目录

第 1 章
String

1.1 Reverse String

Description

Write a function that takes a string as input and returns the string reversed.

Example:

Given s = `"hello"`, return `"olleh"`.

Solution

```
public String reverseString(String s) {
    return new StringBuilder(s).reverse().toString();
}
```

1.2 Reverse String II

Description

Given a string and an integer k, you need to reverse the first k characters for every 2k characters counting from the start of the string. If there are less than k characters left, reverse all of them. If there are less than 2k but greater than or equal to k characters, then reverse the first k characters and left the other as original.

Example:

Input: s = `"abcdefg"`, k = 2

Output: `"bacdfeg"`

Restrictions:

The string consists of lower English letters only.

Length of the given string and k will in the range `[1, 10000]`

Solution

```
public String reverseStr(String s, int k) {
    if (s.length() <= k) {
        return new StringBuilder(s).reverse().toString();
    }
    if (s.length() <= 2 * k) {
        return reverseStr(s.substring(0, k), k) + s.substring(k);
    }
    return reverseStr(s.substring(0, 2 * k), k) + reverseStr(s.substring(2 * k), k);
}
```

1.3 Reverse Words in a String

Description

Given an input string, reverse the string word by word.

For example, Given s = ”the sky is blue”, return ”blue is sky the”.

Clarification:

1. What constitutes a word?

A sequence of non-space characters constitutes a word.

2. Could the input string contain leading or trailing spaces?

Yes. However, your reversed string should not contain leading or trailing spaces.

3. How about multiple spaces between two words?

Reduce them to a single space in the reversed string.

Solution I

```
// 耗时 10ms
public String reverseWords(String s) {
    StringBuilder sb = new StringBuilder();

    boolean inWord = false;
    char[] cc = s.toCharArray();
    for (int i = cc.length - 1, idx = 0; i >= 0; i--) {
        if (cc[i] != ' ') {
            if (!inWord && sb.length() > 0) {
                sb.append(' ');
            }
            inWord = true;
            sb.insert(idx, cc[i]);
        } else if (inWord) {
            idx = sb.length() + 1;
            inWord = false;
        }
    }

    return sb.toString();
}
```

Solution II

```
// 耗时 18ms
public String reverseWords2(String s) {
    StringBuilder sb = new StringBuilder();

    for (int i = 0, j = 0; i < s.length(); ) {
        if (s.charAt(i) == ' ') {
            i++;
            j = i;
        } else if (j >= s.length() || s.charAt(j) == ' ') {
            sb.insert(0, s.substring(i, j) + " ");
            i = j;
        } else {
            j++;
        }
    }

    if (sb.length() > 0) {
        sb.setLength(sb.length() - 1);
    }

    return sb.toString();
}
```

1.4 Reverse Words in a String II

Description

Given an input string, reverse the string word by word. A word is defined as a sequence of non-space characters.

The input string does not contain leading or trailing spaces and the words are always separated by a single space.

For example,

Given s = `"the sky is blue"`,

return `"blue is sky the"`.

Could you do it in-place without allocating extra space?

Solution

```
public void reverseWords(char[] s) {
    for (int i = 0, j = i; i < s.length; ) {
        if (j >= s.length || s[j] == ' ') {
            reverse(s, i, j - 1);
            for (i = j; j < s.length && s[j] == ' '; j++, i = j) ;
        } else {
            j++;
        }
    }
    reverse(s, 0, s.length - 1);
}

private void reverse(char[] s, int start, int end) {
    for (int i = start, j = end; i < j; i++, j--) {
        char t = s[i];
        s[i] = s[j];
        s[j] = t;
    }
}
```

1.5 Reverse Words in a String III

Description

Given a string, you need to reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

Example 1:

Input: `"Let's take LeetCode contest"`

Output: `"s'teL ekat edoCteeL tsetnoc"`

Note: In the string, each word is separated by single space and there will not be any extra space in the string.

Solution

```
public String reverseWords(String s) {
    String[] texts = s.split(" ");
    StringBuilder sb = new StringBuilder();
    for (String text : texts) {
        if (text.length() > 0) {
            sb.append(new StringBuilder(text).reverse().toString()).append(" ");
        }
    }
    return sb.toString().trim();
}
```

1.6 Reverse Vowels of a String

Description

Write a function that takes a string as input and reverse only the vowels of a string.

Example 1:

Given s = `"hello"`, return `"holle"`.

Example 2:

Given s = `"leetcode"`, return `"leotcede"`.

Note:

The vowels does not include the letter `"y"`.

Solution

```
public String reverseVowels(String s) {
    boolean[] flag = new boolean[256];
    for (char c : "aeiouAEIOU".toCharArray()) {
        flag[c] = true;
    }

    StringBuilder sb = new StringBuilder(s);
    for (int i = 0, j = sb.length() - 1; i < j; ) {
        if (!flag[sb.charAt(i)]) {
            i++;
        } else if (!flag[sb.charAt(j)]) {
            j--;
        } else {
            char c = sb.charAt(i);
            sb.setCharAt(i, sb.charAt(j));
            sb.setCharAt(j, c);
            i++;
            j--;
        }
    }

    return sb.toString();
}
```

1.7 Longest Substring Without Repeating Characters

Description

Given a string, find the length of the longest substring without repeating characters.

Examples:

Given `"abcabcbb"`, the answer is `"abc"`, which the length is 3.

Given `"bbbbb"`, the answer is `"b"`, with the length of 1.

Given `"pwwkew"`, the answer is `"wke"`, with the length of 3. Note that the answer must be a substring, `"pwke"` is a subsequence and not a substring.

Solution

```
// 耗时 39ms, 性能挺好
public int lengthOfLongestSubstring(String s) {
    int[] count = new int[256];

    int maxLen = 0;

    for (int i = 0, j = 0; j < s.length(); ) {
        if (count[s.charAt(j)] == 0) {
            count[s.charAt(j)]++;
            maxLen = Math.max(maxLen, j - i + 1);
            j++;
        } else {
            --count[s.charAt(i++)];
        }
    }

    return maxLen;
}
```

1.8 Longest Substring with At Most Two Distinct Characters

Description

Given a string, find the length of the longest substring T that contains at most 2 distinct characters.

For example, Given s = “eceba”,

T is "ece" which its length is 3.

Solution

```
// 7ms
public int lengthOfLongestSubstringTwoDistinct2(String s) {
    int[] count = new int[256];
    int distinct = 0, longest = 0;

    for (int i = 0, j = 0; j < s.length(); j++) {
        if (count[s.charAt(j)]++ == 0) {
            distinct++;
        }

        for ( ; i < j && distinct > 2; ) {
            if (--count[s.charAt(i++)] == 0) {
                --distinct;
            }
        }

        longest = Math.max(longest, j - i + 1);
    }

    return longest;
}
```

1.9 Longest Substring with At Most K Distinct Characters

Description

Given a string, find the length of the longest substring T that contains at most k distinct characters.

For example, Given s = “eceba” and k = 2,

T is "ece" which its length is 3.

Solution

```
/**
 * 思路跟 Longest Substring With At Most Two Distinct Characters 一样，只是给 2 改成 k，
 * 要注意 k 等于 0 时返回 0
 */
// 耗时 9ms
public int lengthOfLongestSubstringKDistinct(String s, int k) {
    if (k == 0) {
        return 0;
    }

    int[] count = new int[256];
    int distinct = 0, longest = 0;

    for (int i = 0, j = 0; j < s.length(); j++) {
        if (count[s.charAt(j)]++ == 0) {
            distinct++;
        }

        for ( ; i < j && distinct > k; ) {
            if (--count[s.charAt(i++)] == 0) {
                --distinct;
            }
        }

        longest = Math.max(longest, j - i + 1);
    }

    return longest;
}
```

1.10 Minimum Window Substring

Description

Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

For example,

S = "ADOBECODEBANC"

T = "ABC"

Minimum window is "BANC".

Note:

If there is no such window in S that covers all characters in T, return the empty string "".

If there are multiple such windows, you are guaranteed that there will always be only one unique minimum window in S.

Solution

```
// 耗时 8ms, 时间复杂度 O(n)
public String minWindow(String s, String t) {
    int[] sc = new int[256], tc = new int[256];

    for (char c : t.toCharArray()) {
        tc[c]++;
    }

    int minStart = 0, minLen = Integer.MAX_VALUE;

    for (int i = 0, j = 0, k = 0; j < s.length(); j++) {
        char c = s.charAt(j);

        if (++sc[c] <= tc[c]) {
            ++k;
        }

        if (k == t.length()) {
            for (; i < j; i++) {
                char cc = s.charAt(i);

                if (tc[cc] == 0) {
                    continue;
                }
                if (sc[cc] <= tc[cc]) {
                    break;
                }
                sc[cc]--;
            }

            if (j - i + 1 < minLen) {
                minLen = j - i + 1;
                minStart = i;
            }
        }
    }

    return minLen != Integer.MAX_VALUE ? s.substring(minStart, minStart + minLen) : "";
}
```

1.11 Sliding Window Maximum

Description

Given an array nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.

For example,

Given nums = `[1,3,-1,-3,5,3,6,7]`, and k = 3.

```
Window position                Max
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

Therefore, return the max sliding window as `[3,3,5,5,6,7]`.

Note:

You may assume k is always valid, ie: $1 \le k \le$ input array's size for non-empty array.

Follow up:

Could you solve it in linear time?

Solution I

```
// 注意 PriorityQueue 的 remove 复杂度是 O(k), 所以本题复杂度是 O(n*k)
// 可以将 PriorityQueue 转成 TreeMap, 复杂度为 O(n*lgk), 耗时 58ms
public int[] maxSlidingWindow(int[] nums, int k) {
    if (nums.length == 0) {
        return new int[0];
    }

    Queue<Integer> queue = new PriorityQueue<Integer>(new Comparator<Integer>() {
        @Override
        public int compare(Integer i1, Integer i2) {
            return i2 - i1;
        }
    });

    for (int i = 0; i < k; i++) {
        queue.offer(nums[i]);
    }

    int[] result = new int[nums.length - k + 1];
    result[0] = queue.peek();

    for (int i = 1; i < result.length; i++) {
        queue.remove(nums[i - 1]);
        queue.offer(nums[i + k - 1]);
        result[i] = queue.peek();
    }

    return result;
}
```

Solution II

```
// 耗时 23ms
// queue 中存的是索引
public int[] maxSlidingWindow2(int[] nums, int k) {
    if (nums.length == 0) {
        return new int[0];
    }

    Deque<Integer> queue = new LinkedList<Integer>();

    int[] result = new int[nums.length - k + 1];

    for (int i = 0; i < nums.length; i++) {
        if (!queue.isEmpty() && queue.peek() <= i - k) {
            queue.removeFirst();
        }

        while (!queue.isEmpty() && nums[queue.peekLast()] < nums[i]) {
            queue.pollLast();
        }

        queue.offerLast(i);

        if (i >= k - 1) {
            result[i - k + 1] = nums[queue.peek()];
        }
    }

    return result;
}
```

1.12 Longest Palindromic Substring

Description

Given a string s, find the longest palindromic substring in s. You may assume that the maximum length of s is 1000.

Example:

Input: `"babad"`

Output: `"bab"`

Note: `"aba"` is also a valid answer.

Example:

Input: `"cbbd"`

Output: `"bb"`

Solution I

```
private int begin, maxLen;

// 耗时 14ms, 平均复杂度 O(n)
public String longestPalindrome(String s) {
    for (int i = 0; i < s.length(); i++) {
        helper(s, i, i);
        helper(s, i, i + 1);
    }
    return s.substring(begin, begin + maxLen);
}

private void helper(String s, int i, int j) {
    for (; i >= 0 && j < s.length() && s.charAt(i) == s.charAt(j); i--, j++) ;
    int len = j - i - 1;
    if (len > maxLen) {
        maxLen = len;
        begin = i + 1;
    }
}
```

Solution II

```
// 动态规划, 耗时 73ms, 复杂度 O(n^2)
public String longestPalindrome2(String s) {
    int len = s.length();

    if (len == 0) {
        return s;
    }

    boolean[][] f = new boolean[len][len];

    int index = 0;
    int max = 1;

    for (int i = 0; i < len; i++) {
        for (int j = 0; j <= i; j++) {
            if (s.charAt(j) == s.charAt(i) && (i - j < 2 || f[j + 1][i - 1])) {
                f[j][i] = true;

                if (i - j + 1 > max) {
                    max = i - j + 1;
                    index = j;
                }
            }
        }
    }

    return s.substring(index, index + max);
}
```

1.13 Shortest Palindrome

Description

Given a string S, you are allowed to convert it to a palindrome by adding characters in front of it. Find and return the shortest palindrome you can find by performing this transformation.

For example:

Given "aacecaaa", return "aaacecaaa".

Given "abcd", return "dcbabcd".

Solution

```
/**
 * 其实只要将 s 与 s 的逆序串拼接在一起，求最长公共子串，逆序串中刨除这个最长公共子串，
 * 就是要加到 s 前面的
 */
public String shortestPalindrome(String s) {
    StringBuilder builder = new StringBuilder(s);
    return builder.reverse().substring(0, s.length() - getCommonLength(s)) + s;
}

private int getCommonLength(String str) {
    StringBuilder builder = new StringBuilder(str);
    String rev = new StringBuilder(str).reverse().toString();
    builder.append("#").append(rev);
    int[] p = new int[builder.length()];
    for (int i = 1; i < p.length; i++) {
        int j = p[i - 1];
        while (j > 0 && builder.charAt(i) != builder.charAt(j)) j = p[j - 1];
        p[i] = j == 0 ? (builder.charAt(i) == builder.charAt(0) ? 1 : 0) : j + 1;
    }
    return p[p.length - 1];
}

/**
 * 更直观的写法
 */
private int getCommonLength2(String str) {
    String rev = new StringBuilder(str).reverse().toString();
    return getCommonLength(str + rev, str.length());
}

private int getCommonLength(String s, int max) {
    for (int i = s.length() - max; i < s.length(); i++) {
        if (s.startsWith(s.substring(i))) {
            return s.length() - i;
        }
    }
    return 0;
}
```

1.14 Count and Say

Description

The count-and-say sequence is the sequence of integers with the first five terms as following:

```
1.     1
2.     11
3.     21
4.     1211
5.     111221
```

1 is read off as `"one 1"` or 11.

11 is read off as `"two 1s"` or 21.

21 is read off as `"one 2, then one 1"` or 1211.

Given an integer n, generate the nth term of the count-and-say sequence.

Note: Each term of the sequence of integers will be represented as a string.

Solution

```
public String countAndSay(int n) {
    String s = "1";
    for (int i = 2; i <= n; i++) {
        s = next(s);
    }
    return s;
}

private String next(String s) {
    StringBuilder sb = new StringBuilder();

    char cur = 0;
    int times = 0;

    for (int i = 0; i < s.length(); i++) {
        if (times == 0) {
            cur = s.charAt(i);
            times = 1;
        } else if (s.charAt(i) == cur) {
            times++;
        } else {
            sb.append(String.format("%d%c", times, cur));
            cur = s.charAt(i);
            times = 1;
        }
    }

    // 这一句千万别掉了
    if (times != 0) {
        sb.append(String.format("%d%c", times, cur));
    }

    return sb.toString();
}
```

1.15 Valid Parentheses

Description

Given a string containing just the characters `'('`, `')'`, `''`, `''`, `'['` and `']'`, determine if the input string is valid.

The brackets must close in the correct order, `"()"` and `"()[]"` are all valid but `"(]"` and `"([)]"` are not.

Solution

```
// 耗时 5ms
public boolean isValid(String s) {
    char[] stack = new char[s.length()];
    int top = -1;

    for (char c : s.toCharArray()) {
        switch (c) {
            case ')':
                if (top >= 0 && stack[top] == '(') {
                    top--;
                } else {
                    return false;
                }
                break;
            case '}':
                if (top >= 0 && stack[top] == '{') {
                    top--;
                } else {
                    return false;
                }
                break;
            case ']':
                if (top >= 0 && stack[top] == '[') {
                    top--;
                } else {
                    return false;
                }
                break;
            default:
                stack[++top] = c;
                break;
        }
    }

    return top < 0;
}
```

1.16 Group Anagrams

Description

Given an array of strings, group anagrams together.

For example, given: `["eat", "tea", "tan", "ate", "nat", "bat"]`,

Return:

```
[
  ["ate", "eat","tea"],
  ["nat","tan"],
  ["bat"]
]
```

Note: All inputs will be in lower-case.

Solution

```
public List<List<String>> groupAnagrams(String[] strs) {
    HashMap<String, List<String>> map = new HashMap<>();

    for (String s : strs) {
        char[] cc = s.toCharArray();
        Arrays.sort(cc);

        String t = new String(cc);

        List<String> list = map.get(t);
        if (list == null) {
            list = new LinkedList<>();
            map.put(t, list);
        }

        list.add(s);
    }

    return new LinkedList<>(map.values());
}
```

1.17 Add Binary

Description

Given two binary strings, return their sum (also a binary string).

For example,

a = "11" b = "1" Return "100".

Solution

```
public String addBinary(String a, String b) {
    StringBuilder sb = new StringBuilder();
    int i = a.length() - 1, j = b.length() - 1, k = 0;
    for ( ; i >= 0 || j >= 0 || k > 0; i--, j--) {
        int i0 = i >= 0 ? a.charAt(i) - '0' : 0;
        int j0 = j >= 0 ? b.charAt(j) - '0' : 0;
        int s = i0 + j0 + k;
        sb.insert(0, s & 1);
        k = s >> 1;
    }
    return sb.toString();
}
```

1.18 String to Integer (atoi)

Description

Implement atoi to convert a string to an integer.

Hint: Carefully consider all possible input cases. If you want a challenge, please do not see below and ask yourself what are the possible input cases.

Notes: It is intended for this problem to be specified vaguely (ie, no given input specs). You are responsible to gather all the input requirements up front.

Analysis

1. 过滤掉前面的空格
2. 考虑正负号
3. 如果溢出则返回上限或下限
4. 解析时遇到非法字符则停止，返回当前结果
5，防御空串

Solution

```
public int myAtoi(String str) {
    long n = 0, sign = 1;

    str = str.trim();

    // 这里要防御空串
    if (str.length() == 0) { return 0; }

    switch (str.charAt(0)) {
        case '-':
            sign = -1;
        case '+':
            str = str.substring(1);
            break;
    }

    for (char c : str.toCharArray()) {
        if (c >= '0' && c <= '9') {
            n = n * 10 + (c - '0');

            if (n * sign > Integer.MAX_VALUE) {
                return Integer.MAX_VALUE;
            } else if (n * sign < Integer.MIN_VALUE) {
                return Integer.MIN_VALUE;
            }
        } else {
            break;
        }
    }
    return (int) (n * sign);
}
```

1.19 Implement strStr()

Description

Implement strStr().

Returns the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

Solution

```
// 这里非常重要的是 i<=len1-len2, 如果没有这个会超时
public int strStr(String haystack, String needle) {
    int l1 = haystack.length(), l2 = needle.length();
    for (int i = 0, j; i + l2 - 1 < l1; i++) {
        for (j = 0; j < l2; j++) {
            if (haystack.charAt(i + j) != needle.charAt(j)) {
                break;
            }
        }
        if (j >= l2) {
            return i;
        }
    }
    return -1;
}
```

1.20 Integer to English Words

Description

Convert a non-negative integer to its english words representation. Given input is guaranteed to be less than 231 - 1.

For example,

```
123 -> "One Hundred Twenty Three"
12345 -> "Twelve Thousand Three Hundred Forty Five"
1234567 -> "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"
```

Solution

```
private final String[] LESS_20 = {
        "", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Eleven",
        "Twelve", "Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"
};

private final String[] LESS_100 = {
        "", "Ten", "Twenty", "Thirty", "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Ninety"
};

private final String[] THOUSANDS = {
        "", "Thousand", "Million", "Billion"
};

/**
 * 1, 别漏了 zero 的情况
 * 2,  当 num % 1000 != 0 的判断别掉了
 * 3,  helper 的返回结果要 trim 一下, 去掉前后多余的空格
 * 4,  最后返回的 sb 要 trim 一下
 */
public String numberToWords(int num) {
    if (num == 0) {
        return "Zero";
    }
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < THOUSANDS.length && num > 0; i++) {
        if (num % 1000 != 0) {
            sb.insert(0, helper(num % 1000).trim() + " " + THOUSANDS[i] + " ");
        }

        num /= 1000;
    }
    return sb.toString().trim();
}

private String helper(int num) {
    if (num < 20) {
        return LESS_20[num];
    } else if (num < 100) {
        return LESS_100[num / 10] + " " + helper(num % 10);
    } else {
        return LESS_20[num / 100] + " Hundred " + helper(num % 100);
    }
}
```

1.21 Multiply Strings

Description

Given two non-negative integers num1 and num2 represented as strings, return the product of num1 and num2.

Note:

1. The length of both num1 and num2 is < 110.
2. Both num1 and num2 contains only digits 0-9.
3. Both num1 and num2 does not contain any leading zero.
4. You must not use any built-in BigInteger library or convert the inputs to integer directly.

Solution

```
public String multiply(String num1, String num2) {
    int len1 = num1.length(), len2 = num2.length();
    int[] result = new int[len1 + len2];
    for (int i = len1 - 1; i >= 0; i--) {
        for (int j = len2 - 1; j >= 0; j--) {
            int res = result[i + j + 1] + (num1.charAt(i) - '0') * (num2.charAt(j) - '0');
            result[i + j + 1] = res % 10;
            result[i + j] += res / 10;
        }
    }

    StringBuilder sb = new StringBuilder();
    for (int n : result) {
        // 这里要去掉头部的"0"
        if (n == 0 && sb.length() == 0) {
            continue;
        }
        sb.append(n);
    }

    /**
     * 这里要注意如果是空要返回 0
     */
    return sb.length() == 0 ? "0" : sb.toString();
}
```

1.22 Compare Version Numbers

Description

Compare two version numbers version1 and version2.

If version1 > version2 return 1, if version1 < version2 return -1, otherwise return 0.

You may assume that the version strings are non-empty and contain only digits and the `.` character.

The `.` character does not represent a decimal point and is used to separate number sequences.

For instance, 2.5 is not `"two and a half"` or `"half way to version three"`, it is the fifth second-level revision of the second first-level revision.

Here is an example of version numbers ordering:

```
0.1 < 1.1 < 1.2 < 13.37
```

Solution