```
start++;
i++;
}
```

# 数组中两个字符串的最小距离

#### 【题目】

给定一个字符串数组 strs, 再给定两个字符串 str1 和 str2, 返回在 strs 中 str1 与 str2 的 最小距离, 如果 str1 或 str2 为 null, 或不在 strs 中, 返回-1。

#### 【举例】

```
strs=["1","3","3","3","2","3","1"], str1="1", str2="2", 返回 2。
strs=["CD"], str1="CD", str2="AB", 返回-1。
```

### 【进阶题目】

如果查询发生的次数有很多,如何把每次查询的时间复杂度降为 O(1)?

## 【难度】

尉 ★★☆☆

## 【解答】

原问题。从左到右遍历 strs,用变量 last1 记录最近一次出现的 str1 的位置,用变量 last2 记录最近一次出现的 str2 的位置。如果遍历到 str1,那么 i-last2 的值就是当前的 str1 和左 边最它最近的 str2 之间的距离。如果遍历到 str2,那么 i-last1 的值就是当前的 str2 和左边最它最近的 str1 之间的距离。用变量 min 记录这些距离的最小值即可。请参看如下的 minDistance 方法。

```
public int minDistance(String[] strs, String str1, String str2) {
    if (str1 == null || str2 == null) {
        return -1;
    }
    if (str1.equals(str2)) {
        return 0;
    }
    int last1 = -1;
```

```
int last2 = -1;
int min = Integer.MAX_VALUE;
for (int i = 0; i != strs.length; i++) {
    if (strs[i].equals(str1)) {
        min = Math.min(min, last2 == -1 ? min : i - last2);
        last1 = i;
    }
    if (strs[i].equals(str2)) {
        min = Math.min(min, last1 == -1 ? min : i - last1);
        last2 = i;
    }
}
return min == Integer.MAX_VALUE ? -1 : min;
}
```

进阶问题。其实是通过数组 strs 先生成某种记录,在查询时通过记录进行查询,本书提供了一种记录的结构供读者参考,如果 strs 的长度为 N,那么生成记录的时间复杂度为  $O(N^2)$ ,记录的空间复杂度为  $O(N^2)$ ,在生成记录之后,单次查询操作的时间复杂度可降为 O(1)。本书实现的记录其实是一个哈希表 HashMap<String, HashMap<String, Integer>>,这是一个 key 为 string 类型、value 为哈希表类型的哈希表。为了描述清楚,我们把这个哈希表叫作外哈希表,把 value 代表的哈希表叫作内哈希表。外哈希表的 key 代表 strs 中的某种字符串,key 所对应的内哈希表表示其他字符串到 key 字符串的最小距离。比如,当 strs 为["1","3","3","3","2","3","1"]时,生成的记录如下(外哈希表):

key	Value (Value 仍为一个哈希表,记为内哈希表)
"l"	("2", 2)->"1"到"2"的最小距离为2
	("3", 1)->"1"到"3"的最小距离为1
"2"	("1", 2)-> "2"到"1"的最小距离为 2
	("3", 1)-> "2"到"3"的最小距离为1
"3"	("1", 1)-> "3"到"1"的最小距离为1
	("2", 1)->"3"到"2"的最小距离为1

如果生成了这种结构的记录,那么查询 strl 和 str2 的最小距离时只用两次哈希查询操作就可以完成。

如下代码的 Record 类就是这种记录结构的具体实现,建立记录过程就是 Record 类的构造函数, Record 类中的 minDistance 方法就是做单次查询的方法。

```
public class Record {
   private HashMap<String, HashMap<String, Integer>> record;
   public Record(String[] strArr) {
```

```
record = new HashMap<String, HashMap<String, Integer>>();
       HashMap<String, Integer> indexMap = new HashMap<String, Integer>();
        for (int i = 0; i != strArr.length; i++) {
            String curStr = strArr[i];
            update(indexMap, curStr, i);
            indexMap.put(curStr, i);
private void update (HashMap < String, Integer > indexMap, String str, int i) {
    if (!record.containsKey(str)) {
       record.put(str, new HashMap<String, Integer>());
    HashMap<String, Integer> strMap = record.get(str);
    for (Entry<String, Integer> lastEntry : indexMap.entrySet()) {
       String key = lastEntry.getKey();
       int index = lastEntry.getValue();
       if (!key.equals(str)) {
            HashMap<String, Integer> lastMap = record.get(key);
            int curMin = i - index;
            if (strMap.containsKey(key)) {
                   int preMin = strMap.get(key);
                   if (curMin < preMin) {
                      strMap.put(key, curMin);
                       lastMap.put(str, curMin);
            } else {
                   strMap.put(key, curMin);
                   lastMap.put(str, curMin);
       }
    }
public int minDistance (String str1, String str2) {
    if (str1 == null || str2 == null) {
           return -1;
    if (str1.equals(str2)) {
           return 0;
    if (record.containsKey(str1) && record.get(str1).containsKey(str2)) {
           return record.get(str1).get(str2);
    return -1;
}
```

}