2-3 trees

Edited from: https://cs.mtsu.edu/~jhankins/files/3110/presentations/2-3Trees.ppt

Motivation

- Binary search trees need O(n) time for search, insertion and deletion.
- Would like to have a tree which self balances itself with every deletion and insertion to maintain as much a balanced tree as possible wherein all leaves are at the same level

Properties

- Each node has either one value or two values

- A node with one value is either a leaf node or has exactly two children (non-null).
  Val(left subtree) < val(node) < val(right subtree)

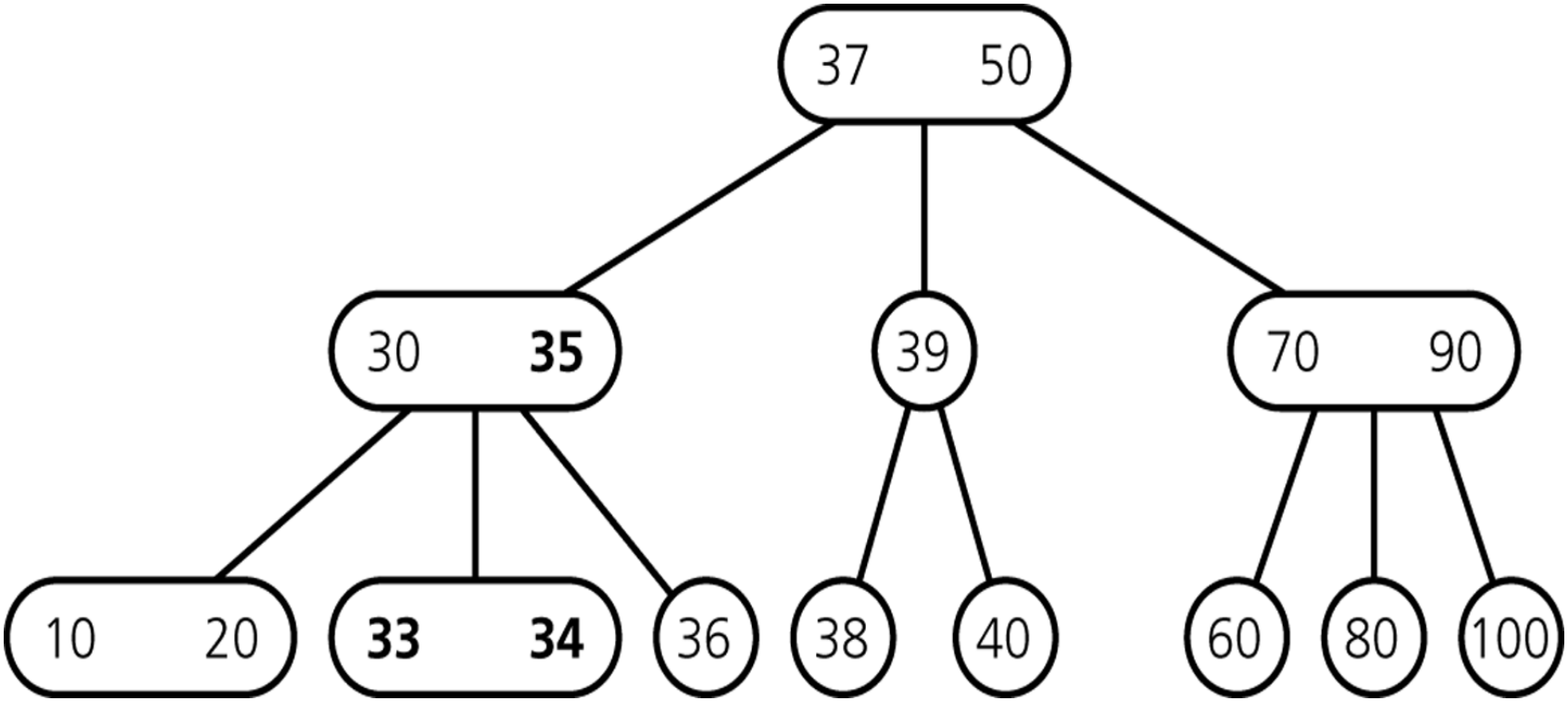
- A node with two values is either a leaf node or has exactly three children (non-null).

Val(left subtree) < val(first value in node) < val(middle subtree)< val(second value in node) < val(right subtree).

- All leaf nodes are at the same level of the tree

- 23 tree has height O(log n). Why?

23 tree



Insertion

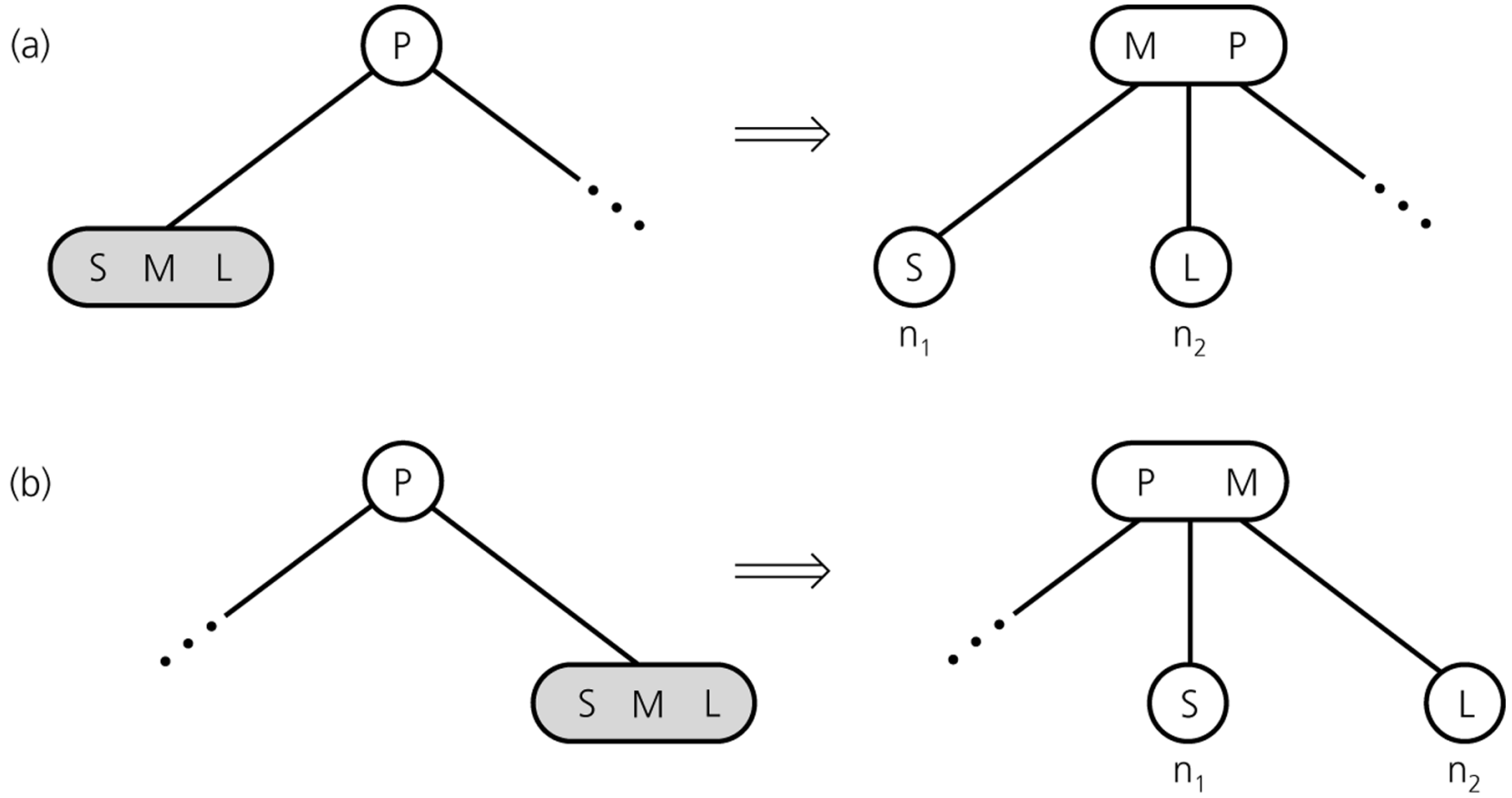
Three cases for insertion

Case 1: Insert into a leaf node with space

Case 2: Insert into a full leaf with parent with space

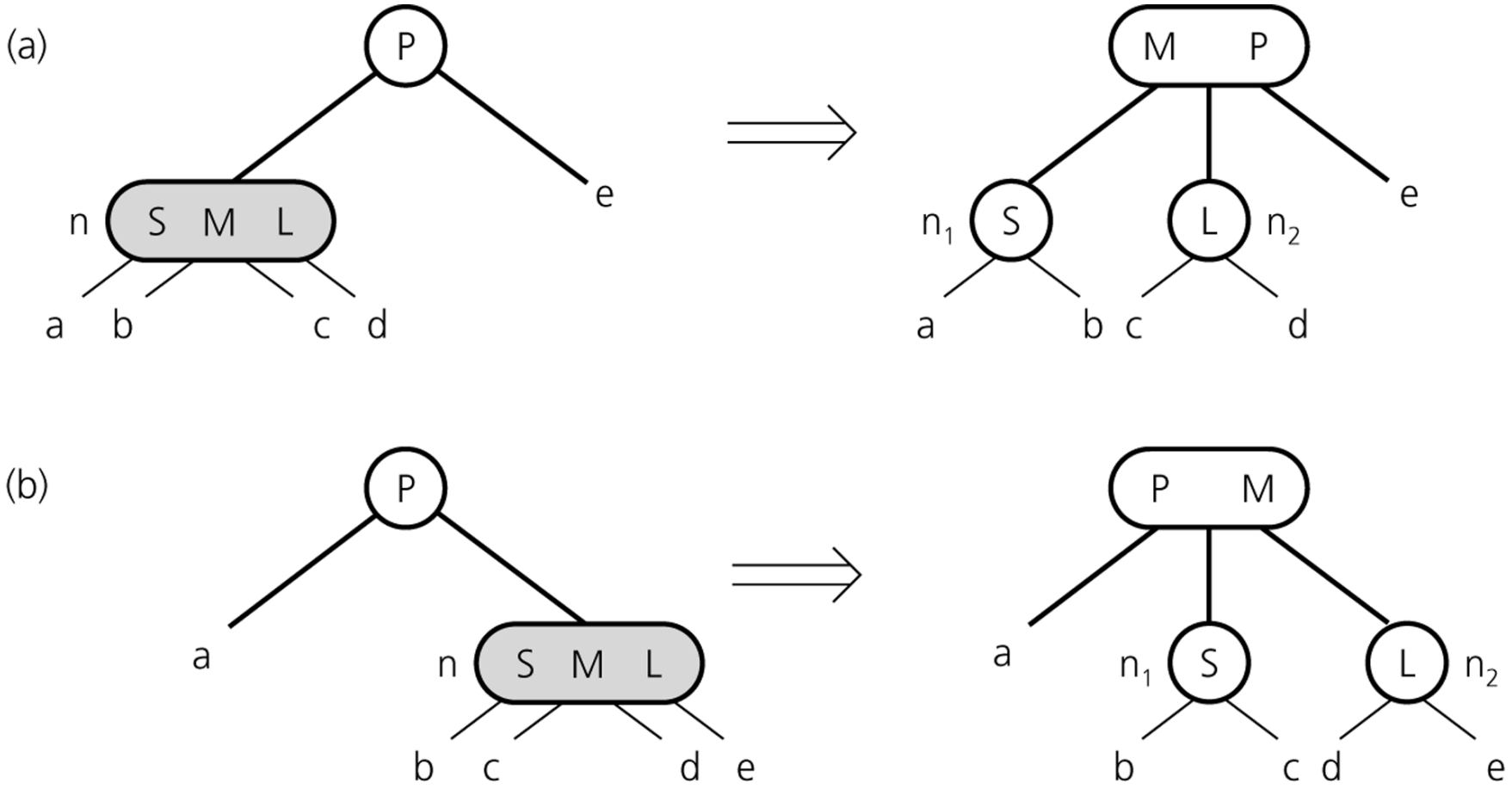
Case 3: Insert into a full leaf with parent without space.

Inserting into full node (Case2)



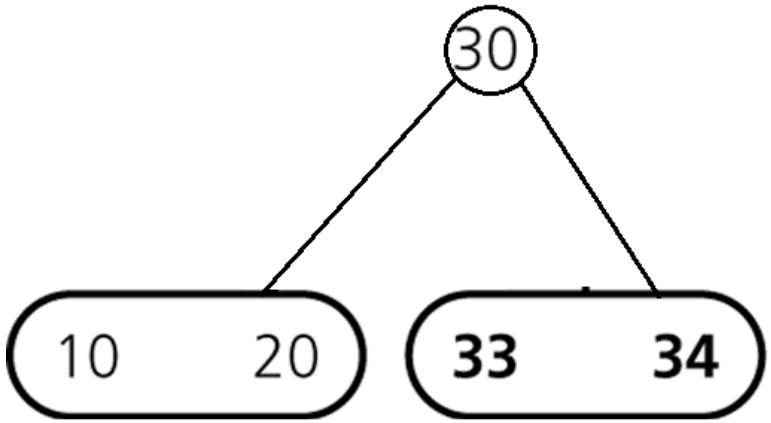
Inserting so far(Case 3)

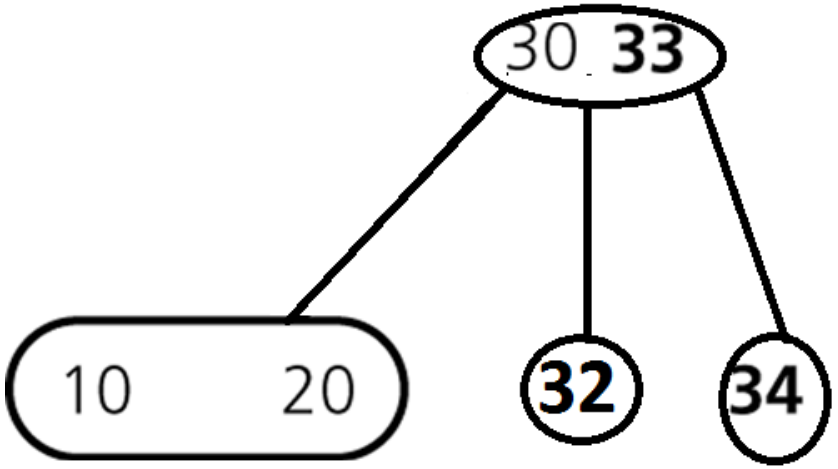
Say node with b,c which was middle child of node with S,L had M inserted. Then M goes up to node SL as shown



Inserting Items

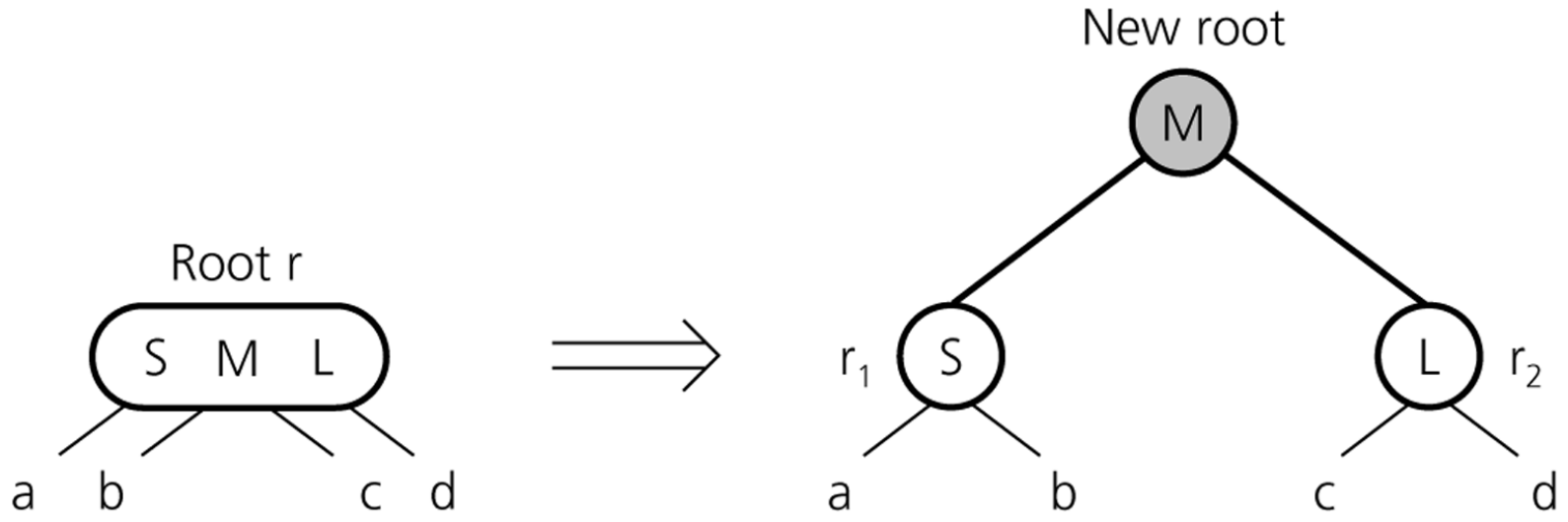
How do we insert 32?





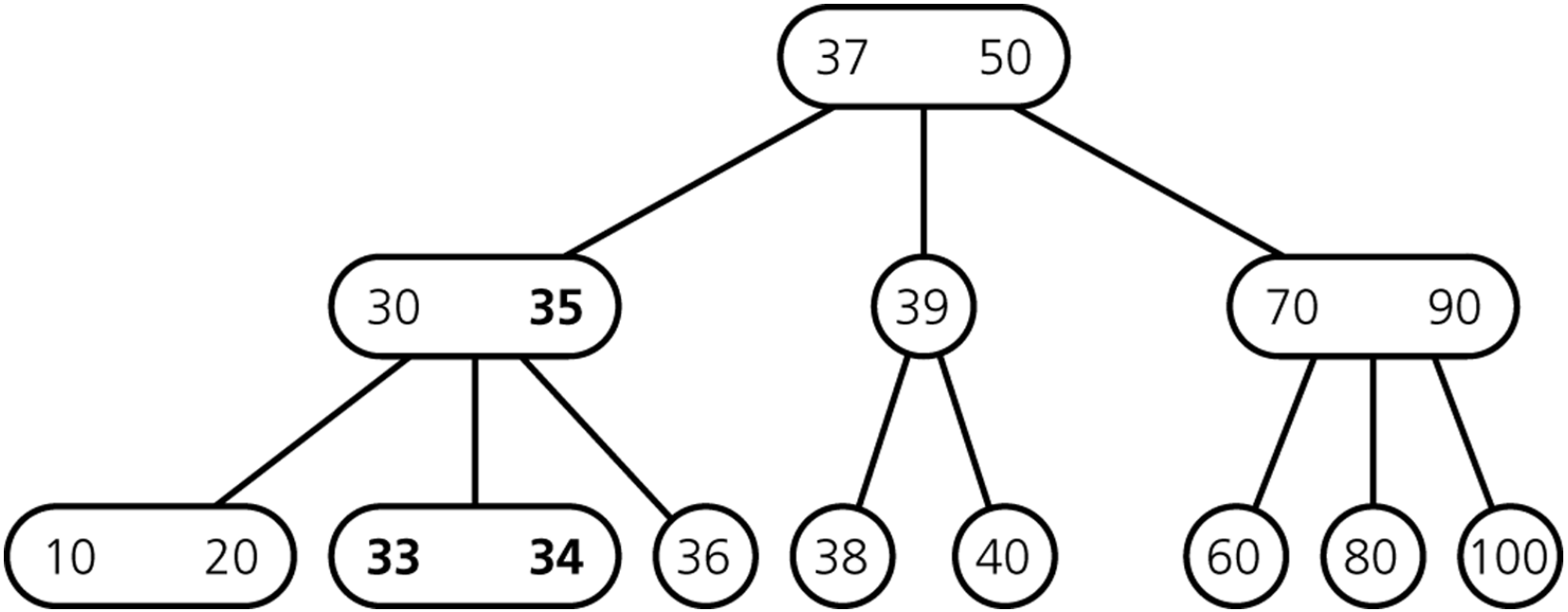
Inserting Items

→ **creating a new root if necessary**
→ **tree grows at the root**



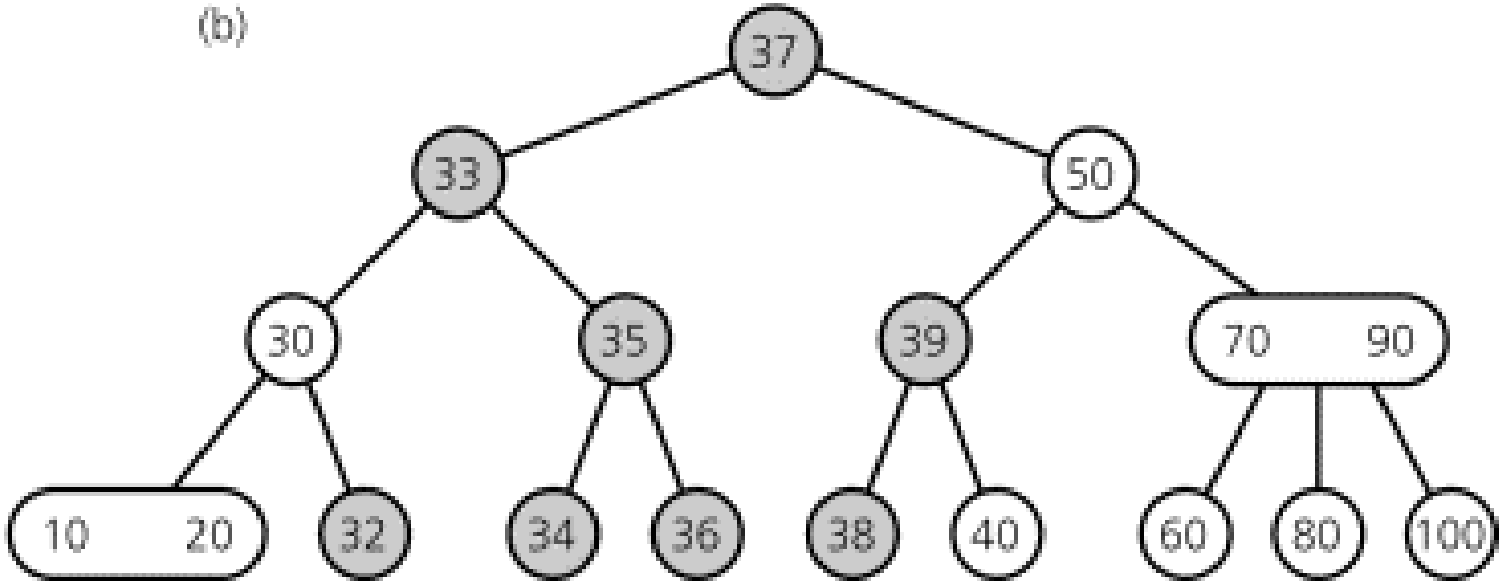
Inserting Items

How do we insert 32?



Inserting Items

Final Result



Insert the following

Insert into the 23 tree in the below sequence

40 30 34 50 36 80 9 37 20 100 38

70 60 39 90 33 35 95

Notes

The height of the 23 tree increases upwards

If a leaf can accommodate the new element then done

Else send the middle element up(iteratively until new root formed)

When an element goes up it takes with it an new pointer to a child increasing the number of children of the parent node from 2 to 3(element found a place) or from 3 to 4 (element splits current node by sending mid element to parent node).

Deletion

Step 1

Deletion happens at the leaf node.

If the node to be deleted is not on a leaf node:
swap it with the closest successor.
The node to be deleted is now in a leaf node !!

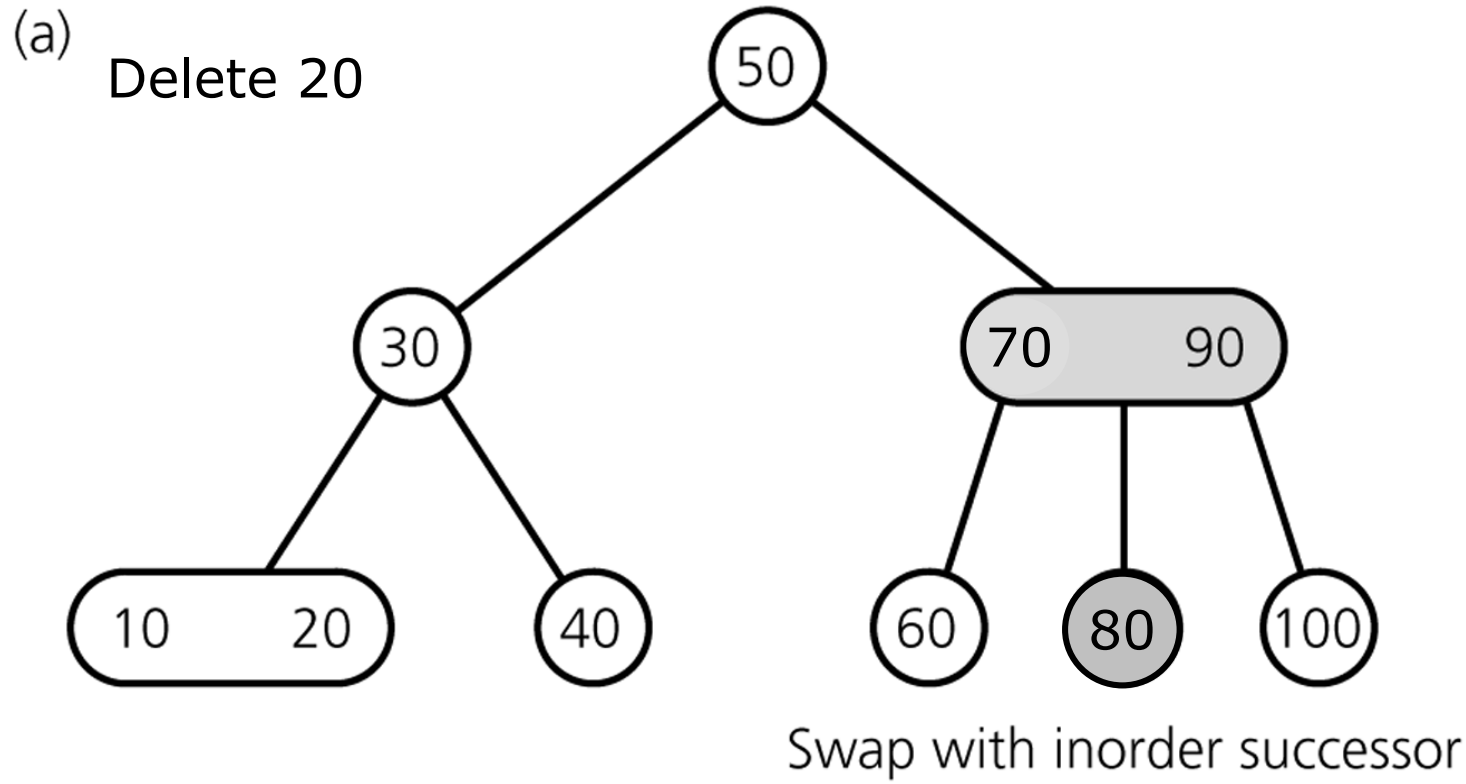
If tree on deletion is not 23 then you either redistribute or merge.

-Redistribution: When an empty node has a two item sibbling(recreates a 23 tree)

-Merging: When an empty node has a sibbling with single item(might need multiple iterations that propogates upwards until a 23 tree is restored)

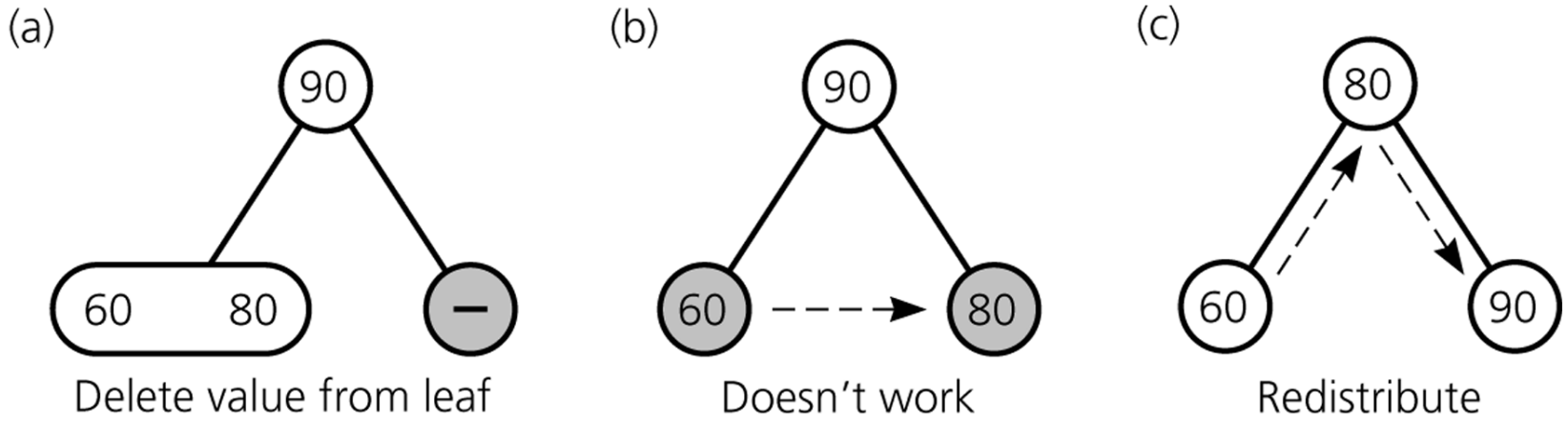
Case 1: two keys

If the leaf has two keys



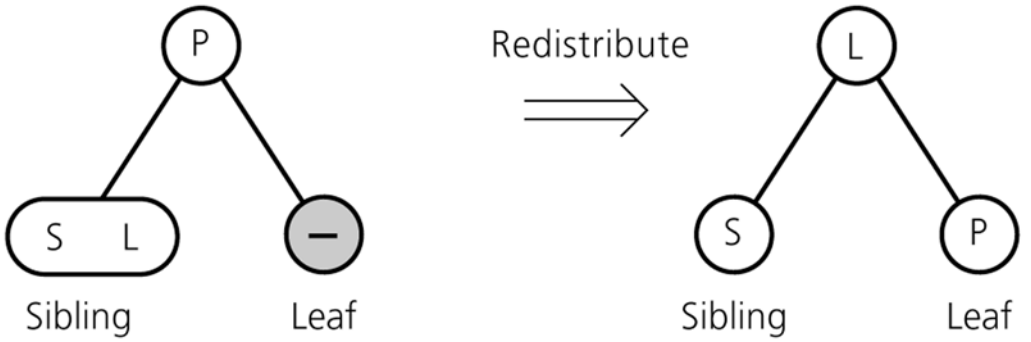
Case 2: Redistribution

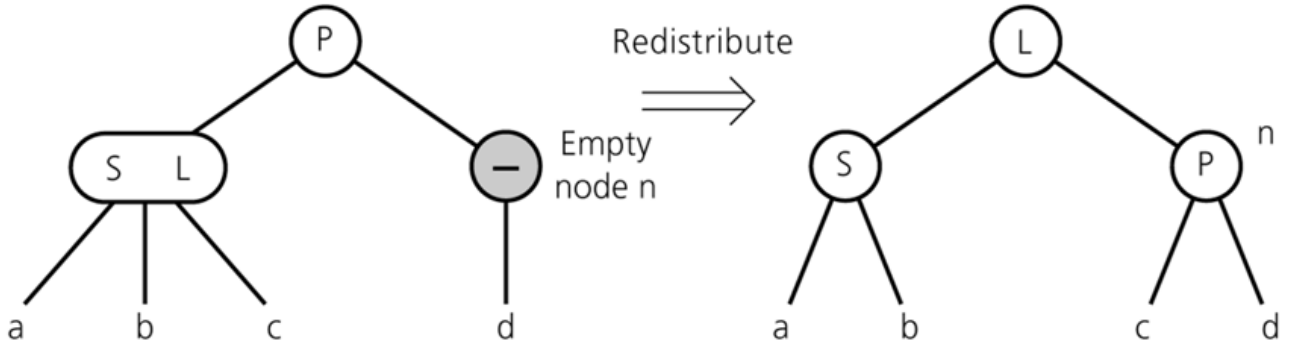
If item to be deleted is a single key and sibbling has two keys



Redistribution

A sibling has 2 items:
→ redistribute item between siblings and parent

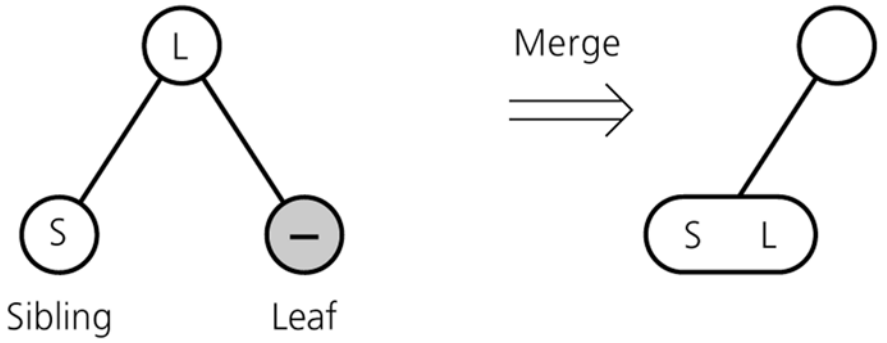


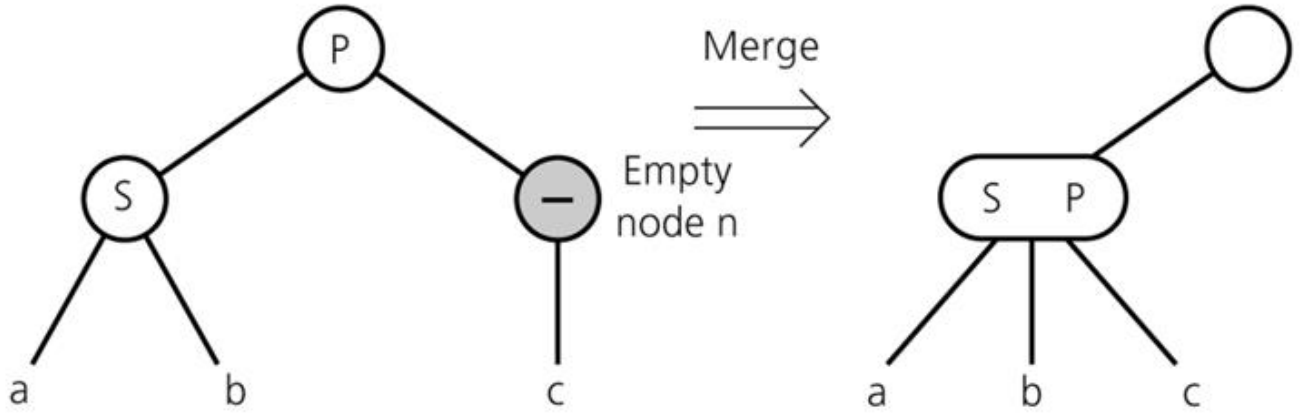


Case 3: Merging

When single key node
to be deleted and sibbling
Has single key

→ move item from parent to sibling
→ adopt child of *n*

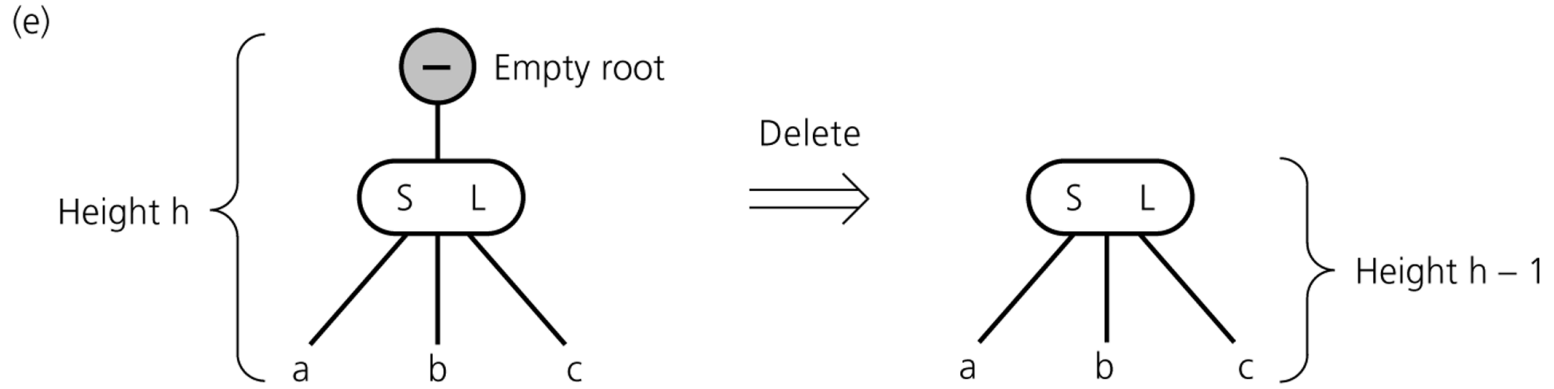




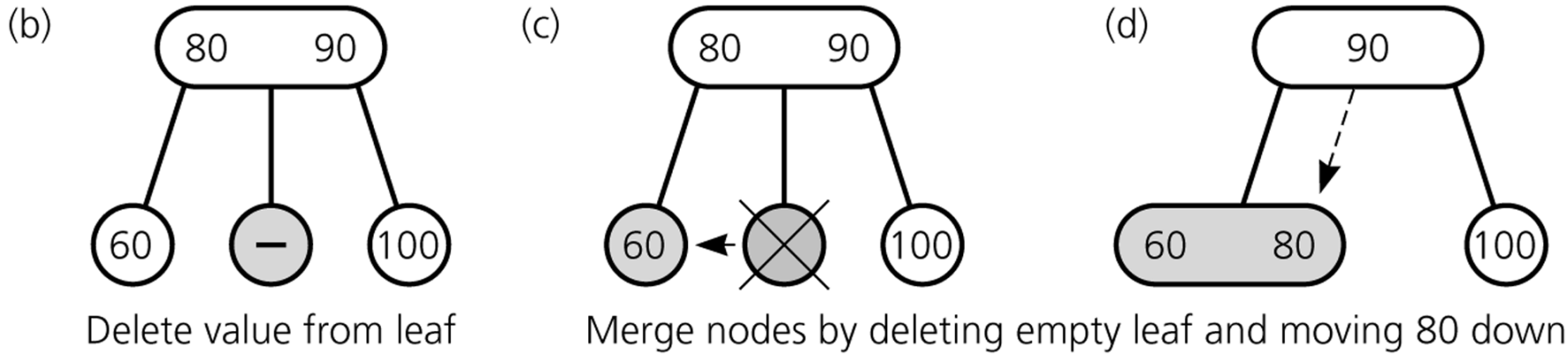
If *n*'s parent ends up without item, apply process recursively

Merging till root

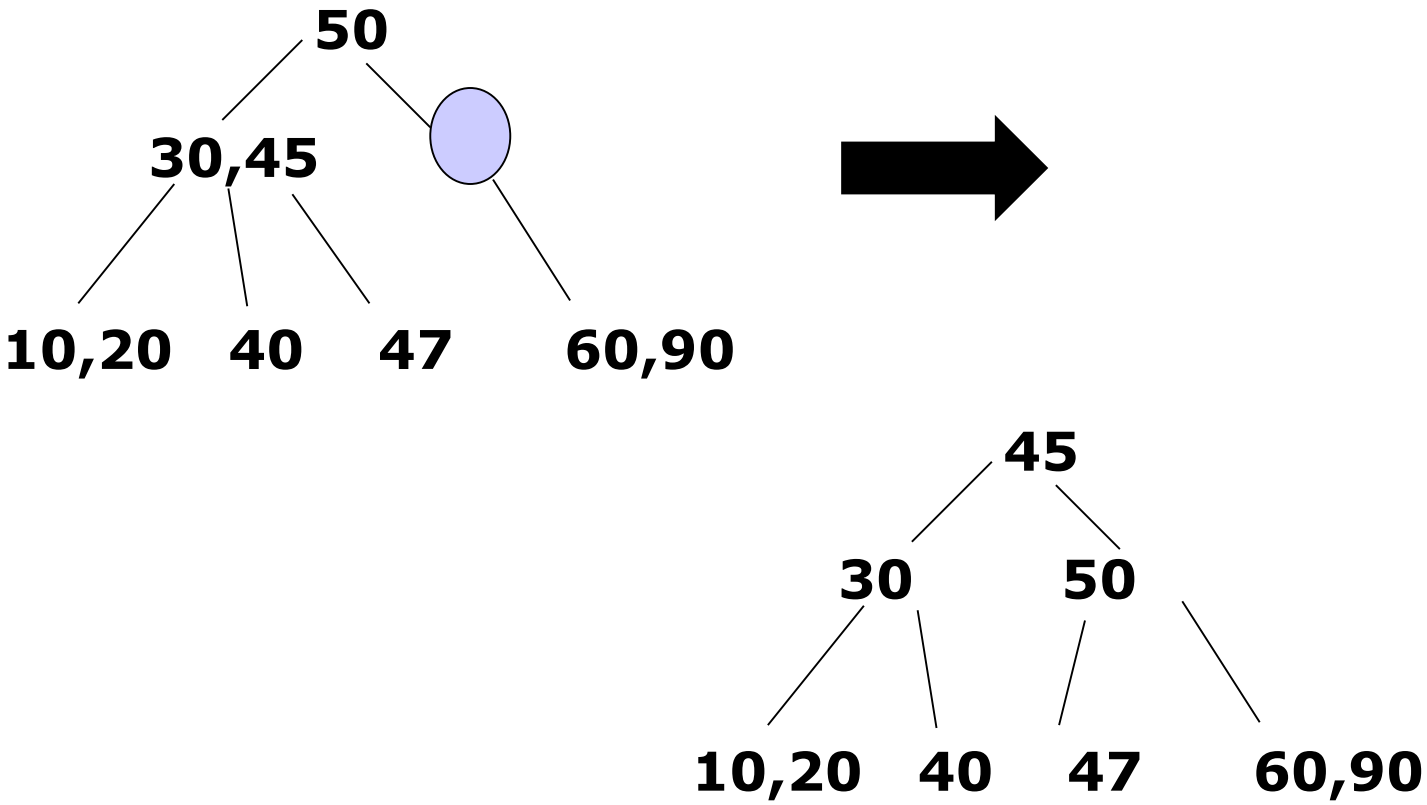
If merging process reaches the root and root is without item
→ delete root



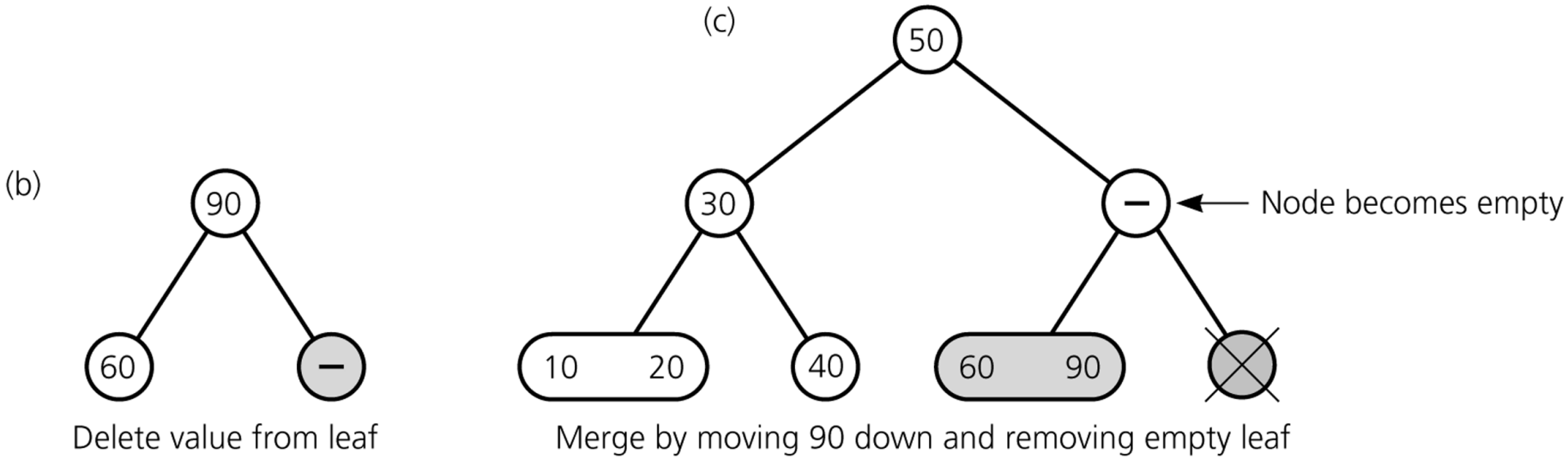
Merge with: double key node



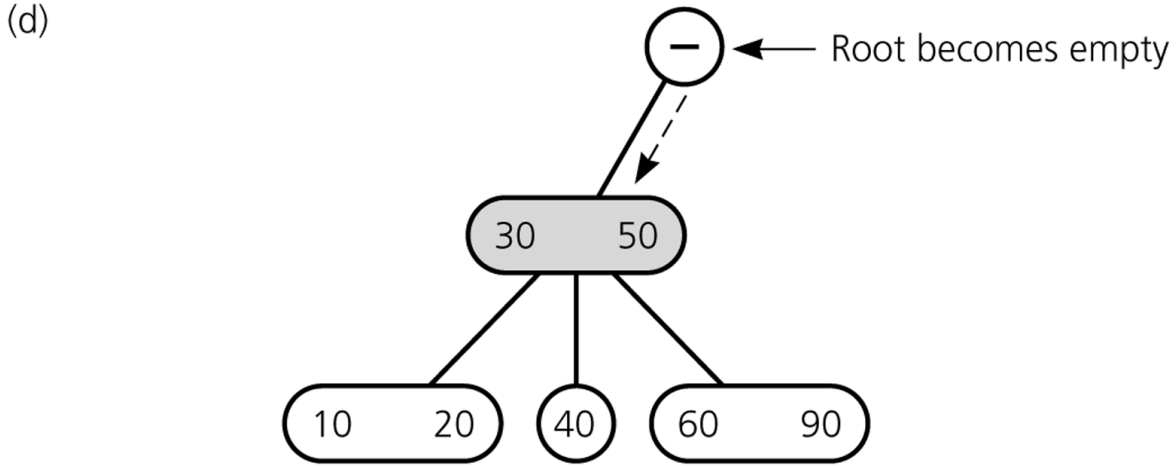
Try



Merge with single key sibbling -1



Example: Merge with single key sibbling -continued



Delete the following

After having Inserted the below
40 30 34 50 36 80 9 37 20 100 38

70 60 39 90 33 35 95

Delete 36, 40,60, 70,80,9 ,30

Conclusion

23 tree is a balanced binary tree satisfying three properties
-leaf nodes at same level
-every 2 node must have three children
-every 1 node must have two children

Insertion: Always at leaf. If no space push middle element up which might propagate till root increasing tree height.

Deletion: Starts at leaf. Three methods
-simple delete: removes one out of two elements in leaf node
- redistribution
- merging: but might propogate upwards till root. Can reduce tree height by 1.