1、双指针技巧套路框架

双指针技巧可以分为两类：1、“快慢指针”类型　　2、“左、右指针”类型

1.1、“快慢指针”类型:解决的链表中的问题,典型：判断链表中是否包含环

1.2、“左、右指针”类型：解决的是数组中的问题,典型：二分搜索

快慢指针:

1.1、判断单链表中是否包含有环

> 算法：快指针在前，慢指针在后，如果链表中有环，则快指针会比慢指针快一圈并相遇;如果链表中没有环，则快指针会遇到Null

```
boolean hasCycle(ListNode head){
 ListNode fast,slow;
 //初始化快指针和慢指针
 fast=slow=head;
 while(fast!=null&&fast.next!=null){
 //快指针前进两步
 fast=fast.next.next;
 //慢指针前进一步
 slow=slow.next;
  if(fast==slow) return true;
  }
  return false;
}
```

1.2已知链表中含有环，返回这个环的起始位置

> 假设fast指针和slow指针相遇时slow指针走了k步,fast指针走了2k步,则两个指针相差k步(环长度的整数倍)
> 设环的起点到相遇点的距离为m步,则从head出发到相遇点的距离为k-m步，且从相遇点到环起点的距离为k-m步
> 综上所述，当fast和slow指针相遇时，将slow指针放在head处然后slow指针和fast指针相遇时就是环的起点

```
ListNode detectCycle(ListNode head){
 ListNode fast,slow;
 fast=slow=head;
 while(fast!=null&&fast.next!=null){
 slow=slow.next;
 fast=fast.next.next;
 if(slow==fast) break;//已知这个链表中存在环
 }
 //此时把slow指针放在head处
```

```
 slow=head;
 while(slow!=fast){
 slow=slow.next;
 fast=fast.next;
 }
 return slow;
}
```

1.3寻找无环单链表的中点

> 单链表的中点是指:假设单链表的长度为n(不包含头结点),则中点为n/2的向下取整

```
ListNode midNode(ListNode head){//head头指针不算是一个单链表中的一个点
 ListNode fast,slow;
 fast=slow=head;
 while(fast!=null&&fast.next!=null){
 fast=fast.next.next;
 slow=slow.next;
 }
 return slow;
}
```

1.4寻找单链表的倒数第k个元素

> 思路:先将快指针**移动k个元素**，然后快慢指针**共同速度**移动，这样当快速指针移动链表末尾**null**时，慢指针所在的位置就是倒数第k个链表节点

```
ListNode fast,slow;
fast=slow=head;
while(k){
 fast=fast.next;
 k--;
}
while(fast){
 fast=fast.next;
 slow=slow.next;
}
return slow;
```

左右指针

左右指针在数组问题中，实际上是指两个索引值,一般初始化为**left=0，right=len(nums)-1**

2.1两数之和

> 输入一个已按照**升序排列**的**有序数组nums和一个目标值target**,在nums中找到两个数使得它们相加之和等于target,请返回**两个数的索引**

```
int[] twoSum(int[] nums,int target){
 int left=0,right=nums.size()-1;
```

```
while(left<right){
int ans=nums[left]+nums[right];
if(ans==target) break;
if(ans<target) left++;
if(ans>target) right--;
}
return [left,right];
}
```

2.2滑动窗口算法--双指针技巧的最高境界,它是快慢指针在数组上的应用

作用：解决一**大串字符串匹配**的问题