**难度: 简单**

给定 S 和 T 两个字符串，当它们分别被输入到空白的文本编辑器后，判断二者是否相等，并返回结果。 # 代表退格字符。

注意：如果对空文本输入退格字符，文本继续为空。

示例 1：

输入：S = "ab#c", T = "ad#c"

输出：true

解释：S 和 T 都会变成 “ac”。

示例 2：

输入：S = "ab##", T = "c#d#"

输出：true

解释：S 和 T 都会变成 “”。

示例 3：

输入：S = "a##c", T = "#a#c"

输出：true

解释：S 和 T 都会变成 “c”。

示例 4：

输入：S = "a#c", T = "b"

输出：false

解释：S 会变成 “c”，但 T 仍然是 “b”。

提示：

1 <= S.length <= 200

1 <= T.length <= 200

S 和 T 只含有小写字母以及字符 '#'。

进阶：

你可以用 O(N) 的时间复杂度和 O(1) 的空间复杂度解决该问题吗？

来源：力扣（LeetCode）

链接：https://leetcode-cn.com/problems/backspace-string-compare

著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。

**思路1(使用栈):**

1). 创建栈,如果不是#就入栈,是#并且栈不为空就退栈

2). 比较剩余留在栈中的字符是否相同

|  |
| --- |
| public boolean backspaceCompare(String S, String T) {  //S字符串退格  String backspaceS = backspace(S);  //T字符串退格  String backspaceT = backspace(T);  //判断两个栈是否都为空  return backspaceS.equals(backspaceT);  }  /\*\*  \*  \* @param s 需要退格的字符串  \* @return 返回字符串  \*/  private String backspace(String s) {  Deque<Character> stack = new ArrayDeque<>();  //转换成字符数组  char[] ch = s.toCharArray();  for (int i = 0; i < ch.length; i++) {  //如果不是#就入栈  if (ch[i] != '#') {  stack.push(ch[i]);  //是#就判断一下栈是否为空,不为空就抛出  }else if (!stack.isEmpty()){  stack.pop();  }  }  StringBuilder sb = new StringBuilder();  //把栈中剩余的字符抛出  while(!stack.isEmpty()) {  sb.append(stack.pop());  }  //转成字符串类型  return sb.toString();  } |

**思路2(模拟栈):**

思路与栈差不多

1). 这里是直接使用springBuilder来模拟栈

|  |
| --- |
| public boolean backspaceCompare2(String S, String T) {  //S字符串退格  String backspaceS = backspace2(S);  //T字符串退格  String backspaceT = backspace2(T);  //判断两个栈是否都为空  return backspaceS.equals(backspaceT);  }  private String backspace2(String s) {  char[] chars = s.toCharArray();  StringBuilder sb = new StringBuilder();  for (int i = 0; i < chars.length; i++) {  if (chars[i] != '#'){  sb.append(chars[i]);  }else {  if (sb.length() > 0) {  sb.deleteCharAt(sb.length()-1);  }  }  }  return sb.toString();  } |

**思路3(使用双指针从尾部进行比较):**

代码很容易理解

|  |
| --- |
| public boolean backspaceCompare3(String S, String T) {  //S的最后节点  int i = S.length() -1;  //S遇到#的次数  int skipS = 0;  //T的最后节点  int j = T.length() -1;  //T遇到#的次数  int skipT = 0;  while(i >= 0 || j >= 0) {  while(i >= 0) {  if (S.charAt(i) == '#') {  skipS++;  i--;  }else if (skipS > 0) {  skipS--;  i--;  }else {  break;  }  }  while(j >= 0) {  if (T.charAt(j) == '#') {  skipT++;  j--;  }else if (skipT > 0) {  skipT--;  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;  }  }  i--;  j--;  }  return true;  } |