CSc 3102: B-trees

Supplementary Notes

- Introduction
- B-tree Properties
- B-tree Insertion
- B-tree Deletion
- Real-world Applications

1 Introduction

An *m-nary* tree is a generalization of the binary tree. The m-ary tree is said to be an m^{th} order tree where each node has at most m children. If $k \leq m$ is the number of children for a given node, then that node must have $k-1$ keys. An $m-nary$ search tree is also a generalization of a binary search tree. A B-tree is a special case of an *m-nary* search tree.

2 The B-tree Properties

Definition 1. An m^{th} order B-tree is a balanced *m-nary* search tree in which

1. Leaf nodes are at the same level.
2. All internal nodes except the root have at most m nonempty children, and at least $\lceil \frac{m}{2} \rceil$ nonempty children.
3. Each internal node has 1 less key than the number of children that it has, and its keys partition the keys in such a way to preserve the search tree property.

4. The root has at most m children, but may have as few as 2 if it is not a leaf, or none if it is the only node.

3 B-tree Insertion

We now work through an example to illustrate the transformations that are required to preserve the B-tree properties while keys are inserted into the tree. The *split* and *merge* operations are used to prevent a node from getting an excess key, too few keys or preserve the depth of the leaf nodes.

Example Trace the insertion of the keys N, G, B, A, H, Q, K, E, M, T, V, F, L, Z, D, X, Y, R, P, and S into an initially empty 5^{th} order B-tree.

Observe that each non-root internal node in a 5^{th}-order B-tree must have at least 2 and at most 4 keys. We insert the keys N, G, B and A into the root node, arranging them in ascending order as shown in figure 1.

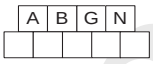

Figure 1: After inserting N, G, B and A

Next, inserting H into the root will lead to an excess key so the node is *split* after the insertion and the median key, G, is moved upward and becomes the key in the new root node. An internal root node may have a single key. After the insertion of H in the tree in figure 1, we get the b-tree shown in figure 2.

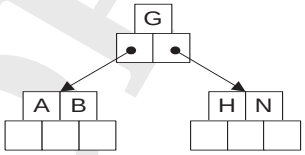

Figure 2: After inserting H

The keys Q, K and E can be readily inserted into leaf nodes since they can accommodate additional keys. See figure 3 for the b-tree after these keys are inserted in the b-tree shown in figure 2.‘

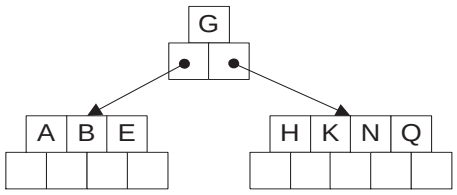

Figure 3: After inserting Q, K and E

When M is inserted, there will be an excess key in the node. This requires that the node is *split* to preserve the b-tree properties. We similarly move M the median key in the root node after the split and obtain the tree shown in figure 4.

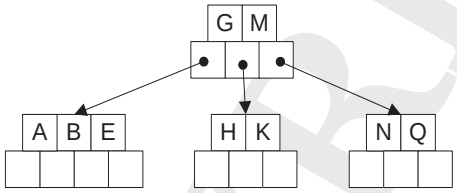

Figure 4: After inserting M

The keys T, V, F and L can be inserted without requiring any further split as shown in figure 5.

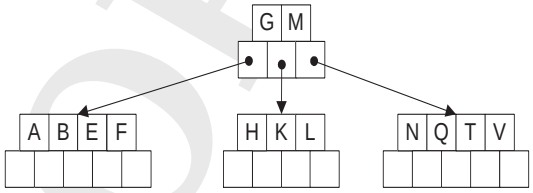

Figure 5: After inserting T, V, F and L

When Z is inserted, the right-most leaf node must be split. The median key in that node, T, is moved up into the parent node and we get the b-tree shown in figure 6.

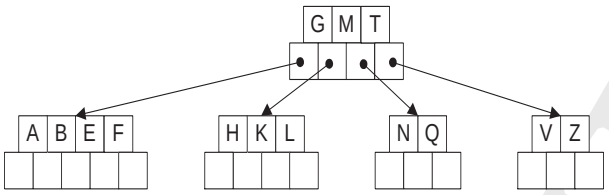

Figure 6: After inserting Z

In order to insert D, the left-most leaf node must be split. We similarly move D, the median key upward to the parent node and we get the b-tree shown in figure 7.

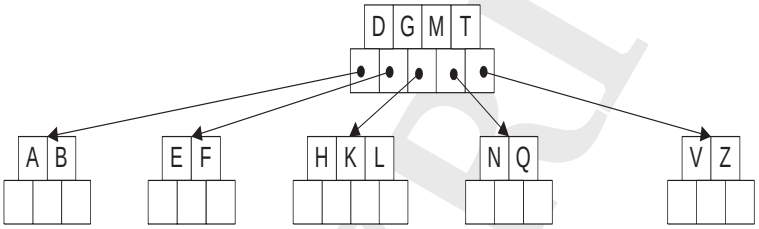

Figure 7: After inserting D

X and Y can be inserted into the node containing V and Z without the node exceeding its capacity. Similarly, R and P can be inserted into the node containing N and Q without the node exceeding its capacity. After the insertion of X, Y, R and P, we get the b-tree shown in figure 8.

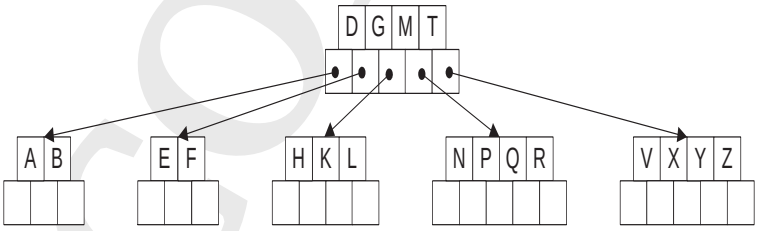

Figure 8: After inserting X, Y, R and P

Inserting S into the leaf node containing N, P, Q, and R requires a split. The median Q is moved up to the parent node. Since the parent node does not have room to accommodate Q, the median in the parent node, M, is moved up to its parent, the root node. After all of these transformations, we get the b-tree shown in figure 9.

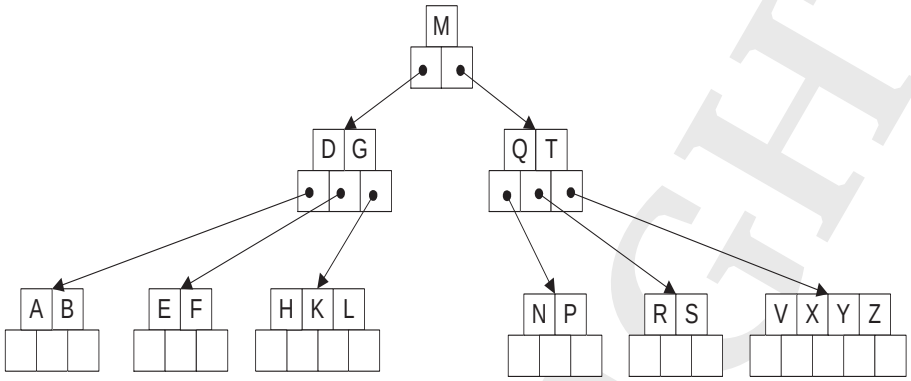

Figure 9: After inserting S

4 B-tree Deletion

When deleting from a B-tree, the transformations that are applied to the tree must preserve its B-tree properties. Nodes with too few keys may be *merged* with their nearby siblings or keys may be *transferred* from nodes with extra keys to those with too few. To illustrate the transformations, we perform a series of deletion starting with the B-tree shown in figure 9.

Deleting H from the leaf node is pretty straightforward since the node has more than the minimum required keys. H is removed from the leaf node and we get the tree shown in figure 10.

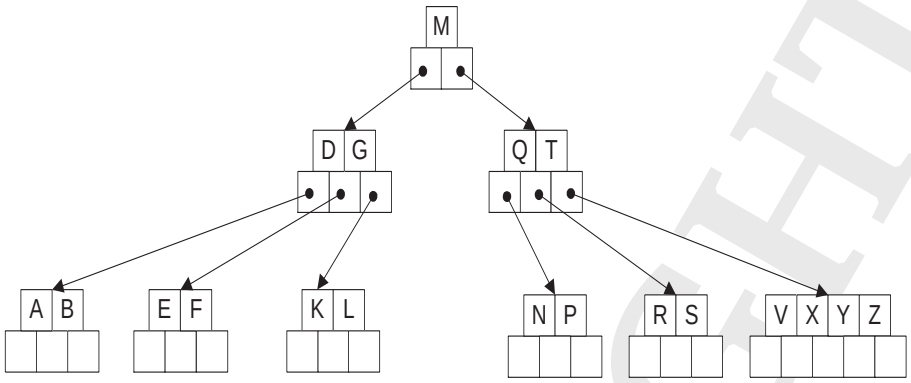

Figure 10: After deleting H

Deleting T involves replacing it with its successor, V. Since the node containing V has an extra key, after T is replaced no further transformation is required and we get the tree shown in figure 11.

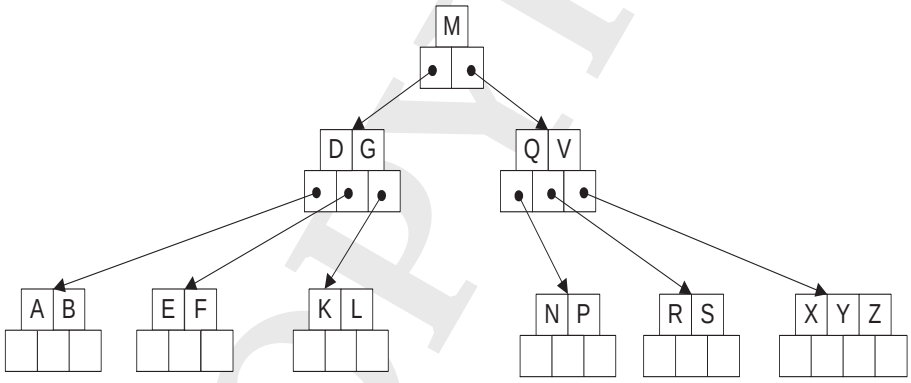

Figure 11: After deleting T

Deleting R from the leaf node involves more than just removing it since the node will be left with only one key, fewer than the required minimum. Since its right sibling has an extra key to spare, we move a key, V, from its parent to replace R and then move X, the extra key, from its sibling to its parent node to replace the V. Each node now has the required minimum number of keys.

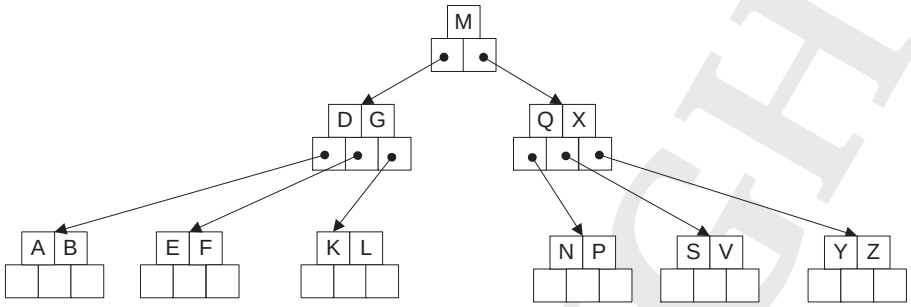

Figure 12: After deleting R

Deleting E is much more involved since its nearby siblings do not have an extra key to spare. So we replace the key E with D, a key from its parent node, [DG].

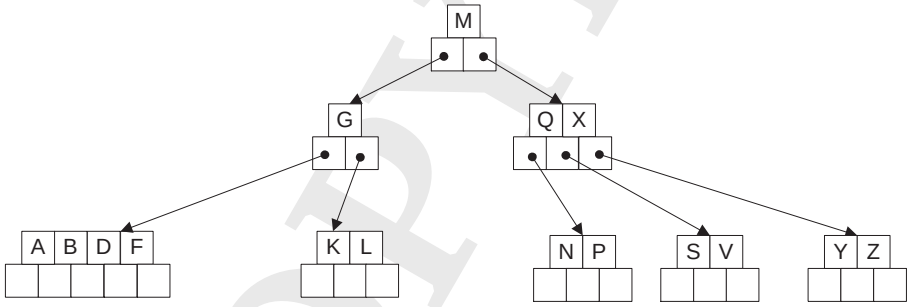

Figure 13: After replacing E with D

The [G] node is now left with fewer than the required minimum number of keys. We merge the nodes [AB] and [DF]. The nearby sibling node [G] does not have an extra key to spare so we move M, a key from the parent node, into the node [G]. This effectively deletes the root. We then merge [GM] and [QX] and the height of tree is reduced by 1.

This demonstrates that deletion from a B-tree beginning at a leaf node may percolate up to the root of the tree and reduce the height by 1.

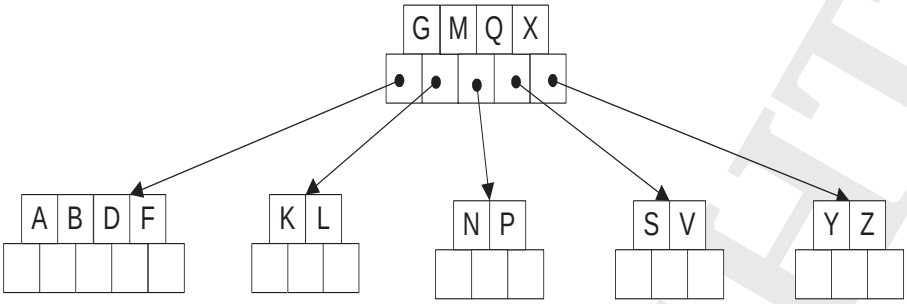

Figure 14: After deleting E

Example Trace the deletion of C from the tree shown in figure 15.

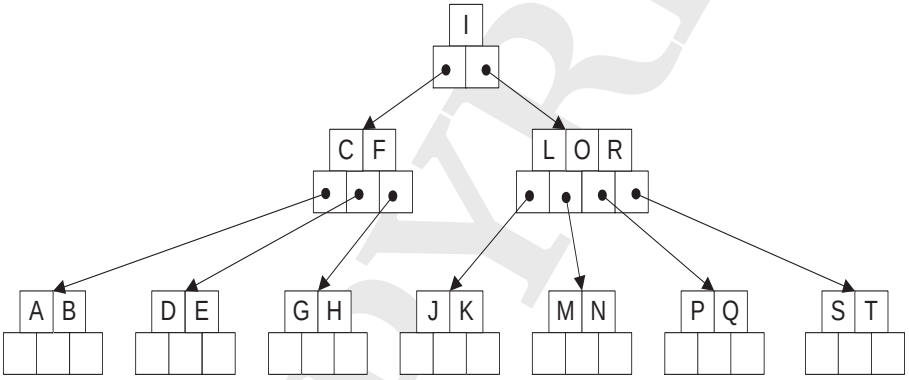

Figure 15: A B-tree

First, we replace C with its successor D. This leaves the leaf node [DE] with only one key, fewer than the required minimum, as shown in figure 16.

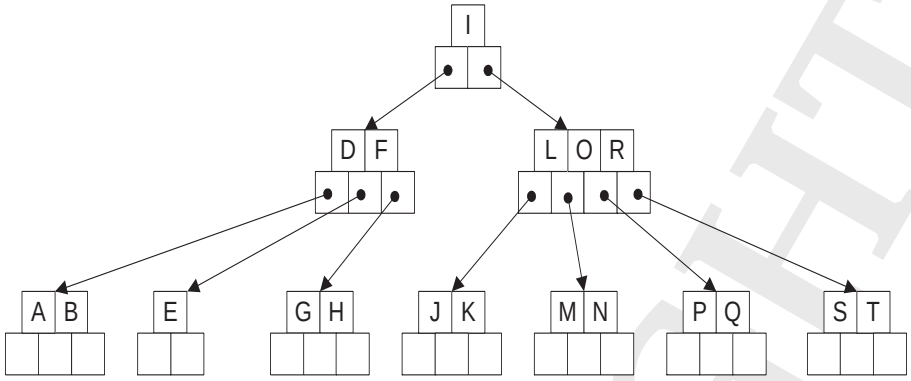

Figure 16: After replacing C

Neither of [E]'s siblings has an extra key to spare. So we move the parent key D into the [E] node and merge it with its sibling [AB] to obtain [ABDE] with [F] as its parent as shown in figure 17.

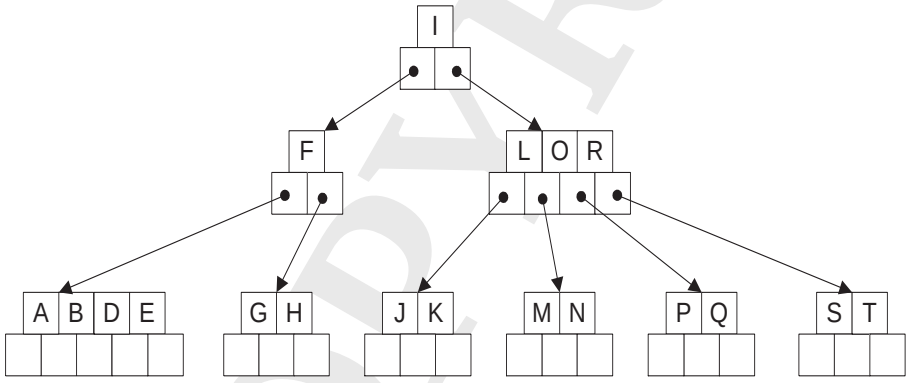

Figure 17: After merging the leaf nodes [AB] and [DE]

Now, this creates another problem. The node [F] now has fewer than the minimum required number of keys and must be augmented. In this case, its nearby sibling [LOR] has an extra key to spare. So we move its parent key, I, into its node and the extra key L into the parent node, as shown in figure 18.

Also, to preserve the search tree property of the B-tree, the node [JK], a node now orphaned, becomes the right-most child of the subtree rooted at [FI].

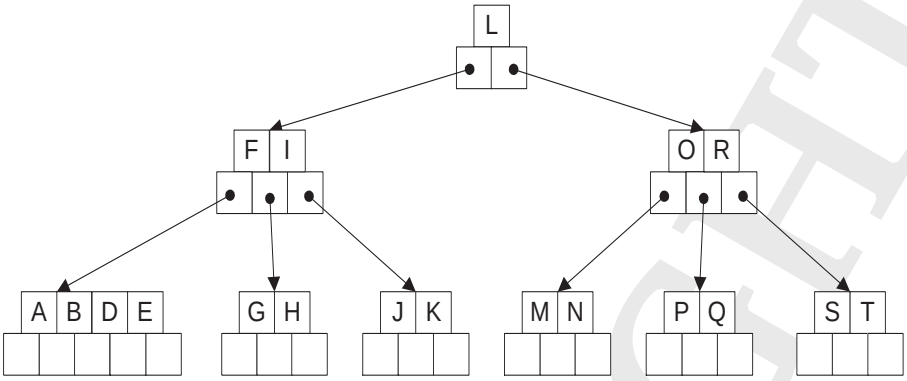

Figure 18: After moving the keys L and I

Observe that in this case, the deletion does not percolate up to the root because a sibling node with an extra key was found. No further transformation is required and the height of the tree remains the same.

5 Search, Traversal and Applications

Searching a B-tree is similar to searching a binary search tree. Similarly, traversal operations in B-trees are analogous to those done in binary trees. Variants of B-trees have been used widely in data transfer in memory hierarchy between cache, memory, disk, etc, and are also used in database software to perform range queries. Variants of B-trees are widely used in file management systems and external sorting applications.

Definition 2. **Preorder traversal** of a B-tree can be done by recursively visiting all the entries in the root node first, then traversing all the subtrees, from left to right, in preorder.

Definition 3. **Postorder traversal** of a B-tree can be done by recursively first traversing all the subtrees of the root, from left to right, in postorder, then visiting all the entries in the root.

Problem 1.

1. Trace the insertion of the keys S, P, R, Y, X, D, Z, L, F, V, T, M, E, K, Q, H, A, B, G and N into an initially empty 4^{th} order B-tree by drawing figures similar to figures 1-9. Draw the tree each time its height increases and after the insertion of N. When splitting a node, use the third key as the median key.

2. Give the preorder traversal of the tree in figure 9.

3. Give the postorder traversal of the tree in figure 9.

4. How would you recursively define inorder traversal of a B-tree? Your proposed definition must guarantee that the keys are visited in non-decreasing order.

5. What is the maximum number of keys that a fifth order B-tree whose height is three can have? What is the minimum number of keys that it can have?

6. Give an expression, in terms of h, for the maximum number of keys an order 5 B-tree whose height is h can have. Give a similar expression for the minimum number of keys.

7. At most how many entries may be examined during a search in a 5^{th} order B-tree of height h.

8. Suppose a 5^{th} order B-tree of height h is minimal with respect to number of entries. At most how many entries are examined during a search?

9. Give an expression for the height h of an order m B-tree that is maximal with respect to entries. Give the expression in terms of m, .