堆

2019年7月1日　　14:47

优先队列（**Priority Queue**）：特殊的“队列”，取出元素的顺序是依照元素的优先权（关键字）大小，而不是元素进入队列的先后顺序。

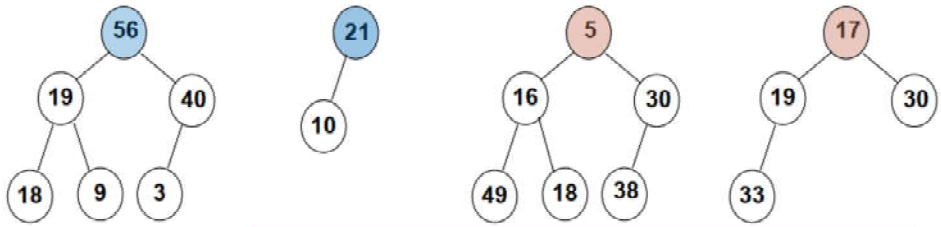
堆的两个特性

☞结构性：用数组表示的完全二叉树；

☞有序性：任一结点的关键字是其子树所有结点的最大值(或最小值)

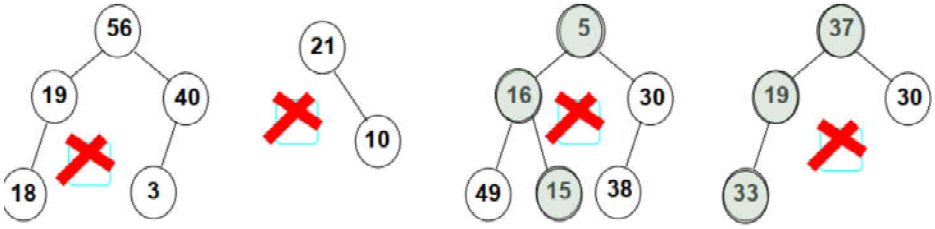
- “最大堆(MaxHeap)”，也称“大顶堆”：最大值
- “最小堆(MinHeap)”，也称“小顶堆”：最小值

【例】最大堆和最小堆



注意：从根结点到任意结点路径上结点序列的有序性！

【例】不是堆



堆的抽象数据类型描述

类型名称：最大堆（**MaxHeap**）

数据对象集：完全二叉树，每个结点的元素值不小于其子结点的元素值

操作集：最大堆**H** ∈ **MaxHeap**，元素**item** ∈ **ElementType**，主要操作有：

- **MaxHeap Create(int MaxSize)**：创建一个空的最大堆。
- **Boolean IsFull(MaxHeap H)**：判断最大堆**H**是否已满。
- **Insert(MaxHeap H, ElementType item)**：将元素**item**插入最大堆**H**。
- **Boolean IsEmpty(MaxHeap H)**：判断最大堆**H**是否为空。
- **ElementType DeleteMax(MaxHeap H)**：返回**H**中最大元素(高优先级)。

堆的数据结构

```
typedef struct HeapStruct *MaxHeap;
struct HeapStruct {
ElementType *Elements; /* 存储堆元素的数组 */
```

int Size; **int Capacity;**	/* 堆的当前元素个数 */
/* 堆的最大容量	*/

```
};
```

最大堆的创建

```
MaxHeap Create( int MaxSize )
{ /* 创建容量为MaxSize的空的最大堆 */
MaxHeap H = malloc( sizeof( struct HeapStruct ) );
H->Elements = malloc( (MaxSize+1) * sizeof(ElementType));
H->Size = 0;
H->Capacity = MaxSize;
H->Elements[0] = MaxData;
/* 定义“哨兵”为大于堆中所有可能元素的值，便于以后更快操作 */
return H;
}
```

把MaxData换成小于堆中所有元素的MinData，同样适用于
创建最小堆。
在线性时间复杂度下建立最大堆。
（1）将*N*个元素按输入顺序存入，先满足完全二叉树的结构特性
（2）调整各结点位置，以满足最大堆的有序特性。

堆的从插入调整

建最大堆，调整思路：从最后一个非叶子节点开始 从下向上从右到左的调整

```
void Insert( MaxHeap H, ElementType item )
{ /* 将元素item 插入最大堆H， 其中H->Elements[0]已经定义为哨兵 */
int i;
if ( IsFull(H) ) {
printf("最大堆已满");
return;
} i
= ++H->Size; /* i指向插入后堆中的最后一个元素的位置 */
for ( ; H->Elements[i/2] < item; i/=2 )
H->Elements[i] = H->Elements[i/2]; /* 向下过滤结点 */
H->Elements[i] = item; /* 将item 插入 */
}
```

H->Element[0] 是哨兵元素，它不小于堆中的最大元素，
控制顺环结束。
***T* (*N*) = O (log *N*)**

堆的删除

用最后一个元素替换根节点

```
ElementType DeleteMax( MaxHeap H )
{ /* 从最大堆H中取出键值为最大的元素， 并删除一个结点 */
int Parent, Child;
ElementType MaxItem, temp;
if ( IsEmpty(H) ) {
printf("最大堆已为空");
return;
}
MaxItem = H->Elements[1]; /* 取出根结点最大值 */
/* 用最大堆中最后一个元素从根结点开始向上过滤下层结点 */
temp = H->Elements[H->Size--];
for( Parent=1; Parent*2<=H->Size; Parent=Child ) {
Child = Parent * 2;
if( (Child!= H->Size) &&
(H->Elements[Child] < H->Elements[Child+1]) )
Child++; /* Child指向左右子结点的较大者 */
```

```
if( temp >= H->Elements[Child] ) break;
else /* 移动temp元素到下一层 */
H->Elements[Parent] = H->Elements[Child];
} H
->Elements[Parent] = temp;
return MaxItem;
}
```