Search in B-Tree
and
Delete in B-Tree

Sinem Özden

150150202

Search in B-Tree

Search in B-Tree

- Search for B-tree is very similar to search in binary search tree. The difference between B-tree and binary search tree is binary tree has two branches at most but B-tree can have multiple branches.
- We can see the this difference when we make a decision to which branch to go.
- We can also say B-tree search is a generalized form of tree search for binary trees.

Search in B-Tree Pseudo Code

In B-Tree Search Linear search is used.

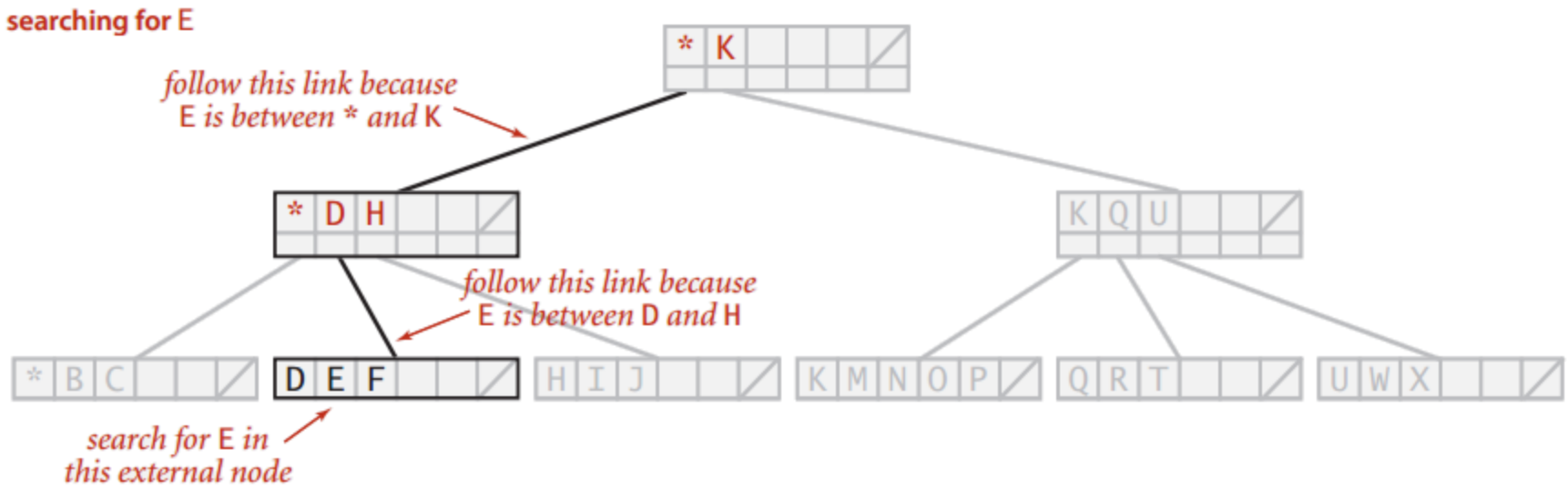
```
B-TREE-SEARCH(x, k)
1  i = 1
2  while i ≤ x.n and k > x.key_i
3       i = i + 1
4  if i ≤ x.n and k == x.key_i
5       return (x, i)
6  elseif x.leaf
7       return NIL
8  else DISK-READ(x.c_i)
9       return B-TREE-SEARCH(x.c_i, k)
```

- Inputs for B-tree search are:
  - the root node x of the sub tree
  - k key to be search in the sub tree
- In the while loop we find the node that closest to the key
- If k equals to key the sub tree x and index i
- If code reached a leaf it returns NIL to state that it couldn’t find the key
- Else it reaches to disk to read/look another sub tree

Big O for B-Tree Search

- For B-Tree search because the data is too big to save in the memory we have to reach the disk to read the data. Big O to accesses from the disk page is $O(h)=O(\log_t n)$.
  - h = height of B-Tree
  - n = number of keys in the tree
  - t = minimum degree(disk block size)
- In lines 1-3 there is a while loop n variables. The n (number of nodes in subtree) is smaller than 2t (disk size). Thus, each node takes at least O(t) time. So the big O is $O(ht) = O(t \log_t n)$.

Search in B-Tree



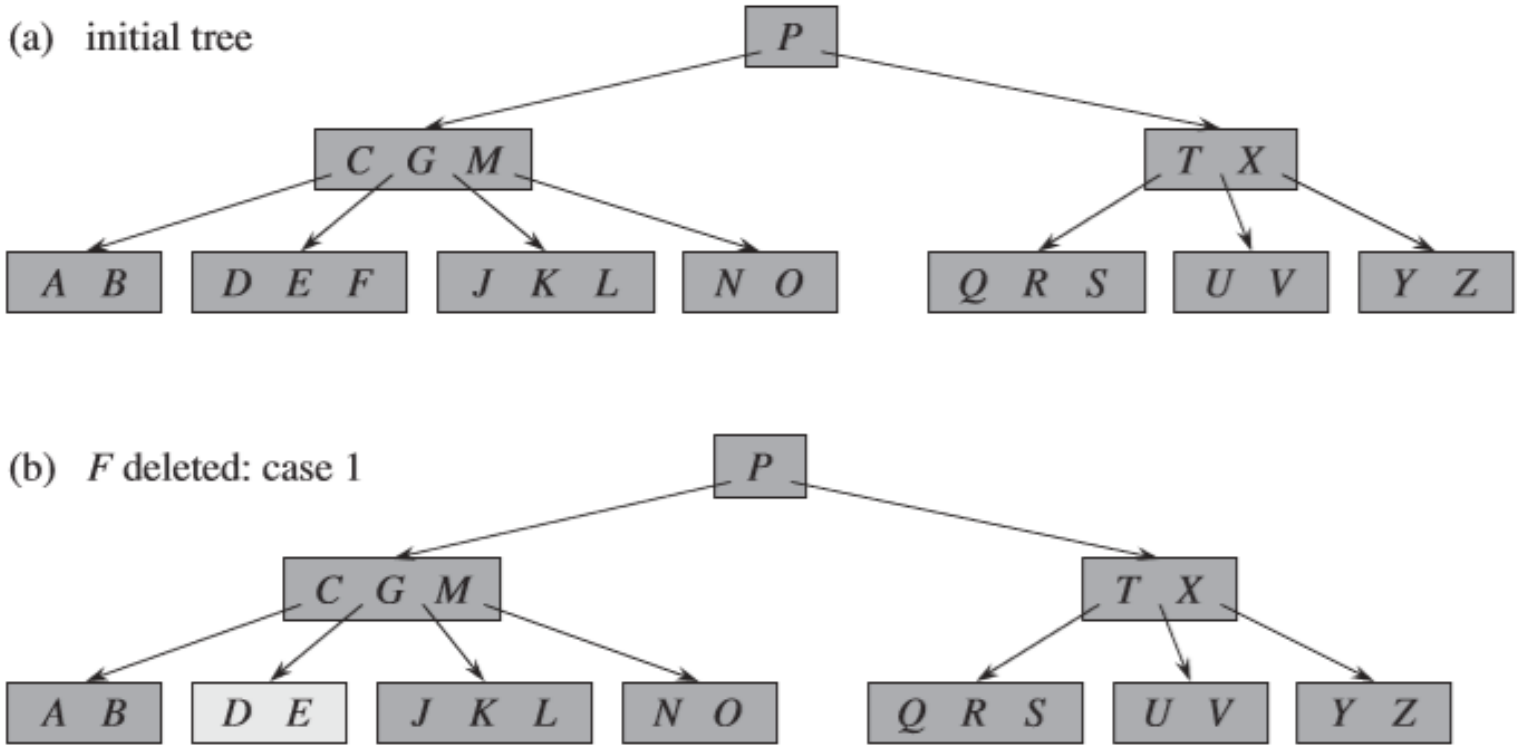
Searching in a B-tree set (M = 6)

Delete in B-Tree

Deleting a Key from B-Tree

- The delete function is very similar to insert function. However, the main difference between these two functions is: we insert key to leaves but we can delete the key from anywhere on tree.
- So after deleting we must ensure that tree is still obey the rules of B-Tree and re arrange the tree if necessary.
- While delete operation continues if there is a node that has minimum number of key the operation might be backed up. To prevent this we ensure that whenever function calls itself the node still has, at least minimum degree t.

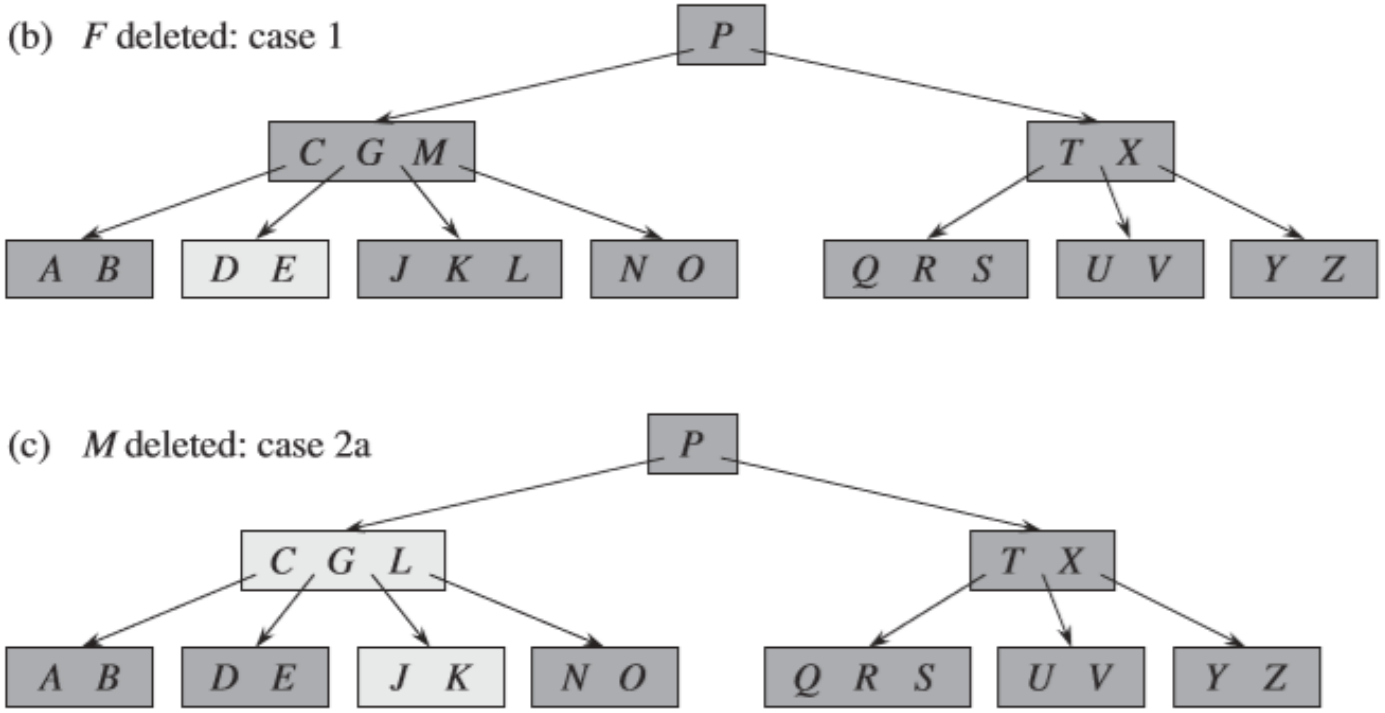
Deleting a Key from B-Tree



Minimum degree for this B-Tree is 3 so all nodes should have at least 3 keys.

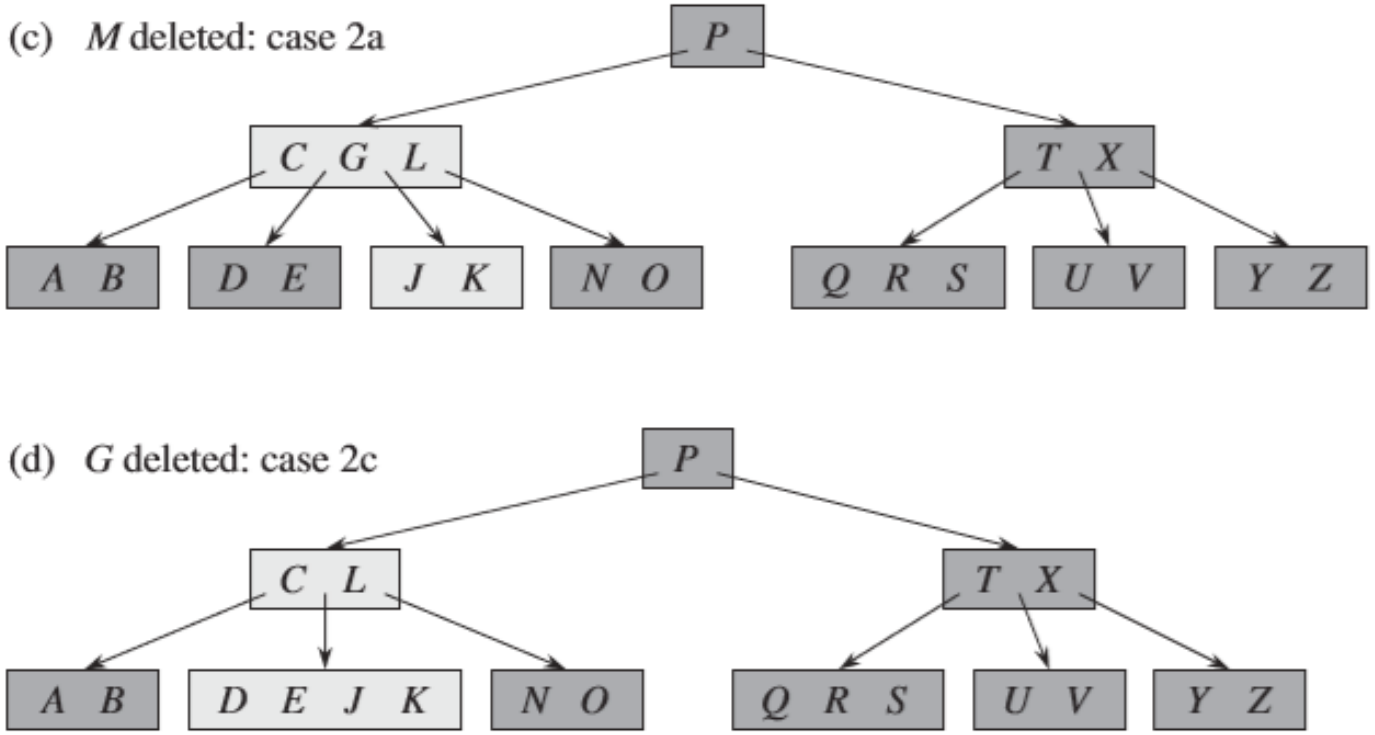
- Case 1: If the key node is a leaf delete the node.

Deleting a Key from B-Tree



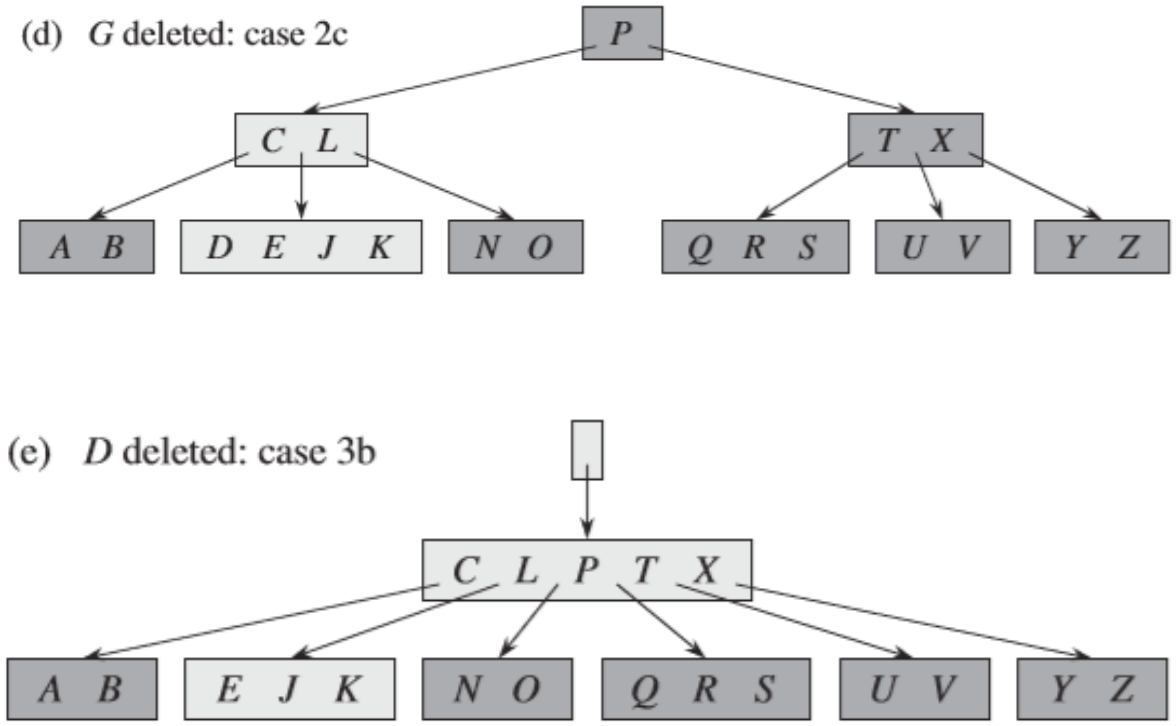
- Case 2a-b: If the key (k) is a internal node (x) find a child has at least t keys that will precedes k. Then recursively delete k and replace it with the child node.

Deleting a Key from B-Tree



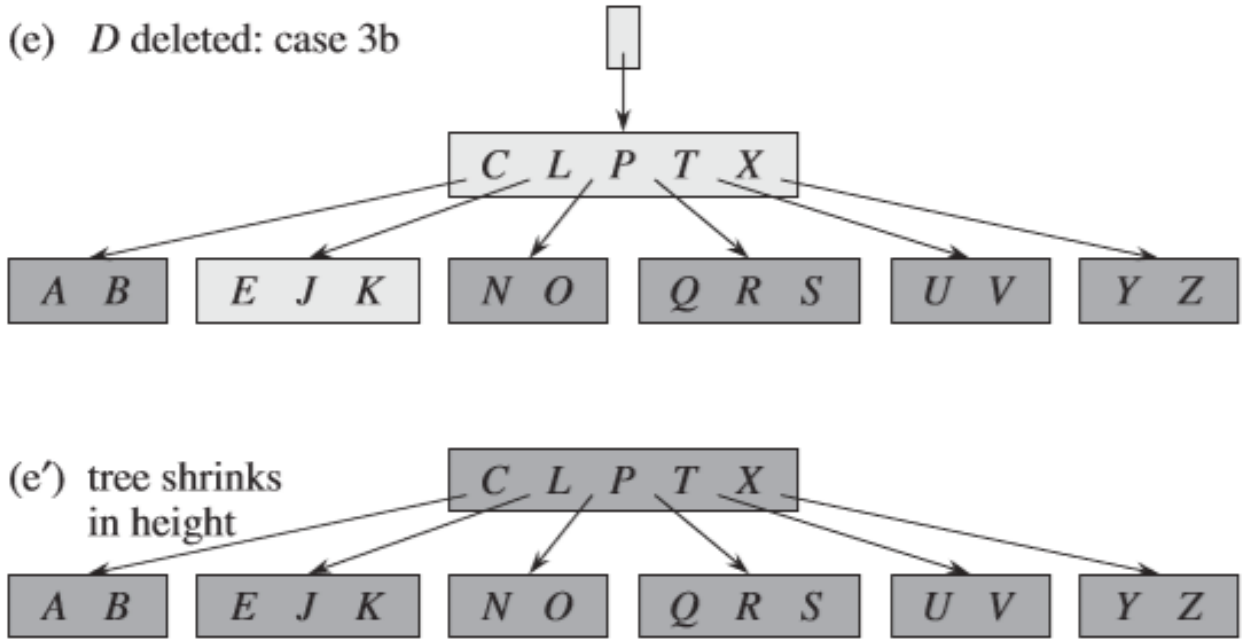
- Case 2c: If the key node (k) is a internal node and there isn't a child that has at least t keys, then merge k and all of its children to left nodes children.
- Then recursively delete k and replace it with the child node.

Deleting a Key from B-Tree



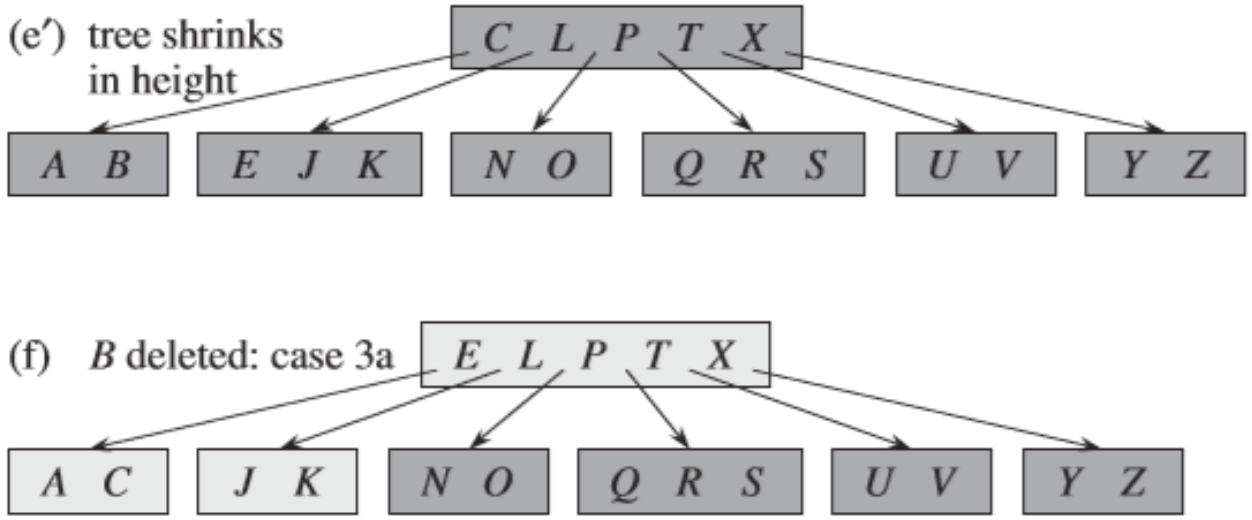
- Case 3b: If the key k is not present in internal node x, determine the root of the appropriate subtree that contain k.
- If node x and it s neighbors have t - 1 keys, merge x with one neighbor.
- Move a key from x down into the new merged node to become the median key for that node.

Deleting a Key from B-Tree



- C L P T X becomes the root node the height gets shorter.

Deleting a Key from B-Tree



- Case 3a: If node x has t -1 keys but has a neighbor with at least t keys, give x a key from it's parent.
- Then, move a key from x's left or right neighbor up to parent node.
- Finally move the appropriate child pointer.

Big O for Deleting a Key from B-Tree

- If the delete function is for a leaf node then the function does one downward pass through the tree, without having to back up. However, when deleting a key in an internal node function again makes a downward but have to return a node to replace the key with its predecessor or successor.
- This procedure involves only O(h) disk operations. ($O(h)=O(\log_t n)$) Also, DISK-READ and DISK-WRITE makes O(1) calls. The CPU time required is $O(th)= O(t \log_t n)$.

The book Introduction to Algorithms is used as a refence.