\*MOST FREQUENTLY OCCURRING DIGIT

import java.io.\*;

import java.util.\*;

class MostFrequentlyOccurringDigit {

public int mostFrequentlyOccurringDigit(int[] input1,int input2){

StringBuilder input = new StringBuilder();

for (int ip : input1) input.append(ip);

int[] freq = new int[10];

for (int j = 0; j < input.length(); j++) {

freq[Integer.parseInt(String.valueOf(input.charAt(j)))]++;

}

int maxFreqIndex = 0;

int maxFreq = 0;

for (int i = 9; i >= 0; i--) {

if (freq[i] > maxFreq) {

maxFreqIndex = i;

maxFreq = freq[i];

}

}

return maxFreqIndex;

}

}

\*DECREASING SEQUENCE

import java.io.\*;

import java.util.\*;

class DecreasingSequence {

public class Result{

public final int output1;

public final int output2;

public Result(int out1, int out2){

output1 = out1;

output2 = out2;

}

}

public Result decreasingSeq(int[] input1,int input2){

int dcrCount = 0;

int longestLen = 0;

int spikeCount = 0;

boolean flag = false;

for (int i = 0; i < input2 - 1; i++) {

if (input1[i] > input1[i + 1]) {

if (flag == false) {

flag = true;

spikeCount++;

}

dcrCount++;

longestLen = dcrCount > longestLen ? dcrCount : longestLen;

} else {

if (flag == true) {

flag = false;

dcrCount = 0;

}

}

}

if (spikeCount > 0) longestLen++;

return new Result(spikeCount, longestLen);

}

}

\*SIMPLE ENCODED ARRAY

import java.io.\*;

import java.util.\*;

class SimpleEncodedArray {

public class Result{

public final int output1;

public final int output2;

public Result(int out1, int out2){

output1 = out1;

output2 = out2;

}

}

public Result findOriginalFirstAndSum(int[] input1,int input2){

int[] out = new int[input1.length];

out[out.length - 1] = input1[input1.length - 1];

for (int i = input1.length - 1; i > 0; i--) {

out[i - 1] = input1[i - 1] - out[i];

}

int sum = 0;

for (int item : out)

sum += item;

return new Result(out[0], sum);

}

}

\*SUM OF DIGITS IN CYCLE

import java.io.\*;

import java.util.\*;

class SumOfSumsOfDigitsInCyclicOrder {

public int sumOfSumsOfDigits(int input1){

String num = String.valueOf(input1);

int sum = 0;

for (int i = 0; i < num.length(); i++) {

for (int j = i; j < num.length(); j++) {

sum += Integer.parseInt(String.valueOf(num.charAt(j)));

}

}

return sum;

}

}

\*SUM OF POWER OF DIGITS

import java.io.\*;

import java.util.\*;

class SumOfSumsOfDigitsInCyclicOrder {

public int sumOfSumsOfDigits(int input1){

String num = String.valueOf(input1);

int sum = 0;

for (int i = 0; i < num.length(); i++) {

for (int j = i; j < num.length(); j++) {

sum += Integer.parseInt(String.valueOf(num.charAt(j)));

}

}

return sum;

}

}

\*ENCODING THREE STRINGS

import java.io.\*;

import java.util.\*;

class EncodingThreeStrings {

public class Result{

public final String output1;

public final String output2;

public final String output3;

public Result(String out1, String out2, String out3){

output1 = out1;

output2 = out2;

output3 = out3;

}

}

public Result encodeThreeStrings(String input1,String input2,String input3){

String[] ip1parts = new String[3];

String[] ip2parts = new String[3];

String[] ip3parts = new String[3];

ip1parts = getParts(input1);

ip2parts = getParts(input2);

ip3parts = getParts(input3);

StringBuilder output1 = new StringBuilder (ip1parts[0] + ip2parts[0] + ip3parts[0]);

StringBuilder output2 = new StringBuilder (ip1parts[1] + ip2parts[1] + ip3parts[1]);

StringBuilder output3 = new StringBuilder (ip1parts[2] + ip2parts[2] + ip3parts[2]);

for (int i = 0; i < output3.length(); i++) {

if (Character.isLowerCase(output3.charAt(i)))

output3.setCharAt(i, Character.toUpperCase(output3.charAt(i)));

else

output3.setCharAt(i, Character.toLowerCase(output3.charAt(i)));

}

return new Result(output1.toString(), output2.toString(), output3.toString());

}

public static String[] getParts(String str) {

int len = str.length();

String[] parts = new String[3];

int partLen = len / 3;

if (len % 3 == 0) {

parts[0] = str.substring(0, partLen);

parts[1] = str.substring(partLen, 2 \* partLen);

parts[2] = str.substring(2 \* partLen, len);

} else if (len % 3 == 1) {

parts[0] = str.substring(0, partLen);

parts[1] = str.substring(partLen, 2 \* partLen + 1);

parts[2] = str.substring(2 \* partLen + 1, len);

} else if (len % 3 == 2) {

parts[0] = str.substring(0, partLen + 1);

parts[1] = str.substring(partLen + 1, 2 \* partLen + 1);

parts[2] = str.substring(2 \* partLen + 1, len);

}

return parts;

}

}

\*IDENTIFY POSSIBLE WORDS

import java.io.\*;

import java.util.\*;

class IdentifyPossibleWords {

public String identifyPossibleWords(String input1,String input2){

input1 = input1.toUpperCase();

StringBuffer output = new StringBuffer();

String[] words = input2.split(":");

int underscoreIndex = input1.indexOf('\_');

for (int i = 0; i < words.length; i++) {

words[i] = words[i].toUpperCase();

if (words[i].length() >= input1.length() &&

input1.replace('\_', words[i].charAt(underscoreIndex)).equals(words[i])) {

output.append(words[i]).append(":");

}

}

if (output.length() == 0) return "ERROR-009";

else return output.toString().substring(0, output.length() - 1);

}

}