```
        @Override
        public int compare(String a, String b) {
                return (a + b).compareTo(b + a);
        }
}

public String lowestString(String[] strs) {
        if (strs == null || strs.length == 0) {
                return "";
        }
        // 根据新的比较方式排序
        Arrays.sort(strs, new MyComparator());
        String res = "";
        for (int i = 0; i < strs.length; i++) {
                res += strs[i];
        }
        return res;
}
```

本题的解法看似非常简单，但解法有效性的证明却比较复杂。在这里不得不提醒读者，这道题的解题方法可以划进贪心算法的范畴，这种有效的比较方式就是我们的贪心策略。

正如本题所展示的一样，贪心策略容易大胆假设，但策略有效性的证明可就不容易求证了。在面试中，如果哪一个题目决定用贪心方法求解，则必须用较大的篇幅去证明你提出的贪心策略是有效的。所以建议面试准备时间不充裕的读者不要轻易去啃有关贪心策略的题目，那将占用大量的时间和精力。

在面试中，实际上也较少出现需要用到贪心策略的题目，造成这个现象有两个很重要的原因，其一是考查贪心策略的面试题目，关键点在于数学上对策略的证明过程，偏离考查编程能力的面试初衷。其二是纯用贪心策略的面试题，解法的正确性完全在于贪心策略的成败，而缺少其他解法的多样性，这样就会使这一类面试题的区分度极差，所以往往不会成为大公司的面试题。贪心策略在算法上的地位当然重要，但对初期准备代码面试的读者来说，性价比不高。

找到字符串的最长无重复字符子串

【题目】

给定一个字符串 str，返回 str 的最长无重复字符子串的长度。

【举例】

str="abcd"，返回 4

str="aabcb"，最长无重复字符子串为"abc"，返回 3。

【要求】

如果 str 的长度为 N，请实现时间复杂度为 $O(N)$的方法。

【难度】

尉 ★★☆☆

【解答】

如果 str 长度为 N，字符编码范围是 M，本题可做到的时间复杂度为 $O(N)$，额外空间复杂度为 $O(M)$。下面介绍这种方法的具体实现。

1．在遍历 str 之前，先申请几个变量。哈希表 map，key 表示某个字符，value 为这个字符最近一次出现的位置。整型变量 pre，如果当前遍历到字符 str[i]，pre 表示在必须以 str[i-1]字符结尾的情况下，最长无重复字符子串开始位置的前一个位置，初始时 pre=-1。整型变量 len，记录以每一个字符结尾的情况下，最长无重复字符子串长度的最大值，初始时，len=0。从左到右依次遍历 str，假设现在遍历到 str[i]，接下来求在必须以 str[i]结尾的情况下，最长无重复字符子串的长度。

2．map(str[i])的值表示之前的遍历中最近一次出现 str[i]字符的位置，假设在 a 位置。想要求以 str[i]结尾的最长无重复子串，a 位置必然不能包含进来，因为 str[a]等于 str[i]。

3．根据 pre 的定义，pre+1 表示在必须以 str[i-1]字符结尾的情况下，最长无重复字符子串的开始位置，也就是说，以 str[i-1]结尾的最长无重复子串是向左扩到 pre 位置停止的。

4．如果 pre 位置在 a 位置的左边，因为 str[a]不能包含进来，而 str[a+1..i-1]上都是不重复的，所以以 str[i]结尾的最长无重复字符子串就是 str[a+1..i]。如果 pre 位置在 a 位置的右边，以 str[i-1]结尾的最长无重复子串是向左扩到 pre 位置停止的。所以以 str[i]结尾的最长无重复子串向左扩到 pre 位置也必然会停止，而且 str[pre+1..i-1]这一段上肯定不含有 str[i]，所以以 str[i]结尾的最长无重复字符子串就是 str[pre+1..i]。

5．计算完长度之后，pre 位置和 a 位置哪一个在右边，就作为新的 pre 值。然后去计算下一个位置的字符，整个过程中求得所有长度的最大值用 len 记录下来返回即可。

具体请参看如下代码中的 maxUnique 方法。

```
public int maxUnique(String str) {
        if (str == null || str.equals("")) {
                return 0;
        }
        char[] chas = str.toCharArray();
        int[] map = new int[256];
        for (int i = 0; i < 256; i++) {
                map[i] = -1;
        }
        int len = 0;
        int pre = -1;
        int cur = 0;
        for (int i = 0; i != chas.length; i++) {
                pre = Math.max(pre, map[chas[i]]);
                cur = i - pre;
                len = Math.max(len, cur);
                map[chas[i]] = i;
        }
        return len;
}
```

找到被指的新类型字符

【题目】

新类型字符的定义如下：

1．新类型字符是长度为 1 或者 2 的字符串。

2．表现形式可以仅是小写字母，例如，"e"；也可以是大写字母+小写字母，例如，"Ab"；还可以是大写字母+大写字母，例如，"DC"。

现在给定一个字符串 str，str 一定是若干新类型字符正确组合的结果。比如"eaCCBi"，由新类型字符"e"、"a"、"CC"和"Bi"拼成。再给定一个整数 k，代表 str 中的位置。请返回被 k 位置指中的新类型字符。

【举例】

str="aaABCDEcBCg"。

1．k=7 时，返回"Ec"。

2．k=4 时，返回"CD"。

3．k=10 时，返回"g"。