12-F3

优先级队列

左式堆：合并算法

邓俊辉

deng@tsinghua.edu.cn

左之左之，君子宜之；右之右之，君子有之

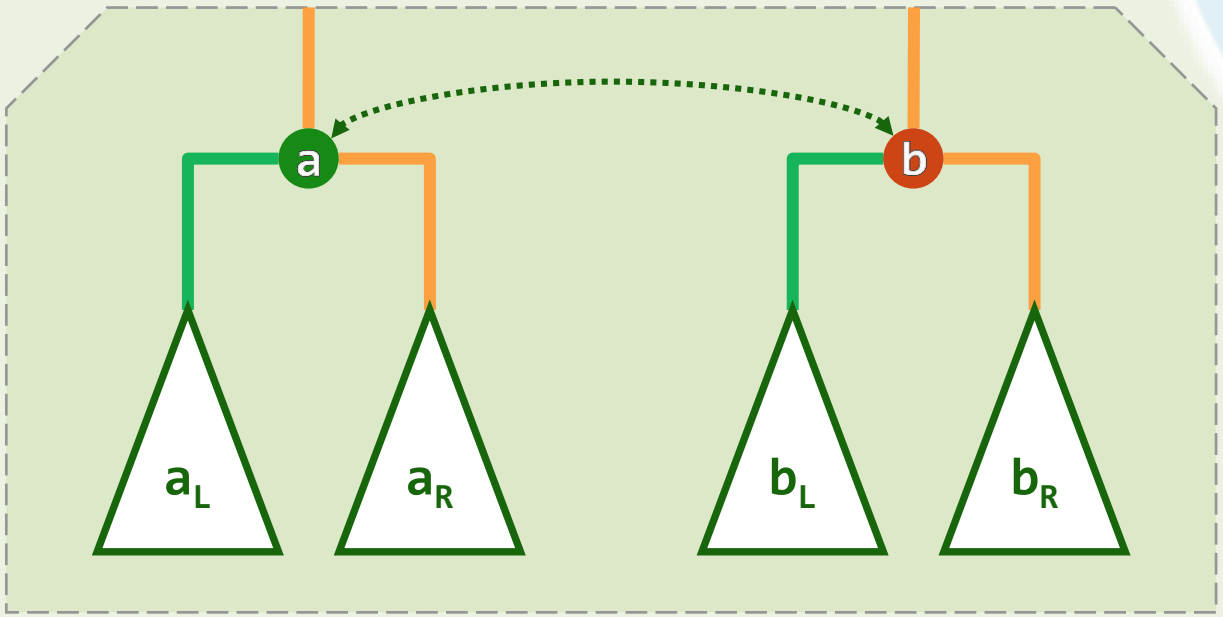
LeftHeap

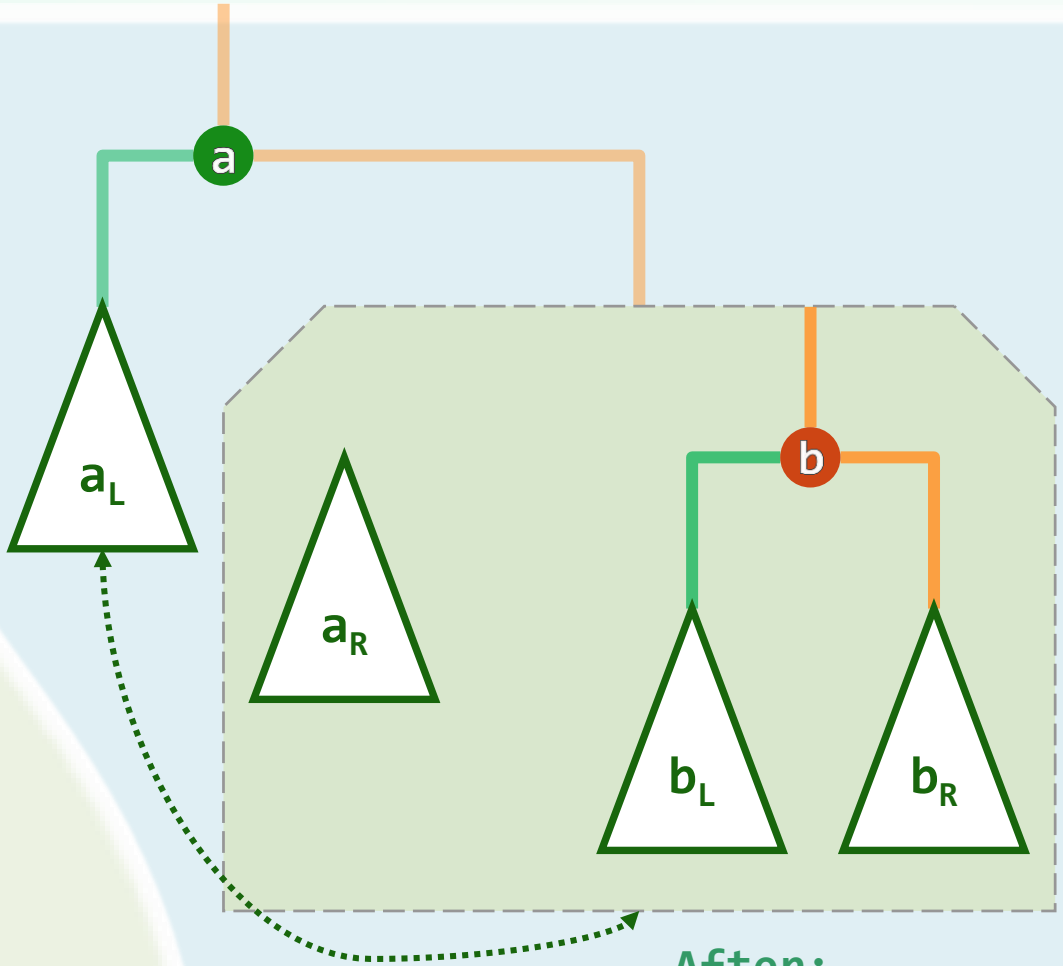
```
template <typename T> //基于二叉树，以左式堆形式实现的优先级队列
class PQ_LeftHeap : public PQ<T>, public BinTree<T> {
public:  T getMax() { return _root->data; }
         void insert(T); T delMax(); //均基于统一的合并操作实现...
         PQ_LeftHeap( PQ_LeftHeap & A, PQ_LeftHeap & B ) {
            _root = merge(A._root, B._root); _size = A._size + B._size;
            A._root = B._root = NULL; A._size = B._size = 0;
         }
};

template <typename T> BinNodePosi<T> merge(BinNodePosi<T>, BinNodePosi<T>);
```

递归：前处理 + 后处理

Before:
compare priority &
swap if necessary



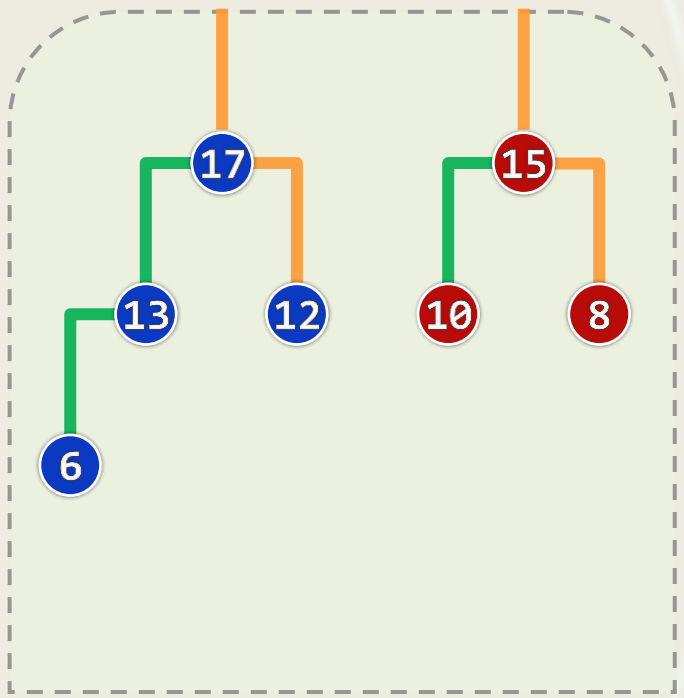


After:
compare NPL &
flip if necessary

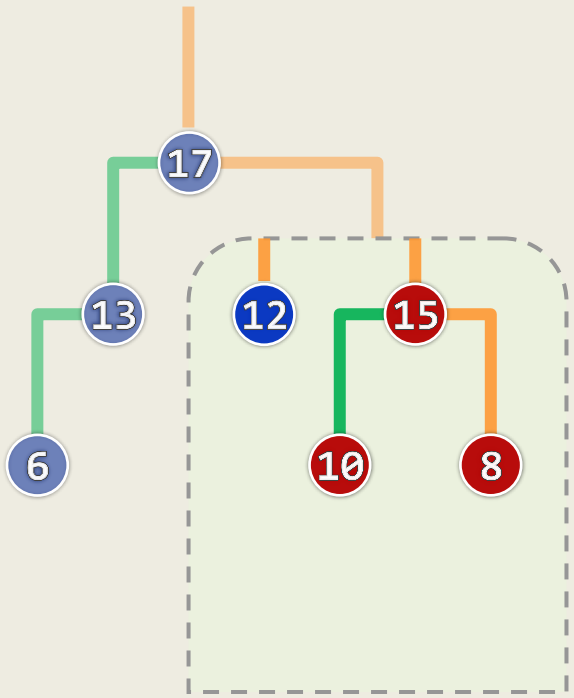
递归实现

```
template <typename T> BinNodePosi<T> merge( BinNodePosi<T> a, BinNodePosi<T> b ) {
   if ( !a ) return b; if ( !b ) return a; //递归基
   if ( lt( a->data, b->data ) ) swap( a, b ); //确保a>=b
   ( a->rc = merge( a->rc, b ) )->parent = a; //将a的右子堆，与b合并
   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，以确保左子堆的npl不小
   a->npl = a->rc ? 1 + a->rc->npl : 1; //更新a的npl
   return a; //返回合并后的堆顶
}
```

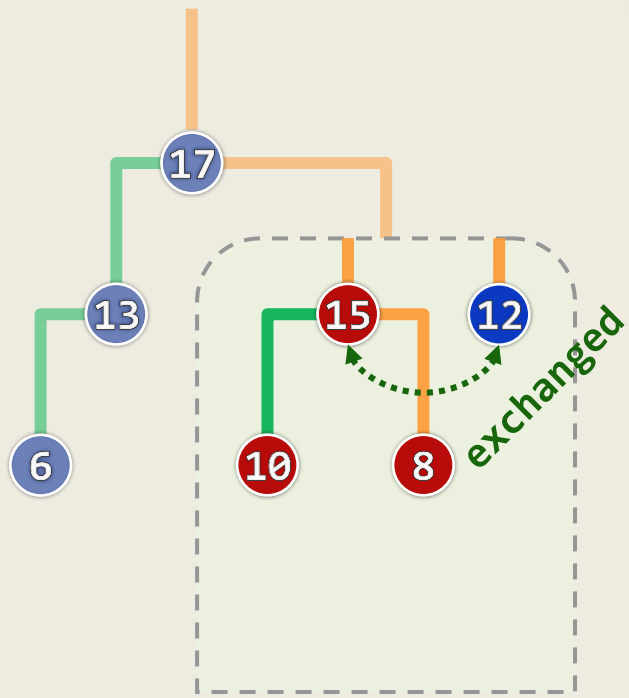
实例（1/5）



(a)

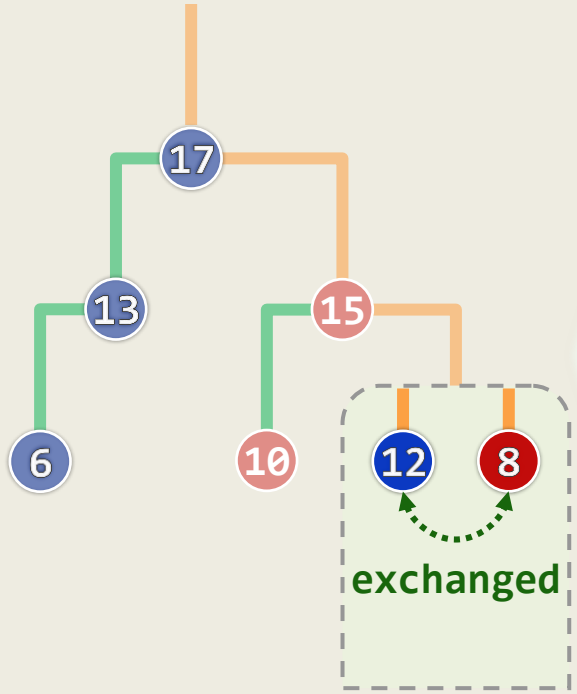


(b)

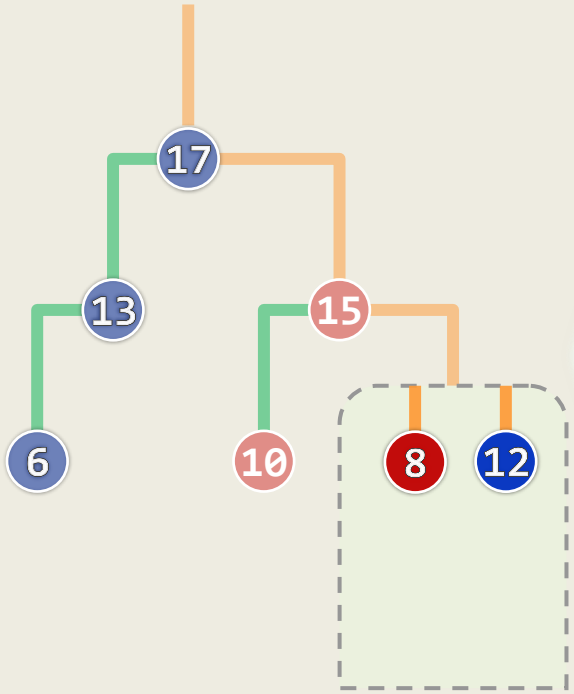


(c)

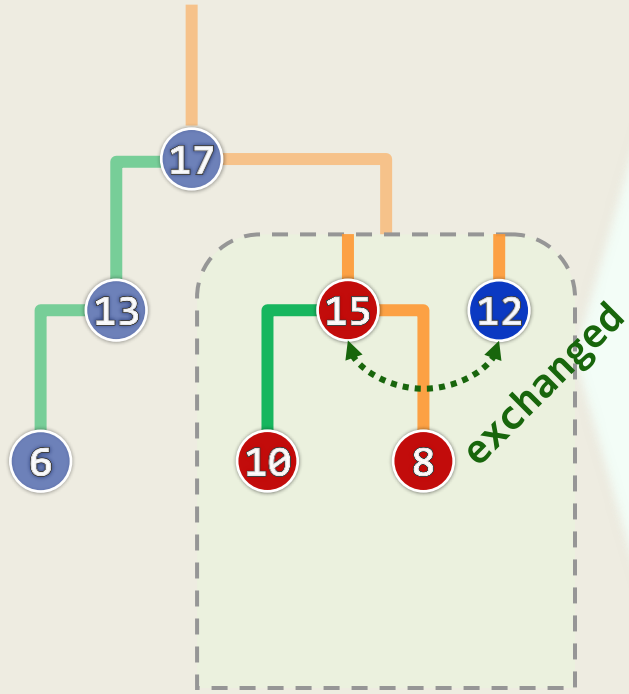
实例（2/5）


17
13
15
6
10
12
8
exchanged

(e)

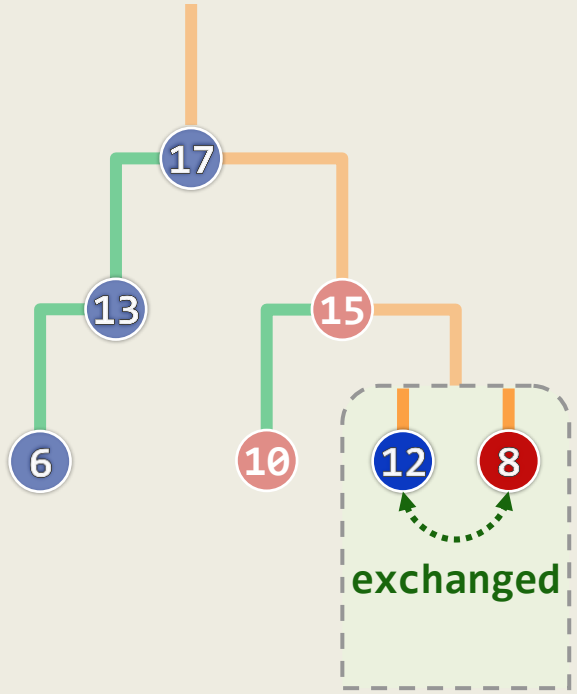

17
13
15
6
10
8
12

(d)

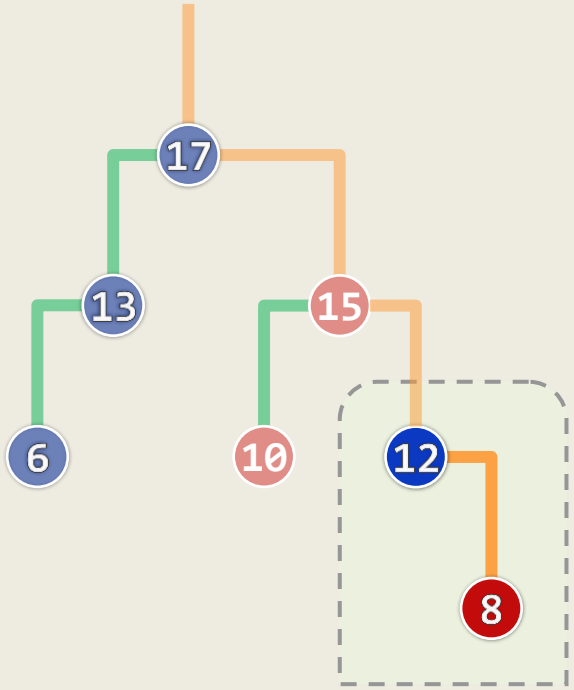

17
13
15
12
6
10
8
exchanged

(c)

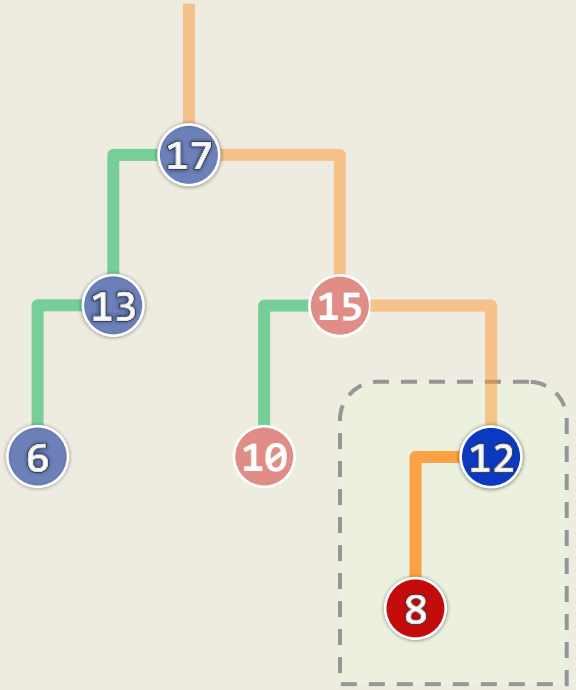
实例（3/5）



(e)

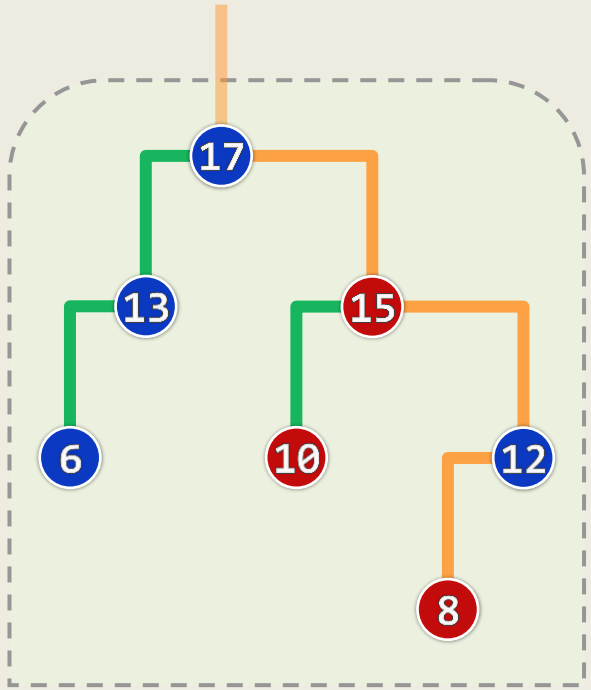


(f)

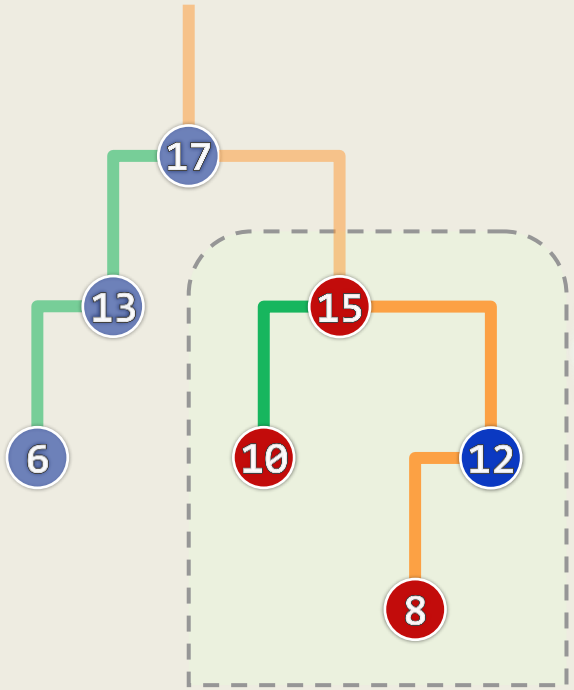


(g)

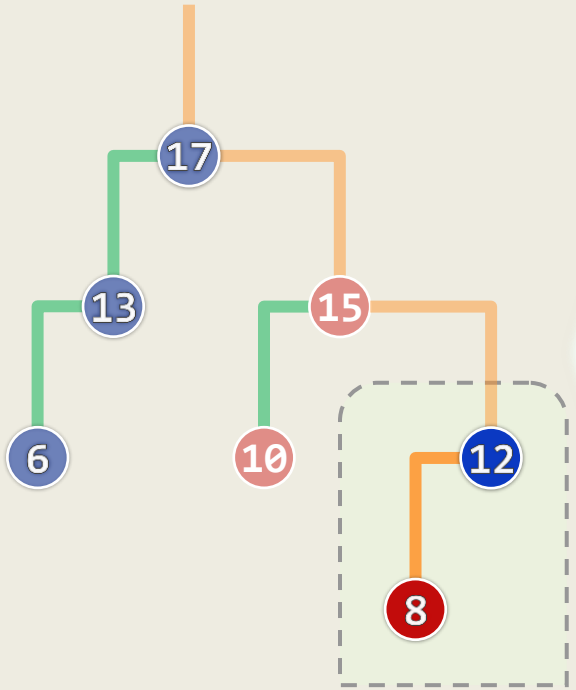
实例（4/5）



(i)

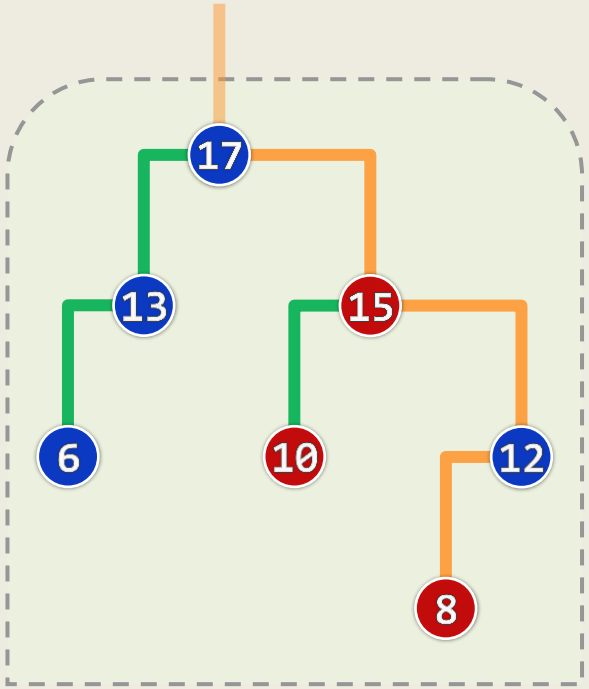


(h)

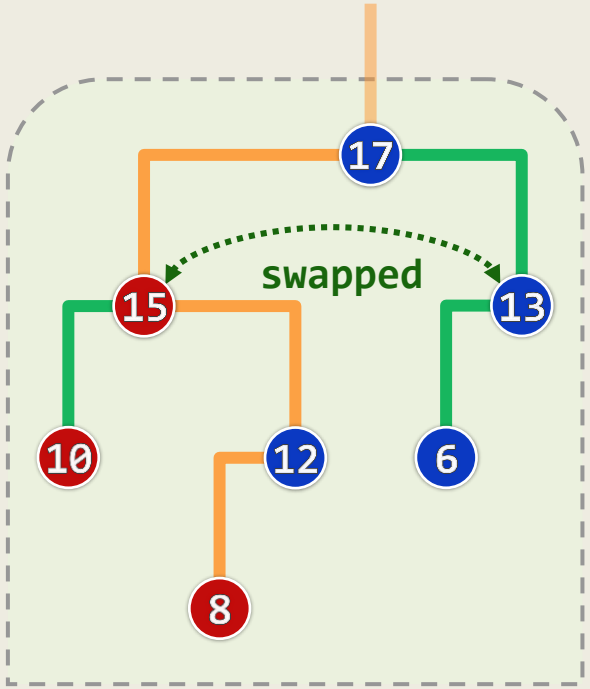


(g)

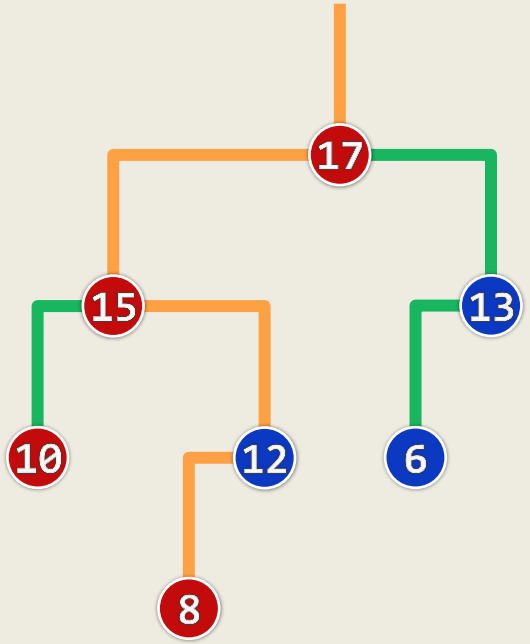
实例（5/5）



(i)



(j)



(k)

迭代实现

```
template <typename T> BinNodePosi<T> merge( BinNodePosi<T> a, BinNodePosi<T> b ) {
   if ( !a ) return b; if ( !b ) return a; //退化情况
   if ( lt( a->data, b->data ) ) swap( a, b ); //确保a>=b
   for ( ; a->rc; a = a->rc ) //沿右侧链做二路归并，直至堆a->rc先于b变空
      if ( lt( a->data, b->data ) ) { b->parent = a; swap( a->rc, b); } //接入b
   (a->rc = b)->parent = a; //直接接入b的残余部分（必然非空）
   for ( ; a; b = a, a = a->parent ) { //从a出发沿右侧链逐层回溯（b == a->rc）
      if ( !a->lc || a->lc->npl < a->rc->npl ) swap( a->lc, a->rc ); //确保npl合法
      a->npl = a->rc ? a->rc->npl + 1 : 1; //更新npl
   }
   return b; //返回合并后的堆顶
} //merge()
```