

# 优先级队列(堆)

## 本节目标

- 掌握堆的概念及实现
- 掌握 PriorityQueue 的使用

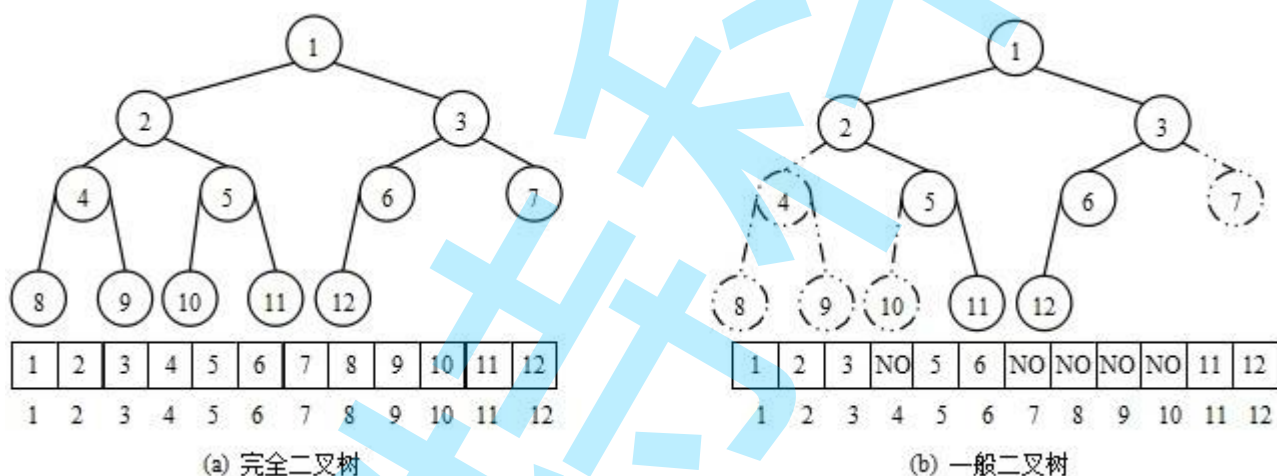
## 1. 二叉树的顺序存储

### 1.1 存储方式

使用数组保存二叉树结构，方式即将二叉树用**层序遍历**方式放入数组中。

一般只适合表示完全二叉树，因为非完全二叉树会有空间的浪费。

这种方式的主要用法就是堆的表示。



### 1.2 下标关系

已知双亲(parent)的下标，则：

左孩子(left)下标 =  $2 * \text{parent} + 1$ ;

右孩子(right)下标 =  $2 * \text{parent} + 2$ ;

已知孩子（不区分左右）(child)下标，则：

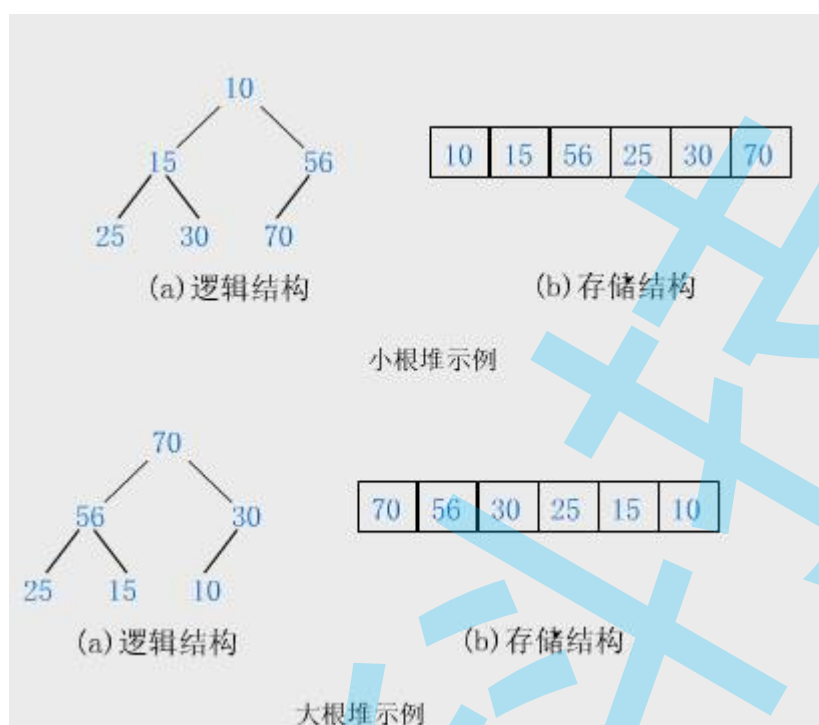
双亲(parent)下标 =  $(\text{child} - 1) / 2$ ;

## 2. 堆(heap)

### 2.1 概念

1. 堆逻辑上是一棵完全二叉树
2. 堆物理上是保存在数组中

3. 满足任意结点的值都大于其子树中结点的值，叫做大堆，或者大根堆，或者最大堆
4. 反之，则是小堆，或者小根堆，或者最小堆



5. 堆的基本作用是，快速找集合中的**最值**

## 2.2 操作-向下调整

**前提：**左右子树必须已经是一个堆，才能调整。

**说明：**

1. array 代表存储堆的数组
2. size 代表数组中被视为堆数据的个数
3. index 代表要调整位置的下标
4. left 代表 index 左孩子下标
5. right 代表 index 右孩子下标
6. min 代表 index 的最小值孩子的下标

**过程（以小堆为例）：**

1. index 如果已经是叶子结点，则整个调整过程结束
  1. 判断 index 位置有没有孩子
  2. 因为堆是完全二叉树，没有左孩子就一定没有右孩子，所以判断是否有左孩子
  3. 因为堆的存储结构是数组，所以判断是否有左孩子即判断左孩子下标是否越界，即  $left \geq size$  越界
2. 确定 left 或 right，谁是 index 的最小孩子 min
  1. 如果右孩子不存在，则  $min = left$
  2. 否则，比较  $array[left]$  和  $array[right]$  值得大小，选择小的为 min
3. 比较  $array[index]$  的值和  $array[min]$  的值，如果  $array[index] \leq array[min]$ ，则满足堆的性质，调整结束
4. 否则，交换  $array[index]$  和  $array[min]$  的值
5. 然后因为 min 位置的堆的性质可能被破坏，所以把 min 视作 index，向下重复以上过程

**图示：**

```
// 调整前
int[] array = { 27,15,19,18,28,34,65,49,25,37 };

// 调整后
int[] array = { 15,18,19,25,28,34,65,49,27,37 };
```



#### 时间复杂度分析:

最坏的情况即图示的情况，从根一路比较到叶子，比较的次数为完全二叉树的高度

即时间复杂度为  $O(\log(n))$

#### 代码:

```
public static void shiftDown(int[] array, int size, int index) {
    int left = 2 * index + 1;
    while (left < size) {
        int min = left;
        int right = 2 * index + 2;
        if (right < size) {
            if (array[right] < array[left]) {
                min = right;
            }
        }
        if (array[index] <= array[min]) {
            break;
        }
        int t = array[index];
        array[index] = array[min];
        array[min] = t;
    }
}
```

```

        index = min;
        left = 2 * index + 1;
    }
}

```

## 2.3 操作-建堆

下面我们给出一个数组，这个数组逻辑上可以看做一颗完全二叉树，但是还不是一个堆，现在我们通过算法，把它构建成一个堆。根节点左右子树不是堆，我们怎么调整呢？这里我们从倒数的第一个非叶子节点的子树开始调整，一直调整到根节点的树，就可以调整成堆。

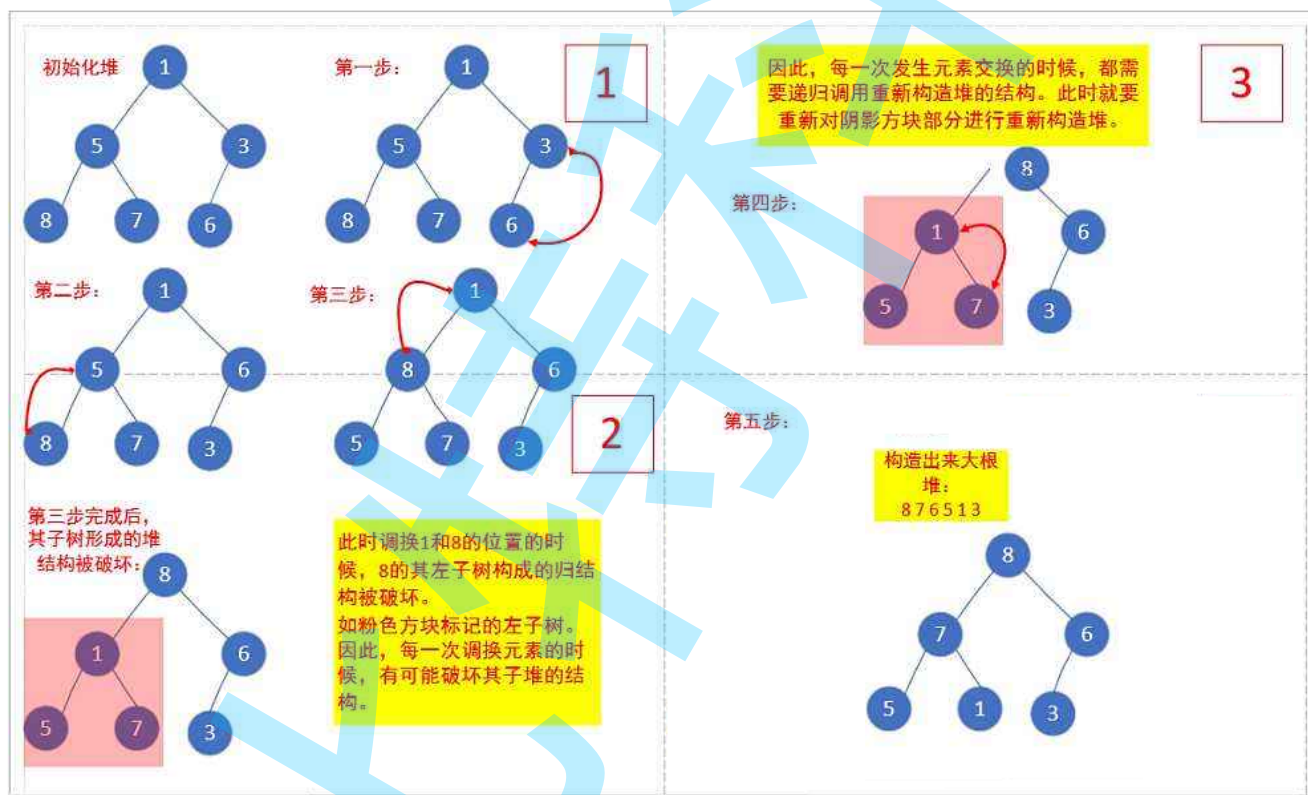
图示（以大堆为例）：

```

// 建堆前
int[] array = { 1,5,3,8,7,6 };

// 建堆后
int[] array = { 8,7,6,5,1,3 };

```



时间复杂度分析：

粗略估算，可以认为是在循环中执行向下调整，为  $O(n * \log(n))$

（了解）实际上是  $O(n)$

[堆排序中建堆过程时间复杂度 \$O\(n\)\$ 怎么来的？](#)

代码：

```
public static void createHeap(int[] array, int size) {  
    for (int i = (size - 1 - 1) / 2; i >= 0; i--) {  
        shiftDown(array, size, i);  
    }  
}
```

## 3. 堆的应用-优先级队列

### 3.1 概念

在很多应用中，我们通常需要根据优先级情况对待处理对象进行处理，比如首先处理优先级最高的对象，然后处理次高的对象。最简单的一个例子就是，在手机上玩游戏的时候，如果有来电，那么系统应该优先处理打进来的电话。

在这种情况下，我们的数据结构应该提供两个最基本的操作，一个是返回最高优先级对象，一个是添加新的对象。这种数据结构就是优先级队列(Priority Queue)

### 3.2 内部原理

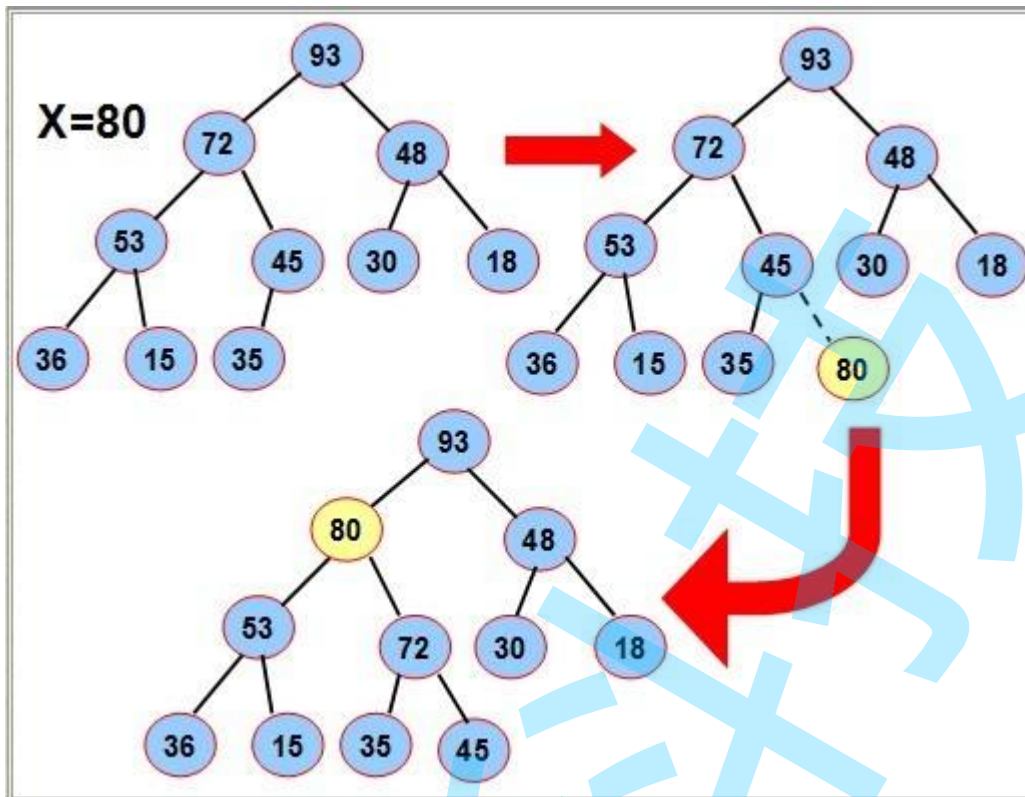
优先级队列的实现方式有很多，但最常见的是使用堆来构建。

### 3.3 操作-入队列

过程（以大堆为例）：

1. 首先按尾插方式放入数组
2. 比较其和其双亲的值的大小，如果双亲的值大，则满足堆的性质，插入结束
3. 否则，交换其和双亲位置的值，重新进行 2、3 步骤
4. 直到根结点

图示：



代码:

```
public static void shiftUp(int[] array, int index) {
    while (index > 0) {
        int parent = (index - 1) / 2;
        if (array[parent] >= array[index]) {
            break;
        }

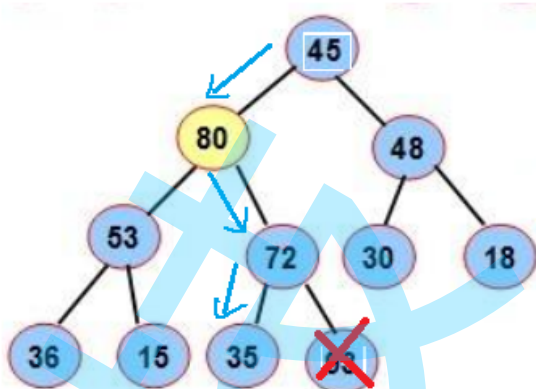
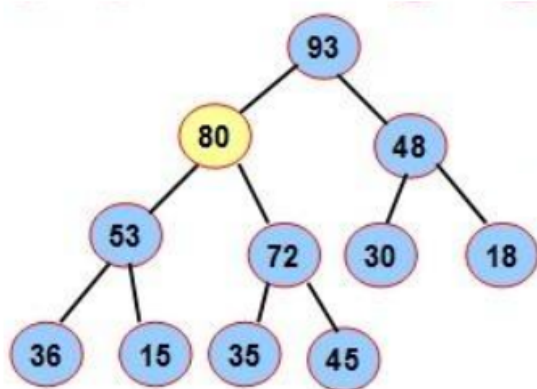
        int t = array[parent];
        array[parent] = array[index];
        array[index] = t;

        index = parent;
    }
}
```

### 3.4 操作-出队列 (优先级最高)

为了防止破坏堆的结构，删除时并不是直接将堆顶元素删除，而是用数组的最后一个元素替换堆顶元素，然后通过向下调整方式重新调整成堆





### 3.5 返回队首元素（优先级最高）

返回堆顶元素即可

### 3.6 代码

```
public class MyPriorityQueue {
    // 演示作用，不再考虑扩容部分的代码
    private int[] array = new int[100];
    private int size = 0;

    public void offer(int e) {
        array[size++] = e;
        shiftUp(array, size - 1);
    }

    public int poll() {
        int oldValue = array[0];
        array[0] = array[--size];
        shiftDown(array, size, 0);
        return oldValue;
    }

    public int peek() {
        return array[0];
    }
}
```

### 3.7 java 中的优先级队列

PriorityQueue implements Queue

错误处理	抛出异常	返回特殊值
入队列	add(e)	offer(e)
出队列	remove()	poll()
队首元素	element()	peek()

## 4. 堆的其他应用-TopK 问题

[拜托，面试别再问我TopK了!!!](#)

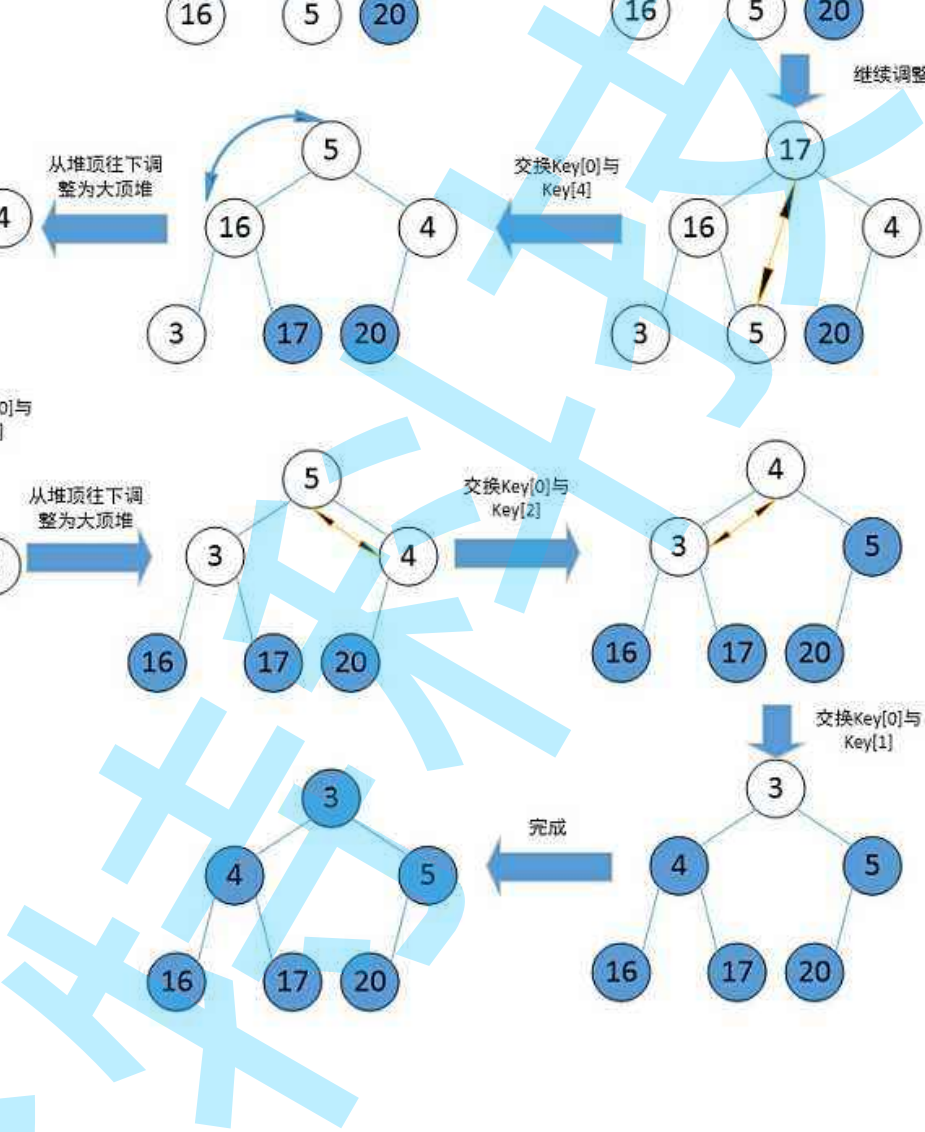
关键记得，找前 K 个最大的，要建 K 个大小的小堆

## 5. 面试题

1. [查找和最小的K对数字](#)

## 6. 堆的其他应用-堆排序





## 内容重点总结

- 堆的基本概念、操作及实现
- 优先级队列
- PriorityQueue 的使用
- TopK 问题
- 堆排序

## 课后作业

- 博客整理堆的相关知识
- 完成课堂代码