目录

Leetcode

最近因为一些事情又开始刷题了，所以这个当作记录刷题笔记的网站吧。

Two Sum

Two Sum

Given an array of integers, return **indices** of the two numbers such that they add up to a specific target.

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

Example:

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

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

这也是我第一次面试的第一个算法，当时想的方法是最暴力的方法，直接两个循环完事，但是时间复杂度是$O(n^2)$ ，那是相当的高o(╥﹏╥)o。后来想了一下找到了一个简单的方法，当时由于太紧张写的不太好。当时的问题方法也稍微有点不同，当时要求的是找到相对应的数字，不是*index*。 先回顾一下当时的算法：

```
int[] another(int[] nums, int target) {
    int head = 0;
    int tail = 0;

    // sort it
    Arrays.sort(nums);

    while (true) {
        int value = nums[head] + nums[tail];

        if (value == target) break;
        else if (value > target) tail -= 1;
        else head += 1;

        if (head == tail) break;
    }

    if (head != tail) return new int[] {nums[head], nums[tail]};
    return null;
}
```

当时想到的是两头同时遍历的方法，但是有个前提要求就是list必须是排过序的，当 `nums` 无限大时排序所消耗的时间可以忽略的，这个方法有点类似于**binary search**。我当时也想过用 `HashMap` ，好像处于什么原因被面试官否定了，whatever不重要的。不闲扯了，我们开始看这题的比较优化的算法，时间复杂度是 `O(n)` 。

```
int[] twoSum(int[] nums, int target) {
    Map<Integer, Integer> map = new HashMap<>();

    for (int i = 0; i < nums.length; i ++) {
        int minus = target - nums[i];
        if (map.containsKey(minus)) {
            return new int[]{map.get(minus), i};
        } else {
            map.put(nums[i], i);
        }
    }

    return null;
}
```

算法思路：

1. 因为是用的 `HashMap` 所以在检测key是否contain的话使用的是hash的方法，所以复杂度是$O(1)$
2. 第7-11行，是来判断这个HashMap是否包含这个num，如果这个num不包含在HashMap里面，然后把该num存到HashMap里面，num当做key，index当做value。这样就相当于把访问过的num和num的index存到HashMap里面，然后再遍历nums的时候得对每个数可以得到一个差值，然后判断这个差值是否存在HashMap里面，如果存在就代表得到这个结果了，如果不存在就把这个 `nums[i]` 存到HashMap。

powered by GitbookFile Modify: 2024-11-27 04:55:52

Add Two Numbers

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 contain a single digit. Add the two numbers and return it as a linked list.

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

Example:

```
Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8
Explanation: 342 + 465 = 807.
```

算法思想：

主要应该考虑一下边界情况：

1. 当sum>=10
2. 两个链表不一样长
3. 在最后一个数字相加的时候sum>=10

第6行的while语句就是为了两条链表不一样长度。第7-8行判断如果该链表不为null就取该val，不然的话就为0（为了不影响下面的计算）。 `int overflow = 0` 是为了解决和超过10的情况，第9行把溢出的数字和v1，v2相加，然后把该结果取余就是结果。然后把 `overflow = sum/10` 得到进位数。每次生成新的数字之后生成一个新的node没然后链接起来。主要是考虑不同的边界情况。

```java
public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
    int overflow = 0;
    if (l1 == null && l2 == null) return null;
    ListNode new_node = new ListNode();
    ListNode head = new_node;
    while (l1 != null || l2 != null) {
        int v1 = (l1 != null) ? l1.val : 0;
        int v2 = (l2 != null) ? l2.val : 0;
        int sum = v1+v2 + overflow;
        overflow = sum / 10;
        if (l1 != null) l1 = l1.next;
        if (l2 != null) l2 = l2.next;
        new_node.val = sum % 10;
        if (l1 != null || l2 != null) {
            new_node.next = new ListNode();
            new_node = new_node.next;
        }
    }
    if (overflow > 0) {
        new_node.next = new ListNode(overflow % 10);
    }
    return head;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Longest Substring Without Repeating Characters

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

Example 1:

```
Input: "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

```
Input: "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

```
Input: "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
             Note that the answer must be a substring, "pwke" is a subsequence and not a
substring.
```

解题方法&思路：

首先说下题意，我刚开始是没理解题意来得（暴露了自己的无知╮(╯▽╰)╭），题意就是匹配到最长的子字符串，并且不能重复。比如 `pwwkew` 可以匹配到 `wke` 最后一个w不可以被匹配到，因为该子字符串已经包含了一个w。

算法解析：~~我们可以使用暴力解法：双循环（但是不推荐)~~。我们可以使用HashMap来解决该问题，因为HashMap查询的时候只需要$O(1)$的时间复杂度。

1. 首先对该字符串进行遍历，如果该字符不在HashMap里面，则把该字符添加到HashMap，取max（HashMap length, ans）
2. 如果该字符已经出现在HashMap里面，从HashMap里面去除一个字符，然后current index保持不变（会对该index进行再次遍历）
3. 循环

有点类似于：当一个字符不在该HashMap，push进去，如果已经在了，pop出来。然后在push之后计算最大的长度是多少。（进去一个如果已经存在就pop出来一个，因为重点是长度，所以不需要关心到底是哪些字符）。

Python伪代码

```
def LongSubString(s):
    while i < len(s):
    if not s[i] in hashmap:
      hashmap[s[i]] = 0
      ans = max(len(hashmap), ans)
    else:
      del hashmap[s[d]]
      d += 1
  return ans
```

Java解决方法：

```
import java.util.HashMap;
import java.util.Map;

public class LongestSubstring {
    public int lengthOfLongestSubstring(String s) {
        int ans = 0;
        int remove = 0;
        Map<Character, Integer> map = new HashMap<>();

        for (int i = 0; i < s.length();) {
            if (!map.containsKey(s.charAt(i))) {
                map.put(s.charAt(i), 0);
                ans = Math.max(map.size(), ans);
                i ++;
            } else {
                map.remove(s.charAt(remove));
                remove += 1;
            }
        }


        return ans;
    }

    public static void main(String[] args) {
        LongestSubstring lon = new LongestSubstring();
        System.out.println(lon.lengthOfLongestSubstring("abcabcbb"));
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Median of two sorted array

There are two sorted arrays **nums1** and **nums2** of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

You may assume **nums1** and **nums2** cannot be both empty.

Example 1:

```
nums1 = [1, 3]
nums2 = [2]

The median is 2.0
```

Example 2:

```
nums1 = [1, 2]
nums2 = [3, 4]

The median is (2 + 3)/2 = 2.5
```

没办法太菜了找不到合适的方法来写了，有想过分开循环，但是这样需要有很多的判断条件，晚会会再尝试的，目前就快的方法就是合并array了。

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

    List<Integer> list = new LinkedList<>();
    int l1 = 0, l2 = 0;

    while (l1 < nums1.length && l2 < nums2.length) {
        if (nums1[l1] < nums2[l2]) {
            list.add(nums1[l1]);
            l1 += 1;
        } else {
            list.add(nums2[l2]);
            l2 += 1;
        }
    }

    // add n2
    if (l1 >= nums1.length) {
        for (int i = l2; i < nums2.length; i ++) list.add(nums2[i]);
    } else {
        for (int i = l1; i < nums1.length; i ++) list.add(nums1[i]);
    }

    int len = list.size();
    if (len % 2 != 0) return list.get(len/2);
    return (double) (list.get(len/2) + list.get(len/2-1)) / 2;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Reverse Integer

reverse integer

Given a 32-bit signed integer, reverse digits of an integer.

Example 1:

```
Input: 123
Output: 321
```

Example 2:

```
Input: -123
Output: -321
```

Example 3:

```
Input: 120
Output: 21
```

本来以为这个题很简单，使用了最简单的算法，但是呢，出现了一个整型溢出的问题o(╥﹏╥)o。

先解析一下思路吧：

1. 首先我们可以用 `x%10` 来得到最后一个数字（因为数字不可能大于10）
2. `x/10` 是把最后一位数去掉
3. `reverse*10` 是为了整体向左移一位，比如(44, 44*10 = 440)多出一个空位，然后加上 `x%10` 的数字

```
public int reverse(int x) {
    int reverse = 0;

    while (x != 0) {
        int last_digit = x % 10;
        reverse = reverse * 10 + last_digit;
        x = x / 10;
    }

    return reverse ;
}
```

这样如果数字区间在$[−2^{31}, 2^{31} − 1]$的话那么就会出现溢出的问题了，大家可以使用 `1534236469` 来尝试一下。

所以这时候我们就需要来做些判断了：

当 `reverse > intMAX/10` 的时候我们基本上就可以确定他会溢出了，比如说 `intMAX = 2147483647` ，假设我们的 `reverse = 214748365` 这时候 `intMAX/10 =214748364` ， `reverse > intMAX` 。这时候如果 `reverse * 10` 我们不管最后一个数字是什么了， `reverse` 已经超出了int取值范围。

正确代码：

```
public int reverse(int x) {
    int reverse = 0;

    while (x != 0) {
        int last_digit = x % 10;
        if (reverse > Integer.MAX_VALUE / 10) return 0;
        if (reverse < Integer.MIN_VALUE / 10) return 0;
        reverse = reverse * 10 + last_digit;
        x = x / 10;
    }

    return reverse ;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

String to Integer (atoi)

String to Integer (atoi)

Implement `atoi` which converts a string to an integer.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str is not a valid integral number, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed.

If no valid conversion could be performed, a zero value is returned.

Note:

- Only the space character `' '` is considered as whitespace character.
- Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range: [−231, 231 − 1]. If the numerical value is out of the range of representable values, INT_MAX (231 − 1) or INT_MIN (−231) is returned.

Example 1:

```
Input: "42"
Output: 42
```

Example 2:

```
Input: "   -42"
Output: -42
Explanation: The first non-whitespace character is '-', which is the minus sign.
             Then take as many numerical digits as possible, which gets 42.
```

Example 3:

```
Input: "4193 with words"
Output: 4193
Explanation: Conversion stops at digit '3' as the next character is not a numerical digit
.
```

Example 4:

```
Input: "words and 987"
Output: 0
Explanation: The first non-whitespace character is 'w', which is not a numerical
             digit or a +/- sign. Therefore no valid conversion could be performed.
```

Example 5:

```
Input: "-91283472332"
Output: -2147483648
Explanation: The number "-91283472332" is out of the range of a 32-bit signed integer.
             Thefore INT_MIN (−231) is returned.
```

这是我写LeetCode以来submit最多的次数╮(╯▽╰)╭，里面条件的限制和符号的判定太烦了。给大家写一下判断条件：

1. 空格只能出现在最前面，一旦出现了-+0-9这些符号，之后再遇到空格就要break
2. -+符号考虑到符号的问题
3. 只能以数字开头的才能提取数字，如果不是就return 0
4. 考虑到整型溢出的问题

就是判断条件有点麻烦，其他的还好，我写的有点乱七八糟的，讲究看下。。。如果有好的idea可以写在comment里面，我会修改的。

```java
public int myAtoi(String str) {
    int res = 0;
    // + or -
    int operation = 1;
    if (str.equals(""))return 0;
    boolean whitespace = true;
    boolean sign = true;

    for (int i = 0; i < str.length(); i ++) {
        char c = str.charAt(i);
        if (whitespace && c == ' ') continue;
        if (!whitespace && c == ' ') break;
        if ((c == '-' || c == '+') && sign) {
            operation = (c == '-')?-1:1;
            sign = false;
            whitespace=false;
            continue;
        }
        if (c < '0' || c > '9') break;
        whitespace = false;
        sign=false;
        if (res*operation > Integer.MAX_VALUE/10 || res*operation < Integer.MIN_VALUE/10)
 {
            if (operation > 0) return Integer.MAX_VALUE;
            else return Integer.MIN_VALUE;
        }

        if (res*operation == Integer.MAX_VALUE/10) if ((int)c-48 >= Integer.MAX_VALUE %10
) return Integer.MAX_VALUE;
        if (res*operation == Integer.MIN_VALUE/10) if ((int)(c-48) * -1 <= Integer.MIN_VA
LUE % 10) return Integer.MIN_VALUE;
        res = res * 10 + (int) c - 48;
    }
    res *= operation;
    return res;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Palindrome Number

palindrome number

Determine whether an integer is a palindrome. An integer is a palindrome when it reads the same backward as forward.

Example 1:

```
Input: 121
Output: true
```

Example 2:

```
Input: -121
Output: false
Explanation: From left to right, it reads -121. From right to left, it becomes 121-. Ther
efore it is not a palindrome.
```

Example 3:

```
Input: 10
Output: false
Explanation: Reads 01 from right to left. Therefore it is not a palindrome.
```

回文数也就是说该数字等于翻转后的数字， `num = reverse(num)` 。首先我们可以确定如果该数字是负数直接false就好了。我们可以使用上一题的思路，把数字翻转然后看是否相等。也可以把边界情况考虑进行。

```
public boolean isPalindrome(int x) {
    if (x < 0) return false;

    int reverse = 0, temp = x;

    while (temp != 0) {
        int last_digit = temp % 10;
        reverse = reverse * 10 + last_digit;
        temp = temp / 10;
    }

    return reverse == x;
}
```

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 `""`.

Example 1:

```
Input: ["flower","flow","flight"]
Output: "fl"
```

Example 2:

```
Input: ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input strings.
```

Note:

All given inputs are in lowercase letters `a-z`.

集体思路：

采用了垂直对比法。比如：

```
s1 = 'flower'

s2 = 'flow'
s3 = 'flight'

我们第一层循环是表示当前string的index, i = 0;
然后得到第一个string的第i个char
第二个循环式为了和剩余的string比对，
如果i的大小等于当前string的长度也就意味着遇到了结束点，比如当i=3的时候，i = s2.length，也就意味着有
结束点了。

c != strs[j].charAt(i) 这就语句是为了判断当前string的第i个char是否和第一个相等，如果不想当也就意味
着可以返回i之前的数据



最后一行意味着strs[0]是最短的字符串，而且strs[0]是comm prefix
```

```java
public String longestCommonPrefix(String[] strs) {
    if (strs == null || strs.length == 0) return "";

    for (int i = 0; i < strs[0].length(); i ++ ) {
        char c = strs[0].charAt(i);

        for (int j = 1; j < strs.length; j ++) {
            if (i == strs[j].length() || c != strs[j].charAt(i))
                return strs[0].substring(0, i);
        }
    }
    return strs[0];
}
```

也可以试着得到最短的string，然后进行循环，思路差不多：

这种情况其实没什么必要，因为可能没循环结束就return了。

```java
public String longestCommonPrefix(String[] strs) {
    if (strs == null || strs.length == 0) return "";

    int min = Integer.MAX_VALUE;
    int index = -1;

    // get the shortest string length
    for (int i = 0; i < strs.length; i ++) {
        if (strs[i].length() < min) {
            min = strs[i].length();
            index = i;
        }
    }

    for (int i  = 0; i < min; i ++) {
        char c = strs[0].charAt(i);

        for (String str : strs) {
            if (c != str.charAt(i)) return str.substring(0, i);
        }
    }

    return strs[index];
}
```

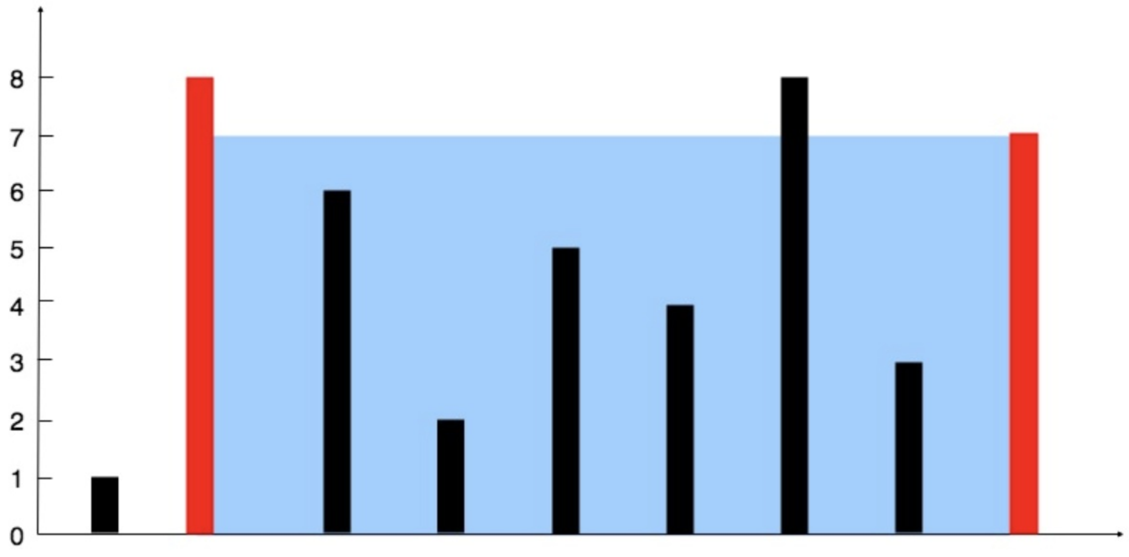
powered by GitbookFile Modify: 2024-11-27 04:38:02

Container With Most Water

Container With Most Water

Given n non-negative integers $a_1, a_2, ..., a_n$, where each represents a point at coordinate (i, a_i). n vertical lines are drawn such that the two endpoints of line i is at (i, a_i) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container and n is at least 2.



The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example:

```
Input: [1,8,6,2,5,4,8,3,7]
Output: 49
```

本来觉得是要用动态规划，可是看了大佬的代码和idea瞬间觉得自己的思路是错的。这个题其实就是变相的求最大面积，我们可以知道长方体的面积是长*高，在遍历这个list的时候长度是一直在减小的，如果长度变小我们只有找到更大的高度才能使面积最大。那么我们可以从两头同时开始遍历，如果 `left < right` 那么left向左移，因为我们要找到higher height，如果 `left >= right` right向左移就行了。这样就会找到最大的area。

ps：不想想太复杂，试着从最基本的信息去推断，比如这个面积，如果一个变量变下，那另一个变量只有变大才能找到最大的。（😁学习到了）

```
public int maxArea(int[] height) {
    int max = 0;
    int l = 0, r = height.length-1;

    while (l < r) {
        max = Math.max(max, Math.min(height[l], height[r]) * Math.abs(r - l));
        if (height[l] < height[r]) l ++;
        else r --;
    }

    return max;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Roman to Integer

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, two is written as `II` in Roman numeral, just two one's added together. Twelve is written as, `XII` , which is simply `X` + `II` . The number twenty seven 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. Input is guaranteed to be within the range from 1 to 3999.

Example 1:

```
Input: "III"
Output: 3
```

Example 2:

```
Input: "IV"
Output: 4
```

Example 3:

```
Input: "IX"
Output: 9
```

Example 4:

```
Input: "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.
```

Example 5:

```
Input: "MCMXCIV"
Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.
```

好像没什么好说的就是简单的字符串操作而已：

```
public int romanToInt(String s) {
    Map<Character, Integer> map = new HashMap<>();
    map.put('I', 1);
    map.put('V', 5);
    map.put('X', 10);
    map.put('L', 50);
    map.put('C', 100);
    map.put('D', 500);
    map.put('M', 1000);

    int res = 0;

    for (int i = 0; i < s.length(); i ++) {
        char c = s.charAt(i);
        if (c == 'V' || c == 'X') {
            if (i > 0) {
                if (s.charAt(i-1) == 'I') res += map.get(c) - 2;
                else res += map.get(c);
            } else res += map.get(c);
        } else if (c == 'L' || c == 'C') {
            if (i > 0) {
                if (s.charAt(i-1) == 'X') res += map.get(c) - 20;
                else res += map.get(c);
            } else res += map.get(c);
        } else if (c == 'D' || c == 'M') {
            if (i > 0) {
                if (s.charAt(i-1) == 'C') res += map.get(c) - 200;
                else res += map.get(c);
            } else res += map.get(c);
        } else {
            res += map.get(c);
        }
    }

    return res;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

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 `""`.

Example 1:

```
Input: ["flower","flow","flight"]
Output: "fl"
```

Example 2:

```
Input: ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input strings.
```

Note:

All given inputs are in lowercase letters `a-z`.

集体思路：

采用了垂直对比法。比如：

```
s1 = 'flower'

s2 = 'flow'
s3 = 'flight'

我们第一层循环是表示当前string的index, i = 0;
然后得到第一个string的第i个char
第二个循环式为了和剩余的string比对，
如果i的大小等于当前string的长度也就意味着遇到了结束点，比如当i=3的时候，i = s2.length，也就意味着有
结束点了。

c != strs[j].charAt(i) 这就语句是为了判断当前string的第i个char是否和第一个相等，如果不想当也就意味
着可以返回i之前的数据


最后一行意味着strs[0]是最短的字符串，而且strs[0]是comm prefix
```

```java
public String longestCommonPrefix(String[] strs) {
    if (strs == null || strs.length == 0) return "";

    for (int i = 0; i < strs[0].length(); i ++ ) {
        char c = strs[0].charAt(i);

        for (int j = 1; j < strs.length; j ++) {
            if (i == strs[j].length() || c != strs[j].charAt(i))
                return strs[0].substring(0, i);
        }
    }
    return strs[0];
}
```

也可以试着得到最短的string，然后进行循环，思路差不多：

这种情况其实没什么必要，因为可能没循环结束就return了。

```java
public String longestCommonPrefix(String[] strs) {
    if (strs == null || strs.length == 0) return "";

    int min = Integer.MAX_VALUE;
    int index = -1;

    // get the shortest string length
    for (int i = 0; i < strs.length; i ++) {
        if (strs[i].length() < min) {
            min = strs[i].length();
            index = i;
        }
    }

    for (int i  = 0; i < min; i ++) {
        char c = strs[0].charAt(i);

        for (String str : strs) {
            if (c != str.charAt(i)) return str.substring(0, i);
        }
    }

    return strs[index];
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Letter Combinations of a Phone Number

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.

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

Example:

```
Input: "23"
Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].
```

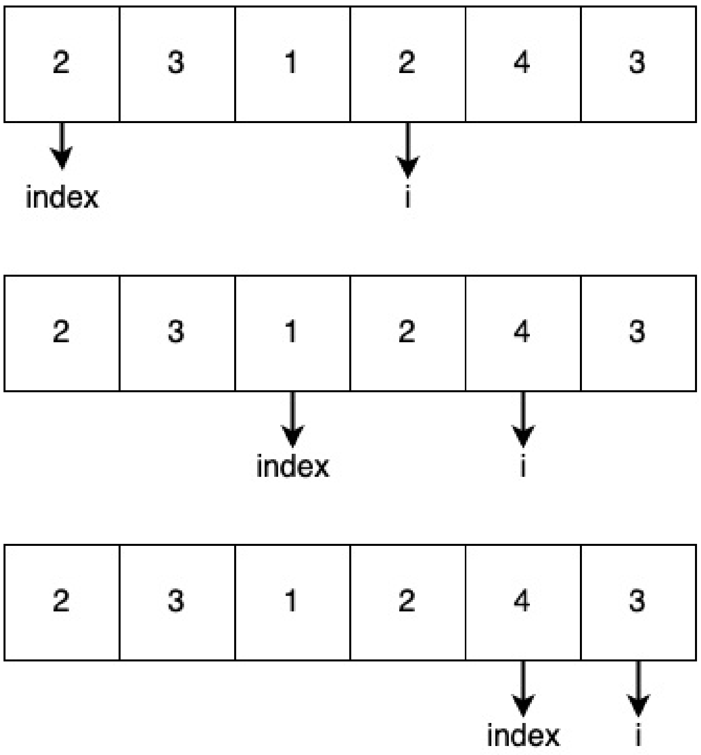
思路1：

每个数字代表几个相对应的数据，把这些代表的数据进行排列组合。这个第一个想到的就是loop，逐层递进，因为如果是三个数字的话，就是只组合成三个字母的string，可以先把1，2组合得到的结果去和3组合。这样就行了，还有别的算法。

```
import java.util.*;

public class LetterCombinationsofaPhoneNumber {
    private List<String> letterCombinations(String digits) {
        List<String> result = new LinkedList<>();
        HashMap<Character, String> maps = new HashMap<>();
        maps.put('2', "abc");
        maps.put('3', "def");
        maps.put('4', "ghi");
        maps.put('5', "jkl");
        maps.put('6', "mno");
        maps.put('7', "pqrs");
        maps.put('8', "tuv");
        maps.put('9', "wyxz");

        // 先实验一下相乘的办法，因为还有很多好的办法，一会慢慢看

        for (int i = 0; i < digits.length(); i ++) {
            if (!maps.containsKey(digits.charAt(i))) continue;

            String str = maps.get(digits.charAt(i));
            result = mul(result, new ArrayList<>(Arrays.asList(str.split(""))));
        }

        return result;
    }
    private List<String> mul(List<String> s1, List<String> s2) {
        if (s1.size() == 0 && s2.size() != 0) {
            return s2;
        }

        if (s1.size() != 0 && s2.size() == 0) {
            return s1;
        }

        List<String> result = new LinkedList<>();
        for (String value : s1) {
            for (String s : s2) {
                result.add(value + s);
            }
        }

        return result;
    }

    public static void main(String[] args) {
        LetterCombinationsofaPhoneNumber l = new LetterCombinationsofaPhoneNumber();
        String str = "23";

        List<String> res = l.letterCombinations(str);

        for (String s : res) {
            System.out.println(s);
        }
    }
}
```

思路2：

想学习一下dfs的算法。。。学习使用dfs和递归的方法，这个算法的思路就是直接访问到最底层的str，然后遍历最底层的list，遍历完之后返回到上一层，然后对上一层进行遍历，同时还会使用到最底层的元素。。。递归有点绕。。


2 3 1 2 4 3
index
i
2 3 1 2 4 3
index
i
2 3 1 2 4 3
index
i

```
private List<String> letterCombinations(String digits) {
    List<String> result = new LinkedList<>();

    if (digits == null || digits.length() == 0) return result;

    HashMap<Character, String> maps = new HashMap<>();
    maps.put('2', "abc");
    maps.put('3', "def");
    maps.put('4', "ghi");
    maps.put('5', "jkl");
    maps.put('6', "mno");
    maps.put('7', "pqrs");
    maps.put('8', "tuv");
    maps.put('9', "wyxz");

    dfs(digits, 0, "", result, maps);

    return result;
}

public void dfs(String digits, int index, String temp, List<String> res, HashMap<Characte
r, String> maps) {
    if (index == digits.length()) {
        res.add(temp);
        return;
    }

    String t = maps.get(digits.charAt(index));

    for (int i = 0; i < t.length(); i ++) {
        dfs(digits, index + 1, temp + t.charAt(i), res, maps);
    }

}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Remove Nth Node From End of List

Remove Nth Node From End of List

Given a linked list, remove the *n*-th node from the end of list and return its head.

Example:

```
Given linked list: 1->2->3->4->5, and n = 2.

After removing the second node from the end, the linked list becomes 1->2->3->5.
```

Note:

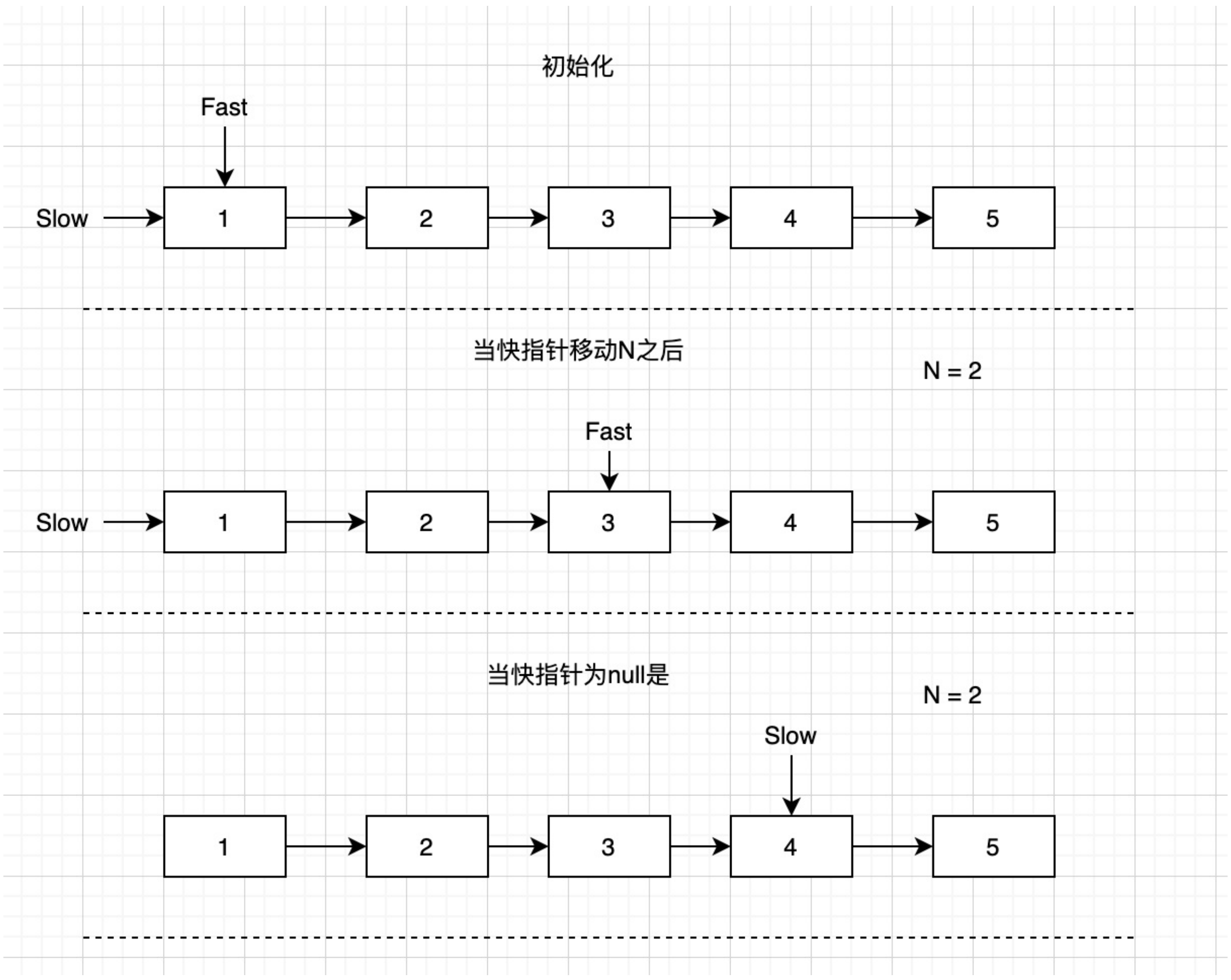
Given *n* will always be valid.

Follow up:

Could you do this in one pass?

解析：

题意就是删除倒数第N的节点。这个可以使用一个快慢指针的方法来解决，先让快的指针移动n个node，然后慢的指针开始移动。这样当快的指针为null的时候，慢的指针刚好就是要删除的节点。



其实可以当 `fast.next == null` 的时候就停止。这样 `slow.next = slow.next.next` 是一样的，因为我的代码用了 `curr` 来判断是否为null（删除的元素是不是head）

```java
public class ListNode {
     int val;
     ListNode next;
     ListNode() {}
     ListNode(int val) { this.val = val; }
     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
}

public ListNode removeNthFromEnd(ListNode head, int n) {
    ListNode curr = null;
    ListNode slow = head;
    ListNode fast = head;


    while (fast != null) {
        if (n <= 0) {
            curr = slow;
            slow = slow.next;
        }
        fast = fast.next;
        n --;
    }

    // means at the head
    if(curr == null) head = head.next;
    if (curr != null) curr.next = slow.next;

    return head;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Valid Parentheses

valid parentheses

Given a string 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.

Note that an empty string is also considered valid.

Example 1:

```
Input: "()"
Output: true
```

Example 2:

```
Input: "()[]{}"
Output: true
```

Example 3:

```
Input: "(]"
Output: false
```

Example 4:

```
Input: "([)]"
Output: false
```

Example 5:

```
Input: "{[]}"
Output: true
```

解题思路：

这个题我使用了stack（栈）的方法来解决这个问题。stack是一种先进后出的数据结构，所以更加适合这种问题。

1. 我们先来初始化一个HashMap来存相对应的括号
2. 先判断该HashMap是否存在该左括号，如果存在就把左括号push到stack里面，
3. 如果不存在先判断stack是否有数据（因为不存在的情况只有该字符是右括号的情况）
4. 然后根据stack pop出来的结果在HashMap找到相对应的右括号，然后判断该字符和HashMap里面的右括

号是否匹配。

```java
public boolean isValid(String s) {
    Stack<Character> stack = new Stack<Character>();
    Map<Character, Character> map = new HashMap<>();
    map.put('[', ']');
    map.put('(', ')');
    map.put('{', '}');

    for (int i = 0; i < s.length(); i ++) {
        if (map.containsKey(s.charAt(i))) {
            stack.push(s.charAt(i));
        } else {
            if (stack.isEmpty()) return false;
            if (!map.get(stack.pop()).equals(s.charAt(i))) return false;
        }
    }

    return stack.isEmpty();
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Merge Two Sorted Lists

Merge two sorted lists

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

Example:

```
Input: 1->2->4, 1->3->4
Output: 1->1->2->3->4->4
```

这个好像没什么说的，就直接对比大小然后merge在一起就行

```
public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
    ListNode head = new ListNode();
    ListNode temp = head;

    while (l1 != null && l2 != null) {
        temp.next = new ListNode();
        temp = temp.next;

        if (l1.val <= l2.val) {
            temp.val = l1.val;
            l1 = l1.next;
        } else {
            temp.val = l2.val;
            l2 = l2.next;
        }
    }

    if (l1 != null) temp.next = l1;
    if (l2 != null) temp.next = l2;

    return head.next;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Generate Parentheses

Generate Parentheses

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

For example, given *n* = 3, a solution set is:

```
[
  "((()))",
  "(()())",
  "(())()",
  "()(())",
  "()()()"
]
```

这个题主要是使用了递归的方法，首先一直添加 `(` ，直到最深度 `n` ，如果超过n肯定不合法了，因为左右括号是相对应的。当左括号达到最深度的时候，开始递归右括号，同时右括号的数量应该小于左括号。伪代码:

```
def find(left, right, depth, str, res):
    if str length == depth * 2:
        res.add(str)

    if left < depth:
        find(left+1, right, depth, str+'(' , res)

    if right < left:
        find(left, right+1, depth, str+')', res)
```

Java代码：

```java
import java.util.LinkedList;
import java.util.List;

public class GenerateParentheses {
    public List<String> generateParenthesis(int n) {
        List<String> list = new LinkedList<>();
        if (n == 0) return list;

        dfs(0,0,n,list, "");
        return list;
    }

    public void dfs(int close, int open, int depth, List<String> res, String str) {
        if (str.length() == depth * 2) {
            res.add(str);
            return;
        }

        if (open < depth) {
            dfs(close, open + 1, depth, res , str + "(");
        }

        if (close < open) {
            dfs(close + 1, open, depth, res, str + ")");
        }
    }

    public static void main(String[] args) {
        GenerateParentheses ge = new GenerateParentheses();
        List<String> res = ge.generateParenthesis(4);

        for (String str : res) {
            System.out.println(str);
        }
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Swap Nodes in Pairs

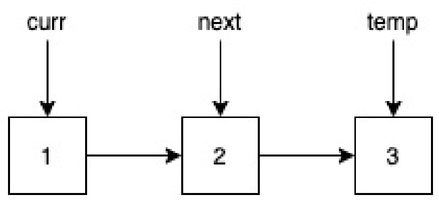
Given a linked list, swap every two adjacent nodes and return its head.

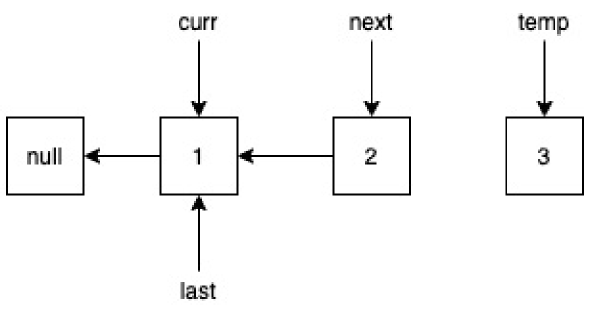
You may **not** modify the values in the list's nodes, only nodes itself may be changed.

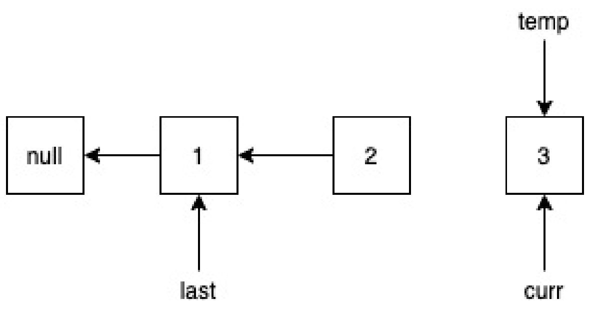
Example:

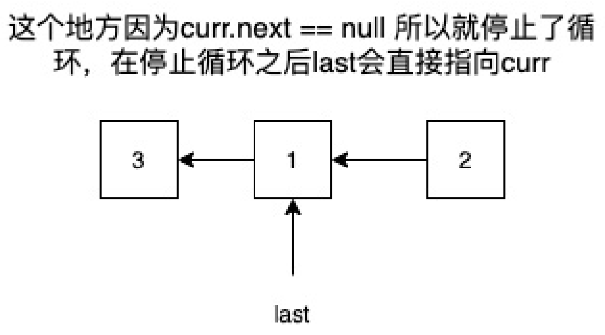
```
Given 1->2->3->4, you should return the list as 2->1->4->3.
```

交换node的位置，两两交换，要考虑的问题有两个，list长度为odd的时候怎么办，怎么去进行loop。








这个地方因为curr.next == null 所以就停止了循环，在停止循环之后last会直接指向curr

```java
public ListNode swapPairs(ListNode head) {
    if (head == null || head.next == null) return head;
    ListNode curr = head;
    head = curr.next;
    ListNode last = null;

    while (curr != null && curr.next != null) {
        ListNode next = curr.next;
        ListNode temp = next.next;

        if (last != null) {
            last.next = curr.next;
        }
        next.next = curr;
        curr.next = null;
        last = curr;

        curr = temp;
    }

    if (curr != null && curr.next == null) last.next = curr;

    return head;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Remove Duplicates from Sorted Array

Remove Duplicates from Sorted Array

Given a sorted array *nums*, remove the duplicates **in-place** such that each element appear only *once* and return the new length.

Do not allocate extra space for another array, you must do this by **modifying the input array in-place** with O(1) extra memory.

Example 1:

```
Given nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums being 1 and 2
 respectively.

It doesn't matter what you leave beyond the returned length.
```

Example 2:

```
Given nums = [0,0,1,1,1,2,2,3,3,4],

Your function should return length = 5, with the first five elements of nums being modifi
ed to 0, 1, 2, 3, and 4 respectively.

It doesn't matter what values are set beyond the returned length.
```

最获取长度的同时需要对array内部数据进行修改：

通过每次记录num，如果不相同的话，对数组的其实位置开始修改，然后curr是正在修改的index。（也会随着i增加而增加，如果所有数字都不一样）只不过是重新赋值了一次而已。

```
public int removeDuplicates(int[] nums) {
    if (nums.length == 0) return 0;
    int dup = nums[0];
    int curr = 1;

    for (int i = 0; i < nums.length; i ++) {
        if (dup != nums[i]) {
            dup = nums[i];
            nums[curr++] = nums[i];
        }
    }

    return curr;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Remove Element

Remove Element

Given an array *nums* and a value *val*, remove all instances of that value **in-place** and return the new length.

Do not allocate extra space for another array, you must do this by **modifying the input array in-place** with O(1) extra memory.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

Example 1:

```
Given nums = [3,2,2,3], val = 3,

Your function should return length = 2, with the first two elements of nums being 2.

It doesn't matter what you leave beyond the returned length.
```

Example 2:

```
Given nums = [0,1,2,2,3,0,4,2], val = 2,

Your function should return length = 5, with the first five elements of nums containing 0
, 1, 3, 0, and 4.

Note that the order of those five elements can be arbitrary.

It doesn't matter what values are set beyond the returned length.
```

这个和前面那个remove duplicate number是一个思路：

这个算法很简单，但是效率贼高。其实很好理解，就是当这个数字不等于val的时候，会用之前的curr（非重复的数字坐标）来替换当前的val。

```
3, 2, 2, 3 & val = 2
curr = 0

when i = 0, val != nums[i] num[curr++] = nums[i] -> nums[0] = nums[0] and curr = 1 now
when i = 1, val == nums[i] continue
when i = 2, val == nums[i] continue
when i = 3, val != nums[i] num[curr++] = nums[i] -> nums[1] = nums[3] and curr = 2 now

so nums = 3, 3, 2, 3 and curr = 2, we do not care index > 1 values,
so we done
```

```java
public int removeElement(int[] nums, int val) {
    int curr = 0;

    for (int i = 0; i < nums.length; i++) {
        if (val != nums[i]) {
            nums[curr++] = nums[i];
        }
    }

    return curr;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Implement strStr()

Implement strStr()

Return the index of the first occurrence of needle in haystack, or **-1** if needle is not part of haystack.

Example 1:

```
Input: haystack = "hello", needle = "ll"
Output: 2
```

Example 2:

```
Input: haystack = "aaaaa", needle = "bba"
Output: -1
```

返回要求的是**index**，我还以为是长度一度迷茫为啥不对**o(╥﹏╥)o**，一个for就能解决的问题，我们先去 `needle` 的第一个字符，然后在循环 `haystack` 的时候如果里面有字符和 `needle` 的第一个字符匹配了就用 `substring` 来接取相对应的长度，然后判断是否和 `needle` 相等（需要判断index的长度是不是超过了 `haystack` 的长度）。

```
public int strStr(String haystack, String needle) {
    if (needle.length() == 0) return 0;
    char c = needle.charAt(0);
    for (int i = 0; i < haystack.length(); i ++) {
        if (haystack.charAt(i) == c) {
            if (i + needle.length() > haystack.length()) return -1;
            if (haystack.substring(i, i+needle.length()).equals(needle)) return i;
        }
    }
    return -1;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Search in Rotated Sorted Array

Search in Rotated Sorted Array

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., `[0,1,2,4,5,6,7]` might become `[4,5,6,7,0,1,2]`).

You are given a target value to search. If found in the array return its index, otherwise return `-1` .

You may assume no duplicate exists in the array.

Your algorithm's runtime complexity must be in the order of $O(\log n)$.

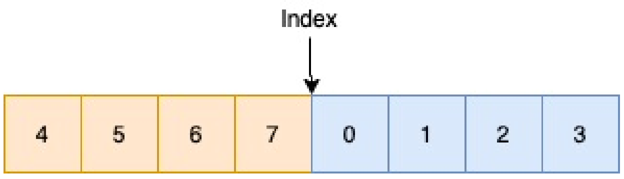
Example 1:

```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```

Example 2:

```
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```

思路，先把分割点找到，就是从哪个地方来做的rotated，比如 `[4,5,6,7,0,1,2]` 那么分割点就是 `index = 3` 的时候，因为是一个递增的过程，所以想找到分割点就很简单。再根据分割点做一个binary search，先分析前半段，如果能找到结果就return，如果找不到就找后半段的内容。



```java
public class SearchRotatedArray {
    public int search(int[] nums, int target) {
        if (nums.length == 0) return -1;
        int left = 0;
        int right = nums.length - 1;
        int num = nums[0];
        int mid = -1;

        for (int i = 1; i < nums.length; i ++) {
            if (num <= nums[i]) num = nums[i];
            else {
                mid = i - 1;
                break;
            }
        }

        if (mid == -1) mid = nums.length - 1;
        right = mid;
        int res = binary_search(left, right, nums, target);
        if (res == -1) {
            left = mid + 1; right = nums.length - 1;
            return binary_search(left, right, nums, target);
        } else {
            return res;
        }
    }

    public int binary_search(int left, int right, int[] nums, int target) {

        while (left <= right) {
            int temp_mid = left + (right - left) / 2;
            if (nums[temp_mid] == target) return temp_mid;
            else if (nums[temp_mid] > target) {
                right = temp_mid - 1;
            } else {
                left = temp_mid + 1;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] nums = new int[] {4,5,6,7,0,1,2};
        SearchRotatedArray s = new SearchRotatedArray();
        int x = s.search(nums, 0);
        System.out.println(x);
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Find First and Last Position of Element in Sorted Array

Find First and Last Position of Element in Sorted Array

Given an array of integers `nums` sorted in ascending order, find the starting and ending position of a given `target` value.

Your algorithm's runtime complexity must be in the order of *O*(log *n*).

If the target is not found in the array, return `[-1, -1]`.

Example 1:

```
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
```

Example 2:

```
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```

使用了一个~二分查找法~，头尾遍历法，不过貌似时间复杂度有点高。。没有别人那么快，花了1ms才通过。一会准备去看看大神的思路，看看有没有好的idea。第23-28行的主要作用是考虑到起始位置和结束位置在同一个地方，当其中有任何一个数字不是-1的时候，代表已经找到这个数字了，但是 `head==tail` 的时候就会结束循环了，所以会进行一个判断。

```
public int[] searchRange(int[] nums, int target) {
    if (nums.length == 0) return new int[] {-1, -1};
    if (nums.length == 1)return  (nums[0] == target) ? new int[] {0, 0}: new int[]{-1, -1
};

    int head = 0, tail = nums.length - 1;
    int[] range = new int[] {-1, -1};

    while (head <= tail) {
        if (nums[head] == target) {
            range[0] = head;
            if (range[1] != -1) break;
        }

        if (nums[tail] == target) {
            range[1] = tail;
            if (range[0] != -1) break;
        }

        if (nums[head] < target) head ++;
        if (nums[tail] > target) tail --;
    }

    if (head == tail) {
        if (range[0] == -1 || range[1] == -1) {
            range[0] = Math.max(range[0], range[1]);
            range[1] = Math.max(range[0], range[1]);
        }
    }

    return range;
}
```

刚去看了一下discussion里面，然后发现自己写的二分查找不叫二分查找，只能算是一个两头遍历的循环。然后根据二分查找的方法写了一份：

```java
public int[] searchRange(int[] nums, int target) {
    if (nums.length == 0) return new int[] {-1, -1};

    int head = 0, tail = nums.length - 1;
    int[] range = new int[] {-1, -1};

    while (head <= tail) {
        int mid = (head + tail) / 2;

        if (nums[mid] > target) {
            tail = mid - 1;
        } else if (nums[mid] < target) {
            head = mid + 1;
        } else {
            // make a loop from head to tail and tail to end
            while (head <= mid) {
                if (nums[head] == target) {
                    range[0] = head;
                    break;
                }
                head ++;
            }

            while (tail >= mid) {
                if (nums[tail] == target) {
                    range[1] = tail;
                    break;
                }
                tail --;
            }
            break;
        }
    }

    return range;
}
```

Search insert position

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Example 1:

```
Input: [1,3,5,6], 5
Output: 2
```

Example 2:

```
Input: [1,3,5,6], 2
Output: 1
```

就是找到合适的位置，然后返回相对应的index，没什么难度：

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

    for (int i = 0; i < nums.length; i ++) {
        if (nums[i] >= target) return i;
    }

    return nums.length;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

39. Combination Sum

Given a **set** of candidate numbers (`candidates`) **(without duplicates)** and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sums to `target` .

The **same** repeated number may be chosen from `candidates` unlimited number of times.

Note:

- All numbers (including `target`) will be positive integers.
- The solution set must not contain duplicate combinations.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7,
A solution set is:
[
  [7],
  [2,2,3]
]
```

Example 2:

```
Input: candidates = [2,3,5], target = 8,
A solution set is:
[
  [2,2,2,2],
  [2,3,3],
  [3,5]
]
```

这个可以使用到回溯算法的思想，从 `i - nums.length` ，每个都会重复，比如当 `i=0` 的时候：

```
2,2,2,2
2,2,2,3
2,2,2,6
2,2,2,7
2,2,3
2,2,6
2,2,7
2,2
2,3
....
```

这样的一个规律。

```java
import java.util.*;

public class CombinationSum {
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> res = new LinkedList<>();
        back_tracking(res, new ArrayList<>(), target, candidates, 0);
        return res;
    }

    // 回溯算法
    public void back_tracking(List<List<Integer>> res, List<Integer> temp, int target, int
[] nums, int index) {
        if (target < 0) return;
        else if (target == 0) res.add(new ArrayList<>(temp));
        else {
            for (int i = index; i < nums.length; i ++) {
                temp.add(nums[i]);
                back_tracking(res, temp, target - nums[i], nums, i);
                temp.remove(temp.size() - 1);
            }
        }
    }

    public static void main(String[] args) {
        CombinationSum cs = new CombinationSum();
        int[] nums = new int[] {2,3,6,7};
        List<List<Integer>> res = cs.combinationSum(nums, 7);

        for (List<Integer> l : res) {
            System.out.println(Arrays.toString(l.toArray()));
        }
    }
}
```

46. Permutations

Given a collection of **distinct** integers, return all possible permutations.

Example:

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

也是一个回溯算法，需要加一个额外的条件，不能重复，不过也是回溯算法的一个套路。伪代码：

```
result = []
def backtrack(path, select_list):
    if condition:
        result.add(path)
        return

    for list in select_list:
        make a select
        backtrack(path, select_list)
        drawback select
```

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;


public class Permutations {
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> res = new LinkedList<>();

        back_track(res, new ArrayList<>(), nums);
        return res;
    }


    public void back_track(List<List<Integer>> res, List<Integer> temp, int[] nums) {
        if (temp.size() == nums.length) {
            res.add(new ArrayList<>(temp));
            return;
        }

        for (int i = 0; i < nums.length; i ++) {
            // 避免重复项
            if (temp.contains(nums[i])) continue;
            temp.add(nums[i]);
            back_track(res, temp, nums);
            temp.remove(temp.size() - 1);
        }
    }

    public static void main(String[] args) {
        int[] nums = new int[] {1,2,3};

        Permutations p = new Permutations();
        List<List<Integer>> res = p.permute(nums);

        for (List<Integer> l : res) {
            System.out.println(Arrays.toString(l.toArray()));
        }
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

48. Rotate Image

You are given an *n* x *n* 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

Note:

You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

Example 1:

```
Given input matrix =
[
  [1,2,3],
  [4,5,6],
  [7,8,9]
],

rotate the input matrix in-place such that it becomes:
[
  [7,4,1],
  [8,5,2],
  [9,6,3]
]
```

emmm这个题我使用了另一个double array应该是不咋对

```java
import java.util.Arrays;

public class RotateImage {
    public void rotate(int[][] matrix) {
        int[][] res = new int[matrix.length][matrix[0].length];

        for (int i = 0; i < matrix[0].length; i ++) {
            int index = 0;
            for (int j = matrix.length - 1; j >= 0; j --) {
                res[i][index++] = matrix[j][i];
            }
        }

        for (int i = 0; i < res.length; i ++) {
            matrix[i] = res[i];
        }
    }

    public static void main(String[] args) {
        int[][] m = new int[][] {{1,2,3},{4,5,6},{7,8,9}};
        RotateImage ri = new RotateImage();
        ri.rotate(m);

        for (int[] ints : m) {
            System.out.println(Arrays.toString(ints));
        }
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Group Anagrams

Group Anagrams

Given an array of strings, group anagrams together.

Example:

```
Input: ["eat", "tea", "tan", "ate", "nat", "bat"],
Output:
[
  ["ate","eat","tea"],
  ["nat","tan"],
  ["bat"]
]
```

Note:

- All inputs will be in lowercase.
- The order of your output does not matter.

把使用同一个字符构成的string放到一起。首先确定是使用HashMap，在遍历数组的时候对每一个string先进行sort一下（或者用hash的方法，我下面有写），排过序的string当做key，如果还有相同的数据就接着添加，如果没有该key就初始化一个数组。

```java
import java.util.*;

public class GroupAnagrams {
    public List<List<String>> groupAnagrams(String[] strs) {
        List<List<String>> list = new LinkedList<>();
        HashMap<String, List<String>> hashMap = new HashMap<>();
        for (String str : strs) {
            hash(str);
            char[] temp_char = str.toCharArray();
            Arrays.sort(temp_char);
            String key = Arrays.toString(temp_char).toString();
            if (hashMap.containsKey(key)) {
                hashMap.get(key).add(str);
            } else {
                List<String> temp = new LinkedList<>();
                temp.add(str);
                hashMap.put(key, temp);
            }
        }

        for (String key : hashMap.keySet()) {
            list.add(hashMap.get(key));
        }
        return list;
    }

    public void hash(String str) {
        int hash = 31;
        for (int i = 0; i < str.length(); i ++) {
            hash += str.charAt(i);
        }
        System.out.println(hash);
    }


    public static void main(String[] args) {
        String[] strs = new String[] {"duy","ill"};
        GroupAnagrams ga = new GroupAnagrams();
        ga.groupAnagrams(strs);
    }
}
```

Pow(x, n)

Pow(x, n)

Implement pow(*x*, *n*), which calculates *x* raised to the power *n* (xn).

Example 1:

```
Input: 2.00000, 10
Output: 1024.00000
```

Example 2:

```
Input: 2.10000, 3
Output: 9.26100
```

Example 3:

```
Input: 2.00000, -2
Output: 0.25000
Explanation: 2-2 = 1/22 = 1/4 = 0.25
```

Note:

- -100.0 < *x* < 100.0
- *n* is a 32-bit signed integer, within the range [−231, 231 − 1]

这个就是简单的求一下平方，然后给保留5位小数就行了，用Python简单的写了一波

```
class Solution(object):
    def myPow(self, x, n):
        """
        :type x: float
        :type n: int
        :rtype: float
        """
        x = x**n

        x = 2**32-1 if x >= 2**32-1 else x
        x = -2**32 if x <= -2**32 else x

        return round(x, 5)
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Maximum Subarray

Maximum Subarray

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

Example:

```
Input: [-2,1,-3,4,-1,2,1,-5,4],
Output: 6
Explanation: [4,-1,2,1] has the largest sum = 6.
```

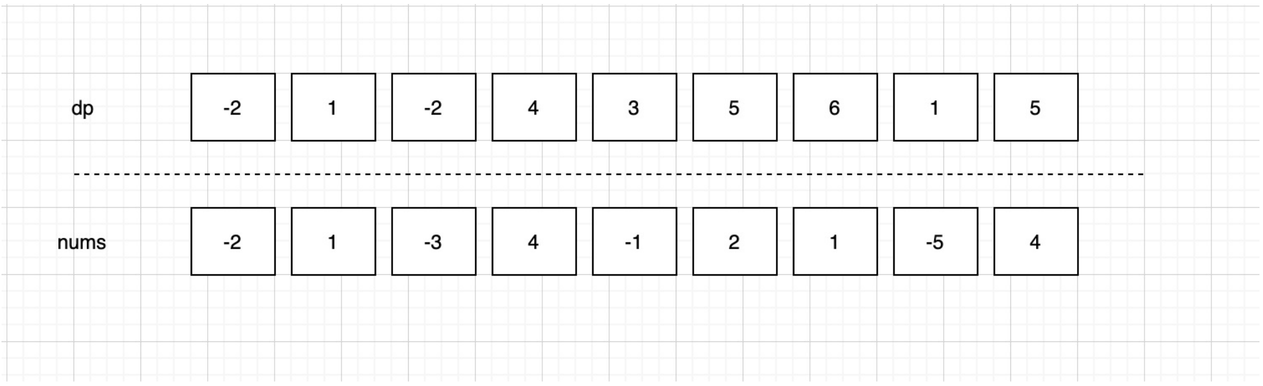
Follow up:

If you have figured out the O(*n*) solution, try coding another solution using the divide and conquer approach, which is more subtle.

题意解析：找到和最大的子序列。

算法解析：

可以使用动态规划来做。初始化一个数组长度和原数组一样，如果 `i-1` 的数字小于零就表示这就不用需要进行加和，因为加负数就是在减小，然后把 `i` 就行赋值。在每次循环的时候找到最大的值就行（ `max` , `dp[i]` 取最大的）。 `dp[i]` 存的结果：如果 `dp[i-1]` 小于零那就是本身，不然就是 `dp[i-1] + num[i]` 加上当前数字。



```
public int maxSubArray(int[] nums) {
    int max = nums[0];
    int[] dp = new int[nums.length];
    dp[0] = nums[0];

    for (int i = 1; i < nums.length; i ++) {
        if (dp[i-1] < 0) {
            dp[i] = nums[i];
        } else {
            dp[i] = dp[i-1] + nums[i];
        }
        max = Math.max(dp[i], max);
    }
    return max;
}
```

看了大佬的思路之后发现了新天地，可以节省一些内存空间，降低空间复杂度：

只需要两个空间的位置就行了，因为我们只需要 `i` 和 `i-1` 这两个参数。

```
public int maxSubArray(int[] nums) {
    int max = nums[0];
    int[] temp = new int[2];
    temp[0] = nums[0];

    for (int i = 1; i < nums.length; i ++) {
        if (temp[(i - 1) % 2] >= 0) {
            temp[i % 2] = temp[(i-1) % 2] + nums[i];
            max = Math.max(temp[i%2], max);
        } else {
            temp[i % 2] = nums[i];
        }
    }

    return max;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

58. Length of Last Word

Given a string *s* consists of upper/lower-case alphabets and empty space characters `' '`, return the length of last word (last word means the last appearing word if we loop from left to right) in the string.

If the last word does not exist, return 0.

Note: A word is defined as a **maximal substring** consisting of non-space characters only.

Example:

```
Input: "Hello World"
Output: 5
```

直接split然后取最后一个，然后直接计算length

```java
public class LengthofLastWord {
    public int lengthOfLastWord(String s) {
        if (s.equals("")) return 0;
        String[] res = s.split(" ");

        if (res.length == 0) return 0;
        return res[res.length - 1].length();
    }

    public static void main(String[] args) {
        String r = "Hello world";

        LengthofLastWord l = new LengthofLastWord();
        System.out.println(l.lengthOfLastWord(r));
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

61. Rotate List

Given a linked list, rotate the list to the right by *k* places, where *k* is non-negative.

Example 1:

```
Input: 1->2->3->4->5->NULL, k = 2
Output: 4->5->1->2->3->NULL
Explanation:
rotate 1 steps to the right: 5->1->2->3->4->NULL
rotate 2 steps to the right: 4->5->1->2->3->NULL
```

Example 2:

```
Input: 0->1->2->NULL, k = 4
Output: 2->0->1->NULL
Explanation:
rotate 1 steps to the right: 2->0->1->NULL
rotate 2 steps to the right: 1->2->0->NULL
rotate 3 steps to the right: 0->1->2->NULL
rotate 4 steps to the right: 2->0->1->NULL
```

做法稍微蠢了一点，我是通过先计算链表的长度，然后把 `k = k % length` 这样就可以得到相对链表来说的第几位。然后遍历链表，当节点的index等于k的时候开始把k之后的node放到链表的起始位置。这样就完成了rotate list。

```
public class RotateList {
    public static class ListNode {
        int val;
        ListNode next;
        ListNode() {}
        ListNode(int val) { this.val = val; }
        ListNode(int val, ListNode next) { this.val = val; this.next = next; }
    }

    public ListNode rotateRight(ListNode head, int k) {
        if (head == null || head.next == null || k == 0) return head;
        ListNode curr = head;

        int length = length(curr);
        k = length - (k % length);
        if(k == length) return head;

        ListNode rota = find_rota(curr, k);
        ListNode temp = rota;

        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = head;
        head = rota;


        return head;
    }

    public ListNode find_rota(ListNode curr, int k) {
        while (k > 1) {
            curr = curr.next;
            k --;
        }

        ListNode node = curr.next;
        curr.next = null;
        return node;

    }

    public int length(ListNode node) {
        int length = 0;
        while (node != null) {
            node = node.next;
            length ++;
        }

        return length;
    }
    public static void main(String[] args) {
        ListNode l1 = new ListNode(1);
        ListNode l2 = new ListNode(2);
        ListNode l3 = new ListNode(3);
//        ListNode l4 = new ListNode(4);
//        ListNode l5 = new ListNode(5);

        l1.next = l2;
        l2.next = l3;
//        l3.next = l4;
//        l4.next = l5;

        RotateList r = new RotateList();
        r.rotateRight(l1, 2);
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Plus One

Given a **non-empty** array of digits representing a non-negative integer, plus one to the integer.

The digits are stored such that the most significant digit is at the head of the list, and each element in the array contain a single digit.

You may assume the integer does not contain any leading zero, except the number 0 itself.

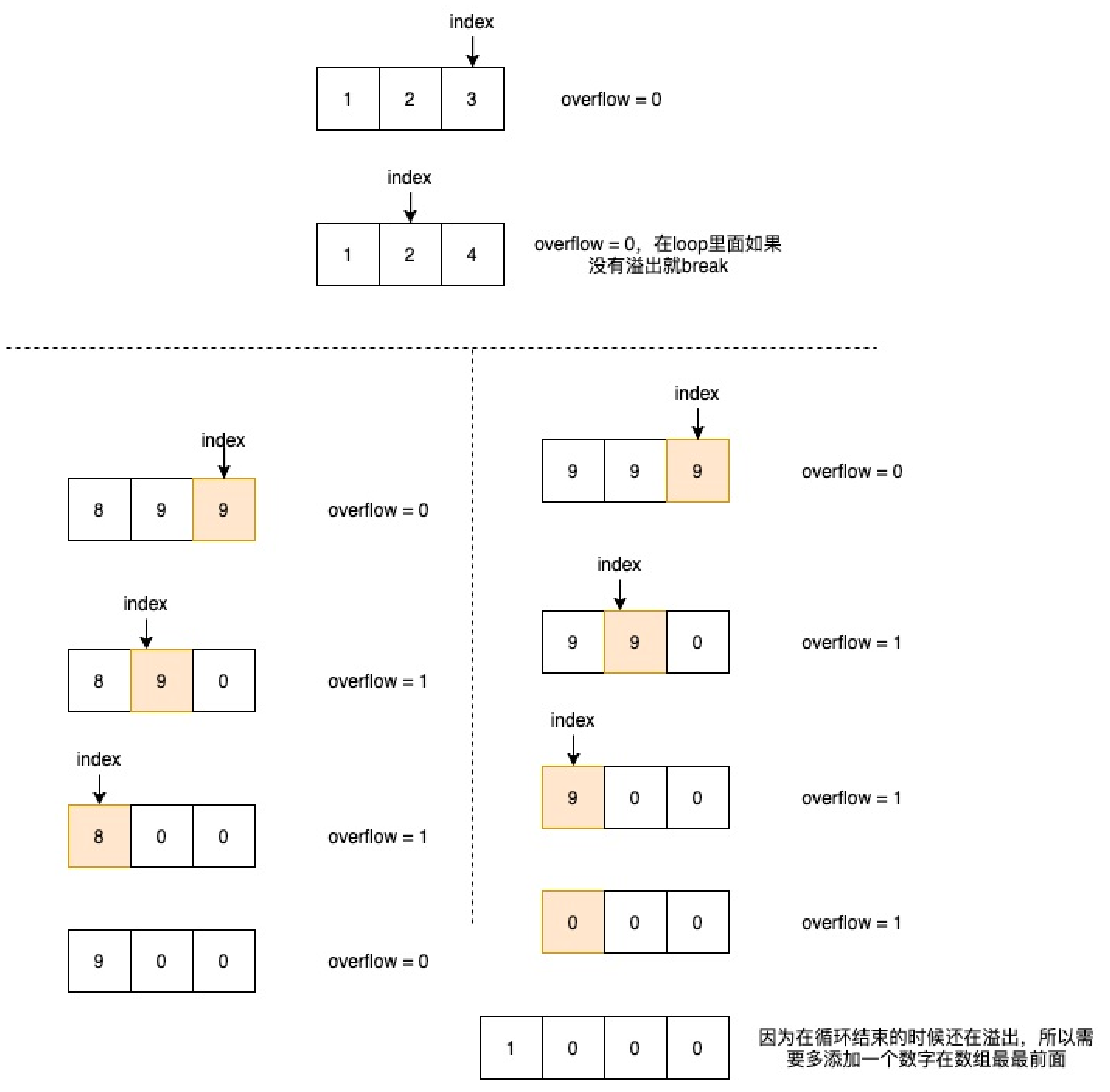
Example 1:

```
Input: [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
```

Example 2:

```
Input: [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321.
```

一个数学的问题，在最后面加1，然后如果大于10，那么前面的数字就需要加一。可以直接不用分类来讨论，我们可以设置一个循环 `index <= 0` ， `index = nums.length - 1` 这样我们从最后面开始加1， `overflow` 是用来判断是否溢出，在循环里面如果没有溢出，就表示前面数字加一的结果不大于10，所以可以直接break。还有一种如果循环结束了依然溢出了，这时候我们需要在array里面多加一个数字在最前面。


index
1 2 3
overflow = 0
index
1 2 4
overflow = 0，在loop里面如果没有溢出就break
index
8 9 9
overflow = 0
index
8 9 0
overflow = 1
index
8 0 0
overflow = 1
9 0 0
overflow = 0
index
9 9 9
overflow = 0
index
9 9 0
overflow = 1
index
9 0 0
overflow = 1
0 0 0
overflow = 1
1 0 0 0
因为在循环结束的时候还在溢出，所以需要多添加一个数字在数组最最前面

```
import java.util.Arrays;
import java.util.stream.IntStream;

public class PlusOne {
    public int[] plusOne(int[] digits) {
        int lens = digits.length - 1;
        int[] nums = digits.clone();
        int overflow = 0;
        nums[lens] += 1;

        while (lens != -1) {
            // every time plus one
            nums[lens] += overflow;
            overflow = nums[lens] / 10;
            if (nums[lens] > 9) nums[lens] %= 10;
            lens --;
            // 当不溢出的时候就可以break了，因为不会进位了
            if (overflow == 0) break;
        }

        // 如果循环结束还溢出，说明还需要多开辟一个内存位置。
        if (overflow != 0) {
            nums = IntStream.concat(Arrays.stream(new int[] {1}), Arrays.stream(nums)).to
Array();
        }
        return nums;
    }

    public static void main(String[] args) {
        int[] nums = new int[] {8,9};
        PlusOne p = new PlusOne();
        int[] res = p.plusOne(nums);
        System.out.println(Arrays.toString(res));
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Climbing Stairs

Climbing Stairs

You are climbing a stair case. It takes *n* steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Note: Given *n* will be a positive integer.

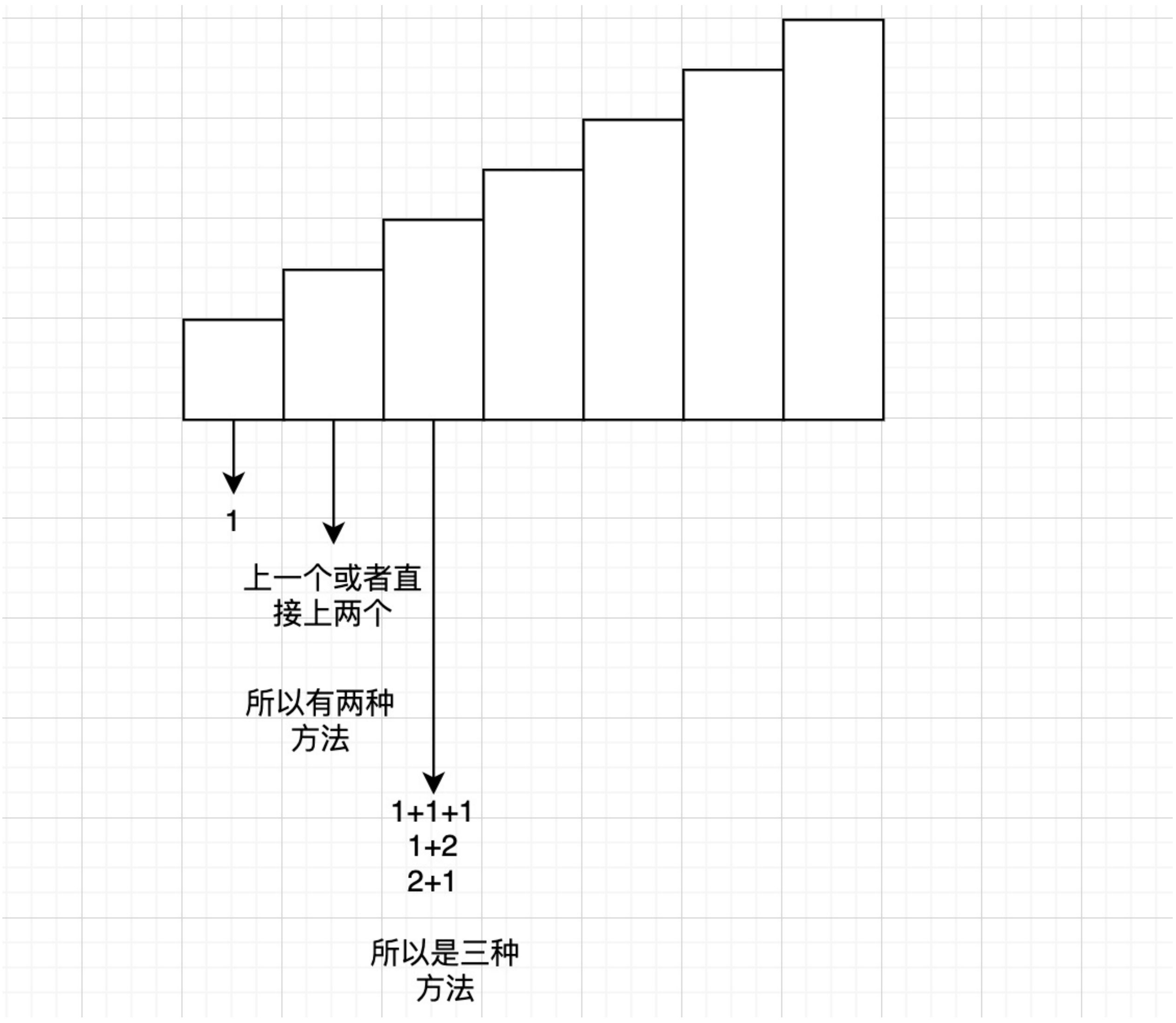
Example 1:

```
Input: 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
```

Example 2:

```
Input: 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
```

这是一个很经典的动态规划的问题。一次可以走一阶楼梯或者两阶楼梯。在第一层的时候肯定只能上一阶，第二层的时候可以是1+1（连续走两个一阶）或者2（直接两阶）所以是两种方法，第三层的时候可以是1+1+1, 1+2, 2+1三种方法。所以可以看出$f(n)=f(n-1)+f(n-2)$的算法。



```java
public int climbStairs(int n) {
    if (n == 1) return 1;
    if (n == 2) return 2;
    int[] array = new int[n];
    array[0] = 1;
    array[1] = 2;

    for (int i = 2; i < n; i ++ ){
        array[i] = array[i-1] + array[i-2];
    }

    return array[n-1];
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

80. Remove Duplicates from Sorted Array II

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

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the **first part** of the array `nums`. More formally, if there are `k` elements after removing the duplicates, then the first `k` elements of `nums` should hold the final result. It does not matter what you leave beyond the first `k` elements.

Return `k` *after placing the final result in the first* `k` *slots of* `nums`.

Do **not** allocate extra space for another array. You must do this by **modifying the input array in-place** with O(1) extra memory.

Example 1:

```
Input: nums = [1,1,1,2,2,3]
Output: 5, nums = [1,1,2,2,3,_]
Explanation: Your function should return k = 5, with the first five elements of nums bein
g 1, 1, 2, 2 and 3 respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).
```

Example 2:

```
Input: nums = [0,0,1,1,1,1,2,3,3]
Output: 7, nums = [0,0,1,1,2,3,3,_,_]
Explanation: Your function should return k = 7, with the first seven elements of nums bei
ng 0, 0, 1, 1, 2, 3 and 3 respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).
```

Solution

一开始理解错了，以为只要返回数量就行，后面发现还需要额外的修改内存数据。其实这个题很简单，只需要用到双指针就行，或者快慢指针的方法，意思都是差不多的。 `idx` 指针负责去遍历 `nums` 数据， `s_idx` 负责记录更新开始的坐标。

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        int idx = 0, s_idx = 0, val = 0, cnt = 0;

        while (idx < nums.size()) {
            val = nums[idx];
            cnt = 0;

            int j = idx;
            while (j < nums.size() && nums[j] == val) {
                cnt ++;
                if (cnt <= 2) nums[s_idx++] = nums[j];
                j ++;
            }

            idx = j;
        }

        return s_idx;
    }
};
```

powered by GitbookFile Modify: 2024-11-27 04:43:10

82. Remove Duplicates from Sorted List II

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only *distinct* numbers from the original list.

Return the linked list sorted as well.

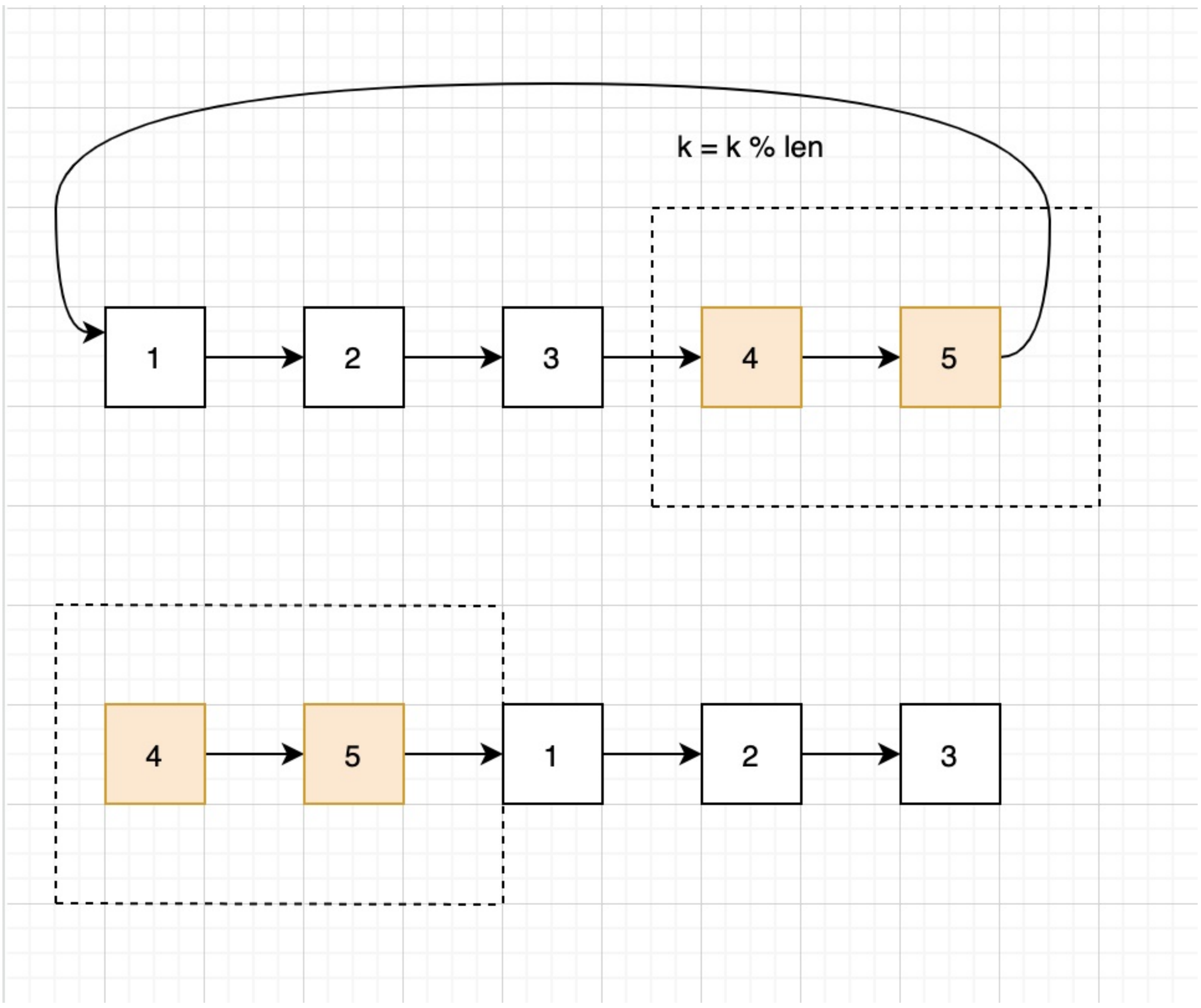
Example 1:

```
Input: 1->2->3->3->4->4->5
Output: 1->2->5
```

Example 2:

```
Input: 1->1->1->2->3
Output: 2->3
```

这个第83的进阶版。只要是有重复的node都需要删除掉。我们需要有一个node（last）来记录上一个访问的node，一个node（curr）来循环。如果 `curr.val == curr.next.val` 就代表着有重复的node了，这时候找到所有重复的node，然后得到最后一个重复node的next，然后使用last链接就行，但是last需要判断是否是null，如果是null的话需要考虑到head的情况。



```
public class RemoveDuplicatesfromSortedListII {
    public static class ListNode {
        int val;
        ListNode next;
        ListNode() {}
        ListNode(int val) { this.val = val; }
        ListNode(int val, ListNode next) { this.val = val; this.next = next; }
    }

    public ListNode deleteDuplicates(ListNode head) {
        if (head == null || head.next == null) return head;

        ListNode last = null;
        ListNode curr = head;

        while (curr.next != null) {
            if (curr.val == curr.next.val) {
                ListNode node = different(curr);
                if (last == null) {
                    head = node;
                } else {
                    last.next = node;
                }
                curr = node;
                if (curr == null) break;
                continue;
            } else {
                last = curr;
            }
            curr = curr.next;
        }

        return head;
    }

    public ListNode different(ListNode node) {
        ListNode last = node;
        while (node.next != null) {
            if (node.val != node.next.val) {
                return node.next;
            }
            last = node;
            node = node.next;
        }
        // 因为是node.next != null 所以需要判断最后一个node是不是和前面的相等
        if (last.val == node.val) return null;
        return node;
    }

    public static void main(String[] args) {
        ListNode l1 = new ListNode(1);
        ListNode l2 = new ListNode(2);
        ListNode l3 = new ListNode(2);
        ListNode l4 = new ListNode(2);
//        ListNode l5 = new ListNode(5);

        l1.next = l2;
        l2.next = l3;
        l3.next = l4;
//        l4.next = l5;

        RemoveDuplicatesfromSortedListII r = new RemoveDuplicatesfromSortedListII();
        r.deleteDuplicates(l1);
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

83. Remove Duplicates from Sorted List

Given a sorted linked list, delete all duplicates such that each element appear only *once*.

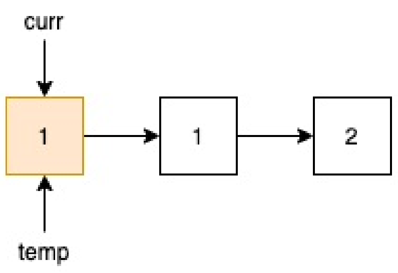
Example 1:

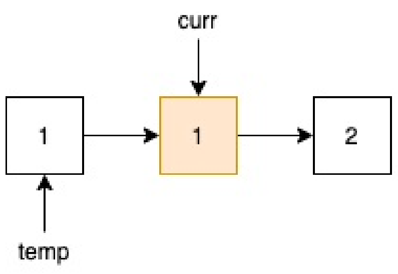
```
Input: 1->1->2
Output: 1->2
```

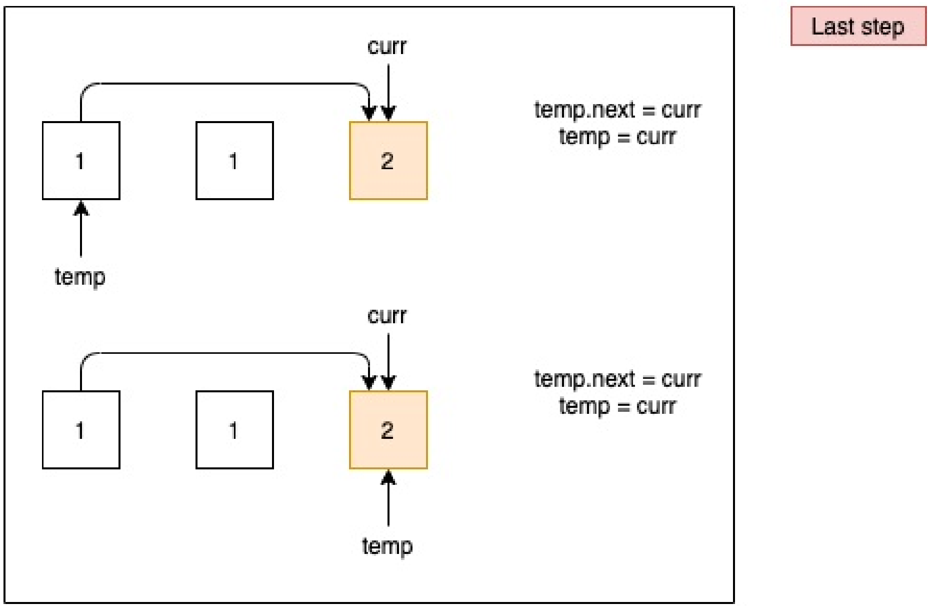
Example 2:

```
Input: 1->1->2->3->3
Output: 1->2->3
```

在已经sorted的链表里面去除重复的值，可以使用连个指针来完成，curr来前进，temp来链接不一样的值。当 `temp.val != curr.val` 的时候代表中间已经省略了同类项，所以直接设置 `temp.next = curr` 然后把curr赋值给temp（ `temp = curr` ）。


curr
1
1
2
temp


curr
1
1
2
temp


Last step
curr
1
1
2
temp.next = curr
temp = curr
temp
curr
1
1
2
temp.next = curr
temp = curr
temp

```
public class RemoveDuplicatesfromSortedList {
    public static class ListNode {
        int val;
        ListNode next;
        ListNode() {}
        ListNode(int val) { this.val = val; }
        ListNode(int val, ListNode next) { this.val = val; this.next = next; }
    }

    public ListNode deleteDuplicates(ListNode head) {
        if (head == null || head.next == null) return head;

        ListNode curr = head.next;
        ListNode temp = head;

        while (curr != null) {
            if (temp.val != curr.val) {
                temp.next = curr;
                temp = curr;
            }
            curr = curr.next;
        }
        temp.next = null;

        return head;
    }


    public static void main(String[] args) {
        ListNode l1 = new ListNode(1);
        ListNode l2 = new ListNode(1);
        ListNode l3 = new ListNode(2);

        l1.next = l2;
        l2.next = l3;
        l3.next = null;

        RemoveDuplicatesfromSortedList re = new RemoveDuplicatesfromSortedList();
        ListNode head = re.deleteDuplicates(l1);

        while (head != null) {
            System.out.println(head.val);
            head = head.next;
        }
    }

}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Merge Sorted Array

Merge Sorted Array

Given two sorted integer arrays *nums1* and *nums2*, merge *nums2* into *nums1* as one sorted array.

Note:

- The number of elements initialized in *nums1* and *nums2* are *m* and *n* respectively.
- You may assume that *nums1* has enough space (size that is greater or equal to *m* + *n*) to hold additional elements from *nums2*.

Example:

```
Input:
nums1 = [1,2,3,0,0,0], m = 3
nums2 = [2,5,6],       n = 3

Output: [1,2,2,3,5,6]
```

这是我今天逛牛客的时候发现的一道题，然后想了一下，按照我现在的思想好像只知道直接merge然后sort。还有一种就是正序插入，但是每个element都需要后移一位，但是这样的效率太低了，看了网上的大神的idea之后就发现了新大陆。因为这个list是拍过序的，那么我们就知道了最后一个元素肯定是最大。那么我们为什么不从后面往前面循环呢，这样不需要元素以为，只需要直接赋值就好了。

```
public void merge(int[] nums1, int m, int[] nums2, int n) {
    // the last position of array
    int last = nums1.length - 1;
    m--; n--;
    while (m > 0 && n > 0) {
        if (nums1[m] > nums2[n]) nums1[last--] = nums1[m--];
        else nums1[last--] = nums2[n--];
    }

    while (m > 0) nums1[last--] = nums1[m--];
    while (n > 0) nums1[last--] = nums2[n--];
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

92. Reverse Linked List II

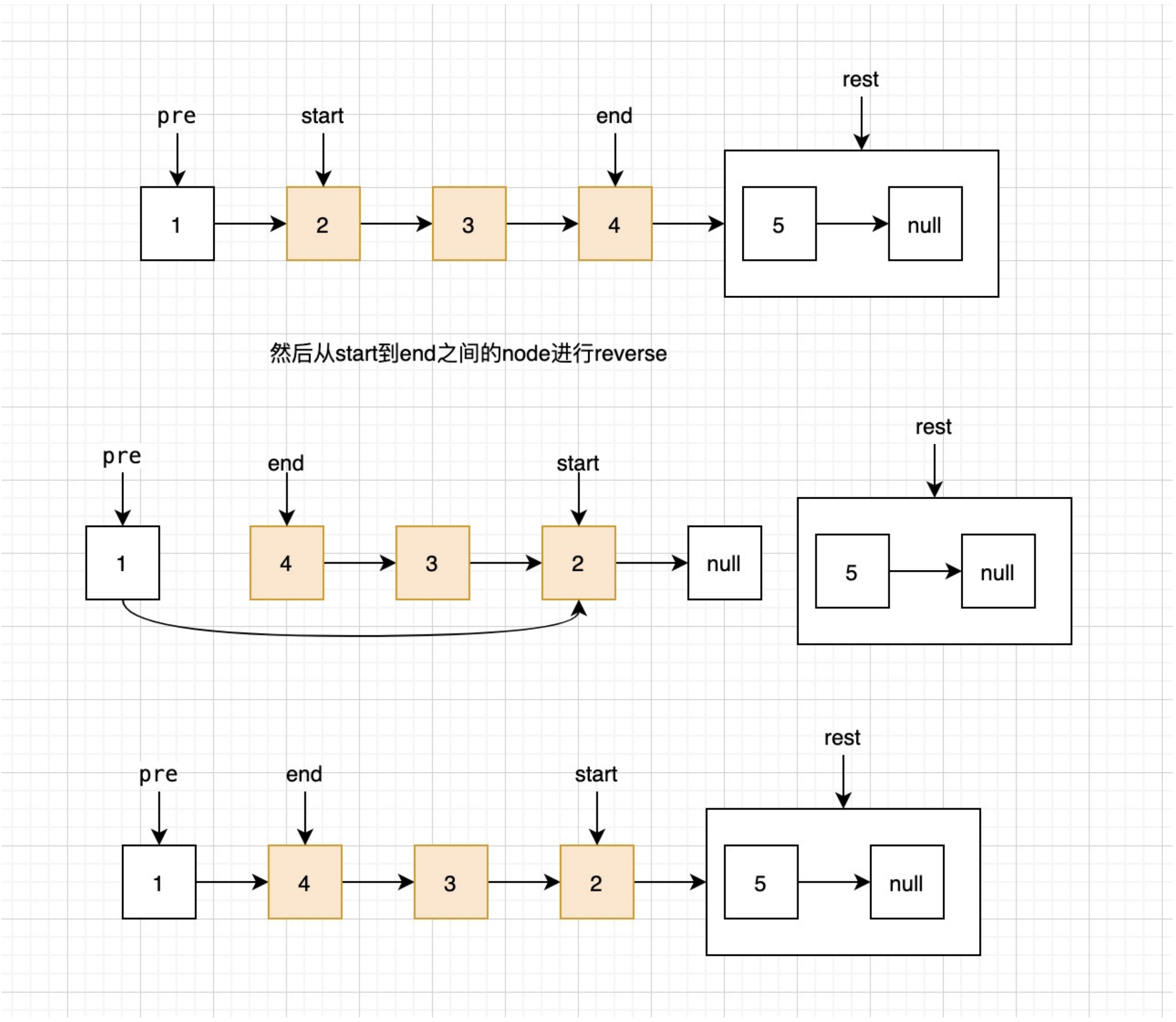
Reverse a linked list from position *m* to *n*. Do it in one-pass.

Note: 1 ≤ *m* ≤ *n* ≤ length of list.

Example:

```
Input: 1->2->3->4->5->NULL, m = 2, n = 4
Output: 1->4->3->2->5->NULL
```

从起始点到结束点进行reverse。首先根据 `curr` 的index遍历到 `m` 的时候记录一下该node `start` ，然后同时记录start前面的一个node `pre` 因为reverse结束还要把链表链接起来。当index遍历到 `n` 的时候记录该node `end` 。然后使用递归进行reverse。



```java
public class ReverseLinkedListII {
    public static class ListNode {
        int val;
        ListNode next;
        ListNode() {}
        ListNode(int val) { this.val = val; }
        ListNode(int val, ListNode next) { this.val = val; this.next = next; }
    }

    public ListNode reverseBetween(ListNode head, int m, int n) {
        if (m >= n || head == null || head.next == null) return head;

        int index = 1;
        ListNode curr = head;
        ListNode start = null;
        ListNode end = null;
        ListNode pre = null;

        while (curr != null) {
            if (index == m) start = curr;
            if (index == n) end = curr;
            if (start == null) pre = curr;
            curr = curr.next;
            ++index;
        }

        if (end == null || start == null) return head;

        ListNode rest = end.next;
        end.next = null;
        reverse(start, start.next);
        // 现在endu应该是head

        if (pre == null) head = end;
        else pre.next = end;

        start.next = rest;

        return head;
    }

    public void reverse(ListNode curr, ListNode next) {
        if (next == null) return;

        reverse(next, next.next);
        next.next = curr;
        curr.next = null;
    }

    public static void main(String[] args) {
        ListNode l1 = new ListNode(1);
        ListNode l2 = new ListNode(2);
        ListNode l3 = new ListNode(3);
        ListNode l4 = new ListNode(4);
        l1.next = l2;
        l2.next = l3;
        l3.next = l4;
        ReverseLinkedListII r = new ReverseLinkedListII();
        r.reverseBetween(l1, 1, 4);
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

100. Same Tree

Given two binary trees, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical and the nodes have the same value.

Example 1:

```
Input:     1         1
          / \       / \
         2   3     2   3

        [1,2,3],   [1,2,3]

Output: true
```

Example 2:

```
Input:     1         1
          /           \
         2             2

        [1,2],     [1,null,2]

Output: false
```

Example 3:

```
Input:     1         1
          / \       / \
         2   1     1   2

        [1,2,1],   [1,1,2]

Output: false
```

思路就是左右两边的node都一样。

```
public class SameTree {
    public boolean isSameTree(TreeNode p, TreeNode q) {
        if (p == null && q == null) return true;
        if (p == null || q == null) return false;

        boolean left = isSameTree(p.left, q.left);
        boolean right = isSameTree(p.right, q.right);

        return p.val == q.val && (left && right);
    }
}
```

101. Symmetric Tree

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

For example, this binary tree `[1,2,2,3,4,4,3]` is symmetric:

```
    1
   / \
  2   2
 / \ / \
3  4 4  3
```

But the following `[1,2,2,null,3,null,3]` is not:

```
    1
   / \
  2   2
   \   \
   3    3
```

其实是和上面那道题一个相反的思路，上面是左右相等，这个是left = right, right = left。

```
public class SymmetricTree {
    public boolean isSymmetric(TreeNode root) {
        if (root == null) return true;
        return mirror(root.left, root.right);
    }
    // 判断左右是否一样
    public boolean mirror(TreeNode p, TreeNode q) {
        if (p == null && q == null) return true;
        if (p == null || q == null) return false;

        boolean left = mirror(p.left, q.right);
        boolean right = mirror(p.right, q.left);

        return p.val == q.val && (left && right);
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

111. Minimum Depth of Binary Tree

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

Note: A leaf is a node with no children.

Example:

Given binary tree `[3,9,20,null,null,15,7]`,

```
    3
   / \
  9  20
    /  \
   15   7
```

return its minimum depth = 2.

关于tree的算法可以去看我另一篇文章https://shunyangli.github.io/2020/05/03/Algorithm/#more

```
public class MinimumDepthofBinaryTree {
    public int minDepth(TreeNode root) {
        if (root == null) return 0;
        if (root.left == null && root.right == null) return 1;

        int left = minDepth(root.left);
        int right = minDepth(root.right);

        if (root.left == null) return 1 + right;
        if (root.right == null) return 1 + left;

        return Math.min(left, right) + 1;
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

112. Path Sum

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

Note: A leaf is a node with no children.

Example:

Given the below binary tree and `sum = 22` ,

```
      5
     / \
    4   8
   /   / \
  11  13  4
 /  \      \
7    2      1
```

return true, as there exist a root-to-leaf path `5->4->11->2` which sum is 22.

```java
public class PathSum {
    public boolean hasPathSum(TreeNode root, int sum) {
        if (root == null) return false;
        if (root.left == null && root.right == null) return root.val == sum;

        boolean left = hasPathSum(root.left, sum - root.val);
        boolean right = hasPathSum(root.right, sum - root.val);

        return left | right;
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Best Time to Buy and Sell Stock

Best Time to Buy and Sell Stock

Say you have an array for which the *i*th element is the price of a given stock on day *i*.

If you were only permitted to complete at most one transaction (i.e., buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Note that you cannot sell a stock before you buy one.

Example 1:

```
Input: [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
             Not 7-1 = 6, as selling price needs to be larger than buying price.
```

Example 2:

```
Input: [7,6,4,3,1]
Output: 0
Explanation: In this case, no transaction is done, i.e. max profit = 0.
```

这个题的思路稍微简单一点，可以使用动态规划来找到最小的买入价格，然后我们就可以根绝在买入价格最低的日期之后，找到卖出价格最高的点，这样收益就是最大的。

```
public int maxProfit(int[] prices) {
    if (prices.length < 2) return 0;
    int max = 0, stock = prices[0];

    for (int price : prices) {
        int receive = price - stock;
          // 找到最大收入
        if (receive >= 0) max = Math.max(max, price-stock);
                // 找到最新的收入
        stock = Math.min(stock, price);
    }

    return max;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

125. Valid Palindrome

Valid Palindrome: Given a string s, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

```
Input: s = "A man, a plan, a canal: Panama"
Output: true
Explanation: "amanaplanacanalpanama" is a palindrome.
```

This one is pretty easy, we just need to ignore the char which is not belong to `a-z, 0-9`. During the processing, we can use double pointer to do that. Set a index from the left and a index from the right.

```python
class Solution:
    def isPalindrome(self, s: str) -> bool:
        s = s.lower()
        start = 0
        end = len(s) - 1

        while start <= end:
            l = s[start]
            r = s[end]

            if not l.isalnum():
                start += 1
                continue

            if not r.isalnum():
                end -= 1
                continue

            if s[start] != s[end]:
                return False

            start += 1
            end -= 1

        return True
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

126. Word Ladder

Given two words (*beginWord* and *endWord*), and a dictionary's word list, find the length of shortest transformation sequence from *beginWord* to *endWord*, such that:

1. Only one letter can be changed at a time.
2. Each transformed word must exist in the word list.

Note:

- Return 0 if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.
- You may assume no duplicates in the word list.
- You may assume *beginWord* and *endWord* are non-empty and are not the same.

Example 1:

```
Input:
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]

Output: 5

Explanation: As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",
return its length 5.
```

词语接龙，一个老外很喜欢的游戏，我觉得他们的test有问题。。。这个数量就很奇怪，不过我还是按照我的方法来，我不觉得我写错了，使用了一个bfs的算法，因为每次改变一个单词其中的一个字符，然后判断新生成的字符是否在wordList之间，然后再判断是不是已经visited过的。

```java
import java.util.*;

public class WordLadder {
    public int ladderLength(String beginWord, String endWord, List<String> wordList) {
        int step = 0;
        if (!wordList.contains(endWord)) return step;
        Queue<String> queue = new LinkedList<>();
        queue.offer(beginWord);
        Set<String> visited = new HashSet<>();

        while (!queue.isEmpty()) {
            ++step;
            for (int x = 0; x < queue.size(); x ++) {
                String str = queue.poll();
                visited.add(str);
                if (str.equals(endWord)) return step;
                for (int i = 0; i < str.length(); i ++) {
                    char[] temp = str.toCharArray();
                    char ch = temp[i];
                    for (char c = 'a'; c <= 'z'; c ++) {
                        if (c == ch) continue;
                        temp[i] = c;
                        String t = new String(temp);
                        if (!wordList.contains(t)) continue;
                        if (visited.contains(t)) continue;
                        queue.offer(t);
                        visited.add(t);
                    }
                }
            }
        }
        return step;
    }


    public static void main(String[] args) {
        WordLadder w = new WordLadder();
        List<String> list = new ArrayList<String>(
                Arrays.asList("hot","dot","dog","lot","log","cog")
        );

        int res = w.ladderLength("hit", "cog", list);

        System.out.println(res);

    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Single Number

Given a **non-empty** array of integers, every element appears *twice* except for one. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

Example 1:

```
Input: [2,2,1]
Output: 1
```

Example 2:

```
Input: [4,1,2,1,2]
Output: 4
```

庆幸一下终于可以独立思考出来一些简单的算法了。这个题其实算是还OK的，可以使用HashMap解决，因为HashMap监测key的时候的时间复杂度是$O(1)$，所以整体时间是$O(n)$。在loop里面去判断该数字是否存在HashMap里面，如果不存在就push进去，如果已经存在就删除。那么最后剩下的那个元素肯定是single（~~单身狗~~）。

```
public int singleNumber (int[] A) {
    // write code here
    Map<Integer, Integer> map = new HashMap<>();
    for (int value : A) {
        if (map.containsKey(value)) {
            map.remove(value);
        } else {
            map.put(value, value);
        }
    }
    return map.get(map.keySet().toArray()[0]);
}
```

Ps: 在discuss里面看到了一个特别骚的操作。。。真的骚操作。。。

使用了异或（exclusive OR简称xor）这个来判断的，异或的话空间复杂度只有$O(1)$。

> $1\oplus0=1$
>
> $1\oplus1=1$
>
> $0\oplus0=0$
>
> $0\oplus1=1$

```
public int singleNumber(int[] nums) {
    int res = 0;
    for (int i = 0; i < nums.length; i++) {
        res ^= nums[i];
    }
    return res;
}
```

行吧。。。牛逼。。

powered by GitbookFile Modify: 2024-11-27 04:38:02

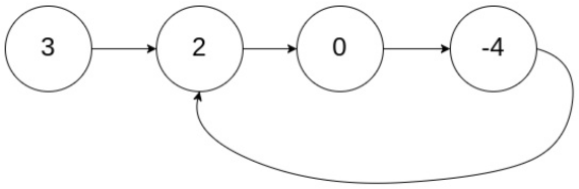
Linked List Cycle

Given a linked list, determine if it has a cycle in it.

To represent a cycle in the given linked list, we use an integer `pos` which represents the position (0-indexed) in the linked list where tail connects to. If `pos` is `-1`, then there is no cycle in the linked list.

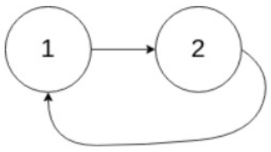
Example 1:

```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where tail connects to
the second node.
```



Example 2:

```
Input: head = [1,2], pos = 0
Output: true
Explanation: There is a cycle in the linked list, where tail connects to
the first node.
```



Example 3:

```
Input: head = [1], pos = -1
Output: false
Explanation: There is no cycle in the linked list.
```

算法思路：

一开始想到的是用HashMap的方式来解决这个问题，但是这样的话空间复杂度就高了很多就是$O(n)$了，然后看了一下大神们的解法，就是利用快慢指针的方法，两种我都试了一下：

1. HashMap的方法

```
public boolean hasCycle(ListNode head) {
    Map<ListNode, Integer> map = new HashMap<>();

    ListNode curr = head;

    while (curr != null) {
        if (map.containsKey(curr)) return true;
        else map.put(curr, 1);

        curr = curr.next;
    }

    return false;
}
```

1. 快慢指针的方法

   这样效率高，内存占用少

```
public boolean hasCycle(ListNode head) {
    ListNode slow = head;
    ListNode fast = head;

    while (fast != null && fast.next != null) {
        fast = fast.next.next;
        slow = slow.next;

        if (slow == fast) return true;
    }
    return false;
}
```

142. Linked List Cycle II

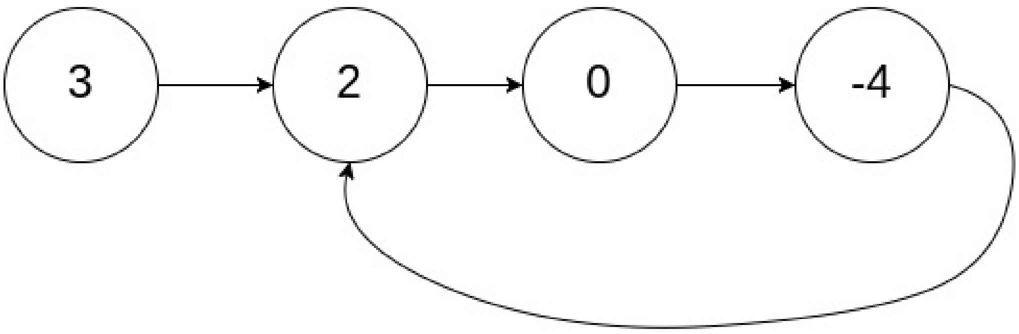
Given a linked list, return the node where the cycle begins. If there is no cycle, return `null`.

To represent a cycle in the given linked list, we use an integer `pos` which represents the position (0-indexed) in the linked list where tail connects to. If `pos` is `-1`, then there is no cycle in the linked list.

Note: Do not modify the linked list.

Example 1:

```
Input: head = [3,2,0,-4], pos = 1
Output: tail connects to node index 1
Explanation: There is a cycle in the linked list, where tail connects to the second node.
```



首先想到的就是HashMap哈哈哈哈，然后他要求说不用额外的空间，在不实用额外的空间的情况下想到了递归+循环的方法，但是时间复杂度比较高。在循环的过程中每个node都进行一个检查环的操作，并且判断在环里面是不是等于该数字，如果不等于的话就代表不是环的一个node，当出现第一个node并且是在环里面的node就代表是环的第一个节点。感觉牺牲了时间换来了空间的优化。。。。。

```
import java.util.HashMap;

public class LinkedListCycleII {
    public static class ListNode {
        int val;
        ListNode next;
        ListNode() {}
        ListNode(int val) { this.val = val; }
        ListNode(int val, ListNode next) { this.val = val; this.next = next; }
    }

    // hashmap的解决方法
//    public ListNode detectCycle(ListNode head) {
//        HashMap<ListNode, Integer> map = new HashMap<>();
//
//        ListNode curr = head;
//        while (curr != null) {
//            if (map.containsKey(curr)) {
//                return curr;
//            } else {
```

```
//                map.put(curr, 1);
//            }
//            curr = curr.next;
//        }
//
//        return null;
//    }

    public ListNode detectCycle(ListNode head) {
        if (head == null || head.next == null) return null;
        ListNode curr = head;

        while (curr.next != null) {
            ListNode temp = loop(curr.next, curr.next.next, curr);
            if (temp != null) return curr;
            curr = curr.next;
        }

        return null;
    }

    public ListNode loop(ListNode slow, ListNode fast, ListNode node) {
        if (slow == null || fast == null || slow.next == null || fast.next == null
                || fast.next.next == null) return null;
        if (slow == fast && slow == node) return slow;
        if (slow == fast) return null;
        return loop(slow.next, fast.next.next, node);
    }


    public static void main(String[] args) {
        ListNode l1 = new ListNode(1);
        ListNode l2 = new ListNode(2);
        ListNode l3 = new ListNode(3);
        ListNode l4 = new ListNode(4);

        l1.next = l2;
        l2.next = l3;
        l3.next = l4;
//        l4.next = l2;

        LinkedListCycleII l = new LinkedListCycleII();
        l.detectCycle(l1);
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

144. Binary Tree Preorder Traversal

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

Example:

```
Input: [1,null,2,3]
   1
    \
     2
    /
   3

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

Follow up: Recursive solution is trivial, could you do it iteratively?

就是一个前序的遍历：根左右

```
import java.util.LinkedList;
import java.util.List;

public class BinaryTreePreorderTraversal {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> res = new LinkedList<>();

        traversal(root, res);
        return res;

    }

    public void traversal(TreeNode root, List<Integer> res) {
        if (root == null) return;

        res.add(root.val);
        traversal(root.left, res);
        traversal(root.right, res);
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Min Stack

Min Stack(https://leetcode.com/problems/min-stack/)

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- getMin() -- Retrieve the minimum element in the stack.

Example 1:

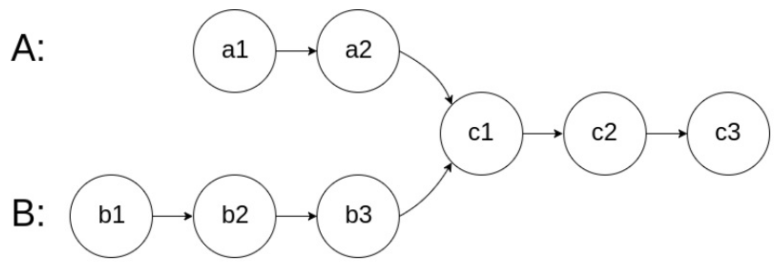
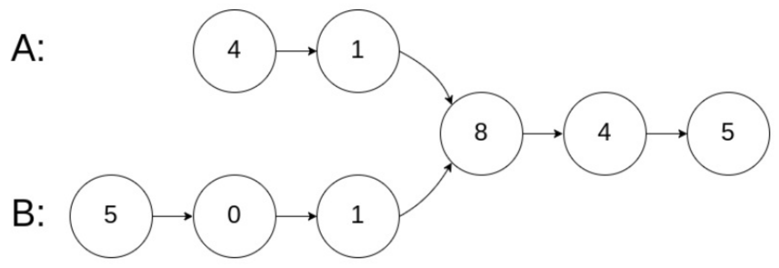
```
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output
[null,null,null,null,-3,null,0,-2]

Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top();    // return 0
minStack.getMin(); // return -2
```

这个只需要知道stack是先进后出的原则就行。

```java
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;

public class MinStack {
    /** initialize your data structure here. */
    private List<Integer> list;
    private int lens;

    public MinStack() {
        this.list = new LinkedList<>();
        this.lens = 0;
    }

    public void push(int x) {
        this.list.add(x);
        this.lens ++;
    }

    public void pop() {
        if (this.list.size() > 0) {
            this.lens --;
            this.list.remove(this.lens);
        }
    }

    public int top() {
        if (this.list.size() > 0) {
            return this.list.get(this.lens - 1);
        }
        return -1;
    }

    public int getMin() {
        return Collections.min(this.list);
    }

    public static void main(String[] args) {
        MinStack m = new MinStack();
        m.push(-2);
        m.push(0);
        m.push(-3);
        System.out.println(m.getMin());
        m.pop();
        System.out.println(m.top());   // return 0
        System.out.println(m.getMin()); // return -2
    }
}
```

Intersection of Two Linked Lists

Intersection of Two Linked Lists

> Write a program to find the node at which the intersection of two singly linked lists begins.
>
> For example, the following two linked lists:
>
> 
>
>
> begin to intersect at node c1.
>
> **Example 1:**
>
> 
>
>
> ```
> Input: intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA =
> 2, skipB = 3
> Output: Reference of the node with value = 8
> Input Explanation: The intersected node's value is 8 (note that this must
> not be 0 if the two lists intersect). From the head of A, it reads as
> [4,1,8,4,5]. From the head of B, it reads as [5,0,1,8,4,5]. There are 2
> nodes before the intersected node in A; There are 3 nodes before the
> intersected node in B.
> ```

也算是我第一次面试的第一道题。当时想的是暴力解决，double循环哈哈哈。这次写的时候想到了一个新的方法就是使用HashMap来解决，最坏的情况是$O(n+m)$，不过也比$O(n^2)$好。先把一个LinkList全部放到HashMap里面，然后循环第二个链表，看是不是有一样的node，如果有就直接return当前的node，如果直到循环结束还没有就 `return null`

```
public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
    ListNode head = null;
    ListNode curr = headA;
    Map<ListNode, Integer> map = new HashMap<>();

    while (curr != null) {
        map.put(curr, curr.val);
        curr = curr.next;
    }

    curr = headB;
    while (curr != null) {
        if (map.containsKey(curr)) {
            head = curr;
            break;
        }
        curr = curr.next;
    }

    return head;
}
```

在面试的时候面试官也给我说了另一种解法，就是假设已知两条链表的长度，然后先把两条链表截取到一样的长度，然后一个循环就能找到相对应的结果。这个我觉得时间复杂度比我的要高，就没写。

powered by GitbookFile Modify: 2024-11-27 04:38:02

162. Find Peak Element

A peak element is an element that is greater than its neighbors.

Given an input array `nums`, where `nums[i] ≠ nums[i+1]`, find a peak element and return its index.

The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.

You may imagine that `nums[-1] = nums[n] = -∞`.

Example 1:

```
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number 2.
```

Example 2:

```
Input: nums = [1,2,1,3,5,6,4]
Output: 1 or 5
Explanation: Your function can return either index number 1 where the peak element is 2,
             or index number 5 where the peak element is 6.
```

Follow up: Your solution should be in logarithmic complexity.

可以采用一个二分查找的方法来处理这个问题，因为已知peak number是 `nums[i] > nums[i+1]` 也就是说如果在array里面出现一个降序就代表着是peak number。这样的话可以使用二分查找，如果 `nums[mid] > nums[mid+1]` 就代表该数字就是peak number，但是之前的可以还有相对应的peak number，所以就是 `search(nums, left, mid)` 。

ps: 也有一种O(n)的方法来查找。

```java
public class FindPeakElement {
//    public int findPeakElement(int[] nums) {
//        for (int i = 0; i < nums.length - 1; i ++) {
//            if (nums[i] > nums[i+1]) return i;
//        }
//
//        return nums.length-1;
//    }

    public int findPeakElement(int[] nums) {
        return search(nums, 0, nums.length - 1);
    }

    public int search(int[] nums, int left, int right) {
        if (left == right) return left;

        int mid = (left + right) / 2;
        if (nums[mid] > nums[mid+1]) return search(nums, left, mid);

        return search(nums, mid+1, right);
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Majority Element

Majority Element

Given an array of size *n*, find the majority element. The majority element is the element that appears **more than** `⌊ n/2 ⌋` times.

You may assume that the array is non-empty and the majority element always exist in the array.

Example 1:

```
Input: [3,2,3]
Output: 3
```

Example 2:

```
Input: [2,2,1,1,1,2,2]
Output: 2
```

看到这一类的题，最先想到的就是使用HashMap来解决这个计数的问题，当计数的结果 `> nums.leng/2` 的时候可以直接return就好了，准没错

```
public int majorityElement(int[] nums) {
    if (nums.length <= 0) return -1;
    int len = nums.length - 1;
    Map<Integer, Integer> map = new HashMap<>();

    for (int i = 0; i <= len; i ++) {
        if (map.containsKey(nums[i])) {
            if (map.get(nums[i]) + 1 >= nums.length/2) return nums[i];
            map.put(nums[i], map.get(nums[i]) + 1);
        } else {
            map.put(nums[i], 1);
        }
    }

    return -1;
}
```

然后看了一下官方的解法，就是先sort一下，如果该数字的个数大于总array长度的一般也就意味着sort完之后取中间那个数字准没错。。。那么简单的方法咋就没想到呢o(╥﹏╥)o

```
public int majorityElement(int[] nums) {
    if (nums.length == 0) return -1;
    Arrays.sort(nums);
    return nums[nums.length/2];
}
```

Rotate Array

Rotate Array

Given an array, rotate the array to the right by *k* steps, where *k* is non-negative.

Follow up:

- Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.
- Could you do it in-place with O(1) extra space?

Example 1:

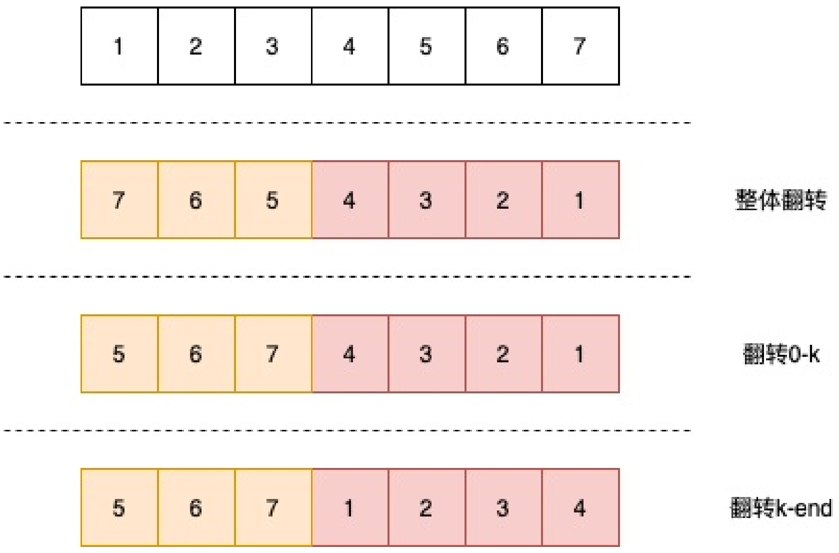
```
Input: nums = [1,2,3,4,5,6,7], k = 3
Output: [5,6,7,1,2,3,4]
Explanation:
rotate 1 steps to the right: [7,1,2,3,4,5,6]
rotate 2 steps to the right: [6,7,1,2,3,4,5]
rotate 3 steps to the right: [5,6,7,1,2,3,4]
```

Example 2:

```
Input: nums = [-1,-100,3,99], k = 2
Output: [3,99,-1,-100]
Explanation:
rotate 1 steps to the right: [99,-1,-100,3]
rotate 2 steps to the right: [3,99,-1,-100]
```

就是根据k的个数，每个element向后移动k位，超出array的话在array的开头继续。

刚开始试了好几种方法，但是都不是特别理想，看了一下discussion，学到了。这个翻转链表实际是有规律的，先把整个array reverse一下，然后再把 `0-k` reverse，然后把 `k-end` reverse一下就可以得到最终的结果。。。很神奇的规律。。。



```
import java.util.Arrays;

/**
 * 这个想法是在leetcode那边看到的，其工作原理就是先reverse一下
 * 然后在对前k个进行reverse，然后对k-len再reverse
 * 这样就能得到结果了
 */
public class RotateArray {
    public void rotate(int[] nums, int k) {
        if (nums.length < 2 || k < 1) return;
        // 整体反转
        reverse(0, nums.length - 1, nums);

        reverse(0, k % nums.length - 1, nums);
        reverse(k % nums.length, nums.length - 1, nums);
    }

    public void reverse(int left, int right, int[] nums) {
        while (left < right) {
            int temp = nums[left];
            nums[left] = nums[right];
            nums[right] = temp;
            left ++;
            right --;
        }
    }

    public static void main(String[] args) {
        RotateArray re = new RotateArray();
        int[] nums = new int[] {1,2};
        re.rotate(nums, 1);

        System.out.println(Arrays.toString(nums));
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Happy Number

Happy Number

Write an algorithm to determine if a number `n` is "happy".

A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it **loops endlessly in a cycle** which does not include 1. Those numbers for which this process **ends in 1** are happy numbers.

Return True if `n` is a happy number, and False if not.

Example:

```
Input: 19
Output: true
Explanation:
12 + 92 = 82
82 + 22 = 68
62 + 82 = 100
12 + 02 + 02 = 1
```

输入一个数字，然后把数字分开，然后平方相加。比如 `19 = 1^2 + 9^2` 得出的数字再继续进行一样的操作，一直简化到该数字到个位数，如果该数字等于1或者7的时候，就可以return true，否则的话就return false

```java
import java.util.LinkedList;
import java.util.List;

public class HappyNumber {
    public boolean isHappy(int n) {
        if (n == 1) return true;
        int sum = n;
        List<Integer> list = new LinkedList<>();

        while (true) {
            list = split(sum);
            sum = 0;
            for (Integer i : list) {
                sum += Math.pow(i, 2);
            }

            if (sum == 1 || sum == 7) {
                return true;
            }

            if (sum < 10) return false;

        }
    }

    public List<Integer> split(int n) {
        List<Integer> list = new LinkedList<>();
        while (n != 0) {
            list.add(n % 10);
            n = n / 10;
        }
        return list;
    }

    public static void main(String[] args) {
        HappyNumber hp = new HappyNumber();
        System.out.println(hp.isHappy(19));
    }
}
```

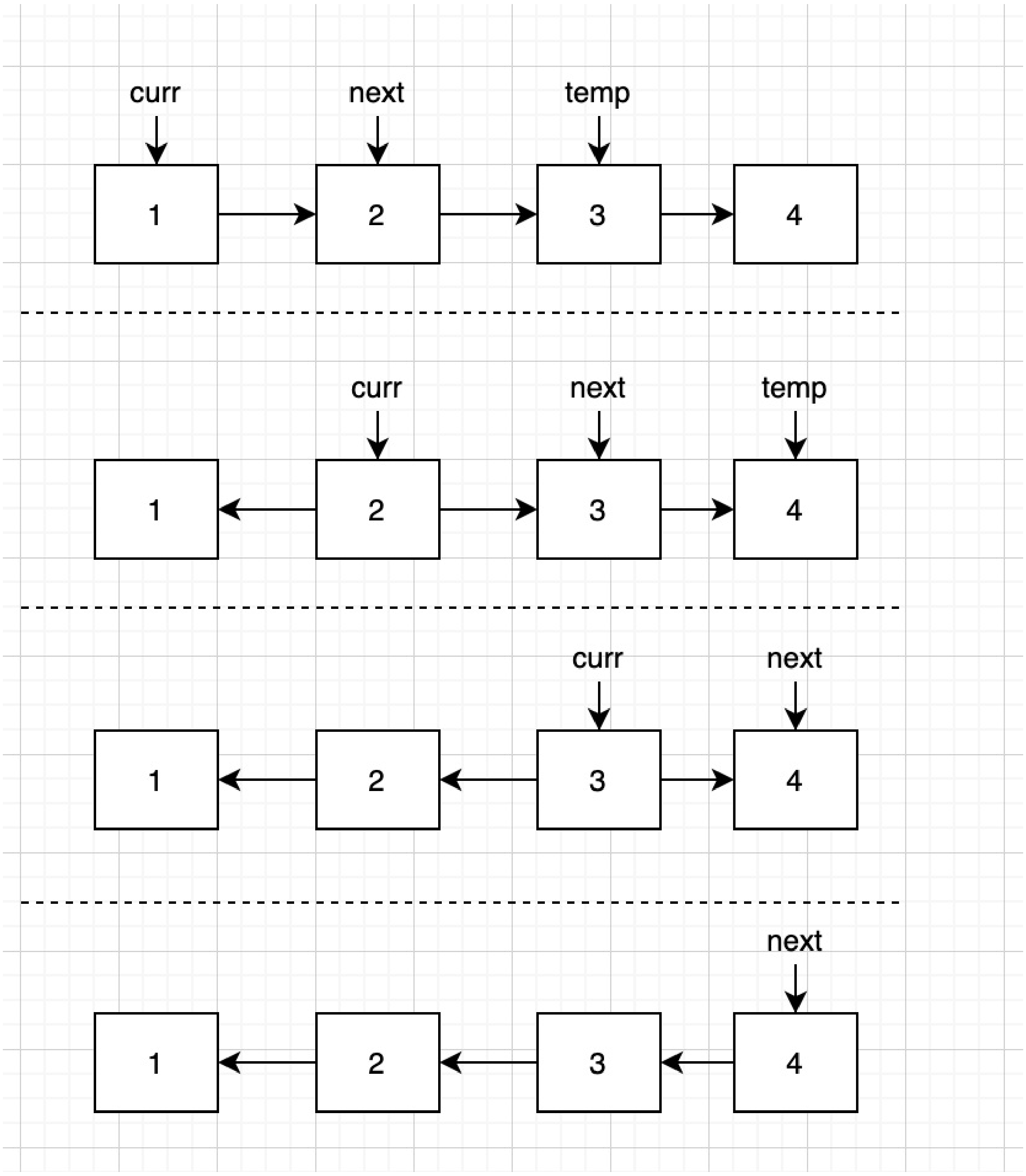
Reverse Linked List

Reverse a singly linked list.

Example:

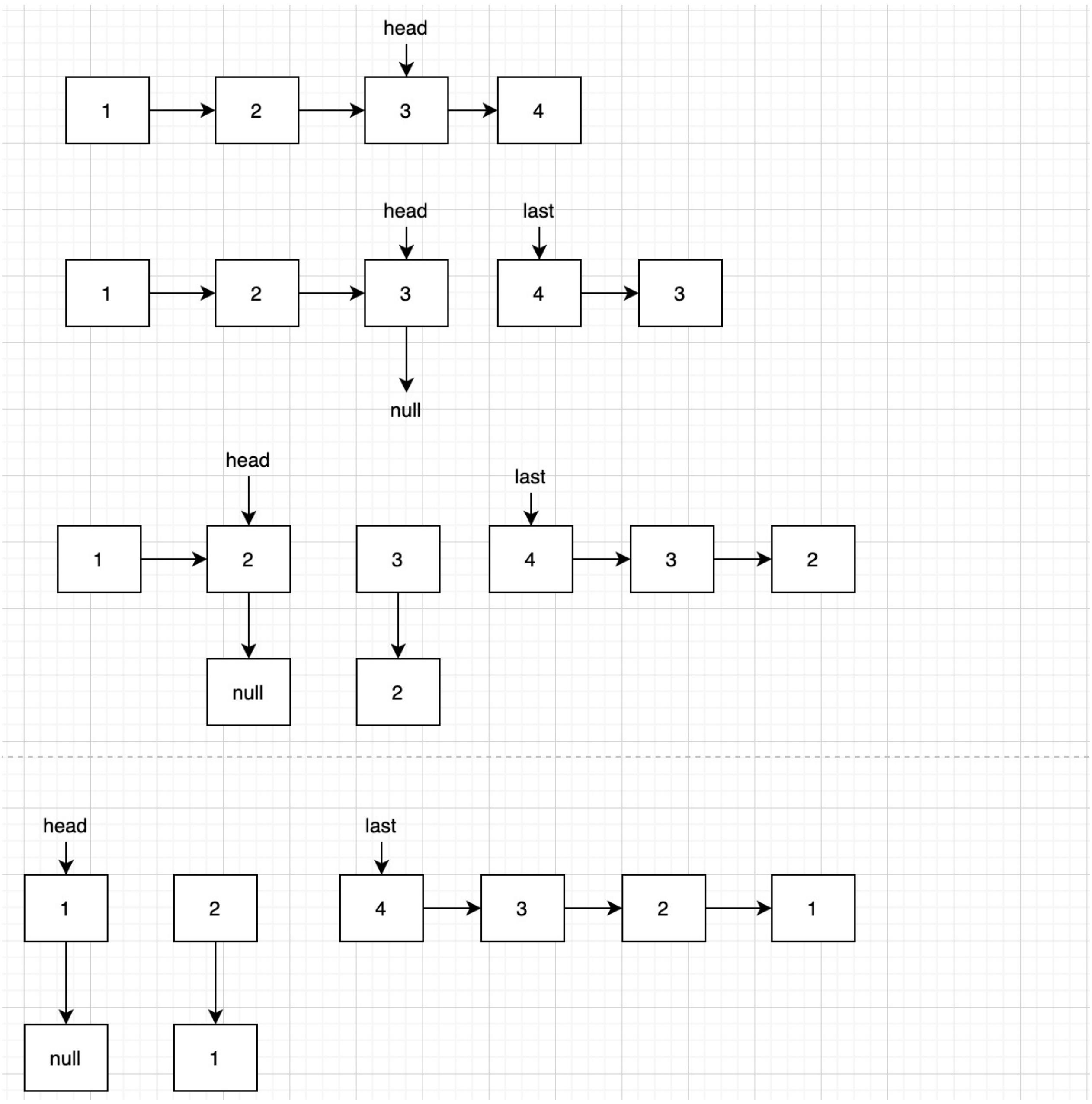
```
Input: 1->2->3->4->5->NULL
Output: 5->4->3->2->1->NULL
```

翻转链表可以循环实现，也可以用递归实现。



```
public ListNode reverseList(ListNode head) {
      if (head == null) return null;
      ListNode curr = head;
      ListNode next = head.next;
      ListNode temp = null;
      curr.next = null;


      while (next != null) {
          temp = next.next;
          next.next = curr;
          curr = next;
          next = temp;
      }

      return curr;
}
```

然后递归的话就刚开始看可能不是特别好理解的。总感觉遇到递归就有点懵逼的感觉。。。



```java
public ListNode reverseList(ListNode head) {
    if (head == null || head.next == null) return head;
    ListNode last = reverseList(head.next);

    head.next.next = head;
    head.next = null;

    return last;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

209. Minimum Size Subarray Sum

Minimum Size Subarray Sum: Given an array of positive integers nums and a positive integer target, return the minimal length of a contiguous subarray [numsl, numsl+1, ..., numsr-1, numsr] of which the sum is greater than or equal to target. If there is no such subarray, return 0 instead.

Example:

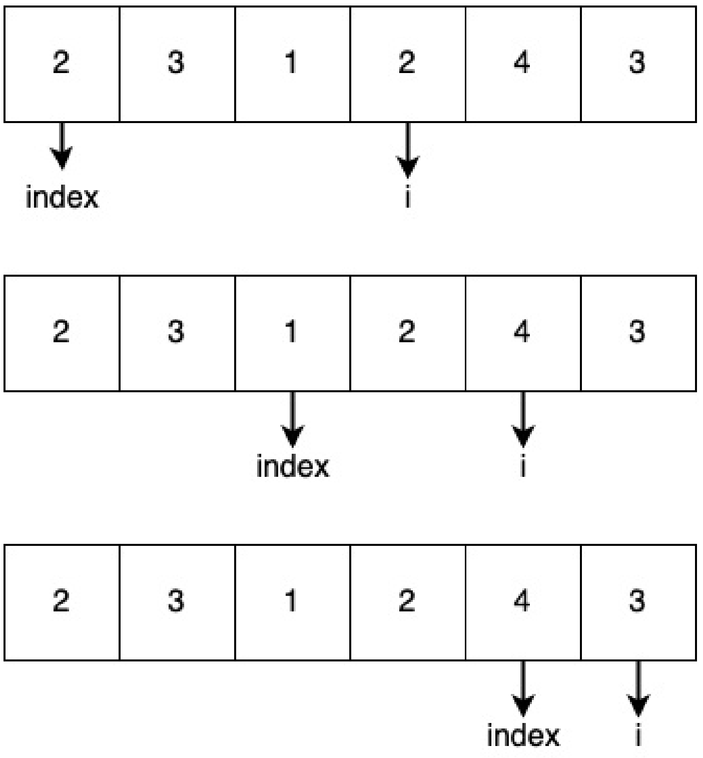
```
Input: target = 7, nums = [2,3,1,2,4,3]
Output: 2
Explanation: The subarray [4,3] has the minimal length under the problem constraint.
```

Brute Force

The basic approach is to use two loops to solve that. The time complexity of this approach is $O(N^2)$. It will cause time limit exceeded.

```python
class Solution:
    def minSubArrayLen(self, target: int, nums: List[int]) -> int:
        length = 9999

        for i in range(0, len(nums)):
            res = nums[i]
            if res >= target:
                length = 1
                continue
            for j in range(i + 1, len(nums)):
                res += nums[j]

                if res >= target:
                    length = min(length, j - i + 1)
                    break
        return length if length != 9999 else 0
```

Advance Approach



To solve this issue, we can use **Sliding Window** to solve this issue. This method is similar to the double pointer. The `index` the left index of this array. When the sum between `i` and `index` is greater or equal to the target, then advance `i` and `index` to find other positions that sum is greater or equal to the target. The basic idea of sliding windows is shown in the Figure.

```
class Solution:
    def minSubArrayLen(self, target: int, nums: List[int]) -> int:
        index = 0
        res = 0
        min_len = 99999999

        for i in range(0, len(nums)):
            res += nums[i]

            while res >= target:
                length = i - index + 1
                min_len = min(length, min_len)
                res -= nums[index]
                index += 1

        return min_len if min_len != 99999999 else 0
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

216. Combination Sum III

Combination Sum III: Find all valid combinations of k numbers that sum up to n such that the following conditions are true:

- Only numbers 1 through 9 are used.
- Each number is used at most once.

Return a list of all possible valid combinations. The list must not contain the same combination twice, and the combinations may be returned in any order. For example:

```
Input: k = 3, n = 7
Output: [[1,2,4]]
Explanation:
1 + 2 + 4 = 7
There are no other valid combinations.
```

This question is pretty similar with the previous one, just add one more constrain (sum = n).

```
class Solution:
    def combinationSum3(self, k: int, n: int) -> List[List[int]]:
        def combine(k, n, index, path, result):
            if len(path) == k:
                if sum(path) == n:
                    result.append([i for i in path])
                return path

            for i in range(index, 10):
                path.append(i)
                combine(k, n, i + 1, path, result)

                path.pop()

            return result

        return combine(k, n, 1, [], [])
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Contains Duplicate

Contains Duplicate

Given an array of integers, find if the array contains any duplicates.

Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

Example 1:

```
Input: [1,2,3,1]
Output: true
```

Example 2:

```
Input: [1,2,3,4]
Output: false
```

Example 3:

```
Input: [1,1,1,3,3,4,3,2,4,2]
Output: true
```

这个题用简单的HashMap做就行，或者用hashset，我这用的是hashset。

```
import java.util.Arrays;
import java.util.HashSet;

public class ContainsDuplicate {
    public boolean containsDuplicate(int[] nums) {
        HashSet<Integer> hashSet = new HashSet<Integer>();

        for (int num : nums) {
            hashSet.add(num);
        }

        return nums.length != hashSet.size();
    }

    public static void main(String[] args) {
        int[] nums = new int[]{1,2,3, 1};
        ContainsDuplicate cd = new ContainsDuplicate();
        System.out.println(cd.containsDuplicate(nums));
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

219. Contains Duplicate II

Given an array of integers and an integer *k*, find out whether there are two distinct indices *i* and *j* in the array such that **nums[i] = nums[j]** and the **absolute** difference between *i* and *j* is at most *k*.

Example 1:

```
Input: nums = [1,2,3,1], k = 3
Output: true
```

Example 2:

```
Input: nums = [1,0,1,1], k = 1
Output: true
```

Example 3:

```
Input: nums = [1,2,3,1,2,3], k = 2
Output: false
```

本来想的是直接用for循环，根据当前数字然后向后走k位，看k位以内有没有和 `nums[i]` 相同的数字，但是貌似bug有点多，放弃了。。改用HashMap了，key存 `nums[i]` ，value存相对应的index。如果HashMap已经有该key则可以计算HashMap里面存的index和当前i的差值，如果小于等于K就可以直接返回了。

```
import java.util.HashMap;
import java.util.Map;

public class ContainsDuplicateII {
    // 放弃了我使用hashmap来写好吧。。。
    public boolean containsNearbyDuplicate(int[] nums, int k) {
        Map<Integer, Integer> map = new HashMap<>();

        for (int i = 0; i < nums.length; i ++) {
            if (map.containsKey(nums[i])) {
                if (i - map.get(nums[i]) <= k) return true;
            }

            map.put(nums[i], i);
        }

        return false;
    }

    public static void main(String[] args) {
        ContainsDuplicateII c = new ContainsDuplicateII();
        int[] nums = new int[] {1,2,3,1,2,3};

        System.out.println(c.containsNearbyDuplicate(nums, 2));
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Palindrome Linked List

Palindrome Linked List

Given a singly linked list, determine if it is a palindrome.

Example 1:

```
Input: 1->2
Output: false
```

Example 2:

```
Input: 1->2->2->1
Output: true
```

Follow up: Could you do it in O(n) time and O(1) space?

思路：

本来想的是用递归的方法算一下和，如果是回文的话结果应该是一样的，想法很美好，现实很残酷。。。LeetCode官方应该也考虑到这个问题了，如果算sum的话会导致整型溢出的问题。。。。不过也算是一个思路吧。o(╥﹏╥)o

```
public boolean isPalindrome(ListNode head) {
    if (head == null) return false;
    if (head.next == null) return true;
    long [] nums = new long[]{0, 0};
    recur(head,nums);

    return nums[0] == nums[1];
}

public void recur(ListNode curr, long[] nums) {
    if (curr == null) return ;

    nums[0] = nums[0]*10 + curr.val;
    recur(curr.next, nums);
    nums[1] = nums[1]*10 + curr.val;
}
```

最近遇到的题不是递归就是动态规划。。。看来这块有点薄弱，有时间要多练习一下。。看了discussion里面，有个大神写的也是递归的方法，类似于先把第一个指针走到结尾，第二个指针指向head，然后对比是不是一样。感觉很牛逼。。不过递归貌似花费时间有点久。

```
boolean flag = true;
public boolean isPalindrome(ListNode head) {
    recur(head, head);
    return flag;
}

public ListNode recur(ListNode p1, ListNode p2) {
    if (p1 == null) return p2;
    ListNode node = recur(p1.next, p2);
    if (node.val != p1.val) flag = false;
    return node.next;
}
```

Ps: 还有一种快慢指针的方法。。就是走到中间，然后把两条链表分开对比就行了，但是感觉有点麻烦，还是不写了。。。

powered by GitbookFile Modify: 2024-11-27 04:38:02

238. Product of Array Except Self

Given an integer array `nums`, return *an array* `answer` *such that* `answer[i]` *is equal to the product of all the elements of* `nums` *except* `nums[i]`.

The product of any prefix or suffix of `nums` is **guaranteed** to fit in a **32-bit** integer.

You must write an algorithm that runs in `O(n)` time and without using the division operation.

Example 1:

```
Input: nums = [1,2,3,4]
Output: [24,12,8,6]
```

Example 2:

```
Input: nums = [-1,1,0,-3,3]
Output: [0,0,9,0,0]
```

Constraints:

- `2 <= nums.length <= 105`
- `30 <= nums[i] <= 30`
- The product of any prefix or suffix of `nums` is **guaranteed** to fit in a **32-bit** integer.

Follow up: Can you solve the problem in `O(1)` extra space complexity? (The output array **does not** count as extra space for space complexity analysis.)

Solution

这个问题给出的提示 `prefix` 和 `suffix` ，那么根据这个提示我们可以得知这个题是需要前后遍历的算。并且该问题要求时间复杂度是 `O(n)` ，我们可以采取左右两边遍历的方法。

根据这个问题本身的要求，算出该数组所有数字的乘积，但是不包含当前数字。首先我们可以使用 `prefix` 从左到右遍历一下：

```
nums: 1 2 3 4
pre:  1 1 2 6
```

我们可以观察到从左到右遍历的话， `nums` 中每个索引对应的数据是前面几个数字的乘积，这样的话我们可以得知从左到右的乘积是多少了。从右往左的时候，我们需要用当前数字乘以上一轮的结果，这样我们就能得到最后结果了。 `prefix` 记录的是从左到右的乘积， `sufix` 记录的是从右到左的乘积（不包含自身）。

```
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        auto ans = vector<int>(nums.size(), 1);

        int prefix = 1;
        for (int i = 0; i < nums.size(); i ++) {
            ans[i] = prefix;
            prefix *= nums[i];
        }

        int suffix = 1;
        for (int i = nums.size() - 1; i >= 0; i -- ) {
            ans[i] = suffix * ans[i];
            suffix *= nums[i];
        }

        return ans;
    }
};
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

242. Valid Anagram

Given two strings `s` and `t` , return true if `t` is an anagram of `s` , and false otherwise.

```
Input: s = "anagram", t = "nagaram"
Output: true
Input: s = "rat", t = "car"
Output: false
```

Brute Force

This issue can be solved by using two loop. But the time complexity is $O(N)$. In addition, it also can be done through sort algorithm, to check whether same. But it is too complex. Here we did not code for this approache.

Advance Approach

We can use `hashmap` to record that how many times they appeared in the string. Therefore, the time complexity is $O(N)$.

```python
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        if len(s) != len(t):
            return False

        data = {}

        for i in s:
            if i not in data:
                data[i] = 1
            else:
                data[i] += 1

        for i in t:
            if i not in data:
                return False
            else:
                if data[i] == 0:
                    return False
                else:
                    data[i] -= 1

        for key, val in data.items():
            if data[key] != 0:
                return False

        return True
```

268. Missing Number

Given an array `nums` containing `n` distinct numbers in the range `[0, n]`, return *the only number in the range that is missing from the array.*

Example 1:

```
Input: nums = [3,0,1]
Output: 2
Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is
 the missing number in the range since it does not appear in nums.
```

Example 2:

```
Input: nums = [0,1]
Output: 2
Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is
 the missing number in the range since it does not appear in nums.
```

Example 3:

```
Input: nums = [9,6,4,2,3,5,7,0,1]
Output: 8
Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is
 the missing number in the range since it does not appear in nums.
```

Constraints:

- `n == nums.length`
- `1 <= n <= 104`
- `0 <= nums[i] <= n`
- All the numbers of `nums` are **unique**.

Follow up: Could you implement a solution using only `O(1)` extra space complexity and `O(n)` runtime complexity?

Solution

用hashmap就能解决。但空间复杂度是 `O(n)` 。如果想要在 `O(1)` 是的空间复杂度中完成，可以通过求和的方式来解决。

```
class Solution {
public:
    int missingNumber(vector<int>& nums) {
        int r_sum = 0;
        int a_sum = 0;

        for (int i = 0; i < nums.size(); i ++) {
            r_sum += i;
            a_sum += nums[i];
        }

        return r_sum + nums.size() - a_sum;
    }
};
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Move Zeroes

Move Zeroes

Given an array `nums`, write a function to move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

Example:

```
Input: [0,1,0,3,12]
Output: [1,3,12,0,0]
```

Note:

1. You must do this **in-place** without making a copy of the array.
2. Minimize the total number of operations.

一开始看到这个想了一下双指针，一个从头开始一个从尾部开始，这样虽然可以把0都放到后面，但是这样就不是顺序来的了。。。看了一下大佬思路，也是双指针（方法思路没错对吧），只不过这个双指针都是从头开始的。先设置一个0的指针 `zero=-1` 因为我们一开始不知道0的位置在那，然后开始循环，如果该数字为0可以对 `zero` 进行赋值了（ `zero == -1` ），当 `zero != -1` 说明之前已经有了0的数字。然后如果该数字不等于0的时候表示可以对之前的zero的index进行替换，前提是 `zero != -1` ，然后判断zero和i的位置，如果i是zero的next说明是只有一个zero，不然的话就代表zero后面还是一个0的数字，所以这时候 `zero++` 就行。

```
public void moveZeroes(int[] nums) {
    int zero = -1;

    for (int i = 0; i < nums.length; i ++) {
        if (nums[i] != 0) {
            if (zero == -1) continue;
            nums[zero] = nums[i];
            nums[i] = 0;
            zero = (zero == i + 1) ? i : zero+1;
        } else {
            if (zero != -1) continue;
            zero = i;
        }
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

287. Find the Duplicate Number

Given an array of integers `nums` containing `n + 1` integers where each integer is in the range `[1, n]` inclusive.

There is only **one repeated number** in `nums`, return *this repeated number*.

You must solve the problem **without** modifying the array `nums` and using only constant extra space.

Example 1:

```
Input: nums = [1,3,4,2,2]
Output: 2
```

Example 2:

```
Input: nums = [3,1,3,4,2]
Output: 3
```

Example 3:

```
Input: nums = [3,3,3,3,3]
Output: 3
```

Constraints:

- `1 <= n <= 105`
- `nums.length == n + 1`
- `1 <= nums[i] <= n`
- All the integers in `nums` appear only **once** except for **precisely one integer** which appears **two or more** times.

Follow up:

- How can we prove that at least one duplicate number must exist in `nums` ?
- Can you solve the problem in linear runtime complexity?

Solution

S1. 可以使用 `hashmap` 的思路来解决这个问题，解决方法比较简单，但是不满足 `O(1)` space的要求。

```
class Solution {
public:
    int findDuplicate(vector<int>& nums) {
        auto maps = unordered_map<int, int>();

        for (int const& num : nums) {
            maps[num] += 1;

            if (maps[num] > 1) return num;
        }

        return -1;
    }
};
```

S2. 使用快慢指针的方法来解决，因为题目中的要求其实已经声明了，数字不会超过数组长度，所以可以使用快慢指针的思路来找到cycle。

```
class Solution {
public:
    int findDuplicate(vector<int>& nums) {
        int slow = nums[0];
        int fast = nums[0];

        // Phase 1: Detect cycle
        do {
            slow = nums[slow];
            fast = nums[nums[fast]];
        } while (slow != fast);

        slow = nums[0];
        while (slow != fast) {
            slow = nums[slow];
            fast = nums[fast];
        }

        return slow; // The duplicate number
    }
};
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

344. Reverse String

Write a function that reverses a string. The input string is given as an array of characters `char[]`.

Do not allocate extra space for another array, you must do this by **modifying the input array in-place** with O(1) extra memory.

You may assume all the characters consist of printable ascii characters.

Example 1:

```
Input: ["h","e","l","l","o"]
Output: ["o","l","l","e","h"]
```

Example 2:

```
Input: ["H","a","n","n","a","h"]
Output: ["h","a","n","n","a","H"]
```

感觉没有什么技术含量，就直接reverse就行。

```
import java.util.Arrays;

public class ReverseString {
    public void reverseString(char[] s) {
        int left = 0;
        int right = s.length - 1;

        while (left < right) {
            char temp = s[left];
            s[left] = s[right];
            s[right] = temp;
            left ++;
            right --;
        }
    }

    public static void main(String[] args) {
        ReverseString re = new ReverseString();
        char[] chars = new char[] {'h', 'e', 'l', 'l', 'o'};
        re.reverseString(chars);

        System.out.println(Arrays.toString(chars));
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

347. Top K Frequent Elements

Given a non-empty array of integers, return the *k* most frequent elements.

Example 1:

```
Input: nums = [1,1,1,2,2,3], k = 2
Output: [1,2]
```

Example 2:

```
Input: nums = [1], k = 1
Output: [1]
```

Note:

- You may assume *k* is always valid, 1 ≤ *k* ≤ number of unique elements.
- Your algorithm's time complexity **must be** better than O(*n* log *n*), where *n* is the array's size.
- It's guaranteed that the answer is unique, in other words the set of the top k frequent elements is unique.
- You can return the answer in any order.

用HashMap可以解决，但是因为Java对HashMap有点点不太友善，我改成Python了，因为可以自定义sort 字典，这样可以根据结果排序了。。。

```
class Solution:
    def topKFrequent(self, nums, k):
        frequence = {}

        for value in nums:
            if value in frequence:
                frequence[value] += 1
            else:
                frequence[value] = 1

        frequence = sorted(frequence.items(), key=lambda x: x[1], reverse=True)
        index = 0
        res = []

        for t in frequence:
            index += 1
            res.append(t[0])
            if index == k:
                break

        return res
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Intersection of Two Arrays II

Intersection of Two Arrays II

Given two arrays, write a function to compute their intersection.

Example 1:

```
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2,2]
```

Example 2:

```
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [4,9]
```

Note:

- Each element in the result should appear as many times as it shows in both arrays.
- The result can be in any order.

这个题就稍微简单一点了，可以用HashMap来记录 `num1` 的元素，value来存数字出现的次数。这样在遍历 `num2` 的时候就可以直接判断了。。。刚开始想到过array的用法，但是因为key是单一的，默默被我放弃了（我就是傻逼。。。），看了一下discussion恍然大悟。。。

```java
import javax.swing.*;
import java.util.*;

/**
 * Input: nums1 = [1,2,2,1], nums2 = [2,2]
 * Output: [2,2]
 * 没啥思路， 傻逼了，想着可以用hashmap但是，想到hashmap的key只能存一个value
 * 但是value可以表示数量啊
 * 傻逼
 *
 */
public class Intersection {
    public int[] intersect(int[] nums1, int[] nums2) {
        HashMap<Integer, Integer> maps = new HashMap<>();
        ArrayList<Integer> res = new ArrayList<Integer>();

        for (Integer num : nums1) {
            if (maps.containsKey(num)) {
                maps.put(num, maps.get(num) + 1);
            } else {
                maps.put(num, 1);
            }
        }

        for (Integer num : nums2) {
            if (maps.containsKey(num)) {
                res.add(num);
                if (maps.get(num) == 1) maps.remove(num);
                else maps.put(num, maps.get(num) - 1);
            }
        }
        int[] result = new int[res.size()];
        for (int i = 0; i < res.size(); i ++) result[i] = res.get(i);
        return result;
    }

    public static void main(String[] args) {
        int[] num1 = new int[] {1,2,2,1};
        int[] num2 = new int[] {2,2};
        Intersection i = new Intersection();
        int[] res = i.intersect(num1, num2);
        System.out.println(Arrays.toString(res));
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

387. First Unique Character in a String

Given a string, find the first non-repeating character in it and return it's index. If it doesn't exist, return -1.

Examples:

```
s = "leetcode"
return 0.

s = "loveleetcode",
return 2.
```

使用了HashMap的方法，当字符只出现一次就设置key为当前char，value设置为index，如果出现超过一次，就标记value是MAX。筛选HashMap里面value不是MAX，并且返回就行了。

```
import java.util.Collections;
import java.util.HashMap;

public class FirstUnique {
    public int firstUniqChar(String s) {
        HashMap<Character, Integer> map = new HashMap<>();

        for (int i = 0; i < s.length(); i ++) {
            char c = s.charAt(i);
            if (map.containsKey(c)) {
                map.put(c, Integer.MAX_VALUE);
            } else {
                map.put(c, i);
            }
        }

        int position = Integer.MAX_VALUE;
        for (Character c : map.keySet()) {
            if (map.get(c) != Integer.MAX_VALUE) {
                if (map.get(c) < position) position = map.get(c);
            }
        }
        if (position == Integer.MAX_VALUE) return -1;
        return position;
    }

    public static void main(String[] args) {
        FirstUnique f = new FirstUnique();
        String str = "loveleetcode";
        System.out.println(f.firstUniqChar(str));
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

Find All Numbers Disappeared in an Array

Find All Numbers Disappeared in an Array

Given an array of integers where 1 ≤ a[i] ≤ *n* (*n* = size of array), some elements appear twice and others appear once.

Find all the elements of [1, *n*] inclusive that do not appear in this array.

Could you do it without extra space and in O(*n*) runtime? You may assume the returned list does not count as extra space.

Example:

```
Input:
[4,3,2,7,8,2,3,1]

Output:
[5,6]
```

题解：数组长度1-n，要判断数组里面出现的数字是不是在1-n之间，如果少了哪个数字就给补上。

比如： `[2,2]` 数组长度 `1-2` ，那么改数组少了1。

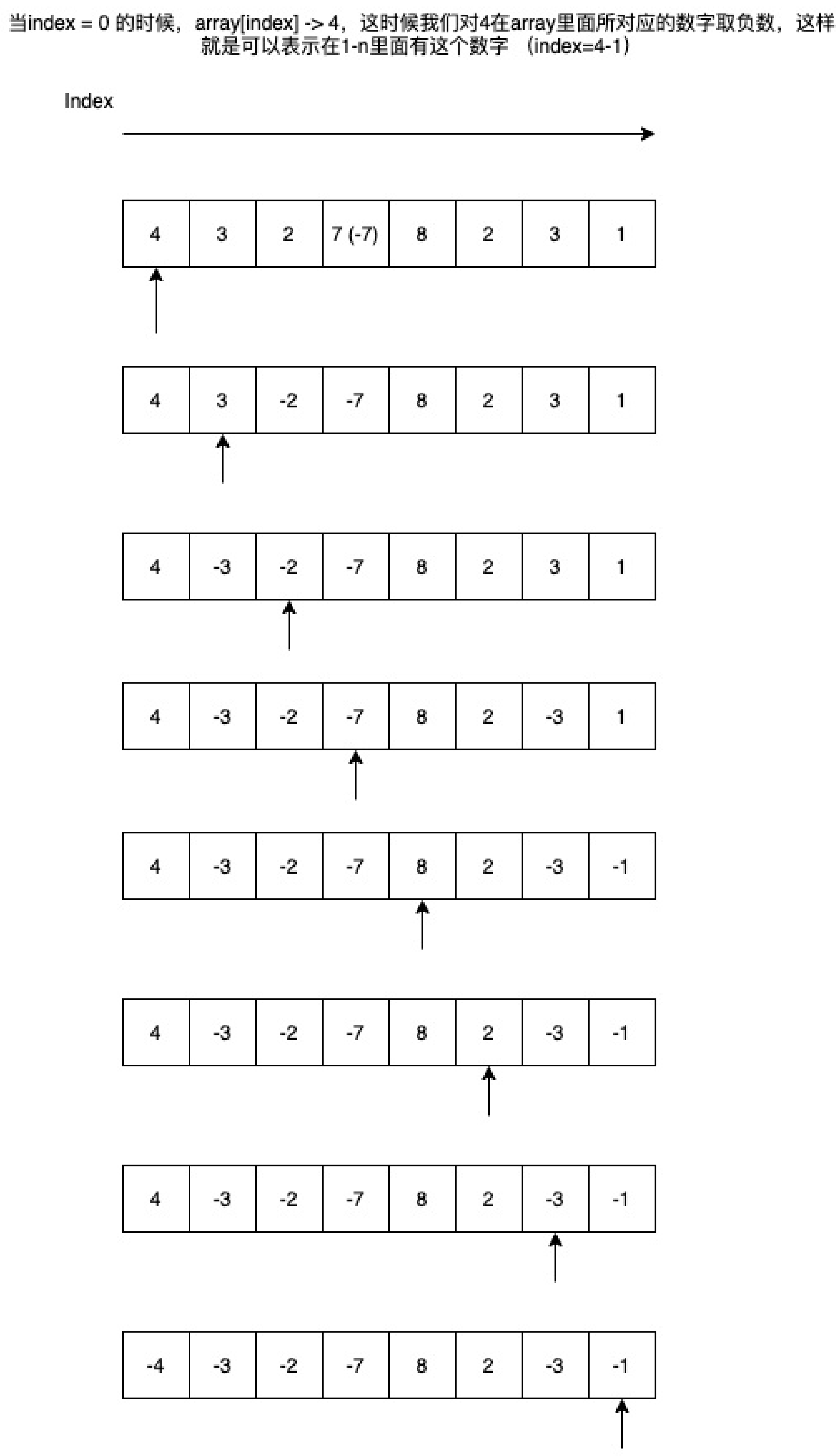
solution 1：

刚开始考虑的是先sort一下，然后判断index在0，末尾和中间的情况进行补全，但是貌似有点浪费时间了。。所以时间复杂度有点高$O(nlog_{n} + n)$。不过也算是一种方法对吧。。。o(╥﹏╥)o

```java
public List<Integer> findDisappearedNumbers(int[] nums) {
    List<Integer> dis = new LinkedList<>();
    if (nums.length == 0) return dis;
    Arrays.sort(nums);

    int num = nums[0];
    while (num > 1) {
        num --;
        dis.add(num);
    }

    num = nums[0];

    for (int i = 1; i < nums.length; i ++) {
        while (num + 1 != nums[i] && num != nums[i]) {
            num++;
            dis.add(num);
        }
        num = nums[i];
    }

    num = nums[nums.length-1];
    while (num != nums.length) {
        num++;
        dis.add(num);
    }
    return dis;
}
```

solution 2：

看了一下discussion感觉他们写的方法屌爆了。。。因为数组里面的数字是在数组长度 `1-n` 之内的，所以这个时候我们可以做一个指示器的格式，把数组里面每个数字所对应的坐标变成负数，然后再寻找一个array里面哪些是负数。很欢喜。。


当index = 0 的时候，array[index] -> 4，这时候我们对4在array里面所对应的数字取负数，这样就是可以表示在1-n里面有这个数字 （index=4-1）

```java
public List<Integer> findDisappearedNumbers(int[] nums) {
    List<Integer> dis = new LinkedList<>();
    if (nums.length == 0) return dis;

    for (int i = 0; i < nums.length; i ++) {
        int index = Math.abs(nums[i]) - 1;
        if (nums[index] > 0) {
            nums[index] = -nums[index];
        }
    }

    for (int i = 0; i < nums.length; i ++) if (nums[i] > 0) dis.add(i + 1);

    return dis;
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

496. Next Greater Element I

The **next greater element** of some element `x` in an array is the **first greater** element that is **to the right** of `x` in the same array.

You are given two **distinct 0-indexed** integer arrays `nums1` and `nums2`, where `nums1` is a subset of `nums2`.

For each `0 <= i < nums1.length`, find the index `j` such that `nums1[i] == nums2[j]` and determine the **next greater element** of `nums2[j]` in `nums2`. If there is no next greater element, then the answer for this query is `-1`.

Return *an array* `ans` *of length* `nums1.length` *such that* `ans[i]` *is the* ***next greater element*** *as described above.*

Example 1:

```
Input: nums1 = [4,1,2], nums2 = [1,3,4,2]
Output: [-1,3,-1]
Explanation: The next greater element for each value of nums1 is as follows:
- 4 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer i
s -1.
- 1 is underlined in nums2 = [1,3,4,2]. The next greater element is 3.
- 2 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer i
s -1.
```

Example 2:

```
Input: nums1 = [2,4], nums2 = [1,2,3,4]
Output: [3,-1]
Explanation: The next greater element for each value of nums1 is as follows:
- 2 is underlined in nums2 = [1,2,3,4]. The next greater element is 3.
- 4 is underlined in nums2 = [1,2,3,4]. There is no next greater element, so the answer i
s -1.
```

Constraints:

- `1 <= nums1.length <= nums2.length <= 1000`
- `0 <= nums1[i], nums2[i] <= 104`
- All integers in `nums1` and `nums2` are **unique**.
- All the integers of `nums1` also appear in `nums2`.

Follow up:

Could you find an `O(nums1.length + nums2.length)` solution?

Solution

和上面那个题思路差不多，只是每个element都是unique的，所以可以额外使用hashmap来方便运算。

```
class Solution {
public:
    vector<int> nextGreaterElement(vector<int>& nums1, vector<int>& nums2) {
        auto stack = vector<int>();
        auto maps = unordered_map<int, int>();

        auto res = vector<int>(nums1.size(), -1);

        if (nums2.size() == 0) return res;



        stack.push_back(nums2[0]);
        maps[nums2[0]] = -1;

        for (int i = 1; i < nums2.size(); i ++) {
            maps[nums2[i]] = -1;

            while (!stack.empty()) {
                int val = stack.back();

                if (val > nums2[i]) break;
                maps[val] = nums2[i];

                stack.pop_back();
            }

            stack.push_back(nums2[i]);
        }

        for (int i = 0; i < nums1.size(); i ++) {
            int val = nums1[i];

            if (maps.find(val) != maps.end()) {
                res[i] = maps[val];
            }
        }

        return res;
    }
};
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

645. Set Mismatch

You have a set of integers `s`, which originally contains all the numbers from `1` to `n`. Unfortunately, due to some error, one of the numbers in `s` got duplicated to another number in the set, which results in **repetition of one** number and **loss of another** number.

You are given an integer array `nums` representing the data status of this set after the error.

Find the number that occurs twice and the number that is missing and return *them in the form of an array*.

Example 1:

```
Input: nums = [1,2,2,4]
Output: [2,3]
```

Example 2:

```
Input: nums = [1,1]
Output: [1,2]
```

Constraints:

- `2 <= nums.length <= 104`
- `1 <= nums[i] <= 104`

Solution

`hashmap` 的解决思路

```
class Solution {
public:
    vector<int> findErrorNums(vector<int>& nums) {
        auto res = vector<int>();
        int r_sum = nums.size() * (nums.size() + 1) / 2;
        auto maps = unordered_map<int, int>();

        for (int const& num : nums) {
            maps[num] += 1;

            if (maps[num] > 1) res.push_back(num);

            if (maps[num] == 1) {
                r_sum -= num;
            }
        }

        res.push_back(r_sum);

        return res;
    }
};
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

704. Binary Search

Given a **sorted** (in ascending order) integer array `nums` of `n` elements and a `target` value, write a function to search `target` in `nums`. If `target` exists, then return its index, otherwise return `-1`.

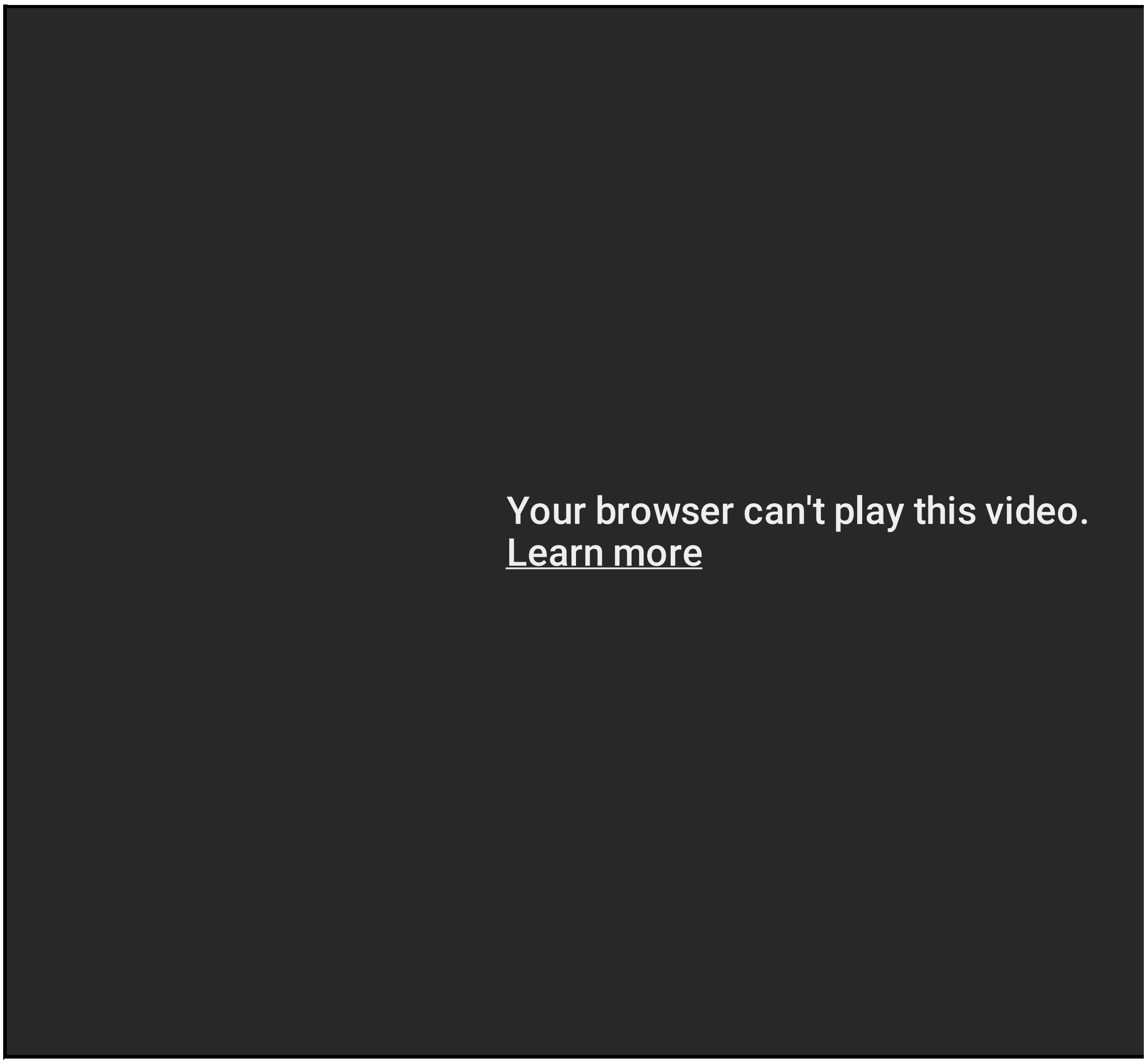
Example 1:

```
Input: nums = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
```

Example 2:

```
Input: nums = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1
```

标准的二分查找法（只能用在排过序的list里面）。有一个算法讲解：


Your browser can't play this video.
Learn more

```java
public class BinarySearch {
    public int search(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;

        while (left <= right) {
            int mid = (left + right) / 2;
            if (nums[mid] == target) return mid;
            else if (nums[mid] > target) right = mid - 1;
            else left = mid + 1;
        }

        return -1;
    }

    public static void main(String[] args) {
        int[] nums = new int[] {-1,0,3,5,9,12};
        BinarySearch bs = new BinarySearch();
        int res = bs.search(nums, 9);
        System.out.println(res);
    }
}
```

powered by GitbookFile Modify: 2024-11-27 04:38:02

739. Daily Temperatures

Given an array of integers `temperatures` represents the daily temperatures, return *an array* `answer` *such that* `answer[i]` *is the number of days you have to wait after the* `ith` *day to get a warmer temperature*. If there is no future day for which this is possible, keep `answer[i] == 0` instead.

Example 1:

```
Input: temperatures = [73,74,75,71,69,72,76,73]
Output: [1,1,4,2,1,1,0,0]
```

Example 2:

```
Input: temperatures = [30,40,50,60]
Output: [1,1,1,0]
```

Example 3:

```
Input: temperatures = [30,60,90]
Output: [1,1,0]
```

Constraints:

- `1 <= temperatures.length <= 105`
- `30 <= temperatures[i] <= 100`

Solution

```
Temperature: [73, 74, 75, 71, 69, 72, 76, 73]
stack = []
res = [0, 0, 0, 0, 0, 0, 0, 0]

1. index = 0
stack = [(0, 73)]

2. index = 1
74 > 73, pop (0, 73) and set res[0] = index - 0
stack = [(1, 74)]
res = [1, 0, 0, 0, 0, 0, 0, 0]

3. index = 2
75 > 74, pop (1, 74) and set res[1] = index - 1
stack = [(2, 75)]
res = [1, 1, 0, 0, 0, 0, 0, 0]

4. index = 3
71 < 75, add (3, 71) into stack
stack = [(2, 75), (3, 71)]

5. index = 4
69 < 71, add (3, 69) into stack
stack = [(2, 75), (3, 71), (4, 69)]

6. index = 5
72 < 69, pop (4, 69) and (3, 71), set res[4] = index - 4, and res[3] = index - 3
stack = [(2, 75), (5, 72)]
res = [1, 1, 0, 2, 1, 0, 0, 0]

7. index = 6
76 > 72 pop (5, 72) and (2, 75), set res[5] = index - 5, and res[2] = index -2
stack = []
res = [1, 1, 4, 2, 1, 1, 0, 0]
```

这个是真是面试题，21年的时候面试的时候遇到了。可以采用 `stack` 数据结构的解决方法。当遍历当前温度的时候去判断 `stack` 里面的元素是否小于当前温度，如果小于的话则 `pop` ，最后结束循环的时候把当前温度添加到 `stack` 中，因为每个温度都要找到比当前大的温度。

```
class Solution {
public:
    vector<int> dailyTemperatures(vector<int>& temperatures) {
        auto stack = std::vector<pair<int, int>>();
        auto res = vector<int>(temperatures.size(), 0);


        for (int i = 0; i < temperatures.size(); i ++) {
            int t = temperatures[i];
            if (stack.size() == 0) {
                stack.push_back(pair(t, i));
                continue;
            }

            while (!stack.empty()) {
                pair<int, int> val = stack.back();

                if (val.first >= t) break;

                res[val.second] = i - val.second;
                stack.pop_back();
            }

            stack.push_back(pair(t, i));

        }
        return res;
    }
};
```

1019. Next Greater Node In Linked List

Next Greater Node In Linked List is a medium problem. We are given a linked list with `head` as the first node. Let's number the nodes in the list: `node_1, node_2, node_3, ... etc.`

Each node may have a next larger value: for node_i, next_larger(node_i) is the node_j.val such that `j > i`, `node_j.val > node_i.val,` and `j` is the smallest possible choice. If such a j does not exist, the next larger value is 0.

Return an array of integers answer, where `answer[i] = next_larger(node_{i+1})`.

Note that in the example inputs (not outputs) below, arrays such as `[2,1,5]` represent the serialization of a linked list with a head node value of 2, second node value of 1, and third node value of 5.

Approach

This problem is pretty similar with Daily Temperatures which can be solved by applying stack. For this issue, we can use `stack` + `hashmap` to solve it with $O(n)$ time complexity.

1. Init stack and the final result.
2. `result` (hashmap) will recored the index as key and greater or 0 value as the value
3. When the `temp` node value is greater then the top value of the stack, update the `result` value and pop the top value of stack.

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def nextLargerNodes(self, head: ListNode) -> List[int]:
        stack = []
        result = {}

        temp = head
        index = 0

        # stack store like (val, pos)
        while (temp != None):

            while len(stack) != 0 and temp.val > stack[len(stack) - 1][0]:
                result[stack[len(stack) - 1][1]] = temp.val
                stack.pop(len(stack) - 1)

            stack.append((temp.val, index))
            result[index] = 0

            index += 1
            temp = temp.next


        return result.values()
```

We can use the following data to give an example:

```
Input: [2,1,5]
Output: [5,5,0]
```

In the following part, we will display the `result` and `stack` storage.

```
1. stack.push((2, 0)) stack = [(2, 0)], result = {0: 0}

2. stack.push((1, 1)), stack = [(2, 0), (1, 1)], result = {0: 0, 1: 0}

3. The node value is 5, greater than 1, then
    1. top value of stack is (1, 1), then result[1] = node.val (5), stack pop (1, 1)
    2. top value of stack is (2, 0), then result[0] = node.val (5), stack pop (2, 0)
    3. End while loop, and insert the (5, 3) into stack and result.

The final value in result is:
result = {
    0: 5,
    1: 5,
    2: 0
}
```

1365. How Many Numbers Are Smaller Than the Current Number

How Many Numbers Are Smaller Than the Current Number is an easy problem. Howerver, it is more interesting than other questions.

Given the array nums, for each `nums[i]` find out how many numbers in the array are smaller than it. That is, for each `nums[i]` you have to count the number of valid j's such that `j != i` and `nums[j] < nums[i]`.

Example 1:

```
Input: nums = [8,1,2,2,3]
Output: [4,0,1,1,3]
Explanation:
For nums[0]=8 there exist four smaller numbers than it (1, 2, 2 and 3).
For nums[1]=1 does not exist any smaller number than it.
For nums[2]=2 there exist one smaller number than it (1).
For nums[3]=2 there exist one smaller number than it (1).
For nums[4]=3 there exist three smaller numbers than it (1, 2 and 2).
```

Approach

Naive approach is brute force. Just use two nest loop can get the result.

```
result = []
for i in range(0, len(nums)):
    num = 0
    for j in range(i, len(nums)):
        if nums[i] > nums[j]:
            num += 1
    result.append(num)
```

The time complexity is `O(n^2)`

powered by GitbookFile Modify: 2024-11-27 04:38:02