**DSA – ASSIGNMENT 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:** true

**Solution. :-**

* Check if the lengths of strings s and t are equal. If not, return false because strings of different lengths cannot be isomorphic.
* Create two dictionaries or hash maps to store the character mappings for s and t. Let's call them s\_map and t\_map.
* Iterate through the characters in s and t simultaneously using a loop:
  + For each character s\_char in s and t\_char in t:
    - If s\_char is already in s\_map and its mapped value is not equal to t\_char, return false because a character cannot map to different characters in the other string.
    - If t\_char is already in t\_map and its mapped value is not equal to s\_char, return false for the same reason.
    - If s\_char is not in s\_map and t\_char is not in t\_map, add s\_char as a key and t\_char as its value to s\_map and t\_map, respectively.
  + After the loop, return true because all characters in s can be replaced with characters from t while preserving the order of characters.

**def isomorphic\_strings(s, t):**

**if len(s) != len(t):**

**return False**

**s\_map = {}**

**t\_map = {}**

**for s\_char, t\_char in zip(s, t):**

**if s\_char in s\_map and s\_map[s\_char] != t\_char:**

**return False**

**if t\_char in t\_map and t\_map[t\_char] != s\_char:**

**return False**

**if s\_char not in s\_map and t\_char not in t\_map:**

**s\_map[s\_char] = t\_char**

**t\_map[t\_char] = s\_char**

**return True**

**s = "egg"**

**t = "add"**

**print(isomorphic\_strings(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

**Solution. :-**

* Create a dictionary or hash map to store the strobogrammatic pairs. Let's call it mapping.
* Initialize two pointers, left and right, pointing to the start and end of the string, respectively.
* While left is less than or equal to right:
  + Get the characters at positions left and right in num. Let's call them left\_char and right\_char, respectively.
  + If left\_char is not a key in mapping or its corresponding value is not equal to right\_char, return False because the characters do not form a valid strobogrammatic pair.
  + Increment left by 1 and decrement right by 1.
* If the loop completes without returning False, return True because the string num is a strobogrammatic number.

**def isStrobogrammatic(num):**

**mapping = {**

**'0': '0',**

**'1': '1',**

**'6': '9',**

**'8': '8',**

**'9': '6'**

**}**

**left = 0**

**right = len(num) - 1**

**while left <= right:**

**left\_char = num[left]**

**right\_char = num[right]**

**if left\_char not in mapping or mapping[left\_char] != right\_char:**

**return False**

**left += 1**

**right -= 1**

**return True**

**num = "69"**

**print(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"

**Solution. :-**

* Initialize an empty result string to store the sum.
* Start two pointers, i and j, at the rightmost digits of num1 and num2, respectively.
* Initialize a carry variable to 0.
* While both i and j are greater than or equal to 0:
  + Convert the characters at positions i and j into integers, x and y, respectively.
  + Compute the sum of x, y, and the carry. Let's call this sum s.
  + Take the remainder of s divided by 10, and convert it back to a character. Append this character to the left side of the result string.
  + Update the carry by taking the integer division of s by 10.
  + Decrement both i and j by 1.
* After the above loop, if i is greater than or equal to 0, iterate from i to 0:
  + Convert the character at position i in num1 to an integer, x.
  + Compute the sum of x and the carry. Let's call this sum s.
  + Take the remainder of s divided by 10, and convert it back to a character. Append this character to the left side of the result string.
  + Update the carry by taking the integer division of s by 10.
* If j is greater than or equal to 0, iterate from j to 0:
  + Convert the character at position j in num2 to an integer, y.
  + Compute the sum of y and the carry. Let's call this sum s.
  + Take the remainder of s divided by 10, and convert it back to a character. Append this character to the left side of the result string.
  + Update the carry by taking the integer division of s by 10.
* If the carry is greater than 0, convert it to a character and append it to the left side of the result string.
* Return the result string.

**def addStrings(num1, num2):**

**result = ""**

**i = len(num1) - 1**

**j = len(num2) - 1**

**carry = 0**

**while i >= 0 or j >= 0:**

**x = int(num1[i]) if i >= 0 else 0**

**y = int(num2[j]) if j >= 0 else 0**

**s = x + y + carry**

**digit = str(s % 10)**

**result = digit + result**

**carry = s // 10**

**i -= 1**

**j -= 1**

**if carry > 0:**

**result = str(carry) + result**

**return result**

**num1 = "11"**

**num2 = "123"**

**print(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"

**Solution. :-**

* Split the input string s into individual words. You can use the split() method in Python, which splits the string on whitespace by default, resulting in a list of words.
* Iterate through each word in the list of words.
* Reverse the characters in each word. One way to achieve this is by using string slicing with a step of -1.
* Join the reversed words back into a single string, using a space as the delimiter. You can use the join() method in Python to accomplish this.
* Return the resulting string.

**def reverseWords(s):**

**words = s.split() # Split the string into words**

**reversed\_words = [word[::-1] for word in words] # Reverse each word**

**return ' '.join(reversed\_words) # Join the reversed words with a space delimiter**

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

**print(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"

**Solution. :-**

* Convert the string s into a list of characters, as strings are immutable in Python.
* Initialize a pointer i to 0 to keep track of the current position in the string.
* While i is less than the length of the string:
  + Reverse the characters at positions i to i+k in the list. You can use list slicing and the reverse() method to achieve this.
  + Update the pointer i by 2k to move to the next set of 2k characters.
* Convert the list of characters back into a string using the join() method.
* Return the resulting string.

**def reverseStr(s, k):**

**chars = list(s)**

**i = 0**

**while i < len(chars):**

**chars[i:i+k] = reversed(chars[i:i+k])**

**i += 2\*k**

**return ''.join(chars)**

**s = "abcdefg"**

**k = 2**

**print(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

**Solution. :-**

* Check if the lengths of s and goal are equal. If they are not equal, return False because s cannot become goal through any number of shifts.
* Concatenate s with itself to create a new string s\_double. This allows us to simulate the shifts on s without actually performing the shifts.
* Check if goal is a substring of s\_double. If it is, return True because s can become goal through some number of shifts.
* If goal is not a substring of s\_double, return False because s cannot become goal through any number of shifts.

**def canShift(s, goal):**

**if len(s) != len(goal):**

**return False**

**s\_double = s + s**

**return goal in s\_double**

**s = "abcde"**

**goal = "cdeab"**

**print(canShift(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".

**Solution. :-**

* Initialize an empty stack to store the characters.
* Iterate through each character c in the input string:
  + If c is not a backspace character ('#'), push it onto the stack.
  + If c is a backspace character ('#'), check if the stack is empty or not.
    - If the stack is not empty, pop the top character from the stack (removing the previous character).
    - If the stack is empty, do nothing.
* After the loop, convert the remaining characters in the stack back into a string.
* Return the resulting string.

**def processString(string):**

**stack = []**

**for c in string:**

**if c != '#':**

**stack.append(c)**

**elif stack:**

**stack.pop()**

**return ''.join(stack)**

**def backspaceCompare(s, t): s = "ab#c"**

**t = "ad#c"**

**print(backspaceCompare(s, t))**

**s\_processed = processString(s)**

**t\_processed = processString(t)**

**return s\_processed == t\_processed**

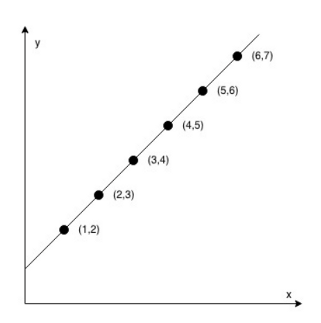
**s = "ab#c"**

**t = "ad#c"**

**print(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:**



**Input:** coordinates = [[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]]

**Output:** true

**Solution. :-**

* If the array coordinates has less than 2 points, return True because a single point or no points always form a straight line.
* Get the x and y coordinates of the first two points, x1, y1, x2, and y2.
* Iterate through the remaining points in coordinates starting from the third point:
  + Get the x and y coordinates of the current point, x, and y.
  + Calculate the slopes slope1 and slope2 between the first two points and the current point:
    - slope1 = (y2 - y1) / (x2 - x1)
    - slope2 = (y - y1) / (x - x1)
  + If slope1 is not equal to slope2, return False because the current point does not lie on the same line.
* If the loop completes without returning False, return True because all points lie on the same straight line.

**def checkStraightLine(coordinates):**

**if len(coordinates) < 2:**

**return True**

**x1, y1 = coordinates[0]**

**x2, y2 = coordinates[1]**

**for i in range(2, len(coordinates)):**

**x, y = coordinates[i]**

**if (y2 - y1) \* (x - x1) != (y - y1) \* (x2 - x1):**

**return False**

**return True**

**coordinates = [[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]]**

**print(checkStraightLine(coordinates))**