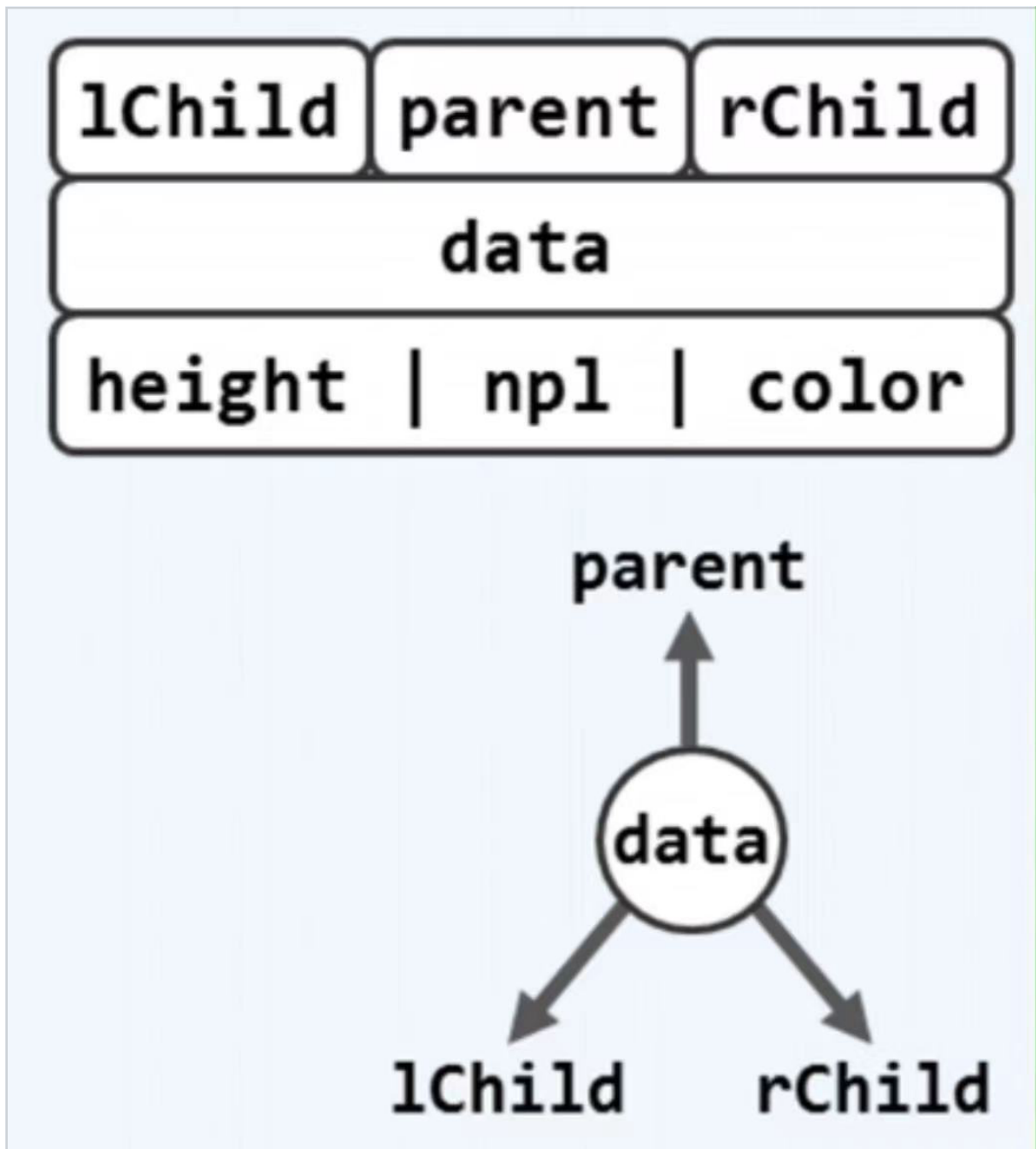


05-D 二叉树实现

#数据结构邓神

binNode模版类

数据组成：



Height:高度

Color：红黑树

npl????

每一个节点都是通过引用去指向其他的节点，每一个节点也都是被其他的节点用引用指向

BinNode模板类

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

❖ template <typename T> struct BinNode {

BinNodePosi(T) parent, lChild, rChild; //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

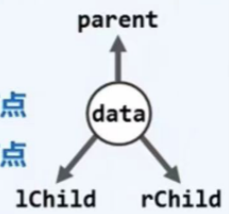
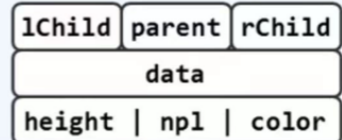
template <typename VST> void travLevel(VST &); //子树层次遍历

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

};



```
#define BinNodePosi(T) BinNode<T>* // 将节点引号 直接变成 位置
```

```
template <typename T> struct BinNode {
```

```
    // 父亲和左右孩子 引用
```

```
    BinNodePosi(T) parent,lChild,rChild;
```

```
    // 数据域
```

```
    T data;
```

```
    // 高度和子树的规模
```

```
    int height;
```

```
    int size(); // 所有后代的总数
```

```
    BinNodePosi(T) insertAsLC(T const &);
```

```
    BinNodePosi(T) insertAsRC(T const &);
```

```
    BinNodePosi(T) succ();
```

```
    // 遍历
```

```
    template <typename VST> void travLevel(VST &); // 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

BinNode接口实现

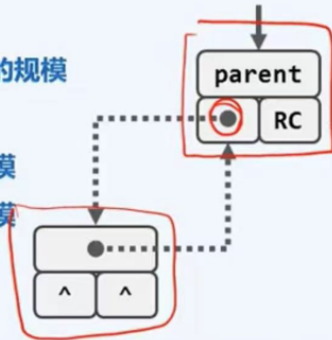
```

❖ template <typename T> BinNodePosi(T) BinNode<T>::insertAsLC(T const & e)
    { return lChild ← new BinNode( e, this ); } // this.LC == NULL

❖ template <typename T> BinNodePosi(T) BinNode<T>::insertAsRC(T const & e)
    { return rChild = 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 (lChild) s += lChild->size(); //递归计入左子树规模
    if (rChild) s += rChild->size(); //递归计入右子树规模
    return s;
} //O(n = |size|)

```



Data Structures & Algorithms (Fall 2013), Tsinghua University

BinTree 模版类

```

// BinTree
template <typename T> class BinTree {
protected:
    int _size;
    BinNodePosi(T) _root;
    virtual int updateHeight(BinNodePosi(T)x);
    void updateHeightAbove(BinNodePosi(T)x);
public:
    int size() const { // 规模
        return _size;
    }
    bool empty() const { // 判空
        return !_root;
    }
    BinNodePosi(T) root() const { // 树根
        return _root;
    }
    /* 子树接入，删除和分离接口 */
    /* 遍历接口 ... */
};

```

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 ... */  
}
```

data Structures & Algorithms (Fall 2013), Tsinghua University

高度更新

高度就是以他为根的子树从他通往最深的叶节点的路径长度

如果只有单节点高度取为 0

不存在任何节点的树高度取为-1

```
#define stature(p) ((p)?(p)->height : -1)
```

一个节点的高度应该等于他的左孩子或者右孩子的高度的更大者的高度再加一（能否用递归算法求出？）

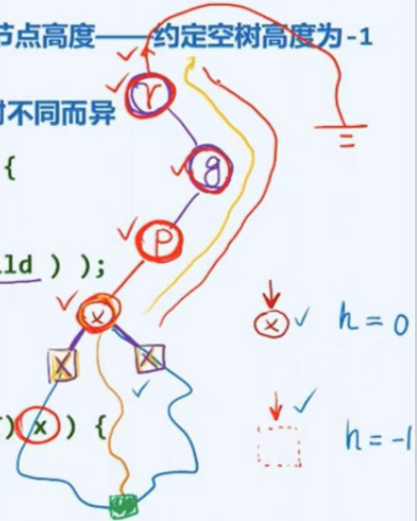
```
#define stature(p) ((p)?(p)->height : -1)  
template <typename T> int BinTree<T>::updateHeight(BinNodePosi(T) x){  
    return x->height = 1 + max(stature(x->lChild), stature(x->rChild));  
} // 采取常规二叉树规则 0(1)
```

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->lChild ), stature( x->rChild ) );
} //此处采用常规二叉树规则，0(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) //可优化：一旦高度未变，即可终止
    { updateHeight(x); x = x->parent; }
} //O( n = depth(x) )
    
```

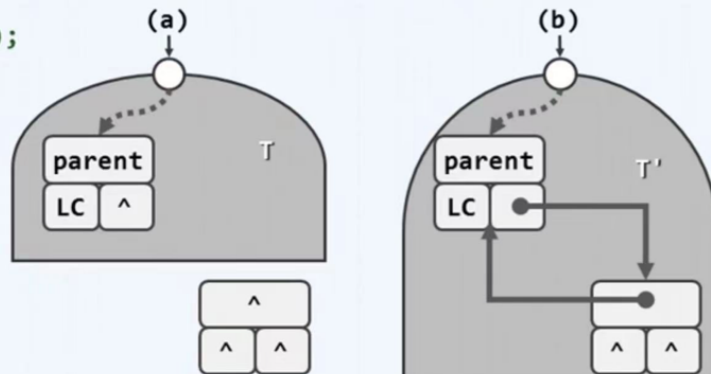


节点插入

节点插入

```

❖ template <typename T> BinNodePosi(T)
BinTree<T>::insertAsRC( BinNodePosi(T) x, T const & e ) { //insertAsLC()对称
    _size++; x->insertAsRC(e); //x祖先的高度可能增加，其余节点必然不变
    updateHeightAbove(x);
    return x->rChild;
}
    
```



// 插入节点

```

template <typename T> BinNodePosi(T) BinTree<T>::insertAsRC(BinNodePosi(T) x, T
const & e){
    _size++;
    x->insertAsRC(e);
    updateHeightAbove(x); // 在增加或者删除任何子节点都需要更新高度
    return x->rChild;
}
    
```

```
template <typename T> BinNodePosi(T) BinTree<T>::insertAsLC(BinNodePosi(T) x, T
const & e){
    _size++;
    x->insertAsLC(e); // 这边这个insertAsLC 是BinNode提供的接口，不是这个BinTree的接
    口，即不是递归，这边也没有递归的意义
    updateHeightAbove(x);
    return x->lChild;
}
```