10.优先级队列

（b2）完全二叉堆：插入与上滤

邓俊辉

deng@tsinghua.edu.cn

算法

- 为插入词条e，只需将e作为末元素接入向量

  //结构性自然保持

  //若堆序性也亦未破坏，则完成

- 否则 //只能是e与其父节点违反堆序性

  e与其父节点换位 //若堆序性因此恢复，则完成

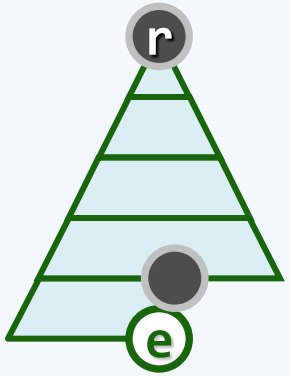
- 否则 //依然只可能是e与其（新的）父节点...

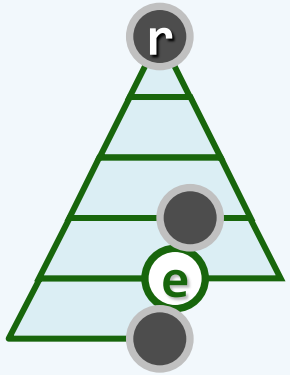
  e再与父节点换位

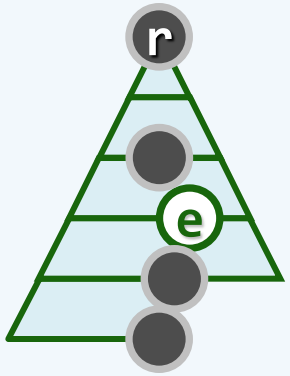
- 不断重复...直到

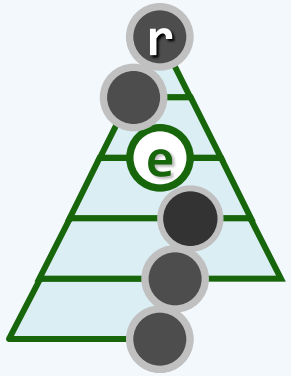
  e与其父亲满足堆序性，或者

  e到达堆顶（没有父亲）

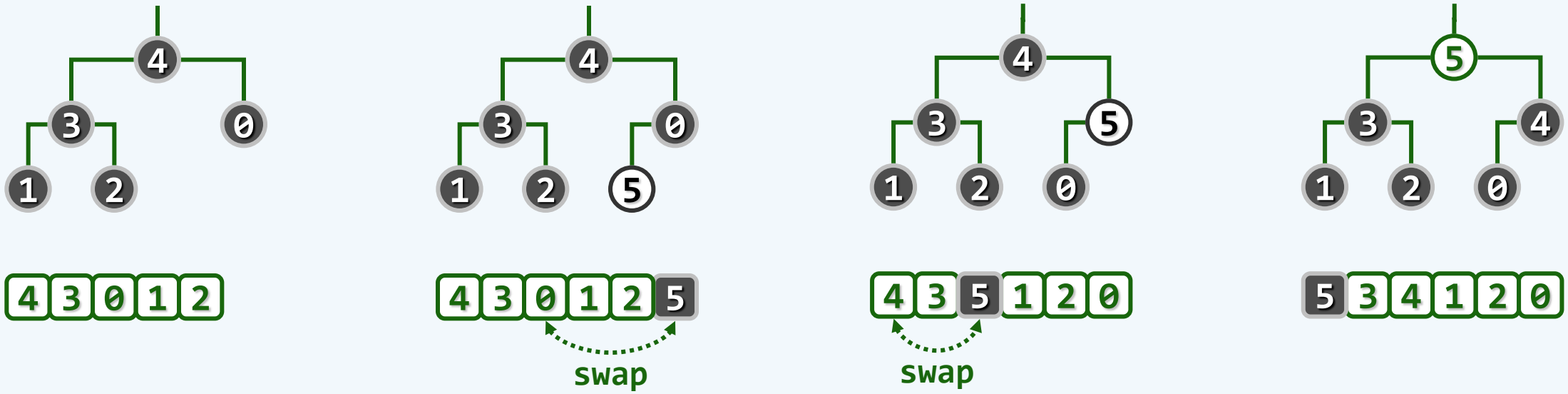








实例


4
3
0
1
2
4 3 0 1 2
4
3
0
1
2
5
4 3 0 1 2 5
swap
4
3
5
1
2
0
4 3 5 1 2 0
swap
5
3
4
1
2
0
5 3 4 1 2 0

实现

```
❖ template <typename T> void PQ_ComplHeap<T>::insert( T e ) //插入
    { Vector<T>::insert( e ); percolateUp( _size - 1 ); }

❖ template <typename T> //对第i个词条实施上滤，i < _size
  Rank PQ_ComplHeap<T>::percolateUp( Rank i ) {
    while ( ParentValid( i ) ) { //只要i有父亲（尚未抵达堆顶），则
      Rank j = Parent( i ); //将i之父记作j
      if ( lt( _elem[i], _elem[j] ) ) break; //一旦父子不再逆序，上滤旋即完成
      swap( _elem[i], _elem[j] ); i = j; //否则，交换父子位置，并上升一层
    } //while
    return i; //返回上滤最终抵达的位置
  }
```

效率

- e与父亲的交换，每次只需$O(1)$时间，且

  每经过一次**交换**，e都会**上升**一层

- 在插入新节点e的整个过程中，

  只有e的**祖先**们，才有可能需要与之交换

- 这里的堆以完全树实现，必平衡，故

  e的祖先至多$O(\log n)$个

- 结论：通过上滤，可在$O(\log n)$时间内

  插入一个新节点，并整体地重新调整为堆