

【第二十九周】经典匹配算法：KMP、Sunday 与 Shift-[AndOr] 算法

1、[28. 实现 strStr\(\)](#)

1. 方法：暴力匹配
2. 让字符串 `needle` 与字符串 `haystack` 的所有长度为 `m` 的子串均匹配一次。主要是判断文本串里是否出现过模式串。
3. 为了减少不必要的匹配，我们每次匹配失败即立刻停止当前子串的匹配，对下一个子串继续匹配。
4. 如果当前子串匹配成功，我们返回当前子串的开始位置即可。如果所有子串都匹配失败，则返回 `-1`。

```
var strStr = function(haystack, needle) {  
    const n = haystack.length, m = needle.length;  
    for (let i = 0; i + m <= n; i++) {  
        let flag = true;  
        for (let j = 0; j < m; j++) {  
            if (haystack[i + j] !== needle[j]) {  
                flag = false;  
                break;  
            }  
        }  
        if (flag) {  
            return i;  
        }  
    }  
    return -1;  
};
```

2、[459. 重复的子字符串](#)

1. 从 0 开始至字符串长度的一半，一个个枚举子字符串，再拿这个子字符串切割字符串 `s`，看它切割后是否每项都是空。
2. 因为要找的是能重复 构成 `s` 的子串，所以 `s` 的长度 必为 该子串的长度 的整数倍

```
/**  
 * @param {string} s  
 * @return {boolean}  
 */
```

```
var repeatedSubstringPattern = function(s) {
    let n = s.length
    for (let i = 0; i < (n - 1) / 2; i++) {
        if ( n % (i + 1) === 0 ) {
            let str = s.slice(0, i + 1)
            if( !s.split(str).join('') ) return true
        }
    }
    return false
};
```

3、3. 无重复字符的最长子串

1. 方法：滑动窗口
2. 保证滑动窗口内不出现重复字符，求最大的滑动窗口的大小
3. Set

```
/**
 * @param {string} s
 * @return {number}
 */
// 滑动窗口
var lengthOfLongestSubstring = function(s) {
    // 哈希集合，记录每个字符是否出现过
    const occ = new Set();
    const n = s.length;
    // 右指针，初始值为 -1，相当于我们在字符串的左边界的左侧，还没有开始移动
    let rk = -1, ans = 0;
    for (let i = 0; i < n; ++i) {
        if (i !== 0) {
            // 左指针向右移动一格，移除一个字符
            occ.delete(s.charAt(i - 1));
        }
        while (rk + 1 < n && !occ.has(s.charAt(rk + 1))) {
            // 不断地移动右指针
            occ.add(s.charAt(rk + 1));
            ++rk;
        }
        // 第 i 到 rk 个字符是一个极长的无重复字符子串
        ans = Math.max(ans, rk - i + 1);
    }
    return ans;
};
```

4、14. 最长公共前缀

1. 方法：链表
2. 当字符串数组长度为 0 时则公共前缀为空，直接返回
3. 令最长公共前缀 `ans` 的值为第一个字符串，进行初始化
4. 遍历后面的字符串，依次将其与 `ans` 进行比较，两两找出公共前缀，最终结果即为最长公共前缀
5. 如果查找过程中出现了 `ans` 为空的情况，则公共前缀不存在直接返回
6. 时间复杂度： $O(s)$ ， s 为所有字符串的长度之和

```
/**
 * @param {string[]} strs
 * @return {string}
 */
var longestCommonPrefix = function(strs) {
    if(strs.length == 0)
        return "";
    let ans = strs[0];
    for(let i = 1; i < strs.length; i++) {
        let j = 0;
        for(; j < ans.length && j < strs[i].length; j++) {
            if(ans[j] != strs[i][j])
                break;
        }
        ans = ans.substr(0, j);
        if(ans === "")
            return ans;
    }
    return ans;
};
```

5、12. 整数转罗马数字

做题思路：

1. 方法：模拟
2. 根据罗马数字的唯一表示法，为了表示一个给定的整数 `num`，我们寻找不超过 `num` 的最大符号值，将 `num` 减去该符号值，然后继续寻找不超过 `num` 的最大符号值，将该符号拼接在上一个找到的符号之后，循环直至 `num` 为 0。最后得到的字符串即为 `num` 的罗马数字表示。
3. 为了方便操作，可以建立一个数值-符号对的列表 `valueSymbols`，按数值从大到小排列。遍历 `valueSymbols` 中的每个数值-符号对，若当前数值 `value` 不超过 `num`，则从 `num` 中不断减去 `value`，直至 `num` 小于 `value`，然后遍历下一个数值-符号对。若遍历中 `num` 为 0 则跳出循环。

补充知识点：罗马数字符号

1. 罗马数字由 7 个不同的单字母符号组成，每个符号对应一个具体的数值。此外，减法规则（如问题描述中所述）给出了额外的 6 个复合符号。这给了我们总共 13 个独特的符号（每个符号由 1 个或 2 个字母组成），如下图所示。

M	→	1000		CM	→	900
D	→	500		CD	→	400
C	→	100		XC	→	90
L	→	50		XL	→	40
X	→	10		IX	→	9
V	→	5		IV	→	4
I	→	1				

2. 我们用来确定罗马数字的规则是：对于罗马数字从左到右的每一位，选择尽可能大的符号值。对于 140，最大可以选择的符号值为 C=100。接下来，对于剩余的数字 40，最大可以选择的符号值为 XL=40。因此，140 的对应的罗马数字为 C+XL=CXL。

```
var intToRoman = function(num) {  
  const valueSymbols = [  
    [1000, "M"],  
    [900, "CM"],  
    [500, "D"],  
    [400, "CD"],  
    [100, "C"],  
    [90, "XC"],  
    [50, "L"],  
    [40, "XL"],  
    [10, "X"],  
    [9, "IX"],  
    [5, "V"],  
    [4, "IV"],  
    [1, "I"]  
  ];  
  const roman = [];  
  for (const [value, symbol] of valueSymbols) {  
    while (num >= value) {  
      num -= value;  
      roman.push(symbol);  
    }  
    if (num == 0) {  
      break;  
    }  
  }  
}
```

```
    }  
  }  
  return roman.join('');  
};
```

6、面试题 01.05. 一次编辑

1. 方法：双指针
2. 双指针遍历，用一个标记位记录是否编辑过。
3. 当出现不相等的字符，就给它一次编辑的机会，后面再出现不相等就返回 `false`。
4. 编辑策略：如果两字符串长度相等，就用替换是最小编辑；不相等的话，长度长的字符串指针走一步就可以了，相当于字符串长的那个删除一个字符。
5. 时间复杂度是 $O(N)$ ，空间复杂度是 $O(1)$ 。因为双指针是同时动的，而且两字符串长度肯定是相等的或者只差1才需要比较。

```
/**  
 * @param {string} first  
 * @param {string} second  
 * @return {boolean}  
 */  
var oneEditAway = function(first, second) {  
  if(Math.abs(first.length-second.length)>1)return false;  
  let i = 0;  
  let j = 0;  
  let edit = false;  
  while(i<first.length&&j<second.length){  
    if(first.charAt(i)===second.charAt(j)){  
      i++;  
      j++;  
    }else{  
      if(!edit){  
        if(first.length===second.length){  
          i++;  
          j++;  
        }else if(first.length<second.length){  
          j++;  
        }else{  
          i++;  
        }  
        edit = true;  
      }else{  
        return false;  
      }  
    }  
  }  
}
```

```
    return true;
};
```

7、214. 最短回文串

1. 找到从 `s` 首位开始的最长回文字符串，再讲不是该串的部分颠倒拼接到 `s` 头部就得到了需要的结果
2. `s` 从 `0->s.length` 枚举 `s` 的子串
3. 判断枚举的子串是否未回文字符串

不是继续枚举

是返回 `s` 中不在该子串部分的颠倒字符 `+s`

```
/**
 * @param {string} s
 * @return {string}
 */
var shortestPalindrome = function (s) {
    let len = s.length,
        str = s.split('').reverse().join('')

    // 正反字符匹配求最长匹配位数
    function kmp(query, pattern) {
        let fail = Array(len).fill(-1)
        // 最长公共前缀，不存在公共前缀填充-1
        for (let i = 1; i < len; ++i) {
            let j = fail[i - 1]
            while (j !== -1 && pattern[j + 1] !== pattern[i]) {
                j = fail[j]
            }
            if (pattern[j + 1] === pattern[i]) {
                fail[i] = j + 1
            }
        }
    }

    // 校验 match记录匹配的位置
    let match = -1
    for (let i = 0; i < len; ++i) {
        // 如果当前不匹配，指针跳跃到前缀位置开始匹配
        while (match !== -1 && pattern[match + 1] !== query[i]) {
            match = fail[match]
        }
        // 如果当前位匹配，逐个向后匹配
        if (pattern[match + 1] === query[i]) {
            ++match
        }
    }
}
```

```

    }
    return match
}

let num = kmp(str, s),
    add = num == len - 1 ? '' : s.substring(num + 1)
return Array.from(add).reverse().join('') + s
}

```

8、1392. 最长快乐前缀

1. 如 ababab, 字符串长度为6。
2. 假设最长快乐前缀为5, 前缀为 ababa, 后缀为 babab, 两者不相等, 不满足条件。
3. 假设最长快乐前缀为4, 前缀为 abab, 后缀为 abab, 两者相等, 满足条件, 即为所求。

```

/**
 * @param {string} s
 * @return {string}
 */
var longestPrefix = function(s) {
    if (s.length <= 1) {
        return ''
    } else {
        let len = s.length - 1
        while (len > 0) {
            const prefix = s.substr(0, len)
            const postfix = s.substr(s.length - len, len)
            if (prefix === postfix) {
                return prefix
            }
            --len
        }
        return ''
    }
};

```