Swap Adjacent in LR String

In a string composed of `'L'`, `'R'`, and `'X'` characters, like `"RXXLRXRXL"`, a move consists of either replacing one occurrence of `"XL"` with `"LX"`, or replacing one occurrence of `"RX"` with `"XR"`. Given the starting string `start` and the ending string `end`, return `True` if and only if there exists a sequence of moves to transform one string to the other.

Example:

```
Input: start = "RXXLRXRXL", end = "XRLXXRRLX"
Output: True
Explanation:
We can transform start to end following these steps:
RXXLRXRXL ->
XRXLRXRXL ->
XRLXRXRXL ->
XRLXXRRXL ->
XRLXXRRLX
```

Note:

1. `1 <= len(start) = len(end) <= 10000`.
2. Both start and end will only consist of characters in `{'L', 'R', 'X'}`.

Solution 1

The idea it to compare 'L' position in start is greater than or equal to that in end, also 'R' position in start is less than or equal to that in end.

The reason is 'L' can move left when 'X' exists before, and 'R' can move right when 'X' exists after, and 'L' and 'R' cannot cross each other, so we do not care about 'X'.

```
class Solution:
    def canTransform(self, start, end):
        """
        :type start: str
        :type end: str
        :rtype: bool
        """
        s = [(c, i) for i, c in enumerate(start) if c == 'L' or c == 'R']
        e = [(c, i) for i, c in enumerate(end) if c == 'L' or c == 'R']
        return len(s) == len(e) and all(c1 == c2 and (i1 >= i2 and c1 == 'L' or i1 <
= i2 and c1 == 'R') for (c1, i1), (c2, i2) in zip(s,e))
```

written by yanzhan2 original link here

Solution 2

The idea is simple. Just get the non-X characters and compare the positions of them.

```
class Solution {
    public boolean canTransform(String start, String end) {
        if (!start.replace("X", "").equals(end.replace("X", "")))
            return false;

        int p1 = 0;
        int p2 = 0;

        while(p1 < start.length() && p2 < end.length()){

            // get the non-X positions of 2 strings
            while(p1 < start.length() && start.charAt(p1) == 'X'){
                p1++;
            }
            while(p2 < end.length() && end.charAt(p2) == 'X'){
                p2++;
            }

            //if both of the pointers reach the end the strings are transformable
            if(p1 == start.length() && p2 == end.length()){
                return true;
            }
            // if only one of the pointer reach the end they are not transformable
            if(p1 == start.length() || p2 == end.length()){
                return false;
            }

            if(start.charAt(p1) != end.charAt(p2)){
                return false;
            }
            // if the character is 'L', it can only be moved to the left. p1 should
be greater or equal to p2.
            if(start.charAt(p1) == 'L' && p2 > p1){
                return false;
            }
            // if the character is 'R', it can only be moved to the right. p2 should
be greater or equal to p1.
            if(start.charAt(p1) == 'R' && p1 > p2){
                return false;
            }
            p1++;
            p2++;
        }
        return true;
    }

}
```

written by Samuel.Y.T.Ji original link here

Solution 3

We see that L can move backwards and R can move forwards. So we can first see if subsequence that contains only L & R is the same for start and end. If so, we can compare the corresponding positions of L and R.
For L in start it need to be somewhere ahead of corresponding L in end (since L can only move backwards).
For R in start, it needs to be somewhere behind of corresponding R in end.

```
bool canTransform(string start, string end) {
       int i=0,j=0, n=start.size();
       while(j < n && i < n){
           while(j < n && end[j] == 'X') ++j;
           while(i < n && start[i] == 'X') ++i;
           if(i==n&&j==n) break;
           if(i==n || j==n || start[i] != end[j]) return false;
           if(start[i] == 'R' && i > j) return false;
           else if(start[i] == 'L' && i < j) return false;
           ++i; ++j;
       }

       return true;
   }
```

written by blackspinner original link here