10.优先级队列

(b3) 完全二叉堆：删除

邓俊辉

deng@tsinghua.edu.cn

算法

- 最大元素始终在堆顶，故为删除之，只需...
- 摘除向量首元素，代之以末元素e

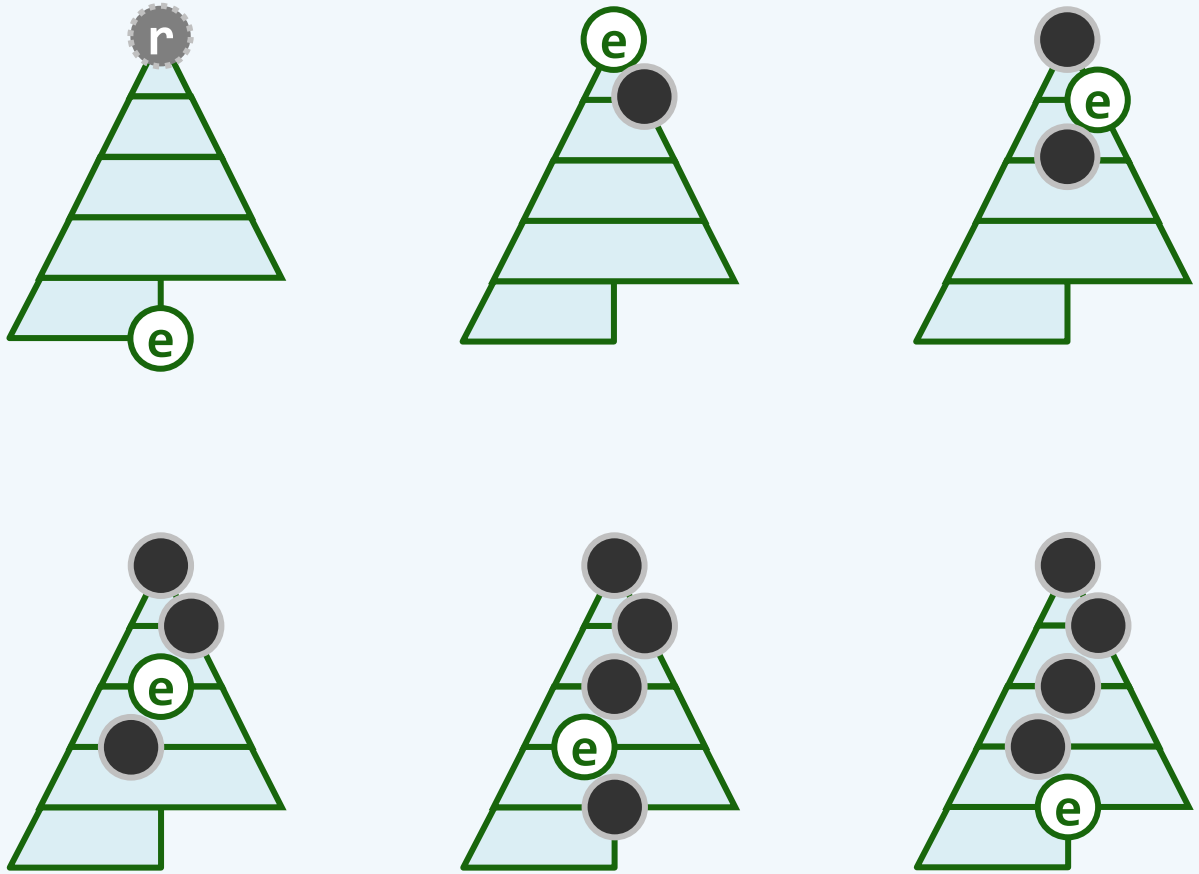
  //结构性保持；若堆序性依然保持则完成
- 否则 //即e与孩子们违背堆序性

  e与孩子中的大者换位

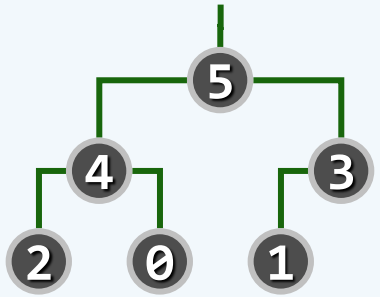
  //若堆序性因此恢复，则完成
- 否则 //即e与其（新的）孩子们...

  e再次与孩子中的大者换位
- 不断重复...直到

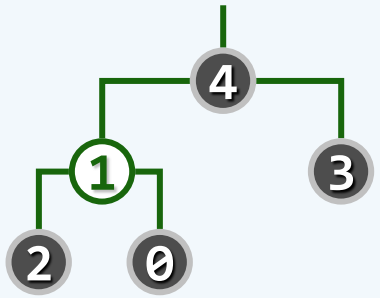
  e满足堆序性，或者已是叶子



实例

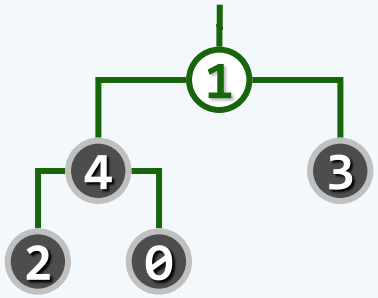


5 4 3 2 0 1



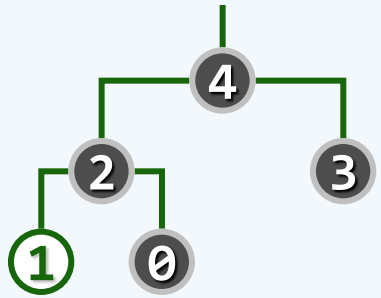
4 1 3 2 0

swap



1 4 3 2 0

swap



4 2 3 1 0

实现

```
❖ template <typename T> T PQ_ComplHeap<T>::delMax() { //删除
     T maxElem = _elem[0]; _elem[0] = _elem[ --_size ]; //摘除堆顶，代之以末词条
     percolateDown( _size, 0 ); //对新堆顶实施下滤
     return maxElem; //返回此前备份的最大词条
   }

❖ template <typename T> //对前n个词条中的第i个实施下滤，i < n
   Rank PQ_ComplHeap<T>::percolateDown( Rank n, Rank i ) {
     Rank j; //i及其（至多两个）孩子中，堪为父者
     while ( i != ( j = ProperParent( _elem, n, i ) ) ) //只要i非j，则
        { swap( _elem[i], _elem[j] ); i = j; } //换位，并继续考察i
     return i; //返回下滤抵达的位置（亦i亦j）
   }
```

效率

- 在下滤的过程中

  每经过一次交换

  e的高度都降低一层

- 故此，在每层至多需要一次交换

- 再次地，由于堆是完全树，故其高度为$O(\log n)$

- 结论：通过下滤，可在$O(\log n)$时间内

  删除堆顶节点，并

  整体重新调整为堆