

21. 合并两个有序链表

21. 合并两个有序链表

难度 **简单**

👤 944

♡ 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

将两个升序链表合并为一个新的升序链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的。

示例:

输入: 1->2->4, 1->3->4

输出: 1->1->2->3->4->4

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        auto dummy = new ListNode(-1);
        auto p = dummy;
        while(l1 && l2)
        {
            if (l1->val < l2->val)
            {
                p->next = l1;
                p = l1;
                l1 = l1->next;
            }
            else
            {
                p->next = l2;
                p = l2;
                l2 = l2->next;
            }
        }
        if (!l1) l1 = l2;
        while(l1)
        {
            p->next = l1;
            p = l1;
            l1 = l1->next;
        }
        return dummy->next;
    }
};
```

22. 括号生成

22. 括号生成

难度 中等

👤 934

❤ 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

数字 n 代表生成括号的对数，请你设计一个函数，用于能够生成所有可能的并且有效的括号组合。

示例：

输入：n = 3

输出：

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

```
class Solution {
public:
    vector<string> res;
    void solve(int l, int r, int n, string cur)
    {
        if(l == n && r == n)
        {
            res.push_back(cur);
            return;
        }
        if(l < n) solve(l + 1, r, n, cur + "(");
        if(r < l) solve(l, r + 1, n, cur + ")");
    }
    vector<string> generateParenthesis(int n) {
        if(n == 0) return res;
        solve(0, 0, n, "");
        return res;
    }
};
```

23. 合并K个排序链表

23. 合并K个排序链表

难度 困难

👍 566

❤ 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

合并 k 个排序链表，返回合并后的排序链表。请分析和描述算法的复杂度。

示例:

输入:

```
[
  1->4->5,
  1->3->4,
  2->6
]
```

输出: 1->1->2->3->4->4->5->6

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* merge2Lists(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;
        }
        cur->next = (l1 != NULL ? l1 : l2);
        return head->next;
    }




    ListNode* mergeKLists(vector<ListNode*> lists) {
        if (lists.size() == 0)
            return NULL;
        if (lists.size() == 1)
            return lists[0];

        int mid = lists.size() / 2;
        vector<ListNode*> left = vector<ListNode*>(lists.begin(), lists.begin()
+ mid);
        vector<ListNode*> right = vector<ListNode*>(lists.begin() + mid,
lists.end());
        ListNode *l1 = mergeKLists(left);
```

```
        ListNode *l2 = mergeKLists(right);
        return merge2Lists(l1, l2);
    }
};
```

24. 两两交换链表中的节点

24. 两两交换链表中的节点

难度 中等  466  收藏  分享  切换为英文  关注  反馈

给定一个链表，两两交换其中相邻的节点，并返回交换后的链表。

你不能只是单纯的改变节点内部的值，而是需要实际的进行节点交换。

示例:

给定 1->2->3->4，你应该返回 2->1->4->3。

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {
        ListNode* dummy = new ListNode(0);
        dummy->next = head;
        ListNode* cur = dummy;

        while (cur != NULL) {
            ListNode* first = cur->next;
            if (first == NULL)
                break;






            ListNode* second = first->next;
            if (second == NULL)
                break;

            // 按照一定的次序，交换相邻的两个结点。
            cur->next = second;
            first->next = second->next;
            second->next = first;

            cur = first;
        }
        return dummy->next;
    }
};
```

25. K 个一组翻转链表

25. K 个一组翻转链表

难度 困难  442  收藏  分享  切换为英文  关注  反馈

给你一个链表，每 k 个节点一组进行翻转，请你返回翻转后的链表。

k 是一个正整数，它的值小于或等于链表的长度。

如果节点总数不是 k 的整数倍，那么请将最后剩余的节点保持原有顺序。

示例：

给你这个链表：1→2→3→4→5

当 $k = 2$ 时，应当返回：2→1→4→3→5

当 $k = 3$ 时，应当返回：3→2→1→4→5

说明：

- 你的算法只能使用常数的额外空间。
- 你不能只是单纯的改变节点内部的值，而是需要实际进行节点交换。

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* reverseKGroup(ListNode* head, int k) {
        ListNode* dummy = new ListNode(0);
        dummy->next = head;
        ListNode* cur = dummy;

        while (cur != NULL) {
            ListNode* first = cur->next;
            ListNode* end = cur;

            for (int i = 0; i < k && end != NULL; i++)
                end = end->next;

            if (end == NULL)
                break;
            //change;
            ListNode* p1 = first;
            ListNode* p2 = first->next;
            while (p1 != end) {
                ListNode* new_p2 = p2->next;
                p2->next = p1;
                p1 = p2;
            }
        }
    }
};
```

```
        p2 = new_p2;
    }

    first -> next = p2;
    cur -> next = end;
    cur = first;
}
return dummy -> next;
};
```

26. 删除排序数组中的重复项

26. 删除排序数组中的重复项

难度 **简单**

👍 1388

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个排序数组，你需要在 **原地** 删除重复出现的元素，使得每个元素只出现一次，返回移除后数组的新长度。

不要使用额外的数组空间，你必须在 **原地** 修改输入数组 并在使用 $O(1)$ 额外空间的条件下完成。

示例 1:

给定数组 `nums = [1,1,2]`，

函数应该返回新的长度 `2`，并且原数组 `nums` 的前两个元素被修改为 `1, 2`。

你不需要考虑数组中超出新长度后面的元素。

示例 2:

给定 `nums = [0,0,1,1,1,2,2,3,3,4]`，

函数应该返回新的长度 `5`，并且原数组 `nums` 的前五个元素被修改为 `0, 1, 2, 3, 4`。

你不需要考虑数组中超出新长度后面的元素。

说明:

为什么返回数值是整数，但输出的答案是数组呢？

请注意，输入数组是以「引用」方式传递的，这意味着在函数里修改输入数组对于调用者是可见的。

你可以想象内部操作如下:

```
// nums 是以“引用”方式传递的。也就是说，不对实参做任何拷贝
int len = removeDuplicates(nums);

// 在函数里修改输入数组对于调用者是可见的。
```

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if(nums.empty()) return 0;
        int k = 1;
        for(int i = 1; i < nums.size(); i++)
        {
            if(nums[i] != nums[i - 1])
                nums[k++] = nums[i];
        }
        return k;
    }
};
```

27. 移除元素

27. 移除元素

难度 **简单** 520 收藏 分享 切换为英文 关注 反馈

给你一个数组 *nums* 和一个值 *val*，你需要 **原地** 移除所有数值等于 *val* 的元素，并返回移除后数组的新长度。

不要使用额外的数组空间，你必须仅使用 $O(1)$ 额外空间并 **原地** 修改输入数组。

元素的顺序可以改变。你不需要考虑数组中超出新长度后面的元素。

示例 1:

给定 *nums* = [3,2,2,3], *val* = 3,

函数应该返回新的长度 2，并且 *nums* 中的前两个元素均为 2。

你不需要考虑数组中超出新长度后面的元素。

示例 2:

给定 *nums* = [0,1,2,2,3,0,4,2], *val* = 2,

函数应该返回新的长度 5，并且 *nums* 中的前五个元素为 0, 1, 3, 0, 4。

注意这五个元素可为任意顺序。

你不需要考虑数组中超出新长度后面的元素。

说明:

为什么返回数值是整数，但输出的答案是数组呢？

请注意，输入数组是以「引用」方式传递的，这意味着在函数里修改输入数组对于调用者是可见的。

你可以想象内部操作如下:

```
// nums 是以“引用”方式传递的。也就是说，不对实参作任何拷贝
```

```

class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        int k = 0;
        for(int i = 0 ; i < nums.size(); i++)
            if(nums[i] != val)
                nums[k++] = nums[i];

        return k;
    }
};

```

28. 实现 strStr()

28. 实现 strStr()

难度 **简单** 412 收藏 分享 切换为英文 关注 反馈

实现 `strStr()` 函数。

给定一个 haystack 字符串和一个 needle 字符串，在 haystack 字符串中找出 needle 字符串出现的第一个位置 (从0开始)。如果不存在，则返回 -1。

示例 1:

输入: haystack = "hello", needle = "ll"
输出: 2

示例 2:

输入: haystack = "aaaaa", needle = "bba"
输出: -1

说明:

当 `needle` 是空字符串时，我们应当返回什么值呢？这是一个在面试中很好的问题。

对于本题而言，当 `needle` 是空字符串时我们应当返回 0。这与C语言的 `strstr()` 以及 Java的 `indexOf()` 定义相符。

```

class Solution {
public:
    int strStr(string haystack, string needle) {
        int n = haystack.size(), m = needle.size();
        for(int i = 0 ; i < n - m + 1; i++)
        {
            bool flag = true;
            for(int j = 0; j < m; j++)
                if(haystack[i + j] != needle[j])
                {
                    flag = false;
                    break;
                }
            if(flag) return i;
        }
    }
};







```



```
    }  
    return -1;  
}  
};
```

29. 两数相除

29. 两数相除

难度 中等  315  收藏  分享  切换为英文  关注  反馈

给定两个整数，被除数 `dividend` 和除数 `divisor`。将两数相除，要求不使用乘法、除法和 `mod` 运算符。

返回被除数 `dividend` 除以除数 `divisor` 得到的商。

整数除法的结果应当截去 (`truncate`) 其小数部分，例如：`truncate(8.345) = 8` 以及 `truncate(-2.7335) = -2`

示例 1:

```
输入: dividend = 10, divisor = 3  
输出: 3  
解释: 10/3 = truncate(3.33333...) = truncate(3) = 3
```

示例 2:

```
输入: dividend = 7, divisor = -3  
输出: -2  
解释: 7/-3 = truncate(-2.33333...) = -2
```

提示:

- 被除数和除数均为 32 位有符号整数。
- 除数不为 0。
- 假设我们的环境只能存储 32 位有符号整数，其数值范围是 $[-2^{31}, 2^{31} - 1]$ 。本题中，如果除法结果溢出，则返回 $2^{31} - 1$ 。

```
class Solution {  
public:  
    int divide(int dividend, int divisor) {  
        const int HALF_INT_MIN = -1073741824;  
        int x = dividend, y = divisor;  
  
        bool sign = (x > 0) ^ (y > 0);  
  
        if (x > 0) x = -x;  
        if (y > 0) y = -y;  
  
        vector<pair<int, int>> ys;  
  
        for (int t1 = y, t2 = -1; t1 >= x; t1 += t1, t2 += t2) {
```

```

        ys.emplace_back(t1, t2);
        if (t1 < HALF_INT_MIN)
            break;
    }

    int ans = 0;
    for (int i = ys.size() - 1; i >= 0; i--)
        if (x <= ys[i].first) {
            x -= ys[i].first;
            ans += ys[i].second;
        }



    if (!sign) {
        if (ans == INT_MIN)
            return INT_MAX;
        ans = -ans;
    }

    return ans;
}
};

```

30. 串联所有单词的子串

30. 串联所有单词的子串

难度 **困难**  245  收藏  分享  切换为英文  关注  反馈

给定一个字符串 **s** 和一些长度相同的单词 **words**。找出 **s** 中恰好可以由 **words** 中所有单词串联形成的子串的起始位置。

注意子串要与 **words** 中的单词完全匹配，中间不能有其他字符，但不需要考虑 **words** 中单词串联的顺序。

示例 1:

输入:

```

s = "barfoothefoobarman",
words = ["foo","bar"]

```

输出: [0,9]

解释:

从索引 0 和 9 开始的子串分别是 "barfoo" 和 "foobar" 。
输出的顺序不重要, [9,0] 也是有效答案。

示例 2:

输入:

```

s = "wordgoodgoodgoodbestword",
words = ["word","good","best","word"]

```

输出: []

```

class Solution {
public:
    int check(string s, int begin, int n, int len, int tot,

```

```

        unordered_map<string, int>& wc, vector<int>& ans) {
// 寻找以begin开始的子串的所有匹配，并返回下一次开始匹配的位置。
unordered_map<string, int> vis;
int count = 0;
for (int i = begin; i < n - len + 1; i += len) {
    string candidate = s.substr(i, len);
    if (wc.find(candidate) == wc.end())
        // 遇到不合法的候选单词，直接返回下一次的开始位置为i+len。
        return i + len;

    while (vis[candidate] == wc[candidate]) {
        // 遇到多余的候选单词，不断从begin开始删除候选单词，直到当前候选单词合法。
        vis[s.substr(begin, len)]--;
        count--;
        begin += len;
    }
    vis[candidate]++; // 插入候选单词。
    count++;
    if (count == tot) // 找到一个位置，记录答案。
        ans.push_back(begin);
}
return n;
}

vector<int> findSubstring(string s, vector<string>& words) {
    vector<int> ans;
    int n = s.length();
    int tot = words.size();
    if (tot == 0) return ans;

    unordered_map<string, int> wc;

    for(int i = 0; i < tot; i++)
        wc[words[i]]++;
    int len = words[0].length();

    for (int offset = 0; offset < len; offset++) // 枚举划分
        for (int begin = offset; begin < n;
            begin = check(s, begin, n, len, tot, wc, ans));

    return ans;
}
};

```

31. 下一个排列

31. 下一个排列

难度 中等 443 收藏 分享 切换为英文 关注 反馈

实现获取下一个排列的函数，算法需要将给定数字序列重新排列成字典序中下一个更大的排列。

如果不存在下一个更大的排列，则将数字重新排列成最小的排列（即升序排列）。

必须原地修改，只允许使用额外常数空间。

以下是一些例子，输入位于左侧列，其相应输出位于右侧列。

1, 2, 3 → 1, 3, 2







3, 2, 1 → 1, 2, 3

1, 1, 5 → 1, 5, 1

```
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        int n = nums.size();
        int j = -1;
        for(int i = n - 2; i >= 0; i --)
            if(nums[i] < nums[i + 1])
            {
                j = i;
                break;
            }
        if(j == -1) reverse(nums.begin(), nums.end());
        else
        {
            for(int i = n - 1; i > j; i --)
                if(nums[i] > nums[j])
                {
                    swap(nums[i], nums[j]);
                    break;
                }
            reverse(nums.begin() + j + 1, nums.end());
        }
    }
};
```

32. 最长有效括号

32. 最长有效括号

难度 困难  591  收藏  分享  切换为英文  关注  反馈

给定一个只包含 '(' 和 ')' 的字符串，找出最长的包含有效括号的子串的长度。

示例 1:

输入: "()"
输出: 2
解释: 最长有效括号子串为 "()"

示例 2:

输入: ")()())"
输出: 4
解释: 最长有效括号子串为 "()()"

```
class solution {
public:

    int work(string s)
    {
        int res = 0;
        for(int i = 0,start = 0,cnt = 0;i <s.size();i ++)
            if(s[i] == '(') cnt ++;
            else
            {
                cnt --;
                if(cnt < 0) start = i + 1,cnt = 0;
                else if(!cnt) res = max(res,i - start + 1);
            }
        return res;
    }
    int longestValidParentheses(string s) {
        int res = work(s);
        reverse(s.begin(),s.end());
        for(auto &c : s) c ^= 1;
        return max(res,work(s));
    }
};
```

33. 搜索旋转排序数组

33. 搜索旋转排序数组

难度 中等

👍 603

❤ 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

假设按照升序排序的数组在预先未知的某个点上进行了旋转。

(例如，数组 `[0, 1, 2, 4, 5, 6, 7]` 可能变为 `[4, 5, 6, 7, 0, 1, 2]`)。

搜索一个给定的目标值，如果数组中存在这个目标值，则返回它的索引，否则返回 `-1` 。

你可以假设数组中不存在重复的元素。

你的算法时间复杂度必须是 $O(\log n)$ 级别。

示例 1:

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

输出: `4`

示例 2:

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

输出: `-1`

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        if(nums.empty()) return -1;

        int l = 0, r = nums.size() - 1;
        while(l < r)
        {
            int mid = l + r >> 1;
            if(nums[mid] <= nums.back()) r = mid;
            else l = mid + 1;
        }


        if(target <= nums.back()) r = nums.size() - 1;
        else l = 0, r--;

        while(l < r)
        {
            int mid = l + r >> 1;
            if(nums[mid] >= target) r = mid;
            else l = mid + 1;
        }

        if(nums[l] == target) return l;
        return -1;
    }
};
```

34. 在排序数组中查找元素的第一个和最后一个位置

34. 在排序数组中查找元素的第一个和最后一个位置

难度 中等  379  收藏  分享  切换为英文  关注  反馈

给定一个按照升序排列的整数数组 `nums`，和一个目标值 `target`。找出给定目标值在数组中的开始位置和结束位置。

你的算法时间复杂度必须是 $O(\log n)$ 级别。

如果数组中不存在目标值，返回 `[-1, -1]`。

示例 1:

输入: `nums = [5,7,7,8,8,10]`, `target = 8`
输出: `[3,4]`

示例 2:

输入: `nums = [5,7,7,8,8,10]`, `target = 6`
输出: `[-1,-1]`

```
class Solution {
public:
    vector<int> searchRange(vector<int>& nums, int target) {
        if(nums.empty()) return {-1,-1};

        int l = 0, r = nums.size() - 1;
        while(l < r)
        {
            int mid = l + r >> 1;
            if(nums[mid] >= target) r = mid;
            else l = mid + 1;
        }

        if(nums[r] != target) return {-1,-1};
        int strat = r;
        l = 0, r = nums.size() - 1;
        while(l < r)
        {
            int mid = l + r + 1 >> 1;
            if(nums[mid] <= target) l = mid;
            else r = mid - 1;
        }
        int end = l;
        return {strat,end};
    }
};
```

35. 搜索插入位置

35. 搜索插入位置

难度 简单

👍 480

📖 收藏

🔗 分享

🌐 切换为英文

👤 关注

🗉 反馈

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。

你可以假设数组中无重复元素。

示例 1:

输入: [1,3,5,6], 5
输出: 2

示例 2:

输入: [1,3,5,6], 2
输出: 1

示例 3:

输入: [1,3,5,6], 7
输出: 4

示例 4:

输入: [1,3,5,6], 0
输出: 0

```
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        int l = 0, r = nums.size();
        while(l < r)
        {
            int mid = l + r >> 1;
            if(nums[mid] >= target) r = mid;
            else l = mid + 1;
        }
        return l;
    }
};
```

36. 有效的数独

36. 有效的数独

难度 中等

305

收藏

分享

切换为英文

关注

反馈

判断一个 9x9 的数独是否有效。只需要根据以下规则，验证已经填入的数字是否有效即可。

1. 数字 1-9 在每一行只能出现一次。
2. 数字 1-9 在每一列只能出现一次。
3. 数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。

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

上图是一个部分填充的有效的数独。

数独部分空格内已填入了数字，空白格用 '.' 表示。

```
class Solution {
public:
    bool isValidSudoku(vector<vector<char>>& board) {
        vector<int> row(9), col(9), squ(9); // 使用三个整型数组判重。
        for (int i = 0; i < 9; i++)
            for (int j = 0; j < 9; j++) {
                if (board[i][j] == '.')
                    continue;
                if (board[i][j] < '1' || board[i][j] > '9') return false;
                int num = board[i][j] - '0';

                // 以row[i] & (1 << num)为例，这是判断第i行中，num数字是否出现过。
                // 即row[i]值的二进制表示中，第num位是否是1。
                // 以下col和squ同理。

                if ((row[i] & (1 << num)) ||
                    (col[j] & (1 << num)) ||
                    (squ[(i / 3) * 3 + (j / 3)] & (1 << num)))
                    return false;

                row[i] |= (1 << num);
                col[j] |= (1 << num);
                squ[(i / 3) * 3 + (j / 3)] |= (1 << num);
            }
        return true;
    }
};
```

37. 解数独

37. 解数独

难度 困难

377

收藏

分享

切换为英文

关注

反馈

编写一个程序，通过已填充的空格来解决数独问题。

一个数独的解法需遵循如下规则：

1. 数字 1-9 在每一行只能出现一次。
2. 数字 1-9 在每一列只能出现一次。
3. 数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。

空白格用 '.' 表示。

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

```
class Solution {
public:
    bool row[9][9] = {0}, col[9][9] = {0}, cell[3][3][9] = {0}; //初始化
    void solveSudoku(vector<vector<char>>& board) {
        for(int i = 0; i < 9; i++)
            for(int j = 0; j < 9; j++)
            {
                char c = board[i][j];
                if(c != '.')
                {
                    int t = c - '1';
                    row[i][t] = col[j][t] = cell[i / 3][j / 3][t] = true;
                }
            }

        dfs(board, 0, 0); //左上角开始走
    }

    bool dfs(vector<vector<char>> &board, int x, int y)
    {
        if(y == 9) x++, y = 0; //出界就去下一行
        if(x == 9) return true; //整个数都做完了
        if(board[x][y] != '.') return dfs(board, x, y + 1); //已经填了数，调到下一个

        for(int i = 0; i < 9; i++)
        {
            if(!row[x][i] && !col[y][i] && !cell[x / 3][y / 3][i])
            {
                board[x][y] = '1' + i; //更新状态
                row[x][i] = col[y][i] = cell[x / 3][y / 3][i] = true;
            }
        }
    }
};
```



```

        if(dfs(board,x,y + 1)) return true;
        row[x][i] = col[y][i] = cell[x / 3][y / 3][i] = false;
        board[x][y] = '.'; //恢复状态
    }
}
return false;
}
};

```

38. 外观数列

38. 外观数列

难度 简单  443  收藏  分享  切换为英文  关注  反馈

「外观数列」是一个整数序列，从数字 1 开始，序列中的每一项都是对前一项的描述。前五项如下：

```

1.    1
2.    11
3.    21
4.    1211
5.    111221

```

1 被读作 "one 1"（“一个一”），即 11。
 11 被读作 "two 1s"（“两个一”），即 21。
 21 被读作 "one 2", "one 1"（“一个二”，“一个一”），即 1211。

给定一个正整数 n ($1 \leq n \leq 30$)，输出外观数列的第 n 项。

注意：整数序列中的每一项将表示为一个字符串。

示例 1:

输入：1
 输出："1"
 解释：这是一个基本样例。

示例 2:

输入：4
 输出："1211"
 解释：当 $n = 3$ 时，序列是 "21"，其中我们有 "2" 和 "1" 两组，"2" 可以读作 "12"，也就是出现频次 = 1 而 值 = 2；类似 "1" 可以读作 "11"。所以答案是 "12" 和 "11" 组合在一起，也就是 "1211"。

```

class Solution {
public:
    string countAndSay(int n) {
        string res = "1";
        for(int i = 0; i < n - 1; i++)
        {
            string ns;
            for(int j = 0; j < res.size(); j++)
            {

```

```

        int k = j;
        while(k < res.size() && res[k] == res[j]) k ++;
        ns += to_string(k - j) + res[j];
        j = k;
    }
    res = ns;
}
return res;
}
};

```

39. 组合总和

39. 组合总和

难度 **中等**  603  收藏  分享  切换为英文  关注  反馈

给定一个无重复元素的数组 `candidates` 和一个目标数 `target`，找出 `candidates` 中所有可以使数字和为 `target` 的组合。

`candidates` 中的数字可以无限制重复被选取。

说明：

- 所有数字（包括 `target`）都是正整数。
- 解集不能包含重复的组合。

示例 1:

```

输入：candidates = [2,3,6,7], target = 7,
所求解集为：
[
  [7],
  [2,2,3]
]

```

示例 2:

```

输入：candidates = [2,3,5], target = 8,
所求解集为：
[
  [2,2,2,2],
  [2,3,3],
  [3,5]
]

```

```

class Solution {
public:
    void solve(int i, vector<int>& candidates, int sum,
               vector<int> &ch, int target, vector<vector<int>>& ans) {
        // 注意这里的 ch 是引用。
        if (sum == target) { // 找到目标值，添加答案。
            ans.push_back(ch);
            return;
        }
        if (i == candidates.size() || sum > target) // 超出范围回溯。

```

```

        return;

        solve(i + 1, candidates, sum, ch, target, ans);

        ch.push_back(candidates[i]);
        solve(i, candidates, sum + candidates[i], ch, target, ans);
        ch.pop_back();
    }

    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        vector<vector<int>> ans;
        sort(candidates.begin(), candidates.end());
        vector<int> ch; // ch 记录已选择的数字。
        solve(0, candidates, 0, ch, target, ans);
        return ans;
    }
};

```

40. 组合总和 II

40. 组合总和 II

难度 中等  238  收藏  分享  切换为英文  关注  反馈

给定一个数组 `candidates` 和一个目标数 `target`，找出 `candidates` 中所有可以使数字和为 `target` 的组合。

`candidates` 中的每个数字在每个组合中只能使用一次。

说明：

- 所有数字（包括目标数）都是正整数。
- 解集不能包含重复的组合。

示例 1:

```

输入：candidates = [10,1,2,7,6,1,5], target = 8,
所求解集为：
[
  [1, 7],
  [1, 2, 5],
  [2, 6],
  [1, 1, 6]
]

```

示例 2:

```

输入：candidates = [2,5,2,1,2], target = 5,
所求解集为：
[
  [1,2,2],
  [5]
]

```

```

class Solution {
public:

```

```

void solve(int i, vector<int>& candidates, int sum,
           vector<int>& ch, int target, vector<vector<int>>& ans) {
    // 注意这里的 ch 是引用。
    if (sum == target) {
        ans.push_back(ch);
        return;
    }
    if (i == candidates.size() || sum > target)
        return;

    for (int j = i + 1; j < candidates.size(); j++)
        if (candidates[j] != candidates[i]) {
            // 不用该层的数字时，需要找到下一个不同的数字。
            solve(j, candidates, sum, ch, target, ans);
            break;
        }

    // 选择该层的数字。
    ch.push_back(candidates[i]);
    solve(i + 1, candidates, sum + candidates[i], ch, target, ans);
    ch.pop_back();
}

vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {
    vector<vector<int>> ans;
    sort(candidates.begin(), candidates.end());
    vector<int> ch; // ch 记录选择的数字。
    solve(0, candidates, 0, ch, target, ans);
    return ans;
}
};

```