### 5. 二叉树

### (e1) 先序遍历

真君曰: "昔吕洞宾居庐山而成仙,鬼

谷子居云梦而得道,今或无此吉地么?"

璞曰:"有,但当遍历耳。"

邓俊辉

deng@tsinghua.edu.cn

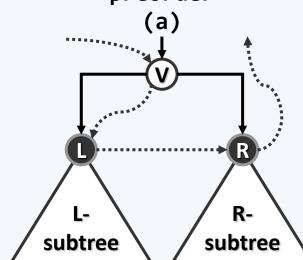
## 遍历

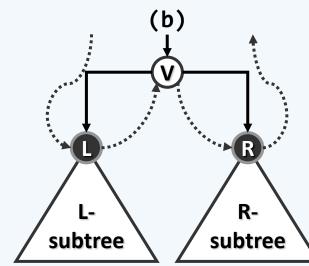
❖ 按照 某种次序 访问树中各节点,每个节点被访问 恰好一次

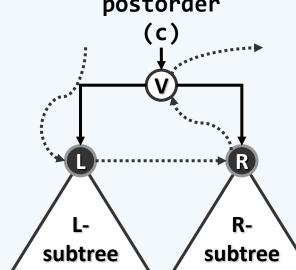
 $T = V \cup L \cup R$ 

❖ 遍历结果 ~ 遍历过程 ~ 遍历次序 ~ 遍历策略

中序 先序 层次(广度) 后序 preorder inorder postorder (b) (a)







### 递归

```
* template <typename T, typename VST>
  void traverse( BinNodePosi(T) x, VST & visit ) {
    if ( !x ) return;
    visit( x->data );
    traverse( x->lChild, visit );
    traverse( x->rChild, visit );
```

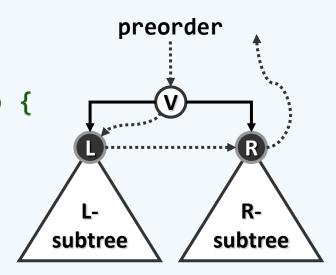
//T(n) = O(1) + T(a) + T(n-a-1) = O(n)

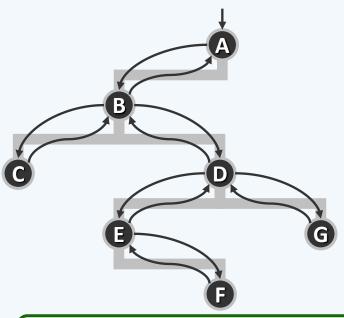
#### ❖ 先序输出文件树结构

c:\> tree.com c:\windows

❖ 挑战: 不依赖递归机制,能否实现先序遍历?

如何实现?效率如何?





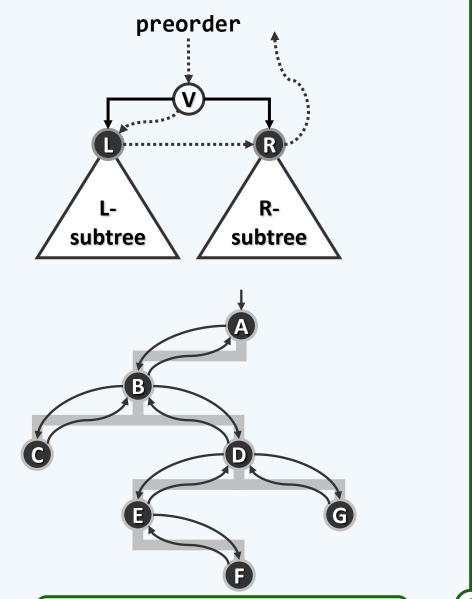
### 迭代1:思路

- ❖ 先序遍历任一二叉树T的过程,无非是 先访问根节点r
  再先后递归地遍历T<sub>L</sub>和T<sub>R</sub>
- ❖ 递归实现中,对左、右子树的递归遍历

都类似于 尾递归

故不难直接消除

- ❖思路:
  - 二分递归 → 迭代 + 单递归 → 迭代 + 栈



### 迭代1:实现

```
❖ template <typename T, typename VST>
                                                       preorder
 void travPre I1( BinNodePosi(T) x, VST & visit ) {
    Stack <BinNodePosi(T)> S; //辅助栈
    if (x) S.push(x); //根节点入栈
                                                   subtree
                                                             subtree
    while ( !S.empty() ) { //在栈变空之前反复循环
       x = S.pop(); visit(x->data); //弹出并访问当前节点
       if ( HasRChild( *x ) ) S.push( x->rChild ); //右孩子先入后出
       if ( HasLChild( *x ) ) S.push( x-> lChild ); //左孩子后入先出
    } //体会以上两句的次序
```

# 迭代1:实例 preorder Rsubtree subtree

### 迭代1:分析

❖ 正确性

无遗落 : 每个节点都会被访问到

归纳假设:若深度为d的节点都能被正确访问到,则深度为d+1的也是

根先: 对于任一子树,根被访问后才会访问其它节点

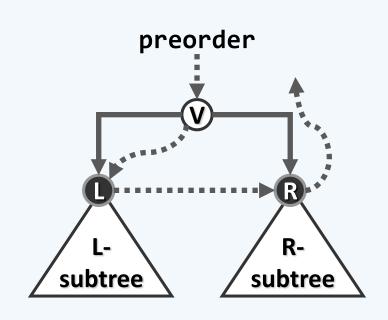
只需注意到:若u是v的真祖先,则u必先于v被访问到

左先右后 : 同一节点的左子树 , 先于右子树被访问

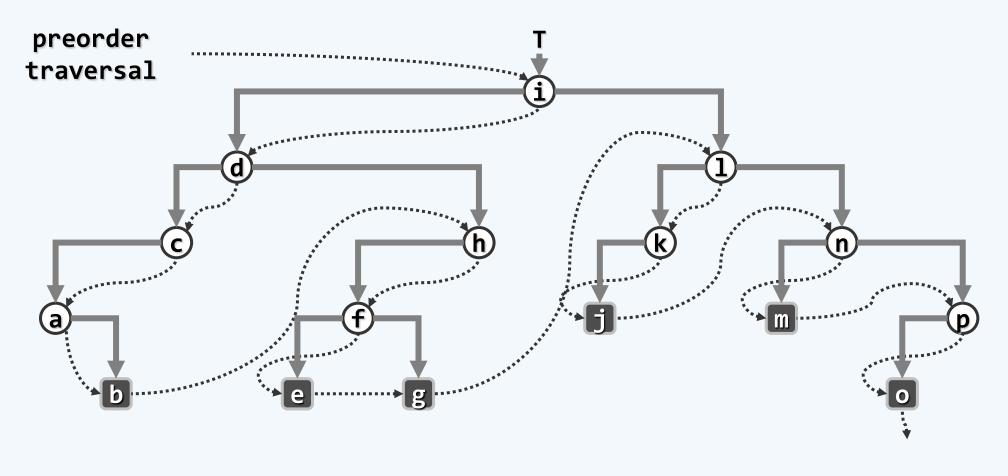
❖ 效率: 0(n)

每步迭代,都有一个节点出栈并被访问 每个节点入/出栈一次且仅一次 每步迭代只需0(1)时间

❖ 以上消除尾递归的思路不易推广,需要另寻他法...



### 迭代2:思路



### 迭代2:思路

❖ 沿着左侧分支
各节点与其右孩子(可能为空)——对应

❖ 从宏观上,整个遍历过程可划分为

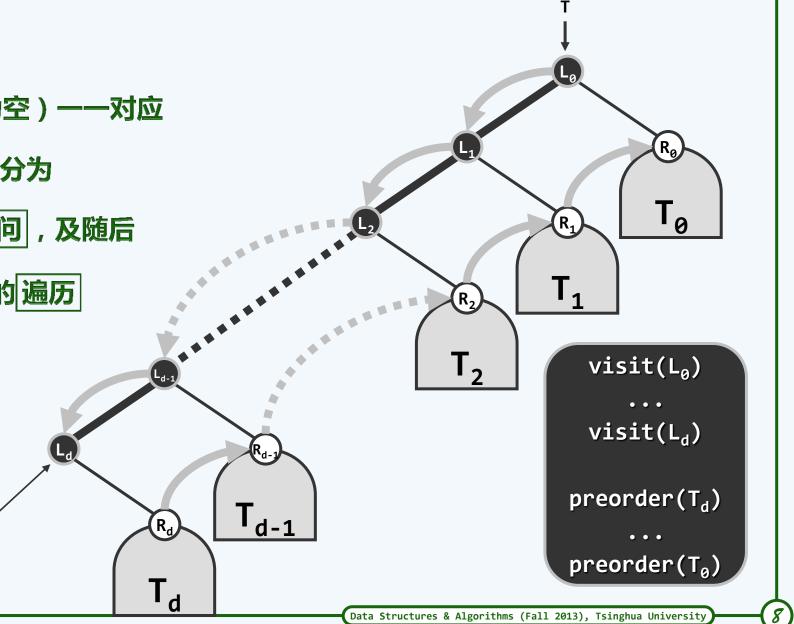
自上而下对 左侧分支 的 访问 , 及随后

自下而上对一系列 右子树 的 遍历

❖ 不同右子树的遍历 相互独立

自成一个子任务

deepest node along left branch



### 迭代2:实现

```
❖ template <typename T, typename VST> //分摊O(1)
 static void visitAlongLeftBranch(
    BinNodePosi(T) x,
    VST & visit,
    Stack <BinNodePosi(T)> & S )
    while (x) { //反复地
      visit(x->data);//访问当前节点
      S.push( | x->rChild | ); //右孩子(右子树) 入栈(将来逆序出栈)
      x = x->1Child; //沿左侧链下行
    } //只有右孩子、NULL可能入栈——增加判断以剔除后者,是否值得?
```

### 迭代2:实现

```
❖ template <typename T, typename VST>
 void travPre I2( BinNodePosi(T) x, VST & visit ) {
    Stack <BinNodePosi(T)> S; //辅助栈
    while (true) { //以 (右)子树 为单位,逐批访问节点
      visitAlongLeftBranch(x, visit, S); //访问子树x的 左侧链 , 右子树 入栈缓冲
      if (S.empty()) break; //栈空即退出
      x = S.pop(); //弹出下一子树的根
    } //#pop = #push = #visit = O(n) = 分摊O(1)
```

### 迭代2:实例

