fit@hcmus

Data Structures and Algorithms

2-3 Trees
2-3-4 Trees

Lecturer: Le Ngoc Thanh
Email: lnthanh@fit.hcmus.edu.vn

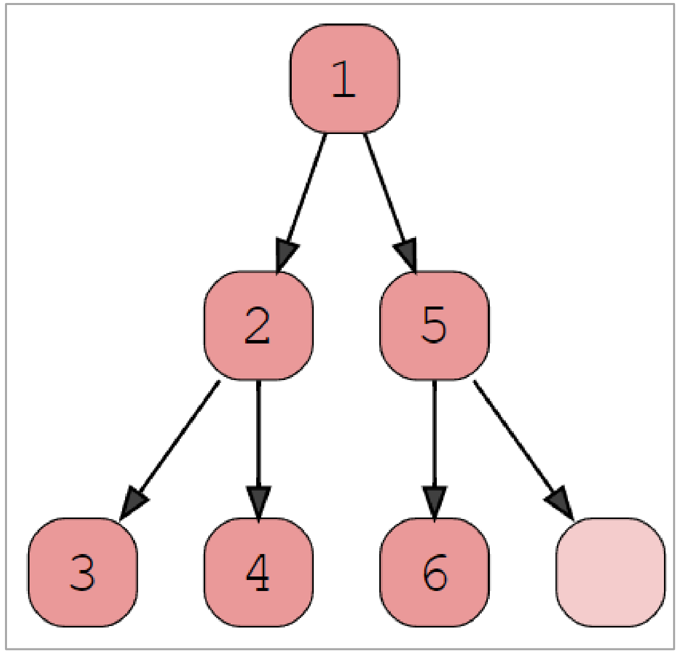
Ho Chi Minh City

Outline

- 2-3 Tree
- 2-3-4 Tree

Reviews

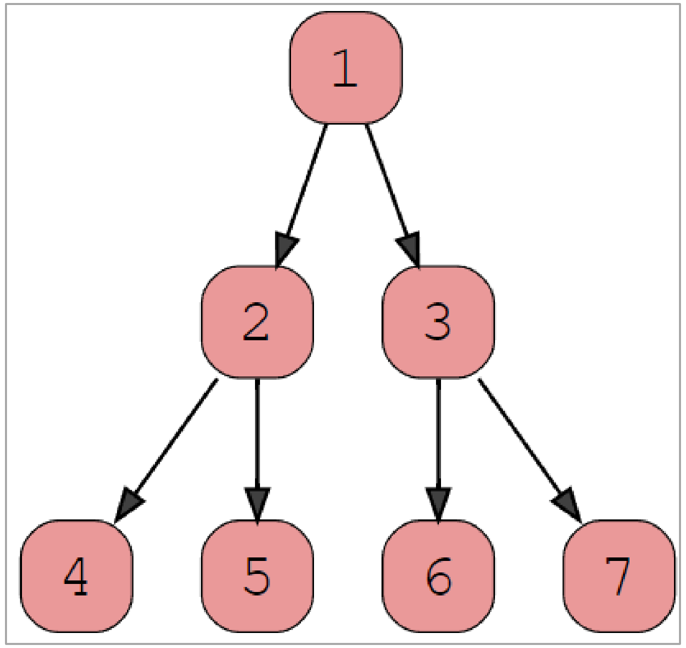
- Complete Tree:
  - Every level, excluding the last, is filled
  - All nodes at the last level are as far left as they can be.



Reviews

fit@hcmus

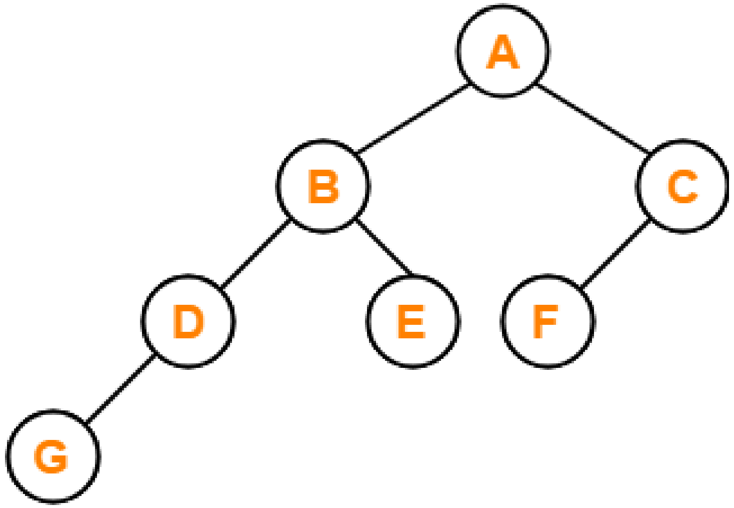
- Perfect Tree:
  - Null links are all the same distance from the root



Do you like perfect tree?
Why? Why not?

Reviews

- Is a AVL tree a perfectly balanced search tree?



AVL Tree

(Height = 3)

Reviews

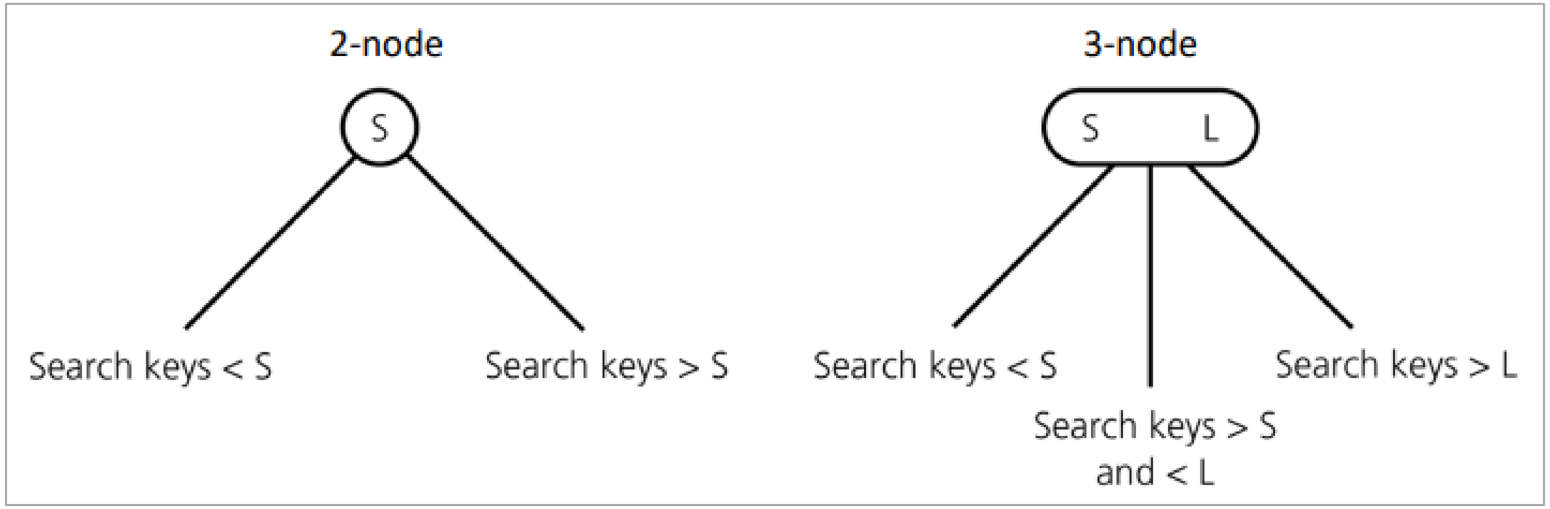
- Is there any way to make a perfectly AVL tree all time?
  - No
- Is there a perfect balanced search tree all time?
  - Yes

2-3 Search Tree

- A 2-3 search tree is a tree that is either empty or
  - Has 2-node, with one key and two links
  - Has 3-node, with two keys and three links

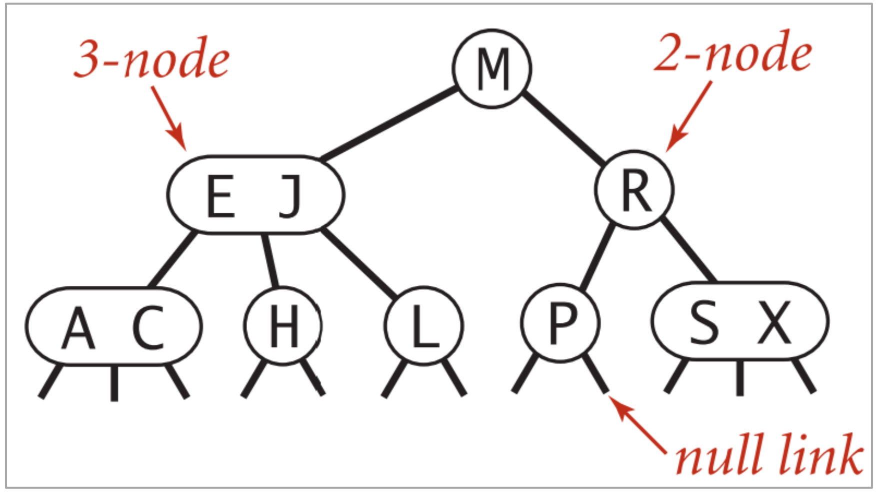
2-3 Search Tree

- A 2-3 search tree is a tree that is either empty or
  - Has 2-node, with one key and two links
  - Has 3-node, with two keys and three links
  - Satisfy value properties as a seach tree
  - All leaves are at the same level in the tree



2-3 Search Tree

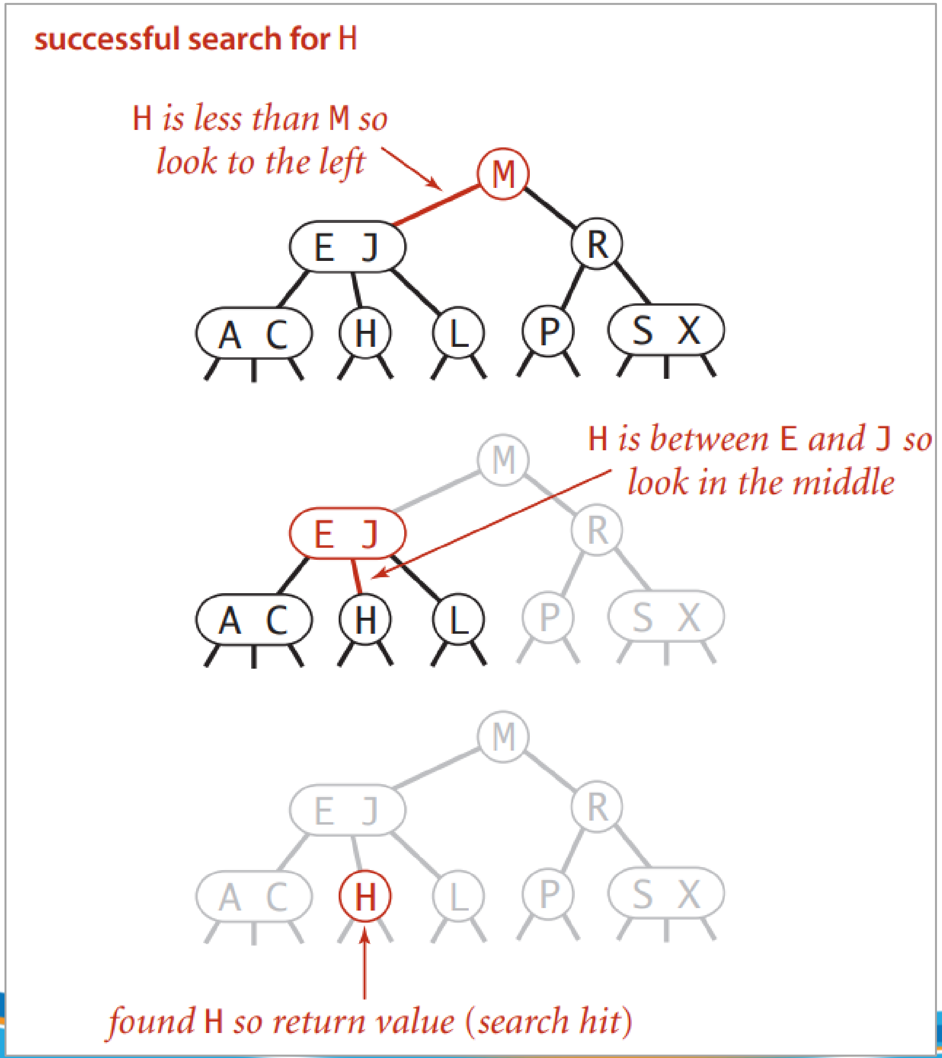
- Example



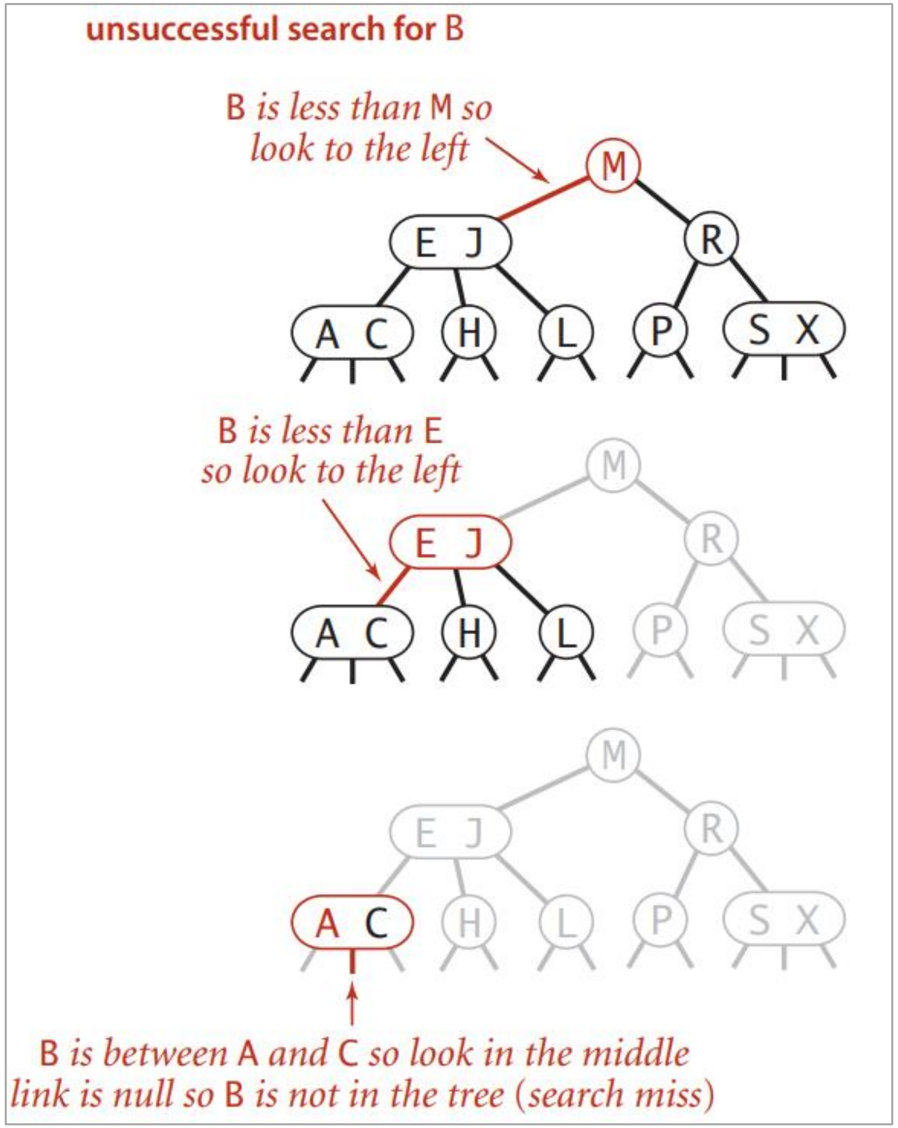
Search an Item

fit@hcmus

- Search in 2-3 tree is same the search algorithm for BST.



Search an Item

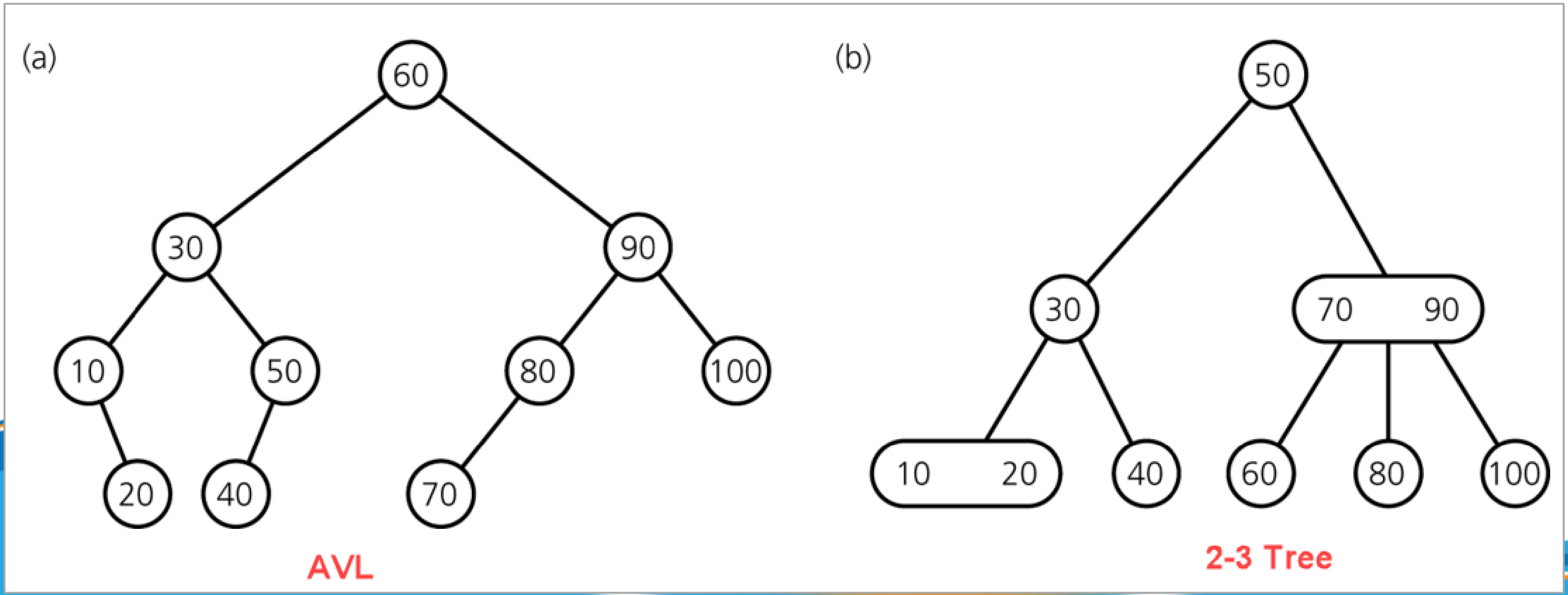


Time Efficiency of Searching

- What is the time efficiency of searching for an item?
  - O(logn)
  - But
    - A AVL tree's height: 1.44log(n+2)-0.328
    - A 2-3 tree's height: 2log(n+1)

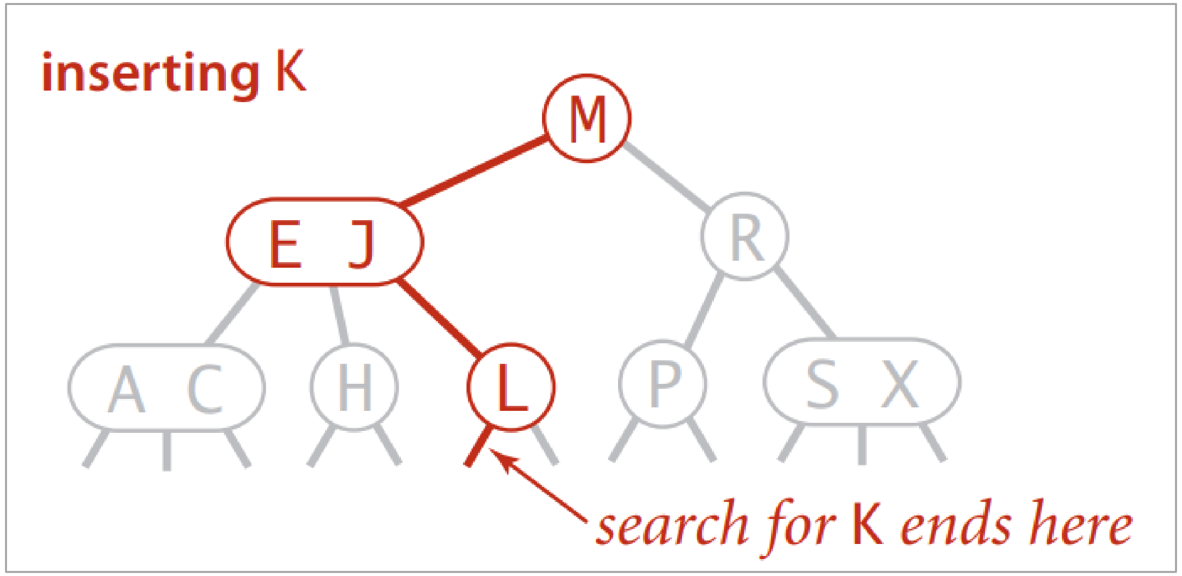
    ⇒ A AVL is slightly faster than a 2-3 tree.

    ⇒ So why we need a 2-3 tree?



Insert an item

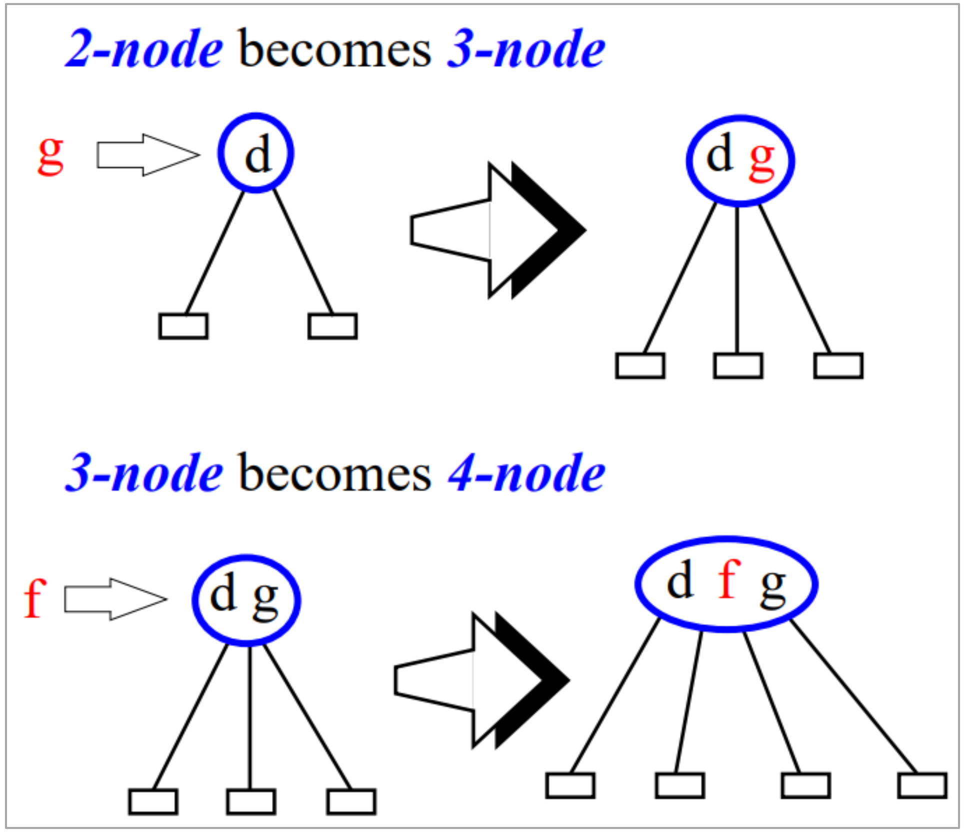
- To insert an item to a 2-3 tree:
  - Do an unsuccessful search and then hook on the node at the bottom.
  - Insert a new item into this node.



Insert an item

fit@hcmus

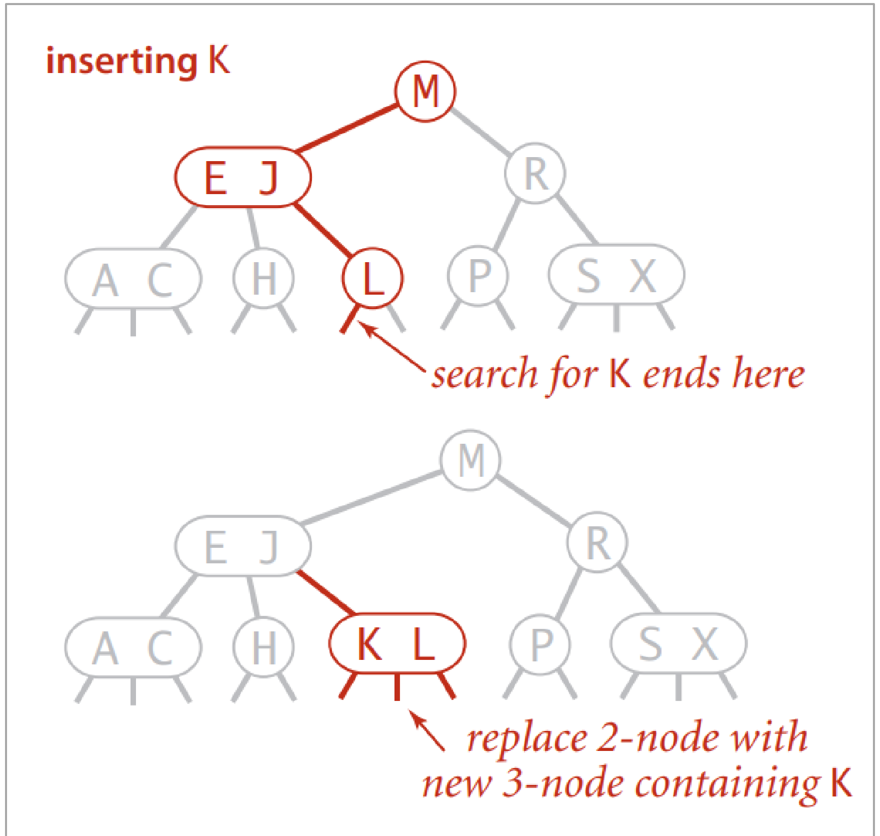
- There are two cases:
  - Insert into a 2-node
  - Insert into a 3-node



Insert an item

fit@hcmus

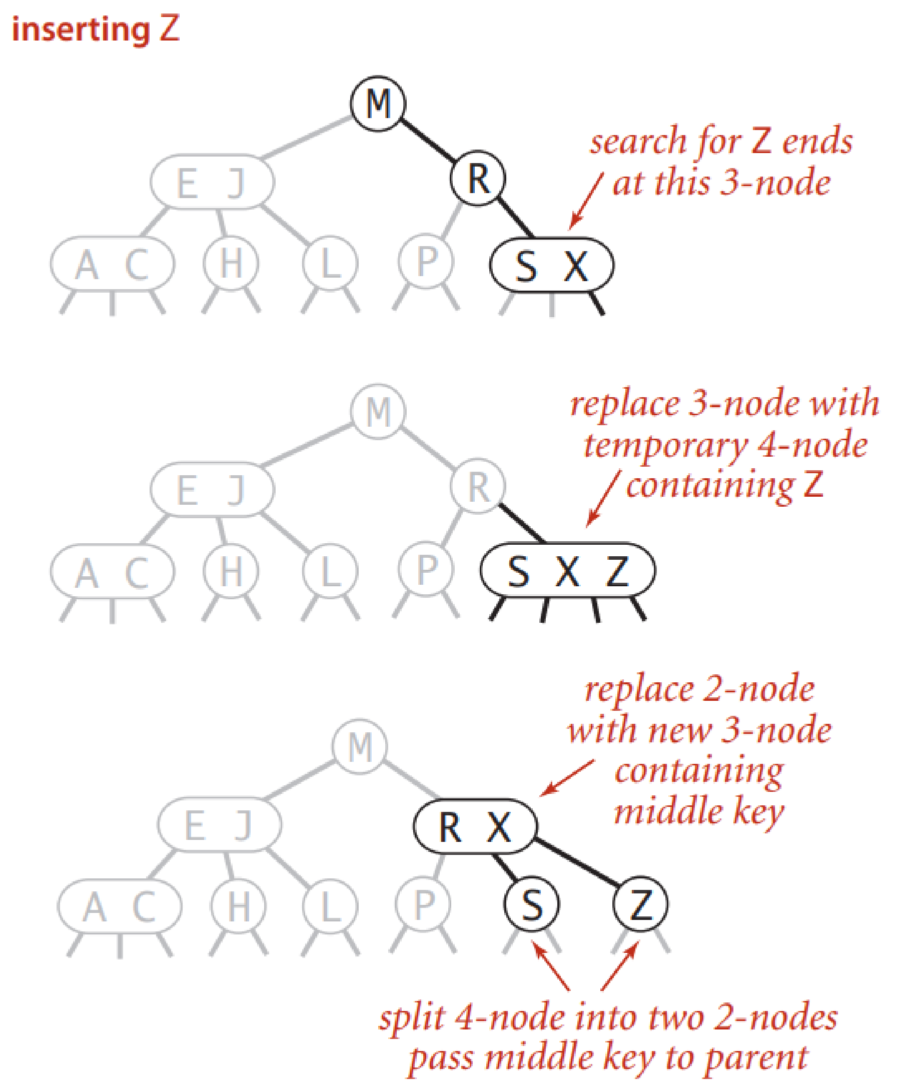
- Insert into a 2-node:
  - Just replace the node with a 3-node containing its key and the new key to be inserted.



Insert into a 2-node

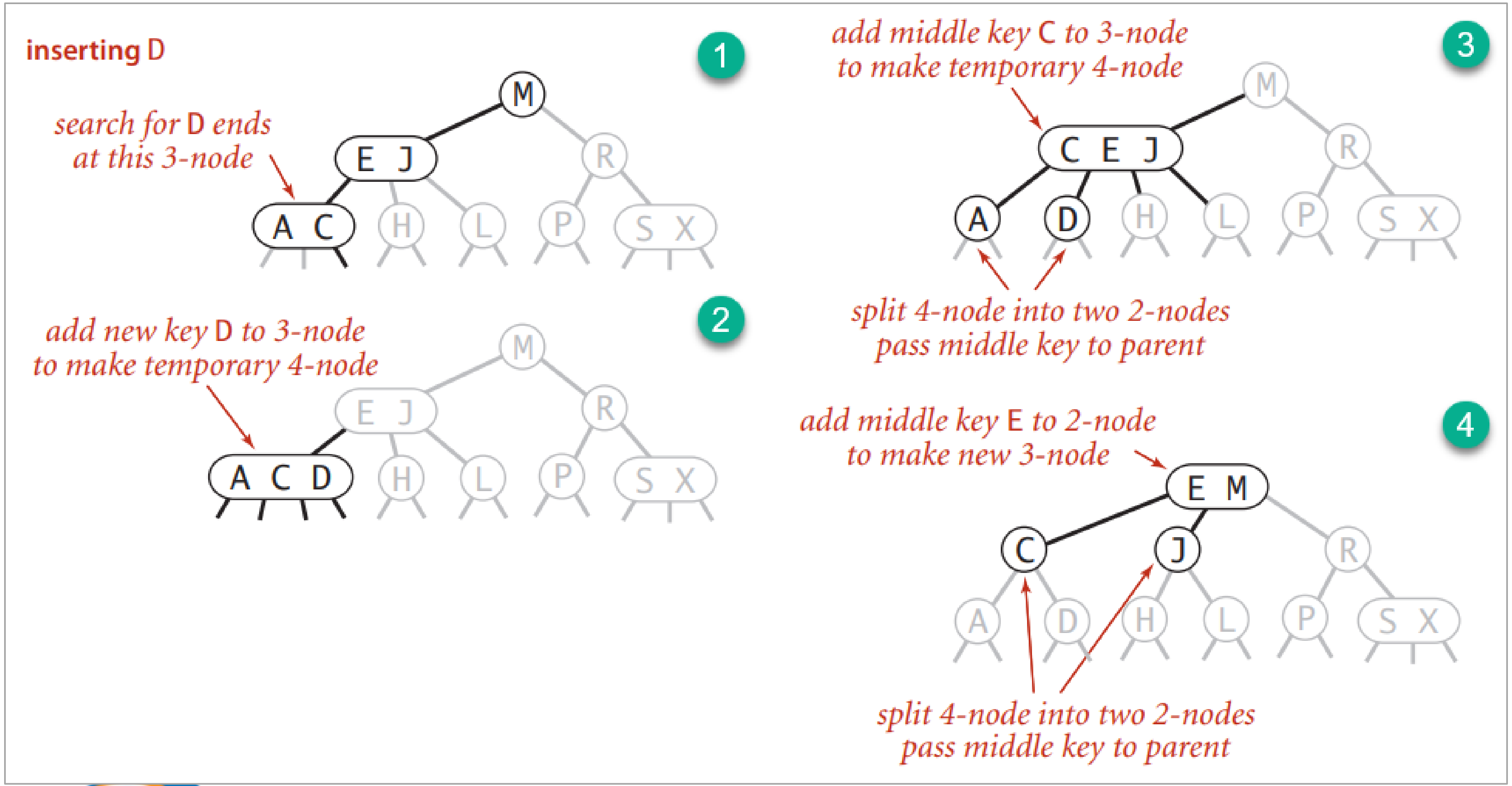
Insert an item

- Insert into a 3-node:
  - The new key to be inserted the leaf and after that …
  - Divide (split) the leaf and move middle value up to parent.
  - Check and fix parent if it is overcrowded and repeat again until root.



Insert into a 3-node whose parent is a 2-node

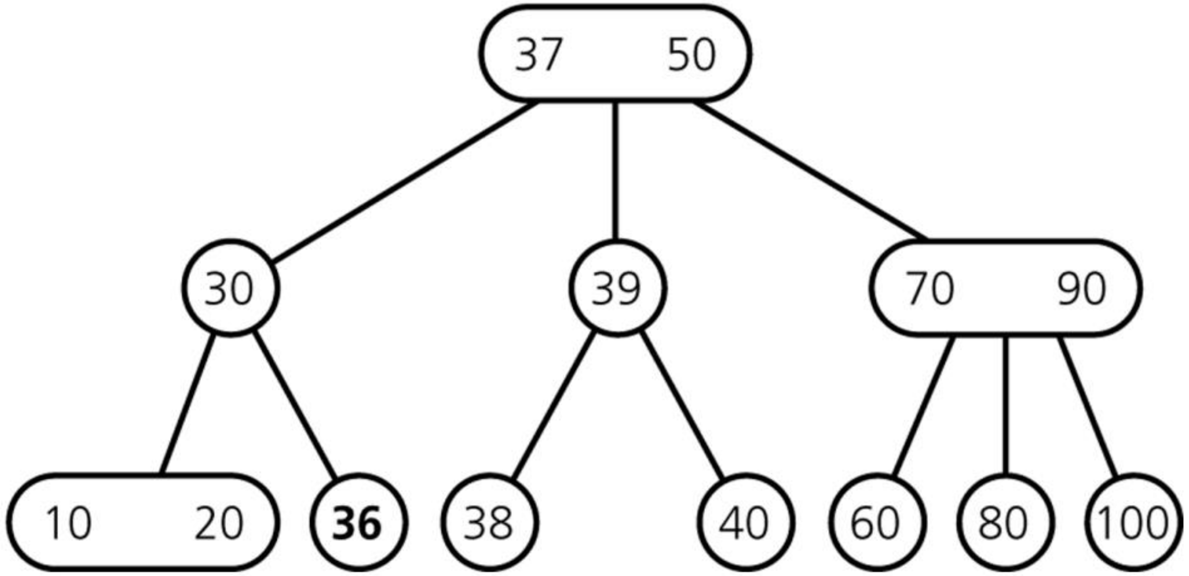
Insert an item



Exercises

- Insert some following values into current 2-3 tree:

    35, 34, 33, 32, 15, 29, 48, 17



Comments

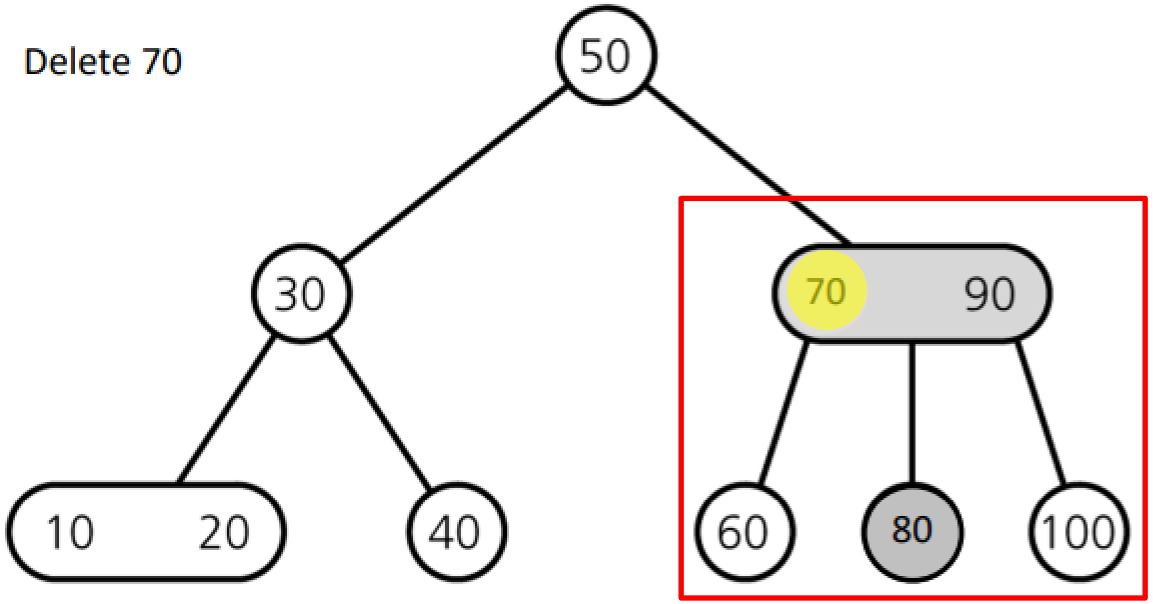
fit@hcmus

- Some comments on inserting:
  - Simple balancing
    - No part of the tree needs to be examined or modified other than the specified nodes and links.
    - Number of links changed is bounded by a small constant.
  - Is better than AVL tree?

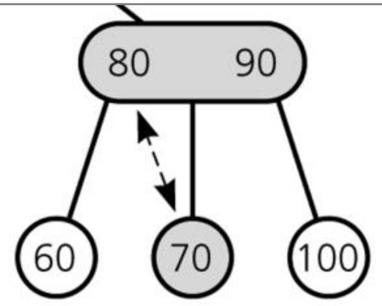
Delete an item

- To delete an item from a 2-3 tree:
  - Do an successful search and then delete that value from the node at the bottom (similar with BST)
  - Deletion leaves a hole in a bottom node so removing the hole without violating the 2-3 tree.

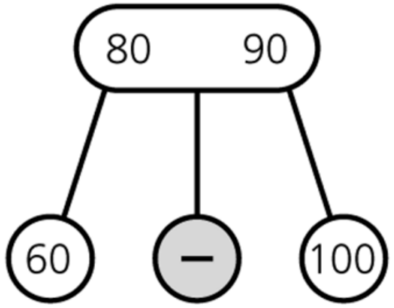
Delete 70



Swap with inorder successor



Swap with inorder successor



Delete value from leaf

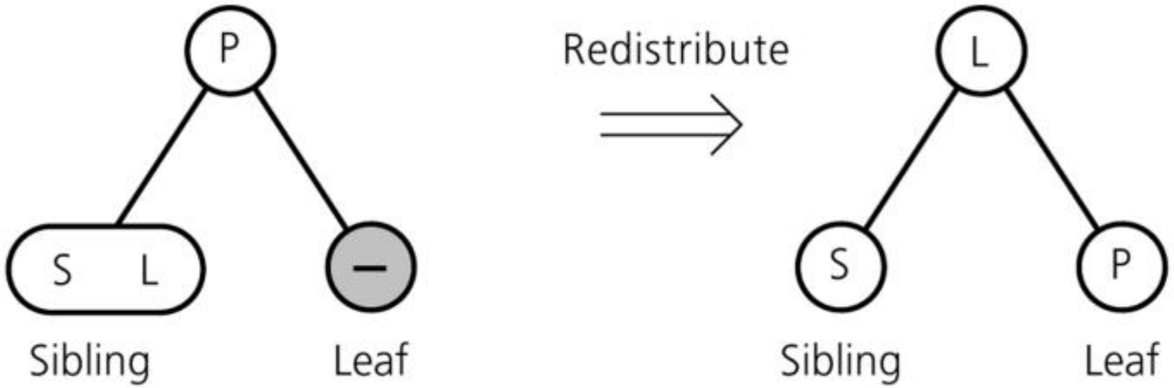
Delete an item

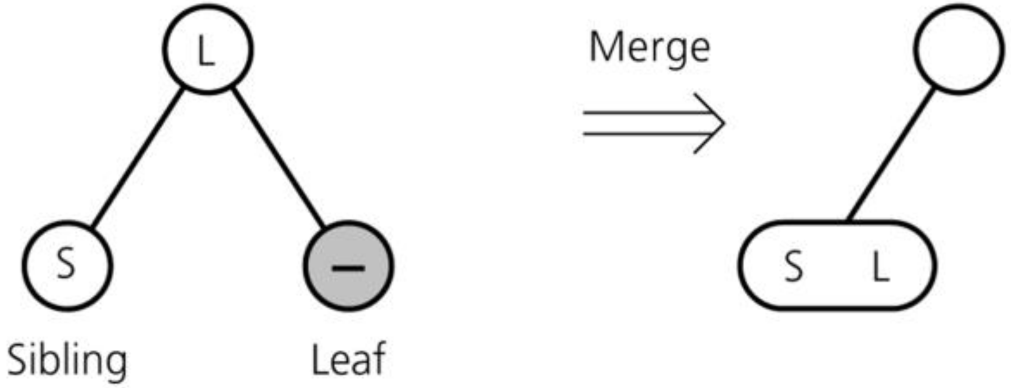
fit@hcmus

- There are 2 main cases:
  - If the node (with hole) is enough (2-node), do nothing.
  - Otherwise (empty):
    - If sibling is rich (3-node), borrow (redistribute) a value from it through parent.
    - If sibling is poor (2-node), join (merge) with sibling and parent. Consequently, new hole is created on parent node. Repeat removing the hole from parent.

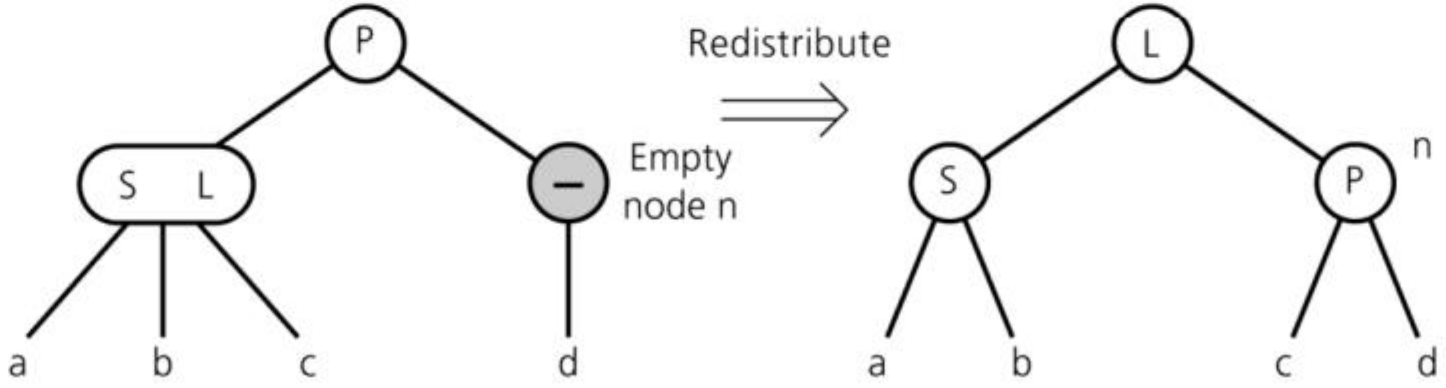
Redistribute and Merge (1)

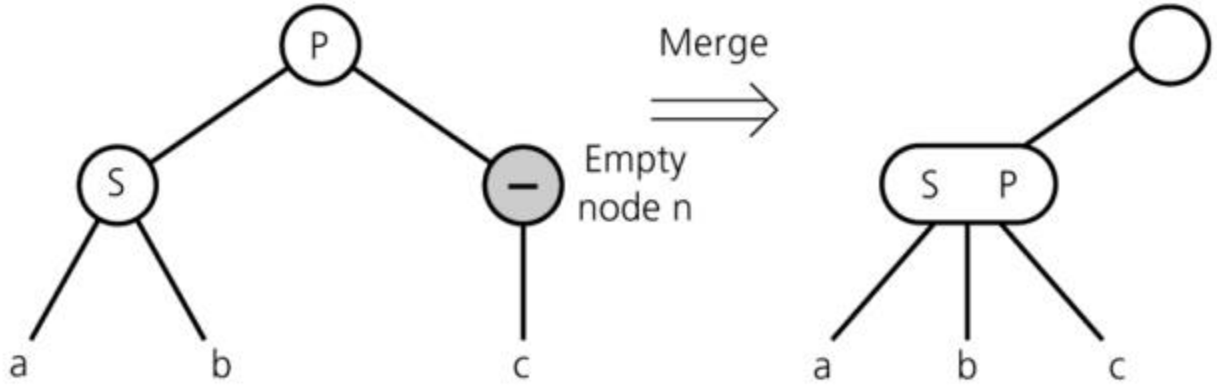
fit@hcmus



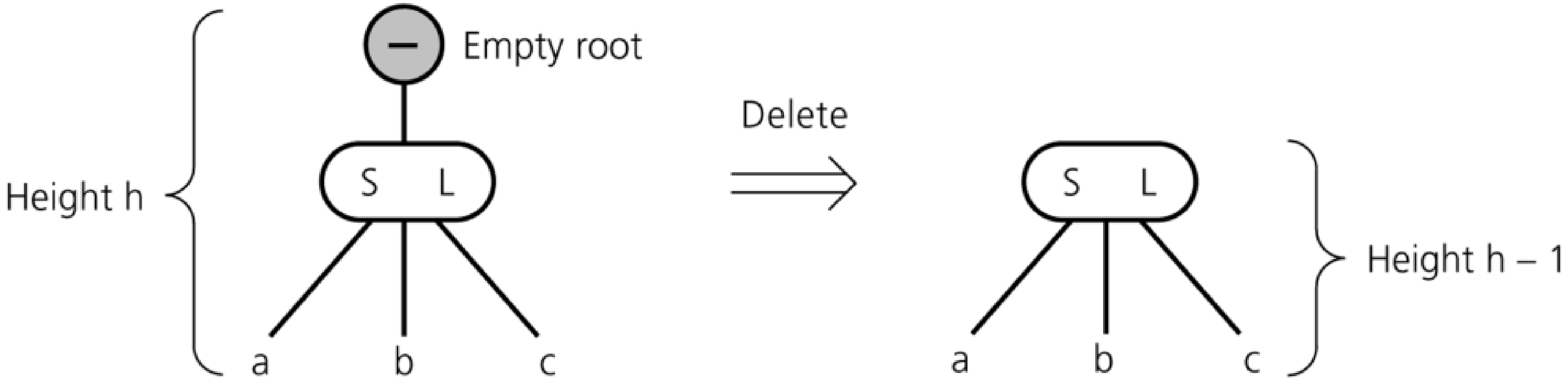


Redistribute and Merge (2)





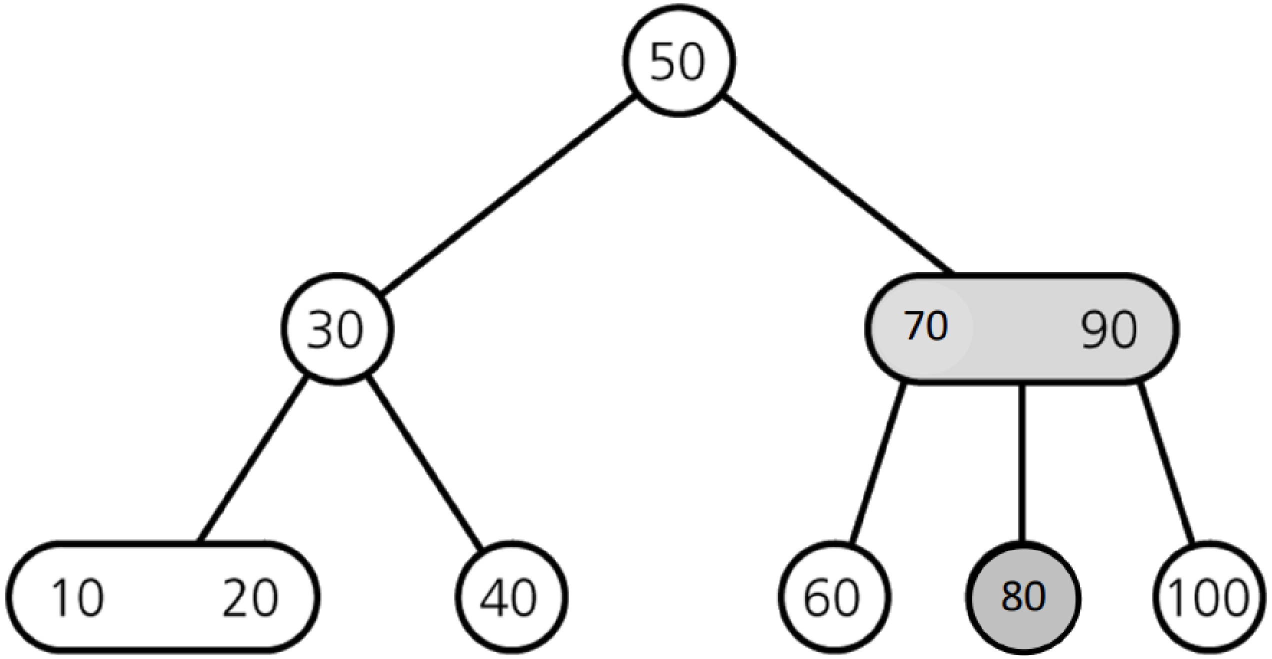
Redistribute and Merge (3)



Exercise

- Delete following values from a 2-3 tree:

  70, 100, 80, 20, 50

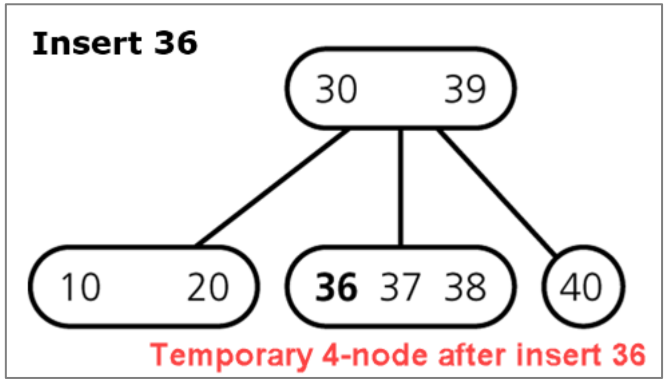


Comments

- Advantages:
  - Perfect balanced
  - Do not use rotation
  - Complexity is O(logn)
- Disadvantages:
  - Walking up the tree to split nodes
  - When insert an item, we create the temporary 4-node.
  - Need to check which value is middle

$\Rightarrow$ Waste space and time

$\Rightarrow$ Solve with a 2-3-4 tree

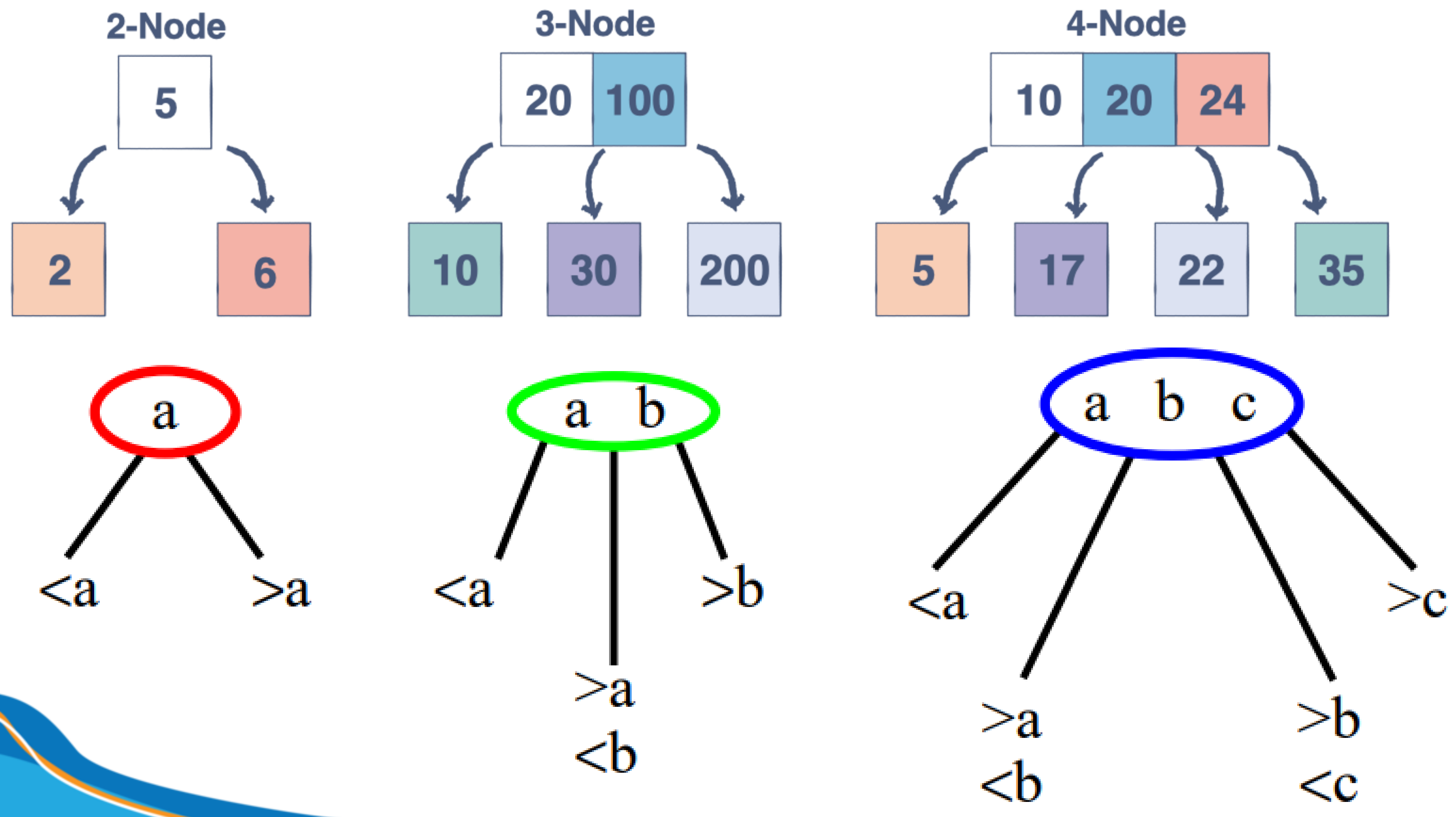


Outline

- 2-3 Tree
- **2-3-4 Tree**

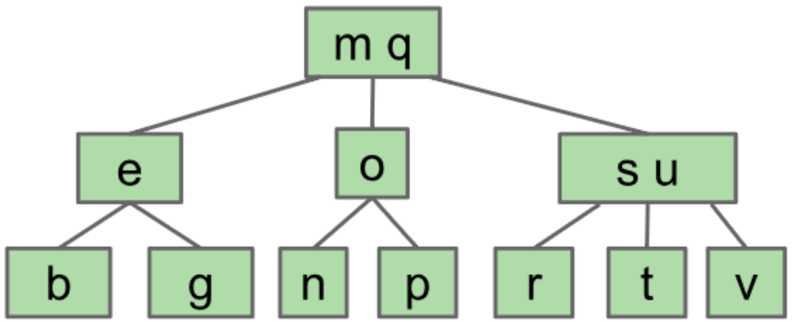
2-3-4 Tree

- A 2-3-4 tree is like a 2-3 tree, but it allows 4-nodes, which are nodes that have four children and three data items.

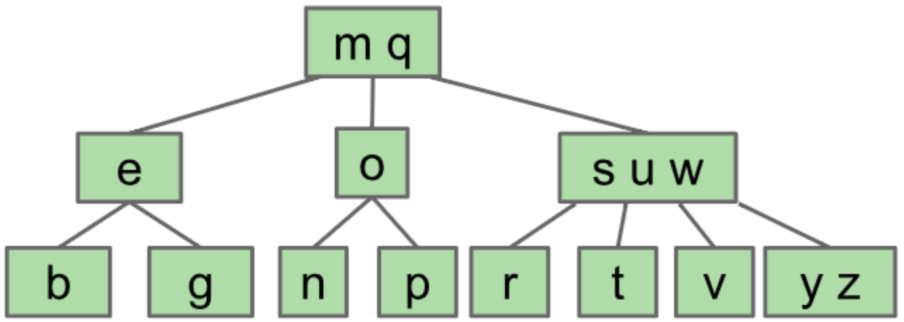


Search, Insert, Delete in a 2-3-4 tree

- Search, Insert, Delete in a 2-3-4 tree are similar to 2-3 tree, but it has more efficient insertion and deletion operations than a 2-3 tree.



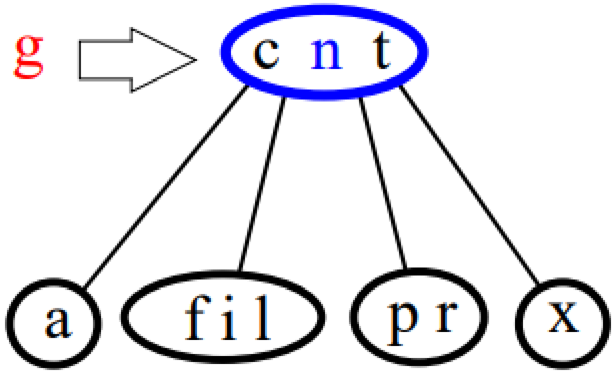
2-3 Tree (L=2):
Max 2 items per node.
Max 3 non-null children per node.



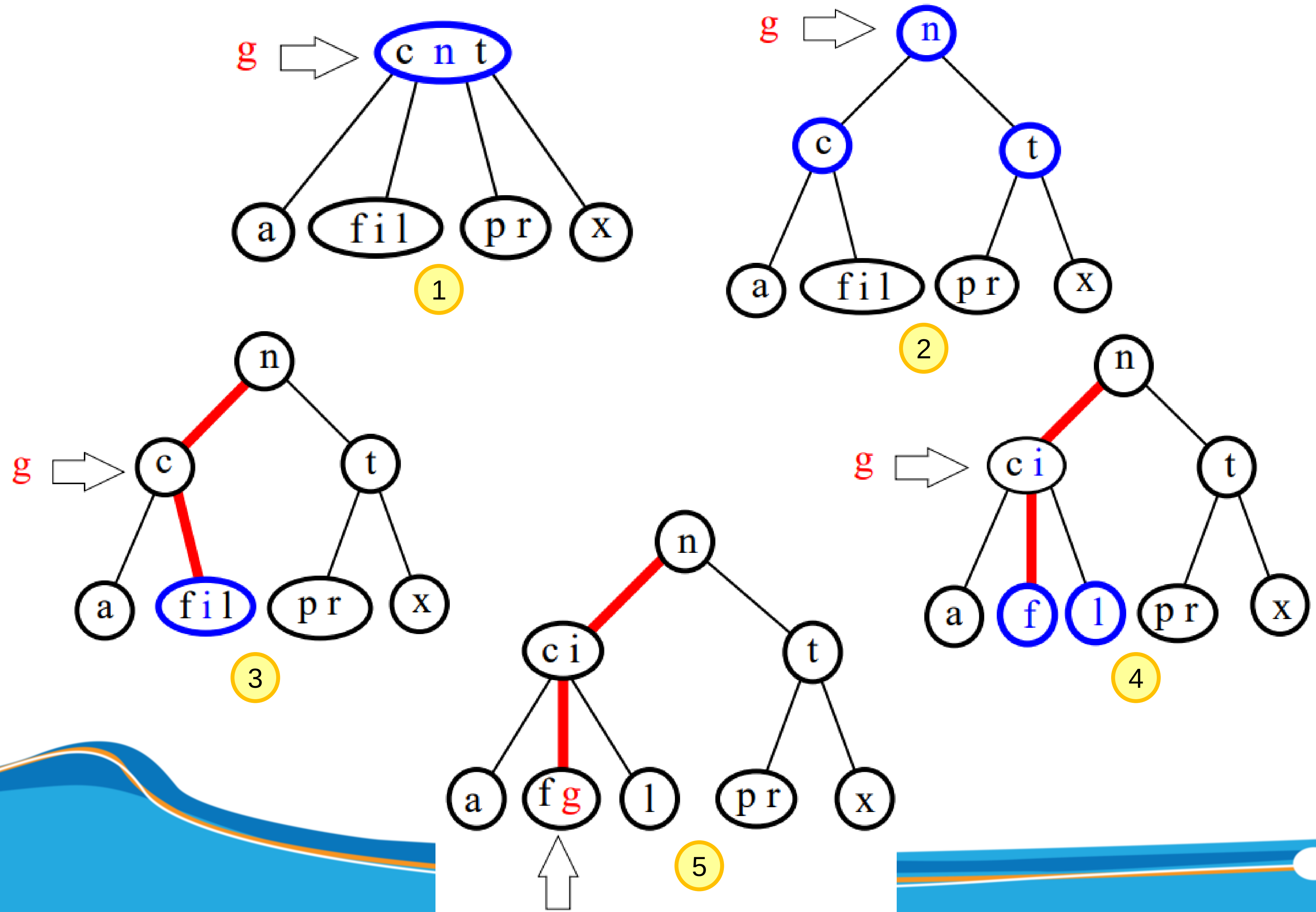
2-3-4 a.k.a. 2-4 Tree (L=3):
Max 3 items per node.
Max 4 non-null children per node.

Top Down Insertion

- For a 2-3 tree:
  - The insertion algorithm traces a path from the root to a leaf and then backs up from the leaf as it splits nodes.
- For a 2-3-4 tree:
  - Insertion can be done in one pass
  - To avoid this return path after reaching a leaf, whenever we reach a 4-node, we break it up into two 2-nodes, and move the middle element up into the parent node.

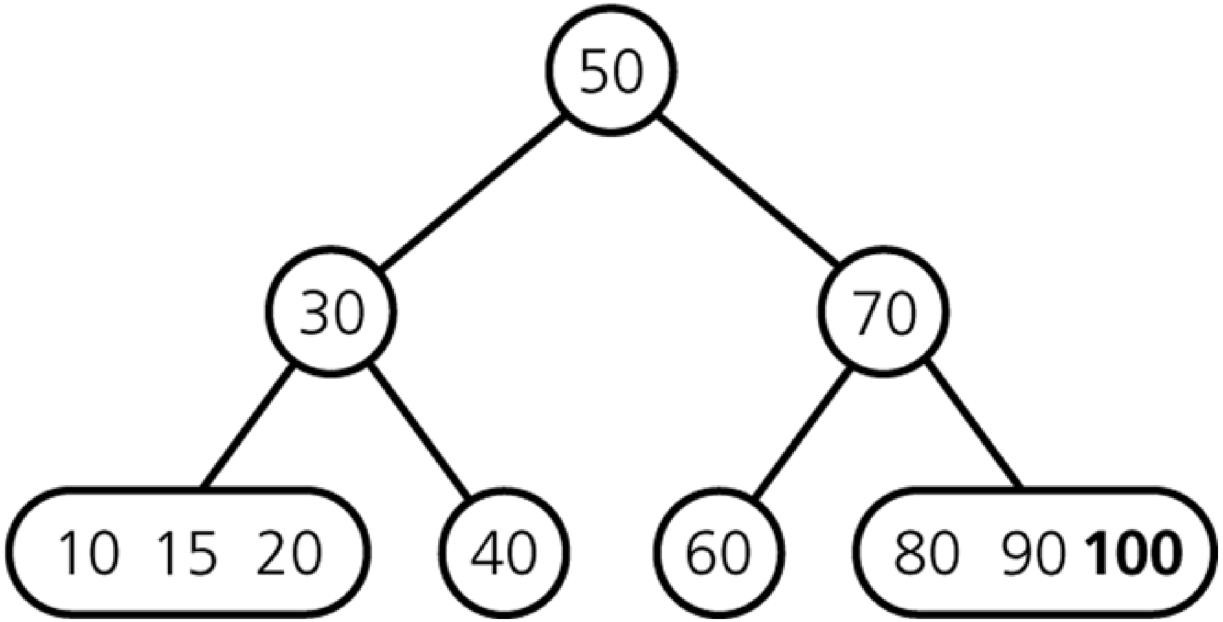


Top Down Insertion



Exercise

- Inserting 60, 30, 10, 20, 50, 40, 70, 80, 15, 90, 100 into a 2-3-4 tree.



Result Tree

Top Down Deletion

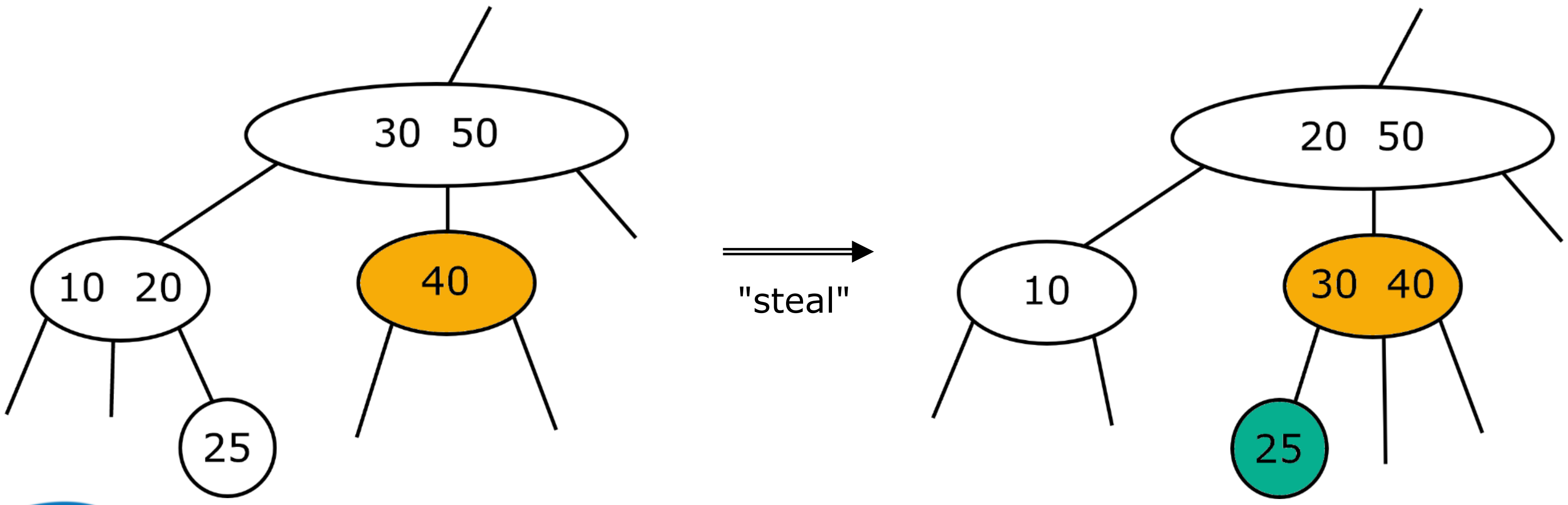
fit@hcmus

- For a 2-3 tree:
  - The deletion algorithm traces a path from the root to a leaf and then backs up from the leaf, fixing empty nodes on the path back up to root.
- For a 2-3-4 tree:
  - Deletion can be done in one pass
  - To avoid this return path after reaching a leaf, we transforms each 2-node into either 3-node or 4-node as soon as it encounters them on the way down the tree from the root to a leaf.
    - Case 1: If an adjacent sibling is a 3-node or 4-node, transfer an item from that sibling to our 2-node.
    - Case 2: If adjacent sibling is a 2-node, merge them.

Top Down Deletion

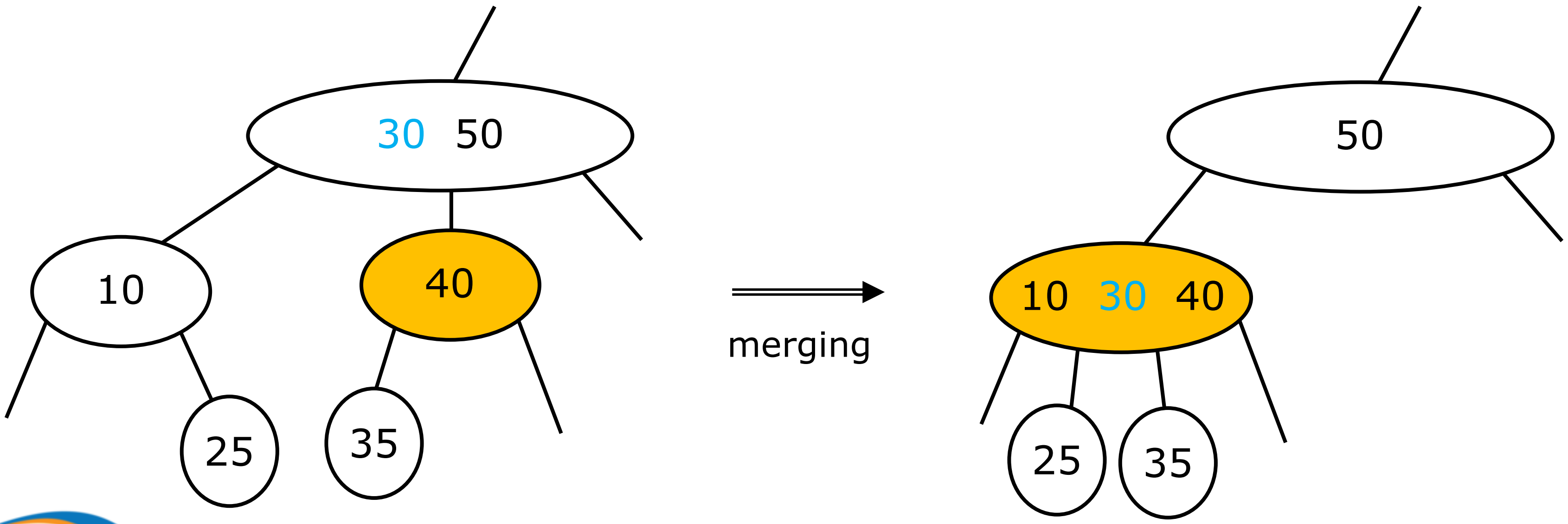
fit@hcmus

- For a 2-3-4 tree:
  - To avoid …
    - Case 1: If an adjacent sibling is a 3-node or 4-node, transfer an item from that sibling to our 2-node.



Top Down Deletion

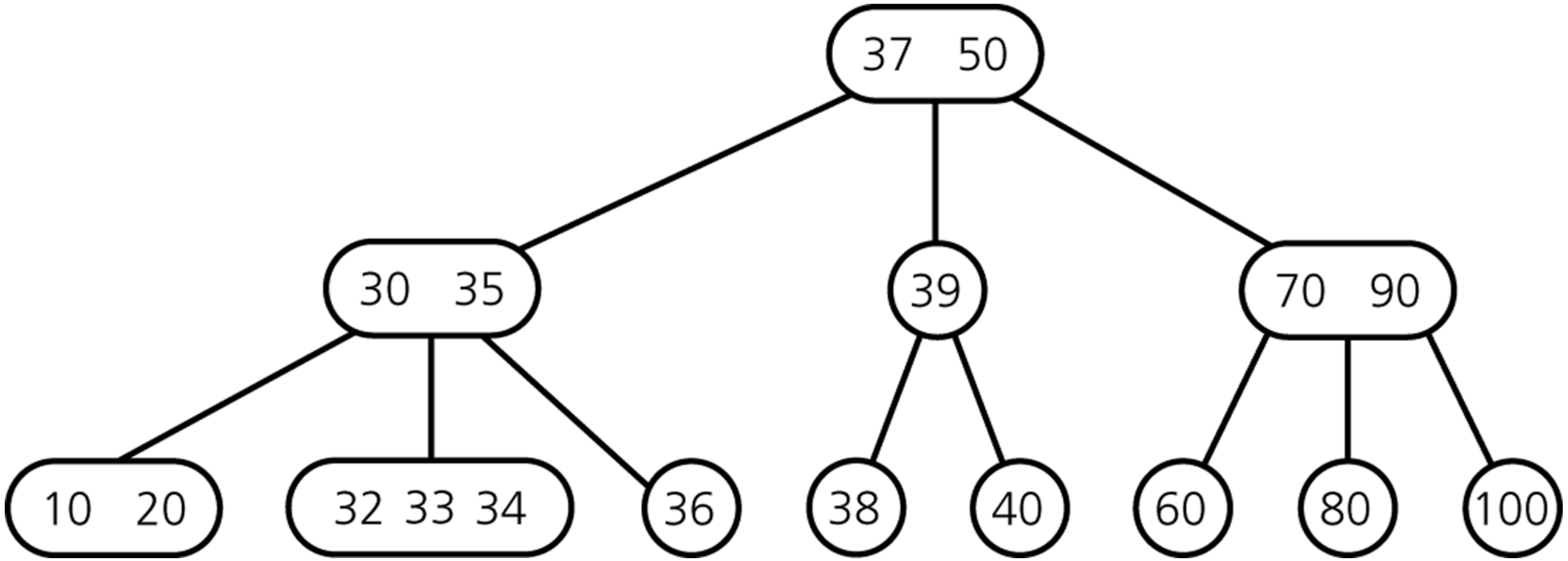
- For a 2-3-4 tree:
  - To avoid …
    - Case 1: …
    - Case 2: If adjacent sibling is a 2-node, merge them.



Exercise

fit@hcmus

- Delete 32, 35, 40, 38, 39, 37, 60 from the following 2-3-4 tree

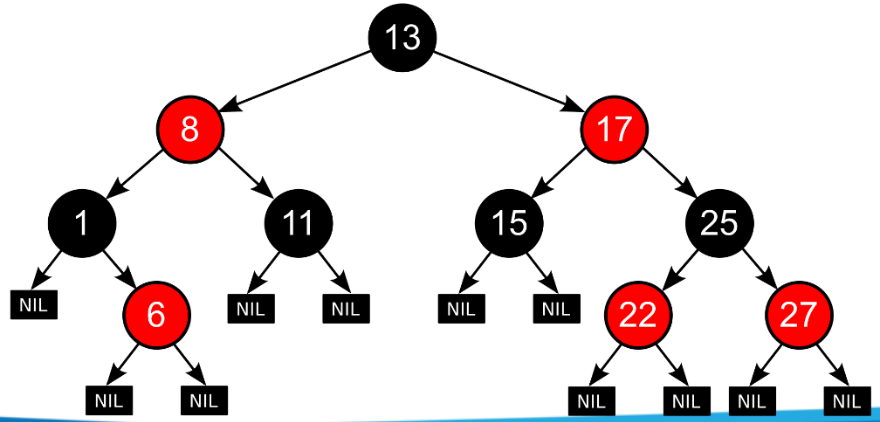


Comments

fit@hcmus

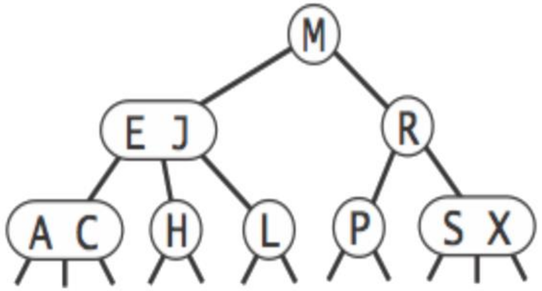
- Advantages:
  - Perfect balanced
  - Time complexity: O(logN)
  - Insertion/deletion performance is more efficient than a 2-3 tree.
- Disadvantages:
  - Different node structures
    - Interconversion of nodes among 2-nodes, 3-nodes and 4-nodes.

=> Solved by Red-Black Tree



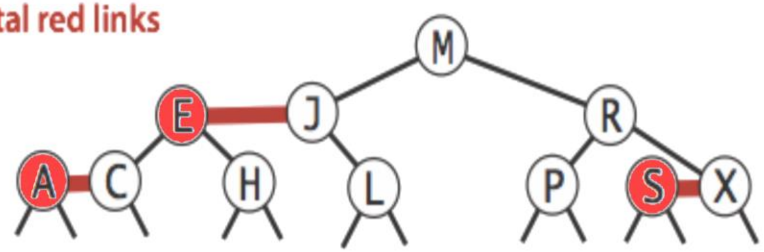
2-3 tree to red-black tree

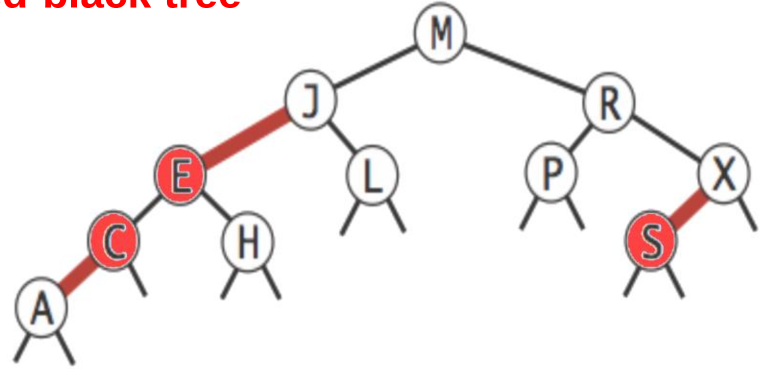
2-3 tree



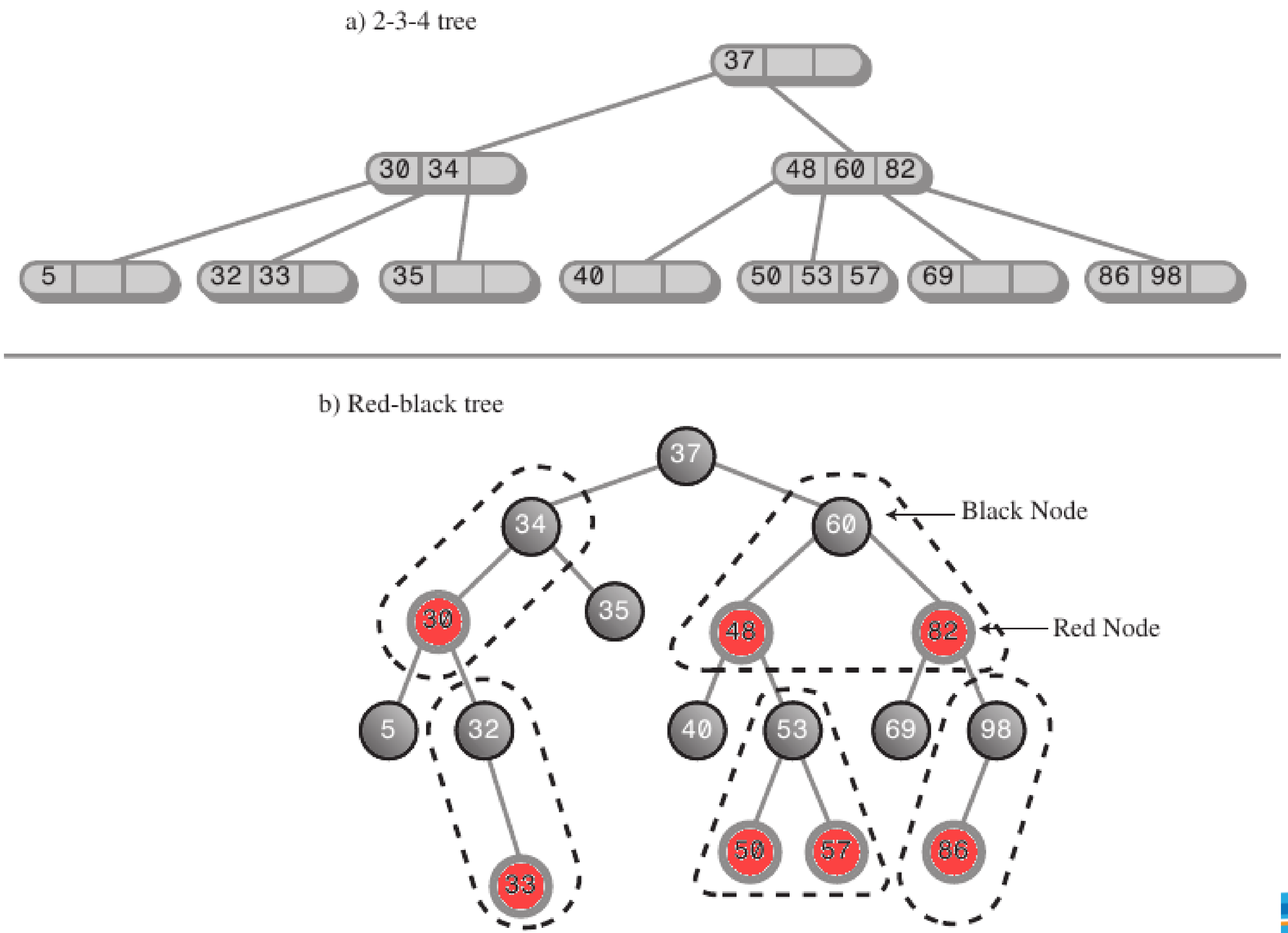
Left-lean red-black tree

horizontal red links





2-3-4 tree to red-black tree



Exercises

- Find an order such that if you add the items 1, 2, 3, 4, 5, 6, and 7 in that order, the resulting 2-3 tree has height 1.

Conclusion

fit@hcmus

- Binary search trees are simple, but they are subject to imbalance which leads to crappy runtime.
- 2-3 trees are balanced, but painful to implement and relatively slow.
- 2-3-4 trees are more effective than 2-3 trees in insertion and deletion.
- LLRBs maintain correspondence with 2-3 tree, Standard Red-Black trees maintain correspondence with 2-3-4 trees.
  - More complex implementation, but significantly faster.

The End