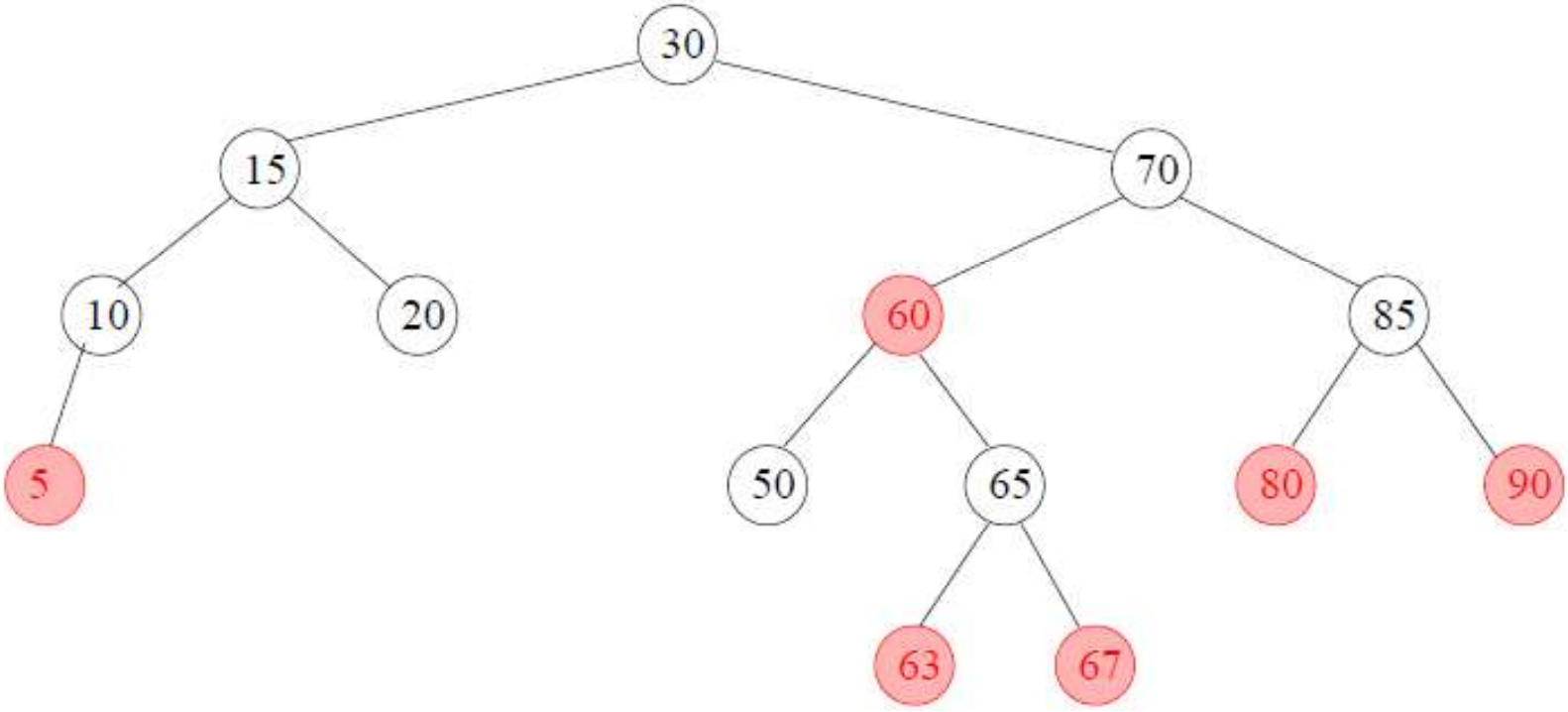
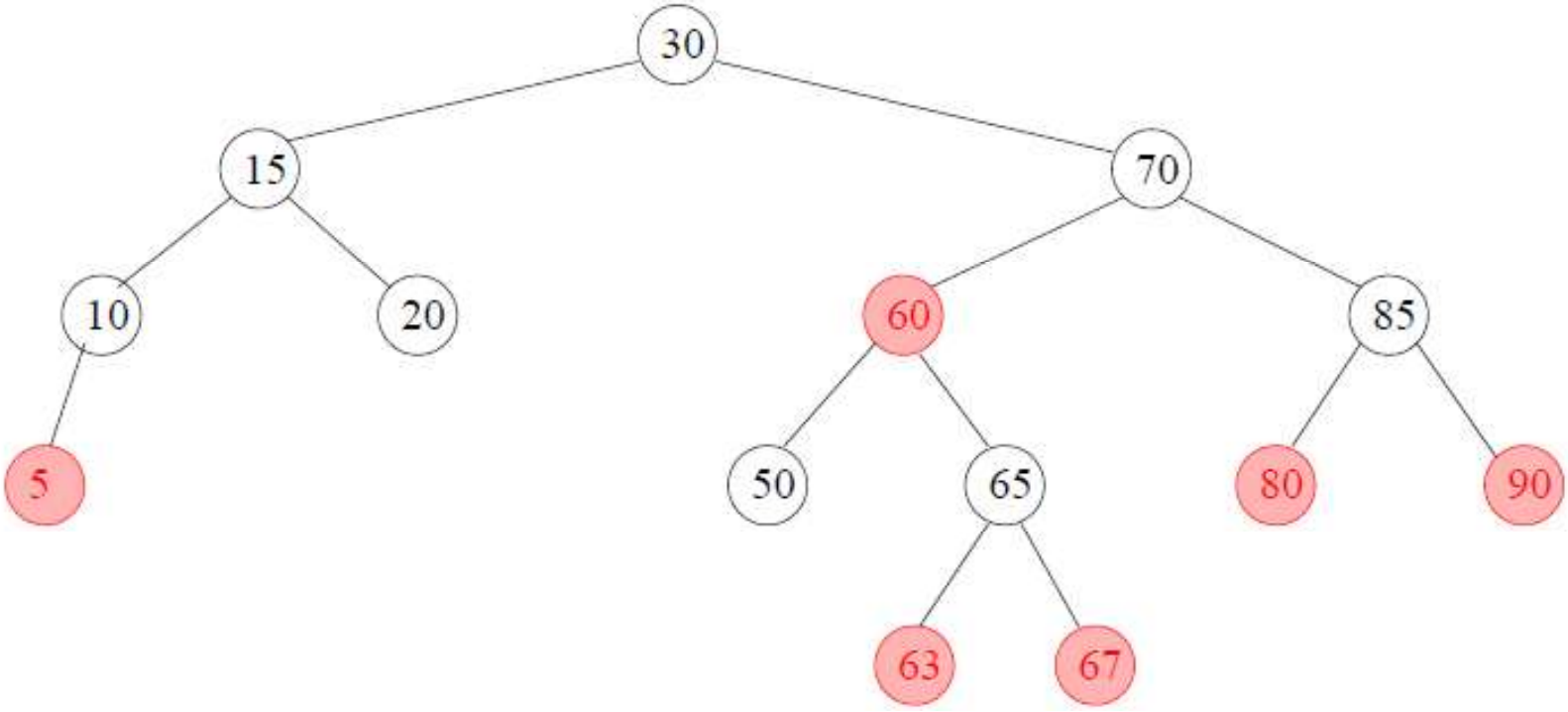
Example 2: Insert Node "64"

- Descend from root…what to change first?



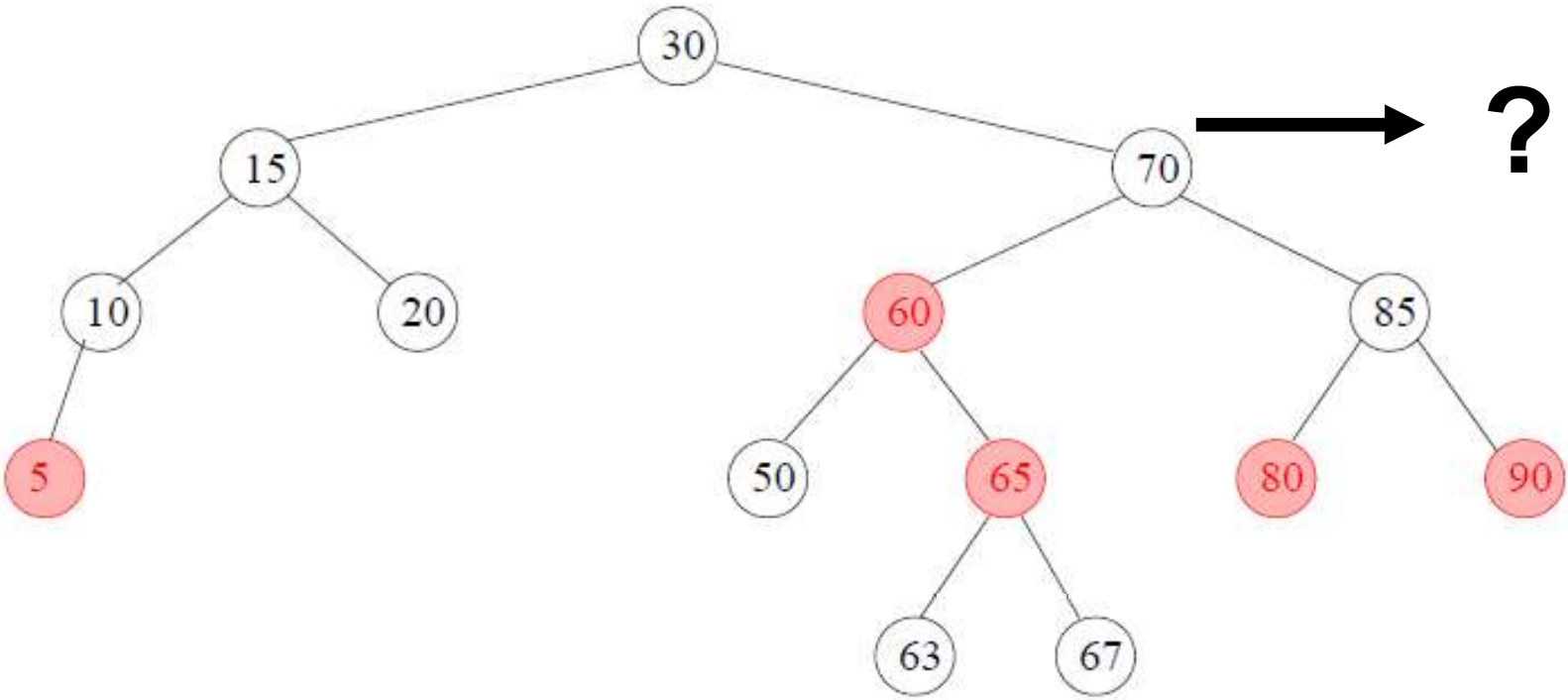
Example 2: Insert Node "64"

- Descend to node containing "65"; note it has **2 red children**; case 2 applies (***X*** is an outside grand-child).



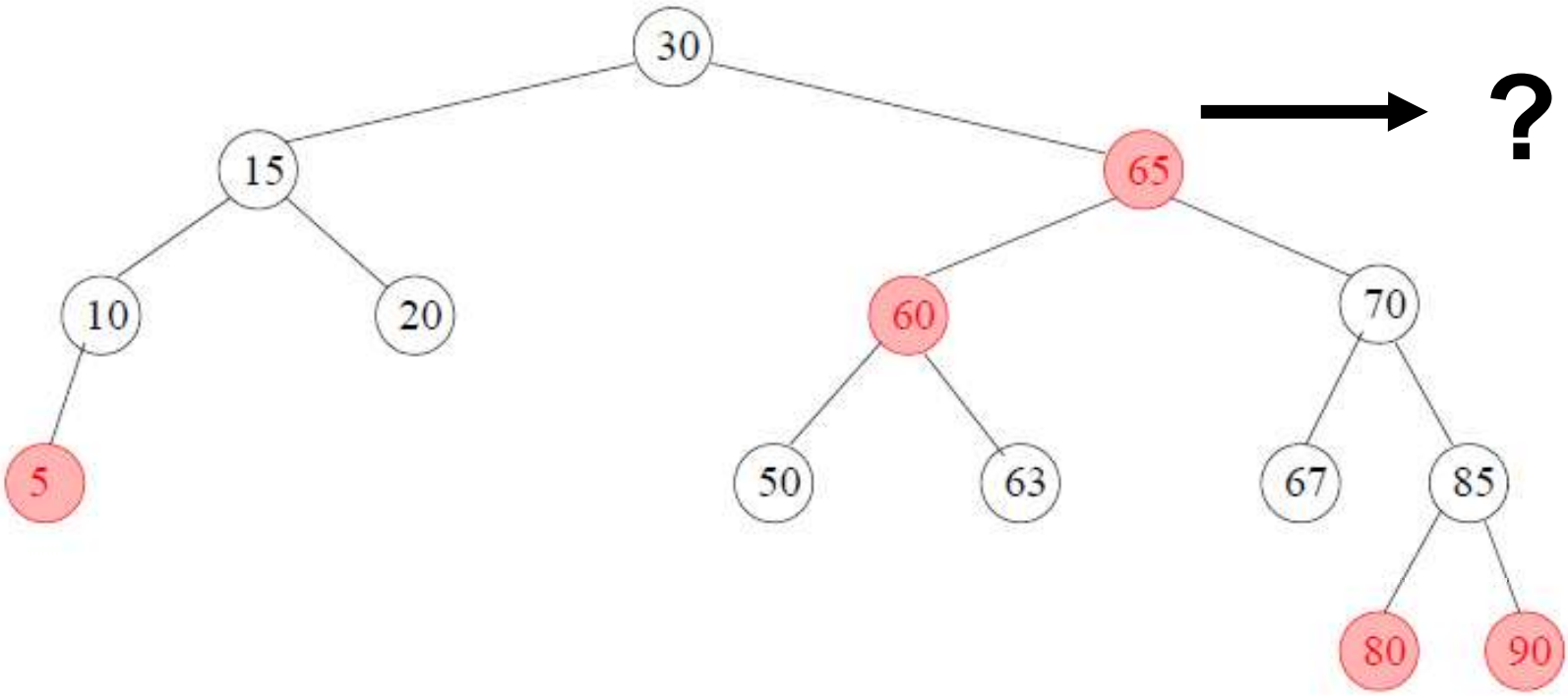
Insert Node "64" (*cont.*)

- Re-colour node ***X*** (node "65") **red**, its children **black**.
- ***X's*** parent is now also **red**, so do *tri-node operation* with ***X***, its parent ("60"), grandparent ("70").



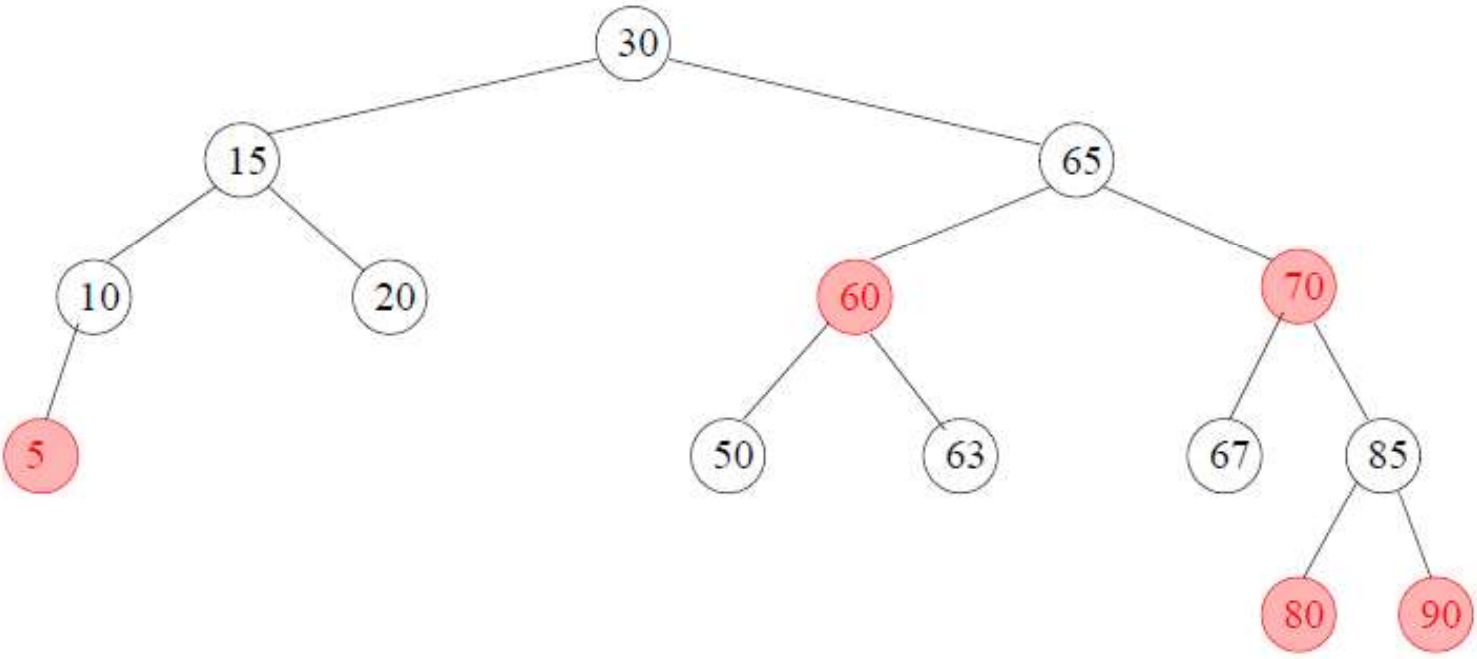
Insert Node "64" (*cont.*)

- ... still need to re-colour ***X ("60")*** and ***G*** **("70").**



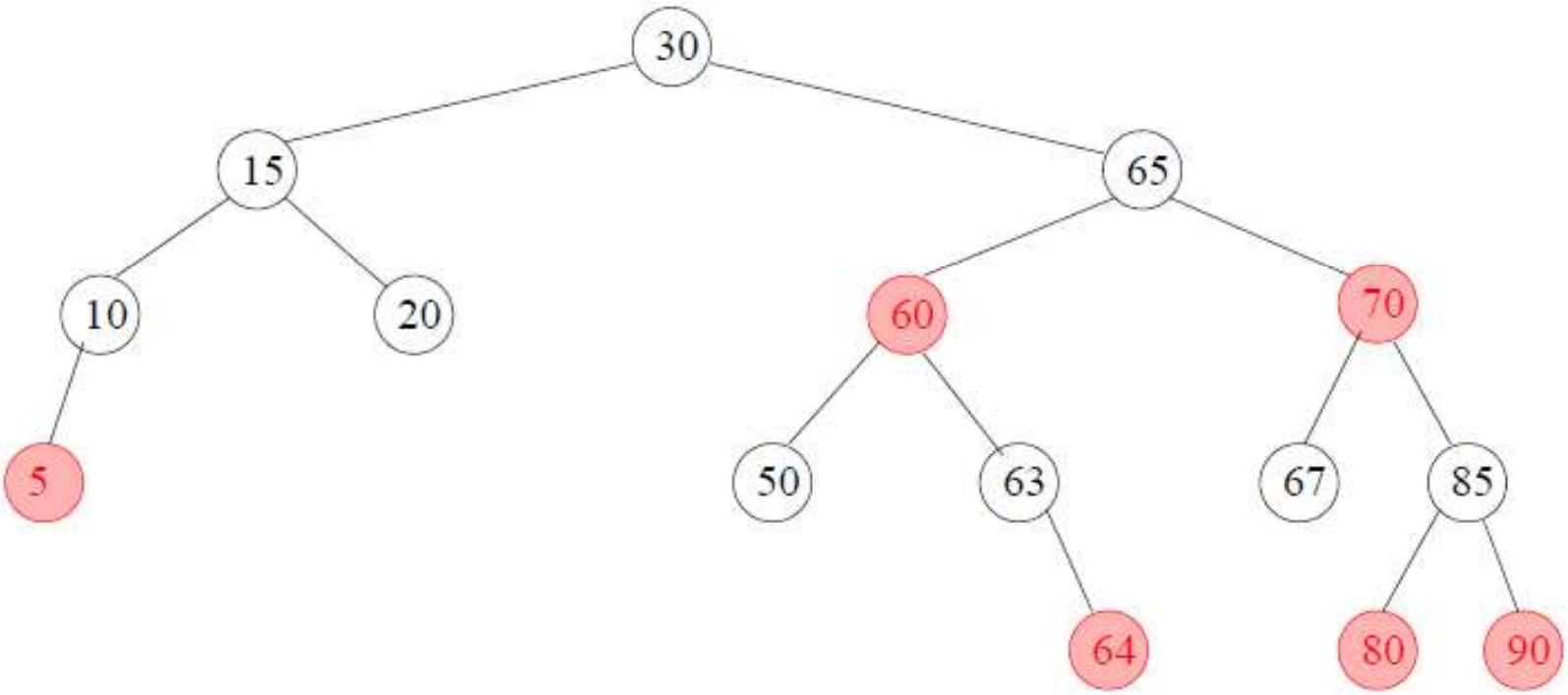
Insert Node "64" (*cont.*)

- Continue descent as ***X*** must now have *4 grand-children*, all **black** (why?), so 2 new **red** nodes (60, 70) will above parent of inserted leaf ("64").



Insert Node “64”, end

- Continue descent, insert “64” as a new **red** leaf.
- Done!



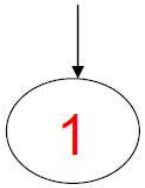
Building a Red-Black Tree

- Pseudo-code “Insert” recap:

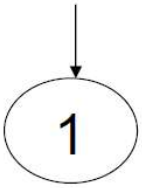
- Start at root:
  - On the way down the tree, if we see a node ***X*** that has **2 red children**, we make ***X*** **red** and its 2 children **black**.
  - If root was coloured **red**, re-colour it **black** (number of black nodes on paths from ***X*** remains unchanged).
  - If ***X's*** parent is **red** then we have **2 consecutive red nodes** (*rule violation!*)
  - To fix, apply *tri-node operation* / *re-colouring*.

Building a Red-Black Tree

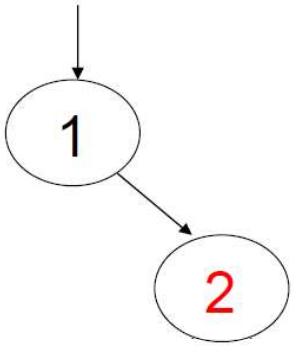
- Insert sorted list = [1 2 3 4 5 6 7]:



- Insert “1”:
  - Treat as leaf so **red**. Check if root – yes – so re-colour to **black**.

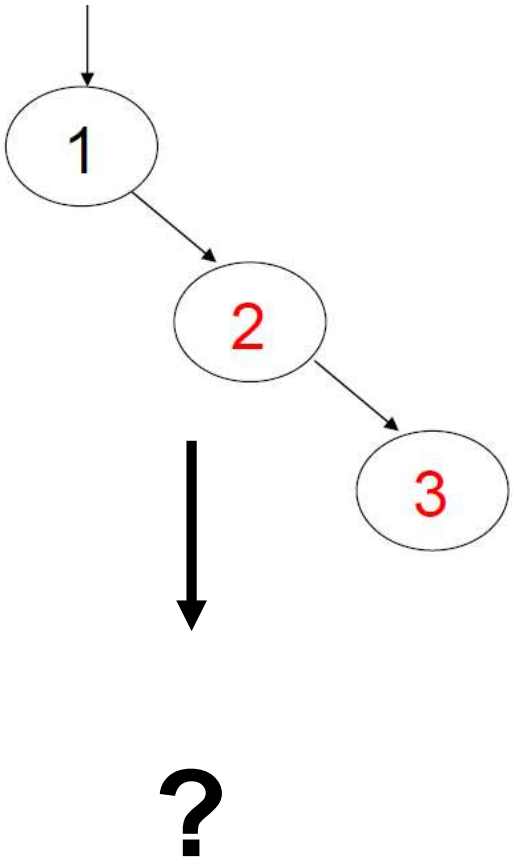


- Insert “2”
  - As above – Make “2” **red**.
  - Parent is **black** so done.



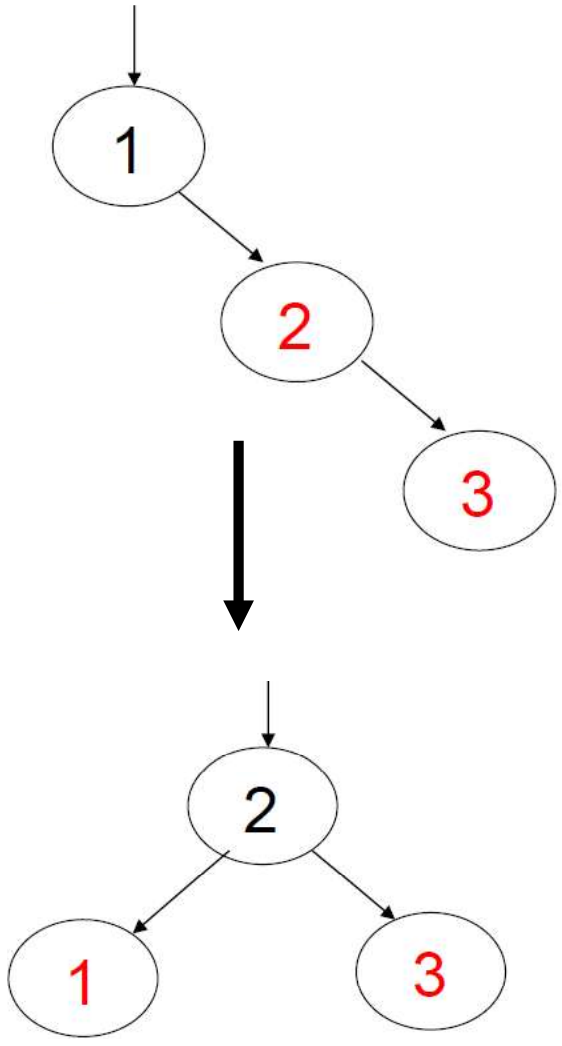
Building a Red-Black Tree

- Insert “3”:
  - Parent is **red**.
  - Parent’s sibling is **black** (null).
  - Which case?
  - *Tri-node restructure*, and *re-colour* …



Building a Red-Black Tree

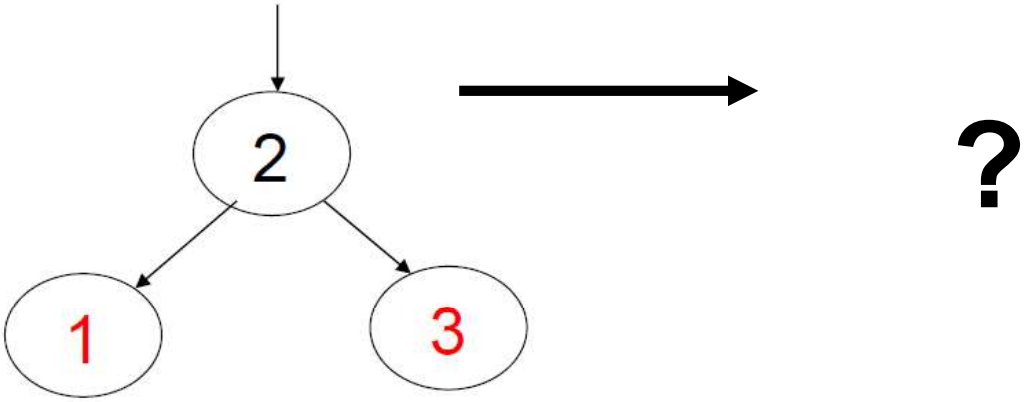
- "3" is outside "grand-child" (Case 4).
- Tri-node restructure [1, 2, 3].
- Re-colour root node.



- **Case 4:** ***P*** is right child of ***G***, and ***X*** is right child of ***P*** (***X*** is "outside grand-child" of ***G***).

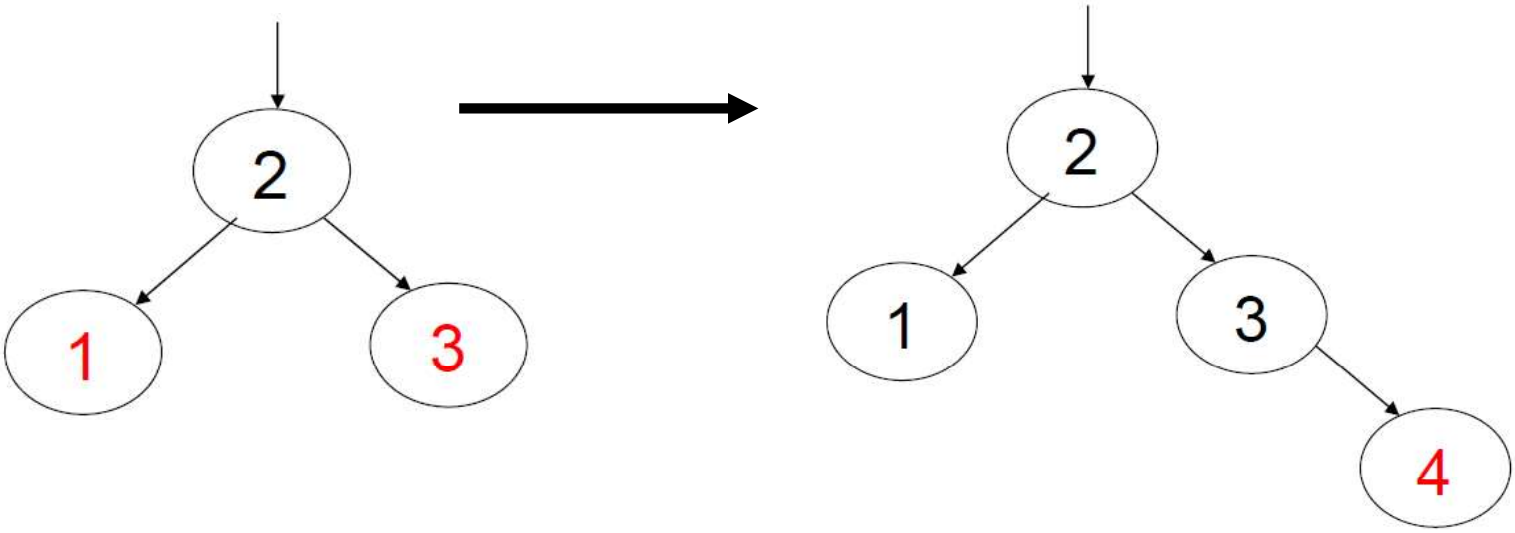
Building a Red-Black Tree

- Insert “4”:
  - Descend from root, see “2” with **2 red children**.
  - Re-colour “2” **red** and children [1, 3] **black**.
  - Then…?



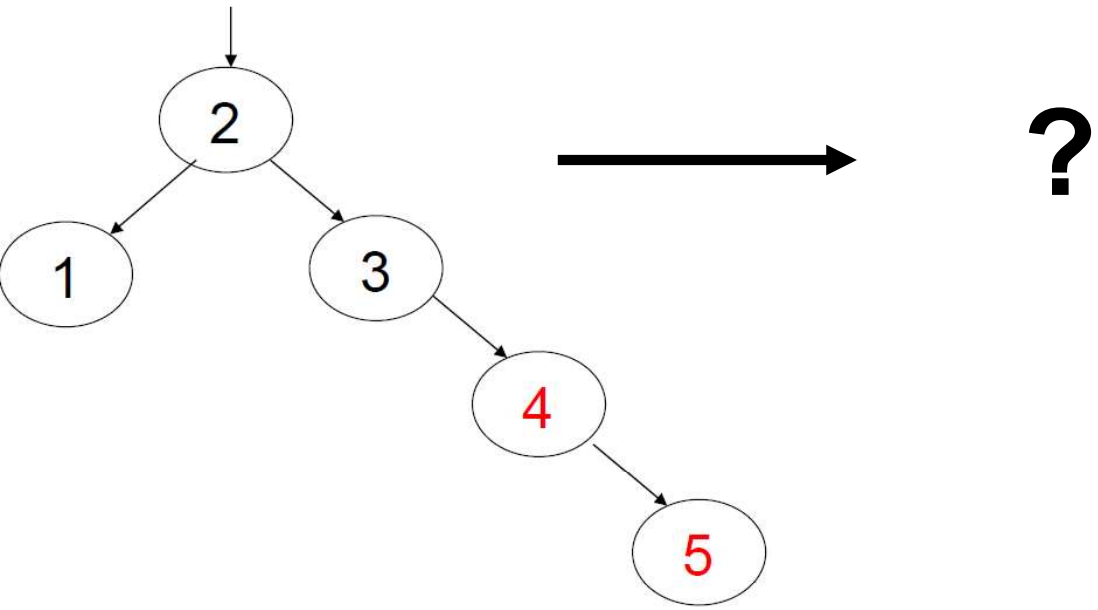
Building a Red-Black Tree

- Re-colour “2” **red** and children **black**.
- Check if “2” is root – yes – re-colour **black**.
- “4” inserted (**red** leaf), parent is **black**, so done.



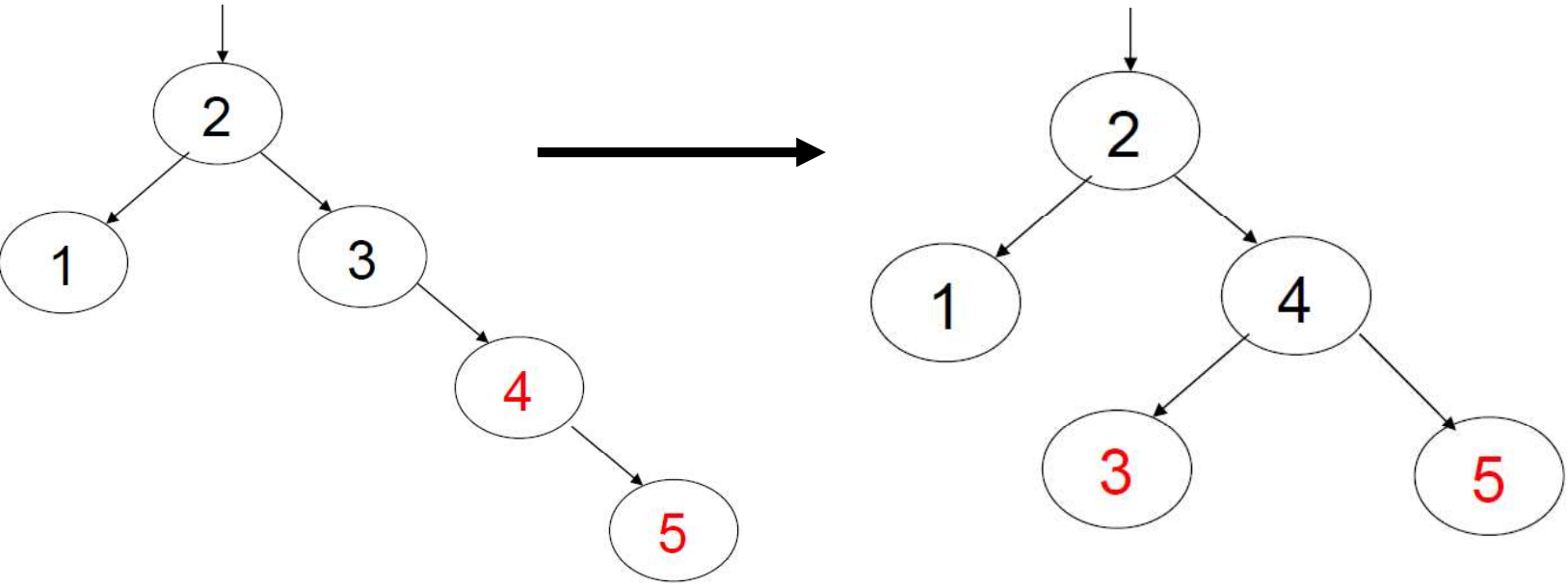
Building a Red-Black Tree

- Insert “5”:
  - Parent of “5” is red – **2 reds** rule violation.
  - *Tri-node operation* needed…which case?



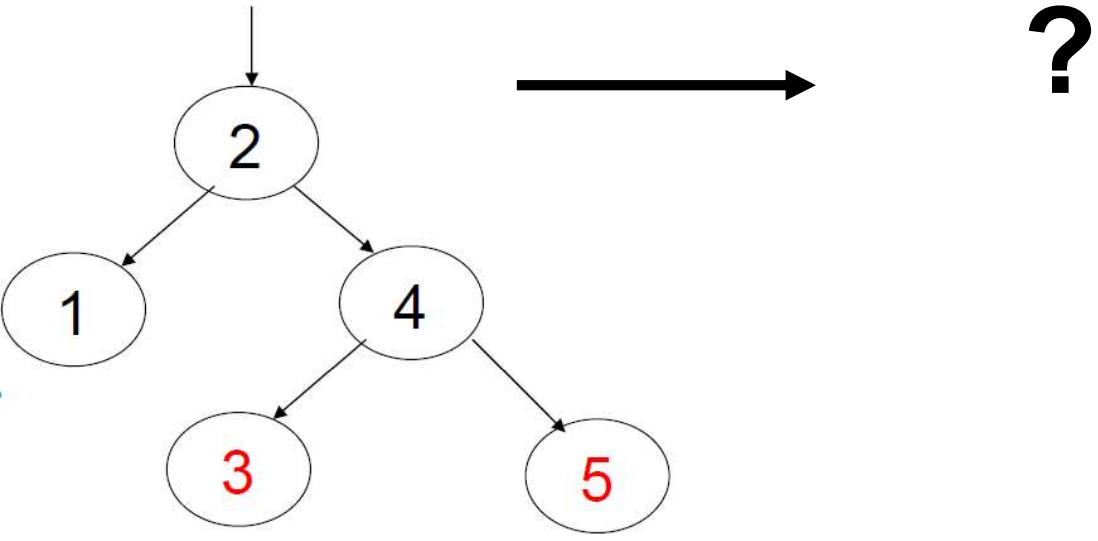
Building a Red-Black Tree

- *Tri-node operation / re-colouring* needed.
- “5” is outside grand-child.
- **Case 4:** ***P*** is right child of ***G***, and ***X*** is right child of ***P*** (***X*** is “outside grandchild” of ***G***).



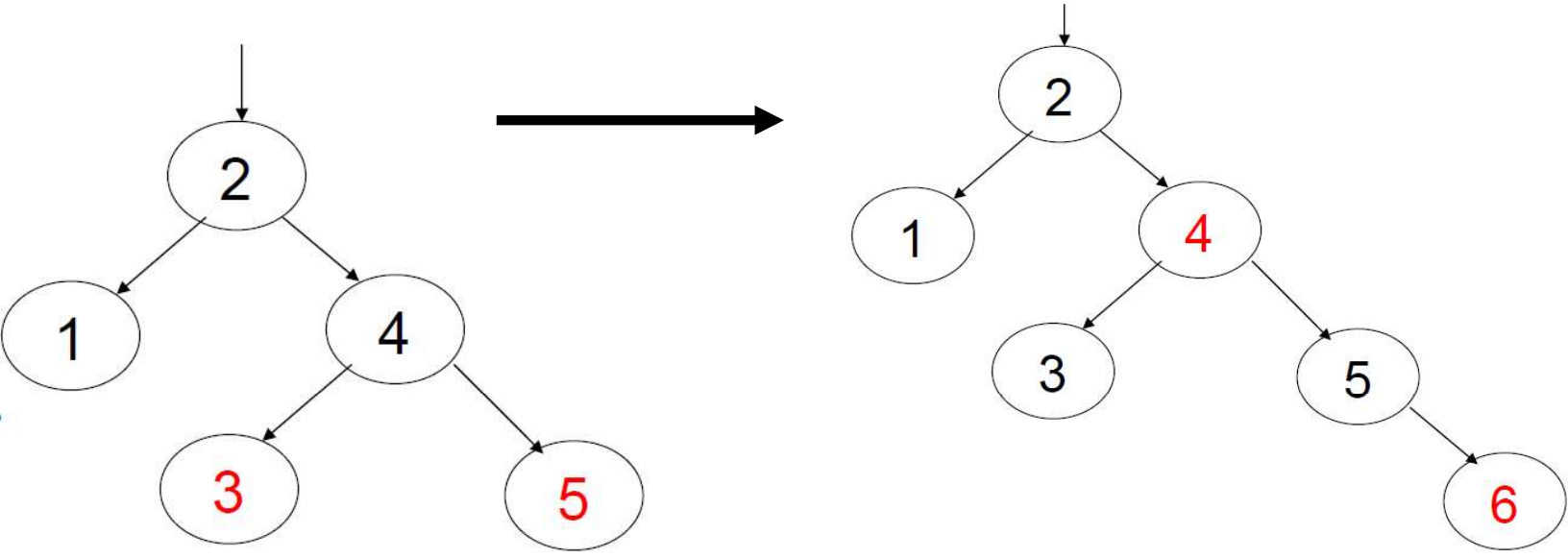
Building a Red-Black Tree

- Insert “6”:
  - Descend from root, see “4” with **2 red children**.
  - *Re-colour?*



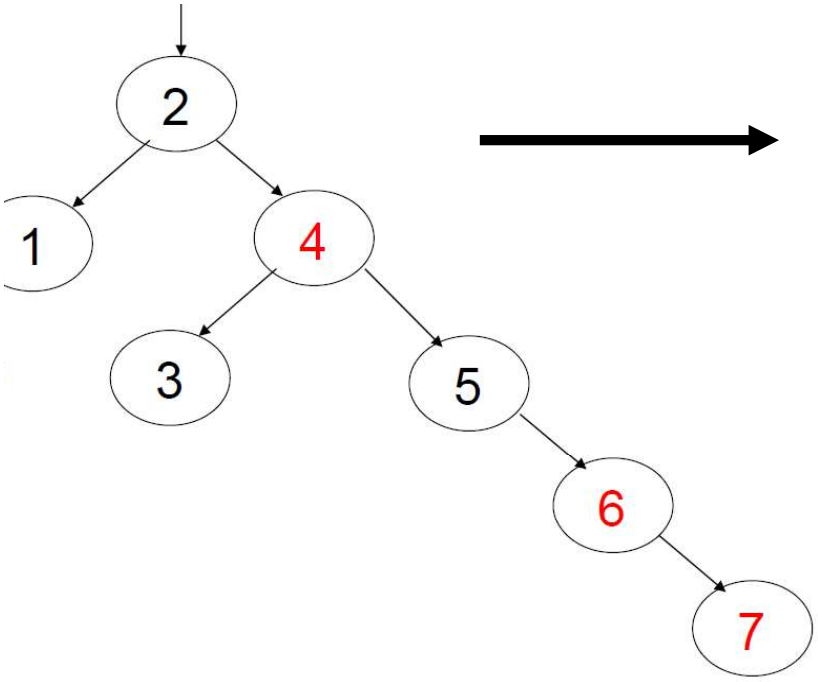
Building a Red-Black Tree

- Make “4” **red** and children **black**.
- Parent of “4” is **black**, so fine.
- Parent of “6” is **black**, so done.



Building a Red-Black Tree

- Insert “7”:
  - Parent “7” is **red** – *2 reds rule violation*.
  - Tri-node restructure / re-colour? – Which case?



Building a Red-Black Tree

- Tri-node restructure: [5, 6, 7], then re-colour.
- “7” is outside grand-child (case 4).