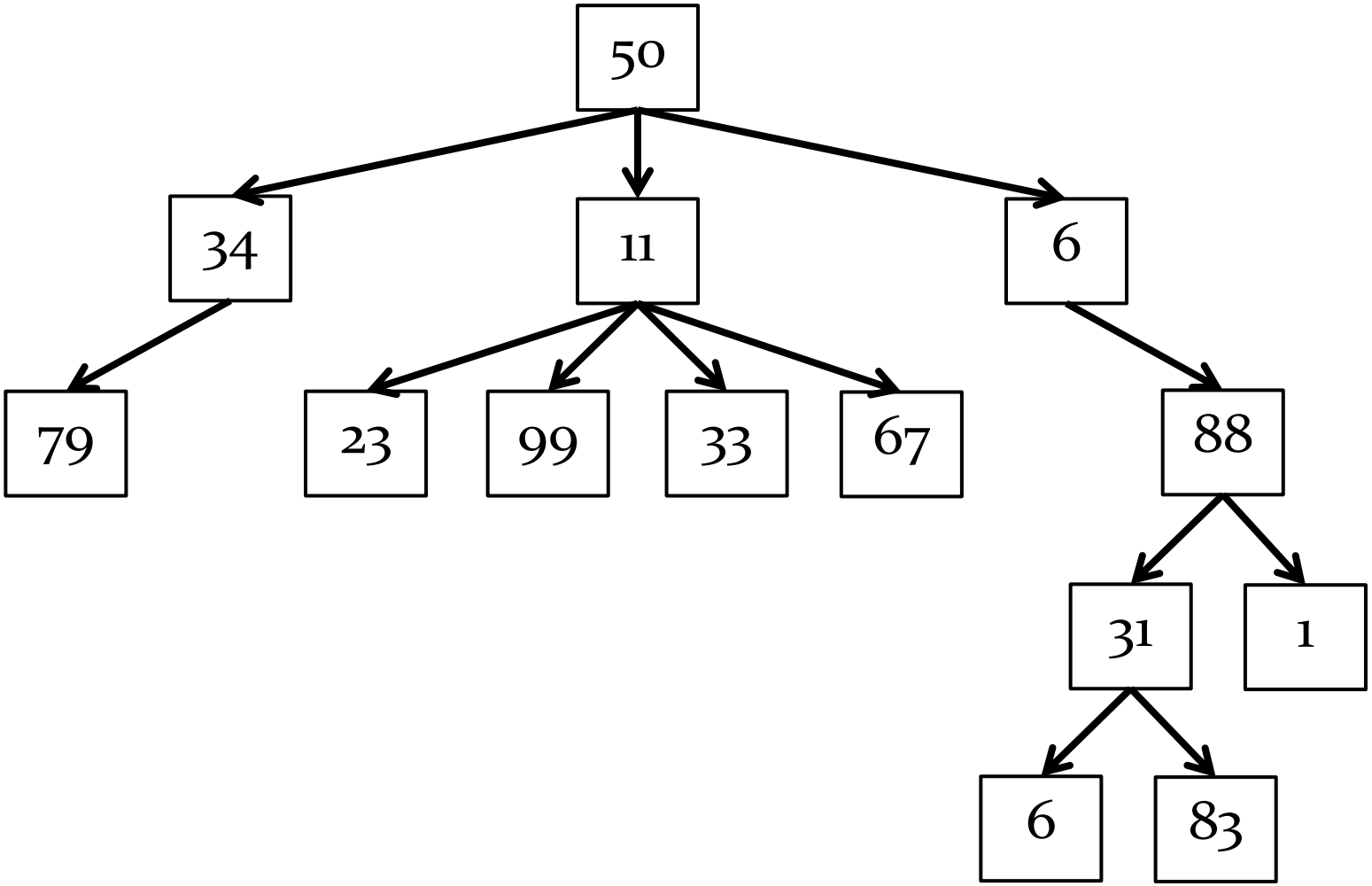
Trees

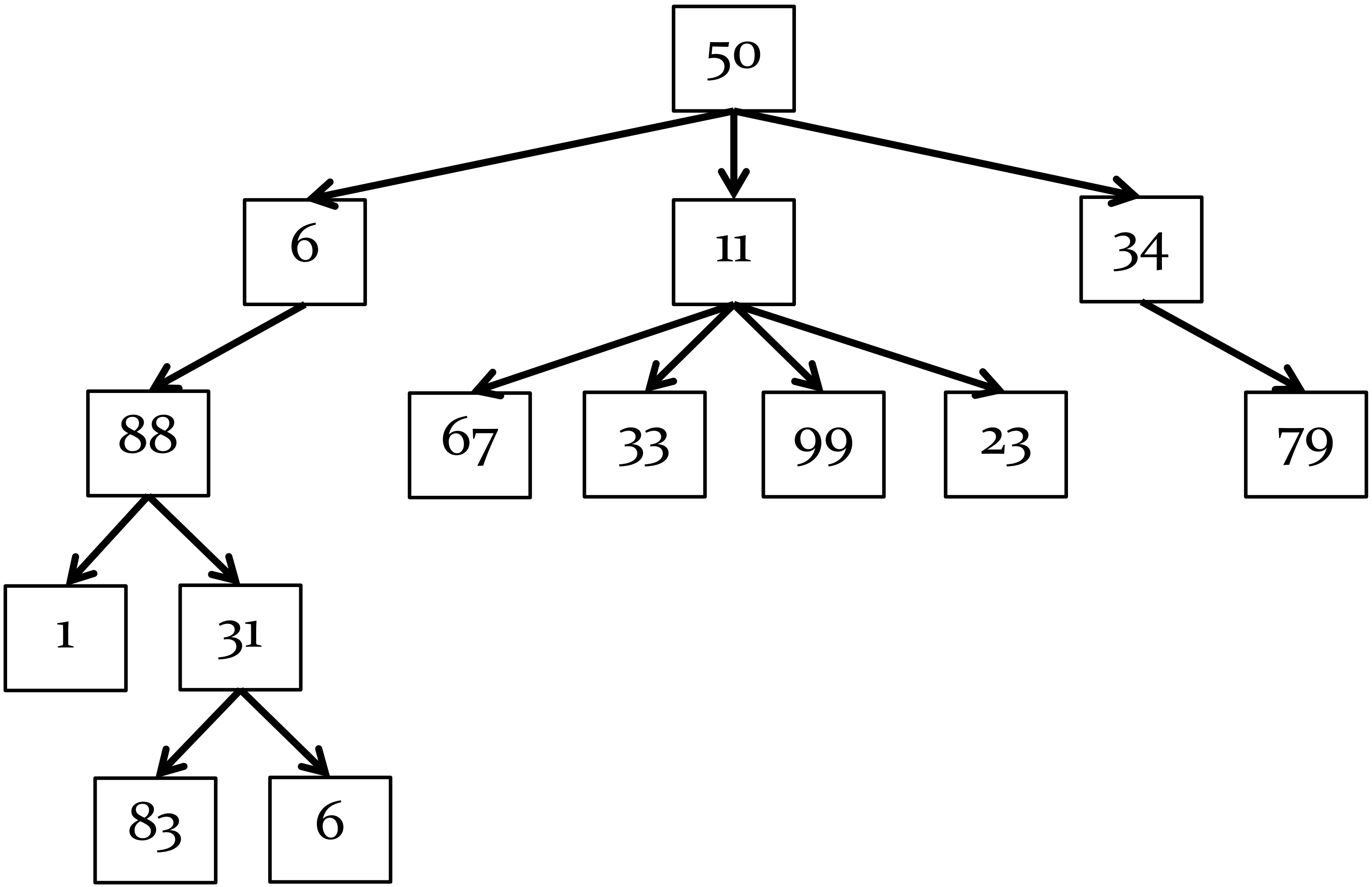
Graphs

- a graph is a data structure made up of nodes
  - each node stores data
  - each node has links to zero or more nodes
    - in graph theory the links are normally called *edges*
- graphs occur frequently in a wide variety of real-world problems
  - social network analysis
    - e.g., six-degrees-of-Kevin-Bacon, Facebook Friend Wheel
  - transportation networks
    - e.g., http://ac.fltmaps.com/en
  - many other examples
    - http://www.visualcomplexity.com/vc/

Trees

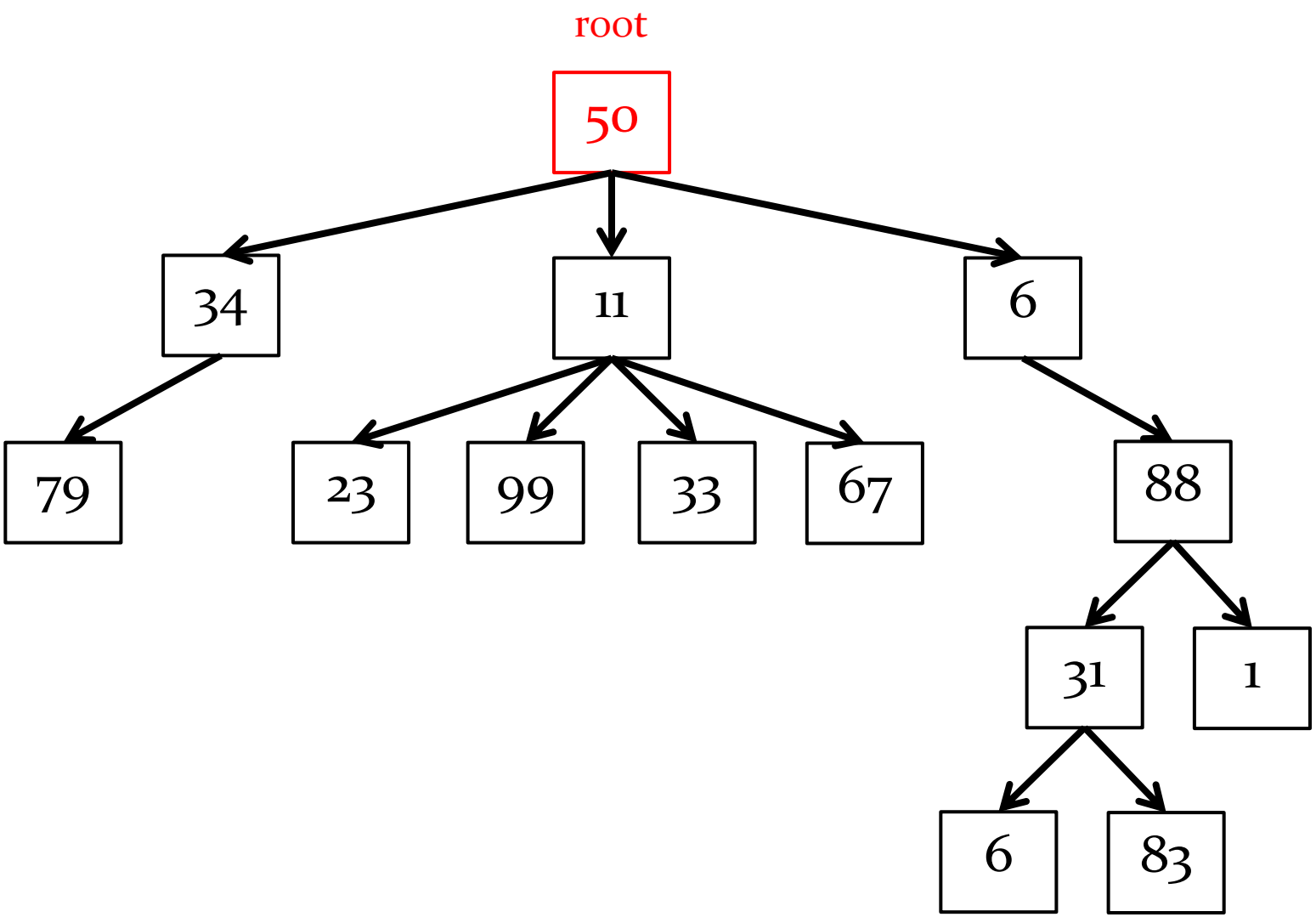
- trees are special cases of graphs
- a tree is a data structure made up of nodes
  - each node stores data
  - each node has links to zero or more nodes in the next level of the tree
    - children of the node
  - each node has exactly one parent node
    - except for the root node


50
34
11
6
79
23
99
33
67
88
31
1
6
83


50
6
11
34
88
67
33
99
23
79
1
31
83
6

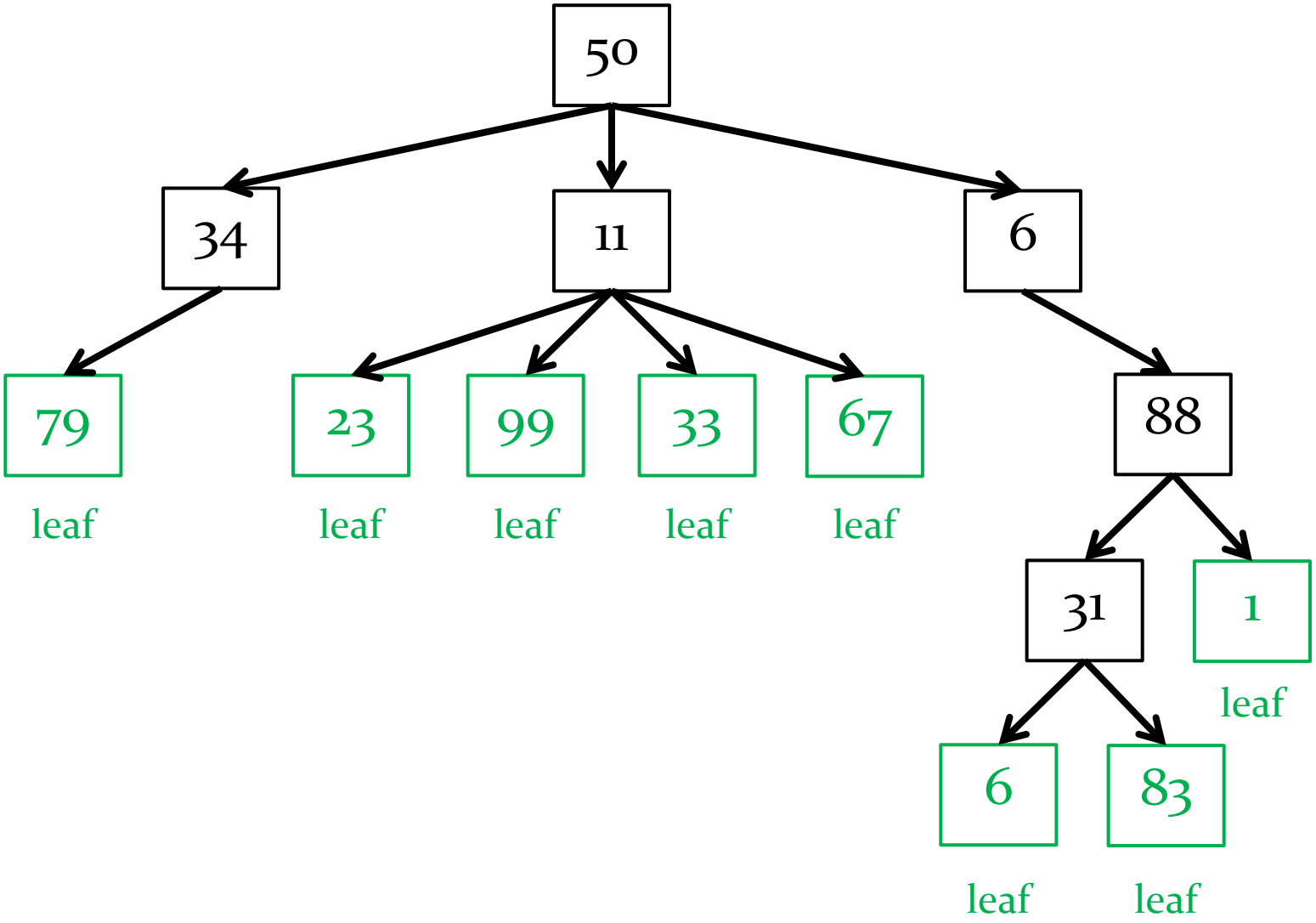
Trees

- the root of the tree is the node that has no parent node
- all algorithms start at the root


root
50
34
11
6
79
23
99
33
67
88
31
1
6
83

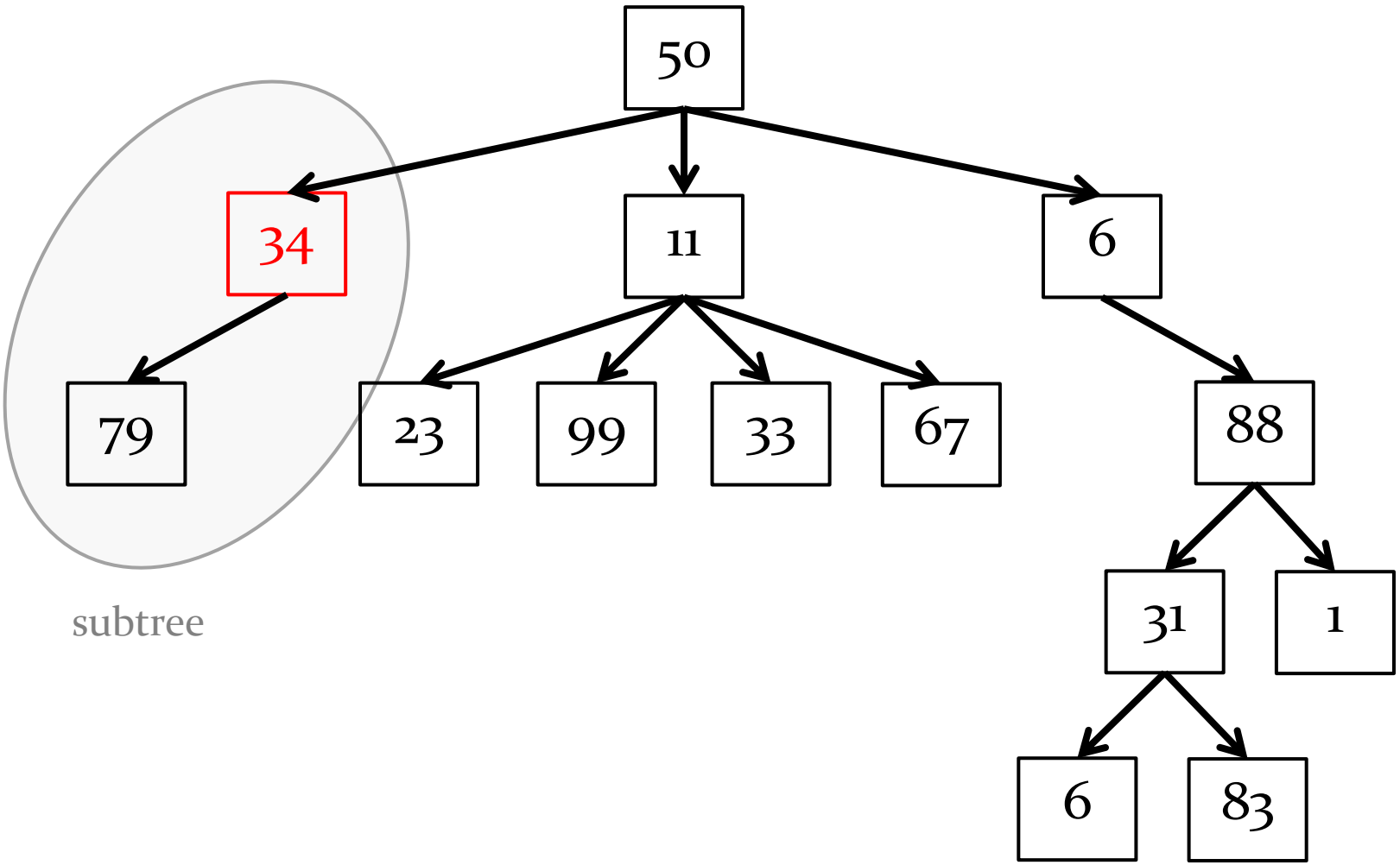
Trees

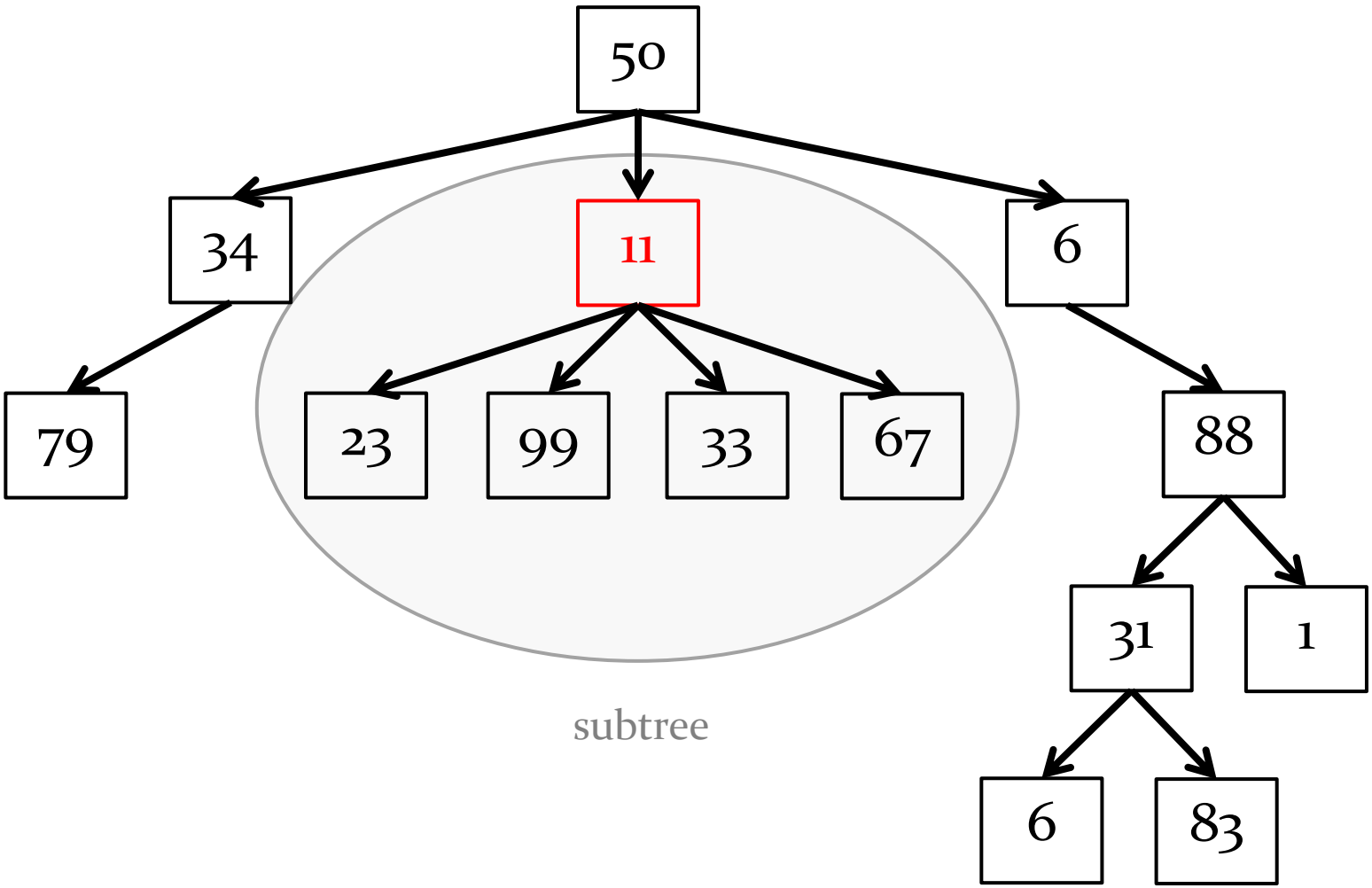
- a node without any children is called a leaf

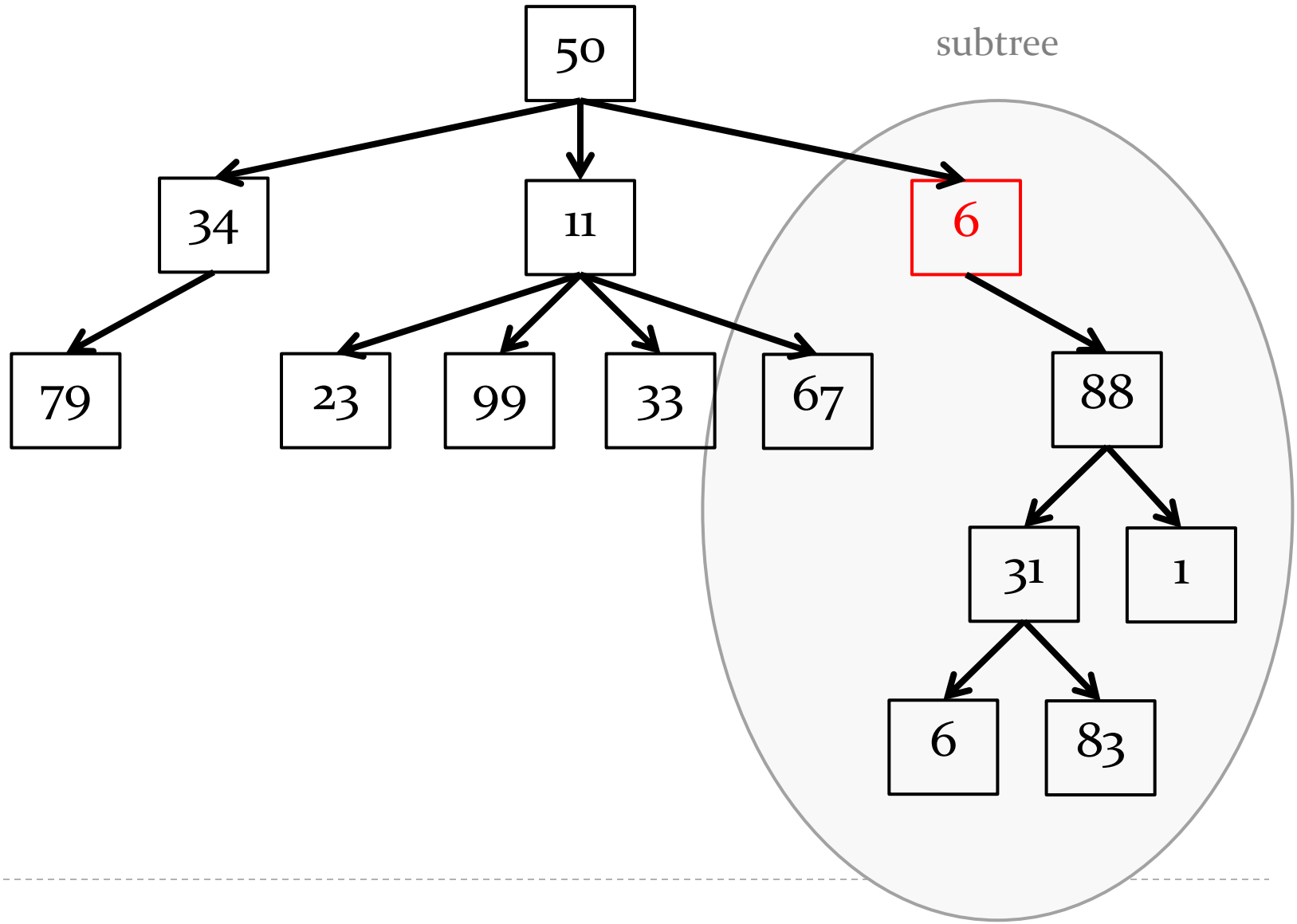

50
34
11
6
79
23
99
33
67
88
leaf
leaf
leaf
leaf
leaf
31
1
leaf
6
83
leaf
leaf

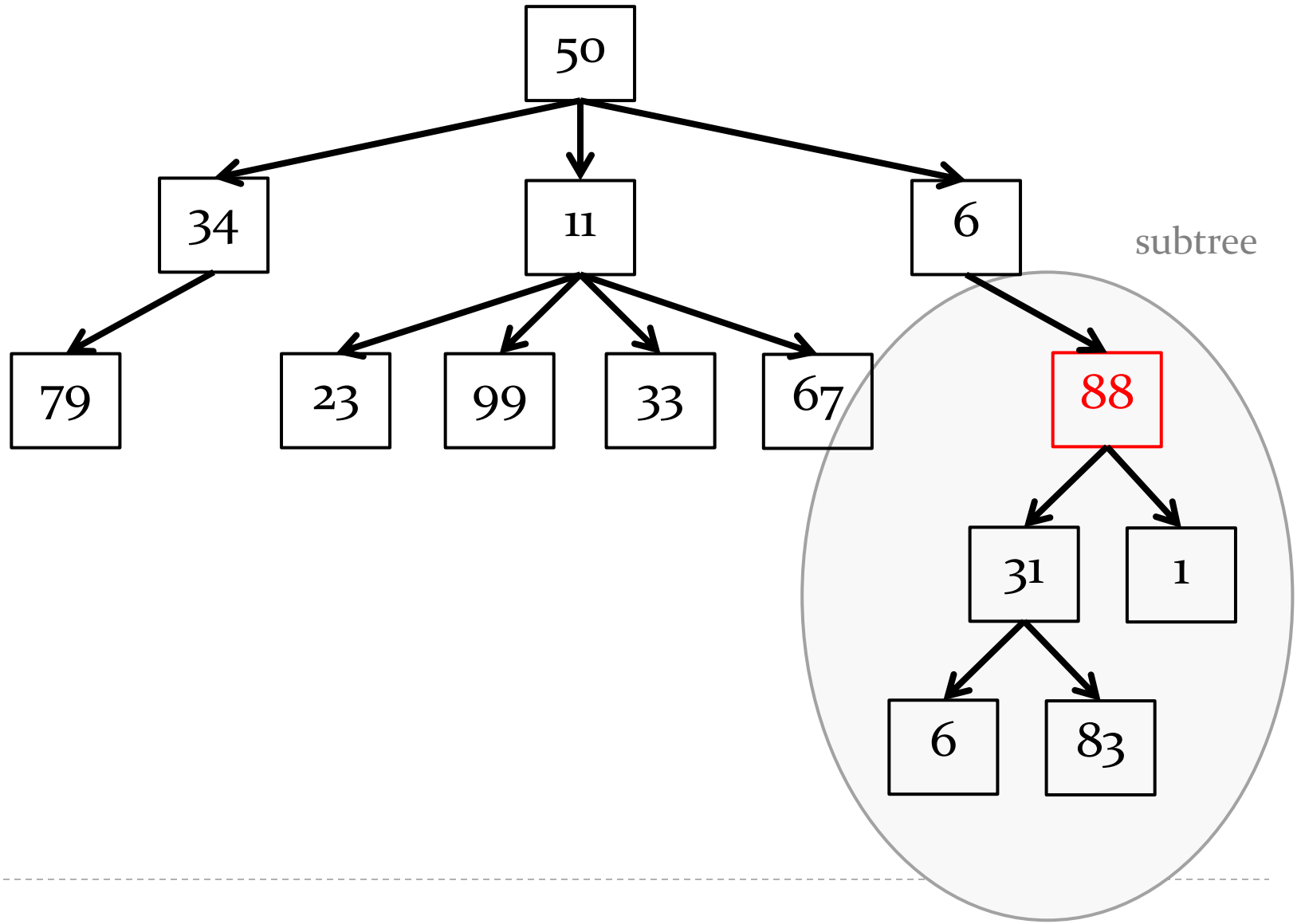
Trees

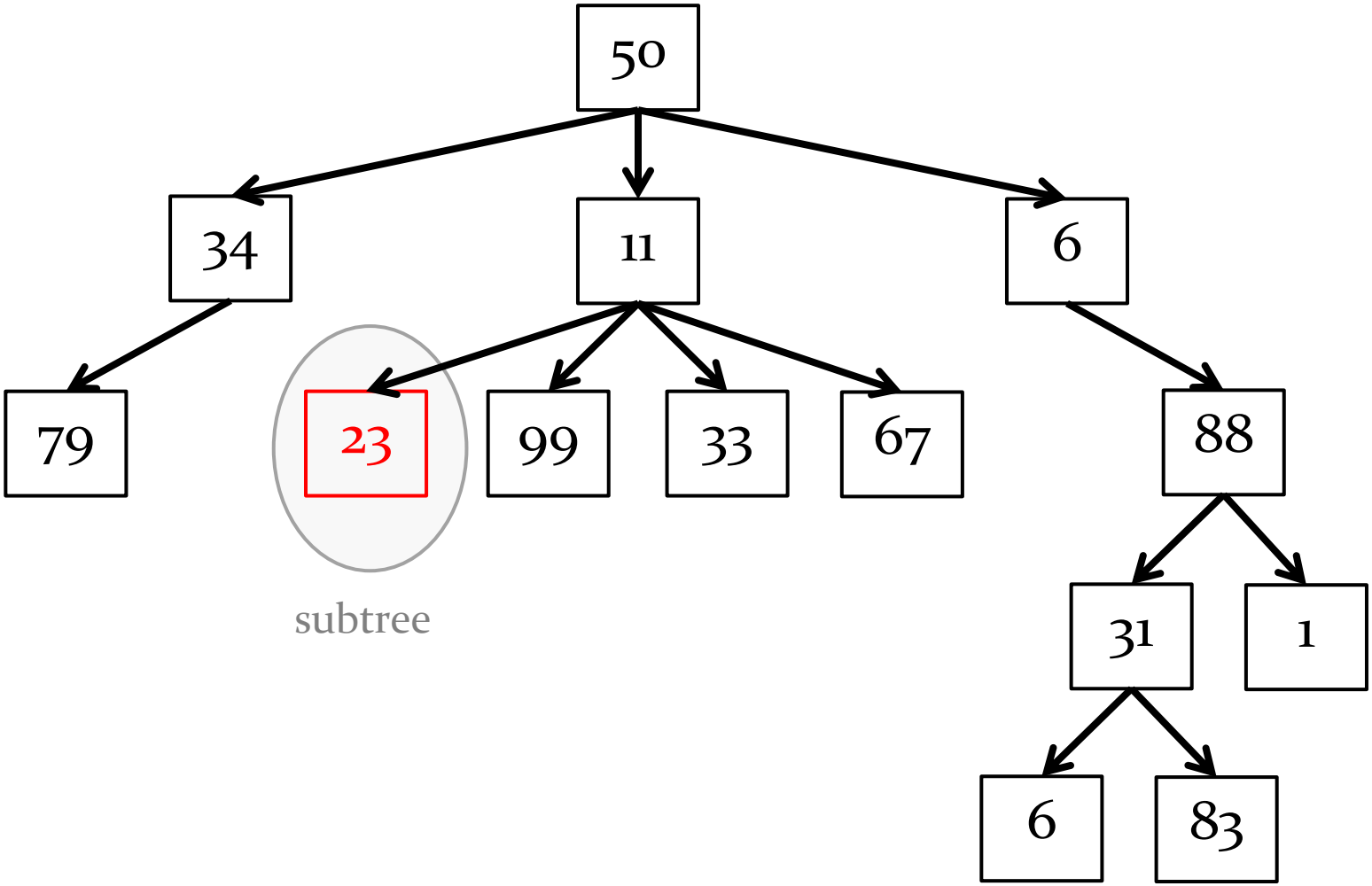
- the recursive structure of a tree means that every node is the root of a tree


50
34
11
6
79
23
99
33
67
88
31
1
6
83
subtree


50
34
11
6
79
23
99
33
67
88
31
1
6
83
subtree

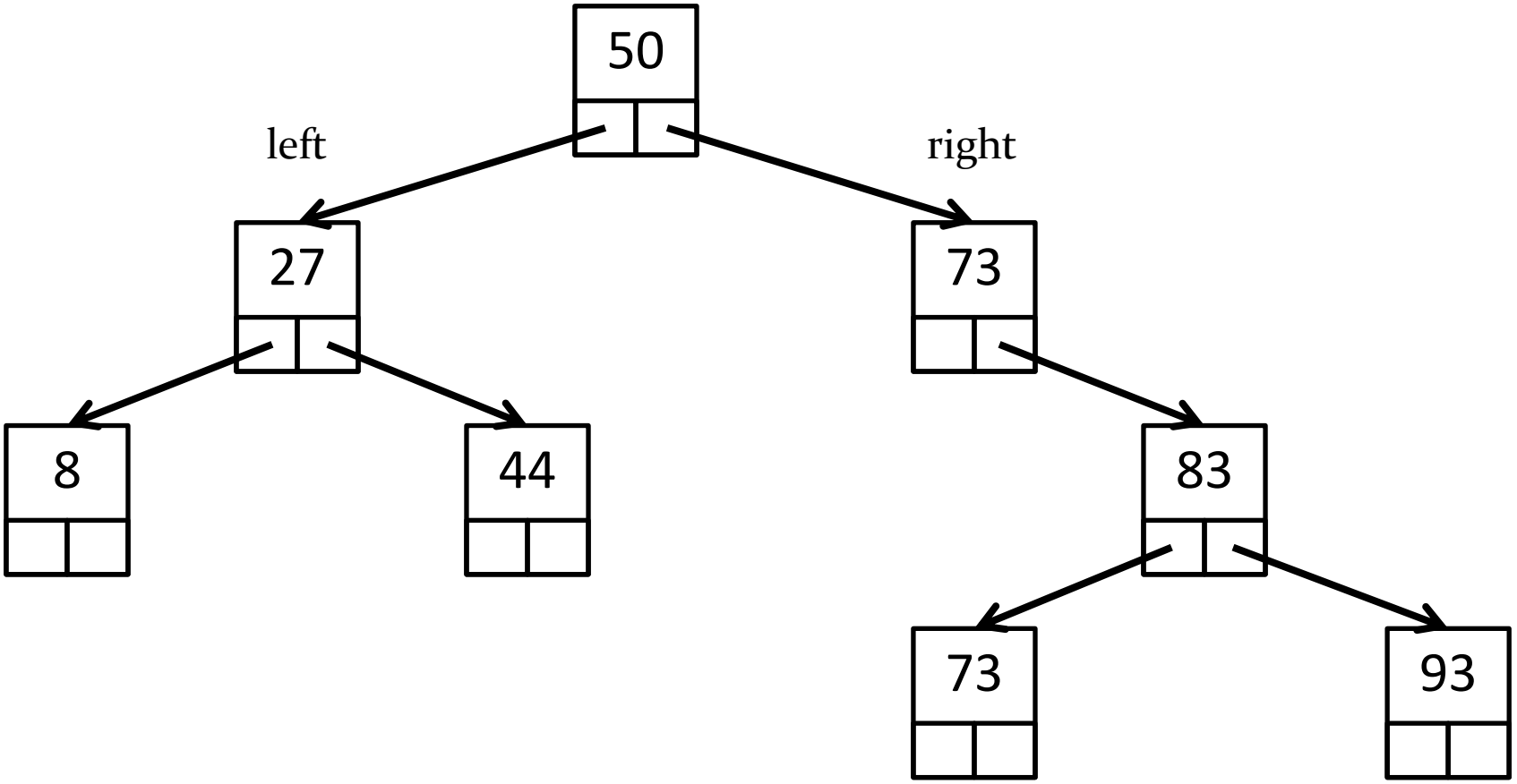

50
subtree
34
11
6
79
23
99
33
67
88
31
1
6
83

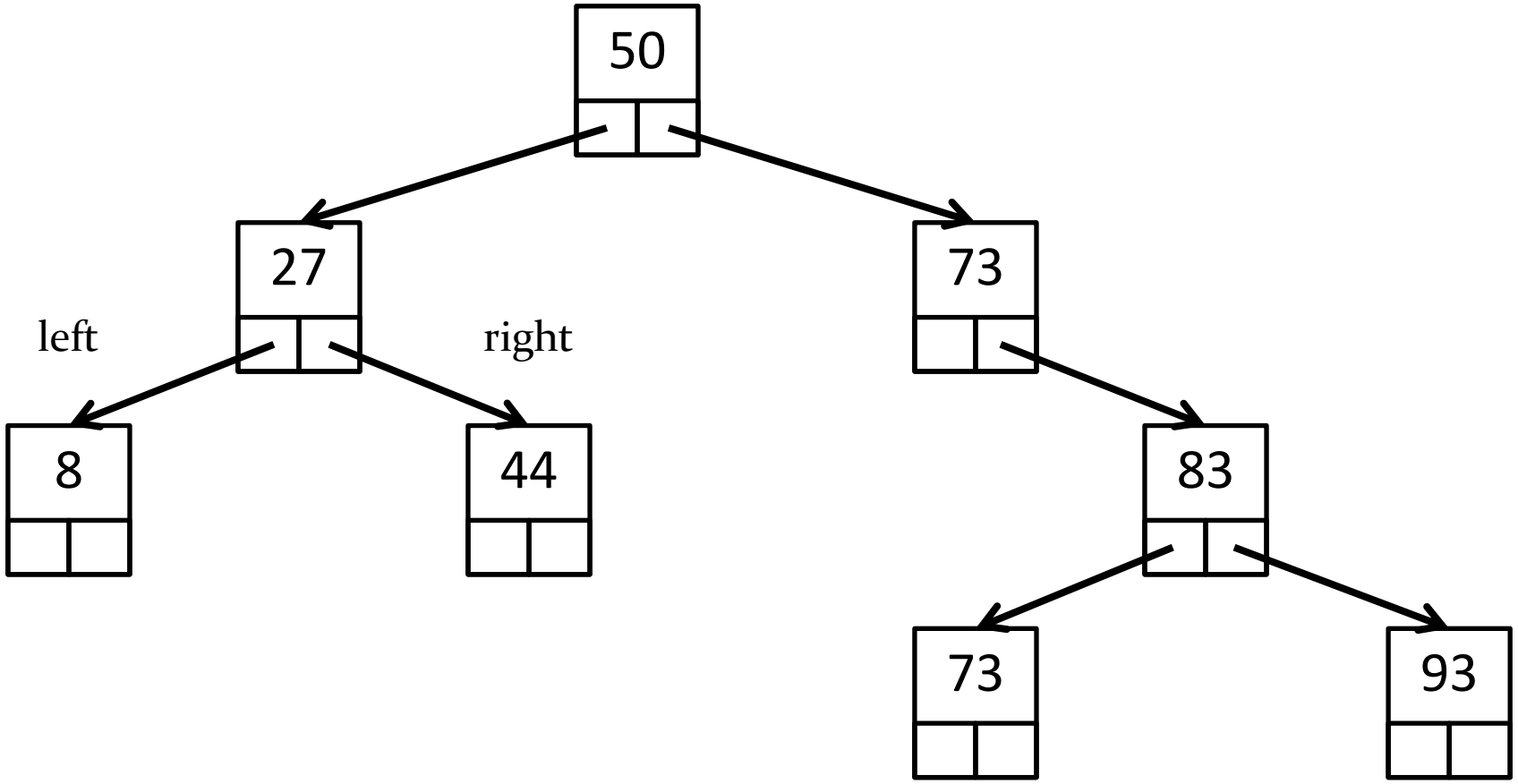

50
34
11
6
subtree
79
23
99
33
67
88
31
1
6
83

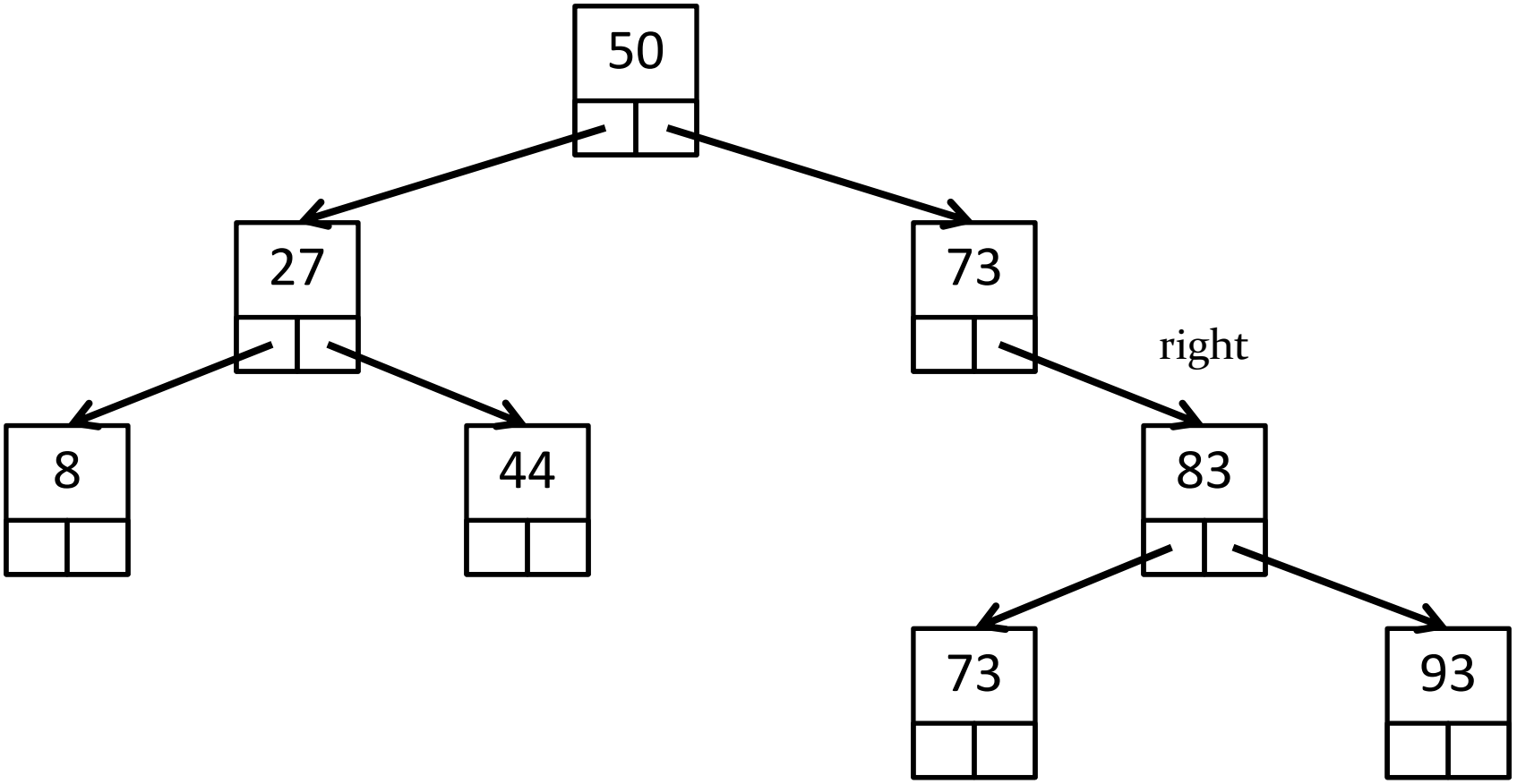

50
34
11
6
79
23
99
33
67
88
subtree
31
1
6
83

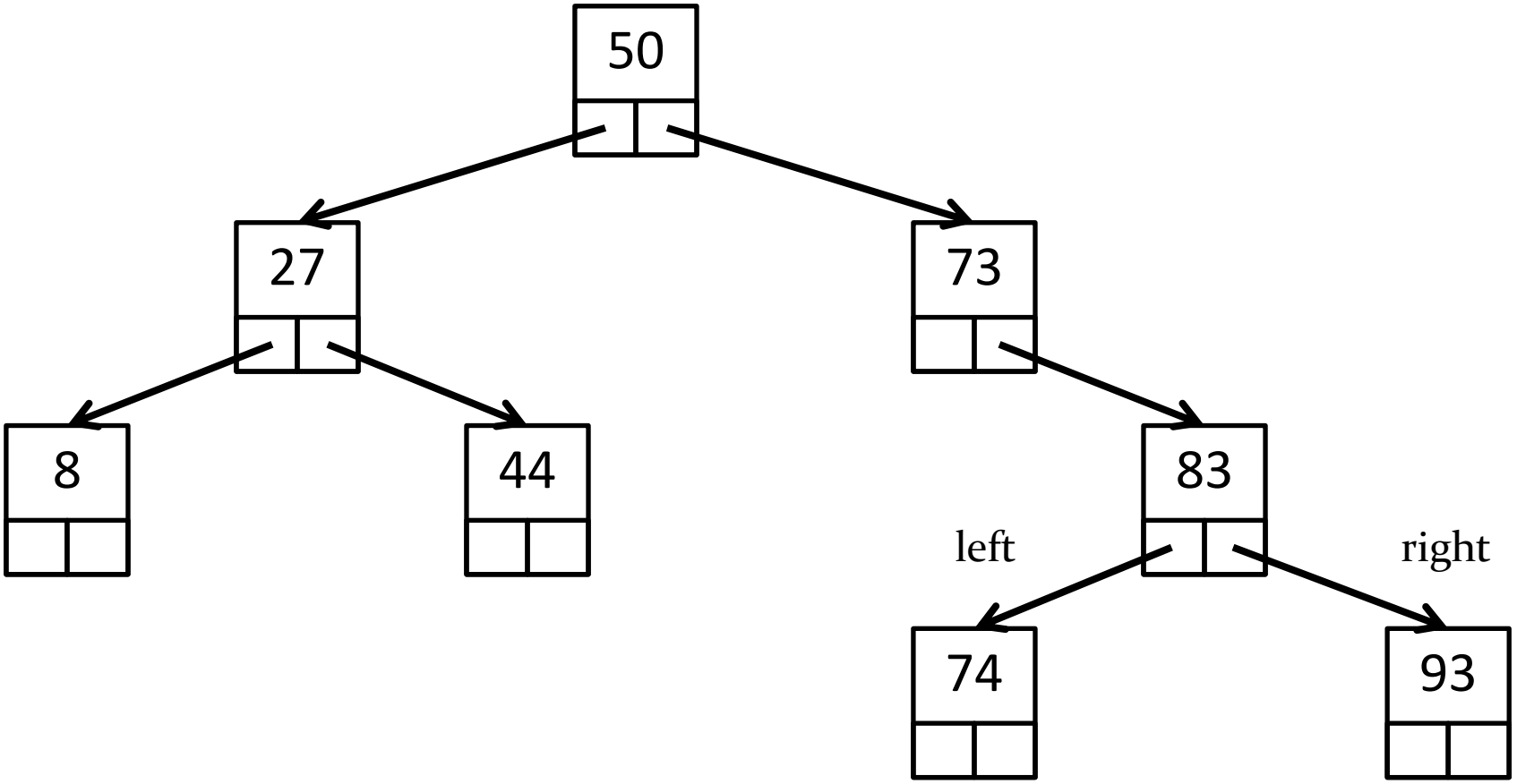
Binary Tree

- a binary tree is a tree where each node has at most two children
  - very common in computer science
  - many variations
- traditionally, the children nodes are called the left node and the right node


50
left
right
27
73
8
44
83
73
93


50
27
73
left
right
8
44
83
73
93


50
27
73
right
8
44
83
73
93


50
27
73
8
44
83
left
right
74
93

Binary Tree Algorithms

- the recursive structure of trees leads naturally to recursive algorithms that operate on trees
- for example, suppose that you want to search a binary tree for a particular element

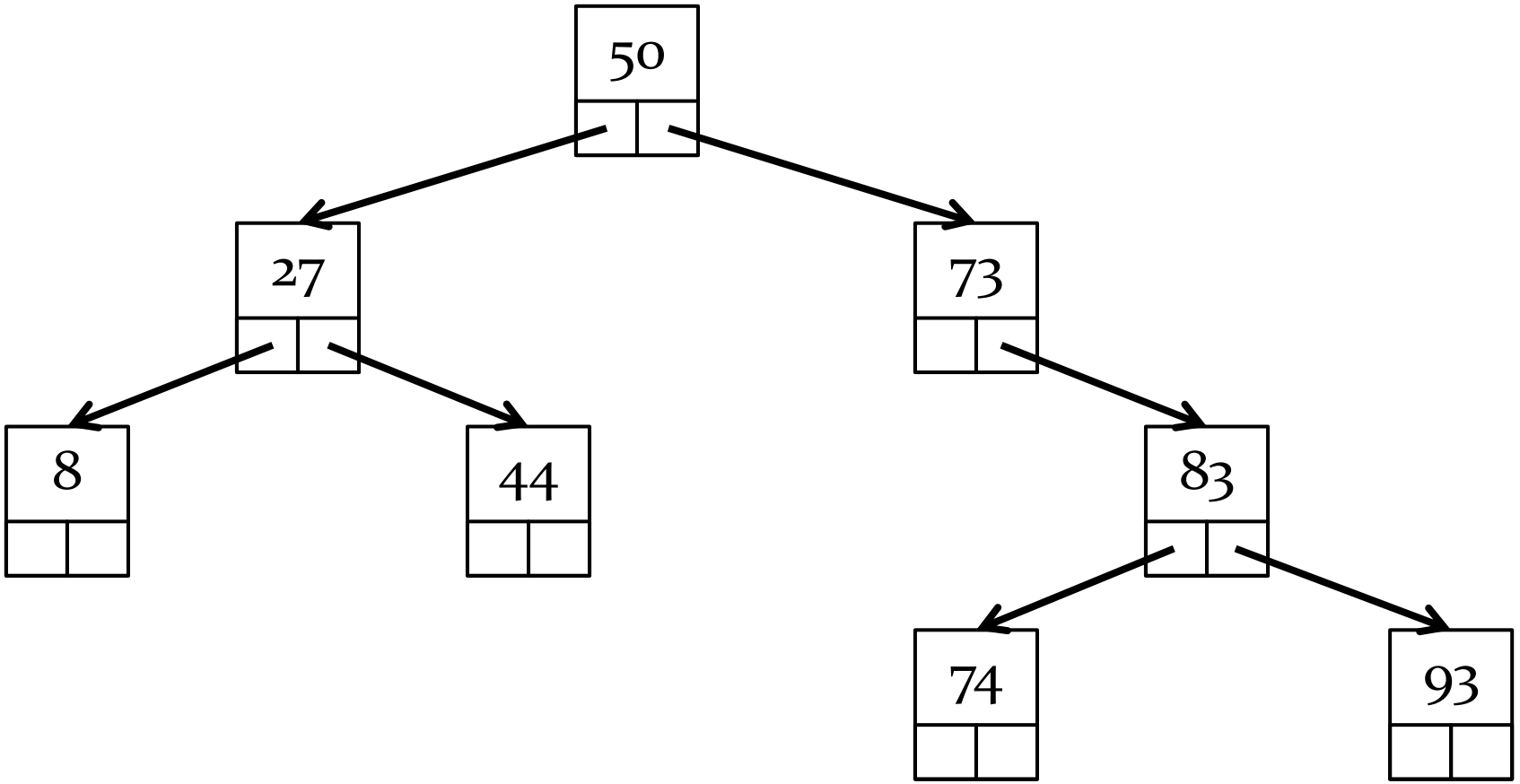
```
public static <E> boolean contains(E element, Node<E> node) {
  if (node == null) {
    return false;
  }
  if (element.equals(node.data)) {
    return true;
  }
  boolean inLeftTree = contains(element, node.left);
  if (inLeftTree) {
    return true;
  }
  boolean inRightTree = contains(element, node.right);
  return inRightTree;
}
```

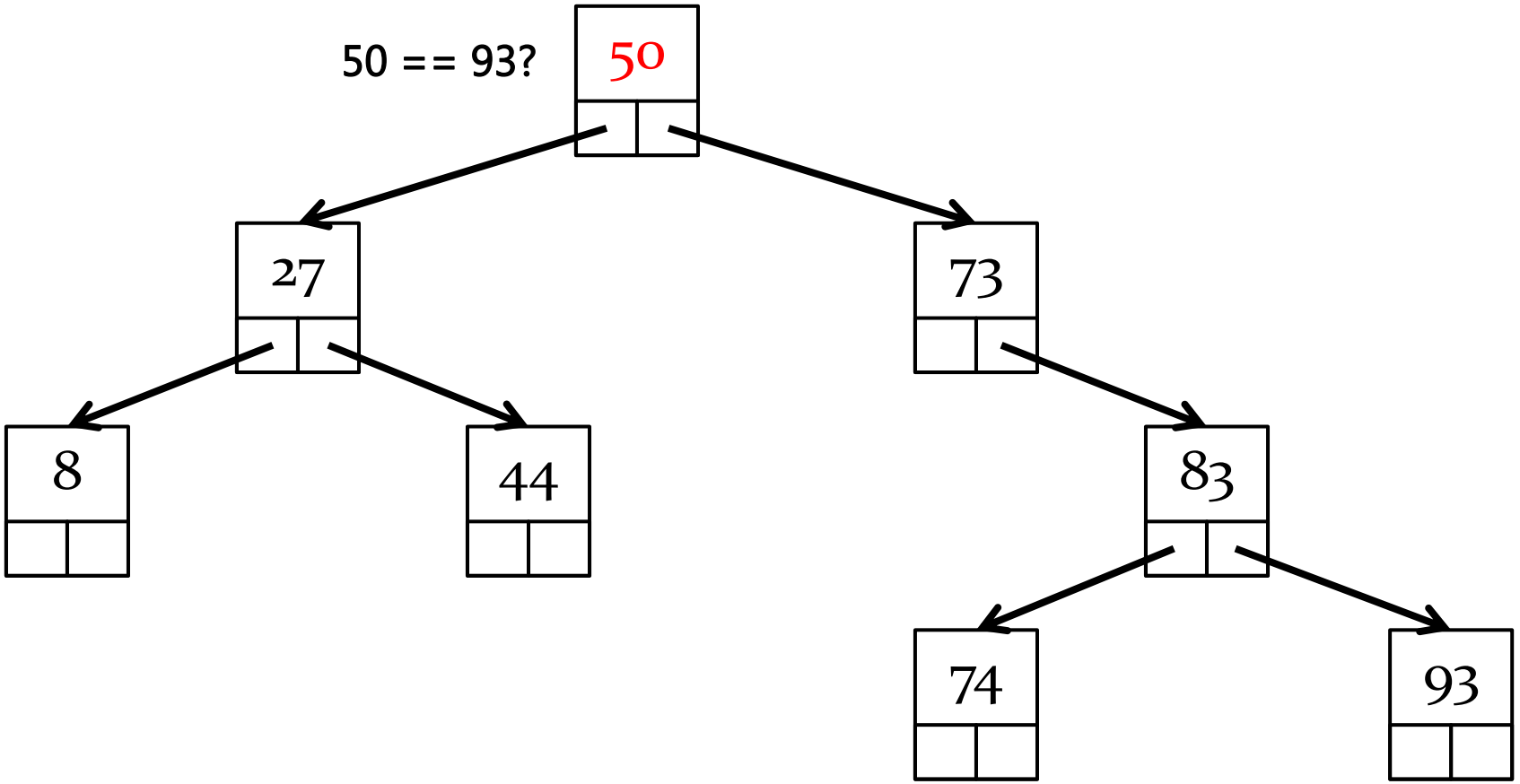
examine root

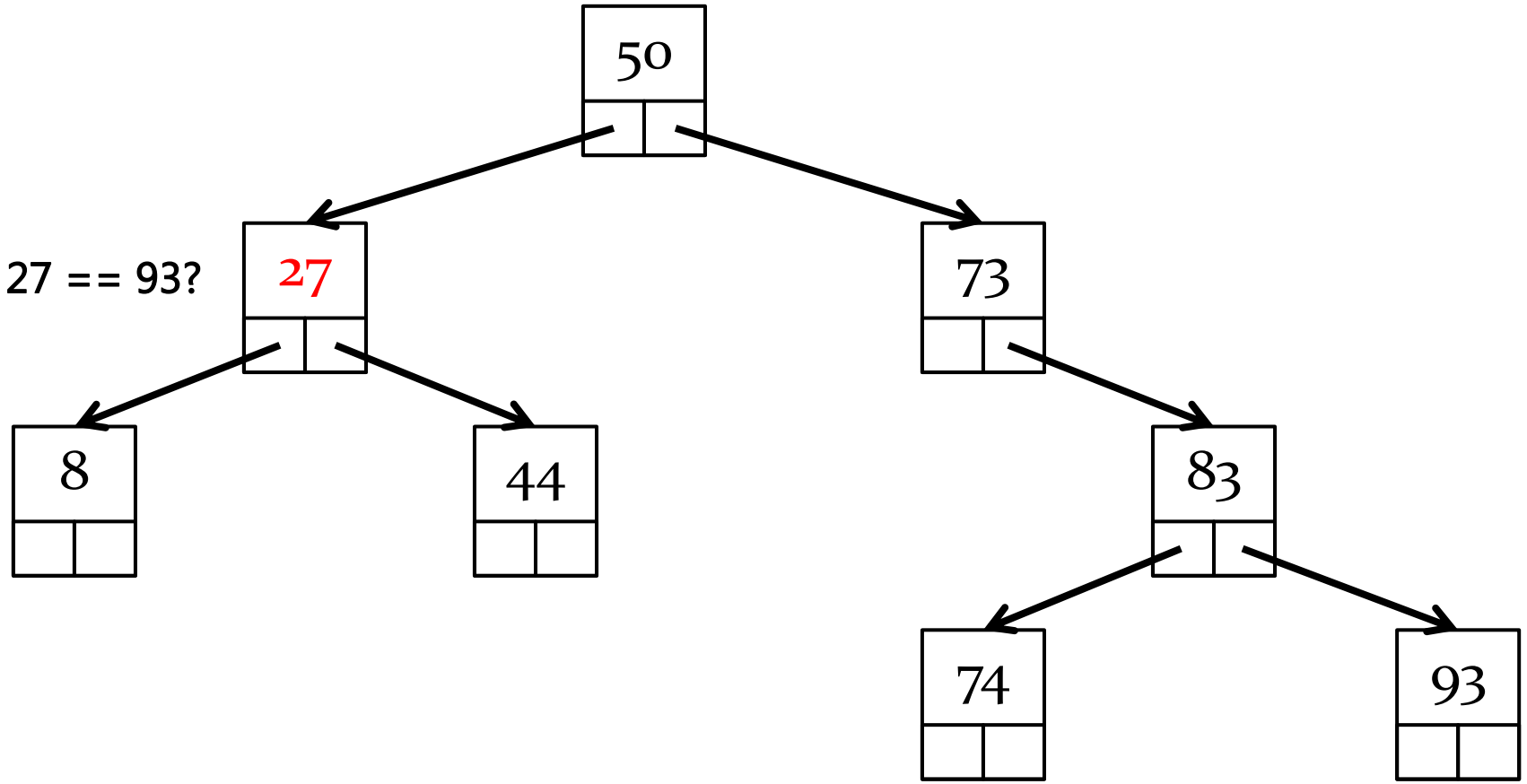
examine left subtree

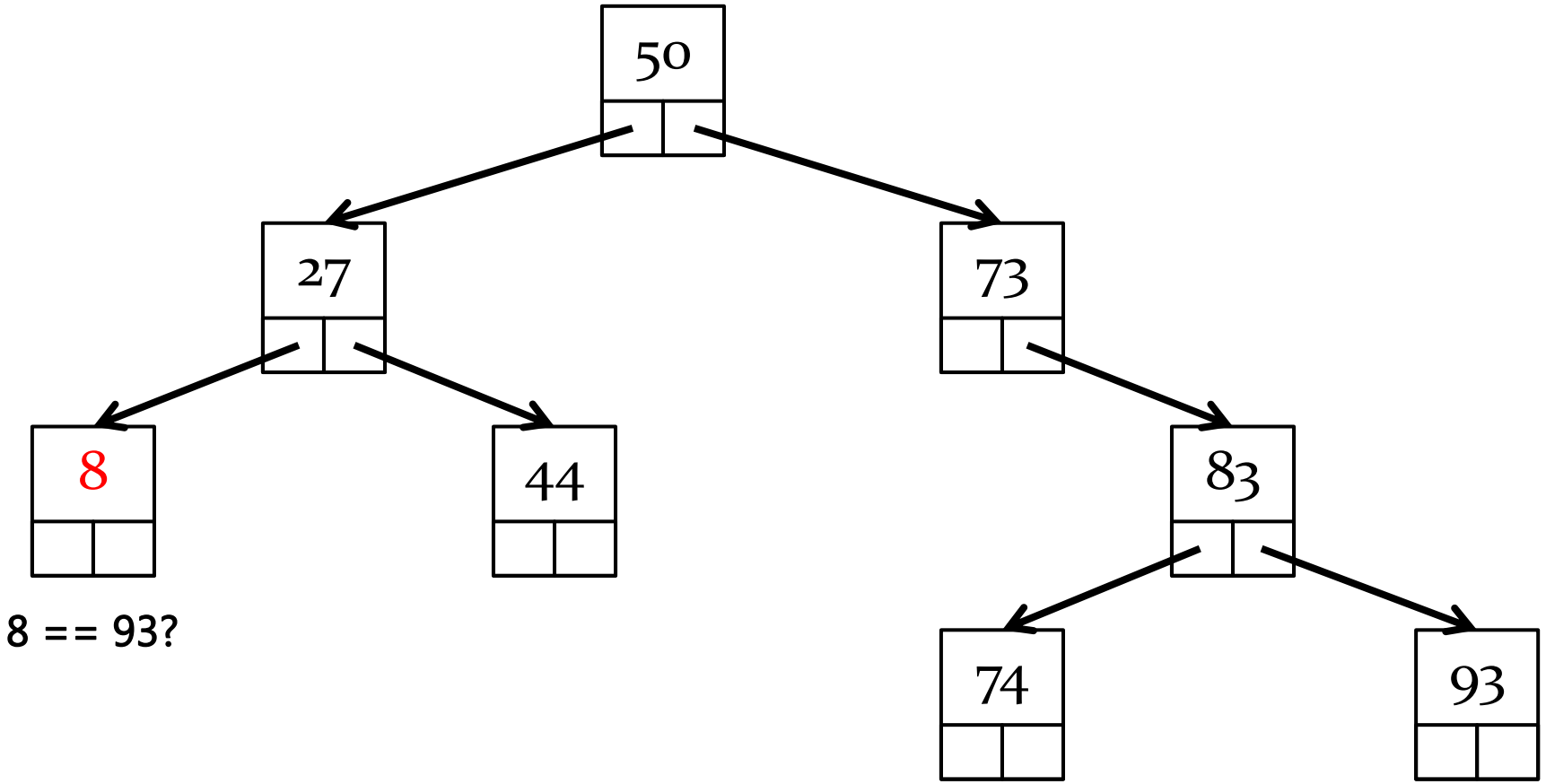
examine right subtree

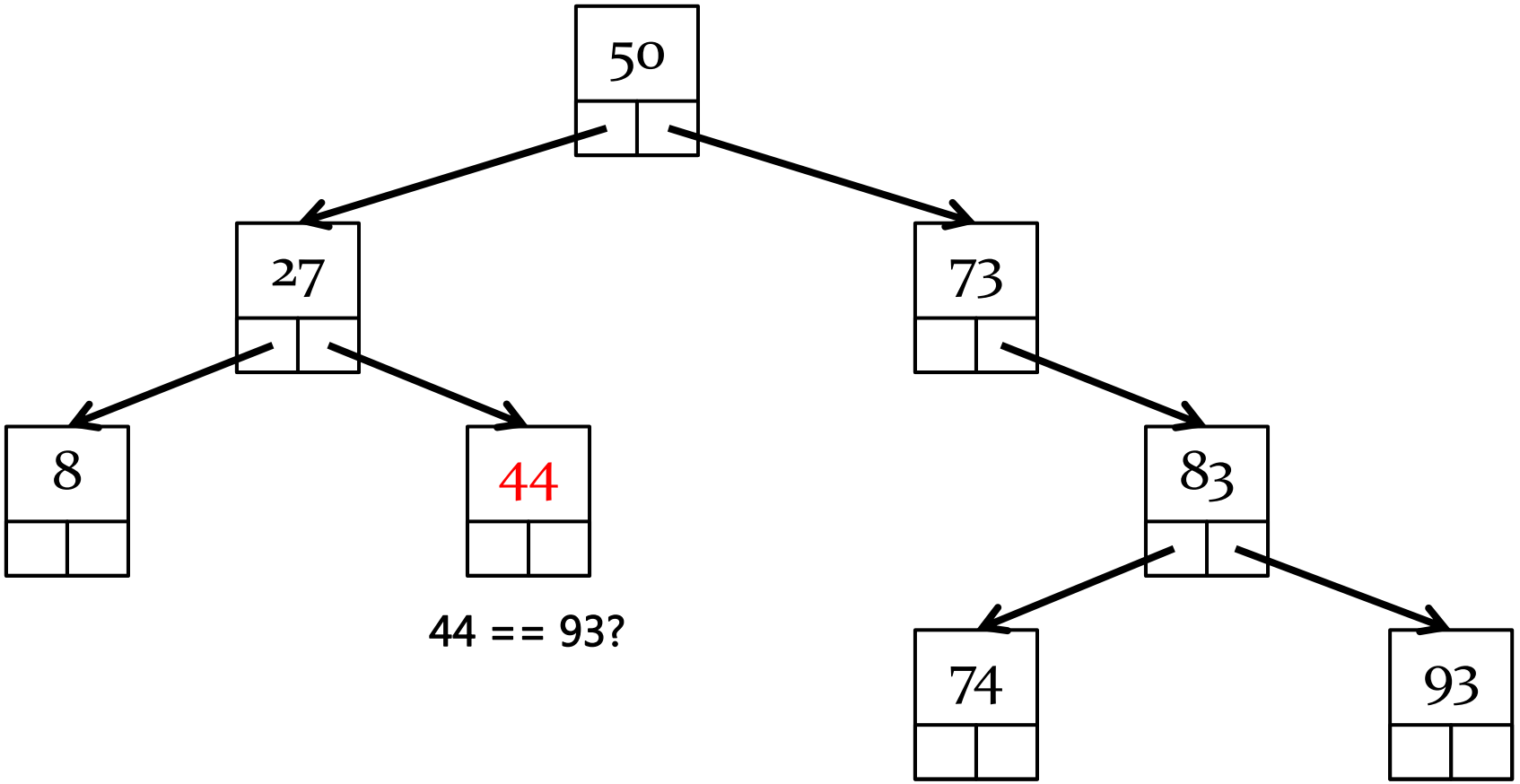
t.contains(93)

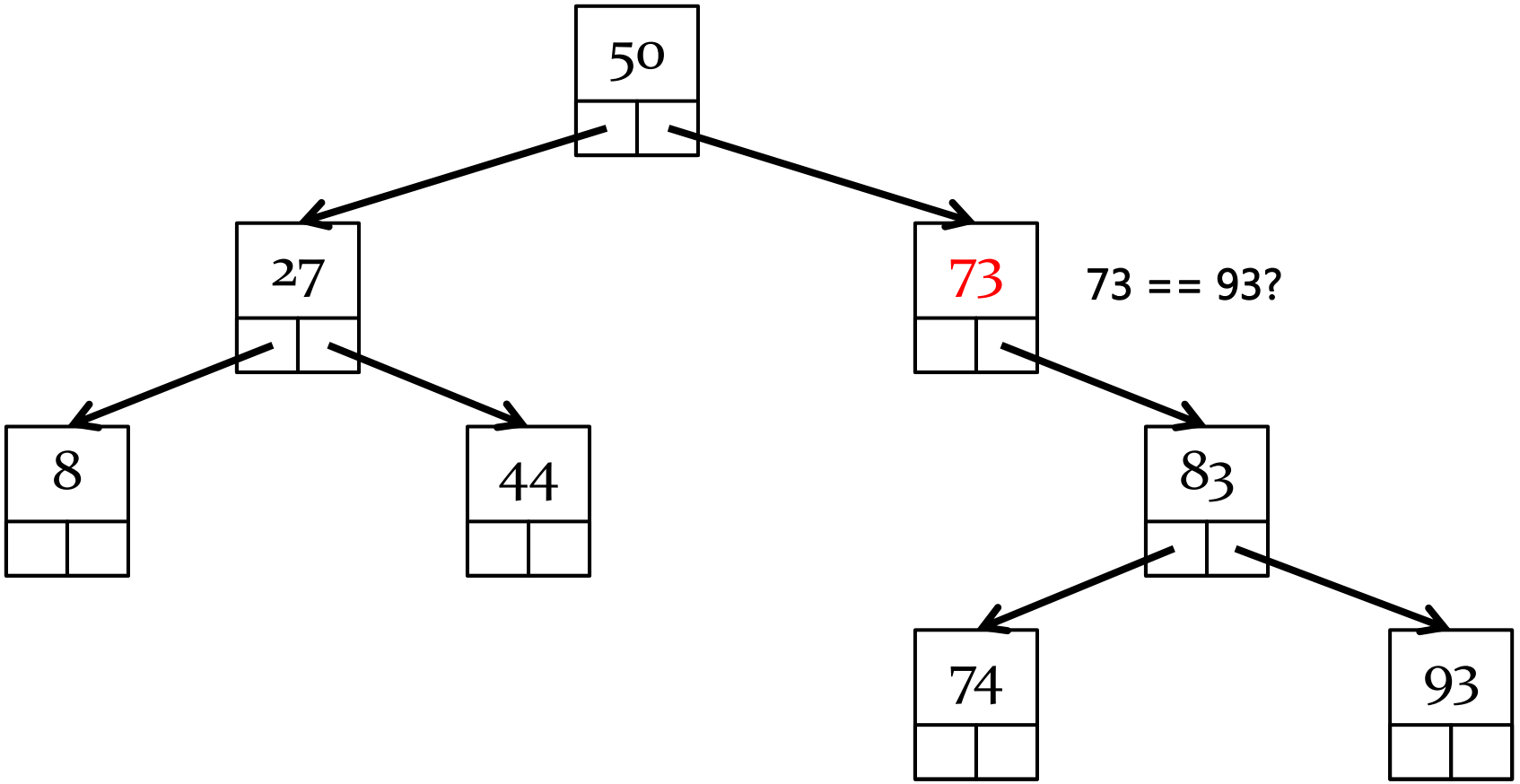


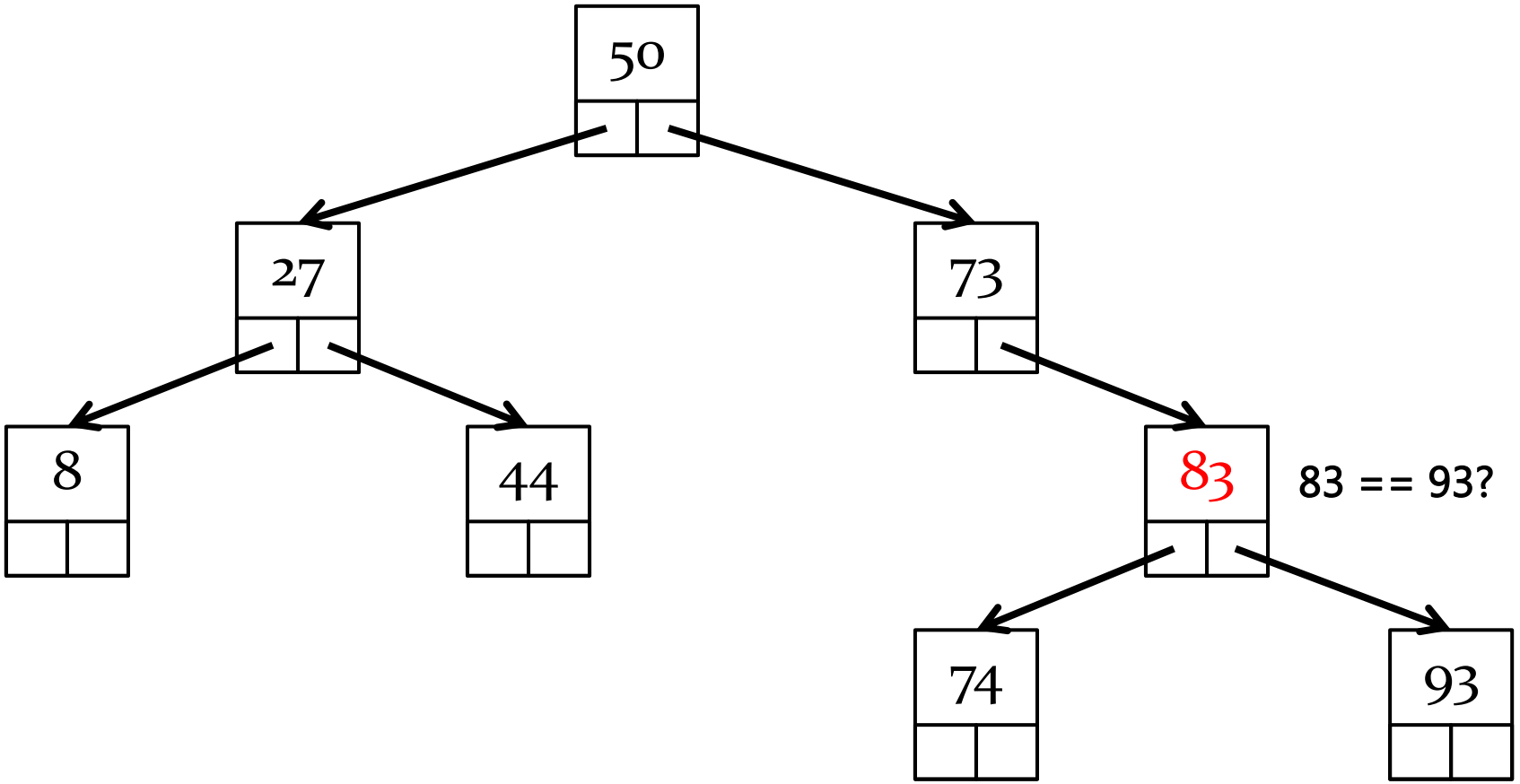

50 == 93?
50
27
73
8
44
83
74
93

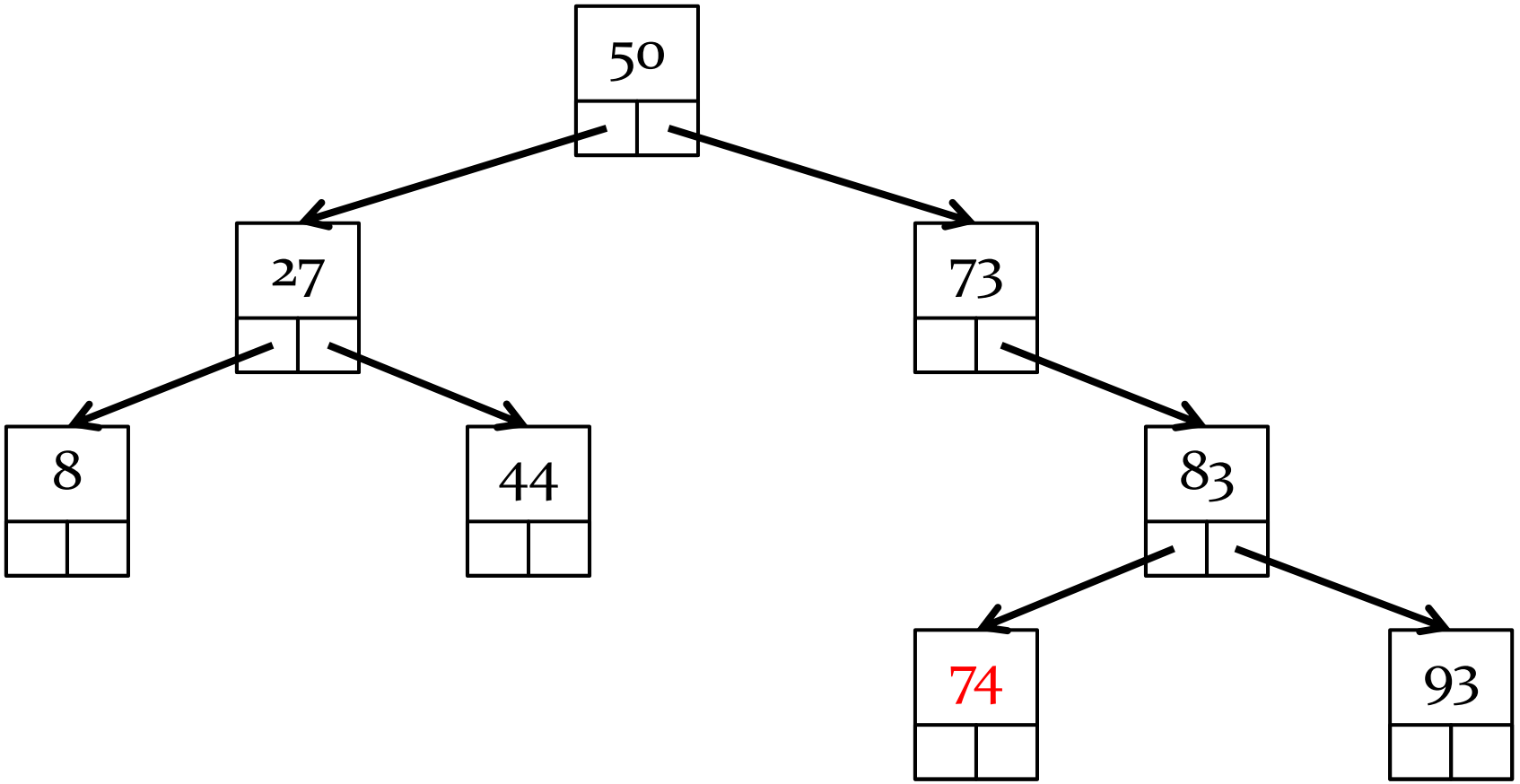

27 == 93?
50
27
73
8
44
83
74
93

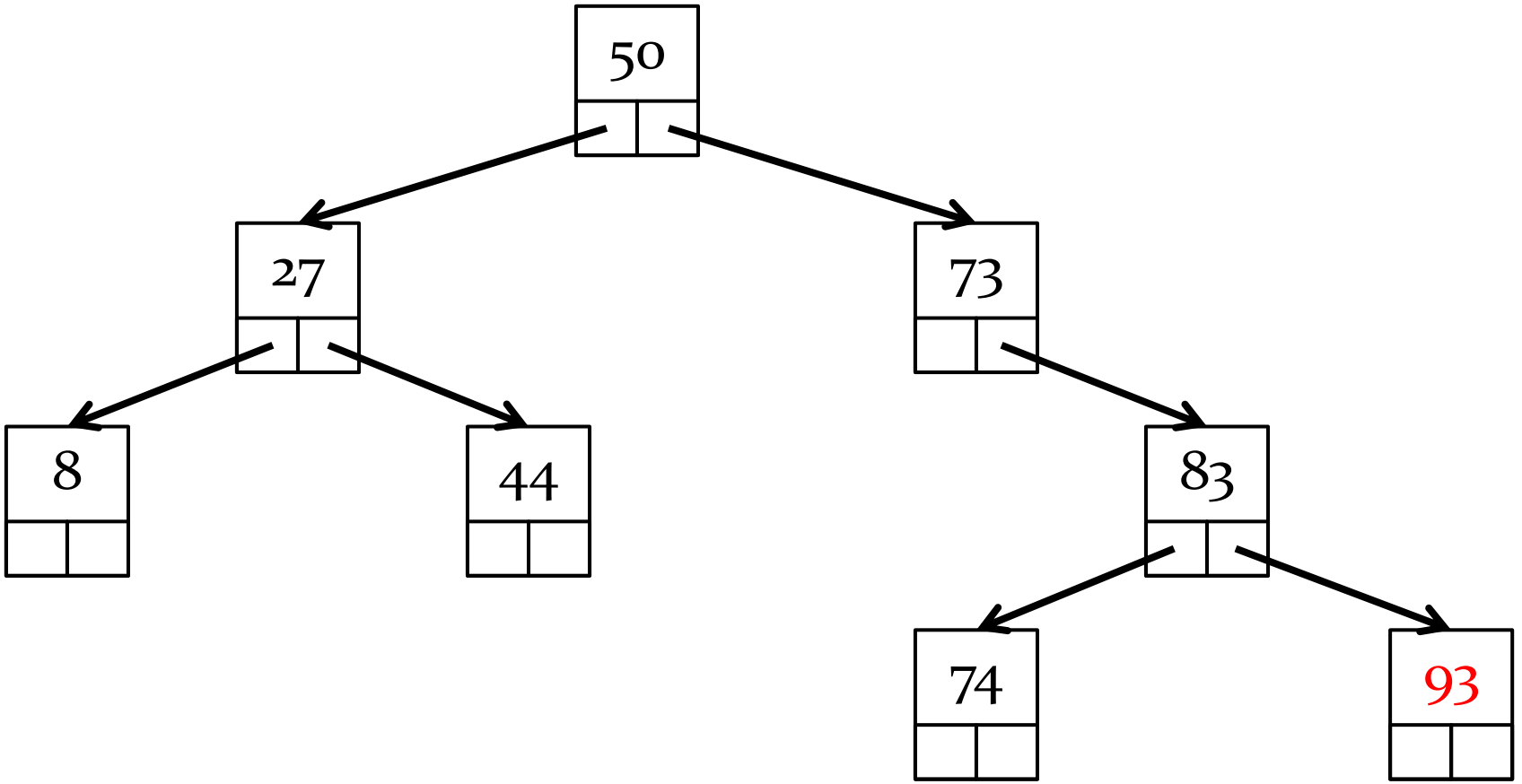

50
27
73
8
44
83
74
93
8 == 93?


50
27
73
8
44
83
44 == 93?
74
93


50
27
73
73 == 93?
8
44
83
74
93


50
27
73
8
44
83
83 == 93?
74
93

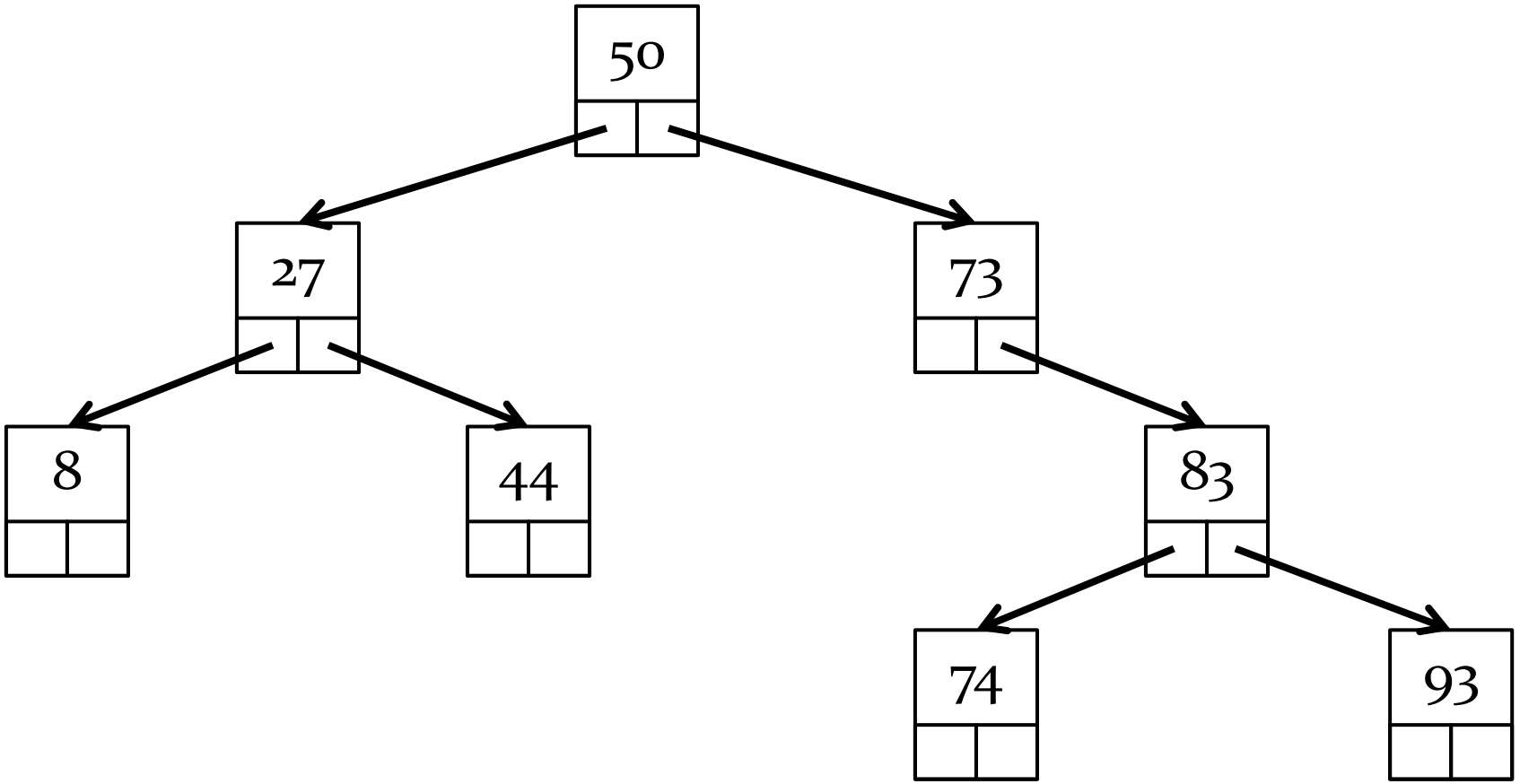

50
27
73
8
44
83
74
93
74 == 93?



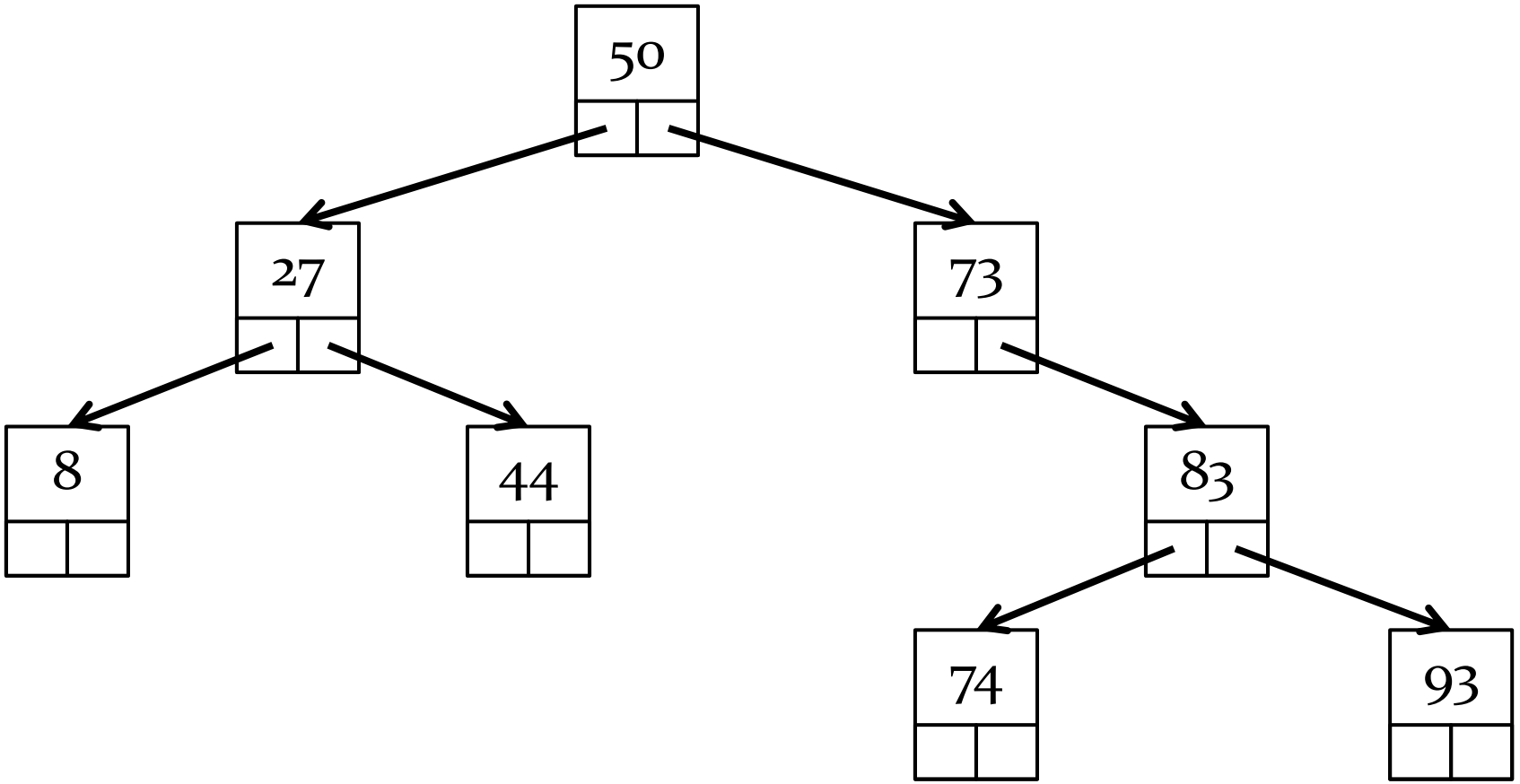
93 == 93?

Iteration

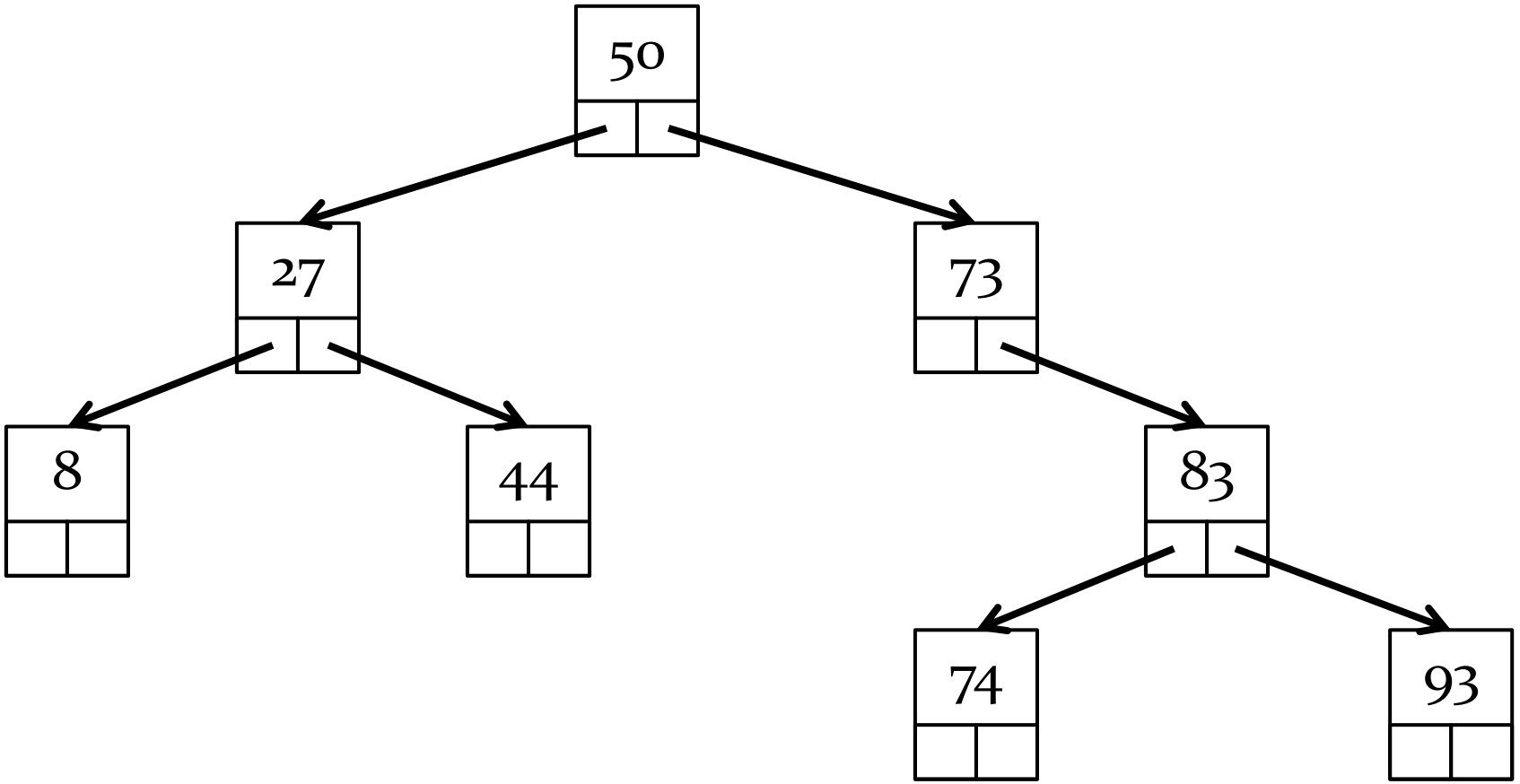
- visiting every element of the tree can also be done recursively
- 3 possibilities based on when the root is visited
  - inorder
    - visit left child, then root, then right child
  - preorder
    - visit root, then left child, then right child
  - postorder
    - visit left child, then right child, then root



inorder: 8, 27, 44, 50, 73, 74, 83, 93



preorder: 50, 27, 8, 44, 73, 83, 74, 93



postorder: 8, 44, 27, 74, 93, 83, 73, 50