1. 两数之和

1. 两数之和

难度 简单 8005

给定一个整数数组 `nums` 和一个目标值 `target`，请你在该数组中找出和为目标值的那 **两个** 整数，并返回他们的数组下标。

你可以假设每种输入只会对应一个答案。但是，你不能重复利用这个数组中同样的元素。

示例:

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

因为 nums[0] + nums[1] = 2 + 7 = 9
所以返回 [0, 1]
```

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int,int> hash;
        for(int i = 0;i < nums.size();i ++)
        {
            if(hash.count(target - nums[i])) return {hash[target -nums[i]],i};
            hash[nums[i]] = i;
        }
        return {-1,-1};
    }
};
```

2. 两数相加

2. 两数相加

难度 中等 4156

给出两个 **非空** 的链表用来表示两个非负的整数。其中，它们各自的位数是按照 **逆序** 的方式存储的，并且它们的每个节点只能存储 **一位** 数字。

如果，我们将这两个数相加起来，则会返回一个新的链表来表示它们的和。

您可以假设除了数字 0 之外，这两个数都不会以 0 开头。

示例:

```
输入：(2 -> 4 -> 3) + (5 -> 6 -> 4)
输出：7 -> 0 -> 8
原因：342 + 465 = 807
```

```cpp
class Solution {
public:
    ListNode *addTwoNumbers(ListNode *l1, ListNode *l2)
    {
        ListNode *res = new ListNode(-1);    //添加虚拟头结点，简化边界情况的判断
        ListNode *cur = res;
        int carry = 0;  //表示进位
        while (l1 || l2)
        {
            int n1 = l1 ? l1->val : 0;
            int n2 = l2 ? l2->val : 0;
            int sum = n1 + n2 + carry;
            carry = sum / 10;
            cur->next = new ListNode(sum % 10);
            cur = cur->next;
            if (l1) l1 = l1->next;
            if (l2) l2 = l2->next;
        }
        if (carry) cur->next = new ListNode(1); //如果最高位有进位，则需在最前面补1.
        return res->next;    //返回真正的头结点
    }
};
```

3. 无重复字符的最长子串

3. 无重复字符的最长子串

难度 中等 3437

给定一个字符串，请你找出其中不含有重复字符的 **最长子串** 的长度。

示例 1:

```
输入: "abcabcbb"
输出: 3
解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。
```

示例 2:

```
输入: "bbbbb"
输出: 1
解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。
```

示例 3:

```
输入: "pwwkew"
输出: 3
解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。
     请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不
是子串。
```

```cpp
class Solution {
```

```
public:
    int lengthOfLongestSubstring(string s) {
        unordered_map<char,int> hash;
        int res = 0;
        for(int i = 0,j = 0;j < s.size();j ++)
        {
            hash[s[j]] ++;
            while(hash[s[j]] > 1) hash[s[i ++]] --;
            res = max(res,j - i + 1);
        }
        return res;
    }
};
```

4. 寻找两个有序数组的中位数

4. 寻找两个有序数组的中位数

难度 困难 2408

给定两个大小为 m 和 n 的有序数组 `nums1` 和 `nums2`。

请你找出这两个有序数组的中位数，并且要求算法的时间复杂度为 O(log(m + n))。

你可以假设 `nums1` 和 `nums2` 不会同时为空。

示例 1:

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

则中位数是 2.0
```

示例 2:

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

则中位数是 (2 + 3)/2 = 2.5
```

```
class Solution {
public:
    string intToRoman(int num) {
        int values[] = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
        string reps[] = {"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX",
"V", "IV", "I"};

        string res;
        int
        for(int i = 0; i < 13; i ++ )
            while(num >= values[i])
            {
                num -= values[i];
                res += reps[i];
```

```
            }
        return res;
    }
};
```

5. 最长回文子串

5. 最长回文子串

难度 中等 1985

给定一个字符串 `s`，找到 `s` 中最长的回文子串。你可以假设 `s` 的最大长度为 1000。

示例 1:

```
输入: "babad"
输出: "bab"
注意: "aba" 也是一个有效答案。
```

示例 2:

```
输入: "cbbd"
输出: "bb"
```

```
class Solution {
public:
    string longestPalindrome(string s) {
        string res;
        for(int i = 0;i < s.size();i ++)
        {
            for(int j = i,k = i;j >= 0 && k < s.size() && s[k] == s[j];j --,k
++)
                if(res.size() < k - j + 1)
                    res = s.substr(j,k - j + 1);

            for(int j = i,k = i + 1;j >= 0 && k < s.size() && s[k] == s[j];j -
-,k ++)
                if(res.size() < k - j + 1)
                    res = s.substr(j,k - j + 1);
        }
        return res;
    }
};
```

6. Z 字形变换

6. Z 字形变换

难度 中等 629

将一个给定字符串根据给定的行数，以从上往下、从左到右进行 Z 字形排列。

比如输入字符串为 "LEETCODEISHIRING" 行数为 3 时，排列如下：

```
L   C   I   R
E T O E S I I G
E   D   H   N
```

之后，你的输出需要从左往右逐行读取，产生出一个新的字符串，比如："LCIRETOESIIGEDHN"。

请你实现这个将字符串进行指定行数变换的函数：

```
string convert(string s, int numRows);
```

示例 1:

```
输入：s = "LEETCODEISHIRING", numRows = 3
输出："LCIRETOESIIGEDHN"
```

示例 2:

```
输入：s = "LEETCODEISHIRING", numRows = 4
输出："LDREOEIIECIHNTSG"
解释：

L     D     R
E   O E   I I
E C   I H   N
T     S     G
```

```
class Solution {
public:
    string convert(string s, int n) {
        if(n == 1) return s;
        string res;
        for(int i = 0;i < n;i ++)
        {
            if(!i || i == n -1)
            {
                for(int j = i;j < s.size();j += 2 * (n - 1)) res += s[j];
            }
            else
            {
                for(int j = i,k = 2 * (n - 1) - i;j < s.size() || k < s.size();j
+= 2 *(n - 1),k += 2 * (n - 1))
                {
                    if(j < s.size()) res += s[j];
                    if(k < s.size()) res += s[k];
```

```
                }
            }
        }
        return res;
    }
};
```

7. 整数反转

7. 整数反转

难度 简单 1809

给出一个 32 位的有符号整数，你需要将这个整数中每位上的数字进行反转。

示例 1:

```
输入：123
输出：321
```

示例 2:

```
输入：-123
输出：-321
```

示例 3:

```
输入：120
输出：21
```

```
class Solution {
public:
    int reverse(int x) {
        string s = to_string(x);
        int n = s.size();
        long long t;
        for(int i = 0; i < n / 2; i++) swap(s[i],s[n - i -1]);
        t = stoi(s);
        return x < 0 ? -t : t;
    }
};
```

8. 字符串转换整数 (atoi)

8. 字符串转换整数 (atoi)

难度 中等 654

请你来实现一个 `atoi` 函数，使其能将字符串转换成整数。

首先，该函数会根据需要丢弃无用的开头空格字符，直到寻找到第一个非空格的字符为止。接下来的转化规则如下：

- 如果第一个非空字符为正或者负号时，则将该符号与之后面尽可能多的连续数字字符组合起来，形成一个有符号整数。
- 假如第一个非空字符是数字，则直接将其与之后连续的数字字符组合起来，形成一个整数。
- 该字符串在有效的整数部分之后也可能会存在多余的字符，那么这些字符可以被忽略，它们对函数不应该造成影响。

注意：假如该字符串中的第一个非空格字符不是一个有效整数字符、字符串为空或字符串仅包含空白字符时，则你的函数不需要进行转换，即无法进行有效转换。

在任何情况下，若函数不能进行有效的转换时，请返回 0 。

提示：

- 本题中的空白字符只包括空格字符 `' '` 。
- 假设我们的环境只能存储 32 位大小的有符号整数，那么其数值范围为 $[-2^{31}, 2^{31} - 1]$。如果数值超过这个范围，请返回 INT_MAX ($2^{31} - 1$) 或 INT_MIN (-2^{31}) 。

示例 1:

```
输入："42"
输出：42
```

示例 2:

```
输入："   -42"
输出：-42
解释：第一个非空白字符为 '-'，它是一个负号。
     我们尽可能将负号与后面所有连续出现的数字组合起来，最后得到 -42 。
```

```cpp
class Solution {
public:
    int myAtoi(string str) {
        int ans = 0,i = 0,flag = 1;
        while(str[i] == ' ') i++;
        if(str[i] == '-'){
            flag = -1;
        }
        if(str[i] == '+' || str[i] == '-') i++;
        while(i < str.size() && isdigit(str[i])){
            int r = str[i] - '0';
            if(ans > INT_MAX / 10 || (ans == INT_MAX / 10 && r > 7)){
                return flag > 0 ? INT_MAX : INT_MIN;
            }
```

```
            ans = ans * 10 + r;
            i++;
        }
        return flag > 0 ? ans : -ans;

    }
};
```

9. 回文数

9. 回文数

难度 简单 993

判断一个整数是否是回文数。回文数是指正序（从左向右）和倒序（从右向左）读都是一样的整数。

示例 1:

```
输入：121
输出：true
```

示例 2:

```
输入：-121
输出：false
解释：从左向右读，为 -121 。 从右向左读，为 121- 。因此它不是一个回文数。
```

示例 3:

```
输入：10
输出：false
解释：从右向左读，为 01 。因此它不是一个回文数。
```

进阶:

你能不将整数转为字符串来解决这个问题吗?

```
class Solution {
 public:
    bool isPalindrome(int x) {
        string str = to_string(x);
        int mid = (str.length() % 2 == 1) ? (str.length()/2) :(str.length()/2-1);
        if(str.length() % 2 == 1)
        {
            for(int i = 1;i + mid < str.length(); i ++){
                if(str[mid + i] != str[mid - i])return false;
            }
            return true;
        }
        else{
            for(int i = 1;i + mid < str.length();i ++){
```

```
                if(str[mid + i] != str[mid - i + 1])return false;
            }
            return true;
        }
    }
};
```

10. 正则表达式匹配

10. 正则表达式匹配

难度 困难 1090

给你一个字符串 `s` 和一个字符规律 `p`，请你来实现一个支持 `'.'` 和 `'*'` 的正则表达式匹配。

```
'.' 匹配任意单个字符
'*' 匹配零个或多个前面的那一个元素
```

所谓匹配，是要涵盖 **整个** 字符串 `s` 的，而不是部分字符串。

说明:

- `s` 可能为空，且只包含从 `a-z` 的小写字母。
- `p` 可能为空，且只包含从 `a-z` 的小写字母，以及字符 `.` 和 `*`。

示例 1:

```
输入：
s = "aa"
p = "a"
输出：false
解释："a" 无法匹配 "aa" 整个字符串。
```

示例 2:

```
输入：
s = "aa"
p = "a*"
输出：true
解释：因为 '*' 代表可以匹配零个或多个前面的那一个元素，在这里前面的元素就是 'a'。因此，字符串 "aa" 可被视为 'a' 重复了一次。
```

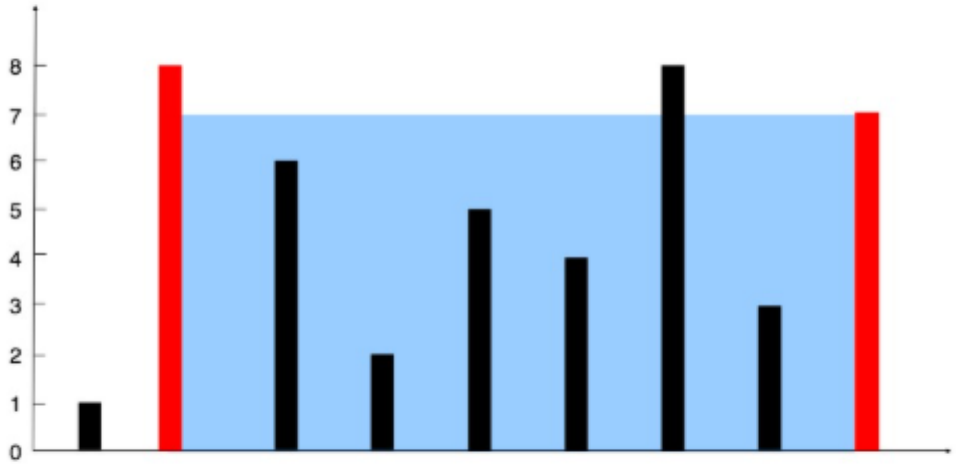
11. 盛最多水的容器

11. 盛最多水的容器

难度 中等 1269

给你 n 个非负整数 a_1，a_2，...，a_n，每个数代表坐标中的一个点 (i, a_i) 。在坐标内画 n 条垂直线，垂直线 i 的两个端点分别为 (i, a_i) 和 $(i, 0)$。找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。

说明：你不能倾斜容器，且 n 的值至少为 2。



图中垂直线代表输入数组 [1,8,6,2,5,4,8,3,7]。在此情况下，容器能够容纳水（表示为蓝色部分）的最大值为 49。

示例：

```
输入：[1,8,6,2,5,4,8,3,7]
输出：49
```

12. 整数转罗马数字

```
class Solution {
public:
    string intToRoman(int num) {
        int values[] = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
        string reps[] = {"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX",
"V", "IV", "I"};

        string res;
        for (int i = 0; i < 13; i ++ )
            while(num >= values[i])
            {
                num -= values[i];
                res += reps[i];
            }
        return res;
    }
};
```

13. 罗马数字转整数

13. 罗马数字转整数

难度 简单 858

罗马数字包含以下七种字符: `I`， `V`， `X`， `L`，`C`，`D` 和 `M`。

字符	数值
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

例如， 罗马数字 2 写做 `II` ，即为两个并列的 1。12 写做 `XII` ，即为 `X` + `II` 。 27 写做 `XXVII`, 即为 `XX` + `V` + `II` 。

通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如 4 不写做 `IIII`，而是 `IV`。数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4 。同样地，数字 9 表示为 `IX`。这个特殊的规则只适用于以下六种情况：

- `I` 可以放在 `V` (5) 和 `X` (10) 的左边，来表示 4 和 9。
- `X` 可以放在 `L` (50) 和 `C` (100) 的左边，来表示 40 和 90。
- `C` 可以放在 `D` (500) 和 `M` (1000) 的左边，来表示 400 和 900。

给定一个罗马数字，将其转换成整数。输入确保在 1 到 3999 的范围内。

示例 1:

```
输入: "III"
输出: 3
```

示例 2:

```
输入: "IV"
输出: 4
```

```
class Solution {
public:
    string intToRoman(int num)
    {
        char* c[4][10] = {
            {"","I","II","III","IV","V","VI","VII","VIII","IX"},
            {"","X","XX","XXX","XL","L","LX","LXX","LXXX","XC"},
            {"","C","CC","CCC","CD","D","DC","DCC","DCCC","CM"},
            {"","M","MM","MMM"}
        };
        string roman;
        roman.append(c[3][num / 1000]);
        roman.append(c[2][num / 100 % 10]);
```

```
            roman.append(c[1][num / 10 % 10]);
            roman.append(c[0][num % 10]);

            return roman;
        }
};
```

14. 最长公共前缀

14. 最长公共前缀

难度 简单 951

编写一个函数来查找字符串数组中的最长公共前缀。

如果不存在公共前缀，返回空字符串 `""`。

示例 1:

```
输入：["flower","flow","flight"]
输出："fl"
```

示例 2:

```
输入：["dog","racecar","car"]
输出：""
解释：输入不存在公共前缀。
```

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& s) {
        string res;
        for(int i = 0 ;i < s.size();i ++)
        {
            for(int j = 0;j < s[0].size();j ++)
            {
                if(s[i][j] != s[i + 1][j] != s[i + 2][j])
                {
                    res = "";
                    break;
                }
                else res =  s[0].substr(0,j);
            }
        }
        return res;
    }
};
```

15. 三数之和

15. 三数之和

难度 中等 1981

给你一个包含 n 个整数的数组 `nums`，判断 `nums` 中是否存在三个元素 a，b，c，使得 $a + b + c = 0$？请你找出所有满足条件且不重复的三元组。

注意：答案中不可以包含重复的三元组。

示例：

```
给定数组 nums = [-1, 0, 1, 2, -1, -4]，

满足要求的三元组集合为：
[
  [-1, 0, 1],
  [-1, -1, 2]
]
```

```cpp
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        vector<vector<int>> res;
        sort(nums.begin(), nums.end());
        for (int st = 0; st < nums.size(); st++) {
            while (st != 0  && st < nums.size() && nums[st] == nums[st - 1])
                st++;
            int l = st + 1, r = nums.size() - 1;
            while (l < r) {
                if (nums[st] + nums[l] + nums[r] == 0) {
                    res.push_back({nums[st], nums[l], nums[r]});
                    do l++;while (l < r && nums[l - 1] == nums[l]);
                    do r--;while (l < r && nums[r] == nums[r + 1]);
                }
                else if (nums[st] + nums[l] + nums[r] < 0) {
                    do l++;while (l < r && nums[l - 1] == nums[l]);
                }
                else {
                    do r--;while (l < r && nums[r] == nums[r + 1]);
                }
            }
        }
        return res;
    }
};
```

16. 最接近的三数之和

16. 最接近的三数之和

难度 中等 398

给定一个包括 *n* 个整数的数组 `nums` 和 一个目标值 `target`。找出 `nums` 中的三个整数，使得它们的和与 `target` 最接近。返回这三个数的和。假定每组输入只存在唯一答案。

例如，给定数组 nums = [-1，2，1，-4]，和 target = 1.

与 target 最接近的三个数的和为 2. (-1 + 2 + 1 = 2).

```
class Solution {
public:
    int threeSumClosest(vector<int>& nums, int target) {
        int ans = nums[0] + nums[1] + nums[2];
        for (int i = 0; i < nums.size(); i++)
            for (int j = i + 1; j < nums.size(); j++)
                for (int k = j + 1; k < nums.size(); k++)
                    if (abs(nums[i] + nums[j] + nums[k] - target) < abs(ans -
target))
                        ans = nums[i] + nums[j] + nums[k];

        return ans;
    }
};
```

17. 电话号码的字母组合

17. 电话号码的字母组合

难度 中等 652

给定一个仅包含数字 `2-9` 的字符串，返回所有它能表示的字母组合。

给出数字到字母的映射如下（与电话按键相同）。注意 1 不对应任何字母。

示例:

> **输入：** "23"
> **输出：** ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

说明:
尽管上面的答案是按字典序排列的，但是你可以任意选择答案输出的顺序。

```
class Solution {
public:
    string chars[8] = {"abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"};

    vector<string> letterCombinations(string digits) {
        if(digits.empty()) return vector<string>();

        vector<string> state(1,"");
        for(auto u : digits)
        {
            vector<string> now;
            for(auto c : chars[u - '2'])
                for(auto s : state)
                    now.push_back(s + c);
            state = now;
        }
        return state;
    }
};
```

18. 四数之和

```
class Solution {
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
        vector<vector<int>> res;
```

```
        sort(nums.begin(), nums.end());
        int n = nums.size();
        for (int i = 0; i < n; i++) {
            while (i > 0 && i < n && nums[i] == nums[i - 1])
                i++;
            for (int j = i + 1; j < n; j++) {
                while (j != i + 1 && j < n && nums[j] == nums[j - 1])
                    j++;
                int l = j + 1, r = n - 1;
                while (l < r) {
                    if (nums[i] + nums[j] + nums[l] + nums[r] == target) {
                        res.push_back({nums[i], nums[j], nums[l], nums[r]});
                        do { l++; } while (l < r && nums[l - 1] == nums[l]);
                        do { r--; } while (l < r && nums[r] == nums[r + 1]);
                    }
                    else if (nums[i] + nums[j] + nums[l] + nums[r] < target) {
                        do { l++; } while (l < r && nums[l - 1] == nums[l]);
                    }
                    else {
                        do { r--; } while (l < r && nums[r] == nums[r + 1]);
                    }
                }
            }
        }
        return res;
    }
};
```

19. 删除链表的倒数第N个节点

19. 删除链表的倒数第N个节点

难度 中等 777

给定一个链表，删除链表的倒数第 *n* 个节点，并且返回链表的头结点。

示例：

```
给定一个链表: 1->2->3->4->5, 和 n = 2.

当删除了倒数第二个节点后，链表变为 1->2->3->5.
```

说明：

给定的 *n* 保证是有效的。

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
```

```
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode *dummy = new ListNode(-1);
        dummy->next = head;
        ListNode *first = dummy, *second = dummy;
        for(int i = 0;i < n;i ++) first = first->next;
        while(first->next)
        {
            first = first->next;
            second = second->next;
        }

        second->next = second->next->next;
        return dummy->next;
    }
};
```

20. 有效的括号

20. 有效的括号

难度 简单 1503

给定一个只包括 `'('`，`')'`，`'{'`，`'}'`，`'['`，`']'` 的字符串，判断字符串是否有效。

有效字符串需满足：

1. 左括号必须用相同类型的右括号闭合。
2. 左括号必须以正确的顺序闭合。

注意空字符串可被认为是有效字符串。

示例 1:

```
输入: "()"
输出: true
```

示例 2:

```
输入: "()[]{}"
输出: true
```

示例 3:

```
输入: "(]"
输出: false
```

示例 4:

```
输入: "([)]"
输出: false
```

示例 5:

```
输入: "{[]}"
输出: true
```

```cpp
class Solution {
public:
    bool isValid(string s) {
        stack<char> stk;
        for (int i = 0; i < s.length(); i++) {
            if (s[i] == '(' || s[i] == '[' || s[i] == '{')
                stk.push(s[i]);
            else if (s[i] == ')') {
                if (stk.empty() || stk.top() != '(')
                    return false;
                stk.pop();
            }
            else if (s[i] == ']') {
                if (stk.empty() || stk.top() != '[')
```

```
                    return false;
                stk.pop();
            }
            else {
                if (stk.empty() || stk.top() != '{')
                    return false;
                stk.pop();
            }
        }
        return stk.empty();
    }
};
```