05-D

二叉树

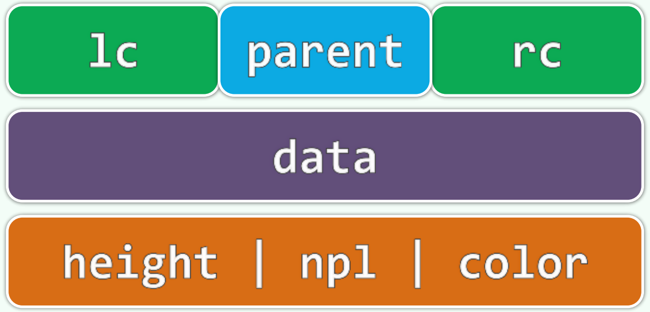
二叉树实现

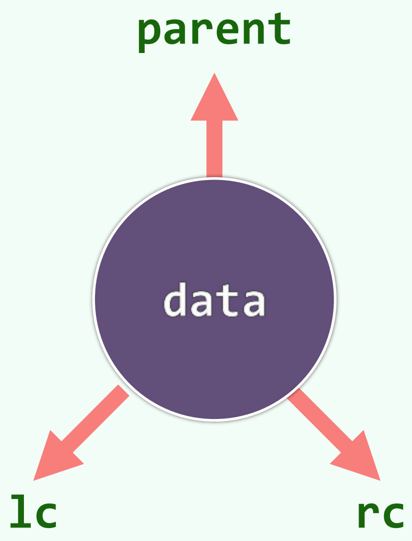
Anyone who loves his father or mother more than me is not worthy of me; anyone who loves his son or daughter more than me is not worthy of me.

邓俊辉
deng@tsinghua.edu.cn

BinNode模板类

```
❖ #define BinNodePosi(T) BinNode<T>* //节点位置

❖ template <typename T> struct BinNode {

      BinNodePosi(T) parent, lc, rc; //父亲、孩子

      T data; int height; int size(); //高度、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 & ); //子树层次遍历

      template <typename VST> void travPre( VST & ); //子树先序遍历

      template <typename VST> void travIn( VST & ); //子树中序遍历

      template <typename VST> void travPost( VST & ); //子树后序遍历

  };
```



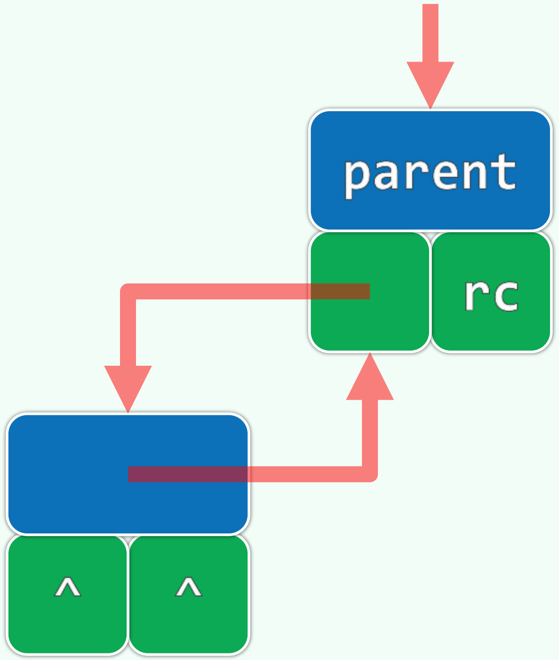


BinNode接口实现

❖
```
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 ); }
```

❖
```
template <typename T>
int BinNode<T>::size() { //后代总数，亦即以其为根的子树的规模
    int s = 1; //计入本身
    if (lc) s += lc->size(); //递归计入左子树规模
    if (rc) s += rc->size(); //递归计入右子树规模
    return s;
} //O( n = |size| )
```



BinTree模板类

```
template <typename T> class BinTree {
protected:
   int _size; //规模
   BinNodePosi(T) _root; //根节点
   virtual int 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:
   int 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 ... */
}
```

高度更新

❖ `#define stature(p) ( (p) ? (p)->height : -1 ) //节点高度——约定空树高度为-1`

❖ template <typename T> //更新节点x高度，具体规则因树不同而异

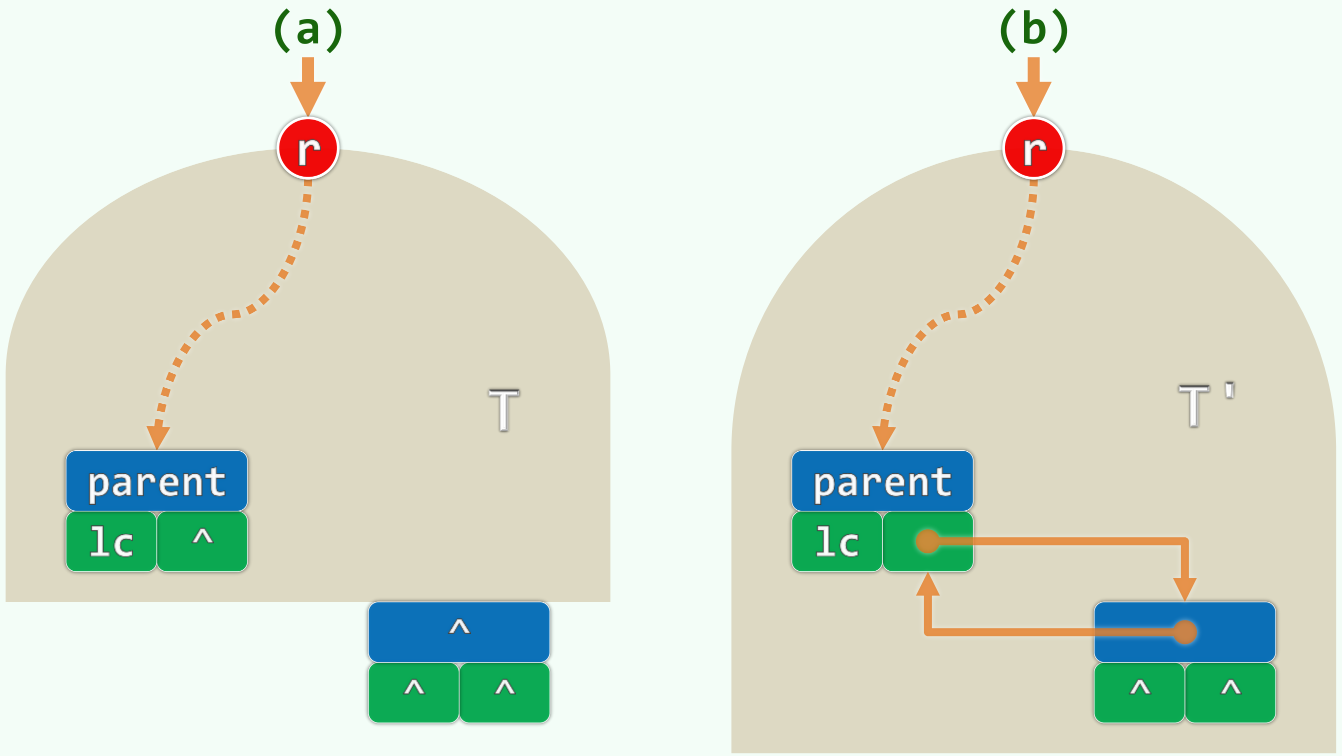
```
int BinTree<T>::updateHeight( BinNodePosi(T) x ) {
   return  x->height  =  1 + max( stature( x->lc ), stature( x->rc ) );
} //此处采用常规二叉树规则，O(1)
```

❖ template <typename T> //更新v及其历代祖先的高度

```
void BinTree<T>::updateHeightAbove( BinNodePosi(T) x ) {
   while (x) { updateHeight(x); x = x->parent; } //可优化
} //O( n = depth(x) )
```

节点插入

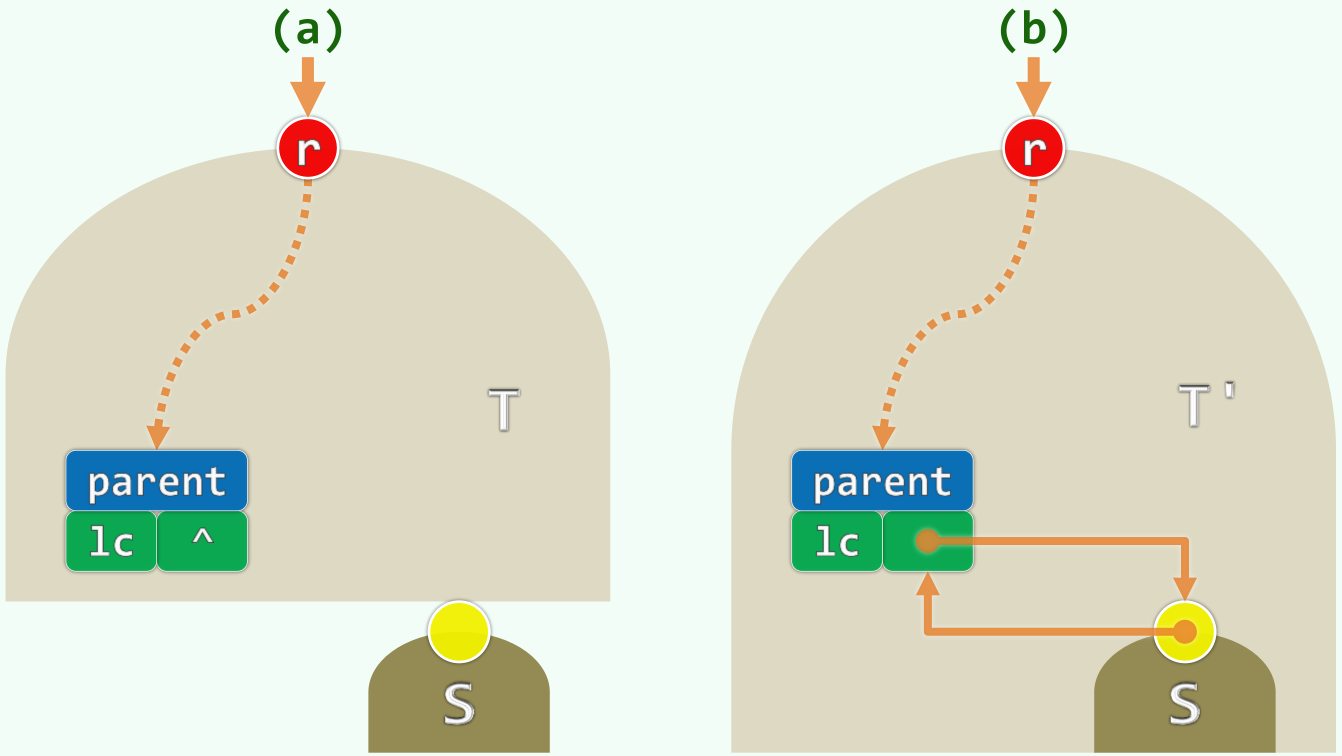
```
template <typename T>
  BinNodePosi(T) BinTree<T>::insertAsRC( BinNodePosi(T) x, T const & e ) {
     _size++;
     x->insertAsRC(e);
     updateHeightAbove(x);
     return x->rc;
  } //insertAsLC()对称
```



❖ x接入后，祖先的高度**可能增加**，其余节点**必然不变**

子树接入

```
template <typename T>
BinNodePosi(T) BinTree<T>::attachAsRC( BinNodePosi(T) x, BinTree<T>* & S ) {
   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;
} //attachAsLC()对称
```



子树删除

❖
```
template <typename T> int BinTree<T>::remove( BinNodePosi(T) x ) {
    FromParentTo( * x ) = NULL;
    updateHeightAbove( x->parent ); //更新祖先高度（其余节点亦不变）
    int n = removeAt(x); _size -= n; return n;
}
```

❖
```
template <typename T> static int removeAt( BinNodePosi(T) x ) {
    if ( ! x ) return 0;
    int n = 1 + removeAt( x->lc ) + removeAt( x->rc );
    release(x->data); release(x); return n;
}
```

子树分离

```
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
}
```