B+ TREES

B Trees. B Trees are multi-way trees. That is each node contains a set of keys and pointers. A B Tree with four keys and five pointers represents the minimum size of a B Tree node. A B Tree contains only data pages.

B Trees are dynamic. That is, the height of the tree grows and contracts as records are added and deleted.

B+ Trees A B+ Tree combines features of ISAM and B Trees. It contains index pages and data pages. The data pages always appear as leaf nodes in the tree. The root node and intermediate nodes are always index pages. These features are similar to ISAM. Unlike ISAM, overflow pages are not used in B+ trees.

The index pages in a B+ tree are constructed through the process of inserting and deleting records. Thus, B+ trees grow and contract like their B Tree counterparts. The contents and the number of index pages reflects this growth and shrinkage.

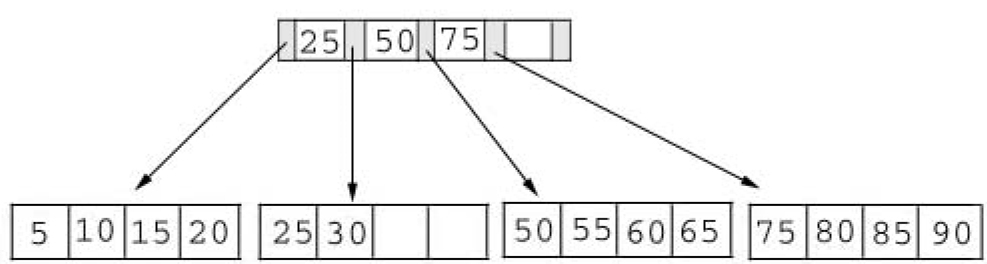
B+ Trees and B Trees use a "fill factor" to control the growth and the shrinkage. A 50% fill factor would be the minimum for any B+ or B tree. As our example we use the smallest page structure. This means that our B+ tree conforms to the following guidelines.

Number of Keys/page	4
Number of Pointers/page	5
Fill Factor	50%
Minimum Keys in each page	2

As this table indicates each page must have a minimum of two keys. The root page may violate this rule.

The following table shows a B+ tree. As the example illustrates this tree does not have a full index page. (We have room for one more key and pointer in the root page.) In addition, one of the data pages contains empty slots.

B+ Tree with four keys



Adding Records to a B+ Tree

The key value determines a record's placement in a B+ tree. The leaf pages are maintained in sequential order AND a doubly linked list (not shown) connects each leaf page with its sibling page(s). This doubly linked list speeds data movement as the pages grow and contract.

We must consider three scenarios when we add a record to a B+ tree. Each scenario causes a different action in the insert algorithm. The scenarios are:

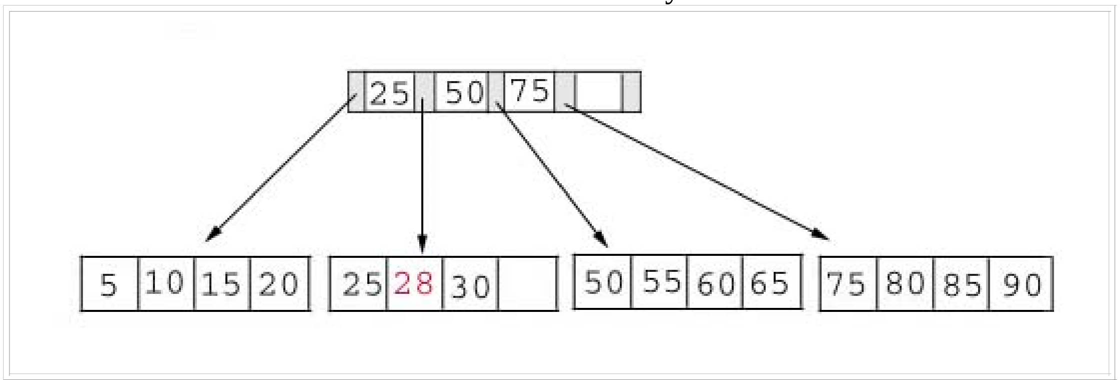
The `insert` algorithm for B+ Trees

Leaf Page Full	Index Page FULL	Action
NO	NO	Place the record in sorted position in the appropriate leaf page
YES	NO	1. Split the leaf page 2. Place Middle Key in the index page in sorted order. 3. Left leaf page contains records with keys below the middle key. 4. Right leaf page contains records with keys equal to or greater than the middle key.
YES	YES	1. Split the leaf page. 2. Records with keys < middle key go to the left leaf page. 3. Records with keys >= middle key go to the right leaf page. 4. Split the index page. 5. Keys < middle key go to the left index page. 6. Keys > middle key go to the right index page. 7. The middle key goes to the next (higher level) index. IF the next level index page is full, continue splitting the index pages.

Illustrations of the insert algorithm

The following examples illlustrate each of the **insert** scenarios. We begin with the simplest scenario: inserting a record into a leaf page that is not full. Since only the leaf node containing 25 and 30 contains expansion room, we're going to insert a record with a key value of 28 into the B+ tree. The following figures shows the result of this addition.

Add Record with Key 28



Adding a record when the leaf page is full but the index page is not

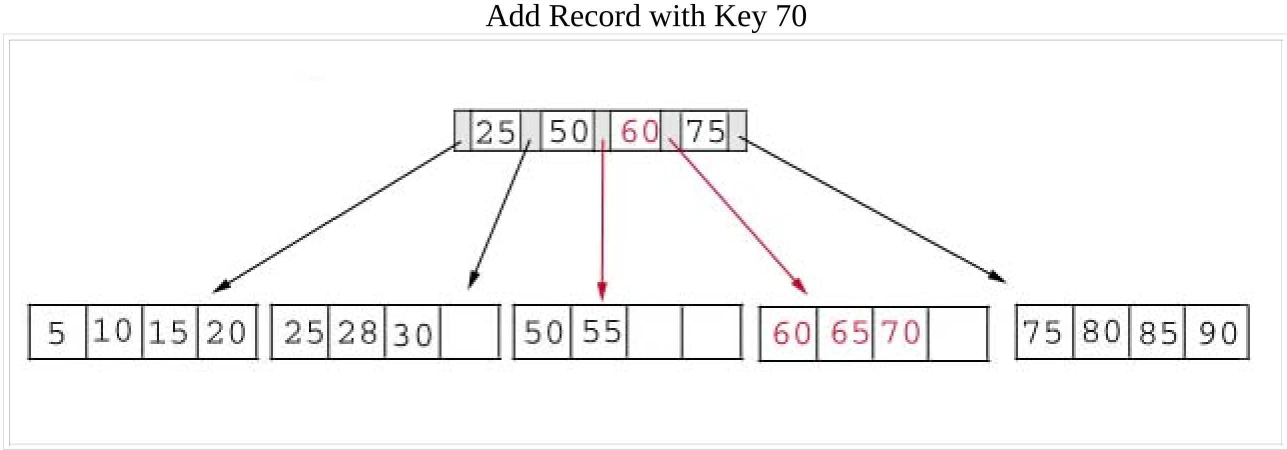
Next, we're going to insert a record with a key value of 70 into our B+ tree. This record should go in the leaf page containing 50, 55, 60, and 65. Unfortunately this page is full. This means that we must split the page as follows:

Left Leaf Page	Right Leaf Page
50 55	60 65 70

The middle key of 60 is placed in the index page between 50 and 75.

The following table shows the B+ tree after the addition of 70.

Add Record with Key 70



Adding a record when both the leaf page and the index page are full

As our last example, we're going to add a record containing a key value of 95 to our B+ tree. This record belongs in the page containing 75, 80, 85, and 90. Since this page is full we split it into two pages:

Left Leaf Page	Right Leaf Page

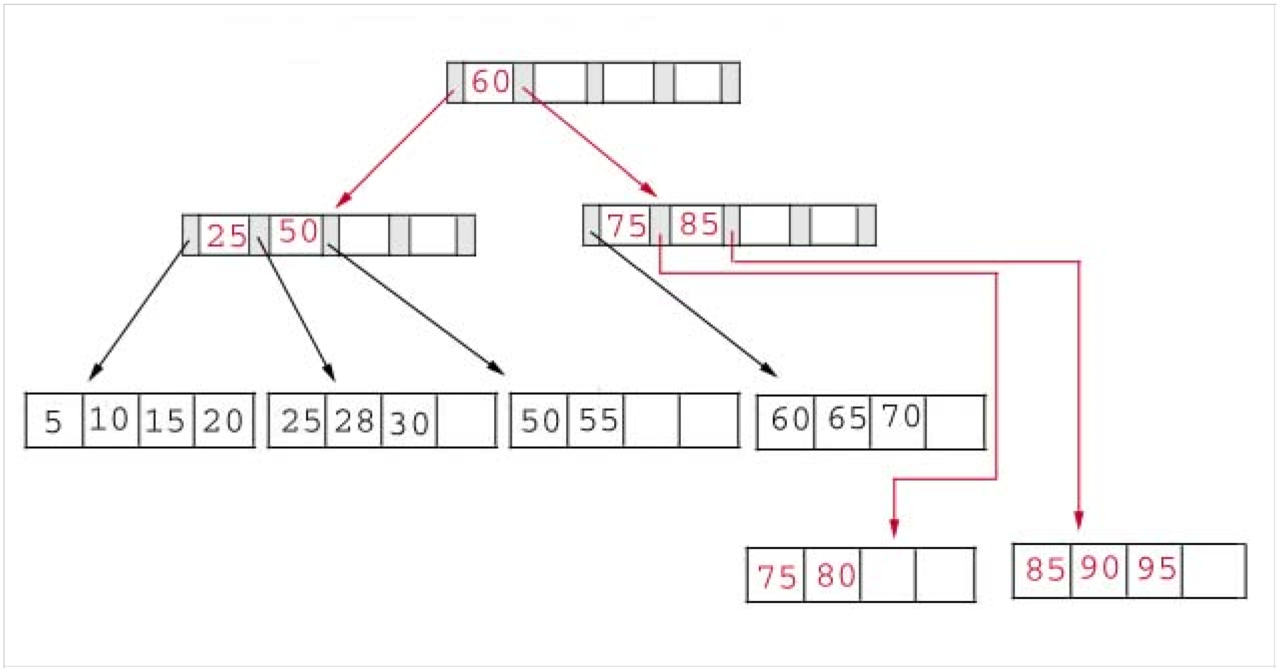
75 80	85 90 95

The middle key, 85, rises to the index page. Unfortunately, the index page is also full, so we split the index page:

Left Index Page	Right Index Page	New Index Page
25 50	75 85	60

The following table illustrates the addition of the record containing 95 to the B+ tree.

Add Record with Key 95

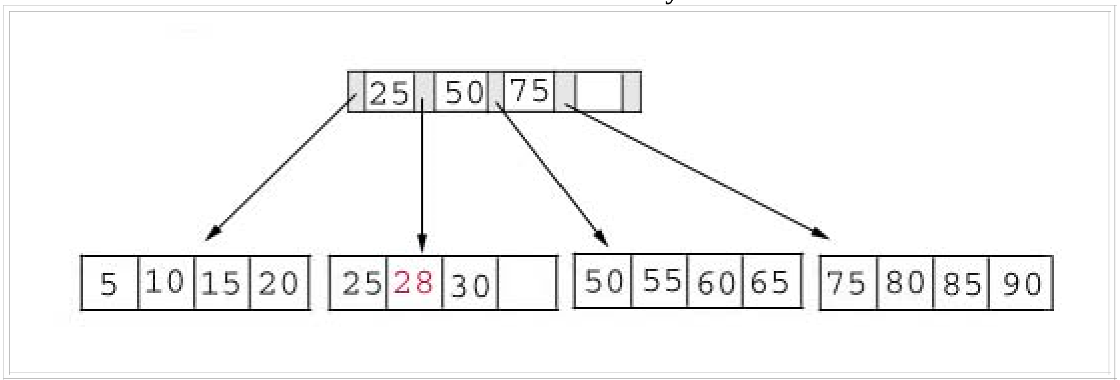


Rotation

B+ trees can incorporate rotation to reduce the number of page splits. A rotation occurs when a leaf page is full, but one of its sibling pages is not full. Rather than splitting the leaf page, we move a record to its sibling, adjusting the indices as necessary. Typically, the left sibling is checked first (if it exists) and then the right sibling.

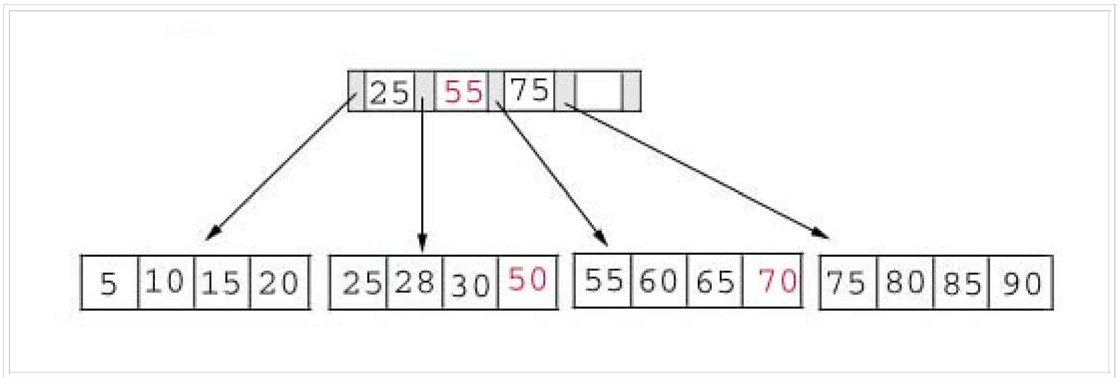
As an example, consider the B+ tree before the addition of the record containing a key of 70. As previously stated this record belongs in the leaf node containing 50 55 60 65. Notice that this node is full, but its left sibling is not.

Add Record with Key 28



Using rotation we shift the record with the lowest key to its sibling. Since this key appeared in the index page we also modify the index page. The new B+ tree appears in the following table.

Illustration of Rotation



Deleting Keys from a B+ tree

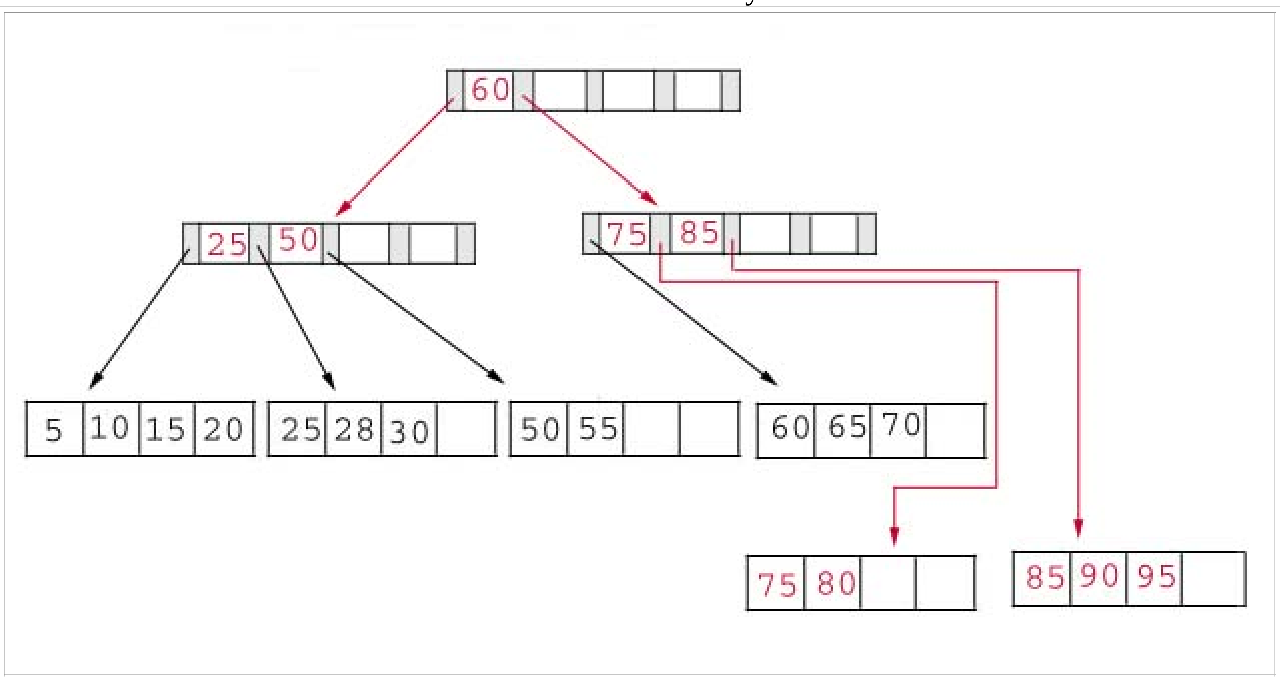
We must consider three scenarios when we delete a record from a B+ tree. Each scenario causes a different action in the delete algorithm. The scenarios are:

The `delete` algorithm for B+ Trees

Leaf Page Below Fill Factor	**Index Page Below Fill Factor**	**Action**
NO	NO	Delete the record from the leaf page. Arrange keys in ascending order to fill void. If the key of the deleted record appears in the index page, use the next key to replace it.
YES	NO	Combine the leaf page and its sibling. Change the index page to reflect the change.
YES	YES	1. Combine the leaf page and its sibling. 2. Adjust the index page to reflect the change. 3. Combine the index page with its sibling. Continue combining index pages until you reach a page with the correct fill factor or you reach the root page.

As our example, we consider the B+ tree after we added 95 as a key. As a refresher this tree is printed in the following table.

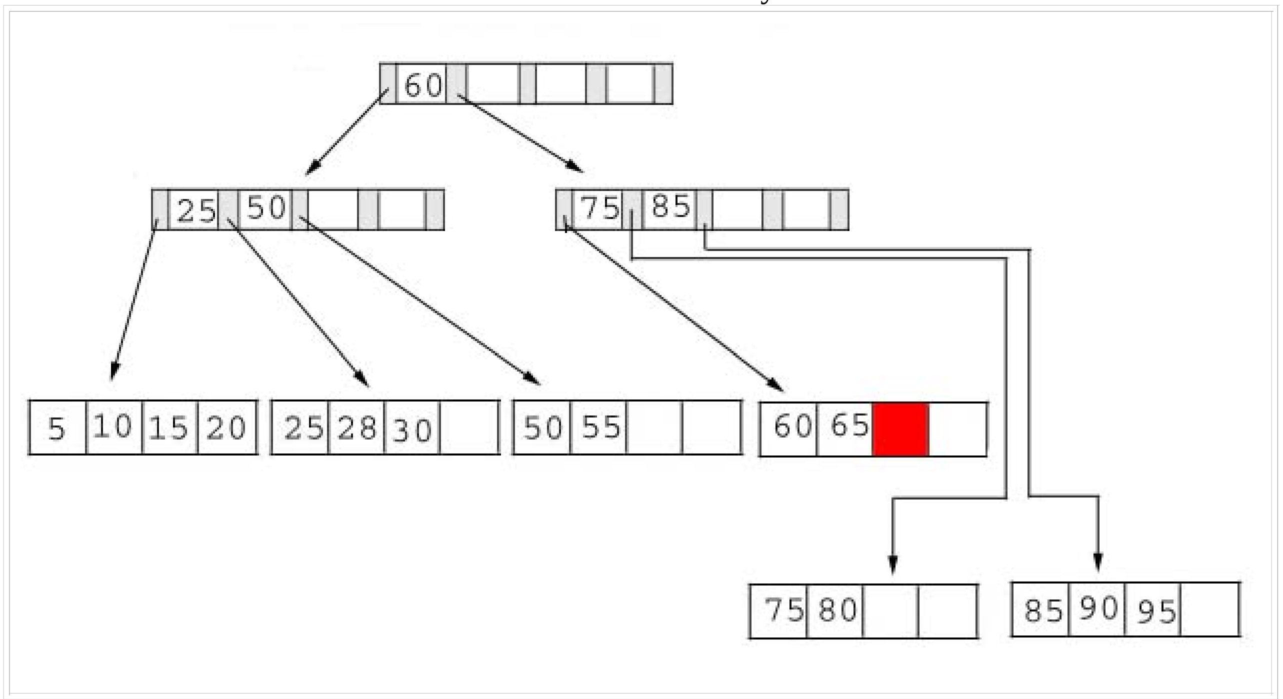
Add Record with Key 95



Delete 70 from the B+ Tree

We begin by deleting the record with key 70 from the B+ tree. This record is in a leaf page containing 60, 65 and 70. This page will contain 2 records after the deletion. Since our fill factor is 50% or (2 records) we simply delete 70 from the leaf node. The following table shows the B+ tree after the deletion.

Delete Record with Key 70

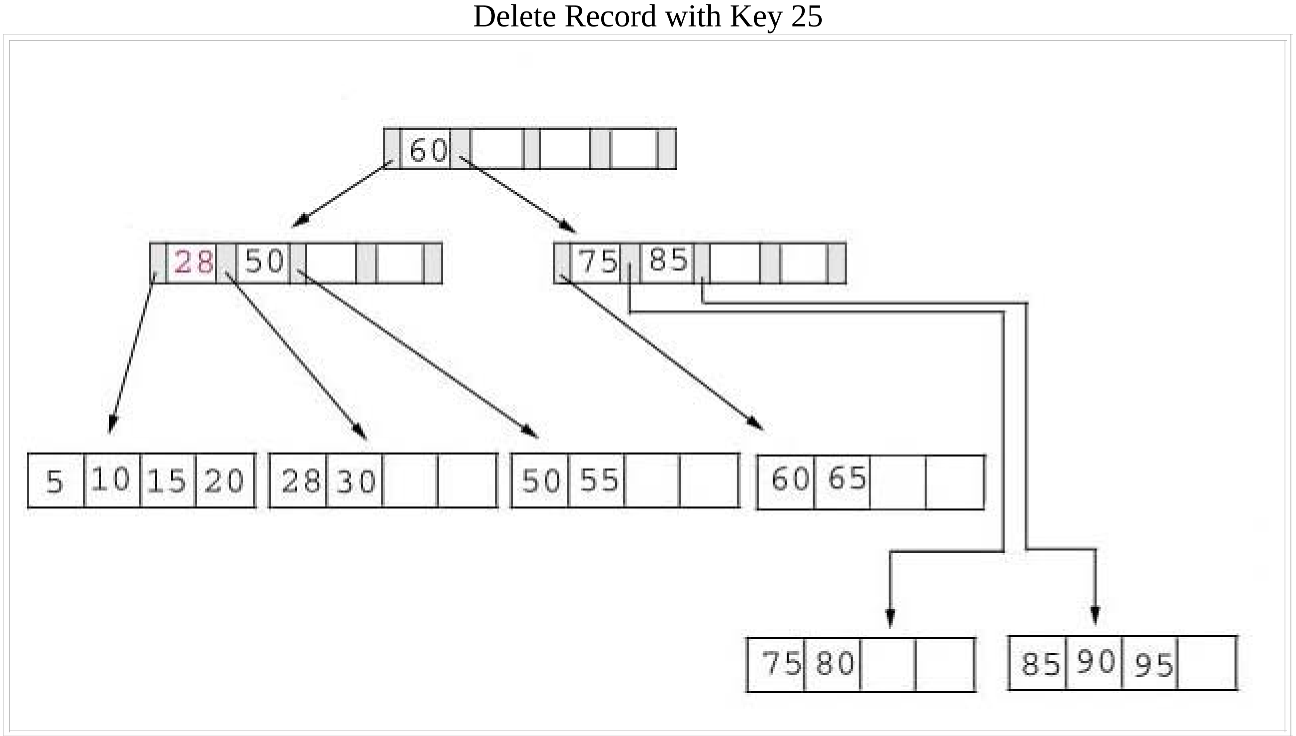


Delete 25 from the B+ tree

Next, we delete the record containing 25 from the B+ tree. This record is found in the leaf node containing 25, 28, and 30. The fill factor will be 50% after the deletion; however, 25 appears in the index page. Thus, when we delete 25 we must replace it with 28 in the index page.

The following table shows the B+ tree after this deletion.

Delete Record with Key 25



Delete 60 from the B+ tree

As our last example, we're going to delete 60 from the B+ tree. This deletion is interesting for several resasons:

1. The leaf page containing 60 (60 65) will be below the fill factor after the deletion. Thus, we must combine leaf pages.
2. With recombined pages, the index page will be reduced by one key. Hence, it will also fall below the fill factor. Thus, we must combine index pages.
3. Sixty appears as the only key in the root index page. Obviously, it will be removed with the deletion.

The following table shows the B+ tree after the deletion of 60. Notice that the tree contains a single index page.

Delete Record with Key 60