**Assignment Questions and Answer 7**

💡 **Question 1**

Given two strings s and t, *determine if they are isomorphic*.

Two strings s and t are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

**Example 1:**

**Input:** s = "egg", t = "add"

**Output:** truepackage Assignment\_7;

public class IsomorphicStrings {

    static boolean isIsomorphic(String s, String t) {

        int map1[]=new int[200];

        int map2[]=new int[200];

        if(s.length()!=t.length())

            return false;

        for(int i=0;i<s.length();i++)

        {

            if(map1[s.charAt(i)]!=map2[t.charAt(i)])

                return false;

            map1[s.charAt(i)]=i+1;

            map2[t.charAt(i)]=i+1;

        }

        return true;

    }

    public static void main(String[] args) {

        String  s = "egg", t = "add";

        System.out.println(isIsomorphic(s, t));

    }

}

💡 **Question 2**

Given a string num which represents an integer, return true *if* num *is a* ***strobogrammatic number***.

A **strobogrammatic number** is a number that looks the same when rotated 180 degrees (looked at upside down).

**Example 1:**

**Input:** num = "69"

**Output:**

True

package Assignment\_7;

import java.util.\*;

public class StrobogrammaticNumber {

    static boolean isStrobogrammatic(String num) {

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

        map.put('6', '9');

        map.put('9', '6');

        map.put('0', '0');

        map.put('1', '1');

        map.put('8', '8');

        int l = 0, r = num.length() - 1;

        while (l <= r) {

            if (!map.containsKey(num.charAt(l))) return false;

            if (map.get(num.charAt(l)) != num.charAt(r))

                return false;

            l++;

            r--;

        }

        return true;

    }

    public static void main(String[] args) {

        String num = "69";

        System.out.println(isStrobogrammatic(num));

    }

}

💡 **Question 3**

Given two non-negative integers, num1 and num2 represented as string, return *the sum of* num1 *and* num2 *as a string*.

You must solve the problem without using any built-in library for handling large integers (such as BigInteger). You must also not convert the inputs to integers directly.

**Example 1:**

**Input:** num1 = "11", num2 = "123"

**Output:**

"134"

package Assignment\_7;

public class AddStrings {

    static String addStrings(String num1, String num2) {

        StringBuilder sb = new StringBuilder();

        int i = num1.length() - 1, j = num2.length() - 1;

        int carry = 0;

        while (i >= 0 || j >= 0) {

            int sum = carry;

            if (i >= 0) sum += (num1.charAt(i--) - '0');

            if (j >= 0) sum += (num2.charAt(j--) - '0');

            sb.append(sum % 10);

            carry = sum / 10;

        }

        if (carry != 0) sb.append(carry);

        return sb.reverse().toString();

    }

    public static void main(String[] args) {

        String  num1 = "11", num2 = "123";

        System.out.println(addStrings(num1, num2));

    }

}

💡 **Question 4**

Given a string s, reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

**Example 1:**

**Input:** s = "Let's take LeetCode contest"

**Output:** "s'teL ekat edoCteeL tsetnoc"

package Assignment\_7;

public class ReverseWordsInAStringIII {

    static String reverseWords(String s) {

        if (s == null || s.length() <= 1) {

            return s;

        }

        char[] str = s.toCharArray();

        int start = 0;

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

            if (str[i] == ' ') {

                reverse(str, start, i - 1);

                start = i + 1;

            } else if (i == str.length - 1) {

                reverse(str, start, i);

            }

        }

        return String.valueOf(str);

    }

    static void reverse(char[] s, int start, int end) {

        while (start < end) {

            char temp = s[start];

            s[start] = s[end];

            s[end] = temp;

            start++;

            end--;

        }

    }

    public static void main(String[] args) {

        String s = "Let's take LeetCode contest";

        System.out.println(reverseWords(s));

    }

}

💡 **Question 5**

Given a string s and an integer k, reverse the first k characters for every 2k characters counting from the start of the string.

If there are fewer than k characters left, reverse all of them. If there are less than 2k but greater than or equal to k characters, then reverse the first k characters and leave the other as original.

**Example 1:**

**Input:** s = "abcdefg", k = 2

**Output:**

"bacdfeg"

package Assignment\_7;

public class ReverseStringII {

    static String reverseStr(String s, int k) {

        int n = s.length();

        StringBuilder sb = new StringBuilder(n);

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

            // Creating the separator for every iteration.

            int j = Math.min(i + k, n);

            // Substring to reverse first k elements

            StringBuilder rev = new StringBuilder(s.substring(i, j));

            rev.reverse();

            sb.append(rev);

            // Substring to append the rest elements.

            int rem = Math.min(i + 2\*k, n);

            sb.append(s.substring(j, rem));

        }

        // Converting back the StringBuilder to String

        return sb.toString();

    }

    public static void main(String[] args) {

        String s = "abcdefg";

        int k = 2;

        System.out.println(reverseStr(s, k));

    }

}

💡 **Question 6**

Given two strings s and goal, return true *if and only if* s *can become* goal *after some number of* ***shifts*** *on* s.

A **shift** on s consists of moving the leftmost character of s to the rightmost position.

* For example, if s = "abcde", then it will be "bcdea" after one shift.

**Example 1:**

**Input:** s = "abcde", goal = "cdeab"

**Output:**

True

package Assignment\_7;

public class RotateString {

    static boolean rotateString(String s, String goal) {

        if(s ==null || goal == null){

            return false;

        }

        if(s.length() != goal.length()) return false;

        if(s.length() == 0){

            return true;

        }

        int i =0, j =0;

        while(i < s.length() && j < goal.length()){

            if(s.charAt(i) == goal.charAt(j)){

                i++; j++;

            }

            else{

                if(j == 0){

                    i++;

                }

                else{

                    j= 0;

                }

            }

        }

        return (s.substring(0,goal.length() - j).equals(goal.substring(j)));

    }

    public static void main(String[] args) {

        String s = "abcde", goal = "cdeab";

        System.out.println(rotateString(s, goal));

    }

}

💡 **Question 7**

Given two strings s and t, return true *if they are equal when both are typed into empty text editors*. '#' means a backspace character.

Note that after backspacing an empty text, the text will continue empty.

**Example 1:**

**Input:** s = "ab#c", t = "ad#c"

**Output:** true

**Explanation:**

Both s and t become "ac".

package Assignment\_7;

public class BackspaceStringCompare {

    static boolean backspaceCompare(String s, String t) {

        int i = s.length() - 1, j = t.length() - 1;

        int skip1 = 0, skip2 = 0;

        for (; i >= 0 || j >= 0; --i, --j) {

            while (i >= 0) {

                if (s.charAt(i) == '#') {

                    ++skip1;

                    --i;

                } else if (skip1 > 0) {

                    --skip1;

                    --i;

                } else {

                    break;

                }

            }

            while (j >= 0) {

                if (t.charAt(j) == '#') {

                    ++skip2;

                    --j;

                } else if (skip2 > 0) {

                    --skip2;

                    --j;

                } else {

                    break;

                }

            }

            if (i >= 0 && j >= 0) {

                if (s.charAt(i) != t.charAt(j)) {

                    return false;

                }

            } else if (i >= 0 || j >= 0) {

                return false;

            }

        }

        return true;

    }

    public static void main(String[] args) {

        String s = "ab#c", t = "ad#c";

        System.out.println(backspaceCompare(s, t));

    }

}

💡 **Question 8**

You are given an array coordinates, coordinates[i] = [x, y], where [x, y] represents the coordinate of a point. Check if these points make a straight line in the XY plane.

**Example 1:**

package Assignment\_7;

public class checkStraightLine {

    static 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;

    }

    public static void main(String[] args) {

        int [][] coordinates = {{1,2},{2,3},{3,4},{4,5},{5,6},{6,7}};

        System.out.println(checkStraightLine(coordinates));

    }

}

