Properties of the B-TREE

Order `m` of the tree

-> At most `m` children
-> At most `m - 1` keys in each node
-> At least `m/2` Children in each node
-> At least `(m/2)-1` keys in each node
Project B-TREE is of order `m = 4`
Tree wil be left biased

Max Child/Node	Min Child/Node	Max Keys/Node	Min Keys/Node
4	2	3	1

Keys in node

-> All keys inside a node have to be sorted
-> All keys in the left subtree are smaller than the keys in the node (Reverse is true)

B-Tree Functions and structure

```
typedef struct btree_node{
  bool leaf;
  int *keys;
  int nb_keys;
  struct btree_node **Children;
} btree_node;

typedef struct btree{
  struct btree_node *root;
} btree;
```

`struct btree_node *create_newNode()`
`struct btree *createBtree()`

`void free_Node(btree_node *node)`
`void free_Tree(btree *tree)`

Operation

Insertion

`void insertKey(int key, btree *tree)`

-> This is the main insertion function. It takes the key to insert and the B-Tree structure. It first checks if the root is full. If so, it creates a new root node, sets the old root as its child, and splits the full root to make room. Then it inserts the key into the correct child. If the root is not full, it directly inserts the key using `insertNonFullKey`.

`insertNonFullKey(btree_node _node, int key)`

-> This function inserts a key into a node that is guaranteed to have space. If the node is a leaf, the key is inserted in sorted order. If the node is internal, it finds the correct child to descend into by comparing with existing keys. If that child is full, it first splits the child. After the split, it decides whether to go left or right, and then it calls itself recursively to insert into the chosen child.

`void insertSplitChild(btree_node *parent, int index)`

-> This function splits the child node at `parent->Children[index]` because it is full. It finds the median key in that child and moves it up into the parent. It creates a new node to hold the right half of the keys and (if not a leaf) the right half of the children. Then it shifts the parent's keys and children to make space for the median and the new right node. This ensures the tree stays balanced and no node exceeds the key limit.

Deletion

Searching

Traversal

Indexing for search by

Save in memory

Other

Sources
https://www.youtube.com/watch?v=aNU9XYYCHu8&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=75