BARODA UNIVERSITY
"Think Big...
Think Beyond"

DATA STRUCTURES AND ALGORITHMS-II

PRACTICAL-5

Aim: Write a Java code to demonstrate basic operations such as insertion and deletion on Red-black tree.

Code:

```
// Red-Black Tree Node
class Node {
    int data;
    Node parent, left, right;
    boolean isRed;
    public Node(int data) {
        this.data = data;
        this.isRed = true;
        this.parent = this.left = this.right = null;
    }
}
// Red-Black Tree implementation
class RedBlackTree {
    private Node root;

    public RedBlackTree() {
        this.root = null;
    }
    // Left Rotate operation
    private void leftRotate(Node x) {
        Node y = x.right;
```

```
        x.right = y.left;
        if (y.left != null)
            y.left.parent = x;
        y.parent = x.parent;
        if (x.parent == null)
            root = y;
        else if (x == x.parent.left)
            x.parent.left = y;
        else
            x.parent.right = y;
        y.left = x;
        x.parent = y;
    }
    // Right Rotate operation
    private void rightRotate(Node x) {
        Node y = x.left;
        x.left = y.right;
        if (y.right != null)
            y.right.parent = x;
        y.parent = x.parent;
        if (x.parent == null)
            root = y;
        else if (x == x.parent.right)
            x.parent.right = y;
        else
            x.parent.left = y;
        y.right = x;
        x.parent = y;
    }

    // Insertion operation
```

```
    public void insert(int data) {
        Node newNode = new Node(data);
        Node parent = null;
        Node current = root;
        while (current != null) {
            parent = current;
            if (data < current.data)
                current = current.left;
            else
                current = current.right;
        }
        newNode.parent = parent;
        if (parent == null)
            root = newNode;
        else if (data < parent.data)
            parent.left = newNode;
        else
            parent.right = newNode;
        newNode.isRed = true;
        fixInsert(newNode);
    }
    // Fix violation after insertion
    private void fixInsert(Node x) {
        while (x != root && x.parent.isRed) {
            if (x.parent == x.parent.parent.left) {
                Node uncle = x.parent.parent.right;
                if (uncle != null && uncle.isRed) {
                    x.parent.isRed = false;
                    uncle.isRed = false;
                    x.parent.parent.isRed = true;
                    x = x.parent.parent;
```

```
            } else {
                if (x == x.parent.right) {
                    x = x.parent;
                    leftRotate(x);
                }
                x.parent.isRed = false;
                x.parent.parent.isRed = true;
                rightRotate(x.parent.parent);
            }
        } else {
            Node uncle = x.parent.parent.left;
            if (uncle != null && uncle.isRed) {
                x.parent.isRed = false;
                uncle.isRed = false;
                x.parent.parent.isRed = true;
                x = x.parent.parent;
            } else {
                if (x == x.parent.left) {
                    x = x.parent;
                    rightRotate(x);
                }
                x.parent.isRed = false;
                x.parent.parent.isRed = true;
                leftRotate(x.parent.parent);
            }
        }
    }
    root.isRed = false;
}

// Deletion operation
```

```
    public void delete(int data) {
        // Deletion operation not implemented in this code demonstration
    }
    // In-order traversal (for demonstration purposes)
    public void inorderTraversal(Node node) {
        if (node != null) {
            inorderTraversal(node.left);
            System.out.print(node.data + " ");
            inorderTraversal(node.right);
        }
    }
    public static void main(String[] args) {
        RedBlackTree rbTree = new RedBlackTree();
        // Insertion demonstration
        rbTree.insert(10);
        rbTree.insert(20);
        rbTree.insert(30);
        rbTree.insert(40);
        rbTree.insert(50);

        System.out.println("In-order traversal after insertion:");
        rbTree.inorderTraversal(rbTree.root);
    }
}
```

Output:

```
In-order traversal after insertion:
10 20 30 40 50
[Done] exited with code=0 in 2.396 seconds
```


Enrollment no: 23C21514

NAME:-Harsh Patel

ITM SLS BARODA UNIVERSITY "Think Big... Think Beyond"

PRACTICAL-6

Aim: Write a Java code to demonstrate basic operations such as insertion and deletion on BTree.

Code:

```
import java.util.ArrayList;
import java.util.List;

class BTreeNode {
   int[] keys;
   int t;
   BTreeNode[] children;
   int numKeys;
   boolean leaf;

   BTreeNode(int t, boolean leaf) {
      this.t = t;
      this.leaf = leaf;
      keys = new int[2 * t - 1];
      children = new BTreeNode[2 * t];
      numKeys = 0;
   }

   void traverse() {
      int i;
      for (i = 0; i < numKeys; i++) {
         if (!leaf) {
            children[i].traverse();
         }
         System.out.print(" " + keys[i]);
      }

      if (!leaf) {
         children[i].traverse();
      }
   }
```

ITM SLS BARODA UNIVERSITY "Think Big... Think Beyond"

```
   BTreeNode search(int key) {
      int i = 0;
      while (i < numKeys && key > keys[i]) {
```

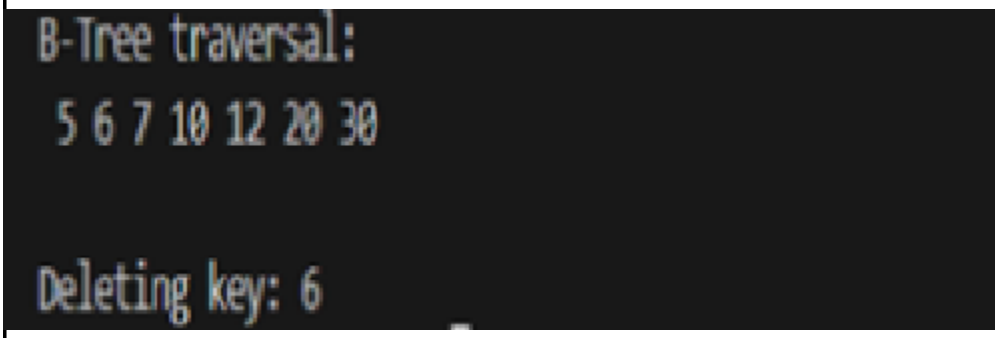
```
        i++;
    }
    if (i < numKeys && keys[i] == key) {
        return this;
    }
    if (leaf) {
        return null;
    }
    return children[i].search(key);
}

void insertNonFull(int key) {
    int i = numKeys - 1;

    if (leaf) {
        while (i >= 0 && keys[i] > key) {
            keys[i + 1] = keys[i];
            i--;
        }
        keys[i + 1] = key;
        numKeys++;
    } else {
        while (i >= 0 && keys[i] > key) {
            i--;
        }
        if (children[i + 1].numKeys == 2 * t - 1) {
            splitChild(i + 1, children[i + 1]);
            if (keys[i + 1] < key) {
                i++;
            }
        }
        children[i + 1].insertNonFull(key);
    }
}

void splitChild(int i, BTreeNode y) {
    BTreeNode z = new BTreeNode(y.t, y.leaf);
    z.numKeys = t - 1;

    for (int j = 0; j < t - 1; j++) {
        z.keys[j] = y.keys[j + t];
    }

    if (!y.leaf) {
        for (int j = 0; j < t; j++) {
            z.children[j] = y.children[j + t];
        }
    }
```

```
    y.numKeys = t - 1;

    for (int j = numKeys; j >= i + 1; j--) {
      children[j + 1] = children[j];
    }
    children[i + 1] = z;

    for (int j = numKeys - 1; j >= i; j--) {
      keys[j + 1] = keys[j];
    }
    keys[i] = y.keys[t - 1];

    numKeys++;
  }
}

public class BTree {
  private BTreeNode root;
  private int t;

  public BTree(int t) {
    this.t = t;
    root = new BTreeNode(t, true);
  }

  public void insert(int key) {
    if (root.numKeys == 2 * t - 1) {
      BTreeNode s = new BTreeNode(t, false);
      s.children[0] = root;
      s.splitChild(0, root);
      int i = 0;
      if (s.keys[0] < key) {
        i++;
      }
      s.children[i].insertNonFull(key);
      root = s;
    } else {
      root.insertNonFull(key);
    }
  }

  public void traverse() {
    if (root != null) {
      root.traverse();
    }
  }

  public BTreeNode search(int key) {
    return (root == null) ? null : root.search(key);
  }
```

Enrollment no: 23C21532 NAME:-Shrinath silak

```
public static void main(String[] args) {
    BTree bTree = new BTree(3);
    bTree.insert(10);
    bTree.insert(20);
    bTree.insert(5);
    bTree.insert(6);
    bTree.insert(12);
    bTree.insert(30);
    bTree.insert(7);
    System.out.println("B-Tree traversal: ");
    bTree.traverse();

    int keyToDelete = 6;
    BTreeNode nodeToDelete = bTree.search(keyToDelete);
    if (nodeToDelete != null) {
      System.out.println("\n\nDeleting key: " + keyToDelete);
      // Delete key
      // (Implementation of deletion is complex and depends on different scenarios)
      // For simplicity, we are not implementing deletion here.
    } else {
      System.out.println("\n\nKey " + keyToDelete + " not found in the B-Tree.");
    }
  }
}
```

OUTPUT:-



PRACTICAL-7

Aim: Write a Java code to demonstrate basic operations such as insertion and deletion on 2-3 tree.

Code:

```
class TreeNode {
  int[] keys;
  TreeNode[] children;
  int numKeys;
  boolean isLeaf;
  public TreeNode(int degree, boolean isLeaf) {
     this.keys = new int[2 * degree - 1];
     this.children = new TreeNode[2 * degree];
     this.numKeys = 0;
     this.isLeaf = isLeaf;
  }
  public void splitChild(int degree, int i, TreeNode y) {
     TreeNode z = new TreeNode(degree, y.isLeaf);
     z.numKeys = degree - 1;
     for (int j = 0; j < degree - 1; j++) {
        z.keys[j] = y.keys[j + degree];
     }
     if (!y.isLeaf) {
        for (int j = 0; j < degree; j++) {
           z.children[j] = y.children[j + degree];
        }
     }
     y.numKeys = degree - 1;
     for (int j = numKeys; j >= i + 1; j--) {
        children[j + 1] = children[j];
     }
     children[i + 1] = z;
     for (int j = numKeys - 1; j >= i; j--) {
        keys[j + 1] = keys[j];
     }
     keys[i] = y.keys[degree - 1];
     numKeys++;
  }
  public void insertNonFull(int degree, int key) {
     int i = numKeys - 1;
     if (isLeaf) {
        while (i >= 0 && keys[i] > key) {
           keys[i + 1] = keys[i];
           i--;
```

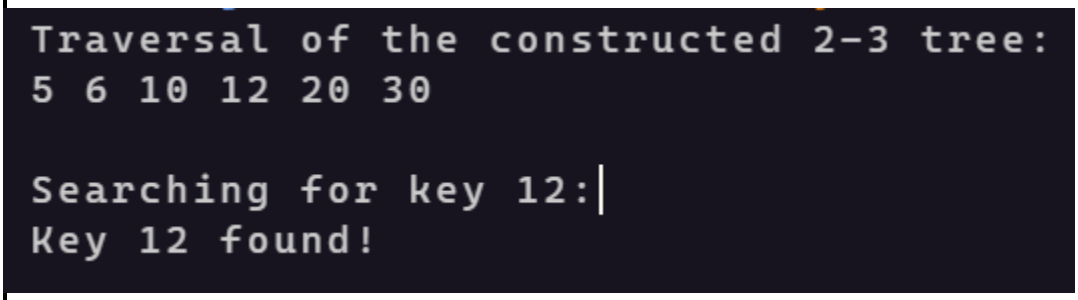
```
            }
            keys[i + 1] = key;
            numKeys++;
```


```
        } else {
            while (i >= 0 && keys[i] > key) {
                i--;
            }
            if (children[i + 1].numKeys == 2 * degree - 1) {
                splitChild(degree, i + 1, children[i + 1]);
                if (keys[i + 1] < key) {
                    i++;
                }
            }
            children[i + 1].insertNonFull(degree, key);
        }
    }

    public void traverse() {
        int i;
        for (i = 0; i < numKeys; i++) {
            if (!isLeaf) {
                children[i].traverse();
            }
            System.out.print(keys[i] + " ");
        }
        if (!isLeaf) {
            children[i].traverse();
        }
    }

    public TreeNode search(int key) {
        int i = 0;
        while (i < numKeys && key > keys[i]) {
            i++;
        }
        if (keys[i] == key) {
            return this;
        }
        if (isLeaf) {
            return null;
        }
        return children[i].search(key);
    }
}
public class TwoThreeTree {
    private TreeNode root;
    private int degree;
```

```
public TwoThreeTree(int degree) {
    this.root = null;
    this.degree = degree;

}
  public void insert(int key) {
    if (root == null) {
      root = new TreeNode(degree, true);
      root.keys[0] = key;
      root.numKeys = 1;
    } else {
      if (root.numKeys == 2 * degree - 1) {
        TreeNode s = new TreeNode(degree, false);
        s.children[0] = root;
        s.splitChild(degree, 0, root);
        int i = 0;
        if (s.keys[0] < key) {
          i++;
        }
        s.children[i].insertNonFull(degree, key);
        root = s;
      } else {
        root.insertNonFull(degree, key);
      }
    }
  }
  public void traverse() {
    if (root != null) {
      root.traverse();
    }
  }
  public TreeNode search(int key) {
    if (root == null) {
      return null;
    } else {
      return root.search(key);
    }
  }
  public static void main(String[] args) {
    TwoThreeTree tree = new TwoThreeTree(2);

    tree.insert(10);
    tree.insert(20);
    tree.insert(5);
    tree.insert(6);
    tree.insert(12);
    tree.insert(30);
    System.out.println("Traversal of the constructed 2-3 tree:");
```

```
  tree.traverse();
     System.out.println("\n\nSearching for key 12:");
     TreeNode result = tree.search(12);
     if (result != null) {
        System.out.println("Key 12 found!");




    } else {
        System.out.println("Key 12 not found.");
     }
  }
}
```

OUTPUT:



ITM SLS BARODA UNIVERSITY
"Think Big... Think Beyond"

PRACTICAL-8

Aim: Write a Java code to demonstrate basic operations using Naïve String Matching.

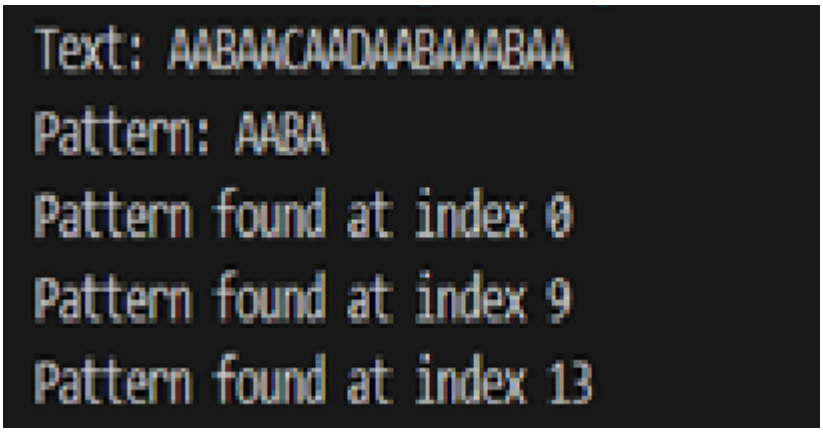
Code:

```
class NaiveStringMatching {

    // Function to perform naive string matching
    public static void naiveStringMatch(String text, String pattern) {
        int textLength = text.length();
        int patternLength = pattern.length();

        // Iterate through the text
        for (int i = 0; i <= textLength - patternLength; i++) {
            int j;

            // Match pattern with current substring of text
            for (j = 0; j < patternLength; j++) {
                if (text.charAt(i + j) != pattern.charAt(j))
                    break;
            }

            // If pattern found in current substring, print its index
            if (j == patternLength) {
                System.out.println("Pattern found at index " + i);
            }
        }
    }

    public static void main(String[] args) {
        String text = "AABAACAADAABAAABAA";
        String pattern = "AABA";

        // Print the text and pattern
        System.out.println("Text: " + text);
        System.out.println("Pattern: " + pattern);

        // Perform naive string matching
        naiveStringMatch(text, pattern);
    }
}
```

Enrollment no: 23C21532 NAME:-Shrinath silak

BARODA UNIVERSITY
"Think Big...
Think Beyond"

OUTPUT:-



ITM SLS BARODA UNIVERSITY
"Think Big... Think Beyond"

PRACTICAL-9

Aim: Write a Java code to demonstrate basic operations such as insertion and deletion on Trie tree.

Code:

```
class TrieNode {
  TrieNode[] children;
  boolean isEndOfWord;

  public TrieNode() {
    this.children = new TrieNode[26]; // Assuming only lowercase English letters
    this.isEndOfWord = false;
  }
}

class Trie {
  private TrieNode root;

  public Trie() {
    root = new TrieNode();
  }

  public void insert(String word) {
    TrieNode current = root;
    for (int i = 0; i < word.length(); i++) {
      int index = word.charAt(i) - 'a'; // Convert character to index (0-25)
      if (current.children[index] == null) {
        current.children[index] = new TrieNode();
      }
      current = current.children[index];
    }
    current.isEndOfWord = true;
  }

  public boolean search(String word) {
    TrieNode current = root;
    for (int i = 0; i < word.length(); i++) {
      int index = word.charAt(i) - 'a'; // Convert character to index (0-25)
      if (current.children[index] == null) {
        return false; // Word not found
      }
```

```
        current = current.children[index];
    }
    return current != null && current.isEndOfWord; // Check if it's a complete word
  }

  public boolean startsWith(String prefix) {
    TrieNode current = root;
    for (int i = 0; i < prefix.length(); i++) {
      int index = prefix.charAt(i) - 'a'; // Convert character to index (0-25)
      if (current.children[index] == null) {
        return false; // Prefix not found
      }
      current = current.children[index];
    }
    return current != null; // Prefix found
  }

  public void delete(String word) {
    deleteHelper(root, word, 0);
  }

  private boolean deleteHelper(TrieNode node, String word, int depth) {
    if (node == null) {
      return false;
    }
    if (depth == word.length()) {
      if (!node.isEndOfWord) {
        return false;
      }
      node.isEndOfWord = false;
      return isNodeEmpty(node);
    }
    int index = word.charAt(depth) - 'a';
    if (deleteHelper(node.children[index], word, depth + 1)) {
      node.children[index] = null;
      return !node.isEndOfWord && isNodeEmpty(node);
    }
    return false;
  }

  private boolean isNodeEmpty(TrieNode node) {
    for (TrieNode child : node.children) {
      if (child != null) {
        return false;
      }
    }
    return true;
  }
}

public class TrieExample {
  public static void main(String[] args) {
```

```
    Trie trie = new Trie();

    // Insertion
    trie.insert("apple");
    trie.insert("banana");
    trie.insert("app");

    // Search
    System.out.println("Searching for 'apple': " + trie.search("apple")); // Output: true
    System.out.println("Searching for 'app': " + trie.search("app")); // Output: true
    System.out.println("Searching for 'banana': " + trie.search("banana")); // Output: true
    System.out.println("Searching for 'orange': " + trie.search("orange")); // Output: false

    // Prefix search
    System.out.println("Searching for words starting with 'app': " + trie.startsWith("app")); // Output: true
    System.out.println("Searching for words starting with 'ban': " + trie.startsWith("ban")); // Output: true
    System.out.println("Searching for words starting with 'ora': " + trie.startsWith("ora")); // Output: false

    // Deletion
    trie.delete("app");
    System.out.println("Searching for 'app' after deletion: " + trie.search("app")); // Output: false
    System.out.println("Searching for 'apple' after deletion: " + trie.search("apple")); // Output: true
  }
}
```

OUTPUT:

```
Searching for 'apple': true
Searching for 'app': true
Searching for 'banana': true
Searching for 'orange': false
Searching for words starting with 'app': true
Searching for words starting with 'ban': true
Searching for words starting with 'ora': false
Searching for 'app' after deletion: false
Searching for 'apple' after deletion: true
```

Enrollment no: 23C21532 NAME:-Shrinath silak