学习情况表

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 杜金瑞 | **学号** | 2020905073 |
| **学院** | 信息工程学院 | **专业** | 计算机科学与技术 |

|  |
| --- |
| 学习情况简述 |
| C:\Users\29043\Documents\Tencent Files\2904326062\Image\C2C\239E78DD0BEF45E9A92BB5558765A47F.jpg |
| 本周练习过的代码（例） |
| public class MaxHeap <E extends Comparable<E>>{   private Array data;   public MaxHeap(int capacity){  data=new Array(capacity) ;  }   public MaxHeap(){  data =new Array();  }   public MaxHeap(E[] arr){  data=new Array<>(arr);  for(int i=parent(arr.length-1);i>=0;i++)  SiftDown(i);  }   public int getSize(){  return data.Getsize();  }   public boolean isEmpty(){  return data.isEmpty();  }   //返回index索引表示元素的父亲节点  public int parent(int index){  if(index==0)  throw new IllegalArgumentException("Don't have parent");  return index-1/2;  }   public int leftChild(int index){  return 2\*index+1;  }   public int rightChild(int index){  return 2\*index+2;  }   public void Add(E e){  data.AddList(e);  Siftup(data.Getsize()-1);  }  private void Siftup(int k){   while(k>0&&data.getData(parent(k)).compareTo(data.getData(k))>0){  data.swap(k,parent(k));  k=parent(k);  }  }   public E findMax(){  if(data.Getsize()==0)  throw new IllegalArgumentException("Empty!");   return (E) data.getData(0);  }  //取出最大元素  public E ExtractMax(){  E ret =findMax();   data.swap(0, data.Getsize()-1);  data.removeLast();  SiftDown(0);   return ret;  }  public void SiftDown(int k){  while(leftChild(k)<data.Getsize()) {  int j = leftChild(k);  if(j+1<data.Getsize() && data.getData(j+1).compareTo(data.getData(j))>0)  j=rightChild(k);   if(data.getData(k).compareTo(data.getData(j))>=0)  break;  data.swap(k,j);  k=j;  }  }   public E replace(E e){  E ret=findMax();  data.setData(0,e);  SiftDown(0);  return ret;  }   public static<E extends Comparable<E>> void Minheap(E[] data,int k) {  while (2 \* k - 1 < data.length) {  int j=2\*k-1;  if(j+1< data.length&&data[j+1].compareTo(data[j])<0)  j++;   if(data[k].compareTo(data[j])<0)  break;   *swap*(data,k,j);  k=j;  }  }   public static <E extends Comparable<E>> void swap(E[] data,int i,int j){  E temp= data[i];  data[i]=data[j];  data[j]=temp;  } }  public interface Queue <E>{   int getSize();  boolean isEmpty();  void enqueue(E e);  E dequeue();  E getFront();  }  public class HeapSort {   private HeapSort(){}   public static<E extends Comparable<E>> void sort(E[] data){  MaxHeap<E> maxHeap=new MaxHeap<>();  for(E e:data)  maxHeap.Add(e);   for(int i=data.length-1;i>=0;i--)  data[i]= maxHeap.ExtractMax();  }   public static <E extends Comparable<E>> void sort2(E[] data){   if(data.length<=1) return;  for(int i= (data.length-2)/2;i>=0;i--)  *SiftDown*(data,i,data.length);  for(int i= data.length-1;i>=0;i--){  *swap*(data,0,i);  *SiftDown*(data,0,i);  }  }  //对于data[0,n)元素形成的最大堆中，对索引k的元素SiftDown  public static <E extends Comparable<E>> void SiftDown(E[] data,int k,int n){  while(2\*k+1<n){  int j=2\*k+1;  if(j+1<n&& data[j+1].compareTo(data[j])>0)  j++;   if(data[j].compareTo(data[k])<=0)  break;  *swap*(data,j,k);  k=j;  }  }  public static <E extends Comparable<E>> void swap(E[] data,int i,int j){  E temp= data[i];  data[i]=data[j];  data[j]=temp;  } }  import javax.xml.bind.annotation.XmlAnyAttribute;  public class PriorityQueue <E extends Comparable<E>> implements Queue<E>{   private MaxHeap<E> maxHeap;   public PriorityQueue(){  maxHeap=new MaxHeap<>();  }   @Override  public boolean isEmpty(){  return maxHeap.isEmpty();  }   @Override  public int getSize(){  return maxHeap.getSize();  }   @Override  public E getFront(){  return maxHeap.findMax();  }   @Override  public void enqueue(E e){  maxHeap.Add(e);  }   @Override  public E dequeue(){  return maxHeap.ExtractMax();  }  }  import java.util.Collections; import java.util.PriorityQueue; class Solution {  public int[] getLeastNumbers(int[] arr, int k) {   PriorityQueue<Integer> q=new PriorityQueue<>(Collections.*reverseOrder*());   for( int i=0;i<k;i++){  q.add(arr[i]);  }   for(int j=k;j<arr.length;j++)  if(!q.isEmpty()&&arr[j]<q.peek()) {  q.remove();  q.add(arr[j]);  }    int[] res=new int[k];  for(int i=0;i<k;i++)  res[i]=q.remove();  return res;  } }  import java.util.PriorityQueue; public class Solution2 {   public int findKthLargest(int[] nums, int k){   PriorityQueue<Integer> q=new PriorityQueue<>();  for(int i=0;i<k;i++)  q.add(nums[i]);    for(int j=k;j<nums.length;j++)  if(!q.isEmpty()&&nums[j]>q.peek()) {  q.remove();  q.add(nums[j]);  }  return q.peek();    } } |