LeetCode题解

极客学院出版

前言

LeetCode（官网：leetcode.com） 是一个美国的在线编程网站，上面主要收集了各大IT公司的笔试面试题，对于应届毕业生找工作是一个不可多得的好帮手。

这个网站的的好处在于它会告诉你测试数据以及你的输出和正确的输出是什么，方便大家调试学习。目前，支持C++和Java等多种语言。

另外它是支持在线编辑，还提供了一个在线运行环境，可以直接看到运行结果。

本教程适合新手找工作作为参考，作者个人刷题总结，重在思路的理解。

关于作者

柯于旺，CSDN2015年度博客之星，欢迎访问我的博客，地址为：blog.csdn.net/nomasp。

QQ交流群：算法交流群

作者邮箱：nomasp@outlook.com

作者微博：nomasp

版权说明

如果你觉得本教程对你有帮助，欢迎打赏

目录

Two Sum

翻译

给定一个整型数组，找出能相加起来等于一个特定目标数字的两个数。

函数 twoSum 返回这两个相加起来等于目标值的数字的索引，且 index1 必须小于 index2。 请记住你返回的答案（包括 index1 和 index2）都不是从 0 开始的。

你可以假定每个输入都有且仅有一个解决方案。

输入: numbers={2, 7, 11, 15}, target=9
输出: index1=1, index2=2

原文

Given an array of integers, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.

You may assume that each input would have exactly one solution.

Input: numbers={2, 7, 11, 15}, target=9
Output: index1=1, index2=2

C++

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        map<int, int> mapping;
        vector<int> result;
        for (int i = 0; i < nums.size(); i++)
        {
            mapping[nums[i]] = i;
        }
        for (int i = 0; i < nums.size(); i++)
        {
            int searched = target - nums[i];
            if (mapping.find(searched) != mapping.end()
                && mapping.at(searched) != i)
            {
                result.push_back(i + 1);
                result.push_back(mapping[searched] + 1);
                break;
            }
        }
        return result;
    }
};
```

Java

```
public class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<Integer,Integer> map=new HashMap<Integer,Integer>();
        int[] result=new int[2];
        for(int i=0;i<nums.length;i++){
            map.put(nums[i], i);
        }
        for(int i=0;i<nums.length;i++){
            int searched=target-nums[i];
            if(map.containsKey(searched)&&map.get(searched)!=i){
                int index=map.get(searched);
                if(index<i){
                    result[0]=map.get(searched)+1;
                    result[1]=i+1;
                }else{
                    result[0]=i+1;
                    result[1]=map.get(searched)+1;
                }
            }
        }
        return result;
    }
}
```

C#（超时）

```
public static int[] TwoSum(int[] nums, int target)
{
    Dictionary<int, int> map = new Dictionary<int, int>();
    int[] result = new int[2];
    for (int i = 0; i < nums.Length; i++)
    {
        map.Add(i, nums[i]);
    }
    for (int i = 0; i < nums.Length; i++)
    {
        if (nums[i] > target)
            continue;
        int searched = target - nums[i];
        if (map.ContainsValue(searched))
        {
            int index = map.Where(x => x.Value == searched).Select(x => x.Key).FirstOrDefault();
            if (index != i)
            {
                if (index < i)
                {
                    result[0] = index + 1;
                    result[1] = i + 1;
                }
                else
                {
                    result[0] = i + 1;
                    result[1] = index + 1;
                }
            }
        }
    }
    return result;
}
```

Add Two Numbers

翻译

给你两个表示两个非负数字的链表。数字以相反的顺序存储，其节点包含单个数字。将这两个数字相加并将其作为一个链表返回。

输入：(2 -> 4 -> 3) + (5 -> 6 -> 4)
输出：7 -> 0 -> 8

原文

You are given two linked lists representing two non-negative numbers. 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.

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8

C++

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        int carry=0;
        ListNode* listNode=new ListNode(0);
        ListNode* p1=l1,*p2=l2,*p3=listNode;

        while(p1!=NULL|p2!=NULL)
        {
            if(p1!=NULL)
            {
                carry+=p1->val;
                p1=p1->next;
            }
            if(p2!=NULL)
            {
                carry+=p2->val;
                p2=p2->next;
            }
            p3->next=new ListNode(carry%10);
            p3=p3->next;
            carry/=10;
        }
        if(carry==1)
            p3->next=new ListNode(1);
        return listNode->next;
    }
};
```

c

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     public int val;
 *     public ListNode next;
 *     public ListNode(int x) { val = x; }
 * }
 */
public class Solution
{
    public ListNode AddTwoNumbers(ListNode l1, ListNode l2)
    {
        int carry = 0;
        ListNode listNode= new ListNode(0);
        ListNode p1 = l1, p2 = l2, p3 = listNode;

        while (p1 != null || p2 != null)
        {
            if (p1 != null)
            {
                carry += p1.val;
                p1 = p1.next;
            }
            if (p2 != null)
            {
                carry += p2.val;
                p2 = p2.next;
            }
            p3.next = new ListNode(carry % 10);
            p3 = p3.next;
            carry /= 10;
        }
        if (carry == 1)
            p3.next = new ListNode(1);
        return listNode.next;
    }
}
```

Java

```
public class Solution {
     public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
         int carry=0;
         ListNode listNode=new ListNode(0);
         ListNode p1=l1,p2=l2,p3=listNode;
         while(p1!=null||p2!=null){
             if(p1!=null){
                 carry+=p1.val;
                 p1=p1.next;
             }
             if(p2!=null){
                 carry+=p2.val;
                 p2=p2.next;
             }
             p3.next=new ListNode(carry%10);
             p3=p3.next;
             carry/=10;
         }
         if(carry==1)
             p3.next=new ListNode(1);
         return listNode.next;
     }
}
```

C++（来源于网络）

```
class Solution {
public:
    ListNode *addTwoNumbers(ListNode *l1, ListNode *l2) {
        int carry=0;
        ListNode* res=new ListNode(0);
        ListNode* head = res;
        while (l1 && l2){
            res->next=new ListNode((l1->val+l2->val+carry)%10);
            carry = (l1->val+l2->val+carry)/10;
            l1=l1->next;
            l2=l2->next;
            res=res->next;
        }
        while (l1){
            res->next=new ListNode((l1->val+carry)%10);
            carry = (l1->val+carry)/10;
            l1=l1->next;
            res=res->next;
        }
        while (l2){
            res->next=new ListNode((l2->val+carry)%10);
            carry = (l2->val+carry)/10;
            l2=l2->next;
            res=res->next;
        }
        if (carry>0){
            res->next = new ListNode(carry);
        }
        return head->next;
    }
};
```

3

Longest Substring Without Repeating Characters

翻译

给定一个字符串，找出其没有重复字符的最大子序列的长度。　　例如，“abcabcbb”的无重复字符的最大子序列是“abc”，它的长度是 3。　　“bbbbb”的最大子序列是“b”，它的长度是 1。

原文

Given a string, find the length of the longest substring without repeating characters. For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3. For "bbbbb" the longest substring is "b", with the length of 1.

Code 1 (C++)

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int len = s.Length;
        int head = 0, index = 0, maxLen = 0;
        bool[] exist = new bool[256];
        for (int i = 0; i < exist.Length; i++)
            exist[i] = false;
        while (index < len){
            if (exist[s[index]]){
                 maxLen = Math.Max(maxLen, index - head);
                 while (s[head] != s[index]){
                     exist[s[head]] = false;
                     head++;
                 }
                 head++; index++;
             }
             else{
                 exist[s[index]] = true;
                 index++;
             }
         }
         maxLen = Math.Max(maxLen, len - head);
         return maxLen;
    }
};
```

Code 1 C

```
public class Solution
{
    public int LengthOfLongestSubstring(string s)
    {
        int len = s.Length;
        int head = 0, index = 0, maxLen = 0;
        bool[] exist = new bool[256];
        for (int i = 0; i < exist.Length; i++)
            exist[i] = false;
        while (index < len)
        {
            if (exist[s[index]])
            {
                maxLen = Math.Max(maxLen, index - head);
                while (s[head] != s[index])
                {
                    exist[s[head]] = false;
                    head++;
                }
                head++; index++;
            }
            else
            {
                exist[s[index]] = true;
                index++;
            }
        }
        maxLen = Math.Max(maxLen, len - head);
        return maxLen;
    }
}
```

Code 1 Java

```
public class Solution {
    public int lengthOfLongestSubstring(String s) {
        int len=s.length();
        int head=0,index=0,maxLen=0;
        boolean[] exist=new boolean[256];
        for(boolean e : exist)
            e=false;
        while(index<len){
            if(exist[s.charAt(index)]){
                maxLen=Math.max(maxLen, index-head);
                while(s.charAt(head)!=s.charAt(index)){
                    exist[s.charAt(head)]=false;
                    head++;
                }
                head++; index++;
            }
            else{
                exist[s.charAt(index)]=true;
                index++;
            }
        }
        return maxLen= Math.max(maxLen,len-head);
    }
}
```

Code 2 C++

```
static int lengthOfLongestSubstring(string s) { int ascii[256]; for(int i=0; i<256; i++) ascii[i] = -1; int start = 0, ans = 0; int i; for(i=0; i<s.size(); i++){ if( -1 != ascii[ s[i] ] ){ if(ans < i-start) ans = i-start; for(int j=start; j<ascii[ s[i] ]; j++) ascii[j] = -1; if(ascii[s[i]] + 1 > start ) start = ascii[s[i]] +1; } ascii[s[i]] = i; } if(ans < i-start) ans = i-start; return ans; }
```

4

Longest Palindromic Substring（最大回文子字符串）

翻译

有两个给定的排好序的数组 nums1 和 nums2，其大小分别为 m 和 n。

找出这两个已排序数组的中位数。

总运行时间的复杂度应该是 O(log(m+n))。

原文

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)).

第 4 章 C

```
public class Solution {
    public double FindMedianSortedArrays(int[] nums1, int[] nums2) {
        int len1=nums1.Length;
        int len2=nums2.Length;
        bool isEven=(nums1.Length+nums2.Length)%2==0;

        int left=(len1+len2+1)/2;
        int right=(len1+len2+2)/2;

        if (isEven)
        {
            var leftValue = findKth(nums1, 0, len1 - 1, nums2, 0, len2 - 1, left);
            var rightValue = findKth(nums1, 0, len1 - 1, nums2, 0, len2 - 1, right);
            return (leftValue + rightValue) / 2.0;
        }
        else
        {
            return findKth(nums1, 0, len1 - 1, nums2, 0, len2 - 1, right);
        }
    }
    public double findKth(int[] A,int lowA,int highA,int[] B,int lowB,int highB,int k)
    {
        if(lowA>highA)
        {
            return B[lowB+k-1];
        }
        if(lowB>highB)
        {
            return A[lowA+k-1];
        }
        int midA=(lowA+highA)/2;
        int midB=(lowB+highB)/2;
        if (A[midA] <= B[midB])
        {
            return k <= midA - lowA + midB - lowB + 1 ?
                this.findKth(A, lowA, highA, B, lowB, midB - 1, k) :
                this.findKth(A, midA + 1, highA, B, lowB, highB, k - (midA - lowA + 1));
        }
        else
        {
            return k <= midA - lowA + midB - lowB + 1 ?
                this.findKth(A, lowA, midA - 1, B, lowB, highB, k) :
                this.findKth(A, lowA, highA, B, midB + 1, highB, k - (midB - lowB + 1));
        }
```

```
    }
}
```

5

ZigZag Conversion（Z型转换）

翻译

给定一个字符串 S，找出它的最大回文子字符串。
你可以假定 S 的最大长度为 1000，
并且这里存在唯一一个最大回文子字符串。

原文

Given a string S, find the longest palindromic substring in S.

You may assume that the maximum length of S is 1000,

and there exists one unique longest palindromic substring.

暴力搜索，O(n3)

```
public static string LongestPalindrome(string s)
{
    int len = s.Length;
    for (int i = len; i > 0; i--)
        for (int j = 1; j <= len + 1 - i; j++)
            if (isPalindrome(s.Substring(j - 1, i)))
                return s.Substring(j - 1, i);
    return null;
}

public static bool isPalindrome(string s)
{
    for (int i = 0; i < s.Length / 2; i++)
        if (s.Substring(i, 1) != s.Substring(s.Length - 1 - i, 1))
            return false;

    return true;
}
```

动态规划，时间：O(n2)，空间：O(n2)

```
public class Solution
{
    public string LongestPalindrome(string s)
    {
        int sLen = s.Length;
        int lonBeg = 0; int maxLen = 1;
        bool[,] DP = new bool[1000, 1000];
        for (int i = 0; i < sLen; i++)
        {
            DP[i, i] = true;
        }
        for (int i = 0; i < sLen - 1; i++)
        {
            if (s[i] == s[i + 1])
            {
                DP[i, i + 1] = true;
                lonBeg = i;
                maxLen = 2;
            }
        }
        for (int len = 3; len <= sLen; len++)      // 字符串长度从3开始的所有子字符串
        {
            for (int i = 0; i < sLen + 1 - len; i++)
            {
                int j = len - 1 + i;       // j为数组尾部的索引
                if (s[i] == s[j] && DP[i + 1, j - 1])
                {
                    DP[i, j] = true;      // i到j为回文
                    lonBeg = i;           // lonBeg为起始索引，等于i
                    maxLen = len;      // maxLen为字符串长度
                }
            }
        }
        return s.Substring(lonBeg, maxLen);
    }
}
```

然而，继续悲剧……

```
Submission Result: Memory Limit Exceeded
```

```
public class Solution
{
    public string LongestPalindrome(string s)
    {
        int nLen = s.Length;
        if (nLen == 0) return "";
        string lonStr = s.Substring(0, 1);
        for (int i = 0; i < nLen - 1; i++)
        {
            string p1 = ExpandAroundCenter(s, i, i);
            if (p1.Length > lonStr.Length)
                lonStr = p1;
            string p2 = ExpandAroundCenter(s, i, i + 1);
            if (p2.Length > lonStr.Length)
                lonStr = p2;
        }
        return lonStr;
    }
    public static string ExpandAroundCenter(string s, int n, int m)
    {
        int l = n, r = m;
        int nLen = s.Length;
        while (l >= 0 && r <= nLen - 1 && s[l] == s[r])
        {
            l--;
            r++;
        }
        return s.Substring(l + 1, r - l - 1);
    }
}
```

好吧，这种方法来源于网络……据说要 O(n2)的时间，但仅要 O(1)的空间！

哎，继续努力了！

6

Reverse Integer（翻转整数）

翻译

字符串“PAYPALISHIRING”通过一个给定的行数写成如下这种 Z 型模式：

P A H N

A P L S I I G

Y I R

然后一行一行的读取：“PAHNAPLSIIGYIR”

写代码读入一个字符串并通过给定的行数做这个转换：

```
string convert(string text, int nRows);
```

调用 convert("PAYPALISHIRING", 3)，应该返回"PAHNAPLSIIGYIR"。

原文

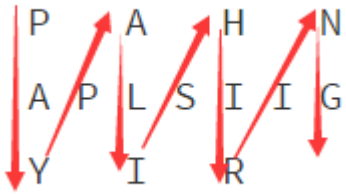
The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

P A H N
A P L S I I G
Y I R

And then read line by line: "PAHNAPLSIIGYIR"
Write the code that will take a string and make this conversion given a number of rows:

string convert(string text, int nRows);
convert("PAYPALISHIRING", 3) should return "PAHNAPLSIIGYIR".

如果还是没明白题目的意思，看下图吧……




```
public class Solution
{
    public string Convert(string s, int numRows)
    {
        if (numRows == 1)
            return s;
        StringBuilder strBuilder = new StringBuilder();
        int lengthOfGroup = 2 * numRows - 2;        // 如上图所示，每组的长度为4
        for (int row = 0; row < numRows; row++)      // 按从第0行到numRows-1行的顺序遍历
        {
            if (row == 0 || row == numRows - 1)         // 此处负责第0行和numRows-1行
            {
                for (int j = row; j < s.Length; j += lengthOfGroup)
                {
                    strBuilder.Append(s[j]);
                }
```

```
            }
            else                    // 此处负责第0行和numRows-1行之间的所有行
            {
                int currentRow = row;           // 在当前行中向右移动（看上图）
                bool flag = true;
                int childLenOfGroup1 = 2 * (numRows - 1 - row);                 //  怎么说呢……中间行的各个索引吧
                int childLenOfGroup2 = lengthOfGroup - childLenOfGroup1;

                while (currentRow < s.Length)
                {
                    strBuilder.Append(s[currentRow]);
                    if (flag)
                        currentRow += childLenOfGroup1;
                    else
                        currentRow += childLenOfGroup2;
                    flag = !flag;
                }
            }
        }
        return strBuilder.ToString();
    }
}
```

C++ 的代码肯定是有的：

```
class Solution {
public:
    string convert(string s, int numRows) {
        if(numRows==1)
            return s;
        string str="";
        int lengthOfGroup=2*numRows-2;
        for(int row=0;row<numRows;row++){
            if(row==0||row==numRows-1){
                for(int currentRow=row;currentRow<s.length();currentRow+=lengthOfGroup){
                    str+=s[currentRow];
                }
            }
            else{
                int currentRow=row;
                bool flag=true;
                int childLenOfGroup1=2*(numRows-1-row);
                int childLenOfGroup2=lengthOfGroup-childLenOfGroup1;
                while(currentRow<s.length()){
                    str+=s[currentRow];
                    if(flag)
```

```
                    currentRow+=childLenOfGroup1;
                else
                    currentRow+=childLenOfGroup2;
                flag=!flag;
            }
        }
    }
    return str;
  }
};
```

至于 Java 嘛，当然也有……不过每次我都是直接把 C# 的代码拷贝过去然后改改就好了。

```
public class Solution {
    public String convert(String s, int numRows) {
          if (numRows == 1)
              return s;
          StringBuilder strBuilder = new StringBuilder();
          int lengthOfGroup = 2 * numRows - 2;
          for(int row=0; row < numRows; row++){
              if (row == 0 || row == numRows - 1){
                  for(int currentRow = row; currentRow < s.length(); currentRow += lengthOfGroup){
                      strBuilder.append(s.charAt(currentRow));
                  }
              }
              else{
                  int currentRow = row;
                  boolean flag = true;
                  int childLenOfGroup1 = 2 * (numRows - 1 - row);
                  int childLenOfGroup2 = lengthOfGroup - childLenOfGroup1;
                  while (currentRow <s.length()){
                      strBuilder.append(s.charAt(currentRow));
                      if (flag)
                          currentRow += childLenOfGroup1;
                      else
                          currentRow += childLenOfGroup2;
                      flag = !flag;
                  }
              }
          }
          return strBuilder.toString();
      }
}
```

7

String to Integer (atoi)（转换到整型）

翻译

翻转一个整型数

例 1：x = 123, 返回 321

例 2：x = -123, 返回 -321

原文

Reverse digits of an integer.

Example1: x = 123, return 321

Example2: x = -123, return -321

Have you thought about this? （来自 LeetCode 官网）

Here are some good questions to ask before coding.

Bonus points for you if you have already thought through this!

If the integer's last digit is 0, what should the output be?

ie, cases such as 10, 100.

Did you notice that the reversed integer might overflow?

Assume the input is a 32-bit integer, then the reverse of 1000000003 overflows.

How should you handle such cases?

For the purpose of this problem,

assume that your function returns 0 when the reversed integer overflows.

Update (2014-11-10):

Test cases had been added to test the overflow behavior.

C

```
public class Solution
{
    public int Reverse(int x)
    {
        try
        {
            string str = Math.Abs(x).ToString();
            string newStr = (x < 0) ? "-" : "";
            for (int i = str.Length - 1; i >= 0; i--)
            {
                newStr += str[i];
            }
            return int.Parse(newStr);
        }
        catch (Exception e)
        {
            return 0;
        }
    }
}
```

C++（来源于网络）

```
class Solution {
public:
    int reverse(int x) {
        int res = 0;
        while(x!=0){
            if(res>INT_MAX/10||res<INT_MIN/10){
                return 0;
            }
            res = res*10 + x%10;
            x = x/10;
        }
        return res;
    }
};
```

8

String to Integer (atoi)（转换到整型）

翻译

实现“atoi”将字符串转换成整型数。

提示：仔细考虑所有可能的输入。如你想要挑战，请不要参阅下面并问问自己都有哪些可能的输入请看。

说明：模糊的指定（没有给定的输入规格）就是为了这个问题。你负责收集所有可能的输入。

atoi 的要求：

函数首先放弃尽可能多的空字符直到找到一个非空白字符。然后从这个字符开始，带上可选的初始加 / 减字符，其后还可能跟着越多越好的数字，并将它们解释成一个数值。

这个字符串可能在这些数字之后包含一些附加的字符，它们可以可以被忽略，并对函数的行为没有影响。

如果字符串 str 中第一个非空格的序列不是一个有效的整型数，或者因为 str 为空或仅有空格字符而不存在这样一个序列，那么不执行任何转换。

如果可以不执行任何有效的转换，则返回零值。如果正确的值在值域范围之外，则返回 INT_MAX（2147483647）或 INT_MIN（-2147483647）。

原文

Implement atoi to convert a string to an integer.

Hint: Carefully consider all possible input cases. If you want a challenge, please do not see below and ask yourself what are the possible input cases.

Notes: It is intended for this problem to be specified vaguely (ie, no given input specs). You are responsible to gather all the input requirements up front.

Requirements for atoi:
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. If the correct value is out of the range of representable values, INT_MAX (2147483647) or INT_MIN (-2147483648) is returned.

英语渣渣实在没看懂题目，不知道有哪些条件，于是就慢慢写代码，根据报错继续改……结果代码改到了 80 行……还是不能完成所有条件，还是从网上荡了一个下来，来日再战！

（好吧，在写博客，也就上面的翻译过程中，我发现题目懂了……）

```
public class Solution
{
    public int MyAtoi(string str)
    {
        if (string.IsNullOrEmpty(str))
        {
            return 0;
        }
        var result = 0;
        var i = 0;
        // clean all the whitespaces in the beginning
```

```
        while (i < str.Length && str[i] == ' ')
        {
            i++;
        }
        // check positive or negative sign
        var sign = 1;
        switch (str[i])
        {
            case '-':
                sign = -1;
                i++;
                break;
            case '+':
                sign = 1;
                i++;
                break;
        }
        // check the rest of numbers
        while (i < str.Length && str[i] >= '0' && str[i] <= '9')
        {
            // check overflow
            try
            {
                checked
                {
                    result = result * 10 + (str[i++] - '0');
                }
            }
            catch (OverflowException)
            {
                return sign == 1 ? int.MaxValue : int.MinValue;
            }
        }
        return sign * result;
    }
}
```

```
1045 / 1045 test cases passed.
Status: Accepted
Runtime: 168 ms
Your runtime beats 16.85% of csharp submissions.
```

9

Palindrome Number（回文数）

翻译

确定一个整数是否是回文数。不能使用额外的空间。

一些提示：

负数能不能是回文数呢？（比如，-1）

如果你想将整数转换成字符串，但要注意限制使用额外的空间。

你也可以考虑翻转一个整数。
然而，如果你已经解决了问题“翻转整数（译者注：LeetCode 第七题），
那么你应该知道翻转的整数可能会造成溢出。
你将如何处理这种情况？

这是一个解决该问题更通用的方法。

原文

Determine whether an integer is a palindrome. Do this without extra space.

Some hints:

Could negative integers be palindromes? (ie, -1)

If you are thinking of converting the integer to string, note the restriction of using extra space.

You could also try reversing an integer.
However, if you have solved the problem "Reverse Integer",
you know that the reversed integer might overflow.
How would you handle such case?

There is a more generic way of solving this problem.

一开始的想法，大不了不新建一个字符串，直接在循环里用就好了，结果居然也可以 accept。

```
public class Solution
{
    public bool IsPalindrome(int x)
    {
        for (int i = 0; i < x.ToString().Length / 2; i++)
        {
            if ((x.ToString())[i] != x.ToString()[x.ToString().Length - i - 1])
            {
                return false;
            }
        }
        return true;
    }
}
```

可是叻：

```
11506 / 11506 test cases passed.
Status: Accepted
Runtime: 244 ms
Your runtime beats 0.97% of csharp submissions.
```

然后修改了一下：

```
public class Solution
{
    public bool IsPalindrome(int x)
    {
        if (x < 0)
            return false;
        // 判断x的长度，比如x=232，div就等于100
        int div = 1;
        while (x / div >= 10)
            div *= 10;
        while (x != 0)
        {
            // 左边开始计数
            int left = x / div;
            // 右边开始计数
            int right = x % 10;
            if (left != right)
                return false;
            x = (x % div) / 10;
            div /= 100;
        }
        return true;
    }
}
```

性能还是不尽如人意呀，不过同样的代码放在 Java 上，貌似 C# 要快一些。

```
11506 / 11506 test cases passed.
Status: Accepted
Runtime: 196 ms
Your runtime beats 21.36% of csharp submissions.
```

下面这份代码是网上找到的，性能和上面的一模一样：

```
public class Solution
{
    public bool IsPalindrome(int x)
    {
        if (x < 0)
            return false;
        else if (x == 0)
            return true;
        else
        {
            int tmp = x;
            int y = 0;
```

```
            while (x != 0)
            {
                y = y * 10 + x % 10;
                x = x / 10;
            }
            if (y == tmp)
                return true;
            else
                return false;
        }
    }
}
```

10

Regular Expression Matching （正则表达式匹配）

翻译

实现支持“.”和“*”的正则表达式匹配。

“.” 匹配支持单个字符

“*” 匹配零个或多个前面的元素

匹配应该覆盖到整个输入的字符串（而不是局部的）。

该函数的原型应该是：

```
bool isMatch(const char * s, const char * p)
```

示例：

isMatch("aa","a") → false

isMatch("aa","aa") → true

isMatch("aaa","aa") → false

isMatch("aa", "a") → true

isMatch("aa", ".") → true

isMatch("ab", ".") → true

isMatch("aab", "ca*b") → true

原文

Implement regular expression matching with support for '.' and '*'.

'.' Matches any single character.

'*' Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

The function prototype should be:

bool isMatch(const char *s, const char *p)

Some examples:

isMatch("aa","a") → false

isMatch("aa","aa") → true

isMatch("aaa","aa") → false

isMatch("aa", "a") → true

isMatch("aa", ".") → true

isMatch("ab", ".") → true

isMatch("aab", "ca*b") → true

C#，递归

```
public class Solution
{
    public bool IsMatch(string s, string p)
    {
        if (p.Length == 0)
            return s.Length == 0;

        if (p.Length == 1)
            return (s.Length == 1) && (p[0] == s[0] || p[0] == '.');

        if (p[1] != '*')
        {
            if (s.Length == 0)
                return false;
            else
                return (s[0] == p[0] || p[0] == '.')
                       && IsMatch(s.Substring(1), p.Substring(1));
        }
        else
        {
            while (s.Length > 0 && (p[0] == s[0] || p[0] == '.'))
            {
                if (IsMatch(s, p.Substring(2)))
                    return true;
                s = s.Substring(1);
            }
            return IsMatch(s, p.Substring(2));
        }
    }
}
```

11

Container With Most Water（最大水容器）

翻译

给定 n 个非负整数 a1,a2,...,an，其中每个代表一个点坐标（i,ai）。

n 个垂直线段例如线段的两个端点在（i,ai）和（i,0）。

找到两个线段，与 x 轴形成一个容器，使其包含最多的水。

备注：你不必倾倒容器。

原文

Given n non-negative integers a1, a2, ..., an,

where each represents a point at coordinate (i, ai).

n vertical lines are drawn

such that the two endpoints of line i is at (i, ai) 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.

题目的意思是，数组中的每个数对应一条线段的长度，索引对应 x 坐标，两个索引可以组成一个底部的宽，高度就是前面所说的线段的长度，而既然是要盛水，高度就是两个线段中较短的一个。

那么该怎么去解题呢？

我水平不行，英文也不行，所以每次一开始都是用最简单的方法，旨在试试有没有理解题目的意思，即便超出时间 / 空间限制也没事。

```
public int MaxAera(int[] height)
{
    int area = 0;
    for (int i = 0; i < height.Length; i++)
    {
        for (int j = i + 1; j < height.Length; j++)
        {
            if (height[i] < height[j])
                area = Math.Max(area, countArea(height, i, j));
        }
    }
    return area;
}

public int countArea(int[] height, int x, int y)
{
    int h = height[x] > height[y] ? height[x] : height[y];
    int info = h * (y - x);
    return info;
}
```

很明显这样是不行的……

那有那些部分可以简化呢？

前面的方法是从数组左侧开始逐个向右遍历所有情况，但明显可以从两侧向中间进发，通过对应的 max 函数来保留最大的面积。

当从左边进入到图中线段 1 位置，右边进入到线段 5 的时候。你不会想着右边继续进入线段 6 和 7，因为你就是从那边过来的。

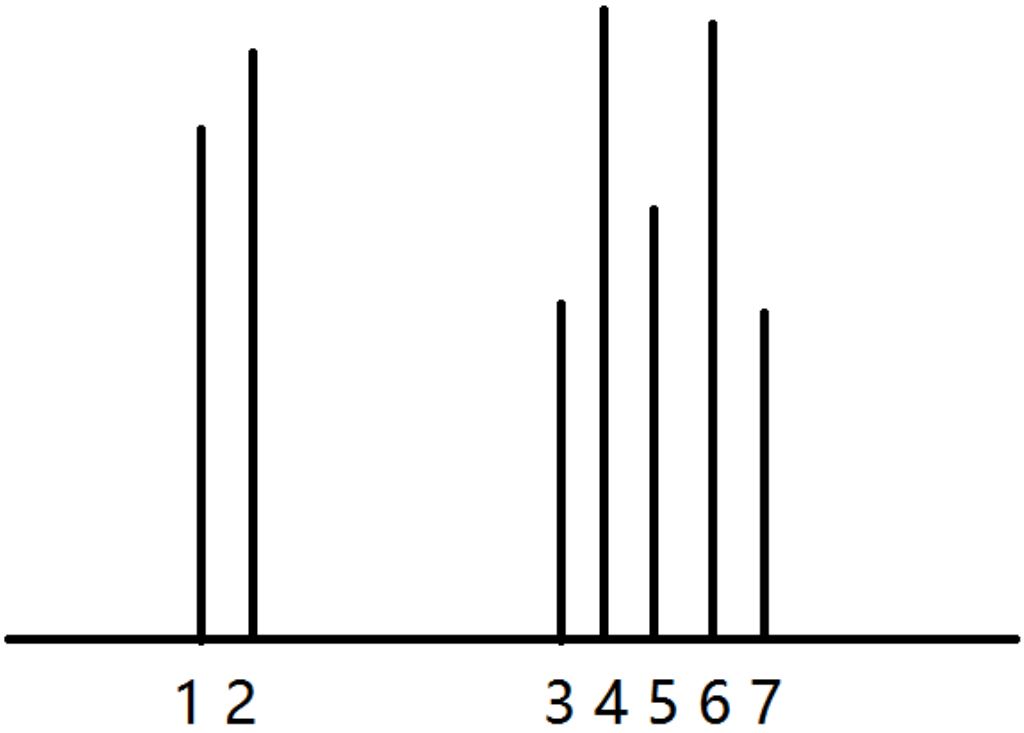
那么是该左边的往右走，还是右边的往左走呢？

如果是右边的往左走，虽然线段 1 变成了线段 2，但是线段1到线段5的距离比线段 2 大，因此面积也大。所以走了之后面积反而小了。

如果是右边的往左走，亲自行脑补：线段 3 和线段 4 是在同一位置，如果是到了线段 4，那么容器的高度将从原本的线段 5 的长度变成线段 1 的长度，（虽然由于距离的变小，总面积仍可能变小，但请继续往下看），而如果到了线段 3，虽然高度变小了，宽度变小了，但，那又何妨呢？因为你的 maxArea 还是在那里的，每次的计算后，当且仅当高度超过原本的高度之后才会覆盖原来的值。

```
maxArea = Max(maxArea,newArea);
```

也就是说，高度如果没有超过，就没有什么影响。



至于你说它会不会因为自增和自减而发生越界，如果

```
int[] height = {10, 1, 2, 3, 4, 5, 6, 7, 11};
```

假设这里的 10 和 11 对应线段 1 和线段 6，请自行脑补：去掉线段 7，既然线段 1 短于线段 6，那么发生的是 left++，而不是 left--。所以，并不会越界的。反之，亦然。

```
public class Solution
{
    public int MaxArea(int[] height)
    {
        int left = 0, right = height.Length - 1;
        int maxArea = 0;
        while (left < right && left >= 0 && right <= height.Length - 1)
        {
            maxArea = Math.Max(maxArea, Math.Min(height[left], height[right]) * (right - left));
            if (height[left] > height[right])
            {
                right--;
            }
            else
            {
```

```
                left++;
            }
        }
        return maxArea;
    }
}
```

明天继续，加油！

12

Integer to Roman（整型数到罗马数）

翻译

给定一个整型数值，将其转换到罗马数字。

输入被保证在 1 到 3999 之间。

原文

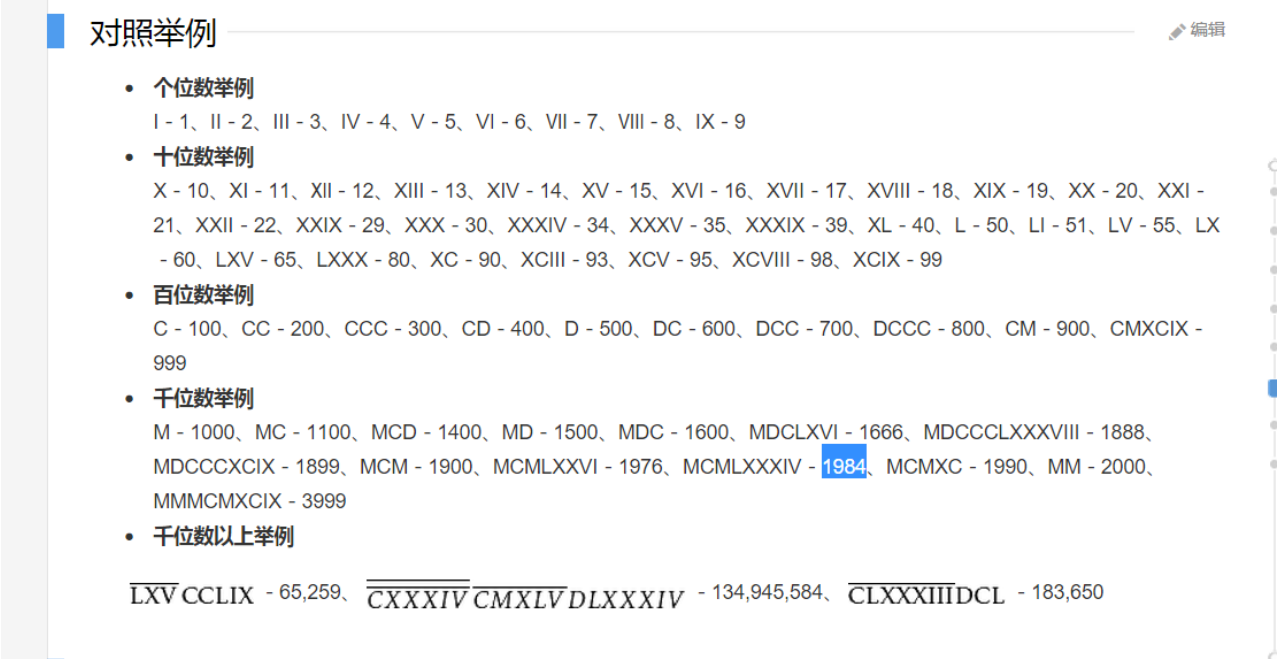
Given an integer, convert it to a roman numeral.

Input is guaranteed to be within the range from 1 to 3999.

我不会告诉你一开始我是用的无数个变量和 if……

后来实在受不了这么多变量就将其写成了枚举，那么接下来就迎刃而解了。

为了让大家理解罗马数是怎么计数的，这里我截了一张图，具体的大家可以自行用微软 Bing 搜索。



对照举例 编辑

- **个位数举例**
  I - 1、II - 2、III - 3、IV - 4、V - 5、VI - 6、VII - 7、VIII - 8、IX - 9
- **十位数举例**
  X - 10、XI - 11、XII - 12、XIII - 13、XIV - 14、XV - 15、XVI - 16、XVII - 17、XVIII - 18、XIX - 19、XX - 20、XXI - 21、XXII - 22、XXIX - 29、XXX - 30、XXXIV - 34、XXXV - 35、XXXIX - 39、XL - 40、L - 50、LI - 51、LV - 55、LX - 60、LXV - 65、LXXX - 80、XC - 90、XCIII - 93、XCV - 95、XCVIII - 98、XCIX - 99
- **百位数举例**
  C - 100、CC - 200、CCC - 300、CD - 400、D - 500、DC - 600、DCC - 700、DCCC - 800、CM - 900、CMXCIX - 999
- **千位数举例**
  M - 1000、MC - 1100、MCD - 1400、MD - 1500、MDC - 1600、MDCLXVI - 1666、MDCCCLXXXVIII - 1888、MDCCCXCIX - 1899、MCM - 1900、MCMLXXVI - 1976、MCMLXXXIV - 1984、MCMXC - 1990、MM - 2000、MMMCMXCIX - 3999
- **千位数以上举例**
  $\overline{\text{LXV}}$CCLIX - 65,259、$\overline{\text{CXXXIV}}\,\overline{\text{CMXLV}}\text{DLXXXIV}$ - 134,945,584、$\overline{\text{CLXXXIII}}$DCL - 183,650

那么代码我就先贴出来了：

```
public class Solution
{
    public string IntToRoman(int num)
    {
        string result = "";
        Type R = typeof(Roman);

        foreach (var r in Enum.GetNames(R).Reverse())
        {
            while (num >= int.Parse(Enum.Format(R, Enum.Parse(R, r), "d")))
            {
                result += r.ToString();
```

```
                num -= int.Parse(Enum.Format(R, Enum.Parse(R, r), "d"));
            }
        }
        return result;
    }
}
public enum Roman
{
    M = 1000,
    CM = 900,
    D = 500,
    CD = 400,
    C = 100,
    XC = 90,
    L = 50,
    XL = 40,
    X = 10,
    IX = 9,
    V = 5,
    IV = 4,
    I = 1
};
```

今天晚些时候我会将 C# 枚举的一些用法贴到博客上，不了解的同学敬请关注。

除了枚举的用法外，我认为这道题中需要你去认真了解这些罗马数的规则，也就是说记得将 9 和 4 这种数也添加到枚举中哦。

那么在 IntToRoman 中都中都在做些什么呢？

- 搭配 Type 和 typeof 新建出来 R
- 用 foreach 遍历枚举中的所有元素
- 切记要加上 Reverse()，至于为什么，大家试试不加就知道了
- 如果 num 比枚举中的数字大，则将其对应的字符串（比如说"M”）添加到 result 中
- 最后在 num 中减掉刚才已经用过的数字，数字也通过枚举来获取
- 最后返回 result

好了，到此为止，准备迎接下一题。

下一道题还是关于罗马数的，不过是从罗马数转换成整型数，欢迎大家访问：传送门：LeetCode 13 Roman to Integer（罗马数到整型数）。

13

Roman to Integer（罗马数到整型数）

翻译

给定一个罗马数字，将其转换到整型数值。

输入被保证在 1 到 3999 之间。

原文

Given a roman numeral, convert it to an integer.

Input is guaranteed to be within the range from 1 to 3999.

一开始就没有构思好，虽然按上一题的套路可以走下去，但结果就是像我下面这样……代码凌乱……

```
public class Solution
{
    public int RomanToInt(string s)
    {
        int result = 0;
        Type R = typeof(Roman);
        string first, second;
        if (s.Length > 1)
        {
            first = s.Substring(0, 1); second = s.Substring(0, 2);
        }
        else
        {
            first = s.Substring(0, 1); second = "";
        }
        foreach (var r in Enum.GetNames(R).Reverse())
        {
            while ((r.Length == 1 && first == r) || (r.Length == 2 && second == r))
            {
                result += int.Parse(Enum.Format(R, Enum.Parse(R, r), "d"));
                int lenR = r.Length, lenS = s.Length;
                if (lenS - lenR < 1)
                    s = "";
                else
                    s = s.Substring(lenR, lenS - lenR);

                if (s.Length > 1)
                {
                    first = s.Substring(0, 1); second = s.Substring(0, 2);
                }
                else if (s.Length == 1)
                {
                    first = s.Substring(0, 1); second = "";
                }
                else
```

```
                {
                    first = ""; second = "";
                }
            }
        }
        return result;
    }

}

public enum Roman
{
    M = 1000,
    CM = 900,
    D = 500,
    CD = 400,
    C = 100,
    XC = 90,
    L = 50,
    XL = 40,
    X = 10,
    IX = 9,
    V = 5,
    IV = 4,
    I = 1
};
```

虽然吧，可以运行……但是效率也太低了，简直不忍直视……于是还是像其他大神学习学习……

下面这段代码深深的打动了我……数组作为数组的索引……我最不常用的用法了……

```
class Solution {
public:
    int romanToInt(string s) {
        unordered_map<char, int> map = {{'I', 1}, {'V', 5}, {'X', 10}, {'L', 50}, {'C', 100}, {'D', 500}, {'M', 1000}};
        int ret = 0;
        for (int idx = 0; idx < s.size(); ++idx) {
            if ((idx < s.size()-1) && (map[s[idx]] < map[s[idx+1]])) {
                ret -= map[s[idx]];
            }
            else {
                ret += map[s[idx]];
            }
        }
        return ret;
```

```
    }
};
```

这个算法呢，充分利用了罗马数字两个数字前后顺序的关系，也就是说如果是 I 在 V 前面，也就是 IV，它代表 4，反之代表 6。再搭配 C++ 的 unordered_map，可以巧妙的通过数组来进行判断而达到对 ret 或加或减的目的。

我需要好好体会了……

14

Longest Common Prefix（最长公共前缀）

翻译

写一个函数（或方法）来寻找一个字符串数组中的最长公共前缀。

原文

Write a function to find the longest common prefix string amongst an array of strings.

释义

"abcdefg"　　"abcdefghijk"　　"abcdfghijk"　　"abcef"

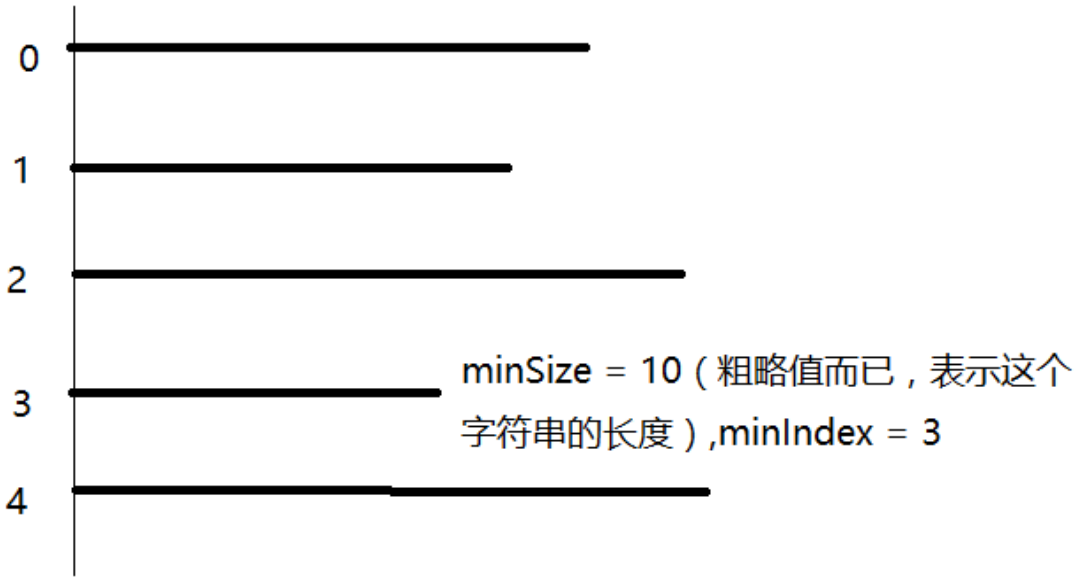
上面的字符串数组的最长公共前缀就是"abc"。

思考

如下图所示，第一步就是要找出该字符串数组中的最短字符串的长度及其序列。

第二步，用 for 循环从第一个字符串到最后一个字符串依次做比较，具体步骤如下：

- 外层 for 循环中用 i 表示字符串长度，从 minSize 一直可以递减到 0
- 初始 result 即为最短字符串（通过 minIndex 确定）的前 i 个字符
- 内层 for 循环中用j表示字符串数组中的索引，依次递增。j 等于 minIndex 时不做操作（因为为同一个字符串不必比较）
- 否则通过临时字符串 temp 来获取索引为 j 的字符的前 i 个字符
- 需要所有的 temp 都与 result 相等
- 如果 j 和 len 相等了，说明已经遍历完所有的字符串
- 每次判断的字符串长度缩减之后都更新 result



代码

```
public class Solution {
    public string LongestCommonPrefix(string[] strs) {
        int len = strs.Length;

        if(len == 0)
            return "";

        string result = "";
        int minSize = 100000;
        int minIndex = 0;

        if(len == 1){
            result = strs[0];
            return result;
        }

        for(int i = 0; i < len; i++){
            int size = strs[i].Length;
            if(size < minSize){
                minSize = size;
                minIndex = i;
            }
        }

        for(int i = minSize; i >= 0; i--){
            result = strs[minIndex].Substring(0,i);

            int j = 0;
            for(; j < len; j++){
                if(j == minIndex)
                    continue;
                string temp = strs[j].Substring(0,i);
                if(result != temp)
                    break;
            }
            if(j == len)
                return result;
        }
        return result;
    }
}
```

15

3 Sum（3 个数的和）

翻译

给定一个有 n 个整数的数组 S，是否存在三个元素 a,b,c 使得 a+b+c=0？找出该数组中所有不重复的 3 个数，它们的和为 0。

备注：
这三个元素必须是从小到大进行排序。
结果中不能有重复的 3 个数。

例如，给定数组 S={-1 0 1 2 -1 4}，一个结果集为：
(-1, 0, 1)
(-1, -1, 2)

原文

Given an array S of n integers,

are there elements a, b, c in S such that a + b + c = 0?

Find all unique triplets in the array which gives the sum of zero.

Note:

Elements in a triplet (a,b,c) must be in non-descending order.

(ie, a ≤ b ≤ c)

The solution set must not contain duplicate triplets.

For example, given array S = {-1 0 1 2 -1 -4},

A solution set is:

(-1, 0, 1)

(-1, -1, 2)

经典方法，可惜我并没有想到这样写……

```
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        vector<vector<int>> result;

        int len = nums.size();
        for (int current = 0; current < len - 2&&nums[current]<=0;current++)
        {
            int front = current + 1, back = len - 1;
            while (front < back)
            {
                if (nums[current] + nums[front] + nums[back] < 0)
                    front++;
                else if (nums[current] + nums[front] + nums[back] > 0)
                    back--;
                else
                {
                    vector<int> v(3);
                        v.push_back(nums[current]);
                        v.push_back(nums[front]);
                        v.push_back(nums[back]);
```

```
                    result.push_back(v);
                    v.clear();
                do {
                    front++;
                } while (front < back&&nums[front - 1] == nums[front]);
                do {
                    back--;
                } while (front < back&&nums[back + 1] == nums[back]);
            }
        }
        while (current < len - 2 && nums[current + 1] == nums[current])
            current++;
    }
    return result;
  }
};
```

继续努力……

和本道题关联密切的题目推荐：

传送门：LeetCode 16 3Sum Closest（最接近的3个数的和）

传送门：LeetCode 18 4Sum（4个数的和）

16

3 Sum Closest（最接近的 3 个数的和）

翻译

给定一个有 n 个整数的数组 S，找出 S 中 3 个数，使其和等于一个给定的数，target。

返回这 3 个数的和，你可以假定每个输入都有且只有一个结果。

例如，给定 S = {-1 2 1 -4}，和 target = 1。

那么最接近 target 的和是 2。(-1 + 2 + 1 = 2)。

原文

Given an array S of n integers,

find three integers in S such that the sum is closest to a given number, target.

Return the sum of the three integers.

You may assume that each input would have exactly one solution.

For example, given array S = {-1 2 1 -4}, and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

思考

也许我已经开始体会到上一题中别人写的方法的思想了。

在这个题目中，我们要做以下几件事：

- 用 sort 对输入的数组进行排序
- 求出长度 len，current 之所以要小于 len-2，是因为后面需要留两个位置给 front 和 back
- 始终保证 front 小于 back
- 计算索引为 current、front、back 的数的和，分别有比 target 更小、更大、相等三种情况
- 更小：如果距离小于 close，那么close 便等于 target-sum，而结果就是 sum。更大的情况同理
- 如果相等，那么要记得将 0 赋值给 close，result 就直接等于 target 了
- 随后为了避免计算重复的数字，用三个 do/while 循环递增或递减它们

代码

```
class Solution
{
public:
    int threeSumClosest(vector<int>& nums, int target) {
        sort(nums.begin(), nums.end());
        int len = nums.size();
        int result = INT_MAX, close = INT_MAX;
        for (int current = 0; current < len - 2; current++) {
            int front = current + 1, back = len - 1;
            while (front < back) {
                int sum = nums[current] + nums[front] + nums[back];
                if (sum < target) {
                    if (target - sum < close) {
                        close = target - sum;
                        result = sum;
                    }
                    front++;
                }
                else if (sum > target) {
                    if (sum - target < close) {
                        close = sum - target;
                        result = sum;
                    }
                    back--;
                }
                else {
                    close = 0;
                    result = target;
                    do {
                        front++;
                    } while (front < back&&nums[front - 1] == nums[front]);
                    do {
                        back--;
                    } while (front < back&&nums[back + 1] == nums[back]);
                }
            }
            while (current < len - 2 && nums[current + 1] == nums[current]) {
                current++;
            }
        }
        return result;
```

```
    }
};
```

和本道题关联密切的题目推荐：

传送门：LeetCode 15 3 Sum（3 个数的和）

传送门：LeetCode 18 4 Sum（4 个数的和）

17

Letter Combinations of a Phone Number（电话号码的字母组合）

翻译

给定一个数字字符串，返回所有这些数字可以表示的字母组合。

一个数字到字母的映射（就像电话按钮）如下图所示。

输入：数字字符串“23”
输出：["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"]

备注:尽管以上答案是无序的，如果你想的话你的答案可以是有序的。

原图

原文

Given a digit string, 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.

Input:Digit string "23"

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

Note:

Although the above answer is in lexicographical order, your answer could be in any order you want.

代码

看样子我还是用 C# 顺手点，一气呵成……主要是用递归，因为不知道 digits 的长度到底是多少，不可能写无数个 foreach 去判断。

```
public class Solution
{
    IList<string> list = new List<string>();
    public Solution()
    {
        list.Insert(0, "abc");
        list.Insert(1, "def");
        list.Insert(2, "ghi");
        list.Insert(3, "jkl");
        list.Insert(4, "mno");
        list.Insert(5, "pqrs");
        list.Insert(6, "tuv");
        list.Insert(7, "wxyz");
    }
    public IList<string> LetterCombinations(string digits)
    {
        IList<string> result = new List<string>();
        if (digits.Length == 0)
            return result;
        if (digits.Length == 1)
        {
            foreach (var a in list.ElementAt(int.Parse(digits[0].ToString()) - 2))
            {
                result.Insert(0, a.ToString());
            }
        }
        int count = 0;
        IList<string> temp = LetterCombinations(digits.Substring(1, digits.Length - 1));
        foreach (var a in list.ElementAt(int.Parse(digits[0].ToString()) - 2))
        {
            foreach (var rest in temp)
            {
                result.Insert(count++, a.ToString() + rest);
            }
        }
        return result;
    }
}
```

以下是复制来的一段 C++ 代码，坦白地说，我写不出来这样的 C++ 代码……

```
class Solution {
public:
    vector<string> letterCombinations(string digits) {
        vector<string> ans;
        if(digits.size() == 0)
            return ans;

        int depth = digits.size();
        string tmp(depth, 0);
        dfs(tmp, 0, depth, ans, digits);
        return ans;
    }

    void dfs(string &tmp, int curdep, int depth, vector<string> &ans, string &digits){
        if(curdep >= depth){
            ans.push_back(tmp);
            return ;
        }
        for(int i = 0; i < dic[digits[curdep] - '0'].size(); ++ i){
            tmp[curdep] = dic[digits[curdep] - '0'][i];
            dfs(tmp, curdep + 1, depth, ans, digits);
        }
        return ;
    }
private:
string dic[10] = {{""},{""},{"abc"},{"def"},{"ghi"},{"jkl"},{"mno"},{"pqrs"},{"tuv"},{"wxyz"}};
};
```

18

4 Sum（4 个数的和）

翻译

给定一个有 n 个数字的数组 S，在 S 中是否存在元素 a，b，c 和 d 的和恰好满足 a + b + c + d = target。

找出数组中所有的不想等的这四个元素，其和等于 target。

备注：

在（a，b，c，d）中的元素必须从小到大排列。（a ≤ b ≤ c ≤ d）
其结果必须不能够重复。

例如，给定 S = {1 0 -1 0 -2 2}，target = 0。
一个结果集为：
(-1, 0, 0, 1)
(-2, -1, 1, 2)
(-2, 0, 0, 2)

原文

Given an array S of n integers,

are there elements a, b, c, and d in S such that a + b + c + d = target?

Find all unique quadruplets in the array which gives the sum of target.

Note:

Elements in a quadruplet (a,b,c,d) must be in non-descending order. (ie, a ⩽ b ⩽ c ⩽ d)

The solution set must not contain duplicate quadruplets.

For example, given array S = {1 0 -1 0 -2 2}, and target = 0.

A solution set is:

(-1, 0, 0, 1)

(-2, -1, 1, 2)

(-2, 0, 0, 2)

代码

具体的方法和前面两道题一样，我就不再赘述了。

```
class Solution {
public:
    vector<vector<int>> fourSum(vector<int>& nums,int target) {
        sort(nums.begin(), nums.end());
        vector<vector<int>> result;

        int len = nums.size();
        for (int current = 0; current < len - 3;current++)
        {
            for(int second = current+1;second<len-2;second++)
            {
                int front = second + 1, back = len - 1;
                while (front < back)
                {
                    if (nums[current]+nums[second] + nums[front] + nums[back]  < target)
                        front++;
                    else if (nums[current] +nums[second]+ nums[front] + nums[back] > target)
                        back--;
                    else
                    {
                        vector<int> v(4);
                        v[0]=nums[current];
                        v[1]=nums[second];
                        v[2]=nums[front];
                        v[3]=nums[back];
                        result.push_back(v);
                        do {
                            front++;
                        } while (front < back&&nums[front - 1] == nums[front]);
                        do {
                            back--;
                        } while (front < back&&nums[back + 1] == nums[back]);
                    }
                }
                while(second < len-2&&nums[second+1]==nums[second])
                    second++;
            }
            while (current < len - 3 && nums[current + 1] == nums[current])
                current++;
```

```
        }
        return result;
    }
};
```

和本道题关联密切的题目推荐：

传送门：LeetCode 15 3 Sum（3 个数的和）

传送门：LeetCode 16 3 Sum Closest（最接近的 3 个数的和）

19

Remove Nth Node From End of List（从列表尾部删除第 N 个结点）

翻译

给定一个链表，移除从尾部起的第 n 个结点，并且返回它的头结点。

例如，给定链表：1->2->3->4->5，n = 2。

在移除尾部起第二个结点后，链表将变成：1->2->3->5。

备注：

给定的 n 是有效的，代码尽量一次通过。

原文

Given a linked list, remove the nth node from the end of list and return its head.

For 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.

Try to do this in one pass.

代码

```
class Solution{
public:
    ListNode* removeNthFromEnd(ListNode* head, int n){
        ListNode newHead(0);
        newHead.next = head;
        count = n;
        return solution1(&newHead);
    }
private:
    int count;
    ListNode* solution1(ListNode* newHead){
        subSol1(newHead);
        return newHead->next;
    }
    bool subSol1(ListNode* node){
        if(!node) return false;
        if(subSol1(node->next)) return true;
        if(count--) return false;
        ListNode *tmp = node->next;
        node->next = tmp->next;
        delete tmp;
        return true;
    }
};
```

20

Valid Parentheses（有效的括号）

翻译

给定一个只包含'('，')'，'{'，'}'，'[' 和']'的字符串，判断这个输入的字符串是否是有效的。

括号必须在正确的形式下闭合，"()" 和"()[]{}" 是有效的，但是 "(]" 和"([)]" 则不是。

原文

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

The brackets must close in the correct order,
"()" and "()[]{}" are all valid but "(]" and "([)]" are not.

代码

```
class Solution {
public:
    bool isValid(string s) {
        if(s.size() % 2 != 0) return 0;
        stack<char> brackets;
        int i = 0;
        while(i < s.size()) {
            if(brackets.empty()) {
                brackets.push(s[i]);
            } else {
                if((brackets.top() == '(' && s[i] == ')') ||
                (brackets.top() == '[' && s[i] == ']') ||
                (brackets.top() == '{' && s[i] == '}')) {
                    brackets.pop();
                } else {
                    brackets.push(s[i]);
                }
            }
            i ++;
        }
        return brackets.size() == 0;
    }
};
```

21

Merge Two Sorted Lists（合并两个已排序的数组）

翻译

合并两个排好序的链表，并返回这个新链表。
新链表应该由这两个链表的头部拼接而成。

原文

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.

代码

```
/**
 * 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) {
        if(l2 == NULL) {
            return l1;
        }
        if(l1 == NULL) {
            return l2;
        }
        if(l1->val > l2->val) {
            ListNode* temp = l2;
            temp->next = mergeTwoLists(l1, l2->next);
            return temp;
        } else {
            ListNode* temp = l1;
            temp->next = mergeTwoLists(l1->next, l2);
            return temp;
        }
    }
};
```

22

Generate Parentheses（生成括号）

翻译

给定一个括号序列，写一个函数用于生成正确形式的括号组合。

例如，给定 n = 3，一个解决方案集是：

"((()))", "(()())", "(())()", "()(())", "()()()"

原文

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

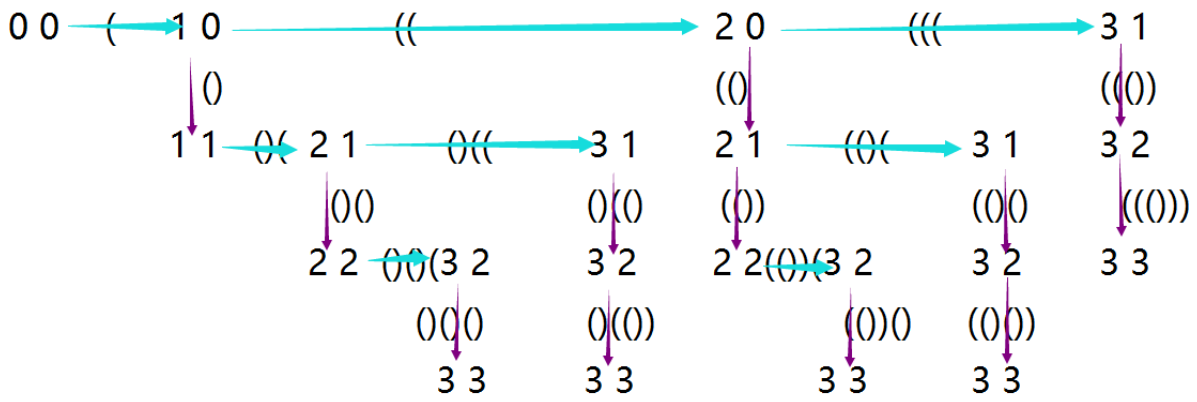
For example, given n = 3, a solution set is:

"((()))", "(()())", "(())()", "()(())", "()()()"

代码

```
class Solution {
public:
    vector<string> result;
    vector<string> generateParenthesis(int n) {
        generate(0, 0, "", n);
        return result;
    }
    void generate(int left, int right, string s, int n) {
        if(right == n) {
            result.push_back(s);
        }
        else
        {
            if(left < n)
            {
                generate(left + 1, right, s + "(", n);
            }
            if(right < left)
            {
                generate(left, right + 1, s + ")", n);
            }
        }
    }
};
```

我自己写的是用排列组合加上第 20 题的代码，有些繁琐不如上面的解法，为了更直观的理解这个递归的过程，就画了下面这个示意图。轻拍……



23

Merge k Sorted Lists（合并 K 个已排序链表）

翻译

合并 K 个已排序的链表，并且将其排序并返回。
分析和描述其复杂性。

原文

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

代码

我们采用分治的方法来解决这个问题，其有 K 个链表，不断将其划分（partition），再将其归并（merge）。

划分的部分并不难，将其不断分成两部分，但是需要注意的是可能出现 start 和 end 相等的情况，这时候就直接 return lists[start] 就可以了。

```
mid = (start + end) / 2
start -- mid                    (1)
mid + 1 -- end                  (2)
```

上面的（1）和（2）就不断的替换更新，不断作为参数写到 partition 函数内。

归并的部分也不难，因为在此之前我们已经遇到过，传送门：LeetCode 21 Merge Two Sorted Lists 。

所以结合起来就是下面这样的：

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*> &lists) {
        return partition(lists, 0, lists.size() - 1);
    }

    ListNode* partition(vector<ListNode*>& lists, int start, int end) {
        if(start == end) {
            return lists[start];
        }

        if(start < end) {
            int mid = (start + end) / 2;
            ListNode* l1 = partition(lists, start, mid);
            ListNode* l2 = partition(lists, mid + 1, end);
            return mergeTwoLists(l1, l2);
        }
        return NULL;
```

```
    }

    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if(l2 == NULL) return l1;
        if(l1 == NULL) return l2;

        if(l1->val > l2->val) {
            ListNode* temp = l2;
            temp->next = mergeTwoLists(l1, l2->next);
            return temp;
        } else {
            ListNode* temp = l1;
            temp->next = mergeTwoLists(l1->next, l2);
            return temp;
        }
    }
};
```

继续学习大神的的解法：

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*> &lists) {
        int size = lists.size();
        if(size == 0) return NULL;
        if(size == 1) return lists[0];

        int i = 2, j;
        while(i / 2 < size) {
            for(j = 0; j < size; j += i) {
                ListNode* p = lists[j];
                if(j + i / 2 < size) {
                    p = mergeTwoLists(p, lists[j + i / 2]);
                    lists[j] = p;
                }
            }
            i *= 2;
        }
```

```
        return lists[0];
    }

    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if(l2 == NULL) return l1;
        if(l1 == NULL) return l2;

        if(l1->val > l2->val) {
            ListNode* temp = l2;
            temp->next = mergeTwoLists(l1, l2->next);
            return temp;
        } else {
            ListNode* temp = l1;
            temp->next = mergeTwoLists(l1->next, l2);
            return temp;
        }
    }
};
```

24

Swap Nodes in Pairs（交换序列中的结点）

翻译

给定一个链表，调换每两个相邻节点，并返回其头部。

例如，
给定 1->2->3->4， 你应该返回的链表是 2->1->4->3。

你的算法必须使用唯一不变的空间。

你也不能修改列表中的值，只有节点本身是可以改变的。

原文

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

For example,

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

Your algorithm should use only constant space.

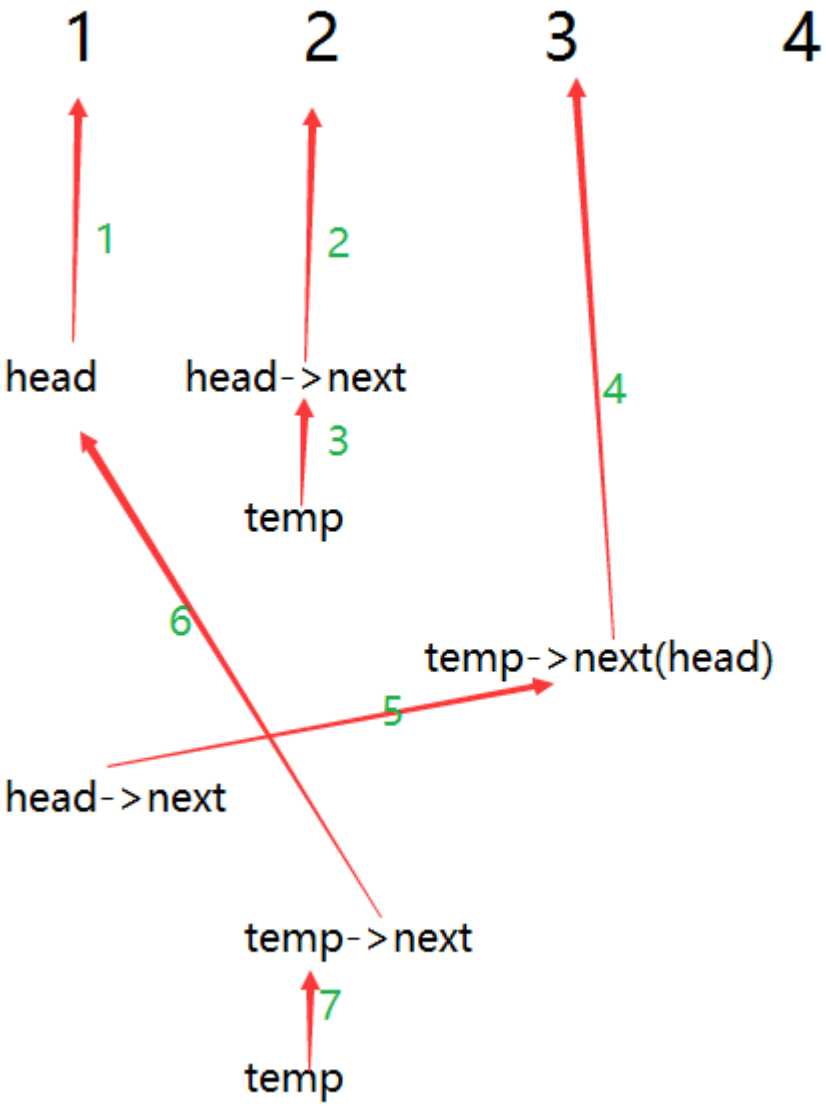
You may not modify the values in the list, only nodes itself can be changed.

分析

我们就以题目中给出的 1，2，3，4 作为示例，将其作为两部分

1，2 -- 3，4

或者我画个图出来更加直观吧……



代码

```
/**
 * 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) {
        if(head == NULL) return NULL;
        if(head->next == NULL) return head;

        ListNode* temp = head->next;
        head->next = swapPairs(temp->next);
        temp->next = head;

        return temp;
    }
};
```

25

Reverse Nodes in k-Group（在K组链表中反转结点）

原文

给定一个链表，在一定时间内反转这个链表的结点，并返回修改后的链表。

如果结点数不是K的倍数，那么剩余的结点就保持原样。

你不应该在结点上修改它的值，只有结点自身可以修改。

只允许使用常量空间。

例如

给定链表： 1->2->3->4->5

对于 k = 2，你应该返回： 2->1->4->3->5

对于 k = 3，你应该返回： 3->2->1->4->5

翻译

Given a linked list, reverse the nodes of a linked list k at a time and return its modified 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 nodes, only nodes itself may be changed.

Only constant memory is allowed.

For example,
Given this linked list: 1->2->3->4->5

For k = 2, you should return: 2->1->4->3->5

For k = 3, you should return: 3->2->1->4->5

代码

下面的代码并不是我的，虽然我想的差不多，但就是这个差不多导致结果的差异性……

任重而道远啊！

```
/**
 * 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) {
        auto node = head;
        for(int i = 0; i < k; ++ i) {
            if(!node) return head;
            node = node->next;
        }

        auto new_head = reverse(head, node);
        head->next = reverseKGroup(node, k);
        return new_head;
    }

    ListNode* reverse(ListNode* start, ListNode* end) {
        ListNode* head = end;

        while(start != end) {
            auto temp = start->next;
            start->next = head;
            head = start;
            start = temp;
        }

        return head;
    }
};
```

26

Remove Duplicates from Sorted Array（从已排序数组中移除重复元素）

翻译

给定一个已排序的数组，删除重复的元素，这样每个元素只出现一次，并且返回新的数组长度。

不允许为另一个数组使用额外的空间，你必须就地以常量空间执行这个操作。

例如，
给定输入数组为 [1,1,2]

你的函数应该返回 length = 2， 其前两个元素分别是 1 和 2。它不关心你离开后的新长度。

原文

Given a sorted array, 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 in place with constant memory.

For example,
Given input array 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 new length.

代码

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if (nums.begin() == nums.end()) return 0;
        vector<int>::iterator itor;
        for (itor = nums.begin(); itor != nums.end() && itor + 1 != nums.end(); ++itor) {
            while (*itor == *(itor + 1)) {
                nums.erase(itor + 1);
                if (itor + 1 == nums.end())
                    break;
            }
        }
        return nums.size();
    }
};
```

27

Implement strStr()（实现 strStr() 函数）

翻译

实现 strStr() 函数。

返回针（needle）在草垛/针垛（haystack）上第一次出现的索引， 如果不存在其中则返回 -1。

其实也就是说字符串 str2 在字符串 str1 中第一次出现的索引而已。

原文

Implement strStr().

Returns the index of the first occurrence of needle in haystack,

or -1 if needle is not part of haystack.

代码

```
class Solution {
public:
    bool compare(string s1, int index, string s2) {
        int count = 0;
        for (int i = 0; i < s2.length(); i++) {
            if (s2[i] == s1[i + index])
                count++;
        }
        if (count == s2.length())
            return true;
        return false;
    }
    int strStr(string haystack, string needle) {
        for (int i = 0; i < haystack.length(); i++) {
            if (haystack[i] == needle[0]) {
                if (compare(haystack, i, needle))
                    return i;
            }
        }
        return -1;
    }
};
```

发现超时了……其实在测试之前就看到了别人的答案……惊呆了……这样都可以？

```
class Solution {
    public:
        int strStr(string haystack, string needle) {

            return haystack.find(needle);

        }
    };
```

也算是长见识了……