459. 重复的子字符串

给定一个非空的字符串，判断它是否可以由它的一个子串重复多次构成。给定的字符串只含有小写英文字母，并且长度不超过10000。

示例：

```
输入: "abab"

输出: True

解释: 可由子字符串 "ab" 重复两次构成。
```

```cpp
class Solution {
public:
    bool repeatedSubstringPattern(string s) {
        int n = s.size(), j = -1;
        vector<int> next(n);
        next[0] = -1;
        for(int i = 1; i < s.size(); i++) {
            while(j != -1 && s[j + 1] != s[i]) j = next[j];
            if(s[j + 1] == s[i]) j++;
            next[i] = j;
        }

        return (next[n - 1] != -1) && (n % (n - (next[n - 1] + 1)) == 0);
    }
};
```

```cpp
class Solution {
public:
    bool repeatedSubstringPattern(string s) {
        int pos = 0;
        char c = s[0];
        while((pos = s.find(c, pos + 1)) != s.npos) {
            if(s.size() % pos != 0) continue;
            int flag = 0;
            for(int i = pos; i < s.size(); ) {
                for(int j = 0; j < pos; j++) {
                    if(s[i] == s[j]) {
                        i++;
                    } else {
                        i++;
                        flag = 1;
                        break;
                    }
                }
            }
            if(flag == 0) return true;
```

```
        }

        if(s.size() % pos != 0) return false;
        return true;
    }
};
```

1392. 最长快乐前缀

「快乐前缀」是在原字符串中既是 非空 前缀也是后缀（不包括原字符串自身）的字符串。

给你一个字符串 `s`，请你返回它的 最长快乐前缀。

如果不存在满足题意的前缀，则返回一个空字符串。

示例：

```
输入: s = "level"
输出: "l"
解释: 不包括 s 自己，一共有 4 个前缀 ("l", "le", "lev", "leve") 和 4 个后缀 ("l", "el", "vel", "evel") 。最长的既是前缀也是后缀的字符串是 "l" 。
```

```
class Solution {
public:
    string longestPrefix(string s) {
        int n = s.size(), j = -1;
        vector<int> next(n);
        next[0] = -1;
        for(int i = 1; i < n; i++) {
            while(j != -1 && s[j + 1] != s[i]) j = next[j];
            if(s[j + 1] == s[i]) j += 1;
            next[i] = j;
        }
        return s.substr(0, next[n - 1] + 1);
    }
};
```

214. 最短回文串

给定一个字符串 ***s***，你可以通过在字符串前面添加字符将其转换为回文串。找到并返回可以用这种方式转换的最短回文串。

示例：

```
输入: s = "aacecaaa"
输出: "aaacecaaa"
```

```cpp
class Solution {
public:
    string shortestPalindrome(string s) {
        string rev_s = s;
        reverse(rev_s.begin(), rev_s.end());
        // aabc#cbaa
        rev_s = s + "#" + rev_s;

        // ------------
        int n = rev_s.size(), j = -1;
        vector<int> next(n);
        next[0] = -1;
        for(int i = 1; i < n; i++) {
            while(j != -1 && rev_s[j + 1] != rev_s[i]) j = next[j];
            if(rev_s[j + 1] == rev_s[i]) j++;
            next[i] = j;
        }
        // -------------

        rev_s = s.substr(next[rev_s.size() - 1] + 1, s.size());
        reverse(rev_s.begin(), rev_s.end());
        return rev_s + s;
    }
};
```

5. 最长回文子串

给你一个字符串 `s`，找到 `s` 中最长的回文子串。

示例：

```
输入: s = "babad"
输出: "bab"
解释: "aba" 同样是符合题意的答案。
```

```cpp
class Solution {
public:
    string getNewString(string &s) {
        string new_s = "#";
        for(int i = 0; s[i]; i++) {
            (new_s += s[i]) += "#";
        }
        return new_s;
    }
    string longestPalindrome(string s) {
        string ns = getNewString(s);
        vector<int> d(ns.size());
        int l = 0, r = -1; // 最远的r
        for(int i = 0; ns[i]; i++) {
            if(i > r) {
```

```
                d[i] = 1;
            } else {
                d[i] = min(r - i, d[l + r - i]);
            }
            // 朴素匹配算法
            while(i - d[i] >= 0 && ns[i - d[i]] == ns[i + d[i]]) {
                d[i]++;
            }
            if(i + d[i] > r && i - d[i] > 0) {
                l = i - d[i];
                r = i + d[i];
            }
        }
        string ret;
        int tmp = -1;
        for(int i = 0; ns[i]; i++) {
            if(tmp >= d[i]) continue;
            tmp = d[i];
            ret = "";
            for(int j = i - d[i] + 1; j < i + d[i]; j++) {
                if(ns[j] == '#') continue;
                ret += ns[j];
            }
        }

        return ret;
    }
};
```

28. 实现 strStr()

实现 **strStr()** 函数。

给你两个字符串 `haystack` 和 `needle` ，请你在 `haystack` 字符串中找出 `needle` 字符串出现的第一个位置（下标从 0 开始）。如果不存在，则返回 `-1` 。

说明：

当 `needle` 是空字符串时，我们应当返回什么值呢？这是一个在面试中很好的问题。

对于本题而言，当 `needle` 是空字符串时我们应当返回 0 。这与 C 语言的 **strstr()** 以及 Java 的 **indexOf()** 定义相符。

示例：

```
输入: haystack = "hello", needle = "ll"
输出: 2
```

```
class Solution {
```

```
public:
    int strStr(string haystack, string needle) {
        int ind[128];
        for(int i = 0; i < 128; i++) ind[i] = -1;
        for(int i = 0; needle[i]; i++) ind[needle[i]] = i;
        int i = 0;
        while(i + needle.size() <= haystack.size()) {
            int flag = 1;
            for(int j = 0; needle[j]; j++) {
                if(haystack[i + j] == needle[j]) continue;
                flag = 0;
                break;
            }
            if(flag) return i;
            i += (needle.size() - ind[haystack[i + needle.size()]]);
        }
        return -1;
    }
};
```

3. 无重复字符的最长子串

给定一个字符串 `s` ，请你找出其中不含有重复字符的 最长子串 的长度。

示例：

```
输入:s = "abcabcbb"
输出:3
解释: 因为无重复字符的最长子串是"abc", 所以其长度为 3。
```

```
class Solution {
public:
    bool check(int l, string &s) {
        int cnt[256] = {0}, k = 0;
        for(int i = 0; s[i]; i++) {
            if(cnt[s[i]] == 0) k += 1;
            cnt[s[i]] += 1;

            if(i >= l) {
                cnt[s[i - l]] -= 1;
                if(cnt[s[i - l]] == 0) k -= 1;
            }
            if(k == l) return true;
        }
        return false;
    }

    int lengthOfLongestSubstring(string s) {
        if(s == "") return 0;
        int head = 1, tail = s.size(), mid;
```

```
        while(head < tail) {
            mid = (head + tail + 1) >> 1;
            if(check(mid, s)) head = mid;
            else tail = mid - 1;
        }
        return head;
    }
};
```

14. 最长公共前缀

编写一个函数来查找字符串数组中的最长公共前缀。

如果不存在公共前缀，返回空字符串 `""`。

示例：

```
输入: strs = ["flower","flow","flight"]
输出: "fl"
```

```
class Solution {
public:
    string Compare(string &a, string &b) {
        string ret = "";
        for(int i = 0; a[i]; i++) {
            if(i == b.size() || a[i] != b[i]) return ret;
            ret += a[i];
        }
        return ret;
    }

    string longestCommonPrefix(vector<string>& strs) {
        string ret = strs[0];
        for(int i = 1; i < strs.size(); i++) {
            ret = Compare(ret, strs[i]);
        }
        return ret;
    }
};
```

面试题 01.05. 一次编辑

字符串有三种编辑操作:插入一个字符、删除一个字符或者替换一个字符。 给定两个字符串，编写一个函数判定它们是否只需要一次(或者零次)编辑。

示例：

```
输入:
first = "pale"
second = "ple"
输出: True
```

```
class Solution {
public:
    bool oneEditAway(string first, string second) {
        if(first.size() < second.size()) swap(first, second);
        int n = first.size(), m = second.size();
        if(abs(n - m) > 1) return false;
        if(abs(n - m) == 0) {
            for(int i = 0, j = 0; i < n; i++) {
                if(first[i] == second[i]) continue;
                j += 1;
                if(j == 2) return false;
            }
            return true;
        }

        int i = 0, j = m - 1;
        while(i <= j && first[i] == second[i]) i++;
        while(i <= j && first[j + 1] == second[j]) --j;

        return i > j;
    }
};
```

12. 整数转罗马数字

罗马数字包含以下七种字符： `I`， `V`， `X`， `L`， `C`， `D` 和 `M`。

```
字符数值
I             1
V             5
X             10
L             50
C             100
D             500
M             1000
```

例如， 罗马数字 2 写做 `II` ，即为两个并列的 1。12 写做 `XII` ，即为 `X` + `II` 。 27 写做 `XXVII`, 即为 `XX` + `V` + `II` 。

通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如 4 不写做 `IIII`，而是 `IV`。数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4 。同样地，数字 9 表示为 `IX`。这个特殊的规则只适用于以下六种情况：

- `I` 可以放在 `V` (5) 和 `X` (10) 的左边，来表示 4 和 9。
- `X` 可以放在 `L` (50) 和 `C` (100) 的左边，来表示 40 和 90。
- `C` 可以放在 `D` (500) 和 `M` (1000) 的左边，来表示 400 和 900。

给你一个整数，将其转为罗马数字。

示例 **1:**

```
输入: num = 3
输出: "III"
```

```
class Solution {
public:
    string RomanString(int d, char a, char b, char c) {
        string ret;
        if(d == 4 || d == 9) {
            ret += a;
            d == 4 ? ret += b : ret += c;
            return ret;
        }
        if(d >= 5) ret += b, d -= 5;
        while(d--) {
            ret += a;
        }
        return ret;
    }
    string output(int &num) {
        if(num >= 1000) {
            int d = num / 1000;
            num -= d * 1000;
            return RomanString(d, 'M', 'M', 'M');
        } else if(num >= 100) {
            int d = num / 100;
            num -= d * 100;
            return RomanString(d, 'C', 'D', 'M');
        } else if(num >= 10) {
            int d = num / 10;
            num -= d * 10;
            return RomanString(d, 'X', 'L', 'C');
        } else {
            int d = num;
            num = 0;
            return RomanString(d, 'I', 'V', 'X');
        }
        return "";
    }
    string intToRoman(int num) {
        string ret;
        while(num) {
            ret += output(num);
        }
```

```
41          return ret;
42      }
43  };
```