05-D

二叉树

二叉树实现

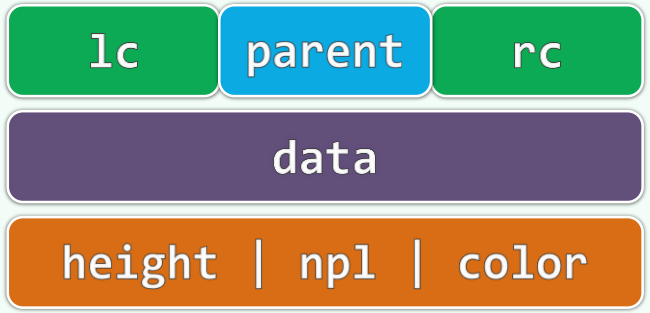
Anyone who loves his father or mother more than me is not worthy of me; anyone who loves his son or daughter more than me is not worthy of me.

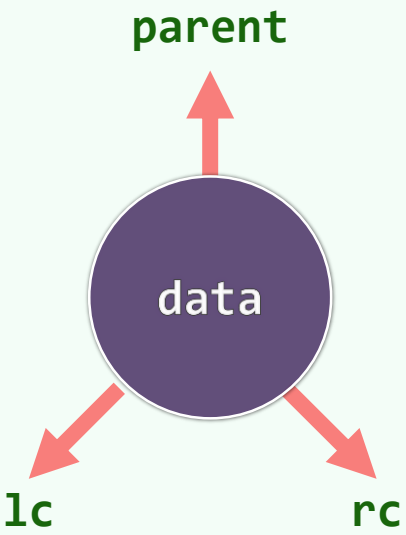
邓俊辉

deng@tsinghua.edu.cn

BinNode模板类

```
template <typename T> using BinNodePosi = BinNode<T>*; //节点位置
template <typename T> struct BinNode {
   BinNodePosi<T> parent, lc, rc; //父亲、孩子
   T data; Rank height, npl; Rank size(); //高度、npl、子树规模
   BinNodePosi<T> insertAsLC( T const & ); //作为左孩子插入新节点
   BinNodePosi<T> insertAsRC( T const & ); //作为右孩子插入新节点
   BinNodePosi<T> succ(); //（中序遍历意义下）当前节点的直接后继
   template <typename VST> void travLevel( VST & ); //层次遍历
   template <typename VST> void travPre( VST & ); //先序遍历
   template <typename VST> void travIn( VST & ); //中序遍历
   template <typename VST> void travPost( VST & ); //后序遍历
};
```

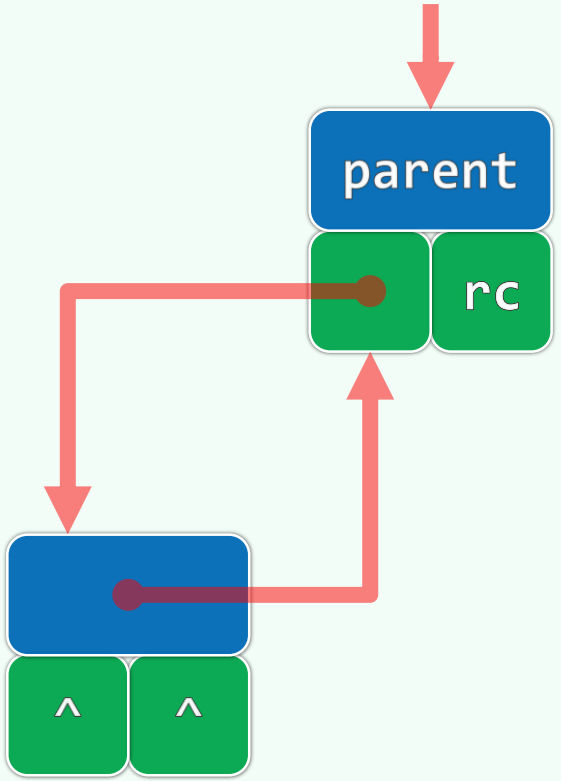




BinNode：引入新节点

```
template <typename T>

BinNodePosi<T> BinNode<T>::insertAsLC( T const & e )

   { return lc = new BinNode( e, this ); }
```

```
template <typename T>

BinNodePosi<T> BinNode<T>::insertAsRC( T const & e )

   { return rc = new BinNode( e, this ); }
```

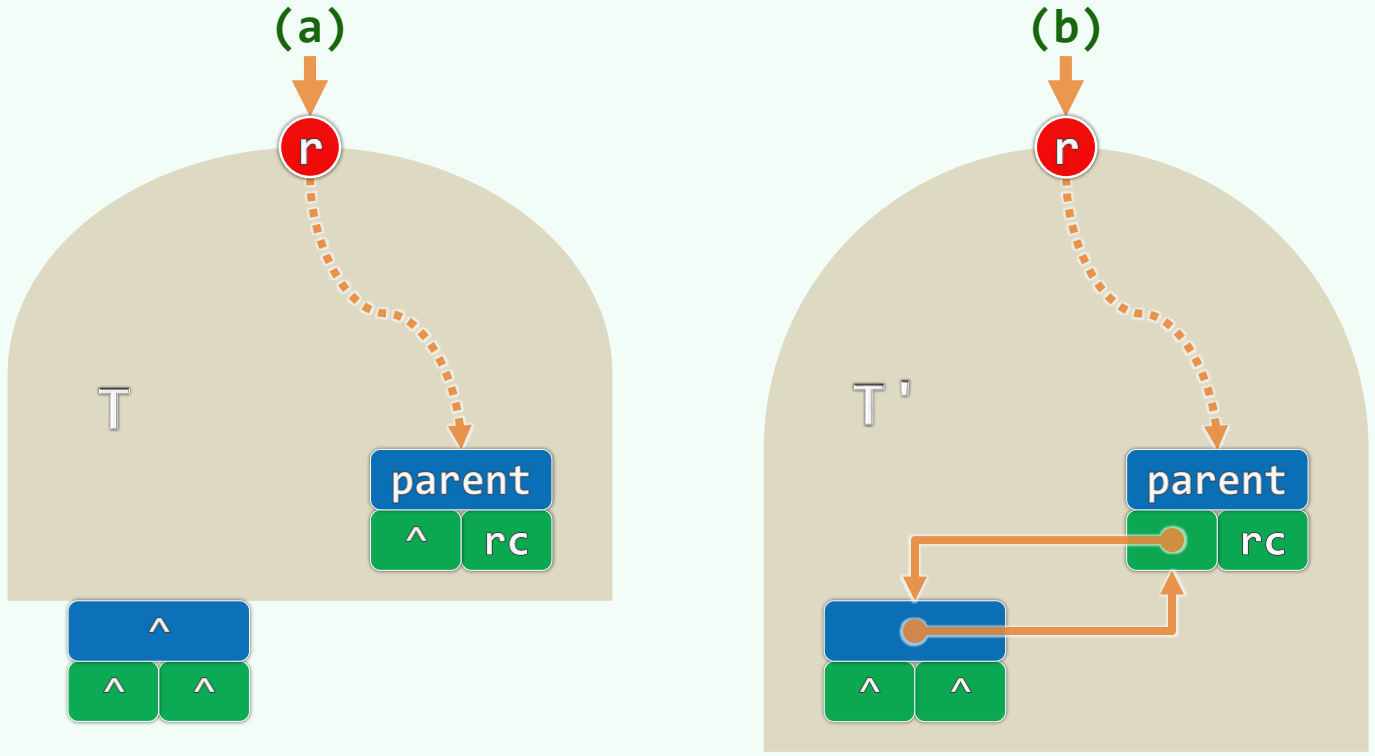


BinTree模板类

```
template <typename T> class BinTree {
protected:
   Rank _size; //规模
   BinNodePosi<T> _root; //根节点
   virtual Rank updateHeight( BinNodePosi<T> x ); //更新节点x的高度
   void updateHeightAbove( BinNodePosi<T> x ); //更新x及祖先的高度
public:
   Rank size() const { return _size; } //规模
   bool empty() const { return !_root; } //判空
   BinNodePosi<T> root() const { return _root; } //树根
   /* ... 子树接入、删除和分离接口；遍历接口 ... */
};
```

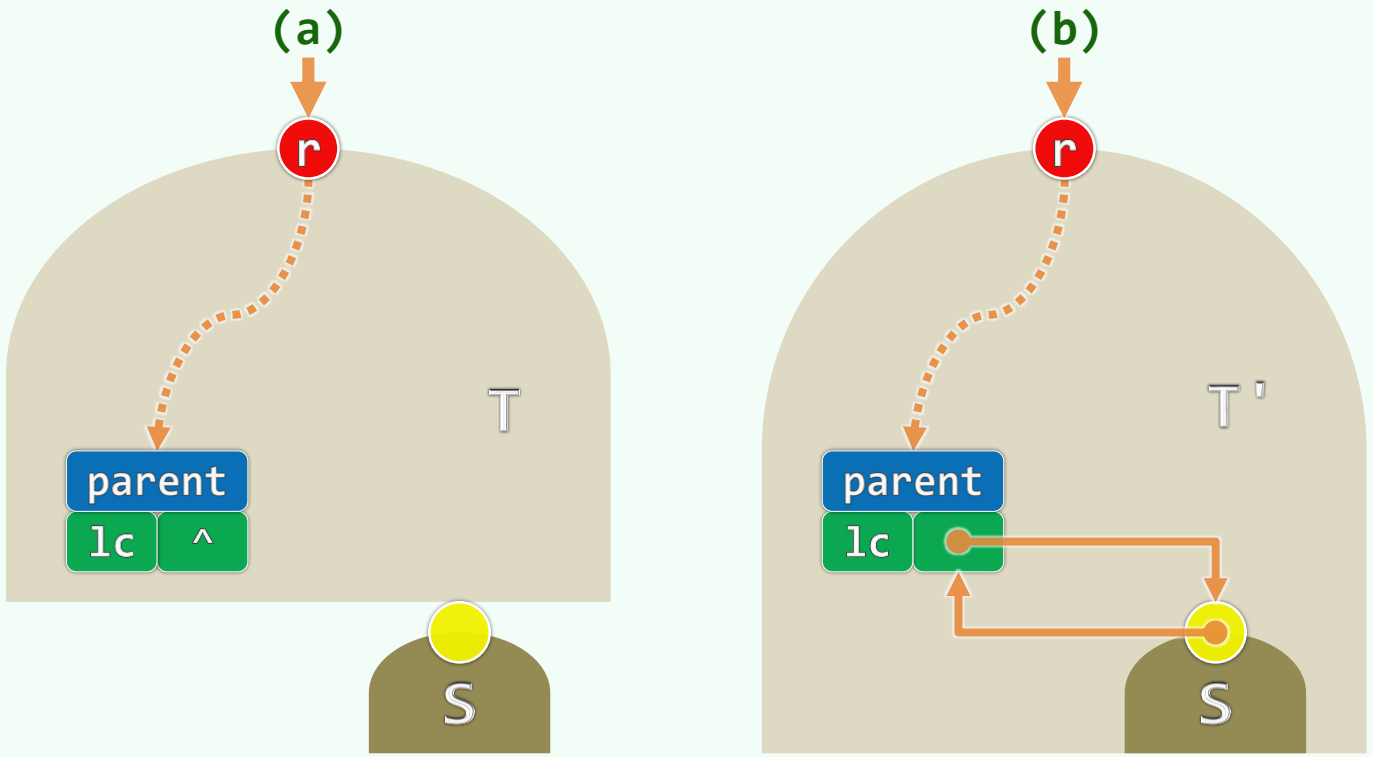
BinTree: 引入新节点

```
BinNodePosi<T> BinTree<T>::insert( BinNodePosi<T> x, T const & e );  //作为右孩子

BinNodePosi<T> BinTree<T>::insert( T const & e, BinNodePosi<T> x ) { //作为左孩子

   _size++;

   x->insertAsLC( e );

   updateHeightAbove( x );

   return x->lc;

}
```



BinTree: 接入子树

```
BinNodePosi<T> BinTree<T>::attach( BinTree<T>* &S, BinNodePosi<T> x );  //接入左子树

BinNodePosi<T> BinTree<T>::attach( BinNodePosi<T> x, BinTree<T>* &S ) { //接入右子树
   if ( x->rc = S->_root )
      x->rc->parent = x;
   _size += S->_size;
   updateHeightAbove(x);
   S->_root = NULL; S->_size = 0;
   release(S); S = NULL;
   return x;
}
```



BinTree：更新高度

```
#define stature(p) ( (int) ( (p) ? (p)->height : -1 ) ) //空树高度-1，以上递推
```

```
template <typename T> //勤奋策略：及时更新节点x高度，具体规则因树不同而异
Rank BinTree<T>::updateHeight( BinNodePosi<T> x ) //此处采用常规二叉树规则，O(1)
   { return  x->height  =  1 + max( stature( x->lc ), stature( x->rc ) ); }
```

```
template <typename T> //更新节点及其历代祖先的高度
void BinTree<T>::updateHeightAbove( BinNodePosi<T> x ) //O( n = depth(x) )
   { while (x) { updateHeight(x); x = x->parent; } } //可优化
```

BinTree: 分离子树

```
template <typename T> BinTree<T>* BinTree<T>::secede( BinNodePosi<T> x ) {
   FromParentTo( * x ) = NULL; updateHeightAbove( x->parent );
// 以上与BinTree<T>::remove()一致；以下还需对分离出来的子树重新封装
   BinTree<T> * S = new BinTree<T>; //创建空树
   S->_root = x; x->parent = NULL; //新树以x为根
   S->_size = x->size(); _size -= S->_size; //更新规模
   return S; //返回封装后的子树
}
```