<aside>

💡 **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: true

</aside>

```
class Solution {
    public static boolean isIsomorphic(String X, String Y)
  {
      // base case
      if (X == null || Y == null) {
         return false;
      }
      // if 'X' and 'Y' have different lengths, they cannot be isomorphic
      if (X.length() != Y.length()) {
         return false;
      }
      // use a map to store a mapping from characters of string 'X' to string 'Y'
      Map<Character, Character> map = new HashMap<>();
      // use set to store a pool of already mapped characters
      Set<Character> set = new HashSet<>();
      for (int i = 0; i < X.length(); i++)
      {
         char x = X.charAt(i), y = Y.charAt(i);
         // if 'x' is seen before
         if (map.containsKey(x))
         {
```

```
            // return false if the first occurrence of `x` is mapped to a
            // different character
            if (map.get(x) != y) {
                return false;
            }
        }
        // if 'x' is seen for the first time (i.e., it isn't mapped yet)
        else {
            // return false if 'y' is already mapped to some other char in 'X'
            if (set.contains(y)) {
                return false;
            }
            // map 'y' to 'x' and mark it as mapped
            map.put(x, y);
            set.add(y);
        }
    }
    return true;
  }
  public static void main(String[] args)
  {
    String X = "ACAB";
    String Y = "XCXY";
    if (isIsomorphic(X, Y)) {
        System.out.println(X + " and " + Y + " are Isomorphic");
    }
    else {
        System.out.println(X + " and " + Y + " are not Isomorphic");
    }
  }
}
```

<aside>

💡 **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

</aside>

```
class Solution {
    public 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;
    }
}
```

<aside>

💡 **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"

</aside>

```
class Solution {
	public String addStrings(String num1, String num2) {
		StringBuilder result = new StringBuilder();
		int r1 = num1.length();
		int r2 = num2.length();
		int carry = 0;
		while(r1>0 || r2>0) {
			int n1 = (r1 > 0) ? (num1.charAt(r1-1) - '0') : 0;
			int n2 = (r2 > 0) ?		(num2.charAt(r2-1) - '0') : 0;
			int sum = (n1 + n2 + carry) % 10;
			carry = (n1 + n2 + carry) / 10;
			result.insert(0, sum);
			r1 -= 1;
			r2 -= 1;
		}
		if(carry > 0) {
			result.insert(0, carry);
		}
		return result.toString();
	}
}
```

<aside>

💡 **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"

</aside>

```
class Solution {
    public String reverseWords(String s) {
        int lastSpaceIndex = -1;
        char[] chArray = s.toCharArray();
        int len = s.length();
        for (int strIndex = 0; strIndex <= len; strIndex++) {
            if (strIndex == len || chArray[strIndex] == ' ') {
                int startIndex = lastSpaceIndex + 1;
                int endIndex = strIndex - 1;
                while (startIndex < endIndex) {
                    char temp = chArray[startIndex];
                    chArray[startIndex] = chArray[endIndex];
                    chArray[endIndex] = temp;
                    startIndex++;
                    endIndex--;
                }
                lastSpaceIndex = strIndex;
            }
        }
        return new String(chArray);
    }

}
```

<aside>

💡 **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”

</aside>

```
class Solution {
    public String reverseStr(String s, int k) {
        char[] a = s.toCharArray();
        for (int start = 0; start < a.length; start += 2 * k) {
            int i = start, j = Math.min(start + k - 1, a.length - 1);
            while (i < j) {
                char tmp = a[i];
                a[i++] = a[j];
                a[j--] = tmp;
            }
        }
        return new String(a);
    }
}
```

<aside>

💡 **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

</aside>

```
class Solution {
    public boolean rotateString(String A, String B) {
        int N = A.length();
        if (N != B.length()) return false;
        if (N == 0) return true;

        //Compute shift table
        int[] shifts = new int[N+1];
        Arrays.fill(shifts, 1);
        int left = -1;
        for (int right = 0; right < N; ++right) {
            while (left >= 0 && (B.charAt(left) != B.charAt(right)))
                left -= shifts[left];
            shifts[right + 1] = right - left++;
        }

        //Find match of B in A+A
        int matchLen = 0;
```

```
        for (char c: (A+A).toCharArray()) {
            while (matchLen >= 0 && B.charAt(matchLen) != c)
                matchLen -= shifts[matchLen];
            if (++matchLen == N) return true;
        }

        return false;
    }
}
```

<aside> **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".

</aside>

```
class Solution {
   public boolean backspaceCompare(String S, String T) {
      int i = S.length() - 1, j = T.length() - 1;
      int skipS = 0, skipT = 0;

      while (i >= 0 || j >= 0) { // While there may be chars in build(S) or build (T)
         while (i >= 0) { // Find position of next possible char in build(S)
            if (S.charAt(i) == '#') {skipS++; i--;}
            else if (skipS > 0) {skipS--; i--;}
            else break;
         }
         while (j >= 0) { // Find position of next possible char in build(T)
            if (T.charAt(j) == '#') {skipT++; j--;}
            else if (skipT > 0) {skipT--; j--;}
            else break;
         }
         // If two actual characters are different
         if (i >= 0 && j >= 0 && S.charAt(i) != T.charAt(j))
```

```
            return false;
        // If expecting to compare char vs nothing
        if ((i >= 0) != (j >= 0))
            return false;
        i--; j--;
      }
      return true;
   }
}
```

<aside>

💡 **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.

</aside>

```
class Solution {
   // Returns the delta Y.
   int getYDiff(int[] a, int[] b) {
      return a[1] - b[1];
   }

   // Returns the delta X.
   int getXDiff(int[] a, int[] b) {
      return a[0] - b[0];
   }

   public boolean checkStraightLine(int[][] coordinates) {
      int deltaY = getYDiff(coordinates[1], coordinates[0]);
      int deltaX = getXDiff(coordinates[1], coordinates[0]);

      for (int i = 2; i < coordinates.length; i++) {
         // Check if the slope between points 0 and i, is the same as between 0 and 1.
         if (deltaY * getXDiff(coordinates[i], coordinates[0])
            != deltaX * getYDiff(coordinates[i], coordinates[0])) {
            return false;
         }
      }
      return true;
   }
}
```