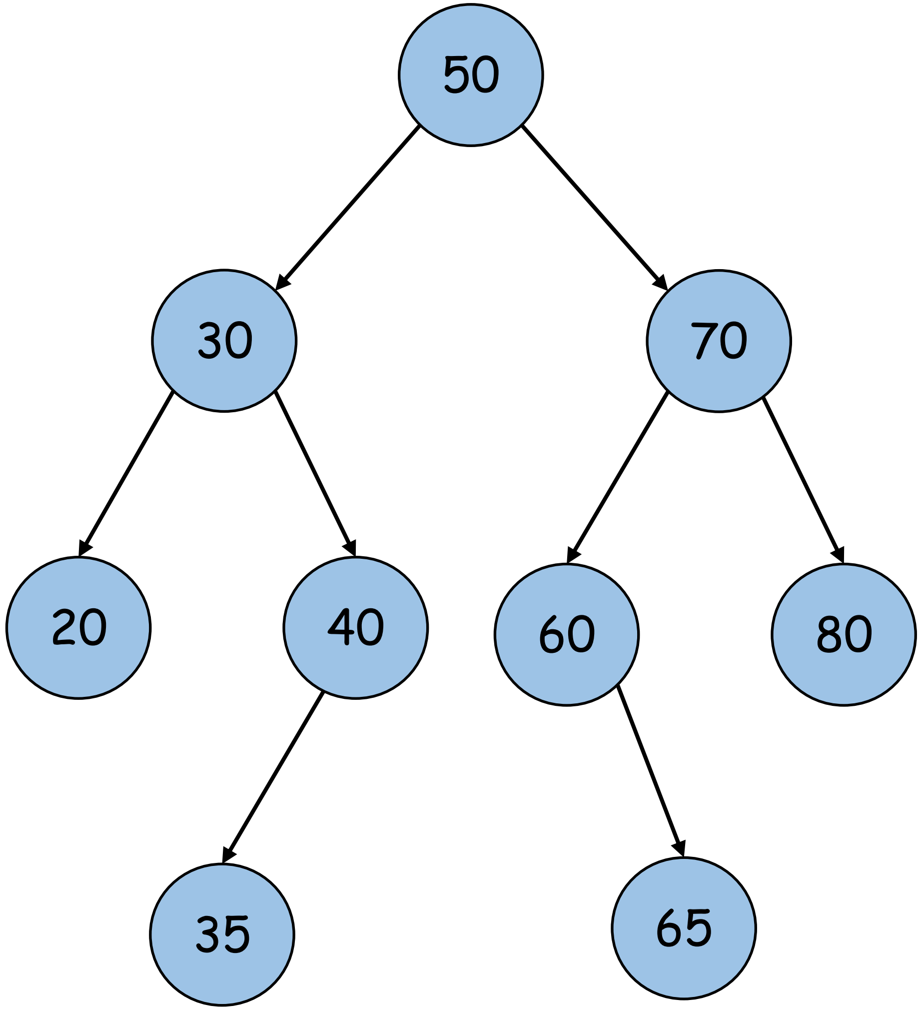
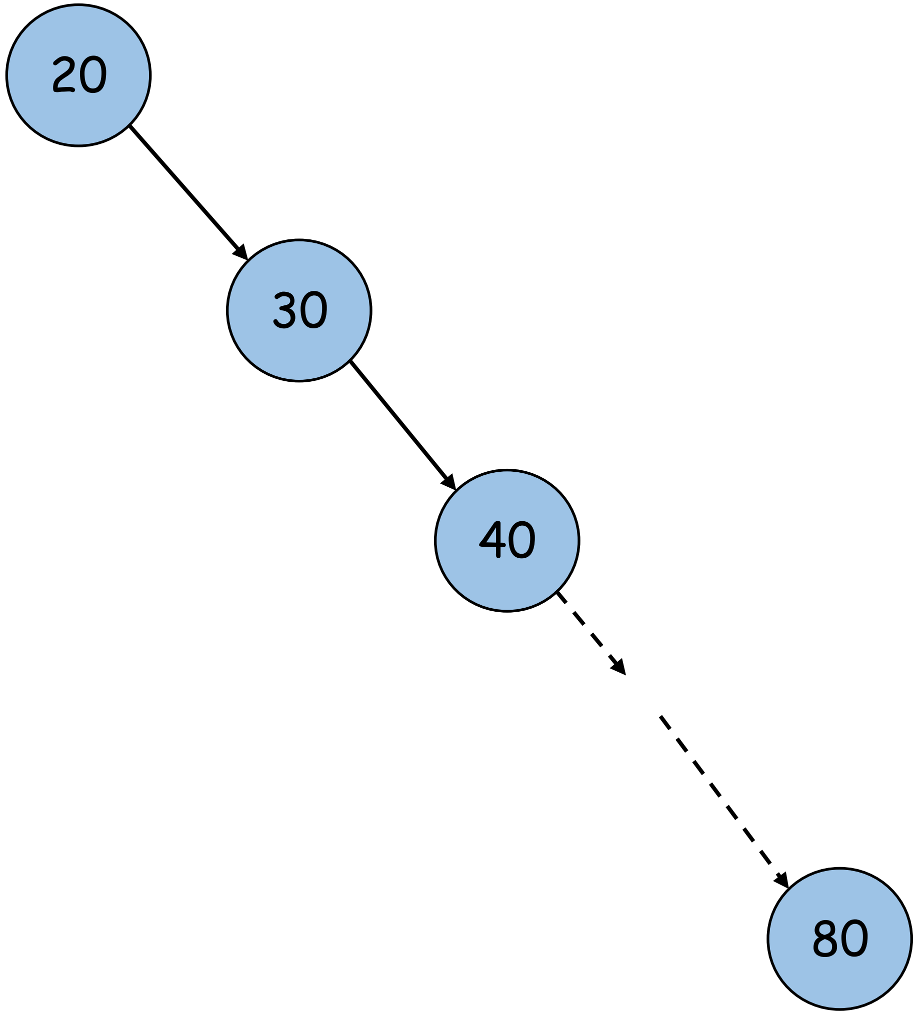
Balanced Trees

Sridhar Alagar

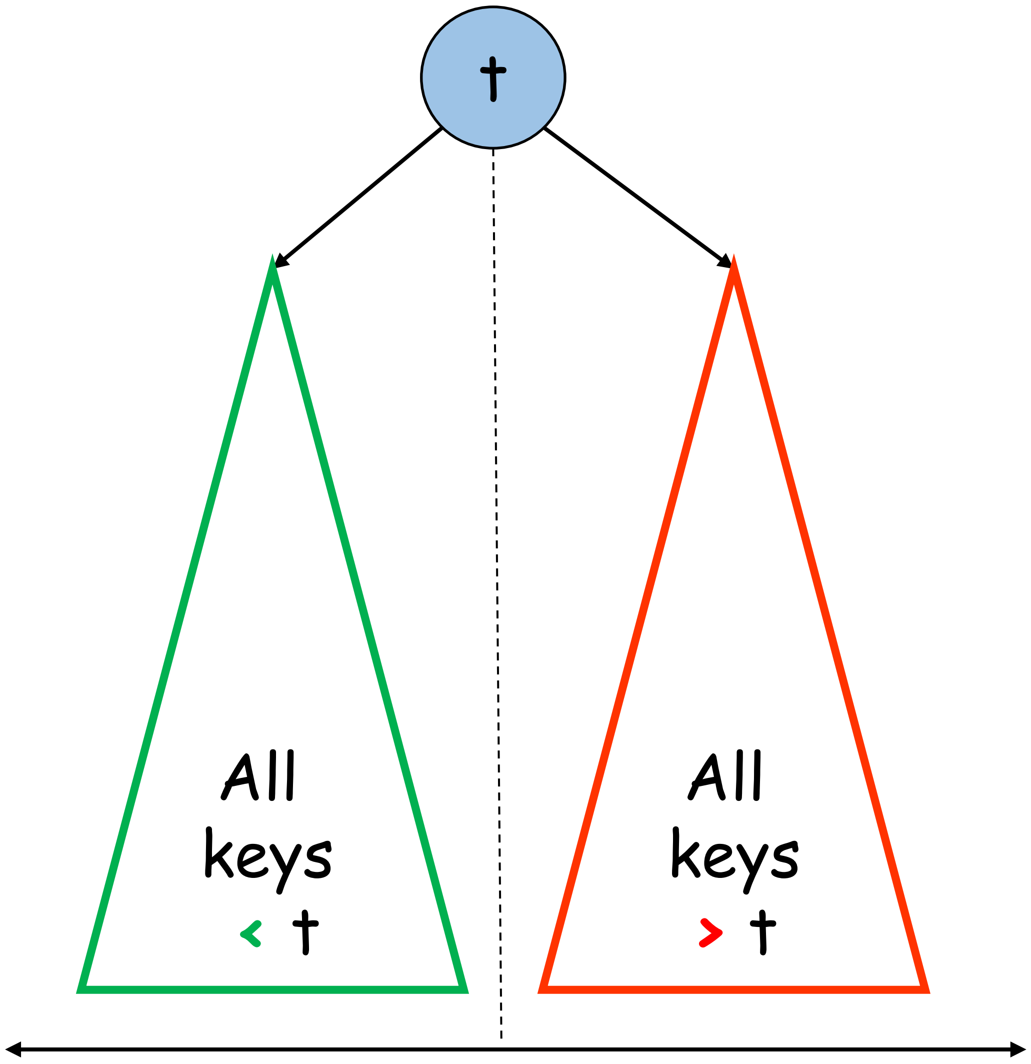
Motivation

Height of the BST can be as high n-1





Binary Search Tree

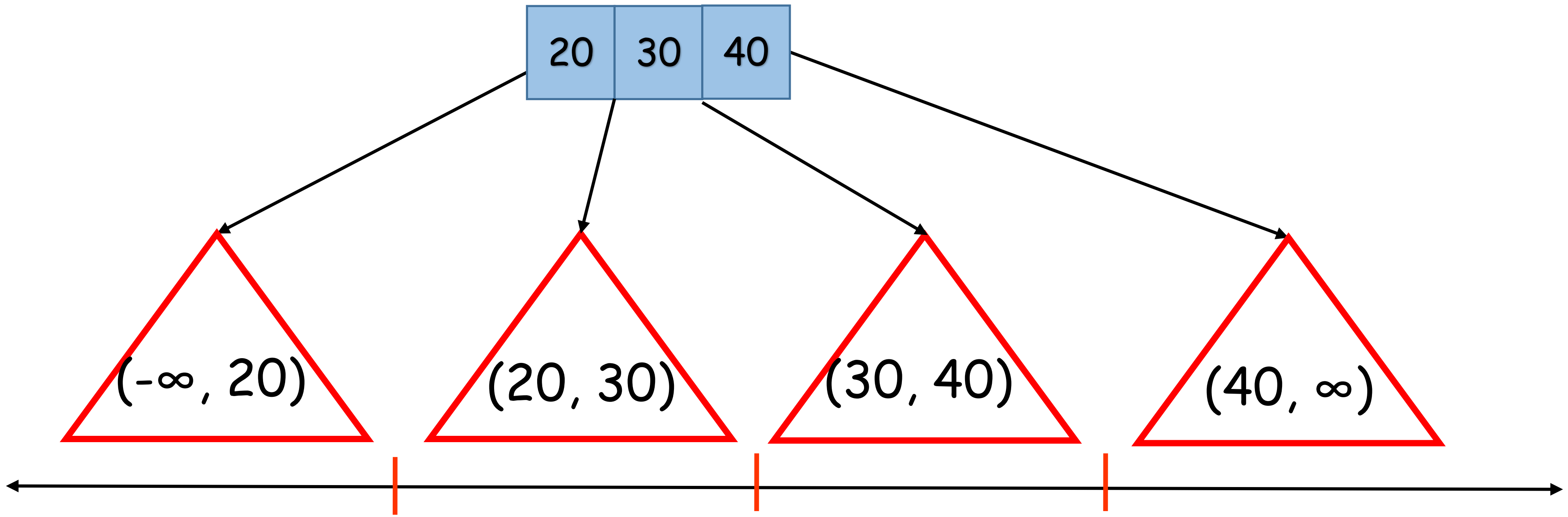


- In a BST, each node stores one key
- Each node splits the key space into two halves

Generalizing BST

- Each node stores many keys in sorted order

- Each node with k keys splits the key space into k+1 regions



Balanced Multiway Search Tree

- Building a <u>balanced</u> multiway search tree is easy
- Just pack them in one node

20	30	35	40	50	60	65	70

- At some point it becomes a bad idea. Why?
- It is just a sorted array
- Need to limit the number of keys in a node. How?

Strategy 1

- Push keys down after the limit exceeded in a node

20 30 70 80 35 40 50 38 39

Strategy 1

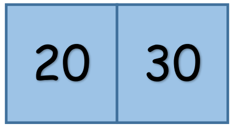
- Push keys down after the limit exceeded in a node

20

30 70 80 35 40 50 38 39

Strategy 1

- Push keys down after the limit exceeded in a node



70 80 35 40 50 38 39

Strategy 1

- Push keys down after the limit exceeded in a node

20	30	70

80 35 40 50 38 39

Strategy 1

- Push keys down after the limit exceeded in a node

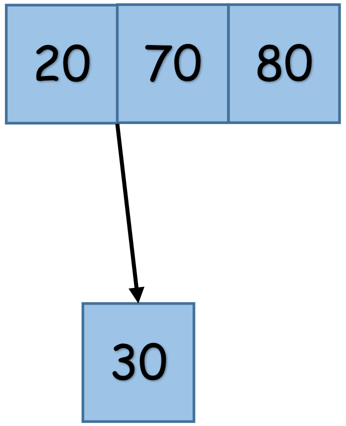
20	30	70	80

Limit exceeded…
push down

35 40 50 38 39

Strategy 1

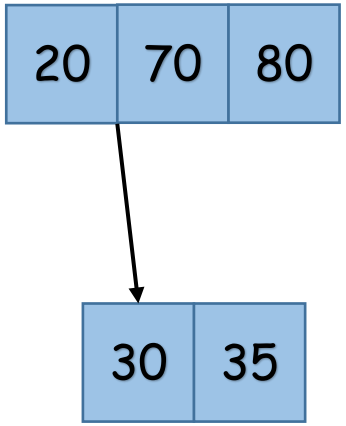
- Push keys down after the limit exceeded in a node



35 40 50 38 39

Strategy 1

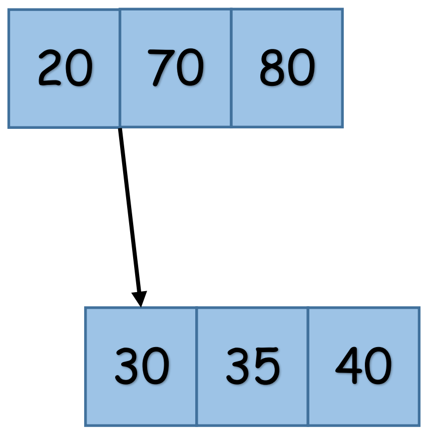
- Push keys down after the limit exceeded in a node



40 50 38 39

Strategy 1

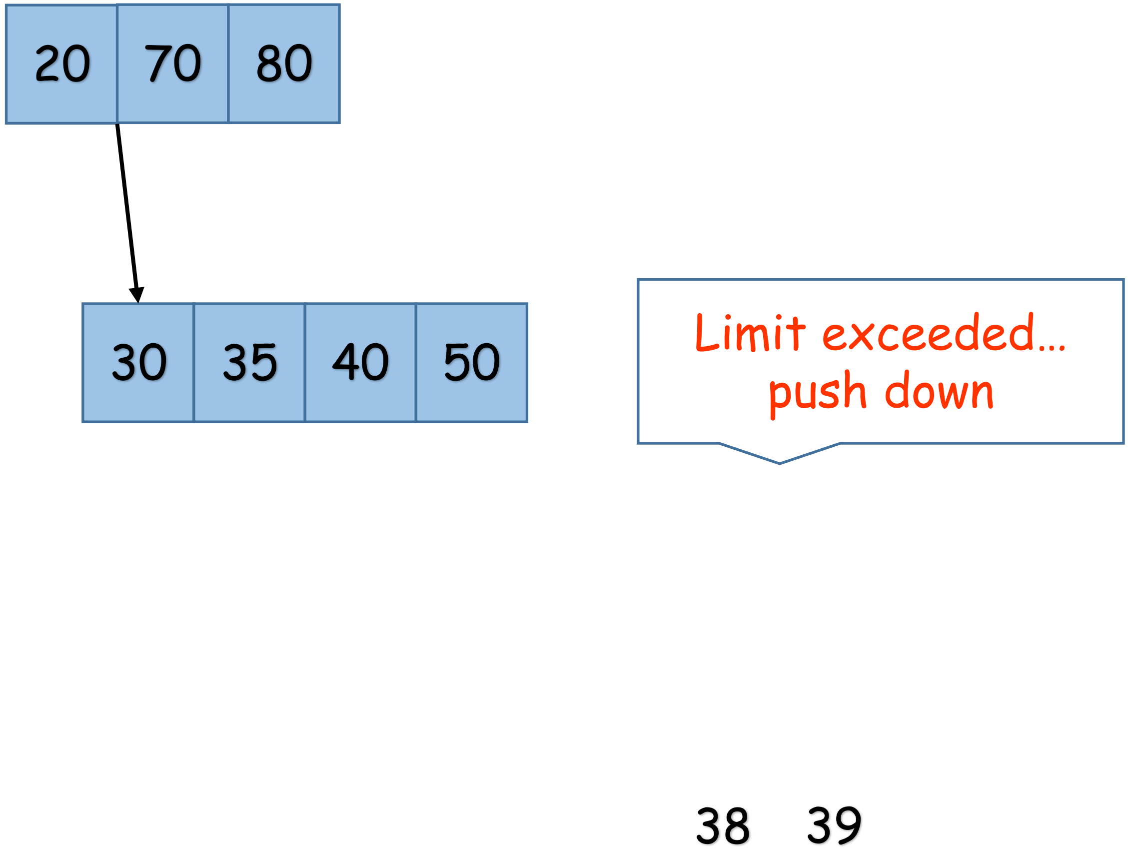
- Push keys down after the limit exceeded in a node



50 38 39

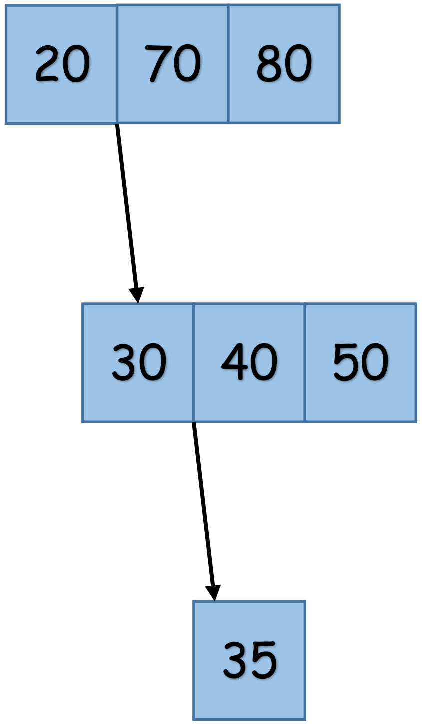
Strategy 1

- Push keys down after the limit exceeded in a node



Strategy 1

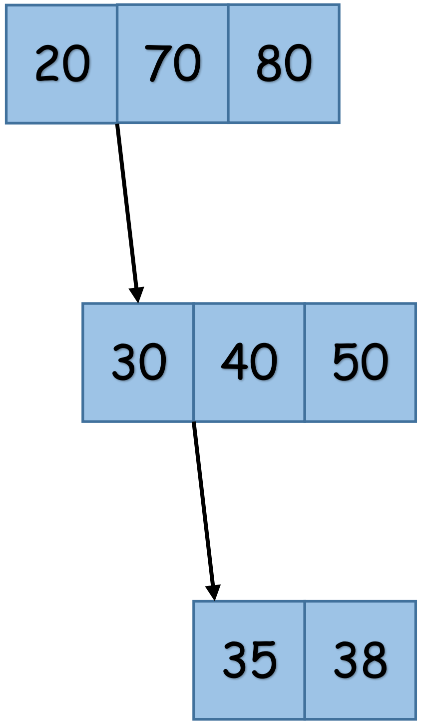
- Push keys down after the limit exceeded in a node



38 39

Strategy 1

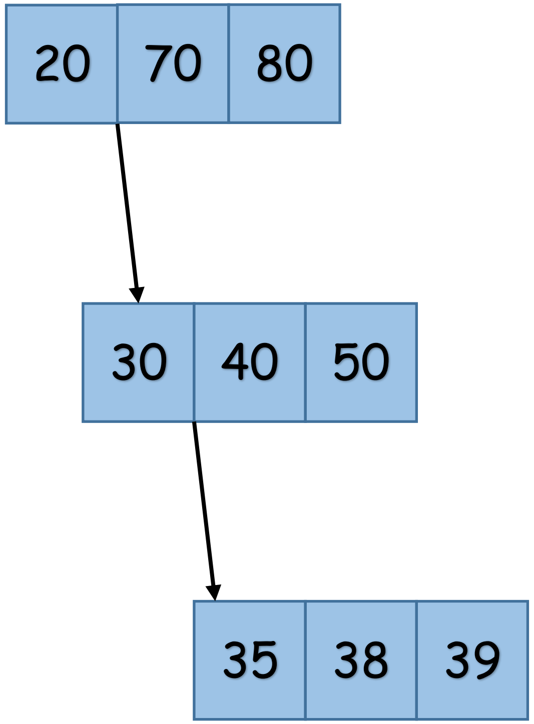
- Push keys down after the limit exceeded in a node



39

Strategy 1

- Push keys down after the limit exceeded in a node



- Easy to implement
- But tree can be unbalanced

Strategy 2

- Split node and push middle key up if limit exceeded

20 30 70 80 35 40 50 38 39

Strategy 2

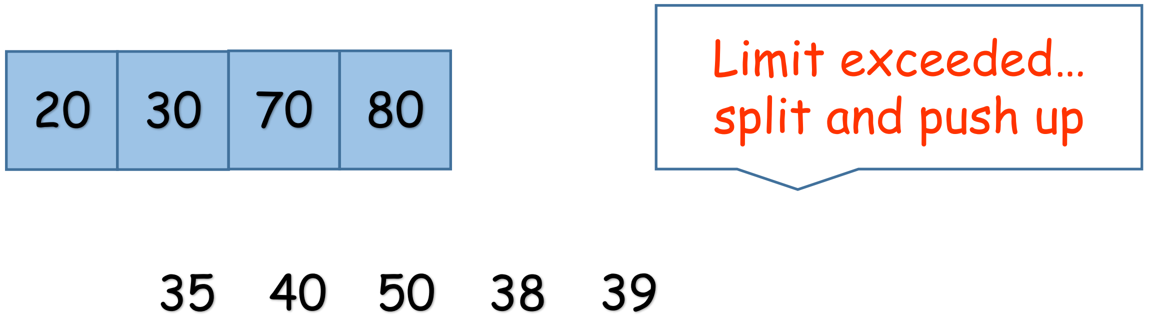
- Split node and push middle key up if limit exceeded

20	30	70

80 35 40 50 38 39

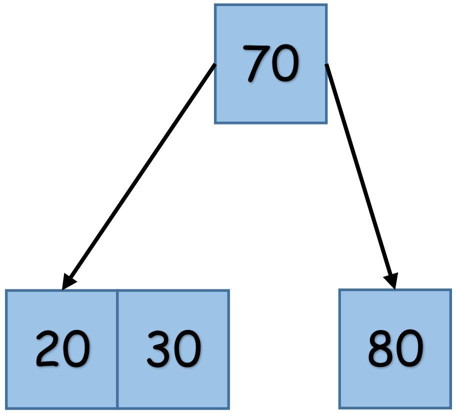
Strategy 2

- Split node and push middle key up if limit exceeded



Strategy 2

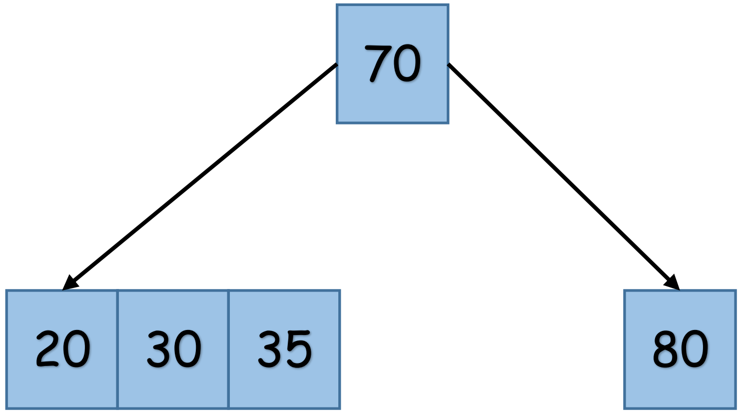
- Split node and push middle key up if limit exceeded



35 40 50 38 39

Strategy 2

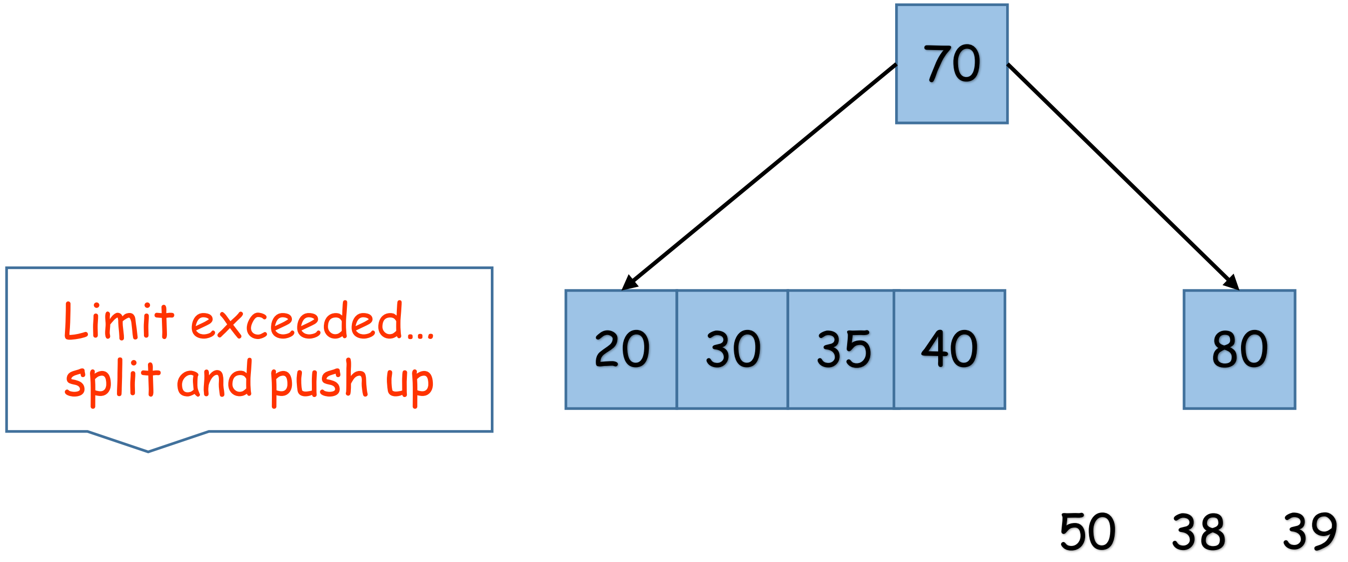
- Split node and push middle key up if limit exceeded



40 50 38 39

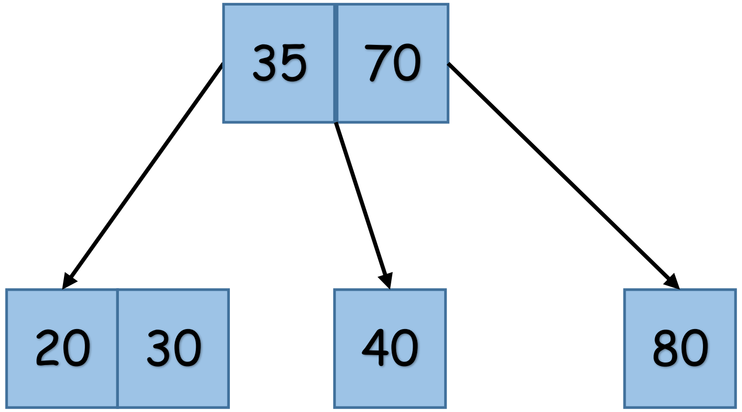
Strategy 2

- Split node and push middle key up if limit exceeded



Strategy 2

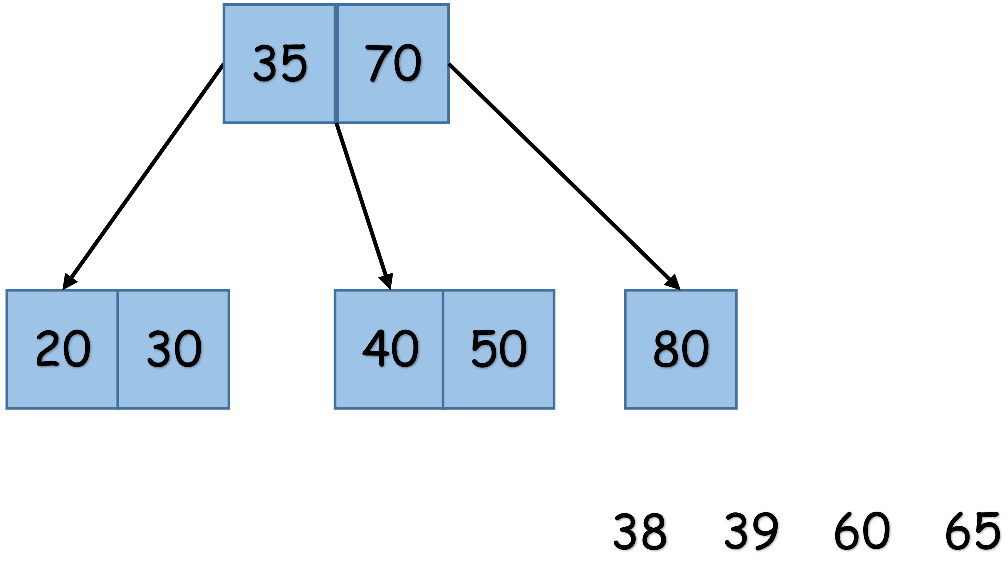
- Split node and push middle key up if limit exceeded



50 38 39 60 65

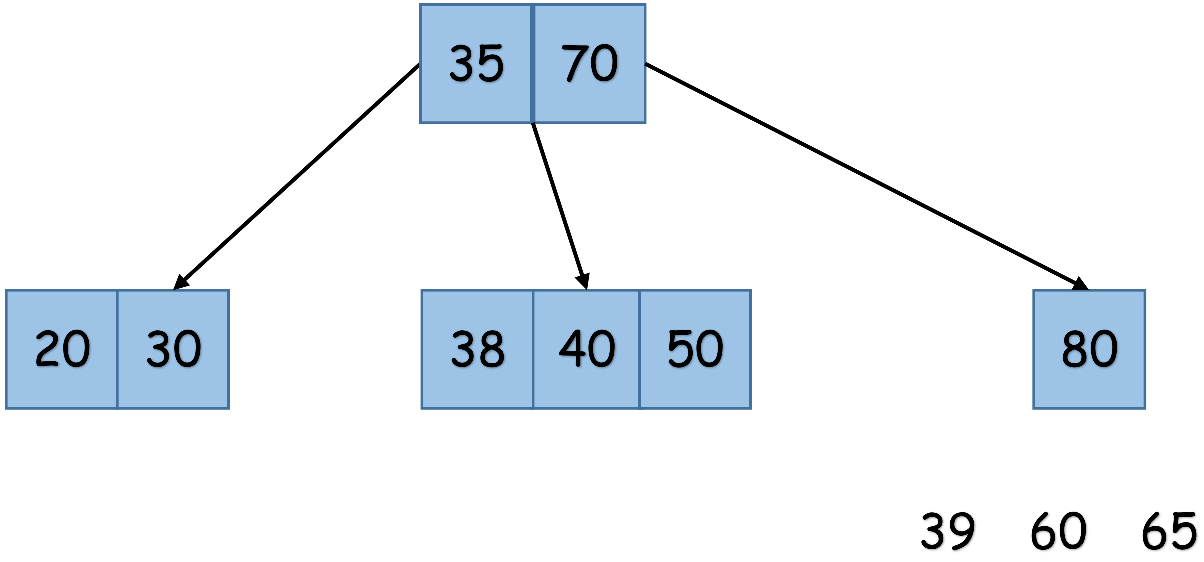
Strategy 2

- Split node and push middle key up if limit exceeded



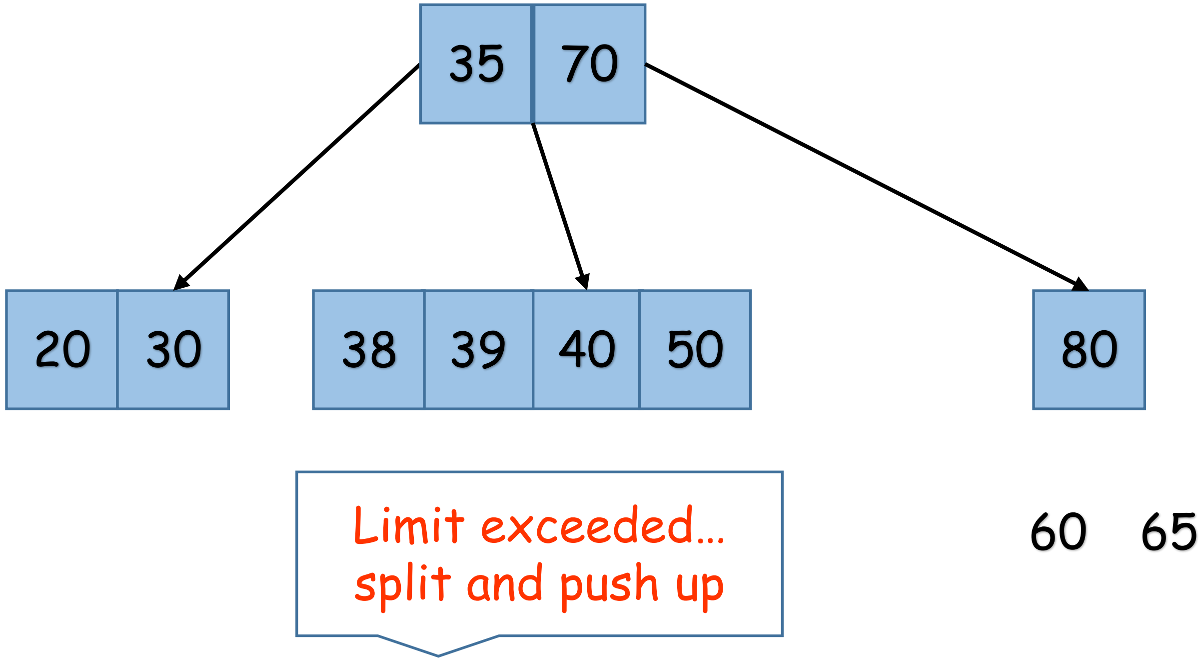
Strategy 2

- Split node and push middle key up if limit exceeded



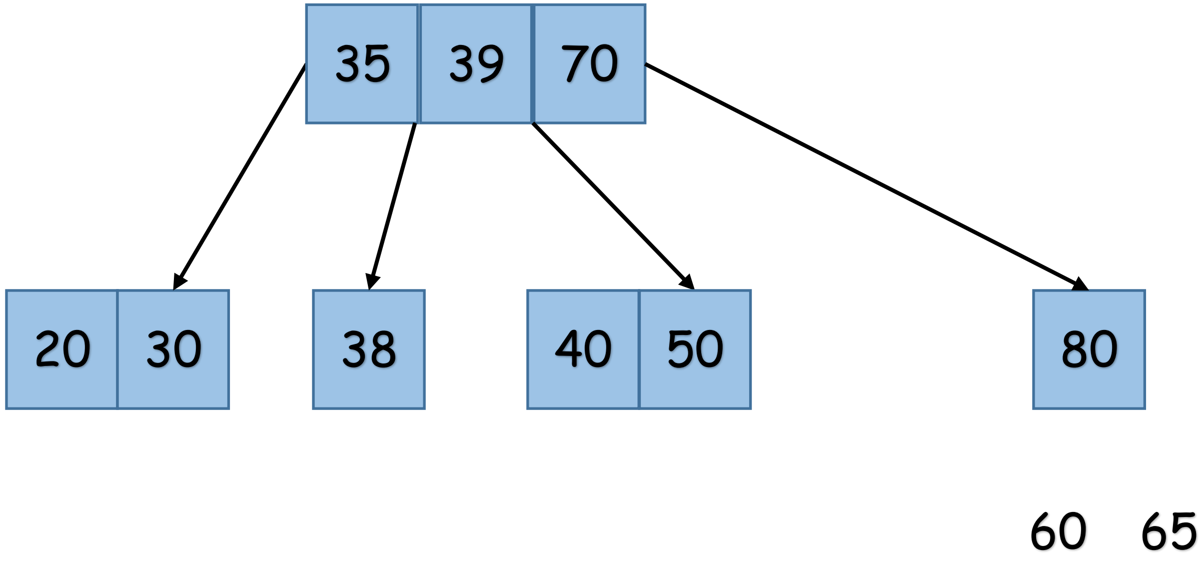
Strategy 2

- Split node and push middle key up if limit exceeded



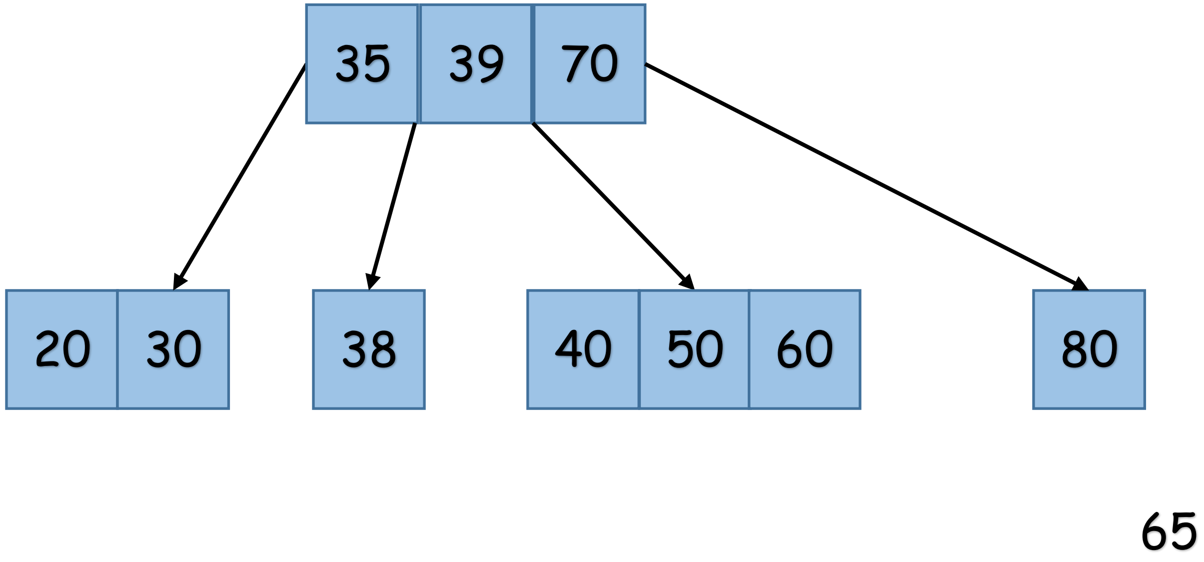
Strategy 2

- Split node and push middle key up if limit exceeded



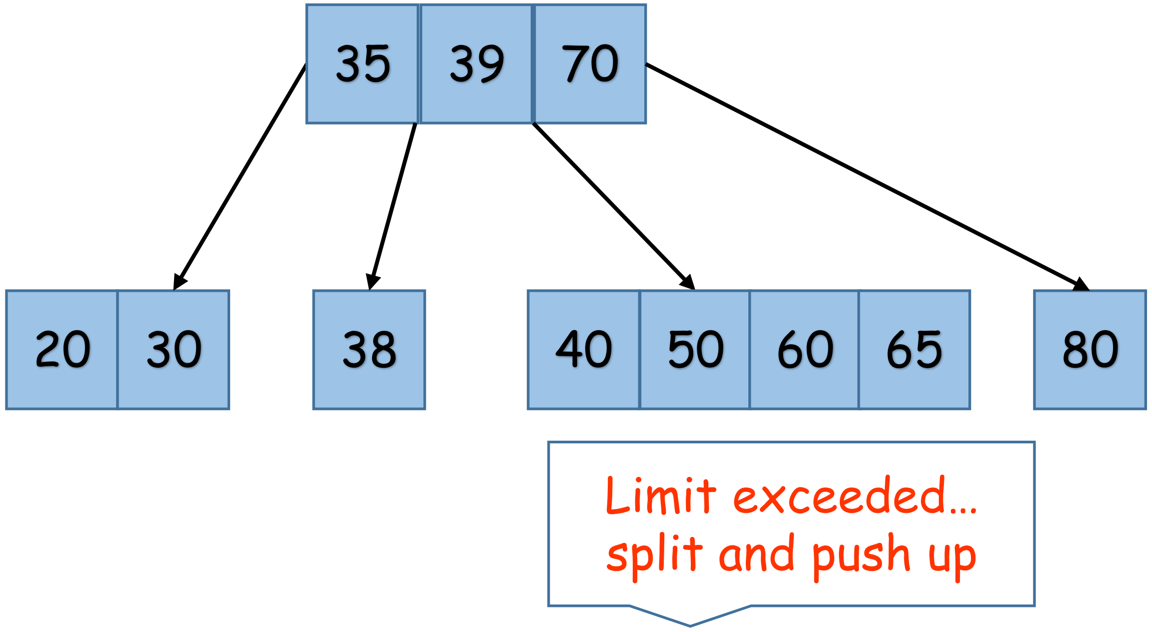
Strategy 2

- Split node and push middle key up if limit exceeded



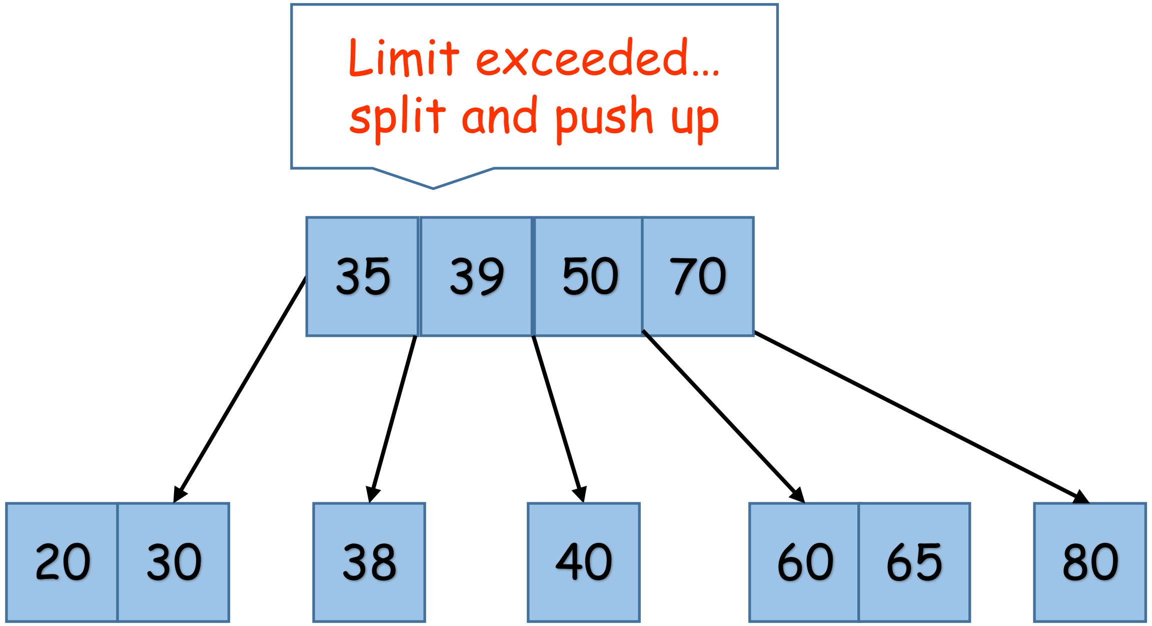
Strategy 2

- Split node and push middle key up if limit exceeded



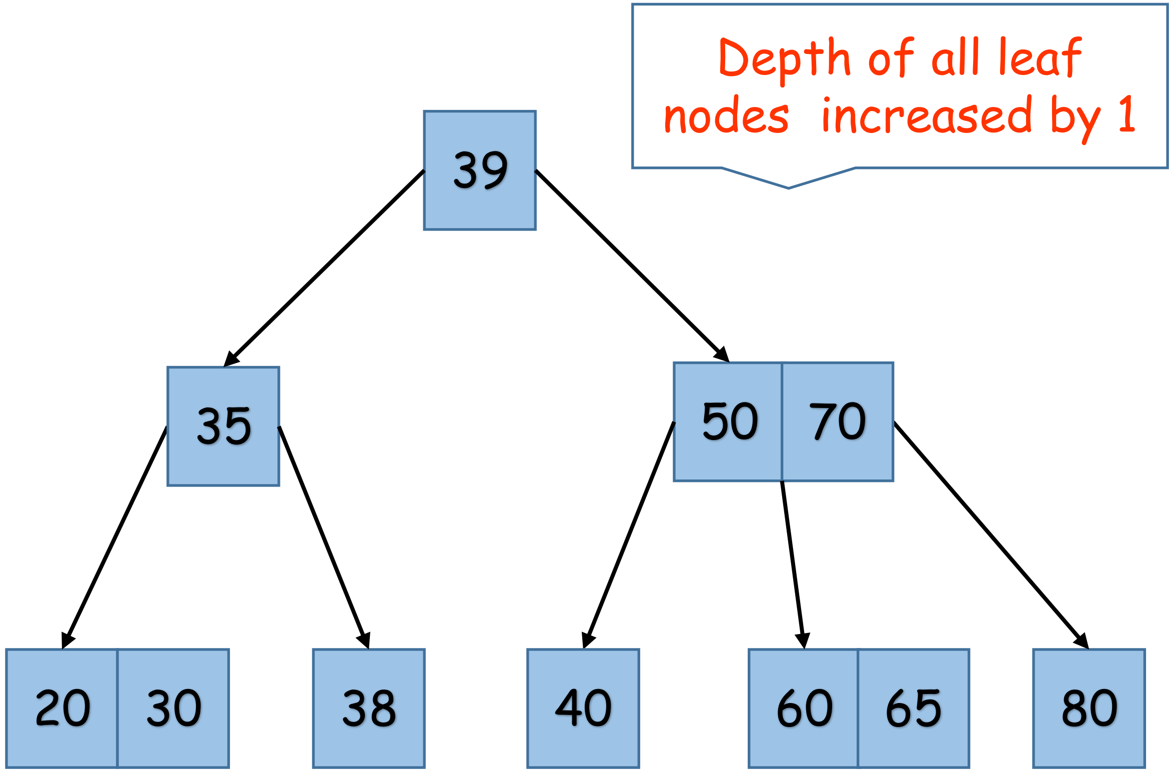
Strategy 2

- Split node and push middle key up if limit exceeded



Strategy 2

- Split node and push middle key up if limit exceeded

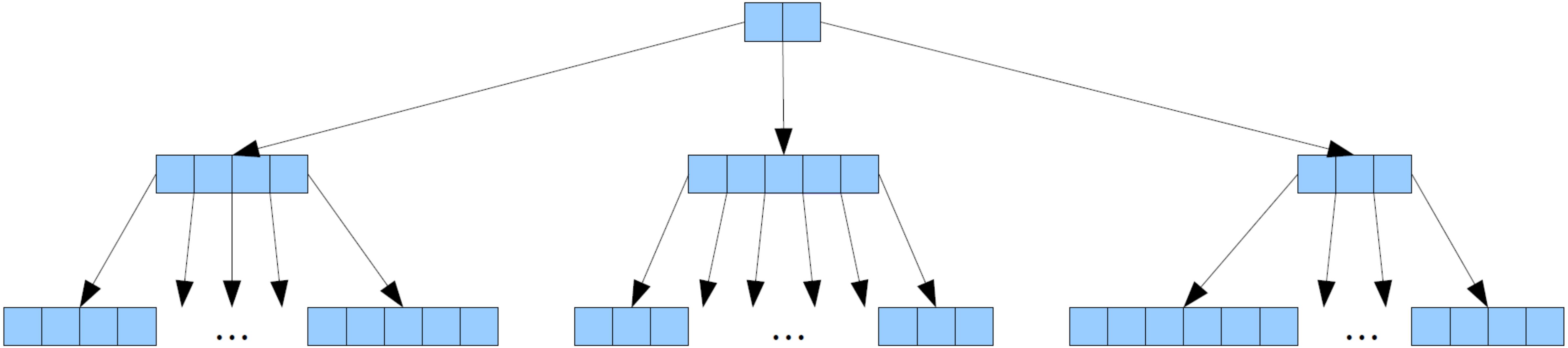


- Slightly trickier to implement
- But tree is balanced. So, search, insert are efficient

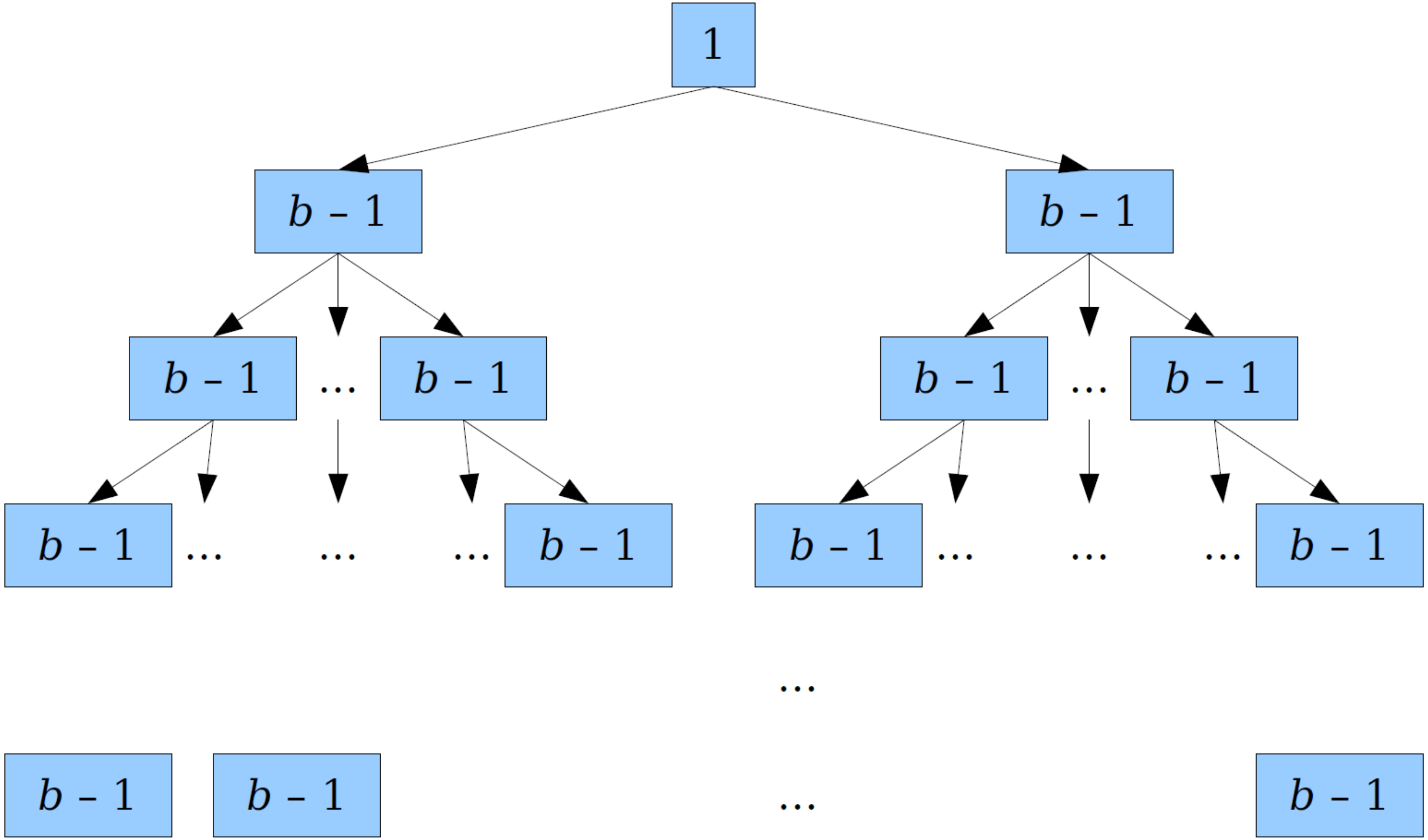
B-trees

A B-tree of order b is a multiway search tree where:

- each node has b-1 to 2b-1 keys, other than root which can have up to b-1 keys
- each node is either a leaf or has one more child than the number of keys
- all leaves are at the same depth



Maximum height of a B-tree? $O(\log_b n)$



1

2(b-1)

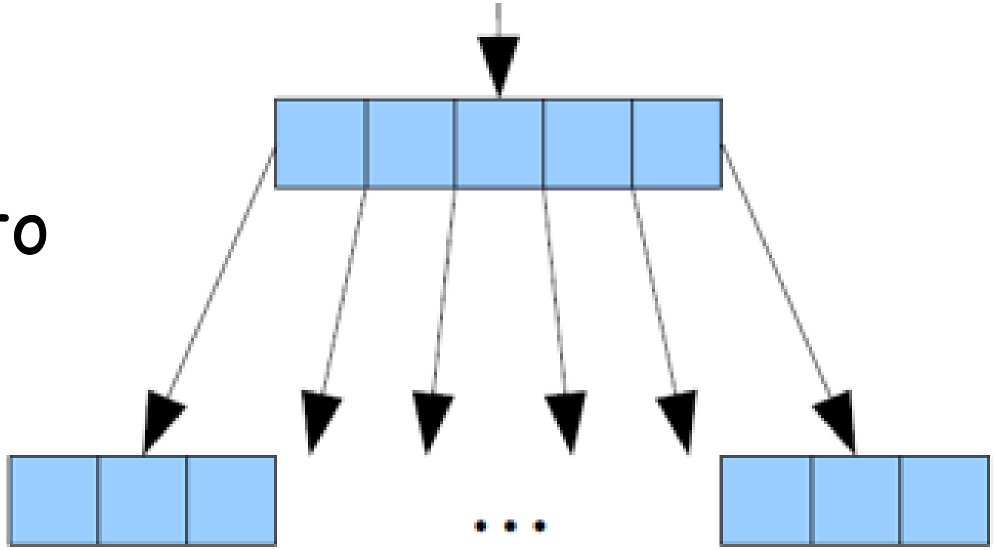
2b(b-1)

$2b^2(b-1)$

$2b^{h-1}(b-1)$

RT for Search

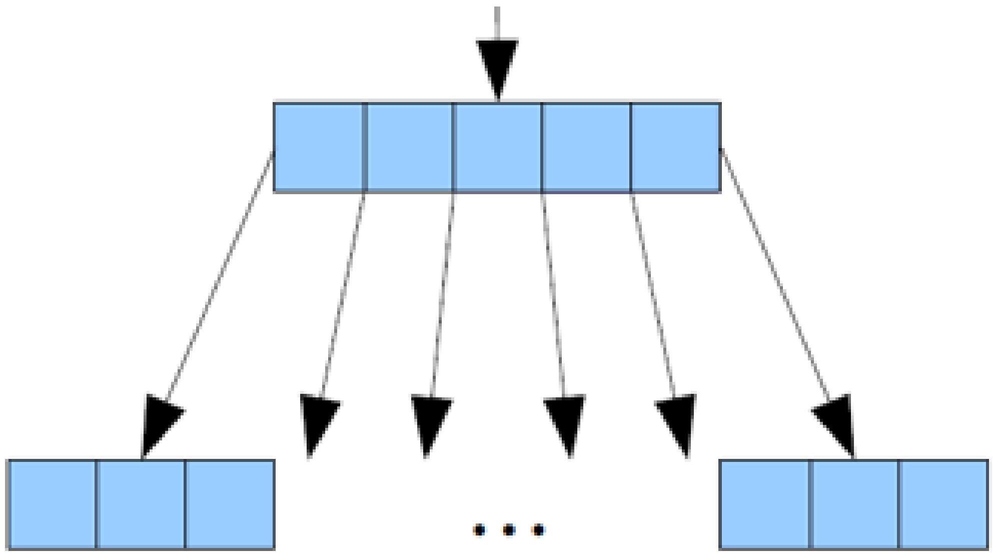
- Search every node
- Find the key or the child to descend to
- RT to search a node:
  Binary search - log b
- RT for search = log b $\log_b n$ = $O(\log n)$



RT for insertion

- Need to visit $\log_b n$ nodes
- May need to split every node $O(b)$
- RT for insertion: $= O(b \log_b n)$

  $= O((b/\log b) \log n)$
- In practice,
  - b = 2 (2-3-4 trees)
  - b = 1024/4096 (B-trees)



External data structures

- If n is too large to fit into memory, nodes are stored in secondary storage
- compute time <<< access time

- Ideal size (branching factor) of node is the size of block (minimum amount of data read from the storage device)
  - Typically, block size is 1K or 4K bytes
- Minimizes the storage device access

Insertion: $O(\log_b n)$

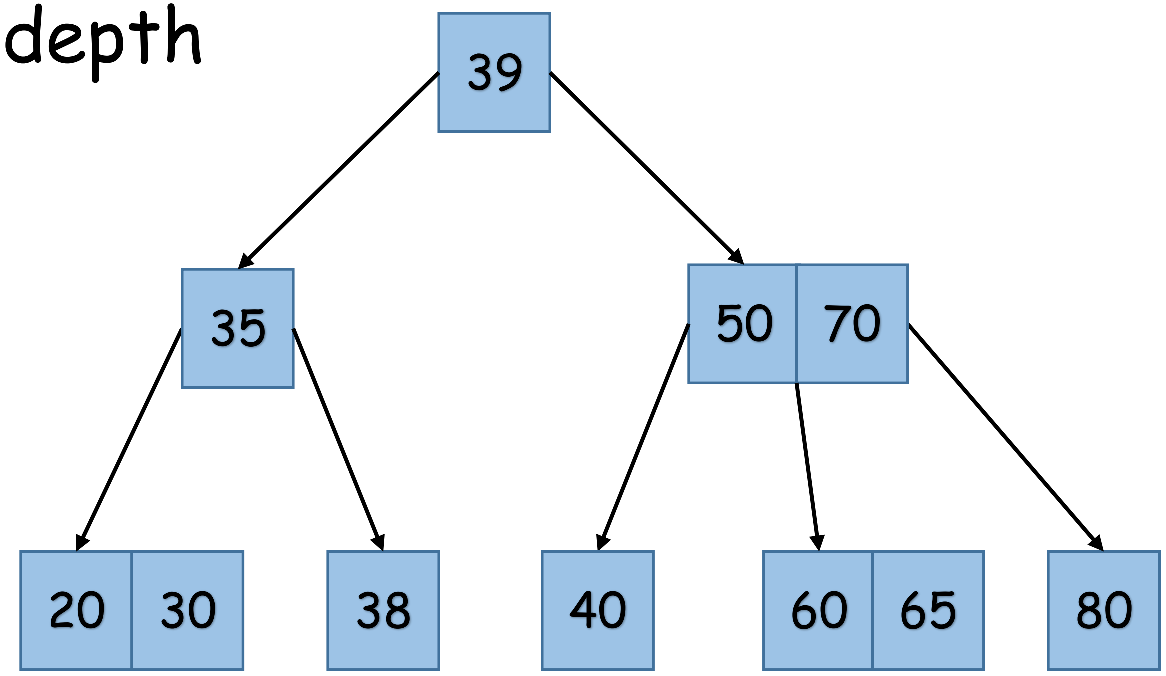
Search: $O(\log_b n)$

If there are billion keys, # disk access = 3

2-3-4 trees

A B-tree with order b = 2

- each node has 1 to 3 keys; root will have 1 key only
- each node is either a leaf or has one more child than the number of keys
- all leaves are at the same depth

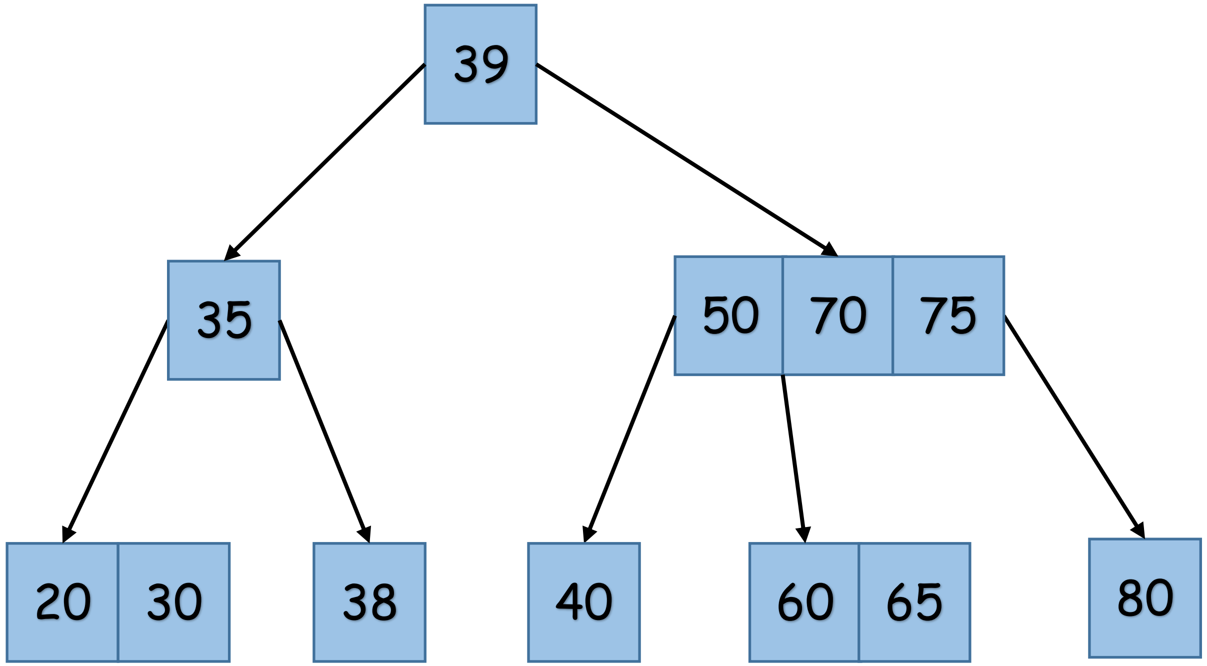


The story so far

- We know how to build a balanced multiway search tree
- 2-3-4 trees when main memory is sufficient
- B-trees for huge amount of keys

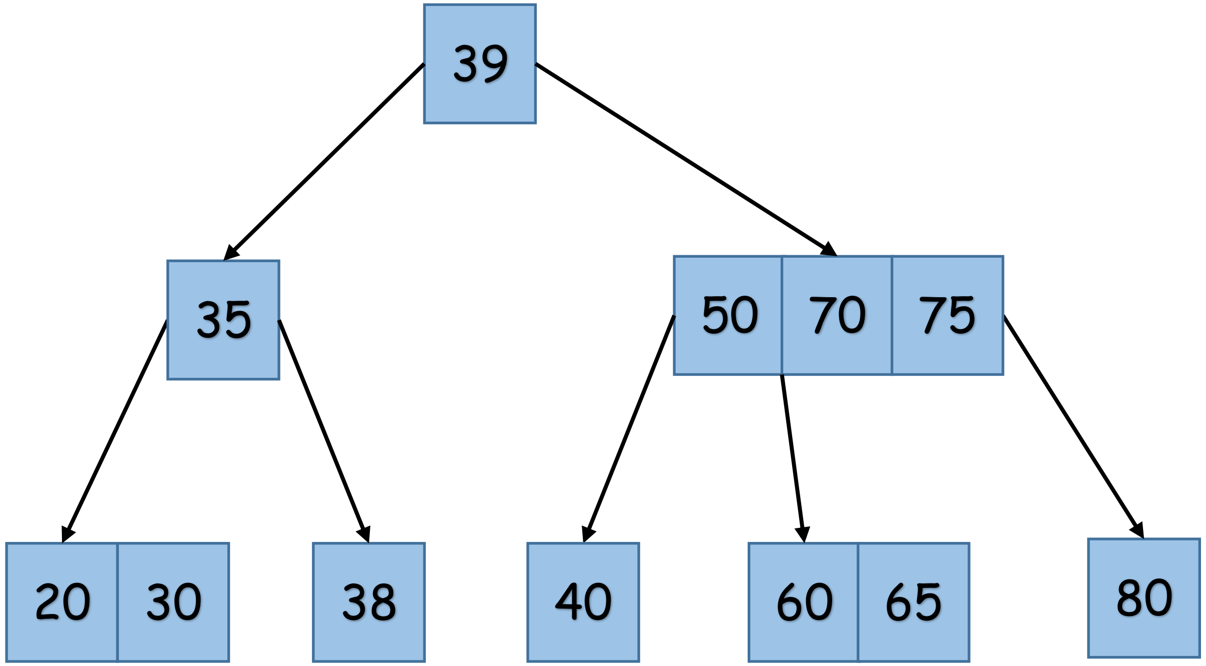
Red Black Trees

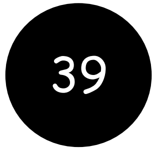
A 2-3-4 tree can be transformed into a corresponding red-black tree



Red Black Trees

A 2-3-4 tree can be transformed into a corresponding red-black tree

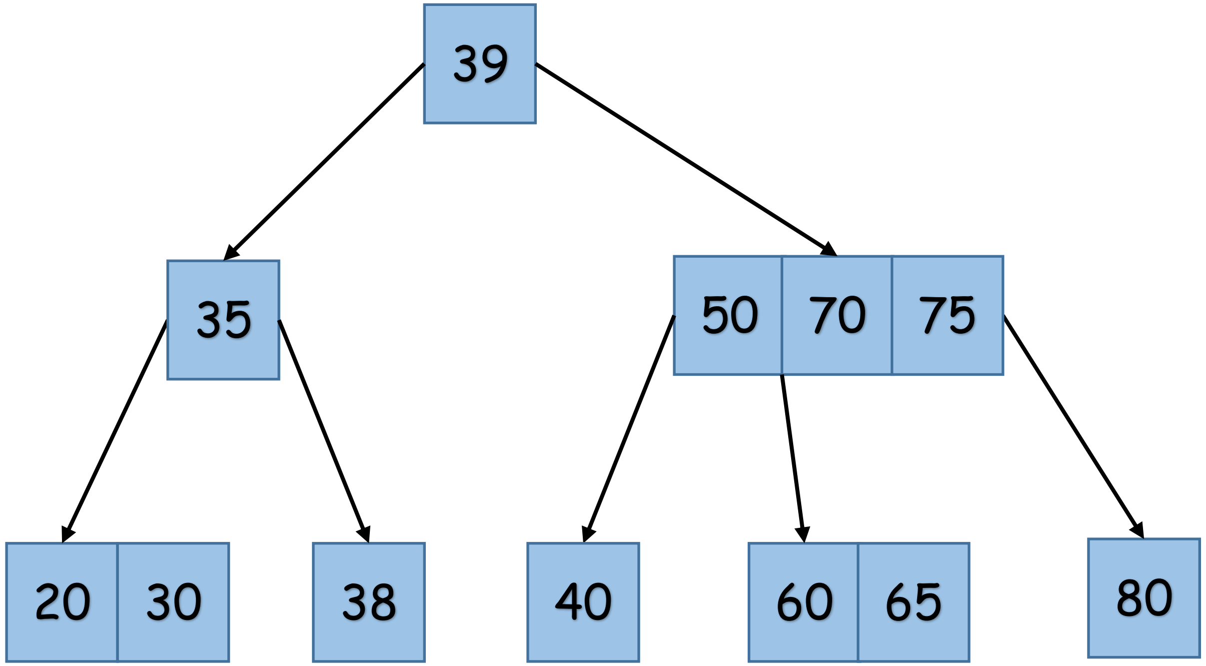


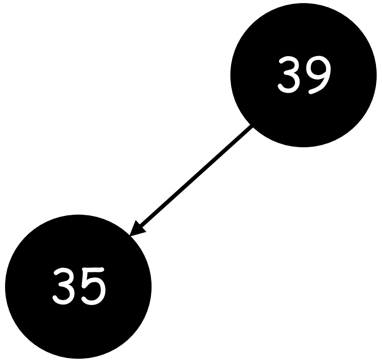


Root Property: The root is black

Red Black Trees

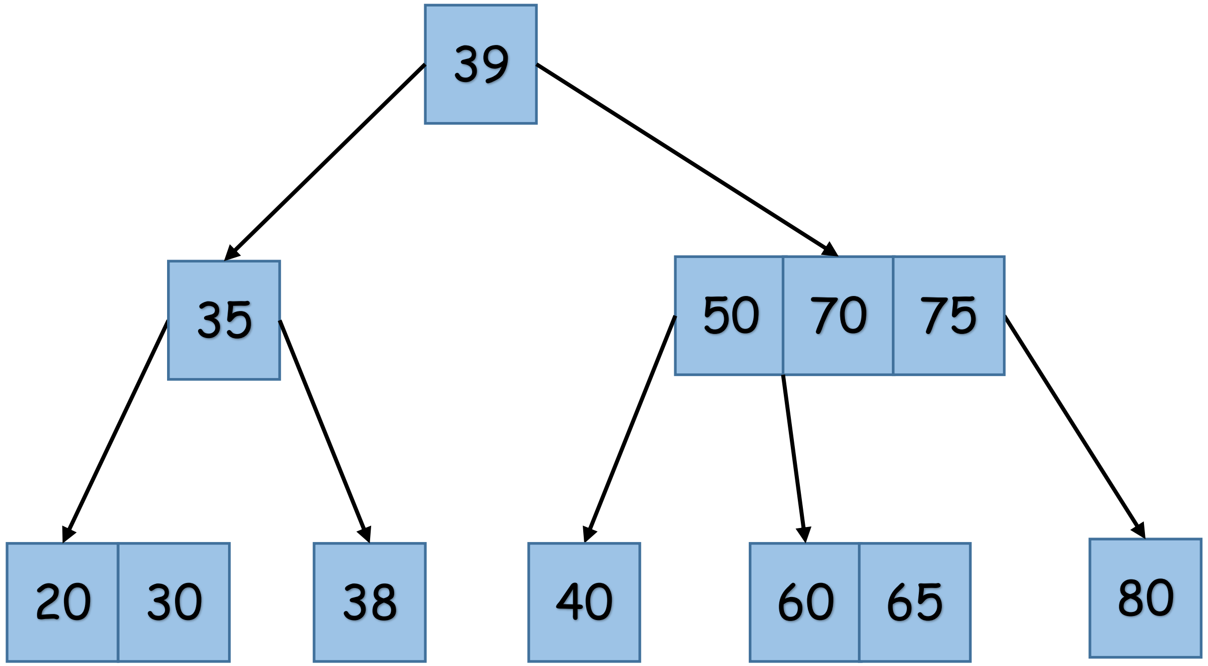
A 2-3-4 tree can be transformed into a corresponding red-black tree

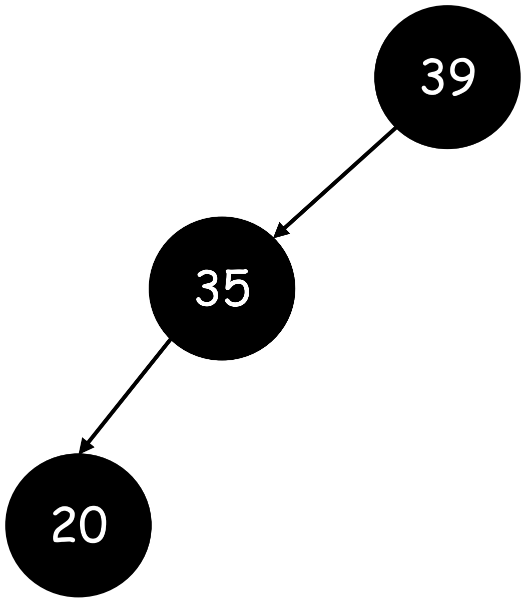




Red Black Trees

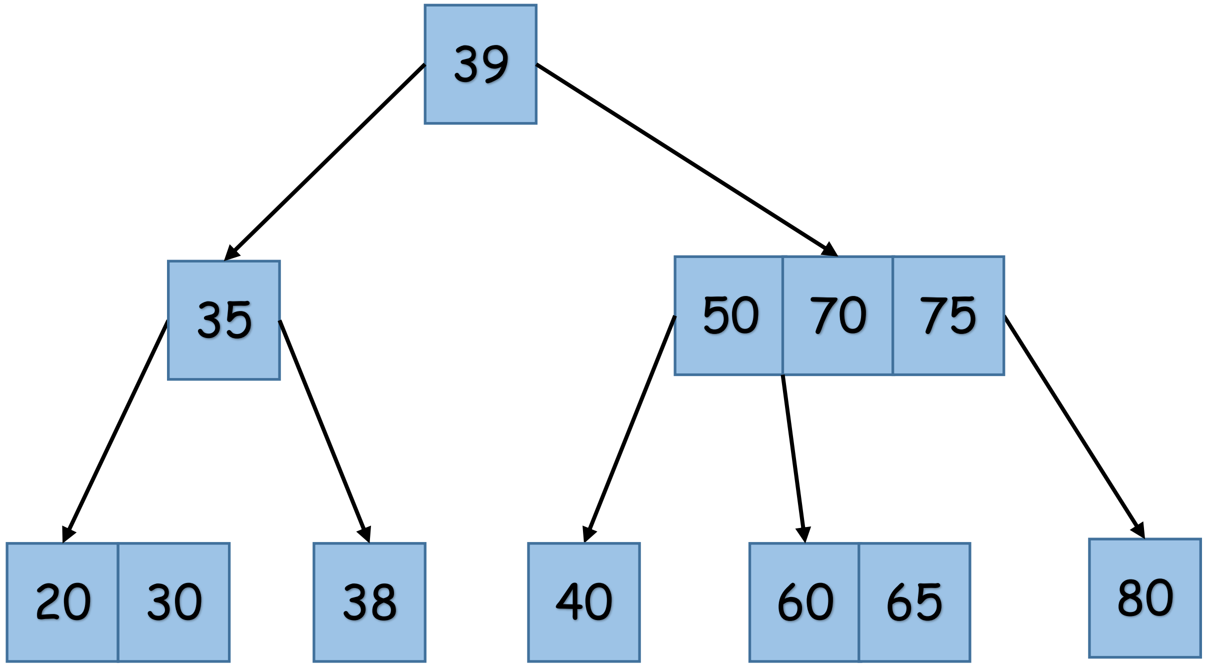
A 2-3-4 tree can be transformed into a corresponding red-black tree

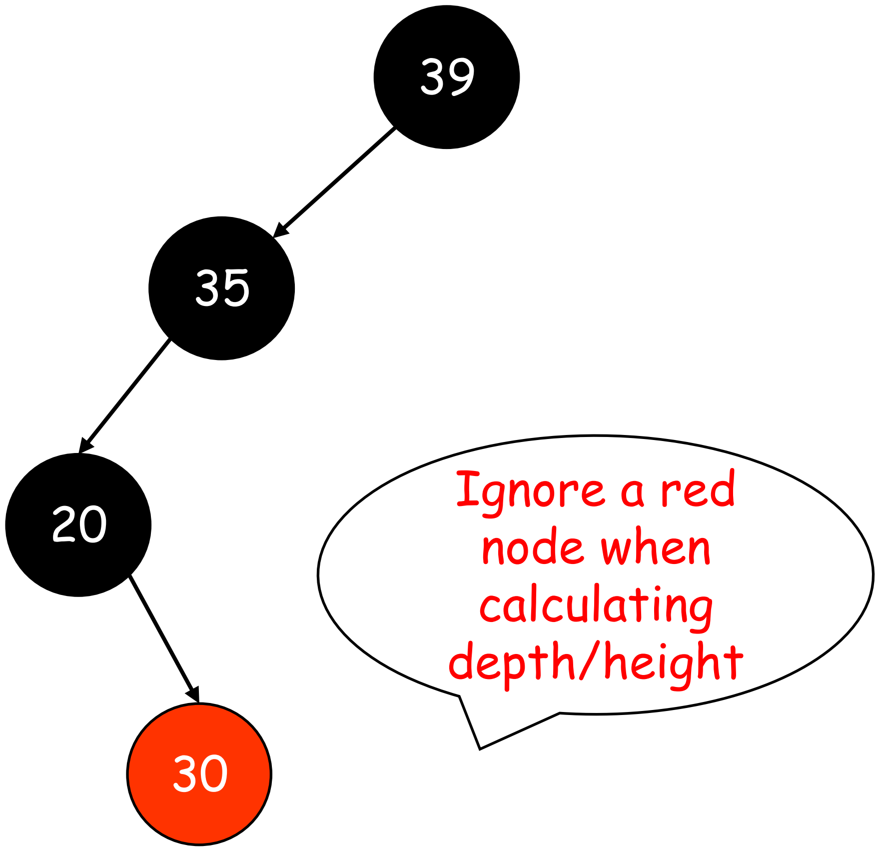




Red Black Trees

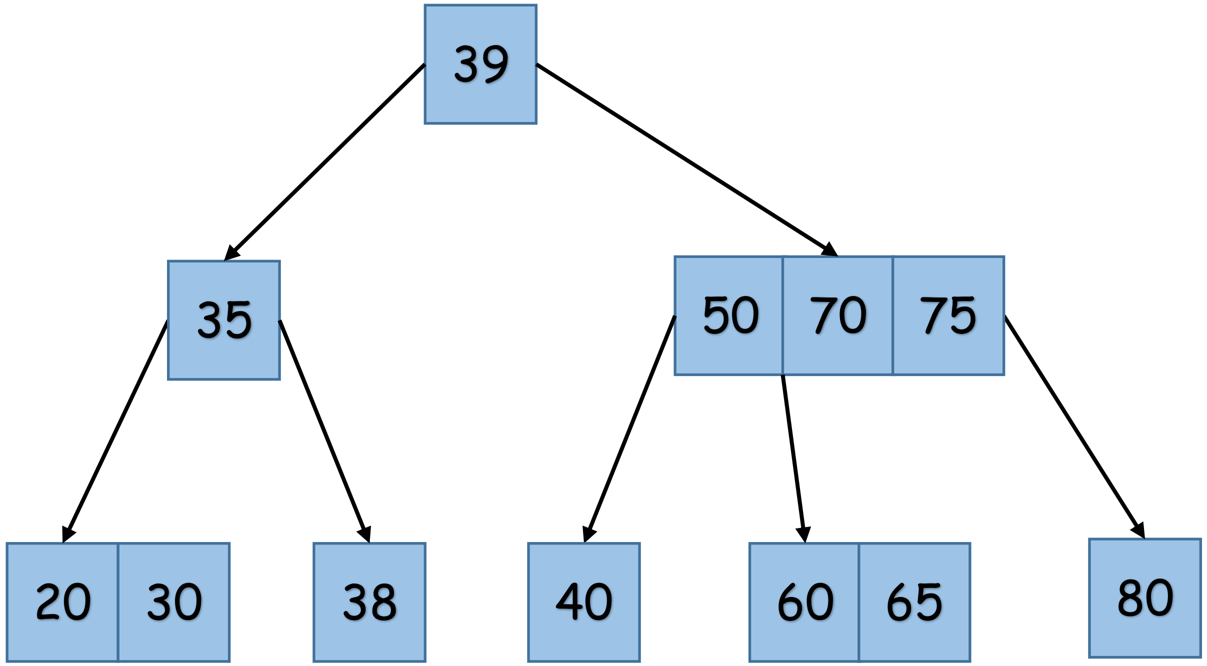
A 2-3-4 tree can be transformed into a corresponding red-black tree

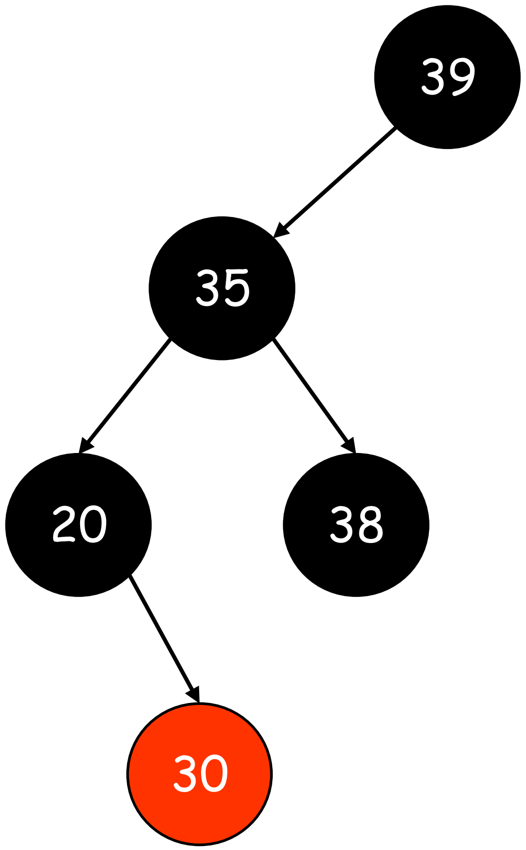




Red Black Trees

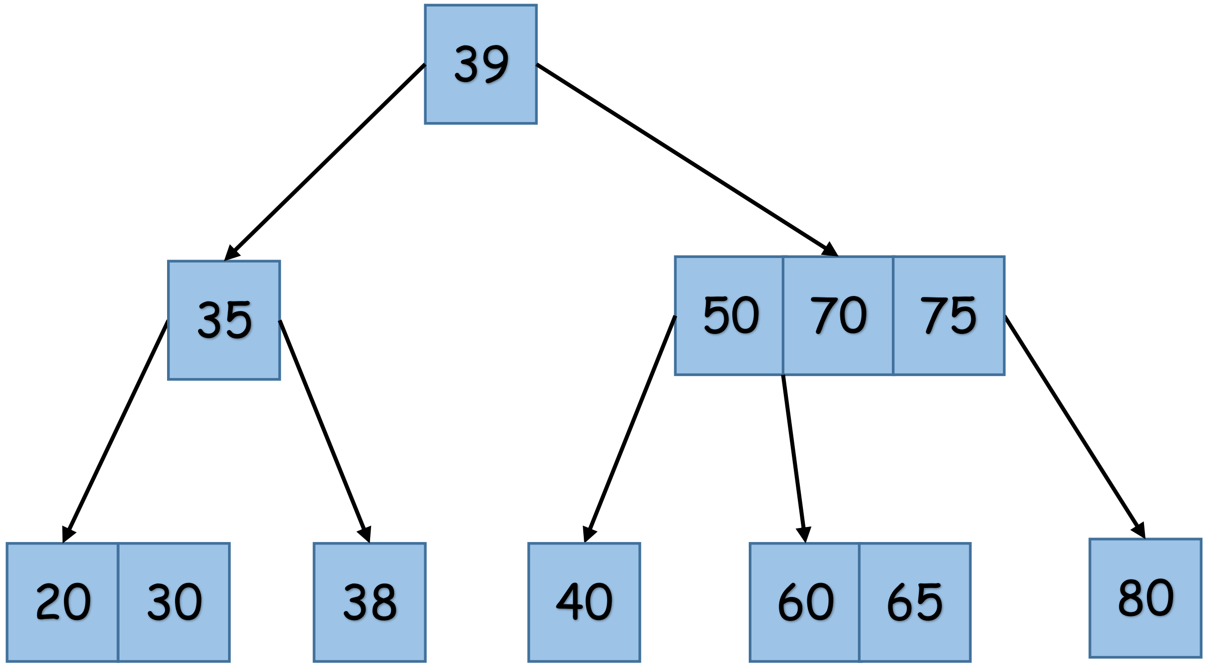
A 2-3-4 tree can be transformed into a corresponding red-black tree

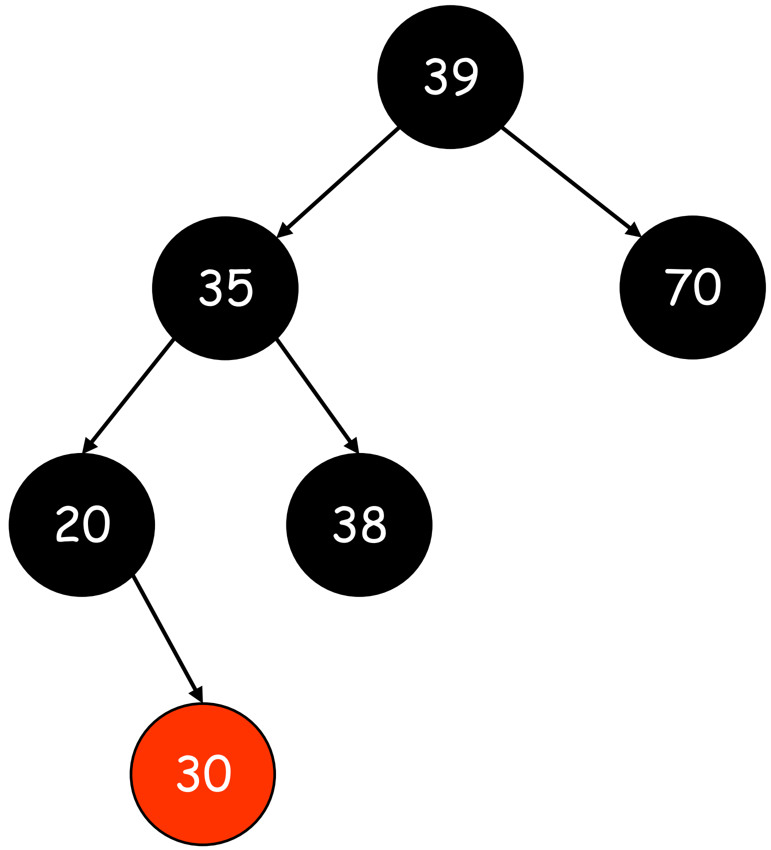




Red Black Trees

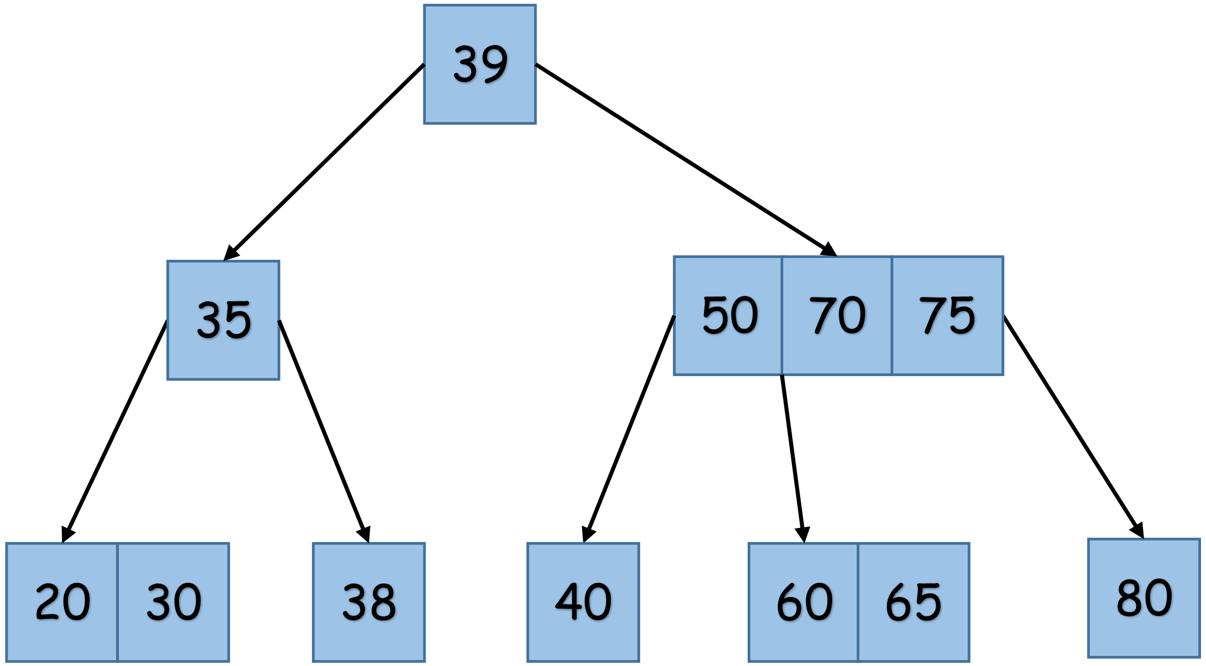
A 2-3-4 tree can be transformed into a corresponding red-black tree

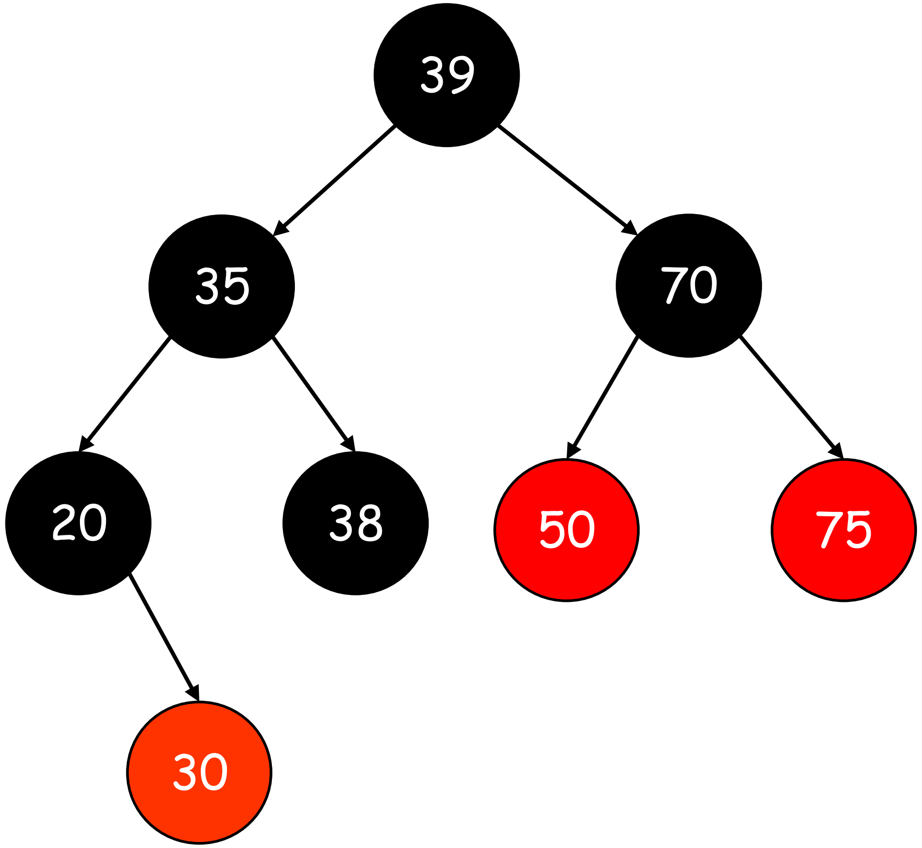




Red Black Trees

A 2-3-4 tree can be transformed into a corresponding red-black tree

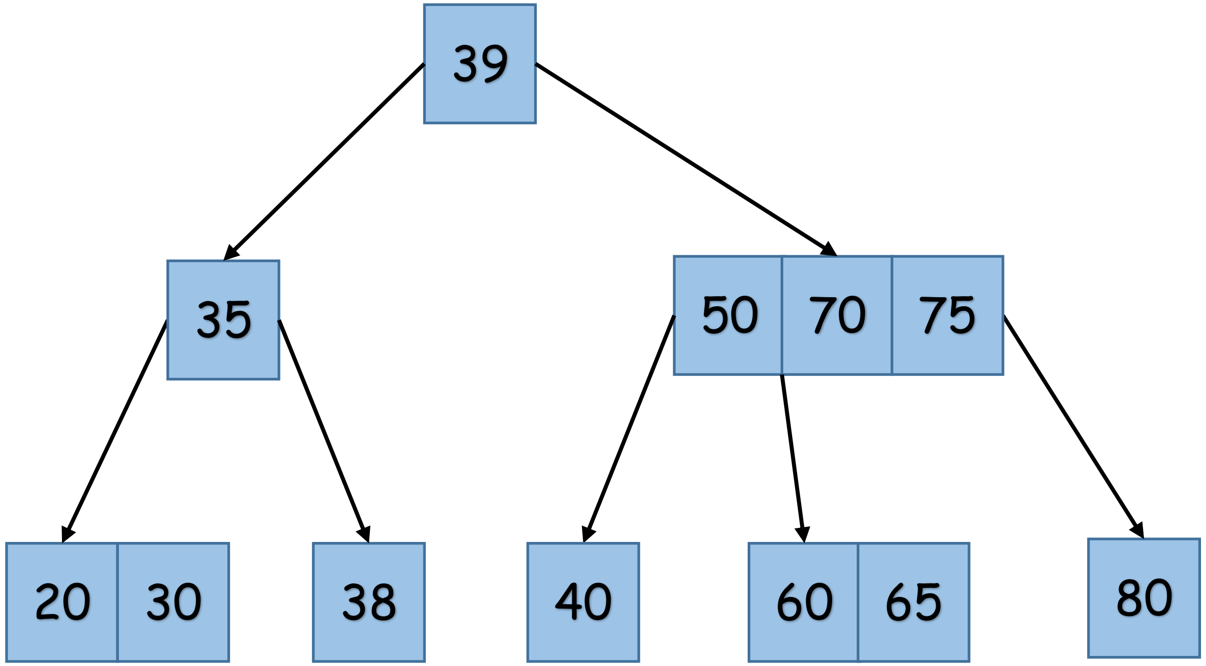


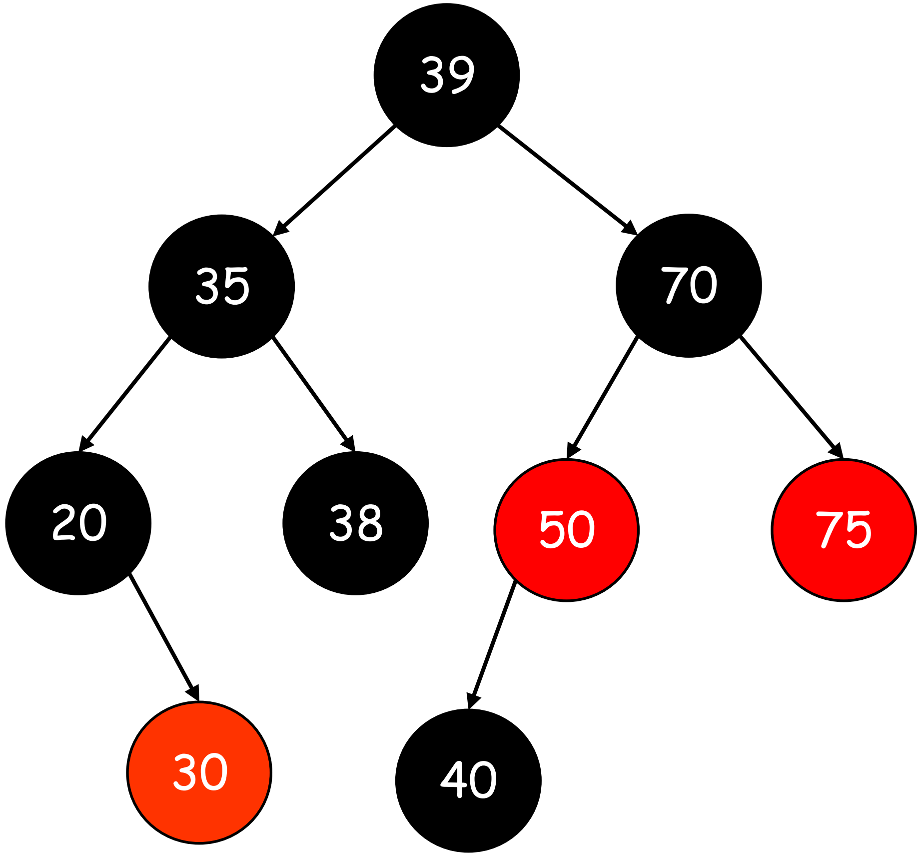


Root Property: The root is black
Red Property: The parent of a red node is black

Red Black Trees

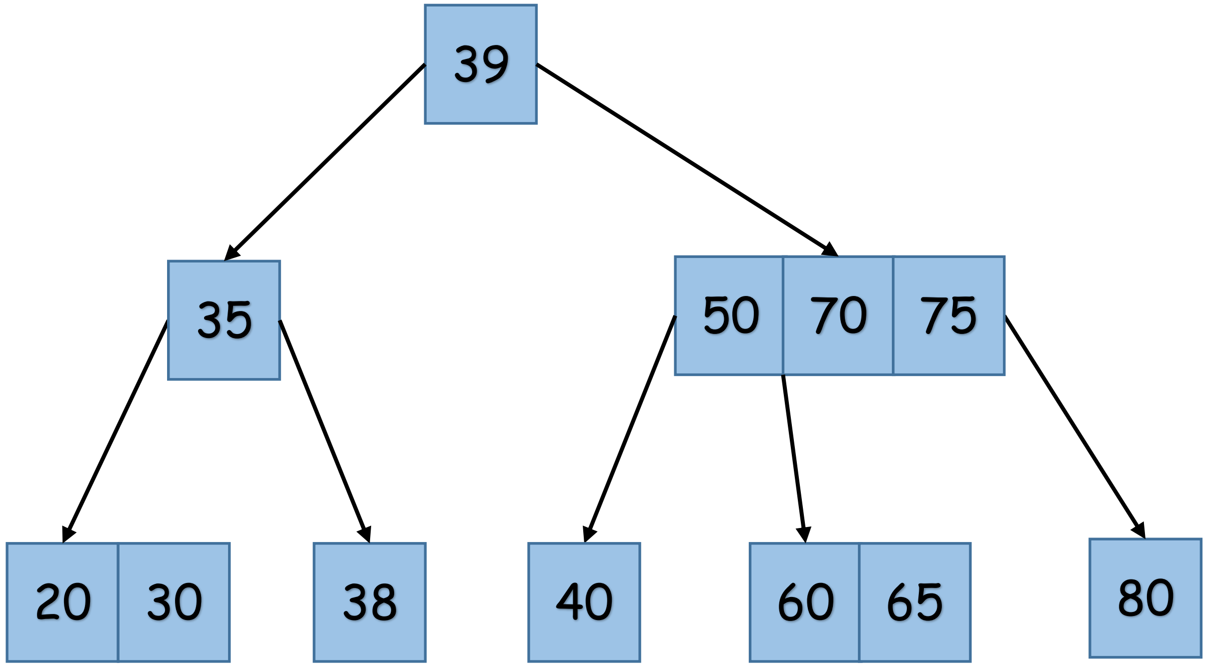
A 2-3-4 tree can be transformed into a corresponding red-black tree

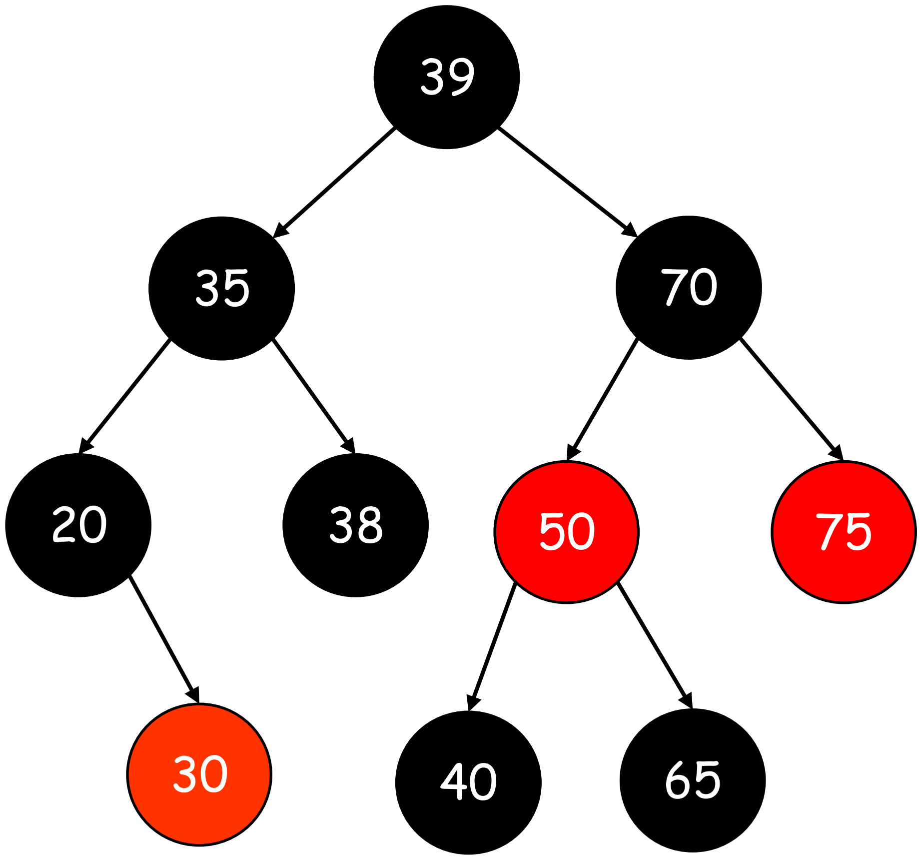




Red Black Trees

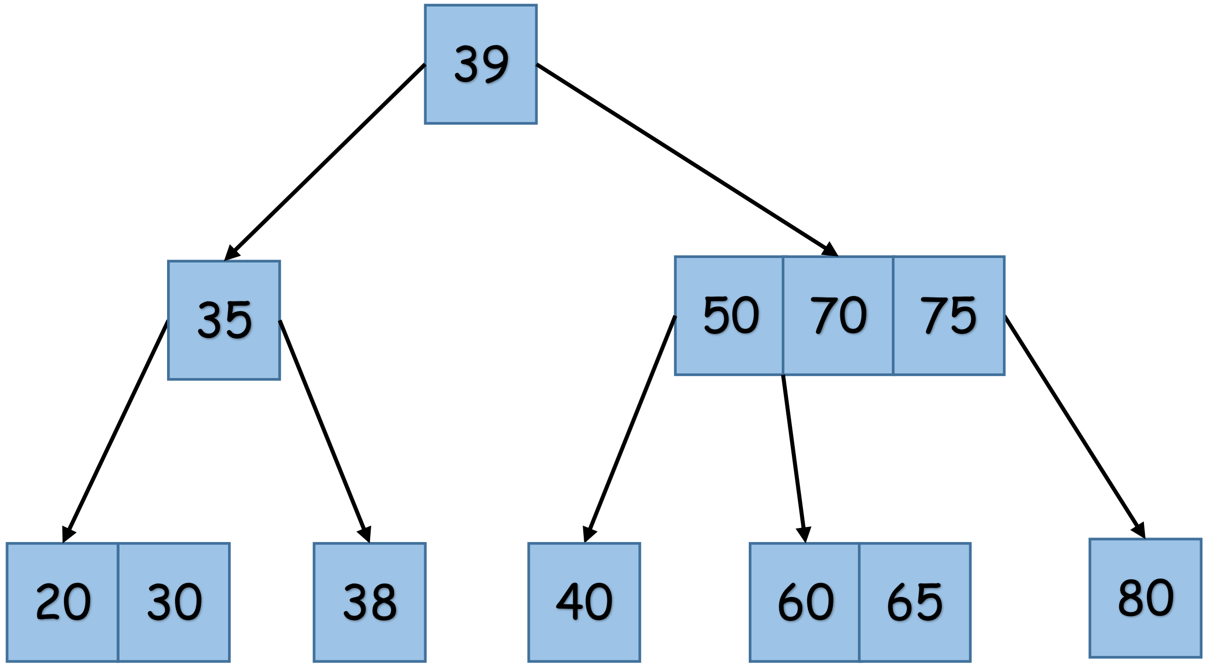
A 2-3-4 tree can be transformed into a corresponding red-black tree

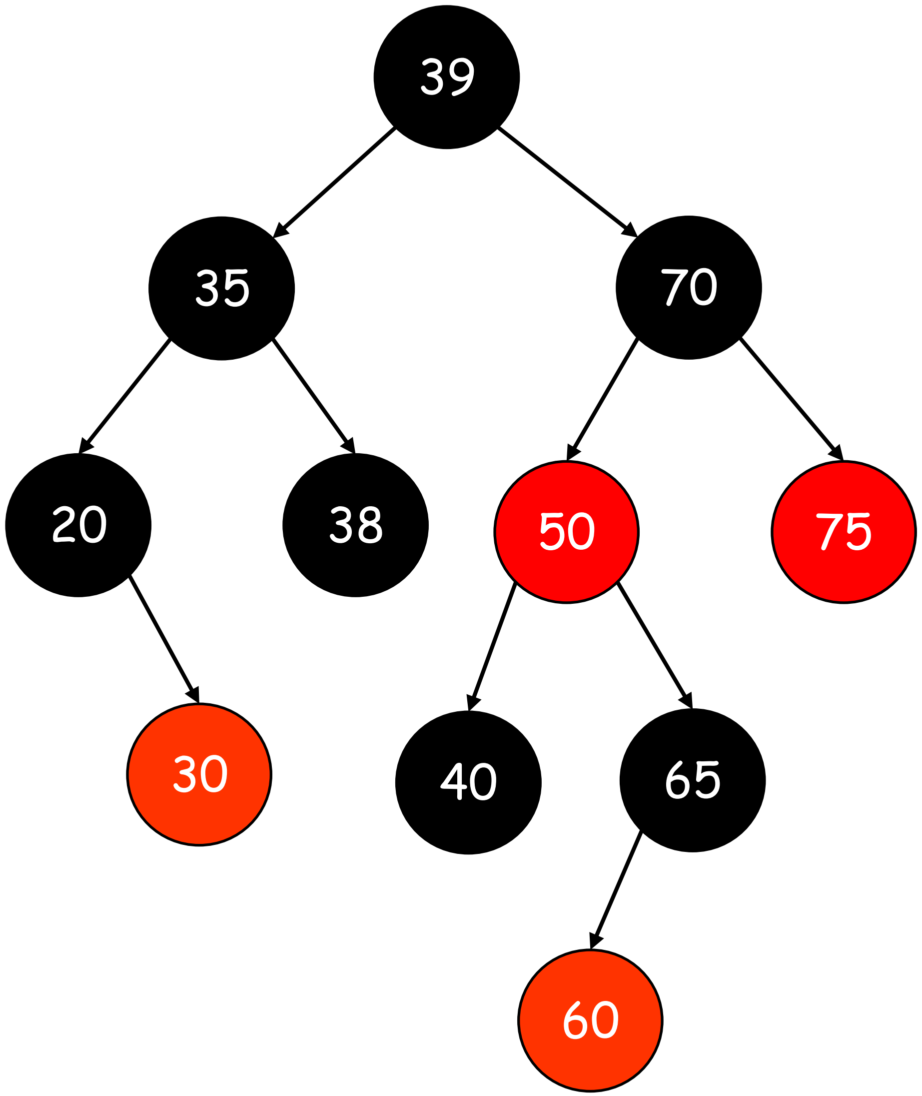




Red Black Trees

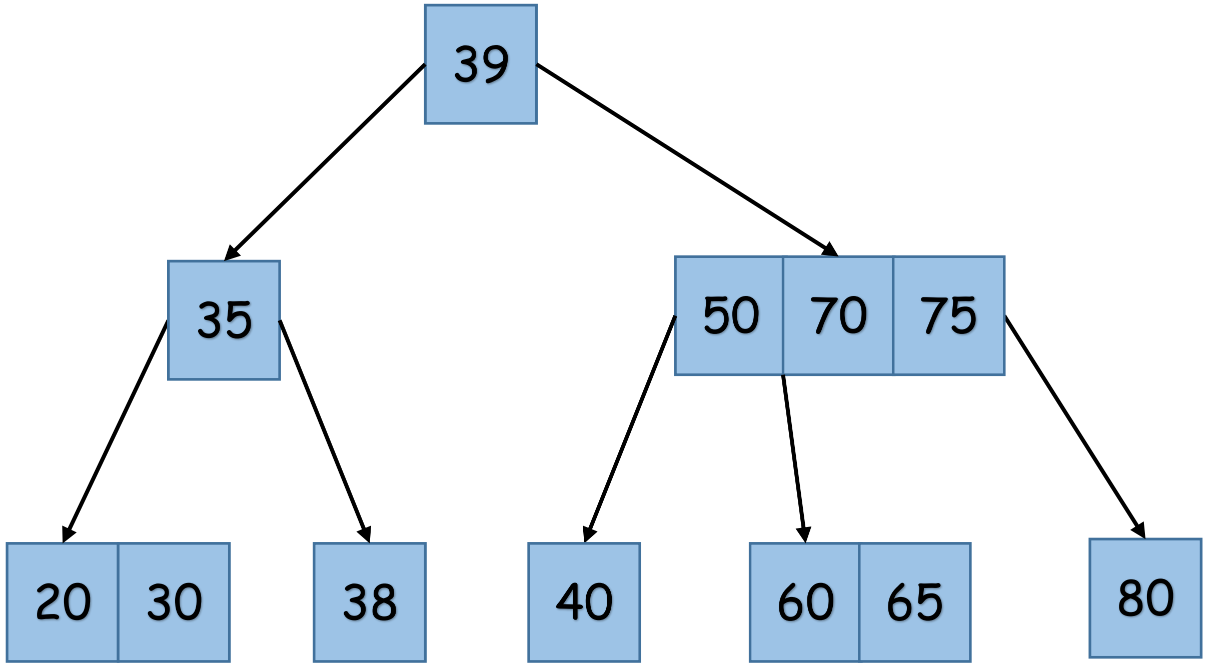
A 2-3-4 tree can be transformed into a corresponding red-black tree

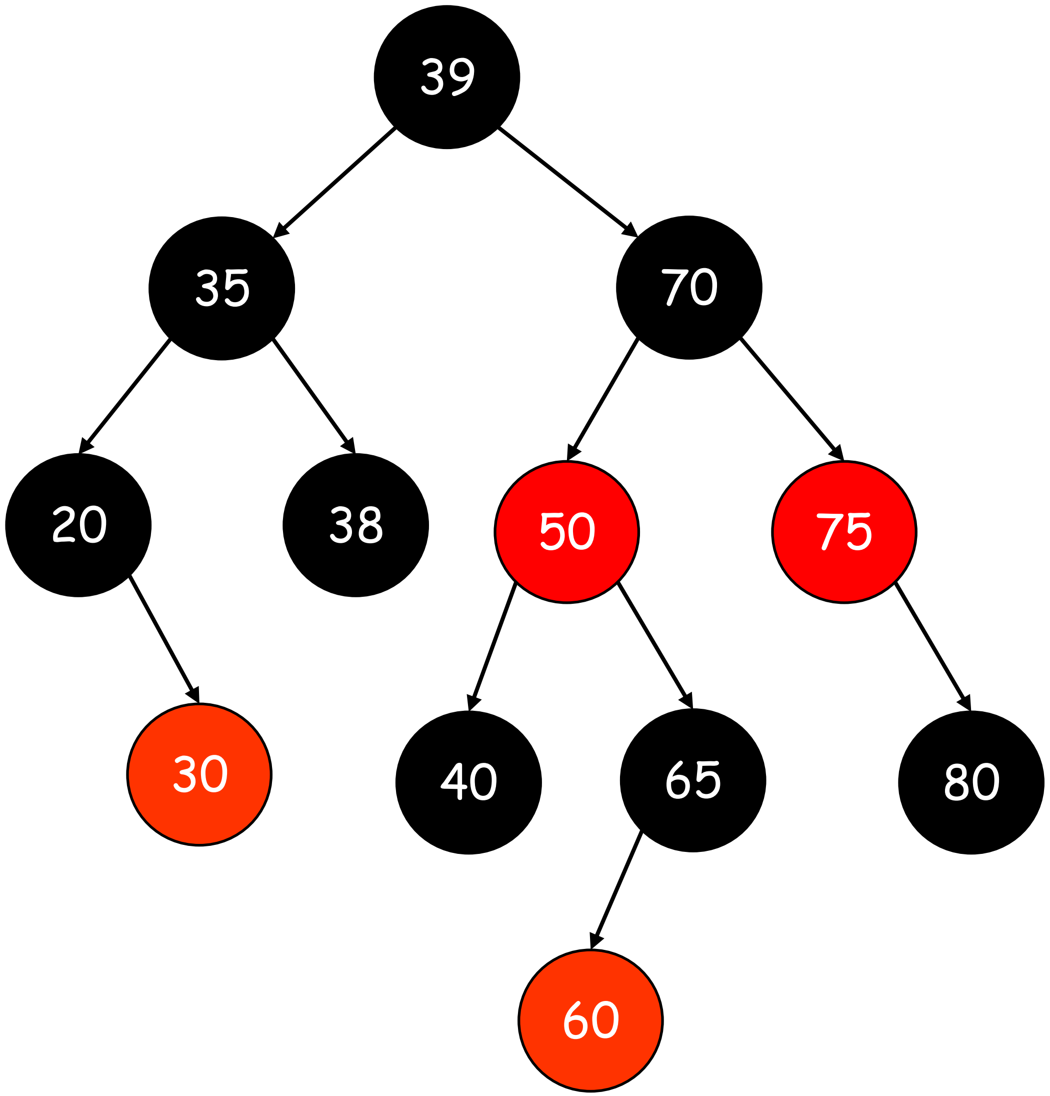




Red Black Trees

A 2-3-4 tree can be transformed into a corresponding red-black tree

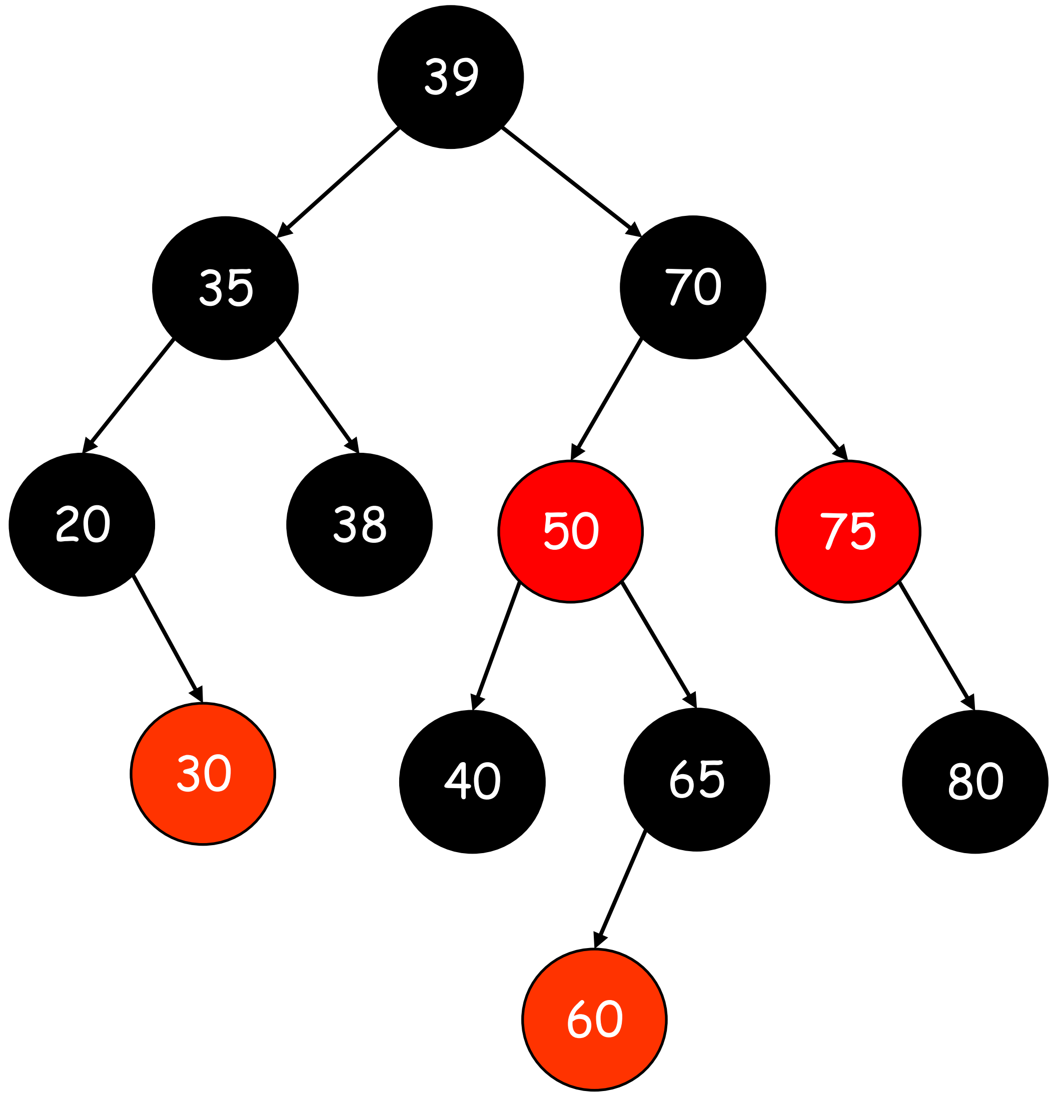




What can you infer about the depth?

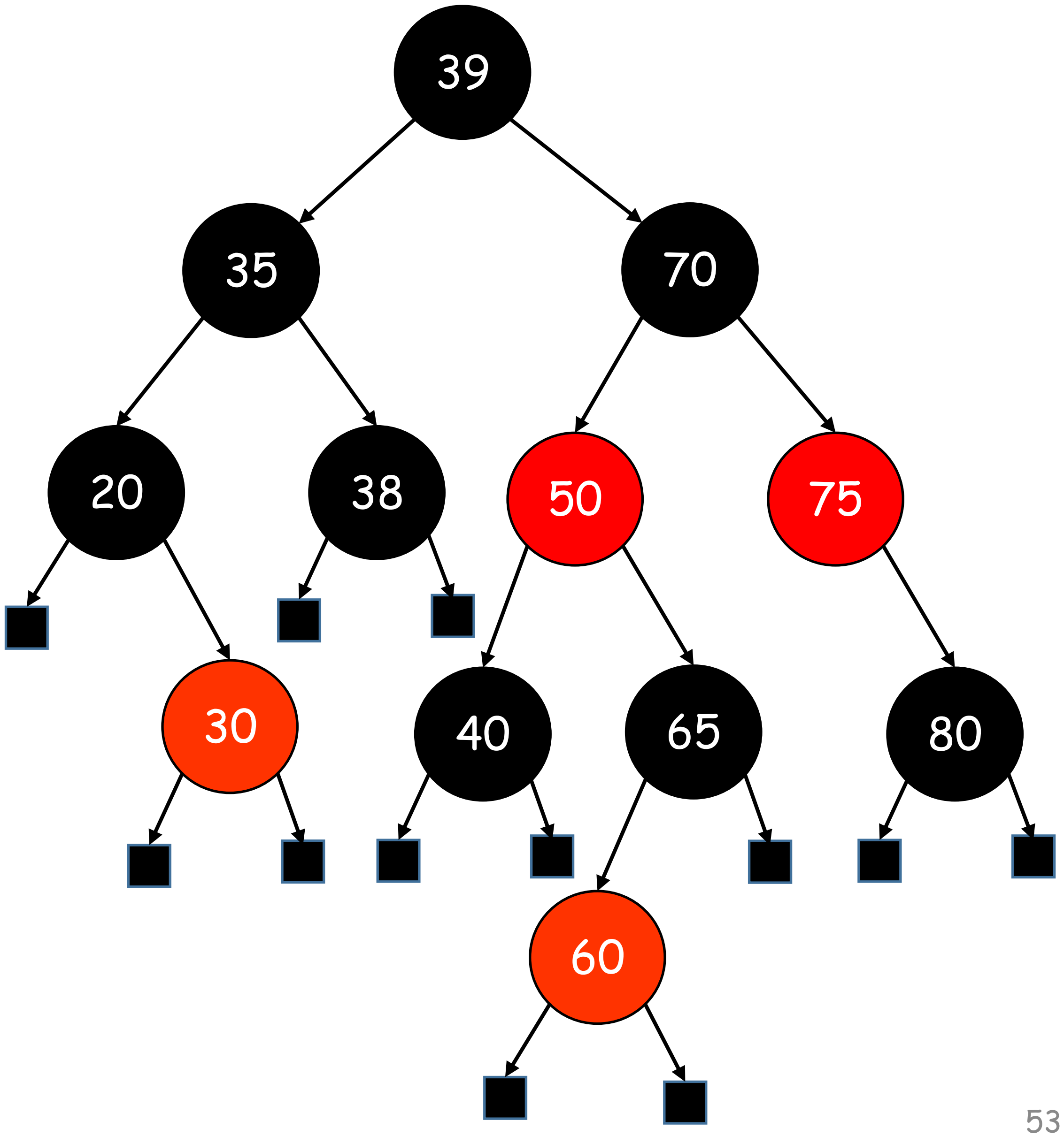
Red Black Trees

<u>Root Property</u>:
The root is black
<u>Red Property</u>:
The parent of a red node is black
<u>Depth Property</u>:
The number of black nodes on each path from the root to a leaf node is the same



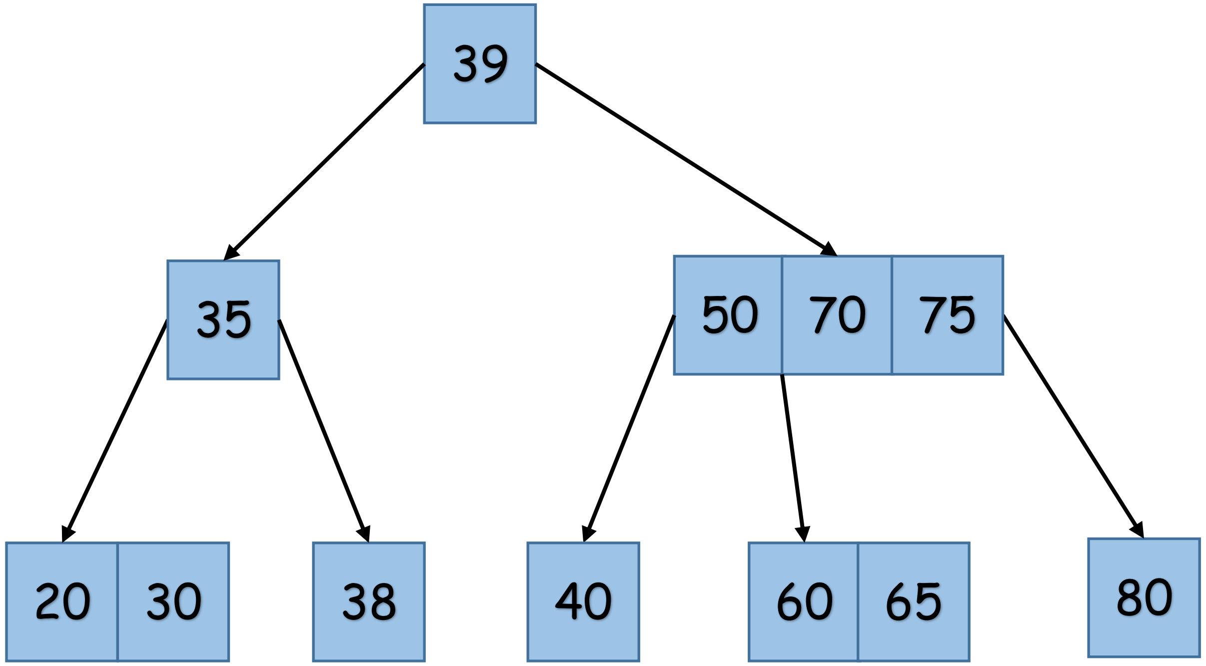
Red Black Trees

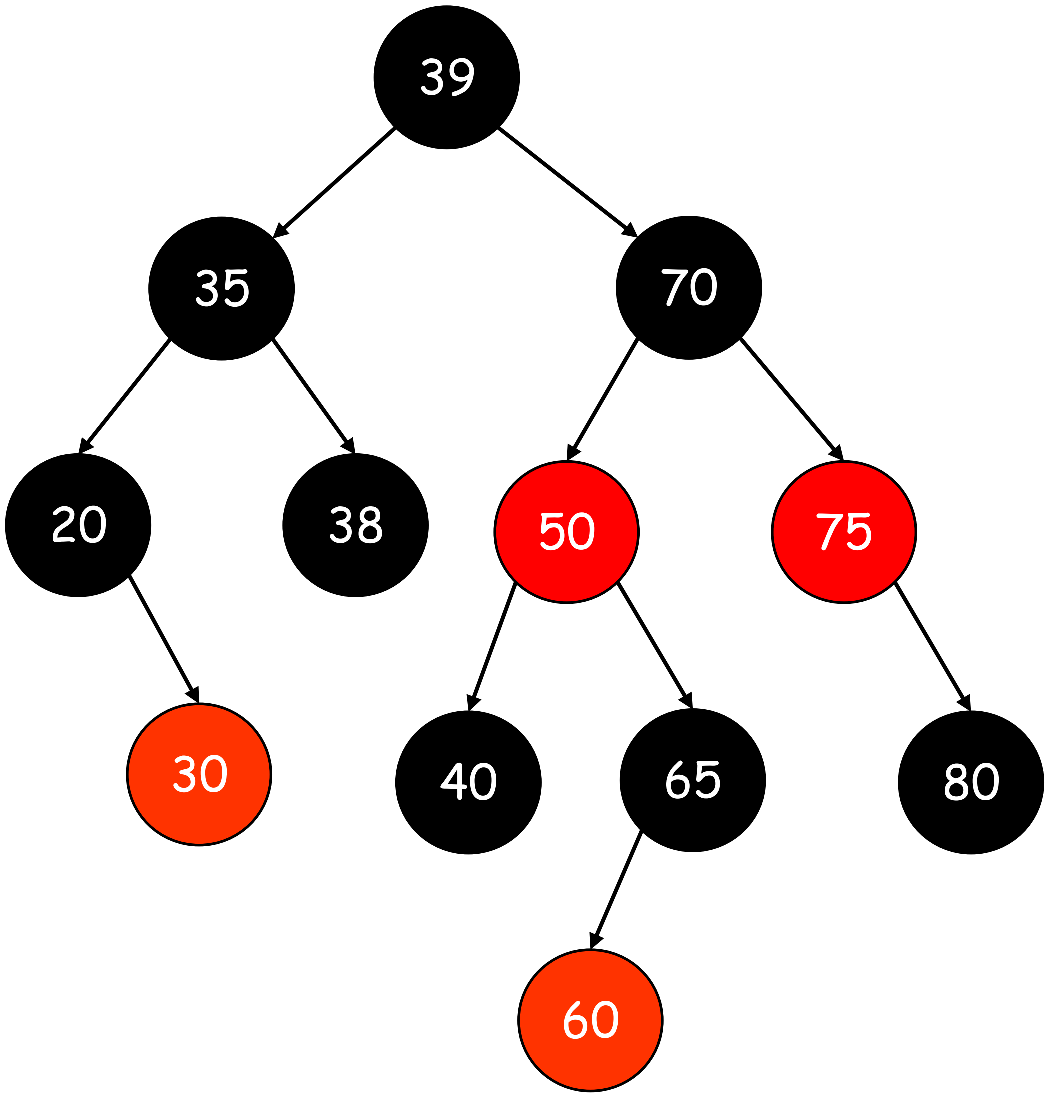
Root Property:
The root is black
Red Property:
The parent of a red node is black
Depth Property:
The number of black nodes on each path from the root to a leaf node is the same
Leaf Nodes:
All leaf nodes are black nil nodes



Red Black Trees

A red-black tree can be transformed into a corresponding 2-3-4 tree

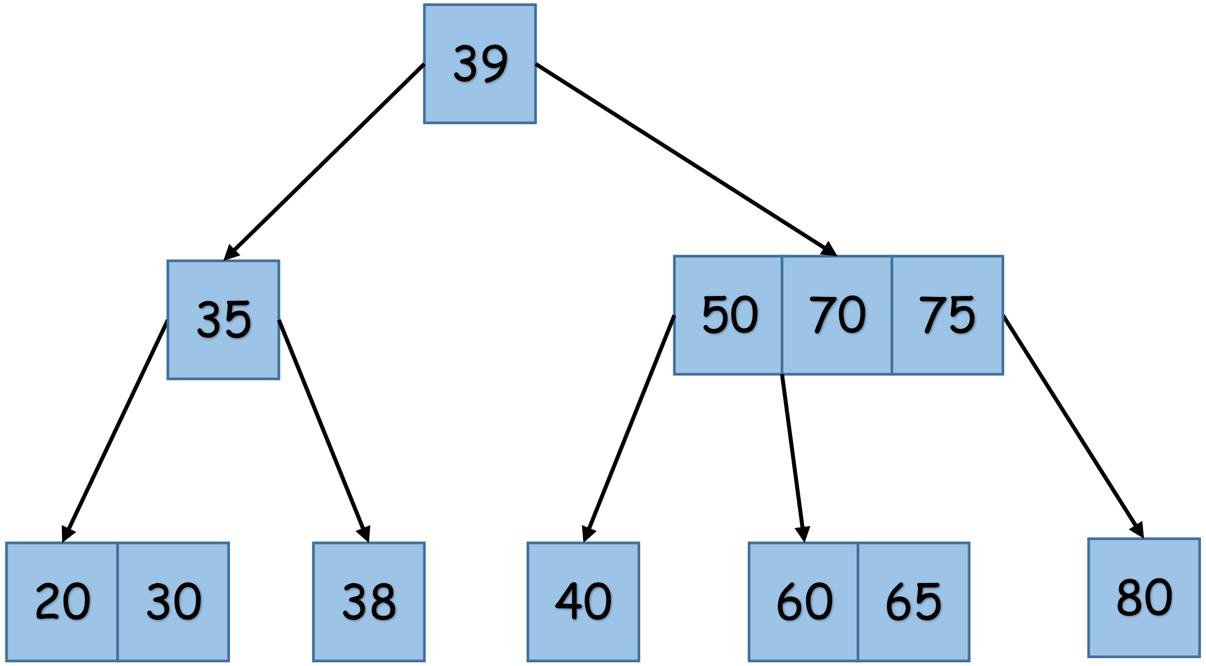


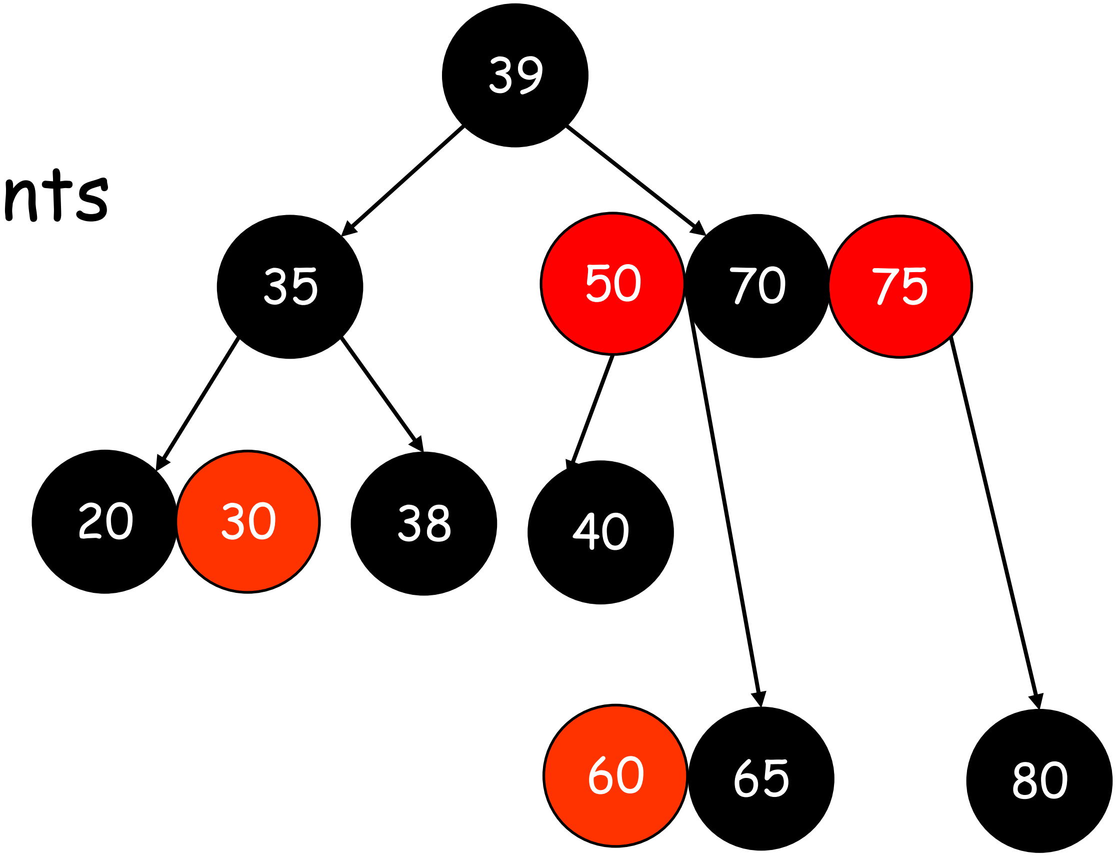


Red Black Trees

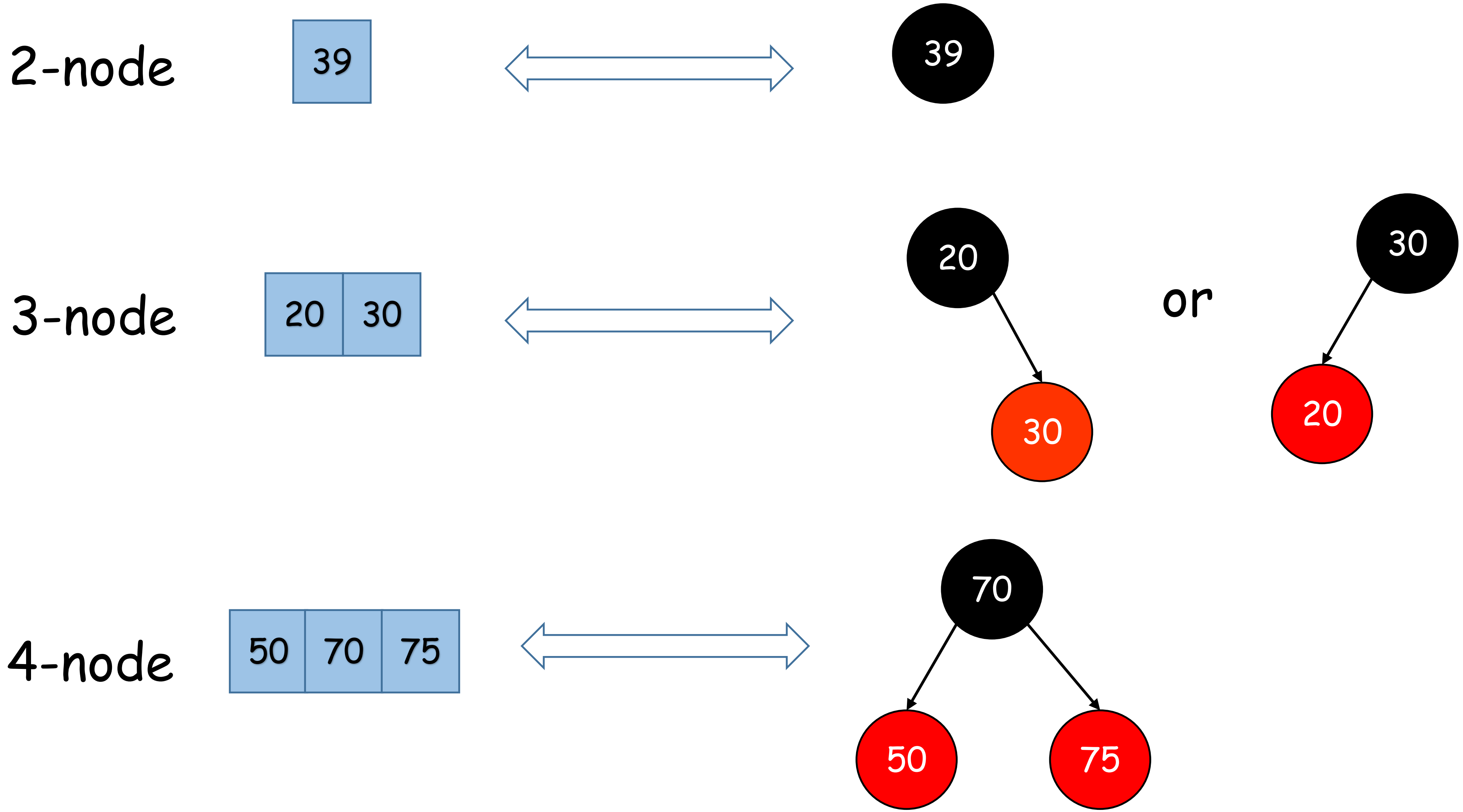
A red-black tree can be transformed into a corresponding 2-3-4 tree

Move the red nodes up to their parents

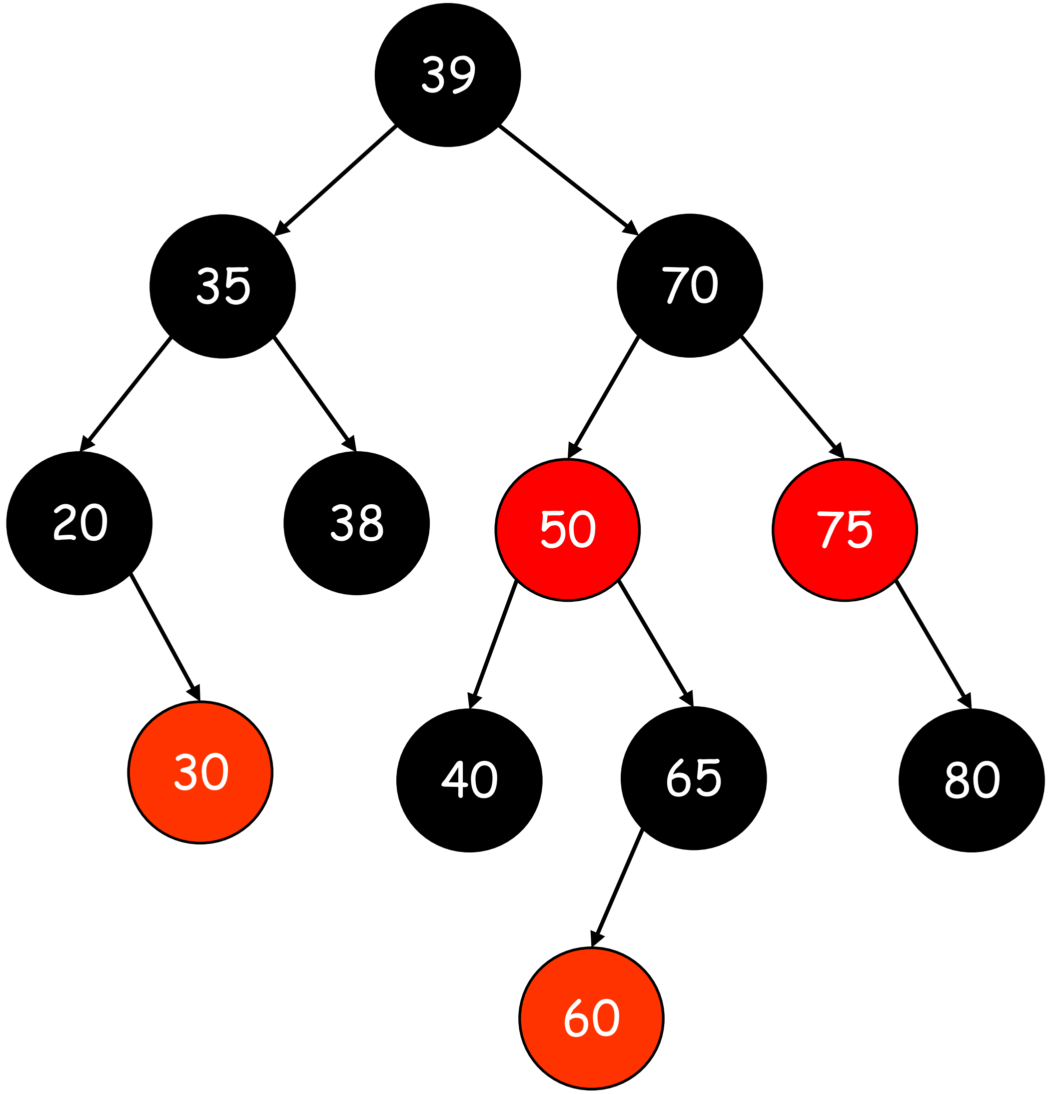




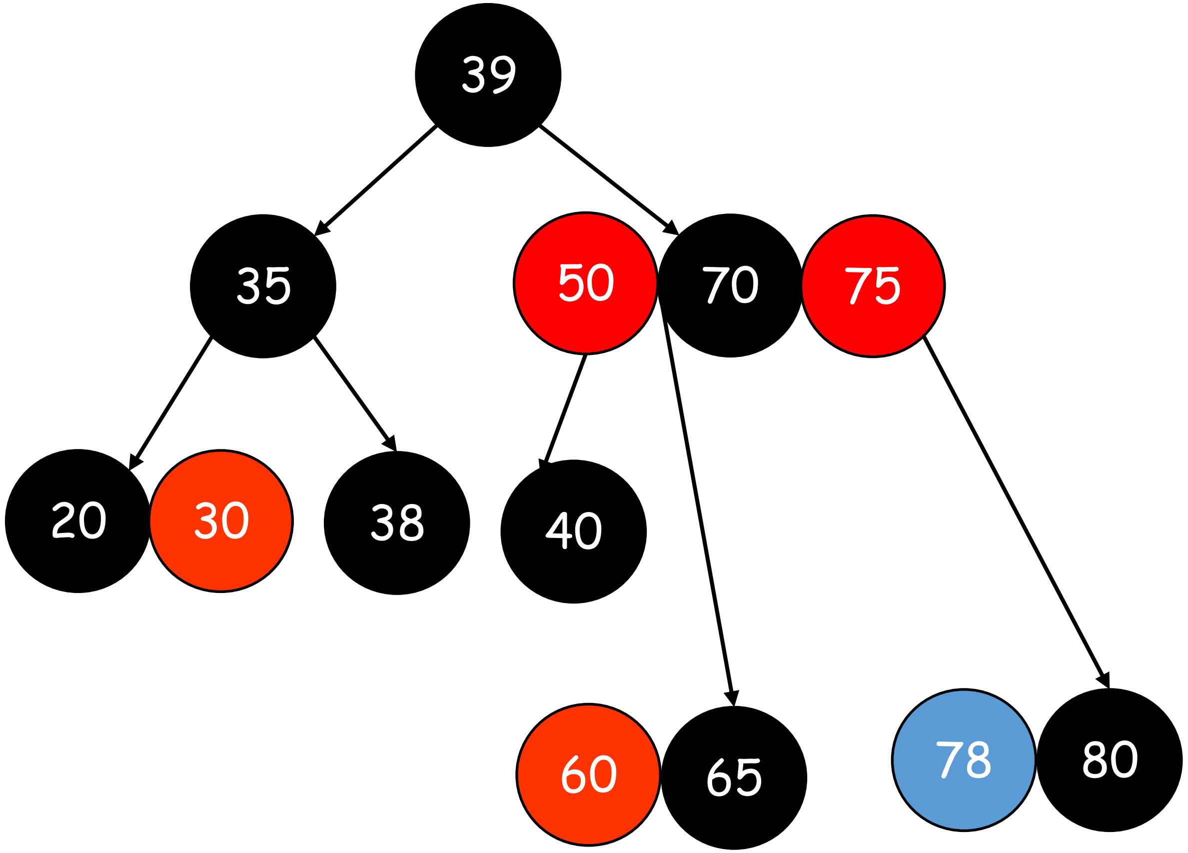
Corresponding Transformation



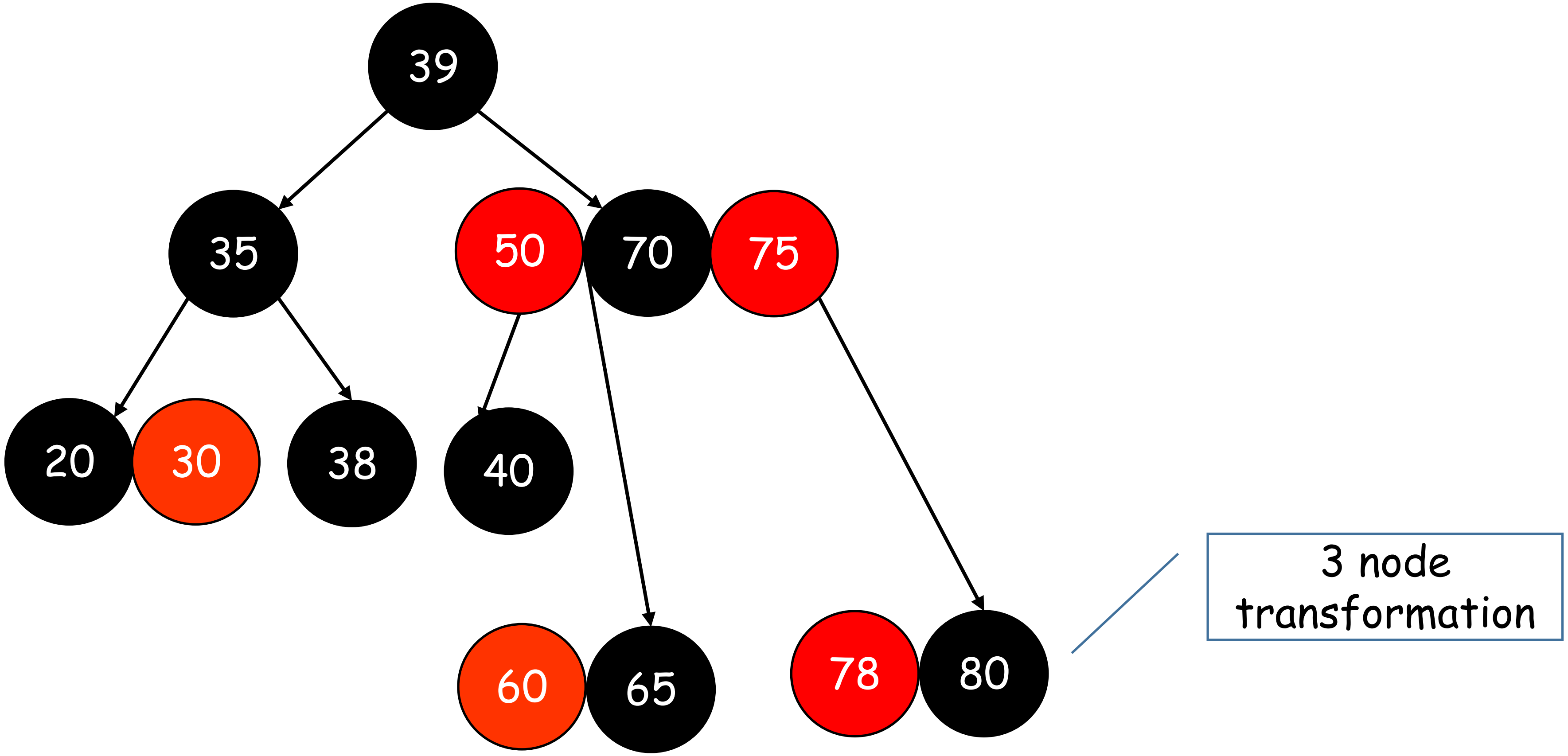
Insert(78)



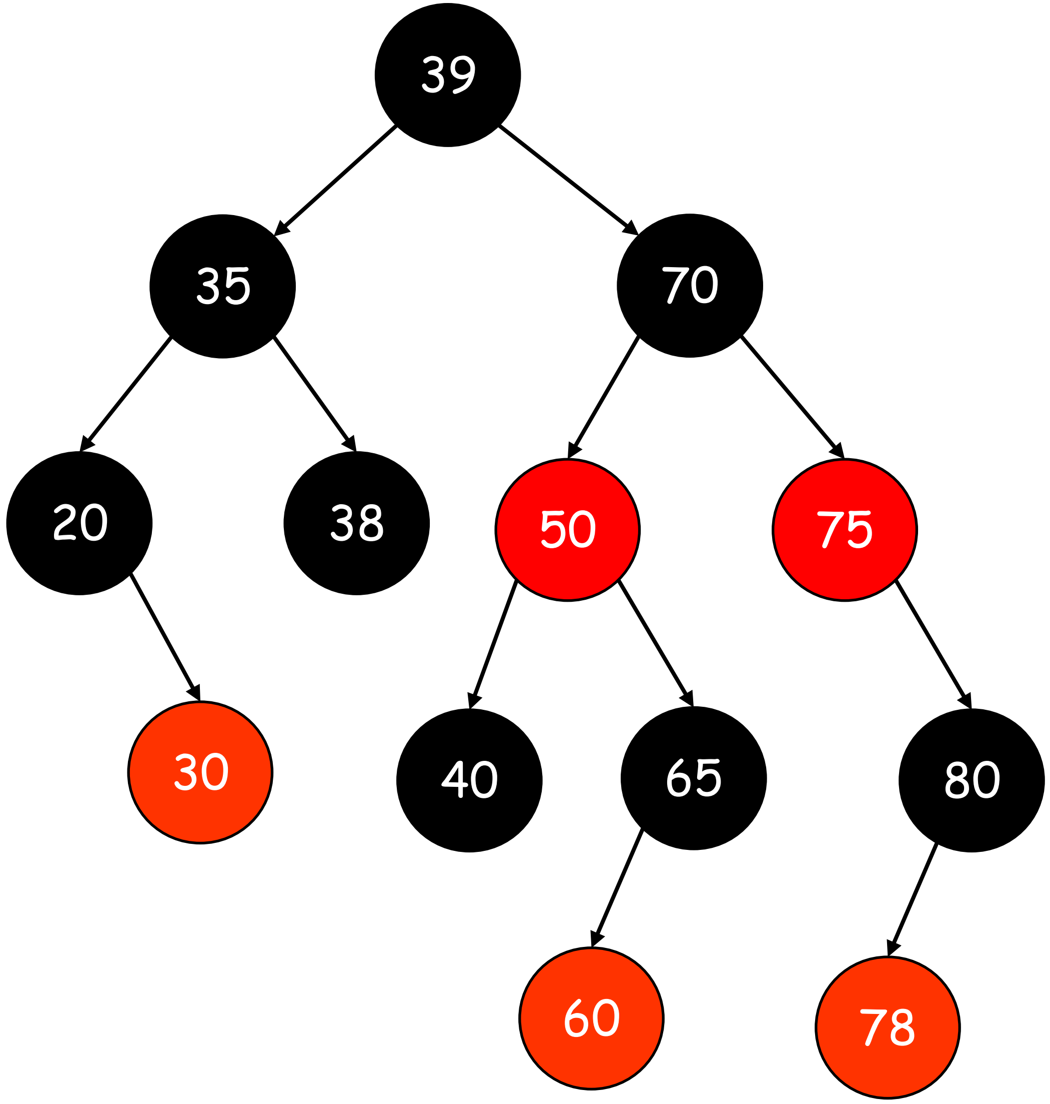
Insert(78)



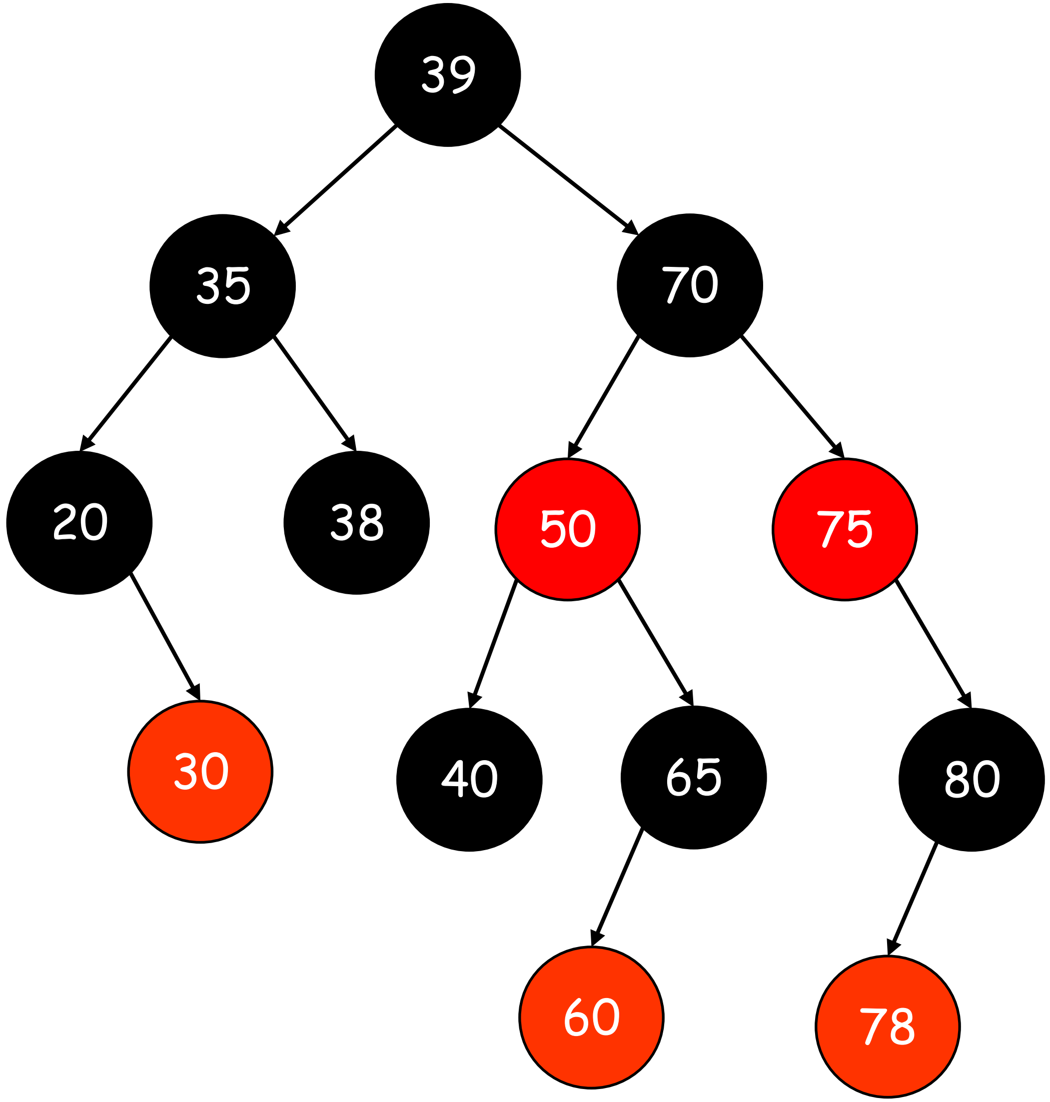
Insert(78)



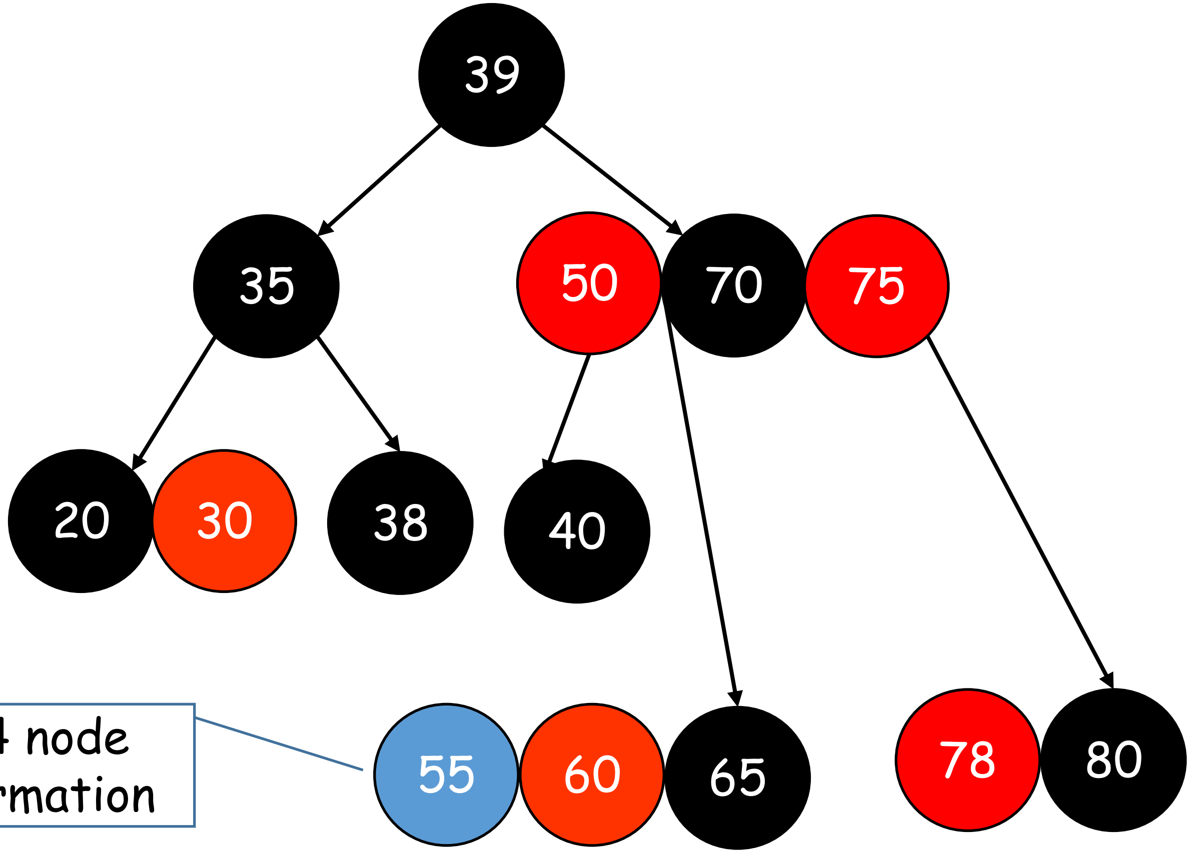
Insert(78)



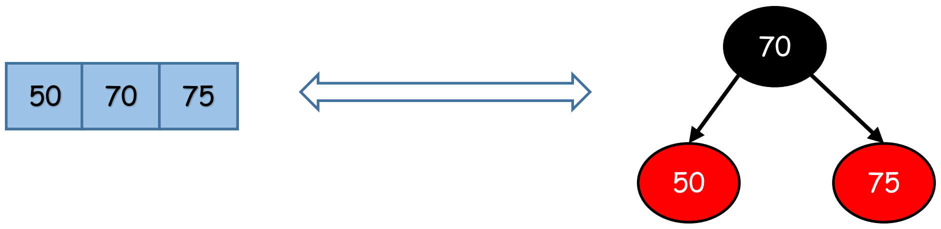
Insert(55)



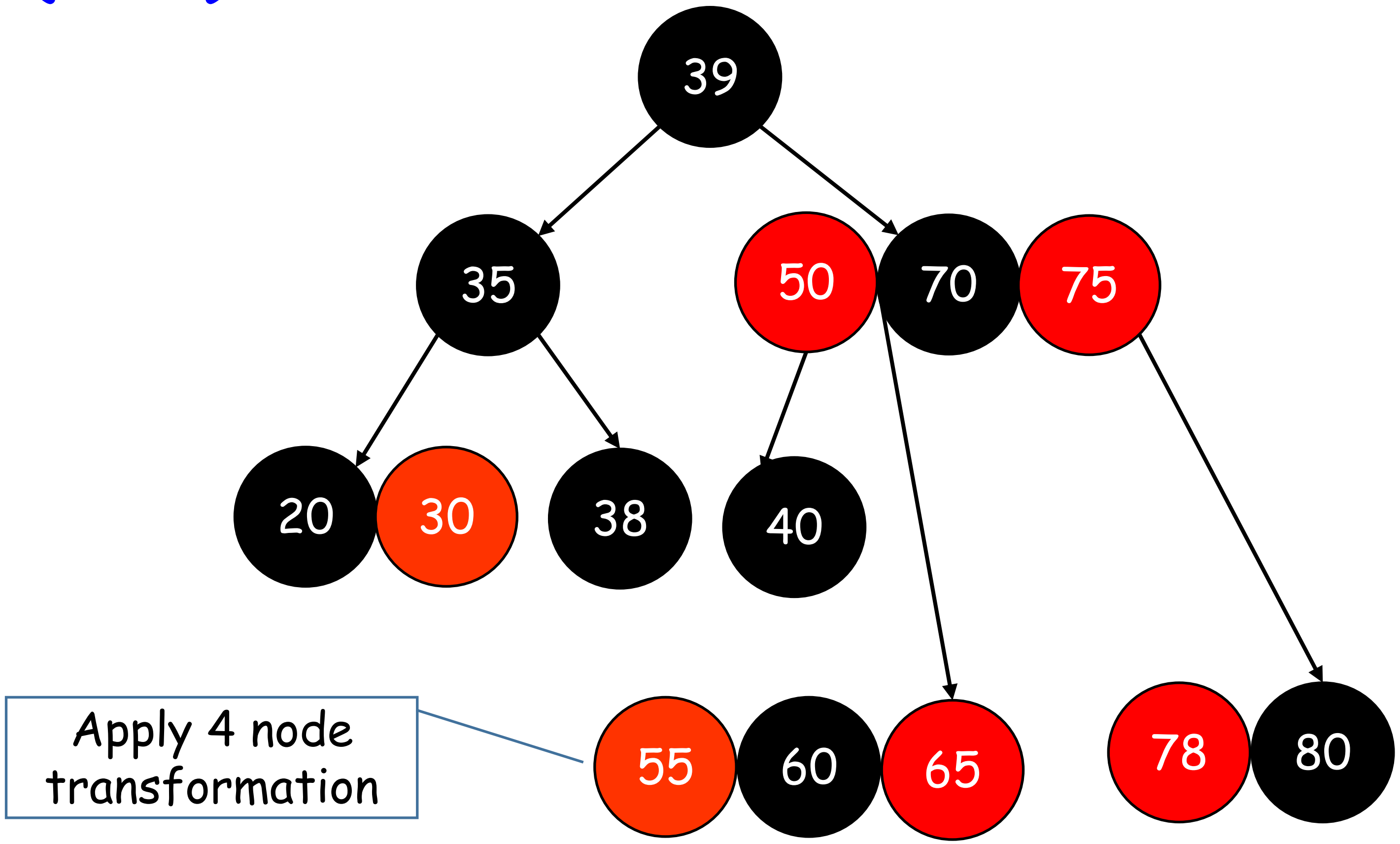
Insert(55)

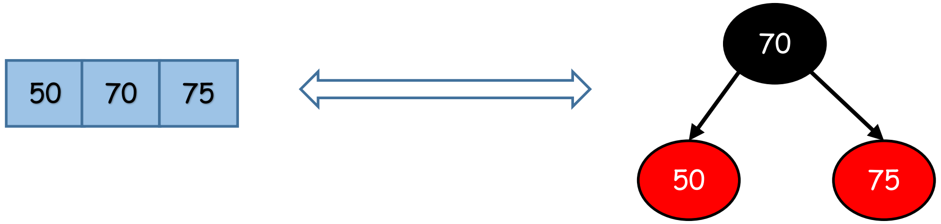


Apply 4 node transformation

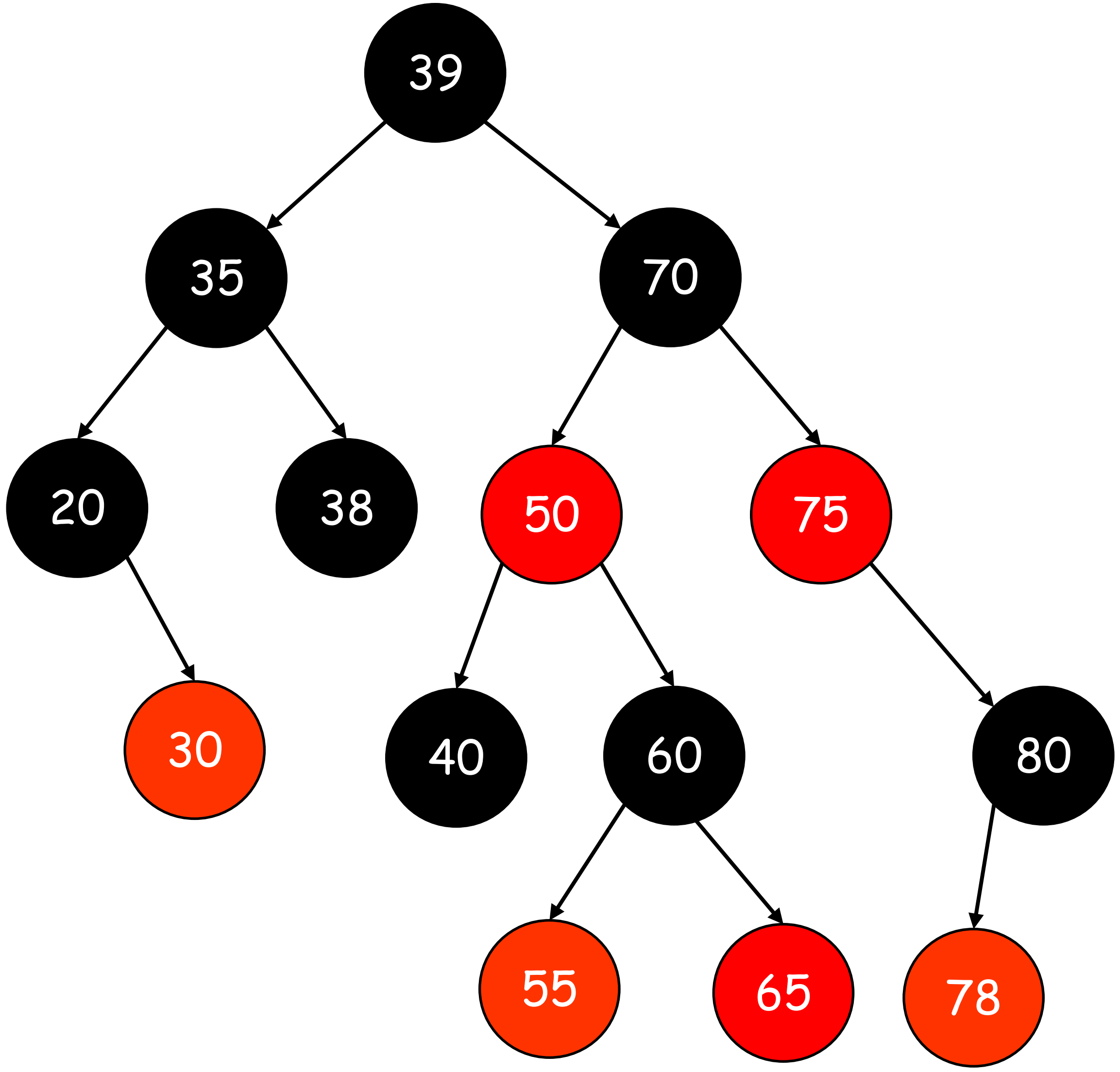


Insert(55)

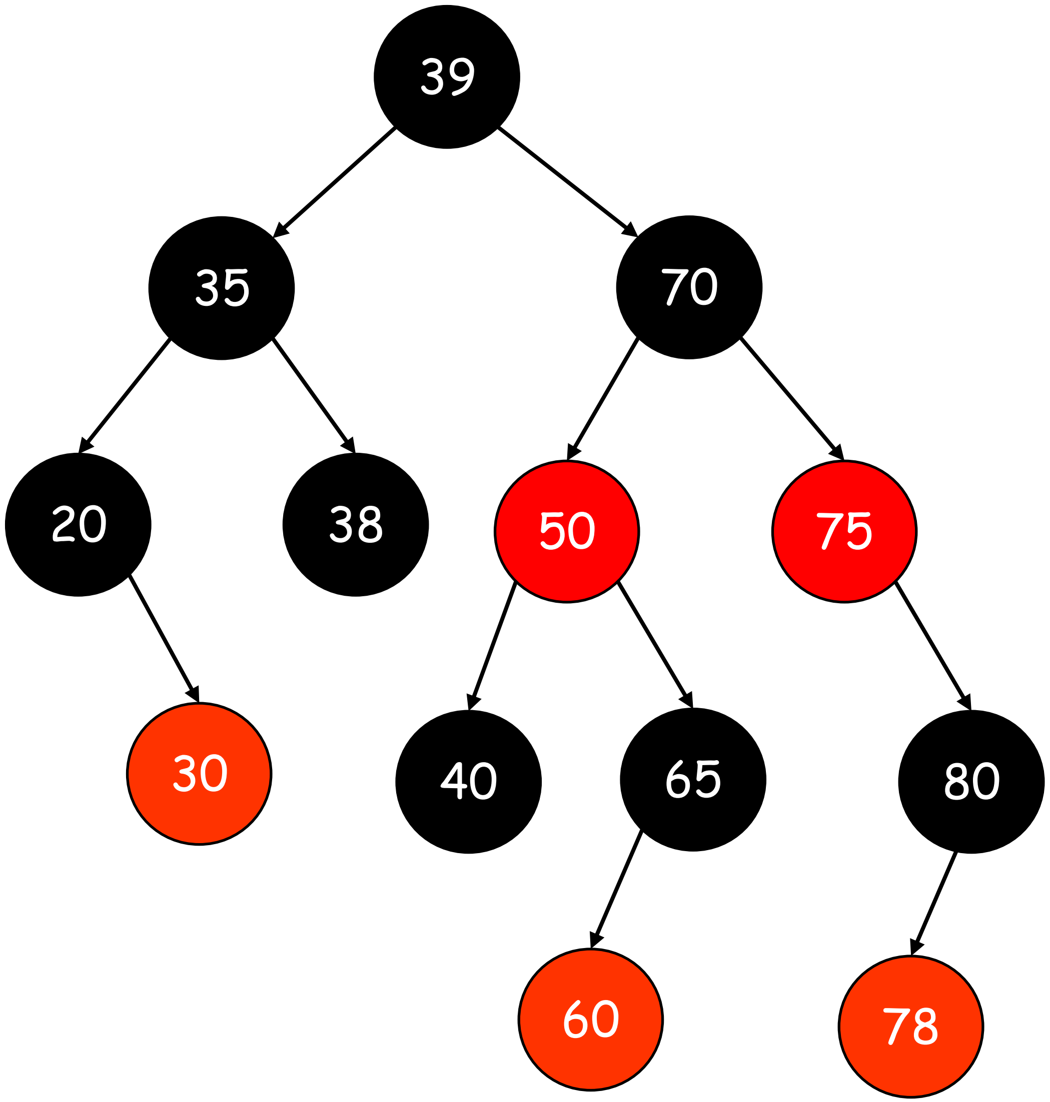

39
35
50
70
75
20
30
38
40
Apply 4 node transformation
55
60
65
78
80


50
70
75
70
50
75

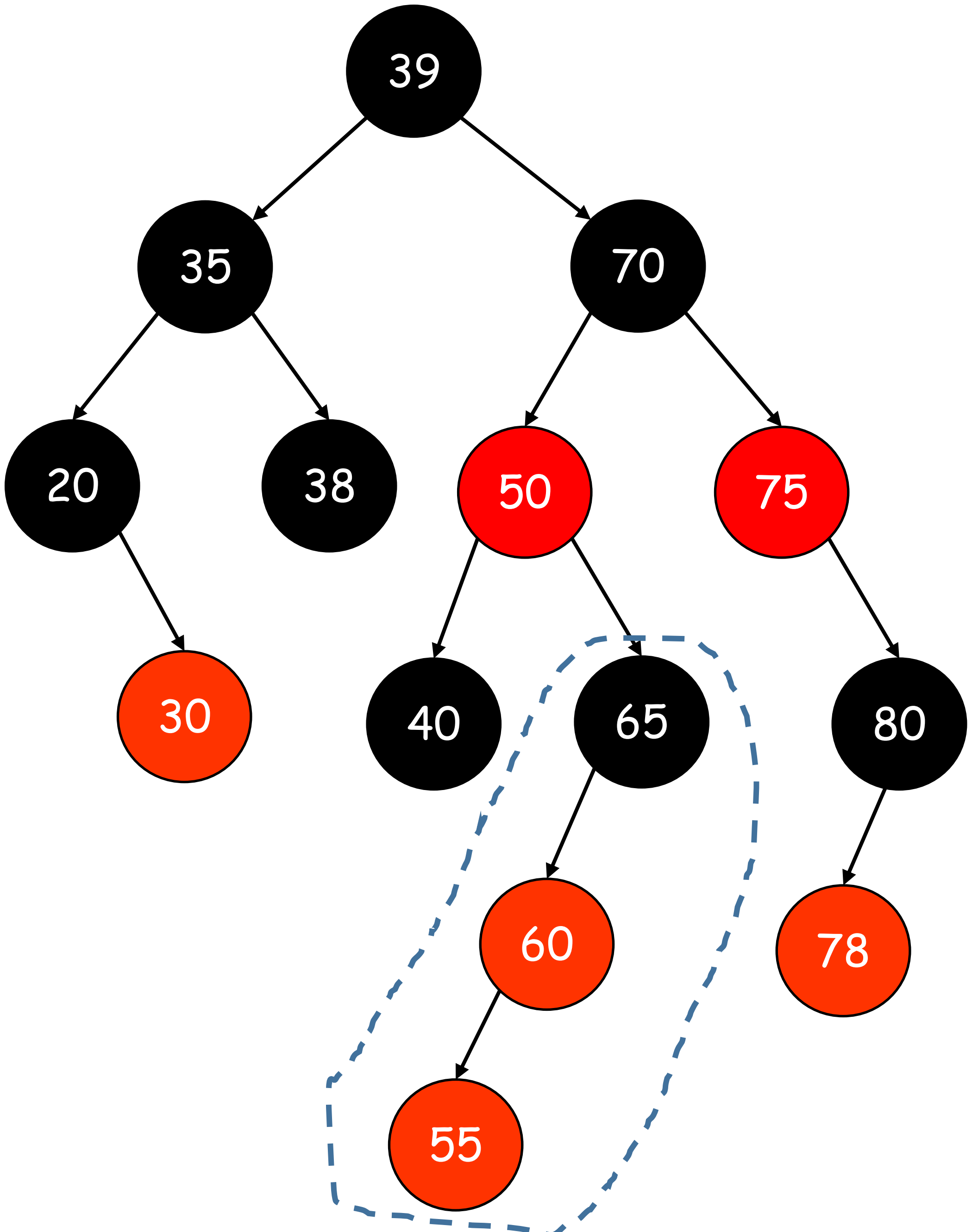
Insert(55)



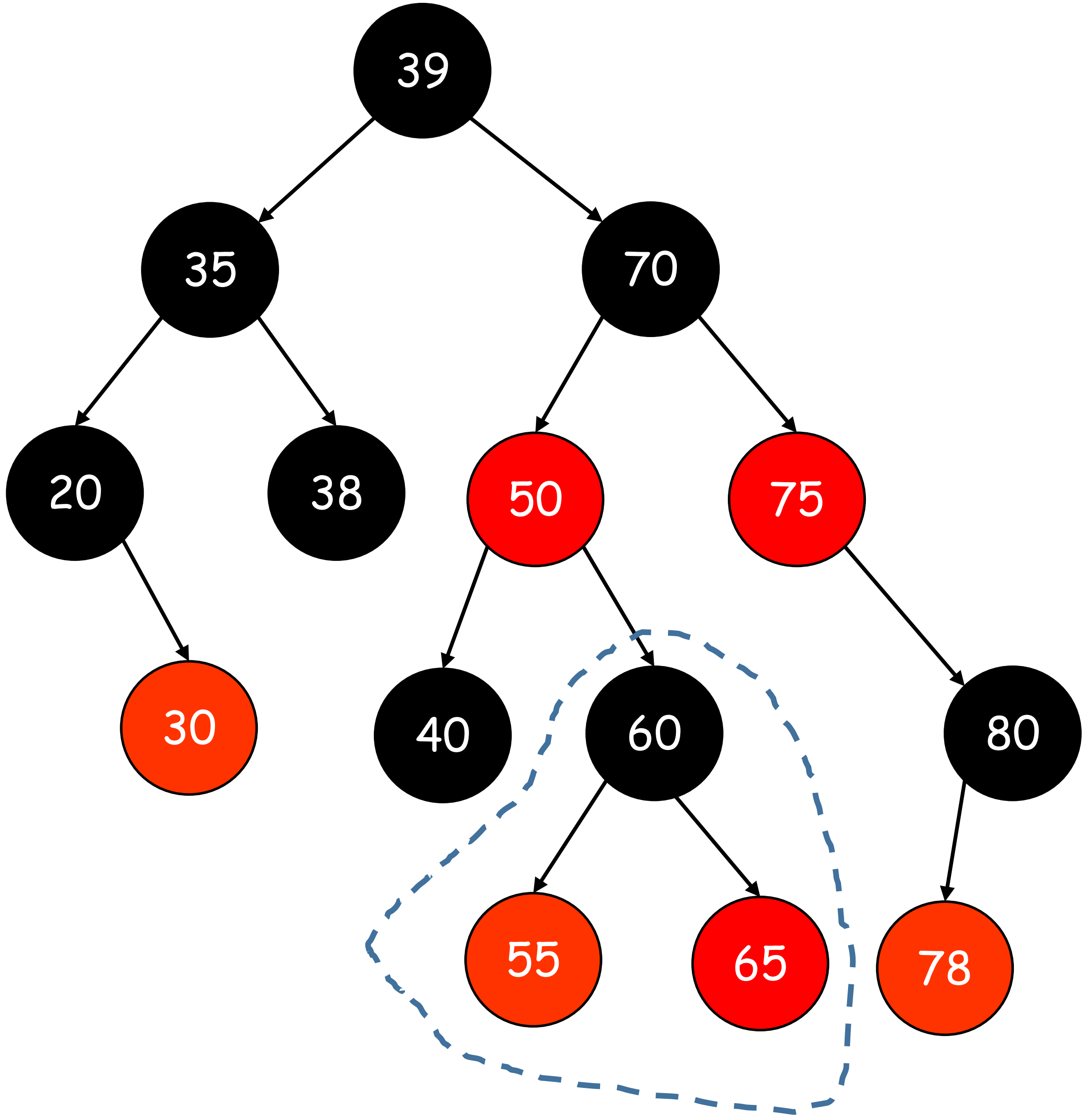
Insert(55)



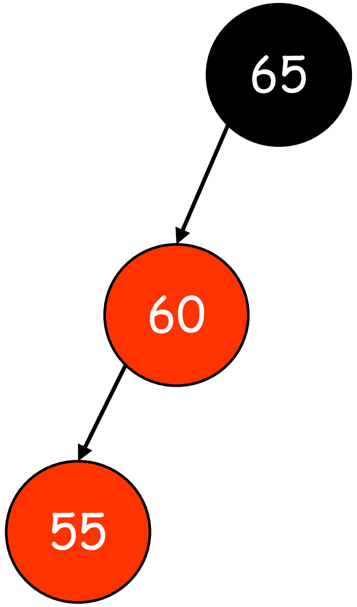
Insert(55)

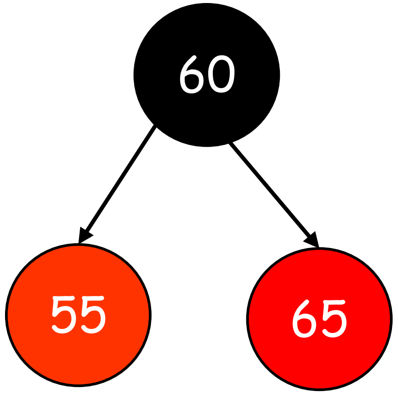


Insert(55)

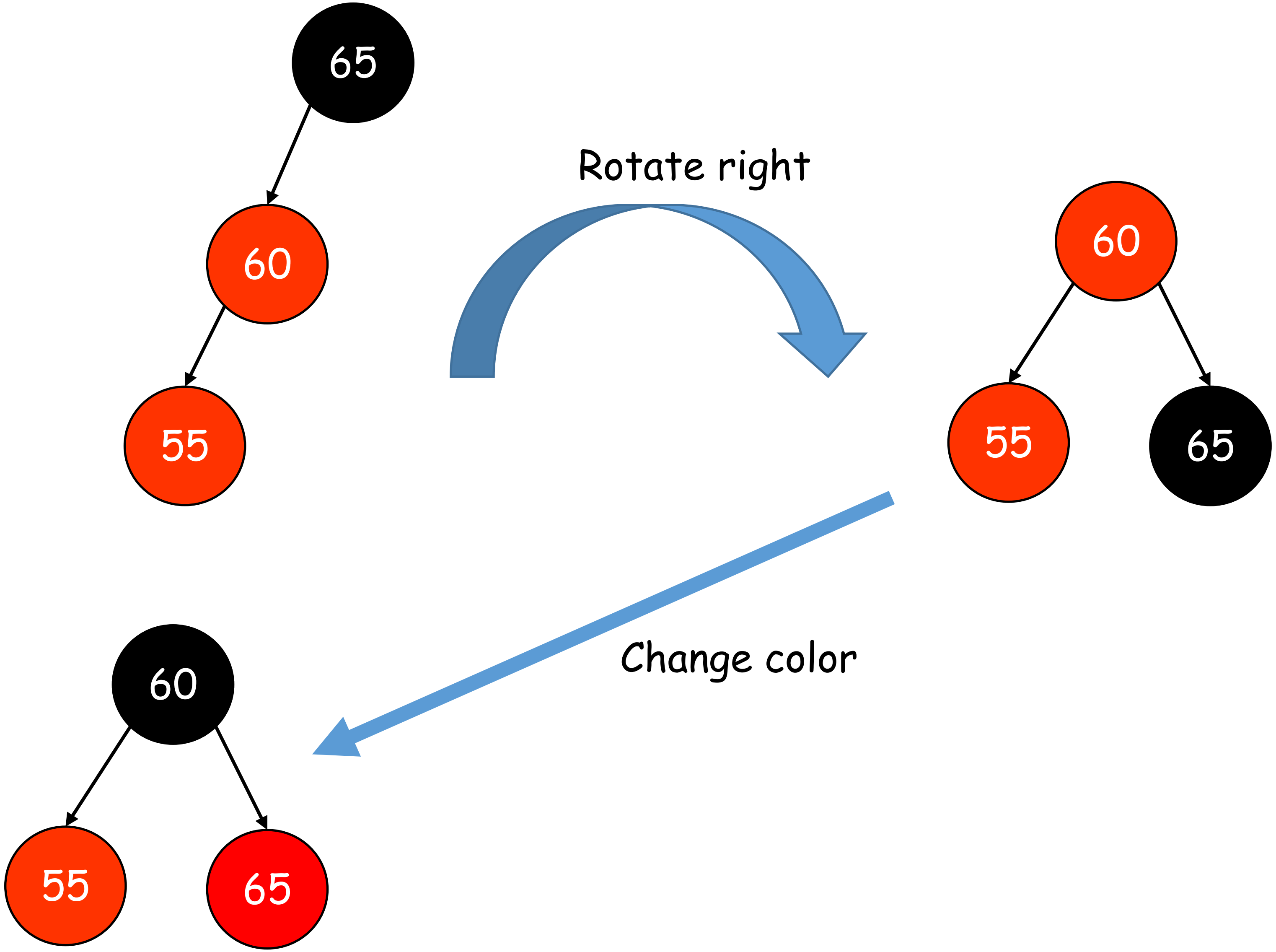


Insert(55)

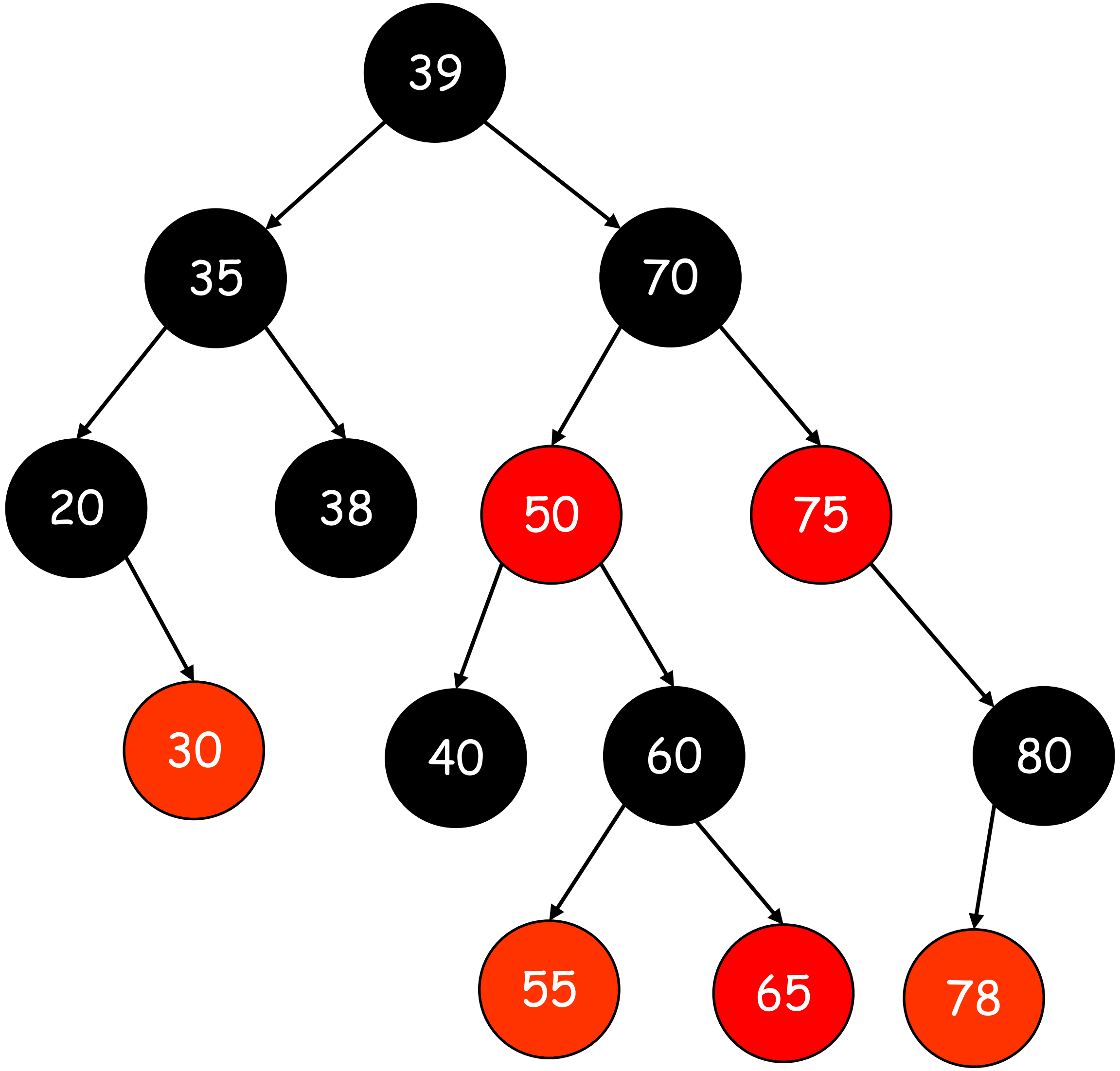




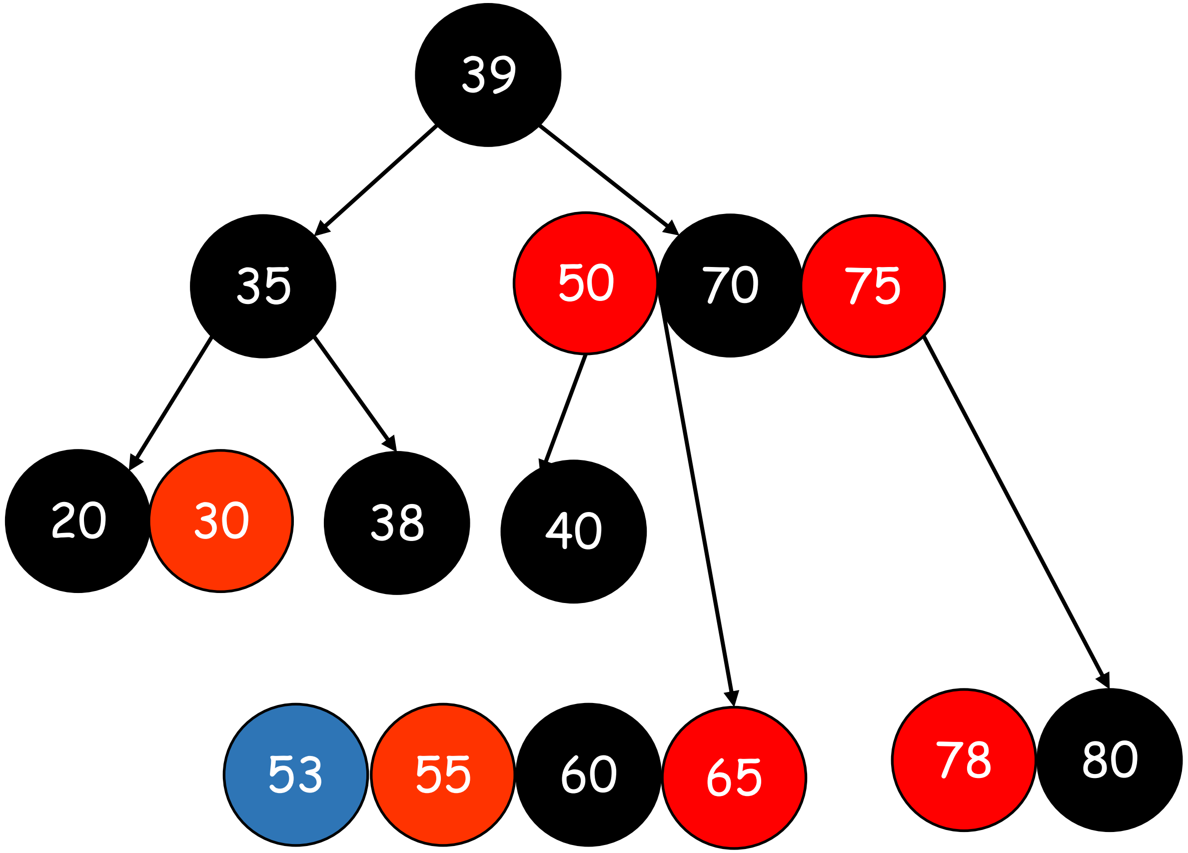
Insert(55)



Insert(53) ?



Insert(53)



Problem of the day

Given an array of integers, and x, find how many pairs of elements of the array sum to x, i.e., how many indexes $i \neq j$ are there with $arr[i] + arr[j] = x$? Array is not sorted, and may contain duplicate elements. Solve the problem using balanced BST with RT = $O(n \log n)$. For example, If arr = {3,3,4,5,3,5,4} then howMany(A,8) returns 7.