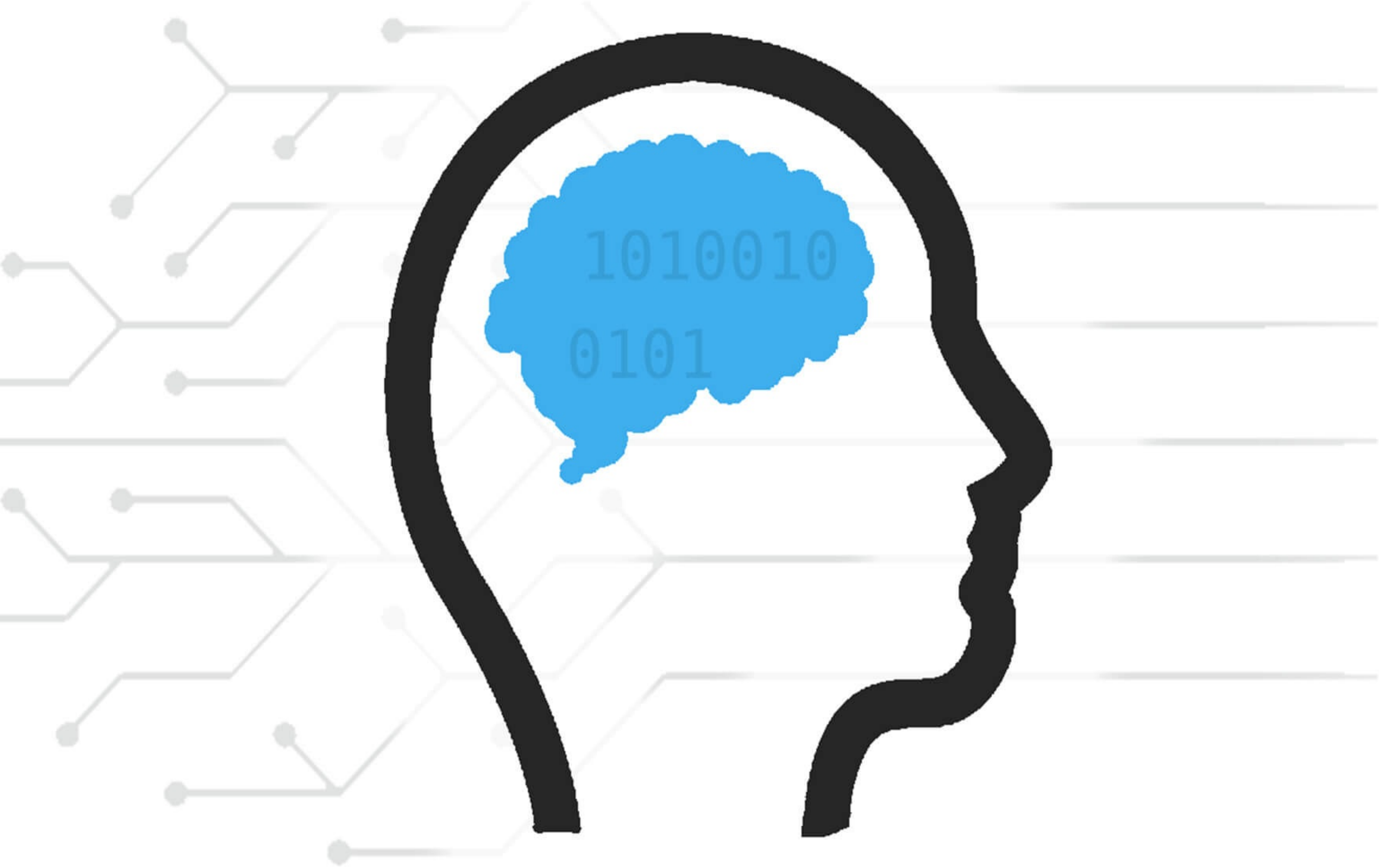


Algorithm

🄯 http://algorithm.yuanbin.me

Table of Contents

Data Structure and Algorithm/leetcode/lintcode

build passing slack 6/325 chat on slack

- English via Data Structure and Algorithm notes
- /leetcode/lintcode
- /leetcode/lintcode

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/algorithm-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/. ~~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, CDN() - Recommended for iPhone/iPad/MAC
  2. PDF: GitHub, Gitbook, CDN() - Recommended for Desktop
  3. MOBI: GitHub, Gitbook, CDN() - Recommended for Kindle
- Site Search via Google: `keywords site:algorithm.yuanbin.me`
- Site Search via Swiftype: Click `Search this site` on the right bottom of webpages

License

Contribution

- English is maintained by @billryan
- is maintained by @billryan, @Shaunwei
- is maintained by @CrossLuna

Other contributors can be found in Contributors to algorithm-exercise

Donation

:)

@billryan ~ `5307`

@billryan

yuanbin2014(at)gmail.com

Wechat

PayPal

yuanbin2014(at)gmail.com friends and family

- `taoli***@gmail.com` , 20

- `*` , 6.66
- `wen***@126.com` , 20.16
- `she***@163.com` , 10
- `*` , 20
- `*` , 50
- `*` , 20
- `don***@163.com` , 5
- `129***@qq.com` , 50
- `130****9675` , 5
- `Tong W***` , 20 $
- `ee.***@gmail.com` , 6.66

CDN / Contributors ///

To Do

- [ ] add multiple languages support, currently , 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
-
-

Part I - Basics

The first part summarizes some of the main aspects of data structures and algorithms, such as implementation and usage.

This chapter consists of the following sections.

Reference

- VisuAlgo - Animated visualizations of data structures and algorithms
- Data Structure Visualizations - An alternative to VisuAlgo
- Sorting Algorithms - Animations comparing various sorting algorithms

Data Structure

This chapter describes the fundamental data structures and their implementations.

String

String-related problems often appear in interview questions. In actual development, strings are also frequently used. Summarized here are common uses of strings in C++, Java, and Python.

Python

```
s1 = str()
# in python, `'`' and `"`' are the same
s2 = "shaunwei" # 'shaunwei'
s2len = len(s2)
# last 3 chars
s2[-3:] # wei
s2[5:8] # wei
s3 = s2[:5] # shaun
s3 += 'wei' # return 'shaunwei'
# list in python is same as ArrayList in java
s2list = list(s3)
# string at index 4
s2[4] # 'n'
# find index at first
s2.index('w')  # return 5, if not found, throw ValueError
s2.find('w') # return 5, if not found, return -1
```

In Python, there's no StringBuffer or StringBuilder. However, string manipulations are fairly efficient already.

Java

```
String s1 = new String();
String s2 = "billryan";
int s2Len = s2.length();
s2.substring(4, 8); // return "ryan"
StringBuilder s3 = new StringBuilder(s2.substring(4, 8));
s3.append("bill");
String s2New = s3.toString(); // return "ryanbill"
// convert String to char array
char[] s2Char = s2.toCharArray();
// char at index 4
char ch = s2.charAt(4); // return 'r'
// find index at first
int index = s2.indexOf('r'); // return 4. if not found, return -1
```

The difference between StringBuffer and StringBuilder is that the former guarantees thread safety. In a single-threaded environment, StringBuilder is more efficient.

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(n2) 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

```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++

```cpp
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

```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

1. corner case: `haystack(source)` and `needle(target)` may be empty string.
2. code convention:
   - space is needed for `==`
   - use meaningful variable names
   - 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.

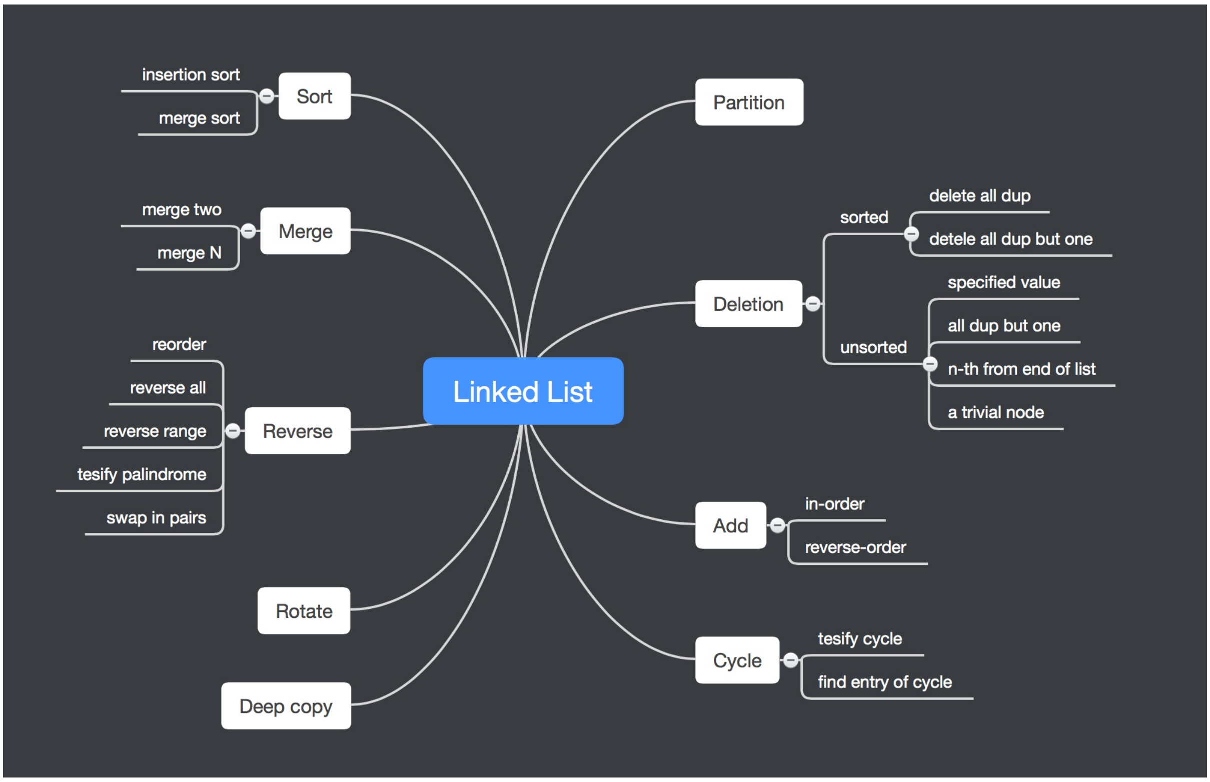
Linked List

This section includes common operations on linked list, such as deletion, insertion, and merging.

Frequently made mistakes:

- Not updating runner-node when traversing linked list
- Not recording head node before traversing
- returning incorrect pointer to node

The image below serves as a summarization.



Reverse Linked List

Question

- leetcode: (206) Reverse Linked List | LeetCode OJ
- lintcode: (35) Reverse Linked List

```
Reverse a linked list.

Example
For linked list 1->2->3, the reversed linked list is 3->2->1

Challenge
Reverse it in-place and in one-pass
```

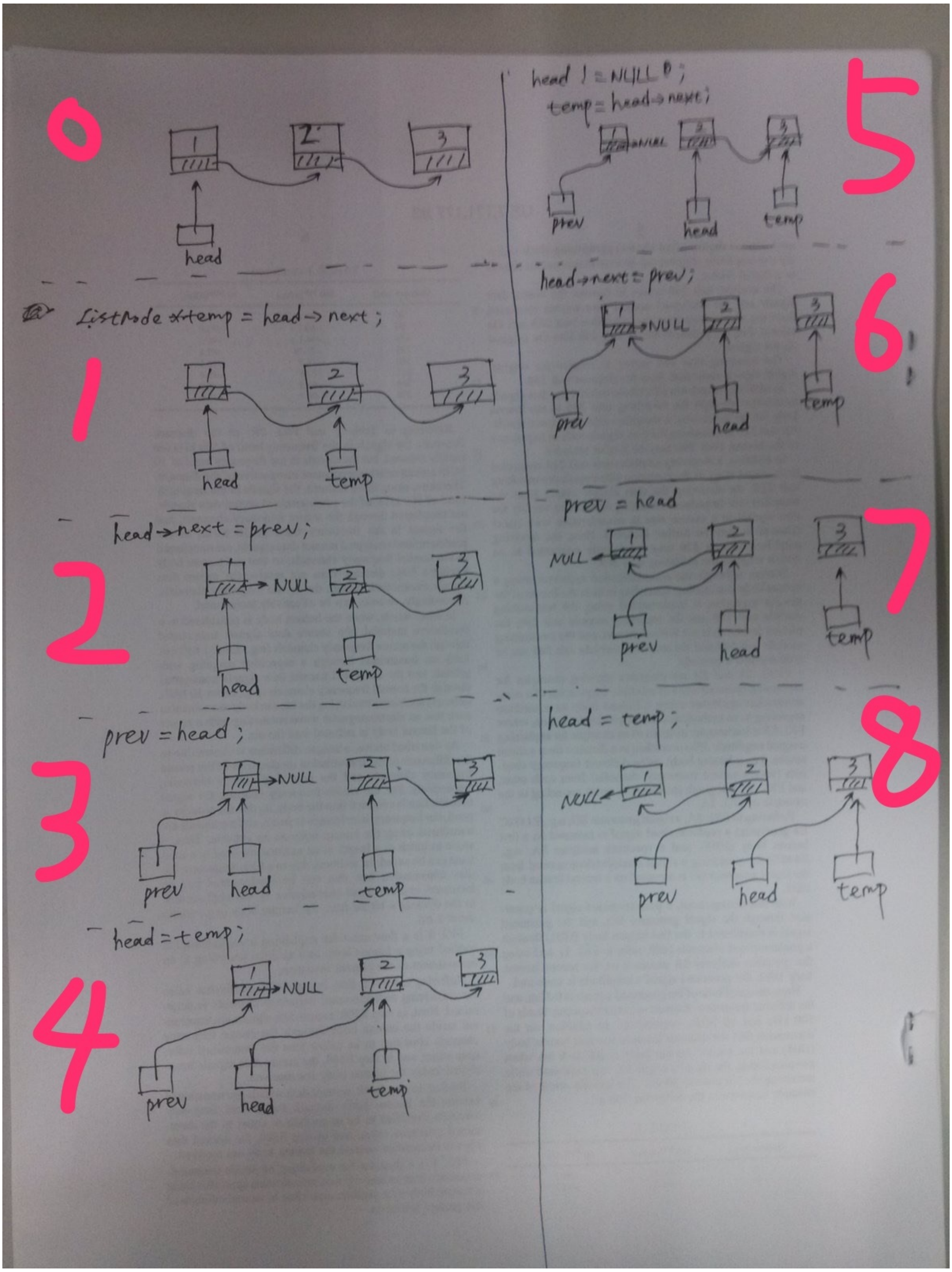
Solution1 - Non-recursively

It would be much easier to reverse an array than a linked list, since array supports random access with index, while singly linked list can ONLY be operated through its head node. So an approach without index is required.

Think about how '1->2->3' can become '3->2->1'. Starting from '1', we should turn '1->2' into '2->1', then '2->3' into '3->2', and so on. The key is how to swap two adjacent nodes.

```
temp = head -> next;
head->next = prev;
prev = head;
head = temp;
```

The above code maintains two pointer, `prev` and `head`, and keeps record of next node before swapping. More detailed analysis:



1. Keep record of next node
2. change `head->next` to `prev`
3. update `prev` with `head`, to keep moving forward
4. update `head` with the record in step 1, for the sake of next loop

Python

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    # @param {ListNode} head
    # @return {ListNode}
    def reverseList(self, head):
        prev = None
        curr = head
        while curr is not None:
            temp = curr.next
            curr.next = prev
            prev = curr
            curr = temp
        # fix head
        head = prev

        return head
```

C++

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* reverse(ListNode* head) {
        ListNode *prev = NULL;
        ListNode *curr = head;
        while (curr != NULL) {
            ListNode *temp = curr->next;
            curr->next = prev;
            prev = curr;
            curr = temp;
        }
        // fix head
        head = prev;

        return head;
    }
};
```

Java

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode prev = null;
        ListNode curr = head;
```

```
        while (curr != null) {
            ListNode temp = curr.next;
            curr.next = prev;
            prev = curr;
            curr = temp;
        }
        // fix head
        head = prev;

        return head;
    }
}
```

Source Code Analysis

Already covered in the solution part. One more word, the assignment of `prev` is neat and skilled.

Complexity

Traversing the linked list leads to ***O(n)*** time complexity, and auxiliary space complexity is ***O(1)***.

Solution2 - Recursively

Three cases when the recursion ceases:

1. If given linked list is null, just return.
2. If given linked list has only one node, return that node.
3. If given linked list has at least two nodes, pick out the head node and regard the following nodes as a sub-linked-list, swap them, then recurse that sub-linked-list.

Be careful when swapping the head node (refer as `nodeY`) and head of the sub-linked-list (refer as 'nodeX'): First, swap `nodeY` and `nodeX` ; Second, assign `null` to `nodeY->next` (or it would fall into infinite loop, and tail of result list won't point to `null`).

Python

```python
"""
Definition of ListNode

class ListNode(object):

    def __init__(self, val, next=None):
        self.val = val
        self.next = next
"""
class Solution:
    """
    @param head: The first node of the linked list.
    @return: You should return the head of the reversed linked list.
                  Reverse it in-place.
    """
    def reverse(self, head):
        # case1: empty list
        if head is None:
            return head
        # case2: only one element list
        if head.next is None:
            return head
        # case3: reverse from the rest after head
        newHead = self.reverse(head.next)
```

```
        # reverse between head and head->next
        head.next.next = head
        # unlink list from the rest
        head.next = None

        return newHead
```

C++

```
/**
 * Definition of ListNode
 *
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: The new head of reversed linked list.
     */
    ListNode *reverse(ListNode *head) {
        // case1: empty list
        if (head == NULL) return head;
        // case2: only one element list
        if (head->next == NULL) return head;
        // case3: reverse from the rest after head
        ListNode *newHead = reverse(head->next);
        // reverse between head and head->next
        head->next->next = head;
        // unlink list from the rest
        head->next = NULL;

        return newHead;
    }
};
```

Java

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode reverse(ListNode head) {
        // case1: empty list
        if (head == null) return head;
        // case2: only one element list
        if (head.next == null) return head;
        // case3: reverse from the rest after head
        ListNode newHead = reverse(head.next);
        // reverse between head and head->next
```

```
        head.next.next = head;
        // unlink list from the rest
        head.next = null;

        return newHead;
    }
}
```

Source Code Analysis

case1 and case2 can be combined.What case3 returns is head of reversed list, which means it is exact the same Node (tail of origin linked list) through the recursion.

Complexity

- - -
- data structures - Reversing a linked list in Java, recursively - Stack Overflow
- C++ |
- iteratively and recursively Java Solution - Leetcode Discuss

Tags