第二课 线程池与任务队列

1.链表复习

1.1相关题目

1.1.1 分隔链表

解题重点:

1.创建两个新链表头来牵引

课堂代码

In [ ]:
```
class Solution:
    def partition(self, head: ListNode, x: int) -> ListNode:
        big_head = ListNode()
        small_head = ListNode()
        cur = head
        big_pre = big_head
        small_pre = small_head
        while cur:#遍历结束条件
            if cur.val >= x:
                big_pre.next = cur
                big_pre = big_pre.next
            else:
                small_pre.next = cur
                small_pre = small_pre.next
            cur = cur.next
        big_pre.next = None
        small_pre.next = big_head.next
        return small_head.next
```

1.1.2 复制带随机指针的链表

解题重点:

1.将复制的链表先保存在原链表，保持原链表的相对关系

2.理解为啥random可以直接random.next，然后复制的链表的random关系就对齐了

3.注意next.next时的判定

课堂代码

In [ ]:
```
class Solution:
    def copyRandomList(self, head: 'Node') -> 'Node':
        if not head:return
        pre = head
        while pre:#复制链表
            q = Node(pre.val, pre.next, pre.random)#新建链表节点,复制next和random
            pre.next = q
            pre = pre.next.next#要走两步，才能到下一个原节点，
        pre = head.next
        while pre:
            if pre.random:
                pre.random = pre.random.next
            pre = pre.next#pre = pre.next.next
            if pre:
```

```
                pre = pre.next
        pre = head.next
        new_head = head.next
        while pre.next:
            head.next = head.next.next
            pre.next = pre.next.next
            head = head.next
            pre = pre.next
        head.next = None
        return new_head
```

2.队列的封装与使用

2.1相关题目

2.1.1 设计循环队列

解题重点:

1.指针下标超界问题，用if或者%将其避开

课堂代码

In [ ]:
```
class MyCircularQueue:

    def __init__(self, k: int):
        self.queue = [-1 for i in range(k)]
        self.front = 0
        self.rear = 0
        self.length = 0
        self.k = k


    def enQueue(self, value: int) -> bool:
        if self.isFull():return False
        self.queue[self.rear] = value
        self.rear = self.rear + 1 if self.rear + 1 < self.k else 0 # (self.rear + 1
        self.length += 1
        return True


    def deQueue(self) -> bool:
        if self.isEmpty():return False
        self.queue[self.front] = -1
        self.front = self.front + 1 if self.front + 1 < self.k else 0
        self.length -= 1
        return True


    def Front(self) -> int:
        return self.queue[self.front]

    def Rear(self) -> int:
        return self.queue[self.rear - 1 if self.rear - 1 >= 0 else self.k - 1]

    def isEmpty(self) -> bool:
        return self.length == 0


    def isFull(self) -> bool:
        return self.length == self.k
```

2.1.2 设计循环双端队列

解题重点:

1.注意front和rear在删除和增加的时候下标的移动方向

2.注意初始值的设定

课堂代码

In [ ]:

```
class MyCircularDeque:

    def __init__(self, k: int):
        """
        Initialize your data structure here. Set the size of the deque to be k.
        """
        self.arr = [-1 for _ in range(k)]
        self.k = k
        self.length = 0
        self.head = 0
        self.tail = 1

    def insertFront(self, value: int) -> bool:
        """
        Adds an item at the front of Deque. Return true if the operation is successful
        """
        if self.isFull():return False
        #print(self.length)
        self.arr[self.head] = value
        self.head = self.head - 1 if self.head - 1 >= 0 else self.k - 1
        self.length += 1
        return True

    def insertLast(self, value: int) -> bool:
        """
        Adds an item at the rear of Deque. Return true if the operation is successful.
        """
        if self.isFull():
            return False
        self.arr[self.tail] = value
        self.tail = (self.tail + 1) % self.k
        self.length += 1
        return True

    def deleteFront(self) -> bool:
        """
        Deletes an item from the front of Deque. Return true if the operation is succe
        """
        if self.isEmpty():return False
        self.head = (self.head + 1) % self.k
        self.arr[self.head] = -1
        self.length -= 1
        return True


    def deleteLast(self) -> bool:
        """
        Deletes an item from the rear of Deque. Return true if the operation is succes
        """
        if self.isEmpty():return
        self.tail = self.tail - 1 if self.tail - 1 >= 0 else self.k - 1
        self.arr[self.tail] = -1
        self.length -= 1
        return True

    def getFront(self) -> int:
        """
        Get the front item from the deque.
```

```
        """
        # print(self.arr)
        # print(self.head)
        # print(self.tail)
        return self.arr[(self.head + 1) % self.k]

    def getRear(self) -> int:
        """
        Get the last item from the deque.
        """
        return self.arr[self.tail - 1 if self.tail - 1 >= 0 else self.k - 1]

    def isEmpty(self) -> bool:
        """
        Checks whether the circular deque is empty or not.
        """
        return self.length == 0

    def isFull(self) -> bool:
        """
        Checks whether the circular deque is full or not.
        """
        return self.length == self.k
```

2.1.3 设计前中后队列

解题重点:

1.前中后队列，使用双向链表加3指针方便操作，注意判断只要1、2元素队列的情况

2.如果用列表则简单至极

课堂代码

In [ ]:

```
class Node:
    def __init__(self, val):
        self.val = val
        self.next = self.prev = None


class FrontMiddleBackQueue:

    def __init__(self):
        self.head = self.tail = self.mid = None
        self.sign = False #作为链表当前奇偶判断，True为奇数，False为偶数


    def pushFront(self, val: int) -> None:
        if not self.head:#判断队列为空的情况
            self.head = self.tail = self.mid = Node(val)
            self.sign = True
            return

        self.head.prev = Node(val)
        self.head.prev.next = self.head
        self.head = self.head.prev #将头节点链接新节点，并将head指针移到前面
        if self.sign:
            self.mid = self.mid.prev
            self.sign = False

        else:
            self.sign = True
```

```
    def pushMiddle(self, val: int) -> None:
        if not self.head:#判断队列为空的情况
            self.head = self.tail = self.mid = Node(val)
            self.sign = True
            return

        node = Node(val)
        if self.sign:#判断当前队列奇偶
            self.sign = False#为奇数则添加元素后变为偶数
            if self.mid.prev:#判断是否是单个元素的队列
                self.mid.prev.next = node
                node.prev = self.mid.prev
                self.mid.prev = node
                node.next = self.mid
                self.mid = node
            else:#如果是单个元素，将头和mid放最前面
                self.mid.prev = node
                node.next = self.mid
                self.head = self.mid = node
        else:#偶数的情况要么0个，要么2个以上
            node.next = self.mid.next
            self.mid.next = node.next.prev = node
            node.prev = self.mid
            self.mid = node
            self.sign = True




    def pushBack(self, val: int) -> None:
        if not self.head:#判断队列为空的情况
            self.head = self.tail = self.mid = Node(val)
            self.sign = True
            return

        self.tail.next = Node(val)
        self.tail.next.prev = self.tail
        self.tail = self.tail.next
        if self.sign:
            self.sign = False

        else:
            self.mid = self.mid.next
            self.sign = True


    def popFront(self) -> int:
        if not self.head:#判断队列为空的情况
            return -1
        res = self.head.val
        self.head = self.head.next
        if not self.head:
            self.mid = self.tail = None
            self.sign = False
            return res

        self.head.prev = None
        if self.sign:
            self.sign = False
        else:
            self.mid = self.mid.next
            self.sign = True
        return res
```

```
    def popMiddle(self) -> int:
        if not self.head:#判断队列为空的情况
            return -1

        res = self.mid.val
        if not self.head.next:#表示只有一个元素的队列情况
            self.head = self.mid = self.tail = None
            self.sign = False
            return res
        if not self.mid.prev:#表示只有两个元素的情况下
            self.head = self.mid = self.tail = self.mid.next
            self.head.prev = None
            self.sign = True
            return res

        self.mid.prev.next = self.mid.next
        self.mid.next.prev = self.mid.prev
        if self.sign:
            self.mid = self.mid.prev
            self.sign = False
        else:
            self.mid = self.mid.next
            self.sign = True
        return res


    def popBack(self) -> int:
        if not self.head:#判断队列为空的情况
            return -1
        res = self.tail.val
        self.tail = self.tail.prev
        if not self.tail:#表示删除前只有一个元素的情况下
            self.mid = self.head = None
            self.sign = False
            return res

        self.tail.next = None
        if self.sign:
            self.mid = self.mid.prev
            self.sign = False

        else:
            self.sign = True
        return res
```

2.1.4 最近请求次数

解题重点:

1.题目理解比较困难

2.按照范围，依次把队列中的数删除（直到在范围内）

课堂代码

In [ ]:
```
class RecentCounter:

    def __init__(self):
        self.requests = []


    def ping(self, t: int) -> int:
        limit_down = max(0, t - 3000)
        self.requests.append(t)
```

```
        while self.requests[0] < limit_down:
            self.requests.pop(0)
        return len(self.requests)
```

3.智力发散题

3.1.1 第K个数

解题重点:

1.理解素因子的关系，第K个数一定由前k-1个数与3、5、7中的某个数相乘所得

2.记录我们使用过的前K-1个数

3.当我们的3、5、7在使用前k-1个数时，满足条件的值对应的3、5、7都要被记录

课堂代码

In [ ]:
```
class Solution:
    def getKthMagicNumber(self, k: int) -> int:
        k_list = [1]
        P3,P5,P7 = 0,0,0
        for i in range(1,k):
            data3 = k_list[P3] * 3 #3--素因子
            data5 = k_list[P5] * 5 #5--素因子
            data7 = k_list[P7] * 7 #7--素因子
            k_list.append(min(min(data3,data5),data7))
            if k_list[i] == data3:P3 += 1
            if k_list[i] == data5:P5 += 1
            if k_list[i] == data7:P7 += 1
        return k_list[k - 1]
```

3.1.2 亲密字符串

解题重点:

1.判断是否长度相等

2.判断不相同的次数，如果是0次或者2次则有机会成为亲密字符串

课堂代码

In [ ]:
```
class Solution:
    def buddyStrings(self, a: str, b: str) -> bool:
        if len(a) != len(b):return False
        same_num = []
        for i in range(len(a)):
            if a[i] != b[i]:
                same_num.append(i)
        if len(same_num) == 2:
            if a[same_num[0]] == b[same_num[1]] and a[same_num[1]] == b[same_num[0]]
                return True
        if len(same_num) == 0:
            return len(a) > len(set(a))
        return False
```

3.1.3柠檬水找零

解题重点:

1.分情况讨论就行

2.注意20元时的找零方法，优先10+5，

课堂代码

In [ ]:
```
class Solution:
    def lemonadeChange(self, bills: List[int]) -> bool:
        five = 0
        ten = 0
        twenty = 0
        for bill in bills:
            if bill == 5:
                five += 1
            elif bill == 10:
                if five > 0:
                    five -= 1
                    ten += 1
                else:
                    return False
            else:
                if ten > 0 and five > 0:#找10快+5快
                    ten -= 1
                    five -= 1
                    twenty += 1
                elif five >= 3:#3个5快
                    five -= 3
                    twenty += 1
                else:
                    return False
        return True
```

3.1.4煎饼排序

解题重点:

1.理解煎饼反转的一个操作

2.每次循环中反转两次，每次先将当前最大的放到第一位，第二次将当前最大的翻转到当前的最后一位，重复这个操作。

3.给的数据特殊，能够通过length判断最大值

课堂代码

In [ ]:
```
class Solution:
    def pancakeSort(self, arr: List[int]) -> List[int]:
        max_val = len(arr)#因为生成的数据特性
        k_list = []
        while max_val > 1:
            max_idx = arr.index(max_val)
            if max_idx != max_val - 1:
               arr[:max_idx + 1] = arr[:max_idx + 1][::-1]#到此为第一次反转
               k_list.append(max_idx + 1)#记录第一次反转
               arr[:max_val] = arr[:max_val][::-1]#到此为第二次反转
               k_list.append(max_val)#记录第二次反转
            max_val -= 1#生成数据的特性
        return k_list
    # arr = [1,2,3,4]
    # print(arr[::-1]) ==>123
```

3.1.5任务调度器

解题重点:

课堂代码

In [ ]:
```
class Solution:
```

```python
    def leastInterval(self, tasks: List[str], n: int) -> int:
        count_list = [0 for _ in range(26)]
        for task in tasks:
            count_list[ord(task) - ord('A')] += 1
        List.sort(count_list)
        max_count = 0
        for i in range(len(count_list)):
            if count_list[25] == count_list[len(count_list) - 1 - i]:
                max_count += 1
            else:
                break
        #通过上述操作，已经统计了最大之的个数
        return max(len(tasks),(count_list[25] - 1) * (n + 1) + max_count)
```