5.1 堆(heap)

什么是堆

- 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。

问题：如何组织优先队列?

- 一般的数组、链表?
- 有序的数组或者链表?
- 二叉搜索树？ AVL树?

若采用数组或链表实现优先队列 这4种方式总有令人不满的复杂度，考虑树

- 数组：

  插入 — 元素总是插入尾部 ~ $\Theta(1)$

  删除 — 查找最大（或最小）关键字 ~ $\Theta(n)$

  从数组中删去需要移动元素 ~ $O(n)$

- 链表:

  插入 — 元素总是插入链表的头部 ~ $\Theta(1)$

  删除 — 查找最大（或最小）关键字 ~ $\Theta(n)$

  删去结点 ~ $\Theta(1)$

- 有序数组:

  插入 — 找到合适的位置 ~ $O(n)$ 或 $O(\log_2 n)$

  移动元素并插入 ~ $O(n)$

  删除 — 删去最后一个元素 ~ $\Theta(1)$

- 有序链表:

  插入 — 找到合适的位置 ~ $O(n)$

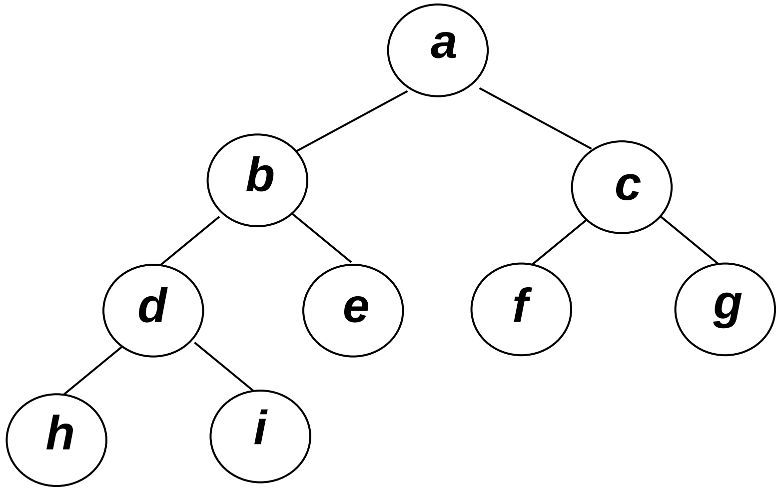
  插入元素 ~ $\Theta(1)$

  删除 — 删除首元素或最后元素 ~ $\Theta(1)$

✍ 是否可以采用二叉树存储结构？

- 二叉搜索树？
- 如果采用二叉树结构，应更关注**插入**还是**删除**？
  - 树结点顺序怎么安排？
  - 树结构怎样？

优先队列的完全二叉树表示



	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*		
0	1	2	3	4	5	6	7	8	9	10	11

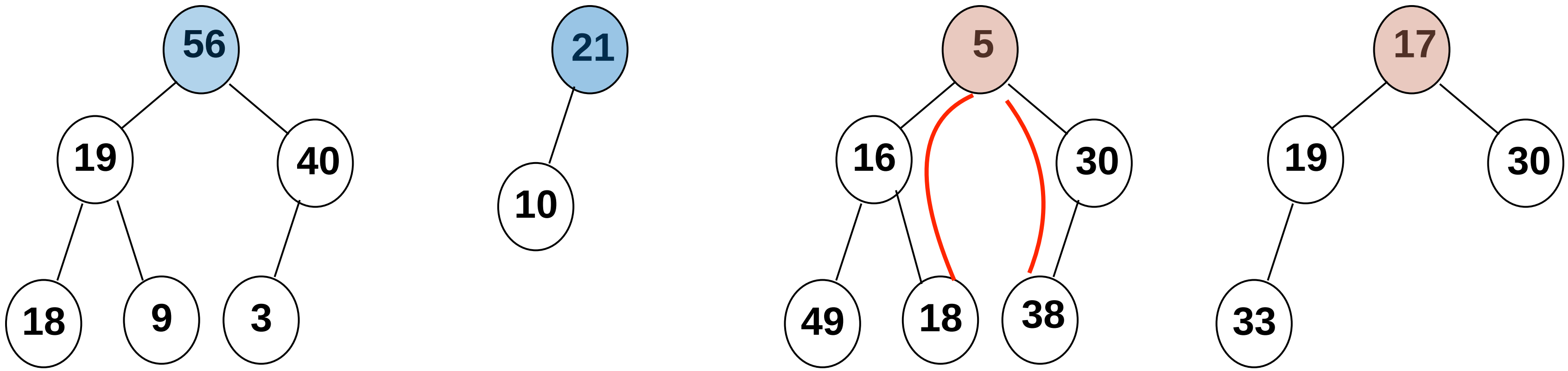
- 堆的两个特性

☞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)

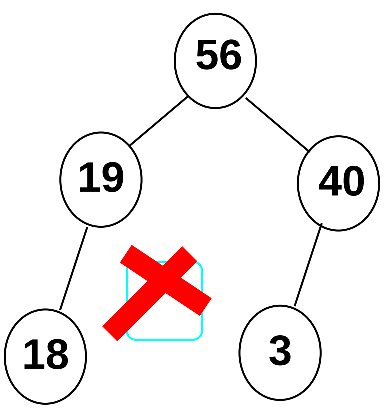
- ❑ “**最大堆(MaxHeap)**”,也称“大顶堆”：最大值
- ❑ “**最小堆(MinHeap)**”,也称“小顶堆”：最小值

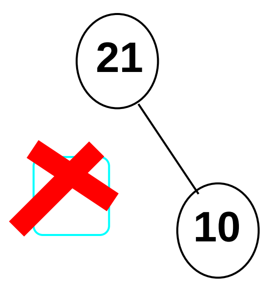
【例】最大堆和最小堆



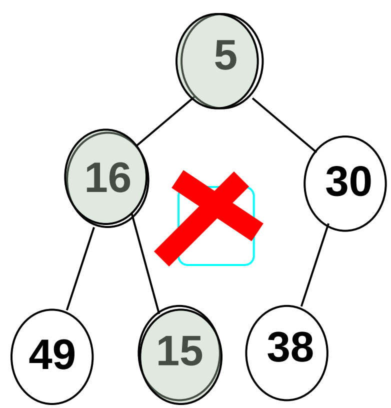
注意：从根结点到任意结点路径上结点序列的有序性！

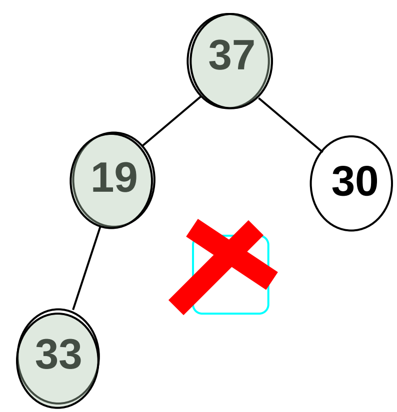
【例】不是堆





不是完全二叉树
（左子结点不能为空）





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)。

最大堆的操作

☞ 最大堆的**创建**

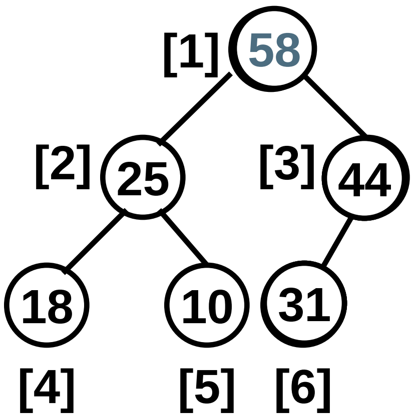
```
typedef struct HeapStruct *MaxHeap;
struct HeapStruct {
        ElementType *Elements; /* 存储堆元素的 */
        int Size;              /* 堆的当前元素个数
        int Capacity;          /* 堆的最大容量
};
```

把MaxData换成
小于堆中所有元素的
MinData，同样适用于
创建**最小堆**。

```
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;
}
```

因为下标从1开始

☞ 最大堆的插入



Case 1 : new_item = 20   20 < 31 ✓

Case 2 : new_item = 35   35 > 31   35 < 44 ✓

Case 3 : new_item = 58   58 > 31   58 > 44   58 < MaxData ✓

❖ 算法：将新增结点插入到从其父结点到根结点的有序序列中

```c
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 */
}
```

父结点

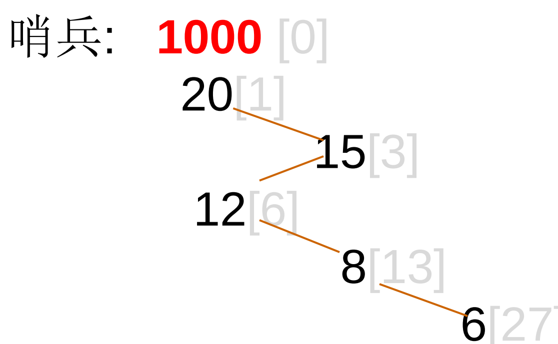
比交换数据要快

❖ 算法：将新增结点插入到从其父结点到根结点的有序序列中

```
void Insert( MaxHeap H, ElementType item )
{ /* 将元素item 插入最大堆H，其中H->Elem... 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 */
}
```

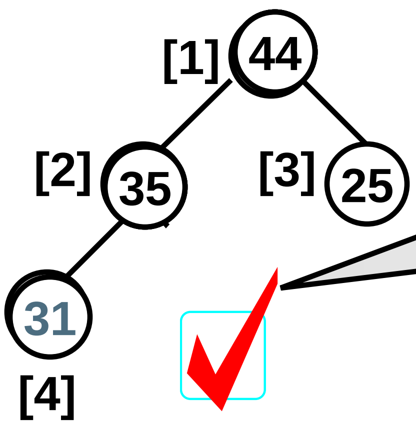
H->Element[0] 是哨兵元素，
它不小于堆中的最大元素，
控制顺环结束。

$T(N) = O(\log N)$



☞ 最大堆的删除

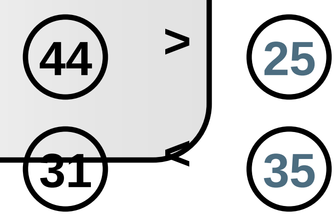
➢ 取出根结点（最大值）元素，同时删除堆的一个结点。



① 把 31 移至根

② 找出31的较大的孩子

为了保持完全二叉树的结构特性，移去的是该结点。



$$T(N) = O(\log N)$$

```
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 */ 最后要return
    /* 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 ) {
初始位置为根结点
        Child = Parent * 2;        判断是否有左儿子
        if( (Child!= H->Size) &&
            (H->Elements[Child] < H->Elements[Child+1]) )
           Child++;  /* Child指向左右子结点的较大者 */  右儿子
        if( temp >= H->Elements[Child] ) break;比左右儿子都大，退出
        else  /* 移动temp元素到下一层 */
           H->Elements[Parent] = H->Elements[Child];
    }
    H->Elements[Parent] = temp;
    return MaxItem;
}
```

☞ 最大堆的建立

建立最大堆：将已经存在的*N*个元素按最大堆的要求存放在一个一维数组中

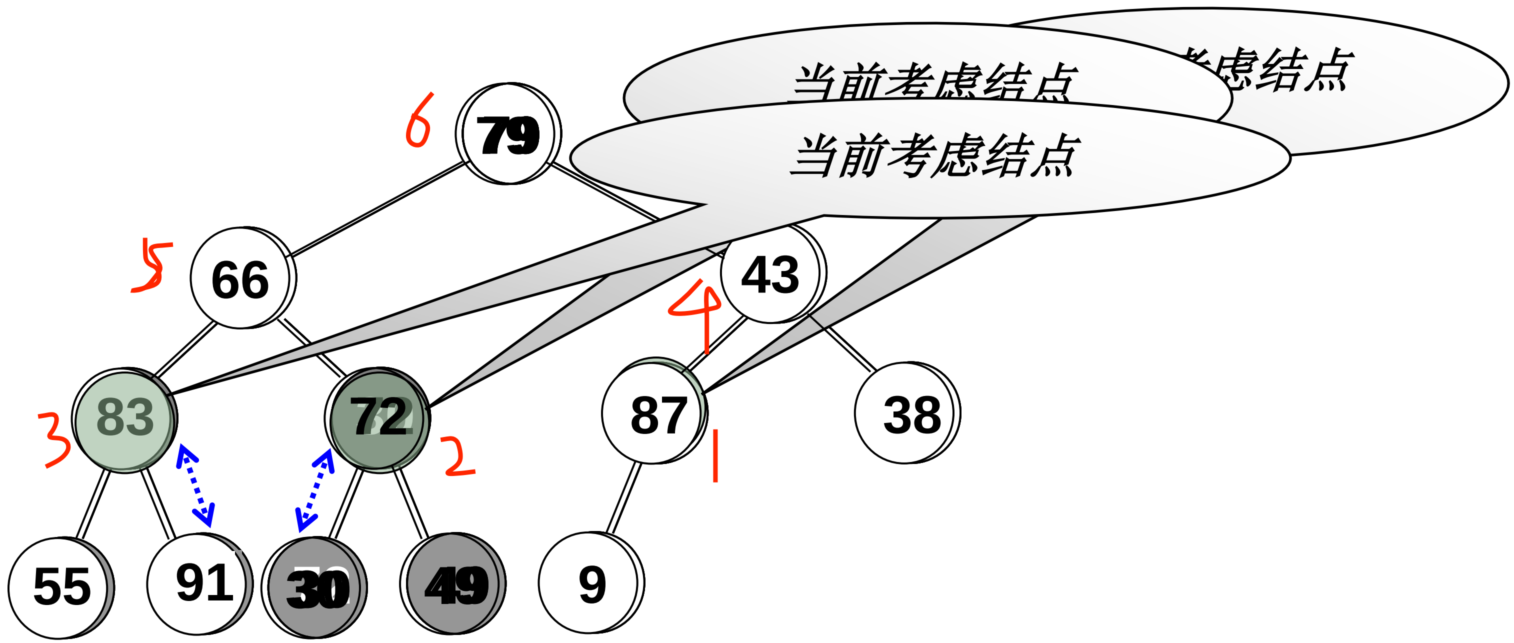
方法1：通过插入操作，将*N*个元素一个个相继插入到一个初始为空的堆中去，其时间代价最大为***O*(*N* log*N*)**。

时间复杂性不好

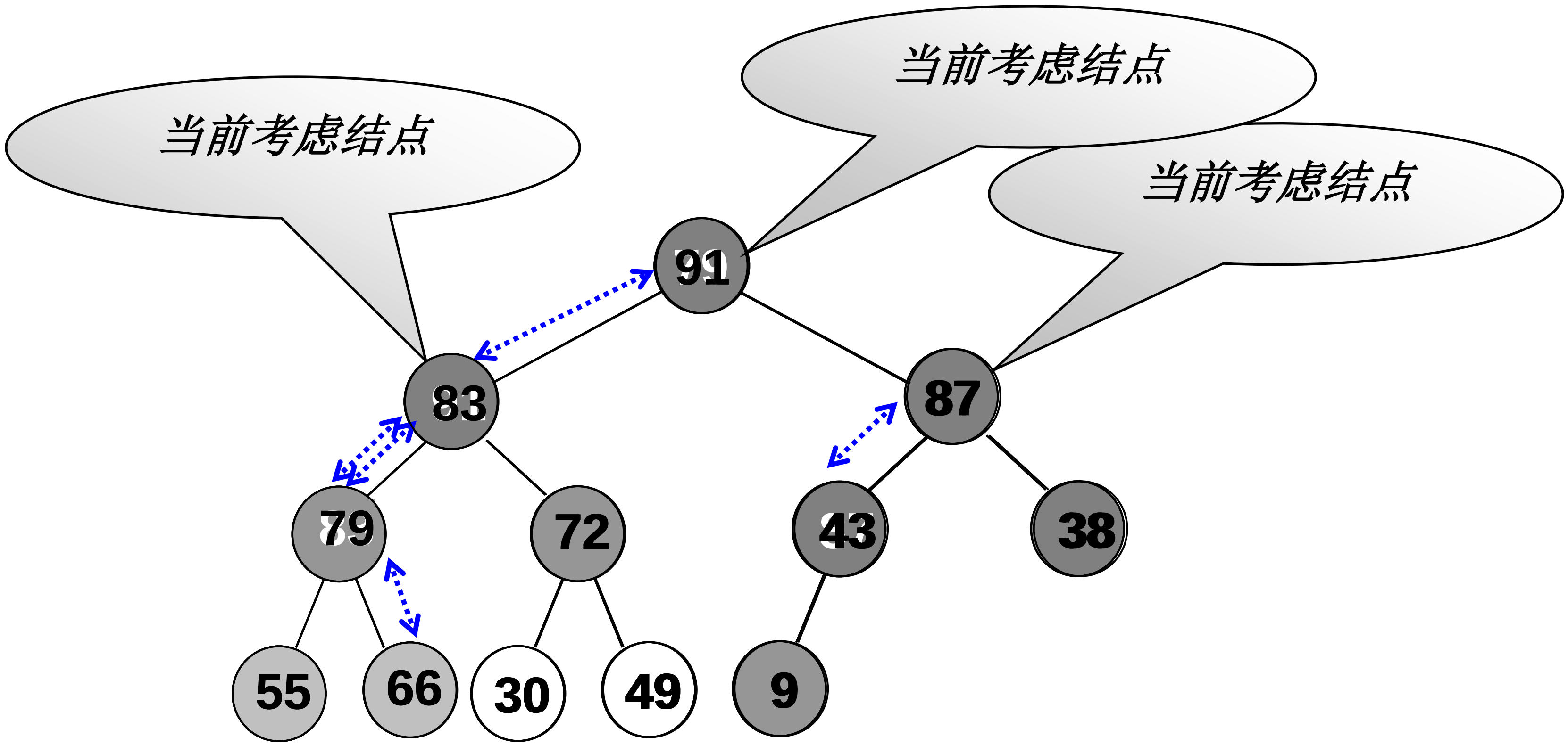
方法2：在线性时间复杂度下建立最大堆。

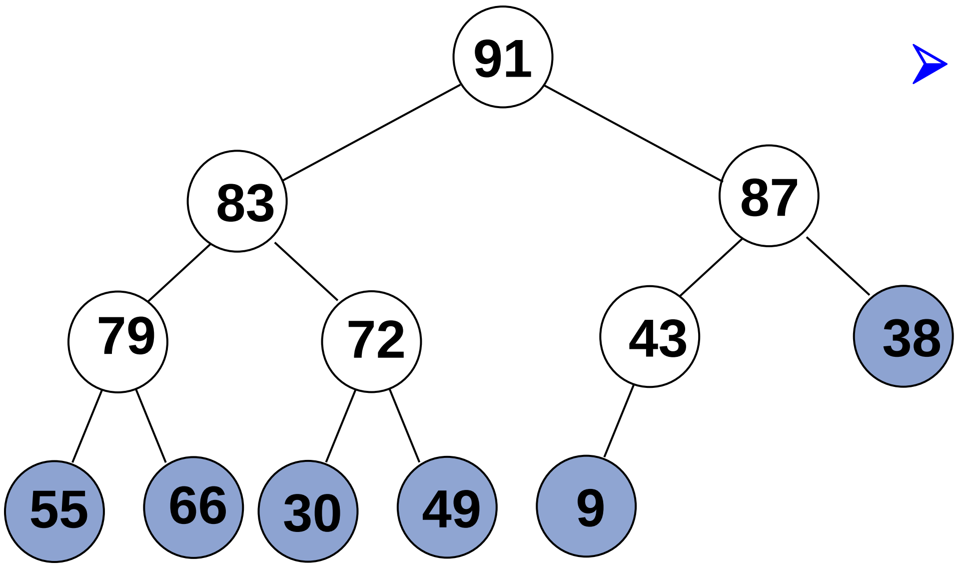
（1）将*N*个元素按输入顺序存入，先满足完全二叉树的结构特性

（2）调整各结点位置，以满足最大堆的有序特性。



调整的操作与删除操作相似，
删除根结点，用最后一个元素替换


当前考虑结点
当前考虑结点
当前考虑结点
91
83
87
79
72
43
38
55
66
30
49
9



- **线性时间复杂度T(n)=O(n)**

结点数	最多交换次数
n/4	1
n/8	2
n/16	3
……	
$n/2^k=1$	$\log_2 n-1$ (k-1)

$$T(n)=\frac{n}{4}+\frac{n}{8}\times 2+\frac{n}{16}\times 3+\ldots+\frac{n}{2^k}\times(k-1)$$

$$2T(n)=\frac{n}{2}+\frac{n}{4}\times 2+\frac{n}{8}\times 3+\ldots+\frac{n}{2^{k-1}}\times(k-1)$$

$$2T(n)-T(n)=\frac{n}{2}+\frac{n}{4}+\frac{n}{8}+\ldots+\frac{n}{2^{k-1}}-\frac{n}{2^k}\times(k-1)\le n-(\log_2 n-1)\le n$$