



Leetcode.typ

Gabriel Wu (@lucifer1004)

January 01, 2026

Contents

| | |
|--|----|
| 0001. Two Sum | 1 |
| 0002. Add Two Numbers | 3 |
| 0003. Longest Substring Without Repeating Characters | 5 |
| 0004. Median of Two Sorted Arrays | 6 |
| 0005. Longest Palindromic Substring | 7 |
| 0006. Zigzag Conversion | 9 |
| 0007. Reverse Integer | 10 |
| 0008. String to Integer (atoi) | 12 |
| 0009. Palindrome Number | 13 |
| 0010. Regular Expression Matching | 14 |
| 0011. Container With Most Water | 16 |
| 0012. Integer to Roman | 18 |
| 0013. Roman to Integer | 20 |
| 0014. Longest Common Prefix | 22 |
| 0015. 3Sum | 23 |
| 0016. 3Sum Closest | 25 |
| 0017. Letter Combinations of a Phone Number | 27 |
| 0018. 4Sum | 29 |
| 0019. Remove Nth Node From End of List | 30 |
| 0020. Valid Parentheses | 31 |
| 0021. Merge Two Sorted Lists | 33 |
| 0022. Generate Parentheses | 34 |
| 0023. Merge k Sorted Lists | 35 |
| 0024. Swap Nodes in Pairs | 36 |
| 0025. Reverse Nodes in k-Group | 37 |
| 0026. Remove Duplicates from Sorted Array (Adapted) | 38 |
| 0042. Trapping Rain Water | 39 |
| 0051. N-Queens | 41 |
| 0094. Binary Tree Inorder Traversal | 42 |
| 0110. Balanced Binary Tree | 44 |
| 0144. Binary Tree Preorder Traversal | 46 |
| 0145. Binary Tree Postorder Traversal | 48 |

0001. Two Sum

Given an array of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`.

You may assume that each input would have **exactly one solution**, and you may not use the same element twice.

You can return the answer in any order.

Test Results

Case 1

nums: [2, 7, 11, 15]

target: 9

| Expected | Your Output |
|----------|-------------|
| [0, 1] | none |

Case 2

nums: [3, 2, 4]

target: 6

| Expected | Your Output |
|----------|-------------|
| [1, 2] | none |

Case 3

nums: [3, 3]

target: 6

| Expected | Your Output |
|----------|-------------|
| [0, 1] | none |

Case 4

nums: [0, 0]

target: 1

| Expected | Your Output |
|----------|-------------|
| [-1, -1] | none |

Case 5

nums: [1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49, 52, 55, 58, 61, 64, 67, 70, 73, 76, 79, 82, 85, 88, 91, 94, 97]

target: 191

| Expected | Your Output |
|----------|-------------|
| [31, 32] | none |

0002. Add Two Numbers

You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Test Results

Case 1

l1: 2 → 4 → 3 → ∅

l2: 5 → 6 → 4 → ∅

| Expected |
|---------------|
| 7 → 0 → 8 → ∅ |

| Your Output |
|-------------|
| none |

Case 2

l1: 0 → ∅

l2: 0 → ∅

| Expected |
|----------|
| 0 → ∅ |

| Your Output |
|-------------|
| none |

Case 3

l1: 9 → 9 → 9 → 9 → 9 → 9 → 9 → ∅

l2: 9 → 9 → 9 → 9 → ∅


| Expected |
|-----------------------------------|
| 8 → 9 → 9 → 9 → 0 → 0 → 0 → 1 → ∅ |

| Your Output |
|-------------|
| none |

Case 4

l1: 2 → 4 → 3 → ∅

l2: 5 → 6 → 4 → 9 → ∅

| Expected | Your Output |
|---|-------------|
|  | none |

0003. Longest Substring Without Repeating Characters

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

Test Results

Case 1

s: "abcabcbb"

| Expected | Your Output |
|----------|-------------|
| 3 | none |

Case 2

s: "bbbbbb"

| Expected | Your Output |
|----------|-------------|
| 1 | none |

Case 3

s: "pwwkew"

| Expected | Your Output |
|----------|-------------|
| 3 | none |

0004. Median of Two Sorted Arrays

Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return the **median** of the two sorted arrays.

The overall run time complexity should be $\mathcal{O}(\log(m + n))$.

Test Results

Case 1

nums1: [1, 3]

nums2: [2]

| Expected | Your Output |
|----------|-------------|
| 2 | none |

Case 2

nums1: [1, 2]

nums2: [3, 4]

| Expected | Your Output |
|----------|-------------|
| 2.5 | none |

Case 3

nums1: [0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99]

nums2: [0, 6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84, 90, 96, 102, 108, 114, 120, 126, 132, 138, 144, 150, 156, 162, 168, 174, 180, 186, 192, 198]

| Expected | Your Output |
|----------|-------------|
| 66.0 | none |

0005. Longest Palindromic Substring

Given a string *s*, return the **longest palindromic substring** in *s*.

Test Results

Case 1

s: "babad"

| Expected | Your Output |
|----------|-------------|
| "bab" | none |

Case 2

s: "cbbd"

| Expected | Your Output |
|----------|-------------|
| "bb" | none |

Case 3

s: "abcdefgfedcb"

| Expected | Your Output |
|---------------|-------------|
| "bcdefgfedcb" | none |

Case 4

s: "accc"

| Expected | Your Output |
|----------|-------------|
| "ccc" | none |

Case 5

s: "a"

| Expected | Your Output |
|----------|-------------|
| "a" | none |

Case 6

s: "aa"

| Expected | Your Output |
|----------|-------------|
| | |

| |
|------|
| "aa" |
|------|

| |
|------|
| none |
|------|

Case 7

s: "asasfsafdaasfsaasa"

| Expected |
|-----------|
| "aasfsaa" |

| Your Output |
|-------------|
| none |

0006. Zigzag Conversion

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this:

```
P   A   H   N
A P L S I I G
Y   I   R
```

And then read line by line: "PAHNAPLSIIGYIR"

Write the code that will take a string and make this conversion given a number of rows.

Test Results

Case 1

s: "PAYPALISHIRING"

numRows: 3

| Expected | Your Output |
|------------------|-------------|
| "PAHNAPLSIIGYIR" | none |

Case 2

s: "PAYPALISHIRING"

numRows: 4

| Expected | Your Output |
|------------------|-------------|
| "PINALSIGYAHRPI" | none |

Case 3

s: "A"

numRows: 1

| Expected | Your Output |
|----------|-------------|
| "A" | none |

0007. Reverse Integer

Given a signed 32-bit integer x , return x with its digits reversed. If reversing x causes the value to go outside the signed 32-bit integer range $[-2^{31}, 2^{31} - 1]$, then return 0.

Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

Test Results

Case 1

x: 123

| Expected |
|----------|
| 321 |

| Your Output |
|-------------|
| none |

Case 2

x: -123

| Expected |
|----------|
| -321 |

| Your Output |
|-------------|
| none |

Case 3

x: 120

| Expected |
|----------|
| 21 |

| Your Output |
|-------------|
| none |

Case 4

x: 0

| Expected |
|----------|
| 0 |

| Your Output |
|-------------|
| none |

Case 5

x: 23498423

| Expected |
|----------|
| 32489432 |

| Your Output |
|-------------|
| none |

Case 6

x: -213898800

| Expected |
|----------|
| -8898312 |

| Your Output |
|-------------|
| none |

Case 7

x: 1534236469

| Expected |
|----------|
| 0 |

| Your Output |
|-------------|
| none |

Case 8

x: 2147483647

| Expected |
|----------|
| 0 |

| Your Output |
|-------------|
| none |

Case 9

x: -2147483648

| Expected |
|----------|
| 0 |

| Your Output |
|-------------|
| none |

0008. String to Integer (atoi)

Implement the `myAtoi(string s)` function, which converts a string to a 32-bit signed integer (similar to C/C++'s `atoi` function).

The algorithm for `myAtoi(string s)` is as follows:

1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
3. Read in next the characters until the next non-digit character or the end of the input is reached. The rest of the string is ignored.
4. Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2).
5. If the integer is out of the 32-bit signed integer range $[-2^{31}, 2^{31} - 1]$, then clamp the integer so that it remains in the range. Specifically, integers less than -2^{31} should be clamped to -2^{31} , and integers greater than $2^{31} - 1$ should be clamped to $2^{31} - 1$.
6. Return the integer as the final result.

Note:

- Only the space character ' ' is considered a whitespace character.
- **Do not ignore** any characters other than the leading whitespace or the rest of the string after the digits.

Test Results

Case 1

s: "42"

| Expected | Your Output |
|----------|-------------|
| 42 | none |

Case 2

s: " -42"

| Expected | Your Output |
|----------|-------------|
| -42 | none |

Case 3

s: "4193 with words"

| Expected | Your Output |
|----------|-------------|
| 4193 | none |

0009. Palindrome Number

Given an integer x , return `true` if x is a **palindrome**, and `false` otherwise.

Test Results

Case 1

x: 121

| Expected |
|----------|
| true |

| Your Output |
|-------------|
| none |

Case 2

x: -121

| Expected |
|----------|
| false |

| Your Output |
|-------------|
| none |

Case 3

x: 10

| Expected |
|----------|
| false |

| Your Output |
|-------------|
| none |

0010. Regular Expression Matching

Given an input string *s* and a pattern *p*, implement regular expression matching with support for '.' and '*' where:

- '.' Matches any single character.
- '*' Matches zero or more of the preceding element.

The matching should cover the **entire** input string (not partial).

Test Results

Case 1

s: "aa"

p: "a"

| Expected |
|----------|
| false |

| Your Output |
|-------------|
| none |

Case 2

s: "aa"

p: "a*"

| Expected |
|----------|
| true |

| Your Output |
|-------------|
| none |

Case 3

s: "ab"

p: ".*"

| Expected |
|----------|
| true |

| Your Output |
|-------------|
| none |

Case 4

s: "aab"

p: "c*a*b"

| Expected |
|----------|
| true |

| Your Output |
|-------------|
| none |

Case 5

s: "mississippi"

p: "mis*is*p*."

| Expected |
|----------|
| false |

| Your Output |
|-------------|
| none |

Case 6

s: "ab"

p: ".*c"

| Expected |
|----------|
| false |

| Your Output |
|-------------|
| none |

Case 7

s: "ab"

p: ".*c*"

| Expected |
|----------|
| true |

| Your Output |
|-------------|
| none |

Case 8

s: "香蕉 x 牛奶"

p: "香.*牛."

| Expected |
|----------|
| true |

| Your Output |
|-------------|
| none |

0011. Container With Most Water

You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the i^{th} line are $(i, 0)$ and $(i, \text{height}[i])$.

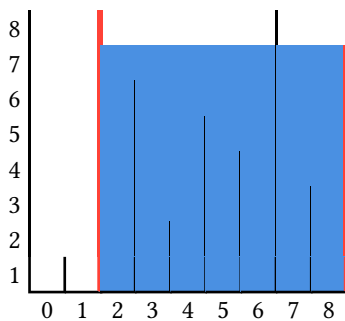
Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return the maximum amount of water a container can store.

Notice that you may not slant the container.

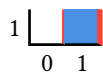
Example 1

height: [1, 8, 6, 2, 5, 4, 8, 3, 7]



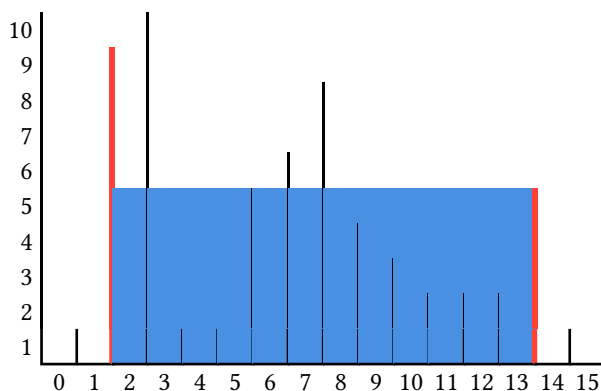
Example 2

height: [1, 1]



Example 3

height: [1, 9, 10, 1, 1, 5, 6, 8, 4, 3, 2, 2, 2, 5, 1, 2]



Test Results

Case 1

height: [1, 8, 6, 2, 5, 4, 8, 3, 7]

| Expected |
|----------|
| 49 |

| Your Output |
|-------------|
| none |

Case 2

height: [1, 1]

| Expected |
|----------|
| 1 |

| Your Output |
|-------------|
| none |

Case 3

height: [1, 9, 10, 1, 1, 5, 6, 8, 4, 3, 2, 2, 2, 5, 1, 2]

| Expected |
|----------|
| 60 |

| Your Output |
|-------------|
| none |

0012. Integer to Roman

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

| Symbol | Value |
|--------|-------|
| I | 1 |
| V | 5 |
| X | 10 |
| L | 50 |
| C | 100 |
| D | 500 |
| M | 1000 |

For example, 2 is written as II in Roman numeral, just two one's added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

- I can be placed before V (5) and X (10) to make 4 and 9.
- X can be placed before L (50) and C (100) to make 40 and 90.
- C can be placed before D (500) and M (1000) to make 400 and 900.

Given an integer, convert it to a roman numeral.

Test Results

Case 1

num: 3

| Expected | Your Output |
|----------|-------------|
| "III" | none |

Case 2

num: 58

| Expected | Your Output |
|----------|-------------|
| "LVIII" | none |

Case 3

num: 1994

| Expected |
|-----------|
| "MCMXCIV" |

| Your Output |
|-------------|
| none |

0013. Roman to Integer

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

| Symbol | Value |
|--------|-------|
| I | 1 |
| V | 5 |
| X | 10 |
| L | 50 |
| C | 100 |
| D | 500 |
| M | 1000 |

For example, 2 is written as II in Roman numeral, just two one's added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

- I can be placed before V (5) and X (10) to make 4 and 9.
- X can be placed before L (50) and C (100) to make 40 and 90.
- C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Test Results

Case 1

s: "III"

| Expected | Your Output |
|----------|-------------|
| 3 | none |

Case 2

s: "LVIII"

| Expected | Your Output |
|----------|-------------|
| 58 | none |

Case 3

s: "MCMXCIV"

| Expected |
|----------|
| 1994 |

| Your Output |
|-------------|
| none |

0014. Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

Test Results

Case 1

strs: ["flower", "flow", "flight"]

| Expected |
|----------|
| "fl" |

| Your Output |
|-------------|
| none |

Case 2

strs: ["dog", "racecar", "car"]

| Expected |
|----------|
| "" |

| Your Output |
|-------------|
| none |

0015. 3Sum

Given an integer array `nums`, return all the triplets `[nums[i], nums[j], nums[k]]` such that $i \neq j$, $i \neq k$, and $j \neq k$, and $nums[i] + nums[j] + nums[k] == 0$.

Notice that the solution set must not contain duplicate triplets.

Test Results

Case 1

nums: [-1, 0, 1, 2, -1, -4]

| Expected | Your Output |
|--|-------------------|
| <code>[[-1, -1, 2], [-1, 0, 1]]</code> | <code>none</code> |

Case 2

nums: [0, 1, 1]

| Expected | Your Output |
|-----------------|-------------------|
| <code>[]</code> | <code>none</code> |

Case 3

nums: [0, 0, 0]

| Expected | Your Output |
|--------------------------|-------------------|
| <code>[[0, 0, 0]]</code> | <code>none</code> |

Case 4

nums: [-10, -7, -4, -1, 2, 5, 8, 11, 14, 17]

| Expected | Your Output |
|---|-------------------|
| <code>[[-10, -7, 17], [-10, -4, 14], [-10, -1, 11], [-10, 2, 8], [-7, -4, 11], [-7, -1, 8], [-7, 2, 5], [-4, -1, 5]]</code> | <code>none</code> |

Case 5

nums: [-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

| Expected | Your Output |
|--|-------------------|
| <code>[[-10, 1, 9], [-10, 2, 8], [-10, 3, 7], [-10, 4, 6], [-9, 0, 9], [-9, 1, 8], [-9, 2, 7], [-9, 3, 6], [-9, 4, 5], [-8, -1, 9], [-8, 0, 8], [-8, 1, 7], [-8, 2, 6], [-8, 3, 5],</code> | <code>none</code> |

$[-7, -2, 9], [-7, -1, 8], [-7, 0, 7], [-7, 1, 6], [-7, 2, 5], [-7, 3, 4], [-6, -3, 9], [-6, -2, 8], [-6, -1, 7], [-6, 0, 6], [-6, 1, 5], [-6, 2, 4], [-5, -4, 9], [-5, -3, 8], [-5, -2, 7], [-5, -1, 6], [-5, 0, 5], [-5, 1, 4], [-5, 2, 3], [-4, -3, 7], [-4, -2, 6], [-4, -1, 5], [-4, 0, 4], [-4, 1, 3], [-3, -2, 5], [-3, -1, 4], [-3, 0, 3], [-3, 1, 2], [-2, -1, 3], [-2, 0, 2], [-1, 0, 1]]$



0016. 3Sum Closest

Given an integer array `nums` of length `n` and an integer `target`, find three integers in `nums` such that the sum is closest to `target`.

Return the sum of the three integers.

You may assume that each input would have exactly one solution.

Test Results

Case 1

nums: [-1, 2, 1, -4]

target: 1

| Expected |
|----------|
| 2 |

| Your Output |
|-------------|
| none |

Case 2

nums: [0, 0, 0]

target: 1

| Expected |
|----------|
| 0 |

| Your Output |
|-------------|
| none |

Case 3

nums: [0, 1, 1]

target: 2

| Expected |
|----------|
| 2 |

| Your Output |
|-------------|
| none |

Case 4

nums: [-10, -7, -4, -1, 2, 5, 8, 11, 14, 17]

target: 20

| Expected |
|----------|
| 21 |

| Your Output |
|-------------|
| none |

Case 5

nums: [-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

target: 30

| Expected |
|----------|
| 24 |

| Your Output |
|-------------|
| none |

0017. Letter Combinations of a Phone Number

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.



Test Results

Case 1

digits: "23"

| Expected | Your Output |
|--|-------------|
| ["ad", "bd", "cd", "ae", "be", "ce", "af", "bf", "cf"] | none |

Case 2

digits: ""

| Expected | Your Output |
|----------|-------------|
| [] | none |

Case 3

digits: "2"

| Expected | Your Output |
|----------|-------------|
|----------|-------------|

["a", "b", "c"]

none

0018. 4Sum

Given an array `nums` of `n` integers, return an array of all the unique quadruplets `[nums[a], nums[b], nums[c], nums[d]]` such that:

- $0 \leq a, b, c, d < n$
- `a, b, c,` and `d` are **distinct**.
- `nums[a] + nums[b] + nums[c] + nums[d] == target`

You may return the answer in **any order**.

Test Results

Case 1

nums: `[1, 0, -1, 0, -2, 2]`

target: `0`

| Expected | Your Output |
|---|-------------------|
| <code>[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]</code> | <code>none</code> |

Case 2

nums: `[2, 2, 2, 2]`

target: `8`

| Expected | Your Output |
|-----------------------------|-------------------|
| <code>[[2, 2, 2, 2]]</code> | <code>none</code> |

Case 3

nums: `[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]`

target: `3`

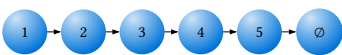
| Expected | Your Output |
|---|-------------------|
| <code>[[-5, 1, 3, 4], [-4, 0, 3, 4], [-4, 1, 2, 4], [-3, -1, 3, 4], [-3, 0, 2, 4], [-3, 1, 2, 3], [-2, -1, 2, 4], [-2, 0, 1, 4], [-2, 0, 2, 3], [-1, 0, 1, 3]]</code> | <code>none</code> |

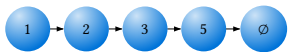
0019. Remove Nth Node From End of List

Given the head of a linked list, remove the nth node from the end of the list and return its head.

Test Results

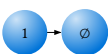
Case 1


head: 
n: 2

| Expected |
|---|
|  |

| Your Output |
|-------------|
| none |

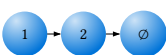
Case 2

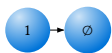
head: 
n: 1

| Expected |
|---|
|  |

| Your Output |
|-------------|
| none |

Case 3

head: 
n: 1

| Expected |
|---|
|  |

| Your Output |
|-------------|
| none |

0020. Valid Parentheses

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

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
 2. Open brackets must be closed in the correct order.
 3. Every close bracket has a corresponding open bracket of the same type.
- *s* consists of parentheses only '()[]{}'.

Test Results

Case 1

s: "()"

| Expected |
|----------|
| true |

| Your Output |
|-------------|
| none |

Case 2

s: "()[{}]"

| Expected |
|----------|
| true |

| Your Output |
|-------------|
| none |

Case 3

s: "[]"

| Expected |
|----------|
| false |

| Your Output |
|-------------|
| none |

Case 4

s: "([)]"

| Expected |
|----------|
| true |

| Your Output |
|-------------|
| none |

Case 5

s: "([)]"

| Expected |
|----------|
|----------|

| Your Output |
|-------------|
|-------------|

false

none

0021. Merge Two Sorted Lists

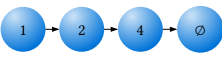
You are given the heads of two sorted linked lists `list1` and `list2`.

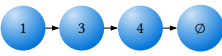
Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.

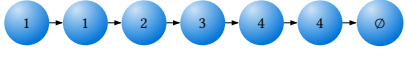
Return the head of the merged linked list.

Test Results

Case 1

list1: 

list2: 


| Expected |
|---|
|  |

| Your Output |
|-------------|
| none |

Case 2

list1: 

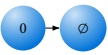
list2: 

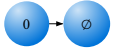
| Expected |
|---|
|  |

| Your Output |
|-------------|
| none |

Case 3

list1: 

list2: 

| Expected |
|---|
|  |

| Your Output |
|-------------|
| none |

0022. Generate Parentheses

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

Test Results

Case 1

n: 1

| Expected |
|----------|
| ["()"] |

| Your Output |
|-------------|
| none |

Case 2

n: 3

| Expected |
|---|
| ["((())", "(())", "()()", "()(())", "()()()"] |

| Your Output |
|-------------|
| none |

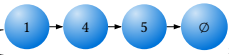
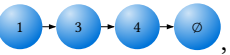
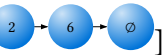
0023. Merge k Sorted Lists

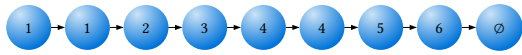
You are given an array of k linked-lists `lists`, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Test Results


Case 1

`lists:` [, , ]

| Expected | Your Output |
|---|-------------|
|  | none |


Case 2

`lists:` []

| Expected | Your Output |
|---|-------------|
|  | none |

Case 3

`lists:` []

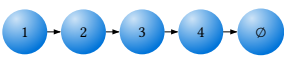
| Expected | Your Output |
|---|-------------|
|  | none |


0024. Swap Nodes in Pairs

Given a linked list, swap every two adjacent nodes and return its head.

Test Results

Case 1


head: 

| Expected |
|---|
|  |

| Your Output |
|-------------|
| none |

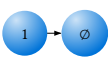
Case 2

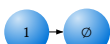
head: 

| Expected |
|---|
|  |

| Your Output |
|-------------|
| none |

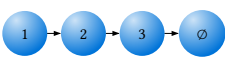
Case 3

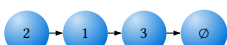
head: 

| Expected |
|---|
|  |

| Your Output |
|-------------|
| none |

Case 4

head: 

| Expected |
|---|
|  |

| Your Output |
|-------------|
| none |

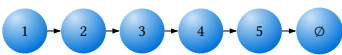
0025. Reverse Nodes in k-Group

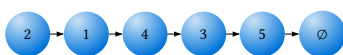
Given the head of a linked list, reverse the nodes of the list k at a time, and return the modified list.

k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes, in the end, should remain as it is.

Test Results

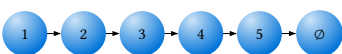
Case 1

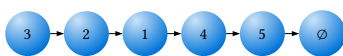
head: 
k: 2

| Expected |
|---|
|  |

| Your Output |
|-------------|
| none |

Case 2

head: 
k: 3

| Expected |
|---|
|  |

| Your Output |
|-------------|
| none |

0026. Remove Duplicates from Sorted Array (Adapted)

- This problem is different from the original version since functions in Typst cannot modify the input array in place.

Given an integer array `nums` sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same.

After removing duplicates, return the unique elements in ascending order.

Test Results

Case 1

nums: [1, 1, 2]

| Expected |
|----------|
| [1, 2] |

| Your Output |
|-------------|
| none |

Case 2

nums: [0, 0, 1, 1, 1, 2, 2, 3, 3, 4]

| Expected |
|-----------------|
| [0, 1, 2, 3, 4] |

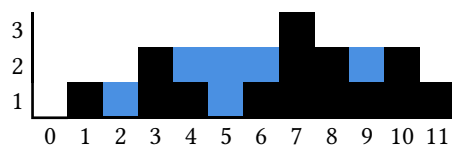
| Your Output |
|-------------|
| none |

0042. Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

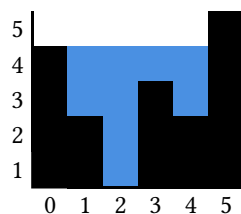
Example 1

height: [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]



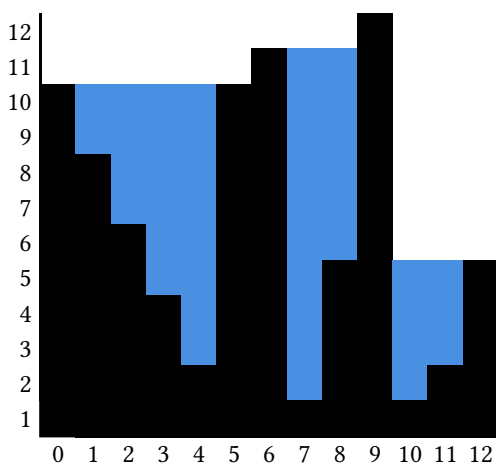
Example 2

height: [4, 2, 0, 3, 2, 5]



Example 3

height: [10, 8, 6, 4, 2, 10, 11, 1, 5, 12, 1, 2, 5]



Test Results

Case 1

height: [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]

| Expected |
|----------|
| 6 |

| Your Output |
|-------------|
| none |

Case 2

height: [4, 2, 0, 3, 2, 5]

| Expected |
|----------|
| 9 |

| Your Output |
|-------------|
| none |

Case 3

height: [10, 8, 6, 4, 2, 10, 11, 1, 5, 12, 1, 2, 5]

| Expected |
|----------|
| 43 |

| Your Output |
|-------------|
| none |

0051. N-Queens

The **n-queens** puzzle is the problem of placing n queens on an $n \times n$ chessboard such that no two queens attack each other.

Given an integer n , return *all distinct solutions* to the **n-queens puzzle**. You may return the answer in **any order**.

Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space, respectively.

Test Results

Case 1

n: 1

| Expected | Your Output |
|--------------|-------------|
| <div>Q</div> | none |

Case 2

n: 2

| Expected | Your Output |
|----------|-------------|
| [] | none |

Case 3

n: 4

| Expected | Your Output | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|-------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|------|
| <table><tr><td>.</td><td>Q</td><td>.</td><td>.</td></tr><tr><td>.</td><td>.</td><td>.</td><td>Q</td></tr><tr><td>Q</td><td>.</td><td>.</td><td>.</td></tr><tr><td>.</td><td>.</td><td>Q</td><td>.</td></tr></table> <hr/> <table><tr><td>.</td><td>.</td><td>Q</td><td>.</td></tr><tr><td>Q</td><td>.</td><td>.</td><td>.</td></tr><tr><td>.</td><td>.</td><td>.</td><td>Q</td></tr><tr><td>.</td><td>Q</td><td>.</td><td>.</td></tr></table> | . | Q | . | . | . | . | . | Q | Q | . | . | . | . | . | Q | . | . | . | Q | . | Q | . | . | . | . | . | . | Q | . | Q | . | . | none |
| . | Q | . | . | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| . | . | . | Q | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Q | . | . | . | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| . | . | Q | . | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| . | . | Q | . | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Q | . | . | . | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| . | . | . | Q | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| . | Q | . | . | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

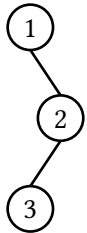
0094. Binary Tree Inorder Traversal

Given the root of a binary tree, return the inorder traversal of its nodes' values.

Test Results

Case 1

root:

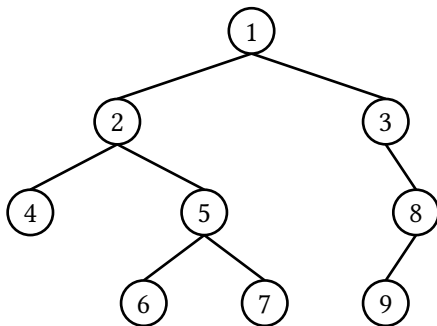


| Expected |
|-----------|
| [1, 3, 2] |

| Your Output |
|-------------|
| none |

Case 2

root:



| Expected |
|-----------------------------|
| [4, 2, 6, 5, 7, 1, 3, 9, 8] |

| Your Output |
|-------------|
| none |

Case 3

root:



| Expected |
|----------|
| [] |

| Your Output |
|-------------|
| none |

Case 4

root:

①

| Expected |
|----------|
| [1] |

| Your Output |
|-------------|
| none |

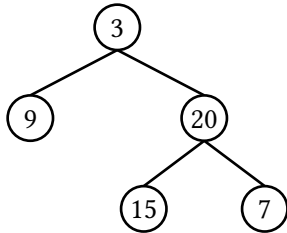
0110. Balanced Binary Tree

Given a binary tree, determine if it is **height-balanced**¹.

Test Results

Case 1

root:

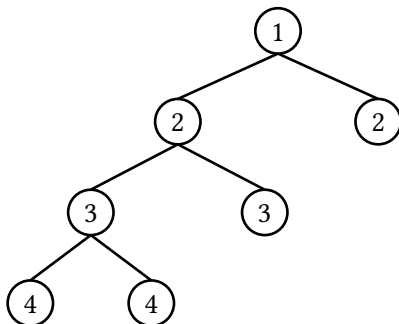


| Expected |
|----------|
| true |

| Your Output |
|-------------|
| none |

Case 2

root:



| Expected |
|----------|
| false |

| Your Output |
|-------------|
| none |

Case 3

root:



¹A **height-balanced** binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.

| Expected |
|----------|
| true |

| Your Output |
|-------------|
| none |

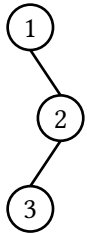
0144. Binary Tree Preorder Traversal

Given the root of a binary tree, return the preorder traversal of its nodes' values.

Test Results

Case 1

root:

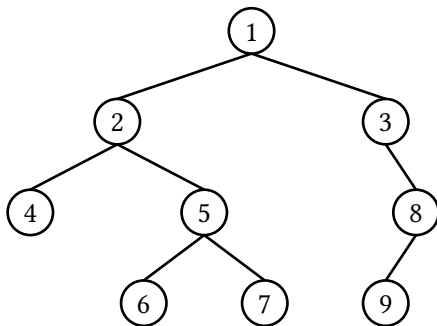


| Expected |
|-----------|
| [1, 2, 3] |

| Your Output |
|-------------|
| none |

Case 2

root:



| Expected |
|-----------------------------|
| [1, 2, 4, 5, 6, 7, 3, 8, 9] |

| Your Output |
|-------------|
| none |

Case 3

root:



| Expected |
|----------|
| [] |

| Your Output |
|-------------|
| none |

Case 4

root:

①

| Expected |
|----------|
| [1] |

| Your Output |
|-------------|
| none |

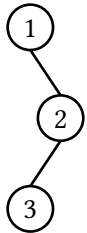
0145. Binary Tree Postorder Traversal

Given the root of a binary tree, return the postorder traversal of its nodes' values.

Test Results

Case 1

root:

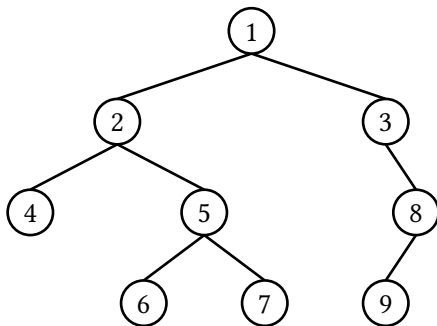


| Expected |
|-----------|
| [3, 2, 1] |

| Your Output |
|-------------|
| none |

Case 2

root:



| Expected |
|-----------------------------|
| [4, 6, 7, 5, 2, 9, 8, 3, 1] |

| Your Output |
|-------------|
| none |

Case 3

root:



| Expected |
|----------|
| [] |

| Your Output |
|-------------|
| none |

Case 4

root:

①

| Expected |
|----------|
| [1] |

| Your Output |
|-------------|
| none |