压缩字符串

给定一组字符，使用原地算法将其压缩。

压缩后的长度必须始终小于或等于原数组长度。

数组的每个元素应该是长度为 1 的**字符**（不是 int 整数类型）。

在完成原地**修改输入数组**后，返回数组的新长度。

进阶：
你能否仅使用 O(1) 空间解决问题？

示例 1：

输入：

```
["a","a","b","b","c","c","c"]
```

输出：

返回 `6` ，输入数组的前 `6` 个字符应该是：`["a","2","b","2","c","3"]`

说明：

`"aa"` 被 `"a2"` 替代。`"bb"` 被 `"b2"` 替代。`"ccc"` 被 `"c3"` 替代。

示例 2：

输入：

```
["a"]
```

输出：

返回 `1` ，输入数组的前 `1` 个字符应该是：`["a"]`

解释：

没有任何字符串被替代。

示例 3：

输入：

```
["a","b","b","b","b","b","b","b","b","b","b","b","b"]
```

输出：

返回 `4` ，输入数组的前 `4` 个字符应该是：`["a","b","1","2"]`。

解释：

由于字符 `"a"` 不重复，所以不会被压缩。`"bbbbbbbbbbbb"` 被 “`b12`” 替代。

注意每个数字在数组中都有它自己的位置。

```
/*
用指针 cur 指向连续块的起始位置，用指针 next 指向不同于该连续块字符的第一个位置，用 count 指针更新字符数组。
next-cur 即为连续块长度，若为 1，则不写入，一趟完成后让 cur 指向 next，即继续扫描下一连续块。
*/
/*
int compress(char* chars, int charsSize){
    int count=0;//修改原数组
    char buff[1000];
    int cur=0;
    int next=0;
    while(next<charsSize)
    {
        while(next<charsSize&&chars[cur]==chars[next])
            next++;//next 寻找下一个不同于当前 cur 连续块字符的第一个位置
        chars[count++]=chars[cur];
        if(next-cur==1)
            continue;
        sprintf(buff,"%d",next-cur);
        for(int i=0;i<strlen(buff);i++)
            chars[count++] = buff[i];

        cur=next;//更新当前连续块的起始位置
    }
```

```
    return count;
}
*/


int compress(char* chars, int charsSize){
    int write = 0;//修改原数组
    char buf[1000];
    for(int read=0, anchor=0;read<charsSize;anchor=read)//需要更新当前连续块的起始位置
    {
        while(read<charsSize&&chars[read]==chars[anchor])
            read++;//寻找下一个不同于当前连续块字符的第一个位置
        chars[write++] = chars[anchor];//原数组覆盖存储无重复的新数据
        if(read-anchor==1)
            continue;
        sprintf(buf,"%d",read-anchor);
        for(int i=0;i<strlen(buf);i++)
            chars[write++] = buf[i];
    }
    return write;
}
```