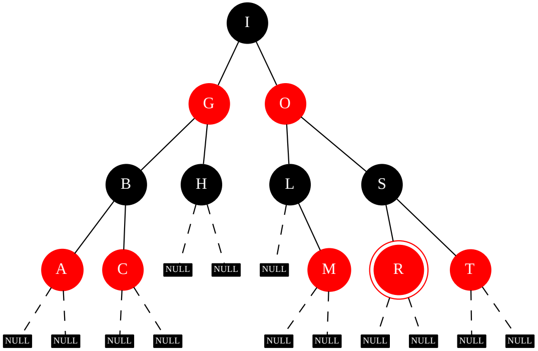
Insertion of 'R' - Finding the right position



Insertion of 'R' - Finding the right position

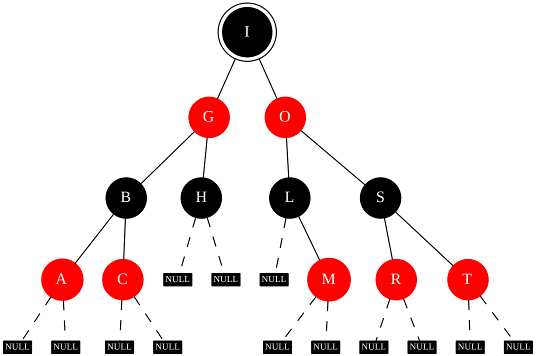


Insertion of 'R' - Key found, overwriting

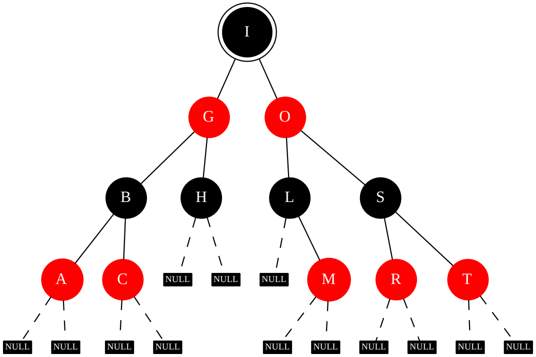


Insertion of 'I'

Insertion of 'I' - Finding the right position

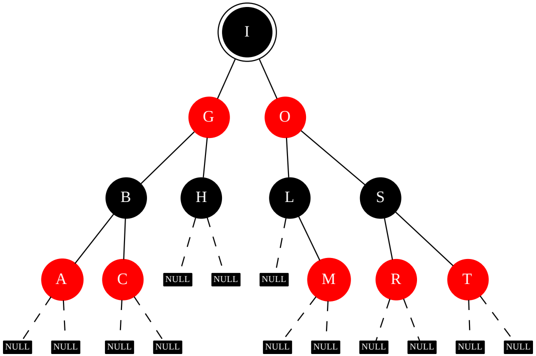


Insertion of 'I' - Key found, overwriting

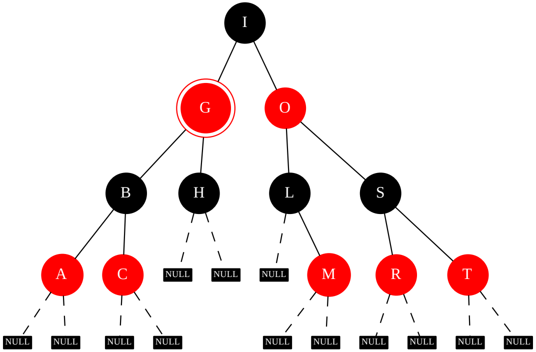


Insertion of 'D'

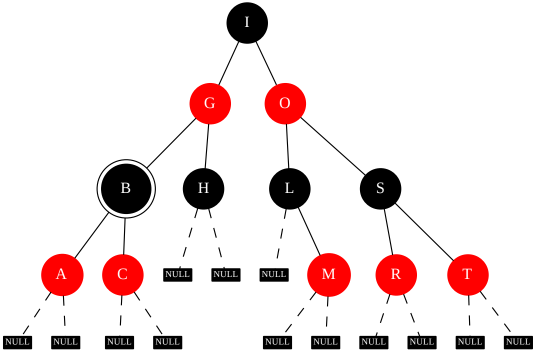
Insertion of 'D' - Finding the right position



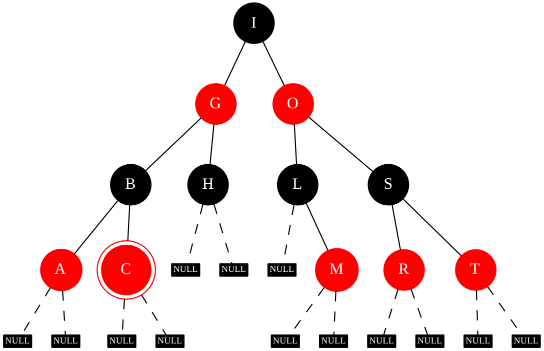
Insertion of 'D' - Finding the right position



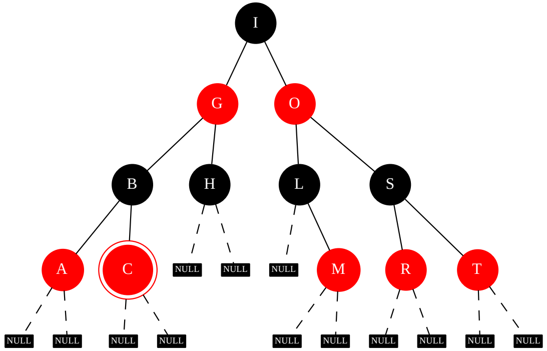
Insertion of 'D' - Finding the right position



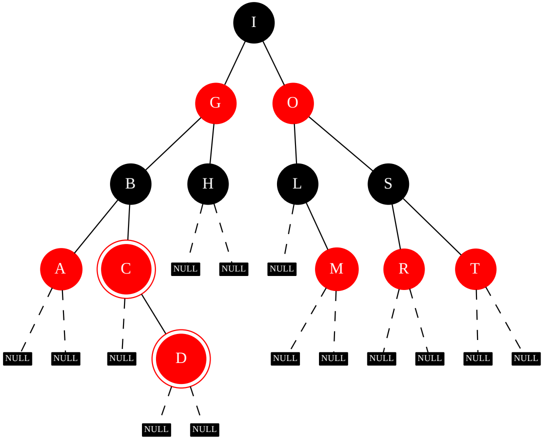
Insertion of 'D' - Finding the right position



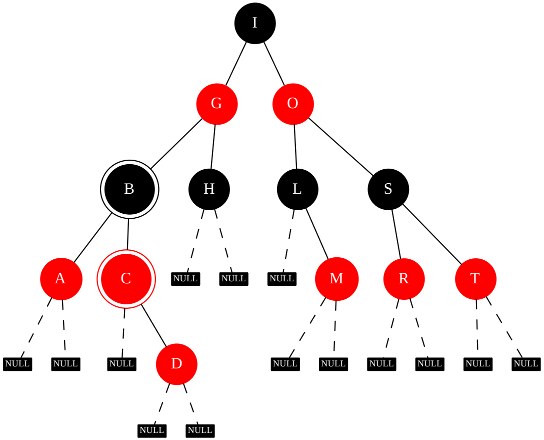
Insertion of 'D' - Inserting as right child of 'C' node



Insertion of 'D' - Property 4 violated, fixup of 'D' needed

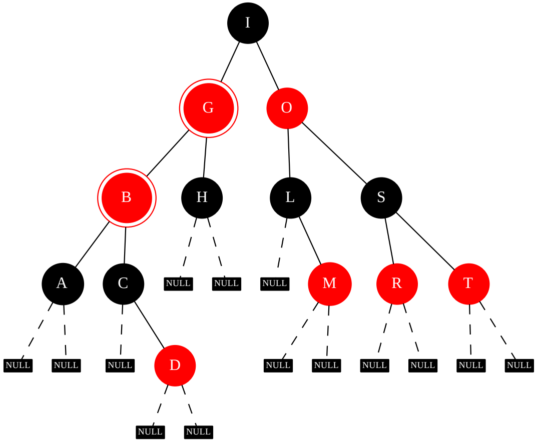


Fixup of 'D' - Case 1

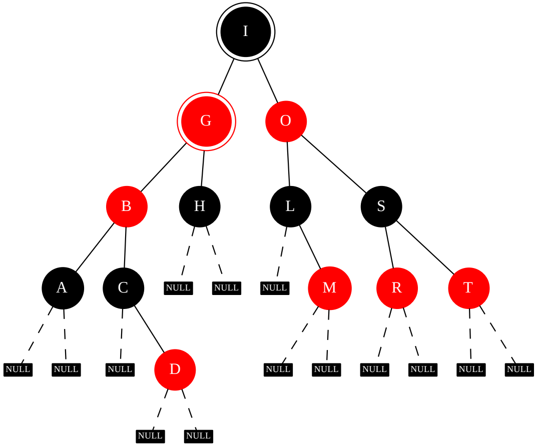


Case 1: the parent ('C', highlighted) is the right child of the grandparent ('B', also highlighted) and the uncle is red. Paint the parent and the uncle black and paint the grandparent red, then recursively fixup the grandparent ('B').

Insertion of 'D' - Property 4 violated, fixup of 'B' needed

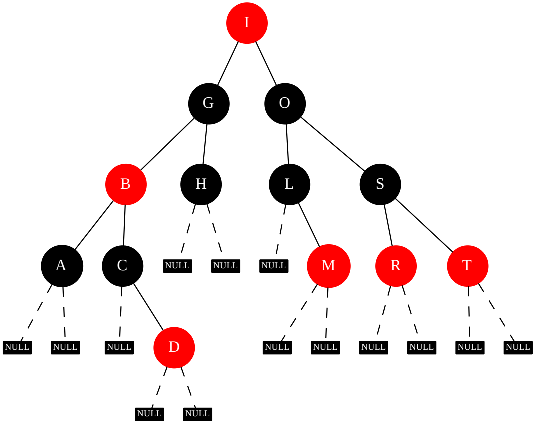


Fixup of 'B' - Case 1

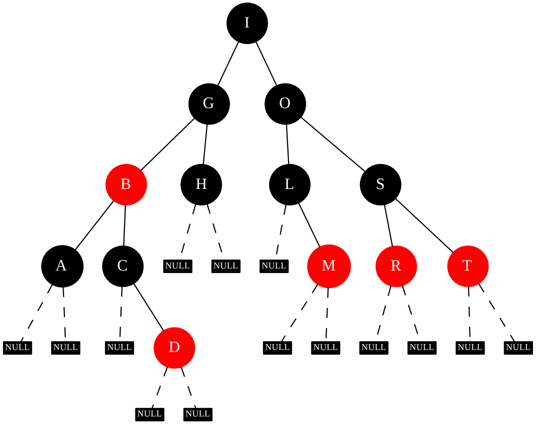


Case 1: the parent ('G', highlighted) is the left child of the grandparent ('I', also highlighted) and the uncle is red. Paint the parent and the uncle black and paint the grandparent red, then recursively fixup the grandparent ('I').

Insertion of 'D' - Fixup of root element: paint black

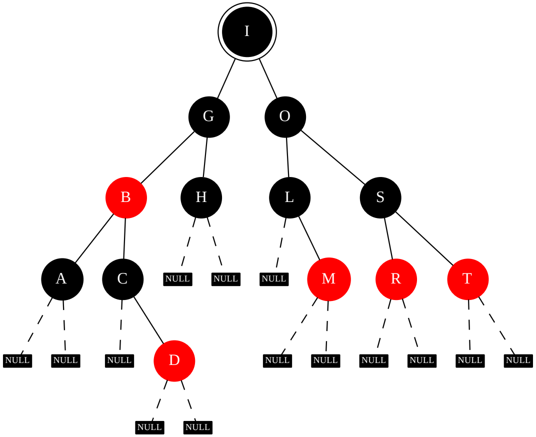


Insertion of 'D' - Outcome

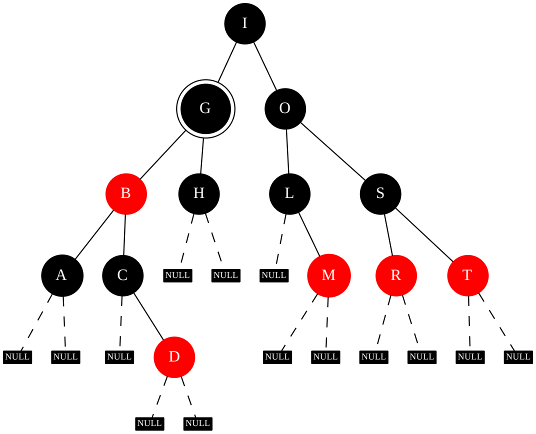


Insertion of 'G'

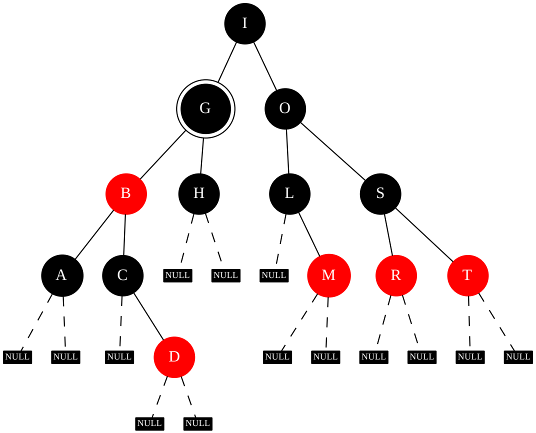
Insertion of 'G' - Finding the right position



Insertion of 'G' - Finding the right position

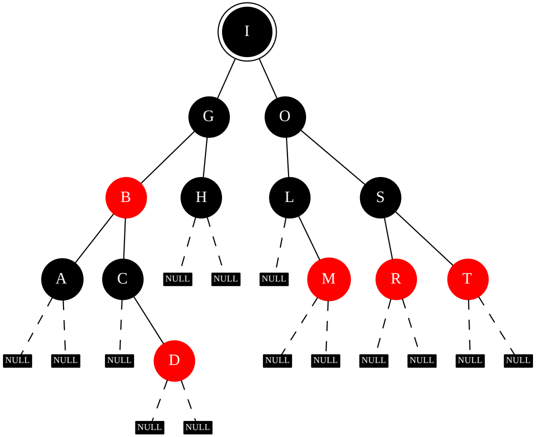


Insertion of 'G' - Key found, overwriting

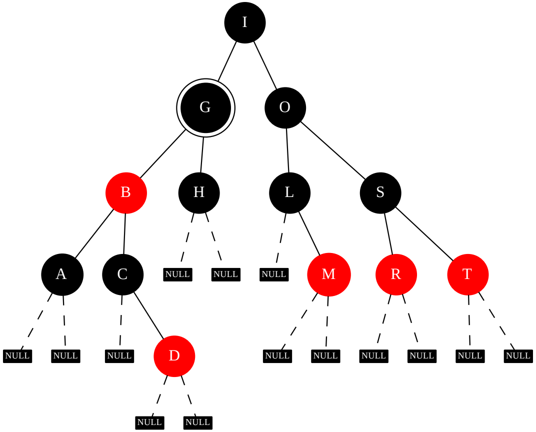


Insertion of 'E'

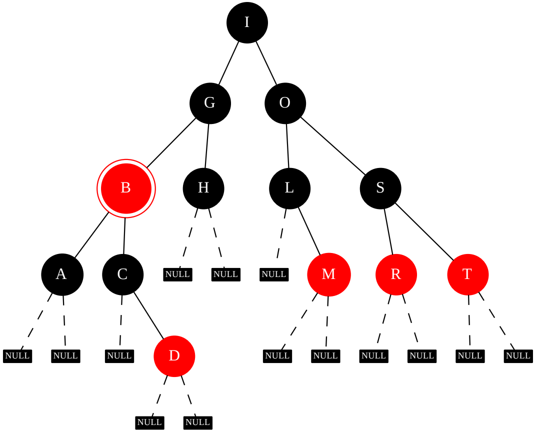
Insertion of 'E' - Finding the right position



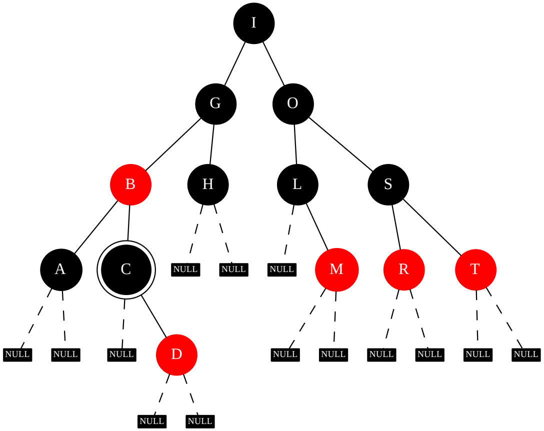
Insertion of 'E' - Finding the right position



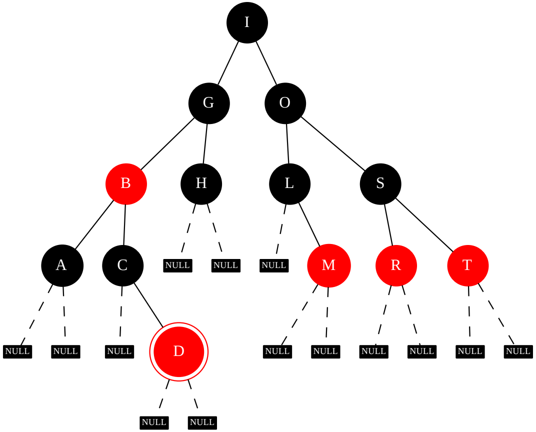
Insertion of 'E' - Finding the right position



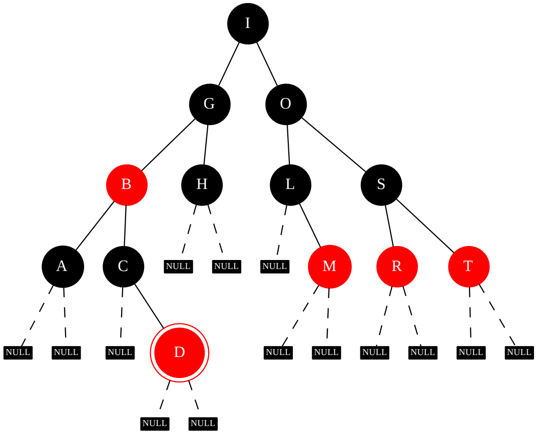
Insertion of 'E' - Finding the right position



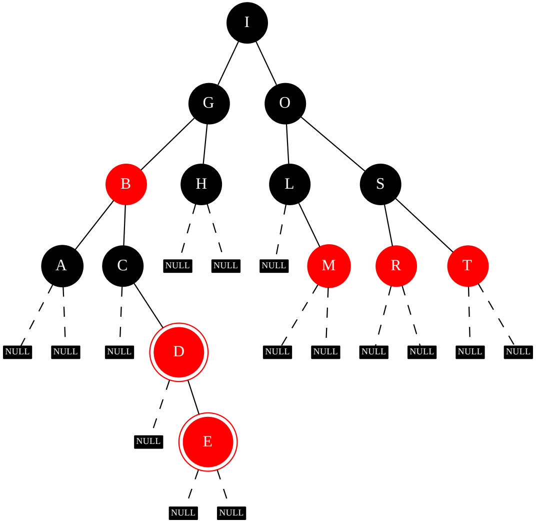
Insertion of ’E’ - Finding the right position



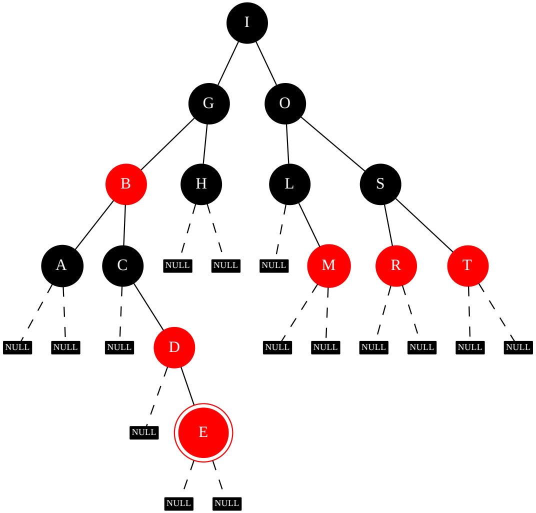
Insertion of 'E' - Inserting as right child of 'D' node



Insertion of 'E' - Property 4 violated, fixup of 'E' needed

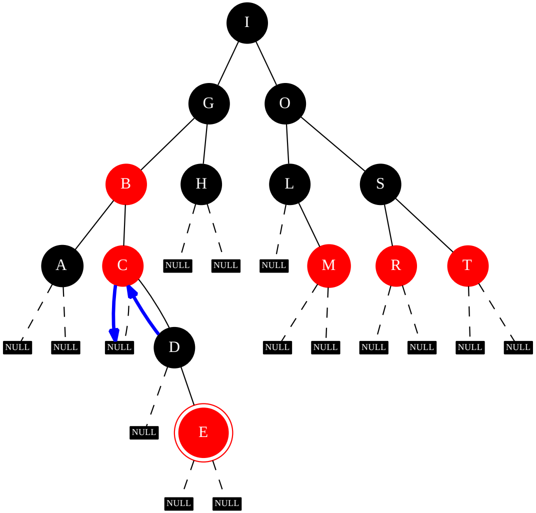


Fixup of 'E' - Case 3



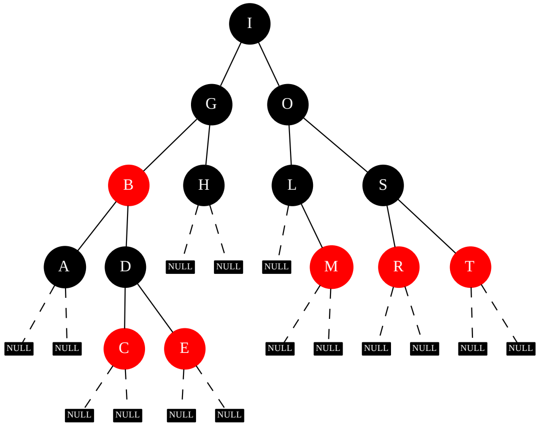
Case 3: the parent ('D') is the right child of the grandparent ('C'), the uncle is black and the node being fixed up ('E', highlighted) is the right child of the parent. Color the parent black, the grandparent red and left-rotate around the grandparent ('C').

Fixup of 'E' - Case 3

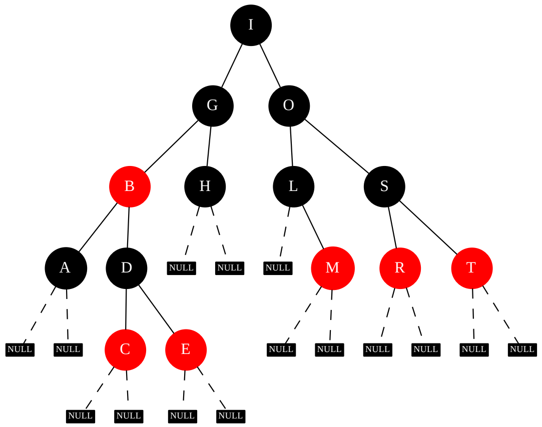


Case 3: the parent ('D') is the right child of the grandparent ('C'), the uncle is black and the node being fixed up ('E', highlighted) is the right child of the parent. Color the parent black, the grandparent red and left-rotate (blue arrows) around the grandparent ('C').

Fixup of 'E' - Done

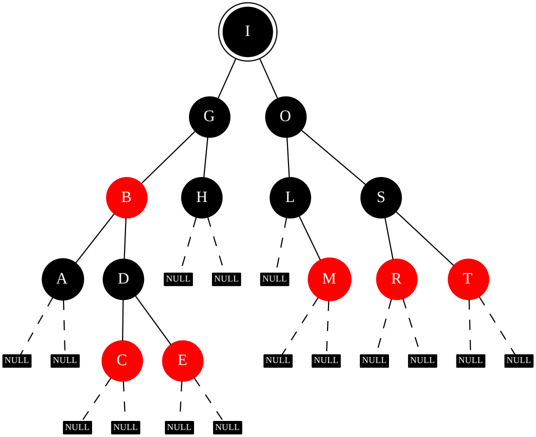


Insertion of 'E' - Outcome

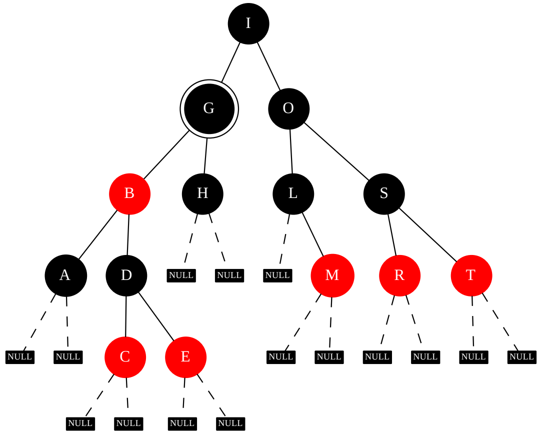


Insertion of ’G’

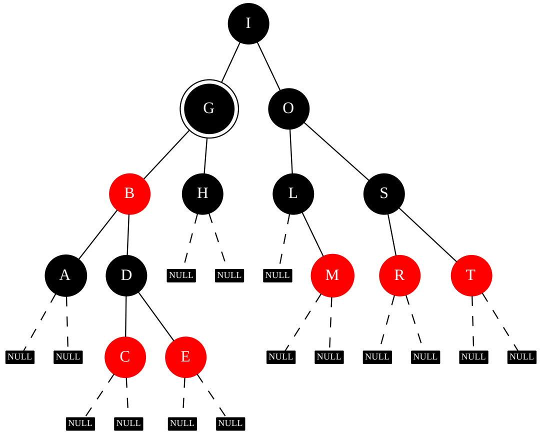
Insertion of 'G' - Finding the right position



Insertion of 'G' - Finding the right position

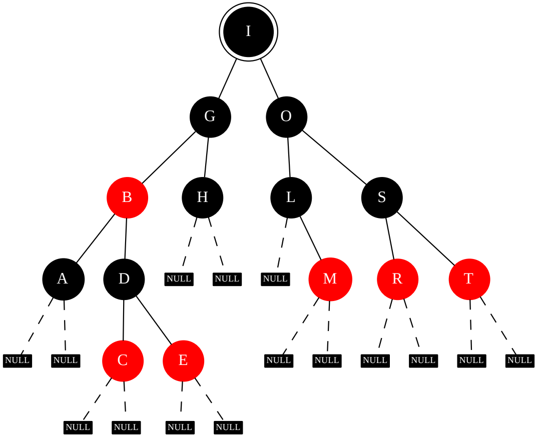


Insertion of 'G' - Key found, overwriting

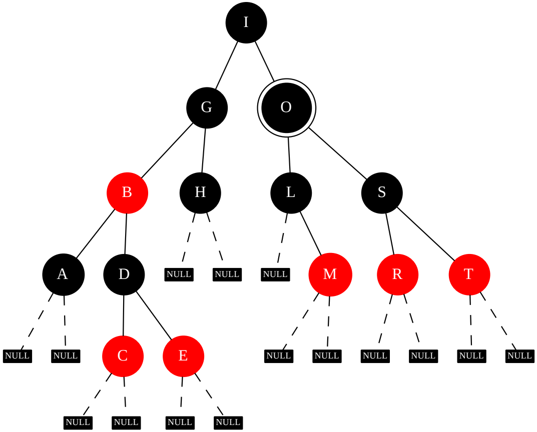


Insertion of 'S'

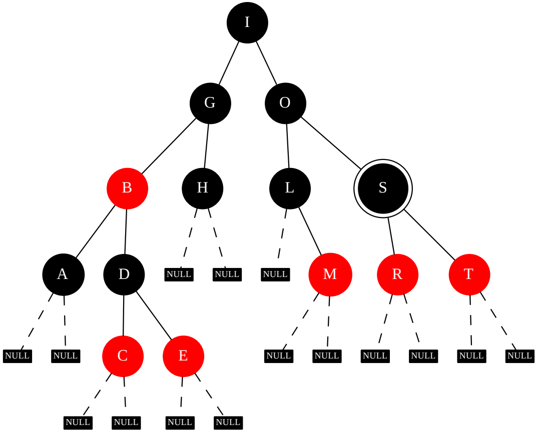
Insertion of 'S' - Finding the right position



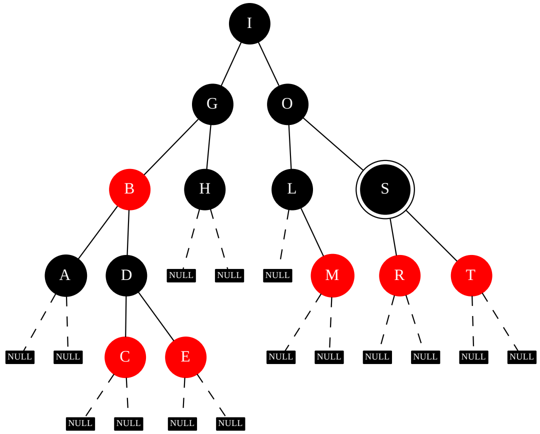
Insertion of 'S' - Finding the right position



Insertion of 'S' - Finding the right position

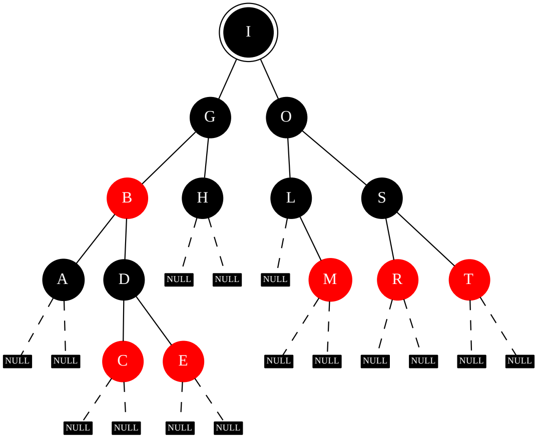


Insertion of 'S' - Key found, overwriting

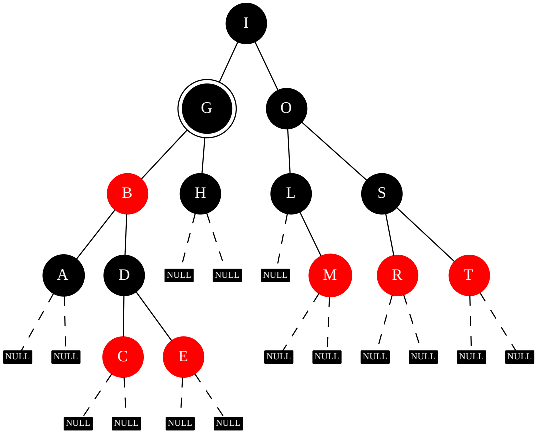


Insertion of '5'

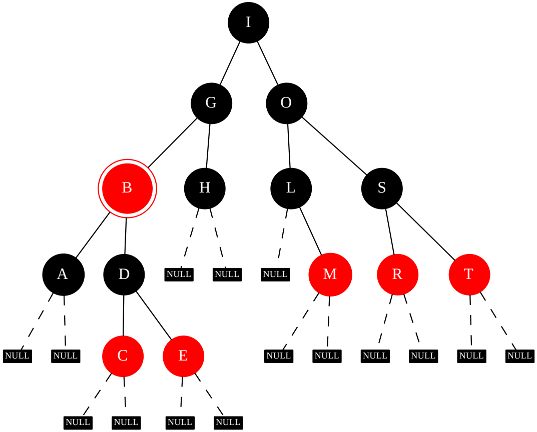
Insertion of '5' - Finding the right position



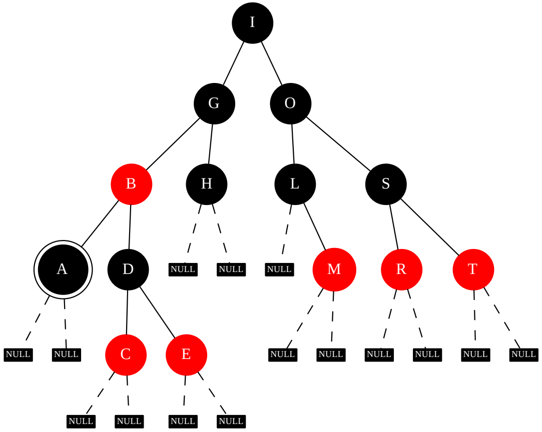
Insertion of '5' - Finding the right position



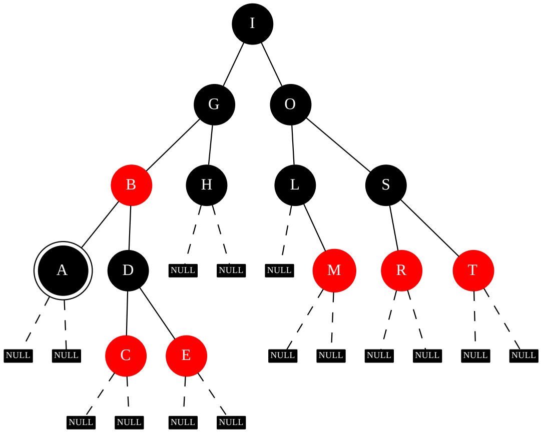
Insertion of '5' - Finding the right position



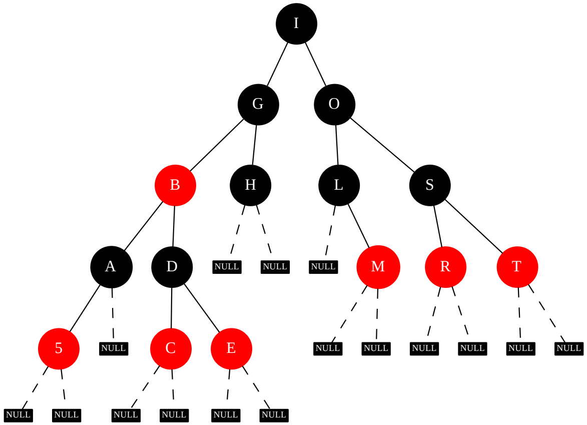
Insertion of '5' - Finding the right position



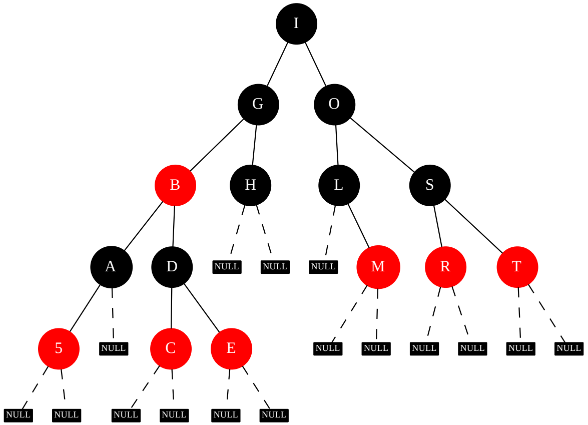
Insertion of '5' - Inserting as left child of 'A' node



Insertion of '5' - No property violated, no fixup needed

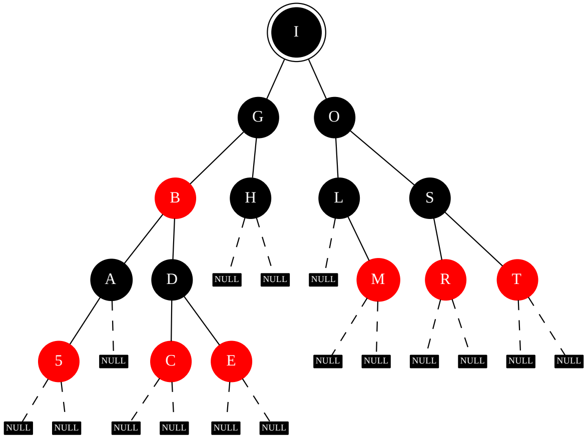


Insertion of '5' - Outcome

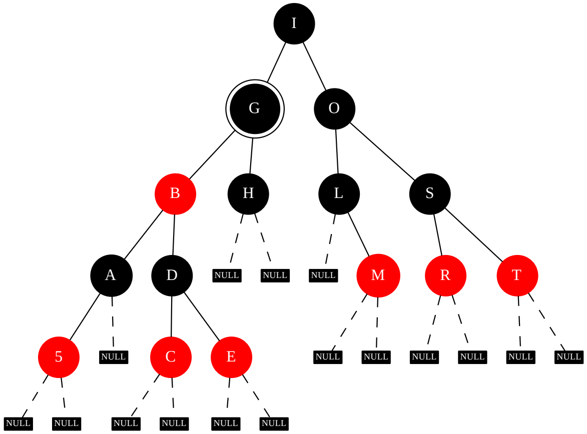


Insertion of '0'

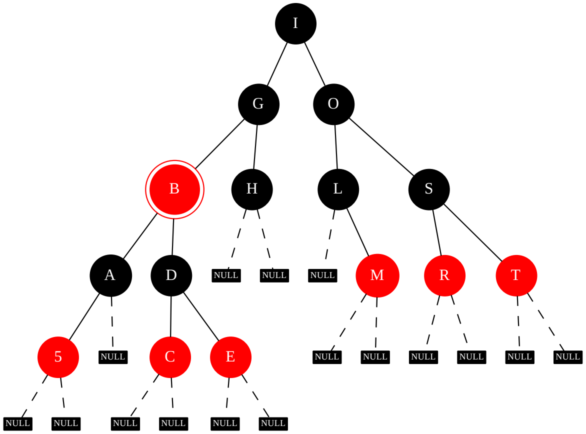
Insertion of '0' - Finding the right position



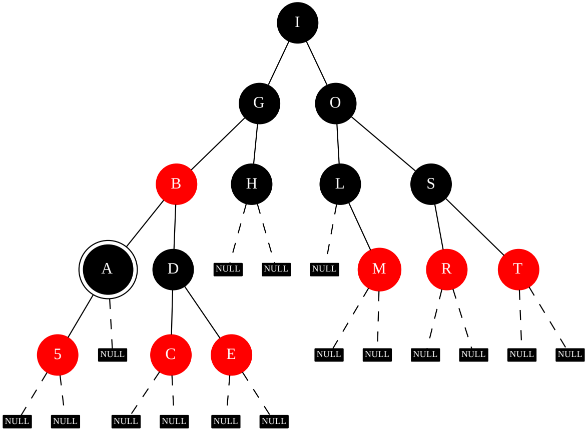
Insertion of '0' - Finding the right position



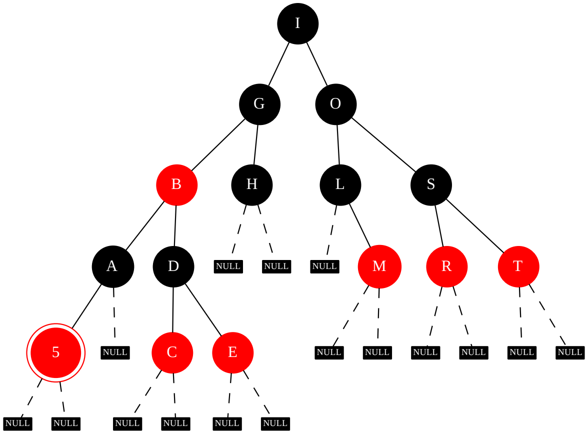
Insertion of '0' - Finding the right position



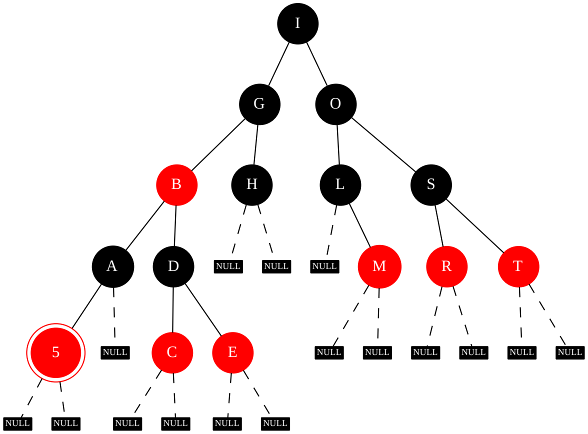
Insertion of '0' - Finding the right position



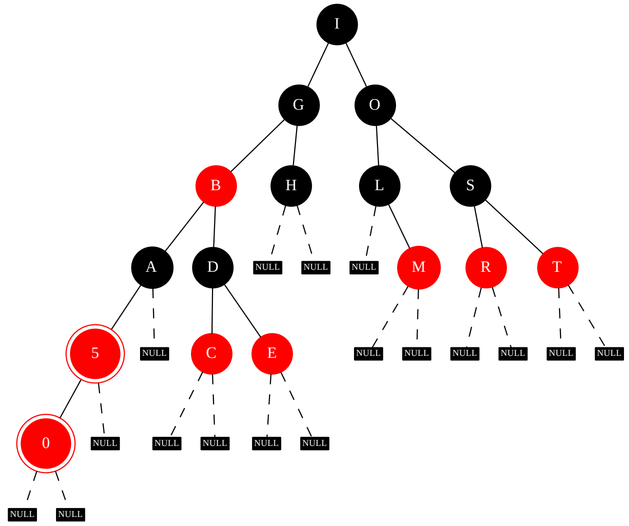
Insertion of '0' - Finding the right position



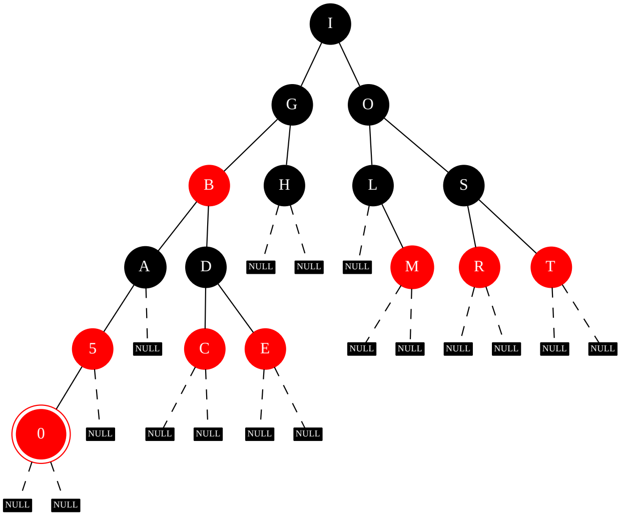
Insertion of '0' - Inserting as left child of '5' node



Insertion of '0' - Property 4 violated, fixup of '0' needed

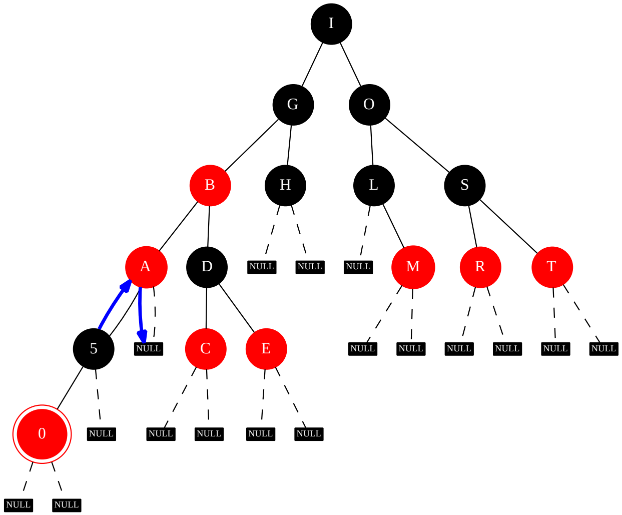


Fixup of '0' - Case 3



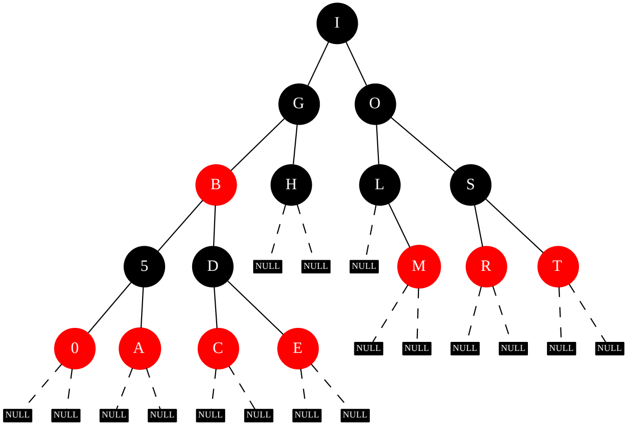
Case 3: the parent ('5') is the left child of the grandparent ('A'), the uncle is black and the node being fixed up ('0', highlighted) is the left child of the parent. Color the parent black, the grandparent red and right-rotate around the grandparent ('A').

Fixup of '0' - Case 3

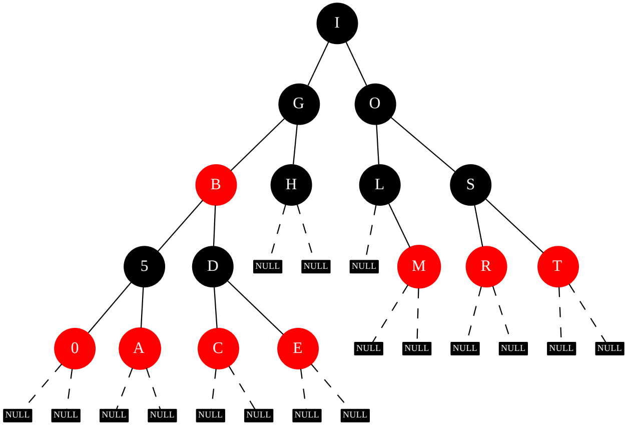


Case 3: the parent ('5') is the left child of the grandparent ('A'), the uncle is black and the node being fixed up ('0', highlighted) is the left child of the parent. Color the parent black, the grandparent red and right-rotate (blue arrows) around the grandparent ('A').

Fixup of ’0’ - Done

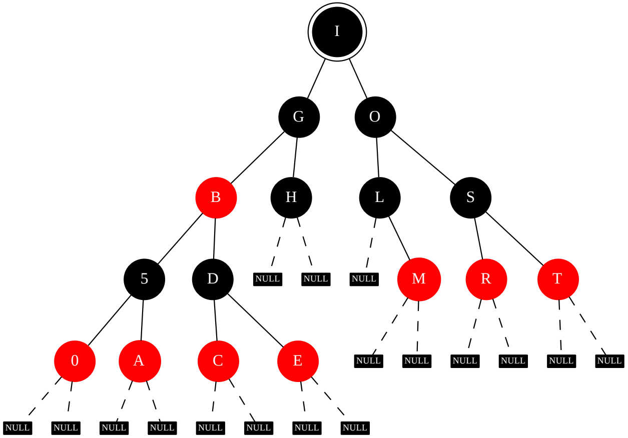


Insertion of '0' - Outcome

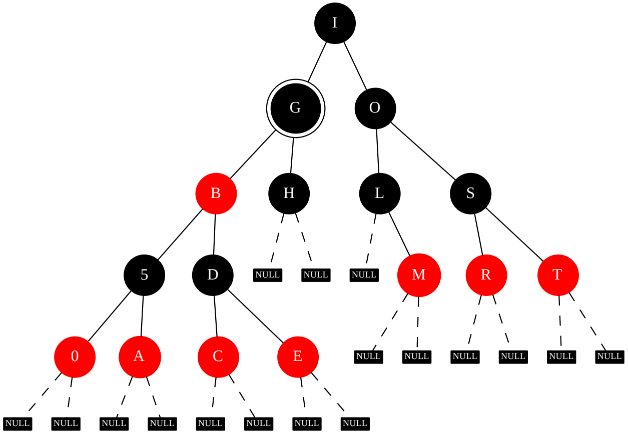


Insertion of '9'

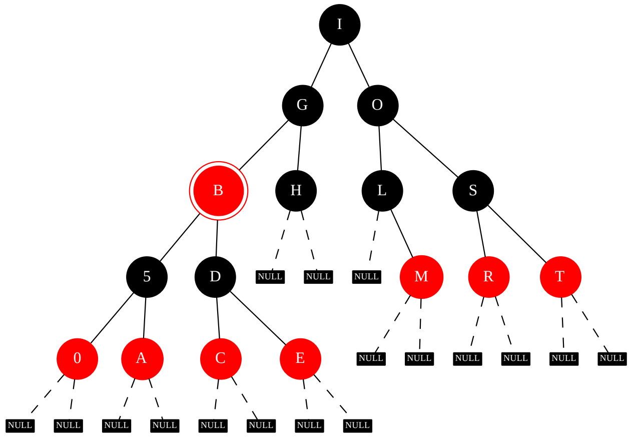
Insertion of '9' - Finding the right position



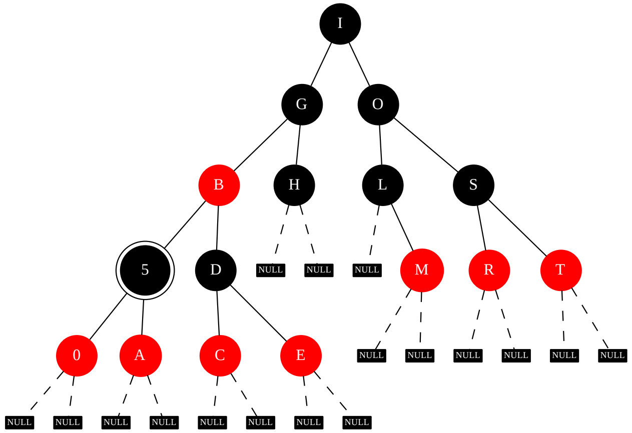
Insertion of '9' - Finding the right position



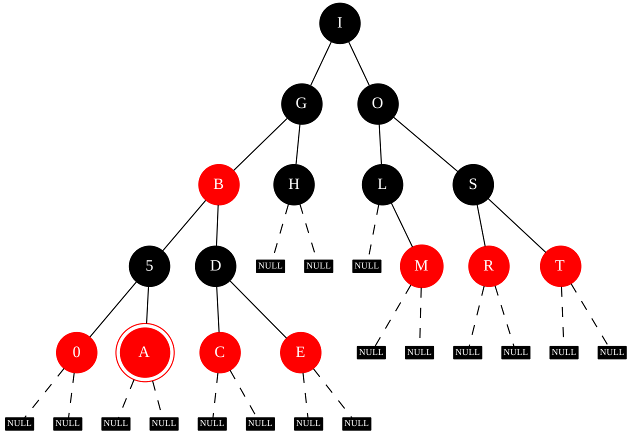
Insertion of '9' - Finding the right position



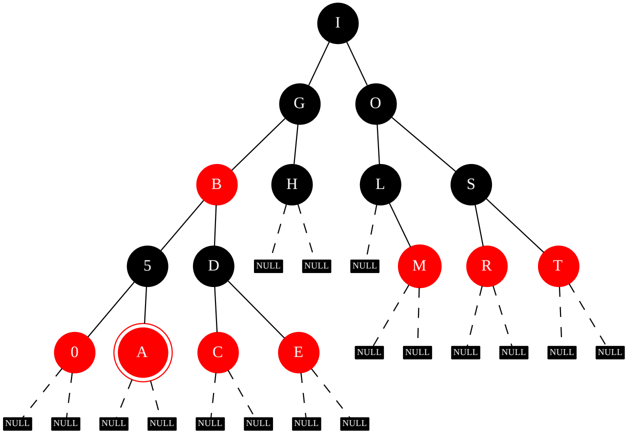
Insertion of '9' - Finding the right position



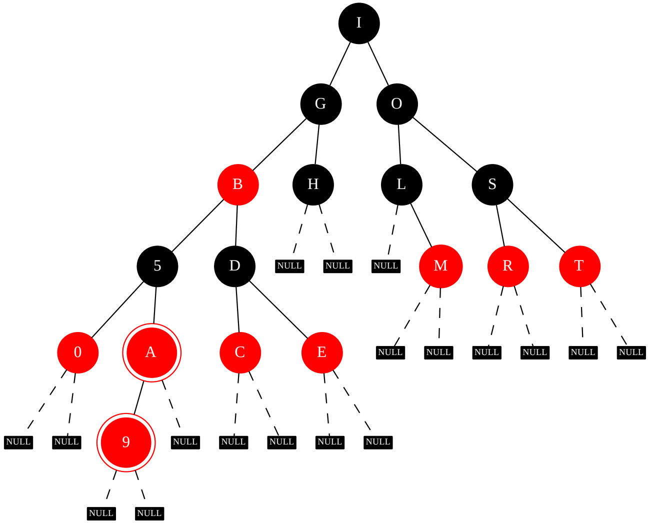
Insertion of '9' - Finding the right position



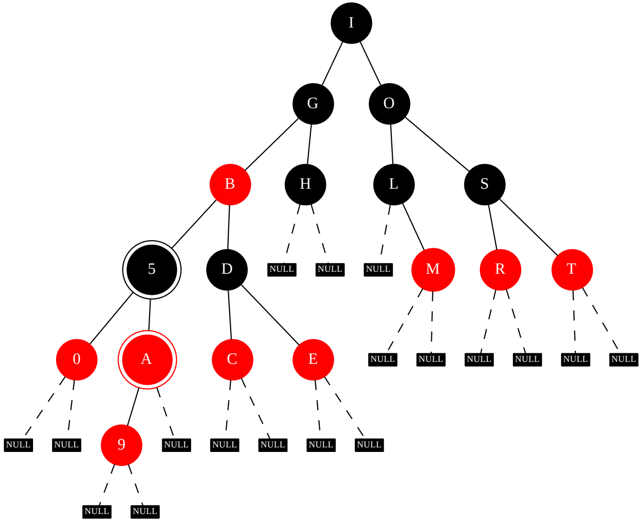
Insertion of '9' - Inserting as left child of 'A' node



Insertion of '9' - Property 4 violated, fixup of '9' needed

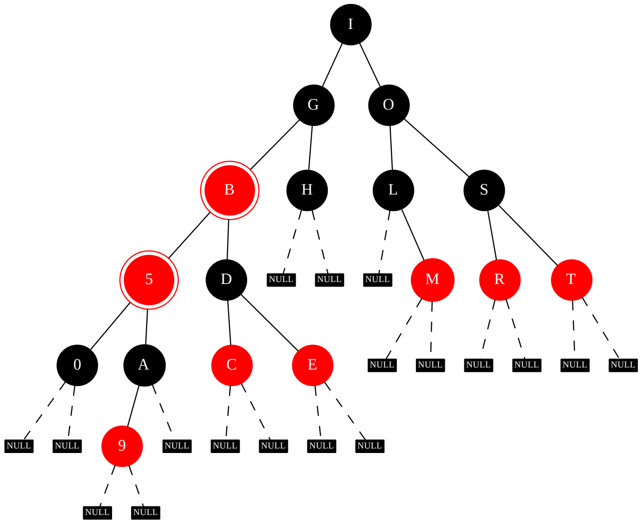


Fixup of '9' - Case 1

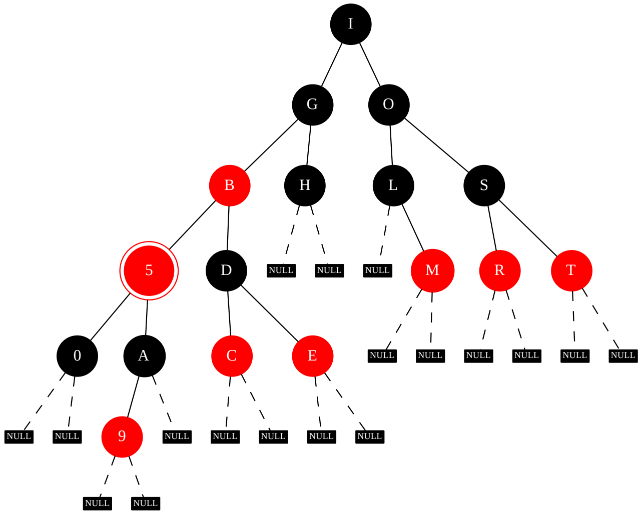


Case 1: the parent ('A', highlighted) is the right child of the grandparent ('5', also highlighted) and the uncle is red. Paint the parent and the uncle black and paint the grandparent red, then recursively fixup the grandparent ('5').

Insertion of '9' - Property 4 violated, fixup of '5' needed

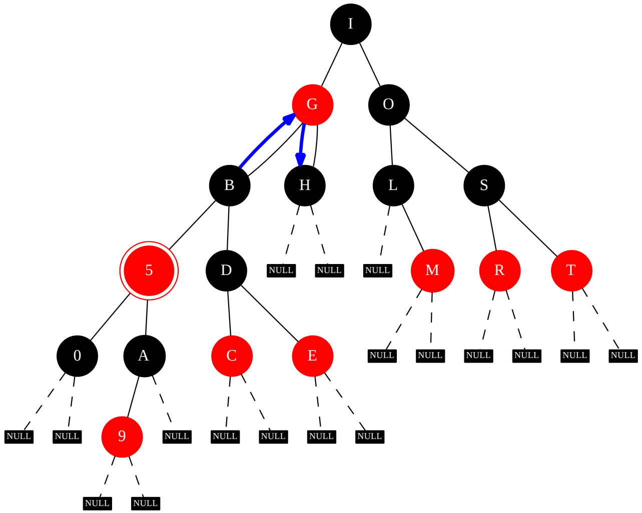


Fixup of '5' - Case 3



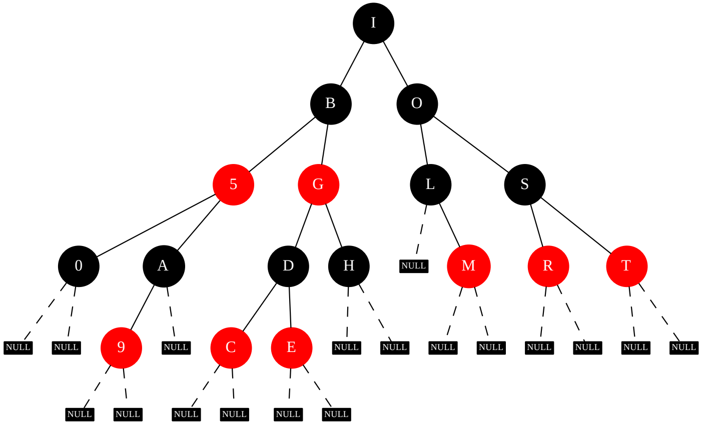
Case 3: the parent ('B') is the left child of the grandparent ('G'), the uncle is black and the node being fixed up ('5', highlighted) is the left child of the parent. Color the parent black, the grandparent red and right-rotate around the grandparent ('G').

Fixup of '5' - Case 3

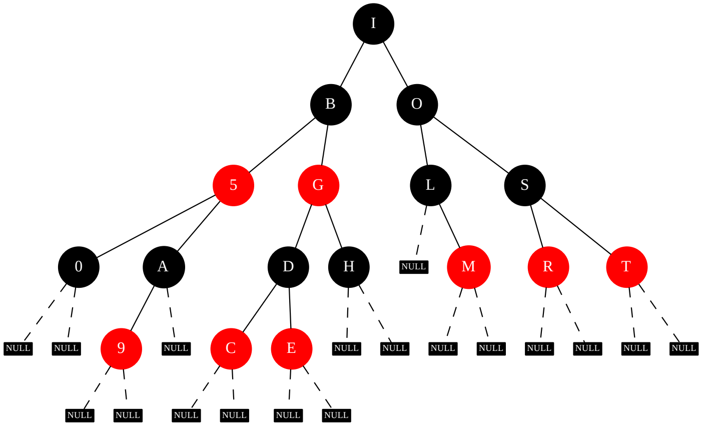


Case 3: the parent ('B') is the left child of the grandparent ('G'), the uncle is black and the node being fixed up ('5', highlighted) is the left child of the parent. Color the parent black, the grandparent red and right-rotate (blue arrows) around the grandparent ('G').

Fixup of '9' - Done

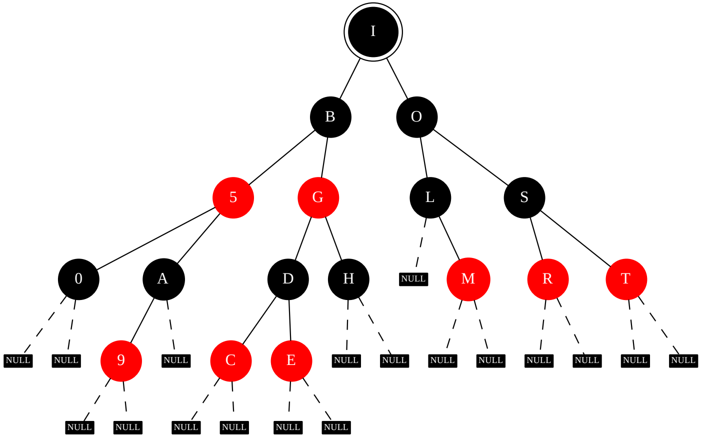


Insertion of '9' - Outcome

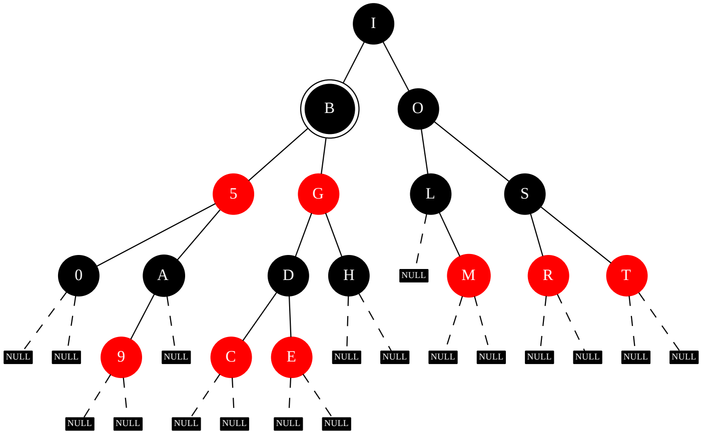


Finding the key 'A'

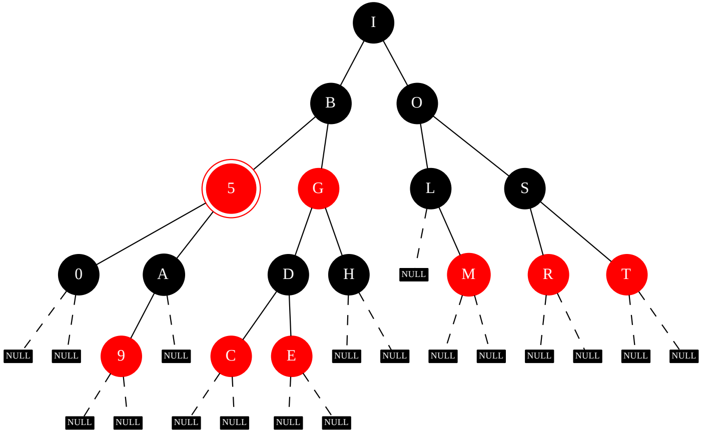
Finding the key 'A' - inspecting element 'I'



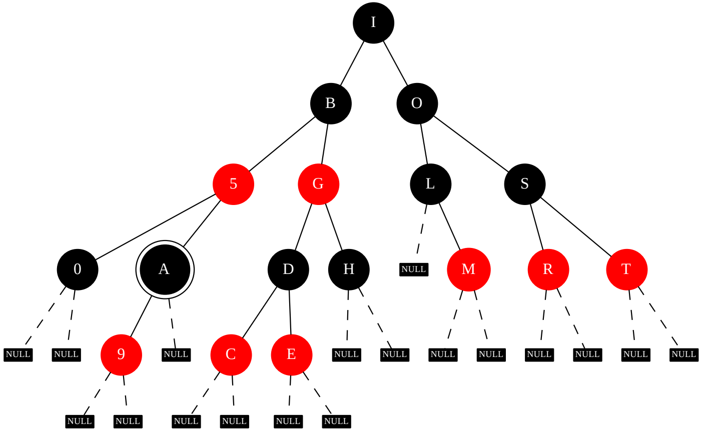
Finding the key 'A' - inspecting element 'B'



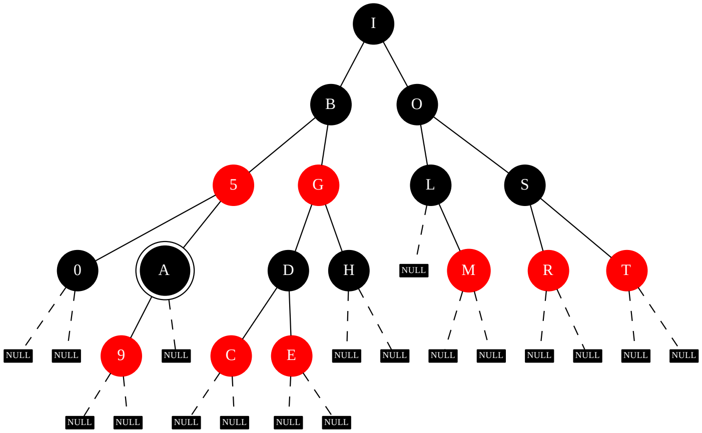
Finding the key 'A' - inspecting element '5'



Finding the key 'A' - inspecting element 'A'

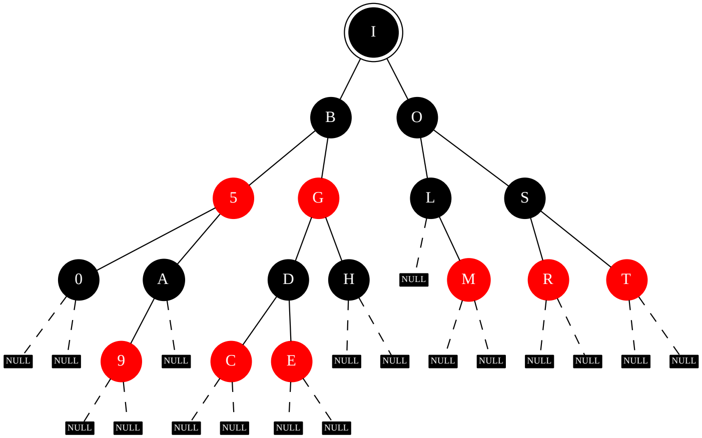


Finding the key 'A' - found

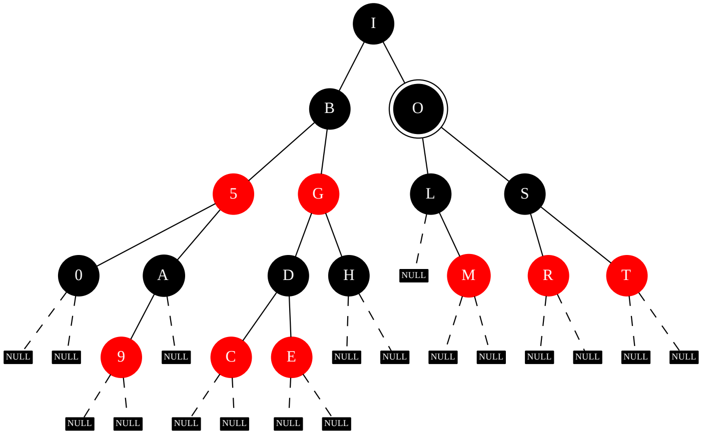


Finding the key 'Z'

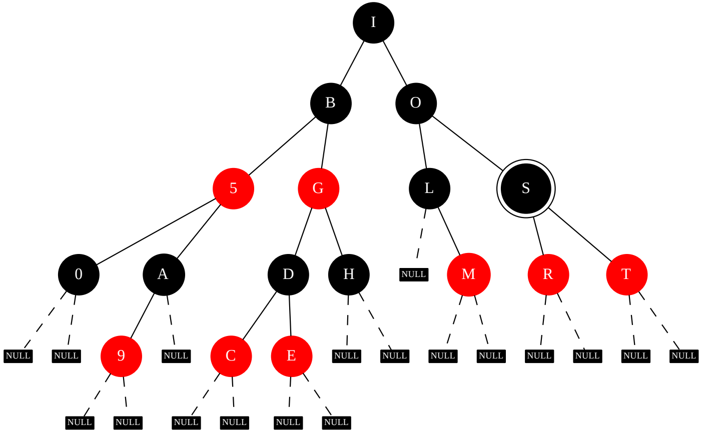
Finding the key 'Z' - inspecting element 'I'



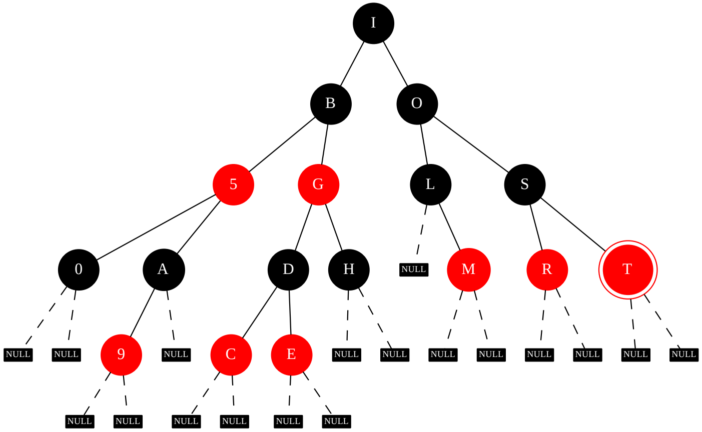
Finding the key 'Z' - inspecting element 'O'



Finding the key 'Z' - inspecting element 'S'



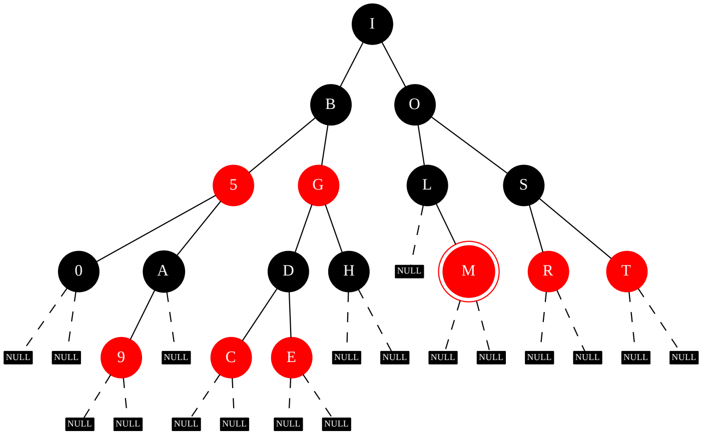
Finding the key 'Z' - inspecting element 'T'



Finding the key 'Z' - failure, key not found

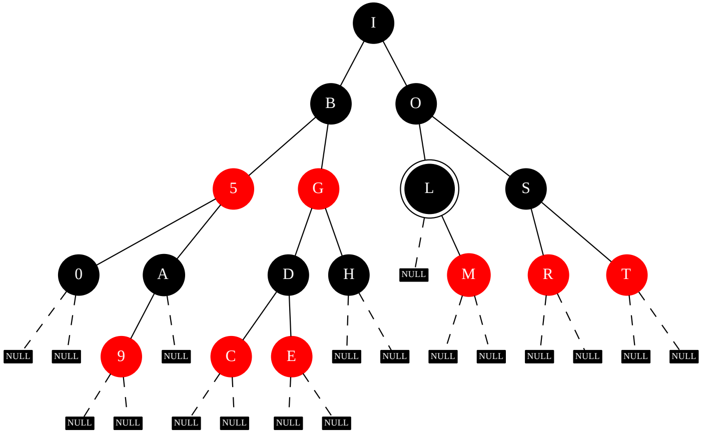
Finding the predecessor of key 'M'

Finding the predecessor of key 'M' - no left subtree



The node 'M' has no left subtree. Therefore its predecessor is its first ancestor to the left.

Finding the predecessor of key 'M' - done: 'L'



The node 'L' is the right child of its parent. Therefore it is the predecessor of 'M'.