COMPETITIVE PROGRAMMING

Assignment-10

2303A51463

B-07

Database Indexing Using B+ Tree

Algorithm:

Insert(key):

1. If tree empty → create leaf node and insert key.
2. Else:
    a. Find correct leaf node.
    b. Insert key in sorted order.
3. If leaf overflows (> m-1 keys):
    a. Split leaf into two halves.
    b. Copy middle key to parent.
4. If parent overflows:
    a. Split internal node.
    b. Promote middle key upward.
5. Repeat until balanced.

Search(Key):

1. Start from root.
2. Compare key with node keys.
3. Move to appropriate child.
4. Reach leaf.
5. If key present → FOUND
6. Else → NOTFOUND

Delete(Key):

1. Search key in leaf.
2. Remove key.
3. If underflow (< minimum keys):
    a. Try borrowing from sibling.
    b. If not possible → merge with sibling.
4. Update parent keys.
5. If root becomes empty → adjust tree height.

Left Traversal:

1. Go to leftmost leaf.
2. Follow leaf next pointers.
3. Print all keys.

ass10.4.py > ...

```python
class Node:
    def __init__(self, order, leaf=False):
        self.order = order
        self.leaf = leaf
        self.keys = []
        self.children = []
        self.next = None

class BPlusTree:
    def __init__(self, order):
        self.root = Node(order, True)
        self.order = order

    def search(self, key):
        node = self.root
        while not node.leaf:
            i = 0
            while i < len(node.keys) and key >= node.keys[i]:
                i += 1
            node = node.children[i]
        return "FOUND" if key in node.keys else "NOTFOUND"

    def insert(self, key):
        root = self.root
        if len(root.keys) == self.order - 1:
            new_root = Node(self.order, False)
            new_root.children.append(self.root)
            self.split_child(new_root, 0)
            self.root = new_root
        self.insert_non_full(self.root, key)

    def insert_non_full(self, node, key):
        if node.leaf:
```

```
        if node.leaf:
            node.keys.append(key)
            node.keys.sort()
        else:
            i = 0
            while i < len(node.keys) and key >= node.keys[i]:
                i += 1
            if len(node.children[i].keys) == self.order - 1:
                self.split_child(node, i)
                if key >= node.keys[i]:
                    i += 1
            self.insert_non_full(node.children[i], key)

    def split_child(self, parent, index):
        order = self.order
        node = parent.children[index]
        new_node = Node(order, node.leaf)
        mid = (order - 1) // 2
        if node.leaf:
            new_node.keys = node.keys[mid:]
            node.keys = node.keys[:mid]
            new_node.next = node.next
            node.next = new_node
            parent.keys.insert(index, new_node.keys[0])
        else:
            parent.keys.insert(index, node.keys[mid])
            new_node.keys = node.keys[mid + 1:]
            node.keys = node.keys[:mid]
            new_node.children = node.children[mid + 1:]
            node.children = node.children[:mid + 1]
```

```
63
64              parent.children.insert(index + 1, new_node)
65
66          def delete(self, key):
67              node = self.root
68              while not node.leaf:
69                  i = 0
70                  while i < len(node.keys) and key >= node.keys[i]:
71                      i += 1
72                  node = node.children[i]
73              if key in node.keys:
74                  node.keys.remove(key)
75
76          def display(self):
77              node = self.root
78              while not node.leaf:
79                  node = node.children[0]
80              while node:
81                  for key in node.keys:
82                      print(key, end=" ")
83                  node = node.next
84              print()
85
86      order = 3
87      bpt = BPlusTree(order)
88      keys = [10, 20, 5, 6, 12, 30, 7]
89      for k in keys:
90          bpt.insert(k)
91      print(bpt.search(12))  # FOUND
92      bpt.delete(6)
93      print(bpt.search(6))   # NOTFOUND
94
```

Output:

```
PS C:\Users\harsh\OneDrive\Desktop\CP> & C:\Users\harsh
thon314\python.exe c:/Users/harsh/OneDrive/Desktop/CP/a
FOUND
NOTFOUND
```

Recently Accessed File Optimization Using Splay Tree

Algorithm:

Insert:

1. Insert like normal **Binary Search Tree**
2. After insertion → perform **SPLAY operation**
3. Move inserted node to root using rotations

Search:

1. Search like BST
2. If found:
    a. Perform **SPLAY**
    b. Move that node to root
    c. Print FOUND
3. Else:
    a. Print NOTFOUND

Delete:

1. First splay the node to root
2. If not found → stop
3. If found:
    a. If left subtree exists:
        i. Find maximum in left subtree
        ii. Splay it
        iii. Attach right subtree
    b. Else:
        i. Root becomes right subtree

Splay Cases:

Case 1:Zig

- Node is child of root → Single rotation

Case 2 :Zig-Zig

- Left-Left OR Right-Right → Double rotation same direction

Case 3: Zig-Zag

- Left-Right OR Right-Left → Double rotation opposite direction

ass10.5.py > SplayTree > splay

```python
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

class SplayTree:
    def __init__(self):
        self.root = None
    def right_rotate(self, x):
        y = x.left
        x.left = y.right
        y.right = x
        return y
    def left_rotate(self, x):
        y = x.right
        x.right = y.left
        y.left = x
        return y
    def splay(self, root, key):
        if root is None or root.key == key:
            return root
        if key < root.key:
            if root.left is None:
                return root
            if key < root.left.key:
                root.left.left = self.splay(root.left.left, key)
                root = self.right_rotate(root)
            elif key > root.left.key:
                root.left.right = self.splay(root.left.right, key)
                if root.left.right:
                    root.left = self.left_rotate(root.left)
            return self.right_rotate(root) if root.left else root
        else:
```

```
        else:
            if root.right is None:
                return root
            if key > root.right.key:
                root.right.right = self.splay(root.right.right, key)
                root = self.left_rotate(root)
            elif key < root.right.key:
                root.right.left = self.splay(root.right.left, key)
                if root.right.left:
                    root.right = self.right_rotate(root.right)
            return self.left_rotate(root) if root.right else root
    def insert(self, key):
        if self.root is None:
            self.root = Node(key)
            return
        self.root = self.splay(self.root, key)
        if self.root.key == key:
            return
        new_node = Node(key)
        if key < self.root.key:
            new_node.right = self.root
            new_node.left = self.root.left
            self.root.left = None
        else:
            new_node.left = self.root
            new_node.right = self.root.right
            self.root.right = None
        self.root = new_node

    def search(self, key):
        self.root = self.splay(self.root, key)
```

```
        if self.root and self.root.key == key:
            print("FOUND")
        else:
            print("NOTFOUND")
    def delete(self, key):
        if self.root is None:
            return
        self.root = self.splay(self.root, key)
        if key != self.root.key:
            return
        if self.root.left is None:
            self.root = self.root.right
        else:
            temp = self.root.right
            self.root = self.splay(self.root.left, key)
            self.root.right = temp
    def preorder(self, root):
        if root:
            print(root.key, end=" ")
            self.preorder(root.left)
            self.preorder(root.right)
tree = SplayTree()
files = [50, 30, 70, 20, 40, 60, 80]
for f in files:
    tree.insert(f)
tree.search(40)
tree.search(60)
tree.delete(30)
print("Tree structure (Preorder):")
tree.preorder(tree.root)

```

Output:

```
PS C:\Users\harsh\OneDrive\Desktop\CP> & C:\Users\harsh\Ap
thon314\python.exe c:/Users/harsh/OneDrive/Desktop/CP/ass1
FOUND
FOUND
Tree structure (Preorder):
20 40 60 50 70 80
```