8.优先队列

定义及图示

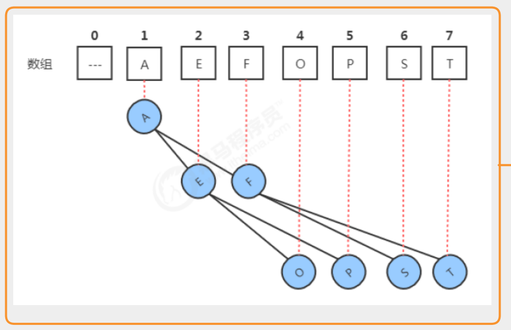
普通的队列是一种先进先出的数据结构，元素在队列尾追加，而从队列头删除。在某些情况下，我们可能需要找出队列中的最大值或者最小值，例如使用一个队列保存计算机的任务，一般情况下计算机的任务都是有优先级的，我们需要在这些计算机的任务中找出优先级最高的任务先执行，执行完毕后就需要把这个任务从队列中移除。普通的队列要完成这样的功能，需要每次遍历队列中的所有元素，比较并找出最大值，效率不是很高，这个时候，我们就可以使用一种特殊的队列来完成这种需求，优先队列。

最大优先队列

- API设计
  - 类名：MaxPriorityQueue
  - 构造方法
    - MaxPriorityQueue(int capacity)：创建容量为capacity的MaxPriorityQueue对象
  - 成员方法
    - 1.private boolean less(int i,int j)：判断堆中索引i处的元素是否小于索引j处的元素
    - 2.private void exch(int i,int j):交换堆中i索引和j索引处的值
    - 3.public T delMax():删除队列中最大的元素,并返回这个最大元素
    - 4.public void insert(T t)：往队列中插入一个元素
    - 5.private void swim(int k):使用上浮算法，使索引k处的元素能在堆中处于一个正确的位置
    - 6.private void sink(int k):使用下沉算法，使索引k处的元素能在堆中处于一个正确的位置
    - 7.public int size():获取队列中元素的个数
    - 8.public boolean isEmpty():判断队列是否为空
  - 成员变量
    - 1.private T[] imtes : 用来存储元素的数组
    - 2.private int N：记录堆中元素的个数

最小优先队列

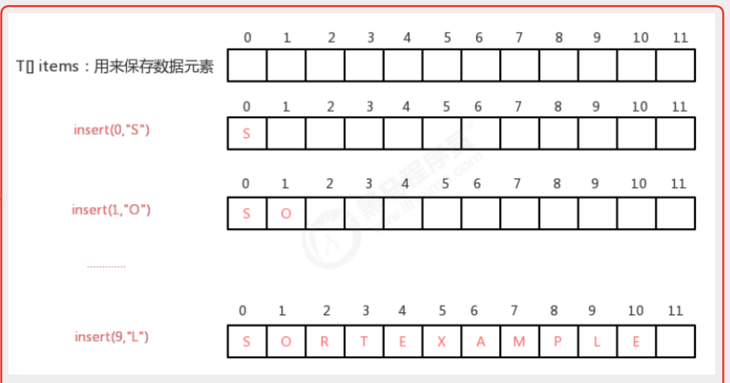
- 最小堆
  - 特性
    - 1.最小的元素放在数组的索引1处。
    - 2.每个结点的数据总是小于等于它的两个子结点的数据。
  - 图示



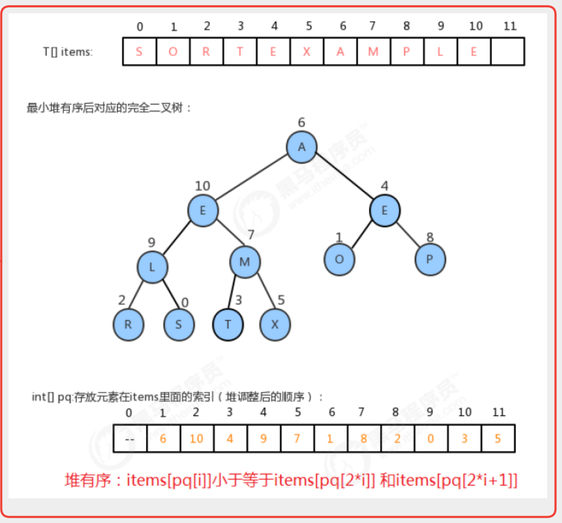
- API设计
  - 类名：MinPriorityQueue
  - 构造方法
    - MinPriorityQueue(int capacity)：创建容量为capacity的MinPriorityQueue对象
  - 成员方法
    - 1.private boolean less(int i,int j)：判断堆中索引i处的元素是否小于索引j处的元素
    - 2.private void exch(int i,int j):交换堆中i索引和j索引处的值
    - 3.public T delMin():删除队列中最小的元素,并返回这个最小元素
    - 4.public void insert(T t)：往队列中插入一个元素
    - 5.private void swim(int k):使用上浮算法，使索引k处的元素能在堆中处于一个正确的位置
    - 6.private void sink(int k):使用下沉算法，使索引k处的元素能在堆中处于一个正确的位置
    - 7.public int size():获取队列中元素的个数
    - 8.public boolean isEmpty():判断队列是否为空
  - 成员变量
    - 1.private T[] imtes : 用来存储元素的数组
    - 2.private int N：记录堆中元素的个数

索引优先队列

- 设计思路
  - 步骤一
    - 存储数据时，给每一个数据元素关联一个整数，例如insert(int k,T t),我们可以看做k是t关联的整数，那么我们的实现需要通过k这个值，快速获取到队列中t这个元素，此时有个k这个值需要具有唯一性。
    - 最直观的想法就是我们可以用一个T[] items数组来保存数据元素，在insert(int k,T t)完成插入时，可以把k看做是items数组的索引，把t元素放到items数组的索引k处，这样我们再根据k获取元素t时就很方便了，直接就可以拿到items[k]即可。



  - 步骤二
    - 步骤一完成后的结果，虽然我们给每个元素关联了一个整数，并且可以使用这个整数快速的获取到该元素，但是，items数组中的元素顺序是随机的，并不是堆有序的，所以，为了完成这个需求，我们可以增加一个数组int[]pq,来保存每个元素在items数组中的索引，pq数组需要堆有序，也就是说，pq[1]对应的数据元素items[pq[1]]要小于等于pq[2]和pq[3]对应的数据元素items[pq[2]]和items[pq[3]]。



  - 步骤三
    - 通过步骤二的分析，我们可以发现，其实我们通过上浮和下沉做堆调整的时候，其实调整的是pq数组。如果需要对items中的元素进行修改，比如让items[0]="H",那么很显然，我们需要对pq中的数据做堆调整，而且是调整pq[9]中元素的位置。但现在就会遇到一个问题，我们修改的是items数组中0索引处的值，如何才能快速的知道需要挑中pq[9]中元素的位置呢?
    - 最直观的想法就是遍历pq数组，拿出每一个元素和0做比较，如果当前元素是0，那么调整该索引处的元素即可，但是效率很低。
    - 我们可以另外增加一个数组，int[] qp,用来存储pq的逆序。例如：在pq数组中：pq[1]=6;
    - 那么在qp数组中，把6作为索引，1作为值，结果是：qp[6]=1;
    - 当有了pq数组后，如果我们修改items[0]="H"，那么就可以先通过索引0，在qp数组中找到qp的索引：qp[0]=9,
    - 那么直接调整pq[9]即可。

  - API设计
    - 类名：IndexMinPriorityQueue
    - 构造方法
      - IndexMinPriorityQueue(int capacity)：创建容量为capacity的IndexMinPriorityQueue对象
    - 成员方法
      - 1.private boolean less(int i,int j)：判断堆中索引i处的元素是否小于索引j处的元素
      - 2.private void exch(int i,int j):交换堆中i索引和j索引处的值
      - 3.public int delMin():删除队列中最小的元素,并返回该元素关联的索引
      - 4.public void insert(int i,T t)：往队列中插入一个元素,并关联索引i
      - 5.private void swim(int k):使用上浮算法，使索引k处的元素能在堆中处于一个正确的位置
      - 6.private void sink(int k):使用下沉算法，使索引k处的元素能在堆中处于一个正确的位置
      - 7.public int size():获取队列中元素的个数
      - 8.public boolean isEmpty():判断队列是否为空
      - 9.public boolean contains(int k):判断k对应的元素是否存在
      - 10.public void changeItem(int i, T t):把与索引i关联的元素修改为为t
      - 11.public int minIndex():最小元素关联的索引
      - 12.public void delete(int i):删除索引i关联的元素
    - 成员变量
      - 1.private T[] imtes : 用来存储元素的数组
      - 2.private int[] pq:保存每个元素在items数组中的索引，pq数组需要堆有序
      - 3.private int [] qp:保存qp的逆序，pq的值作为索引，pq的索引作为值
      - 4.private int N：记录堆中元素的个数