LEETCODE 1 TO 100

SOLUTION IN JAVA

1. [Two Sum](https://leetcode.com/problems/two-sum)

class Solution {

public int[] twoSum(int[] nums, int target) {

Map<Integer, Integer> numToIndex = new HashMap<>();

for (int i = 0; i < nums.length; ++i) {

if (numToIndex.containsKey(target - nums[i]))

return new int[] {numToIndex.get(target - nums[i]), i};

numToIndex.put(nums[i], i);

}

throw new IllegalArgumentException();

}

}

# [Add Two Numbers](https://leetcode.com/problems/add-two-numbers)

# class Solution {

# public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

# ListNode dummy = new ListNode(0);

# ListNode curr = dummy;

# int carry = 0;

# while (l1 != null || l2 != null || carry > 0) {

# if (l1 != null) {

# carry += l1.val;

# l1 = l1.next;

# }

# if (l2 != null) {

# carry += l2.val;

# l2 = l2.next;

# }

# curr.next = new ListNode(carry % 10);

# carry /= 10;

# curr = curr.next;

# }

# return dummy.next;

# }

# }

# [3. Longest Substring Without Repeating Characters](https://leetcode.com/problems/longest-substring-without-repeating-characters)

# class Solution {

# public int lengthOfLongestSubstring(String s) {

# int ans = 0;

# int[] count = new int[128];

# for (int l = 0, r = 0; r < s.length(); ++r) {

# ++count[s.charAt(r)];

# while (count[s.charAt(r)] > 1)

# --count[s.charAt(l++)];

# ans = Math.max(ans, r - l + 1);

# }

# return ans;

# }

# }

# [4. Median of Two Sorted Arrays](https://leetcode.com/problems/median-of-two-sorted-arrays)

# class Solution {

# public double findMedianSortedArrays(int[] nums1, int[] nums2) {

# final int n1 = nums1.length;

# final int n2 = nums2.length;

# if (n1 > n2)

# return findMedianSortedArrays(nums2, nums1);

# int l = 0;

# int r = n1;

# while (l <= r) {

# final int partition1 = (l + r) / 2;

# final int partition2 = (n1 + n2 + 1) / 2 - partition1;

# final int maxLeft1 = partition1 == 0 ? Integer.MIN\_VALUE : nums1[partition1 - 1];

# final int maxLeft2 = partition2 == 0 ? Integer.MIN\_VALUE : nums2[partition2 - 1];

# final int minRight1 = partition1 == n1 ? Integer.MAX\_VALUE : nums1[partition1];

# final int minRight2 = partition2 == n2 ? Integer.MAX\_VALUE : nums2[partition2];

# if (maxLeft1 <= minRight2 && maxLeft2 <= minRight1)

# return (n1 + n2) % 2 == 0

# ? (Math.max(maxLeft1, maxLeft2) + Math.min(minRight1, minRight2)) \* 0.5

# : Math.max(maxLeft1, maxLeft2);

# else if (maxLeft1 > minRight2)

# r = partition1 - 1;

# else

# l = partition1 + 1;

# }

# throw new IllegalArgumentException();

# }

# }

# [5.Longest Palindromic Substring](https://leetcode.com/problems/longest-palindromic-substring)

# class Solution {

# public String longestPalindrome(String s) {

# if (s.isEmpty())

# return "";

# // (start, end) indices of the longest palindrome in s

# int[] indices = {0, 0};

# for (int i = 0; i < s.length(); ++i) {

# int[] indices1 = extend(s, i, i);

# if (indices1[1] - indices1[0] > indices[1] - indices[0])

# indices = indices1;

# if (i + 1 < s.length() && s.charAt(i) == s.charAt(i + 1)) {

# int[] indices2 = extend(s, i, i + 1);

# if (indices2[1] - indices2[0] > indices[1] - indices[0])

# indices = indices2;

# }

# }

# return s.substring(indices[0], indices[1] + 1);

# }

# private int[] extend(final String s, int i, int j) {

# for (; i >= 0 && j < s.length(); --i, ++j)

# if (s.charAt(i) != s.charAt(j))

# break;

# return new int[] {i + 1, j - 1};

# }

# }

# [6. Zigzag Conversion](https://leetcode.com/problems/zigzag-conversion)

# class Solution {

# public String convert(String s, int numRows) {

# StringBuilder sb = new StringBuilder();

# List<Character>[] rows = new List[numRows];

# int k = 0;

# int direction = numRows == 1 ? 0 : -1;

# for (int i = 0; i < numRows; ++i)

# rows[i] = new ArrayList<>();

# for (final char c : s.toCharArray()) {

# rows[k].add(c);

# if (k == 0 || k == numRows - 1)

# direction \*= -1;

# k += direction;

# }

# for (List<Character> row : rows)

# for (final char c : row)

# sb.append(c);

# return sb.toString();

# }

# }

# [7. Reverse Integer](https://leetcode.com/problems/reverse-integer)

# class Solution {

# public int reverse(int x) {

# long ans = 0;

# while (x != 0) {

# ans = ans \* 10 + x % 10;

# x /= 10;

# }

# [8. String to Integer (atoi)](https://leetcode.com/problems/string-to-integer-atoi)

# class Solution {

# public int myAtoi(String s) {

# s = s.strip();

# if (s.isEmpty())

# return 0;

# final int sign = s.charAt(0) == '-' ? -1 : 1;

# if (s.charAt(0) == '+' || s.charAt(0) == '-')

# s = s.substring(1);

# long num = 0;

# for (final char c : s.toCharArray()) {

# if (!Character.isDigit(c))

# break;

# num = num \* 10 + (c - '0');

# if (sign \* num <= Integer.MIN\_VALUE)

# return Integer.MIN\_VALUE;

# if (sign \* num >= Integer.MAX\_VALUE)

# return Integer.MAX\_VALUE;

# }

# return sign \* (int) num;

# }

# }

# [9. Palindrome Number](https://leetcode.com/problems/palindrome-number)

# [10. Regular Expression Matching](https://leetcode.com/problems/regular-expression-matching)

# class Solution {

# public boolean isMatch(String s, String p) {

# final int m = s.length();

# final int n = p.length();

# // dp[i][j] := true if s[0..i) matches p[0..j)

# boolean[][] dp = new boolean[m + 1][n + 1];

# dp[0][0] = true;

# for (int j = 0; j < p.length(); ++j)

# if (p.charAt(j) == '\*' && dp[0][j - 1])

# dp[0][j + 1] = true;

# for (int i = 0; i < m; ++i)

# for (int j = 0; j < n; ++j)

# if (p.charAt(j) == '\*') {

# // The minimum index of '\*' is 1.

# final boolean noRepeat = dp[i + 1][j - 1];

# final boolean doRepeat = isMatch(s, i, p, j - 1) && dp[i][j + 1];

# dp[i + 1][j + 1] = noRepeat || doRepeat;

# } else if (isMatch(s, i, p, j)) {

# dp[i + 1][j + 1] = dp[i][j];

# }

# return dp[m][n];

# }

# private boolean isMatch(final String s, int i, final String p, int j) {

# return j >= 0 && p.charAt(j) == '.' || s.charAt(i) == p.charAt(j);

# }

# }

# [11. Container With Most Water](https://leetcode.com/problems/container-with-most-water)

# class Solution {

# public int maxArea(int[] height) {

# int ans = 0;

# int l = 0;

# int r = height.length - 1;

# while (l < r) {

# final int minHeight = Math.min(height[l], height[r]);

# ans = Math.max(ans, minHeight \* (r - l));

# if (height[l] < height[r])

# ++l;

# else

# --r;

# }

# return ans;

# }

# }

# [12. Integer to Roman](https://leetcode.com/problems/integer-to-roman)

# class Solution {

# public String intToRoman(int num) {

# final int[] values = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};

# final String[] symbols = {"M", "CM", "D", "CD", "C", "XC", "L",

# "XL", "X", "IX", "V", "IV", "I"};

# StringBuilder sb = new StringBuilder();

# for (int i = 0; i < values.length; ++i) {

# if (num == 0)

# break;

# while (num >= values[i]) {

# num -= values[i];

# sb.append(symbols[i]);

# }

# }

# return sb.toString();

# }

# }

# [13. Roman to Integer](https://leetcode.com/problems/roman-to-integer)

# class Solution {

# public int romanToInt(String s) {

# int ans = 0;

# int[] roman = new int[128];

# roman['I'] = 1;

# roman['V'] = 5;

# roman['X'] = 10;

# roman['L'] = 50;

# roman['C'] = 100;

# roman['D'] = 500;

# roman['M'] = 1000;

# for (int i = 0; i + 1 < s.length(); ++i)

# if (roman[s.charAt(i)] < roman[s.charAt(i + 1)])

# ans -= roman[s.charAt(i)];

# else

# ans += roman[s.charAt(i)];

# return ans + roman[s.charAt(s.length() - 1)];

# }

# }

# [14. Longest Common Prefix](https://leetcode.com/problems/longest-common-prefix)

# class Solution {

# public String longestCommonPrefix(String[] strs) {

# if (strs.length == 0)

# return "";

# 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))

# return strs[0].substring(0, i);

# return strs[0];

# }

# }

# [15. 3Sum](https://leetcode.com/problems/3sum)

# class Solution {

# public List<List<Integer>> threeSum(int[] nums) {

# if (nums.length < 3)

# return new ArrayList<>();

# List<List<Integer>> ans = new ArrayList<>();

# Arrays.sort(nums);

# for (int i = 0; i + 2 < nums.length; ++i) {

# if (i > 0 && nums[i] == nums[i - 1])

# continue;

# int l = i + 1;

# int r = nums.length - 1;

# while (l < r) {

# final int sum = nums[i] + nums[l] + nums[r];

# if (sum == 0) {

# ans.add(Arrays.asList(nums[i], nums[l++], nums[r--]));

# while (l < r && nums[l] == nums[l - 1])

# ++l;

# while (l < r && nums[r] == nums[r + 1])

# --r;

# } else if (sum < 0) {

# ++l;

# } else {

# --r;

# }

# }

# }

# return ans;

# }

# }

# [16. 3Sum Closest](https://leetcode.com/problems/3sum-closest)

# class Solution {

# public int threeSumClosest(int[] nums, int target) {

# int ans = nums[0] + nums[1] + nums[2];

# Arrays.sort(nums);

# for (int i = 0; i + 2 < nums.length; ++i) {

# if (i > 0 && nums[i] == nums[i - 1])

# continue;

# int l = i + 1;

# int r = nums.length - 1;

# while (l < r) {

# final int sum = nums[i] + nums[l] + nums[r];

# if (sum == target)

# return sum;

# if (Math.abs(sum - target) < Math.abs(ans - target))

# ans = sum;

# if (sum < target)

# ++l;

# else

# --r;

# }

# }

# return ans;

# }

# }

# [17. Letter Combinations of a Phone Number](https://leetcode.com/problems/letter-combinations-of-a-phone-number)

# class Solution {

# public List<String> letterCombinations(String digits) {

# if (digits.isEmpty())

# return new ArrayList<>();

# List<String> ans = new ArrayList<>();

# dfs(digits, 0, new StringBuilder(), ans);

# return ans;

# }

# private static final String[] digitToLetters = {"", "", "abc", "def", "ghi",

# "jkl", "mno", "pqrs", "tuv", "wxyz"};

# private void dfs(String digits, int i, StringBuilder sb, List<String> ans) {

# if (i == digits.length()) {

# ans.add(sb.toString());

# return;

# }

# for (final char c : digitToLetters[digits.charAt(i) - '0'].toCharArray()) {

# sb.append(c);

# dfs(digits, i + 1, sb, ans);

# sb.deleteCharAt(sb.length() - 1);

# }

# }

# }

# [18. 4Sum](https://leetcode.com/problems/4sum)

class Solution {

public List<List<Integer>> fourSum(int[] nums, int target) {

List<List<Integer>> ans = new ArrayList<>();

Arrays.sort(nums);

nSum(nums, 4, target, 0, nums.length - 1, new ArrayList<>(), ans);

return ans;

}

// Finds n numbers that add up to the target in [l, r].

private void nSum(int[] nums, long n, long target, int l, int r, List<Integer> path,

List<List<Integer>> ans) {

if (r - l + 1 < n || target < nums[l] \* n || target > nums[r] \* n)

return;

if (n == 2) {

// Similar to the sub procedure in 15. 3Sum

while (l < r) {

final int sum = nums[l] + nums[r];

if (sum == target) {

path.add(nums[l]);

path.add(nums[r]);

ans.add(new ArrayList<>(path));

path.remove(path.size() - 1);

path.remove(path.size() - 1);

++l;

--r;

while (l < r && nums[l] == nums[l - 1])

++l;

while (l < r && nums[r] == nums[r + 1])

--r;

} else if (sum < target) {

++l;

} else {

--r;

}

}

return;

}

for (int i = l; i <= r; ++i) {

if (i > l && nums[i] == nums[i - 1])

continue;

path.add(nums[i]);

nSum(nums, n - 1, target - nums[i], i + 1, r, path, ans);

path.remove(path.size() - 1);

}

}

}

# [19. Remove Nth Node From End of List](https://leetcode.com/problems/remove-nth-node-from-end-of-list)

} class Solution {

public ListNode removeNthFromEnd(ListNode head, int n) {

ListNode slow = head;

ListNode fast = head;

while (n-- > 0)

fast = fast.next;

if (fast == null)

return head.next;

while (fast.next != null) {

slow = slow.next;

fast = fast.next;

}

slow.next = slow.next.next;

return head;

}

# [20. Valid Parentheses](https://leetcode.com/problems/valid-parentheses)

class Solution {

public boolean isValid(String s) {

Deque<Character> stack = new ArrayDeque<>();

for (final char c : s.toCharArray())

if (c == '(')

stack.push(')');

else if (c == '{')

stack.push('}');

else if (c == '[')

stack.push(']');

else if (stack.isEmpty() || stack.pop() != c)

return false;

return stack.isEmpty();

}

}

# [21. Merge Two Sorted Lists](https://leetcode.com/problems/merge-two-sorted-lists)

class Solution {

public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

if (list1 == null || list2 == null)

return list1 == null ? list2 : list1;

if (list1.val > list2.val) {

ListNode temp = list1;

list1 = list2;

list2 = temp;

}

list1.next = mergeTwoLists(list1.next, list2);

return list1;

}

}

# [22. Generate Parentheses](https://leetcode.com/problems/generate-parentheses)

class Solution {

public List<String> generateParenthesis(int n) {

List<String> ans = new ArrayList<>();

dfs(n, n, new StringBuilder(), ans);

return ans;

}

private void dfs(int l, int r, final StringBuilder sb, List<String> ans) {

if (l == 0 && r == 0) {

ans.add(sb.toString());

return;

}

if (l > 0) {

sb.append("(");

dfs(l - 1, r, sb, ans);

sb.deleteCharAt(sb.length() - 1);

}

if (l < r) {

sb.append(")");

dfs(l, r - 1, sb, ans);

sb.deleteCharAt(sb.length() - 1);

}

}

}

# [23. Merge k Sorted Lists](https://leetcode.com/problems/merge-k-sorted-lists)

class Solution {

public ListNode mergeKLists(ListNode[] lists) {

ListNode dummy = new ListNode(0);

ListNode curr = dummy;

Queue<ListNode> minHeap = new PriorityQueue<>((a, b) -> a.val - b.val);

for (final ListNode list : lists)

if (list != null)

minHeap.offer(list);

while (!minHeap.isEmpty()) {

ListNode minNode = minHeap.poll();

if (minNode.next != null)

minHeap.offer(minNode.next);

curr.next = minNode;

curr = curr.next;

}

return dummy.next;

}

}

# [24. Swap Nodes in Pairs](https://leetcode.com/problems/swap-nodes-in-pairs)

**class Solution {**

**public ListNode swapPairs(ListNode head) {**

**final int length = getLength(head);**

**ListNode dummy = new ListNode(0, head);**

**ListNode prev = dummy;**

**ListNode curr = head;**

**for (int i = 0; i < length / 2; ++i) {**

**ListNode next = curr.next;**

**curr.next = next.next;**

**next.next = curr;**

**prev.next = next;**

**prev = curr;**

**curr = curr.next;**

**}**

**return dummy.next;**

**}**

**private int getLength(ListNode head) {**

**int length = 0;**

**for (ListNode curr = head; curr != null; curr = curr.next)**

**++length;**

**return length;**

**}**

**}**

# [25. Reverse Nodes in k-Group](https://leetcode.com/problems/reverse-nodes-in-k-group)

**class Solution {**

**public ListNode reverseKGroup(ListNode head, int k) {**

**if (head == null)**

**return null;**

**ListNode tail = head;**

**for (int i = 0; i < k; ++i) {**

**// There are less than k nodes in the list, do nothing.**

**if (tail == null)**

**return head;**

**tail = tail.next;**

**}**

**ListNode newHead = reverse(head, tail);**

**head.next = reverseKGroup(tail, k);**

**return newHead;**

**}**

**// Reverses [head, tail).**

**private ListNode reverse(ListNode head, ListNode tail) {**

**ListNode prev = null;**

**ListNode curr = head;**

**while (curr != tail) {**

**ListNode next = curr.next;**

**curr.next = prev;**

**prev = curr;**

**curr = next;**

**}**

**return prev;**

**}**

**}**

# [26. Remove Duplicates from Sorted Array](https://leetcode.com/problems/remove-duplicates-from-sorted-array)

**class Solution {**

**public int removeDuplicates(int[] nums) {**

**int i = 0;**

**for (final int num : nums)**

**if (i < 1 || num > nums[i - 1])**

**nums[i++] = num;**

**return i;**

**}**

**}**

# [27. Remove Element](https://leetcode.com/problems/remove-element)

**class Solution {**

**public int removeElement(int[] nums, int val) {**

**int i = 0;**

**for (final int num : nums)**

**if (num != val)**

**nums[i++] = num;**

**return i;**

**}**

**}**

# [28. Find the Index of the First Occurrence in a String](https://leetcode.com/problems/find-the-index-of-the-first-occurrence-in-a-string)

**class Solution {**

**public int strStr(String haystack, String needle) {**

**final int m = haystack.length();**

**final int n = needle.length();**

**for (int i = 0; i < m - n + 1; ++i)**

**if (haystack.substring(i, i + n).equals(needle))**

**return i;**

**return -1;**

**}**

**}**

# [29. Divide Two Integers](https://leetcode.com/problems/divide-two-integers)

class Solution {

public int divide(long dividend, long divisor) {

// -2^{31} / -1 = 2^31 will overflow, so return 2^31 - 1.

if (dividend == Integer.MIN\_VALUE && divisor == -1)

return Integer.MAX\_VALUE;

final int sign = dividend > 0 ^ divisor > 0 ? -1 : 1;

long ans = 0;

long dvd = Math.abs(dividend);

long dvs = Math.abs(divisor);

while (dvd >= dvs) {

long k = 1;

while (k \* 2 \* dvs <= dvd)

k \*= 2;

dvd -= k \* dvs;

ans += k;

}

return sign \* (int) ans;

}

}

# [30. Substring with Concatenation of All Words](https://leetcode.com/problems/substring-with-concatenation-of-all-words)

**class Solution {**

**public List<Integer> findSubstring(String s, String[] words) {**

**if (s.isEmpty() || words.length == 0)**

**return new ArrayList<>();**

**final int k = words.length;**

**final int n = words[0].length();**

**List<Integer> ans = new ArrayList<>();**

**Map<String, Integer> count = new HashMap<>();**

**for (final String word : words)**

**count.merge(word, 1, Integer::sum);**

**for (int i = 0; i <= s.length() - k \* n; ++i) {**

**Map<String, Integer> seen = new HashMap<>();**

**int j = 0;**

**for (; j < k; ++j) {**

**final String word = s.substring(i + j \* n, i + j \* n + n);**

**seen.merge(word, 1, Integer::sum);**

**if (seen.get(word) > count.getOrDefault(word, 0))**

**break;**

**}**

**if (j == k)**

**ans.add(i);**

**}**

**return ans;**

**}**

**}**

# [31. Next Permutation](https://leetcode.com/problems/next-permutation)

**class Solution {**

**public void nextPermutation(int[] nums) {**

**final int n = nums.length;**

**int i;**

**for (i = n - 2; i >= 0; --i)**

**if (nums[i] < nums[i + 1])**

**break;**

**if (i >= 0)**

**for (int j = n - 1; j > i; --j)**

**if (nums[j] > nums[i]) {**

**swap(nums, i, j);**

**break;**

**}**

**reverse(nums, i + 1, n - 1);**

**}**

**private void reverse(int[] nums, int l, int r) {**

**while (l < r)**

**swap(nums, l++, r--);**

**}**

**private void swap(int[] nums, int i, int j) {**

**final int temp = nums[i];**

**nums[i] = nums[j];**

**nums[j] = temp;**

**}**

**}**

# [32. Longest Valid Parentheses](https://leetcode.com/problems/longest-valid-parentheses)

**class Solution {**

**public int longestValidParentheses(String s) {**

**final String s2 = ")" + s;**

**int dp[] = new int[s2.length()];**

**for (int i = 1; i < s2.length(); ++i)**

**if (s2.charAt(i) == ')' && s2.charAt(i - dp[i - 1] - 1) == '(')**

**dp[i] = dp[i - 1] + dp[i - dp[i - 1] - 2] + 2;**

**return Arrays.stream(dp).max().getAsInt();**

**}**

**}**

# [33. Search in Rotated Sorted Array](https://leetcode.com/problems/search-in-rotated-sorted-array)

**class Solution {**

**public int search(int[] nums, int target) {**

**int l = 0;**

**int r = nums.length - 1;**

**while (l <= r) {**

**final int m = (l + r) / 2;**

**if (nums[m] == target)**

**return m;**

**if (nums[l] <= nums[m]) { // nums[l..m] are sorted.**

**if (nums[l] <= target && target < nums[m])**

**r = m - 1;**

**else**

**l = m + 1;**

**} else { // nums[m..n - 1] are sorted.**

**if (nums[m] < target && target <= nums[r])**

**l = m + 1;**

**else**

**r = m - 1;**

**}**

**}**

**return -1;**

**}**

**}**

# [34. Find First and Last Position of Element in Sorted Array](https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array)

**class Solution {**

**public int[] searchRange(int[] nums, int target) {**

**final int l = firstGreaterEqual(nums, target);**

**if (l == nums.length || nums[l] != target)**

**return new int[] {-1, -1};**

**final int r = firstGreaterEqual(nums, target + 1) - 1;**

**return new int[] {l, r};**

**}**

**private int firstGreaterEqual(int[] A, int target) {**

**int l = 0;**

**int r = A.length;**

**while (l < r) {**

**final int m = (l + r) / 2;**

**if (A[m] >= target)**

**r = m;**

**else**

**l = m + 1;**

**}**

**return l;**

**}**

**}**

# [35. Search Insert Position](https://leetcode.com/problems/search-insert-position)

**class Solution {**

**public int searchInsert(int[] nums, int target) {**

**int l = 0;**

**int r = nums.length;**

**while (l < r) {**

**final int m = (l + r) / 2;**

**if (nums[m] == target)**

**return m;**

**if (nums[m] < target)**

**l = m + 1;**

**else**

**r = m;**

**}**

**return l;**

**}**

**}**

# [36. Valid Sudoku](https://leetcode.com/problems/valid-sudoku)

**class Solution {**

**public boolean isValidSudoku(char[][] board) {**

**Set<String> seen = new HashSet<>();**

**for (int i = 0; i < 9; ++i)**

**for (int j = 0; j < 9; ++j) {**

**if (board[i][j] == '.')**

**continue;**

**final char c = board[i][j];**

**if (!seen.add(c + "@row" + i) || //**

**!seen.add(c + "@col" + j) || //**

**!seen.add(c + "@box" + i / 3 + j / 3))**

**return false;**

**}**

**return true;**

**}**

**}**

# [37. Sudoku Solver](https://leetcode.com/problems/sudoku-solver)

**class Solution {**

**public void solveSudoku(char[][] board) {**

**dfs(board, 0);**

**}**

**private boolean dfs(char[][] board, int s) {**

**if (s == 81)**

**return true;**

**final int i = s / 9;**

**final int j = s % 9;**

**if (board[i][j] != '.')**

**return dfs(board, s + 1);**

**for (char c = '1'; c <= '9'; ++c)**

**if (isValid(board, i, j, c)) {**

**board[i][j] = c;**

**if (dfs(board, s + 1))**

**return true;**

**board[i][j] = '.';**

**}**

**return false;**

**}**

**private boolean isValid(char[][] board, int row, int col, char c) {**

**for (int i = 0; i < 9; ++i)**

**if (board[i][col] == c || board[row][i] == c ||**

**board[3 \* (row / 3) + i / 3][3 \* (col / 3) + i % 3] == c)**

**return false;**

**return true;**

**}**

**}**

# [38. Count and Say](https://leetcode.com/problems/count-and-say)

**class Solution {**

**public String countAndSay(int n) {**

**StringBuilder sb = new StringBuilder("1")**

**while (--n > 0) {**

**StringBuilder next = new StringBuilder();**

**for (int i = 0; i < sb.length(); ++i) {**

**int count = 1;**

**while (i + 1 < sb.length() && sb.charAt(i) == sb.charAt(i + 1)) {**

**++count;**

**++i;**

**}**

**next.append(count).append(sb.charAt(i));**

**}**

**sb = next;**

**}**

**return sb.toString();**

**}**

**}**

# [39. Combination Sum](https://leetcode.com/problems/combination-sum)

**class Solution {**

**public List<List<Integer>> combinationSum(int[] candidates, int target) {**

**List<List<Integer>> ans = new ArrayList<>();**

**Arrays.sort(candidates);**

**dfs(0, candidates, target, new ArrayList<>(), ans);**

**return ans;**

**}**

**private void dfs(int s, int[] candidates, int target, List<Integer> path,**

**List<List<Integer>> ans) {**

**if (target < 0)**

**return;**

**if (target == 0) {**

**ans.add(new ArrayList<>(path));**

**return;**

**}**

**for (int i = s; i < candidates.length; ++i) {**

**path.add(candidates[i]);**

**dfs(i, candidates, target - candidates[i], path, ans);**

**path.remove(path.size() - 1);**

**}**

**}**

**}**

# [40. Combination Sum II](https://leetcode.com/problems/combination-sum-ii)

**class Solution {**

**public List<List<Integer>> combinationSum2(int[] candidates, int target) {**

**List<List<Integer>> ans = new ArrayList<>();**

**Arrays.sort(candidates);**

**dfs(0, candidates, target, new ArrayList<>(), ans);**

**return ans;**

**}**

**private void dfs(int s, int[] candidates, int target, List<Integer> path,**

**List<List<Integer>> ans) {**

**if (target < 0)**

**return;**

**if (target == 0) {**

**ans.add(new ArrayList<>(path));**

**return;**

**}**

**for (int i = s; i < candidates.length; ++i) {**

**if (i > s && candidates[i] == candidates[i - 1])**

**continue;**

**path.add(candidates[i]);**

**dfs(i + 1, candidates, target - candidates[i], path, ans);**

**path.remove(path.size() - 1);**

**}**

**}**

**}**

# [41. First Missing Positive](https://leetcode.com/problems/first-missing-positive)

**class Solution {**

**public int firstMissingPositive(int[] nums) {**

**final int n = nums.length;**

**for (int i = 0; i < n; ++i)**

**while (nums[i] > 0 && nums[i] <= n && nums[i] != nums[nums[i] - 1])**

**swap(nums, i, nums[i] - 1);**

**for (int i = 0; i < n; ++i)**

**if (nums[i] != i + 1)**

**return i + 1;**

**return n + 1;**

**}**

**private void swap(int[] nums, int i, int j) {**

**final int temp = nums[i];**

**nums[i] = nums[j];**

**nums[j] = temp;**

**}**

**}**

# [42. Trapping Rain Water](https://leetcode.com/problems/trapping-rain-water)

**class Solution {**

**public int trap(int[] height) {**

**final int n = height.length;**

**int ans = 0;**

**int[] l = new int[n]; // l[i] := max(height[0..i])**

**int[] r = new int[n]; // r[i] := max(height[i..n))**

**for (int i = 0; i < n; ++i)**

**l[i] = i == 0 ? height[i] : Math.max(height[i], l[i - 1]);**

**for (int i = n - 1; i >= 0; --i)**

**r[i] = i == n - 1 ? height[i] : Math.max(height[i], r[i + 1]);**

**for (int i = 0; i < n; ++i)**

**ans += Math.min(l[i], r[i]) - height[i];**

**return ans;**

**}**

**}**

# [43. Multiply Strings](https://leetcode.com/problems/multiply-strings)[¶](https://walkccc.me/LeetCode/problems/43/#43-multiply-strings)

**class Solution {**

**public String multiply(String num1, String num2) {**

**final int m = num1.length();**

**final int n = num2.length();**

**StringBuilder sb = new StringBuilder();**

**int[] pos = new int[m + n];**

**for (int i = m - 1; i >= 0; --i)**

**for (int j = n - 1; j >= 0; --j) {**

**final int multiply = (num1.charAt(i) - '0') \* (num2.charAt(j) - '0');**

**final int sum = multiply + pos[i + j + 1];**

**pos[i + j] += sum / 10;**

**pos[i + j + 1] = sum % 10;**

**}**

**for (final int p : pos)**

**if (p > 0 || sb.length() > 0)**

**sb.append(p);**

**return sb.length() == 0 ? "0" : sb.toString();**

**}**

**}**

# [44. Wildcard Matching](https://leetcode.com/problems/wildcard-matching)

**class Solution {**

**public boolean isMatch(String s, String p) {**

**final int m = s.length();**

**final int n = p.length();**

**// dp[i][j] := true if s[0..i) matches p[0..j)**

**boolean[][] dp = new boolean[m + 1][n + 1];**

**dp[0][0] = true;**

**for (int j = 0; j < p.length(); ++j)**

**if (p.charAt(j) == '\*')**

**dp[0][j + 1] = dp[0][j];**

**for (int i = 0; i < m; ++i)**

**for (int j = 0; j < n; ++j)**

**if (p.charAt(j) == '\*') {**

**final boolean matchEmpty = dp[i + 1][j];**

**final boolean matchSome = dp[i][j + 1];**

**dp[i + 1][j + 1] = matchEmpty || matchSome;**

**} else if (isMatch(s, i, p, j)) {**

**dp[i + 1][j + 1] = dp[i][j];**

**}**

**return dp[m][n];**

**}**

**private boolean isMatch(final String s, int i, final String p, int j) {**

**return j >= 0 && p.charAt(j) == '?' || s.charAt(i) == p.charAt(j);**

**}**

**}**

# [45. Jump Game II](https://leetcode.com/problems/jump-game-ii)

**class Solution {**

**public int jump(int[] nums) {**

**int ans = 0;**

**int end = 0;**

**int farthest = 0;**

**for (int i = 0; i < nums.length - 1; ++i) {**

**farthest = Math.max(farthest, i + nums[i]);**

**if (farthest >= nums.length - 1) {**

**++ans;**

**break;**

**}**

**if (i == end) { // Visited all the items on the current level.**

**++ans; // Increment the level.**

**end = farthest; // Make the queue size for the next level.**

**}**

**}**

**return ans;**

**}**

**}**

# [46. Permutations](https://leetcode.com/problems/permutations)

**class Solution {**

**public List<List<Integer>> permute(int[] nums) {**

**List<List<Integer>> ans = new ArrayList<>();**

**dfs(nums, new boolean[nums.length], new ArrayList<>(), ans);**

**return ans;**

**}**

**private void dfs(int[] nums, boolean[] used, List<Integer> path, List<List<Integer>> ans) {**

**if (path.size() == nums.length) {**

**ans.add(new ArrayList<>(path));**

**return;**

**}**

**for (int i = 0; i < nums.length; ++i) {**

**if (used[i])**

**continue;**

**used[i] = true;**

**path.add(nums[i]);**

**dfs(nums, used, path, ans);**

**path.remove(path.size() - 1);**

**used[i] = false;**

**}**

**}**

**}**

# [47. Permutations II](https://leetcode.com/problems/permutations-ii)

**class Solution {**

**public List<List<Integer>> permuteUnique(int[] nums) {**

**List<List<Integer>> ans = new ArrayList<>();**

**Arrays.sort(nums);**

**dfs(nums, new boolean[nums.length], new ArrayList<>(), ans);**

**return ans;**

**}**

**private void dfs(int[] nums, boolean[] used, List<Integer> path, List<List<Integer>> ans) {**

**if (path.size() == nums.length) {**

**ans.add(new ArrayList<>(path));**

**return;**

**}**

**for (int i = 0; i < nums.length; ++i) {**

**if (used[i])**

**continue;**

**if (i > 0 && nums[i] == nums[i - 1] && !used[i - 1])**

**continue;**

**used[i] = true;**

**path.add(nums[i]);**

**dfs(nums, used, path, ans);**

**path.remove(path.size() - 1);**

**used[i] = false;**

**}**

**}**

**}**

# [48. Rotate Image](https://leetcode.com/problems/rotate-image)

**class Solution {**

**public void rotate(int[][] matrix) {**

**for (int i = 0, j = matrix.length - 1; i < j; ++i, --j) {**

**int[] temp = matrix[i];**

**matrix[i] = matrix[j];**

**matrix[j] = temp;**

**}**

**for (int i = 0; i < matrix.length; ++i)**

**for (int j = i + 1; j < matrix.length; ++j) {**

**final int temp = matrix[i][j];**

**matrix[i][j] = matrix[j][i];**

**matrix[j][i] = temp;**

**}**

**}**

**}**

# [49. Group Anagrams](https://leetcode.com/problems/group-anagrams)

**class Solution {**

**public List<List<String>> groupAnagrams(String[] strs) {**

**Map<String, List<String>> keyToAnagrams = new HashMap<>();**

**for (final String str : strs) {**

**char[] chars = str.toCharArray();**

**Arrays.sort(chars);**

**String key = String.valueOf(chars);**

**keyToAnagrams.computeIfAbsent(key, k -> new ArrayList<>()).add(str);**

**}**

**return new ArrayList<>(keyToAnagrams.values());**

**}**

**}**

# [50. Pow(x, n)](https://leetcode.com/problems/powx-n)[¶](https://walkccc.me/LeetCode/problems/50/#50-powx-n)

**class Solution {**

**public double myPow(double x, long n) {**

**if (n == 0)**

**return 1;**

**if (n < 0)**

**return 1 / myPow(x, -n);**

**if (n % 2 == 1)**

**return x \* myPow(x, n - 1);**

**return myPow(x \* x, n / 2);**

**}**

**}**

# [51. N-Queens](https://leetcode.com/problems/n-queens)

**class Solution {**

**public List<List<String>> solveNQueens(int n) {**

**List<List<String>> ans = new ArrayList<>();**

**char[][] board = new char[n][n];**

**for (int i = 0; i < n; ++i)**

**Arrays.fill(board[i], '.');**

**dfs(n, 0, new boolean[n], new boolean[2 \* n - 1], new boolean[2 \* n - 1], board, ans);**

**return ans;**

**}**

**private void dfs(int n, int i, boolean[] cols, boolean[] diag1, boolean[] diag2, char[][] board,**

**List<List<String>> ans) {**

**if (i == n) {**

**ans.add(construct(board));**

**return;**

**}**

**for (int j = 0; j < cols.length; ++j) {**

**if (cols[j] || diag1[i + j] || diag2[j - i + n - 1])**

**continue;**

**board[i][j] = 'Q';**

**cols[j] = diag1[i + j] = diag2[j - i + n - 1] = true;**

**dfs(n, i + 1, cols, diag1, diag2, board, ans);**

**cols[j] = diag1[i + j] = diag2[j - i + n - 1] = false;**

**board[i][j] = '.';**

**}**

**}**

**private List<String> construct(char[][] board) {**

**List<String> listBoard = new ArrayList<>();**

**for (int i = 0; i < board.length; ++i)**

**listBoard.add(String.valueOf(board[i]));**

**return listBoard;**

**}**

**}**

# [52. N-Queens II](https://leetcode.com/problems/n-queens-ii)

**class Solution {**

**public int totalNQueens(int n) {**

**dfs(n, 0, new boolean[n], new boolean[2 \* n - 1], new boolean[2 \* n - 1]);**

**return ans;**

**}**

**private int ans = 0**

**private void dfs(int n, int i, boolean[] cols, boolean[] diag1, boolean[] diag2) {**

**if (i == n) {**

**++ans;**

**return;**

**}**

**for (int j = 0; j < cols.length; ++j) {**

**if (cols[j] || diag1[i + j] || diag2[j - i + n - 1])**

**continue;**

**cols[j] = diag1[i + j] = diag2[j - i + n - 1] = true;**

**dfs(n, i + 1, cols, diag1, diag2);**

**cols[j] = diag1[i + j] = diag2[j - i + n - 1] = false;**

**}**

**}**

# [53. Maximum Subarray](https://leetcode.com/problems/maximum-subarray)

**class Solution {**

**public int maxSubArray(int[] nums) {**

**// dp[i] := the maximum sum subarray ending in i**

**int[] dp = new int[nums.length];**

**dp[0] = nums[0];**

**for (int i = 1; i < nums.length; ++i)**

**dp[i] = Math.max(nums[i], dp[i - 1] + nums[i]);**

**return Arrays.stream(dp).max().getAsInt();**

**}**

**}**

# [54. Spiral Matrix](https://leetcode.com/problems/spiral-matrix)

**class Solution {**

**public List<Integer> spiralOrder(int[][] matrix) {**

**if (matrix.length == 0)**

**return new ArrayList<>();**

**final int m = matrix.length;**

**final int n = matrix[0].length;**

**List<Integer> ans = new ArrayList<>();**

**int r1 = 0;**

**int c1 = 0;**

**int r2 = m - 1;**

**int c2 = n - 1;**

**// Repeatedly add matrix[r1..r2][c1..c2] to `ans`.**

**while (ans.size() < m \* n) {**

**for (int j = c1; j <= c2 && ans.size() < m \* n; ++j)**

**ans.add(matrix[r1][j]);**

**for (int i = r1 + 1; i <= r2 - 1 && ans.size() < m \* n; ++i)**

**ans.add(matrix[i][c2]);**

**for (int j = c2; j >= c1 && ans.size() < m \* n; --j)**

**ans.add(matrix[r2][j]);**

**for (int i = r2 - 1; i >= r1 + 1 && ans.size() < m \* n; --i)**

**ans.add(matrix[i][c1]);**

**++r1;**

**++c1;**

**--r2;**

**--c2;**

**}**

**return ans;**

**}**

**}**

# [55. Jump Game](https://leetcode.com/problems/jump-game)

**class Solution {**

**public boolean canJump(int[] nums) {**

**int i = 0;**

**for (int reach = 0; i < nums.length && i <= reach; ++i)**

**reach = Math.max(reach, i + nums[i]);**

**return i == nums.length;**

**}**

**}**

# [56. Merge Intervals](https://leetcode.com/problems/merge-intervals)

**class Solution {**

**public int[][] merge(int[][] intervals) {**

**List<int[]> ans = new ArrayList<>();**

**Arrays.sort(intervals, (a, b) -> (a[0] - b[0]))**

**for (int[] interval : intervals)**

**if (ans.isEmpty() || ans.get(ans.size() - 1)[1] < interval[0])**

**ans.add(interval);**

**else**

**ans.get(ans.size() - 1)[1] = Math.max(ans.get(ans.size() - 1)[1], interval[1]);**

**return ans.toArray(int[][] ::new);**

**}**

**}**

# [57. Insert Interval](https://leetcode.com/problems/insert-interval)

**class Solution {**

**public int[][] insert(int[][] intervals, int[] newInterval) {**

**final int n = intervals.length;**

**List<int[]> ans = new ArrayList<>();**

**int i = 0;**

**while (i < n && intervals[i][1] < newInterval[0])**

**ans.add(intervals[i++]);**

**// Merge overlapping intervals.**

**while (i < n && intervals[i][0] <= newInterval[1]) {**

**newInterval[0] = Math.min(newInterval[0], intervals[i][0]);**

**newInterval[1] = Math.max(newInterval[1], intervals[i][1]);**

**++i;**

**}**

**ans.add(newInterval);**

**while (i < n)**

**ans.add(intervals[i++]);**

**return ans.toArray(int[][] ::new);**

**}**

**}**

# [58. Length of Last Word](https://leetcode.com/problems/length-of-last-word)

**class Solution {**

**public int lengthOfLastWord(String s) {**

**int i = s.length() - 1;**

**while (i >= 0 && s.charAt(i) == ' ')**

**--i;**

**final int lastIndex = i;**

**while (i >= 0 && s.charAt(i) != ' ')**

**--i;**

**return lastIndex - i;**

**}**

**}**

# [59. Spiral Matrix II](https://leetcode.com/problems/spiral-matrix-ii)

**class Solution {**

**public int[][] generateMatrix(int n) {**

**int[][] ans = new int[n][n];**

**int count = 1;**

**for (int min = 0; min < n / 2; ++min) {**

**final int max = n - min - 1;**

**for (int i = min; i < max; ++i)**

**ans[min][i] = count++;**

**for (int i = min; i < max; ++i)**

**ans[i][max] = count++;**

**for (int i = max; i > min; --i)**

**ans[max][i] = count++;**

**for (int i = max; i > min; --i)**

**ans[i][min] = count++;**

**}**

**if (n % 2 == 1)**

**ans[n / 2][n / 2] = count;**

**return ans;**

**}**

**}**

# [60. Permutation Sequence](https://leetcode.com/problems/permutation-sequence)

**class Solution {**

**public String getPermutation(int n, int k) {**

**StringBuilder sb = new StringBuilder();**

**List<Integer> nums = new ArrayList<>();**

**int[] fact = new int[n + 1]; // fact[i] := i!**

**for (int i = 1; i <= n; ++i)**

**nums.add(i);**

**Arrays.fill(fact, 1);**

**for (int i = 2; i <= n; ++i)**

**fact[i] = fact[i - 1] \* i;**

**--k; // 0-indexed**

**for (int i = n - 1; i >= 0; --i) {**

**final int j = k / fact[i];**

**k %= fact[i];**

**sb.append(nums.get(j));**

**nums.remove(j);**

**}**

**return sb.toString();**

**}**

**}**

# [61. Rotate List](https://leetcode.com/problems/rotate-list)

**class Solution {**

**public ListNode rotateRight(ListNode head, int k) {**

**if (head == null || head.next == null || k == 0)**

**return head;**

**int length = 1;**

**ListNode tail = head;**

**for (; tail.next != null; tail = tail.next)**

**++length;**

**tail.next = head; // Circle the list.**

**final int t = length - k % length;**

**for (int i = 0; i < t; ++i)**

**tail = tail.next;**

**ListNode newHead = tail.next;**

**tail.next = null;**

**return newHead;**

**}**

**}**

# [62. Unique Paths](https://leetcode.com/problems/unique-paths)

**class Solution {**

**public int uniquePaths(int m, int n) {**

**// dp[i][j] := the number of unique paths from (0, 0) to (i, j)**

**int[][] dp = new int[m][n];**

**Arrays.stream(dp).forEach(A -> Arrays.fill(A, 1));**

**for (int i = 1; i < m; ++i)**

**for (int j = 1; j < n; ++j)**

**dp[i][j] = dp[i - 1][j] + dp[i][j - 1];**

**return dp[m - 1][n - 1];**

**}**

**}**

# [63. Unique Paths II](https://leetcode.com/problems/unique-paths-ii)

**class Solution {**

**public int uniquePathsWithObstacles(int[][] obstacleGrid) {**

**final int m = obstacleGrid.length;**

**final int n = obstacleGrid[0].length;**

**// dp[i][j] := the number of unique paths from (0, 0) to (i, j)**

**long[][] dp = new long[m + 1][n + 1];**

**dp[0][1] = 1; // Can also set dp[1][0] = 1.**

**for (int i = 1; i <= m; ++i)**

**for (int j = 1; j <= n; ++j)**

**if (obstacleGrid[i - 1][j - 1] == 0)**

**dp[i][j] = dp[i - 1][j] + dp[i][j - 1];**

**return (int) dp[m][n];**

**}**

**}**

# [64. Minimum Path Sum](https://leetcode.com/problems/minimum-path-sum)

**class Solution {**

**public int minPathSum(int[][] grid) {**

**final int m = grid.length;**

**final int n = grid[0].length;**

**for (int i = 0; i < m; ++i)**

**for (int j = 0; j < n; ++j)**

**if (i > 0 && j > 0)**

**grid[i][j] += Math.min(grid[i - 1][j], grid[i][j - 1]);**

**else if (i > 0)**

**grid[i][0] += grid[i - 1][0];**

**else if (j > 0)**

**grid[0][j] += grid[0][j - 1];**

**return grid[m - 1][n - 1];**

**}**

**}**

# [65. Valid Number](https://leetcode.com/problems/valid-number)

**class Solution {**

**public boolean isNumber(String s) {**

**s = s.trim();**

**if (s.isEmpty())**

**return false;**

**boolean seenNum = false;**

**boolean seenDot = false;**

**boolean seenE = false;**

**for (int i = 0; i < s.length(); ++i) {**

**switch (s.charAt(i)) {**

**case '.':**

**if (seenDot || seenE)**

**return false;**

**seenDot = true;**

**break;**

**case 'e':**

**case 'E':**

**if (seenE || !seenNum)**

**return false;**

**seenE = true;**

**seenNum = false;**

**break;**

**case '+':**

**case '-':**

**if (i > 0 && s.charAt(i - 1) != 'e' && s.charAt(i - 1) != 'E')**

**return false;**

**seenNum = false;**

**break;**

**default:**

**if (!Character.isDigit(s.charAt(i)))**

**return false;**

**seenNum = true;**

**}**

**}**

**return seenNum;**

**}**

**}**

# [66. Plus One](https://leetcode.com/problems/plus-one)

# class Solution {

# public int[] plusOne(int[] digits) {

# for (int i = digits.length - 1; i >= 0; i--) {

# if (digits[i] < 9) {

# ++digits[i];

# return digits;

# }

# digits[i] = 0;

# }

# int[] ans = new int[digits.length + 1];

# ans[0] = 1;

# return ans;

# }

# }

# [67. Add Binary](https://leetcode.com/problems/add-binary)

# class Solution {

# public String addBinary(String a, String b) {

# StringBuilder sb = new StringBuilder();

# int carry = 0;

# int i = a.length() - 1;

# int j = b.length() - 1;

# while (i >= 0 || j >= 0 || carry == 1) {

# if (i >= 0)

# carry += a.charAt(i--) - '0';

# if (j >= 0)

# carry += b.charAt(j--) - '0';

# sb.append(carry % 2);

# carry /= 2;

# }

# return sb.reverse().toString();

# }

# }

# [68. Text Justification](https://leetcode.com/problems/text-justification)

# class Solution {

# public List<String> fullJustify(String[] words, int maxWidth) {

# List<String> ans = new ArrayList<>();

# List<StringBuilder> row = new ArrayList<>();

# int rowLetters = 0;

# for (final String word : words) {

# // If we place the word in this row, it will exceed the maximum width.

# // Therefore, we cannot put the word in this row and have to pad spaces

# // for each word in this row.

# if (rowLetters + row.size() + word.length() > maxWidth) {

# final int spaces = maxWidth - rowLetters;

# if (row.size() == 1) {

# // Pad all the spaces after row[0].

# for (int i = 0; i < spaces; ++i)

# row.get(0).append(" ");

# } else {

# // Evenly pad all the spaces to each word (expect the last one) in

# // this row.

# for (int i = 0; i < spaces; ++i)

# row.get(i % (row.size() - 1)).append(" ");

# }

# final String joinedRow =

# row.stream().map(StringBuilder::toString).collect(Collectors.joining(""));

# ans.add(joinedRow);

# row.clear();

# rowLetters = 0;

# }

# row.add(new StringBuilder(word));

# rowLetters += word.length();

# }

# final String lastRow =

# row.stream().map(StringBuilder::toString).collect(Collectors.joining(" "));

# StringBuilder sb = new StringBuilder(lastRow);

# final int spacesToBeAdded = maxWidth - sb.length();

# for (int i = 0; i < spacesToBeAdded; ++i)

# sb.append(" ");

# ans.add(sb.toString());

# return ans;

# }

# }

# [69. Sqrt(x)](https://leetcode.com/problems/sqrtx)

# class Solution {

# public int mySqrt(long x) {

# long l = 1;

# long r = x + 1;

# while (l < r) {

# final long m = (l + r) / 2;

# if (m > x / m)

# r = m;

# else

# l = m + 1;

# }

# // l := the minimum number s.t. l \* l > x

# return (int) l - 1;

# }

# }

# [70. Climbing Stairs](https://leetcode.com/problems/climbing-stairs)

# class Solution {

# public int climbStairs(int n) {

# // dp[i] := the number of ways to climb to the i-th stair

# int[] dp = new int[n + 1];

# dp[0] = 1;

# dp[1] = 1;

# for (int i = 2; i <= n; ++i)

# dp[i] = dp[i - 1] + dp[i - 2];

# return dp[n];

# }

# }

# [71. Simplify Path](https://leetcode.com/problems/simplify-path)

# ` class Solution {

# public String simplifyPath(String path) {

# final String[] dirs = path.split("/");

# Stack<String> stack = new Stack<>();

# for (final String dir : dirs) {

# if (dir.isEmpty() || dir.equals("."))

# continue;

# if (dir.equals("..")) {

# if (!stack.isEmpty())

# stack.pop();

# } else {

# stack.push(dir);

# }

# }

# return "/" + String.join("/", stack);

# }

# }

# [72. Edit Distance](https://leetcode.com/problems/edit-distance)

# class Solution {

# public int minDistance(String word1, String word2) {

# final int m = word1.length();

# final int n = word2.length();

# // dp[i][j] := the minimum number of operations to convert word1[0..i) to

# // word2[0..j)

# int[][] dp = new int[m + 1][n + 1];

# for (int i = 1; i <= m; ++i)

# dp[i][0] = i;

# for (int j = 1; j <= n; ++j)

# dp[0][j] = j;

# for (int i = 1; i <= m; ++i)

# for (int j = 1; j <= n; ++j)

# if (word1.charAt(i - 1) == word2.charAt(j - 1))

# dp[i][j] = dp[i - 1][j - 1];

# else

# dp[i][j] = Math.min(dp[i - 1][j - 1], Math.min(dp[i - 1][j], dp[i][j - 1])) + 1;

# return dp[m][n];

# }

# }

# [73. Set Matrix Zeroes](https://leetcode.com/problems/set-matrix-zeroes)

# class Solution {

# public void setZeroes(int[][] matrix) {

# final int m = matrix.length;

# final int n = matrix[0].length;

# boolean shouldFillFirstRow = false;

# boolean shouldFillFirstCol = false;

# for (int j = 0; j < n; ++j)

# if (matrix[0][j] == 0) {

# shouldFillFirstRow = true;

# break;

# }

# for (int i = 0; i < m; ++i)

# if (matrix[i][0] == 0) {

# shouldFillFirstCol = true;

# break;

# }

# // Store the information in the first row and the first column.

# for (int i = 1; i < m; ++i)

# for (int j = 1; j < n; ++j)

# if (matrix[i][j] == 0) {

# matrix[i][0] = 0;

# matrix[0][j] = 0;

# }

# // Fill 0s for the matrix except the first row and the first column.

# for (int i = 1; i < m; ++i)

# for (int j = 1; j < n; ++j)

# if (matrix[i][0] == 0 || matrix[0][j] == 0)

# matrix[i][j] = 0;

# // Fill 0s for the first row if needed.

# if (shouldFillFirstRow)

# for (int j = 0; j < n; ++j)

# matrix[0][j] = 0;

# // Fill 0s for the first column if needed.

# if (shouldFillFirstCol)

# for (int i = 0; i < m; ++i)

# matrix[i][0] = 0;

# }

# }

# [74. Search a 2D Matrix](https://leetcode.com/problems/search-a-2d-matrix)

# class Solution {

# public boolean searchMatrix(int[][] matrix, int target) {

# if (matrix.length == 0)

# return false;

# final int m = matrix.length;

# final int n = matrix[0].length;

# int l = 0;

# int r = m \* n;

# while (l < r) {

# final int mid = (l + r) / 2;

# final int i = mid / n;

# final int j = mid % n;

# if (matrix[i][j] == target)

# return true;

# if (matrix[i][j] < target)

# l = mid + 1;

# else

# r = mid;

# }

# return false;

# }

# }

# [75. Sort Colors](https://leetcode.com/problems/sort-colors)

# class Solution {

# public void sortColors(int[] nums) {

# int zero = -1;

# int one = -1;

# int two = -1;

# for (final int num : nums)

# if (num == 0) {

# nums[++two] = 2;

# nums[++one] = 1;

# nums[++zero] = 0;

# } else if (num == 1) {

# nums[++two] = 2;

# nums[++one] = 1;

# } else {

# nums[++two] = 2;

# }

# }

# }

# [76. Minimum Window Substring](https://leetcode.com/problems/minimum-window-substring)

# class Solution {

# public String minWindow(String s, String t) {

# int[] count = new int[128];

# int required = t.length();

# int bestLeft = -1;

# int minLength = s.length() + 1;

# for (final char c : t.toCharArray())

# ++count[c];

# for (int l = 0, r = 0; r < s.length(); ++r) {

# if (--count[s.charAt(r)] >= 0)

# --required;

# while (required == 0) {

# if (r - l + 1 < minLength) {

# bestLeft = l;

# minLength = r - l + 1;

# }

# if (++count[s.charAt(l++)] > 0)

# ++required;

# }

# }

# return bestLeft == -1 ? "" : s.substring(bestLeft, bestLeft + minLength);

# }

# }

# [77. Combinations](https://leetcode.com/problems/combinations)

# class Solution {

# public List<List<Integer>> combine(int n, int k) {

# List<List<Integer>> ans = new ArrayList<>();

# dfs(n, k, 1, new ArrayList<>(), ans);

# return ans;

# }

# private void dfs(int n, int k, int s, List<Integer> path, List<List<Integer>> ans) {

# if (path.size() == k) {

# ans.add(new ArrayList<>(path));

# return;

# }

# for (int i = s; i <= n; ++i) {

# path.add(i);

# dfs(n, k, i + 1, path, ans);

# path.remove(path.size() - 1);

# }

# }

# [78. Subsets](https://leetcode.com/problems/subsets)

# class Solution {

# public List<List<Integer>> subsets(int[] nums) {

# List<List<Integer>> ans = new ArrayList<>();

# dfs(nums, 0, new ArrayList<>(), ans);

# return ans;

# }

# private void dfs(int[] nums, int s, List<Integer> path, List<List<Integer>> ans) {

# ans.add(new ArrayList<>(path));

# for (int i = s; i < nums.length; ++i) {

# path.add(nums[i]);

# dfs(nums, i + 1, path, ans);

# path.remove(path.size() - 1);

# }

# }

# }

# [79. Word Search](https://leetcode.com/problems/word-search)

# class Solution {

# public boolean exist(char[][] board, String word) {

# for (int i = 0; i < board.length; ++i)

# for (int j = 0; j < board[0].length; ++j)

# if (dfs(board, word, i, j, 0))

# return true;

# return false;

# }

# private boolean dfs(char[][] board, String word, int i, int j, int s) {

# if (i < 0 || i == board.length || j < 0 || j == board[0].length)

# return false;

# if (board[i][j] != word.charAt(s) || board[i][j] == '\*')

# return false;

# if (s == word.length() - 1)

# return true;

# final char cache = board[i][j];

# board[i][j] = '\*';

# final boolean isExist = dfs(board, word, i + 1, j, s + 1) || //

# dfs(board, word, i - 1, j, s + 1) || //

# dfs(board, word, i, j + 1, s + 1) || //

# dfs(board, word, i, j - 1, s + 1);

# board[i][j] = cache;

# return isExist;

# }

# }

# [80. Remove Duplicates from Sorted Array II](https://leetcode.com/problems/remove-duplicates-from-sorted-array-ii)

# class Solution {

# public int removeDuplicates(int[] nums) {

# int i = 0;

# for (final int num : nums)

# if (i < 2 || num > nums[i - 2])

# nums[i++] = num;

# return i;

# }

# }

# [81. Search in Rotated Sorted Array II](https://leetcode.com/problems/search-in-rotated-sorted-array-ii)

# class Solution {

# public boolean search(int[] nums, int target) {

# int l = 0;

# int r = nums.length - 1;

# while (l <= r) {

# final int m = (l + r) / 2;

# if (nums[m] == target)

# return true;

# if (nums[l] == nums[m] && nums[m] == nums[r]) {

# ++l;

# --r;

# } else if (nums[l] <= nums[m]) { // nums[l..m] are sorted

# if (nums[l] <= target && target < nums[m])

# r = m - 1;

# else

# l = m + 1;

# } else { // nums[m..n - 1] are sorted

# if (nums[m] < target && target <= nums[r])

# l = m + 1;

# else

# r = m - 1;

# }

# }

# return false;

# }

# }

# [82. Remove Duplicates from Sorted List II](https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii)

# class Solution {

# public ListNode deleteDuplicates(ListNode head) {

# ListNode dummy = new ListNode(0, head);

# ListNode prev = dummy;

# while (head != null) {

# while (head.next != null && head.val == head.next.val)

# head = head.next;

# if (prev.next == head)

# prev = prev.next;

# else

# prev.next = head.next;

# head = head.next;

# }

# return dummy.next;

# }

# }

# [83. Remove Duplicates from Sorted List](https://leetcode.com/problems/remove-duplicates-from-sorted-list)

# class Solution {

# public ListNode deleteDuplicates(ListNode head) {

# ListNode curr = head;

# while (curr != null) {

# while (curr.next != null && curr.val == curr.next.val)

# curr.next = curr.next.next;

# curr = curr.next;

# }

# return head;

# }

# }

# [84. Largest Rectangle in Histogram](https://leetcode.com/problems/largest-rectangle-in-histogram)

# class Solution {

# public int largestRectangleArea(int[] heights) {

# int ans = 0;

# Deque<Integer> stack = new ArrayDeque<>();

# for (int i = 0; i <= heights.length; ++i) {

# while (!stack.isEmpty() && (i == heights.length || heights[stack.peek()] > heights[i])) {

# final int h = heights[stack.pop()];

# final int w = stack.isEmpty() ? i : i - stack.peek() - 1;

# ans = Math.max(ans, h \* w);

# }

# stack.push(i);

# }

# return ans;

# }

# }

# [85. Maximal Rectangle](https://leetcode.com/problems/maximal-rectangle)

# class Solution {

# public int maximalRectangle(char[][] matrix) {

# if (matrix.length == 0)

# return 0;

# int ans = 0;

# int[] hist = new int[matrix[0].length];

# for (char[] row : matrix) {

# for (int i = 0; i < row.length; ++i)

# hist[i] = row[i] == '0' ? 0 : hist[i] + 1;

# ans = Math.max(ans, largestRectangleArea(hist));

# }

# return ans;

# }

# private int largestRectangleArea(int[] heights) {

# int ans = 0;

# Deque<Integer> stack = new ArrayDeque<>();

# for (int i = 0; i <= heights.length; ++i) {

# while (!stack.isEmpty() && (i == heights.length || heights[stack.peek()] > heights[i])) {

# final int h = heights[stack.pop()];

# final int w = stack.isEmpty() ? i : i - stack.peek() - 1;

# ans = Math.max(ans, h \* w);

# }

# stack.push(i);

# }

# return ans;

# }

# }

# [86. Partition List](https://leetcode.com/problems/partition-list)

# class Solution {

# public ListNode partition(ListNode head, int x) {

# ListNode beforeHead = new ListNode(0);

# ListNode afterHead = new ListNode(0);

# ListNode before = beforeHead;

# ListNode after = afterHead;

# for (; head != null; head = head.next)

# if (head.val < x) {

# before.next = head;

# before = head;

# } else {

# after.next = head;

# after = head;

# }

# after.next = null;

# before.next = afterHead.next;

# return beforeHead.next;

# }

# }

# [87. Scramble String](https://leetcode.com/problems/scramble-string)

# class Solution {

# public boolean isScramble(String s1, String s2) {

# if (s1.equals(s2))

# return true;

# final String hashKey = s1 + "+" + s2;

# if (mem.containsKey(hashKey))

# return mem.get(hashKey);

# int[] count = new int[128];

# for (int i = 0; i < s1.length(); ++i) {

# ++count[s1.charAt(i)];

# --count[s2.charAt(i)];

# }

# for (final int freq : count)

# if (freq != 0) {

# mem.put(hashKey, false);

# return false;

# }

# for (int i = 1; i < s1.length(); ++i) {

# if (isScramble(s1.substring(0, i), s2.substring(0, i)) &&

# isScramble(s1.substring(i), s2.substring(i))) {

# mem.put(hashKey, true);

# return true;

# }

# if (isScramble(s1.substring(0, i), s2.substring(s2.length() - i)) &&

# isScramble(s1.substring(i), s2.substring(0, s2.length() - i))) {

# mem.put(hashKey, true);

# return true;

# }

# }

# mem.put(hashKey, false);

# return false;

# }

# private Map<String, Boolean> mem = new HashMap<>();

# }

# [88. Merge Sorted Array](https://leetcode.com/problems/merge-sorted-array)

# class Solution {

# public void merge(int[] nums1, int m, int[] nums2, int n) {

# int i = m - 1; // nums1's index (the actual nums)

# int j = n - 1; // nums2's index

# int k = m + n - 1; // nums1's index (the next filled position)

# while (j >= 0)

# if (i >= 0 && nums1[i] > nums2[j])

# nums1[k--] = nums1[i--];

# else

# nums1[k--] = nums2[j--];

# }

# }

# [89. Gray Code](https://leetcode.com/problems/gray-code)

# class Solution {

# public List<Integer> grayCode(int n) {

# List<Integer> ans = new ArrayList<>();

# ans.add(0);

# for (int i = 0; i < n; ++i)

# for (int j = ans.size() - 1; j >= 0; --j)

# ans.add(ans.get(j) | 1 << i);

# return ans;

# }

# }

# [90. Subsets II](https://leetcode.com/problems/subsets-ii)

# class Solution {

# public List<List<Integer>> subsetsWithDup(int[] nums) {

# List<List<Integer>> ans = new ArrayList<>();

# Arrays.sort(nums);

# dfs(nums, 0, new ArrayList<>(), ans);

# return ans;

# }

# private void dfs(int[] nums, int s, List<Integer> path, List<List<Integer>> ans) {

# ans.add(new ArrayList<>(path));

# for (int i = s; i < nums.length; ++i) {

# if (i > s && nums[i] == nums[i - 1])

# continue;

# path.add(nums[i]);

# dfs(nums, i + 1, path, ans);

# path.remove(path.size() - 1);

# }

# }

# }

# [91. Decode Ways](https://leetcode.com/problems/decode-ways)

# class Solution {

# public int numDecodings(String s) {

# final int n = s.length();

# // dp[i] := the number of ways to decode s[i..n)

# int[] dp = new int[n + 1];

# dp[n] = 1; // ""

# dp[n - 1] = isValid(s.charAt(n - 1)) ? 1 : 0;

# for (int i = n - 2; i >= 0; --i) {

# if (isValid(s.charAt(i)))

# dp[i] += dp[i + 1];

# if (isValid(s.charAt(i), s.charAt(i + 1)))

# dp[i] += dp[i + 2];

# }

# return dp[0];

# }

# private boolean isValid(char c) {

# return c != '0';

# }

# private boolean isValid(char c1, char c2) {

# return c1 == '1' || c1 == '2' && c2 < '7';

# }

# }

# [92. Reverse Linked List II](https://leetcode.com/problems/reverse-linked-list-ii)

# class Solution {

# public ListNode reverseBetween(ListNode head, int left, int right) {

# if (left == 1)

# return reverseN(head, right)

# head.next = reverseBetween(head.next, left - 1, right - 1);

# return head;

# }

# private ListNode reverseN(ListNode head, int n) {

# if (n == 1)

# return head;

# ListNode newHead = reverseN(head.next, n - 1);

# ListNode headNext = head.next;

# head.next = headNext.next;

# headNext.next = head;

# return newHead;

# }

# }

# [93. Restore IP Addresses](https://leetcode.com/problems/restore-ip-addresses)

# class Solution {

# public List<String> restoreIpAddresses(final String s) {

# List<String> ans = new ArrayList<>();

# dfs(s, 0, new ArrayList<>(), ans);

# return ans;

# }

# private void dfs(final String s, int start, List<String> path, List<String> ans) {

# if (path.size() == 4 && start == s.length()) {

# ans.add(String.join(".", path));

# return;

# }

# if (path.size() == 4 || start == s.length())

# return;

# for (int length = 1; length <= 3; ++length) {

# if (start + length > s.length()) // out-of-bounds

# return;

# if (length > 1 && s.charAt(start) == '0') // leading '0'

# return;

# final String num = s.substring(start, start + length);

# if (Integer.parseInt(num) > 255)

# return;

# path.add(num);

# dfs(s, start + length, path, ans);

# path.remove(path.size() - 1);

# }

# }

# }

# [94. Binary Tree Inorder Traversal](https://leetcode.com/problems/binary-tree-inorder-traversal)

# class Solution {

# public List<Integer> inorderTraversal(TreeNode root) {

# List<Integer> ans = new ArrayList<>();

# Deque<TreeNode> stack = new ArrayDeque<>();

# while (root != null || !stack.isEmpty()) {

# while (root != null) {

# stack.push(root);

# root = root.left;

# }

# root = stack.pop();

# ans.add(root.val);

# root = root.right;

# }

# return ans;

# }

# }

# [95. Unique Binary Search Trees II](https://leetcode.com/problems/unique-binary-search-trees-ii)

# class Solution {

# public List<TreeNode> generateTrees(int n) {

# if (n == 0)

# return new ArrayList<>();

# return generateTrees(1, n);

# }

# private List<TreeNode> generateTrees(int min, int max) {

# if (min > max)

# return Arrays.asList((TreeNode) null);

# List<TreeNode> ans = new ArrayList<>();

# for (int i = min; i <= max; ++i)

# for (TreeNode left : generateTrees(min, i - 1))

# for (TreeNode right : generateTrees(i + 1, max)) {

# ans.add(new TreeNode(i));

# ans.get(ans.size() - 1).left = left;

# ans.get(ans.size() - 1).right = right;

# }

# return ans;

# }

# }

# [96. Unique Binary Search Trees](https://leetcode.com/problems/unique-binary-search-trees)

# class Solution {

# public int numTrees(int n) {

# // dp[i] := the number of unique BST's that store values 1..i

# int[] dp = new int[n + 1];

# dp[0] = 1;

# dp[1] = 1;

# for (int i = 2; i <= n; ++i)

# for (int j = 0; j < i; ++j)

# dp[i] += dp[j] \* dp[i - j - 1];

# return dp[n];

# }

# }

# [97. Interleaving String](https://leetcode.com/problems/interleaving-string)

# class Solution {

# public boolean isInterleave(String s1, String s2, String s3) {

# final int m = s1.length();

# final int n = s2.length();

# if (m + n != s3.length())

# return false;

# boolean[][] dp = new boolean[m + 1][n + 1];

# dp[0][0] = true;

# for (int i = 1; i <= m; ++i)

# dp[i][0] = dp[i - 1][0] && s1.charAt(i - 1) == s3.charAt(i - 1);

# for (int j = 1; j <= n; ++j)

# dp[0][j] = dp[0][j - 1] && s2.charAt(j - 1) == s3.charAt(j - 1);

# for (int i = 1; i <= m; ++i)

# for (int j = 1; j <= n; ++j)

# dp[i][j] = dp[i - 1][j] && s1.charAt(i - 1) == s3.charAt(i + j - 1) ||

# dp[i][j - 1] && s2.charAt(j - 1) == s3.charAt(i + j - 1);

# return dp[m][n];

# }

# }

# [98. Validate Binary Search Tree](https://leetcode.com/problems/validate-binary-search-tree)

# class Solution {

# public boolean isValidBST(TreeNode root) {

# return isValidBST(root, null, null);

# }

# private boolean isValidBST(TreeNode root, TreeNode minNode, TreeNode maxNode) {

# if (root == null)

# return true;

# if (minNode != null && root.val <= minNode.val)

# return false;

# if (maxNode != null && root.val >= maxNode.val)

# return false;

# return //

# isValidBST(root.left, minNode, root) && //

# isValidBST(root.right, root, maxNode);

# }

# }

# [99. Recover Binary Search Tree](https://leetcode.com/problems/recover-binary-search-tree)

# class Solution {

# public void recoverTree(TreeNode root) {

# inorder(root);

# swap(x, y);

# }

# private TreeNode pred = null;

# private TreeNode x = null; // the first wrong node

# private TreeNode y = null; // the second wrong node

# private void inorder(TreeNode root) {

# if (root == null)

# return

# inorder(root.left);

# if (pred != null && root.val < pred.val) {

# y = root;

# if (x == null)

# x = pred;

# else

# return;

# }

# pred = root;

# inorder(root.right);

# }

# private void swap(TreeNode x, TreeNode y) {

# final int temp = x.val;

# x.val = y.val;

# y.val = temp;

# }

# }

# [100. Same Tree](https://leetcode.com/problems/same-tree)

# class Solution {

# public boolean isSameTree(TreeNode p, TreeNode q) {

# if (p == null || q == null)

# return p == q;

# return p.val == q.val && //

# isSameTree(p.left, q.left) && //

# isSameTree(p.right, q.right);

# }

# }