B Tree

Module 2

Motivation for B-Trees

- Index structures for large datasets cannot be stored in main memory
- Storing it on disk requires different approach to efficiency.
- M-way trees are mainly used in external searching, i.e. in situations where data is to be retreived from secondary storage like disk files.
- Assuming that a disk spins at 3600 RPM, one revolution occurs in 1/60 of a second, or 16.7ms
- Crudely speaking, one disk access takes about the same time as 200,000 instructions

Motivation (cont.)

- Assume that we use an AVL tree to store about 20 million records
- We end up with a **very** deep binary tree with lots of different disk accesses; $\log_2 20{,}000{,}000$ is about 24, so this takes about 0.2 seconds
- We know we can't improve on the log *n* lower bound on search for a binary tree
- But, the solution is to use more branches and thus reduce the height of the tree!
  - As branching increases, depth decreases

Motivation (cont.)

- As we know, access time for secondary storage is much higher than that of primary storage, and thus our aim is to minimize the number of file accesses.
- To do so, the height of the tree is reduced by forming an m-way tree. Lesser the height, more efficient is the external search.
- Although the height of an M-way tree is less, there is a chance for further reduction which can be achieved by balancing the tree.
- That's where B tree comes in.
- A B Tree (height Balanced m-way search Tree) is a special type of M-way tree which balances itself.

Definition of a B-tree

- A B-tree of order m is an m-way tree (i.e., a tree where each node may have up to m children) in which:
  1. the number of keys in each non-leaf node is one less than the number of its children and these keys partition the keys in the children in the fashion of a search tree
  2. all leaves are on the same level
  3. all non-leaf nodes (except the root) have at least $\lceil m/2 \rceil$ children
  4. the root is either a leaf node, or it has from two to m children
  5. a leaf node contains no more than $m - 1$ keys

Definition of a B-tree

- B tree of order m have following properties.
- Every node have maximum m children.
- Minimum children

  leaf=0

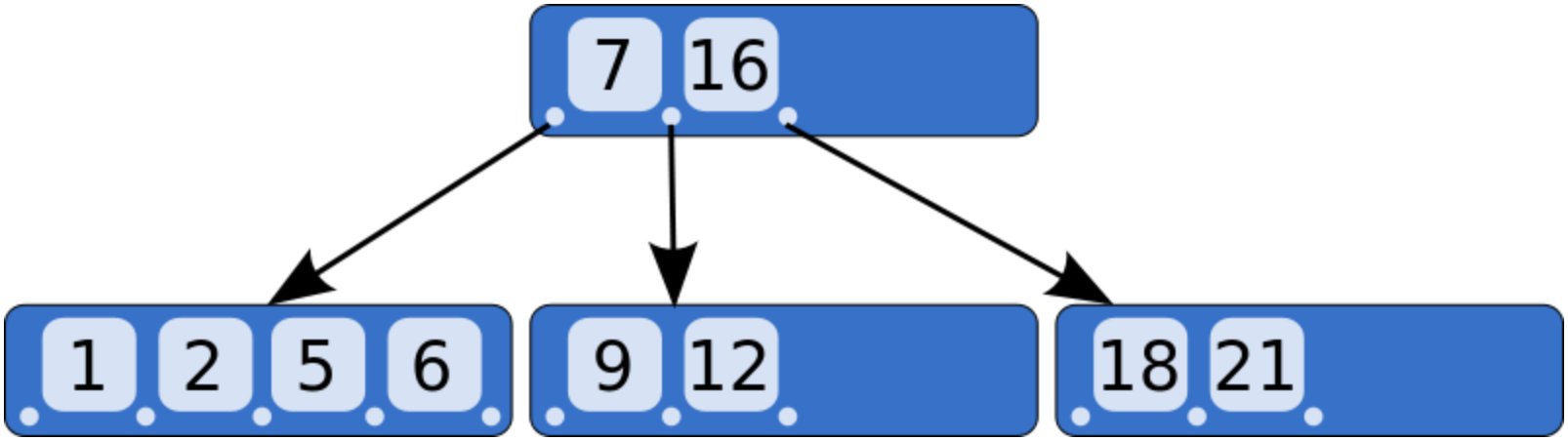
  root=2

  internal node=(m/2)
- Every node have maximum m-1 keys
- Minimum Keys

  root=1

  all other nodes=(m/2)-1

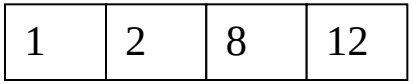
B-Tree: Example



Constructing a B-tree

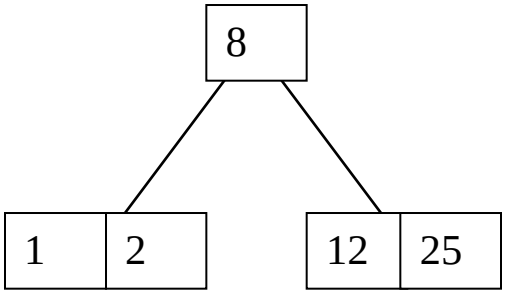
- Suppose we start with an empty B-tree and keys arrive in the following order:

  1 12 8 2 25 6 14 28 17 7 52 16 48 68 3 26 29 53 55 45
- We want to construct a B-tree of order 5
- The first four items go into the root:

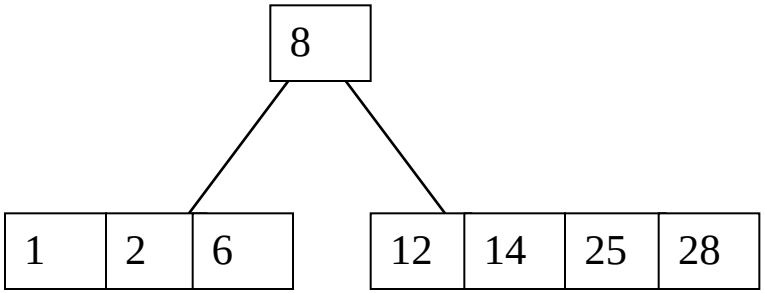


- To put the fifth item in the root would violate condition 5
- Therefore, when 25 arrives, pick the middle key to make a new root

Constructing a B-tree (contd.)

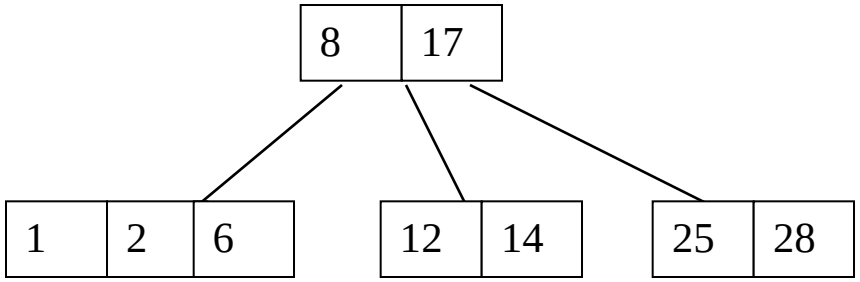


6, 14, 28 get added to the leaf nodes:

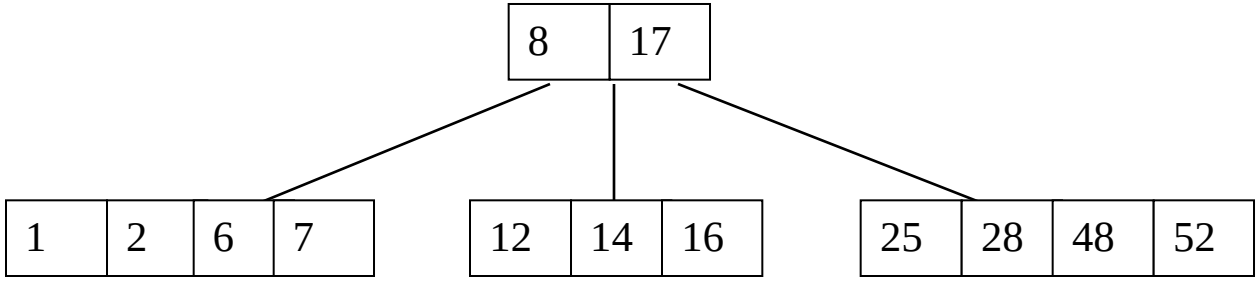


Constructing a B-tree (contd.)

Adding 17 to the right leaf node would over-fill it, so we take the middle key, promote it (to the root) and split the leaf

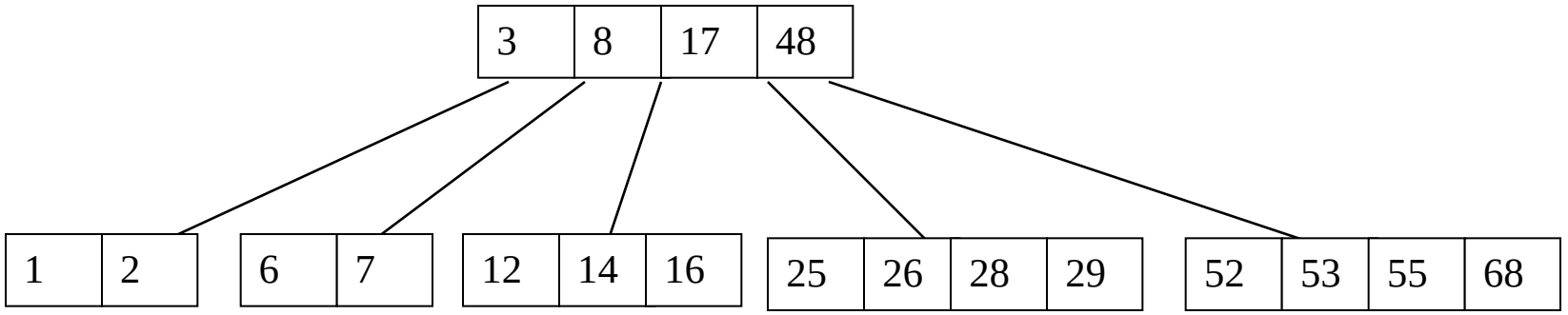


7, 52, 16, 48 get added to the leaf nodes

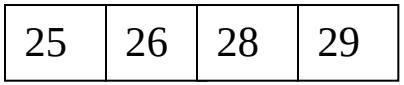


Constructing a B-tree (contd.)

Adding 68 causes us to split the right most leaf, promoting 48 to the root, and adding 3 causes us to split the left most leaf, promoting 3 to the root; 26, 29, 53, 55 then go into the leaves

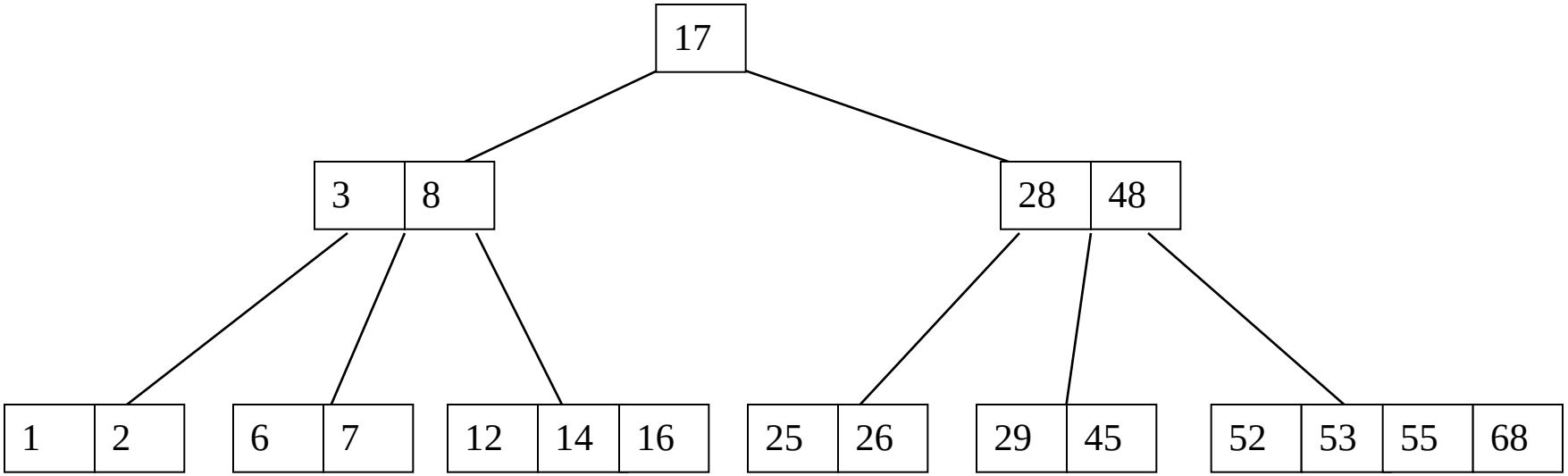


Adding 45 causes a split of



and promoting 28 to the root then causes the root to split

Constructing a B-tree (contd.)



Inserting into a B-Tree

- Attempt to insert the new key into a leaf
- If this would result in that leaf becoming too big, split the leaf into two, promoting the middle key to the leaf's parent
- If this would result in the parent becoming too big, split the parent into two, promoting the middle key
- This strategy might have to be repeated all the way to the top
- If necessary, the root is split in two and the middle key is promoted to a new root, making the tree one level higher

Exercise in Inserting a B-Tree

- Insert the following keys to a 5-way B-tree:
- 3, 7, 9, 23, 45, 1, 5, 14, 25, 24, 13, 11, 8, 19, 4, 31, 35, 56

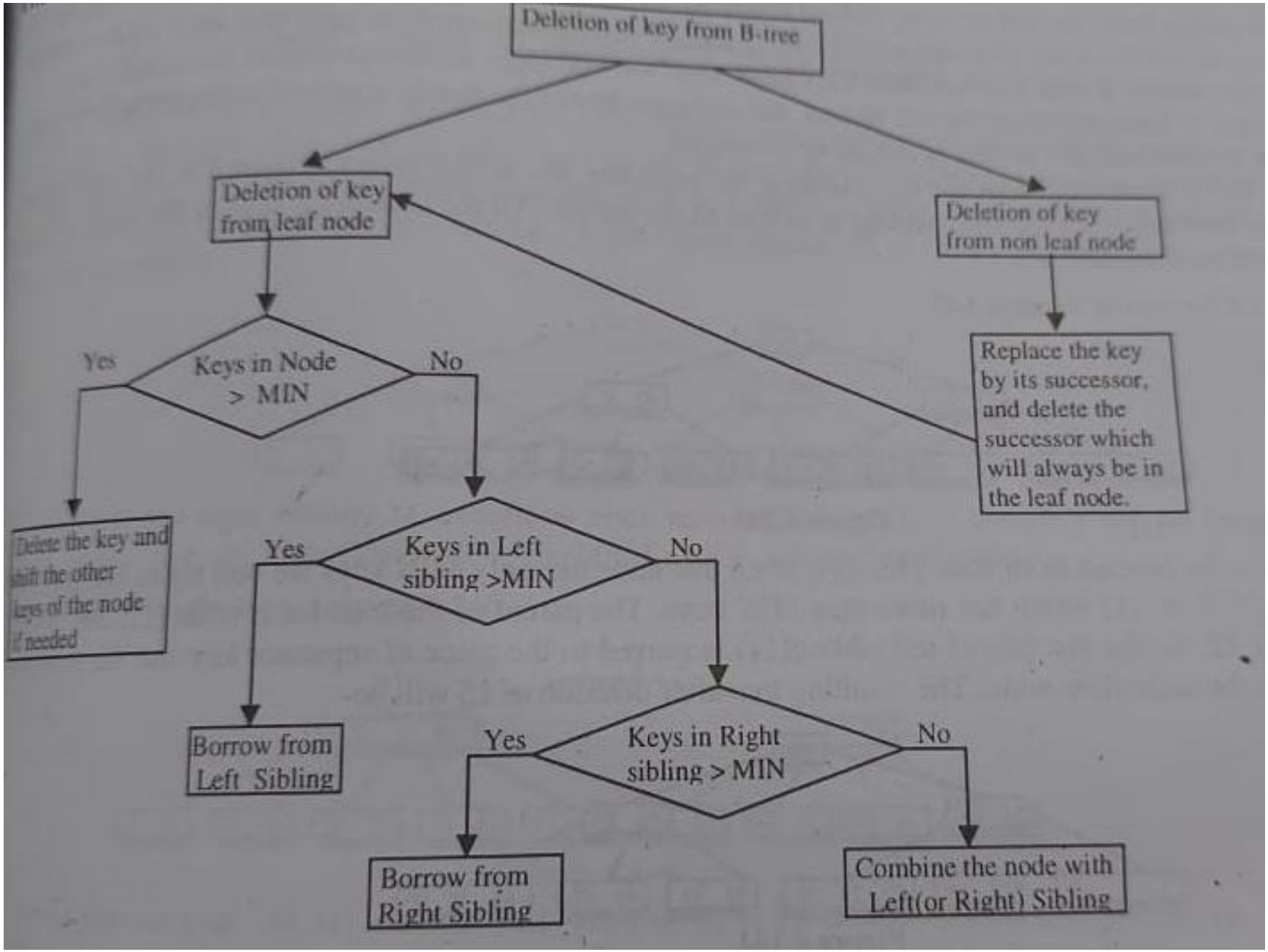
Removal from a B-tree

- During insertion, the key always goes *into* a *leaf*. For deletion we wish to remove *from* a leaf. There are three possible ways we can do this:
  - 1 - If the key is already in a leaf node, and removing it doesn't cause that leaf node to have too few keys, then simply remove the key to be deleted.
  - 2 - If the key is *not* in a leaf then it is guaranteed (by the nature of a B-tree) that its predecessor or successor will be in a leaf -- in this case we can delete the key and promote the predecessor or successor key to the non-leaf deleted key's position.

Removal from a B-tree (2)

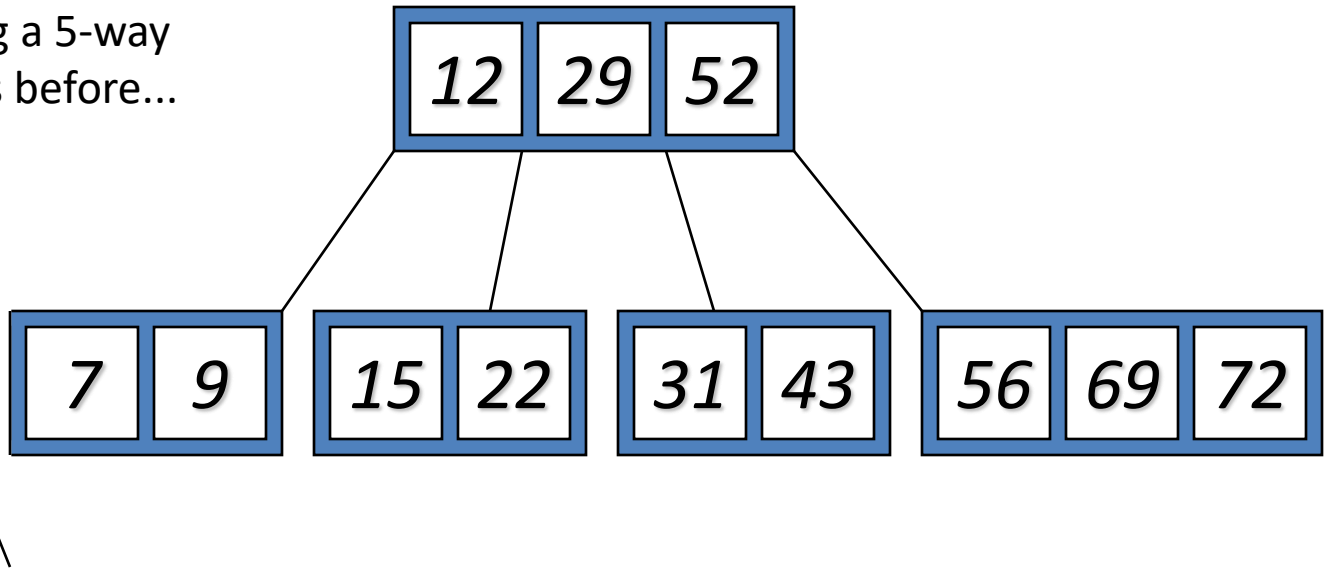
- If (1) or (2) lead to a leaf node containing less than the minimum number of keys then we have to look at the siblings immediately adjacent to the leaf in question:
  - 3: if one of them has more than the min. number of keys then we can promote one of its keys to the parent and take the parent key into our lacking leaf
  - 4: if neither of them has more than the min. number of keys then the lacking leaf and one of its neighbours can be combined with their shared parent (the opposite of promoting a key) and the new leaf will have the correct number of keys; if this step leave the parent with too few keys then we repeat the process up to the root itself, if required

Deletion Flowchart



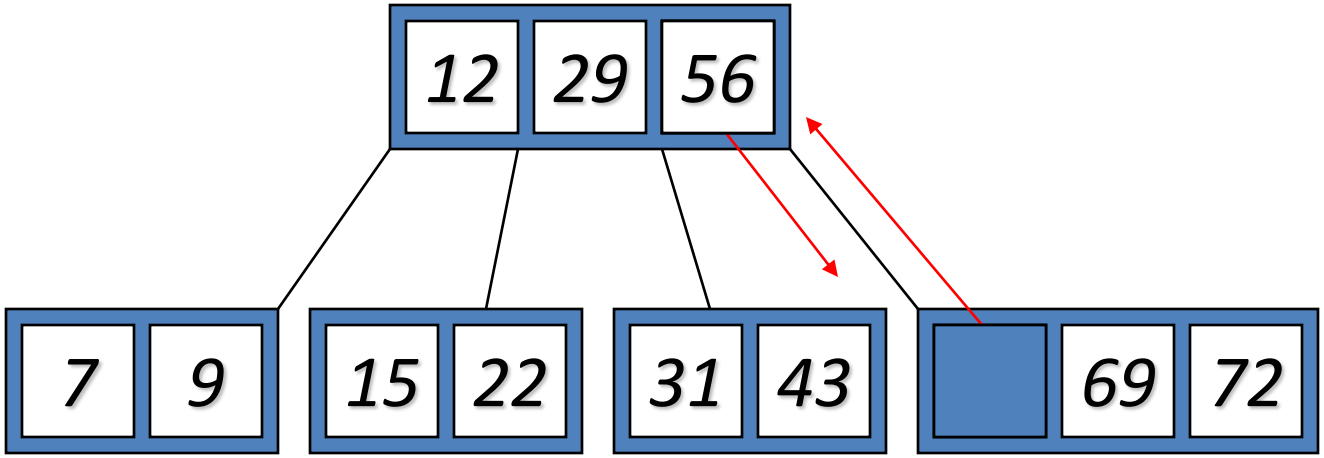
Type #1: Simple leaf deletion

Assuming a 5-way B-Tree, as before...

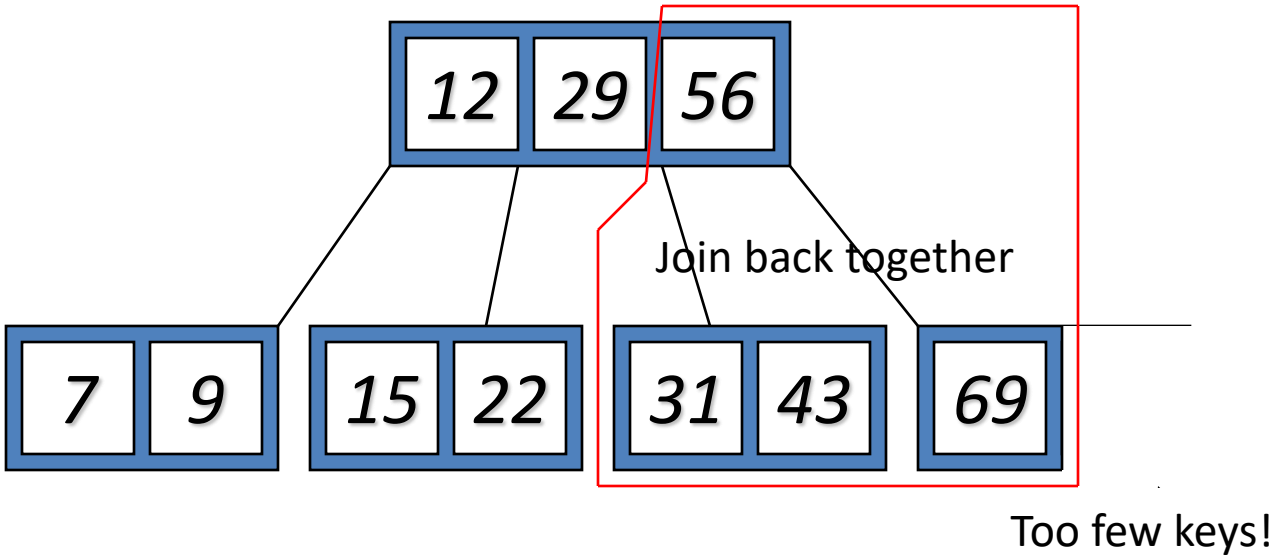


Delete 2: Since there are enough keys in the node, just delete it

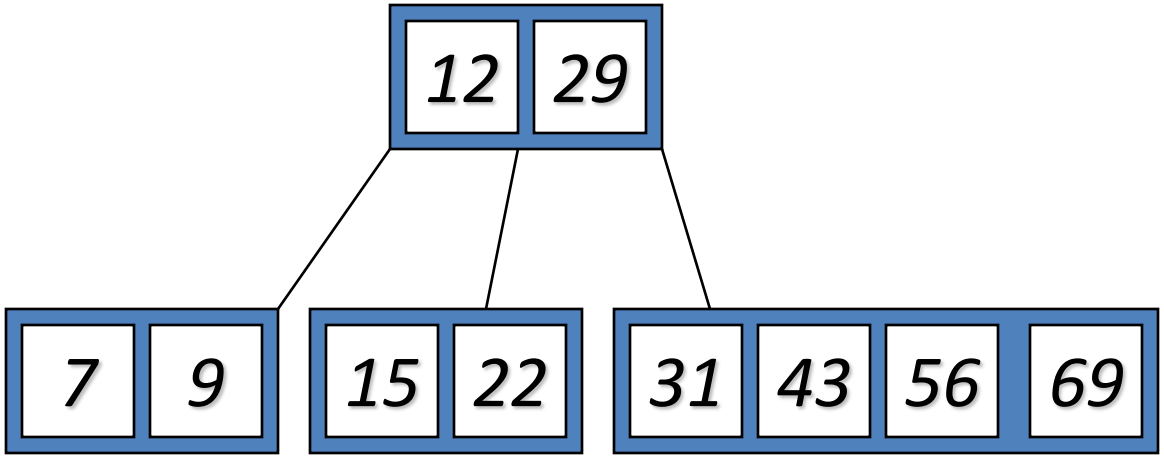
Type #2: Simple non-leaf deletion



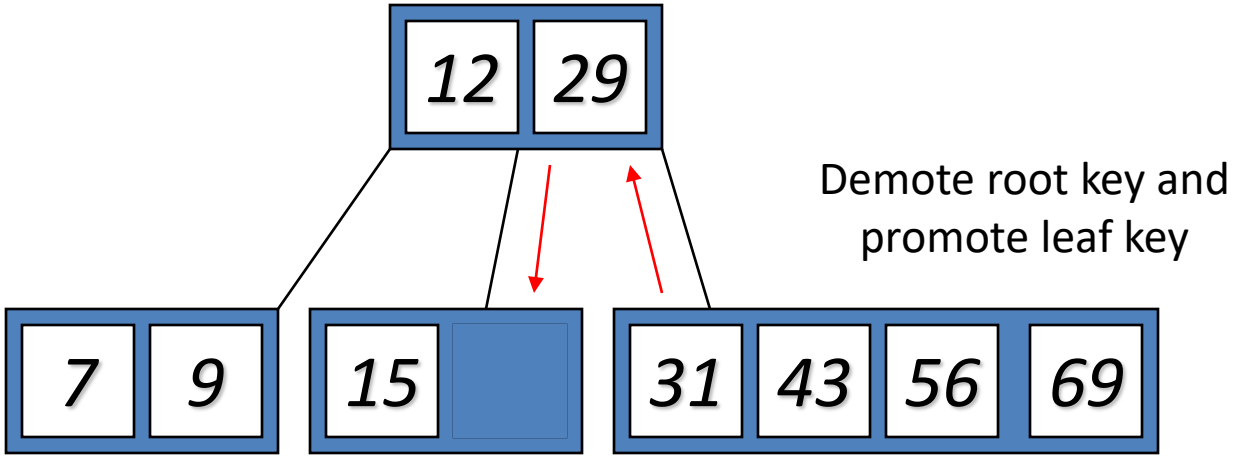
Type #4: Too few keys in node and its siblings



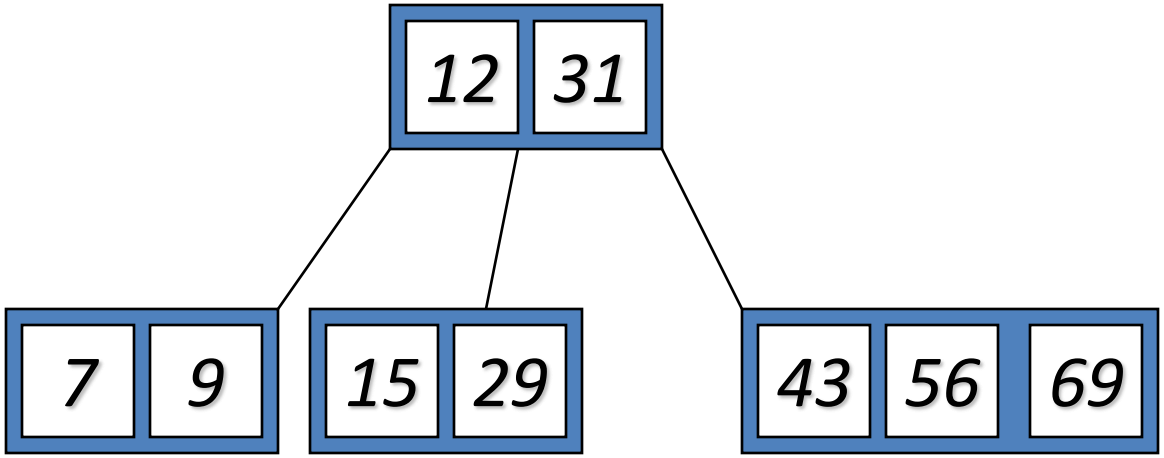
Type #4: Too few keys in node & its siblings



Type #3: Enough siblings



Type #3: Enough siblings



Exercise in Removal from a B-Tree

- Given 5-way B-tree created by these data:

  3, 7, 9, 23, 45, 1, 5, 14, 25, 24, 13, 11, 8, 19, 4, 31, 35, 56
- Add these further keys: 2, 6,12
- Delete these keys: 4, 5, 7, 3, 14

Analysis of B-Trees

- Max. no. of items in a B-tree of order m and height h:

  root $m - 1$

  level 1 $m(m - 1)$

  level 2 $m^2(m - 1)$

  . . .

  level h $m^h(m - 1)$

- So, the total number of items is

  $(1 + m + m^2 + m^3 + \ldots + m^h)(m - 1) =$

  $[(m^{h+1} - 1)/ (m - 1)] (m - 1) = \mathbf{m^{h+1} - 1}$

- When $m = 5$ and $h = 2$ this gives $5^3 - 1 = 124$

Comparing Trees

- Binary trees
  - Can become *unbalanced* and *lose* their good time complexity (big O)
  - AVL trees are strict binary trees that *overcome the balance problem*
  - Heaps remain balanced but only *prioritise* (not order) the keys
- Multi-way trees
  - B-Trees can be *m*-way, they can have any (odd) number of children
  - One B-Tree, the 2-3 (or 3-way) B-Tree, *approximates* a permanently balanced binary tree, exchanging the AVL tree's balancing operations for insertion and (more complex) deletion operations