

12B2 插入和上滤

#数据结构邓神

上滤

算法

 xuetangX.com
学堂在线

❖ 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



一旦堆序性被破坏了，就与其父亲节点交换，所以时间复杂度为 $O(\log N)$

❖ 否则 //只能是 e 与其**父**节点违反堆序性

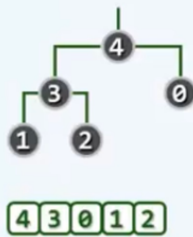
e 与其父节点换位 //若堆序性因此恢复，则完成

不一定一次就完全解决，但是也没有关系，最多也就是到ROOT节点

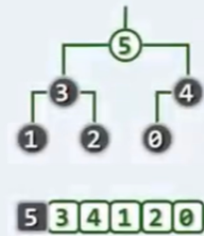
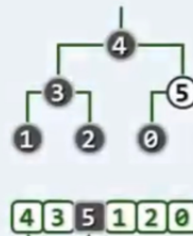
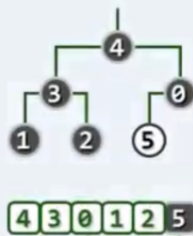
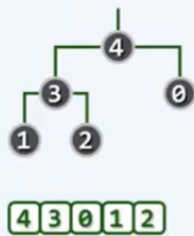
这一过程，亦即所谓的
上滤（percolate up）

实例

实例



这里所标注的数值
都是元素的优先级
而非其所对应的秩



实现

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
}
```

```
void insert(T e){
    _elem.push_back(e);
    percolateUp(_elem.size() - 1);
}
```

```

}

bool ParentVaild(const Rank& i){
    const Rank ROOT = 0;
    return ROOT != i;
}

Rank percolateUp(Rank i){
    while(ParentVaild(i)){
        Rank j = Parent(i);
        if(_elem[i] <= _elem[j]){
            break;
        }
        swap(_elem[i],_elem[j]);
        i = j;
    }
    return i;
}

```

效率

完全二叉树是理想平衡的二叉树

树高可以完美控制在 $\log n$

所有的迭代可以完美控制在 $\log n$

就渐进的意义而言 很好

但是就常系数的意义而言还是不行

但是 swap 函数需要三次交换操作

所以可能会多达 $3\log N$

我们可以对新插入的节点做一个备份，先去判断是否上移动，直到无序上移动在赋值进去

$3\log N \sim \log N + 2$

大小比较操作也可以改进