```
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.TimeUnit;

public class BST {


    public static class Node {
        //instance variable of Node class
        public Substring data;
        public Node left;
        public Node right;

        //constructor
        public Node(Substring data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }

    // instance variable
    public Node root;


    public BST() {
        this.root = null;
    }

    public static Random randomNum = new Random();

    // insert method to insert the new Data
    public void insert(Substring substring) {
        this.root = insert(root, substring);
    }

    public Node insert(Node root, Substring newData) {
        // Base Case: root is null or not
        if (root == null) {
            // Insert the new data, if root is null.
            root = new Node(newData);
            // return the current root to his sub tree
            return root;
        }
        // Here checking for root data is greater or equal to newData or not
        else if ((root.data.substring.compareTo(newData.substring) > 0)) {
            // if current root data is greater than the new data then now process
the left sub-tree
            root.left = insert(root.left, newData);
        }
        else if ((root.data.substring.compareTo(newData.substring) < 0)) {
            // if current root data is less than the new data then now process the
right sub-tree
            root.right = insert(root.right, newData);
        }
        else{
            root.data.occurrences += 1;
        }
        return root;
```

```
    }


    public void deleteANode(Node node) {
        deleteNode(this.root, node);
    }

    private Node deleteNode(Node root, Node node) {
        // check for node initially
        if (root == null) {
            return null;
        } else if ((node.data.substring.compareTo(root.data.substring) <
0)/*node.data.length() < root.data.length()*/) {
            // process the left sub tree
            root.left = deleteNode(root.left, node);
        } else if ((node.data.substring.compareTo(root.data.substring) > 0)) {
            // process the right sub tree
            root.right = deleteNode(root.right, node);
        } else if(root.data==node.data){
            // case 3: 2 child
            if (root.left != null && root.right != null) {
                String lmax = findMaxData(root.left);
                Substring smax = new Substring(lmax);
                root.data.substring = lmax;
                root.left = deleteNode(root.left, new Node(smax));
                return root;
            }
            //case 2: one child
            // case i-> has only left child
            else if (root.left != null) {
                return root.left;
            }
            // case ii-> has only right child
            else if (root.right != null) {
                return root.right;
            }
            //case 1:- no child
            else {
                return null;
            }
        }
        return root;
    }

    // inorder successor of given node
    public String findMaxData(Node root) {
        if (root.right != null) {
            return findMaxData(root.right);
        } else {
            return root.data.substring;
        }
    }

    // calls for the search method
    public boolean search(String data) {
        return search(this.root, data);
    }

    //Searches through the BST
```

```
    private boolean search(Node root, String data) {
        if (root == null) {
            System.out.println(data + " does not exist");
            return false;
        } else if (root.data.substring.compareTo(data) == 0) {
            System.out.println(data + " exists with occurrence " +
root.data.occurrences);
            return true;
        } else if (root.data.substring.compareTo(data) > 0) {
            return search(root.left, data);
        }
        return search(root.right, data);
    }

    public void preorder(){
        preorder(root);
        System.out.println();
    }

    //prints the BST in preorder sequence
    public void preorder(Node node){
        if(node!=null){
            System.out.println(node.data.substring+"-"+node.data.occurrences);
            preorder(node.left);
            preorder(node.right);
        }
    }

    public void destroy(){
        destroy(root);
    }

    public void destroy(Node node){
        if(node!=null){
            destroy(node.left);
            deleteANode(new Node(node.data));
            destroy(node.right);
        }
    }

    public static String StringGenerator(Random rand, int nLength, int
subStringLength, int subStringLoopCount) {

        int num;
        int j;
        String stringInput = "";

        for(j = 0; j < nLength; j++) {
            num = rand.nextInt(4);
            if (num == 0)
                stringInput += 'a';

            if (num == 1)
                stringInput += 'c';

            if (num == 2)
                stringInput += 'g';

            if (num == 3)
```

```
                stringInput += 't';
        }
        System.out.println("The DNA Sequence: " + stringInput);

        return stringInput;
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter DNA Length: ");
        int dnaLength = input.nextInt();
        System.out.print("Enter Substring Lengths: ");
        int subStringLength = input.nextInt();
        int subStringLoopCount = dnaLength - subStringLength + 1;
        String DNAstring = StringGenerator(randomNum, dnaLength, subStringLength,
subStringLoopCount);
        // Creating the object of BinarySearchTree class
        BST bst = new BST();
        // call the method insert
        int i;
        long start = System.nanoTime();
        for(i = 0; i < subStringLoopCount; i++){
            Substring temp = new Substring(DNAstring.substring(i,
i+subStringLength));
            bst.insert(temp);
        }

        System.out.println("Binary Tree in preorder :");
        bst.preorder();
        long end = System.nanoTime();

        long convert = TimeUnit.MILLISECONDS.convert(end-start,
TimeUnit.NANOSECONDS);
        System.out.print("\nTime in miliseconds to solve K-mer distribution: ");
        System.out.println(convert);

        System.out.print("Search tree for a substring:");
        input.nextLine();
        String stringInput = input.nextLine();
        System.out.println(bst.search(stringInput));
        input.close();

        System.out.println("Binary Tree destroying");
        bst.destroy();

    }
}
```