**Java exercises**

Contents

[ **References** 3](#_Toc146535794)

[ **Strings** 3](#_Toc146535795)

[ reverse 3](#_Toc146535796)

[ rotation 3](#_Toc146535797)

[ generate all rotations of a given string 3](#_Toc146535798)

[ sort 3](#_Toc146535799)

[ string segmentation 4](#_Toc146535800)

[ print all subsequences of a string 5](#_Toc146535801)

[ find all palindrome substrings 5](#_Toc146535802)

[ longest palindrome in a string 5](#_Toc146535803)

[ anagram 6](#_Toc146535804)

[ parenthesis checker 7](#_Toc146535805)

[ longest common prefix in an array 8](#_Toc146535806)

[ multiply two strings 9](#_Toc146535807)

[ edit distance 10](#_Toc146535808)

[ generate all binary strings from given pattern 10](#_Toc146535809)

[ **Arrays** 11](#_Toc146535810)

[ sum of two values 11](#_Toc146535811)

[ sum of two values (leetcode 1. Two Sum) 11](#_Toc146535812)

[ cloning arrays 12](#_Toc146535813)

[ reverse an array or string 12](#_Toc146535814)

[ rotation of array 12](#_Toc146535815)

[ search+insert+delete in an unsorted array 13](#_Toc146535816)

[ search+insert+delete in a sorted array 13](#_Toc146535817)

[ generate all subarrays 13](#_Toc146535818)

[ find the largest three distinct elements in an array 13](#_Toc146535819)

[ find second largest element in an array 13](#_Toc146535820)

[ move all zeroes to end of array 14](#_Toc146535821)

[ kth largest element in an array 14](#_Toc146535822)

[ kth largest element in a stream 15](#_Toc146535823)

[ Kth largest element in a stream (leetcode 703. Kth Largest Element in a Stream) 16](#_Toc146535824)

[ Others 17](#_Toc146535825)

[ **Linked List** 17](#_Toc146535826)

[ insert 17](#_Toc146535827)

[ delete 17](#_Toc146535828)

[ search 18](#_Toc146535829)

[ search: Program for Nth node from the end of a Linked List 18](#_Toc146535830)

[ find Length of a Linked List 18](#_Toc146535831)

[ reverse 19](#_Toc146535832)

[ delete Node with a given key 19](#_Toc146535833)

[ copy linked list with arbitrary pointer 20](#_Toc146535834)

[ **Stacks** 20](#_Toc146535835)

[ stack using singly linked list 21](#_Toc146535836)

[ stack in-built functions 21](#_Toc146535837)

[ **Queues** 21](#_Toc146535838)

[ single queue with array 21](#_Toc146535839)

[SingleQueueWithArray.java 21](#_Toc146535840)

[ **Tree** 21](#_Toc146535841)

[ mirror binary tree nodes 21](#_Toc146535842)

[ check if two binary trees are identical 21](#_Toc146535843)

[ **Graphs** 22](#_Toc146535844)

[ minimum spanning tree 22](#_Toc146535845)

[ **Sorting and searching** 23](#_Toc146535846)

[ find low/high index 23](#_Toc146535847)

[ merge overlapping intervals 23](#_Toc146535848)

[ **Dynamic programming** 24](#_Toc146535849)

[ largest sum subarray 24](#_Toc146535850)

[ **Math and stats** 25](#_Toc146535851)

[ is the number valid 25](#_Toc146535852)

[ **Backtracking** 26](#_Toc146535853)

[ all possible braces 26](#_Toc146535854)

[ **Design** 27](#_Toc146535855)

[ implement LRU cache 27](#_Toc146535856)

* **References**

Java Google Coding Interview questions

<https://www.codinginterview.com/google>

<https://www.codinginterview.com/google-interview-questions>

<https://leetcode.com/problemset/all/>

<https://www.geeksforgeeks.org/must-do-coding-questions-for-companies-like-amazon-microsoft-adobe/>

<https://www.geeksforgeeks.org/dsa-sheet-by-love-babbar/>

<https://www.geeksforgeeks.org/stuck-in-programming-get-the-solution-from-these-10-best-websites/>

<https://www.geeksforgeeks.org/top-15-websites-for-coding-challenges-and-competitions/>

<https://practice.geeksforgeeks.org/explore?page=1&sortBy=submissions>

[https://practice.geeksforgeeks.org/explore?page=1&company[]=Google&category[]=Strings&sortBy=submissions](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Google&category%5b%5d=Strings&sortBy=submissions)

<https://practice.geeksforgeeks.org/problem-of-the-day>

<https://www.educative.io/blog/google-coding-interview>

* **Strings**
* reverse

<https://www.geeksforgeeks.org/program-reverse-string-iterative-recursive/?ref=lbp>

* rotation

<https://www.geeksforgeeks.org/left-rotation-right-rotation-string-2/?ref=lbp>

* generate all rotations of a given string

<https://www.geeksforgeeks.org/generate-rotations-given-string/>

* sort

<https://www.geeksforgeeks.org/sort-string-characters/?ref=lbp>

* print the frequency of each character in alphabetical order

<https://www.geeksforgeeks.org/print-the-frequency-of-each-character-in-alphabetical-order/?ref=lbp>

* swap characters in a string

<https://www.geeksforgeeks.org/swap-characters-in-a-string/?ref=lbp>

* how to insert characters in a string at a certain position?

<https://www.geeksforgeeks.org/how-to-insert-characters-in-a-string-at-a-certain-position/?ref=lbp>

* program to check if two strings are same or not

<https://www.geeksforgeeks.org/program-to-check-if-two-strings-are-same-or-not/?ref=lbp>

* remove all occurrences of a character in a string

<https://www.geeksforgeeks.org/remove-all-occurrences-of-a-character-in-a-string/?ref=lbp>

* string segmentation, work break problem

Problem Statement

You are given a dictionary of words and a large input string. You have to find out whether the input string can be completely segmented into the words of a given dictionary. The following two examples elaborate on the problem further.

A close-up of a chart

Description automatically generated

Hint:

Recursion

Memoization

Companies: Google

ref:

139. Word Break

<https://leetcode.com/problems/word-break/description/>

Word Break | Dynamic Programming | Leetcode #139

<https://www.youtube.com/watch?v=th4OnoGasMU>

Solution:

* + StringSegmentation.java: Recursion
  + StringSegmentation2.java (better): Recursion

WB[]: each recursion

A white board with writing on it

Description automatically generated

For a string of length = 4 (WB[code]), it does 8 calculation of string (WB[string]).

For a string of length = 3 (WB[ode]), it does 4 calculation of string (WB[string]).

For a string of length = 2 (WB[de]), it does 2 calculation of string (WB[string]).

For a string of length = 1 (WB[e]), it does 1 calculation of string (WB[string]).

Then Runtime Complexity is: O(2n)

A white board with writing on it

Description automatically generated

The worst case is:

When each character of the string is presented in the dictionary and ended time for all partitions points. Each partitions points can have a partitions or not, then Time Complexity is O(2n).

* Use Set for fast lookup

In a recursion function first validate the best case, in this case when is empty string.

A person writing on a white board

Description automatically generated

* + StringSegmentation3.java (better): Memoization

Used for large inputs and dictionary sizes.

See imagen StringSegmentation2

WB[de] = appear many times, must apply memorization

A white board with writing on it

Description automatically generated

* + StringSegmentation4.java (much better): Tabulation Dynamic Programming

Used for repeating subproblems

A white board with writing on it

Description automatically generated

A person writing on a white board

Description automatically generated

A white board with writing on it

Description automatically generated

* + StringSegmentation5.java: Back Tracking

Word Break Problem using Backtracking

Given a valid sentence without any spaces between the words and a dictionary of valid English words, find all possible ways to break the sentence into individual dictionary words.

Example:

Consider the following dictionary

{ i, like, sam, sung, samsung, mobile, ice,

and, cream, icecream, man, go, mango}

Input: "ilikesamsungmobile"

Output: i like sam sung mobile

i like samsung mobilek8

Input: "ilikeicecreamandmango"

Output: i like ice cream and man go

i like ice cream and mango

i like icecream and man go

i like icecream and mango

The **Dynamic Programming** solution only finds whether it is **possible to break a word or not**. Here we need to print **all possible word breaks**.

* print all subsequences of a string

<https://www.geeksforgeeks.org/print-subsequences-string/>

* find all palindrome substrings

Problem Statement

Given a string find all **non-single letter** substrings that are palindromes. For instance:

Hint

Find substrings

A blue rectangular box with black text

Description automatically generated

Companies: Google

ref:

Can be use Manacher’s algorithm

* + FindAllPalindromeSubstrings.java
* count all palindrome sub-strings in a string

Given a string, the task is to count all palindrome substring in a given string. Length of palindrome substring is greater than or equal to 2.

Examples:

Input : str = "abaab"

Output: 3

Explanation :

All palindrome substring are :

"aba", "aa", "baab"

Input : str = "abbaeae"

Output: 4

Explanation :

All palindrome substring are :

"bb", "abba", "aea", "eae":

* + FindAllPalindromeSubstrings.java
  + CountPalindromeSubstrings.java
* find all distinct palindromic sub-strings of a given string

Given a string of lowercase ASCII characters, find all distinct continuous palindromic sub-strings of it.

Examples:

Input: str = "abaaa"

Output: Below are 5 palindrome sub-strings

a

aa

aaa

aba

b

Input: str = "geek"

Output: Below are 4 palindrome sub-strings

e

ee

g

k

* + FindAllDistinctPalindromeSubstrings.java
* palindrome in a string

<https://www.geeksforgeeks.org/find-number-distinct-palindromic-sub-strings-given-string/> (PENDING)

<https://www.geeksforgeeks.org/check-given-string-rotation-palindrome/> (PENDING)

* longest palindrome in a string

Given a string S, find the longest palindromic substring in S.**Substring of string S:** S[ i . . . . j ] where 0 ≤ i ≤ j < len(S)**. Palindrome string:** A string that reads the same backward. More formally, S is a palindrome if reverse(S) = S.**In case of conflict**, return the substring which occurs first ( with the least starting index).

**Example 1:**

**Input:**

S = "aaaabbaa"

**Output:** aabbaa

**Explanation**: The longest Palindromic

substring is "aabbaa".

**Example 2:**

**Input**:

S = "abc"

**Output:** a

**Explanation**: "a", "b" and "c" are the

longest palindromes with same length.

The result is the one with the least

starting index.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **longestPalin()**which takes the string S as input and returns the longest palindromic substring of S.

**Expected Time Complexity:**O(|S|2).  
**Expected Auxiliary Space:**O(1).

**Constraints:**  
1 ≤ |S| ≤ 103

Companies: Google

Refs:

LongestPalindromeString.java

[https://practice.geeksforgeeks.org/problems/longest-palindrome-in-a-string3411/1?page=1&company[]=Google&category[]=Strings&sortBy=submissions](https://practice.geeksforgeeks.org/problems/longest-palindrome-in-a-string3411/1?page=1&company%5b%5d=Google&category%5b%5d=Strings&sortBy=submissions)

better approach: use Manacher’s algorithm.

<https://www.geeksforgeeks.org/manachers-algorithm-linear-time-longest-palindromic-substring-part-1/>

Time Complexity: O(N), N length of the input string.

Space Complexity: O(N), N length of the modified input string.

* anagram

Given two strings **a**and **b**consisting of lowercase characters. The task is to check whether two given strings are an anagram of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different. For example, act and tac are an anagram of each other.

**Note:**

* If the strings are anagrams you have to **return True or else return False**
* **|s|**represents the length of string s.

**Example 1:**

**Input:**a = geeksforgeeks, b = forgeeksgeeks

**Output:** YES

**Explanation:** Both the string have same characters with

same frequency. So, both are anagrams.

**Example 2:**

**Input:**a = allergy, b = allergic

**Output:** NO

**Explanation:** Characters in both the strings are

  not same, so they are not anagrams.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **isAnagram()** which takes the string **a** and string **b** as input parameter and check if the two strings are an anagram of each other. The function returns true if the strings are anagram else it returns false.

**Expected Time Complexity:**O(|a|+|b|).  
**Expected Auxiliary Space:**O(Number of distinct characters).

**Constraints:**  
1 ≤ |a|,|b| ≤ 105

Companies: Google

Refs:

IsAnagram.java

Anagram2.java (better)

* parenthesis checker

Given an expression string **x**. Examine whether the pairs and the orders of {,},(,),[,] are correct in exp.  
For example, the function should return 'true' for exp = [()]{}{[()()]()} and 'false' for exp = [(]).

**Note:**The drive code prints "balanced" if function return true, otherwise it prints "not balanced".

**Example 1:**

**Input**:

{([])}

**Output**:

true

**Explanation**:

{ ( [ ] ) }. Same colored brackets can form

balanced pairs, with 0 number of

unbalanced bracket.

**Example 2:**

**Input**:

()

**Output**:

true

**Explanation**:

(). Same bracket can form balanced pairs,

and here only 1 type of bracket is

present and in balanced way.

**Example 3:**

**Input**:

([]

**Output**:

false

**Explanation**:

([]. Here square bracket is balanced but

the small bracket is not balanced and

Hence , the output will be unbalanced.

**Your Task:**  
This is a **function**problem. You only need to complete the function **ispar()**that takes a **string**as a **parameter**and returns a boolean value **true**if **brackets**are **balanced**else **returns false**. The **printing**is done **automatically**by the **driver code**.  
  
**Expected Time Complexity**: O(|x|)  
**Expected Auixilliary Space**: O(|x|)  
  
**Constraints:**  
1 ≤ |x| ≤ 32000

Companies: Google

Refs:

ParenthesisChecker.java

<https://www.geeksforgeeks.org/check-for-balanced-parentheses-in-an-expression/>

* longest common prefix in an array

Given an array of **N** strings, find the longest common prefix among all strings present in the array.

**Example 1:**

**Input:**

N = 4

arr[] = {geeksforgeeks, geeks, geek,

  geezer}

**Output:** gee

**Explanation**: "gee" is the longest common

prefix in all the given strings.

**Example 2:**

**Input**:

N = 2

arr[] = {hello, world}

**Output:** -1

**Explanation**: There's no common prefix

in the given strings.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **longestCommonPrefix()**which takes the string array **arr**[] and its size **N** as inputs and **returns** the longest common prefix common in all the strings in the array. If there's no prefix common in all the strings, return "-1".

**Expected Time Complexity:**O(N\*min(|arri|)).  
**Expected Auxiliary Space:**O(min(|arri|)) for result.

**Constraints:**  
1 ≤ N ≤ 103  
1 ≤ |arri| ≤ 103

Companies: Google

Refs:

LongestCommonPrefix.java

* multiply two strings

Given two numbers as strings s1 and s2. Calculate their Product.  
  
**Note:** The numbers can be negative andYou are not allowed to use any built-in function or convert the strings to integers. There can be zeros in the begining of the numbers. You don't need to specify '+' sign in the begining of positive numbers.

**Example 1:**

**Input:**

**s1 =** "0033"

**s2 =** "2"

**Output:**

66

**Example 2:**

**Input:**

**s1 =** "11"

**s2 =** "23"

**Output:**

253

**Your Task:**You don't need to read input or print anything. Your task is to complete the function **multiplyStrings()** which takes two strings s1 and s2 as input and returns their product as a string.  
  
**Expected Time Complexity:**O(n1\* n2)  
**Expected Auxiliary Space:**O(n1+ n2); where n1 and n2 are sizes of strings s1 and s2 respectively.

**Constraints:**  
1 ≤ length of s1 and s2 ≤ 103

Refs:

MultiplyTwoStrings.java

* edit distance

Given two strings **s** and **t.**Return the minimum number of operations required to convert **s**to **t**.  
The possible operations are permitted:

1. Insert a character at any position of the string.
2. Remove any character from the string.
3. Replace any character from the string with any other character.

**Example 1:**

**Input:**

s = "geek", t = "gesek"

**Output:** 1

**Explanation:** One operation is required

inserting 's' between two 'e's of s.

**Example 2:**

**Input :**

s = "gfg", t = "gfg"

**Output:**

0

**Explanation:** Both strings are same.

**Your Task:**  
You don't need to read or print anything. Your task is to complete the function **editDistance()**which takes strings s and t as input parameters and returns the minimum number of operation to convert the string **s** to string**t**.

**Expected Time Complexity:**O(|s|\*|t|)  
**Expected Space Complexity:**O(|s|\*|t|)

**Constraints:**  
1 ≤ Length of both strings ≤ 100  
Both the strings are in lowercase.

Refs:

EditDistance.java

* generate all binary strings from given pattern

Refs:

GenerateAllBinaryStringsGivenPattern.java

* **Arrays**
* sum of two values

Given an array of integers and a value, determine if there are any two integers in the array whose sum is equal to the given value. Return true if the sum exists and return false if it does not.

Example:

Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,

return [0, 1].

Companies: Google

Refs:

SumOfTwoValues.java

* sum of two values (leetcode 1. 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.

Example 1:

Input: nums = [2,7,11,15], target = 9

Output: [0,1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Example 2:

Input: nums = [3,2,4], target = 6

Output: [1,2]

Example 3:

Input: nums = [3,3], target = 6

Output: [0,1]

Constraints:

2 <= nums.length <= 104

-109 <= nums[i] <= 109

-109 <= target <= 109

Only one valid answer exists.

Companies: Google

Refs:

SumOfTwoValuesLeetcode1.java

* cloning arrays

Refs:

CloningArray.java

* reverse an array or string

Given an array (or string), the task is to reverse the array/string.

Examples :

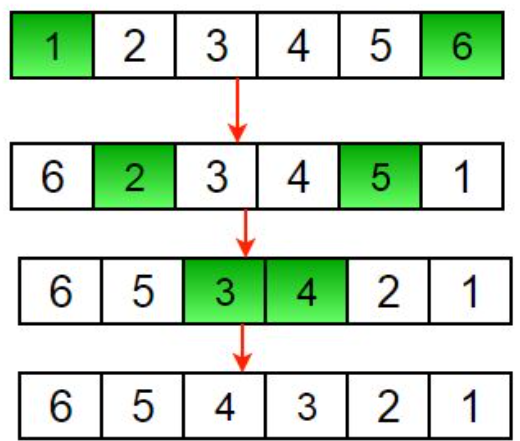
Input : arr[] = {1, 2, 3}

Output : arr[] = {3, 2, 1}

Input : arr[] = {4, 5, 1, 2}

Output : arr[] = {2, 1, 5, 4}

* iterative way:



* rotation of array

<https://www.geeksforgeeks.org/complete-guide-on-array-rotations/>

* Approach: The Reversal Algorithm: better solution

Original array:

A green rectangle with numbers

Description automatically generated

Separate it out into first ‘d’ elements and ‘n-d’ elements.

A diagram of a number

Description automatically generated

Reverse the first ‘d’ elements

A green rectangular object with numbers

Description automatically generated

Reverse last (N-d) elements

A green rectangular object with numbers

Description automatically generated

Reverse the whole array.

A green rectangular object with black text

Description automatically generated

* **Applications**:

**Searching** in a sorted and rotated array: binary search

**Game** development: game mechanics, rotating a game board, game objects.

**Audio** and **video** processing: effects such as pitch shifting, time stretching, stereo panning.

**Text** editing: undo and redo functionality.

* search+insert+delete in an unsorted array

Linear Search

* search+insert+delete in a sorted array

Binary Search

* binary search in sorted array

Refs:

BinarySearchInSortedArray.java

BinarySearchInSortedArray2.java

* generate all subarrays

Given an array, generate all the possible subarrays of the given array using recursion.

Examples:

Input : [1, 2, 3]

Output : [1], [1, 2], [2], [1, 2, 3], [2, 3], [3]

Input : [1, 2]

Output : [1], [1, 2], [2]

<https://www.geeksforgeeks.org/generating-subarrays-using-recursion/>

* find the largest three distinct elements in an array

Given an array with all distinct elements, find the largest three elements. Expected time complexity is O(n) and extra space is O(1).

Examples :

Input: arr[] = {10, 4, 3, 50, 23, 90}

Output: 90, 50, 23

* find second largest element in an array

Given an array of integers, our task is to write a program that efficiently finds the second-largest element present in the array.

Example:

Input: arr[] = {12, 35, 1, 10, 34, 1}

Output: The second largest element is 34.

Explanation: The largest element of the array is 35 and the second largest element is 34

Input: arr[] = {10, 5, 10}

Output: The second largest element is 5.

Explanation: The largest element of the array is 10 and the second largest element is 5

Input: arr[] = {10, 10, 10}

Output: The second largest does not exist.

Explanation: Largest element of the array is 10 there is no second largest element

* move all zeroes to end of array

Given an array of random numbers, Push all the zero’s of a given array to the end of the array. For example, if the given arrays is {1, 9, 8, 4, 0, 0, 2, 7, 0, 6, 0}, it should be changed to {1, 9, 8, 4, 2, 7, 6, 0, 0, 0, 0}. The order of all other elements should be same. Expected time complexity is O(n) and extra space is O(1).

Example:

Input : arr[] = {1, 2, 0, 4, 3, 0, 5, 0};

Output : arr[] = {1, 2, 4, 3, 5, 0, 0, 0};

Input : arr[] = {1, 2, 0, 0, 0, 3, 6};

Output : arr[] = {1, 2, 3, 6, 0, 0, 0};

* kth largest element in an array

Given an integer k and an array of size n consisting of unique integers. Find the kth largest element in **this** array.

Examples

Input : arr : [2,1,4,6,3,9,7], k : 2

Output : 7

Input : arr : [12,65,34,90], k : 4

Output : 12

Ref:

KthLargestElement1.java

KthLargestElement2.java

KthLargestElement3.java

KthLargestElement4.java (better)

* kth largest element in a stream

Given an input stream **arr[]**of **n** integers. Find the **K**th largest element (not **K**th largest unique element) after insertion of each element in the stream and if the **Kth** largest element doesn't exist, the answer will be -1 for that insertion.  return a list of size n after all insertions.

**Example 1:**

**Input**:

k = 4, n = 6

arr[] = {1, 2, 3, 4, 5, 6}

**Output**:

-1 -1 -1 1 2 3

**Explanation**:

k = 4

For 1, the 4th largest element doesn't

exist so answer will be -1.

For 2, the 4th largest element doesn't

exist so answer will be -1.

For 3, the 4th largest element doesn't

exist so answer will be -1.

For 4, the 4th largest element is 1.

For 5, the 4th largest element is 2.

For 6, the 4th largest element is 3.

**Example 2:**

**Input**:

k = 1, n = 2

arr[] = {3, 4}

**Output**:

3 4

**Explanation**:

For the 1st and 2nd element the 1st largest

element is itself.

Companies: Google

Refs:

KthLargestElementStream.java

<https://practice.geeksforgeeks.org/problems/kth-largest-element-in-a-stream2220/1>

Kth Largest Element in a Stream | GeeksforGeeks Java

<https://www.youtube.com/watch?v=qk6jJzSPCxk>

* Kth largest element in a stream (leetcode 703. Kth Largest Element in a Stream)

Find the Kth largest element in a stream of numbers. The **class** should have the following two things:​

1. The constructor of the **class** should accept an integer array containing initial numbers from the stream and an integer K.
2. The **class** should expose a function add(**int** num) which will store the given number and **return** the Kth largest number.

Design a class to find the kth largest element in a stream. Note that it is the kth largest element in the sorted order, not the kth distinct element.

Implement KthLargest class:

KthLargest(int k, int[] nums) Initializes the object with the integer k and the stream of integers nums.

int add(int val) Appends the integer val to the stream and returns the element representing the kth largest element in the stream.

**Example 1:**

**Input**: ["KthLargest", "add", "add", "add", "add", "add"]

[[3, [4, 5, 8, 2]], [3], [5], [10], [9], [4]]

**Output**: [null, 4, 5, 5, 8, 8]

**Explanation**:

KthLargest kthLargest = new KthLargest(3, [4, 5, 8, 2]);

kthLargest.add(3); // return 4

kthLargest.add(5); // return 5

kthLargest.add(10); // return 5

kthLargest.add(9); // return 8

kthLargest.add(4); // return 8

**Constraints**:

1 <= k <= 104

0 <= nums.length <= 104

-104 <= nums[i] <= 104

-104 <= val <= 104

At most 104 calls will be made to add.

It is guaranteed that there will be at least k elements in the array when you search for the kth element.

Companies: Google

Refs:

KthLargestElementStreamLeetcode703.java

703. Kth Largest Element in a Stream

<https://leetcode.com/problems/kth-largest-element-in-a-stream/>

https://www.educative.io/blog/google-coding-interview-questions?eid=5082902844932096

* Others

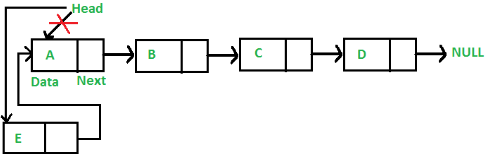
Segregate even and odd numbers using Lomuto’s Partition Scheme

<https://www.geeksforgeeks.org/segregate-even-odd-numbers-set-3/>

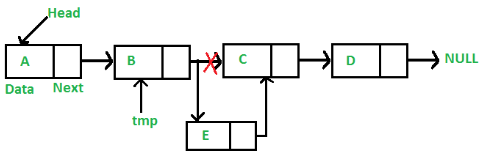
Print left rotation of array in O(n) time and O(1) space

<https://www.geeksforgeeks.org/print-left-rotation-array/>

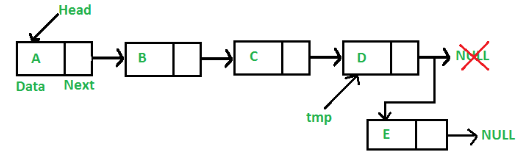
* **Linked List**
* insert
* Insert a Node at the Front/Beginning of Linked List:



* Insert a Node after a given Node in Linked List:



* Insert a Node at the end of Linked List:



* delete
* Delete a Node recursively at the Beginning/End/Middle of Linked List:

A diagram of a computer code

Description automatically generated

* Delete a Node at a given Node in Linked List
* Delete a Node at the end of Linked List
* Delete the entire Linked List
* search
* Search element, return true if exists, else returns false.
* Search element by index.

Example:

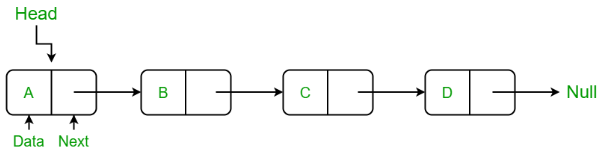
Input: 1->10->30->14, index = 2

Output: 30

The node at index 2 is 30

* search: Program for Nth node from the end of a Linked List

Given a Linked List and a number **N**, write a function that returns the value at the Nth node from the end of the Linked List.



Examples:

Input: 1 -> 2 -> 3 -> 4, N = 3

Output: 2

Input: 35 -> 15 -> 4 -> 20, N = 4

Output: 35

Refs:

SearchNthNodeFromEnd.java

* find Length of a Linked List

Uses **tail recursion** instead of **regular recursion**.

Refs:

SinglyLinkedList.java

* reverse

Input: Head of following linked list

1->2->3->4->NULL

Output: Linked list should be changed to,

4->3->2->1->NULL

Input: Head of following linked list

1->2->3->4->5->NULL

Output: Linked list should be changed to,

5->4->3->2->1->NULL

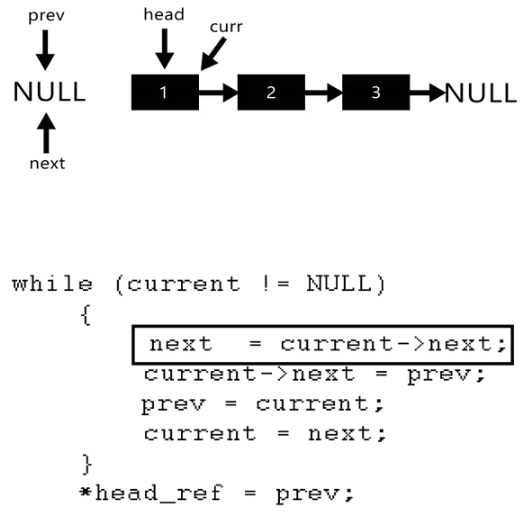
Input: NULL

Output: NULL

Input: 1->NULL

Output: 1->NULL

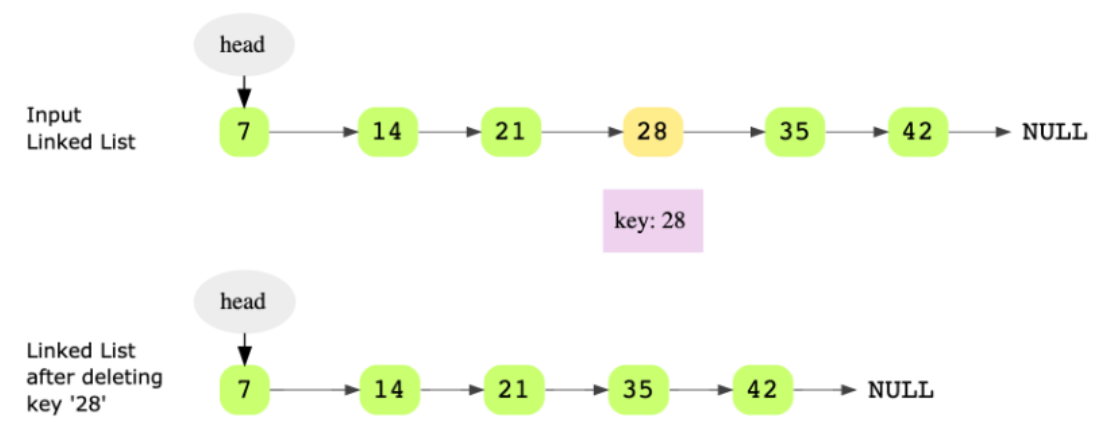
* reverse a Linked List by Iterative method:



* Other methods: using Recursion, Tail Recursive, Stack.
* delete Node with a given key

**Problem Statement**

We are given the head of a linked list and a key. We have to delete the node that contains this given key. The following two examples elaborate on this problem further.

****

Hint: Keep track of previous pointer

Companies: Google

Refs:

DeleteNodeWithGivenKey.java

<https://leetcode.com/problems/delete-node-in-a-linked-list/description/>

<https://leetcode.com/problems/remove-linked-list-elements/>

<https://practice.geeksforgeeks.org/problems/delete-a-node-in-single-linked-list/1>

* copy linked list with arbitrary pointer

**Problem Statement**

You are given a linked list where the node has two pointers. The first is the regular **next** pointer. The second pointer is called **arbitrary\_pointer** and it can point to any node in the linked list. Your job is to write code to make a deep copy of the given linked list. Here, **deep copy means that any operations on the original list (inserting, modifying and removing) should not affect the copied list**.

Here’s an example of a linked list with arbitrary pointers connected.

A diagram of a number

Description automatically generated

Hint: Hash table

Companies: Google

Refs:

CopyLinkedListWithArbitraryPointer.java

CopyLinkedListWithArbitraryPointer2.java (better)

<https://www.educative.io/copy-linked-list-with-arbitrary-pointer> <https://www.youtube.com/watch?v=OvpKeraoxW0>

* **Stacks**
* stack using singly linked list

<https://www.geeksforgeeks.org/implement-a-stack-using-singly-linked-list/?ref=lbp>

* stack in-built functions
* **Queues**
* single queue with array

SingleQueueWithArray.java

* single queue with linked list

SingleQueueWithLinkedList.java

* **Tree**
* mirror binary tree nodes

<https://www.educative.io/mirror-binary-tree-nodes>

Companies: Google

Refs:

MirrorBinaryTreeNodes.java (better)

MirrorBinaryTreeNodes2.java

<https://www.youtube.com/watch?v=FtdyIHBaKjc>

* check if two binary trees are identical

Problem Statement

Given the roots of two binary trees, determine if these trees are identical or not. Identical trees have the same layout and data at each node. Consider the following two identical binary trees that have the same layout and data.

A diagram of a number

Description automatically generated with medium confidence

It is not necessary that trees that have the same data are identical trees. Trees that have the exact same data may not be structurally identical. For example if you look at the two trees below, although they have the same data, they are not identical.

A diagram of numbers and a number

Description automatically generated with medium confidence

Hint

Depth first traversal

Recursion

Companies: Google

Refs:

CheckTwoBinaryTreesIdentical.java

* **Graphs**
* minimum spanning tree

Problem Statement:

Find the minimum spanning tree of a connected, undirected graph with weighted edges.

Consider the following graph.

A diagram of a triangle with blue circles and black lines

Description automatically generated

The minimum spanning tree of the above graph would be:

A diagram of a triangle with circles and lines

Description automatically generated

Hint:

Minimum weight edge

Companies: Google

Refs:

MinimumSpanningTree.java

* **Sorting and searching**
* find low/high index

Problem Statement:

Given a sorted array of integers, return the low and high index of the given key. You must return -1 if the indexes are not found.

The array length can be in the millions with many duplicates.

In the following example, according to the the key, the low and high indices would be:

key: 1, low = 0 and high = 0

key: 2, low = 1 and high = 1

key: 5, low = 2 and high = 9

key: 20, low = 10 and high = 10

Hint:

Binary Search

A number in a box

Description automatically generated with medium confidence

Companies: Google

Refs:

FindLowHighIndex.java

* merge overlapping intervals

Problem Statement:

You are given an array (list) of interval pairs as input where each interval has a start and end timestamp. The input array is sorted by starting timestamps. You are required to merge overlapping intervals and return a new output array.

Consider the input array below. Intervals (1, 5), (3, 7), (4, 6), (6, 8) are overlapping so they should be merged to one big interval (1, 8). Similarly, intervals (10, 12) and (12, 15) are also overlapping and should be merged to (10, 15).

A green and blue rectangles with black numbers

Description automatically generated

Hint:

Try the linear scan

Use the pair class defined in the code below to handle pairs of time stamps

Companies: Google

Refs:

MergeOverlapingIntervals.java

* sorting

<https://www.geeksforgeeks.org/sorting-algorithms/>

* **Dynamic programming**
* largest sum subarray

Problem Statement:

In the array below, the largest sum subarray starts at index 3 and ends at 6, and with the largest sum being 12.



Hint:

Use Kadane’s algorithm

Solution Explanation:

Runtime complexity: The runtime complexity of this solution is linear, O(n).

Memory complexity: The memory complexity of this solution is constant, O(1).

The basic idea of **Kadane’s algorithm** is to scan the entire array and at each position find the maximum sum of the subarray ending there. This is achieved by keeping a current\_max for the current array index and a global\_max. The algorithm is as follows:

current\_max = A[0]

global\_max = A[0]

for i = 1 -> size of A

if current\_max is less than 0

then current\_max = A[i]

otherwise

current\_max = current\_max + A[i]

if global\_max is less than current\_max

then global\_max = current\_max

Companies: Google

Refs:

LargestSumSubarray.java

LargestSumSubarray2.java (better)

* **Math and stats**
* is the number valid

Problem Statement:

Given an input string, determine if it makes a valid number or not. For simplicity, assume that white spaces are not present in the input.

4.325 is a valid number.

1.1.1 is NOT a valid number.

222 is a valid number.

22. is NOT a valid number.

0.1 is a valid number.

22.22. is NOT a valid number.

1. is NOT a valid number.

Hint:

Use **state machine**

Solution Breakdown:

We’ll use the state machine below to check if a number is valid or not. The initial state is ‘Start’. We’ll process each character to identify the next state. The input string is not a valid number if we reach an UNKNOWN state at any point or if it ends in a DECIMAL.

A diagram of a mathematical function

Description automatically generated

We are not looking at the sign (+ or -) at the start of a number in the state machine. If there is a sign at the start of the input string, we start processing the string for the state machine after that sign character.

Companies: Google

Refs:

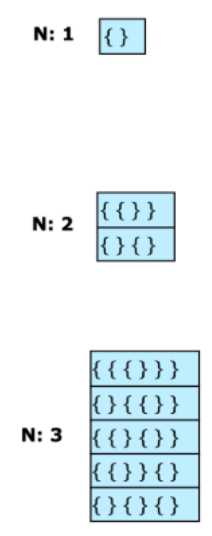
IsNumberValid.java

<https://www.geeksforgeeks.org/introduction-of-algorithmic-state-machines-asms/>

<https://www.geeksforgeeks.org/introduction-of-finite-automata/>

* **Backtracking**
* all possible braces

Problem Statement: Print all braces combinations for a given value n so that they are balanced. Here are a few examples:



Hint: Recursion

Solution Breakdown:

The solution is to maintain counts of left\_braces and right\_braces. The basic algorithm is as follows:​

left\_braces count: 0

right\_braces count: 0

if left\_braces count is less than n:

add left\_braces and recurse further

if right\_braces count is less than left\_braces count:

add right\_braces and recurse further

stop recursing when left\_braces and right\_braces counts are both equal to n

For n=2: {{}}

{}{}

A screenshot of a computer

Description automatically generated

For n=3:

A diagram of a structure

Description automatically generated

Companies: Google

Refs:

PrintAllBalancedBraceCombinations.java

PrintAllBalancedBraceCombinations2.java (better)

LeetCode 22. Generate Parentheses | Generate All Balanced Parentheses | Parentheses Permutations (explain better)

<https://www.youtube.com/watch?v=FrTsckBmdyM>

<https://www.youtube.com/watch?v=_L3dvtG8Jxw> (explain hindi)

<https://www.youtube.com/watch?v=gBC_Fd8EE8A>

<https://www.geeksforgeeks.org/introduction-to-backtracking-data-structure-and-algorithm-tutorials/>

* **Design**
* implement LRU cache

Problem Statement:

Least Recently Used (LRU) is a common caching strategy. It defines the policy to evict elements from the cache to make room for new elements when the cache is full, meaning it discards the least recently used items first.

Let’s take an example of a cache that has a capacity of 4 elements. We cache elements 1, 2, 3 and 4.

A green square with black numbers

Description automatically generated

The diagram above represents the cache state after first access of all four elements. We now need to cache another element “5”.

A green square with black numbers

Description automatically generated

In LRU cache, we evict the least recently used element (in this case “1”) in case a new element needs to be cached. Now “2” is next in line to be evicted if a new element needs to be cached. Let’s see what happens when “2” is accessed again.

A green square with black numbers

Description automatically generated

Now “3” becomes the next in line to be evicted from the cache.

Hint:

Doubly linked list

Hashing

Think about evictions

Companies: Google

Refs:

LRUCache.java