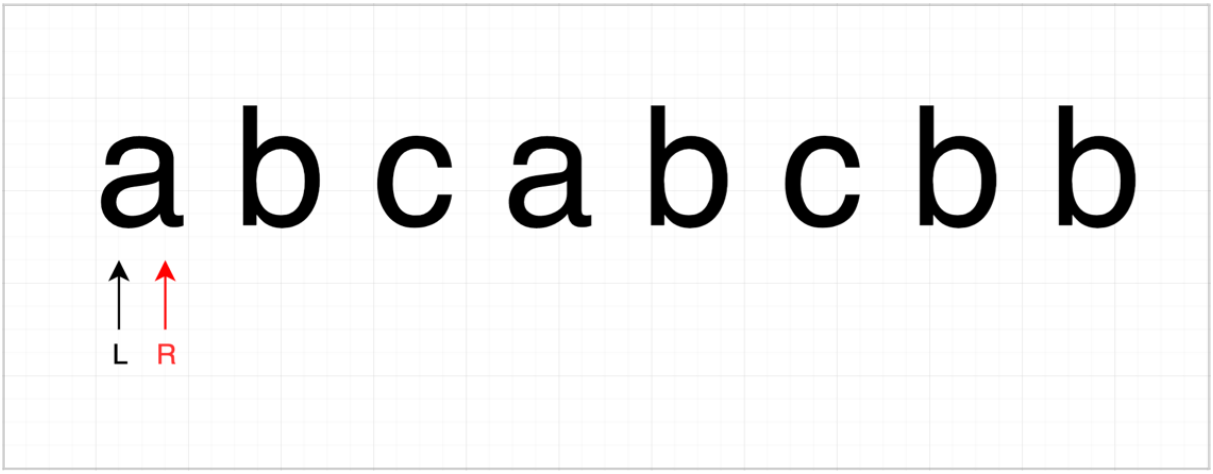
3. 无重复字符的最长子串

本题比较简单的思路是通过双指针来找到无重复的最长子串，具体优化是在双指针中进行优化，我们先看双指针的思路。

初始化，maxLength = 1, 双指针L, R 分别指向字符串开始位置：

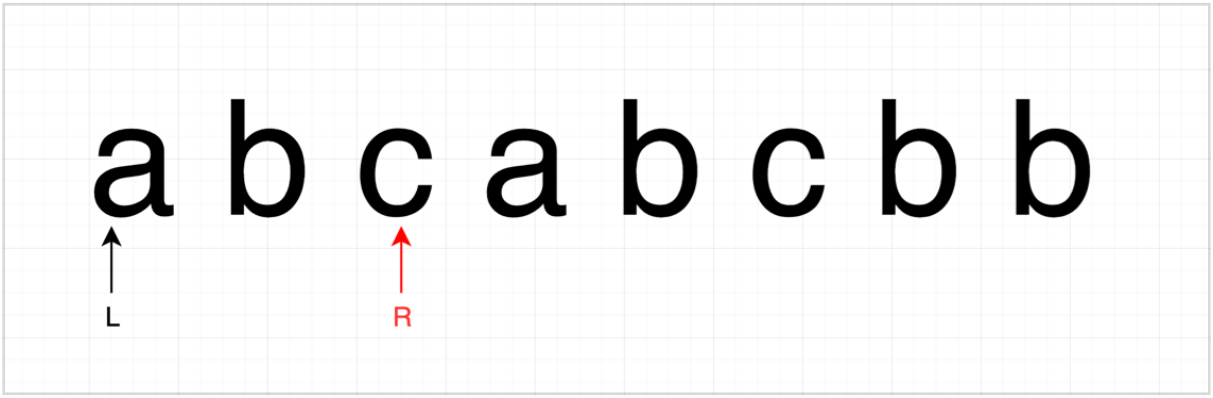


此时最长子串长度为 maxLength = Max(maxLength, R - L)

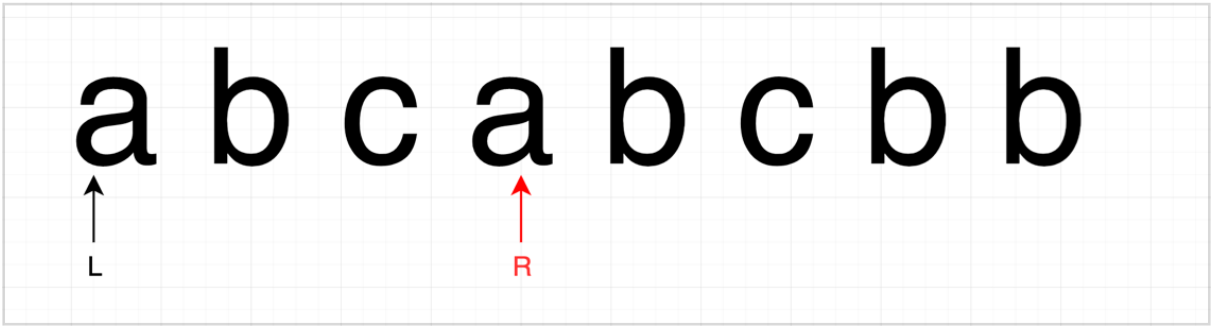
接着每次 R 右移一个位置，当右移之后，R 指向一个新的字符，首先判断 L~R-1 内 是否有该字符，若有，说明出现了重复；否则 继续下一次右移

每次移动后 需要更新一次 maxLength 的值：

maxLength = Max(maxLength, R - L)

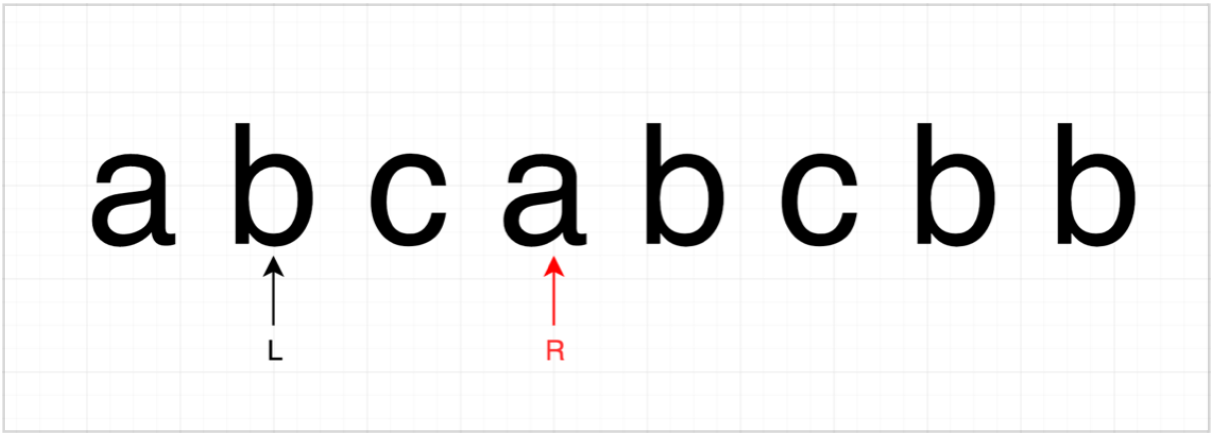


当 R 指向 'c' 时，Str[L~R-1] 内（即 "ab" 子串）没有 字符 c 出现，所以 R 可以继续右移。



当 R 指向 'a' 时，Str[L~R-1] 内出现了字符 a，

然后更新 L 为 Str[L~R-1]内出现字符 a 位置的下一个位置，即 b 的位置：



接着 R 可以继续右移，当 R 到达 最右边的位置时，再更新一次 maxLength 即可，最后返回 maxLength

这里的双指针优化的点在于，当我们在子串 Str[L:R-1]内匹配时，可以使用一个 map 来降低匹配时间复杂度，提高效率。同时本题中只会出现ascii字符，我们完全可以使用一个 大小为 256 的一个 int 数组来替代 map 的作用，减少空间使用。

AC 代码参考：

```
func lengthOfLongestSubstring(s string) int {
    Exist := make([]int, 256)
    for i, _ := range Exist {
        Exist[i] = -1
    }
    start, end := 0, 0
    ans := 0
    for idx, _ := range s {
        if i := Exist[s[idx]]; i!=-1 {
            if start < i {
                start = i
            }
        }
        Exist[s[idx]] = idx+1 // 从该位置开始不重复
        end = idx
        ans = max(ans, end - start + 1)
    }
    return ans
}
```

```go
func max(a, b int) int {
    if a > b {
        return a
    }
    return b
}
```