第一周学习总结

1. 一维数组坐标变换
   数组的 delete 一般是通过赋值来改变确定坐标位置的元素值来实现的；
   数组的操作一般是通过遍历坐标，不断 swap 两个坐标的值来实现。

2. 翻转链表
   Python 的一行代码实现，简洁明了：
   ```
   def reverselist(self, head):
       prev, cur = None, head
       while cur:
           cur.next, prev, cur = prev, cur, cur.next
       return prev
   ```

3. 有序数组或链表合并
   数组：while 循环迭代遍历
   ```
   def mergeArrays(self, nums1, m, nums2, n):
       while m > 0 and n > 0:
           if nums1[m-1] >= nums2[n-1]:
               nums1[m+n-1] = nums1[m-1]
               m -=1
           else:
               nums1[m+n-1] = nums2[n-1]
               n -= 1
       if n > 0:
           nums1[:n] = nums2[:n]
   ```

   链表：递归调用遍历
   ```
   public ListNode mergeTwoLists(ListNode l1, ListNode l2){
       if(l1 == null) return l2;
       if (l2 == null) return l1;

       if (l1.val < l2.val){
           l1.next = mergeTwoLists(l1.next, l2);
           return l1;
       }
       else{
           l2.next = mergeTwoLists(l1, l2.next);
           return l2;
       }
   }
   ```

4. 对称和类似剥洋葱型结构的问题应该想到用 stack 来解决
   一维数组遍历 + stack 可以解决很多原本需要 n 平方时间复杂度的问题

一、优先队列源代码分析

Class PriorityQueue<E>

`add(E e)` `offer(E e)`

```
public boolean add(E e) { return offer(e); }
```

实际执行方法：

```
public boolean offer(E e) {
        if (e == null)
            throw new NullPointerException();
        modCount++;
        int i = size;
        if (i >= queue.length)
            grow(i + 1);
        size = i + 1;
        if (i == 0)
            queue[0] = e;
        else
            siftUp(i, e);
        return true;
    }
```

添加 E type 的数据到 PriorityQueue 里去,这里涉及到 siftUp(i, e)方法，具体操作见源码：

```
private void siftUp(int k, E x) {
        if (comparator != null)
            siftUpUsingComparator(k, x);
        else
            siftUpComparable(k, x);
```

这里涉及到比较器 comparator

`comparator()`

比较器方法

优先队列的头是基于自然排序的最小元素，如果存在比较器即 comparator!=nullz 就按照比较器排序，siftUpUsingComparator(k, x)将在 k 位置上添加进去的元素 x 进行操作，底层实现是在一个排序树中将 x 不断和它的 parent node 进行比较，往上移动，直到它等于或大于它的 parent node 为止。具体由下面两个方法实现：

```
private void siftUpComparable(int k, E x) {
        Comparable<? super E> key = (Comparable<? super E>) x;
        while (k > 0) {
            int parent = (k - 1) >>> 1;
            Object e = queue[parent];
            if (key.compareTo((E) e) >= 0)
                break;
```

```
                queue[k] = e;
                k = parent;
            }
            queue[k] = key;
        }


private void siftUpUsingComparator(int k, E x) {
            while (k > 0) {
                int parent = (k - 1) >>> 1;
                Object e = queue[parent];
                if (comparator.compare(x, (E) e) >= 0)
                    break;
                queue[k] = e;
                k = parent;
            }
            queue[k] = x;
        }
```

remove(Object o)

移除元素

非空返回 true，空元素返回 false

```
public boolean remove(Object o) {
            int i = indexOf(o);
            if (i == -1)
                return false;
            else {
                removeAt(i);
                return true;
            }
        }
```

具体实现：

```
private E removeAt(int i) {
            // assert i >= 0 && i < size;
            modCount++;
            int s = --size;
            if (s == i) // removed last element
                queue[i] = null;
            else {
                E moved = (E) queue[s];
                queue[s] = null;
                siftDown(i, moved);
                if (queue[i] == moved) {
                    siftUp(i, moved);
                    if (queue[i] != moved)
```

```
                    return moved;
                }
            }
            return null;
        }
```

`contains(Object o)`

判断队列中是否存在一个元素，存在就返回 true

```
public boolean contains(Object o) {
        return indexOf(o) != -1;
    }
```

具体实现：

```
private int indexOf(Object o) {
        if (o != null) {
            for (int i = 0; i < size; i++)
                if (o.equals(queue[i]))
                    return i;
        }
        return -1;
    }
```

还有 peek()方法

```
public E peek() {
        return (size == 0) ? null : (E) queue[0];
    }
```

队列空返回 null，不空返回队首元素。

二．Deque 实例代码改写

将代码中所有 push 函数改为 addfirst 函数即可。