



# Algorithm

# Table of Contents

Preface	0
FAQ	1
Guidelines for Contributing	1.1
Contributors	1.2
Part I - Basics	2
Basics Data Structure	3
String	3.1
Linked List	3.2
Binary Tree	3.3
Huffman Compression	3.4
Queue	3.5
Heap	3.6
Stack	3.7
Set	3.8
Map	3.9
Graph	3.10
Basics Sorting	4
Bubble Sort	4.1
Selection Sort	4.2
Insertion Sort	4.3
Merge Sort	4.4
Quick Sort	4.5
Heap Sort	4.6
Bucket Sort	4.7
Counting Sort	4.8
Radix Sort	4.9
Basics Algorithm	5
Divide and Conquer	5.1
Binary Search	5.2
Math	5.3
Greatest Common Divisor	5.3.1
Prime	5.3.2
Knapsack	5.4
Counting Problem	5.5
Probability	5.6
Shuffle	5.6.1
Bitmap	5.7
Basics Misc	6
Bit Manipulation	6.1

Part II - Coding	7
String	8
strStr	8.1
Two Strings Are Anagrams	8.2
Compare Strings	8.3
Anagrams	8.4
Longest Common Substring	8.5
Rotate String	8.6
Reverse Words in a String	8.7
Valid Palindrome	8.8
Longest Palindromic Substring	8.9
Space Replacement	8.10
Wildcard Matching	8.11
Length of Last Word	8.12
Count and Say	8.13
Integer Array	9
Remove Element	9.1
Zero Sum Subarray	9.2
Subarray Sum K	9.3
Subarray Sum Closest	9.4
Recover Rotated Sorted Array	9.5
Product of Array Exclude Itself	9.6
Partition Array	9.7
First Missing Positive	9.8
2 Sum	9.9
3 Sum	9.10
3 Sum Closest	9.11
Remove Duplicates from Sorted Array	9.12
Remove Duplicates from Sorted Array II	9.13
Merge Sorted Array	9.14
Merge Sorted Array II	9.15
Median	9.16
Partition Array by Odd and Even	9.17
Kth Largest Element	9.18
Binary Search	10
Binary Search	10.1
Search Insert Position	10.2
Search for a Range	10.3
First Bad Version	10.4
Search a 2D Matrix	10.5
Search a 2D Matrix II	10.6
Find Peak Element	10.7

Search in Rotated Sorted Array	10.8
Search in Rotated Sorted Array II	10.9
Find Minimum in Rotated Sorted Array	10.10
Find Minimum in Rotated Sorted Array II	10.11
Median of two Sorted Arrays	10.12
Sqrt x	10.13
Wood Cut	10.14
Math and Bit Manipulation	11
Single Number	11.1
Single Number II	11.2
Single Number III	11.3
O1 Check Power of 2	11.4
Convert Integer A to Integer B	11.5
Factorial Trailing Zeroes	11.6
Unique Binary Search Trees	11.7
Update Bits	11.8
Fast Power	11.9
Hash Function	11.10
Count 1 in Binary	11.11
Fibonacci	11.12
A plus B Problem	11.13
Print Numbers by Recursion	11.14
Majority Number	11.15
Majority Number II	11.16
Majority Number III	11.17
Digit Counts	11.18
Ugly Number	11.19
Plus One	11.20
Linked List	12
Remove Duplicates from Sorted List	12.1
Remove Duplicates from Sorted List II	12.2
Remove Duplicates from Unsorted List	12.3
Partition List	12.4
Add Two Numbers	12.5
Two Lists Sum Advanced	12.6
Remove Nth Node From End of List	12.7
Linked List Cycle	12.8
Linked List Cycle II	12.9
Reverse Linked List	12.10
Reverse Linked List II	12.11
Merge Two Sorted Lists	12.12
Merge k Sorted Lists	12.13

Reorder List	12.14
Copy List with Random Pointer	12.15
Sort List	12.16
Insertion Sort List	12.17
Palindrome Linked List	12.18
Delete Node in the Middle of Singly Linked List	12.19
LRU Cache	12.20
Rotate List	12.21
Swap Nodes in Pairs	12.22
Remove Linked List Elements	12.23
<b>Binary Tree</b>	<b>13</b>
Binary Tree Preorder Traversal	13.1
Binary Tree Inorder Traversal	13.2
Binary Tree Postorder Traversal	13.3
Binary Tree Level Order Traversal	13.4
Binary Tree Level Order Traversal II	13.5
Maximum Depth of Binary Tree	13.6
Balanced Binary Tree	13.7
Binary Tree Maximum Path Sum	13.8
Lowest Common Ancestor	13.9
Invert Binary Tree	13.10
Diameter of a Binary Tree	13.11
Construct Binary Tree from Preorder and Inorder Traversal	13.12
Construct Binary Tree from Inorder and Postorder Traversal	13.13
Subtree	13.14
Binary Tree Zigzag Level Order Traversal	13.15
Binary Tree Serialization	13.16
<b>Binary Search Tree</b>	<b>14</b>
Insert Node in a Binary Search Tree	14.1
Validate Binary Search Tree	14.2
Search Range in Binary Search Tree	14.3
Convert Sorted Array to Binary Search Tree	14.4
Convert Sorted List to Binary Search Tree	14.5
Binary Search Tree Iterator	14.6
<b>Exhaustive Search</b>	<b>15</b>
Subsets	15.1
Unique Subsets	15.2
Permutations	15.3
Unique Permutations	15.4
Next Permutation	15.5
Previous Permutation	15.6
Permutation Index	15.7

Permutation Index II	15.8
Permutation Sequence	15.9
Unique Binary Search Trees II	15.10
Palindrome Partitioning	15.11
Combinations	15.12
Combination Sum	15.13
Combination Sum II	15.14
Minimum Depth of Binary Tree	15.15
Word Search	15.16
<b>Dynamic Programming</b>	<b>16</b>
Triangle	16.1
Backpack	16.2
Backpack II	16.3
Minimum Path Sum	16.4
Unique Paths	16.5
Unique Paths II	16.6
Climbing Stairs	16.7
Jump Game	16.8
Word Break	16.9
Longest Increasing Subsequence	16.10
Palindrome Partitioning II	16.11
Longest Common Subsequence	16.12
Edit Distance	16.13
Jump Game II	16.14
Best Time to Buy and Sell Stock	16.15
Best Time to Buy and Sell Stock II	16.16
Best Time to Buy and Sell Stock III	16.17
Best Time to Buy and Sell Stock IV	16.18
Distinct Subsequences	16.19
Interleaving String	16.20
Maximum Subarray	16.21
Maximum Subarray II	16.22
Longest Increasing Continuous subsequence	16.23
Longest Increasing Continuous subsequence II	16.24
Egg Dropping Puzzle	16.25
Maximal Square	16.26
<b>Graph</b>	<b>17</b>
Find the Connected Component in the Undirected Graph	17.1
Route Between Two Nodes in Graph	17.2
Topological Sorting	17.3
Word Ladder	17.4
Bipartite Graph Part I	17.5

Data Structure	18
Implement Queue by Two Stacks	18.1
Min Stack	18.2
Sliding Window Maximum	18.3
Longest Words	18.4
Heapify	18.5
Kth Smallest Number in Sorted Matrix	18.6
Problem Misc	19
Nuts and Bolts Problem	19.1
String to Integer	19.2
Insert Interval	19.3
Merge Intervals	19.4
Minimum Subarray	19.5
Matrix Zigzag Traversal	19.6
Valid Sudoku	19.7
Add Binary	19.8
Reverse Integer	19.9
Gray Code	19.10
Find the Missing Number	19.11
N Queens	19.12
N Queens II	19.13
Minimum Window Substring	19.14
Continuous Subarray Sum	19.15
Continuous Subarray Sum II	19.16
Longest Consecutive Sequence	19.17
Part III - Contest	20
Google APAC	21
APAC 2015 Round B	21.1
Problem A. Password Attacker	21.1.1
APAC 2016 Round D	21.2
Problem A. Dynamic Grid	21.2.1
Microsoft	22
Microsoft 2015 April	22.1
Problem A. Magic Box	22.1.1
Problem B. Professor Q's Software	22.1.2
Problem C. Islands Travel	22.1.3
Problem D. Recruitment	22.1.4
Microsoft 2015 April 2	22.2
Problem A. Lucky Substrings	22.2.1
Problem B. Numeric Keypad	22.2.2
Problem C. Spring Outing	22.2.3
Microsoft 2015 September 2	22.3

Problem A. Farthest Point	22.3.1
Appendix I Interview and Resume	23
Interview	23.1
Resume	23.2
Appendix II System Design	24
The System Design Process	24.1
Statistics	24.2
System Architecture	24.3
Scalability	24.4

# Data Structure and Algorithm/leetcode/lintcode



- English via [Data Structure and Algorithm notes](#)
- 简体中文请戳 [数据结构与算法/leetcode/lintcode题解](#)
- 繁體中文請瀏覽 [資料結構與演算法/leetcode/lintcode題解](#)

## Introduction

This work is some notes of learning and practicing data structures and algorithm.

1. Part I is some brief introduction of basic data structures and algorithm, such as, linked lists, stack, queues, trees, sorting and etc.
2. Part II is the analysis and summary of programming problems, and most of the programming problems come from <https://leetcode.com/>, <http://www.lintcode.com/>, <http://www.geeksforgeeks.org/>, <http://hihocoder.com/>, <https://www.topcoder.com/>.
3. Part III is the appendix of resume and other supplements.

This project is hosted on <https://github.com/billryan/algorith-exercise> and rendered by [Gitbook](#). You can star the repository on the GitHub to keep track of updates. Another choice is to subscribe channel `#github_commit` via Slack [https://ds-algo.slack.com/messages/github\\_commit/](https://ds-algo.slack.com/messages/github_commit/). ~~RSS feed is under development.~~

Feel free to access <http://slackin4ds-algo.herokuapp.com> for Slack invite automation.

You can view/search this document online or offline, feel free to read it. :)

- Online(Rendered by Gitbook): <http://algorithm.yuanbin.me>
- Offline(Compiled by Gitbook and Travis-CI):
  1. EPUB: [GitHub](#), [Gitbook](#), [GitCafe\(mainland China\)](#) - Recommended for iPhone/iPad/MAC
  2. PDF: [GitHub](#), [Gitbook](#), [GitCafe\(mainland China\)](#) - Recommended for Desktop
  3. MOBI: [GitHub](#), [Gitbook](#), [GitCafe\(mainland China\)](#) - Recommended for Kindle
- Site Search via Google: `keywords site:algorithm.yuanbin.me`
- Site Search via Swifttype: Click `Search this site` on the right bottom of webpages

## License

This work is licensed under the **Creative Commons Attribution-ShareAlike 4.0 International License**. To view a copy of this license, please visit <http://creativecommons.org/licenses/by-sa/4.0/>

## To Do

- [ ] add multiple languages support, currently 繁體中文, 简体中文 are available
- [x] explore nice writing style
- [x] add implementations of `Python`, `C++`, `Java` code
- [x] add time and space complexity analysis
- [x] summary of basic data structure and algorithm
- [x] add CSS for online website <http://algorithm.yuanbin.me>
- [x] add proper Chinese fonts for PDF output

## FAQ - Frequently Asked Question

Some guidelines for contributing and other questions are listed here.

### How to Contribute?

- Access [Guidelines for Contributing](#) for details.

## Guidelines for Contributing

- Access English via [Guidelines for Contributing](#)
- 繁體中文請移步 [貢獻指南](#)
- 简体中文请移步 [贡献指南](#)

# Contributors

Contributors are listed here.

## Maintainer

- [English](#) is maintained by [@billryan](#), [@niangaotuantuan](#)
- [简体中文](#) is maintained by [@billryan](#), [@Shaunwei](#), [@niangaotuantuan](#)
- [繁體中文](#) is maintained by [@CrossLuna](#)

You can find other contributors in [Contributors to billryan/algorithm-exercise](#).

## Donation

For privacy protection, some personal information are omitted.

- taoli\*\*@gmail.com , 支付宝转账
- 张亚\* , 支付宝转账
- 俞卓\* , 支付宝转账
- 季\* , 支付宝转账

## Part I - Basics

# Data Structure

# String

String related topics are discussed in this chapter.

In order to re-use most of the memory of an existing data structure, internal implementation of string is immutable in most programming languages(Java, Python). Take care if you want to modify character in place.

# strStr

## Question

- leetcode: [Implement strStr\(\) | LeetCode OJ](#)
- lintcode: [lintcode - \(13\) strstr](#)

## Problem Statement

For a given source string and a target string, you should output the **first** index(from 0) of target string in source string.

If target does not exist in source, just return `-1`.

## Example

If source = `"source"` and target = `"target"`, return `-1`.

If source = `"abcdabcdefg"` and target = `"bcd"`, return `1`.

## Challenge

$O(n^2)$  is acceptable. Can you implement an  $O(n)$  algorithm? (hint: *KMP*)

## Clarification

Do I need to implement KMP Algorithm in a real interview?

- Not necessary. When you meet this problem in a real interview, the interviewer may just want to test your basic implementation ability. But make sure your confirm with the interviewer first.

## Problem Analysis

It's very straightforward to solve string match problem with nested for loops. Since we must iterate the target string, we can optimize the iteration of source string. It's unnecessary to iterate the source string if the length of remaining part does not exceed the length of target string. We can only iterate the valid part of source string. Apart from this naive algorithm, you can use a more effective algorithm such as KMP.

## Python

```
class Solution:
    def strStr(self, source, target):
        if source is None or target is None:
            return -1

        for i in range(len(source) - len(target) + 1):
            for j in range(len(target)):
                if source[i + j] != target[j]:
                    break
                else: # no break
                    return i
        return -1
```

## C

```

int strStr(char* haystack, char* needle) {
    if (haystack == NULL || needle == NULL) return -1;

    const int len_h = strlen(haystack);
    const int len_n = strlen(needle);
    for (int i = 0; i < len_h - len_n + 1; i++) {
        int j = 0;
        for (; j < len_n; j++) {
            if (haystack[i+j] != needle[j]) {
                break;
            }
        }
        if (j == len_n) return i;
    }

    return -1;
}

```

## C++

```

class Solution {
public:
    int strStr(string haystack, string needle) {
        if (haystack.empty() && needle.empty()) return 0;
        if (haystack.empty()) return -1;
        if (needle.empty()) return 0;
        // in case of overflow for negative
        if (haystack.size() < needle.size()) return -1;

        for (int i = 0; i < haystack.size() - needle.size() + 1; i++) {
            string::size_type j = 0;
            for (; j < needle.size(); j++) {
                if (haystack[i + j] != needle[j]) break;
            }
            if (j == needle.size()) return i;
        }

        return -1;
    }
};

```

## Java

```

public class Solution {
    public int strStr(String haystack, String needle) {
        if (haystack == null && needle == null) return 0;
        if (haystack == null) return -1;
        if (needle == null) return 0;

        for (int i = 0; i < haystack.length() - needle.length() + 1; i++) {
            int j = 0;
            for (; j < needle.length(); j++) {
                if (haystack.charAt(i+j) != needle.charAt(j)) break;
            }
            if (j == needle.length()) return i;
        }

        return -1;
    }
}

```

## Source Code Analysis

- corner case: `haystack(source)` and `needle(target)` may be empty string.

2. code convention:
  - o space is needed for `==`
  - o use meaningful variable names
  - o put a blank line before declaration `int i, j;`
3. declare `j` outside for loop if and only if you want to use it outside.

Some Pythonic notes: [4. More Control Flow Tools](#) section 4.4 and [if statement - Why does python use 'else' after for and while loops?](#)

## Complexity Analysis

nested for loop,  $O((n - m)m)$  for worst case.