

12F3 合并算法

#数据结构邓神

LeftHeap模版类

LeftHeap

✧ template <typename T> //基于二叉树，以左式堆形式实现的优先级队列

⇒ class PQ_LeftHeap : public PQ<T>, public BinTree<T> {

public:

void insert(T); // (按比较器确定的优先级次序) 插入元素

T getMax() { return _root->data; } //取出优先级最高的元素

T delMax(); //删除优先级最高的元素

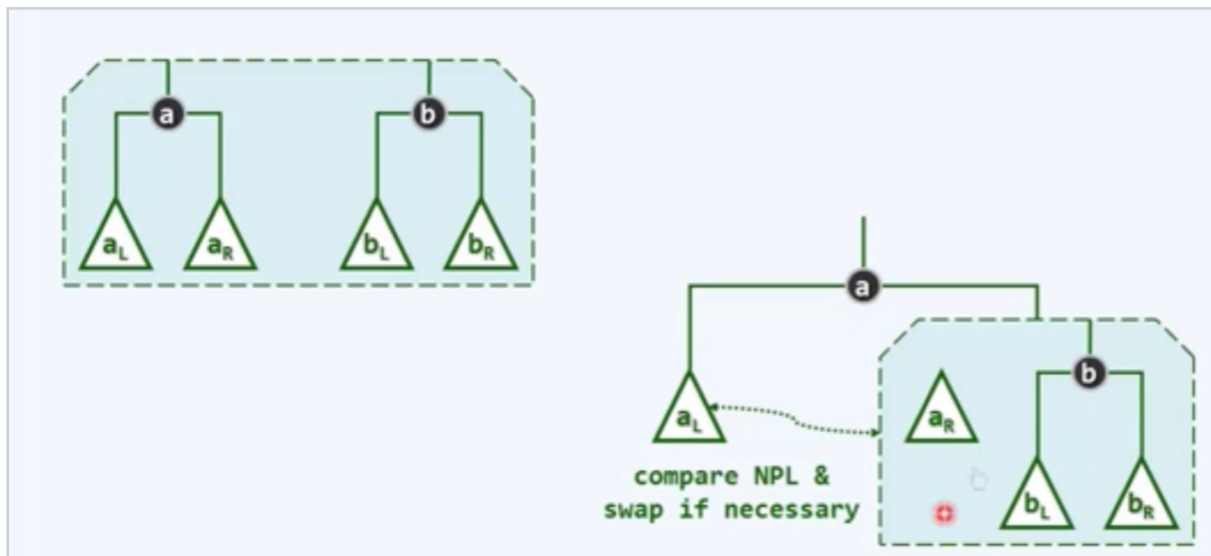
}; //主要接口，均基于统一的合并操作实现...

```
template <typename T> class PQ_LeftHeap : public PQ<T> , public BinTree <T> {
public:
    void insert(T);
    T getMax(){
        return _root->data;
    }
    T delMax();
};
```

左式堆不满足结构结构性
物理结构不再保持紧凑性

```
template <typename T>
static BinNodePosi(T) merge(BinNodePosi(T), BinNodePosi(T));
// 通过外部函数的形式给出具体算法
```

算法



在合并后如果NPL逆序就互换左右子树

实现

实现

```
template <typename T>
static BinNodePosi(T) merge( BinNodePosi(T) [a], BinNodePosi(T) [b] ) {
    if ( ! [a] ) return [b]; //递归基
    if ( ! [b] ) return [a]; //递归基
    if ( lt( [a]->data, [b]->data ) ) swap( [b], [a] ); //一般情况：首先确保b不大
    [a]->rc = merge( [a]->rc, [b] ); //将a的右子堆，与b合并
    [a]->parent = [a]; //并更新父子关系
    if ( ! [a]->lc || [a]->lc->npl < [a]->rc->npl ) //若有必要
        swap( [a]->lc, [a]->rc ); //交换a的左、右子堆，以确保右子堆的npl不大
    [a]->npl = [a]->rc ? [a]->rc->npl + 1 : 1; //更新a的npl
    return [a]; //返回合并后的堆顶
}
```

Handwritten notes in the image:

- $a \geq b$ (with a curved arrow pointing to the swap condition)
- Red circles around `merge` and `[a]->rc` in the recursive call.
- Red circles around `[a]` and `[b]` in the base cases.

Data Structures (Spring 2014), Tsinghua University

```
template <typename T> static BinNodePosi(T)
merge(BinNodePosi(T), BinNodePosi(T)) {
    if (! a) {
        return b;
    }
    if (! b) {
        return a;
    }
}
```

```
if ( a->data < b->data ){
  swap(b,a);
}
a->rc = merge(a->rc,b);
a->rc->parent = a;
if (! a->lc || a->lc->npl < a->rc->npl){
  swap(a->lc,a->rc);
}
a->npl = a->rc ? a->rc->npl + 1 : 1;
return a;
}
```