剑指offer-python版（5-8）

5用两个栈实现队列

题目：用两个栈来实现一个队列，完成队列的Push和Pop操作。 队列中的元素为int类型。

思路：第一个栈临时保存插入的数据，当调用弹出函数的时候，如果stack2不为空则直接弹出；为空则把stack1中的数据全部弹出放到stack2中。

这样不会存在冲突，而且由于stack2保存的是以前的老数据，弹出一定都符合队列的规律

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** \_\_init\_\_(self):
4. self.stack1=[]
5. self.stack2=[]
6. **def** push(self, node):
7. # write code here
8. **return** self.stack1.append(node)
9. **def** pop(self):
10. # return xx
11. **if** self.stack2:
12. **return** self.stack2.pop()
13. **else**:
14. **while** self.stack1:
15. self.stack2.append(self.stack1.pop())
16. **return** self.stack2.pop()

6旋转数组的最小数字

题目：把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。 输入一个非减排序的数组的一个旋转，输出旋转数组的最小元素。 例如数组{3,4,5,1,2}为{1,2,3,4,5}的一个旋转，该数组的最小值为1。 NOTE：给出的所有元素都大于0，若数组大小为0，请返回0。

思路1：（暴力搜索，以后要避免）根据给定的数组特点，从左到右遍历数组元素，当首次遇到数组中某个元素比上一个元素小时，该元素就是我们需要的元素

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** minNumberInRotateArray(self, rotateArray):
4. # write code here
5. **if** **not** rotateArray:
6. **return** 0
7. lenArr=len(rotateArray)
8. min=rotateArray[0]
9. **for** i **in** range(1,lenArr):
10. **if** rotateArray[i] >= min:
11. min = rotateArray[i]
12. **else**:
13. **return** rotateArray[i]

思路2：旋转数组也算是一种排序数组，可以用二分法解题，使时间复杂度最低控制在O(lgN)。（一开始有些顾虑觉得旋转后就不可以使用二分法了）用变量mid定位到数组的中间位置，将数组头部的值与mid处的值进行比较，不断缩小数组。当mid处的值等于数组头部的值时，即当代码需要遍历时，可用一行代码代替for循环：

1. **def** minNumberInRotateArray(self, rotateArray):
2. **if** **not** rotateArray:
3. **return** 0
4. **if** len(rotateArray)==2:
5. **return** rotateArray[1]
7. mid = int(len(rotateArray)/2)
8. **if** rotateArray[mid] > rotateArray[0]:
9. **return** self.minNumberInRotateArray(rotateArray[mid:])
10. **elif** rotateArray[mid] < rotateArray[0]:
11. **return** self.minNumberInRotateArray(rotateArray[:mid+1])
12. **else**:
13. **return** self.minNumberInRotateArray(rotateArray[1:]

思路3：左右指针

时间复杂度和空间复杂度都比较低，但是如果稍微不注意，还是会掉到小坑里，比如如果用mid处的值和数组左边的值比较，会有什么不同；else语句中如果不让right处的指向左移一位，而是让left处的指针右移一位会怎么样，都是需要稍加留意的地方

1. **def** minNumberInRotateArray(self, rotateArray):
2. left = 0
3. right = len(rotateArray)-1
4. **while** left < right:
5. mid = int((left+right)/2)
6. **if** rotateArray[mid] > rotateArray[right]:
7. left = mid+1
8. **elif** rotateArray[mid] < rotateArray[right]:
9. right = mid
10. **else**:
11. right -= 1
12. **return** rotateArray[left]

7斐波那契数列

题目：大家都知道斐波那契数列，现在要求输入一个整数n，请你输出斐波那契数列的第n项（从0开始，第0项为0）。

n<=39

思路：直接使用前两个数字的和相加得到第三个数字的方式能把时间复杂度降到O（n）。

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** Fibonacci(self, n):
4. # write code here
5. a,b=0,1
6. **for** i **in** range(n):
7. a,b=b,a+b
8. **return** a

思路2：递归容易有很多的重复计算，所以使用一个字典来存储已经计算好的值

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** \_\_init\_\_(self):
4. self.keep = {0:0, 1:1}
6. **def** Fibonacci(self, n):
7. **if** n **in** self.keep:
8. **return** self.keep[n]
9. **else**:
10. fn = self.Fibonacci(n - 1) + self.Fibonacci(n - 2)
11. self.keep[n] = fn
12. **return** fn

8跳台阶

题目：一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法（先后次序不同算不同的结果）

思路：典型的动态规划题目

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** jumpFloor(self, number):
4. # write code here
5. **If number==0:**
6. **Return 0**
7. **if** number==1:
8. **return** 1
9. **if** number==2:
10. **return** 2
11. dp[number]=jumpFloor[number-1]+jumpFloor[number-2]
12. **return** dp[number]

思路二：斐波那契的简化版

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** jumpFloor(self, number):
4. # write code here
5. ans = [];
6. ans.append(0);
7. ans.append(1);
8. ans.append(2);
9. **for** i **in** range(3 , number+1):
10. ans.append(ans[i-1]+ans[i-2]);
11. **return** ans[number];