B-Trees

November 13, 2023

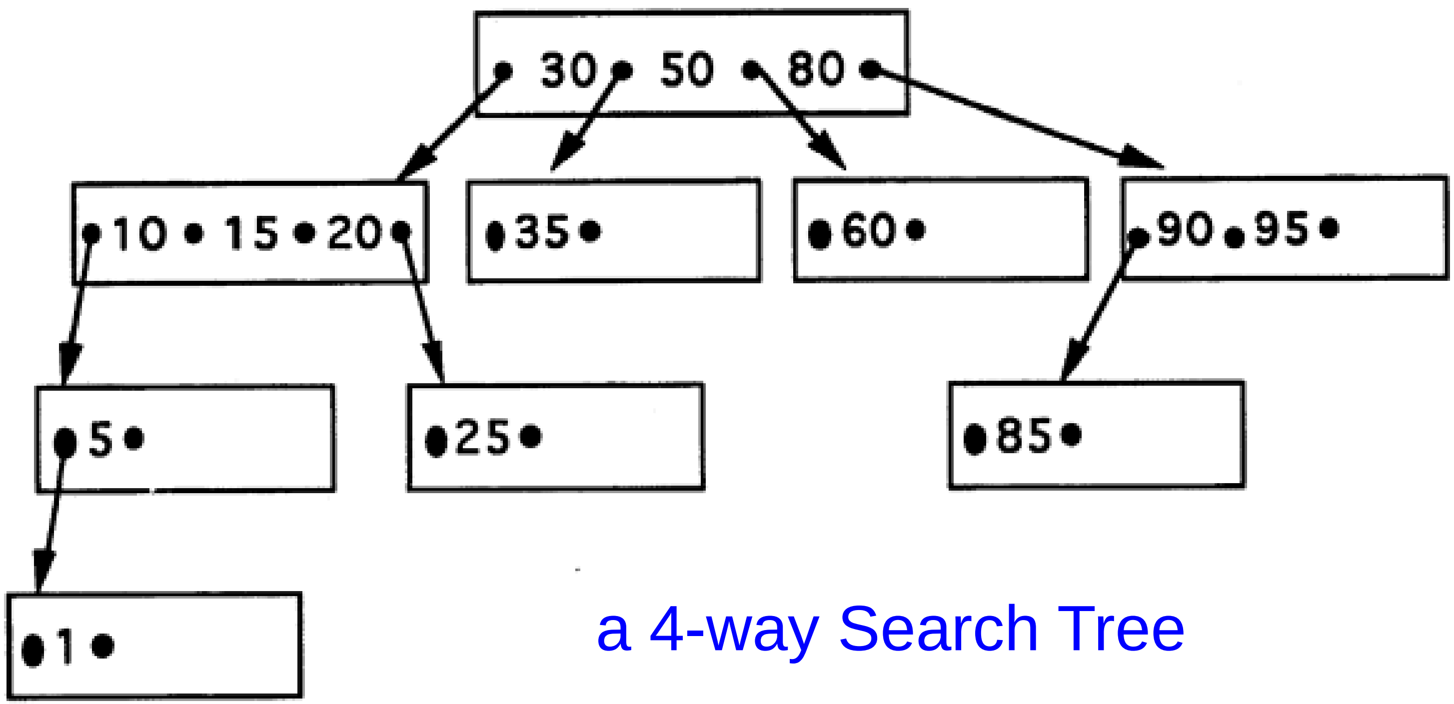
External Data Storage

- Difference in access times for silicon memory (nsec) Vs. disk (msec) is enormous
- Balanced binary trees are nice structures, if data is stored in main memory
- Idea: reduce disk accesses
  - Control the access to disks by storing “consecutive” nodes on the same page and reduce the page access
- Information is divided into a no. of equal-sized pages of bits that appear consecutively within cylinders, and each disk read/write is of one/more entire pages

External Data Storage

- For external storage, height of the tree should be minimized, to minimize the disk access
- To achieve the minimization of height, each node has as many subtrees as possible
- An m-path search tree with the goal to minimize the accesses while retrieving a key from a file

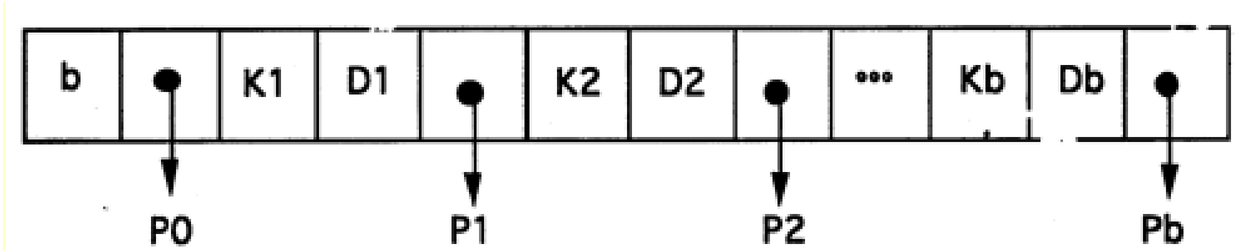
m-way Search Tree



a 4-way Search Tree

- A generalization of binary search trees
- An m-way tree is a tree in which all the nodes have a degree $\leq$ m

m-way Search tree: node structure



- K_i and D_i are the key/data pairs, $1 \leq i \leq b$
- b is the no. of keys currently stored in the node
- P_i are the pointers to subtrees of the node, $0 \leq i \leq b$
- The keys are stored in ascending order within a node: $K_1 \leq K_2 \leq K_3 \leq \ldots\ldots K_i \leq K_{i+1}$ for $1 \leq i \leq b$
- All the keys stored in the subtree of P_i are smaller than the key K_{i+1} for $0 \leq i \leq b\text{-}1$
- All the keys stored in the subtree of P_b are bigger than the key K_b
- The subtrees P_i, $0 \leq i \leq b$ are also m-way trees

m-way Search tree – Imp Points

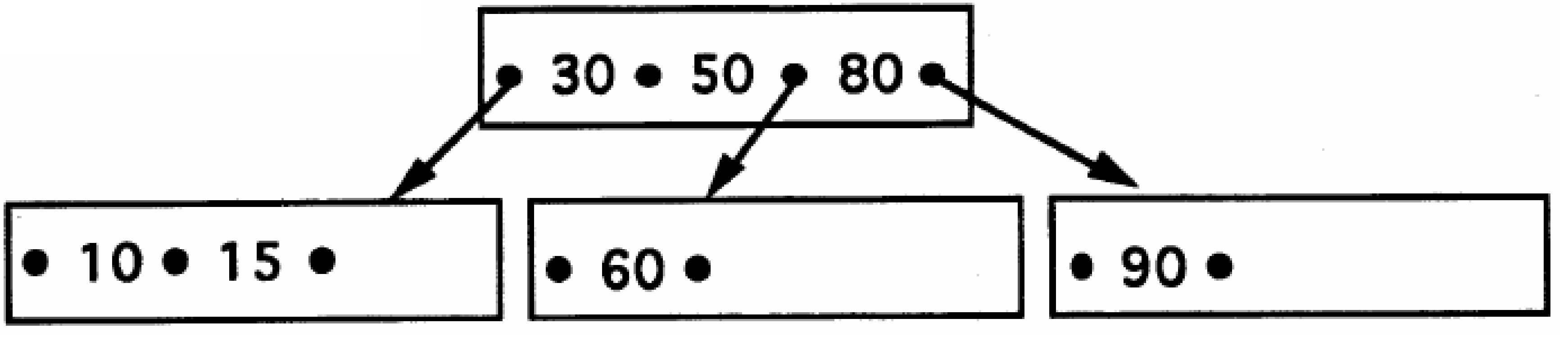
- Max. value of b = m-1, if tree is an m-way tree
- m – order of the tree
- Degree of each node can reach a maximum of m (each node has at most m child nodes)
- The entries Di are either data (saved together with a key) or pointers to satellite data, this means the m-way tree is an index of saved data
- Limits for the height for an m-way tree with n keys

$$\log_m(n+1) \le h \le n$$

m-way Search tree - Insertion

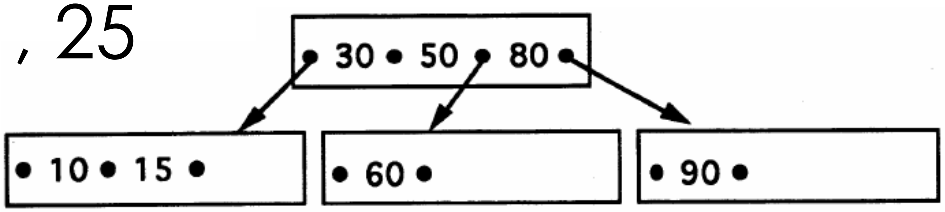
- Insertion in a 4-way search tree
- Insert sequence: 30, 50, 80

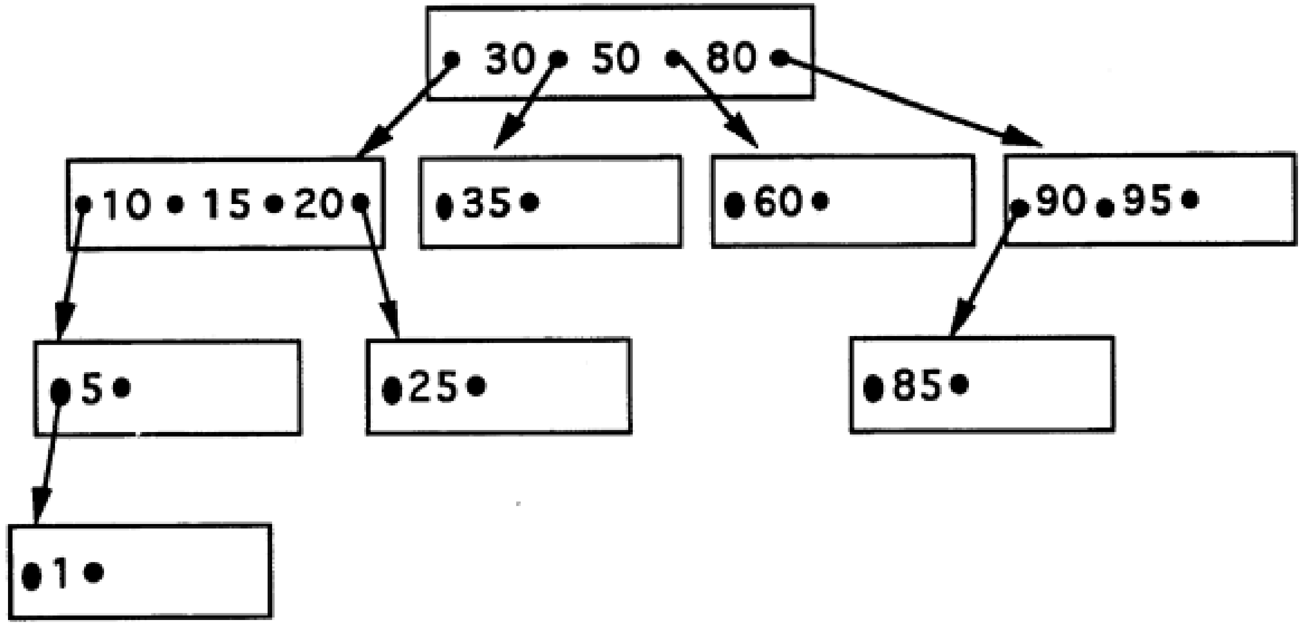
- Then 10, 15, 60, 90



m-way Search tree - Insertion

- Then insert 20, 35, 5, 95, 1, 25





- Keys in inter nodes are both keys and splitters
- Searching in a node: both possibilities: sequential as well as binary search

m-way Search tree - Drawbacks

Let's construct a 3-way search tree:

List A: 10, 15, 20, 25, 30, 35, 40, 45

List B: 20, 35, 40, 10, 15, 25, 30, 45

Feel the difference in two trees constructed with the same keys.

- The tree is not balanced
- Leaves located on different tree levels
- No balancing algorithm on update operations
- Bad storage space usage, degenerates into linked lists

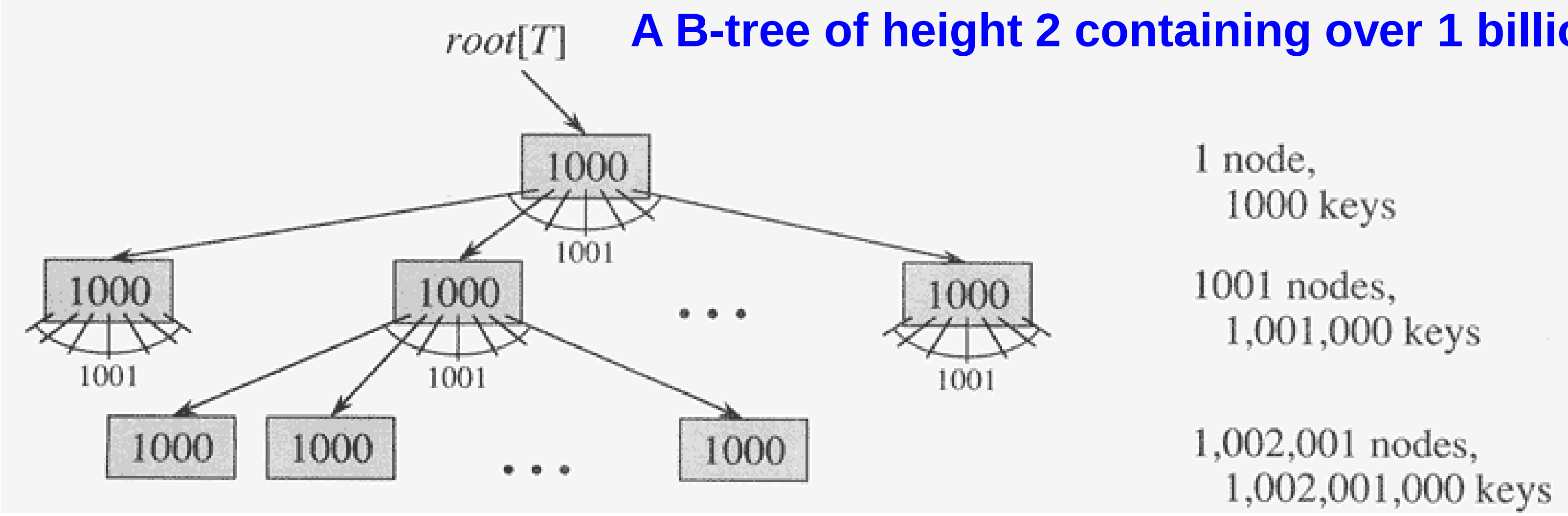
B-trees (balanced m-way trees)

- Proposed by R. Bayer and E. M. McCreigh, 1972
- Balanced search trees designed to work well on disks or other direct access secondary storage devices
- Quite large branching factor, usually determined by characteristics of the disk unit used
- Many database systems use B-trees, or variants of B-trees, to store information

B-trees

- A B-tree node is usually as large as a whole disk page **(This limits the number of children a B-tree node can have)**
- Branching factors: 50 to 2000 are most common
- A large branching factor reduces the height of the tree and the number of disk accesses required to find any key

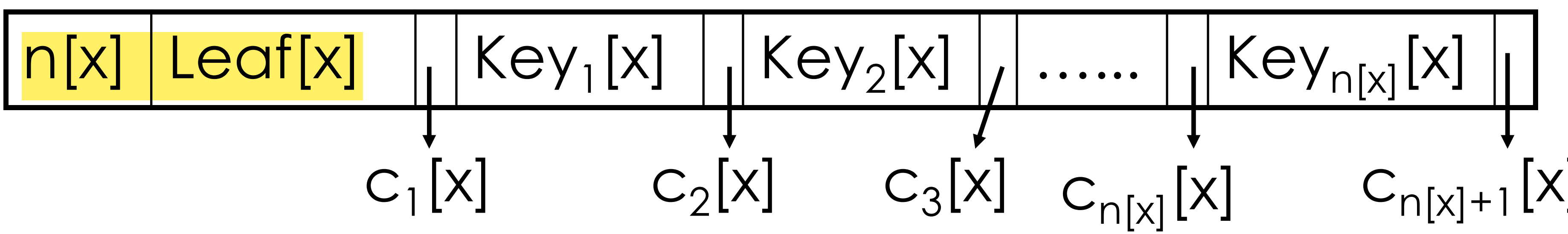
A B-tree of height 2 containing over 1 billion keys



B-trees - Definition

A B-tree T is a rooted tree with

- Structure of a node: same as a node in m-way tree + one Boolean value (T for a leaf; F for an internal node)
- Keys and pointers to subtrees: same fashion



Structure of a node x

- All leaves have the same depth → tree's height h

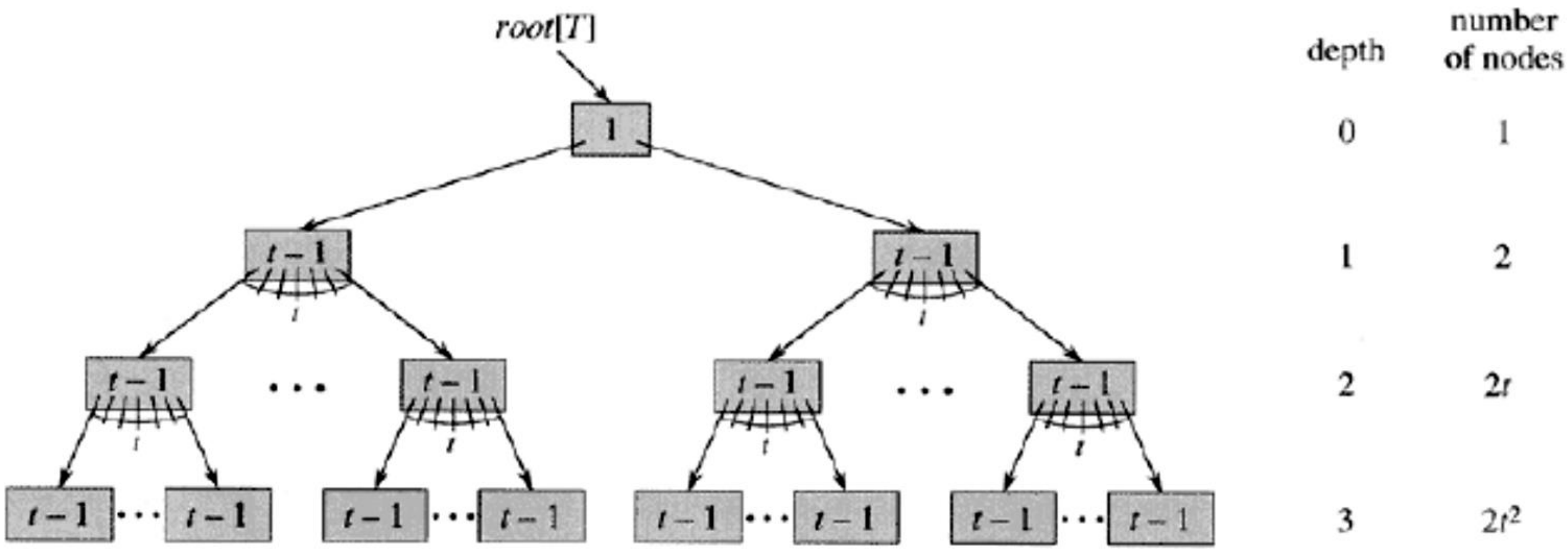
B-trees - Definition

- Lower and upper bounds on the no. of keys a node can have
- Bounds are expressed in terms of a fixed integer t ≥ 2, which is minimum degree of the B-tree
  - ❖ Every node other than root must have at least t-1 keys
    - ➢ If the tree is nonempty, the root must have at least one key
  - ❖ Every node can contain at most 2t-1 keys
- Remember a B-tree is atleast half full
- Simplest B-tree: t = 2
  - ❖ Every internal node has either 2, 3, or 4 children: 2-3-4 tree / 2-4 tree

In other texts: min keys t, max keys 2t

B-trees – worst case height

- The no. of disk accesses required for most operations on a B-tree ∝ height of the B-tree



- The no. n of keys satisfies the inequality:

$n \geq 1 + 2(t-1) + 2t(t-1) + 2t^2(t-1) + \ldots\ldots\, 2t^{h-1}(t-1)$

$n \geq 2t^h - 1 \quad \rightarrow \quad h \leq \log_t((n+1)/2)$

B-trees – search

- n[x]+1- way branching decision
- Procedure takes as input a pointer to the root node x of a subtree and a key k to be searched for in that subtree
- The no. of disk pages accessed is $O(h) = O(\log_t n)$

B-trees – inserting a key

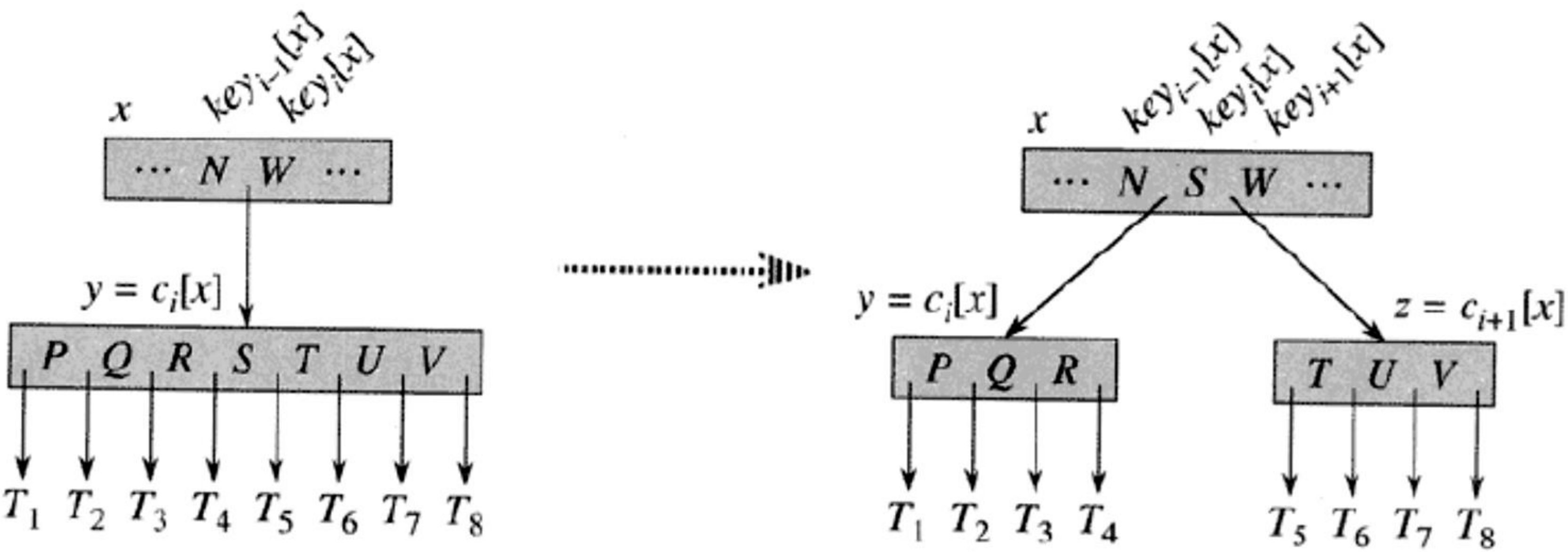
- Complicated than inserting a key into a BST
- As with BST, search for the leaf position at which to insert the new key
- We can not simply create a new leaf node and insert it

B-trees – inserting a key

- Insert the new key into an existing leaf node
  - But we can not insert the new key into a leaf node that is full
- Split operation:
  - Split a full node y around its median key $key_t[y]$ node into 2 nodes having (t-1) keys each
  - The median key moves up into y's parent to identify the dividing point b/w the two new trees
  - If y's parent is also full, it must be split before the new key can be inserted
  - Need to split full nodes can propagate all the way up the tree

B-trees – inserting a key (Cont..)

- Splitting a node with t=4

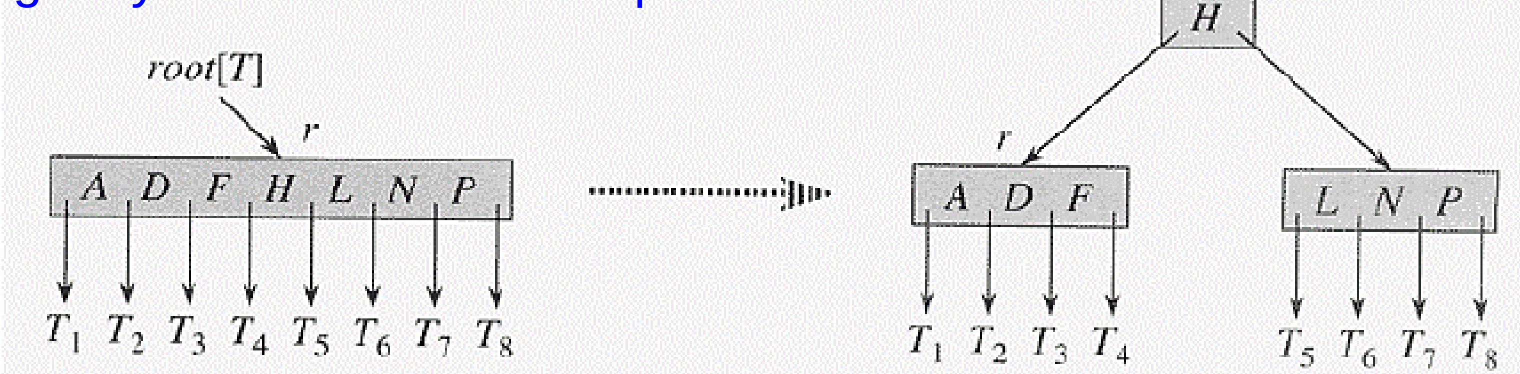


Node y splits into two nodes, y and z. The median key S of Y is moved up into y's parent.

B-trees – inserting a key (Cont..)

- Inserting a key in a single pass down the tree
- As you travel down the tree searching for the position where the new key belongs, split each full node along the way (including the leaf itself)
- So, whenever you split a full node y, you are assured that its parent is not full

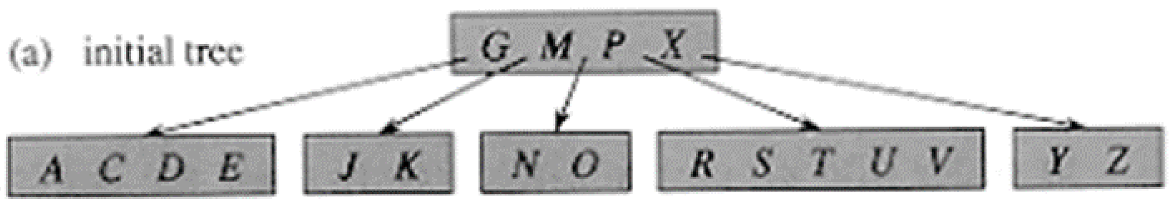
Splitting the root with t = 4. B-tree grows in height by 1 when the root is split.



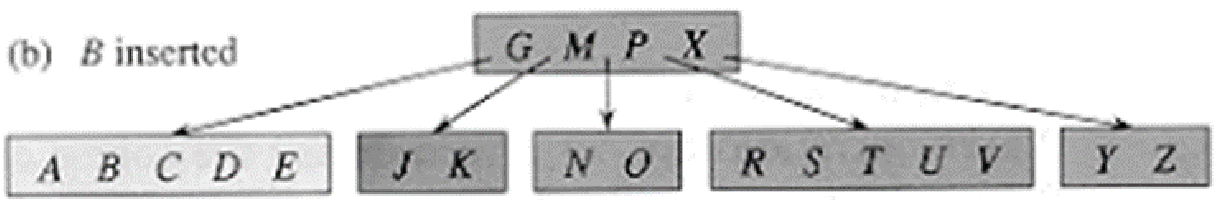
Unlike a BST, a B-tree increases in height at the top instead of at the bottom

B-trees – insertion example

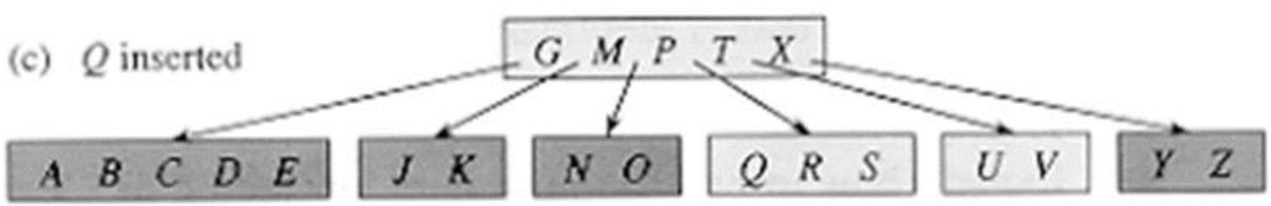
Here t=3. A node can have at most 5 keys.



Insert B:

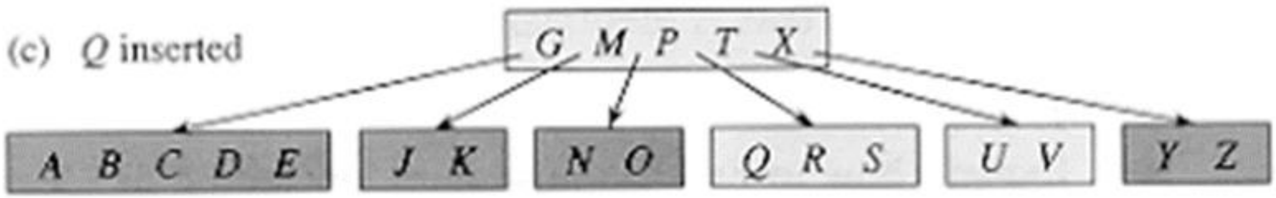


Insert Q:



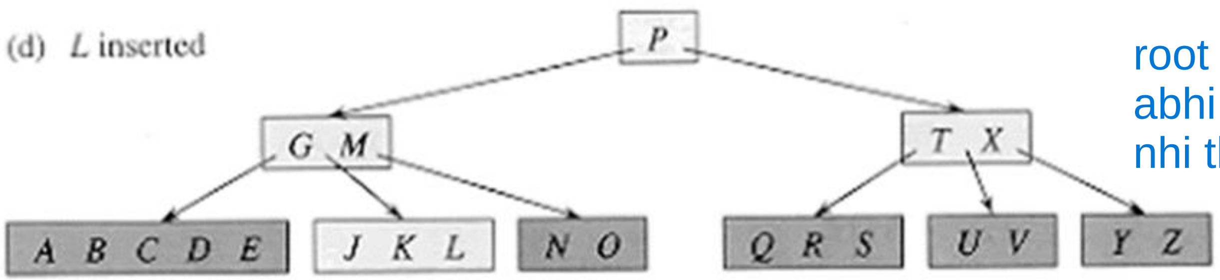
The node RSTUV is split into two nodes RS and UV, the key T is moved up to the root, and Q is inserted in the leftmost of the two halves

B-trees – insertion example (Cont..)



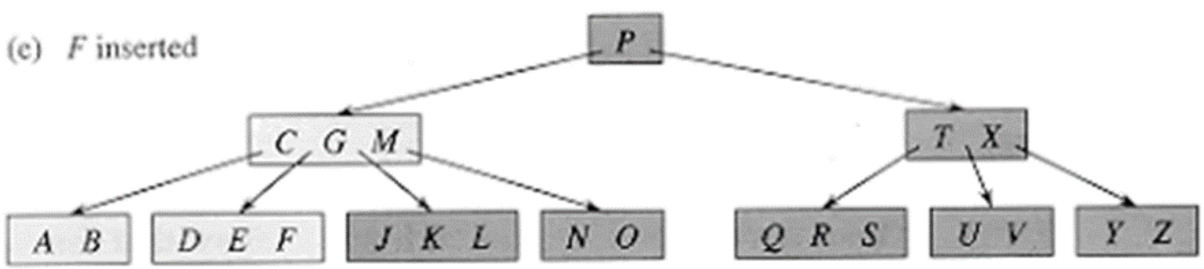
Next insertion in tree will result in split of root as it is full now

Insert L:



root ko kiu split kra abhi toh need bhi nhi thi

Insert F:



The node ABCDE is split before F is inserted into the rightmost of the two halves (the DE node)

B-trees – deleting a key

- More complicated
  - ❖ A key may be deleted from leaf as well as from internal node
  - ❖ Deletion from an internal node requires that the node's children be rearranged
- Need to ensure that a node does not get too small during deletion
  - ❖ Except that the root is allowed to have fewer than the minimum no. of keys, though it is not allowed to have more than the maximum no. of keys

B-trees – deleting a key

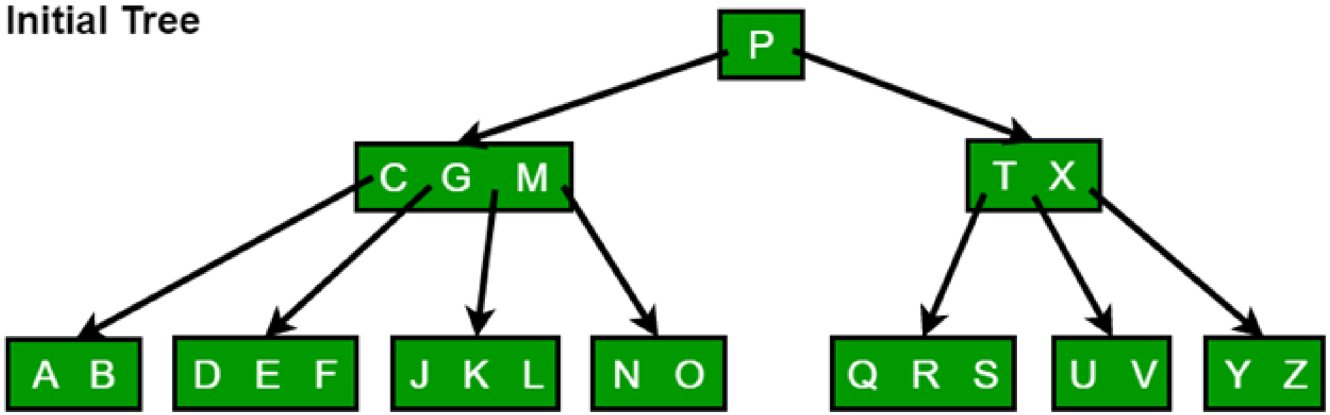
- Simple approach: Take appropriate action if a node (other than the root) along the path to where the key is to be deleted has the min. no. of keys

- Delete the key k from the subtree rooted at x

B-trees – deleting a key (Cont..)

- Case 1: key k is in node x and x is a leaf.
  Delete the key k from x.

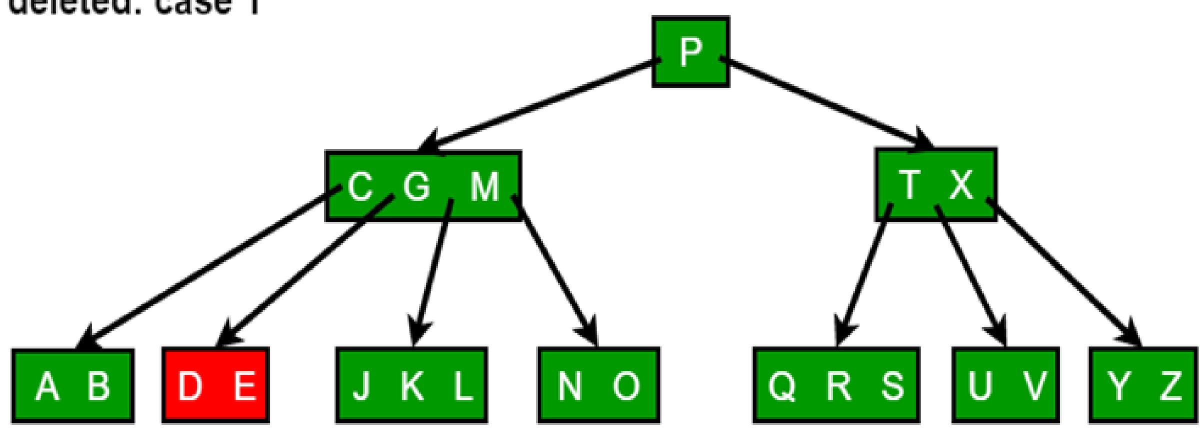
(a) Initial Tree



t=3 for this tree, so a node (other than the root) cannot have fewer than 2 keys

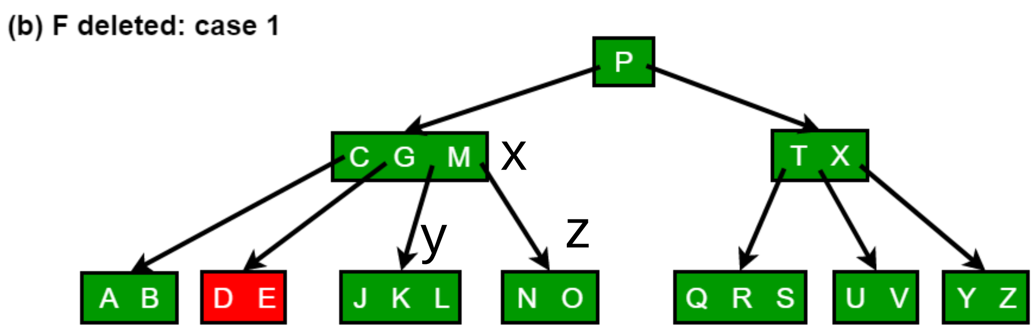
Delete F:

(b) F deleted: case 1

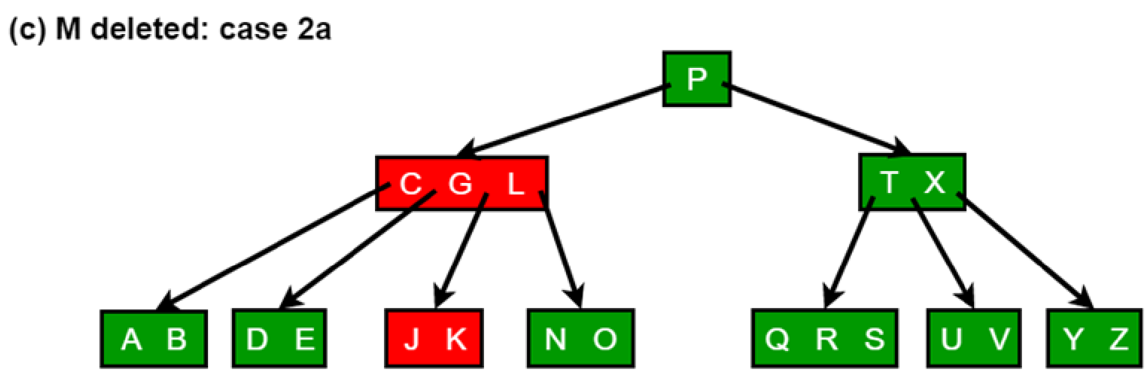


B-trees – deleting a key (Cont..)

- Case 2: k is in node x and x is an internal node.
  a. If the child y that precedes k in node x has at least t keys, then find the predecessor k' of k in the subtree rooted at y. Recursively delete k', and replace k by k' in x
  (finding k' and deleting it can be performed in a single downward pass)



Delete M:



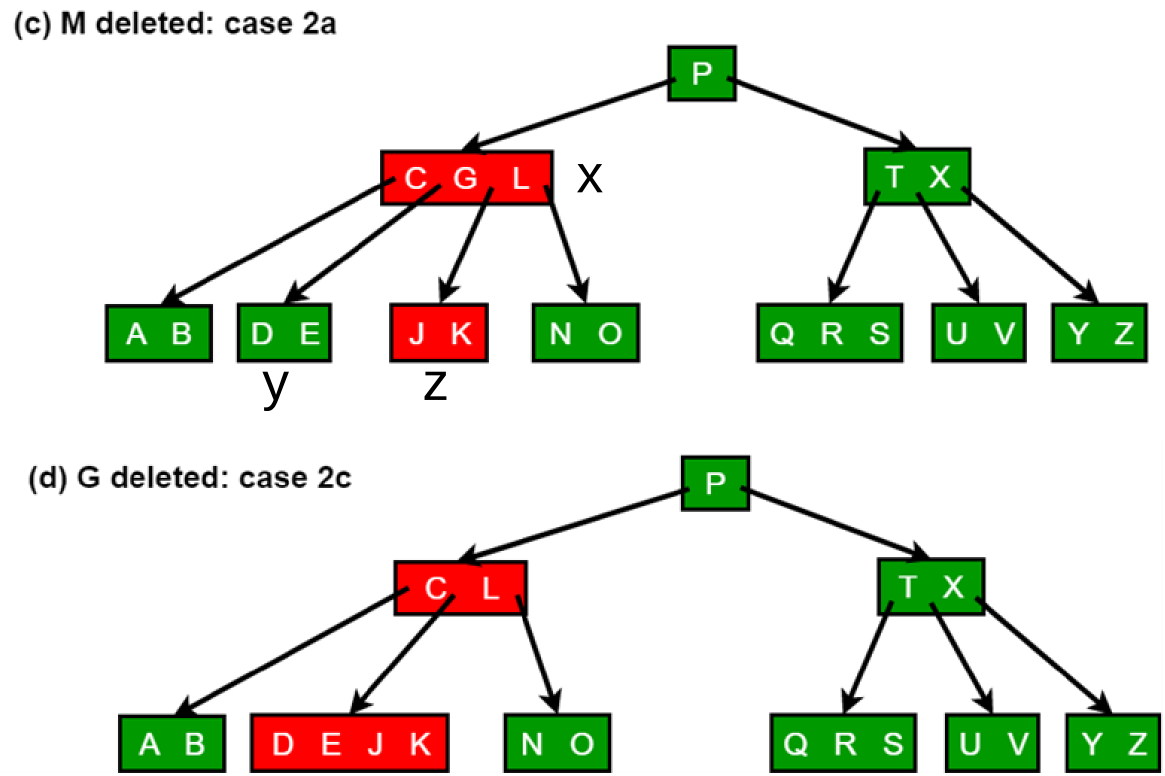
B-trees – deleting a key (Cont..)

- Case 2: k is in node x and x is an internal node.
  b. Symmetrically, if the child z that follows k in node x has at least t keys, then find the successor k' of k in the subtree rooted at z. Recursively delete k', and replace k by k' in x. (finding k' and deleting it can be performed in a single downward pass)

B-trees – deleting a key (Cont..)

- Case 2: k is in node x and x is an internal node.
  - c. If both y and z have only (t-1) keys, merge k and all of z into y, so that x loses both k and the pointer to z, and y now contains (2t-1) keys. Then, free z and recursively delete k from y.

Delete G:



B-trees – deleting a key (Cont..)

- Case 3: key k is not present in internal node x.

Determine the root $c_i[x]$ of appropriate subtree that must contain k, if k is in the tree at all.

If $c_i[x]$ has only (t-1) keys, execute step 3a or 3b as necessary to guarantee that we descend to a node containing at least t keys.

Then, finish by recursing on the appropriate child of x.

B-trees – deleting a key (Cont..)

- Case 3a: if $c_i[x]$ has only (t-1) keys, but has an immediate sibling with at least t keys,

Give $c_i[x]$ an extra key by moving a key from x down into $c_i[x]$,

Move a key from $c_i[x]$'s immediate left or right sibling up into x,

Move the appropriate child pointer from the sibling into $c_i[x]$.

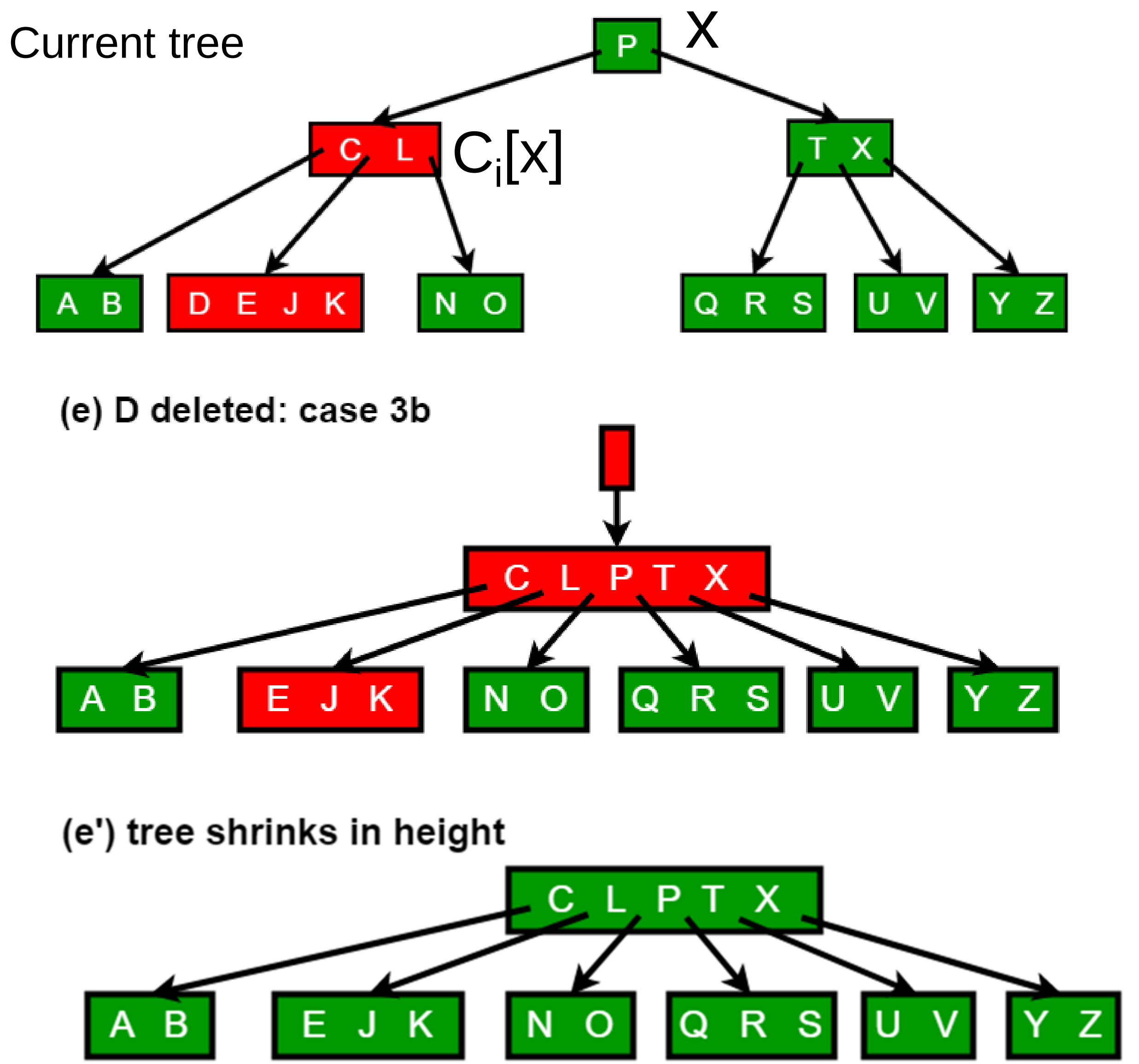
B-trees – deleting a key (Cont..)

- Case 3b: if $c_i[x]$ and both of its immediate siblings have only (t-1) keys,

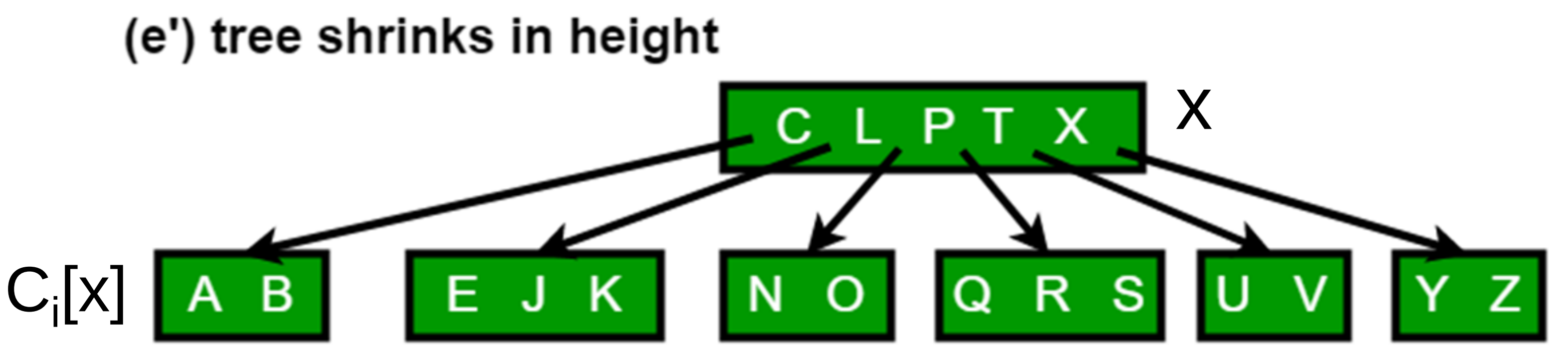
  Merge $c_i[x]$ with one sibling, which involves moving a key from x down into the new merged node to become the median key for that node.

B-trees – deleting a key (Cont..)

Delete D:



B-trees – deleting a key (Cont..)



Delete B: