Q1.public class Solution {

public boolean isIsomorphic(String s, String t) {

if (s.length() != t.length()) {

return false;

}

Map<Character, Character> mapping = new HashMap<>();

Set<Character> mappedChars = new HashSet<>();

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

char charS = s.charAt(i);

char charT = t.charAt(i);

if (mapping.containsKey(charS)) {

if (mapping.get(charS) != charT) {

return false;

}

} else {

if (mappedChars.contains(charT)) {

return false;

}

mapping.put(charS, charT);

mappedChars.add(charT);

}

}

return true;

}

}

Q2.public class Solution {

public boolean isStrobogrammatic(String num) {

int left = 0;

int right = num.length() - 1;

while (left <= right) {

char c1 = num.charAt(left);

char c2 = num.charAt(right);

if (!isStrobogrammaticPair(c1, c2)) {

return false;

}

left++;

right--;

}

return true;

}

private boolean isStrobogrammaticPair(char c1, char c2) {

// Check if the pair of characters is a valid strobogrammatic pair

// 0 and 0, 1 and 1, 6 and 9, 8 and 8, 9 and 6 are valid pairs

return (c1 == '0' && c2 == '0') ||

(c1 == '1' && c2 == '1') ||

(c1 == '6' && c2 == '9') ||

(c1 == '8' && c2 == '8') ||

(c1 == '9' && c2 == '6');

}

}

Q3.public class Solution {

public String addStrings(String num1, String num2) {

StringBuilder result = new StringBuilder();

int carry = 0;

int i = num1.length() - 1;

int j = num2.length() - 1;

while (i >= 0 || j >= 0 || carry > 0) {

int digit1 = i >= 0 ? num1.charAt(i) - '0' : 0;

int digit2 = j >= 0 ? num2.charAt(j) - '0' : 0;

int sum = digit1 + digit2 + carry;

carry = sum / 10;

int digit = sum % 10;

result.insert(0, digit);

i--;

j--;

}

return result.toString();

}

}

Q4.public class Solution {

public String reverseWords(String s) {

String[] words = s.split(" ");

StringBuilder result = new StringBuilder();

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

String reversedWord = reverseString(words[i]);

result.append(reversedWord);

if (i < words.length - 1) {

result.append(" ");

}

}

return result.toString();

}

private String reverseString(String s) {

char[] chars = s.toCharArray();

int i = 0;

int j = s.length() - 1;

while (i < j) {

char temp = chars[i];

chars[i] = chars[j];

chars[j] = temp;

i++;

j--;

}

return new String(chars);

}

}

Q5.public class Solution {

public String reverseStr(String s, int k) {

char[] chars = s.toCharArray();

int n = chars.length;

for (int i = 0; i < n; i += 2 \* k) {

int left = i;

int right = Math.min(i + k - 1, n - 1);

while (left < right) {

char temp = chars[left];

chars[left] = chars[right];

chars[right] = temp;

left++;

right--;

}

}

return new String(chars);

}

}

Q6.public class Solution {

public boolean rotateString(String s, String goal) {

if (s.length() != goal.length()) {

return false;

}

String rotated = s + s;

return rotated.contains(goal);

}

}

Q7.import java.util.Stack;

public class Solution {

public boolean backspaceCompare(String s, String t) {

return buildString(s).equals(buildString(t));

}

private String buildString(String str) {

Stack<Character> stack = new Stack<>();

for (char c : str.toCharArray()) {

if (c != '#') {

stack.push(c);

} else if (!stack.isEmpty()) {

stack.pop();

}

}

StringBuilder sb = new StringBuilder();

while (!stack.isEmpty()) {

sb.append(stack.pop());

}

return sb.toString();

}

}

Q8.public class Solution {

public boolean checkStraightLine(int[][] coordinates) {

int x0 = coordinates[0][0];

int y0 = coordinates[0][1];

int x1 = coordinates[1][0];

int y1 = coordinates[1][1];

for (int i = 2; i < coordinates.length; i++) {

int x = coordinates[i][0];

int y = coordinates[i][1];

if ((y1 - y0) \* (x - x0) != (y - y0) \* (x1 - x0)) {

return false;

}

}

return true;

}

}