符统计的阶段变为遇到字符的阶段；遇到 str[6]=='b'，一个新的字符出现了，此时令 sum+=num（即 sum=100），sum 表示目前原字符串走到什么位置了，此时发现 sum 并未到达 index 位置，说明还要继续遍历，记录下遇到了字符'b'，即 cur='b'，然后令 num=0，因为字符'a'的统计已经完成，现在 num 开始表示字符'b'的连续数量。也就是说，每遇到一个新的字符，都把上一个已经完成的统计数 num 加到 sum 上，再看 sum 是否到达 index，如果已到达，就返回上一个字符 cur，如果没到达，就继续遍历。

3．每个字符的统计都在遇到新字符时加到 sum 上，所以当遍历完成时，最后一个字符的统计数并不会加到 sum 上，最后要单独加。

具体过程请参看如下代码中的 getCharAt 方法。

```
public char getCharAt(String cstr, int index) {
        if (cstr == null || cstr.equals("")) {
                return 0;
        }
        char[] chs = cstr.toCharArray();
        boolean stage = true;
        char cur = 0;
        int num = 0;
        int sum = 0;
        for (int i = 0; i != chs.length; i++) {
                if (chs[i] == '_') {
                        stage = !stage;
                } else if (stage) {
                        sum += num;
                        if (sum > index) {
                                return cur;
                        }
                        num = 0;
                        cur = chs[i];
                } else {
                        num = num * 10 + chs[i] - '0';
                }
        }
        return sum + num > index ? cur : 0;
}
```

判断字符数组中是否所有的字符都只出现过一次

【题目】

给定一个字符类型数组 chas[]，判断 chas 中是否所有的字符都只出现过一次，请根据以下不同的两种要求实现两个函数。

【举例】

chas=['a','b','c']，返回 true；chas=['1','2','1']，返回 false。

【要求】

1．实现时间复杂度为 $O(N)$的方法。

2．在保证额外空间复杂度为 $O(1)$的前提下，请实现时间复杂度尽量低的方法。

【难度】

按要求 1 实现的方法　士 ★☆☆☆

按要求 2 实现的方法　尉 ★★☆☆

【解答】

要求 1。遍历一遍 chas，用 map 记录每种字符的出现情况，这样就可以在遍历时发现字符重复出现的情况，map 可以用长度固定的数组实现，也可以用哈希表实现。具体请参看如下代码中的 isUnique1 方法。

```
public boolean isUnique1(char[] chas) {
        if (chas == null) {
                return true;
        }
        boolean[] map = new boolean[256];
        for (int i = 0; i < chas.length; i++) {
                if (map[chas[i]]) {
                        return false;
                }
                map[chas[i]] = true;
        }
        return true;
}
```

要求 2。整体思路是先将 chas 排序，排序后相同的字符就放在一起，然后判断有没有重复字符就会变得非常容易，所以问题的关键是选择什么样的排序算法。因为必须保证额外空间复杂度为 $O(1)$，所以本题是考查面试者对经典排序算法在额外空间复杂度方面的理解程度。首先，任何时间复杂度为 $O(N)$的排序算法做不到额外空间复杂度为 $O(1)$，因为这些排序算法不是基于比较的排序算法，所以有多少个数都得“装下”，然后按照一定顺序“倒出”来完成排序。具体细节请读者查阅相关图书中有关桶排序、基数排序、计数排序等内容。然后看时间复杂度 $O(N\log N)$的排序算法，常见的有归并排序、快速排序、希尔排序和

堆排序。归并排序首先被排除，因为归并排序中有两个小组合并成一个大组的过程，这个过程需要辅助数组才能完成，尽管归并排序可以使用手摇算法将额外空间复杂度降至 $O(1)$，但这样最差情况下的时间复杂度会因此上升至 $O(N^2)$。快速排序也被排除，因为无论选择递归实现还是非递归实现，快速排序的额外空间复杂度最低，为 $O(\log N)$，不能达到 $O(1)$的程度。希尔排序同样被排除，因为希尔排序的时间复杂度并不固定，成败完全在于步长的选择，如果选择不当，时间复杂度会变成 $O(N^2)$。这四种经典排序中，只有堆排序可以做到额外空间复杂度为 $O(1)$的情况下，时间复杂度还能稳定地保持 $O(N\log N)$。那么堆排序就是答案，面试者似乎只要写出堆排序的大体过程，要求 2 的实现就能完成。

但遗憾的是，虽然堆排序的确是答案，但大部分资料提供的堆排序的实现却是基于递归函数实现的。而我们知道递归函数需要使用函数栈空间，这样堆排序的额外空间复杂度就增加至 $O(\log N)$。所以，如果真正想达到要求 2 的实现，面试者需要用非递归的方式实现堆排序。要求 2 的实现请参看如下代码中的 isUnique2 方法，其中的 heapSort 方法是堆排序的非递归实现。

```
public boolean isUnique2(char[] chas) {
        if (chas == null) {
                return true;
        }
        heapSort(chas);
        for (int i = 1; i < chas.length; i++) {
                if (chas[i] == chas[i - 1]) {
                        return false;
                }
        }
        return true;
}

public void heapSort(char[] chas) {
        for (int i = 0; i < chas.length; i++) {
                heapInsert(chas, i);
        }
        for (int i = chas.length - 1; i > 0; i--) {
                swap(chas, 0, i);
                heapify(chas, 0, i);
        }
}

public void heapInsert(char[] chas, int i) {
        int parent = 0;
        while (i != 0) {
                parent = (i - 1) / 2;
                if (chas[parent] < chas[i]) {
                        swap(chas, parent, i);
```

```
                        i = parent;
                } else {
                        break;
                }
        }
}

public void heapify(char[] chas, int i, int size) {
        int left = i * 2 + 1;
        int right = i * 2 + 2;
        int largest = i;
        while (left < size) {
                if (chas[left] > chas[i]) {
                        largest = left;
                }
                if (right < size && chas[right] > chas[largest]) {
                        largest = right;
                }
                if (largest != i) {
                        swap(chas, largest, i);
                } else {
                        break;
                }
                i = largest;
                left = i * 2 + 1;
                right = i * 2 + 2;
        }
}

public void swap(char[] chas, int index1, int index2) {
        char tmp = chas[index1];
        chas[index1] = chas[index2];
        chas[index2] = tmp;
}
```

在有序但含有空的数组中查找字符串

【题目】

给定一个字符串数组 strs[]，在 strs 中有些位置为 null，但在不为 null 的位置上，其字符串是按照字典顺序由小到大依次出现的。再给定一个字符串 str，请返回 str 在 strs 中出现的最左的位置。

【举例】

strs=[null,"a",null,"a",null,"b",null,"c"]，str="a"，返回 1。