6

MULTIWAY TREES

Unit Structure:

6.1 Definition of a Multi-way Tree
6.2 B-Trees

6.1. DEFINITION OF MULTI-WAY TREE:

A Multi-way tree is a tree which each node contains one or more keys.

A Multi-way (or m-way) search tree of order m is a tree in which

- each node has m or less than m sub-trees and
- contains one less key than its sub-trees
- e.g. The figure below is a multi-way search tree of order 4 because
  - the root has 4 sub-trees and
  - It contains 3 keys (.i.e. root contains one less key than no. of sub-trees)

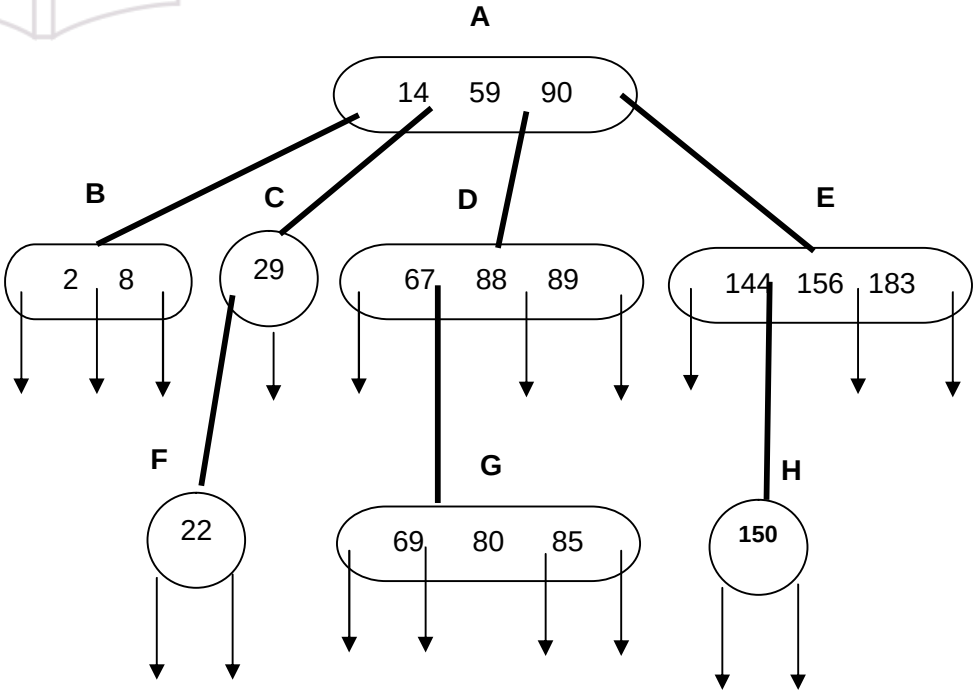


Fig.6.1. Multi-way search tree

- The keys in each node are sorted
- All the keys in a sub-tree S_0 are less than or equal to K_0
  - .e.g. keys of the sub-tree B (.i.e. 2 and 8) are less than first key of the root A (.i.e.14)
- All keys in the sub-tree Sj (1 < j < m-2) are greater than Kj-1 and less than or equal to Kj
  - .e.g. the second sub-tree of A contains the keys 29 & 22 which are greater than 14 (the first key of A) and less than 59 (the second key of A)

6.2 B-TREES :

6.2.1 Introduction

A B-tree is a specialized multiway tree designed especially for use on disk. In a B-tree each node may contain a large number of keys. The number of subtrees of each node, then, may also be large. A B-tree is designed to branch out in this large number of directions and to contain a lot of keys in each node so that the height of the tree is relatively small. This means that only a small number of nodes must be read from disk to retrieve an item. The goal is to get fast access to the data, and with disk drives this means reading a very small number of records. Note that a large node size (with lots of keys in the node) also fits with the fact that with a disk drive one can usually read a fair amount of data at once.

6.2.2 Definition :

A B-tree of order m is a multi-way search tree of order m such that:

- All leaves are on the bottom level.
- Each non-root node contains at least (m-1)/2 keys
- Each node can have at most m children
- For each node, if k is the actual number of children in the node, then k - 1 is the number of keys in the node

A B-Tree that is a multi-way search tree of order 4 has to fulfill the following conditions related to the ordering of the keys:

- The keys in each node are in ascending order.
- At the root node the following is true:
  - The sub-tree starting at first branch of the root has only keys that are less than the first key of the root

- The sub-tree starting at the second branch of root has only keys that are greater than first key of root and at the same time less than second key of root.
- The sub-tree starting at third branch has only keys that are greater than second key of root and at the same time less than third key of root.
- The sub-tree starting at the last branch .i.e. the fourth branch has only keys that are greater than the third key of root.

Example

The following is an example of a B-tree of order 4.

This means that all non root nodes must have at least (4 -1)/ 2 = 1 children

Of course, the maximum number of children that a node can have is 4 (so 3 is the maximum number of keys).

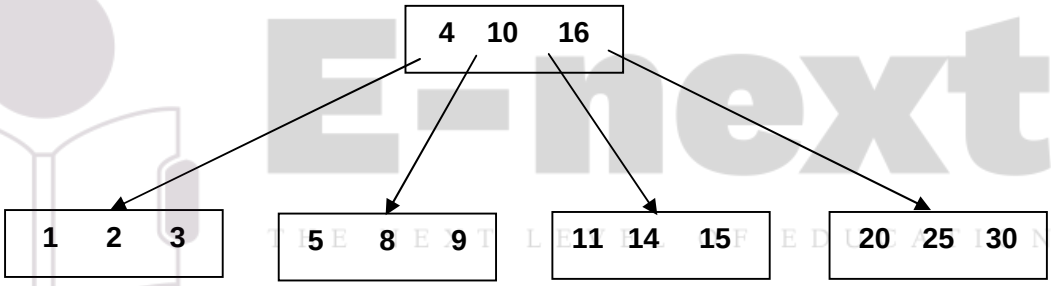


Fig.6.2: B Tree of order 4

In the above diagram the following are true:

- The sub-tree starting at first branch of the root has the keys 1,2,3 that are less than the first key of the root .i.e. 4
- The sub-tree starting at the second branch of root has the keys 5, 8, 9 that are greater than first key of root which is 4 and at the same time less than second key of root which is 10.
- The sub-tree starting at third branch has the keys 11, 14, 15 that are greater than second key of root which is 10 and at the same time less than third key of root which is 16.
- The sub-tree starting at the last branch .i.e. the fourth branch has the keys 20, 25, 30 that are greater than the third key of root which is 16.

6.2.3 B- Tree Operations:

In this section we will study the following operations

1. Insertion in a B-Tree
2. Deletion in a B-Tree

6.2.3.1 Insertion in a B-Tree:

In a B-Tree Insertion takes place at the leaf node

1. Search for the leaf node where the data is to be entered

2. If the node has less than m-1 entries then the new data are simply inserted in the sequence.

3. If the node is full, the insertion causes overflow. Therefore split the node into two nodes. Bring the median (middle) data to its parents left entries of the median to be copied to the left sub-tree and right entries copied into right sub-tree.

   While splitting we come across the following circumstances

   - The new key is less than the median key
   - The new key is the median key
   - The new key is greater than the median key

If the new key is less than or equal to the median key the new data belongs to the left or original node.

If the new key is greater than the median key, the data belongs to the new node

Example:

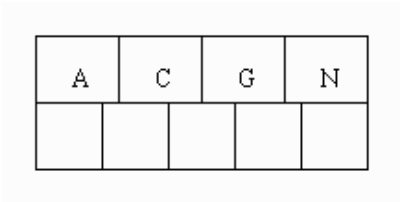
Construct a B-Tree of order 5 which is originally empty from the following data:

C N G A H E K Q M F W L T Z D P R X Y S

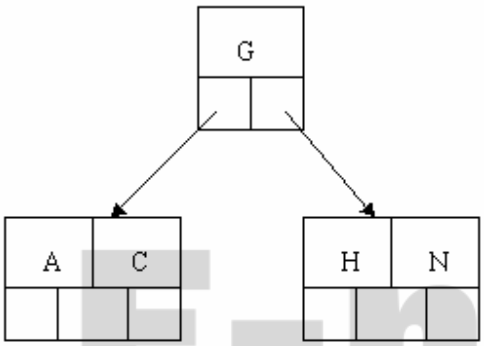
Order 5 means that a node can have a maximum of 5 children and 4 keys.

All nodes other than the root must have a minimum of 2 keys.

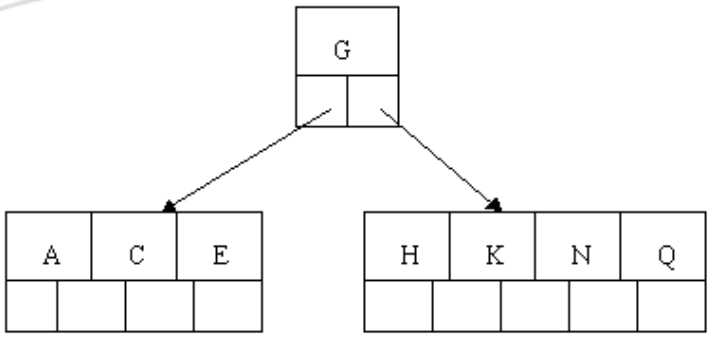
The first 4 letters get inserted into the same node, resulting in this picture



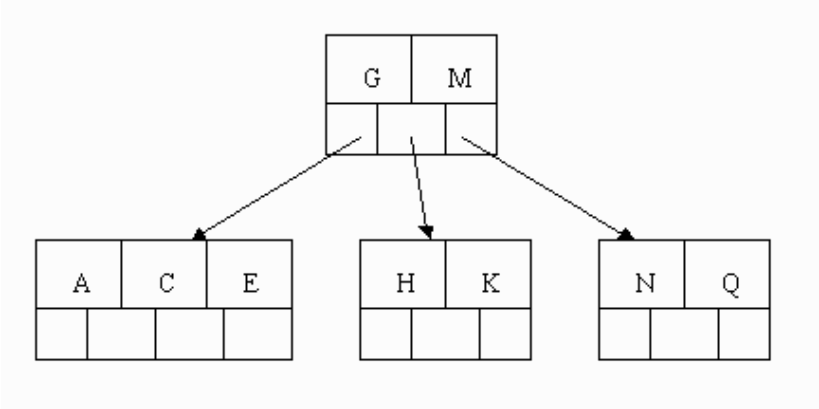
When we try to insert the H, we find no room in this node, so we split it into 2 nodes, moving the median item G up into a new root node. Note that in practice we just leave the A and C in the current node and place the H and N into a new node to the right of the old one.



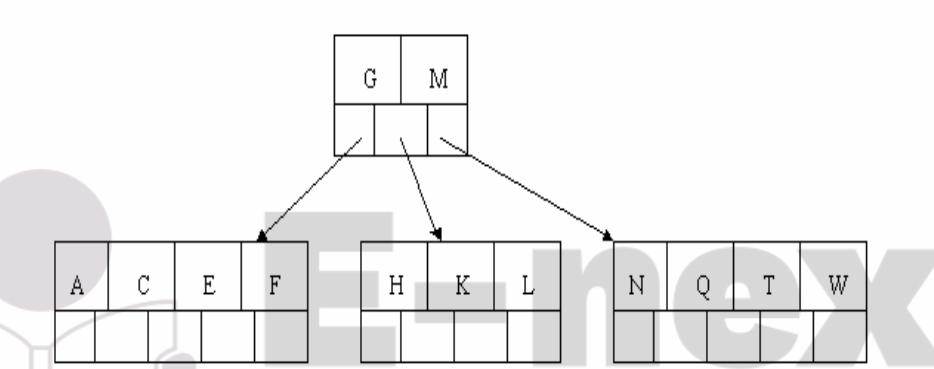
Inserting E, K, and Q proceeds without requiring any splits:



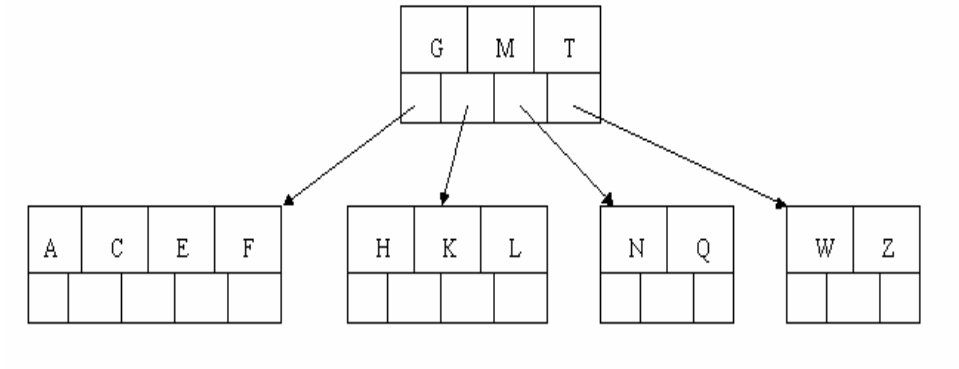
Inserting M requires a split. Note that M happens to be the median key and so is moved up into the parent node.



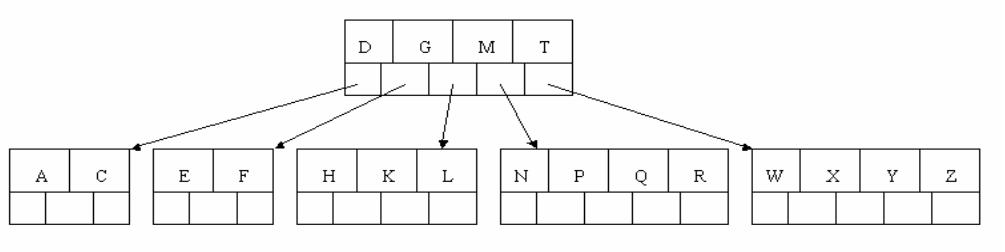
The letters F, W, L, and T are then added without needing any split.



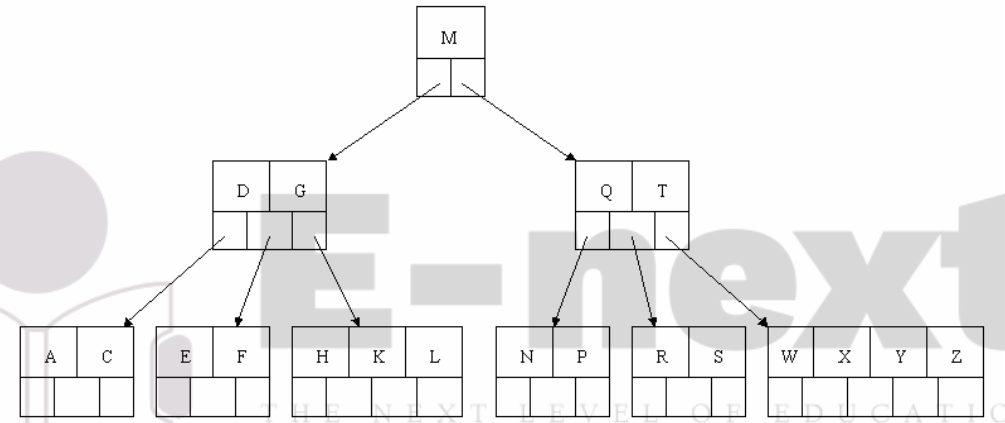
When Z is added, the rightmost leaf must be split. The median item T is moved up into the parent node. Note that by moving up the median key, the tree is kept fairly balanced, with 2 keys in each of the resulting nodes.



The insertion of D causes the leftmost leaf to be split. D happens to be the median key and so is the one moved up into the parent node. The letters P, R, X, and Y are then added without any need of splitting:



Finally, when S is added, the node with N, P, Q, and R splits, sending the median Q up to the parent. However, the parent node is full, so it splits, sending the median M up to form a new root node. Note how the 3 pointers from the old parent node stay in the revised node that contains D and G.



6.2.3.2 Deletion in B-Tree:

In a B-Tree Deletion takes place at the leaf node

1. Search for the entry to be deleted.
2. If found then continue, else terminate.
3. If it is a leaf simply delete it.
4. If it is an internal node find a successor from its sub-tree and replace it. There are two data items that can be substituted, either the immediate predecessor or immediate successor
5. After deleting if the node has less than the minimum entries (underflow) try Balancing or Combining then continue else terminate

Balancing:

It shifts the data among nodes to reestablish the integrity of the tree. Because it does not change the structure of the tree. We balance a tree by rotating an entry from one sibling to another

through parent. The direction of rotation depends upon the number of siblings in the left or right sub-trees.

.e.g. deletion of R in the next example involves balancing to maintain the integrity of the B Tree

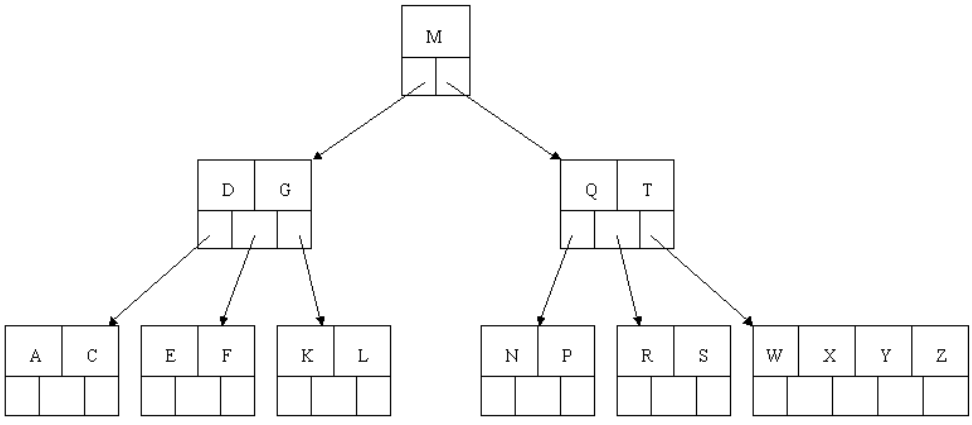
Combining:

Combining joins the data from an under-flowed entry, a minimal sibling and a parent node. The result is one node with maximum entries and an empty node that must be recycled. It makes no difference which sub-tree has under-flowed, we combine all the nodes into left sub-tree. Once this has been done we recycle the right sub-tree.

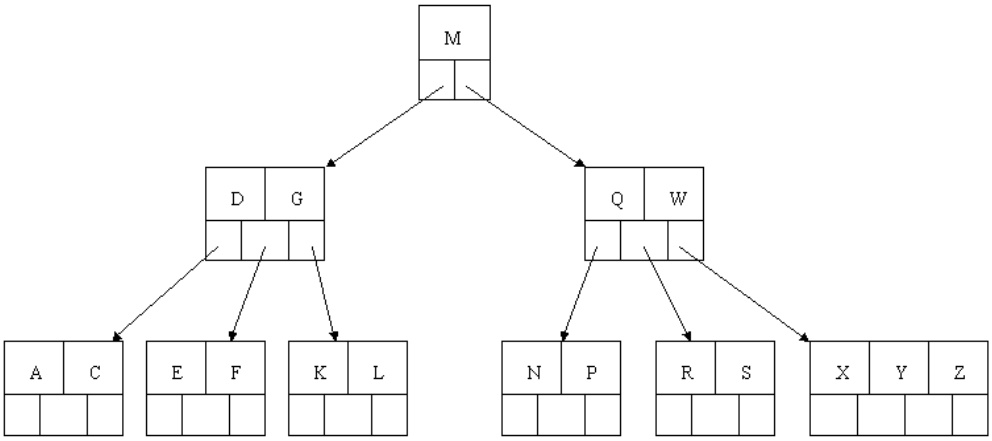
E.g. Deletion of D in the next example results in an underflow and then involves combining of data items.

Example: Deletion in B- Tree

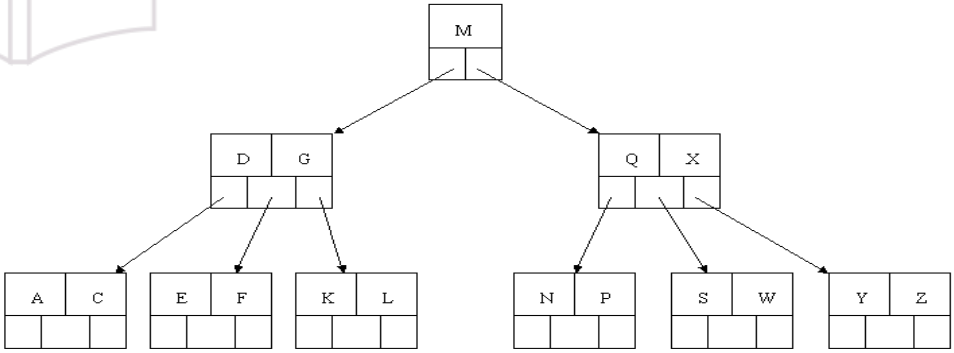
In the B-tree as we left it at the end of the last section, delete H. Of course, we first do a lookup to find H. Since H is in a leaf and the leaf has more than the minimum number of keys, this is easy. We move the K over where the H had been and the L over where the K had been. This gives:



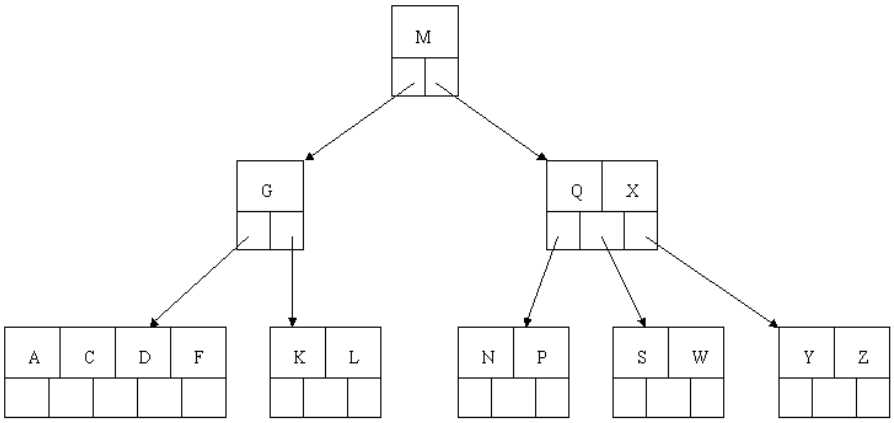
Next, delete the T. Since T is not in a leaf, we find its successor (the next item in ascending order), which happens to be W, and move W up to replace the T. That way, what we really have to do is to delete W from the leaf, which we already know how to do, since this leaf has extra keys. In ALL cases we reduce deletion to a deletion in a leaf, by using this method.



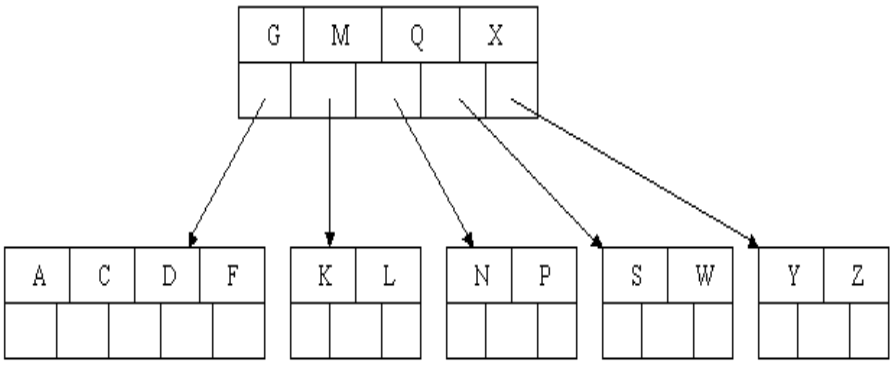
Next, delete R. Although R is in a leaf, this leaf does not have an extra key; the deletion results in a node with only one key, which is not acceptable for a B-tree of order 5. If the sibling node to the immediate left or right has an extra key, we can then borrow a key from the parent and move a key up from this sibling. In our specific case, the sibling to the right has an extra key. So, the successor W of S (the last key in the node where the deletion occurred), is moved down from the parent, and the X is moved up. (Of course, the S is moved over so that the W can be inserted in its proper place.)



Finally, let's delete E. This one causes lots of problems. Although E is in a leaf, the leaf has no extra keys, nor do the siblings to the immediate right or left. In such a case the leaf has to be combined with one of these two siblings. This includes moving down the parent's key that was between those of these two leaves. In our example, let's combine the leaf containing F with the leaf containing A C. We also move down the D.

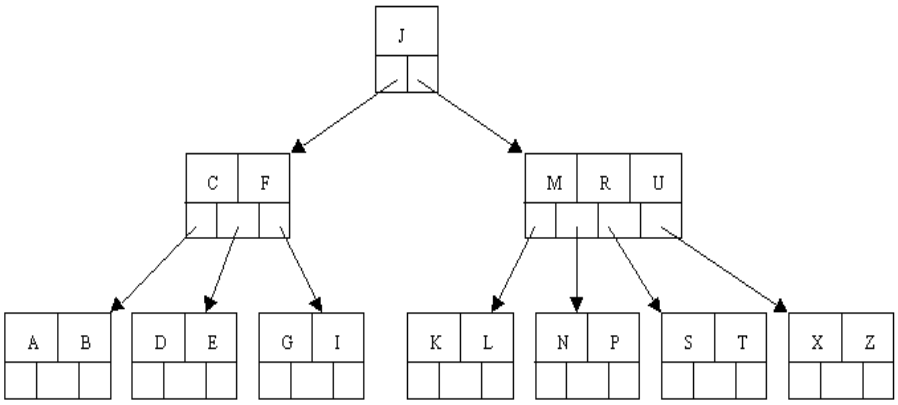


Of course, you immediately see that the parent node now contains only one key, G. This is not acceptable. If this problem node had a sibling to its immediate left or right that had a spare key, then we would again "borrow" a key. Suppose for the moment that the right sibling (the node with Q X) had one more key in it somewhere to the right of Q. We would then move M down to the node with too few keys and move the Q up where the M had been. However, the old left subtree of Q would then have to become the right subtree of M. In other words, the N P node would be attached via the pointer field to the right of M's new location. Since in our example we have no way to borrow a key from a sibling, we must again combine with the sibling, and move down the M from the parent. In this case, the tree shrinks in height by one.

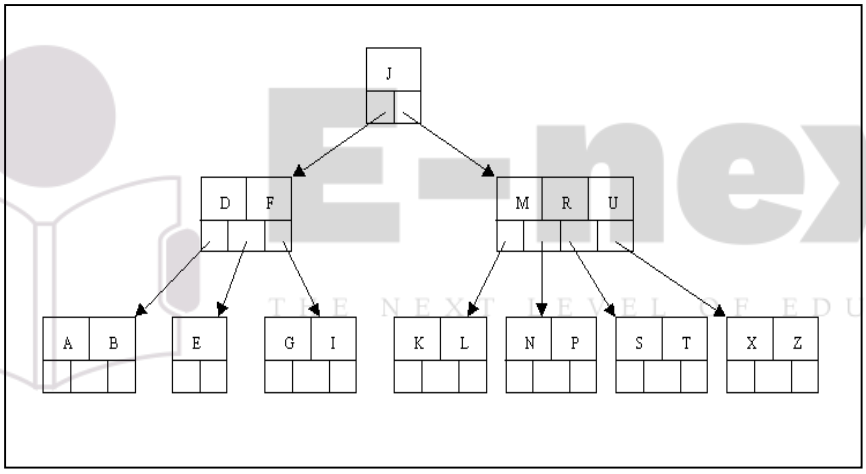


Another Example

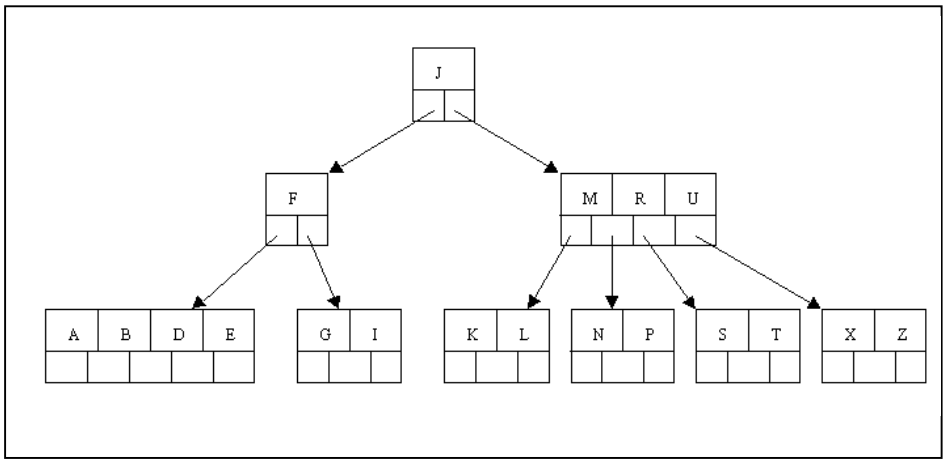
Here is a different B-tree of order 5. Let's try to delete C from it.



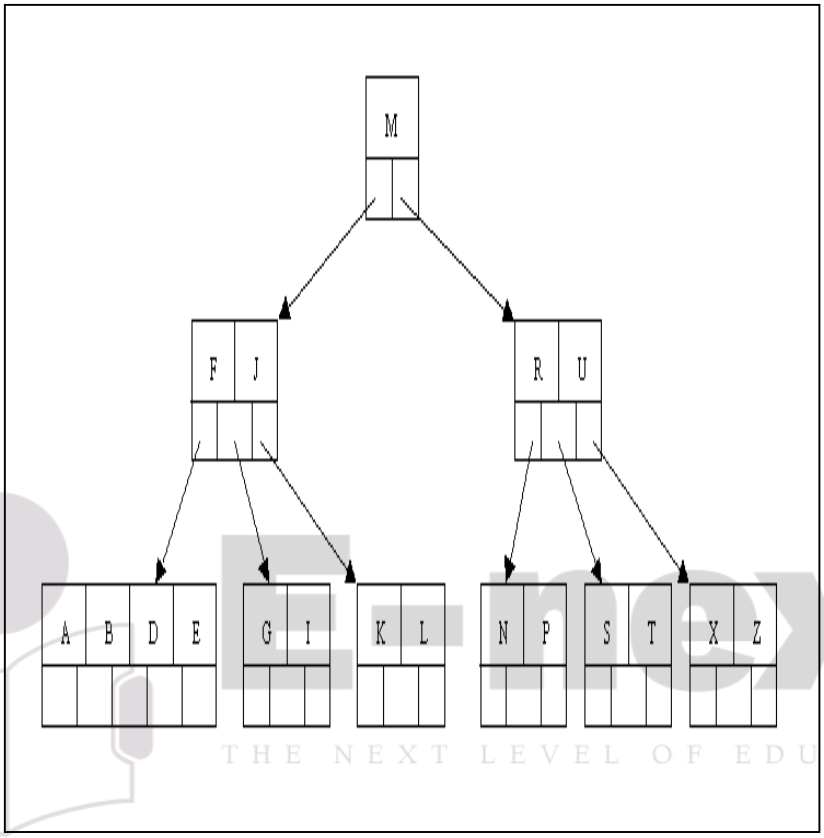
We begin by finding the immediate successor, which would be D, and move the D up to replace the C. However, this leaves us with a node with too few keys.



Since neither the sibling to the left or right of the node containing E has an extra key, we must combine the node with one of these two siblings. Let's consolidate with the A B node.



But now the node containing F does not have enough keys. However, its sibling has an extra key. Thus we borrow the M from the sibling, move it up to the parent, and bring the J down to join the F. Note that the K L node gets re-attached to the right of the J.



❖❖❖❖