LeetCode 杂鱼的末日篇

为保证题目的完整性，在此将没有分类的题目的解法简单介绍。

1. Largest Number，。

```
    public String largestNumber(int[] num) {
        if(num == null || num.length == 0){
            return "";
        }
        String[] cache = new String[num.length];
        for(int i = 0; i < num.length; i++){
            cache[i] = String.valueOf(num[i]);
        }
        Arrays.sort(cache, new Comparator<String>(){
            public int compare(String s1, String s2){
                return (s1 + s2).compareTo(s2 + s1);
            }
        });
        if(cache[cache.length - 1].equals("0")){
            return "0";
        }
        StringBuilder res = new StringBuilder();
        for(int i = cache.length - 1; i >= 0; i--){
            res.append(cache[i]);
        }
        return res.toString();
    }
```

2. Compare Version Numbers，。

```
    public int compareVersion(String version1, String version2) {
        String[] ver1 = version1.split("\\.");
        String[] ver2 = version2.split("\\.");
        for(int i = 0; i < Math.max(ver1.length, ver2.length); i++){
            int gap = (i < ver1.length ? Integer.parseInt(ver1[i]) : 0) - (i < ver2.length ?
Integer.parseInt(ver2[i]) : 0);
            if(gap != 0){
                return gap > 0 ? 1 : -1;
            }
        }
        return 0;
    }
```

3. Maximum Gap， 。

```
    public int maximumGap(int[] num) {
        if(num == null || num.length < 2){
            return 0;
        }

        // Find the min and max elements in the list.
        int min = Integer.MAX_VALUE;
        int max = Integer.MIN_VALUE;
        for(int elem : num){
            min = Math.min(min, elem);
            max = Math.max(max, elem);
        }

        // Put the n elements into (n-1) buckets.
        double div = (max - min) * 1.0 / (num.length - 1);

        // bucket[i]    : min value in the bucket i/2;
        // bucket[i+1]: max value in the bucket i/2;
        // Note: the elements are all non-negatives.
        int[] buckets = new int[num.length * 2];
        for(int elem : num){
            int i = (int)((elem - min) / div) * 2;
            buckets[i] = buckets[i] == 0 ? elem : Math.min(elem, buckets[i]);
            buckets[i + 1] = buckets[i + 1] == 0 ? elem : Math.max(elem, buckets[i + 
1]);
        }

        // Calculate the maximum distance between buckets,
        // which is aslo the maximum gap between elements.
        int last = min;
        int maxGap = Integer.MIN_VALUE;
        for(int i = 0; i < buckets.length; i += 2){
            // no element in this bucket.
            if(buckets[i] == 0){
                continue;
            }
            maxGap = Math.max(maxGap, buckets[i] - last);
            last = buckets[i + 1];
        }
        return maxGap;
    }
```

4. Pascal’s Triangle， 。

```
    public ArrayList<ArrayList<Integer>> generate(int numRows) {
        ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();
        if(numRows <= 0){
            return res;
        }
        ArrayList<Integer> item = new ArrayList<Integer>();
        ArrayList<Integer> pre = item;
        item.add(1);
        res.add(item);
        for(int i = 1; i < numRows; i++){
            item = new ArrayList<Integer>();
            item.add(1);
            for(int j = 1; j < i; j++){
                item.add(pre.get(j - 1) + pre.get(j));
            }
            item.add(1);
            pre = item;
            res.add(item);
        }
        return res;
    }
```

Part 2 – Sequence Dynamic Programming

这类题目是动态规划当中中等难度的，递推量相对固定，但递推式需要根据题目的要求做出相应的变化。同样当我们明确了4个主要元素，问题就会迎刃而解。

1）确定递推量。res[i]一般表示第i个位置/数字/字母的状态；

2）推导递推式。一般可以研究res[i]之前的res[j]对res[i]的影响；

3）计算初始条件。res[0]/res[n - 1]；

4）考虑存储历史信息的空间维度（可选）。一般就是一维空间，不可优化。

在 LeetCode 和 LintCode 中关于 Sequence DP 的题目有：

1. Climbing Stairs
2. Jump Game
3. Jump Game II
4. Palindrome Partitioning II
5. Word Break
6. Longest Increasing Subsequence

1. Pascal's Triangle II，。

```
public ArrayList<Integer> getRow(int rowIndex) {
        ArrayList<Integer> res = new ArrayList<Integer>();
        if(rowIndex < 0){
                return res;
        }
        res.add(1);
        for(int i = 1; i <= rowIndex; i++){
                for(int j = i - 1; j > 0; j--){
                        res.set(j, res.get(j) + res.get(j - 1));
                }
                res.add(1);
        }
        return res;
}
```

2. Evaluate Reverse Polish Notation，。

```
public int evalRPN(String[] tokens) {
        if(tokens == null || tokens.length == 0){
                return 0;
        }
        LinkedList<Integer> stack = new LinkedList<Integer>();
        for(int i = 0; i < tokens.length; i++){
                switch(tokens[i]){
                        case "+":
                                stack.push(stack.pop() + stack.pop());
                                break;
                        case "-":
                                stack.push(-stack.pop() + stack.pop());
                                break;
                        case "*":
                                stack.push(stack.pop() * stack.pop());
                                break;
                        case "/":
                                int divisor = stack.pop();
                                int dividend = stack.pop();
                                stack.push(dividend / divisor);
                                break;
                        default:
                                stack.push(Integer.parseInt(tokens[i]));
                }
        }
        return stack.pop();
}
```

3. Spiral Matrix，。

```
public ArrayList<Integer> spiralOrder(int[][] matrix) {
    ArrayList<Integer> res = new ArrayList<Integer>();
    if(matrix == null || matrix.length == 0 || matrix[0].length == 0){
        return res;
    }
    int min = Math.min(matrix.length, matrix[0].length);
    int layerNum = min / 2;
    for(int layer = 0; layer < layerNum; layer++){
        for(int i = layer; i < matrix[0].length - 1 - layer; i++){
            res.add(matrix[layer][i]);
        }
        for(int i = layer; i < matrix.length - 1 - layer; i++){
            res.add(matrix[i][matrix[0].length - 1 - layer]);
        }
        for(int i = matrix[0].length - 1 - layer; i > layer; i--){
            res.add(matrix[matrix.length - 1 - layer][i]);
        }
        for(int i = matrix.length - 1 - layer; i > layer; i--){
            res.add(matrix[i][layer]);
        }
    }
    if((min & 1) == 1){
        if(matrix.length == min){
            for(int i = layerNum; i <= matrix[0].length - 1 - layerNum; i++){
                res.add(matrix[layerNum][i]);
            }
        } else {
            for(int i = layerNum; i <= matrix.length - 1 - layerNum; i++){
                res.add(matrix[i][layerNum]);
            }
        }
    }
    return res;
}
```

4. Spiral Matrix II，。

```
public int[][] generateMatrix(int n) {
    int[][] res = new int[n][n];
    if(n <= 0){
        return res;
    }
    int layerNum = n / 2;
    int num = 1;
    for(int layer = 0; layer < layerNum; layer++){
        for(int i = layer; i < n - 1 - layer; i++){
            res[layer][i] = num++;
        }
        for(int i = layer; i < n - 1 - layer; i++){
            res[i][n - 1 - layer] = num++;
        }
        for(int i = n - 1 - layer; i > layer; i--){
            res[n - 1 - layer][i] = num++;
        }
        for(int i = n - 1 - layer; i > layer; i--){
            res[i][layer] = num++;
        }
    }
    if(n % 2 == 1){
        res[layerNum][layerNum] = num;
    }
    return res;
}
```

4. Rotate Image，。

```
    public void rotate(int[][] matrix) {
        if(matrix == null || matrix.length == 0 || matrix[0].length == 0){
            return;
        }
        int n = matrix.length;
        int layerNum = n / 2;
        for(int layer = 0; layer < layerNum; layer++){
            for(int i = layer; i < n - 1 - layer; i++){
                int temp = matrix[layer][i];
                matrix[layer][i] = matrix[n - 1 - i][layer];
                matrix[n - 1 - i][layer] = matrix[n - 1 - layer][n - 1 - i];
                matrix[n - 1 - layer][n - 1 - i] = matrix[i][n - 1 - layer];
                matrix[i][n - 1 - layer] = temp;
            }
        }
    }
```

5. Remove Element，。

```
    public int removeElement(int[] A, int elem) {
        if(A == null || A.length == 0){
            return 0;
        }
        int idx = 0;
        for(int i = 0; i < A.length; i++){
            if(A[i] != elem){
                A[idx++] = A[i];

            }
        }
        return idx;
    }
```

6. Valid Sudoku， 。

```
public boolean isValidSudoku(char[][] board) {
    if(board == null || board.length != 9 || board[0].length != 9){
        return false;
    }
    HashSet<Integer> set = new HashSet<Integer>();
    for(int i = 0; i < 9; i++){
        set.clear();
        for(int j = 0; j < 9; j++){
            if(board[i][j] != '.'){
                int value = board[i][j] - '0';
                if(set.contains(value)){
                    return false;
                } else {
                    set.add(value);
                }
            }
        }
    }
    for(int j = 0; j < 9; j++){
        set.clear();
        for(int i = 0; i < 9; i++){
            if(board[i][j] != '.'){
                int value = board[i][j] - '0';
                if(set.contains(value)){
                    return false;
                } else {
                    set.add(value);
                }
            }
        }
    }
    for(int block = 0; block < 9; block++){
        set.clear();
        for(int i = block / 3 * 3; i < block / 3 * 3 + 3; i++){
            for(int j = block % 3 * 3; j < block % 3 * 3 + 3; j++){
                if(board[i][j] != '.'){
                    int value = board[i][j] - '0';
                    if(set.contains(value)){
                        return false;
                    } else {
                        set.add(value);
                    }
                }
            }
        }
    }
    return true;
}
```

Part 3 – Two Sequences Dynamic Programming

这类题目是动态规划乃至全LeetCode题目当中最难的题目，4个主要元素完全不固定，递推量和递推式都要因题而异而且不好确定。但做多了还是有些许规律可循的，大部分是针对字符串处理的动态规划题目，理清s字符串第i个字符和p字符串第j个字符之间的关系即可。

1）确定递推量。res[i][j]表示第一个字符串前i个字符配上第二个字符串前j个字符的状态；

2）推导递推式。res[i][j]一般研究第i个字符和第j个字符的匹配关系；

3）计算初始条件。res[i][0]/res[0][i]；

4）考虑存储历史信息的空间维度（可选）。一般就是一维空间，不可优化。

在 LeetCode 和 LintCode 中关于 Two Sequences DP 的题目有：

1. Longest Common Subsequence
2. Longest Common Substring
3. Edit Distance
4. Distinct Subsequence
5. Interleaving String

1. Length of Last Word，。

```
public int lengthOfLastWord(String s) {
    if(s == null || s.length() == 0){
        return 0;
    }
    int end = s.length() - 1;
    while(end >= 0 && s.charAt(end) == ' '){
        end--;
    }
    int start = end;
    while(start >= 0 && s.charAt(start) != ' '){
        start--;
    }
    return end - start;
}
```

2. Container With Most Water， 。

```
    public int maxArea(int[] height) {
        if(height == null || height.length == 0){
            return 0;
        }
        int maxArea = 0;
        int left = 0;
        int right = height.length - 1;
        while(left < right){
            int min = Math.min(height[left], height[right]);
            maxArea = Math.max(maxArea, min * (right - left));
            if(height[left] == min){
                left++;
            } else {
                right--;
            }
        }
        return maxArea;
    }
```

3. Trapping Rain Water， 。

```
    // Solution 1 - Left & Right Search
    public int trap(int[] A) {
        if(A == null || A.length == 0){
            return 0;
        }
        int res = 0;
        int max = 0;
        int[] left = new int[A.length];
        for(int i = 0; i < A.length; i++){
            left[i] = max;
            max = Math.max(max, A[i]);
        }
        max = 0;
        for(int i = A.length - 1; i >= 0; i--){
            int min = Math.min(left[i], max);
            max = Math.max(max, A[i]);
            if(min > A[i]){
                res += min - A[i];
            }
        }
        return res;
    }

    // Solution 2 – Two Pointers
    public int trap(int[] A) {
        if(A == null || A.length == 0){
            return 0;
        }
        int res = 0;
        int left = 0;
        int right = A.length - 1;
        while(left < right){
            int min = Math.min(A[left], A[right]);
            if(min == A[left]){
                left++;
                while(left < right && A[left] <= min){
                    res += min - A[left];
                    left++;
                }
            } else {
                right--;
                while(left < right && A[right] <= min){
                    res += min - A[right];
                    right--;
                }
            }
        }
        return res;
    }
```

4. Longest Palindromic Substring， 。

```
// Solution 1 - Brute Force
public String longestPalindrome(String s) {
    if(s == null || s.length() == 0){
        return "";
    }
    String res = "";
    for(int k = 0; k < s.length() * 2 - 1; k++){
        int i = k / 2;
        int j = k / 2;
        if((k & 1) == 1){
            j++;
        }
        String item = helper(s, i, j);
        if(item.length() > res.length()){
            res = item;
        }
    }
    return res;
}

private String helper(String s, int left, int right){
    while(left >= 0 && right <= s.length() - 1 && s.charAt(left) == s.charAt(right)){
        left--;
        right++;
    }
    return s.substring(left + 1, right);
}

// Solution 2 - Dynamic Programming
public String longestPalindrome(String s) {
    if(s == null || s.length() == 0){
        return "";
    }
    String res = "";
    boolean[][] map = new boolean[s.length()][s.length()];
    for(int i = s.length() - 1; i >= 0; i--){
        for(int j = i; j < s.length(); j++){
            if(s.charAt(i) == s.charAt(j) && (j - i <= 2 || map[i + 1][j - 1])){
                map[i][j] = true;
                if(j - i + 1 > res.length()){
                    res = s.substring(i, j + 1);
                }
            }
        }
    }
    return res;
}
```

5. Set Matrix Zeroes， 。

```
public void setZeroes(int[][] matrix) {
    if(matrix == null || matrix.length == 0 || matrix[0].length == 0){
        return;
    }
    boolean firstRow = false;
    boolean firstCol = false;
    for(int i = 0; i < matrix.length; i++){
        if(matrix[i][0] == 0){
            firstCol = true;
            break;
        }
    }
    for(int j = 0; j < matrix[0].length; j++){
        if(matrix[0][j] == 0){
            firstRow = true;
            break;
        }
    }
    for(int i = 1; i < matrix.length; i++){
        for(int j = 1; j < matrix[0].length; j++){
            if(matrix[i][j] == 0){
                matrix[i][0] = 0;
                matrix[0][j] = 0;
            }
        }
    }
    for(int i = 1; i < matrix.length; i++){
        for(int j = 1; j < matrix[0].length; j++){
            if(matrix[i][0] == 0 || matrix[0][j] == 0){
                matrix[i][j] = 0;
            }
        }
    }
    if(firstRow){
        for(int j = 0; j < matrix[0].length; j++){
            matrix[0][j] = 0;
        }
    }
    if(firstCol){
        for(int i = 0; i < matrix.length; i++){
            matrix[i][0] = 0;
        }
    }
}
```

ZigZad Conversion，。

```
public String convert(String s, int nRows) {
    if(s == null || s.length() == 0 || nRows <= 0){
        return "";
    }
    if(nRows == 1){
        return s;
    }
    StringBuilder res = new StringBuilder();
    int len = nRows * 2 - 2;
    for(int i = 0; i < nRows; i++){
        for(int j = i; j < s.length(); j += len){
            res.append(s.charAt(j));
            if(i != 0 && i != nRows - 1 && j + len - 2 * i < s.length()){
                res.append(s.charAt(j + len - 2 * i));
            }
        }
    }
    return res.toString();
}
```

Merge Intervals，。

```
public ArrayList<Interval> merge(ArrayList<Interval> intervals) {
        ArrayList<Interval> res = new ArrayList<Interval>();
        if(intervals == null || intervals.size() == 0){
            return res;
        }
        Collections.sort(intervals, new Comparator<Interval>(){
            public int compare(Interval i1, Interval i2){
                return i1.start != i2.start ? i1.start - i2.start : i1.end - i2.end;
            }
        });
        Interval last = intervals.get(0);
        res.add(last);
        int idx = 1;
        while(idx < intervals.size()){
            if(last.end >= intervals.get(idx).start){
                last.end = Math.max(last.end, intervals.get(idx).end);
            } else {
                last = intervals.get(idx);
                res.add(last);
            }
            idx++;
        }
        return res;
    }
```

Insert Interval，。

```
    public ArrayList<Interval> insert(ArrayList<Interval> intervals, Interval newInterval) {
        ArrayList<Interval> res = new ArrayList<Interval>();
        if(intervals == null || intervals.size() == 0){
            res.add(newInterval);
            return res;
        }
        int idx = 0;
        while(idx < intervals.size() && intervals.get(idx).end < newInterval.start){
            res.add(intervals.get(idx));
            idx++;
        }
        res.add(newInterval);
        if(idx < intervals.size()){
            newInterval.start = Math.min(newInterval.start, intervals.get(idx).start);
        }
        while(idx < intervals.size() && intervals.get(idx).start <= newInterval.end){
            newInterval.end = Math.max(newInterval.end, intervals.get(idx).end);
            idx++;
        }
        while(idx < intervals.size()){
            res.add(intervals.get(idx));
            idx++;
        }
        return res;
    }
```

Next Permutation，。

```
    public void nextPermutation(int[] num) {
        if(num == null || num.length == 0){
            return;
        }
        int p = num.length - 2;
        while(p >= 0 && num[p] >= num[p + 1]){
            p--;
        }
        if(p == -1){
            reverse(num, 0, num.length - 1);
        } else {
            int q = num.length - 1;
            while(p < q && num[q] <= num[p]){
                q--;
            }
            int temp = num[p];
            num[p] = num[q];
            num[q] = temp;
            reverse(num, p + 1, num.length - 1);
        }
    }

    private void reverse(int[] num, int left, int right){
        while(left < right){
            int temp = num[left];
            num[left] = num[right];
            num[right] = temp;
            left++;
            right--;
        }
    }
```

Count and Say，。

```
public String countAndSay(int n) {
    if(n <= 0){
        return "";
    }
    StringBuilder res = new StringBuilder();
    res.append(1);
    for(int i = 2; i <= n; i++){
        StringBuilder curRes = new StringBuilder();
        int count = 1;
        for(int j = 0; j < res.length(); j++){
            if(j == res.length() - 1 || res.charAt(j) != res.charAt(j + 1)){
                curRes.append(count);
                curRes.append(res.charAt(j));
                count = 1;
            } else {
                count++;
            }
        }
        res = curRes;
    }
    return res.toString();
}
```

Excel Sheet Column Title，。

```
public String convertToTitle(int n) {
    StringBuilder res = new StringBuilder();
    while(n > 0){
        n--;
        res.insert(0, (char)('A' + n % 26));
        n /= 26;
    }
```

```
        return res.toString();
    }
```

Excel Sheet Column Number， 。

```
    public int titleToNumber(String s) {
        if(s == null || s.length() == 0){
            return -1;
        }
        int res = 0;
        for(int i = 0; i < s.length(); i++){
            res = res * 26 + s.charAt(i) - 'A' + 1;
        }
        return res;
    }
```

Gray Code， 。

```
    public ArrayList<Integer> grayCode(int n) {
        ArrayList<Integer> res = new ArrayList<Integer>();
        if(n < 0){
            return res;
        }
        if(n == 0){
            res.add(0);
            return res;
        }
        res.add(0);
        res.add(1);
        for(int i = 2; i <= n; i++){
            int size = res.size();
            for(int j = size - 1; j >= 0; j--){
                res.add(res.get(j) + (1 << i - 1));
            }
        }
        return res;
    }
```

Gas Station， 。

```
    public int canCompleteCircuit(int[] gas, int[] cost) {
        if(gas == null || gas.length == 0 || cost == null || cost.length == 0 || gas.length != cost.length){
            return -1;
        }
        int total = 0;
        int sum = 0;
        int pointer = -1;
        for(int i = 0; i < gas.length; i++){
            int diff = gas[i] - cost[i];
            total += diff;
            sum += diff;
            if(sum < 0){
                sum = 0;
                pointer = i;
            }
        }
        return total >= 0 ? pointer + 1 : -1;
    }
```

Candy，。

```
    public int candy(int[] ratings) {
        if(ratings == null || ratings.length == 0){
            return 0;
        }
        int[] res = new int[ratings.length];
        res[0] = 1;
        for(int i = 1; i < ratings.length; i++){
            if(ratings[i] > ratings[i - 1]){
                res[i] = res[i - 1] + 1;
            } else {
                res[i] = 1;
            }
        }
        int total = res[ratings.length - 1];
        for(int i = ratings.length - 2; i >= 0; i--){
            if(ratings[i] > ratings[i + 1]){
                res[i] = Math.max(res[i], res[i + 1] + 1);
            }
            total += res[i];
        }
        return total;
    }
```

Anagrams，。

```
    public ArrayList<String> anagrams(String[] strs) {
        ArrayList<String> res = new ArrayList<String>();
        if(strs == null || strs.length == 0){
            return res;
        }
        HashMap<String, ArrayList<String>> map = new HashMap<String,
ArrayList<String>>();
        for(int i = 0; i < strs.length; i++){
            char[] charArray = strs[i].toCharArray();
            Arrays.sort(charArray);
            String keyString = new String(charArray);
            ArrayList<String> item = null;
            if(map.containsKey(keyString)){
                item = map.get(keyString);
            } else {
                item = new ArrayList<String>();
            }
            item.add(strs[i]);
            map.put(keyString, item);
        }
        for(ArrayList<String> item : map.values()){
            if(item.size() > 1){
                res.addAll(item);
            }
        }
        return res;
    }
```

Surrounded Regions，。

```
    public void solve(char[][] board) {
        if(board == null || board.length == 0 || board[0].length == 0){
            return;
        }
        for(int i = 0; i < board.length; i++){
            fill(i, 0, board);
            fill(i, board[0].length - 1, board);
        }
        for(int j = 0; j < board[0].length; j++){
            fill(0, j, board);
            fill(board.length - 1, j, board);
        }
        for(int i = 0; i < board.length; i++){
            for(int j = 0; j < board[0].length; j++){
                if(board[i][j] == 'O'){
                    board[i][j] = 'X';
                } else if(board[i][j] == '#'){
                    board[i][j] = 'O';
                }
            }
        }
    }

    private void fill(int i, int j, char[][] board){
        if(board[i][j] != 'O'){
            return;
        }
        board[i][j] = '#';
        int code = i * board[0].length + j;
        LinkedList<Integer> queue = new LinkedList<Integer>();
        queue.offer(code);
        while(!queue.isEmpty()){
            int curCode = queue.poll();
            int row = curCode / board[0].length;
            int col = curCode % board[0].length;
            if(row > 0 && board[row - 1][col] == 'O'){
                board[row - 1][col] = '#';
                queue.offer((row - 1) * board[0].length + col);
            }
            if(row < board.length - 1 && board[row + 1][col] == 'O'){
                board[row + 1][col] = '#';
                queue.offer((row + 1) * board[0].length + col);
            }
            if(col > 0 && board[row][col - 1] == 'O'){
                board[row][col - 1] = '#';
                queue.offer(row * board[0].length + col - 1);
            }
            if(col < board[0].length - 1 && board[row][col + 1] == 'O'){
                board[row][col + 1] = '#';
                queue.offer(row * board[0].length + col + 1);
            }
        }
    }
```

Valid Palindrome， 。

```
    public boolean isPalindrome(String s) {
        if(s == null || s.length() == 0){
            return true;
        }
        int left = 0;
        int right = s.length() - 1;
        while(left < right){
            while(left < right && !isValid(s.charAt(left))){
                left++;
            }
            while(left < right && !isValid(s.charAt(right))){
                right--;
            }
            if(!isSame(s.charAt(left), s.charAt(right))){
                return false;
            }
            left++;
            right--;
        }
        return true;
    }

    private boolean isSame(char a, char b){
        if(a >= 'A' && a <= 'Z'){
            a = (char)(a - 'A' + 'a');
        }
        if(b >= 'A' && b <= 'Z'){
            b = (char)(b - 'A' + 'a');
        }
        return a == b;
    }

    private boolean isValid(char c){
        return (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9');
    }
```

Longest Common Prefix， 。

```
public String longestCommonPrefix(String[] strs) {
    if(strs == null || strs.length == 0){
        return "";
    }
    StringBuilder res = new StringBuilder();
    boolean isSame = true;
    for(int i = 0; i < strs[0].length(); i++){
        for(int j = 1; j < strs.length; j++){
            if(i < strs[j].length() && strs[j].charAt(i) == strs[0].charAt(i)){
                continue;
            } else {
                isSame = false;
                break;
            }
        }
        if(isSame){
            res.append(strs[0].charAt(i));
        } else {
            return res.toString();
        }
    }
    return res.toString();
}
```

Longest Substring Without Repeating Characters， 。

```
// Solution 1 - Window + HashSet
public int lengthOfLongestSubstring(String s) {
    if(s == null || s.length() == 0){
        return 0;
    }
    int maxLength = 0;
    int start = 0;
    int end = 0;
    HashSet<Character> set = new HashSet<Character>();
    while(end < s.length()){
        if(set.contains(s.charAt(end))){
            maxLength = Math.max(maxLength, end - start);
            while(s.charAt(start) != s.charAt(end)){
                set.remove(s.charAt(start));
                start++;
            }
            start++;
            end++;
        } else {
            set.add(s.charAt(end));
            end++;
        }
    }
    maxLength = Math.max(maxLength, end - start);
    return maxLength;
}

// Solution 2 - Last Index Dictionary
public int lengthOfLongestSubstring(String s) {
    if(s == null || s.length() == 0){
        return 0;
    }
    int maxLength = 0;
    int start = 0;
    int idx = 0;
    HashMap<Character, Integer> map = new HashMap<Character, Integer>();
    while(idx < s.length()){
        if(map.containsKey(s.charAt(idx))){
            int pre = map.get(s.charAt(idx));
            if(pre >= start){
                start = pre + 1;
            }
        }
        map.put(s.charAt(idx), idx);
        maxLength = Math.max(maxLength, idx - start + 1);
        idx++;
    }
    return maxLength;
}
```

Minimum Window Substring， 。

```
public String minWindow(String S, String T) {
    if(S == null || S.length() < T.length()){
        return "";
    }
    HashMap<Character, Integer> map = new HashMap<Character, Integer>();
    for(int i = 0; i < T.length(); i++){
        char tChar = T.charAt(i);
        if(map.containsKey(tChar)){
            map.put(tChar, map.get(tChar) + 1);
        } else {
            map.put(tChar, 1);
        }
    }
    int left = 0;
    int right = 0;
    int count = 0;
    int minStart = 0;
    int minLength = S.length() + 1;
    while(right < S.length()){
        char sChar = S.charAt(right);
        if(map.containsKey(sChar)){
            map.put(sChar, map.get(sChar) - 1);
            if(map.get(sChar) >= 0){
                count++;
            }
        }
        while(count == T.length()){
            if(right - left + 1 < minLength){
                minLength = right - left + 1;
                minStart = left;
            }
            char leftChar = S.charAt(left);
            if(map.containsKey(leftChar)){
                map.put(leftChar, map.get(leftChar) + 1);
                if(map.get(leftChar) > 0){
                    count--;
                }
            }
            left++;
        }
        right++;
    }
    if(minLength > S.length()){
        return "";
    }
    return S.substring(minStart, minStart + minLength);
}
```

Substring with Concatenation of All Words，。

```
public ArrayList<Integer> findSubstring(String S, String[] L) {
    ArrayList<Integer> res = new ArrayList<Integer>();
    if(S == null || S.length() == 0 || L == null || L.length == 0){
        return res;
    }
    HashMap<String, Integer> pattern = new HashMap<String, Integer>();
    for(int i = 0; i < L.length; i++){
        if(pattern.containsKey(L[i])){
            pattern.put(L[i], pattern.get(L[i]) + 1);
        } else {
            pattern.put(L[i], 1);
        }
    }
    for(int i = 0; i < L[0].length(); i++){
        HashMap<String, Integer> match = new HashMap<String, Integer>();
        int count = 0;
        int last = i;
        for(int j = i; j <= S.length() - L[0].length(); j += L[0].length()){
            String piece = S.substring(j, j + L[0].length());
            if(pattern.containsKey(piece)){
                if(match.containsKey(piece)){
                    match.put(piece, match.get(piece) + 1);
                } else {
                    match.put(piece, 1);
                }
                if(match.get(piece) <= pattern.get(piece)){
                    count++;
                } else {
                    while(match.get(piece) > pattern.get(piece)){
                        String temp = S.substring(last, last + L[0].length());
                        match.put(temp, match.get(temp) - 1);
                        if(match.get(temp) < pattern.get(temp)){
                            count--;
                        }
                        last += L[0].length();
                    }
                }
                if(count == L.length){
                    res.add(last);
                }
            } else {
                match.clear();
                count = 0;
                last = j + L[0].length();
            }
        }
    }
    return res;
}
```

Simplify Path，。

```
    public String simplifyPath(String path) {
        if(path == null || path.length() == 0){
            return "";
        }
        StringBuilder res = new StringBuilder();
        LinkedList<String> stack = new LinkedList<String>();
        int idx = 0;
        while(idx < path.length()){
            int start = idx;
            StringBuilder piece = new StringBuilder();
            while(idx < path.length() && path.charAt(idx) != '/'){
                piece.append(path.charAt(idx));
                idx++;
            }
            if(idx != start){
                String subPath = piece.toString();
                if(subPath.equals("..")){
                    if(!stack.isEmpty()){
                        stack.pop();
                    }
                } else if(!subPath.equals(".")){
                    stack.push(subPath);
                }
            }
            idx++;
        }
        if(!stack.isEmpty()){
            String[] strArr = stack.toArray(new String[stack.size()]);
            for(int i = strArr.length - 1; i >= 0; i--){
                res.append("/" + strArr[i]);
            }
        }
        if(stack.isEmpty()){
            return "/";
        }
        return res.toString();
    }
```

Text Justification，。

```
public ArrayList<String> fullJustify(String[] words, int L) {
    ArrayList<String> res = new ArrayList<String>();
    if(words == null || words.length == 0){
        return res;
    }
    int count = 0;
    int last = 0;
    for(int i = 0; i < words.length; i++){
        if(count + words[i].length() + (i - last) > L){
            int spaceNum = 0;
            int extraNum = 0;
            if(i - 1 - last > 0){
                    spaceNum = (L - count) / (i - 1 - last);
                    extraNum = (L - count) % (i - 1 - last);
                }
                StringBuilder curLine = new StringBuilder();
                for(int j = last; j < i; j++){
                    curLine.append(words[j]);
                    if(j < i - 1){
                        for(int k = 0; k < spaceNum; k++){
                            curLine.append(" ");
                        }
                        if(extraNum > 0){
                            curLine.append(" ");
                        }
                        extraNum--;
                    }
                }
                for(int j = curLine.length(); j < L; j++){
                    curLine.append(" ");
                }
                res.add(curLine.toString());
                count = 0;
                last = i;
            }
            count += words[i].length();
        }
        StringBuilder lastLine = new StringBuilder();
        for(int i = last; i < words.length; i++){
            lastLine.append(words[i]);
            if(lastLine.length() < L){
                lastLine.append(" ");
            }
        }
        for(int i = lastLine.length(); i < L; i++){
            lastLine.append(" ");
        }
        res.add(lastLine.toString());
        return res;
    }
```

Valid Number，。

```
    public boolean isNumber(String s) {
        if(s == null){
            return false;
        }
        s = s.trim();
        if(s.length() == 0){
            return false;
        }
        boolean dotFlag = false;
        boolean eFlag = false;
        for(int i = 0; i < s.length(); i++){
            switch(s.charAt(i)){
                case '.':
                    if(dotFlag || eFlag || (i == 0 || !(s.charAt(i - 1) >= '0' && s.charAt(i -
1) <= '9')) && (i == s.length() - 1 || !(s.charAt(i + 1) >= '0' && s.charAt(i + 1) <= '9'))){
                        return false;
                    }
                    dotFlag = true;
                    break;
                case '+':
                case '-':
                    if((i > 0 && !(s.charAt(i - 1) == 'e' || s.charAt(i - 1) == 'E')) || (i ==
s.length() - 1 || !(s.charAt(i + 1) >= '0' && s.charAt(i + 1) <= '9' || s.charAt(i + 1) == '.'))){
                        return false;
                    }
                    break;
                case 'e':
                case 'E':
                    if(eFlag || i == 0 || i == s.length() - 1){
                        return false;
                    }
                    eFlag = true;
                    break;
                case '9':
                case '8':
                case '7':
                case '6':
                case '5':
                case '4':
                case '3':
```

```
            case '2':
            case '1':
            case '0':
                break;
            default:
                return false;
        }
    }
    return true;
}
```

First Missing Positive， 。

```
public int firstMissingPositive(int[] A) {
    if(A == null || A.length == 0){
        return 1;
    }
    for(int i = 0; i < A.length; i++){
        if(A[i] > 0 && A[i] <= A.length && A[i] != i + 1 && A[A[i] - 1] != A[i]){
            int temp = A[A[i] - 1];
            A[A[i] - 1] = A[i];
            A[i] = temp;
            i--;
        }
    }
    for(int i = 0; i < A.length; i++){
        if(A[i] != i + 1){
            return i + 1;
        }
    }
    return A.length + 1;
}
```

Fraction to Recurring Decimal， 。

```
public String fractionToDecimal(int numerator, int denominator) {
    if(numerator == 0){
        return "0";
    }
    StringBuilder res = new StringBuilder();
    res.append(((numerator ^ denominator) >>> 31) == 1 ? "-" : "");
    long num = Math.abs((long)numerator);
    long den = Math.abs((long)denominator);
    res.append(num / den);
    num %= den;
    if(num == 0){
        return res.toString();
    }
    res.append(".");
    HashMap<Long, Integer> map = new HashMap<Long, Integer>();
    map.put(num, res.length());
    while(num != 0){
        num *= 10;
        res.append(num / den);
        num %= den;
        if(map.containsKey(num)){
            int index = map.get(num);
            res.insert(index, "(");
            res.append(")");
            break;
        } else {
            map.put(num, res.length());
        }
    }
    return res.toString();
}
```