05-D

二叉树

二叉树实现

Two roads diverged in a yellow wood

And sorry I could not travel both

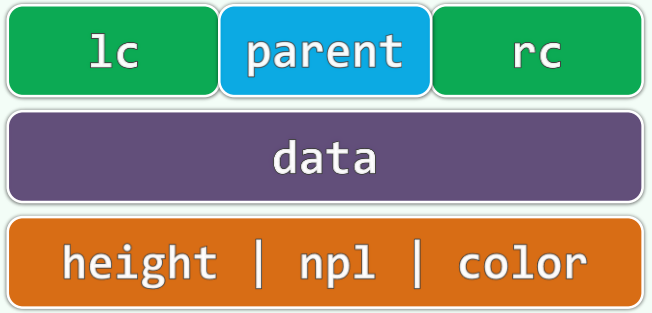
Anyone who loves his father or mother more than me is not worthy of me; anyone who loves his son or daughter more than me is not worthy of me.

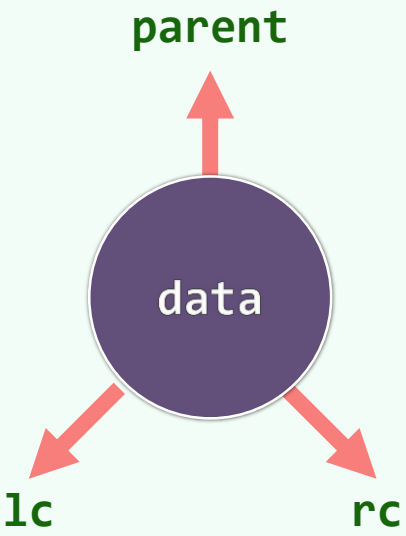
邓俊辉

deng@tsinghua.edu.cn

BinNode模板类

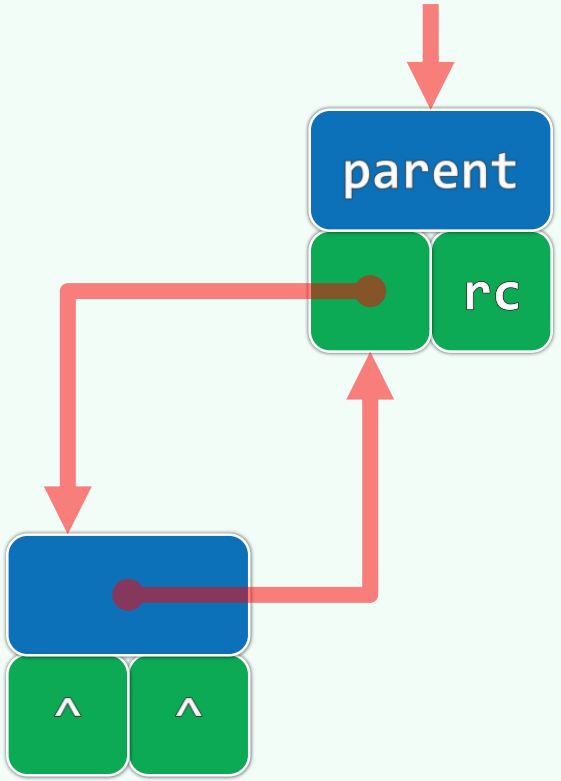
```
template <typename T> using BinNodePosi = BinNode<T>*; //节点位置

template <typename T> struct BinNode {
   BinNodePosi<T> parent, lc, rc; //父亲、孩子
   T data; Rank height, npl; RBColor color; //数据、高度、npl、颜色

   Rank size(); Rank updateHeight(); void updateHeightAbove(); //更新规模、高度

   BinNodePosi<T> insertLc( T const & ); //插入左孩子
   BinNodePosi<T> insertRc( T const & ); //插入右孩子

   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：插入新节点

```
template <typename T>
BinNodePosi<T> BinNode<T>::insertLc( T const & e )
   { return lc = new BinNode<T>( e, this ); }
```




```
template <typename T>
BinNodePosi<T> BinNode<T>::insertRc( T const & e )
   { return rc = new BinNode<T>( e, this ); }
```

BinNode：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 BinNode<T>::updateHeight() //此处采用常规二叉树规则，O(1)
   { return  height  =  1 + max( stature( lc ), stature( rc ) ); }
```

```
template <typename T> //更新节点及其历代祖先的高度
void BinNode<T>::updateHeightAbove() //更新当前节点及其祖先的高度，O( n = depth(x) )
   { for ( BinNodePosi<T> x = this; x; x = x->parent ) x->updateHeight(); } //可优化
```

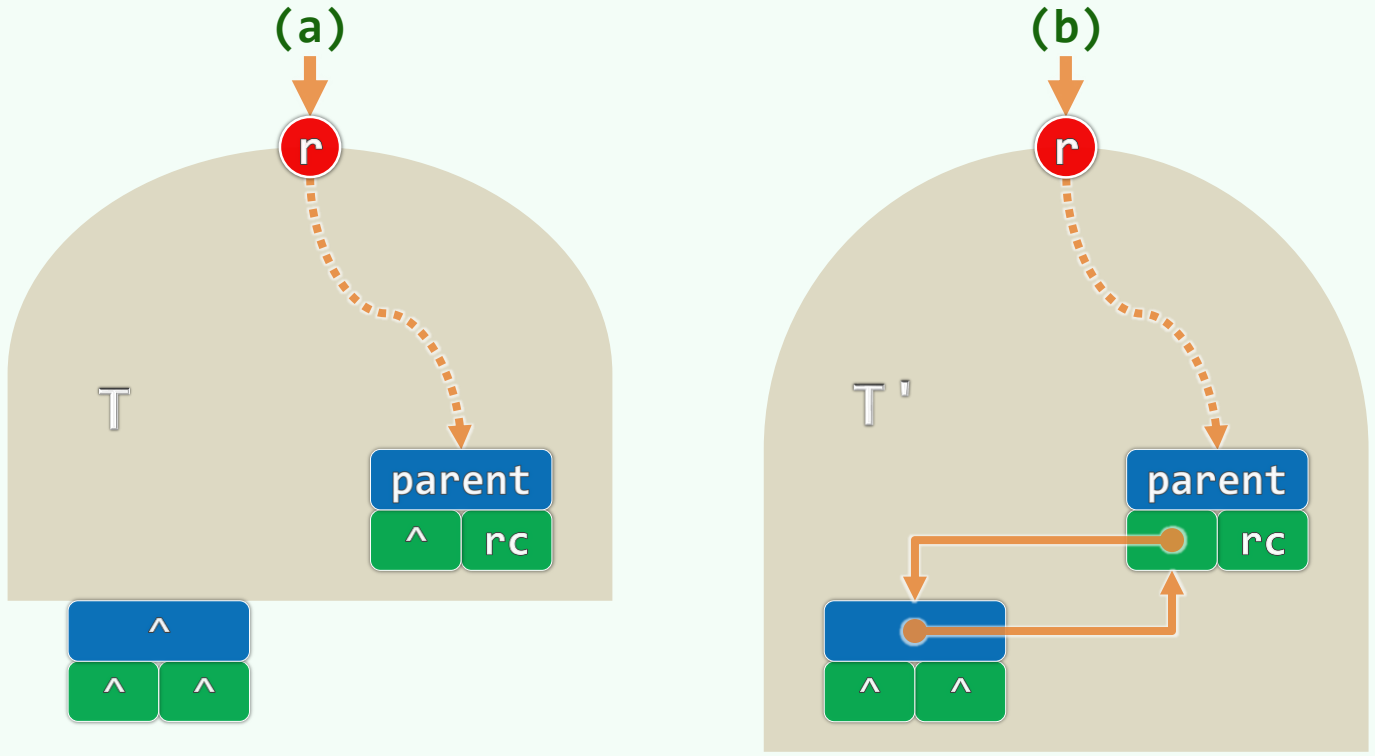
BinTree模板类

```
template <typename T> class BinTree {
protected: Rank _size; BinNodePosi<T> _root;
public:  Rank size() const { return _size; }; bool empty() const { return !_root; }
         BinNodePosi<T> root() const { return _root; }
         BinNodePosi<T> insert( T const& ); //插入根节点
         BinNodePosi<T> insert( T const&, BinNodePosi<T> ); //插入左孩子
         BinNodePosi<T> insert( BinNodePosi<T>, T const& ); //插入右孩子
         BinNodePosi<T> attach( BinTree<T>, BinNodePosi<T> ); //接入左子树
         BinNodePosi<T> attach( BinNodePosi<T>, BinTree<T> ); //接入右子树
         Rank remove( BinNodePosi<T> ); //子树删除
         BinTree<T>* secede( BinNodePosi<T> ); //子树分离
}
```

BinTree: 插入新节点

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

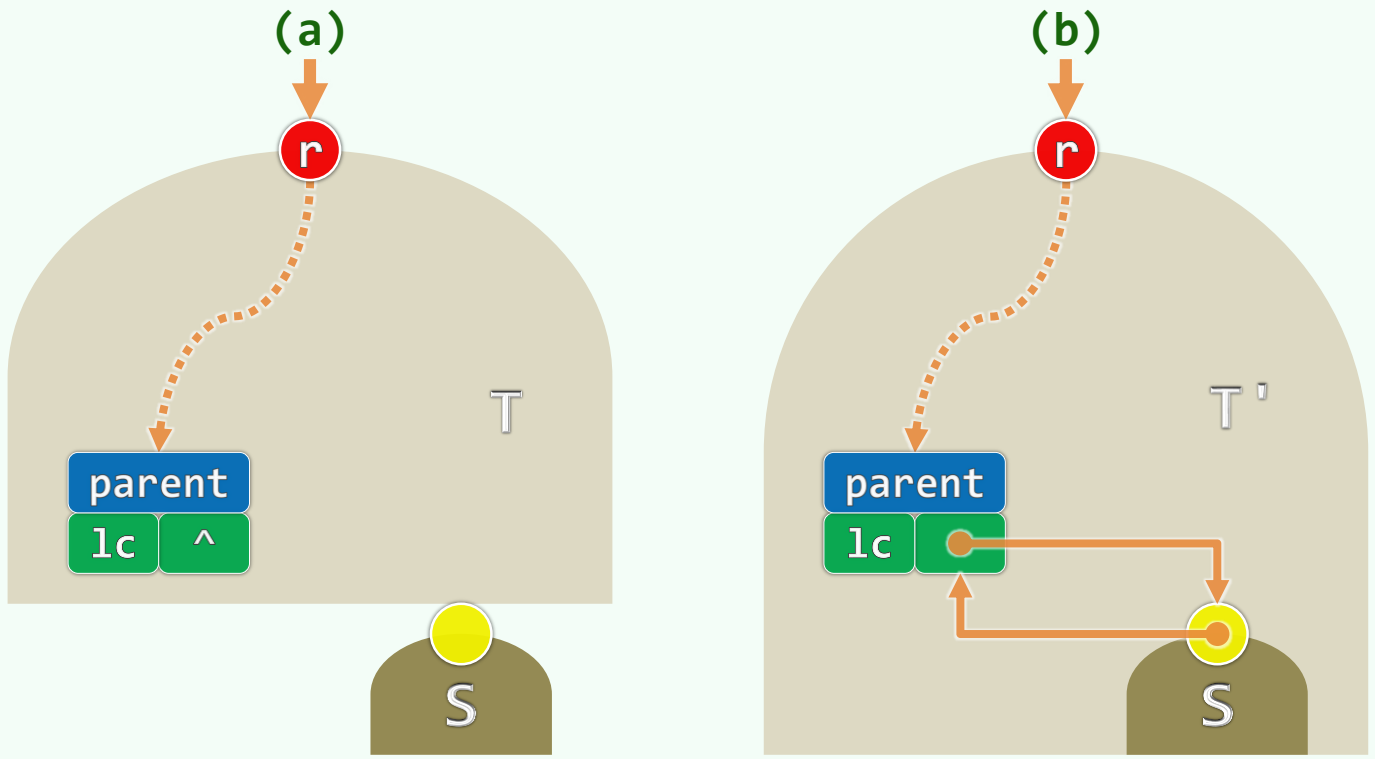
```
BinNodePosi<T> BinTree<T>::insert( T const & e, BinNodePosi<T> x ) { //作为左孩子
   _size++;
   x->insertLc( e );
   x->updateHeightAbove();
   return x->lc;
}
```



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;
   x->updateHeightAbove();
   S._root = NULL;
   S._size = 0;
   return x;
}
```



BinTree: 分离子树

```
template <typename T> BinTree<T>* BinTree<T>::secede( BinNodePosi<T> x ) {
   FromParentTo( x ) = NULL; x->parent->updateHeightAbove();
// 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
}
```