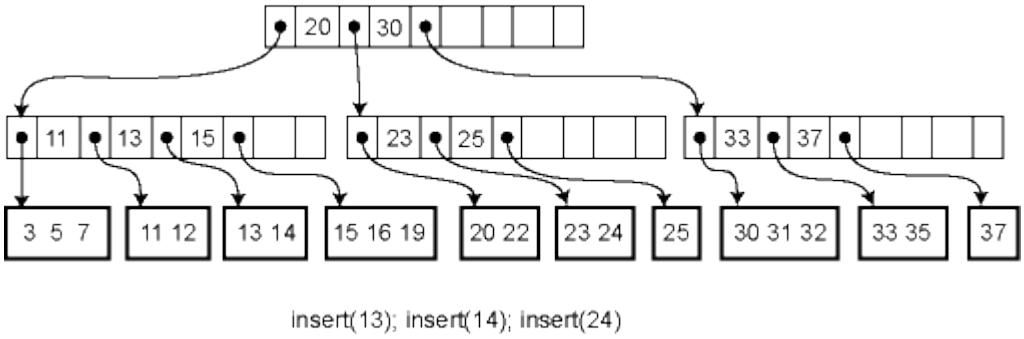
Assignment 3 : Btrees

Introduction:

- Btrees are data structures used mainly for data retrieval from disk. . The elements (ordered according to their keys), are stored in groups, each group called a node.
- The properties of a Btree are as follows:
- The degree is given by t.
- The minimum number of elements in a node is t-1, except the root. The root can have even 1 element.
- Maximum number of elements in a node is 2t -1.
- New elements are always inserted into the leaves of the tree.
- All leaves are at the same depth in the tree.

Btrees as stated before are commonly used for information retrieval, when all the data cannot be stored in main memory at one point in time. In such implementations, only a few nodes of the btree are in main memort at any point of time. Once the necessary access/modification is done, they are written back to secondary memory.



insert(13); insert(14); insert(24)

Implementation:

The implementation of Btrees has been done in 2 ways:

• **Btree stored in files**: The whole Btree is stored in secondary memory, and only a few required nodes are brought into memory at a time. This invovles explicit writes and reads from files. This has also been implemented in C.

• Completely in memory. The whole Btree is stored in memory at all times. This restricts the size of the tree to the size of available main memory space. The language of implementation is C.

The functions implemented are :

• **Insert** into the tree

• **Delete** an key from the tree

• **Search** for a key in the tree The nodes of the tree are stored in an array. Hence the top level data structure of the Btree is an array of nodes. Each node inturn is an array of records. A record is a structure containing the key, and other information pertaining to a particular record.

The algorithm used for the implementation of the above functions in both the memory anf file based methods are the same, hence providing the same interface to users.

Insert: It is implemented ina one pass method. This means that when a node is full, the split does not propagate backwards, instead when an internal node is full, it is split in anticipation of an element being inserted into it.

Delete: This is also a one pass implementation, which merges nodes which have t-1 elements in anticipation that another element might be deleted from them. However, some cases cannot be handled by a one pass method, and hence in one case there is backward propagation of merging . Deletion in a tree requires that many edge cases be handled, and hence is quite complex.

Search: It starts with the root, and recursively calls the corresponding child until the key is found, and prints the record information if it is found, else prints “key not found” if it is not present.

Performance Analysis:

The number of records in the tree is considered to be n in all cases.

Insert: Disk accesses: O(h) ; h is height -> log t n ; number of disk accesses by split : O(1) CPU time: each node has a maximum of 2t-1 records. O(t*h)

Delete: Disk accesses : O(h) ; h is height -> log t n CPU time : O(t*h)

Search: Disk accesses: O(h) ; h is height -> log t n CPU time: each node has a maximum of 2t-1 records. O(t*h)

Conclusions:

• A Btree is an efficient data structure for information retrieval.

• Since a parent node and its child node are stored near each other, when a page is brought into memory, there are high chances that no more disk accesses are required to bring in child nodes.

• This again, depends on the size of the node, and how many nodes can be accommodated in a single page.