Daily Practice Problem Date 11/12/2023

Problem 1: Prime Subtraction Operation

```
package javaPractic;

import java.util.ArrayList;
import java.util.Arrays;

public class PrimeSubOperation {   //class created

    static ArrayList<Integer> primes = new ArrayList<Integer>();

    public static void getPrimes(){  //constructor created

        boolean[] visited = new boolean[10001];   //object created

        for(int i = 2; i <= 10000; i++){   //loop iterating
            if(!visited[i]){
                visited[i] = true;
                primes.add(i);

                int j = i;
                while(j <= 10000){   //condition checking
                    visited[j] = true;
                    j += i;
                }
            }
        }

    }

    public static boolean primeSubOperation(int[] nums) {  //function created
        getPrimes();    //function calling

        for(int i = 0; i < nums.length; i++){  //loop iterating
            int prime = 0;
            if(i == 0){
                for(int j : primes){
                    if(j >= nums[i]){
                        break;
                    }

                    prime = j;
                }

                nums[i] -= prime;
            }else{                      // if(prime != 0)primes.remove((Integer)prime);
                for(int j : primes){
                    if(j >= nums[i]){
                        break;
                    }

                    if(nums[i-1] < (nums[i] - j)){
                        prime = j;
                    }
                }

                nums[i] -= prime;
                                    // if(prime != 0) primes.remove((Integer)prime);

            }
        }

        System.out.println(Arrays.toString(nums));

        for(int i = 1; i < nums.length; i++){   //loop iterating
```

```
            if(nums[i-1] >= nums[i]){
                return false;
            }
        }

        return true;  //return true
    }
    public static void main(String[] args) {   //main program created
            int num[] = {4,9,6,10};
            System.out.println(primeSubOperation(num));  //function calling
      }
}
```

Problem 2: Find K the permutation sequence

```
package javaPractic;
import java.util.*;

public class PermutationSequence {   //class created

static int findFirstNumIndex(int k, int n){   //function created
if (n == 1)
      return 0;
n--;  //decrement

int first_num_index;
int n_partial_fact = n;

while (k >= n_partial_fact && n > 1) {  //n_actual_fact = n!
      n_partial_fact = n_partial_fact * (n - 1);
      n--;
}

first_num_index = k / n_partial_fact;

k = k % n_partial_fact;
return first_num_index;
}
static String findKthPermutation(int n, int k){    //function created

String ans = "";   //Store final answer

HashSet<Integer> s = new HashSet<>();
for (int i = 1; i <= n; i++)
      s.add(i);

Vector<Integer> v = new Vector<>();
v.addAll(s);

int itr = v.elementAt(0);   //Mark the first position
k = k - 1;

for (int i = 0; i < n; i++) {    //loop iterating
      int index = findFirstNumIndex(k, n - i);

      if(index < v.size()) {    //condition checking
      ans += ((v.elementAt(index).toString()));
      v.remove(index);
      }
      else
      ans += String.valueOf(itr + 2);
      itr = v.elementAt(0);
}
return ans;
}
public static void main(String[] args){   //main program created
int n = 3, k = 4;
```

```
String kth_perm_seq = findKthPermutation(n, k);   //function calling
System.out.print(kth_perm_seq + "\n");
}
}
```

Problem 3: Single Number

```
package javaPractic;
public class SingleNumber {   //class created

    public static int singleNumber(int[] nums) {   //function created
        int result=0;
        for(int i=0; i<nums.length; i++) {   //loop iterating
            result = result^nums[i];
        }
        return result;
    }
    public static void main(String[] args) {   //main program created
            int[] nums= {2,2,1};
            System.out.println(singleNumber( nums));   //function calling
      }
}
```

Problem 4: Generate Parenthesis

```
package javaPractic;

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

public class GenerateParenthesis {    //class created
      public static void main(String[] args) {   //main program created
            int n = 3;
            System.out.println(generateParenthesis( n));   //function calling
      }
    static List<String> answer = new ArrayList();
    public static List<String> generateParenthesis(int n) {  //function created
        find("",n,n);
        return answer;
    }
    private static void find(String s, int a, int b) {  //function created
        if (a != 0) find(s + "(", a-1, b);
        if (b != 0 && a < b) find(s + ")", a, b-1);
        if (a == 0 && b == 0) answer.add(s);
    }

}
```

Problem 5: Combinations

```
package javaPractic;

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

public class Combinations {    //class created
    public static List<List<Integer>> combine(int n, int k) {   //function created
        List<List<Integer>> result = new ArrayList<List<Integer>>();
        if (k > n || k < 0) {    //condition checking
            return result;
        }
        if (k == 0) {   //condition checking
            result.add(new ArrayList<Integer>());
            return result;
        }
        result = combine(n - 1, k - 1);
        for (List<Integer> list : result) {    //loop created
```

```
            list.add(n);
        }
        result.addAll(combine(n - 1, k));  //calling function
        return result;
    }
    public static void main(String[] args) {   //main program created
            int k = 1;
            int n = 1;
            System.out.println(combine( n,  k));  //function calling
      }
}
```

Problem 6: Sum root to leaf numbers

```
package javaPractic;
import javax.swing.tree.TreeNode;

      class Node {   // A binary tree node
            int data;
            Node left, right;

            Node(int item)
            {
                  data = item;
                  left = right = null;
            }
      }

      public class SumNumberLeaf {
      private Node root;

      {
            Node root;
      }

            int treePathsSumUtil(Node node, int val)
            {

                  if (node == null)  // Base case
                        return 0;

                  val = (val * 10 + node.data);   // Update val
                  if (node.left == null && node.right == null)  // if current node is
leaf, return the current value of val
                        return val;

                  return treePathsSumUtil(node.left, val)+ treePathsSumUtil(node.right,
val);
            }

            int treePathsSum(Node node)   // A wrapper function over treePathsSumUtil()
            {
                  return treePathsSumUtil(node, 0);
            }
            public static void main(String args[]) {  //main program created
                  SumNumberLeaf tree = new SumNumberLeaf();
                  tree.root = new Node(1);
                  tree.root.left = new Node(2);
                  tree.root.right = new Node(3);


                  System.out.print("Sum of all paths is " +
tree.treePathsSum(tree.root));   //function calling
            }
      }
```

Problem 7:Evaluate reverse polish notation

```
package javaPractic;
import java.util.Stack;

public class ERPNotation {   //class created

    public static int evalRPN(String[] tokens) {  //function created
        Stack<Integer> st = new Stack<>();
        for(String t : tokens){   //loop
            if("+-*/".contains(t))   //checking condition
              st.push(eval(st.pop(), st.pop(), t));
            else
              st.push(Integer.parseInt(t));
        }
        return st.pop();
    }
     private static int eval(int b, int a, String op){   //function created
            if("+".equals(op))
            return a+b;
            else if("-".equals(op))
            return a-b;
            else if("*".equals(op))
            return a*b;
            else
            return a/b;    //return value
        }

     public static void main(String[] args) {    //main program created
            String[] tokens = {"2","1","+","3","*"};
            System.out.println(evalRPN( tokens));   //function calling
      }
}
```

Problem 8: Word break

```
package javaPractic;
import java.util.*;

public class WordBreak{   //class created

      private static Set<String> dictionary = new HashSet<>();    // set to hold
dictionary values

      public static void main(String []args){   //main program created

            String temp_dictionary[] = {"mobile","samsung","sam","sung",
"man","mango","icecream","and", "go","i","like","ice","cream"};

            for (String temp :temp_dictionary){   // loop to add all strings in
dictionary set
                  dictionary.add(temp);
            }

            System.out.println(wordBreak("ilikesamsung"));

      }

      public static boolean wordBreak(String word){   //function created
            int size = word.length();

            if (size == 0)  // base case
            return true;

            for (int i = 1; i <= size; i++){   //else check for all words

                  if (dictionary.contains(word.substring(0,i)) &&
```

```
                                wordBreak(word.substring(i,size)))
                        return true;
                }
                        // if all cases failed then return false
                return false;
        }
}
```

Program 9 : Right Side View

```
package javaPractic;

class Node {    //A binary tree node

        int data;
        Node left, right;

        Node(int item)
        {
                data = item;
                left = right = null;
        }
}
class Max_level {  //class to access maximum level by reference
        int max_level;
}
public class RightSideView {  //class created
        Node root;
        Max_level max = new Max_level();

        void rightViewUtil(Node node, int level,Max_level max_level){

                if (node == null)    // Base Case
                        return;
                if (max_level.max_level < level) {    // If this is the last Node of its
level
                        System.out.print(node.data + " ");
                        max_level.max_level = level;
                }
                rightViewUtil(node.right, level + 1, max_level);
                rightViewUtil(node.left, level + 1, max_level);
        }

        void rightView() { rightView(root); }

        void rightView(Node node){  // A wrapper over rightViewUtil()
                rightViewUtil(node, 1, max);
        }
        public static void main(String args[]){   //main program created
                RightSideView tree = new RightSideView();
                tree.root = new Node(1);
                tree.root.left = new Node(2);
                tree.root.right = new Node(3);
                tree.root.left.left = new Node(4);
                tree.root.left.right = new Node(5);
                tree.root.right.left = new Node(6);
                tree.root.right.right = new Node(7);
                tree.root.right.left.right = new Node(8);

                tree.rightView(); // function calling
        }
}
```