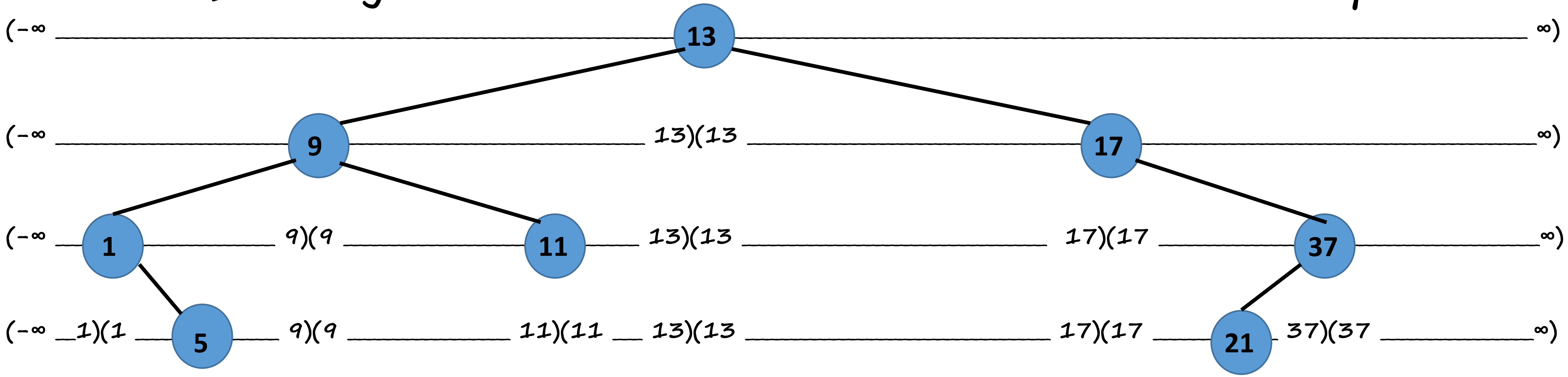
2-3 trees and Red-Black trees

Motivation

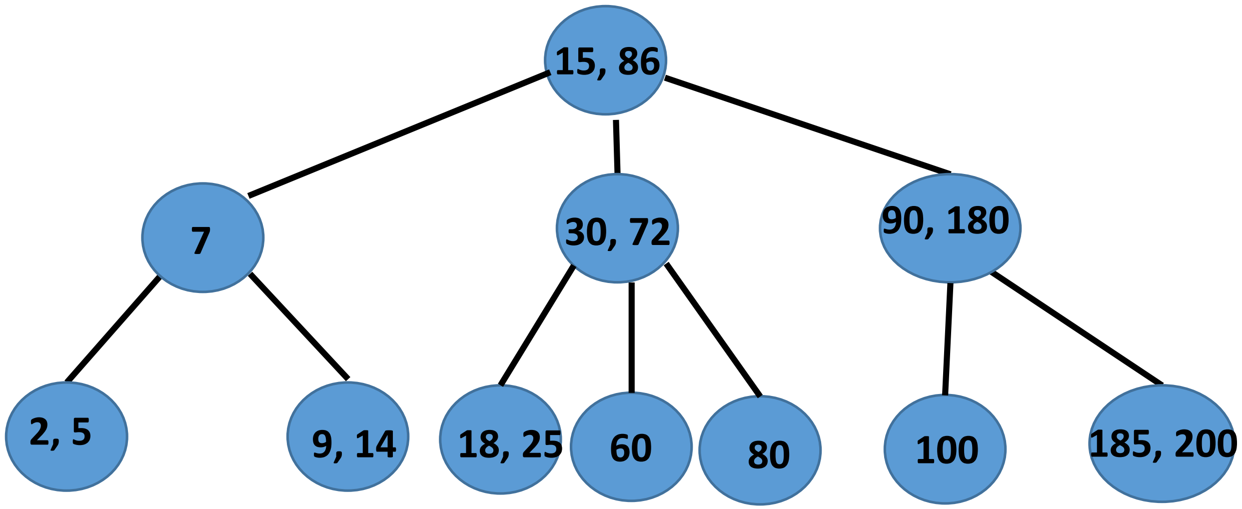
- In a BST
  - there is exactly one key at each node ≡ Associate an interval (a,b) with the key such that it lies in this interval.
  - non-leaf nodes have one or two children ≡ The key at the node partitions the associated interval into two subintervals such that the key at each children lies in the corresponding subinterval.
- In a BST, the key at a node cuts the associated interval into two parts.

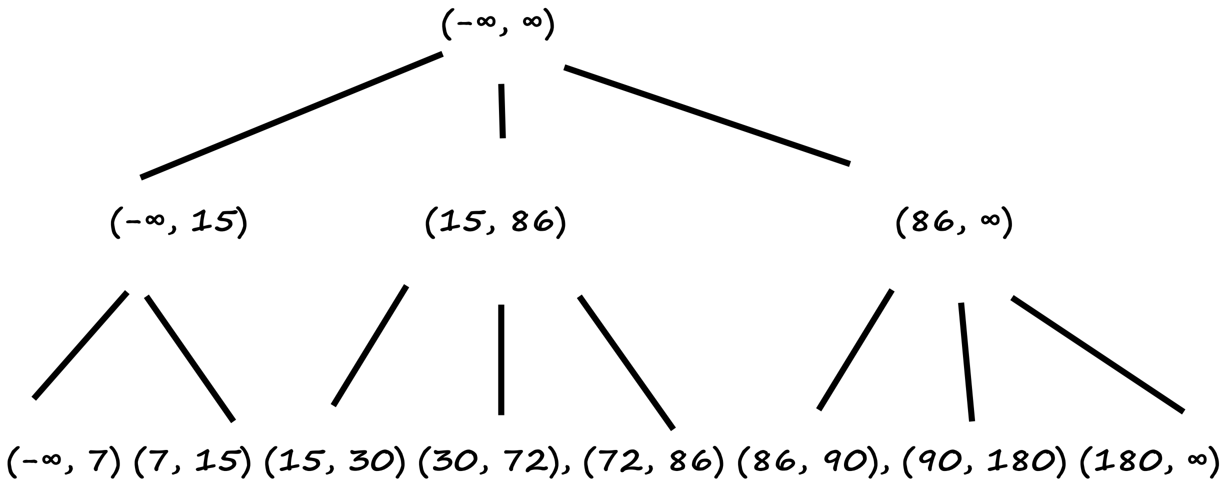


- Can this idea be extended to have more than one key at a node? If yes, then how would the search work?

2-3 trees

- Suppose we permit up to two keys at a node, say $k_1 < k_2$. Then,
  - Associate an interval (a,b) with the keys such that $(k_1, k_2) \leq (a, b)$.
  - The keys at the node partitions the associated interval into three subintervals such that the key at each children lies in the corresponding subinterval; viz., (a, k_1), (k_1, k_2), (k_2, b).
- This creates a search tree such that
  - each node has one or two keys,
  - accordingly, each non-leaf node has two or three children
  - to make searching faster (≡ height shorter), all the leaf nodes are kept at the same level.





Height of 2-3 trees on n nodes

- In a 2-3 tree, all the leaf nodes are at same level.
- The height of the tree on n nodes is the shortest when each node is a 3-node. So,

$$\min h = \log_3 n \approx 0.631 \log_2 n$$

- The height of the tree on n nodes is the longest when each node is a 2-node. So,

$$\max h = \log_2 n$$

Thus,

$$h = O(\log_2 n)$$

Note: A 2-3 tree of height 12 to 20 can hold up to a million keys.
A 2-3 tree of height 18 to 30 can hold up to a billion keys.

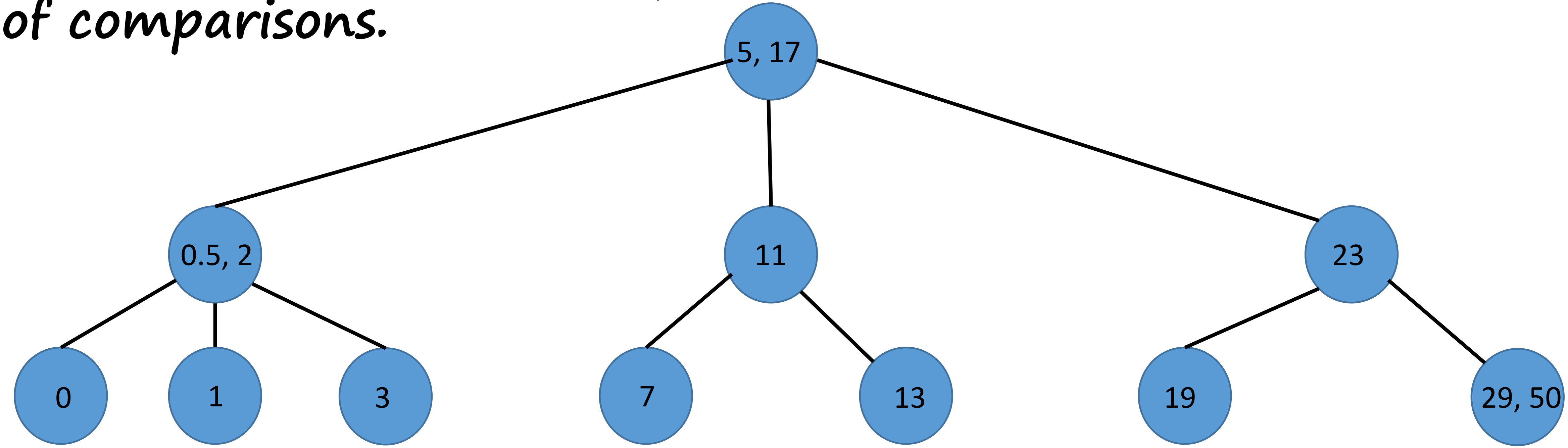
Exercise

- A 2-3 tree is a search tree in which
  - each non-leaf node has two or three children (one or two keys)
  - all the leaf nodes are at the same level
- The height of a 2-3 tree on n nodes is O(log n)
- What is a 2 tree? What is its common name? Does a 2 tree exist for any number of nodes?
- If h is the height of a 2-3 tree then what is the possible number of nodes on the tree?
- Does a 2-3 tree exist for any set of keys?
- Is there a one-one correspondence between a 2-3 tree and some BST?

Searching in a 2-3 tree

$O(\log n)$

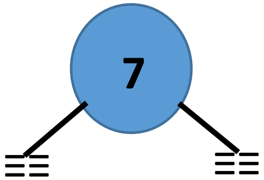
- Search(x, T)
  - Start the search at the root node. Take the
    - Left branch if $x < k_1$ where k_1, k_2 are the keys of the current node.
    - Middle branch if $k_1 < x < k_2$
    - Right branch if $x > k_2$
- **Note:** Search key x is to be compared with at most 2 keys at each node (on the search path). So, 2h is the maximum number of comparisons.



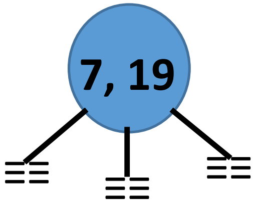
Inserting in a 2-3 tree

Construct a 2-3 tree for the set {7, 19, 11, 23, 29, 2, 13, 17, 21, 5, 3, 1, -1, 0}

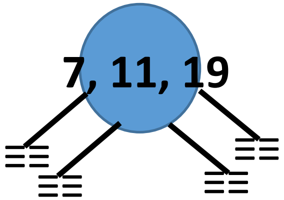
Insert 7



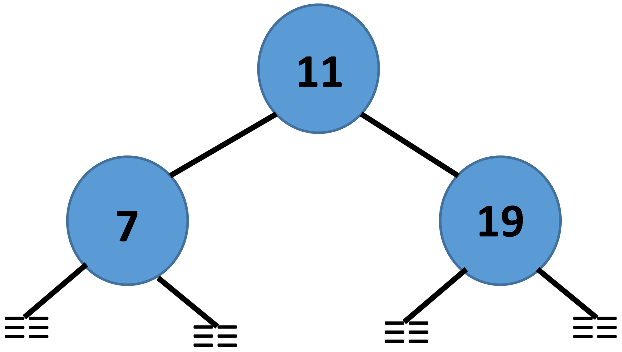
Insert 19



Insert 11



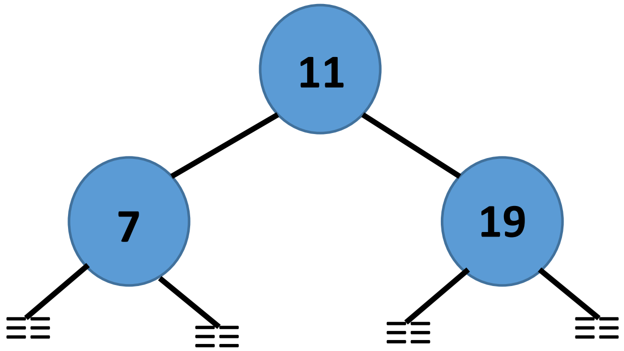
Create a 4-node, temporarily.



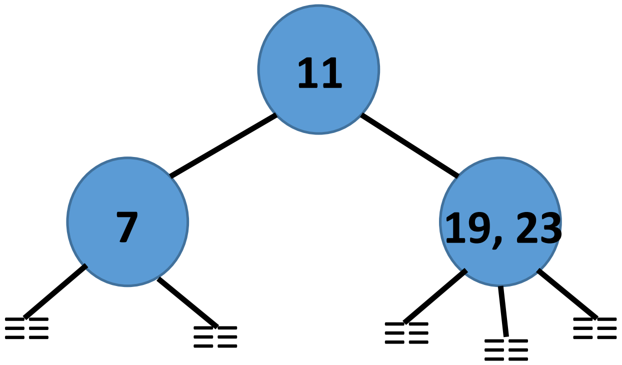
Move the middle key of the 4-node to its parent and convert the temporary node into two 2-nodes.

Inserting in a 2-3 tree

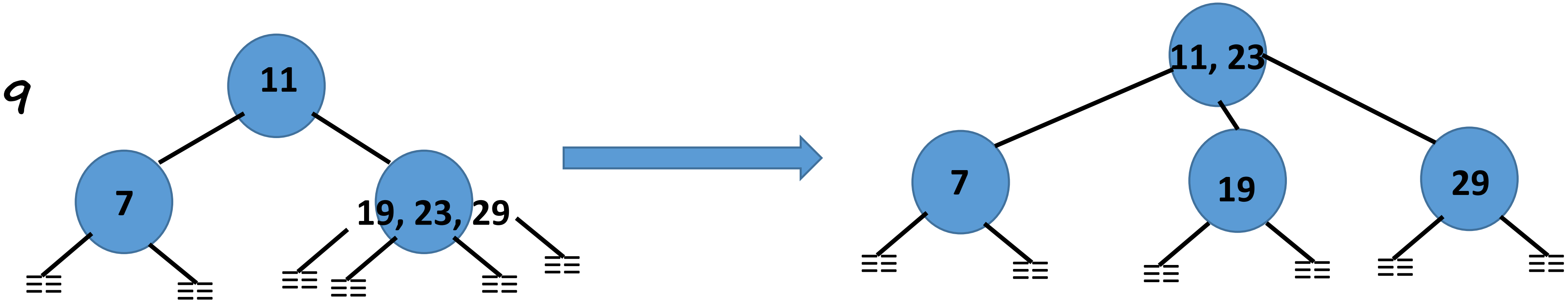
Construct a 2-3 tree for the set {7, 19, 11, 23, 29, 2, 13, 17, 21, 5, 3, 1, -1, 0}



Insert 23

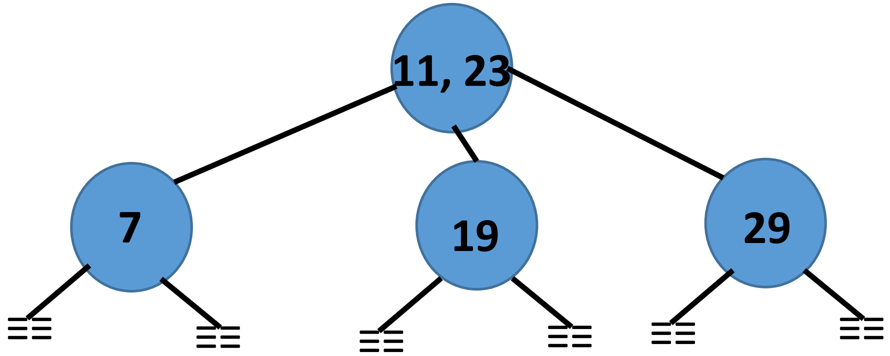


Insert 29

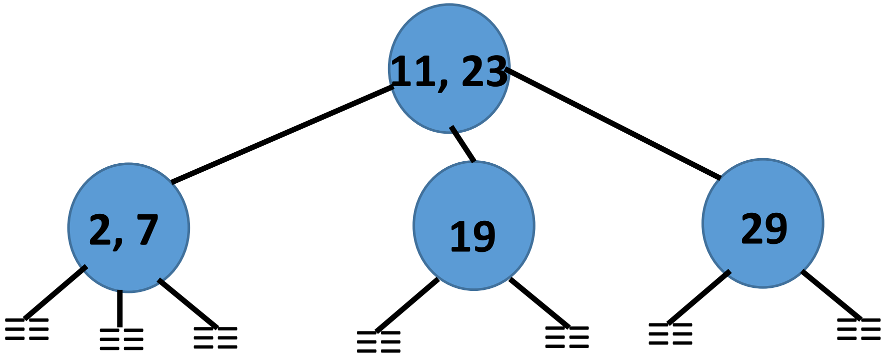


Inserting in a 2-3 tree

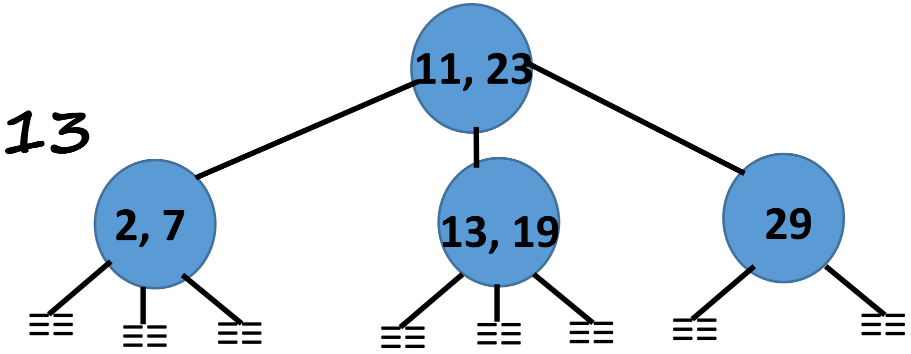
Construct a 2-3 tree for the set {7, 19, 11, 23, 29, 2, 13, 17, 21, 5, 3, 1, -1, 0}



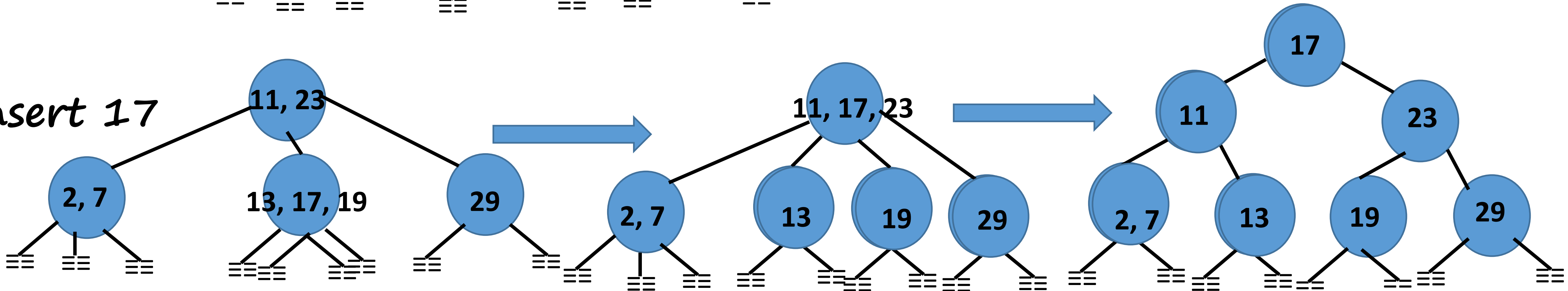
Insert 2



Insert 13

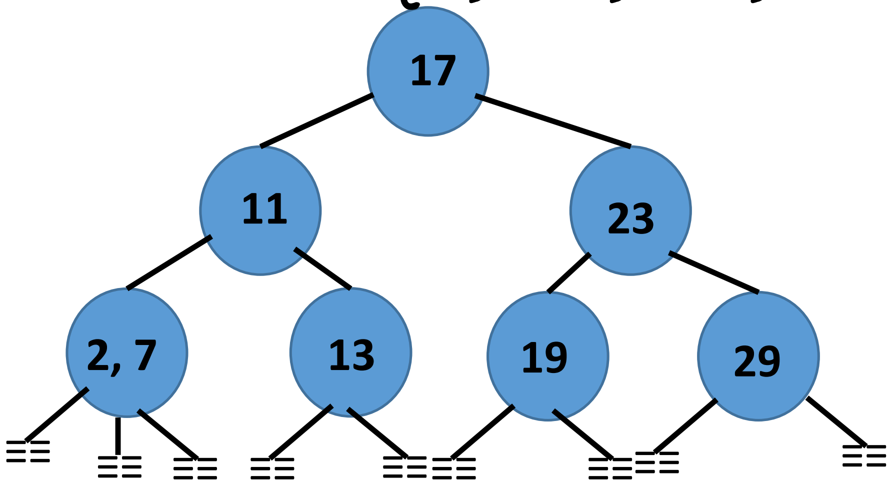


Insert 17

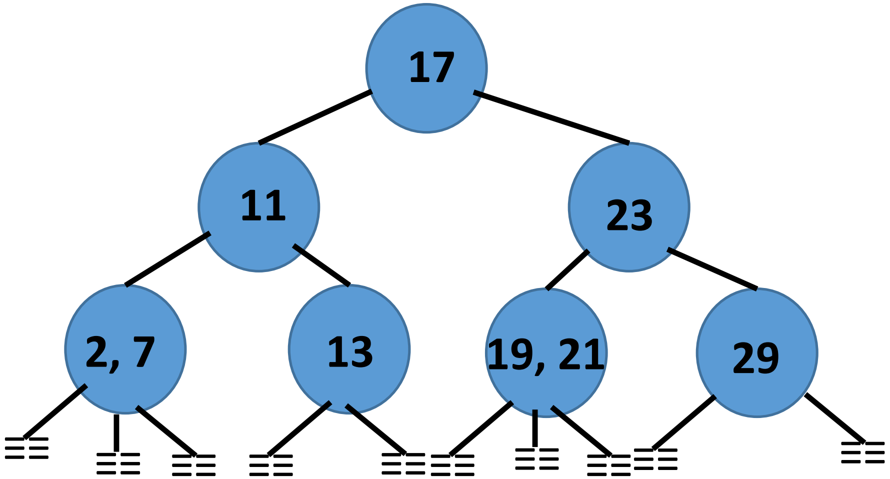


Inserting in a 2-3 tree

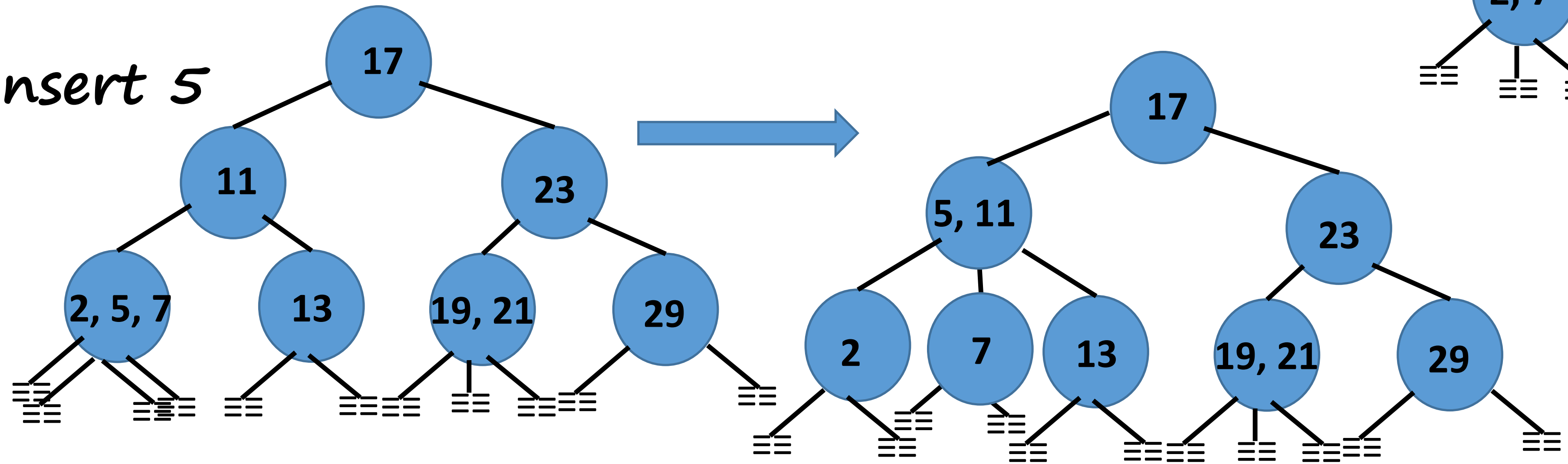
Construct a 2-3 tree for the set {7, 19, 11, 23, 29, 2, 13, 17, 21, 5, 3, 1, -1, 0}



Insert 21



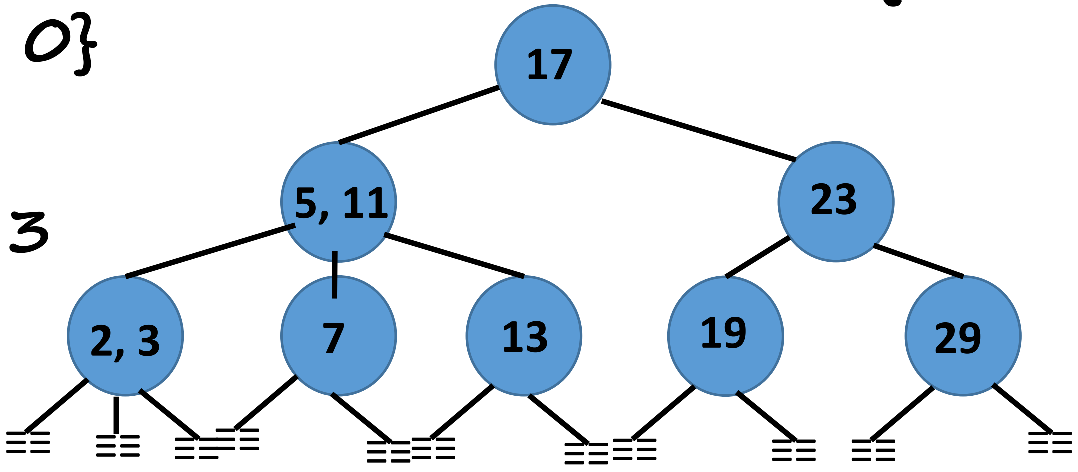
Insert 5



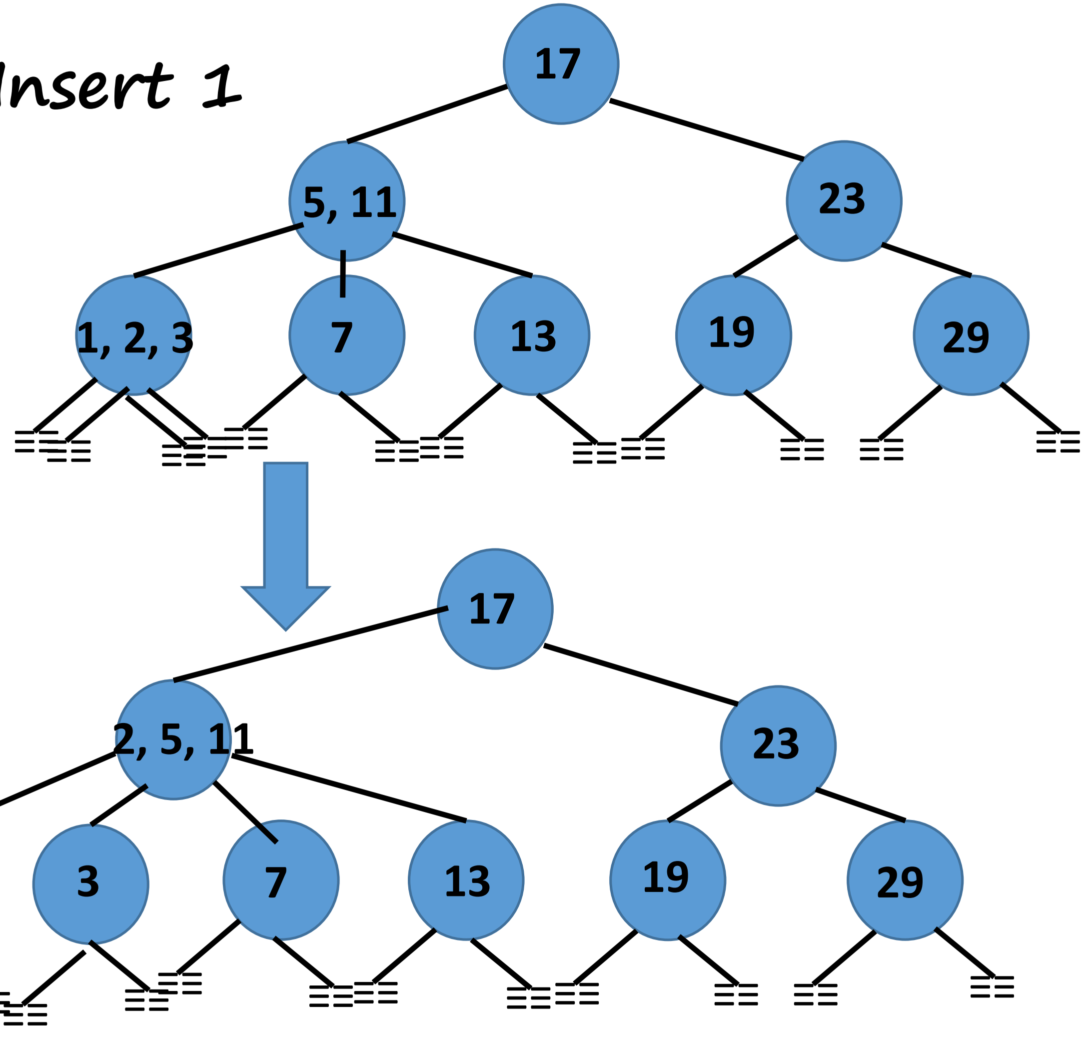
Inserting in a 2-3 tree

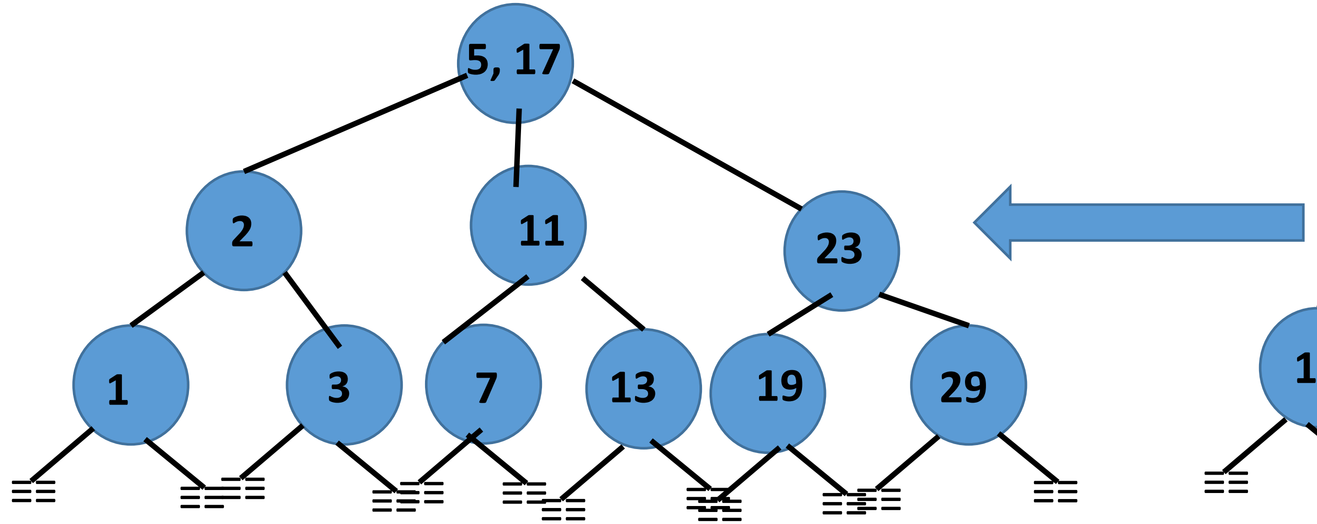
Construct a 2-3 tree for the set {7, 19, 11, 23, 29, 2, 13, 17, 21, 5, 3, 1, -1, 0}

Insert 3



Insert 1

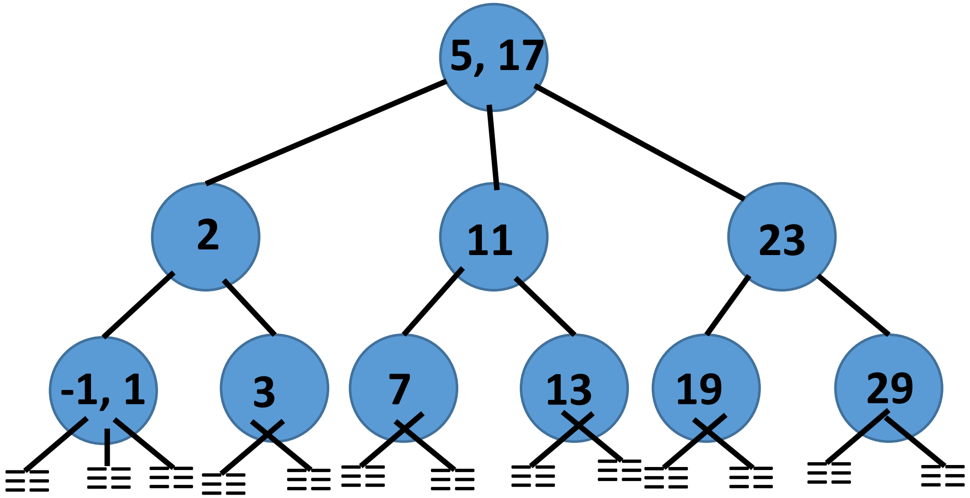




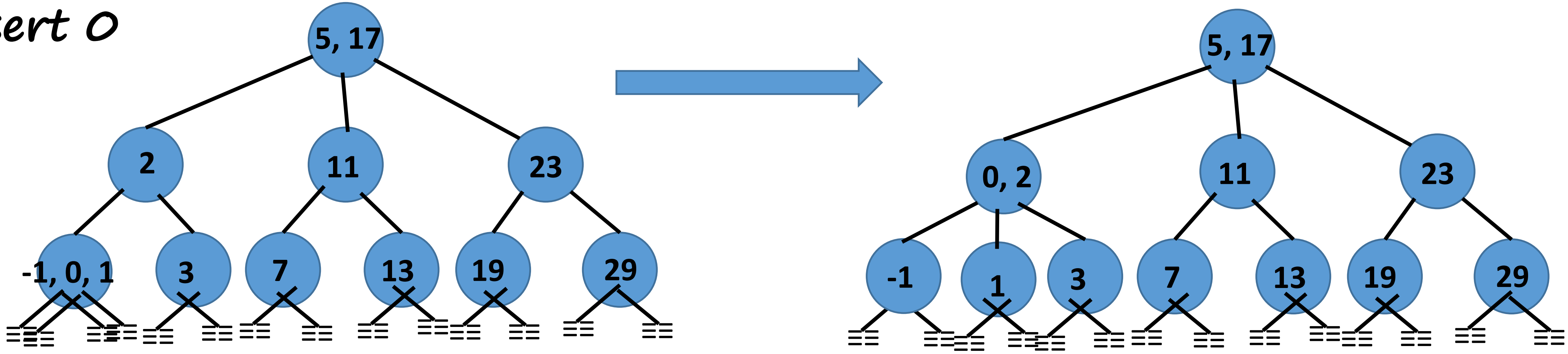
Inserting in a 2-3 tree

Construct a 2-3 tree for the set {7, 19, 11, 23, 29, 2, 13, 17, 21, 5, 3, 1, -1, 0}

Insert -1



Insert 0



Insertion: Process and Run-time

- To insert key β into a 2-node leaf x which has key α in it: $O(1)$
  - Make x into a 3-node (leaf).
  - Place β before or after α according as $\beta < \alpha$ or $\alpha < \beta$.

- Changes to 2-3 property, if any:
  - A 2-node leaf becomes a 3-node leaf.
  - There is no disturbance/changes made elsewhere on the tree, so every node is a 2-node or a 3-node.
  - There is no change to the height/level of any node; so the leaf nodes continue to remain at the same level.

Results in a 2-3 tree

Insertion: Process and Run-time

- To insert key β into a 3-node leaf x which has keys α, γ ($\alpha < \gamma$) in it:
  - Make x into a 4-node (leaf), temporarily.
  - Place β before α or between α and γ or after γ according as $\beta < \alpha$ or $\alpha < \beta$ and $\beta < \gamma$ or $\beta > \gamma$. Let the key arrangement be $a < b < c$
  - Push b into Parent[x] at an appropriate key position; create a new 2-node if Parent[x] DNE with b as its key.
  - Split leaf node x into two (leaf) 2-nodes x_1 and x_2 with a and c as the keys, respectively.
  - Make Parent[x] the parent of x_1 and x_2.
  - Repeat the splitting process up the tree, if necessary.
    - If the root node becomes a 4-node (temporarily) then grow the tree by one level.

Run-time for a single split = $O(1)$
\# of splits $\leq$ # on nodes on the search path $\leq 1 +$ height of the tree
Total run-time = $O(h) = O(\log n)$

Insertion: Process and Run-time

- Changes to 2-3 property, if any
  - The node which is temporarily made into a 4-node splits into two 2-nodes after pushing the middle key (in the 4-node) to its parent.
    - If there is no parent, then a new 2-node is created (as the parent node) and this becomes the root of the tree.
    - If the parent becomes a 4-node then it undergoes the splitting process.
    - Nodes on the search path which are temporarily a 4-node change into two 2-nodes and the parent of this node becomes a 3-node (or a 2-node).

      All the nodes are 2-nodes are 3-nodes.
    - The tree if it grows, due to splitting, then it grows at the root. So, the leaves remain at the same level.

      Results in a 2-3 tree

Insertion of a key into a 2-3 tree retains 2-3 property in $O(h) = O(\log n)$ time.

Exercise

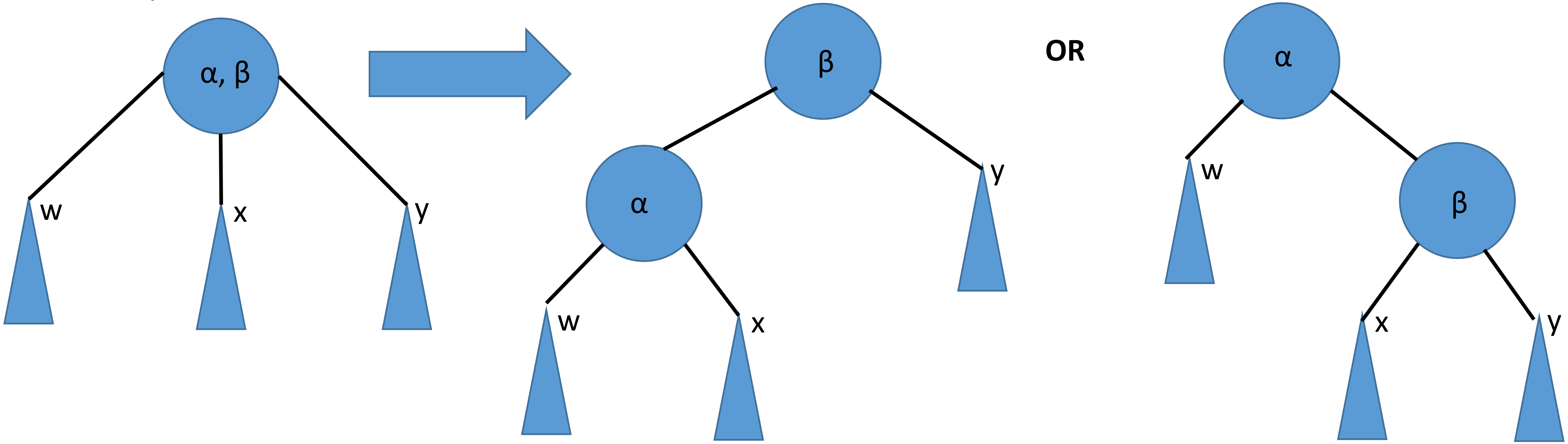
- Delete a key from a 3-node leaf.
- Delete a key from a 2-node leaf. [The parent looses a child in this case. How to retain the 2/3-nodeness of the parent?]
- Delete a key from a non-leaf 2-node. [Is the node itself lost? How to find new parent(s) for the orphaned children? How to retain the 2/3-nodeness of the new parent(s)? How to maintain the common level for the leaves?]
- Delete a key from a non-leaf 3-node. [Does the 3-node become a 2-node? How to merge its children so that it is a 2-node? How to maintain the common level for the leaves?]

  - In a BST, we always delete a leaf node (through replacement keys)!

Red-Black Trees: Motivation

Represent a 3-node using two 2-nodes so as to convert a 2-3 tree into an equivalent BST:

Idea-1: Replace a 3-node by two 2-nodes and making one of these as the parent of the other (and for one of the three children).

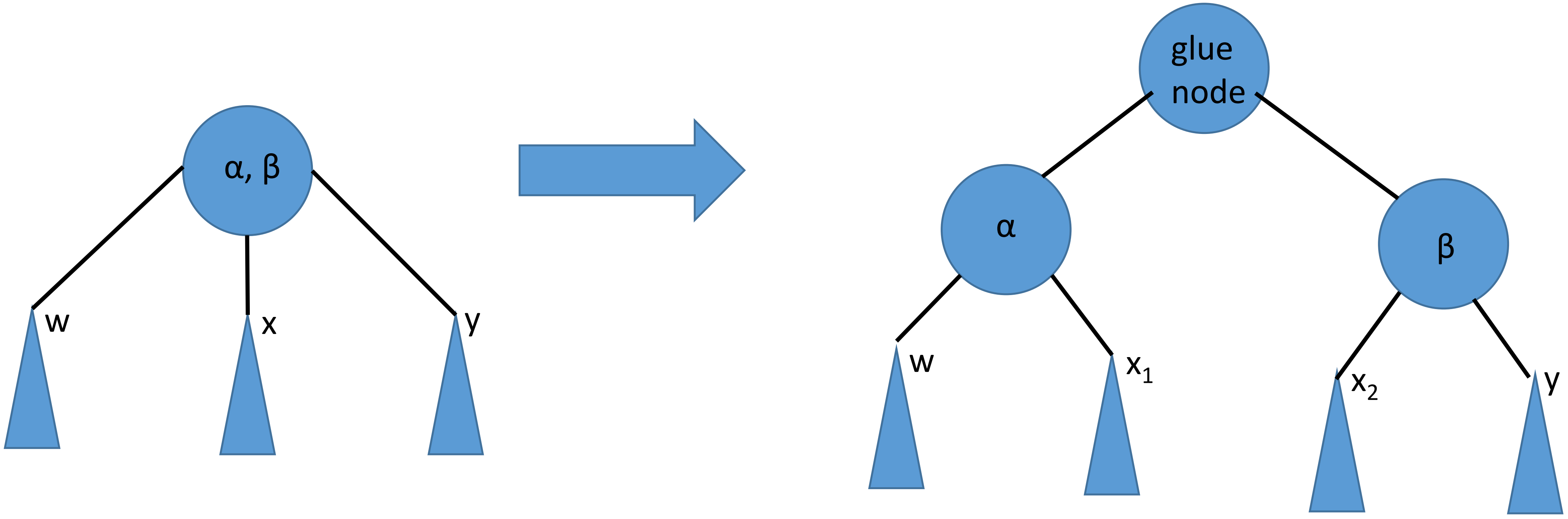


Demerit: Cannot (uniquely) map the BST to a 2-3 tree.

Red-Black Trees: Motivation

Represent a 3-node using two 2-nodes so as to convert a 2-3 tree into an equivalent BST:

Idea-2: Split the 3-node into two 2-nodes and use a "glue" node.

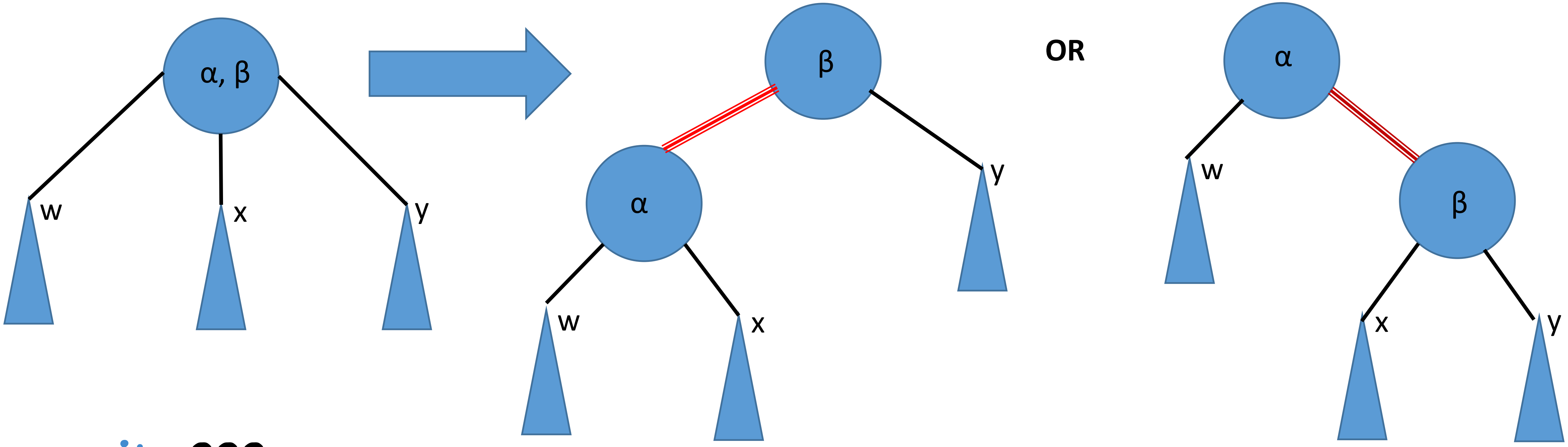


Demerit: Messy code, sometime the glue node may be a dummy node.

Red-Black Trees: Motivation

Represent a 3-node using two 2-nodes so as to convert a 2-3 tree into an equivalent BST:

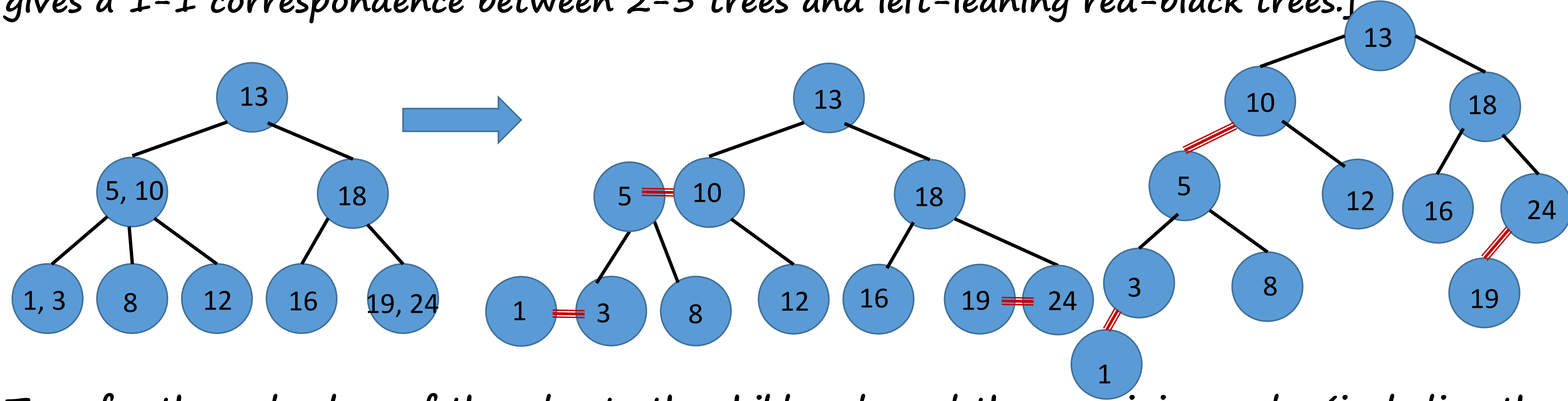
Idea-3: Replace a 3-node by two 2-nodes and making one of these as the parent of the other (and for one of the three children) and assign (red) colour to the glue edge.



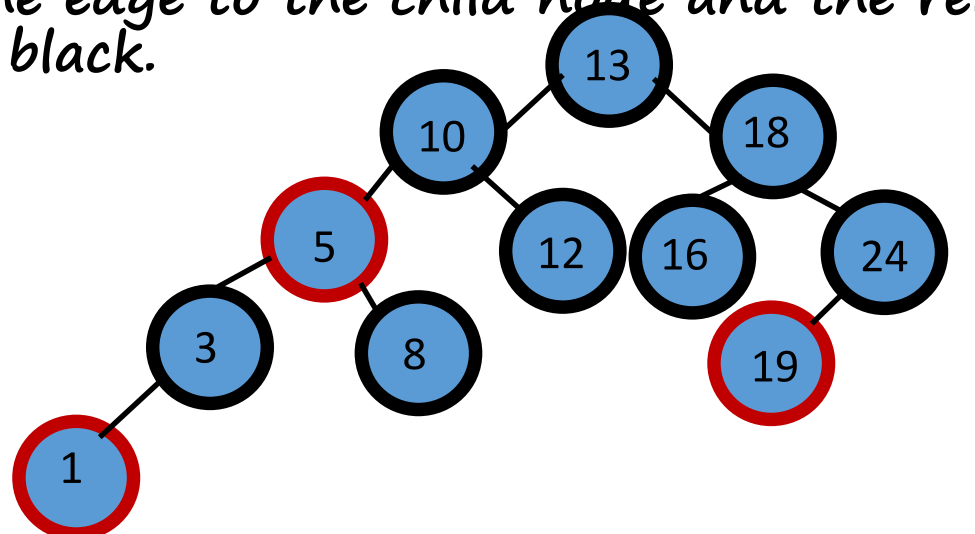
Demerit: ???

Left-leaning Red-Black Trees

Replace a 3-node by two 2-nodes and a glue edge such that the red links lean left. [This gives a 1-1 correspondence between 2-3 trees and left-leaning red-black trees.]

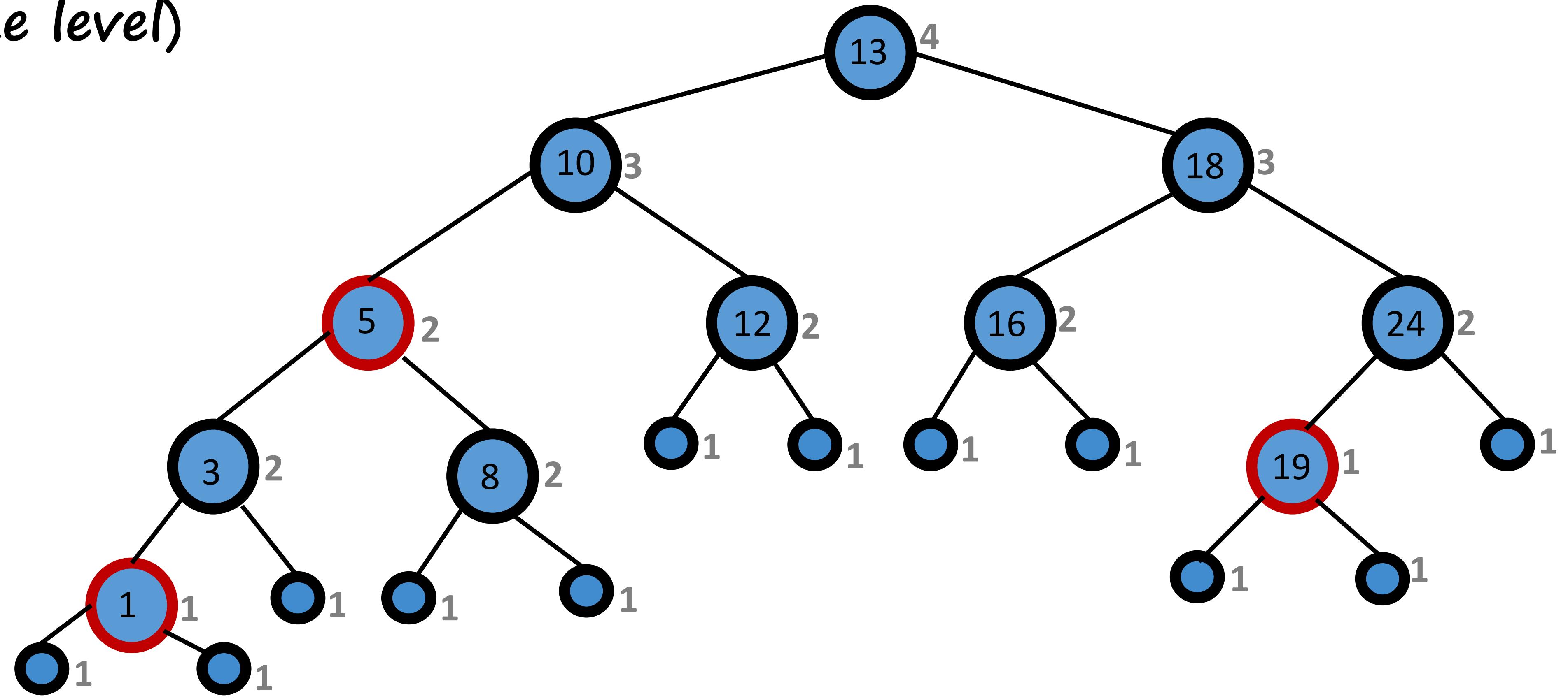


Transfer the red colour of the edge to the child node and the remaining nodes (including the sentinels/nulls) are coloured black.



Left-leaning Red-Black Trees

- A BST
- No two red nodes are adjacent (no node has two red links on it)
- Red nodes are left-child of their parent (red links lean left)
- Every path from a node to its descendant sentinel node has same number of black nodes; i.e., has same black height (leaves in a 2-3 tree are at the same level)



Operations on a Red-Black Tree

- Since the black height of a node along each child remains the same, the height of a red-black tree is O(log n).
- The black height of a node can be augmented to the BST.
- The colours on nodes can be augmented using one (additional) bit per node of a BST.

- Search – As in a BST. $O(h) = O(\log n)$

- Insertion – The splitting of a 3-node in a 2-3 tree translates into rotation or colour flip. $O(h) = O(\log n)$

- Deletion – Requires rebalancing of black height (through rotations) and/or colour flip. $O(h) = O(\log n)$