

## 05-F 中序遍历

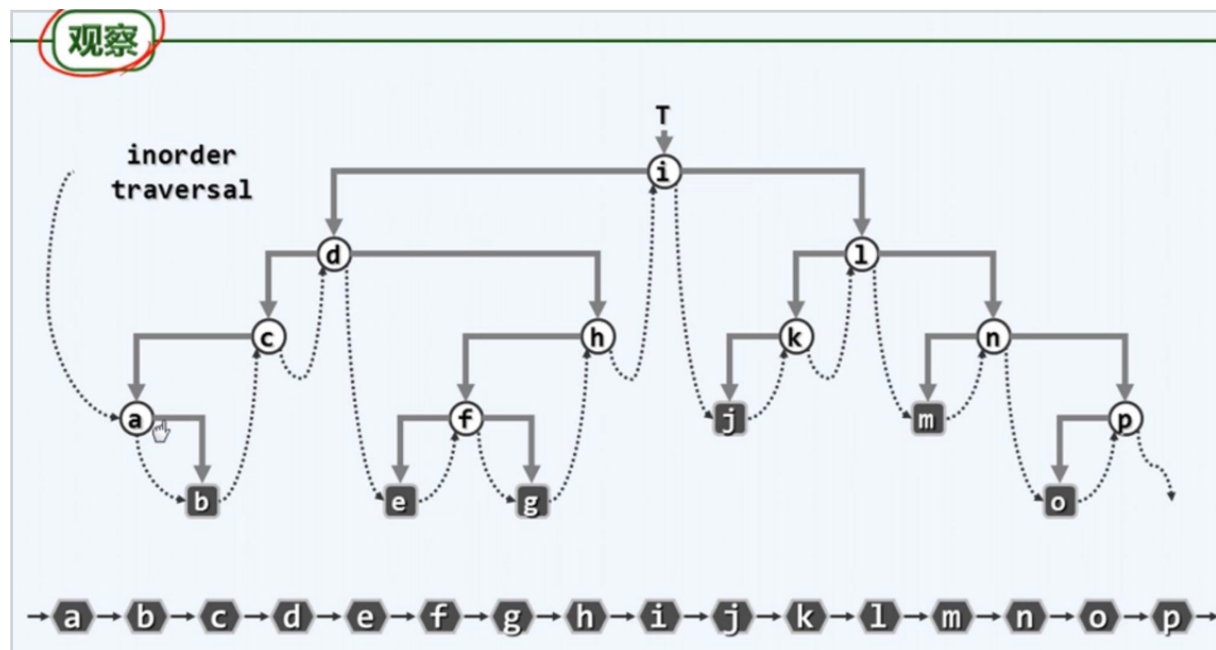
#数据结构邓神

### 递归

// 中序遍历

```
template <typename T,typename VST> void Mid_traverse(BinNodePosi(T) x,VST &
visit){
    if (!x){
        return;
    }
    Mid_traverse(x->lChild,visit);
    visit(x->data);
    traverse(x->rChild,visit);
} //  $T(n) = T(a) + O(1) + T(n-a-1) = O(n)$ 
```

### 观察

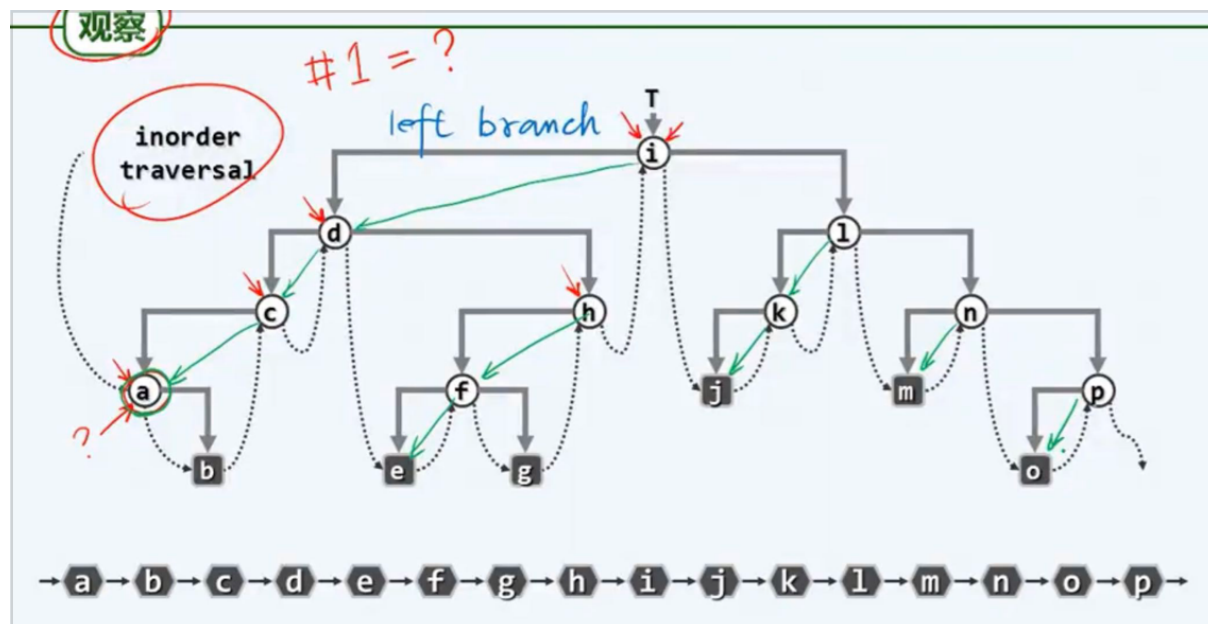


通过这张图我们可以简单明了的发现，当中序搜索时

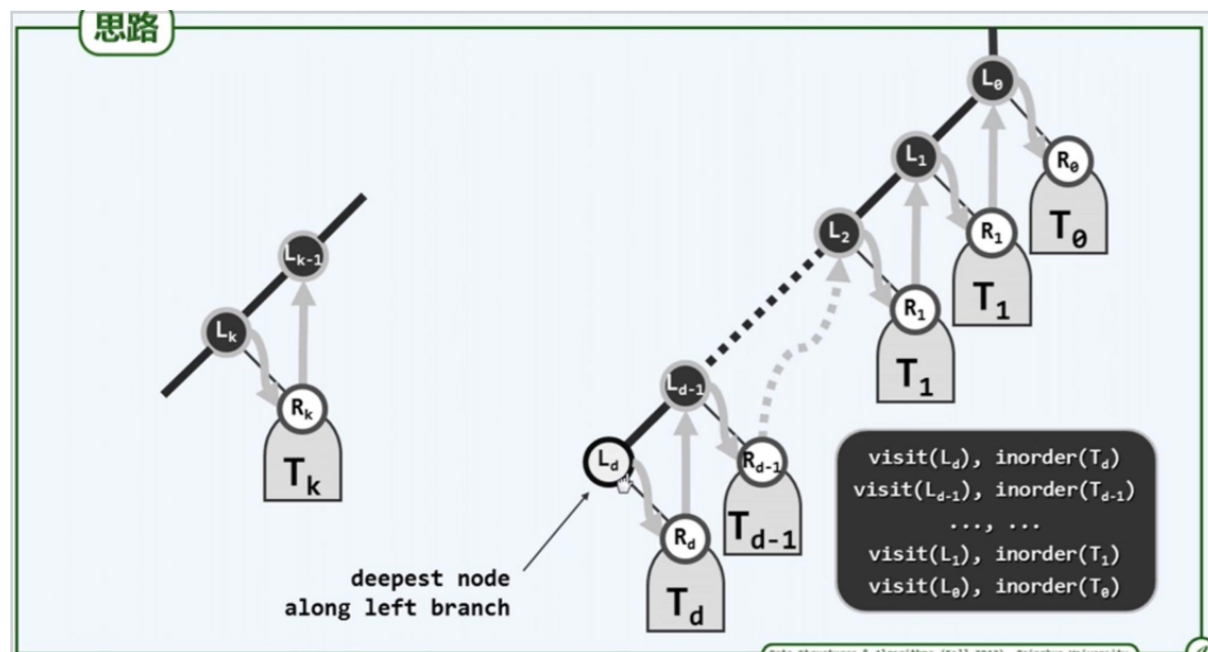
首先算法从 i 开始，除非算法扫描完成整个一颗左子树，否则就不会扫描 i 所以 i 会把控制权交给 d，d 也是一样，除非 d 的左子树被扫描完毕，也不会扫描 d，就会继续转交，到 c c 给 a，a 没有左子树，可以扫描 a

与先序遍历非常类似，也是沿着左侧分支逐层向下，找到第一个左孩子的节点

有点像先序遍历但是反过来，哈哈哈



思路



构思

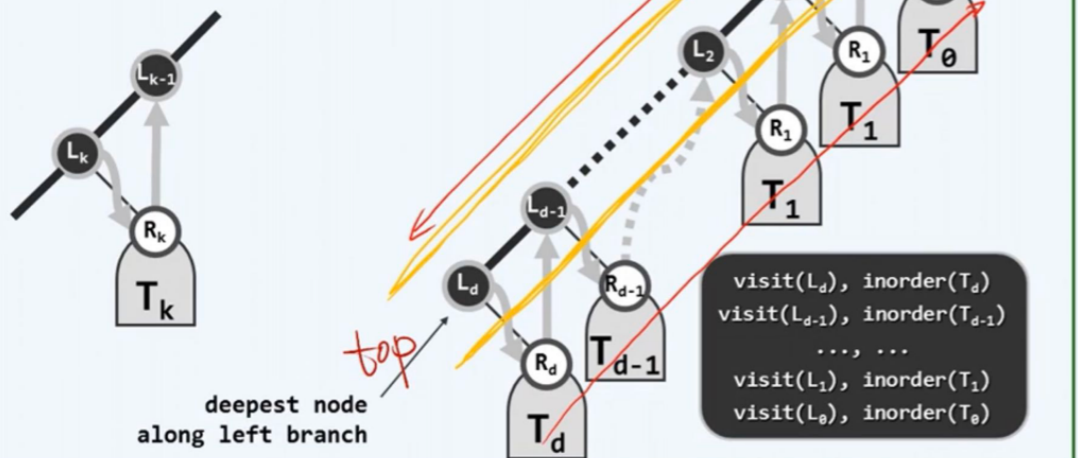
从根出发沿着左分支下行，直到最深的节点，在其中不断将根节点和右子树入栈，然后到了最深处，不断出栈 !!!!

❖ 从根出发沿左分支下行，直到最深的节点  
——它就是全局首先被访问者

### 思路

❖ 从根出发沿**左分支**下行，直到最深的节点  
——它就是全局首先被访问者

stack  
LIFO



## 实现

```
// 中序遍历 迭代
```

```
template <typename T> static void goAlongLeftBranch(BinNodePosi(T)
```

```
x, stack<BinNodePosi(T)> & S){
```

```
while(x){
```

```
S.push(x); // 入栈当前节点
```

```
x = x->lchild; // 向左侧深入
```

}

}

```
template <typename T,typename VST> void travIN_I1(BinNodePosi(T) x,VST & visit)
```

{

```
stack<BinNodePosi(T)> S;
```

```
while (true){
```

```
goAlongLeftBranch(x,S);
```

```
if (S.empty()){
```

```
break;
```

}

```
x = S.top();
```

```
S.pop();
```

```
visit(x->data);
```

```
x = x->rChild;
```

}

}

## 实现

```

❖ template <typename T>
    static void goAlongLeftBranch( BinNodePosi(T) x, Stack <BinNodePosi(T)> & S )
    { while (x) { S.push(x); x = x->lChild; } } //反复地入栈，沿左分支深入

❖ template <typename T, typename V> void travIn_I1( BinNodePosi(T) x, V& visit ) {
    ✓ Stack <BinNodePosi(T)> S; //辅助栈
    while (true) { //反复地
        goAlongLeftBranch( x, S ); //从当前节点出发，逐批入栈
        if ( S.empty() ) break; //直至所有节点处理完毕
        x = S.pop(); //x的左子树或为空，或已遍历（等效于空），故可以
        visit( x->data ); //立即访问之
        x = x->rChild; //再转向其右子树（可能为空，留意处理手法）
    }
}

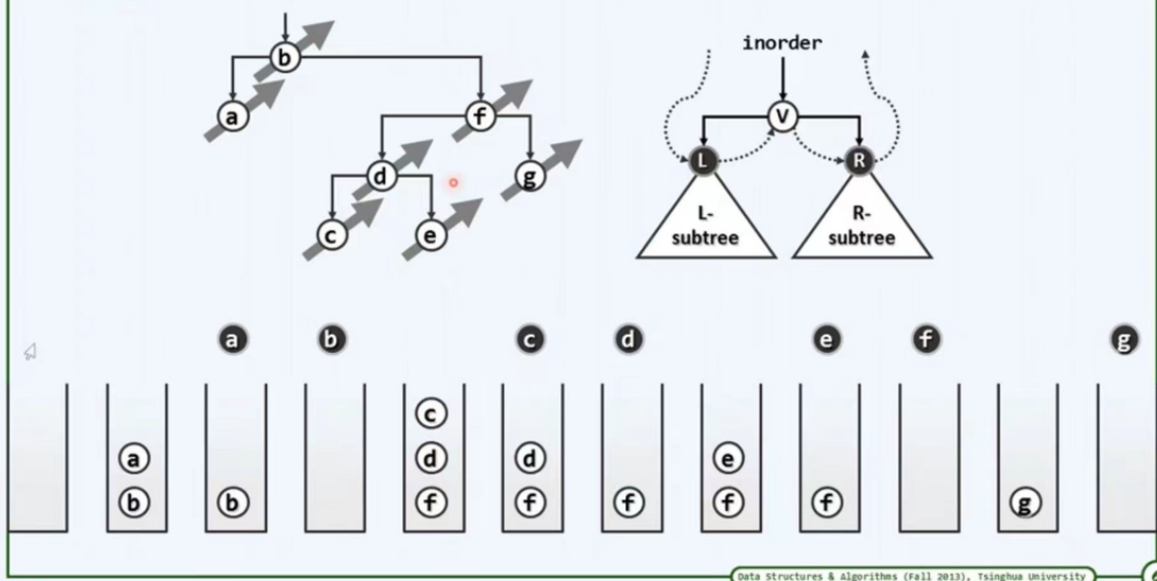
```

Data Structures & Algorithms (Fall 2013), Tsinghua University

5

## 实例

### 实例



Data Structures & Algorithms (Fall 2013), Tsinghua University

6

## 分摊分析

### 回到算法

```

template <typename T> static void goAlongLeftBranch(BinNodePosi(T)
x,stack<BinNodePosi(T)> & S){
    while(x){
        S.push(x); // 入栈当前节点
        x = x->lchild; // 向左侧深入
    }
}

```

```

    }
}
template <typename T,typename VST> void travIN_I1(BinNodePosi(T) x,VST & visit)
{
    stack<BinNodePosi(T)> S;
    while (true){
        goAlongLeftBranch(x,S);
        if (S.empty()){
            break;
        }
        x = S.top();
        S.pop();
        visit(x->data);
        x = x->rChild;
    }
}
}

```

外循环至少 $O(n)$ ，内循环也可能要 $O(n)$ ？难道总体的复杂度为  $O(n^2)$  ???

整个算法呈现一个迭代形式，

但是内部循环迭代次数所有的和在一起也无非是  $O(n)$ ，所以  $O(n^2)$  的时间复杂度是一个假象，他是永远到不了的。

总的算法复杂度类似于  $O(2n)$  要远远小于递归版本的常系数

## 分摊分析