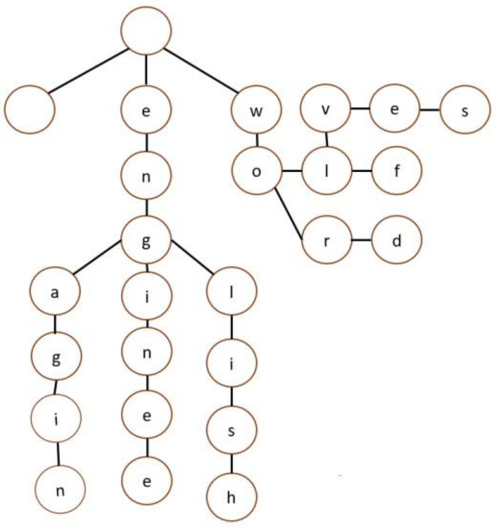
CMPE 343

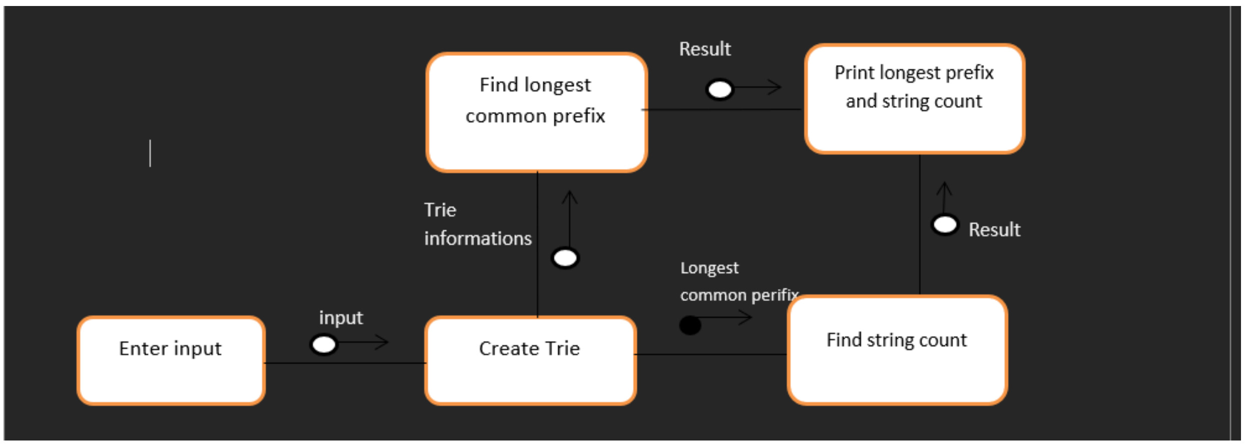
PROGRAMMING HOMEWORK-4 REPORT

Students Information	
Course	CMPE 343 – Data Structure and Algorithms-II
Name, Surname	Ahmet Berkay SARIASLAN – 54490639066 Zülal KARIN - 12622989076

Problem Statement and Code Design

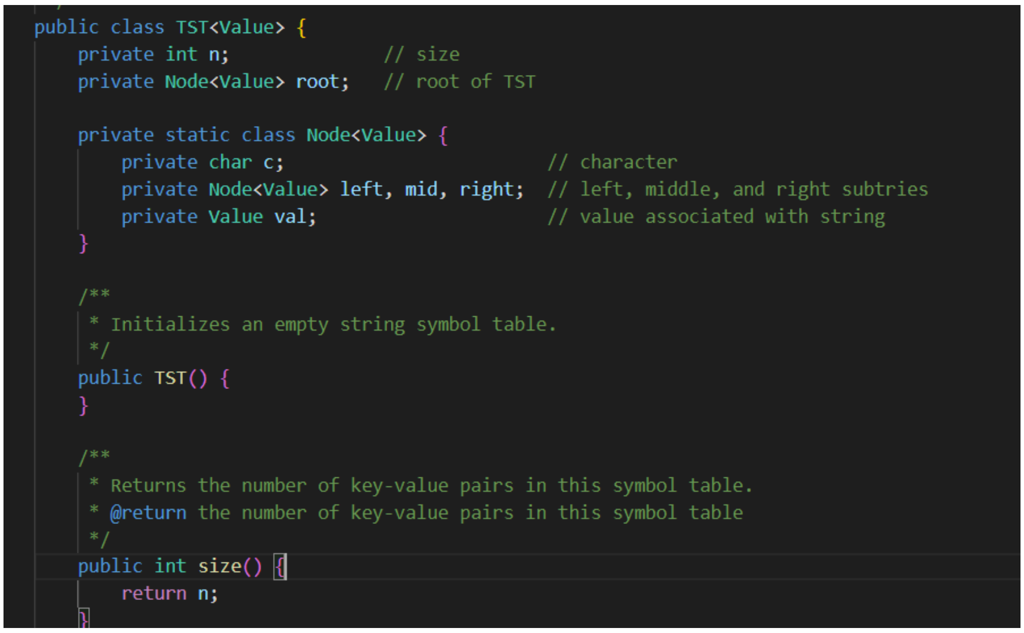
Task 1 : In this task, we need to interact with the argument number and get the largest number of XORs from the given 9-bit numbers requested from us. So, we kept the result of this interaction in the structure of Trie. For each number, we tried to print the number XOR, which is the largest after comparison.
Task 2 : In this task, we need to access the longest common prefix from the words given to us. As a priority for this, we have created our Trie structure. When using this Trie structure, we always kept the key values constant. Because we didn't use the "key" values of the words directly in this task. On the other hand, we also printed the number of longest common prefix that we found. In addition, we will share the Trie structure that we created ourselves before taking office in the continuation of the report.



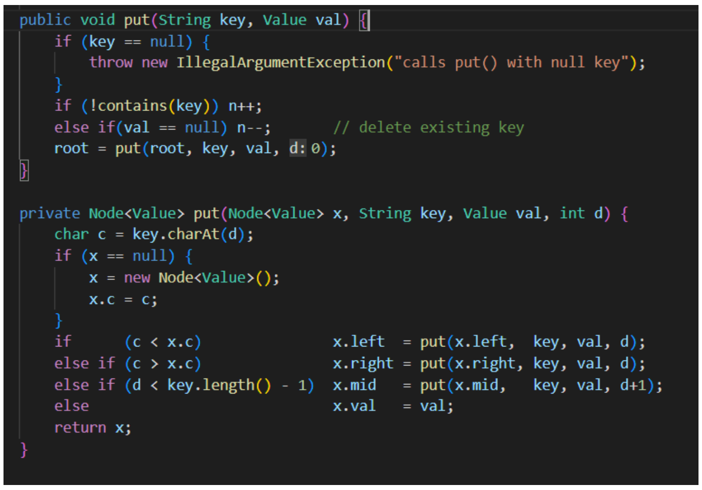


Part-2 structure diagram

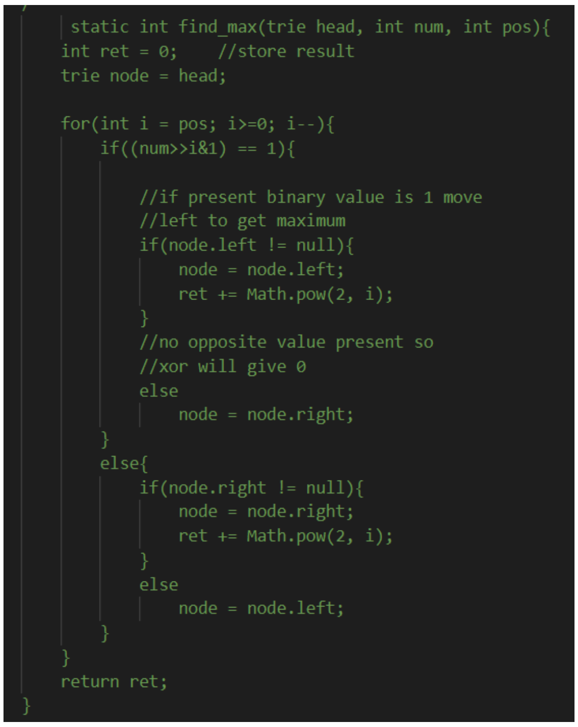
At this point, we set size to 2. Because our digit consists of 0 and 1.



```java
public class TST<Value> {
    private int n;              // size
    private Node<Value> root;   // root of TST

    private static class Node<Value> {
        private char c;                        // character
        private Node<Value> left, mid, right;  // left, middle, and right subtries
        private Value val;                     // value associated with string
    }

    /**
     * Initializes an empty string symbol table.
     */
    public TST() {
    }

    /**
     * Returns the number of key-value pairs in this symbol table.
     * @return the number of key-value pairs in this symbol table
     */
    public int size() {
        return n;
    }
```



```java
public void put(String key, Value val) {
    if (key == null) {
        throw new IllegalArgumentException("calls put() with null key");
    }
    if (!contains(key)) n++;
    else if(val == null) n--;       // delete existing key
    root = put(root, key, val, d: 0);
}

private Node<Value> put(Node<Value> x, String key, Value val, int d) {
    char c = key.charAt(d);
    if (x == null) {
        x = new Node<Value>();
        x.c = c;
    }
    if      (c < x.c)               x.left  = put(x.left,  key, val, d);
    else if (c > x.c)               x.right = put(x.right, key, val, d);
    else if (d < key.length() - 1)  x.mid   = put(x.mid,   key, val, d+1);
    else                            x.val   = val;
    return x;
}
```

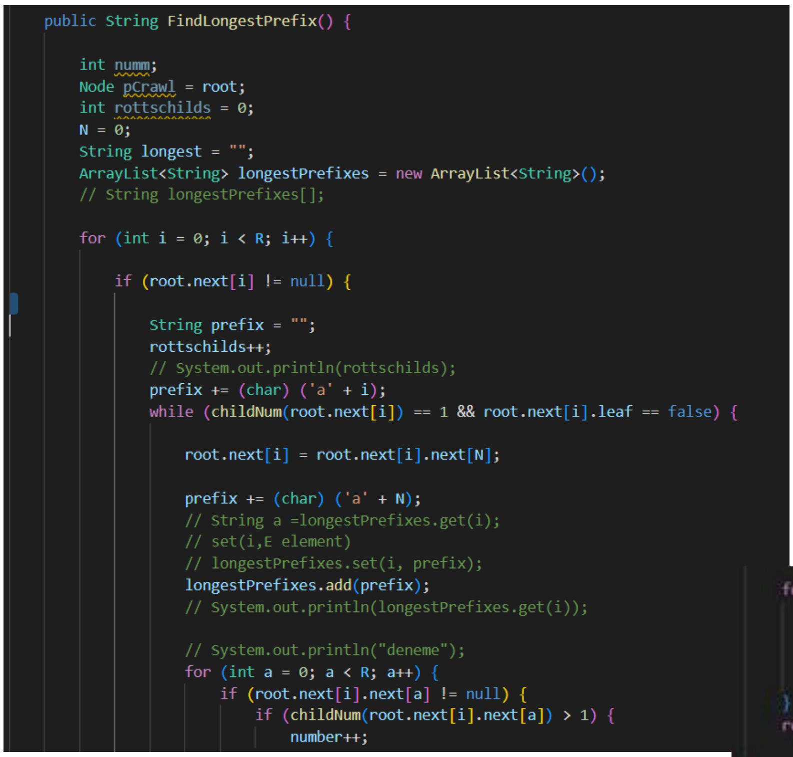


```java
static int find_max(trie head, int num, int pos){
    int ret = 0;    //store result
    trie node = head;

    for(int i = pos; i>=0; i--){
        if((num>>i&1) == 1){

            //if present binary value is 1 move
            //left to get maximum
            if(node.left != null){
                node = node.left;
                ret += Math.pow(2, i);
            }
            //no opposite value present so
            //xor will give 0
            else
                node = node.right;
        }
        else{
            if(node.right != null){
                node = node.right;
                ret += Math.pow(2, i);
            }
            else
                node = node.left;
        }
    }
    return ret;
}
```

At this point, we are editing and implementing Trie with the help of Symbol Table. We add data to our Trie structure thanks to the put method. Then, thanks to the maximum xor method, we process the given digit and add it to our Trie structure.

Part 2

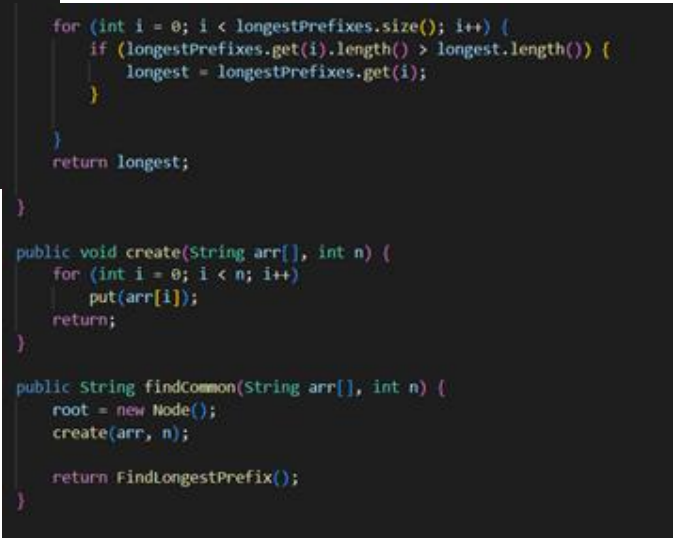
At this point, we quickly formed a Trie and immediately began operation. Also at this point, we have updated the size because we are dealing with Strings.



```java
public String FindLongestPrefix() {

    int numm;
    Node pCrawl = root;
    int rottschilds = 0;
    N = 0;
    String longest = "";
    ArrayList<String> longestPrefixes = new ArrayList<String>();
    // String longestPrefixes[];

    for (int i = 0; i < R; i++) {

        if (root.next[i] != null) {

            String prefix = "";
            rottschilds++;
            // System.out.println(rottschilds);
            prefix += (char) ('a' + i);
            while (childNum(root.next[i]) == 1 && root.next[i].leaf == false) {

                root.next[i] = root.next[i].next[N];

                prefix += (char) ('a' + N);
                // String a =longestPrefixes.get(i);
                // set(i,E element)
                // longestPrefixes.set(i, prefix);
                longestPrefixes.add(prefix);
                // System.out.println(longestPrefixes.get(i));

                // System.out.println("deneme");
                for (int a = 0; a < R; a++) {
                    if (root.next[i].next[a] != null) {
                        if (childNum(root.next[i].next[a]) > 1) {
                            number++;
```

First we find all the children of the root. Then we stop by each of the children we find one by one and continue to search on the loop until they have more than one child. And when we get to the root with more than one child, we add it to the string.

We compare the children we got after we got to the root. Thus, we can determine the longest string.



```java
    for (int i = 0; i < longestPrefixes.size(); i++) {
        if (longestPrefixes.get(i).length() > longest.length()) {
            longest = longestPrefixes.get(i);
        }

    }
    return longest;

}

public void create(String arr[], int n) {
    for (int i = 0; i < n; i++)
        put(arr[i]);
    return;
}

public String findCommon(String arr[], int n) {
    root = new Node();
    create(arr, n);

    return FindLongestPrefix();
}
```

Final Assessment

- First, we designed a Trie according to the inputs given to calculate the steps we will follow. We have provided checks such as whether each child has been visited with form cycles. Thanks to this task, we focused on Trie data structures.
- The most challenging for us part was getting to the children. We tried to reach children with different root numbers, and the this step forced us.
- In this task was an opportunity for us to closely observe the Trie data structure. We also remembered the conversion structures, such as the ASCII table.