```
/*
给定一个字符串 S 和一个字符 C。返回一个代表字符串 S 中每个字符到字符串 S 中的字符 C 的最短距离的数组。

示例 1:

输入: S = "loveleetcode", C = 'e'
输出: [3, 2, 1, 0, 1, 0, 0, 1, 2, 2, 1, 0]
说明:

字符串 S 的长度范围为 [1, 10000]。
C 是一个单字符，且保证是字符串 S 里的字符。
S 和 C 中的所有字母均为小写字母。

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/shortest-distance-to-a-character
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:

- 暴力法:
  - 首选遍历一次S,找到所有C的索引
  - 再遍历一次S,然后依次计算距离并保存最小值
  - 时间复杂度为$O(M*N)$,其中M为S的长度，N是S中C的个数.
- 双索引法:
  - 遍历S的所有位置，若当前位置的值不为C,就从当前位置开始分别往S的左半部分和右半部分遍历直到遇到C，然后计算距离保存即可.

方法一:C++_暴力法

```
class Solution
{

    public:
        vector<int> shortestToChar(string S, char C)
        {
            vector<int>  ret_val        ;
            vector<int>  temp_ind       ;
            int          i       =   0  ;
            int          j       =   0  ;
            int          min_ind =   0   ;

            /*找到S中所有C的索引*/
            for(i=0;i<S.size();i++)
            {
                if(C==S.at(i))
                {
                    temp_ind.push_back(i);
                }
            }
```

```
            /*遍历S中的每一个元素并计算*/
            for(i=0;i<S.size();i++)
            {
                min_ind = 10000;
                if(S.at(i)==C)
                {
                    min_ind = 0;
                }
                else
                {
                    for(j=0;j<temp_ind.size();j++)
                    {
                        if(abs( temp_ind[j] - i ) < min_ind)
                        {
                            min_ind = abs( temp_ind[j] - i );
                        }
                    }
                }
                ret_val.push_back(min_ind);
            }
            return ret_val;
        }
};
/*
执行结果:
通过
显示详情
执行用时 :20 ms, 在所有 C++ 提交中击败了47.42%的用户
内存消耗 :8.9 MB, 在所有 C++ 提交中击败了74.80%的用户
*/
```

方法二:C++_双索引法

```
class Solution
{
    public:
        vector<int> shortestToChar(string S, char C)
        {
            vector<int> ret_val ;
            int     len     =   S.size();
            int     i       =   0;
            int     left    =   0;
            int     right   =   0;
            char    ch      =   0;
            int     min_ind =   0;

            for(i = 0 ; i < len ; i++)
            {
                ch = S.at(i);
                if(ch == C)
                {
                    ret_val.push_back(0);
                }
                else
                {
```

```
                    left    = i - 1;
                    right   = i + 1;
                    while(1)
                    {
                        if(  ( left >=0)
                           &&( C == S.at(left))
                        )
                        {
                            ret_val.push_back(i-left);
                            break;
                        }
                        else
                        {
                            left--;
                        }

                        if(   (right < len)
                            &&(C == S.at(right))
                        )
                        {
                            ret_val.push_back(right-i);
                            break;
                        }
                        else
                        {
                            right++;
                        }
                    }
                }
            }
            return ret_val;
        }
};

/*
执行结果:
通过
显示详情
执行用时 :12 ms, 在所有 C++ 提交中击败了96.01%的用户
内存消耗 :8.9 MB, 在所有 C++ 提交中击败了77.17%的用户
*/
```

AlimyBreak
2019.10.15