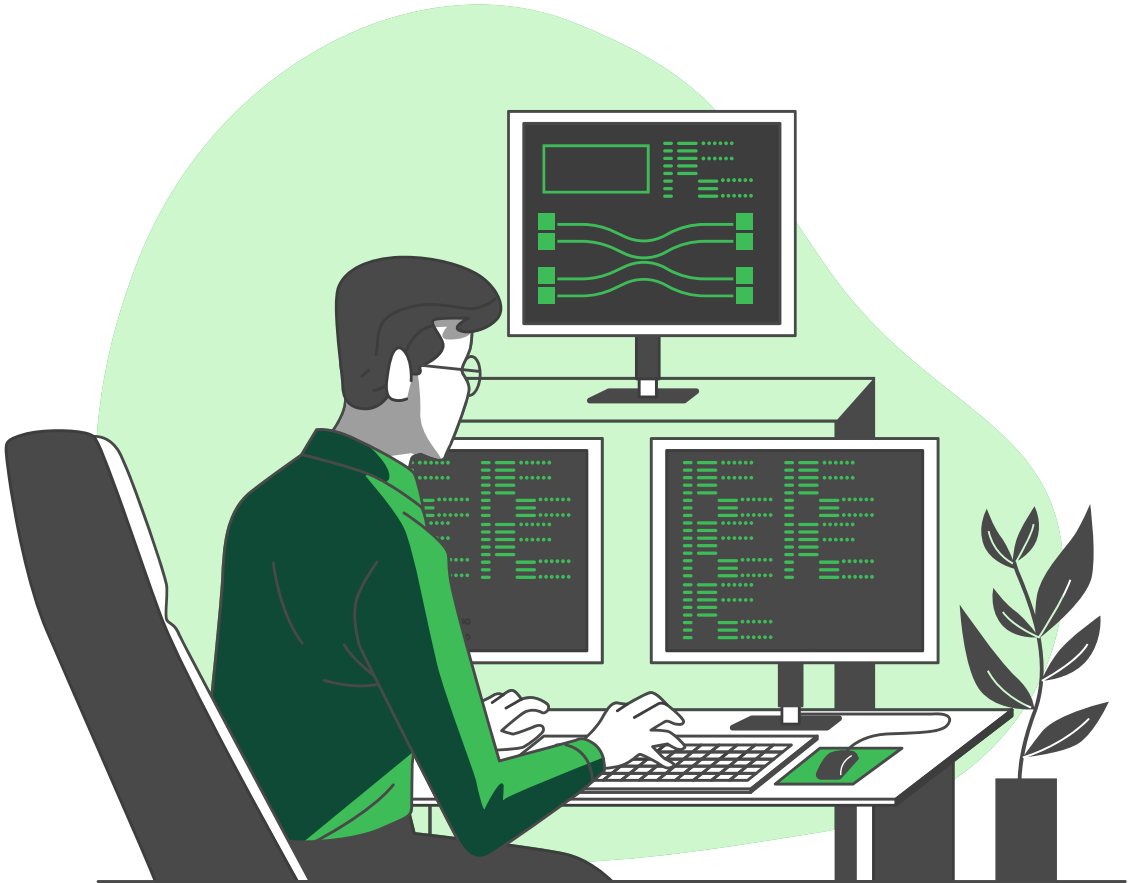
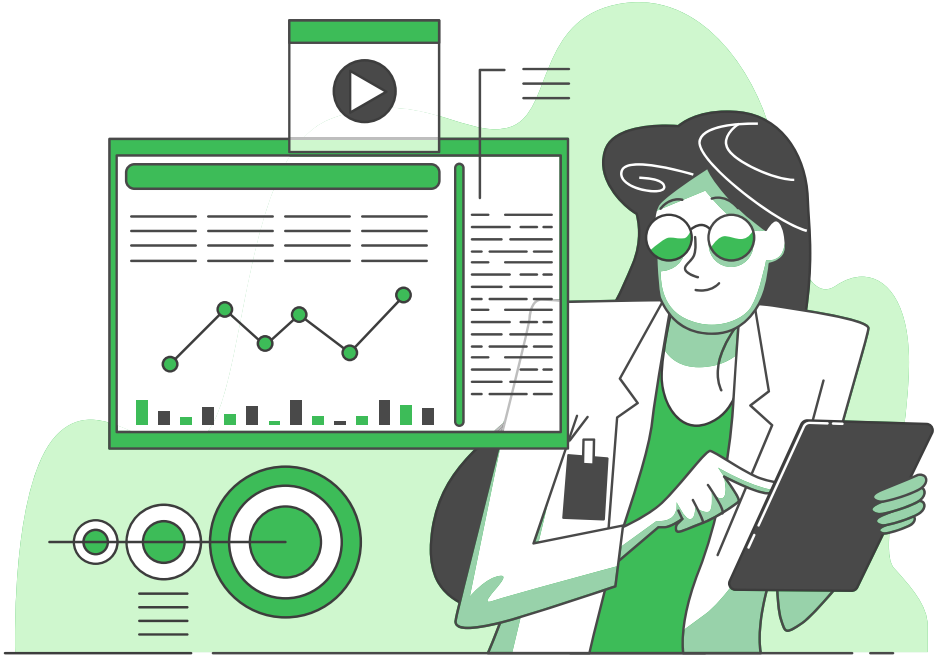


B-Tree

CS 251 - Data Structures and Algorithms

Note:
Slides complement the discussion in class

Table of Contents



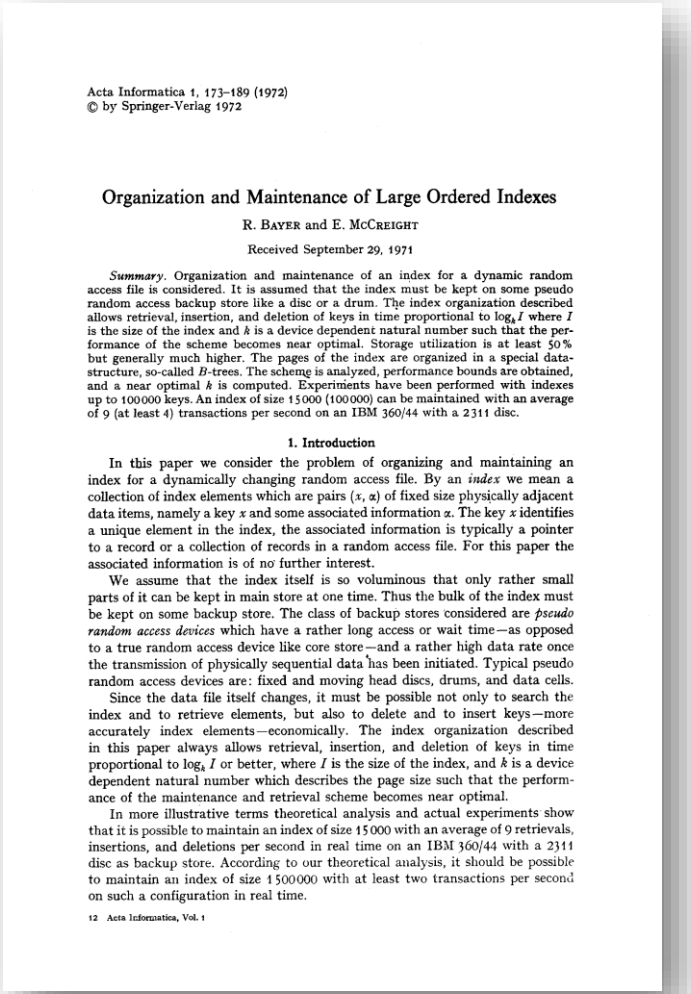
01

B-Tree

Enforcing balance

Bayer, R., & McCreight, E. M. (1972). Organization and Maintenance of Large Ordered Indexes. Acta Informatica, 1(3), 173–189.

https://link.springer.com/article/10.1007/BF00288683



Acta Informatica 1, 173–189 (1972)

Organization and Maintenance of Large Ordered Indexes

R. Bayer and E. McCreight

Received September 29, 1971

Summary. Organization and maintenance of an index for a dynamic random access file is considered. It is assumed that the index must be kept on some pseudo random access backup store like a disc or a drum. The index organization described allows retrieval, insertion, and deletion of keys in time proportional to $\log_k I$ where I is the size of the index and k is a device dependent natural number such that the performance of the scheme becomes near optimal. Storage utilization is at least 50% but generally much higher. The pages of the index are organized in a special data-structure, so-called B-trees. The scheme is analyzed, performance bounds are obtained, and a near optimal k is computed. Experiments have been performed with indexes up to 100000 keys. An index of size 15000 (100000) can be maintained with an average of 9 (at least 4) transactions per second on an IBM 360/44 with a 2311 disc.

1. Introduction

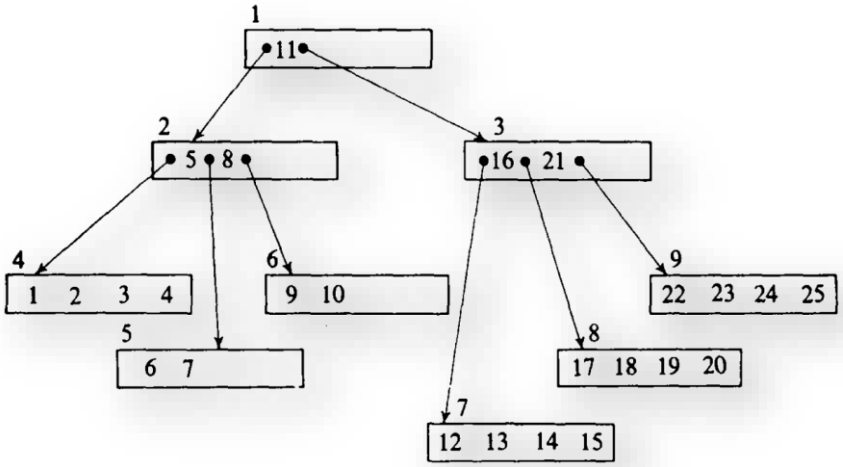
In this paper we consider the problem of organizing and maintaining an index for a dynamically changing random access file. By an *index* we mean a collection of index elements which are pairs (x, α) of fixed size physically adjacent data items, namely a key x and some associated information α. The key x identifies a unique element in the index, the associated information is typically a pointer to a record or a collection of records in a random access file. For this paper the associated information is of no further interest.

We assume that the index itself is so voluminous that only rather small parts of it can be kept in main store at one time. Thus the bulk of the index must be kept on some backup store. The class of backup stores considered are *pseudo random access devices* which have a rather long access or wait time—as opposed to a true random access device like core store—and a rather high data rate once the transmission of physically sequential data has been initiated. Typical pseudo random access devices are: fixed and moving head discs, drums, and data cells.

Since the data file itself changes, it must be possible not only to search the index and to retrieve elements, but also to delete and to insert keys—more accurately index elements—economically. The index organization described in this paper always allows retrieval, insertion, and deletion of keys in time proportional to $\log_k I$ or better, where I is the size of the index, and k is a device dependent natural number which describes the page size such that the performance of the maintenance and retrieval scheme becomes near optimal.

In more illustrative terms theoretical analysis and actual experiments show that it is possible to maintain an index of size 15000 with an average of 9 retrievals, insertions, and deletions per second in real time on an IBM 360/44 with a 2311 disc as backup store. According to our theoretical analysis, it should be possible to maintain an index of size 1500000 with at least two transactions per second on such a configuration in real time.

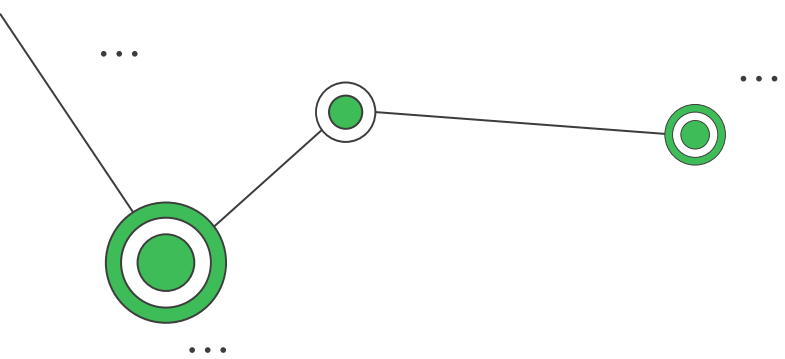
B-Tree



Bayer, R., & McCreight, E. M. (1972). Organization and Maintenance of Large Ordered Indexes. Acta Informatica, 1(3), 173–189.

Developed to meet the growing need for data structures that could efficiently manage large datasets on disk.

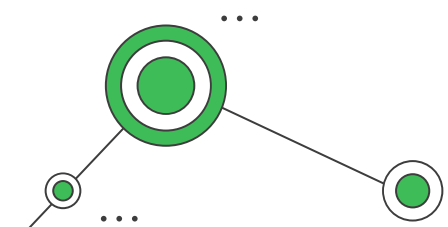
B-trees maintain their balance through a dynamic process of node splitting and redistribution, ensuring that operations like insertion, deletion, and searching can be performed with logarithmic time complexity



B-Tree Properties

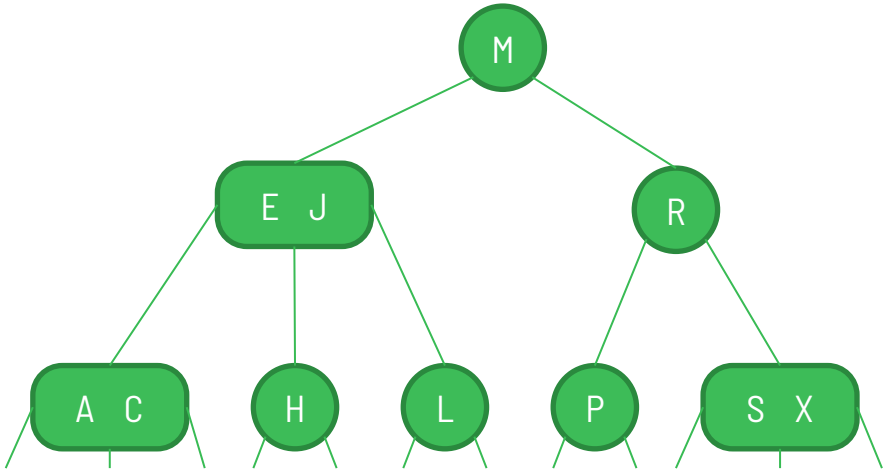
These properties follow Knuth's definition for B-Trees of order m:

- Every node has at most m children.
- Every internal node has at least $\left\lceil \frac{m}{2} \right\rceil$ children.
- The root node has at least two children unless it is a leaf.
- All leaves appear on the same level.
- A non-leaf node with k children contains $k - 1$ keys.



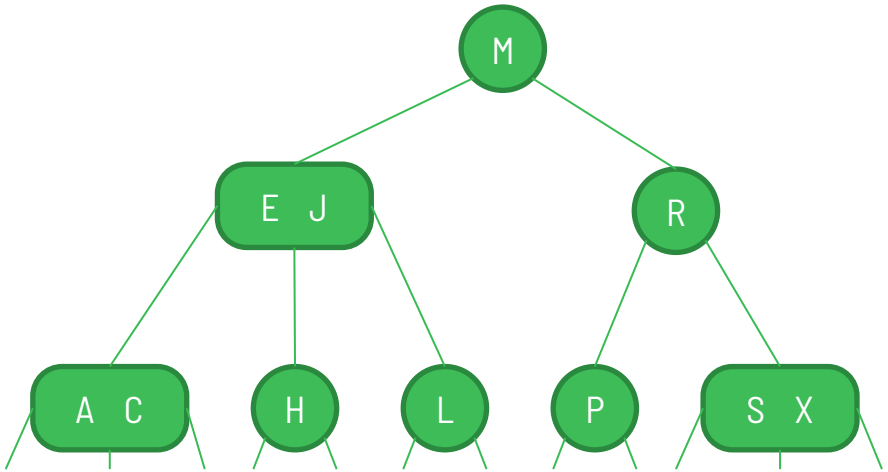
Knuth, Donald (1998). Sorting and Searching. The Art of Computer Programming. Vol. 3 (Second ed.). Addison-Wesley. ISBN 0-201-89685-0. Section 6.2.4: Multiway Trees, pp. 481–491.

2-3 Tree (B-Tree of order 3)

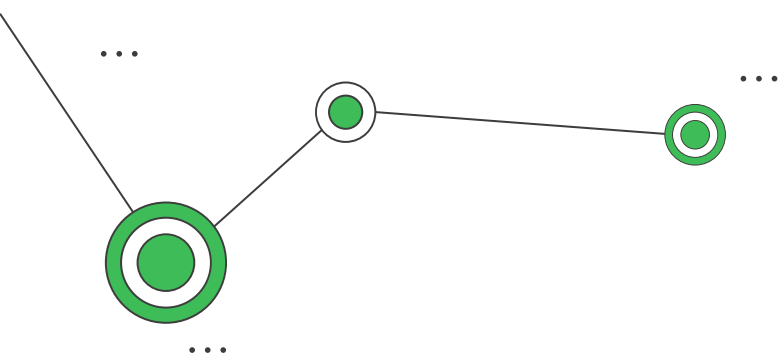


- Key idea: A flexible tree for maintaining balance done less expensively than with regular binary search trees.
- A 2-3 tree allows nodes to have: 1 key and 2 links (2-node) or 2 keys and tree links (3-node).

2-3 Tree Properties

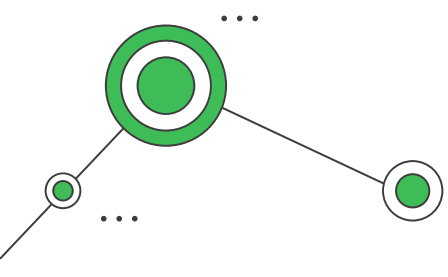


- Maintains order.
- Every null link is at the same distance from the root.
- Q: What is the height of the tree given n nodes?
  A: Between $\log_3(n)$ and $\log_2(n)$.



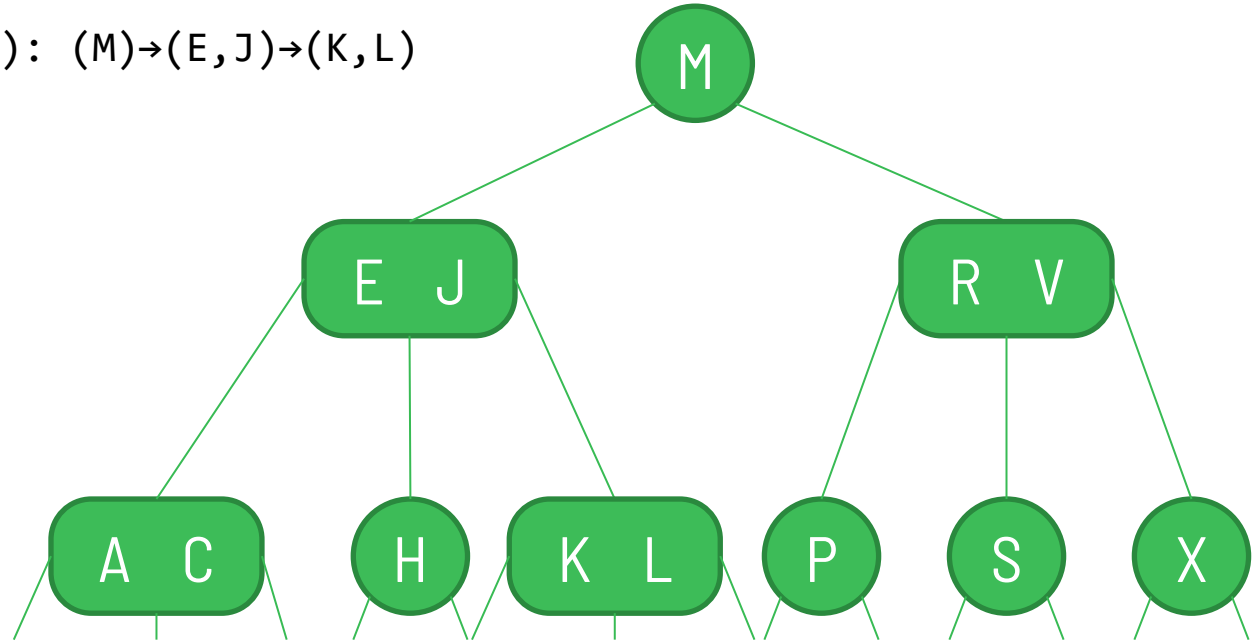
B-Tree Search

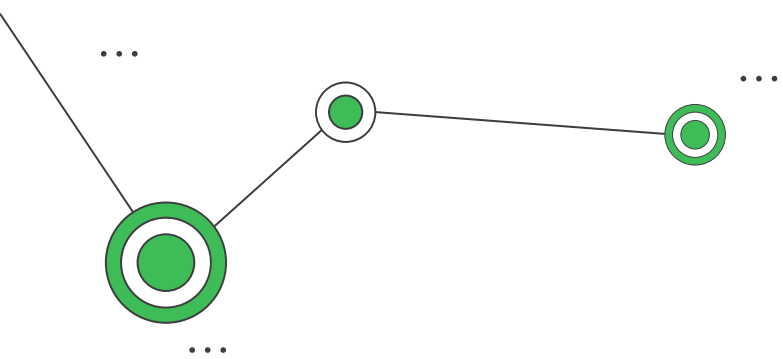
```
algorithm search(root:node, x:item) → node
   i ← 0
   while i < root.nitems and x > root.item[i] do
      i ← i + 1
   end while

   if i < root.nitems and x = root.item[i] then
      return (root, i)
   end if

   if root is a leaf then
      return null
   end if

   return search(root.children[i], x)
end algorithm
```



Search

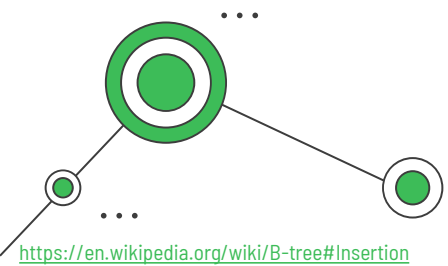
Search(L): (M)→(E,J)→(K,L)



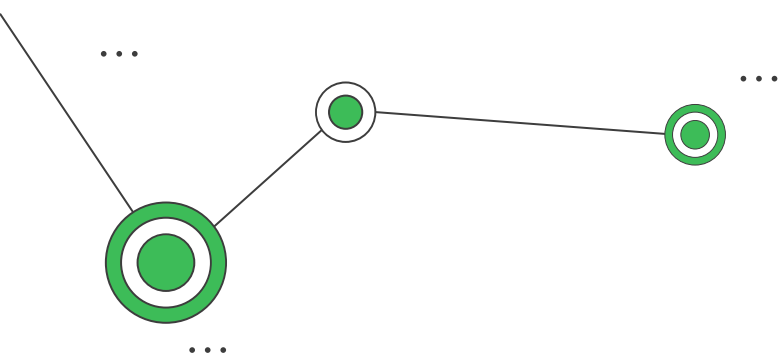


B-Tree Insert Insights

- Search for the leaf that might contain the new item.
- If the node contains fewer than the maximum allowed number of items, then there is room for the new item. Insert the new item in the node, keeping the node's items ordered.
- Otherwise, the node is full, evenly split it into two nodes.

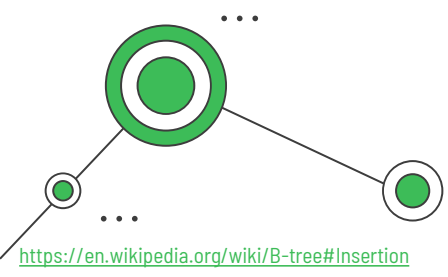


https://en.wikipedia.org/wiki/B-tree#Insertion



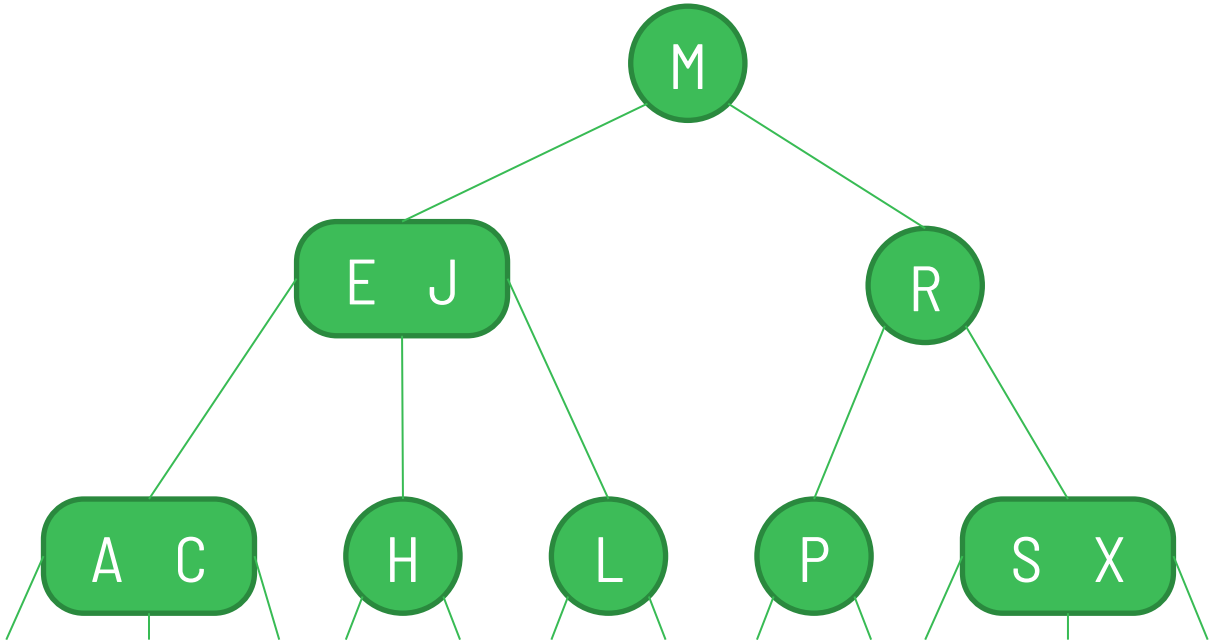
Node Split

- Chose a median item from among the leaf's items and the new item that is being inserted.
- Items less than the median are put in the new left node and items greater than the median are put in the new right node, with the median acting as a separation value.
- The separation value is inserted in the node's parent, which may cause it to be split, and so on. If the node has no parent (i.e., the node was the root), create a new root above this node (increasing the height of the tree).

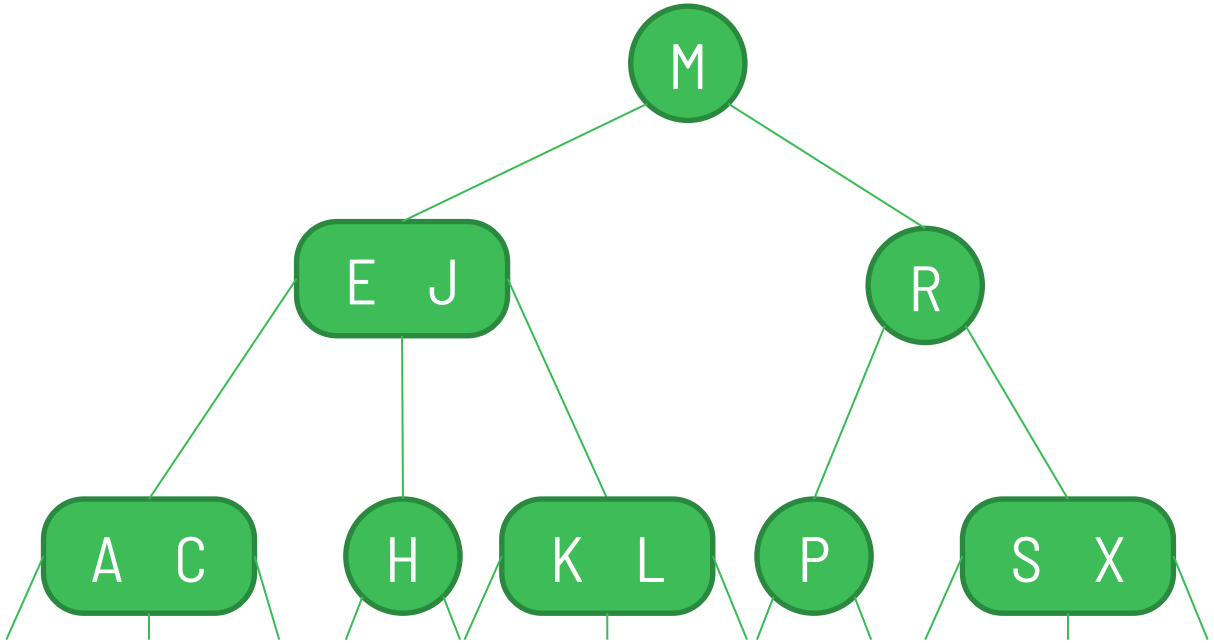


https://en.wikipedia.org/wiki/B-tree#Insertion

Insert(K) in a 2-3 Tree

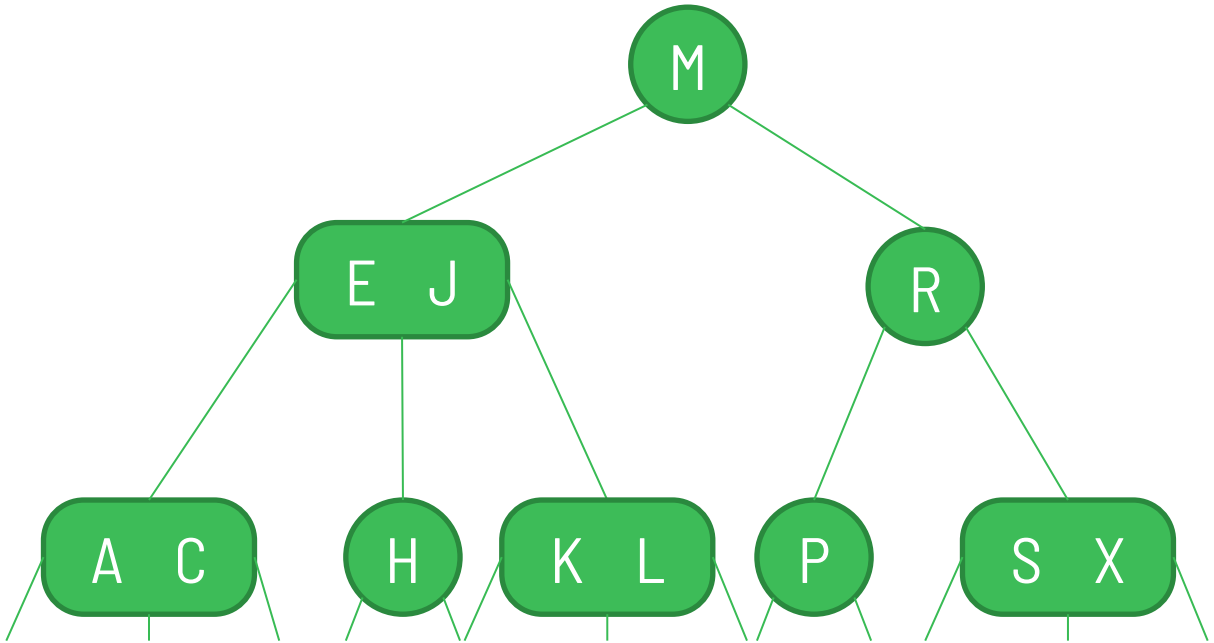


Insert(K) in a 2-3 Tree

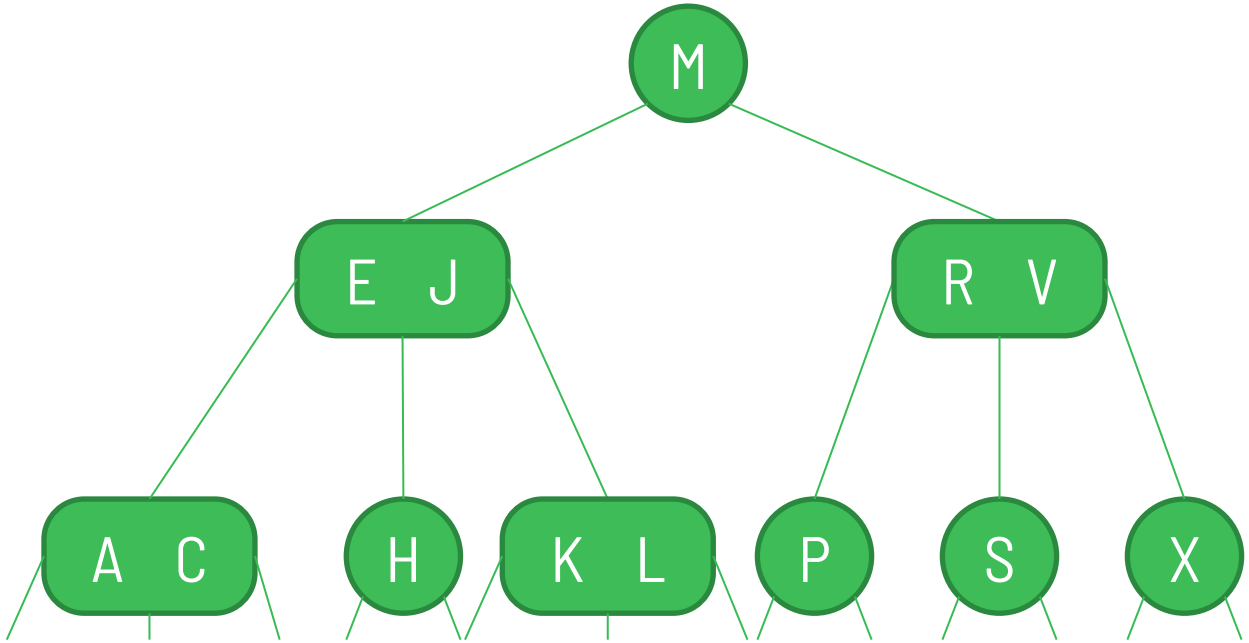


Split a 4-Node

Insert(V) in a 2-3 Tree



Insert(V) in a 2-3 Tree

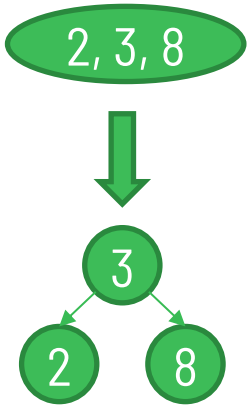


Insert in a 2-3 Tree: 8, 2, 3, 4, 0, 9, 6, 5, 1

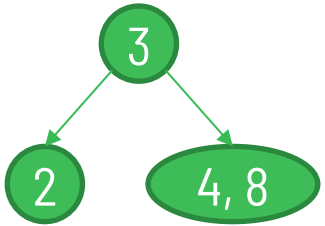
8:

2:

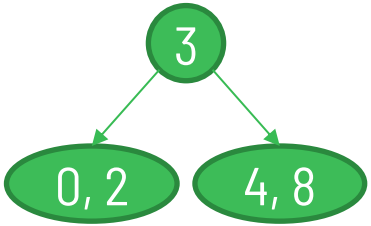
3:



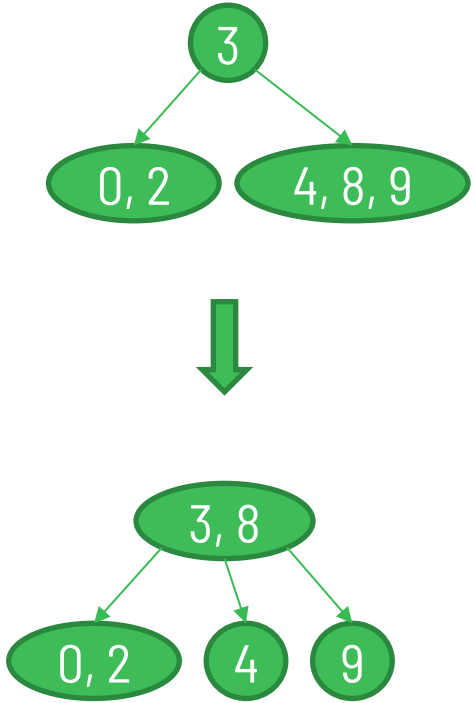
4:



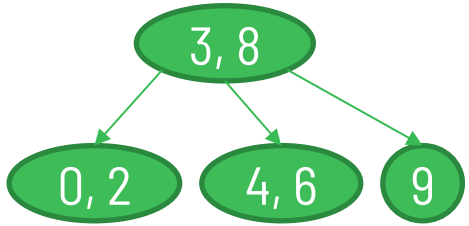
0:



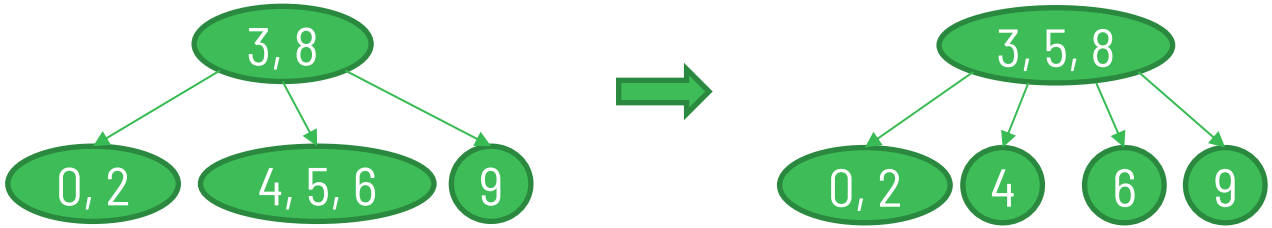
9:

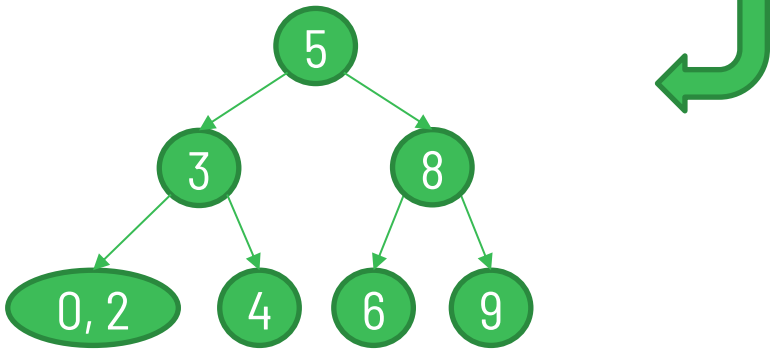


6:

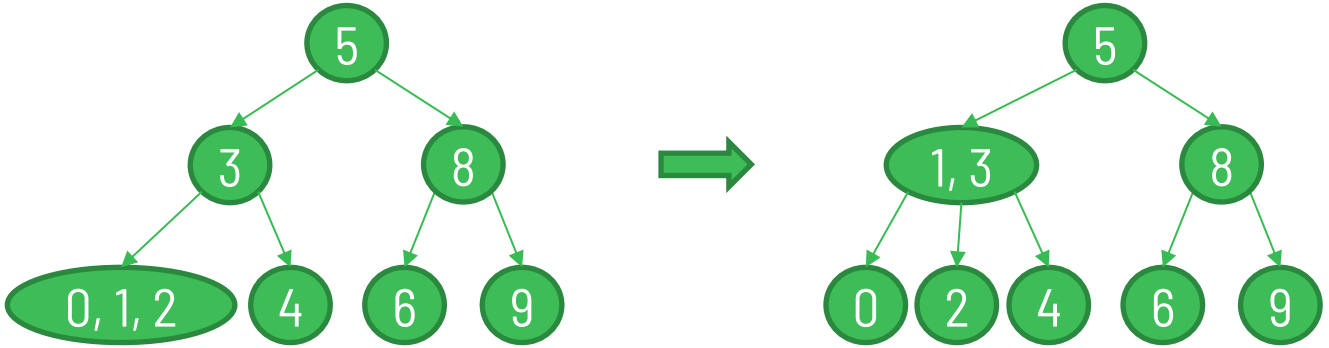


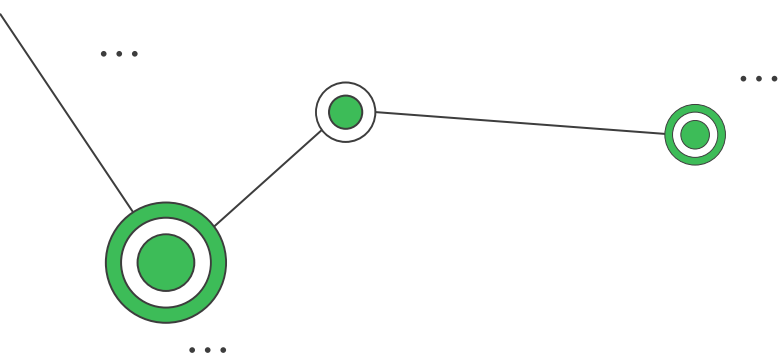
5:





1:





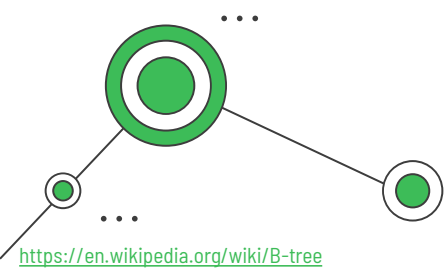
B-Tree Delete Insights

Strategy: Locate and delete the item, then restructure the tree to retain its invariants.

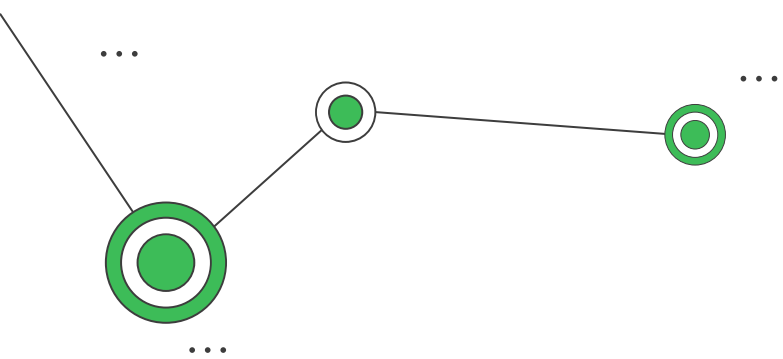
Ideal: Delete from a leaf with more than an item.

Important:

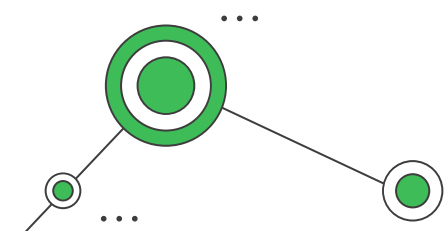
1. An item in an internal node is a separator for its child nodes.
2. Deleting an item may put its node under the minimum number of elements and children.



https://en.wikipedia.org/wiki/B-tree



B-Tree Delete Cases



Delete from a leaf:

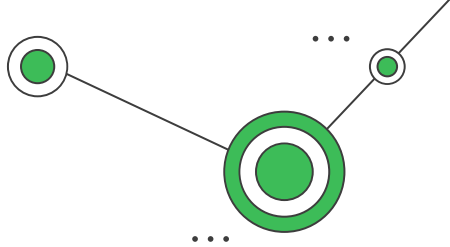
- Search for the value to delete.
- If the value is in a leaf node, simply delete it from the node.
- If underflow happens, rebalance the tree.

Delete from an internal node:

- Choose a new separator, remove it from the leaf node it is in, and replace the item to be deleted with the new separator.
- The previous step deleted an item from a leaf node. If that leaf node is now deficient (has fewer than the required number of nodes), then **rebalance** the tree starting from the leaf node.

https://en.wikipedia.org/wiki/B-tree

B-Tree Rebalancing (1)

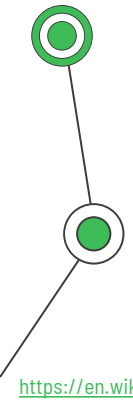


If the deficient node's right sibling exists and has more than the minimum number of items, then rotate left:

1. Copy the separator from the parent to the end of the deficient node (the separator moves down; the deficient node now has the minimum number of items)
2. Replace the separator in the parent with the first item of the right sibling (right sibling loses one node but still has at least the minimum number of items)
3. The tree is now balanced

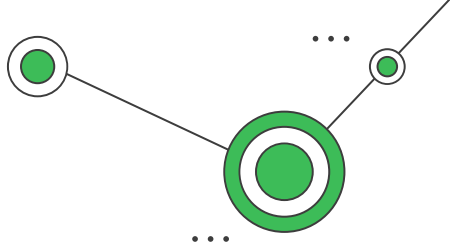
Otherwise, if the deficient node's left sibling exists and has more than the minimum number of items, then rotate right:

1. Copy the separator from the parent to the start of the deficient node (the separator moves down; deficient node now has the minimum number of items)
2. Replace the separator in the parent with the last element of the left sibling (left sibling loses one node but still has at least the minimum number of items)
3. The tree is now balanced



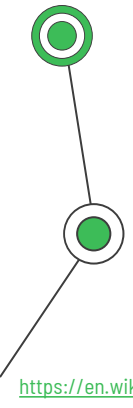
https://en.wikipedia.org/wiki/B-tree

B-Tree Rebalancing (2)



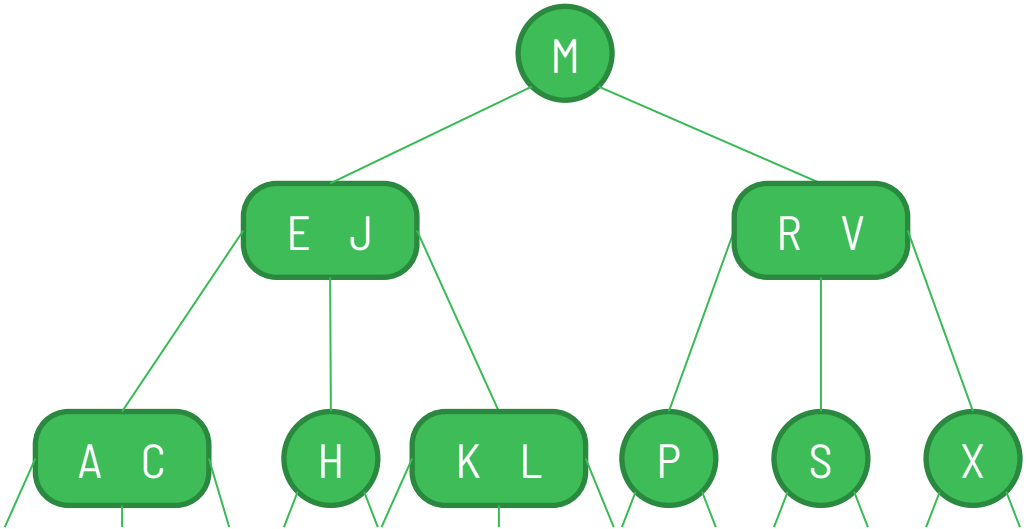
Otherwise, if both immediate siblings have only the minimum number of items, then merge with a sibling sandwiching their separator taken off from their parent.

1. Copy the separator to the end of the left node (the left node may be the deficient node or it may be the sibling with the minimum number of items)
2. Move all items from the right node to the left node (the left node now has the maximum number of items, and the right node – empty)
3. Remove the separator from the parent along with its empty right child (the parent loses an item)
    a. If the parent is the root and now has no items, then free it and make the merged node the new root (tree becomes shallower)
    b. Otherwise, if the parent has fewer than the required number of items, then rebalance the parent



https://en.wikipedia.org/wiki/B-tree

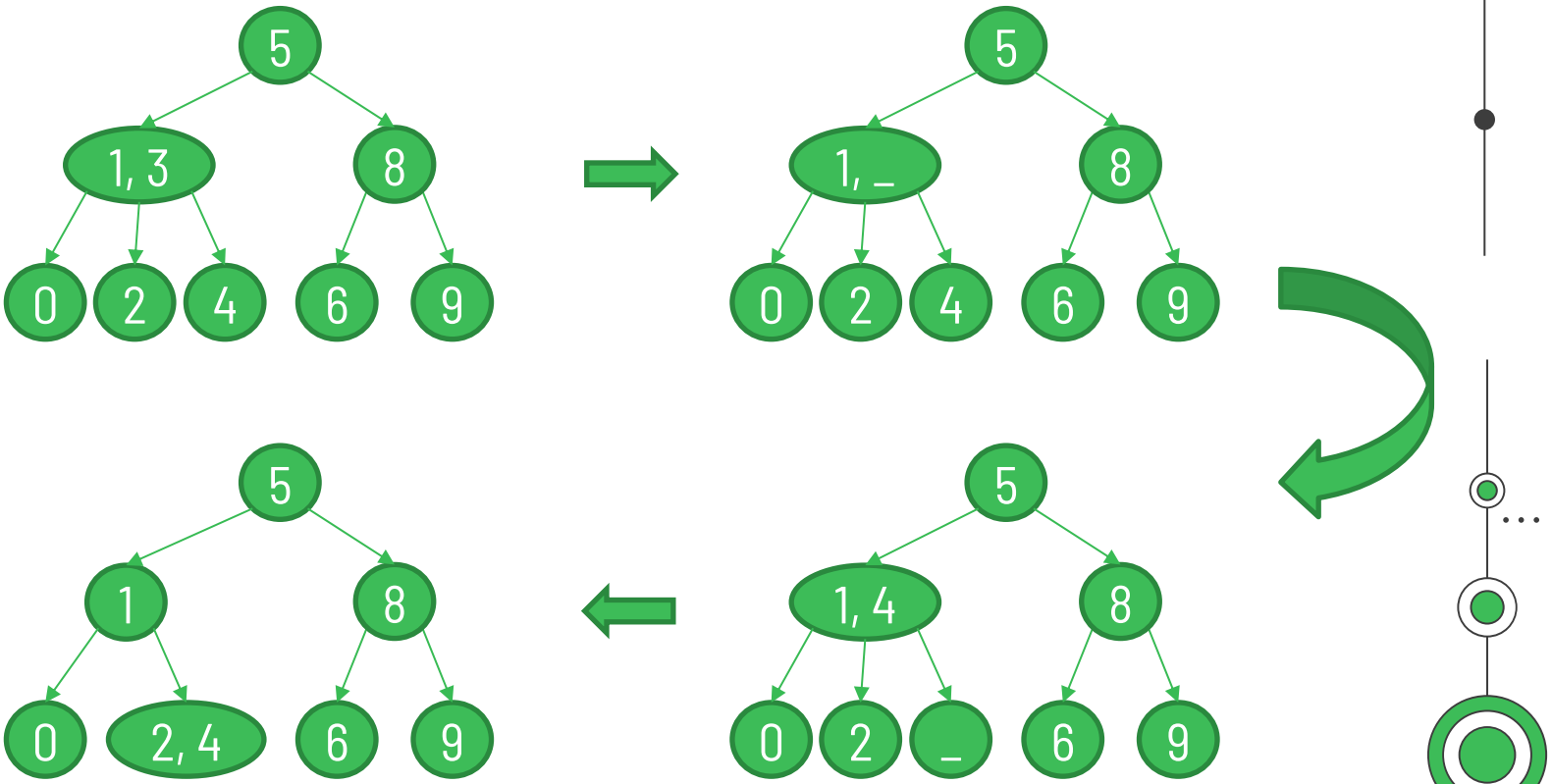
Delete in 2-3 Tree



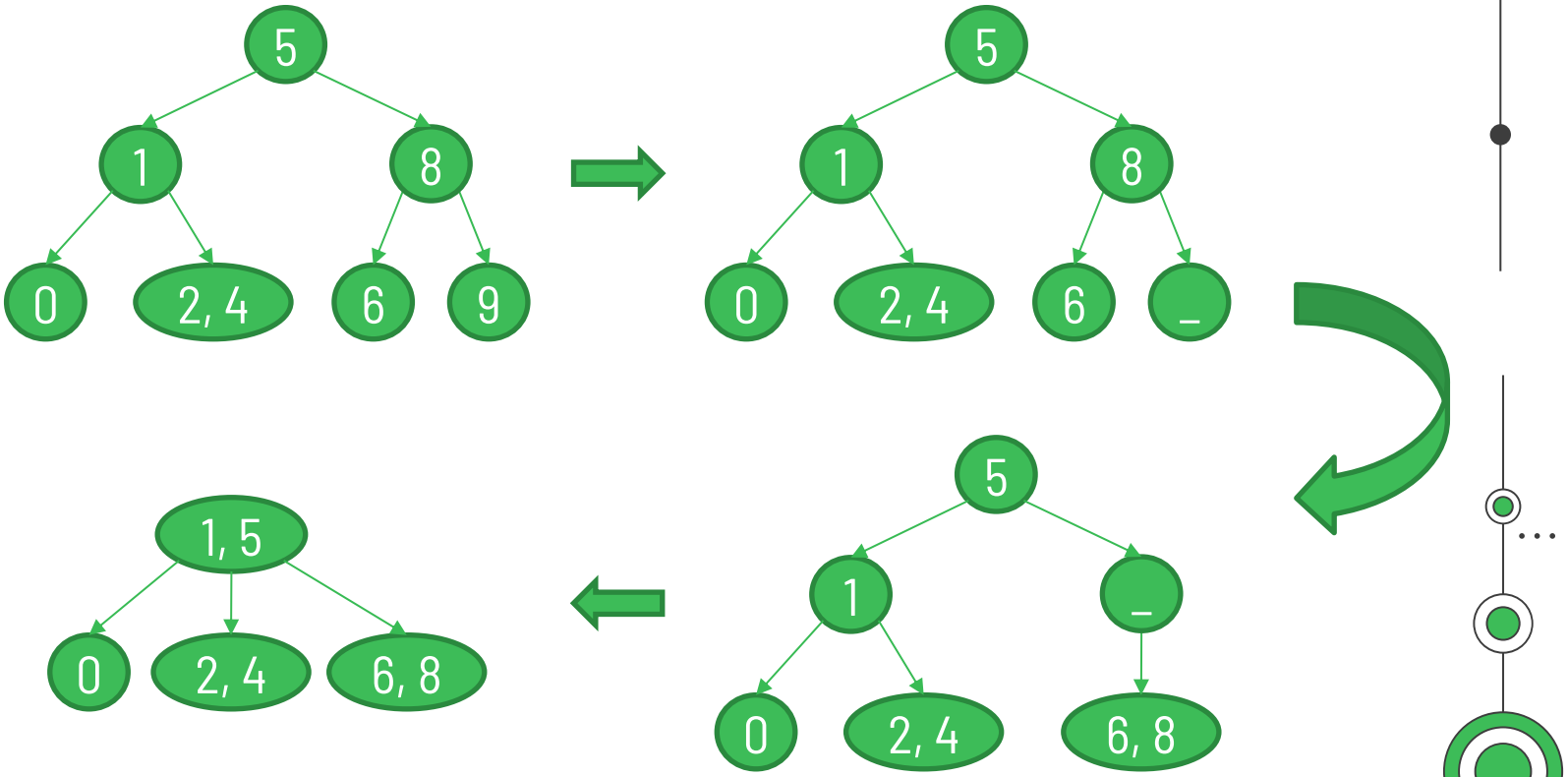
Easy to delete a key from a 3-node or a 4-node at the bottom of the tree. It is tricky to delete from a 2-node.

Idea: Use the inorder predecessor or successor (if any) to replace a deleted item. Merge subtrees accordingly.

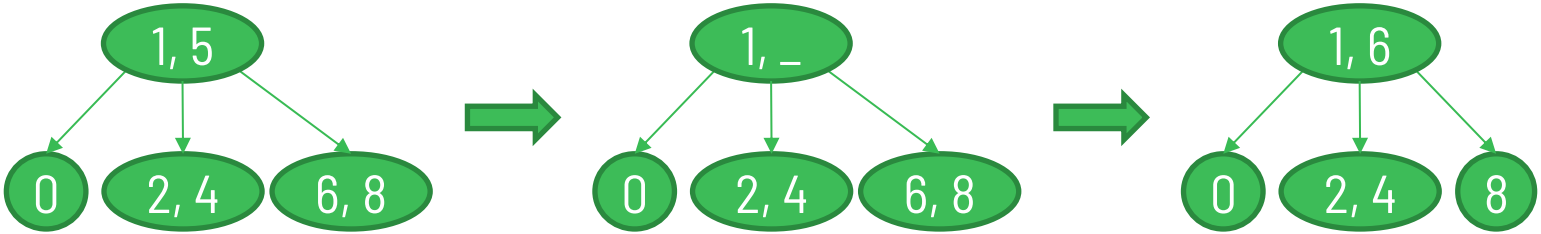
Delete 3:

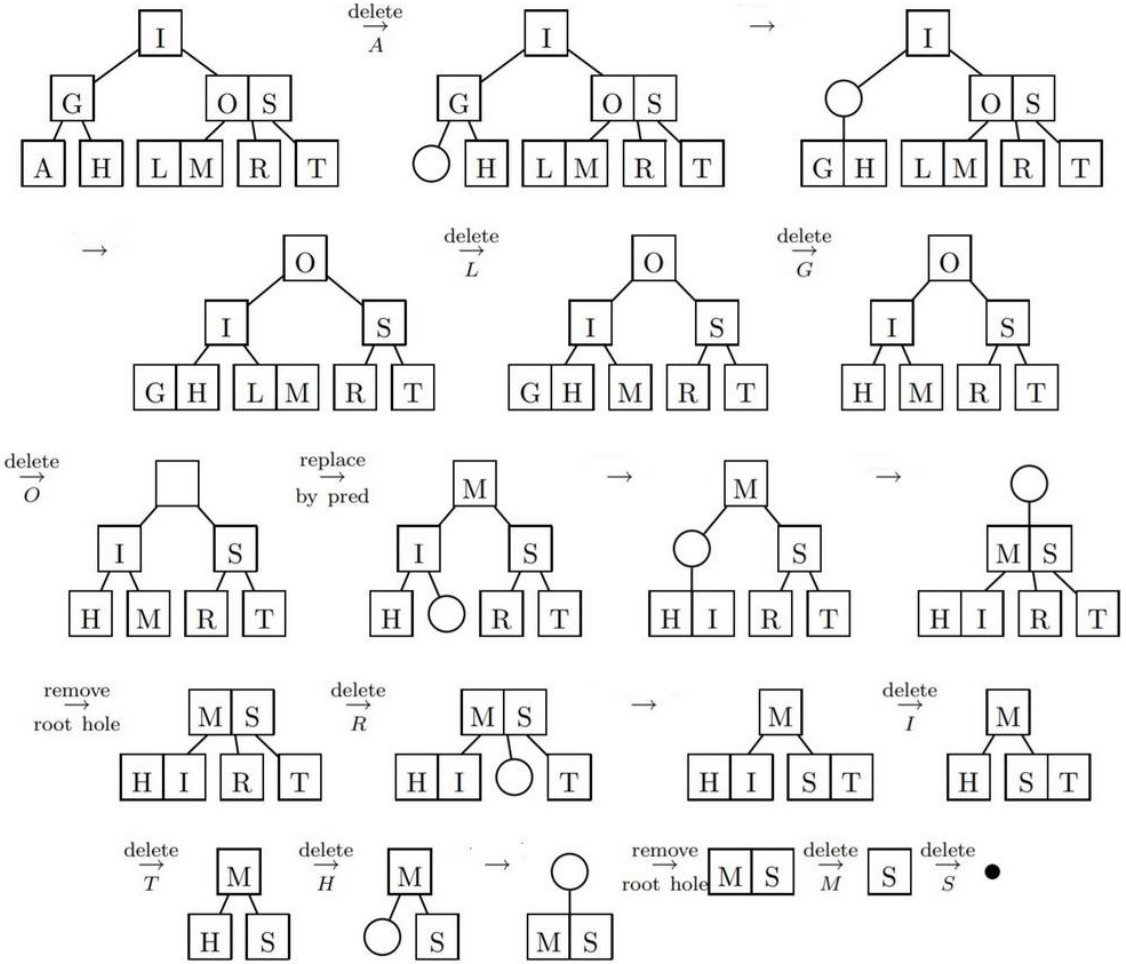


Delete 9:



Delete 5:





https://iq.opengenus.org/2-3-trees/

One Tree Less!

Do you have any questions?