20. Valid Parentheses

Easy Topics Companies Hint

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

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

- `1 <= s.length <= 10^4`
- `s` consists of parentheses only `'()[]{}'`.

Seen this question in a real interview before? 1/4

Yes No

Problem List

Run Submit

Code

Java Auto

```java
class Solution {
    public boolean isValid(String s) {
      Stack<Character> stack = new Stack<>();
      for (char c : s.toCharArray()) {
        if (c == '(')
          stack.push(')');
        else if (c == '{')
          stack.push('}');
        else if ( c== '[')
          stack.push(']');
        else {
          if (stack.isEmpty()) return false;
          if (stack.pop() != c) return false;
        }
      }
      return stack.isEmpty();
    }
}


//       Stack<Character> stack=new Stack<>();
//       for(int i=0; i<s.length(); i++ ){
//         char c = s.charAt(i);
//         if(c=='(' || c=='[' || c=='{') {
//           stack.push(c);
//           continue;
//         }
//         if(c==')') {
//           if(stack.empty()) return false;
//           if(stack.peek() != '(' ) return false;
//           stack.pop();
//         }
//         if(c==']') {
//           if(stack.empty()) return false;
//           if(stack.peek() != '[' ) return false;
//           stack.pop();
//         }
//         if(c=='}') {
//           if(stack.empty()) return false;
//           if(stack.peek() != '{' ) return false;
//           stack.pop();
//         }

//       }
//       return stack.empty();

```

Saved to local

Testcase Test Result

21. Merge Two Sorted Lists

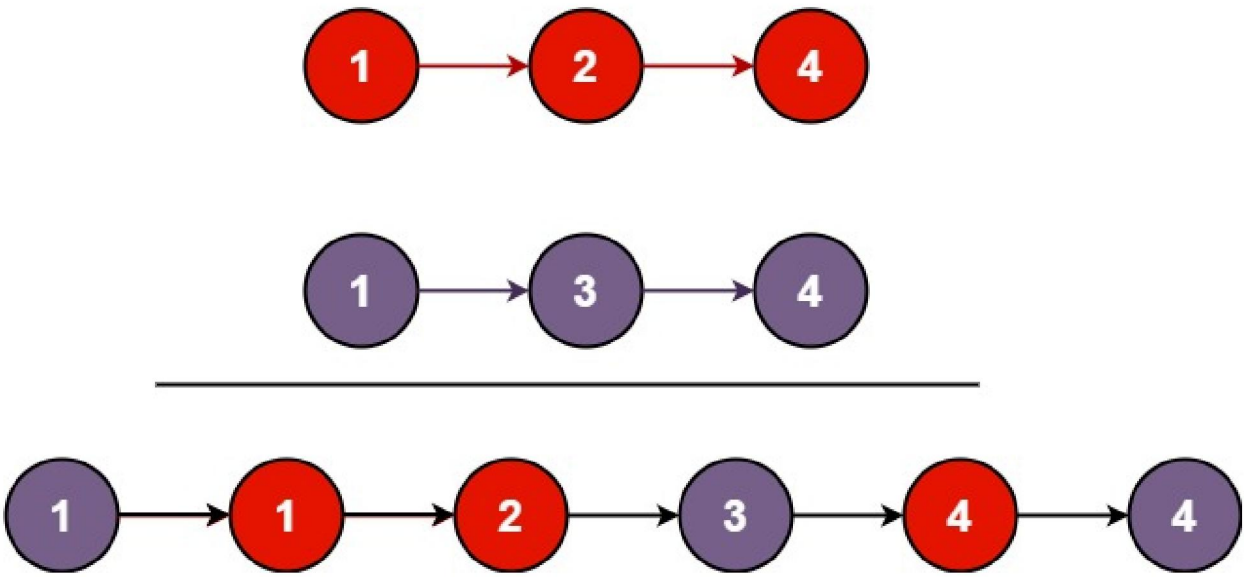
Easy Topics Companies

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

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

Return *the head of the merged linked list*.

Example 1:




```
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]
```

Example 2:

```
Input: list1 = [], list2 = []
Output: []
```

Example 3:

```
Input: list1 = [], list2 = [0]
Output: [0]
```

Code

Java Auto

```java
/**
 * Definition for singly-linked list.
 * 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; }
 * }
 */
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
      ListNode head = new ListNode(0);
      ListNode cur = head;
      while(l1!=null && l2!=null) {
        if(l1.val<l2.val) {
          cur.next=l1;
          l1=l1.next;
        } else {
          cur.next=l2;
          l2=l2.next;
        }
        cur=cur.next;
      }
      if(l1!=null)
        cur.next=l1;
      if(l2!=null)
        cur.next=l2;
      return head.next;

    }
}
```

Saved to local

Testcase Test Result

22. Generate Parentheses

Medium Topics Companies

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

Example 1:

```
Input: n = 3
Output: ["((()))","(()())","(())()","()(())","()()()"]
```

Example 2:

```
Input: n = 1
Output: ["()"]
```

Constraints:

- `1 <= n <= 8`

Seen this question in a real interview before? 1/4

Yes No

Accepted 1.6M Submissions 2.2M Acceptance Rate 73.6%

Topics

Companies

Similar Questions

Discussion (63)

20K 63

Run Submit

Code

Java Auto

```java
class Solution {
    public List<String> generateParenthesis(int n) {
      List<String> ret = new ArrayList<>();
      helper(0, 0, n, "", ret);
      return ret;
    }
    public void helper(int open, int close, int max, String curStr, List<String> ret) {
      if (open == max && close == max) {
        ret.add(curStr);
        return;
      }
      if (open < max) {
        helper(open + 1, close, max, curStr + "(", ret);
      }
      if (open > close) {
        helper(open, close + 1, max, curStr + ")", ret);
      }
    }

}
```

Saved to local

Testcase Test Result

23. Merge k Sorted Lists

Hard Topics Companies

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

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

Example 1:

```
Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
[
  1->4->5,
  1->3->4,
  2->6
]
merging them into one sorted list:
1->1->2->3->4->4->5->6
```

Example 2:

```
Input: lists = []
Output: []
```

Example 3:

```
Input: lists = [[]]
Output: []
```

Constraints:

- `k == lists.length`
- `0 <= k <= 10^4`
- `0 <= lists[i].length <= 500`
- `-10^4 <= lists[i][j] <= 10^4`

Problem List

Run Submit

Code

Java Auto

```java
/**
 * Definition for singly-linked list.
 * 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; }
 * }
 */
class Solution {
    public ListNode mergeKLists(ListNode[] lists) {
      if(lists==null || lists.length==0) return null;

      PriorityQueue<ListNode> pq=new PriorityQueue<>((x,y)->x.val-y.val);

      ListNode head = new ListNode(0);
      ListNode cur=head;

      for(ListNode n : lists)   if(n!=null) pq.offer(n);
      while(!pq.isEmpty()) {
        cur.next=pq.poll();
        cur=cur.next;
        if(cur.next!=null) pq.offer(cur.next);
        cur.next=null;

      }
      return head.next;

    }
}
```

Saved to local

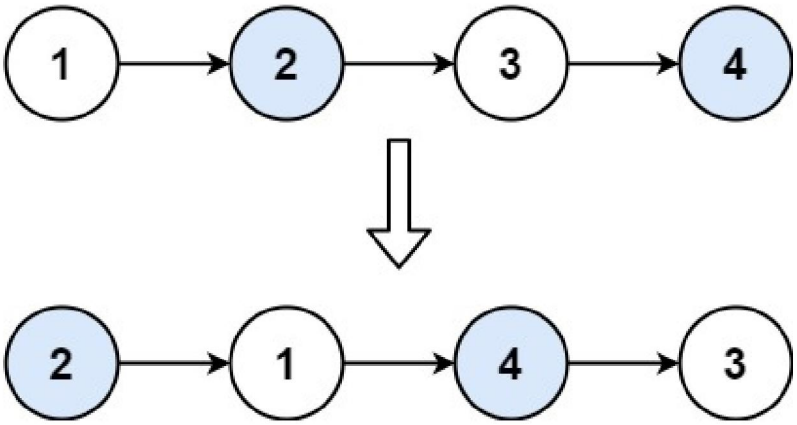
Testcase Test Result

24. Swap Nodes in Pairs

Medium Topics Companies

Given a linked list, swap every two adjacent nodes and return its head. You must solve the problem without modifying the va list's nodes (i.e., only nodes themselves may be changed.)

Example 1:




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

Example 2:

```
Input: head = []
Output: []
```

Example 3:

```
Input: head = [1]
Output: [1]
```

Constraints:

- The number of nodes in the list is in the range `[0, 100]`.
- `0 <= Node.val <= 100`

Run Submit

Code

Java Auto

```java
/**
 * Definition for singly-linked list.
 * 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; }
 * }
 */
class Solution {
    public ListNode swapPairs(ListNode head) {
      ListNode newH = new ListNode(0);
      newH.next=head;
      ListNode n0=newH;
      ListNode n1;
      ListNode n2;
      ListNode n3;
      while(true) {
        n1=n0.next;  if(n1==null) return newH.next;
        n2=n1.next;  if(n2==null) return newH.next;
        n3=n2.next;

        n1.next=n3;
        n2.next=n1;
        n0.next=n2;

        n0=n1;

      }//end while

    }
}
```

Saved to local

Testcase Test Result

Description Editorial Solutions Submissions

25. Reverse Nodes in k-Group

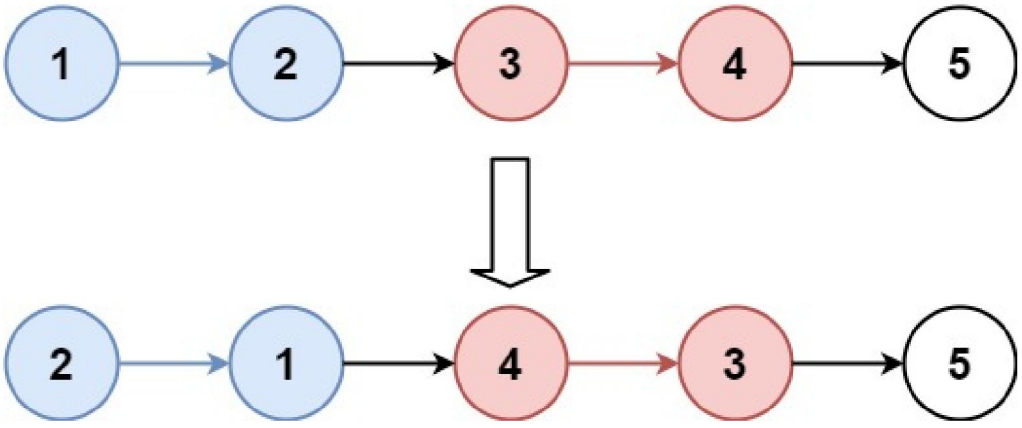
Hard Topics Companies

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

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

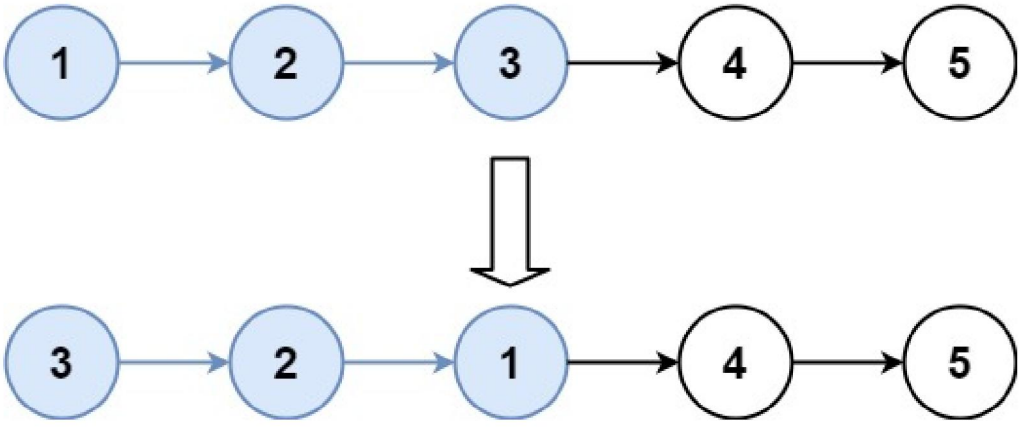
You may not alter the values in the list's nodes, only nodes themselves may be changed.

Example 1:




```
Input: head = [1,2,3,4,5], k = 2
Output: [2,1,4,3,5]
```

Example 2:



Problem List

Run Submit

Code

Java Auto

```java
 */
class Solution {
    public ListNode reverseKGroup(ListNode head, int k) {
      if (head==null) return null;
      ListNode last = head;
      int cnt = 1;
      while (cnt++ < k && last != null) {
        last = last.next;
      }
      if (cnt < k || last ==null ) return head;
      ListNode next = last.next;
      last.next = null;

      ListNode leftTail = reverse(head, last);
      ListNode rightHead = reverseKGroup(next, k);
      leftTail.next = rightHead;


      return last;
    }

    // public ListNode reverse(ListNode head, ListNode last ) {
    //   Stack<ListNode> stack = new Stack<>();
    //   ListNode cur = head;
    //   while (cur != last) {
    //     stack.push(cur);
    //     cur=cur.next;
    //   }
    //   while(!stack.empty()) {
    //     ListNode t = stack.pop();
    //     cur.next = t;
    //     cur = t;
    //   }
    //   return cur;
    // }
    public ListNode reverse(ListNode head, ListNode last ) { //last is not used
      ListNode pre = null;
      last.next = null;
      ListNode cur = head;
      while (cur != null) {
        ListNode next = cur.next;
        cur.next = pre;
        pre = cur;
        cur = next;

      }
      return head;
    }
}
```

Saved to local

Testcase Test Result

26. Remove Duplicates from Sorted Array

Easy Topics Companies Hint

Given an integer array `nums` sorted in **non-decreasing order**, remove the duplicates **in-place** such that each unique elemen **once**. The **relative order** of the elements should be kept the **same**. Then return *the number of unique elements in* `nums`.

Consider the number of unique elements of `nums` to be `k`, to get accepted, you need to do the following things:

- Change the array `nums` such that the first `k` elements of `nums` contain the unique elements in the order they were presen initially. The remaining elements of `nums` are not important as well as the size of `nums`.
- Return `k`.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;
for (int i = 0; i < k; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be **accepted**.

Example 1:

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

Example 2:

```
Input: nums = [0,0,1,1,1,2,2,3,3,4]
```

Problem List

Run Submit

Code

Java Auto

```java
class Solution {
    public int removeDuplicates(int[] nums) {
      if(nums.length<2) return nums.length;
      int ret=1;
      for(int i=1; i<nums.length; i++ ) {
        if(nums[i]==nums[i-1])
          continue;
        nums[ret++]=nums[i];
      }
      return ret;

    }
}
```

Saved to local

Testcase Test Result

27. Remove Element

Easy Topics Companies Hint

Given an integer array `nums` and an integer `val`, remove all occurrences of `val` in `nums` **in-place**. The order of the element changed. Then return *the number of elements in* `nums` *which are not equal to* `val`.

Consider the number of elements in `nums` which are not equal to `val` be `k`, to get accepted, you need to do the following t

- Change the array `nums` such that the first `k` elements of `nums` contain the elements which are not equal to `val`. The rem elements of `nums` are not important as well as the size of `nums`.
- Return `k`.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int val = ...; // Value to remove
int[] expectedNums = [...]; // The expected answer with correct length.
                            // It is sorted with no values equaling val.

int k = removeElement(nums, val); // Calls your implementation

assert k == expectedNums.length;
sort(nums, 0, k); // Sort the first k elements of nums
for (int i = 0; i < actualLength; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be **accepted**.

Example 1:

```
Input: nums = [3,2,2,3], val = 3
Output: 2, nums = [2,2,_,_]
Explanation: Your function should return k = 2, with the first two elements of nums being
It does not matter what you leave beyond the returned k (hence they are underscores).
```

Example 2:

Problem List

Run Submit

Code

Java Auto

```java
class Solution {
    public int removeElement(int[] nums, int val) {
      int ret=0;
      for(int i=0; i<nums.length; i++) {
        if(nums[i]!=val) {
          nums[ret++]=nums[i];
        }

      }//end for
      return ret;

    }
}
```

Saved to local

Testcase Test Result

28. Find the Index of the First Occurrence in a String

Easy Topics Companies

Given two strings `needle` and `haystack`, return the index of the first occurrence of `needle` in `haystack`, or `-1` if `needle` i `haystack`.

Example 1:

```
Input: haystack = "sadbutsad", needle = "sad"
Output: 0
Explanation: "sad" occurs at index 0 and 6.
The first occurrence is at index 0, so we return 0.
```

Example 2:

```
Input: haystack = "leetcode", needle = "leeto"
Output: -1
Explanation: "leeto" did not occur in "leetcode", so we return -1.
```

Constraints:

- `1 <= haystack.length, needle.length <= 10^4`
- `haystack` and `needle` consist of only lowercase English characters.

Seen this question in a real interview before? 1/4

Yes No

Accepted 2.1M Submissions 5.1M Acceptance Rate 40.9%

Topics

Companies

5.1K 164

Run Submit

Code

Java Auto

```java
class Solution {
    public int strStr(String haystack, String needle) {
      int lenh=haystack.length();
      int lenn=needle.length();
      for(int i=0; i+lenn<=lenh; i++) {
        int j=0;
        for(j=0; j<lenn; j++ ) {
          if(haystack.charAt(i+j)!=needle.charAt(j))
            break;
        }
        if(j==lenn)
          return i;

      }//end for
      return -1;

    }
}
```

Saved to local

Testcase Test Result

29. Divide Two Integers

Medium Topics Companies

Given two integers `dividend` and `divisor`, divide two integers **without** using multiplication, division, and mod operator.

The integer division should truncate toward zero, which means losing its fractional part. For example, `8.345` would be trunca and `-2.7335` would be truncated to `-2`.

Return *the **quotient** after dividing* `dividend` *by* `divisor`.

Note: Assume we are dealing with an environment that could only store integers within the **32-bit** signed integer range: `[-2 1]`. For this problem, if the quotient is **strictly greater than** $2^{31} - 1$, then return $2^{31} - 1$, and if the quotient is **strictly less** then return -2^{31}.

Example 1:

```
Input: dividend = 10, divisor = 3
Output: 3
Explanation: 10/3 = 3.33333.. which is truncated to 3.
```

Example 2:

```
Input: dividend = 7, divisor = -3
Output: -2
Explanation: 7/-3 = -2.33333.. which is truncated to -2.
```

Constraints:

- $-2^{31} <= dividend, divisor <= 2^{31} - 1$
- `divisor != 0`

Seen this question in a real interview before? 1/4

Yes No

4.8K 164

Run Submit

Code

Java Auto

```java
class Solution {
    public int divide(int dividend, int divisor) {
        if (dividend == Integer.MIN_VALUE && divisor == -1 )
            return Integer.MAX_VALUE;
        int sign = 1;
        if (dividend < 0 && divisor >0 || dividend > 0 && divisor < 0 )
            sign = -1;
        long ldividend = Math.abs((long)dividend);
        long ldivisor = Math.abs((long)divisor);
        if (ldividend < ldivisor) return 0;
        long ret = 0;

        while (ldividend >= ldivisor) {
            long tmp = ldivisor;
            int p = 1;
            while (ldividend >= ( tmp << 1)) {
                tmp = tmp << 1;
                p = p << 1;
            }
            ret += p;
            ldividend -= tmp;
        }
        if (sign < 0 )
            ret=-ret;
        return (int)ret;

    }
}
```

Saved to local

Testcase Test Result

30. Substring with Concatenation of All Words

Hard Topics Companies

You are given a string `s` and an array of strings `words`. All the strings of `words` are of **the same length**.

A **concatenated substring** in `s` is a substring that contains all the strings of any permutation of `words` concatenated.

- For example, if `words = ["ab","cd","ef"]`, then `"abcdef"`, `"abefcd"`, `"cdabef"`, `"cdefab"`, `"efabcd"`, and `"efcdab"` concatenated strings. `"acdbef"` is not a concatenated substring because it is not the concatenation of any permutation of

Return *the starting indices of all the concatenated substrings in* `s`. You can return the answer in **any order**.

Example 1:

```
Input: s = "barfoothefoobarman", words = ["foo","bar"]
Output: [0,9]
Explanation: Since words.length == 2 and words[i].length == 3, the concatenated substring
of length 6.
The substring starting at 0 is "barfoo". It is the concatenation of ["bar","foo"] which i
permutation of words.
The substring starting at 9 is "foobar". It is the concatenation of ["foo","bar"] which i
permutation of words.
The output order does not matter. Returning [9,0] is fine too.
```

Example 2:

```
Input: s = "wordgoodgoodgoodbestword", words = ["word","good","best","word"]
Output: []
Explanation: Since words.length == 4 and words[i].length == 4, the concatenated substring
of length 16.
There is no substring of length 16 in s that is equal to the concatenation of any permuta
words.
We return an empty array.
```

Example 3:

```
Input: s = "barfoofoobarthefoobarman", words = ["bar","foo","the"]
Output: [6,9,12]
Explanation: Since words.length == 3 and words[i].length == 3, the concatenated substring
```

Problem List

Run Submit

Code

Java Auto

```java
class Solution {
    public List<Integer> findSubstring(String s, String[] words) {
      List<Integer> ret=new ArrayList<>();
      if(s.length()==0 || words.length==0) return ret;
      Map<String, Integer> map  = new HashMap<>();
      for(String s1 : words) {
        //map.computeIfAbsent(s,0);
        map.compute(s1,(x,y)->(y==null) ? 1 : y+1);
      }
      int n=s.length();
      int m=words.length;
      int l=words[0].length();
      for(int i=0; i<=n-l*m; i++) {
        Map<String, Integer> maptmp = new HashMap<>();
        int j;
        for( j=0; j<m; j++) {
          String stmp=s.substring(i+j*l, i+j*l+l);
          if(!map.containsKey(stmp)) break;
          maptmp.compute(stmp, (x,y)->(y==null)?1:y+1);
          if(maptmp.get(stmp)>map.get(stmp)) break;
        }
        if(j==m) ret.add(i);
      }
      return ret;

    }
}
```

Saved to local

Testcase Test Result

31. Next Permutation

Medium Topics Companies

A **permutation** of an array of integers is an arrangement of its members into a sequence or linear order.

- For example, for `arr = [1,2,3]`, the following are all the permutations of `arr`: `[1,2,3], [1,3,2], [2, 1, 3], [2, 3, [3,1,2], [3,2,1]`.

The **next permutation** of an array of integers is the next lexicographically greater permutation of its integer. More formally, permutations of the array are sorted in one container according to their lexicographical order, then the **next permutation** of permutation that follows it in the sorted container. If such arrangement is not possible, the array must be rearranged as the l order (i.e., sorted in ascending order).

- For example, the next permutation of `arr = [1,2,3]` is `[1,3,2]`.
- Similarly, the next permutation of `arr = [2,3,1]` is `[3,1,2]`.
- While the next permutation of `arr = [3,2,1]` is `[1,2,3]` because `[3,2,1]` does not have a lexicographical larger rearra

Given an array of integers `nums`, *find the next permutation of* `nums`.

The replacement must be **in place** and use only constant extra memory.

Example 1:

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

Example 2:

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

Example 3:

```
Input: nums = [1,1,5]
Output: [1,5,1]
```

Constraints:

Problem List

Run Submit

Code

Java Auto

```java
class Solution {
    public void nextPermutation(int[] nums) {
      int i=nums.length-1;
      //from right to left, find the 1st case where nums[i-1]<nums[i]
      while(i>0 && nums[i-1]>=nums[i]) i--;
      int left=i;
      if(left>0) {
        i=nums.length-1;
        while(i>=left && nums[i]<=nums[left-1]) i--;
        int t=nums[left-1];
        nums[left-1]=nums[i];
        nums[i]=t;
      }
      int right=nums.length-1;
      //now, [left, right] are desc ordered, reverse them
      while(left<right) {
        int t=nums[left];
        nums[left]=nums[right];
        nums[right]=t;
        left++;
        right--;
      }

    }
}


// 如果从末尾往前看，数字逐渐变大，到了2时才减小的，然后再从后往前找第一个比2大的数字，是3，那么我们交换2和3，再把此时3后一下即可，步骤如下：

// 1    2    7    4    3    1

// 1    2    7    4    3    1

// 1    3    7    4    2    1

// 1    3    1    2    4    7
```

Saved to local

Testcase Test Result

32. Longest Valid Parentheses

Hard Topics Companies

Given a string containing just the characters `'('` and `')'`, return *the length of the longest valid (well-formed) parentheses su*

Example 1:

```
Input: s = "(()"
Output: 2
Explanation: The longest valid parentheses substring is "()".
```

Example 2:

```
Input: s = ")()())"
Output: 4
Explanation: The longest valid parentheses substring is "()()".
```

Example 3:

```
Input: s = ""
Output: 0
```

Constraints:

- `0 <= s.length <= 3 * 10^4`
- `s[i]` is `'('`, or `')'`.

Seen this question in a real interview before? 1/4

Yes No

Accepted 670.7K Submissions 2M Acceptance Rate 33.4%

Topics

11.9K 41

Problem List

Run Submit

Code

Java Auto

```java
  public int sol1(String s) {
    Stack<Integer> stack = new Stack<>();
    int start = 0;
    int ret = 0;
    for (int i = 0; i < s.length(); i++) {
      char c = s.charAt(i);
      if (c == '(') {
        stack.push(i);
      } else if (c == ')' ) {
        if (stack.empty()) {
          start = i + 1;
        } else {
          stack.pop();
          if (stack.empty())
            ret = Math.max(ret, i - start + 1);
          else
            ret = Math.max(ret, i - stack.peek());
        }
      }
    }
    return ret;
  }

  public int longestValidParentheses(String s) {
    // return sol1(s);
    return sol2(s);
  }

  public int sol2(String s) {
    int maxans = 0;
    Stack<Integer> stack = new Stack<>();
    //the top of the stack is the last index that mismatch
    stack.push(-1);
    for (int i = 0; i < s.length(); i++) {
        if (s.charAt(i) == '(') {
          stack.push(i);
          continue;
        }
        stack.pop();
        //now it's ')'
        if (stack.empty()) {
            //current ')' mismatch, push the current index
            stack.push(i);
        } else {
            maxans = Math.max(maxans, i - stack.peek());
        }

    }
    return maxans;
```

Saved to local

Testcase Test Result

Description Editorial Solutions Submissions

33. Search in Rotated Sorted Array

Medium Topics Companies

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` (`1 <= k < nums.length`) su resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For examp `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of* `target` *if it is in* `nums`, *or* `-1` *if* `nums`.

You must write an algorithm with `O(log n)` runtime complexity.

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
```

Example 3:

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

Constraints:

- `1 <= nums.length <= 5000`
- `-10^4 <= nums[i] <= 10^4`
- All values of `nums` are **unique**.
- `nums` is an ascending array that is possibly rotated.

24.8K 161

Problem List

Run Submit

Code

Java Auto

```java
class Solution {
    public int search(int[] nums, int target) {
      int left=0;
      int right=nums.length-1;
      while(left<=right) {

        int mid=(right-left)/2+left;
        //System.out.println("before"+left +" "+mid + " "+right);
        if(nums[mid]==target) return mid;
        if(nums[mid]>=nums[left]) {
          if(nums[mid]>target && target>=nums[left]) right=mid-1;
          else left=mid+1;
        }else {
          if(nums[mid]<target && target<=nums[right]) left=mid+1;
          else right=mid-1;
        }
        //System.out.println("after"+left +" "+right);
      }//end while

      return -1;
    }
}
```

Saved to local

Testcase Test Result

34. Find First and Last Position of Element in Sorted Array

Medium Topics Companies

Given an array of integers `nums` sorted in non-decreasing order, find the starting and ending position of a given `target` val

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

You must write an algorithm with `O(log n)` runtime complexity.

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]
```

Example 3:

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

Constraints:

- `0 <= nums.length <= 10^5`
- `-10^9 <= nums[i] <= 10^9`
- `nums` is a non-decreasing array.
- `-10^9 <= target <= 10^9`

Problem List

Run Submit

Code

Java Auto

```java
class Solution {
    public int[] searchRange(int[] nums, int target) {
      int left=0; int right=nums.length-1;
      int[] ret=new int[]{-1,-1};
      if(nums.length==0)
        return ret;
      while(left<right) {
        int mid=(left+right)/2;
        if(nums[mid]<target)
          left=mid+1;  //in case left==mid, always +1
        else
          right=mid;
      }

      if(nums[left]!=target) return ret;

      right=nums.length-1;
      ret[0]=left;

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

      return ret;

    }
}
```

Saved to local

Testcase Test Result

35. Search Insert Position

Easy Topics Companies

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index whe if it were inserted in order.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

- `1 <= nums.length <= 10^4`
- `-10^4 <= nums[i] <= 10^4`
- `nums` contains **distinct** values sorted in **ascending** order.
- `-10^4 <= target <= 10^4`

Seen this question in a real interview before? 1/4

Yes No

Problem List

Run Submit

Code

Java Auto

```java
class Solution {
    public int searchInsert(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;
        if(nums[mid]<target)
          left=mid+1;
        else
          right=mid-1;

      }//while
      return left;
    }


}
```

Saved to local Upgrade to Cloud Saving

Testcase Test Result

36. Valid Sudoku

Medium Topics Companies

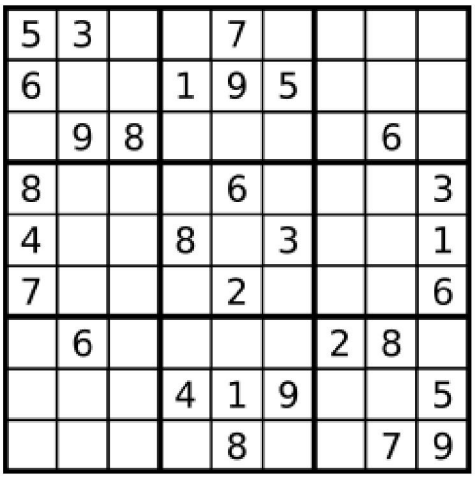
Determine if a `9 x 9` Sudoku board is valid. Only the filled cells need to be validated **according to the following rules**:

1. Each row must contain the digits `1-9` without repetition.
2. Each column must contain the digits `1-9` without repetition.
3. Each of the nine `3 x 3` sub-boxes of the grid must contain the digits `1-9` without repetition.

Note:

- A Sudoku board (partially filled) could be valid but is not necessarily solvable.
- Only the filled cells need to be validated according to the mentioned rules.

Example 1:



```
Input: board =
[["5","3",".",".","7",".",".",".","."]
,["6",".",".","1","9","5",".",".","."]
,[".","9","8",".",".",".",".","6","."]
,["8",".",".",".","6",".",".",".","3"]
,["4",".",".","8",".","3",".",".","1"]
,["7",".",".",".","2",".",".",".","6"]
```

Run Submit

Code

Java Auto

```java
class Solution {

    public boolean checkRow(char[][] board) {
      for(int i=0; i<9; i++) {
        boolean[] row=new boolean[9];

        for(int j=0; j<9; j++) {
          if(board[i][j]!='.') {
            int val_r=board[i][j]-'1';

            if(row[val_r]) return false;
            else row[val_r]=true;
          }

        }//for
      }
      return true;
    }

    public boolean checkCol(char[][] board) {
      for(int i=0; i<9; i++) {
        boolean[] col=new boolean[9];

        for(int j=0; j<9; j++) {
          if(board[j][i]!='.') {
            int val_c=board[j][i]-'1';
            if(col[val_c]) return false;
            else col[val_c]=true;
          }
        }//for
      }
      return true;
    }

    public boolean checkBox(char[][] board) {
      for (int base_i=0; base_i<9; base_i+=3) {
        for (int base_j=0; base_j<9; base_j+=3) {
          boolean[] box=new boolean[9];
          for (int i=0; i<3; i++) {
            for (int j=0; j<3; j++) {
              int x = base_i+i;
              int y=base_j+j;
              if(board[x][y]!='.') {
                int val = board[x][y]-'1';
                if(box[val]) return false;
                else box[val]=true;
              }
            }
```

Saved to local

Testcase Test Result

Run Submit

Code

Java Auto

```java
    public boolean checkBox(char[][] board) {
      for (int base_i=0; base_i<9; base_i+=3) {
        for (int base_j=0; base_j<9; base_j+=3) {
          boolean[] box=new boolean[9];
          for (int i=0; i<3; i++) {
            for (int j=0; j<3; j++) {
              int x = base_i+i;
              int y=base_j+j;
              if(board[x][y]!='.') {
                int val = board[x][y]-'1';
                if(box[val]) return false;
                else box[val]=true;
              }
            }
          }

        }
      }
      return true;
    }



    public boolean sol3(char[][] board) {
      return checkRow(board) && checkCol(board) && checkBox(board);
    }
    public boolean isValidSudoku(char[][] board) {
      // return sol1(board);
      return sol3(board);
    }

    public boolean sol1(char[][] board) {
      for(int i=0; i<9; i++) {
        boolean[] row=new boolean[9];
        boolean[] col=new boolean[9];
        boolean[] cell=new boolean[9];

        int base_r=(i/3)*3;
        int base_c=(i%3)*3;

        for(int j=0; j<9; j++) {

          if(board[i][j]!='.') {
            int val_r=board[i][j]-'1';

            if(row[val_r]) return false;
            else row[val_r]=true;
          }
```

Saved to local

Testcase Test Result

Code

Java Auto

```java
    }
    public boolean isValidSudoku(char[][] board) {
      // return sol1(board);
      return sol3(board);
    }

    public boolean sol1(char[][] board) {
      for(int i=0; i<9; i++) {
        boolean[] row=new boolean[9];
        boolean[] col=new boolean[9];
        boolean[] cell=new boolean[9];

        int base_r=(i/3)*3;
        int base_c=(i%3)*3;

        for(int j=0; j<9; j++) {

          if(board[i][j]!='.') {
            int val_r=board[i][j]-'1';

            if(row[val_r]) return false;
            else row[val_r]=true;
          }

          if(board[j][i]!='.') {
            int val_c=board[j][i]-'1';
            if(col[val_c]) return false;
            else col[val_c]=true;
          }

          int r=base_r+j/3;
          int c=base_c+j%3;
          if(board[r][c]!='.') {
            int val_cell=board[r][c]-'1';
            if(cell[val_cell]) return false;
            else cell[val_cell] = true;
          }

        }//for




      }
      return true;
    }

}
```

Saved to local

Testcase Test Result

37. Sudoku Solver

Hard Topics Companies

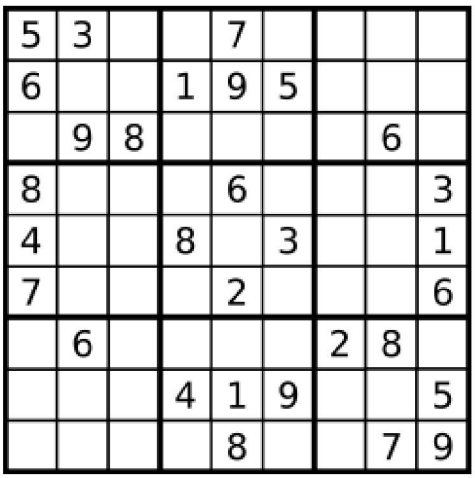
Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules**:

1. Each of the digits `1-9` must occur exactly once in each row.
2. Each of the digits `1-9` must occur exactly once in each column.
3. Each of the digits `1-9` must occur exactly once in each of the 9 `3x3` sub-boxes of the grid.

The `'.'` character indicates empty cells.

Example 1:



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

```
Input: board = [["5","3",".",".","7",".",".",".",".",".],["6",".",".","1","9","5",".",".",".
[".","9","8",".",".",".",".",".","6",".],["8",".",".",".","6",".",".",".","3"],
["4",".",".","8",".","3",".",".",".","1"],["7",".",".",".",".","2",".",".",".","6"],
[".","6",".",".",".",".",".","2","8",".],[".",".",".",".","4","1","9",".",".","5"],
[".",".",".",".",".","8",".",".",".","7","9"]]
Output: [["5","3","4","6","7","8","9","1","2"],["6","7","2","1","9","5","3","4","8"],
["1","9","8","3","4","2","5","6","7"],["8","5","9","7","6","1","4","2","3"],
["4","2","6","8","5","3","7","9","1"],["7","1","3","9","2","4","8","5","6"],
["9","6","1","5","3","7","2","8","4"],["2","8","7","4","1","9","6","3","5"],
```

Problem List

Run Submit

Code

Java Auto

```java
class Solution {
    public void solveSudoku(char[][] board) {
      if(board==null || board.length!=9 || board[0].length!=9) return;
      solve(board);
    }
    public boolean solve(char[][] board) {
      for(int i=0; i<9; i++){
        for(int j=0; j<9; j++) {
          if(board[i][j]=='.') {
            for(char c='1';c<='9';c++) {

              if(valid(board,i,j,c)) {
                board[i][j]=c;
                if(solve(board))
                  return true;
                else
                  board[i][j]='.';
              }
            }
            //no solution return false
            return false;
          }//i,j
        }

      }
      return true;

    }
    public boolean valid(char[][] board, int i, int j, char c) {
      for(int n=0; n<9; n++)
        if(board[i][n]==c) return false;
      for(int n=0; n<9; n++)
        if(board[n][j]==c) return false;
      int left=(i/3)*3;
      int right=left+2;
      int top=(j/3)*3;
      int bot=top+2;
      for(int m=left; m<=right; m++ ) {
        for(int n=top; n<=bot; n++){
          if(board[m][n]==c) return false;
        }
      }
      return true;
    }


}
```

Saved to local

Testcase Test Result

38. Count and Say

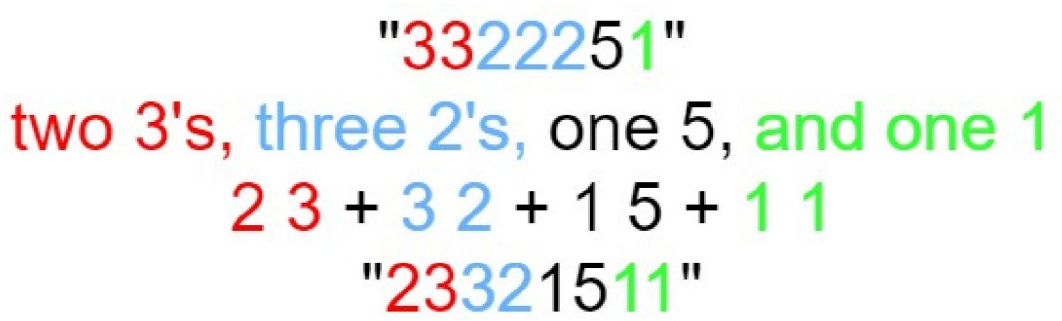
Medium Topics Companies Hint

The **count-and-say** sequence is a sequence of digit strings defined by the recursive formula:

- `countAndSay(1) = "1"`
- `countAndSay(n)` is the way you would "say" the digit string from `countAndSay(n-1)`, which is then converted into a diffe string.

To determine how you "say" a digit string, split it into the **minimal** number of substrings such that each substring contains e unique digit. Then for each substring, say the number of digits, then say the digit. Finally, concatenate every said digit.

For example, the saying and conversion for digit string `"3322251"`:



Given a positive integer `n`, return *the* `n^th` *term of the* ***count-and-say*** *sequence.*

Example 1:

```
Input: n = 1
Output: "1"
Explanation: This is the base case.
```

Example 2:

```
Input: n = 4
Output: "1211"
Explanation:
countAndSay(1) = "1"
countAndSay(2) = say "1" = one 1 = "11"
countAndSay(3) = say "11" = two 1's = "21"
```

Problem List

Run Submit

Code

Java Auto

```java
class Solution {
    public String countAndSay(int n) {
      String s="1";
      int m=1;
      while(m++<n) {
        StringBuilder sb = new StringBuilder();
        int i=1;
        int cnt=1;
        char c=s.charAt(0);
        while(i<s.length()) {
          if(s.charAt(i)!=c) {
            sb.append(Integer.toString(cnt));
            sb.append(c);
            cnt=1;
            c=s.charAt(i);
          } else {
            cnt++;
          }
          i++;
        }
        sb.append(Integer.toString(cnt));
        sb.append(c);

        s=sb.toString();
      }
      return s;
    }
}
```

Saved to local

Testcase Test Result

39. Combination Sum

Medium Topics Companies

Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all **unique combinations** of* `car` *where the chosen numbers sum to* `target`. You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the fre least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to `target` is less than `150` combina given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```

Example 2:

```
Input: candidates = [2,3,5], target = 8
Output: [[2,2,2,2],[2,3,3],[3,5]]
```

Example 3:

```
Input: candidates = [2], target = 1
Output: []
```

Constraints:

- `1 <= candidates.length <= 30`
- `2 <= candidates[i] <= 40`
- All elements of `candidates` are **distinct**.

Problem List

Run Submit

Code

Java Auto

```java
class Solution {

    public List<List<Integer>> combinationSum(int[] candidates, int target) {
      List<List<Integer>> ret = new ArrayList<>();
      //sorting for shortcut only
      Arrays.sort(candidates);
      helper(candidates, ret, new ArrayList<Integer>(), 0, target);
      return ret;
    }
    public void helper(int[] candidates, List<List<Integer>> ret, List<Integer> curList, int start, int target)
      if(target<0)
        return;
      if(target==0) {
        ret.add(new ArrayList<Integer>(curList));
      }

      for(int i=start; i<candidates.length; i++)       {
        if (candidates[i]>target) break; //shotcut
        curList.add(candidates[i]);
        //as you can repeat the same, so don't increase i as start
        helper(candidates, ret, curList, i, target-candidates[i]);
        curList.remove(curList.size()-1);
      }
    }

}

```

Saved to local

Testcase Test Result