777. Swap Adjacent in LR String

Leetcode Link

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 1:

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

Example 2:

Input: `start = "X", end = "L"`
Output: `false`

Solution

Solution

The first observation we can make is that the two moves can be described as the following: shift `L` to the left and shift `R` to the right. Since `L` and `R` cannot be swapped with each other, the relative order of `L` and `R` letters will **never** change.

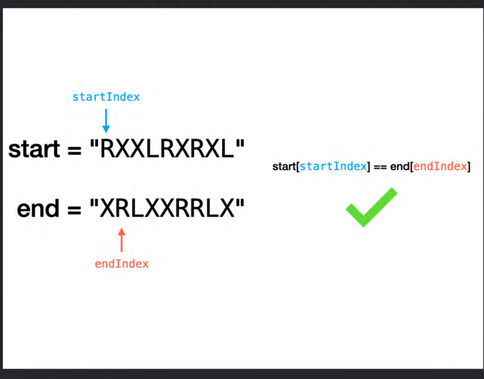
Let's label `L` and `R` as valid letters.

Our first condition for a transformation from `start` to `end` is that both `start` and `end` must have the same number of valid letters. In addition, the first valid letter in `start` must match the first valid letter in `end`, the second valid letter in `start` must match the second valid letter in `end`, and so on until the last.

We can also observe that for a transformation to exist, the i^{th} valid letter in `start` must be able to move to the position of the i^{th} valid letter in `end`. We'll denote `startIndex` as the index of the i^{th} valid letter in `start` and `endIndex` as the index of the i^{th} valid letter in `end`. There are two cases to consider:

- The valid letter is `L`. Since `L` can only move left, a transformation exists when `startIndex >= endIndex`.
- The valid letter is `R`. Since `R` can only move right, a transformation exists when `startIndex <= endIndex`.

We can implement this using the idea of Two Pointers to keep track of `startIndex` and `endIndex` for every valid letter.



Time Complexity

Let's denote N as the length of both strings `start` and `end`.

Since we use Two Pointers to iterate through both strings once, our time complexity is $O(N)$.

Time Complexity: $O(N)$

Space Complexity

Space Complexity: $O(1)$

C++ Solution

```cpp
class Solution {
    public:
    bool canTransform(string start, string end) {
        int n = start.size();
        int startIndex = 0;
        int endIndex = 0;
        while (startIndex < n || endIndex < n) {
            while (startIndex < n &&
                   start[startIndex] == 'X') {  // find next valid letter in start
                startIndex++;
            }
            while (endIndex < n &&
                   end[endIndex] == 'X') {  // find next valid letter in end
                endIndex++;
            }
            if (startIndex == n && endIndex == n) {  // both reached the end
                return true;
            }
            if (startIndex == n || endIndex == n) {  // different number of valid letters
                return false;
            }
            if (start[startIndex] != end[endIndex]) {  // different valid letter
                return false;
            }
            if (start[startIndex] == 'R' && startIndex > endIndex) {  // wrong direction
                return false;
            }
            if (start[startIndex] == 'L' && startIndex < endIndex) {  // wrong direction
                return false;
            }
            startIndex++;
            endIndex++;
        }
        return true;
    }
};
```

Java Solution

```java
class Solution {
    public boolean canTransform(String start, String end) {
        int n = start.length();
        int startIndex = 0;
        int endIndex = 0;
        while (startIndex < n || endIndex < n) {
            while (startIndex < n
                   && start.charAt(startIndex) == 'X') { // find next valid letter in start
                startIndex++;
            }
            while (endIndex < n
                   && end.charAt(endIndex) == 'X') { // find next valid letter in end
                endIndex++;
            }
            if (startIndex == n && endIndex == n) { // both reached the end
                return true;
            }
            if (startIndex == n || endIndex == n) { // different number of valid letters
                return false;
            }
            if (start.charAt(startIndex)
                != end.charAt(endIndex)) { // different valid letter
                return false;
            }
            if (start.charAt(startIndex) == 'R'
                && startIndex > endIndex) { // wrong direction
                return false;
            }
            if (start.charAt(startIndex) == 'L'
                && startIndex < endIndex) { // wrong direction
                return false;
            }
            startIndex++;
            endIndex++;
        }
        return true;
    }
}
```

Python Solution

```python
class Solution:
    def canTransform(self, start: str, end: str) -> bool:
        n = len(start)
        startIndex = 0
        endIndex = 0
        while startIndex < n or endIndex < n:
            while (
                startIndex < n and start[startIndex] == "X"
            ):  # find next valid letter in start
                startIndex += 1
            while (
                endIndex < n and end[endIndex] == "X"
            ):  # find next valid letter in end
                endIndex += 1
            if startIndex == n and endIndex == n:  # both reached the end
                return True
            if startIndex == n or endIndex == n:  # different number of valid letters
                return False
            if start[startIndex] != end[endIndex]:  # different valid letter
                return False
            if start[startIndex] == "R" and startIndex > endIndex:  # wrong direction
                return False
            if start[startIndex] == "L" and startIndex < endIndex:  # wrong direction
                return False
            startIndex += 1
            endIndex += 1
        return True

```